DETERMINATION OF OPTIMAL OPERATING SCHEMES FOR A HYDROPOWER RESERVOIR UNDER ENVIRONMENTAL CONSTRAINTS

By

Amelia R. Shaw

Dissertation

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

in

Environmental Engineering

May 11, 2018

Nashville, Tennessee

Approved:

Eugene J. LeBoeuf, Ph.D., P.E.

Mark McDonald, Ph.D., P.E.

George Hornberger, Ph.D.

Mark Ellingham, Ph.D.

Boualem Hadjerioua, Ph.D.

In memory of my grandmother, who raised my dad to become the kind of man who encouraged his daughter to play in the dirt. For that, I am forever grateful.

Marilyn Lease Shaw

1925 - 2018

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Eugene LeBoeuf, for first introducing me to hydropower operations and its environmental impacts, and for his continued support. I also want to thank my committee members for the time they spent guiding this research. Drs. Mark McDonald and Bo Hadjerioua were integral to the success of this project. I appreciate the thorough mathematics review provided by Dr. Mark Ellingham. Lastly, thanks to Dr. George Hornberger for his mentorship and for reminding me to step back and examine the big picture.

Funding for this work was provided by the U.S. Department of Energy Wind and Water Program under contract DE-AC05-00OR22725 through Oak Ridge National Laboratory and the U.S. Department of Energy Office of Energy Efficiency and Renewable Energy under Award Number DE-EE0002668 through the Hydro Research Foundation. The Vanderbilt University Department of Civil and Environmental Engineering provided additional financial support, and I thank the faculty and staff for supporting the research and career development of graduate students. Additional scholarship and fellowship support was provided by the National Precast Concrete Association, Chi Epsilon, the American Water Resources Association, and the Air and Waste Management Association Southern Section.

Thank you to Heather Smith Sawyer for helping me learn the ins and outs of CE-QUAL-W2. Bob Sneed and Jeff Gregory from the U.S. Army Corps of Engineers Nashville District provided reservoir operations data, models, and lots of guidance. Gergely Varga from the Institute for Software Integrated Systems trimmed years from this work with his data management tools. I greatly appreciate the support and resources provided by the Hydro Research Foundation and want to especially thank Brenna Vaughn, Deborah Linke, and Mike Sale for welcoming me and the other "Hydro Fellows" to the world of hydropower with open arms.

I had the honor of spending the first part of 2017 as a Mirzayan Fellow at the National Academy of Engineering. I want to thank Randy Atkins for his excellent mentorship, the NAE Program Office staff for providing so many opportunities to get involved, and Dr. Anne-Marie Mazza for inspiring years of fellows through her leadership of the program. My brilliant class of fellows made the experience so fulfilling, and I thank them for sharing their passion for science policy.

Last but certainly not least, none of this would have been possible without the support of my family and friends. I have to thank my parents for always prioritizing my education. The importance of the camaraderie provided by my fellow graduate students really can't be measured, and I've made many lifelong friends during my time here. My bookworm of a fiancé Sam deserves a round of applause for reviewing and helping improve my writing. More importantly, I appreciate him for always believing in me even at times when I struggled to believe in myself.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS AND ACRONYMS

ACO	ant colony optimization
ALGA	Augmented Lagrangian Genetic Algorithm
AME	absolute mean error
ANN	artificial neural network
BOD	biochemical oxygen demand
CalSim	California Water Resources Simulation Model
CBOD	carbonaceous biochemical oxygen demand
CSO	combined sewer overflow
CSV	comma separated values
CWMS	Corps Water Management System
DO	dissolved oxygen
DOP	dynamic optimization problem
DP	dynamic programming
DSS	decision support system
EFDC	Environmental Fluid Dynamics Code
EGO	efficient global optimization
EIF	expected improvement function
GA	genetic algorithm
GEM	Gaussian emulator machine
HBMO	honey bees mating optimization

HEC	Hydrologic Engineering Center
HEC-PRM	Hydrologic Engineering Center Prescriptive Reservoir Model
HSPF	Hydrologic Simulation Program Fortran
JDAY	Julian day
kNN	k-nearest neighbors
LP	linear programming
MBO	marriage in honey bees optimization
MINLP	mixed integer nonlinear programming
MW	megawatt
MWh	megawatt-hour
NARX	nonlinear autoregressive network with exogenous inputs
NLP	nonlinear programming
OECD	Organization for Economic Cooperation and Development
PSO	particle swarm optimization
RBF	radial basis function
SA	simulated annealing
SMS	Surface Water Modeling System
SVM	support vector machine
SVR	support vector regression
SWAT	Soil and Water Assessment Tool
TCD	temperature control device

TDG	total dissolved gas
TDS	total dissolved solids
TKN	total Kjeldahl nitrogen
TMDL	total maximum daily load
TN	total nitrogen
TOC	total organic carbon
TVA	Tennessee Valley Authority
USACE	U.S. Army Corps of Engineers
USBR	U.S. Bureau of Reclamation
USEPA	U.S. Environmental Protection Agency
WASP	Water Quality Analysis Simulation Program
WQM	water quality model
WRIMS	Water Resource Integrated Modeling System

Chapter I

INTRODUCTION

One of the largest challenges facing societies worldwide is energy scarcity. Global energy consumption is projected to grow by 48% over the 28-year period from 2012 to 2040 (*U.S. Department of Energy*, 2016a). This growth stems largely from a consumption increase in countries outside the Organization for Economic Cooperation and Development (OECD), whose membership consists of 35 countries worldwide, most of which are advanced (*The Organisation for Economic Co-operation and Development (OECD)*, 2018); however, energy use is still predicted to increase by 18% in OECD member countries, which includes the United States (*U.S. Department of Energy*, 2016a). While fossil fuels will continue to dominate world energy use, renewable resources are the fastest growing electricity source, rising by 2.9% each year worldwide and 1.8% each year in the United States through 2040 (*U.S. Department of Energy*, 2016a).

The majority of hydroelectric power derives from dammed river systems, in which impounded water's potential energy drives turbines and generates electricity. There are currently over 45,000 large (over 10^7 cubic meters of storage as defined by *Graf* (2005)) dams worldwide (*McCartney*, 2009). Twenty-seven percent of the projected growth in worldwide renewables is expected to come from hydroelectric power (*U.S. Department of Energy*, 2016a), primarily from construction of new, large, and gravity concrete and earth dammed systems. While other countries are still actively constructing large conventional dams, the U.S. has witnessed a sharp decline in new large dam construction since the 1970s (Figure I.1), primarily due to concerns over adverse environmental impacts (Endangered Species Act of 1973, Clean Water Act of 1977). U.S. hydroelectric power generation is projected to increase by 0.1% annually (*U.S. Department of Energy*, 2016a), corresponding to 1.7% of U.S. renewables growth. This is expected to be derived from hydropower development at existing non-powered dams, additional pumped-storage facilities, new small in-stream hydropower, and improved turbine and generator efficiencies through equipment upgrades and optimized reservoir and turbine operations procedures (*U.S. Department of Energy*, 2015). This growth is important as hydropower can supplement power demands, especially as a responsive and flexible power genera-

tion source during peak demand periods, which thermal electric power sources and other renewables cannot deliver (*U.S. Department of Energy*, 2016b). Without construction of new large hydropower projects, the projected increase of hydroelectric power must come from improved equipment efficiencies and optimized operation procedures. This research focuses on the latter idea.

The general environmental impacts of dams and hydropower operations are well-known, but the exact impacts of a particular dam are difficult to predict due to unique characteristics of aquatic ecosystems (*Friedl and Wuest*, 2002; *McCartney*, 2009). Hydropower plants typically operate on a "peaking" schedule, supplying additional electricity to the power grid during high demand periods. This can result in flow fluctuations, impacting downstream fish habitats (*Jager and Smith*, 2008). Globally on average, damming triples river water residence times (*Covich*, 1993). Reduced flow velocities enhance sedimentation rates upstream of dams, and the reduced sediment loads and fluctuating velocities can enhance erosion downstream (*McCartney*, 2009). The resulting large mass of still water absorbs heat and may result in stratification, where surface water layers are considerably warmer than deeper layers. If release locations are deep in the reservoir, the reservoir releases can be considerably cooler than would occur under a natural regime (*McCartney*, 2009). Drought and warm weather exacerbate this due to greater differences in water densities between the cool deep water and warmer surface waters (*Dortch*, 1997). Thermal stratification reduces vertical exchanges, which can create anoxic conditions in deep water layers. If outflow structure elevations lie in oxygen-depleted regions of a reservoir, discharge waters may also be oxygen-depleted. When most of the energy of the release is dedicated to power production, this leaves little energy for reaeration (*Dortch*, 1997). This water may also have reduced levels of other compounds, leading to a poor downstream assimilative capacity; this can be especially harmful in river reaches which receive wastewater and other effluents (*Friedl and Wuest*, 2002). Temperature and dissolved oxygen (DO) are primarily the greatest water quality interest for reservoirs, as temperature regulates biotic growth rates and oxygen is necessary to sustain life within waterbodies (*Dortch*, 1997). Studies of U.S. Army Corps of Engineers (USACE) and the Tennessee Valley Authority (TVA) water resources projects in the southeastern U.S. revealed significant dam tailwater quality DO issues (*Kennedy and Gaugush*, 1988; *Hayes et al.*, 1998; *Higgins and Brock*, 1999). The greatest needs associated with dams relate to tailwater quality, especially for hydropower projects where structural design and the desire to meet maximum turbine efficiency reduces reaeration during power generation (*Kennedy*

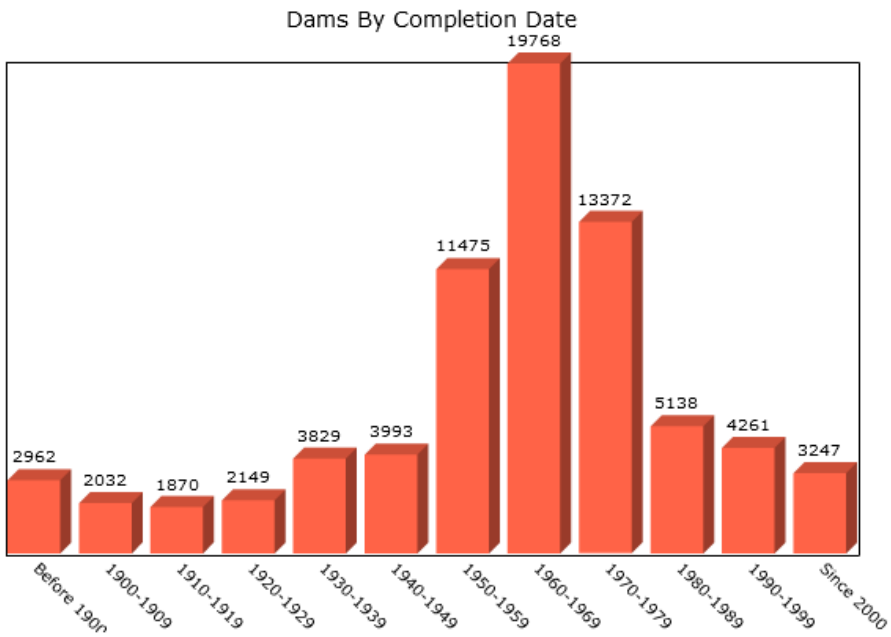


Figure I.1: Dams in the United States by completion date (*U.S. Army Corps of Engineers*, 2013a).

and Gaugush, 1988). Additionally, nuclear and coal power plants rely on river flow for condenser cooling water and must comply with regulatory temperature limits before discharging the cooling water into the river (*U.S. Environmental Protection Agency*, 2016). Consequently, there is great value in managing reservoir temperatures to minimize thermal power plant derating, especially during warm weather periods. Reservoir operators must also consider how warm water releases from thermal plants and peaking cold water released from hydropower dams can produce sudden temperature changes, which may negatively impact sensitive fish species, particularly during winter.

Tradeoffs are made when considering both water quantity and quality objectives, often resulting in a desire for flow release decision-making that benefits water quality in conjunction with other project demands, such as flood abatement or energy production (*Loftis et al.*, 1985). There are three primary mechanisms that can improve water quality: (i) pretreatment or control of inflows, (ii) in-pool management or treatment techniques, and (iii) outflow management (*Dortch*, 1997). Outflow management is the most commonly used method, as controlling outflow rates, outlet locations, and timing of releases can impact both in-pool and release water quality by influencing in-pool water quality gradients (*Dortch*, 1997; *Price and Meyer*, 1992). Outflow decision-making represents the primary focus of this research work.

Reservoirs with hydropower capabilities are generally operated with the primary goal of maximizing energy production while meeting legal water regulations (*Jager and Smith*, 2008). The optimization of reservoir operations has been extensively studied, with initial studies focusing on water quantity constraints and more recent studies integrating constraints related to wildlife and water quality. The limited number of studies which consider water quality have not employed state-of-the-art two-dimensional high-fidelity water quality models (WQMs), instead incorporating one-dimensional coarse-grid models or minimum flow requirements deemed to support sufficient water quality (*Jager and Smith*, 2008). For example, *Hayes et al.* (1998) integrated the quasi-2D coarse-grid water quality DORM-II model of the upper Cumberland River basin in the southeastern United States into an optimal control model to analyze water quality improvement opportunities through operational changes. While computationally feasible, this work included simplifications such as 24 hour periods of generation, stratification defined by two well-mixed vertical layers with no mixing between layers, and simplified heat transfer and reaeration equations. Optimizing operations for a single reservoir under simulated environmental constraints has proven computationally difficult, and expanding to multireservoir systems is even more challenging (*Dhar and Datta*, 2008). A technique for integrating high-fidelity water quality simulation models within a hydropower decision support system would provide reservoir releases which better meet defined objectives and constraints.

I.1 Plan of Research

Presented here is an approach for computing globally optimal power generation schemes for a hydropower reservoir using high-fidelity WQMs, surrogate modeling techniques, and multidimensional optimization methods. The combination of these approaches allows for the inclusion of high-fidelity water quality constraints within dam release decision making on an operational timescale, as well as comparison between resulting optimal schemes and current operating procedures. This methodology reveals a power generation benefit while maintaining water quality standards or minimizing water quality standard violations.

The primary objective is to perform simulation and optimization for determination of flow releases from turbines and control structures along river systems with consideration of power produc-

tion, navigability, temperature, water quality, and flood risk. The general workflow for this process is shown in Figure I.2. To determine optimal releases, high-fidelity spatial and temporal information are needed on system hydraulics and water quality. This information is generally managed on an individual system basis, and can be estimated by high-fidelity models such as the CE-QUAL-W2 model (*Cole and Wells*, 2007), which is currently used by the USACE and TVA to model the Cumberland and Tennessee Rivers, respectively. A section of the Cumberland River containing two USACE hydropower projects (Old Hickory and Cordell Hull reservoirs) is used as a prototype system (Figure I.3). These run-of-the-river type hydropower facilities have small storage capacities which are sensitive to smaller timescale variations in inflows and outflows (*Ferreira and Teegavarapu*, 2012); therefore, short-term operations planning on daily or hourly timescales is highly valuable.

As expressed by *Bartholow et al.* (2001), there is a need to link optimization software with the CE-QUAL-W2 model, which would allow managers to satisfy both downstream and in-reservoir water quality objectives. Previously *Dhar and Datta* (2008) developed a method for determining optimal short-term operation of a single reservoir to control downstream water quality through a linked simulation (CE-QUAL-W2) and optimization (elitist genetic algorithm) process. Their methodology is limited by time requirements of the simulation model, which could be improved through development of parallel code or use of metamodels. Metamodels, also known as response surface models, surrogates, or emulators, mimic the behavior of a simulation model with substantial computational savings (*Forrester et al.*, 2008).

Chapter II details the state-of-the-art of research in the areas of reservoir modeling and operations, surrogate modeling techniques, and hydropower systems optimization. Following chapters detail work encompassing three main objectives, all centered around the goal of exploring optimal operational schemes while maintaining water quality. In Chapter III, construction of surrogate WQMs and integration of these models within an optimization application is described. This is applied to a single multipurpose reservoir with hydropower capabilities, and the surrogate-enabled optimizer is used to explore the trade-offs between spillway and hydropower flow releases. Chapter IV focuses on the optimizer itself, exploring modifications to the optimization algorithm which improve solution quality. Random immigrants replacement, a technique to improve genetic algorithm (GA) population diversity when solving dynamic optimization problems, and soliciting additional surrogate model training data adaptively mid-optimization are both investigated. Chapter V looks

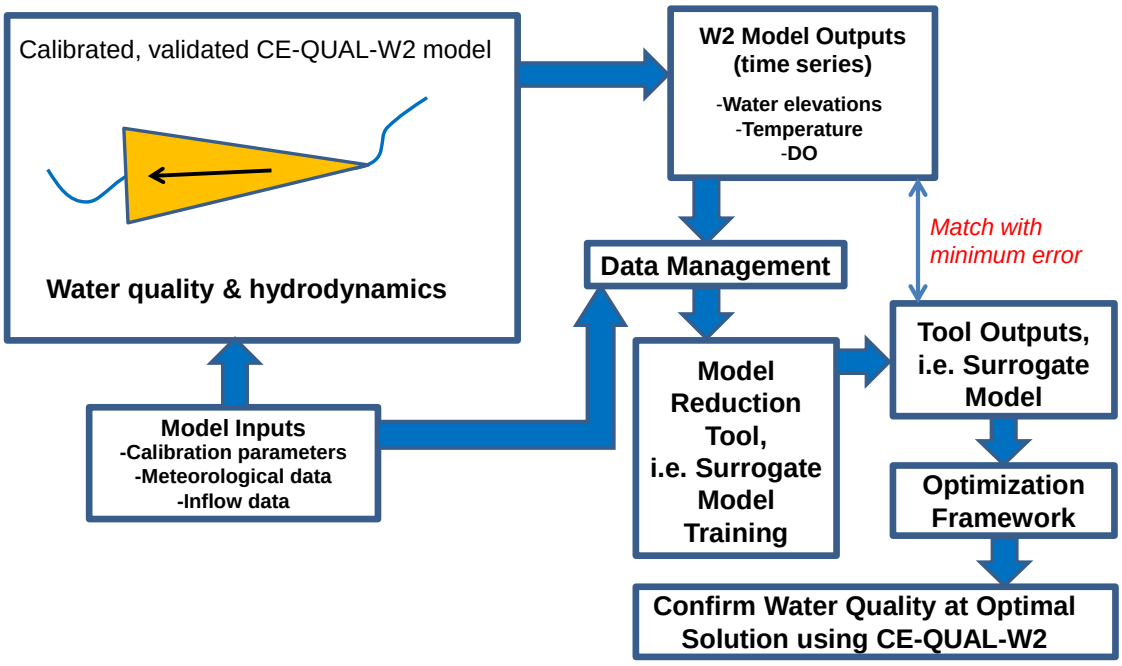


Figure I.2: Methodology overall approach.

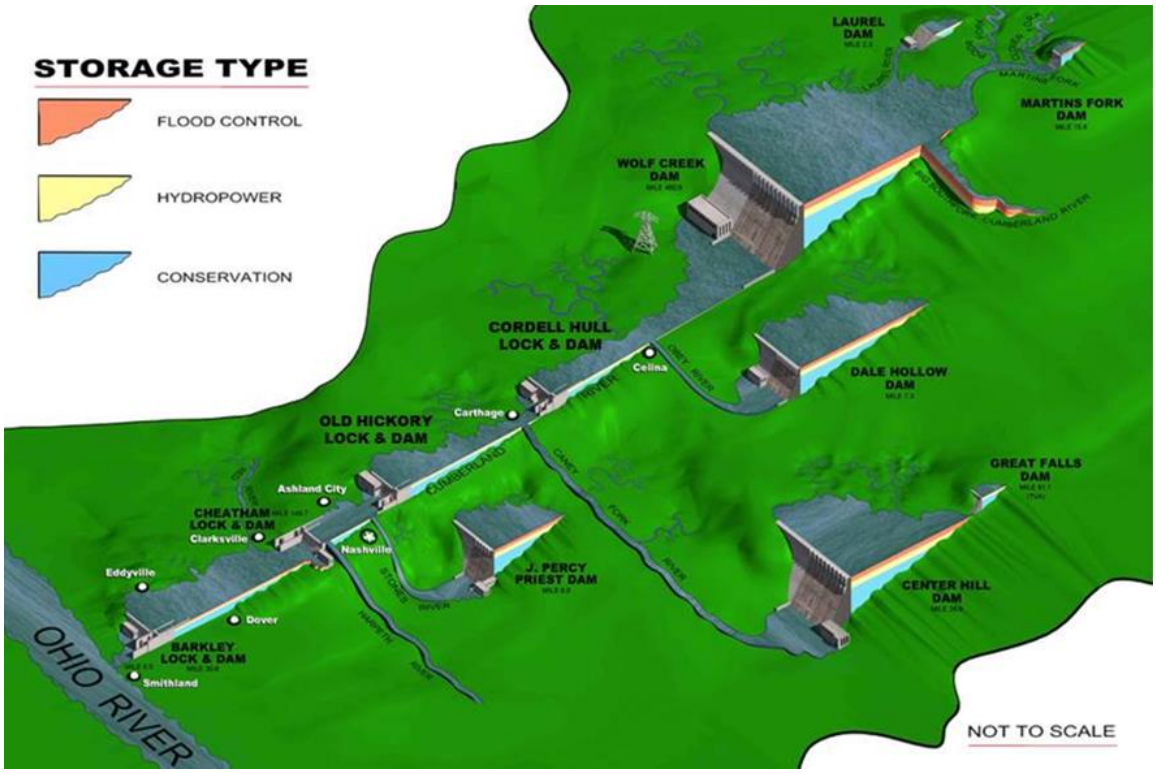


Figure I.3: Cumberland River System (courtesy of Nashville District of the U.S. Army Corps of Engineers).

toward expanding this work to a system of reservoirs by performing a necessary exploration of the feedbacks exhibited between two reservoirs connected in series. Determination of the sensitivity of downstream water quality due to changes in upstream operations is examined. Chapter VI provides concluding thoughts and proposed areas of future work.

Chapter II

STATE-OF-THE-ART LITERATURE REVIEW

Developing an optimization tool which incorporates water quality parameters requires integrating mathematical and modeling methods from several independent fields of study. An understanding of the strengths and weaknesses of available techniques in these fields, including their use in previous applications, can be gained from the following literature review.

II.1 Reservoir Modeling and Operations

In many hydropower systems, there is a desire to improve water quality outcomes by modifying operations or applying other mitigation techniques. We discuss currently employed mitigation techniques below. Reservoir modeling is an integral piece of this research, including both hydrodynamic and water quality components. There is extensive research in this area, with recent research growth due in part to improvements in computational abilities. General and reservoir-specific water quality and hydrodynamic models have various characteristics; here, we discuss the advantages, disadvantages, and applications of such models. This includes detailed coverage of CE-QUAL-W2, a two-dimensional hydrodynamic and WQM which has simulated over 2,300 surface water bodies worldwide, including over 300 manmade reservoir applications (*Portland State University*, 2007). Additionally, we discuss incorporating hydrodynamic models in power generation management systems, which attempt to optimize hydropower performance.

II.1.1 Environmental Mitigation Techniques for Hydropower Systems

Hydropower operations can negatively impact river system water quality. Impounded dams can reduce flow velocities, increase sedimentation rates upstream, reduce sediment loads downstream, and enhance erosion (*McCartney*, 2009). Stratification of water temperature and constituent concentrations may occur, reducing vertical exchanges. DO levels, water temperatures, and ensuring adequate water quality and quantity (i.e., environmental flows) for aquatic species are the primary

water quality concerns within controlled river systems (*U.S. Department of Energy*, 2016b).

Dortch (1997) states that there are three primary efforts that can improve water quality: (i) pretreatment or control of inflows, (ii) in-pool management or treatment techniques, and (iii) outflow management. Pretreating of reservoir inflows requires watershed control and land management planning, and engaging multiple stakeholders beyond river operators alone. In-pool management and treatment techniques include pumps which supply oxygenated water to the turbine penstock intakes to improve release aeration, line diffusers to increase oxygen concentrations in the forebay, disrupting or preventing stratification using water jets, sediment removal to increase volume and reduce toxicity, and aquatic plant harvesting and phosphorus inactivation by adding aluminum sulfate or sodium aluminate for algal control (*Dortch*, 1997; *U.S. Department of Energy*, 2016b). Outflow management is the most common method, as methods such as controlling outflow rates, outlet locations, and timing of releases can impact both in-pool and release water quality by influencing in-pool water quality gradients (*Dortch*, 1997; *Price and Meyer*, 1992). Outflow management methods include using temperature control devices for selective withdrawal of cold water for fisheries downstream, auto-venting turbines that add oxygen to hydropower releases, and mixing warm turbine releases with cold water bypass releases to provide a cooler downstream environment (*U.S. Department of Energy*, 2016b). Modifying dam releases has also been successful for producing flow regimes that maintain sensitive species. For example, incorporating flow pulses along the Putah Creek in California created favorable spawning and rearing conditions and maintained stable base ecological flows in order to regain native fish populations (*Poff and Schmidt*, 2016). Additional detail on in-pool management, treatment techniques, and outflow management can be found in *Price and Meyer* (1992) and *Dortch* (1997).

Studies of USACE and TVA water resources projects in the southeastern U.S. revealed significant dam tailwater quality DO issues (*Hayes et al.*, 1998; *Higgins and Brock*, 1999). In the early 1990s, TVA implemented the Reservoir Release Improvement program to improve water quality and provide a minimum constant flow at 20 TVA river system projects. DO mitigation techniques included oxygen and air injection, surface water pumping, turbine venting, oxygen line diffusion, and reregulation and aeration weirs (*Mobley and Brock*, 1995; *Higgins and Brock*, 1999). These actions resulted in reduction in the total number of days below DO targets in a year for the 16 projects with aeration improvements reduced from the historic average of 1,346 days per year to 454, 424,

231, and 267 days per year for 1994, 1995, 1996, and 1997, respectively. TVA also observed improvements in both benthic macroinvertebrate and fish communities overall. More recently, the USACE Nashville District installed a fixed-cone release valve at Percy Priest Dam on the Cumberland River (*Batick*, 2011) and Duke Energy installed aerating turbines at their Bridgewater Project in North Carolina in order to improve downstream DO levels (*U.S. Department of Energy*, 2016b).

WQMs can simulate the impacts of mitigation techniques such as the ones mentioned here, allowing managers to determine appropriate site-specific designs and operating schemes for these mitigation technologies (*U.S. Department of Energy*, 2016b). *Saito et al.* (2001) used a WQM to forecast changes in phytoplankton production due to installation of a temperature control device enabling selective withdrawal at the dam at Shasta Lake, California, and then linked this model to a food web-energy transfer model to assess impact further up the food web. The authors concluded that modeling can aid in the challenging task of predicting reservoir impacts of new dam operations. *Shirangi et al.* (2008) combined a water quality simulation model with conflict resolution theory to determine improved operational strategies for reservoir selective withdrawal. *Caliskan and Elci* (2009) used the 3D Environmental Fluid Dynamics Code (EFDC) numerical model to analyze the effect of selective withdrawal from four outlets at a reservoir in Turkey on water temperatures, as well as the impact on mixing and thermal stratification.

II.1.2 Hydrodynamic and Water Quality Modeling for Rivers and Reservoirs

The following subsections detail the early Streeter-Phelps equation model as well as a selection of 1D, 2D, and 3D hydrodynamic and WQMs that are available and described in the literature. This discussion focuses on water quality modeling capabilities, especially for DO calculation.

Streeter-Phelps

First developed in 1925, the Streeter-Phelps model describes the relationship between DO and biochemical oxygen demand (BOD). It is considered the pioneer work in the field of water quality modeling. Streeter and Phelps performed numerous studies on oxygen demand and depletion in the Ohio River (*Streeter and Phelps*, 1925) and developed the Streeter-Phelps equation:

$$D = D_0 e^{-k_a t} + \frac{k_d L_0}{k_a - k_r}\left(e^{-k_r t} - e^{-k_a t}\right) \tag{II.1}$$

where D is the DO saturation deficit, D_0 is the initial DO deficit at time $t = 0$, L_0 is the ultimate BOD, k_a is the reaeration rate, k_r is the total deoxygenation rate, and k_d is the decomposition rate (*Chapra*, 1997).

The Streeter-Phelps model ties together decomposition of organic matter and oxygen reaeration mechanisms for computation of DO in a sewage-receiving stream (*Chapra*, 1997). Without the availability of computers, model solutions were closed-form, with applications limited to linear kinetics, simple geometries, and steady-state conditions. The original model assumes only plug flow advection with no mixing occurring and only a single DO source and sink. With the advent of computers, expanded models were developed which incorporate photosynthesis, respiration, and sediment oxygen demand (*O'Connor*, 1960). *Thomann* (1963) expanded the Streeter-Phelps model to allow for multi segment systems.

QUAL

The QUAL series of models begins in the late 1960s with the development of the one-dimensional QUAL-I stream model by the Texas Water Development Board (*Brown and Barnwell*, 1987). QUAL-I simulated conservative constituents, temperature, BOD, and DO in a steady flow river (*Grenney et al.*, 1978). Tufts University and the U.S. Environmental Protection Agency (USEPA) expanded the model to add additional constituents (ammonia, nitrate, coliform, phosphate, and algae) and named the QUAL-II model (*Cox*, 2003; *Grenney et al.*, 1978). Further enhancements led to the "enhanced QUAL-II" model, or QUAL2E (*Chapra*, 1997). QUAL2E is a one-dimensional model for stream flow and water quality, capable of simulating up to 15 water quality determinants in a river and tributary system. It allows for multiple waste discharges, withdrawals, tributary flows, and incremental inflow and outflow, and can operate in steady-state or dynamic modes. When used dynamically, the effects of meteorological variations and DO diurnal variations due to algal growth and respiration can be studied, but dynamic forcing functions cannot be modeled (*Brown and Barnwell*, 1987).

Other enhanced versions now exist. QUAL2E-UNCAS adds uncertainty analysis features to

the steady-state simulation mode. Three options are available: sensitivity analysis, first order error analysis, and Monte Carlo simulation. QUAL2K 2002 (*Park and Lee*, 2002) expands the QUAL2E computational structure and adds new constituent interactions, such as algal BOD, denitrification, and DO change caused by fixed plant. Another version, QUAL2Kw, was developed by Pelletier and Chapra, modifying their QUAL2K 2003 model (of no relationship to Park and Lee's QUAL2K 2002) (*Kannel et al.*, 2011). QUAL2Kw includes the ability to model unequally spaced reaches, multiple loadings input to any reach, non-living particulate organic matter, and two forms of carbonaceous BOD (CBOD) to represent organic carbon. It also includes a GA to automatically calibrate kinetic rate parameters. The Washington State Department of Ecology used QUAL2Kw to study total maximum daily load for temperature, nutrients, DO, and pH in the Wenatchee River (*Cristea and Pelletier*, 2005), model DO in the Bagmati River in Nepal (*Kannel et al.*, 2007), model DO and pH in the Umpqua River in Oregon (*Turner et al.*, 2009), and assist in automatic calibration of the QUAL2K 2003 model for the Gangneung Namdaecheon River in Korea (*Cho and Ha*, 2010).

Delft3D

Delft3D is a an open source modeling suite for simulation in 2-D and 3-D. It contains modules for simulating flow (Delft3D-FLOW), sediment transport (Delft3D-SED), morphology (Delft3D-MOR), waves (Delft3D-WAVE), water quality (Delft3D-WAQ), and ecology (Delft3D-ECO) (*Deltares*, 2015). The modules are dynamically interfaced for data exchange and embedded in a graphical user interface. Delft3D also includes pre-processing and post-processing modules capable of preparing grid oriented data, performing tidal analysis of time series data, visualization and animation of results, and connection to ArcGIS® and MATLAB®. The hydrodynamic module calculates non-steady flow and transport based on the full Navier-Stokes equations with the shallow water approximation and can be applied to studies on salt intrusion in estuaries, lake thermal stratification, cooling water intakes, waste water outlets, transport of dissolved material, river flows, floodplains with and without vegetation, and reservoir siltation and degradation below dams. The water quality computations solve the advection-diffusion equation and include the complete natural cycles of carbon, nitrogen, phosphorus, silicon, oxygen, sediments, bacteria, salinity, temperature, heavy metals, and organic micro-pollutants. Water quality processes are formulated using linear or non-linear functions available in a library covering 140 standard substances. Constituents are considered

"passive," meaning their concentrations are assumed to have no influence on transport processes. The water quality module can be used for analyzing water balance, sewage outfalls, nutrient cycling and eutrophication, sedimentation, and recirculation of cooling water from power and desalination plants.

Most commonly, Delft3D is used for coastal and estuarial studies. *Lee and Qu* (2004) used the Delft3D-FLOW model in three dimensions to model the advective transport of red tides in the Pearl River Estuary in Hong Kong. They determined bloom initiation locations that correspond to the tidal and wind conditions during individual fish kill events in the 1998 massive red tide. *El Serafy and Mynett* (2008) modeled the hourly stratification and circulation in the Osaka Bay in Japan using Delft3D-FLOW in three dimensions and investigated improvement of daily operational forecasts of salinity and current profiles using an ensemble Kalman filter-based steady state Kalman filter (EnKF-based SSKF). *Dissanayake et al.* (2012) explored the morphodynamic response to future sea level rise using a large inlet/basin system located on the Dutch Wadden Sea.

Delft3D is less commonly applied to rivers and lakes. *Kacikoc and Beyhan* (2014) used the Delft3D flow and water quality modules to build and calibrate a WQM of a vertically well-mixed lake in Turkey. The application of Delft3D on river systems has typically been for sediment transport studies. *Edmonds and Slingerland* (2008) investigated the stability of fine-grained delta networks using the flow and morphology modules. *Bos* (2011) used the model to address the morphological effects of river sediment diversions on the final 110 km of the Lower Mississippi River, analyzing the conflicting interests of delta building and maintaining navigable waterways. He determined the best site from which to divert sediment into the delta and minimize future erosion.

Water Quality Analysis Simulation Program (WASP)

The Water Quality Analysis Simulation Program (WASP) is a dynamic compartment-modeling program for water systems, developed by the USEPA. It incorporates advection, dispersion, point and diffuse mass loading, and boundary exchange in one, two, or three dimensions (*Wool et al.*, 2002). The version 6.0 system consists of two standalone programs: DYNHYD5 for hydrodynamics and WASP6 for water quality. The basic principle behind both programs is conservation of mass, and the hydrodynamics program also conserves momentum in both time and space. Other hydrodynamic programs have been successfully linked to the WASP WQM. For example, EFDC was

used for hydrodynamic calculations and linked with WASP6 for water quality simulation in order to build a three-dimensional estuary model aimed at evaluating total maximum daily load (TMDL) scenarios (*Wool et al.*, 2003).

Water quality computations are made using kinetic subroutines, which originate from a library or can be written by the user. This ability to customize subroutines makes the WASP ideal for problem-specific models. Two subroutines are included with the version 6.0 model: TOXI and EUTRO. The TOXI subroutine models "toxic pollution," such as organic chemicals, metals, sediments, and tracers. EUTRO models "conventional pollution," including DO, BOD, nutrients, and eutrophication. More submodels have been included in the latest version (WASP7), including an advanced EUTRO (Periphyton), MERCURY, and HEAT. Early versions of WASP were capable of simulating the transport and transformation of 8 state variables, while WASP7 can simulation 10-14 state variables ("depending on how they are counted") (*Kannel et al.*, 2011). DO can be modeled at many levels of complexity depending on available information, ranging from the basic Streeter-Phelps BOD-DO relationship to a nonlinear DO balance. The WASP model has been used to analyze the influence of sediment resuspension in Lake Okeechobee (*James et al.*, 1997), study phytoplankton productivity and nutrient dynamics in a large South Carolina reservoir (*Tufford and McKellar*, 1999), assess management scenarios related to urban effluent loads in the Thermaikos Gulf (*Nikolaidis et al.*, 2006), determine the effects of aquatic macrophytes and hydropower operations on DO concentrations in a shallow tailwater reservoir (*Stansbury and Admiraal*, 2004), and predict concentrations of atrazine in Lake Michigan (*Rygwelski et al.*, 1999).

RMA2/RMA4

RMA2 and RMA4 are 2D, depth-averaged, finite-element models for hydrodynamics and water quality transport, respectively. The RMA models are part of the TABS-MD (Multi-Dimensional) Numerical Modeling System and the Surface Water Modeling System (SMS) (*Camp*, 2009). RMA2 models free-surface and sub-critical flows without regard for vertical stratification. It uses a finite element solution of the Reynolds-averaged Navier-Stokes equations for turbulent flow for both steady and unsteady problems (*Donnell et al.*, 2006). RMA4 models the advective-diffusive transport of up to 6 constituents, either conservative or non-conservative with a first order decay, and utilizes the hydrodynamics provided by RMA4 or another hydrodynamics model (*Letter et al.*, 2011). RMA4

water quality computations can be made on a 1D or 2D finite element grid.

Using RMA2, modelers have determined water levels and flow distribution around islands, flows at bridges with relief openings, flows into and out of off-channel hydropower plants and pumping plant channels, flows at river junctions, wetland water body circulation and transport, and general water surface elevations and flow patterns in rivers, reservoirs, and estuaries (*Donnell et al.*, 2006). *Crowder and Diplas* (2006) and *Stewart et al.* (2005) used RMA2 hydrodynamic models for fish habitat flow studies. Using RMA4, modelers have defined horizontal salinity distributions and intrusion, traced power plant temperature effects, calculated residence times, optimized outlet placement, identified critical areas for pollutant spills, evaluated turbidity plumes, monitored game and fish habitat water quality, and defined mixing zones (*Letter et al.*, 2011). *Xu et al.* (2008) used RMA2 and RMA4 to model and predict water quality for a Chinese tidal river network.

Environmental Fluid Dynamics Code (EFDC)

EFDC, first developed by the Virginia Institute of Marine Science at The College of William and Mary, solves "three-dimensional, vertically hydrostatic, free surface, turbulent averaged equations of motions for a variable density fluid" (*Hamrick*, 1996). The EFDC model can also be configured as a one-dimensional or two-dimensional model in either horizontal or vertical planes. It is appropriate for surface water systems, including rivers, lakes, estuaries, reservoirs, wetlands, and coastal regions (*Ji et al.*, 2002). It allows for drying and wetting in shallow areas and has the ability to simulate discharge control structures, including weirs, spillways, and culverts (*Hamrick*, 1996). The code is written in FORTRAN-77 and requires no internal source code modifications for applications to specific sites; however, since the code is in the public domain source code modifications are possible. The preprocessor generates the computational grid and interpolates bathymetry and initial conditions (salinity and temperature) based on observed data. EFDC's water quality capabilities are limited to temperature, transport of conservative substances, sediment transport, and eutrophication processes (*Ji et al.*, 2002), but the model is capable of outputting hydrodynamic solutions in formats intended for easy linkage to WQMs, such as WASP5 (*Camp*, 2009). Postprocessing capabilities include time series analysis at user specified locations, plotting, and animations.

Virginia's James and York River estuaries were the first waterbodies modeled using EFDC. For the Chesapeake Bay estuary, EFDC has simulated pollutant and pathogenic organism transport,

power plan cooling water discharges, oyster and crab larvae transport, and dredging and dredge spoil disposal alternatives (*Hamrick*, 1996). *Ji et al.* (2002) used EFDC to build a 1D hydrodynamic, sediment, and toxic model of the Blackstone River in Massachusetts, simulating concentrations of sediments and five metals over three storm events. *Jin et al.* (2002a) assessed vertical thermal and wind-driven mixing in Lake Okeechobee, Florida using a three-dimensional EFDC model. *Caliskan and Elci* (2009) also employed EFDC for a stratified reservoir, looking at selective withdrawal in a reservoir in Turkey on a 30-minute timestep. The authors determined withdrawal from the bottom of four available outlets best encouraged mixing in the water column and reduced anoxia. *Anderson* (2010) modeled Lake Elsinore in southern California in three dimensions using EFDC under the effects of a proposed pumped-storage facility for hydropower generation. The author's simulations revealed variations in surface elevation associated with pumping and generation, but limited overall effect on sediment resuspension or stratification in the lake. *Xia et al.* (2010) employed the EFDC model to simulate distributions of DO, salinity, temperature, and nutrients in the Caloosahatchee River Estuary in southwestern Florida, concluding that tidal forcing greatly influences deep layer DO concentrations in the estuary.

CE-QUAL-W2

CE-QUAL-W2 is a two-dimensional hydrodynamic and WQM used for simulating rivers, lakes, reservoirs, and estuaries since 1975. The spatial grid is laterally averaged, making it well-suited for modeling long narrow water bodies; it is not an appropriate model for water bodies with lateral water quality gradients. The model uses a finite-different approximation to laterally averaged partial differential equations for the governing equations (*Kuo et al.*, 2006). The governing equations shown below are comprised of x-momentum (horizontal momentum) (II.2), z-momentum (vertical momentum) (II.3), continuity (II.4), the equation of state (II.5), the free surface equation (II.6), and conservation of mass/heat (II.7). These six equations are shown below, where U represents horizontal velocity (m/s), W represents vertical velocity (m/s), B represents channel width, P represents pressure, τ_{xx} represents turbulent shear stress acting in the x-direction on the x-face of the control volume, τ_{xz} represents turbulent shear stress acting in the x-direction on the z-face of the control volume, α represents the channel slope angle (where slope, S_0, is equal to $\tan\alpha$), ρ represents density, q represents inflow per unit width, T_w represents water temperature, Φ represents concentration

or temperature, g represents gravitational acceleration, η represents water surface location, D_x and D_z represent longitudinal and vertical dispersion coefficients, q_Φ represents lateral inflow or outflow mass flow rate of constituent per unit volume, and S_Φ represents a laterally averaged source or sink term (*Cole and Wells*, 2007).

$$\frac{\partial UB}{\partial t}+\frac{\partial UUB}{\partial x}+\frac{\partial WUB}{\partial z}=gB\sin\alpha+g\cos\alpha B\frac{\partial \eta}{\partial x} - \frac{g\cos\alpha B}{\rho}\int_\eta^z \frac{\partial \rho}{\partial x}dz+\frac{1}{\rho}\frac{\partial B\tau_{xx}}{\partial x}+\frac{1}{\rho}\frac{\partial B\tau_{xz}}{\partial z}+qBU_x \tag{II.2}$$

$$0=g\cos\alpha-\frac{1}{\rho}\frac{\partial P}{\partial z} \tag{II.3}$$

$$\frac{\partial UB}{\partial x}+\frac{\partial WB}{\partial z}=qB \tag{II.4}$$

$$\rho=f(T_w,\Phi_{TDS},\Phi_{ISS}) \tag{II.5}$$

$$B_\eta\frac{\partial \eta}{\partial t}=\frac{\partial}{\partial x}\int_\eta^h UBdz-\int_\eta^h qBdz \tag{II.6}$$

$$\frac{\partial B\Phi}{\partial t}+\frac{\partial UB\Phi}{\partial x}+\frac{\partial WB\Phi}{\partial z}-\frac{\partial (BD_x\frac{\partial \Phi}{\partial x})}{\partial x}-\frac{\partial (BD_z\frac{\partial \Phi}{\partial z})}{\partial z}=q_\Phi B+S_\Phi B \tag{II.7}$$

CE-QUAL-W2 models physical, chemical, and biological processes including temperature, DO, nutrients, algae, and sediments. This complex dynamic model's detailed computational abilities include residence time; pH; total dissolved gases; multiple phytoplankton, zooplankton, and macrophyte groups; derived constituents including total nitrogen (TN), total Kjeldahl nitrogen (TKN), and total organic carbon (TOC); and allows users to define additional constituent subroutines to be included in the water quality algorithm (*Mooij et al.*, 2010). The model includes features that allow users to add branches and tributaries, link multiple water bodies, and incorporate various types of inflow and outflow structures. The model code is written in FORTRAN and is open-source, allowing users to make modifications as desired. The spatial grid resolution is user-defined, while the temporal resolution is determined by time stepping routines which attempt to limit numerical instability (*Cole and Wells*, 2007).

CE-QUAL-W2 has been used widely throughout the United States. In one of the earliest published applications of CE-QUAL-W2, the hydrodynamics and water quality of DeGray Lake in Arkansas were accurately simulated by CE-QUAL-W2 (*Martin*, 1988). *Adams et al.* (1997) em-

ployed a CE-QUAL-W2 model of the Cheatham Reservoir (on the Cumberland River, located downstream of Nashville, TN) to determine the impacts of combined sewer overflow (CSO) discharges, concluding that they had little influence on the DO levels in the reservoir. Using a CE-QUAL-W2 model of Shasta Lake in northern California, *Bartholow et al.* (2001) employed multivariable testing, a structured design-of-experiments method, to minimize computational expense while analyzing the potential impacts of adding a temperature control device (TCD) selective withdrawal structure. It was determined that early spring water surface elevation and reservoir storage had a much greater influence on hypolimnetic nutrient levels than the TCD. The Shasta Lake CE-QUAL-W2 model was linked to a food web-energy transfer model in order to assess the impacts of phytoplankton availability on fish (*Saito et al.*, 2001). *Deliman and Gerald* (2002) modeled the Conowingo Reservoir in the Chesapeake Bay watershed with the goal of studying sediment and nutrient trapping; they made code modifications to account for three distinct particle settling classes and incorporate scour. By comparing the results from CE-QUAL-W2 to results from a one-dimensional Hydrologic Simulation Program Fortran (HSPF) WQM, the authors concluded that CE-QUAL-W2 better matched measured DO values and performed similarly to the HSPF model for other constituents. *Bowen and Hieronymus* (2003) employed CE-QUAL-W2 with code modifications to study the impacts of nitrogen TMDL reductions on the Neuse River Estuary in North Carolina. Modifications involved inclusion of three separate algal groups (a feature later incorporated in release versions of CE-QUAL-W2), addition of a linear relationship to correlate light attenuation to salinity, and allowances for users to define algal boundary conditions as chlorophyll a concentrations rather than algal organic matter. The prediction of load reduction required to reach acceptable water quality levels as determined by CE-QUAL-W2 closely matched the results of two previous studies of this estuary, one developed using EFDC and WASP and another formulated as a Bayesian probability network model. *Debele et al.* (2008) linked CE-QUAL-W2 with a Soil and Water Assessment Tool (SWAT) model in order to simulate the Cedar Creek Reservoir and its upland watershed in Texas. After calibration, CE-QUAL-W2 was able to reproduce most observed hydrodynamic and water quality variables; however, some constituent measurements (ammonium/ammonia, total phosphorus, and total nitrogen) failed to be reproduced due to poor input data quality and propagation of errors stemming from upstream assumptions. There are numerous additional studies incorporating CE-QUAL-W2 WQMs in the United States, including those by

Garvey et al. (1998), *Annear and Wells* (2002), *Nestler et al.* (2002), *Lung and Bai* (2003), *Sullivan et al.* (2003), *Xu et al.* (2007), *Berger and Wells* (2008), *Dhar and Datta* (2008), *Wang and Yang* (2008), *Chung and Gu* (2009), *Huang and Liu* (2010), *Lee and Foster* (2013), and *Singleton et al.* (2013).

International modeling studies regularly employ CE-QUAL-W2 as well. *Kurup et al.* (2000) compared the modeling capabilities of two laterally averaged, two-dimensional models, TISAT (*Bloss et al.*, 1988) and CE-QUAL-W2, for a stratified Australian estuary. The authors determined that CE-QUAL-W2 exhibited far fewer numerical diffusion effects and better predicted surface salinity. *Kuo et al.* (2003) produced a calibrated model of the Feitsui Reservoir in Taiwan and concluded that a 50% reduction of total phosphate load would shift the reservoir's trophic state from eutrophic/mesotrophic to oligotrophic. Additionally, thermocline depths for two other stratified reservoirs in Taiwan under different climate conditions (temperature and sub-tropical climates) have been correctly predicted using CE-QUAL-W2 (*Kuo et al.*, 2006). *Chung and Oh* (2006) studied the impacts of turbidity during monsoon season on a Korean reservoir using a calibrated and verified CE-QUAL-W2 model, in anticipation of developing a real-time turbidity monitoring and modeling system. *Afshar et al.* (2011) developed an automatic calibration process and demonstrated using the Karkheh Reservoir in Iran as a case study. Other uses of CE-QUAL-W2 outside of the United States include *Gunduz et al.* (1998), *Saloranta* (2006), *Choi et al.* (2007), *Norton and Bradford* (2009), *Bonalumi et al.* (2012), and *Saadatpour and Afshar* (2013).

II.1.3 Decision Support Systems

Decision support systems (DSSs) enable decision makers to utilize available data and models in a user-friendly environment. Decision makers, including managers, engineers, and operators, are then able to compare alternatives and scenarios. DSSs for reservoir operations often include many connected modules, including database management, inflow modeling and forecasting, and monthly or real-time operation simulation and optimization (*Karamouz et al.*, 2005). These systems should be designed with the end-user in mind, and usually with the goal of a seamless transition between these underlying modules. This section describes a few of the primary general DSSs for evaluating and planning reservoir operations.

HEC-3/HEC-5/HEC-ResSim

HEC-ResSim is a generalized reservoir/river system simulation model produced by the USACE Hydrologic Engineering Center (HEC). HEC-ResSim is a component of the larger Corps Water Management System (CWMS), allowing it to be used in combination with the HEC-DSS data storage tool and other HEC models. HEC-3 and HEC-5 are predecessors of the HEC-ResSim model (*Wurbs*, 2005). HEC-3, developed in 1965-1966, simulates operation of reservoir systems for conservation purposes. HEC-5, initially released in 1973, duplicates HEC-3's capabilities with the addition of simulation of flood control capabilities for real-time operations. HEC-5 allows for variable time intervals, meaning larger timesteps may be used for normal or low flows while hourly data may be used during flood conditions. HEC-5 also has the ability to compute expected flood damages and water supply and hydroelectric power yields. A version containing one-dimensional water quality computations, HEC-5Q, can compute release requirements to satisfy downstream water quality targets (*Dortch*, 1997).

Development of HEC-ResSim began in 1996, with the latest version released in 2013 (*U.S. Army Corps of Engineers*, 2013b). HEC-ResSim allows modelers to perform project studies as well as allowing reservoir operators to monitor during real-time events. The tool is comprised of a graphical user interface, a reservoir operation simulator, data management capabilities, and tools for graphics and results reporting. The tool allows for timesteps to vary from 15 minutes to 1 day. Users can define operating goals, pool zones, release requirements, hydropower requirements, downstream control requirements (*Wurbs*, 2005), but water quality computations are not included in this tool. Employing HEC-ResSim, *Reis et al.* (2011) investigated malaria control around a reservoir in Ethiopia, *Park and Kim* (2014) analyzed the impacts of climate change on water and hydropower supply for a multipurpose dam in South Korea, *Ziaei et al.* (2012) determined monthly operating rules for a reservoir system in Iran, and *Piman et al.* (2013) looked at the impacts of future dam development in the Mekong River basin.

HEC-PRM

The Prescriptive Reservoir Model (HEC-PRM) is a network flow programming model used for determining generalized reservoir system releases based on minimizing costs "associated with various purposes including hydroelectric power, recreation, water supply, navigation, and flood control"

(*Wurbs*, 2005). HEC-PRM employs a substantially different modeling approach from HEC-3/HEC-5/HEC-ResSim and has not been as widely applied. User-supplied bounds on flows and storages are reflected as constraints, while the objective function of the network problem consists of the sum of linear approximations of penalty functions (*U.S. Army Corps of Engineers*, 2003a). HEC-PRM applications have generally used a monthly time interval for long-term planning. The model assumes future flows are known and performs computations simultaneously over all time intervals.

In an effort to address competing water users during drought conditions, USACE first developed HEC-PRM for studies of two major systems in the Missouri and Columbia River basins. The Missouri River study included six mainstem reservoirs to determine operation plans over a 90 year period of historical data (*Lund and Ferreira*, 1996). The only environmental concern included was maintenance of flows for sand bar nesting birds (*Wurbs*, 2005). Simulation modeling tested the final rules. USACE applied HEC-PRM to a review of the Columbia River basin operations at 14 reservoirs, with an objective function reflecting penalties representing hydropower, flood control, navigation, salmon and steelhead fish seasonal flows, water supply, and recreation (*U.S. Army Corps of Engineers*, 2003b). This study used gaged monthly streamflows from 1928 to 1978, adjusted to 1980 basin development conditions. *Draper et al.* (2003) and *Jenkins et al.* (2004) detail optimization of water systems in California using the California Value Integrated Network model, which includes HEC-PRM along with data from simulation models and economic values. This large model includes 51 reservoirs, 28 groundwater basins, 19 urban water demand areas, 24 agricultural economic demand areas, and 39 environmental flow locations, all modeled on a monthly timestep using historical data over 1922-1993. *Watkins and Moser* (2006) describe how HEC-PRM was used to study the operations of the Panama Canal system, analyzing the trade-off between hydroelectric power generation and navigation requirements. They also used the tool to look at the impacts of the Panama Canal expansion. Additionally, HEC-PRM enabled multiobjective reservoir operations optimization of the Upper Mississippi system of 14 reservoirs (*Faber and Harou*, 2006).

MODSIM

MODSIM is a river basin management decision support system developed by Colorado State University and the Bureau of Reclamation's Pacific North West Region (*Rani and Moreira*, 2010). It is designed for "developing improved basin wide and regional strategies for short-term water man-

agement, long-term operational planning, drought contingency planning, water rights analysis and resolving conflicts between urban, agricultural, and environmental concerns" (*Labadie and Larson*, 2007). MODSIM's graphical user interface allows for easy connection to database management components and a network flow optimization model, which contains objective function and constraints that are automatically constructed without requiring any user background in optimization or programming. The objective function provides a means to achieve system targets and demands. The flow allocation problem is modeled at each timestep of a network flow optimization problem solved with RELAX-IV, a Langrangian relaxation algorithm. Nonlinearies are handled using a successive approximations solution procedure (*Sulis and Sechi*, 2013). MODSIM includes hydropower generation capacity and production computations, as well as simulation of stochastically generated inflows and demands for use in Monte Carlo analysis. According to the version 8.1 user manual (*Labadie and Larson*, 2007), MODSIM has modeled reservoir systems in Brazil (*Srdjevic et al.*, 2004), Egypt, the Phillippines, the Dominican Republic, Korea, and extensively across the western United States, as well as the Sirvan basin in Iran (*Shourian et al.*, 2008). MODSIM is distributed as freeware online and allows for user customization and recoding in any of the several .NET languages provided with the .NET Framework.

Several studies integrate MODSIM water quantity computations with water quality objectives. *de Azevedo et al.* (2000) assessed six management alternatives for a river basin in Sao Paulo, Brazil using a combination of modified versions of the network flow allocation model MODSIM and the stream flow routing and WQM QUAL2E-UNCAS. Their study addresses both water supply (total reliability, total vulnerability, and total resiliency) and water quality (stream standard compliance reliability, water quality index, spatial uniformity of water quality, and temporal uniformity of water quality) performance measures. First the MODSIM model simulates many potential operational scenarios with respect to established priorities, and then the basin flows are input into the QUAL2E-UNCAS model to simulate concentrations of DO, BOD, total nitrogen, total phosphorus, and fecal coliform. The fidelity of this study was limited to annual quarters (temporal) and one-dimensional computations at 12 stations (spatial). *Dai and Labadie* (2001) improved this process by linking QUAL2E with MODSIMQ, a modified form of MODSIM with two additional water quality constraints. Successive relaxation is invoked to relax these additional constraints during initial estimation of the flow solution, and then these flows are input back into the QUAL2E

model and concentrations are updated. The process is iterated until convergence of water quality concentrations.

RiverWare

RiverWare is a generalized river basin modeling tool developed and maintained by the Center for Advanced Decision Support for Water and Environmental Systems at the University of Colorado Boulder (*Zagona et al.*, 2001). Its development was supported by TVA and the U.S. Bureau of Reclamation (USBR) (*Gastelum and Cullom*, 2013). It has the capability to model hydrology and hydrologic processes, hydropower production and energy uses, and water rights and account transactions (*Center for Advanced Decision Support for Water and Environmental Systems (CADSWES)*, 2015). It uses empirical relationships to model basic water quality, including total dissolved solids (TDS), DO, and temperature. Work is currently underway to include total dissolved gas (TDG) estimation within RiverWare, largely as a function of releases (*Magee*, 2015; *Witt et al.*, 2017). RiverWare includes a "point-and-click" graphical interface, allowing users to visualize and construct a network of simulation objects, linkages, and select applicable physical process algorithms for each. With computational timesteps ranging from 1 hour to 1 year, RiverWare can be applied for both scheduling and long-term planning.

RiverWare operates primarily in one of three modes: pure simulation, rule-based simulation, and optimization (*Magee*, 2015). Pure simulation involves calculating system outputs given a complete set of inputs, i.e. discharge flows. Rule-based simulation allows the user to employ prioritized if-then rules to determine solutions. These rules contain logic for operating the system and are expressed in the RiverWare Policy Language, an interpreted language developed exclusively for RiverWare (*Center for Advanced Decision Support for Water and Environmental Systems (CADSWES)*, 2015). The optimization mode employs a preemptive linear goal programming approach, which optimizes multi-objective problems with user-ranked prioritized goals formulated as soft constraints. Hydropower production is often the primary objective, which is incorporated within the algorithm as a lower-priority constraint. RiverWare linearizes nonlinear variables in order to employ a robust CPLEX linear programming solver; this means that solutions found are approximate and may be local optima, not global. A post-optimization rule-based simulation is often performed (*Magee*, 2015).

Water managers employed the RiverWare environment for release scheduling on both power and nonpower reservoirs. In 1996 TVA began performing daily scheduling modelings using RiverWare (*Zagona et al.*, 2001). Since then TVA has used the optimization routine to schedule the 35 reservoirs on the Tennessee River with as many as 800 active user specified constraints (*Biddle*, 2001; *Eschenbach et al.*, 2001). Using 6 hour timesteps over an operating forecast period of one week, the TVA RiverWare optimization model had a computational time of about 5 minutes. They additionally employ RiverWare to build hourly models when this resolution is needed. In 1996 the USBR transitioned from their Colorado River Simulation System, first developed in the 1970s, to RiverWare for long-term monthly planning on the Colorado River and nine tributaries (*Zagona et al.*, 2001). Fifty operating policy-based rules are incorporated. It also includes TDS modeling, but these calculations ignore temperature effects, precipitation, and ion exchange; additionally, reservoirs are assumed to be completely mixed throughout. The USBR also employs RiverWare for determining monthly operations on the Colorado River and daily operations on the three Lower Colorado projects (Hoover Dam, Davis Dam, and Parker Dam). RiverWare has been linked with the three-dimensional groundwater model MODFLOW and applied to the Middle Rio Grande Basin in New Mexico (*Valerio et al.*, 2010).

CalSim/WRIMS

The Water Resource Integrated Modeling System (WRIMS), formerly referred to as the California Water Resources Simulation Model (CalSim) and renamed to avoid confusion with its specific application to the California system, is a simulation model for planning and management of large river basins (*Draper et al.*, 2004). CalSim-I was developed by the California State Department of Water Resources and the USBR for application to the State Water Project and the federal Central Valley Project, and later enhanced the CalSim-II and CalSim 3.0 versions. The model employs the Water Resources Engineering Simulation Language to allow users to define the system, priorities, and operational constraints; this language is based on the Java language and structured query language (SQL) statements. Constraints may be expressed as either hard or soft. Users supply model information as text files in a defined tree structure and time series data in HEC-DSS files. Water is routed through the system network using the XA solver, a mixed integer LP solver. Because it is not a detailed operations model, CalSim cannot capture forecasts and actual operations of project

facilities; however, the flexibility of the model allows it to simulate the impacts of complex new environmental water demands (*Wang et al.*, 2011).

The CalSim-II model representing the Central Valley Project-State Water Project system includes 24 surface reservoirs and their interconnected flows. It simulates operations on a monthly timestep, including complex water right permit requirements and project sharing agreements. These include transport fish flows and water quality standards that are translated into flow equivalents. Salinity is estimated externally at four water quality stations by an artificial neural network (ANN) which has been previously trained using a one-dimensional hydrodynamic finite difference model of the channel system (*Draper et al.*, 2004).

II.2 Surrogate Modeling Techniques

Computer simulation models attempt to replicate the behavior of natural systems using physically-based mathematical equations and assumptions, when appropriate. These models are utilized in numerous problem categories, including "prediction, optimization, operational management, design space exploration, sensitivity analysis, and uncertainty analysis" (*Razavi et al.*, 2012a). The degree of realism a simulation model exhibits refers to its fidelity. Models that are considered "high-fidelity" are better able to reproduce real-world systems, but may also require a large amount of computational time. Depending on the intended application in which a model will be employed, computer models may need to be run hundreds or thousands of times; computational expense quickly becomes prohibitive (*Razavi et al.*, 2012b). Surrogate modeling methods have been developed to overcome this hurdle. Surrogate modeling, also known as metamodeling, model emulation, proxy modeling, and functional mapping, can be thought of as the creation of a "model of a model" to approximate a simulation model response. The model response surface is a function of the input variables that influence the original simulation model. Computationally expensive simulation models are models of the true environment; therefore, if the real system is considered to be a "black box" model, associated simulation models can be considered metamodels which predict the response of the original system.

Differing from response surface surrogates, lower-fidelity surrogates are simply less-detailed versions of original simulation models. They retain "the main body of processes modeled in the

original simulation model" (*Razavi et al.*, 2012a). Examples of lower-fidelity surrogates include coarse grid and large numerical time step versions of high-fidelity simulation models, which generally have fine spatial grids and small time steps. This literature review will not cover lower-fidelity surrogates; instead the focus is on response surface methods, which are not structured mathematically similar to an original model.

Surrogate models are commonly used as replacements for expensive simulation codes to be included within optimization problems. Metamodel quality is important, as metamodel-enabled optimization performance has been found to be much more dependent on surrogate accuracy than the search technique (*Johnson and Rogers*, 2000; *Zou et al.*, 2007). Metamodels can also be used to aid in model calibration, deal with noisy or missing data, and assist in determining relationships between variables and their levels of influence on a particular outcome (*Forrester et al.*, 2008). *Razavi et al.* (2012a) provides six problem characteristics that should be considered when choosing a surrogate modeling technique:

1. Whether the surrogate will be used for either searching or sampling. Search analyses include optimization problems and uncertainty-based calibration procedures.
2. Computational budget constraints. This may limit the number of original model evaluations available to construct and train a surrogate.
3. Problem dimensionality. As the number of input variables increases, surrogate modeling may become infeasible.
4. Number of outputs required. For example, multi-output surrogates are required for problems where outputs of interest vary with time and space.
5. Exact emulation versus inexact emulation. Simply put, "an emulator is a statistical approximation of a simulator" (*O'Hagan*, 2006). Should the surrogate match all training data exactly, or be a smoothed approximation?
6. Availability of original simulation model developers, as they can provide insight into surrogate performance in relation to the original model.

II.2.1 Design of Experiments

Creation of a surrogate model typically starts with a design of experiments, which will generate an initial sample of training data. The response surface will be computed to fit this set of initial data and, depending on the model form, parameter values are estimated. Space-filling strategies are employed to ensure that the set of training data captures all model behaviors within the bounds of exploration. Common techniques to produce a space-filling set are Latin hypercube sampling, symmetric Latin hypercube sampling, full factorial design, fractional factorial design, and central composite design. For a large number of design variables, deterministic methods (e.g., full factorial design, fractional factorial design, and central composite design) may become computationally expensive. Random methods (e.g., Latin hypercube sampling and symmetric Latin hypercube sampling) can be scaled up to accommodate a large number of design variables, lessening computational expense (*Razavi et al.*, 2012a).

The selection of training data depends on the original model. If a surrogate is being used to replicate field data, a space-filling sampling plan can be implemented from the onset. In the case of a high-fidelity computer model, multiple runs may be required in order to achieve an adequate set, and even then there is no guarantee that the set will be space-filling. The size of the training data set is important; if the set is too large computational savings are diminished, but if the set is too small it may not capture detailed behavior of the original model. Search spaces can become very large for high-dimensional problems, resulting in a large number of training points to cover the space sufficiently (*Razavi et al.*, 2012a). *O'Hagan* (2006) provides a comparison of a 25-D space versus a 5-D space, noting that 200 training points will lead to sparse coverage and dense coverage for each, respectively.

The minimum number of training points required as well as the maximum number of training points that will still allow feasibility are partly determined by the function approximation technique (*Razavi et al.*, 2012a). Techniques that require as many correlation functions as training points, such as kriging, radial basis functions (RBFs), and Gaussian emulator machines (GEMs), become computationally expensive as the training set grows. GEM applications suffer from this the most, but it is also especially true for kriging, in which the determination of correlation parameters is performed by maximum likelihood estimation. Design sites in kriging applications are "typically less

than a few thousand" (*Razavi et al.*, 2012a). RBFs can handle a larger number of training points, but the correlation parameter tuning process may become computationally challenging (*Razavi et al.*, 2012a). ANNs are capable of handling a very large number of training sites; for example, *Broad et al.* (2005) used 10,000 data points to calibrate an ANN surrogate for a water distribution system simulation.

Dimensionality also plays a role. *O'Hagan* (2006) notes that there is little coverage in the literature related to high-dimensional kriging surrogates used in practice, but that kriging metamodeling can likely be employed effectively on current computing platforms for problems up to 50-D. *Jones et al.* (1998) found that at least $n = 10k$ space-filling initial points, where k is the dimension size, are necessary for kriging and RBF models; however, *Sóbester et al.* (2005) notes that "rules of thumb" such as this have not been rigorously proven and that (in the context of employing surrogate models in optimization frameworks) "to date there is no clear understanding of how this figure should be chosen and what influence the choice has on the performance of the optimizer." *Sóbester et al.* (2005) concluded from numerical experiments using an uncertainty-based, metamodel-enabled optimizer that an initial sample size between 35% and 60% of the total computational budget is appropriate. If the size is too large, points are extraneously placed in a space-filling manner (rather than in regions of interest). If the size is too small, the results of an expected improvement-based objective function become nearly meaningless. *Razavi et al.* (2012b) suggest employing a screening method for high-dimensional problems in which the design space is screened to "identify and remove decision variables that are less important." Unfortunately, this process can be difficult and may decrease approximation accuracy if relevant parameters are fixed via screening.

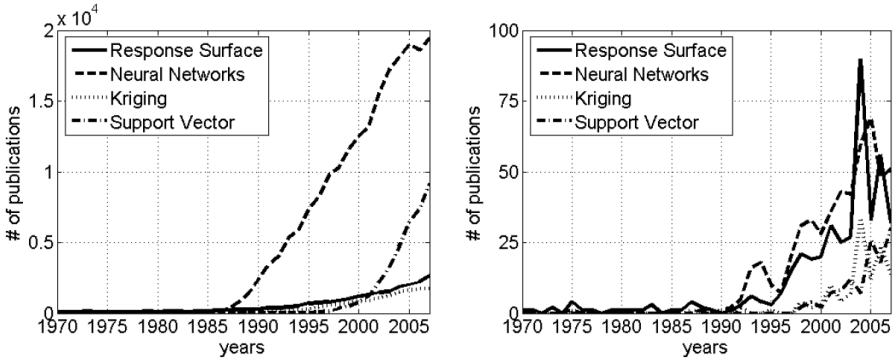
II.2.2 Function Approximation Models

Response surface surrogate modeling encompasses numerous techniques, which fall under the main categories of exact and inexact emulators. An exact emulator fits training sites exactly with no error, while inexact emulators allow for smoothing of noisy data sets. Typically, inexact emulator models are suitable for replicating physical experiments, which tend to have some element of random noise, while exact emulators are appropriate for approximating deterministic computer models (*Razavi et al.*, 2012a). *Viana and Haftka* (2008) searched the Publish or Perish software system and

Google Scholar databases to determine how the number of publications related to surrogate modeling has changed over time. Figure II.1(a) shows their findings over all research fields, while Figure II.1(b) narrows the research field to just the optimization arena. In their study, "response surface" refers to polynomial response surface methods. While support vector regression and ANNs are the most dominant published forms for surrogate modeling overall, in optimization problems all techniques are fairly equal in number in the literature as of the year 2008. A later update of this study of the literature revealed the continuation of these trends (*Viana et al.*, 2014). These four commonly-employed categories of function approximation models are discussed in detail in this section, in addition to radial basis function models, which are closely related to kriging, and Shepard's method for inverse distance weighting, as it can be engaged as a surrogate model. Relevant applications of these and other surrogate models in the water resources literature are covered in section II.2.4.

Polynomial Response Surface Models

Box and Wilson (1951) introduced the earliest work in response surface surrogates. In their classic paper, they developed a process to find optimal operating conditions for chemical production using polynomial functions to estimate output dependent on several input variables. Their work has become the basis of response surface methodology. Other techniques that typically incorporate polynomial models as function approximations, including Taylor series expansion and trust-region methods, can be thought of as early applications of the response surface concept (*Razavi et al.*, 2012a).



(a) Major surrogate techniques in all areas

(b) Major surrogate techniques in Optimization.

Figure II.1: Evolution of surrogate modeling publications (*Viana and Haftka*, 2008).

An m-order polynomial approximation of the true response f as a function of sampling points $\mathbf{X} = \{x^{(1)}, x^{(2)}, \ldots, x^{(n)}\}^\intercal$ is written as

$$\widehat{f}(m, x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \cdots + w_m x^m = \sum_{i=0}^{n} w_i x^i \tag{II.8}$$

Using the true response vector $\mathbf{y} = \{y^{(1)}, y^{(2)}, \ldots, y^{(n)}\}^\intercal$, the vector of weights can be determined by least squares (*Forrester et al.*, 2008). Other function forms can be used, including exponentials (*Blanning*, 1975), but polynomials are most common due to their simplicity, minimal expense, and clarity of parameter sensitivity (*Fen et al.*, 2009). Since prediction errors can occur at training data locations, polynomial surrogates are inexact emulators; however, if prior knowledge suggests that the original function may be of a similar form to a polynomial, it becomes be a strong option (*Razavi et al.*, 2012a). Polynomial models are typically not applicable to models with more than 10 input variables or when the response surface is highly nonlinear (*Simpson et al.*, 2001). Non-linear, multi-model, multi-dimensional design landscapes are often encountered in engineering problems. The ranges of variables can be reduced through trust-region methods, but for highly dimensional problems obtaining the amount of data necessary to estimate high-order polynomial terms may not be viable (*Forrester and Keane*, 2009).

Modelers are tasked with selecting the polynomial order size, m. *Razavi et al.* (2012a) state that second-order polynomial functions are the most popular order size employed as response functions; however, greater values of m generate more accurate predictions, but may overfit noisy data if too many terms are allowed. *Forrester et al.* (2008) suggest using cross-validation to determine an appropriate value for m. Cross-validation involves splitting the training data into several equal subsets, removing each subset individually, fitting the model, and determining prediction errors at all input locations. This process is performed for several values of m, and the value with the lowest prediction error is chosen. More information about cross-validation can be found in the work of *Viana et al.* (2010).

Inverse Distance Weighting (Shepard's Method)

Shepard's method is an inverse distance weighting method for construction of global interpolations "by blending local interpolants using local-support weight functions" (*Thacker et al.*, 2010). It

is useful for constructing interpolations from irregularly spaced data points. In his paper introducing the original form of the method, Shepard states the desire to develop a smooth two-dimensional interpolation function, meaning the response surface is continuous and once differentiable. He concludes that this method is generalizable to higher dimensional spaces. Shepard also notes that "the function should be suitable for computer application at reasonable cost" (*Shepard*, 1968).

The original Shepard algorithm is a local method characterized as weighted sums of local approximations f_k with weights $W_k(\mathbf{x})$ that when normalized as a set form a partition of unity. The overall support is considered local because the weight functions have local support; in other words, they are nonzero near the region of interest and go to zero at farther distances. For a set of irregularly-spaced data points $\{\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^n\}$ and associated scalar values f_i for each point, an interpolated approximation for the underlying function can be written as

$$\widehat{f}(\mathbf{x}) = \frac{\sum_{k=1}^{n} W_k(\mathbf{x}) f_k}{\sum_{k=1}^{n} W_k(\mathbf{x})} \tag{II.9}$$

where the weight functions are defined by

$$W_k(\mathbf{x}) = \frac{1}{\|\mathbf{x} - \mathbf{x}^{(k)}\|^p} \tag{II.10}$$

where typically $p = 2$, but can be set to other values (*Thacker et al.*, 2010). Weight functions can be written in various forms, including a Gaussian form of

$$W_k(\mathbf{x}) = e^{-\|\mathbf{x} - \mathbf{x}^{(k)}\|^2/(2\sigma^2)} \tag{II.11}$$

as used for applications in *Fasshauer* (2007). The original form of Shepard's method's benefits include implementation simplicity, no required parameters to be tuned, ability to work in any dimensional space, and capability to interpolate scattered data on any grid and with coinciding nodes. Deficiencies include slow performance with large datasets and large weights for distant nodes in high-dimensional spaces (*ALGLIB*, 2014).

Franke and Nielson (1980) propose a modified Shepard's method which allows for greater local support and replaces the nodal values (f_k) with a local approximation function $P_k(\mathbf{x})$. Weight

functions for the modified Shepard's method can be written as

$$W_k(\mathbf{x}) = \left[\frac{(R_w^{(k)} - \|\mathbf{x} - \mathbf{x}^{(k)}\|)_+}{R_w^{(k)} \|\mathbf{x} - \mathbf{x}^{(k)}\|} \right]^2 \tag{II.12}$$

where the constant $R_w^{(k)}$ is a radius about the point $x^{(k)}$ in which training points are allowed to influence prediction. Franke and Nielson suggest using the relationship $R_w = \frac{D}{2}\sqrt{\frac{N_w}{n}}$ where D is the maximum Euclidean distance between any two data points and N_w is a positive integer parameter that must be tuned. *Renka* (1988) tested several variations of Shepard's method and tuned this parameter by testing values of N_w, seeking to minimize error. Additional parameter considerations are required for non-constant values of $P_k(\mathbf{x})$, such as polynomial functions (*Thacker et al.*, 2010). Modified Shepard's method improves performance for large datasets and eliminates "flat spots" near nodes when combined with a polynomial function, but computational expense may increase for high-dimensional spaces (above 5) (*ALGLIB*, 2014).

Radial Basis Function (RBF) Models

RBF models approximate smooth, continuous functions as a combination of weighted symmetrical basis functions. *Sóbester* (2003) relates this process to synthesizers which imitate the sounds of various musical instruments by weighting a combination of tones. Bases are centered at training points in the space, resulting in interpolated outcomes. Assuming data is noise-free, as is the case when data is collected from deterministic computer simulations, an approximation of the true response f as a function of sampling points $\mathbf{X} = \{x^{(1)}, x^{(2)}, \ldots, x^{(n)}\}^\intercal$ is written as

$$\widehat{f}(\mathbf{x}) = \mathbf{w}^\intercal \Psi = \sum_{i=1}^{n_c} w_i \psi(\|\mathbf{x} - c^{(i)}\|) \tag{II.13}$$

where n_c is the total number of basis centers, $c^{(i)}$ is the ith basis function center, ψ are the basis functions, and Ψ is a vector containing basis function values evaluated at the Euclidean distance between prediction sites and centers (*Forrester et al.*, 2008). RBF models can be augmented by adding a polynomial term to equation (II.13), which may provide additional global support (*Elsayed et al.*, 2012). Basis functions can be of many mathematical forms, including linear, cubic, and thin plate spline. Gaussian, multiquadric, and inverse multiquadric basis functions can provide better sensitivity, but require the estimation of additional parameters to specify the spread of basis

function influence. Gaussian basis functions allow modelers to easily estimate prediction error at any location, making them a popular choice. Since basis functions are symmetric in all directions, RBF models treat all influencing variables equally; to eliminate the influence of varying variables units and scales, input data is generally normalized to a [0,1] interval (*Razavi et al.*, 2012a).

The weights vector is computed by $\mathbf{w} = \mathbf{G}^{-1}\mathbf{y}$, where $\mathbf{G}$, the Gram matrix, is defined by $\mathbf{G}_{i,j} = \psi(\|\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\|)$ for $i, j = 1, \ldots, n_c$ (*Forrester et al.*, 2008). If two training points in the set are very closely located to each other, $\mathbf{G}$ may become ill-conditioned (*Micchelli*, 1986) and the computation of the weights vector becomes numerically unstable. The correct estimation of the weights vector $\mathbf{w}$ allows the model to accurately simulate at training point locations, but it is also important to carefully set additional parameters in order to minimize errors in the remainder of the design space. This can be performed by finding the parameters that produce the minimum error estimate during cross-validation (*Forrester et al.*, 2008).

Gaussian Basis (Kriging) Models

The kriging model method, also known as Gaussian process modeling, was first developed by and named after Danie Krige, a South African mining engineer who used the method to estimate gold ore spatial patterns (*Krige*, 1951). The kriging model consists of a combination of localized basis functions, also known as correlation functions. The most commonly used is an exponentially decaying correlation function of the form

$$\psi^{(i)} = e^{-\sum_{j=1}^{k} \theta_j |x_j^{(i)} - x_j|^{p_j}} \tag{II.14}$$

where k is the number of input variables and θ_j are correlation or width parameters (*Simpson et al.*, 2001). The kriging basis function above is mathematically similar to the Gaussian RBF form, with two notable differences. The vector $\theta = \{\theta_1, \theta_2, \ldots, \theta_k\}^\intercal$ of correlation parameters allows each variable to have a unique basis function width parameter, and p_j is a "smoothness" parameter than can be tuned. Larger values of the correlation parameter θ_j result in extended influence, and by comparing values a dominant input variable can roughly be inferred (*Forrester et al.*, 2008). By allowing independent correlation parameters for each input dimension, sensitivities to units of measurement are negligible. This suggests that normalizing input data to unity is not as important in a

kriging model as it is for RBF models (*Jones*, 2001). *Razavi et al.* (2012a) suggest that large correlation parameter values indicate nonlinear behaviors in that particular dimension and small values indicate a smooth function with minimal variances. Larger values of p_j increase the smoothness of the Gaussian basis curves, while very small values suggest no correlation between a point and its neighboring space; in other words, the function is discontinuous at this location. When all values of p_j are fixed at 2 and all values of θ_j are equal, the kriging basis function is the same as the Gaussian (*Forrester et al.*, 2008). Considering this, kriging models can be either exact or inexact emulators depending on parameter choice (*Elsayed et al.*, 2012).

The kriging method treats interpolated outcome values as regionalized variables, which have characteristics of both random and deterministic variables. Regionalized variables continuously vary in space, assuming that points near each other are spatially correlated and points far from one another are statistically independent (*Elsayed et al.*, 2012). The kriging prediction function is written as

$$\widehat{y}(\mathbf{x}) = \widehat{\mu} + \boldsymbol{\psi}^{\mathsf{T}}\boldsymbol{\Psi}^{-1}(\mathbf{y} - \mathbf{1}\widehat{\mu}) \tag{II.15}$$

where $\widehat{\mu}$ is the expected mean value, $\boldsymbol{\psi}$ is a vector of correlations between training data and the prediction, Ψ is the correlation matrix, and y is the vector of observed sample values. A detailed derivation of (II.15) can be found in *Forrester et al.* (2008). Like RBF models, kriging models may be augmented with a polynomial function to provide additional global support; this is often taken to be a constant term, as shown above in (II.15) (*Srivastava et al.*, 2004). In total, the model has $2k+2$ parameters: $\widehat{\mu}$, $\widehat{\sigma}^2$, $\{\theta_1, \theta_2, \ldots, \theta_k\}$, and $\{p_1, p_2, \ldots, p_k\}$. These can be computed by maximum likelihood estimation (*Elsayed et al.*, 2012); however, due to the expense of estimating the correlation and smoothness parameters, kriging is most useful for cases where the original simulation model is exceptionally computationally intensive (e.g., computational fluid dynamics models) (*Forrester and Keane*, 2009).

The kriging method "treats the deterministic response of a computer model as a realization of a stochastic process, thereby providing a statistical basis for fitting" (*Razavi et al.*, 2012a). The estimated mean square error for a kriging model at a location x in the design space can be computed by

$$s^2(\mathbf{x}) = \sigma^2 \left[\mathbf{1} - \psi^\intercal \Psi^{-1} \psi + \frac{\mathbf{1} - \mathbf{1}^\intercal \Psi^{-1} \psi}{\mathbf{1}^\intercal \Psi^{-1} \mathbf{1}} \right] \tag{II.16}$$

as given in *Forrester and Keane* (2009). This allows kriging models to be easily used for approximating uncertainty at any given point in the design space, which makes it a popular choice for surrogate-based optimization.

Support Vector Regression (SVR)

Support vector machine (SVM) theory was first developed at AT&T Bell Laboratories in the 1990s, making it a newer family of methods (*Forrester and Keane*, 2009). SVM is traditionally a classification approach rather than a method for function approximation (*Basudhar et al.*, 2012). Methods have been developed from SVM theory that can be used for approximation, including support vector regression (SVR). SVR can be thought of as an extension of RBF and kriging methods due to many similarities (*Forrester et al.*, 2008).

SVR models incorporate a margin ε in which errors are acceptable in the sample data, and these errors are not allowed to affect predictions. Training points within the $\pm\varepsilon$ band, also called the ε-tube, are ignored for prediction. The predictor is defined only by exterior points and points on the region boundary; these training points form support vectors (*Forrester et al.*, 2008). SVR's ability to reduce noise sensitivity makes it useful for noisy models and inexact emulation (*Razavi et al.*, 2012a). SVR models also incorporate a user defined constant C, which determines the linear rate of influence loss for points outside of the ε-tube (*Forrester et al.*, 2008).

The SVR prediction formulation is similar to that of the kriging model, consisting of the sum of weighted basis functions and the bias term μ. Basis functions are also referred to as kernels in SVM literature; popular choices include linear, d degree homogeneous polynomial, d degree inhomogeneous polynomial, Gaussian, and kriging. A lengthy derivation involving constrained convex quadratic optimization and introduction of Langrange multipliers results in a prediction function of the form

$$\widehat{y}(\mathbf{x}) = \mu + \sum_{i=1}^{n} (\alpha^{+(i)} - \alpha^{-(i)})(\mathbf{x}^{(i)} \cdot \mathbf{x}) \tag{II.17}$$

Basis functions of various forms are incorporated via space mapping and kernel substitution, and

support vectors can be found by forming a dual variable optimization problem. The bias term μ can be computed through exploiting the idea that at the solution of the dual variable optimization problem the products between dual variables and constraints go to zero; this is one of the Karush-Kuhn-Tucker conditions for optimality. The user-defined constant C governs "trade-off between model complexity and the degree to which errors larger than ε are tolerated" and can be computed by testing values of varying orders of magnitude and selecting the one with the lowest resulting RMSE. C can be sensitive to the scaling, so the values in $\mathbf{y}$ should be normalized to unity. To properly assign ε, the source of data must be considered. The precision limits of measurement can be used for ε if training data comes from physical experiments, but for data stemming from deterministic computer simulations ε can be calculated by using the v-SVR technique (*Forrester et al.*, 2008). The two parameters ε and C are mutually dependent, meaning a change in one may influence the effect of the other on prediction (*Razavi et al.*, 2012a).

SVR is a powerful prediction method for large, high-dimensional data sets, but due to the method being relatively young there is little implementation of its use in engineering design in the literature. Another possible reason for its limited use is the lack of large amounts of data in some high-dimensional engineering design problems. In these cases, it may be necessary to use all available data for model training, and SVR's fundamental idea of incorporating data subsets becomes unattractive. Also, SVR training time is longer than other surrogate methods, making SVR models difficult to implement in problems that involve surrogate refinement within an optimization loop (*Forrester and Keane*, 2009).

Artificial Neural Networks (ANNs)

Feedforward ANNs are flexible tools for function approximation composed of neurons assembled into a multi-layer architecture. They have been used for a variety of complex problems including speech and handwriting recognition, face recognition, currency exchange rate prediction, chemical processes optimization, cancerous cell identification, and spacecraft trajectory prediction (*Cheng and Titterington*, 1994). The neurons are multiple linear regression models with a nonlinear transformation on $\mathbf{y}$. If input variables to each neuron are given by $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$, then the predicted output can be written as

$$y = \frac{1}{1 + e^{-\eta/T}} \tag{II.18}$$

where $\eta = \sum_{i=1}^{n} w_i \mathbf{x}_i + \beta$. β represents the "bias value" of a neuron, T is a user-defined slope parameter, and w_i are model weights (*Simpson et al.*, 2001).

There are two main steps in constructing an ANN. First the architecture must be specified, and secondly the network must be trained. Modelers specify the model architecture through several parameters, including the number of hidden layers, number of neurons in each hidden layer, and the form of transfer functions. These decisions can be subjective, but processes have been developed for structure development. These include methods based on growing or pruning strategies, network geometrical interpretation, and Bayesian statistics. Unfortunately, these methods can be computationally extensive as they involve testing a variety of network structures; considering this, the appropriate architecture of ANN applications in the literature are generally decided by trial-and-error (*Razavi et al.*, 2012a). The architecture parameters are combined, ANN models are trained for these network configurations, and the architecture resulting in the lowest error metric measured on the test set is chosen (*Liong et al.*, 2001; *Zou et al.*, 2007; *Shrestha et al.*, 2009). As in all modeling approaches, the smallest architecture with an acceptably low error should be used to minimize computational expense, both during training and prediction. Networks involving "tens of thousands of parameters" have been successfully built, but data management and calculation of model parameters can be very expensive. Once the architecture is defined, model weights are determined when a training process converges upon minimized validation errors; this is often performed by back-propagation (*Simpson et al.*, 2001). Training is typically performed multiple times, as there may be many sets of weights that can represent the training data satisfactorily.

ANNs can be used as inexact emulators for noisy data sources or exact emulators for deterministic computer code. With a large enough structure ANNs can perform exact emulation of deterministic code, but this may lead to poor performance in unsampled areas of the design space and a risk of overfitting (*Razavi et al.*, 2012a). Considering this, ANNs are more suitable for physical experiments than deterministic experiments (*Razavi et al.*, 2012b). *Tamura and Tateishi* (1997) proved theoretically that ANNs with two hidden layers require fewer hidden neurons to perform as exact emulators as compared to ANNs with only one hidden layer; however, in a review of response

surface modeling literature *Razavi et al.* (2012a) conclude for water resources applications single hidden layer ANNs are most popular. ANNs are capable of handling large amounts of training data and it is generally believed that more input data results in a better-generalized model; however, large amounts of data can require additional computational time for training and may trap the training process at a local (rather than global) solution (*Zou et al.*, 2007). Finally, it should be noted that some references, including the MATLAB® Neural Network Toolbox, consider RBFs as a type of feedforward ANN (*Razavi et al.*, 2012a).

II.2.3 Analysis Frameworks

Once a surrogate model is built, it can be utilized in frameworks of various types. There are four main "families" of surrogate-enabled frameworks, and each specific type may only be applicable for certain uses (i.e., searching versus sampling) (*Razavi et al.*, 2012a). Framework development is an important step in the utilization of surrogate models for practical problems and must be considered in the initial planning stages, because certain surrogate-enabled frameworks can be more easily implemented using specific surrogate model forms. A key feature in the frameworks discussed below is search point selection method, which can be performed as one-stage or two-stage. Most current approaches employ two-step search point methods (*Jones*, 2001).

Basic Sequential Framework (Off-Line)

The simplest analysis framework which employs metamodels is the basic sequential framework. It can also be referred to as an off-line framework because the metamodel requires no updating during analyses. This framework follows a three step process:

1. Develop a design of experiments in which a predetermined number of samples are taken throughout the feasible space and, in the case of a search analysis, objective function values at each location are evaluated by the original simulation model.
2. A surrogate model is built and parameters are tuned.
3. The surrogate model can be substituted in place of the original simulation model for performing time-intensive analyses.

Since the majority of computational budget is allocated during the design of experiments, the number of locations sampled initially is much higher than in the more-advanced frameworks discussed

later. This one-stage training point selection method can provide a globally stronger surrogate model initially, but the model may not accurately represent the original model in regions of interest. This could lead to failure in both search and sampling applications (*Razavi et al.*, 2012a). In order to avoid poor performance in regions near optimal conditions, *Bliznyuk et al.* (2008) narrowed the search region by applying optimization techniques directly on the original model, and then fit a surrogate model only in the local optimal region. While this may be beneficial for off-line problems where global accuracy is not required, applying optimization procedures on original simulation models may not be computationally feasible.

Adaptive-Recursive Framework

The adaptive-recursive framework is similar to a basic sequential framework, with the addition of surrogate refinement using a two-stage point selection process. This framework also follows a three step process:

1. Develop a design of experiments in which a predetermined number of samples are taken throughout the feasible space and, in the case of a search analysis, objective function values at each location are evaluated by the original simulation model.
2. A surrogate model is built and parameters are tuned.
3. Identify regions of interest using a search or sampling algorithm, sample additional points in this region using the original simulation model, and repeat Steps 2 and 3 until convergence is reached.

When used for optimization searching, the best point found during the framework process is generally considered the final optimal solution (*Razavi et al.*, 2012a). *Zou et al.* (2007) employed an adaptive strategy for ANN-enabled optimization of a water quality modeling problem, citing previous linked ANN optimization studies which failed to perform well under off-line sampling. While the adaptive-recursive framework seeks to address the drawbacks of the off-line method, there are cases where this method may fail to find solutions in the true function optimal region (*Jones*, 2001). This may result in situations where new sampling points are added in close proximity to preexisting training points (thereby adding no additional knowledge for response surface training) or may converge to local optimal solutions.

Metamodel-Embedded Evolution Framework

The metamodel-enabled evolution framework is similar to the adaptive-recursive framework but is designed for use with evolutionary optimization procedures. With this method, an initial sampling plan stemming from a formal design of experiments is not required. Rather, first a population-based optimization algorithm such as a GA is used for several generations, computing function values from the original simulation model. These data points are used to fit a surrogate model. In all subsequent generations, individuals are evaluated by either the surrogate or the original model using a pre-defined process, which has been termed evolution control by *Jin et al.* (2002b). Jin explains that this can be performed two ways: either by designating a certain number of individuals (called controlled individuals) within each generation to be evaluated using the original fitness function, or to introduce controlled generations in which all individuals in that generation are evaluated by the original fitness function. All other individuals are evaluated by the surrogate model. Depending on the approach taken, modelers must decide either the number of controlled individuals or controlled generations; the process can be made further complex by adaptively changing these parameters as the optimization algorithm progresses. The surrogate model is refitted occasionally as training points are added to the set. In order for an optimization process to find global optima under this framework, the evolutionary algorithm chosen must be a global optimizer and any individual in any generation should have some probability of being solved through the original simulation model. Otherwise, failure modes similar to those occurring in an adaptive-recursive framework are possible (*Razavi et al.*, 2012a). It is also important that the initial collection individuals are well-distributed and approximate the response surface well, as all following generations are conditioned from this set of individuals. If this is not fulfilled, the evolutionary optimization algorithm may fail to find a global solution (*Broad et al.*, 2005).

Approximation Uncertainty-Based Framework

The approximation uncertainty-based framework relies on the basic shell of the adaptive-recursive framework while incorporating surrogate model uncertainty in the sampling decision process. This method has been extensively used in structural (*Bichon et al.*, 2013; *Sóbester et al.*, 2005), aerospace (*Basudhar et al.*, 2012; *Queipo et al.*, 2005), manufacturing (*Boukouvala and Ierapetritou*, 2013; *Chen et al.*, 2012; *Huang et al.*, 2006), and petroleum engineering (*Horowitz et al.*, 2010; *Queipo*

et al., 2002) fields, but with the exception of the work of *Mugunthan and Shoemaker* (2006) and (*di Pierro et al.*, 2009) it has not been well-employed in the water resources arena. While the adaptive-recursive framework assumes surrogate approximate values to be true, this may not be so in many regions of the design space, including at globally optimally regions. This technique relies on an approximation uncertainty quantity, which is readily available in certain surrogate forms including kriging and Gaussian RBF models. The three steps involved in this framework are:

1. Develop a design of experiments in which a predetermined number of samples are taken throughout the feasible space and, in the case of a search analysis, objective function values at each location are evaluated by the original simulation model.
2. A surrogate model is built and parameters are tuned.
3. Optimize a new surface function, which balances a desire to minimize model uncertainty and find globally optimal results.

The third step aims to balance exploration and exploitation (*Razavi et al.*, 2012a). Different methods have been developed to perform the third step, but the maximization of an expected improvement function (EIF) approach can be considered the most advanced. An EIF can be used to select training data to be added to the surrogate model of optimization results by calculating the "expectation that any point in the search space will provide a better solution than the current best solution based on the expected values and variances predicted" by the current surrogate model (*Bichon et al.*, 2013). The EIF at any location x for a kriging metamodel prediction can be expressed as

$$EI(\mathbf{x}) = \left(f(\mathbf{x}^*) - \mu_{\widehat{f}}(\mathbf{x})\right)\Phi\left(\frac{f(\mathbf{x}^*) - \mu_{\widehat{f}}(\mathbf{x})}{\sigma_{\widehat{f}}(\mathbf{x})}\right) + \sigma_{\widehat{f}}(\mathbf{x})\phi\left(\frac{f(\mathbf{x}^*) - \mu_{\widehat{f}}(\mathbf{x})}{\sigma_{\widehat{f}}(\mathbf{x})}\right) \tag{II.19}$$

where $f(\mathbf{x}^*)$ is the current best function value located at $\mathbf{x}^*$ found by the optimization routine, $\mu_{\widehat{f}}(\mathbf{x})$ is the mean of the kriging prediction at $\mathbf{x}$, $\sigma_{\widehat{f}}(\mathbf{x})$ is the standard deviation of the kriging prediction at $\mathbf{x}$, and Φ and ϕ are the standard normal cumulative distribution and probability density functions. A global optimization routine must be used to determine the maximum of the EIF; the branch-and-bound algorithm (*Jones et al.*, 1998), the DIRECT method (*Bichon et al.*, 2013), and GAs (*di Pierro et al.*, 2009) have been used successfully for this application.

Developed by *Jones et al.* (1998), the efficient global optimization (EGO) algorithm is a commonly-

used optimizer which utilizes an EIF for sampling point search. EGO works well when the function shape and smoothness are generally well-estimated from an initial collection of training points; however, if this is badly approximated due to poorly distributed design sites, the process may converge slowly or prematurely stall (*Jones*, 2001; *Razavi et al.*, 2012a). EGO will not attempt to add training points identical to those already in the set, but as the optimizer converges there is potential to create an ill-conditioned correlation matrix in the kriging model due to newly-added points being located near previously sampled points in the training set. This can be overcome by using an uncertainty ratio to remove points that are deemed "too close" to other points or employing a "layering" method which "uses separate kriging models for short and long correlation lengths" (*Bichon et al.*, 2013). The EGO algorithm's initial formulation is intended for single objective optimization, but it has been extended to perform multiobjective optimization as well. ParEGO (*Knowles*, 2006) does this by applying weighting factors to aggregate all objectives into a single function, SMS-EGO (*Ponweiser et al.*, 2008) incorporates multiple surrogates to simulate multiple objectives, and Shinkyu and Obayashi's multi-EGO procedure embeds a multiobjective GA into an EGO-based framework (*Shinkyu and Obayashi*, 2005).

II.2.4 Response Surface Surrogate Usage in Water Resources

Just as *Viana and Haftka* (2008) found in their literature search over all fields, earlier applications of metamodeling in water resources generally incorporated regression or ANN models. Kriging, RBF, and SVM models have gained popularity in recent years, as well as the combination of multiple surrogate model forms. Surrogate models have been employed in water resources applications for various purposes, with the two primary purposes being to aid in calibration parameter selection and for use within optimization routines for operations and design. Automatic calibration applies an optimization algorithm to an objective function which aims to minimize the error between predictions and measured values (*Shoemaker et al.*, 2007). Automatic calibration can be superior to traditional "trial-and-error" methods, which can be inefficient, oversubjective, and unreliable (*Zou et al.*, 2007). Surrogate models have also been used within optimization routines as replacements for high-fidelity models, which are sometimes necessary for computing constraint and objective function values.

Surrogates in Automatic Calibration Procedures

The majority of water resources publications using metamodels to aid automatic calibration routines have been designed for watershed models. *Liong et al.* (2001), *Khu and Werner* (2003), and *Khu et al.* (2004) used ANN metamodels in automatic calibration procedures to find optimal parameter values for the rainfall-runoff models HydroWorks, the Storm Water Management Model, and MIKE 11/NAM, respectively. These procedures use feedforward ANNs to estimate the response of the catchment model, allowing for faster search by GA of the parameter space. In both *Liong et al.* (2001) and *Khu and Werner* (2003), the ANN metamodel is not fit over a set of uniform training points found from a formal DoE, but rather initial optimization trials are conducted on the original simulation and the evaluated points from this process are used for fitting. *Liong et al.* (2001) found that a network with three hidden layers which is trained by data from six storm events accurately reproduces the original HydroWorks model in all regions of the parameter space; however, in regions near closely spaced training points a linear interpolation approach performs just as well. *Khu and Werner* (2003) and *Khu et al.* (2004) both use a single hidden layer. To avoid overfitting the ANN model, *Khu and Werner* (2003) employ the early stopping approach; while this procedure results in a savings of 80% of full evaluations, it can limit the number of unique design sites available for training, testing, and validation sets. Additional studies have developed automatic calibration procedures for the SWAT watershed model using various surrogate model forms. *Shoemaker et al.* (2007) incorporate RBF models within an evolution framework, screening offspring by estimated fitness predicted by the RBF model and then confirming optimal values with the computationally expensive SWAT model. In comparing the results of the evolutionary algorithm combined with RBF approximation to other calibration methods, they conclude that it is "the most effective algorithm when there was a severe limitation on the number of simulations that can be performed" and methods with model approximation "should be seriously considered as alternatives to widely used methods such as SCE [Shuffled Complex Evolution] and evolutionary algorithms without function approximation when the complexity of the simulation model limits the number of simulations that can feasibly be done." *Zhang et al.* (2009) approximated the SWAT model by one-hidden-layer ANN and SVM, tested both methods on two watersheds in the eastern United States, and determined that the SVM form resulted in better generalized models than those constructed using ANNs. *Razavi et al.* (2012b) compared the behavior of two SWAT metamodel-enabled calibration

optimizers, kriging-GA and Multistart Local Metric Stochastic RBF, with two optimizers without metamodeling, dynamically dimensioned search and GA. They concluded kriging-GA and dynamically dimensioned search performed similarly in all computational budget settings, with kriging-GA performing slightly better when a harsh limit is placed on the number of allowable function evaluations.

Computationally expensive groundwater models can also be calibrated via surrogate-enabled procedures. *Rizzo and Dougherty* (1994) used a neural kriging network, which consists of both training and spatial interpolation phases, to estimate hydraulic conductivity fields in both two- and three-dimensional aquifer models using limited field data. *Johnson and Rogers* (2000) tested the accuracy of using linear regression and ANN models for automatic calibration of the 2D finite-difference groundwater model SUTRA, using simulated annealing techniques to search the parameter space. The authors included linear approximator tests, which failed to reproduce the high-fidelity model, in their study to avoid "the pitfall of addressing a problem with an unnecessarily complex method," but acknowledged that from the onset they did not anticipate that they would perform well. *Mugunthan et al.* (2005) tested two RBF-based function approximation methods (*Regis and Shoemaker*, 2004; *Gutmann*, 2001) within various optimization algorithms for autocalibration of chlorinated ethene biodegradation in an aquifer. The original simulation model, DECHLOR, is a multispecies reactive transport model that uses the finite different model MODFLOW for flow computations and the reactive transport model RT3D for contaminant transport computations. For their field case study, the original model requires 2.5 hours to complete a single simulation, making it very poorly suited for use directly within an optimization routine. This routine computes objective function values at each evaluation point through the original groundwater model and then fits an RBF surface to aid in optimization search. Both function approximation models performed well, with the model developed by *Regis and Shoemaker* (2004) performing best for minimizing overall errors in the final calibrated model form.

Automatic calibration routines have also been developed for surface water body models which incorporate surrogate model forms. *Zou et al.* (2007) demonstrated how an adaptive ANN-GA approach can determine values for 19 calibration parameters which minimize errors in relation to measured values for a eutrophication model (WASP5/EUTRO) linked to a previously calibration CE-QUAL-W2 hydrodynamic model. The 19 calibration parameters were first determined through

a sensitivity analysis, and various ANN models were created to emulate the eutrophication model. The authors determined that an adaptive ANN-GA procedure (which starts with a limited training set and adaptively adds additional information during optimization) converges closer to the global optimal solution than a one-step ANN-GA process (which starts with a robust training set but no additional training data is added during optimization). The total computational time from training data generation through optimization for this method is about 6.5 days of continuous computation, which largely consists of training data generation and ANN training time. *Huang and Liu* (2010) performed a similar analysis for calibration of a CE-QUAL-W2 hydrodynamic and WQM, in which 26 calibration parameters were determined by sensitivity analysis in terms of their ability to predict 6 hydrodynamic and water quality outputs (including vertical profile measurements). They also concluded an adaptive procedure performs better than one-step and that the largest computational expense comes from generation of training data through runs of the original high-fidelity model. *Ostfeld and Salomons* (2005) also demonstrated a routine for autocalibration of a CE-QUAL-W2 model using a k-nearest neighbors algorithm (kNN) for approximating the error resulting from various parameter combinations. A GA was used for searching. Two application locations were used: a hypothetical reservoir was used to tune the GA-kNN parameters, while a model of the Lower Columbia Slough water body was used to demonstrate autocalibration for temperature and DO prediction. The coupled GA-kNN algorithm produced results similar to those of a pure GA (without model reduction), while reducing computational expense.

Surrogates in Operations and Design Optimization

One of the earliest examples of surrogate-enabled optimization in water resources to minimize computational expense can be found in the work of *Alley* (1986), which expanded on the work of *Gorelick et al.* (1984) by creating response functions of computationally expensive contaminant transport models using polynomial regression. These regressions are functions of pumping-recharge rates at several wells, which form the decision variables of a groundwater contamination concentration minimization optimization problem, and are generated from the results of multiple transport simulation model runs. *Lefkoff and Gorelick* (1990)'s work expanded on Alley's by using regression to predict salt mass, rather than concentration, in an irrigated stream-aquifer system in the Arkansas Valley in southeastern Colorado. Although this study did not employ optimization

in the formal sense, the salt transport surrogate results were incorporated into a larger economic-hydrologic-agronomic model which serves as a tool for analyzing the relationship between crop mixing and profit in farming. This linked model system could be further formalized within an optimization routine to determine optimal trade-off points. *Cooper et al.* (1998) also developed a simulation/regression/optimization model for optimization of the oil recovery process from groundwater, expanding to a non-steady state problem. Response functions for residual oil and free oil were created using outputs from multiple runs of the ARMOS 2D finite element flow simulator, and verification of the surrogate-enabled optimization results by ARMOS simulation show small error levels.

Noting a need to expand these ideas to surface water applications, *Ejaz and Peralta* (1995) incorporated water quality processes from the QUAL2E simulation model within a simulation-optimization model via simplified regression equations. From the results of numerous systematic QUAL2E simulations, regression equations with a traditional mass balance form best fit all constituent response data with the exception of DO, which required a more detailed equation as a function of mass flow rates of BOD5, total nitrogen, and chlorophyll a. A verification step was included following nonlinear optimization to confirm that regression equations predicted acceptably close to QUAL2E. *Saad et al.* (1996) employed RBF ANNs to decompose the optimal operating policies obtained through dynamic programming for a reservoir system, which were combined to form one equivalent reservoir of equal potential energy. Using historical flow records, 500 equally likely deterministic inflow sequences were generated as inputs, and a year's optimal operations and corresponding potential energy were found for each on a monthly timestep. This formed the data set used for ANN training, and a fuzzy clustering approach was used to compute RBF parameters. *Neelakantan and Pundarikanthan* (1999) also used an ANN for simulation of a reservoir system's operation as substitution for a conventional simulation model, with the goal of maximizing drinking water supply. The monthly conventional mass-balance simulation model inputs and results were used to train a three-layer feedforward ANN, which was then embedded within a nonlinear optimization algorithm. Training each ANN required 8 hours of computational time, but the ANN model was reported to run 300 times faster than the conventional model. Solving the optimization problem took as long as 15 days of continuous computations using the conventional model, but only a few hours with the ANN model. *Castelletti et al.* (2010) used response surface methods to optimize the

number and location of water quality rehabilitation devices (i.e., mixers) in order to improve overall water quality in the Googong Reservoir in Australia. The 3-D coupled hydrodynamic-ecological model ELCOM-CAEDYM was used to compute training data for linear interpolators, RBF ANNs, and inverse distance weighting; the authors termed this step as the "learning phase." Then during the "planning phase," an approximate solution to the design problem is found. The learning and planning phases are performed iteratively to improve performance near optimal solutions(s), and at each iteration the response surface form with the smallest errors was chosen. Their results showed that significant improvements were possible by simply moving the currently installed mixers and that an additional pair of mixers would further improve destratification. To solve this design optimization problem using what-if analysis would "require about 5.5 years of computation with a modern computer" according to the authors.

II.3 Optimization of Hydropower Systems

Various techniques have been employed for hydropower optimization. Early studies employed linear programming (LP), which entails short computational times but requires functions to be linear or linearizable. This is often not the case for hydropower generation problems. A step up from LP, nonlinear programming (NLP) algorithms do not have the linear function requirement. NLP requires all functions to be differentiable, which may not be the case for hydropower systems. Dynamic programming (DP) methods have been popular in hydropower optimization tool development due to their ability to handle nonconvex and discontinuous functions and structure which emulates the multistage decision-making process involved in reservoir system operations (*Labadie*, 2004). The curse of dimensionality arises in these types of problems, which has led to various DP modifications to lessen the computational time of high-dimensional problems.

More recently, heuristic programming methods have become popular for investigating hydropower optimal operating patterns. In contrast to traditional derivative-based methods, heuristic techniques are less-structured, can rely on both quantitative and qualitative information, and can handle complexities including multiple objectives, uncertainty, nonlinearity, and discontinuities. Although convergence to an optimal solution cannot be guaranteed, heuristic methods are generally capable of locating global optima in all but the most complex problems, where traditional methods converge to

local optima (*Rani and Moreira*, 2010). These benefits may come at a computational cost by requiring more function evaluations than traditional optimization methods, but evolutionary or population-based methods allow for parallel computations (*Rani and Moreira*, 2010). Evolutionary methods that have been used for hydropower optimization applications in the literature include GAs, simulated annealing, ant colony optimization, particle swarm optimization, and honey bees mating optimization. These techniques have all been used in hydropower-related studies, but the literature is limited in comparison to traditional derivative-based methods.

Multiobjective reservoir optimization applications using both traditional and heuristic optimization approaches have sought to analyze the trade-off between a variety of outcomes including power generation, flood control, and water supply/quality. *Fontane et al.* (1997) employed stochastic DP to quantify optimal monthly releases for a 12-month period in terms of hydropower generation, flood control, water supply, and recreational demands. Using a GA, *Teegavarapu et al.* (2013) analyzed the trade-offs between power generation and downstream water quality using a simplistic one-dimensional decay process on a daily timescale, *Chen et al.* (2016) performed daily and hourly reservoir system scheduling subject to fish flow and other competing constraints, and *Liu et al.* (2011) incorporated minimization of flood risk on a daily timestep. These applications all assumed a well-mixed system or were performed in one spatial dimension.

II.3.1 Classic Methods

Linear Programming (LP)

LP is one of the most popular methods for reservoir system optimization due its many advantages, which include efficiency, ability to solve large-scale problems, global convergence guarantee, no initial solutions required to start the algorithm, duality theory to assist in sensitivity analyses, and ease of problem setup and solution using readily available software packages (*Labadie*, 2004). The most notable limitation of this technique is the requirement of objective and constraint functions to be linear or linearizable and convex. These limitations can be overcome in some cases by extension methods including separable LP, successive LP, and binary, integer, and mixed integer LP; however, many reservoir systems are represented by highly nonlinear or discontinuous functions associated with reservoir hydrodynamics, power generation, and water quality. These are either not appropriate

for or cannot be efficiently solved by LP, even with extension methods.

Simple reservoir optimization problems have been solved using LP techniques. *Ponnambalam et al.* (1989) solved for monthly turbine releases for two reservoirs connected in series over a 40 year period, resulting in 880 decision variables and 3680 constraints. They compared the performance of simplex and interior point algorithms, concluding that the interior point method converges in far fewer steps for large problems. *Crawley and Dandy* (1993) used linear goal programming to identify monthly optimal operating policies for a much larger reservoir system in South Australia, with the objective of minimizing pumping costs from a nearby river for reservoir fill. The authors used separable programming to piece-wise linearize the nonlinear pumping cost curves. *Needham et al.* (2000) analyzed the flood-control procedures for three U.S. Army Corps of Engineers reservoirs using a mixed integer LP model, concluding that coordinated releases may be unnecessary to minimize flood damage by showing this to be true for 8 of the 10 largest flood events on record. Additional application of optimization by LP for reservoir operations include (*Martin*, 1983), (*Martin*, 1995), *Lee et al.* (2006), *Seifi and Hipel* (2001), *Ziaei et al.* (2012), and *Mousavi et al.* (2004).

Nonlinear Programming (NLP)

Because many reservoir systems cannot be realized by linear or linearizable functions, NLP techniques have been employed in previous optimization applications. NLP has the disadvantages of slow convergence, leading to large computation time requirements. There is also no guarantee of find global optima, demonstrated by NLP algorithms often converging to local optima instead. The Karush-Kuhn-Tucker conditions for constrained nonlinear programming optimality may not be computationally feasible for many large-scale nonlinear problems (*Hiew*, 1987). Because of this, constrained NLP problems are often solved using penalty and barrier constraint-handling methods, which require careful choice of penalty weights and may not converge to the true feasible optimum. As noted by *Rani and Moreira* (2010), software packages are available which can solve large scale nonlinear optimization problems; regardless, global optimality proves difficult for practical applications employing NLP.

This is a broad family of techniques which includes sequential linear programming (*Barros et al.*, 2003; *Grygier and Stedinger*, 1985), sequential quadratic programming (*Tejada-Guibert et al.*, 1990; *Finardi et al.*, 2005), the augmented Lagrangian method (also known as the method

of multipliers) (*Arnold et al.*, 1994; *Naresh and Sharma*, 2002; *Finardi and Scuzziato*, 2013), and the generalized reduced gradient method (*Sale et al.*, 1982; *Unver and Mays*, 1990). All of these methods require differentiable objective and constraint functions, which may not be the case for hydropower systems due to the presence of discontinuities often associated with turbine operations. *Hiew* (1987) compared various nonlinear algorithms for optimization of a system of hydropower reservoirs and concluded the sequential linear programming method to be the most efficient. Using mixed integer nonlinear programming (MINLP), *Teegavarapu and Simonovic* (2000) optimized power generation revenues for a system of 4 hydropower plants with daily scheduling and *Ferreira and Teegavarapu* (2012) formulated a single run-of-the-river hydropower reservoir optimization problem on a daily timestep over a 15 day operating period. They included a simplistic downstream water quality constraint to explore dam operations' ability to counteract a downstream pollutant point source. Although formulated as a MINLP, the authors opted to solve the problem using GAs, noting that the "reduced gradient based method used initially in this study as optimization solver provided unsatisfactory (i.e., non-optimal) solutions."

Dynamic Programming (DP)

DP methods are able to address nonconvex and discontinuous functions and their structure emulates the multistage decision-making process involved in reservoir system operations (*Labadie*, 2004). DP breaks the original problem into subproblems that are then solved in stages sequentially. For each subproblem, an optimal cost-to-go function is developed which represents the optimal value accumulated from the current period going forward, as a function of an initial state condition. For the majority of reservoir applications, the state consists of reservoir storage. If additional states are relevant to the constraint and objective formulations, such as the inclusion of water quality or additional reservoirs, the size of the problem grows quickly; this has been coined the "curse of dimensionality" associated with DP. Discrete DP overcomes difficulties due to nonlinear, nonconvex, and discontinuous objective and constraint functions (*Labadie*, 2004).

The earliest application of determining optimal operating rules for a single multi-purpose reservoir using deterministic DP was performed by *Hall et al.* (1968). Their technique provided for what were considered to be "complex constraints" at the time, including time-variable flood control reservations; mandatory fish, wildlife, and recreational releases; and navigation minimum flows. This

resulted in an optimal schedule of releases for each month given a price schedule. *Stedinger et al.* (1984) developed a stochastic DP model to define releases from a dam in the Nile River Basin based on the best inflow forecast as a hydrologic state variable, resulting in improved operations compared to using the proceeding period's inflow as the state variable. *Georgakakos et al.* (1997) used a combination of dynamic programming and optimal control method modules to maximize firm energy generation of the Lanier-Allatoona-Carters hydropower system across multiple timescales (instantaneously, hourly, and daily).

Optimization of many linked reservoirs becomes computationally infeasible using the original DP formulation, which is the reason much of the hydropower optimization by DP literature involves modified DP approaches. *Castelletti et al.* (2007) employed neuro-dynamic programming, which approximates Bellman functions with ANNs, for reservoir network management. *Yi et al.* (2003) solved a multireservoir unit allocation problem using dynamic programming with successive approximation, a technique which "replaces the original multidimensional problem with a sequence of 1D problems" and whose computational expense increases linearly with respect to the problem size. *Wang et al.* (2005) was able to solve a problem combining multiobjective optimization (hydropower, water supply, and flood control), a multireservoir system (three reservoirs in parallel), and stochastic inflows using a combination of modifications. These included a constraint technique (to transform the optimization to a single objective form) and combined decomposition iteration and simulation analysis to overcome the dimensionality problem. *El-Awar et al.* (1998), *Yurtal et al.* (2005), and *Zhao et al.* (2014) also employed modified DP approaches to solve for optimal hydropower reservoir operations.

II.3.2 Heuristic Algorithms

Genetic Algorithms (GAs)

GAs, first introduced by *Holland* (1975), are a family of algorithms based on the mechanics of genetics and natural selection. They use a variety of methods to transition from one generation population to the next, including genetic operators such as inheritance, mutation, selection, and crossover. Populations of candidate solutions are evolved toward better solutions in an iterative process which rewards feasible, near-optimal solutions. Candidate solutions are copied into the

next generation, mutated, and combined stochastically based on their assigned fitness levels,. This attempts to balance exploration of solutions from new areas of the design space and exploitation of solutions already found in regions of high fitness. This process terminates when stopping criteria has been reached; examples of these criteria include a maximum number of generations or solutions, a satisfactory fitness level, or a population homogeneity level being reached.

One of the earliest introductions of genetics algorithms in the water resources literature comes from *Esat and Hall* (1994), where GAs were used to solve the "four-reservoir problem." This problem concerns a system of four reservoirs, with both parallel and series connections, operated over twelve 2 hour periods (a total of 24 hours), searching for optimal releases with constraints related to flood control and turbine capacities. The authors concluded that as system size increases, computational expense for DDDP increases exponentially while the expense of GAs increase linearly. *Wardlaw and Sharif* (1999) solved the same "four-reservoir problem" as well as a more complex 10-reservoir problem, testing sensitivities to various GA settings. *Oliveira and Loucks* (1997) combined a genetic search algorithm with simulation models to determine optimal operating policy rules for several multireservoir systems, focusing on satisfying joint water demands and joint energy requirements. Similarly, *Suiadee and Tingsanchali* (2007) used a combined simulation-GA optimization model to determine optimal monthly reservoir rule curves for a single reservoir in Thailand, with the objective function equal to the maximum net system benefit subject to irrigation constraints and the monthly releases computed by the simulation model. *Ahmed and Sarma* (2005), *Chang and Chang* (2001), and *Cheng et al.* (2008) each employed various forms of GA for determining optimal reservoir operations.

GAs have been used in combination with surface WQMs. *Kerachian and Karamouz* (2007) determined optimal operating rules for the Ghomrud Reservoir-River system in Iran for water quality management using a stochastic GA-based conflict resolution technique. A one-dimensional WQM simulating thermal stratification and water quality at releases from different outlets was used, as well as simulation of pollutants in the downstream river. This one-dimensional model was based on the existing Ghomrud HEC-5Q model, which could not be easily linked to the optimization model. *Ostfeld and Salomons* (2005) and *Huang and Liu* (2010) coupled hybrid GAs and ANN models for calibration of surface water quality CE-QUAL-W2 models. *Ostfeld and Salomons* (2005) reduced computational time by implementing a "hurdle race" approach which halts CE-QUAL-W2 simula-

tions early if a threshold is not met during simulation, while *Huang and Liu* (2010) combined a GA with a local search method to improve the results while reducing expense. *Dhar and Datta* (2008) linked a CE-QUAL-W2 model with an elitist GA to determine optimal reservoir operation policy with the aim of maintaining water quality downstream of the reservoir while minimizing the storage deviation from target storage. The authors employed this method on a hypothetical reservoir on the upstream end of the Middle Willamette River in Oregon, USA for daily operating decisions over a 10 day management period. They concluded with the note that "[d]evelopment of parallel code or use of metamodels (e.g. ANNs) may be very useful in reducing the CPU time" and that those modifications would "make it feasible to solve larger and more complex real-life optimal reservoir system operation problems."

Simulated Annealing (SA)

First introduced by *Kirkpatrick et al.* (1983), SA is a global search method which emulates the annealing process in glasses and metals to find optimal solutions for large systems. Using a temperature parameter, simulated annealing solves an optimization problem by theoretically maximizing strength and minimizing brittleness. Early water resources applications of this technique were for groundwater management problems, with the first reservoir operations optimization application performed by *Teegavarapu and Simonovic* (2002). They used the technique to optimize a four-reservoir system for hydropower and irrigation needs, including a simulation model for computing reservoir states during optimization. They solved a weekly problem on a half-day timestep and showed that SA provides similar results to a mixed integer NLP problem. Then they expanded the decision space by solving for hourly operations over a weekly horizon, which the SA algorithm was able to solve in a computationally feasible manner. *Tospornsampan et al.* (2005) compared the performance of using simulated annealing and GAs for determining monthly operations over 3 years for a multi-reservoir system with diversions, with the goal of minimizing irrigation deficits. Their results showed SA to be more efficient than GA for their application, generating higher quality solutions and requiring less computational time. *Li and Wei* (2008) also found SA to perform better than GA while optimizing a 3-reservoir system in series for electricity generation maximization. Of the methods they tested, the authors determined that their improved GA-SA algorithm produced the highest quality solutions at a lower computational time than the traditional unimproved GA-SA

algorithm. *Chiu et al.* (2007) also employed a hybrid GA-SA for optimizing the operation scheme of a single reservoir in Taiwan, concluding that the method results in superior performance as well as reduced computational time due to parallel analyses.

Ant Colony Optimization (ACO)

ACO is a heuristic technique based on observations of the behavioral patterns of ant colonies. Certain ant species are capable of finding shortest paths by using pheromone communication. ACO aims to emulate the shortest path search capabilities of these species (*Dorigo and Stützle*, 2004). Examples of ACO use in hydropower optimization applications are limited. *Kumar and Reddy* (2006) compared ACO to real coded GA for optimization of a multi-purpose reservoir in India and determined that the ACO algorithm converges to more globally optimal results than GA does.The developed models were used to determine operations on a monthly timestep for both short-term and long-term horizons. Optimization objectives were minimizing flood risk, minimizing irrigation deficits, and maximizing hydropower production; no water quality objectives or constraints were considered. *Jalali et al.* (2007) used a special version of the ACO algorithm to overcome ACO's difficulty handling continuous problems. A random mesh of the search space was used to minimize the chance of missing the global optimum, and the algorithm is also capable of handling discrete and continuous decision variables. The algorithm was tested on a complex 10-reservoir problem, which is "beyond the capacity of traditional DP and is difficult with variants such as DDDP [discrete differential dynamic programming], but is relatively simple to solve by LP." The system consists of reservoirs in parallel and series and was optimized over 12 operating periods with the goal of maximizing hydropower production. ACO was able to reach solutions which were 99.8% of the known global solutions. *Madadgar and Afshar* (2009) extended the initial ACO discrete space search method to continuous domains, improved algorithm performance and efficiency with the addition of an adaptation operator and explorer ants, and tested their algorithm on well-known benchmark problems and a single hydropower reservoir optimization problem with the objective of minimizing the sum of relative generation deficits from the installed capacity over 240 monthly operating periods.

Particle Swarm Optimization (PSO)

PSO is a technique for searching continuous nonlinear functions inspired by bird flocking and fish schooling behavior (*Eberhart and Kennedy*, 1995). It can solve many of the same types of problems as GAs. PSO is similar to a GA while overcoming some of GA's challenges, including being able to retain an active memory of good solutions. Unlike a GA, there are no evolution operators. Instead, each potential solution is assigned a random velocity, and then these "particles" are "flown through hyperspace." There are only two variables that must be defined by the user: maximum velocity and an acceleration constant.

Kumar and Reddy (2007) employed elitist-mutated PSO to determine operation plans for a multipurpose reservoir. Elitist-mutated PSO improves the standard PSO algorithm by adding an elitist-mutation mechanism. In their study, Kumar and Reddy applied elitist-mutated PSO to a hypothetical case and then to a realistic case, the Bhadra reservoir in India, which serves irrigation and hydropower generation purposes. The system was optimized on a monthly time step, for both 1 year (short-term) and 15 year (long-term) problems. This study concluded that elitist-mutated PSO performs better than both standard PSO and GAs, by yielding better solutions with fewer function evaluations. Similarly, *Zhang et al.* (2013) used a modified PSO approach to determine optimal hourly discharge rates for 10 cascading hydroelectric plants in a multi-reservoir system, with the goals of minimizing power deficit and uniformly distributing deficit if it should occur. This was achieved using a multi-elite guide PSO, which incorporated an archive set which preserves elite solutions. Multi-elite guide PSO produced improved solutions and converged quickly in comparison with other methods.

Honey Bees Mating Optimization (HBMO or MBO)

Another swarm-based algorithm is the HBMO method, which is inspired by the mating behavior of honey-bees in nature. This algorithm typically captures the bees' genetic potentiality, environment, and colony social conditions in order to converge to optimal solutions. *Haddad et al.* (2006) tested this algorithm on a water resources application for the first time. First it was applied to several benchmark constrained and unconstrained mathematical functions. Then the authors applied this algorithm to optimize single reservoir monthly operations over 5 years, aiming to minimize deviations between releases and target demands. They concluded that the HBMO algorithms per-

forms similarly well to GAs. More recently, *Dariane and Farahmandfar* (2013) applied the similar marriage in honey bees optimization (MBO) algorithm to determine 47 years of monthly operations for a three-reservoir system under irrigation and environmental flow requirements. This represented a problem with a very large number of decision variables. Their experiments revealed that MBO proved to be superior to other algorithms tested, including GA, ACO, PSO, and elitist-mutation PSO. The authors conclude by stating that "development of a hybrid algorithm consisting of MBO and any of the GA or elitist-mutation PSO algorithms could be considered in future research to further aid in solving complex optimisation problems with a large number of decision variables."

II.4 Gaps in the Literature and Research Advancement

This chapter summarizes the scope of the literature on reservoir modeling and operations, surrogate modeling techniques, and hydropower systems optimization. There is extensive documentation of a variety of hydrodynamic and WQMs capable of modeling waterbodies. These models have been applied to study the water quality impacts of an assortment of changes to natural and engineered systems; however, these studies typically apply to long-term planning and design purposes, not real-time operation. Although DSSs such as RiverWare and HEC-3/HEC-5/HEC-ResSim are powerful tools for determining optimal real-time hydropower operations, they have at most limited capabilities for considering water quality. When considered, water quality metrics are assessed by derived relationships between releases and water quality outcomes. This may not be adequate for river systems with strong temporal or spatial water quality gradients in areas of concern. These tools also cannot assess water quality system-wide, potentially missing areas of concern such as thermal plant cooling water withdrawal and release points or sensitive species spawning grounds.

Hydropower optimization objectives and constraints are typically represented by nonlinear and discontinuous functions. Most hydropower optimization studies have relied upon classic optimization algorithms that involve simplified function forms, linearization, and a focus solely on water quantity rather than quality. We observed recent growth in applying heuristic optimization methods for determining optimal hydropower operations, but the literature is limited in terms of applications for planning at an operational, rather than seasonal, timescale. While some studies such as those by *Kerachian and Karamouz* (2007) and *Dhar and Datta* (2008) considered water quality, none have

done so using a timestep of operational fidelity and high-fidelity WQM simulation. A primary reason for this lies in the high computational expense of high-fidelity WQMs and their structure being ill-suites for direct use in complex optimization problems. To counteract this, surrogate modeling approaches have been applied to water resources operations and design optimization applications. Even so, this approach has not yet been applied to solve for real-time hydropower operations optimization subject to constraints informed by high-fidelity WQMs.

This dissertation presents a foundation for developing a DSS capable of providing optimized real-time operational guidance for a hydropower system with refined water quality considerations. Optimized operations are influenced by robust WQMs capable of simulating water quality gradients, which may require high spatial and temporal model resolution. Integrating WQMs within a discontinuous, nonlinearized optimization problem that can be solved with limited computational resources is achieved by using surrogate modeling techniques.

Chapter III

HYDROPOWER OPTIMIZATION USING ARTIFICIAL NEURAL NETWORK SURROGATE MODELS OF A HIGH-FIDELITY HYDRODYNAMICS AND WATER QUALITY MODEL

III.1 Introduction

Reservoirs with hydropower capabilities are generally operated to maximize energy generation while meeting other water management policies and regulations (*Jager and Smith*, 2008). The optimization of reservoir operations is extensively studied (*Labadie*, 2004), with initial studies primarily focusing on water quantity constraints (*Hall et al.*, 1968; *Martin*, 1983; *Grygier and Stedinger*, 1985; *Arnold et al.*, 1994; *Teegavarapu and Simonovic*, 2000; *Chang and Chang*, 2001; *Seifi and Hipel*, 2001; *Teegavarapu and Simonovic*, 2002; *Yi et al.*, 2003; *Barros et al.*, 2003; *Cheng et al.*, 2008) and more recent studies integrating constraints related to ecosystems and water quality (*Hayes et al.*, 1998; *Chaves and Kojiri*, 2007; *Kerachian and Karamouz*, 2007; *Dhar and Datta*, 2008; *Ferreira and Teegavarapu*, 2012; *Castelletti et al.*, 2014). The inclusion of water quality as a constraint has been limited in that studies have not employed state-of-the-art multi-dimension high-fidelity hydrodynamic and WQMs, but instead generally incorporate one-dimensional or quasi two-dimensional coarse-grid models (*Hayes et al.*, 1998; *Jager and Smith*, 2008; *Ferreira and Teegavarapu*, 2012). Fidelity is defined here as a measure of similarity between a real-life system and a synthetic system, or model; in terms of time and space, this can also be called model resolution. Extending reservoir optimization modeling to multi-dimension and/or high-fidelity greatly increases computational requirements, even for a single reservoir under simulated environmental constraints (e.g., (*Dhar and Datta*, 2008)). The need for high-fidelity models within optimization schemes has come of age, driven by increased computational capabilities (*Castelletti et al.*, 2010)

and by increased requirements to meet specific points of compliance with greater accuracy.

Hydropower optimization efforts to date have not incorporated high-fidelity WQMs on an operations timescale, where operating decisions are made every hour or less, but rather for long-term seasonal or yearly planning. Additionally, such models often employ either one-dimensional WQMs, utilize relatively low spatial resolution, or both. Low temporal and spatial resolution restricts applications timescales and limits the ability to capture well the complex hydrodynamic and water quality interactions at water release points and other points of compliance of interest such as in vicinity of sensitive species areas or thermal electric water intake and discharge zones. Further, many optimization methods require linearity and differentiable functions, which cannot be addressed by numerical models. Lastly, both traditional and heuristic optimization routines often require significant numbers of objective and constraint evaluations, hindering the use of computationally expensive models.

The optimization of hydropower-equipped reservoir operations subject to numerous constraints is typically realized by a high-dimensional, non-linear, discontinuous problem formulation (*Labadie*, 2004), presenting a challenge in determining globally optimal solutions. Computationally-efficient gradient-based solvers can converge to local optima (especially for high-dimensional problems) and require known analytical function forms in order to compute gradients (*Labadie*, 2004; *Jin*, 2005). Reservoir operations are, by their nature, dynamic, and dynamic programming has been heavily employed in this area; however, this approach is not feasible for high-dimensional problems. The inclusion of water quality constraints is feasible when employing simple differentiable function approximations of water quality and hydrodynamic processes; however, this is not the case when including computationally-demanding simulation models within optimization routines. A heuristic global optimization method overcomes these challenges and allows for inclusion of high-fidelity models within constraints by use of surrogate models (*Forrester et al.*, 2008).

Here, we describe an advancement for computing optimal hourly power generation schemes for a hydropower reservoir through use of computationally-demanding WQMs, surrogate modeling techniques, and optimization methods. Optimal schemes are those in which water quality and other constraints are met as closely as possible, while flows are passed through hydropower turbines to produce maximum power value. Due to problem complexity and the use of heuristic methods, "optimal solution" here refers to the best solution found by the global solver employed. This study presents the development and application of an approach where the predictive power of the high-

fidelity hydrodynamic and WQM CE-QUAL-W2 is successfully emulated using an ANN model, which is then integrated into a GA-based optimization scheme to inform scheduling on an operations timescale of reservoir operations subject to high-fidelity spatial and temporal constraints (*Smith Sawyer et al.*, 2013; *Shaw et al.*, 2013, 2015, 2016, 2017).

This architecture allows for inclusion of water quality constraints in the decision-making process and for comparison between resulting optimal schemes and current operating procedures, all at high spatial and temporal accuracy. This provides a means for stratified reservoir operators to determine preferred releases on an operational timescale, maximizing power output while minimizing spill volumes necessary to maintain water quality standards. To date, no such approach exists on an operational timescale at a resolution that captures water quality gradients in dynamic, stratified reservoirs.

III.2 Case Study Description

The USACE Nashville District operates nine hydropower projects along the Cumberland River in Tennessee and Kentucky, USA (*U.S. Army Corps of Engineers*, 1998). The Cumberland River and its tributaries form the Cumberland River Basin (Figure III.1). The Cumberland River reservoirs' water levels are set by guide curves, which define volumes of water dedicated to purposes including power, flood, and minimum storage.

Old Hickory reservoir, a mainstem multipurpose reservoir for navigation, hydropower, and recreation located upstream of Nashville, Tennessee, has a backwater distance of 97.3 miles and is retained by a combination earthfill and concrete-gravity dam. Outflow structures are 6 tainter gates and 4 Kaplan hydropower turbine units, with a total installed capacity of 100 megawatts (MWs). The run-of-river Old Hickory project exhibits little fluctuation in water level due to navigation and recreation requirements; consequently, a review of historical operations reveals that Old Hickory's turbines consistently operate at or near their defined rating of 25 MW. Release decision projections are typically made 10 days in advance; additionally, operations are defined on an hourly or finer timescale and in terms of number of active turbines and spill gate settings.

Temperatures and water quality constituents of concern, including DO, are highly stratified vertically and longitudinally during the warm months. The Nashville District employs the CE-

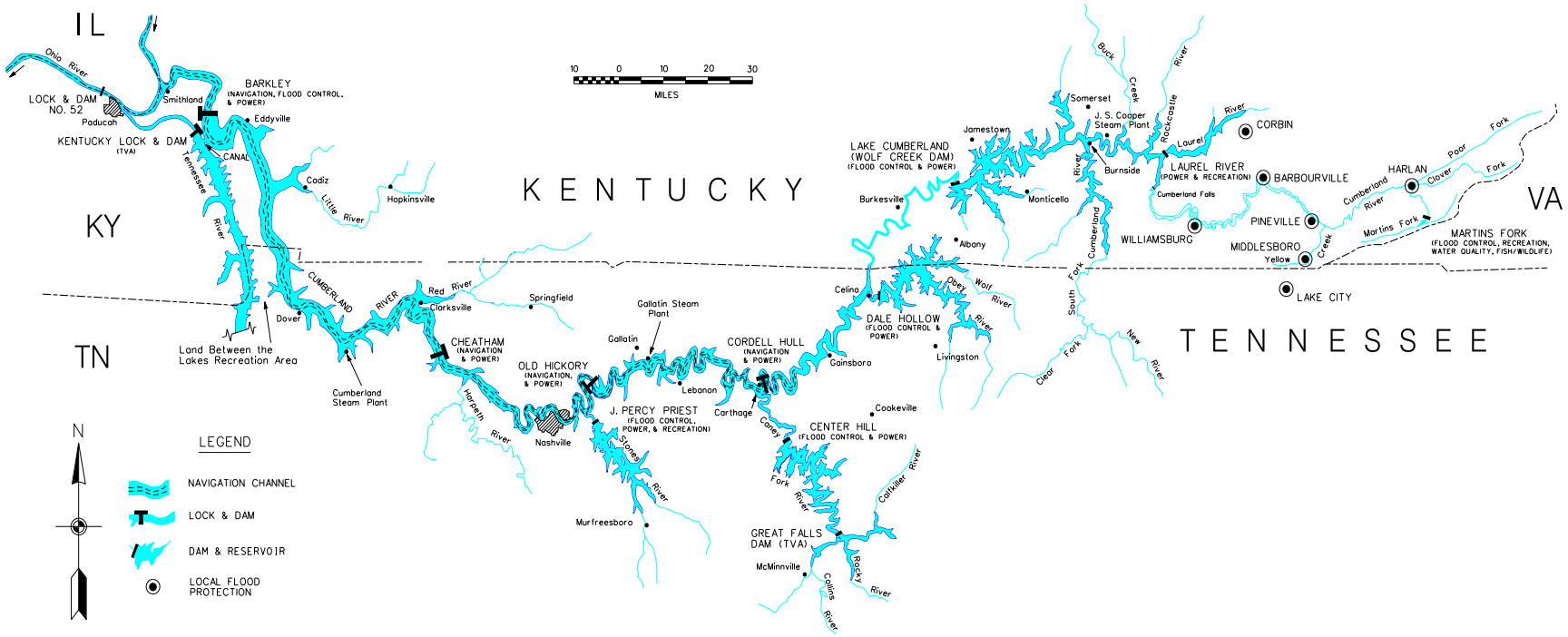


Figure III.1: Dam projects in the Cumberland River Basin (adapted from figure courtesy of Nashville District of the U.S. Army Corps of Engineers).

QUAL-W2 hydrodynamic and WQM for Old Hickory reservoir, allowing them to more accurately estimate water quality at points of compliance, to include releases and locations (both depth and river mile) of sensitive species; however, they do not currently directly incorporate the model within decision support systems for reservoir operations.

III.3 Optimization Problem Formulation

To determine optimal operations of Old Hickory reservoir, problems are formulated to determine turbine operations that generate maximum power value, subject to operational constraints. The problems are nonlinear with integer decision variables $\{x_1, x_2, ..., x_n\}$, representing the number of active turbines at each hour $i = 1 : n$. Optimization is performed for a defined planning period, in this case 10 days, a typical river system scheduling operational period (*U.S. Army Corps of Engineers*, 1998). Computational expense increases substantially as the number of decision variables grows; therefore, the planning period is divided into daily sub-problems which are solved consecutively.

Old Hickory reservoir must fulfill many requirements, which are formulated as a set of hard and soft constraints. The algorithm seeks to meet soft constraints, but if they are not fulfilled completely the algorithm still proceeds. Soft constraints are integrated into the objective function by use of a penalty parameter. Several hard constraints and a single soft constraint applied in the experiments are described below and in Table III.4. The optimization problem objective and constraints can be written as follows and explained below. Equations III.2-III.8 are firm constraints on the problem that must be satisfied.

$$\underset{X}{\text{minimize}} \quad -\left[\sum_{i=1}^{n} C(i)\cdot x_i \cdot r\right] + \left[d\cdot(e_f - e_t)^2\right] \tag{III.1}$$

$$\text{s.t.} \quad p_l \le E(x_1, x_2, ..., x_i)_i \le p_u, \ \ \forall \ \ i = 1 : n \tag{III.2}$$

$$\sum_{i}^{i+z} x_i \ge 1, \ \ \forall \ \ i = 1 : (n-z) \tag{III.3}$$

$$\mid x_{i+1} - x_i \mid \le c, \ \ \forall \ \ i = 1 : (n-1) \tag{III.4}$$

$$(x_i \le x_{i+1} \le x_{i+2} \le x_{i+3}) \vee (x_i \ge x_{i+1} \ge x_{i+2} \ge x_{i+3}),$$
$$\forall \ \ i = 1 : (n-3) \tag{III.5}$$

$$\{x_i \in \mathbb{Z} \mid 0 \le x_i \le a\}, \ \ \forall \ \ i = 1 : n \tag{III.6}$$

$$\frac{\sum_{i=1}^{|S|} \max(0, o_l - o_i)}{|S|} \le 0 \tag{III.7}$$

$$\frac{\sum_{i=1}^{|S|} \max(0, t_l - t_i)}{|S|} \le 0 \tag{III.8}$$

III.3.1 Objective Function and Soft Constraint

The objective (Equation III.1) is to maximize (formulated as a minimization as is convention) the value of hydropower produced over a set planning period. n is the number of hours in the planning period, $C(i)$ is the power value at time i, and r is the turbine power rating in MW. A cost curve defines the relationship between the value of power production and the time of day, which is important due to changes in electricity demand and the use of hydropower traditionally as peaking power to supplement thermal power production. If no cost curve is provided, i.e., $C(i) = 1$ for all values of i, the problem is equivalent to maximizing the total power generated over the planning period. The employed cost curve (Figure III.2) was created using Old Hickory reservoir historical operating patterns to estimate a relationship between time of day and generation. This approach is intended to be used for planning, not for real-time grid balancing, so a historically-based cost curve is appropriate.

The second term in Equation III.1 is a penalty term representing a soft constraint, penalizing deviations of final water level e_f from the final target elevation e_t. This restricts the solution from draining to the bottom of the power pool at the end of each daily optimized sub-problem. Briefly,

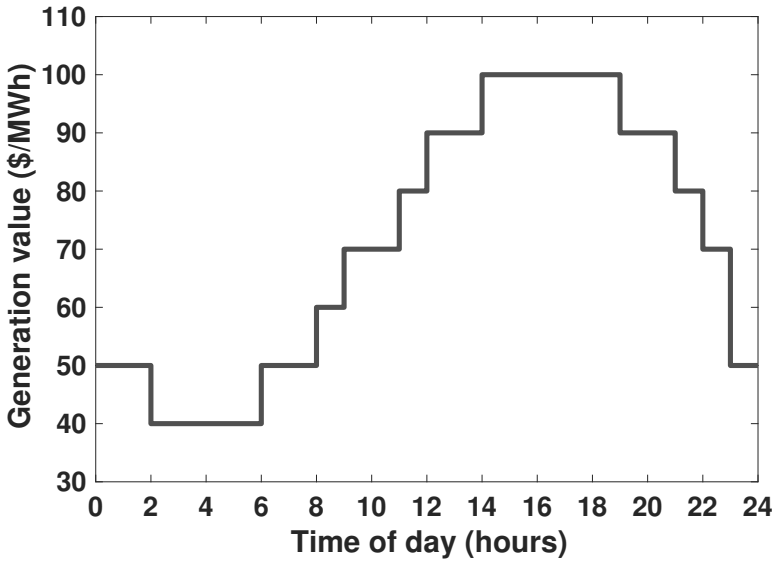


Figure III.2: Cost curve used in optimization applications.

for each daily sub-problem potential solution the final water level elevation is found, the penalty is computed, and a deduction to the objective function value is made for water level elevations below target levels. Prior to the start of the GA solver, a penalty coefficient is computed using linear interpolation:

$$d = y_{projected} \cdot \left(v_u + (v_l - v_u) \frac{p_T - p_l}{p_u - p_l} \right) \tag{III.9}$$

where d is the penalty coefficient in dollars per meter below target (or megawatt-hour, MWh, per meter below target if no cost curve is provided), $y_{projected}$ is the estimated power value under projected operations for the sub-problem optimization time period (in dollars if a cost curve is provided, otherwise in MWh), p_T is the target water level elevation at the end of the time period, p_l and p_u are lower and upper bounds of the power pool, respectively, and v_l and v_u are scaling coefficients with $v_l \leq v_u$. The penalty coefficient is greater the closer the target water level elevation is to the bottom of the power pool. Scaling coefficients are a function of the value of power and reservoir generation capacity, with larger coefficients aligned with increased penalty. For reservoirs with total capacities of 100 MW, like the one used in this study, and a cost curve with value magnitudes in the range of \$40-\$100/MWh as assumed here, values of $v_l = 500$ and $v_u = 1000$ perform well.

III.3.2 Hard Constraints

Equation III.2 sets lower and upper bounds (p_l and p_u, respectively) on water levels. $E(x_1, x_2, ..., x_i)$ is an elevation model that predicts water level elevations for all timesteps $1 : i$. For reservoirs operated on a seasonal guide curve, p_l and p_u are typically set to the lower and upper bounds of the power pool. The simplified water level elevation model assumes the water level to be consistent along the entire reservoir and is a function of all inflows and outflows. Spill flow is often engaged to improve downstream water quality. An average spill flowrate for each daily sub-problem is computed during elevation calculations based on turbine releases, inflows, and user-provided midnight target elevation values. First, water level elevation is computed based on the hourly turbine settings assuming no spill release. If the final elevation for the sub-problem is less than the target elevation, spill remains zero. If the final elevation is greater than the target elevation, an average spill flowrate for the sub-problem is assigned which results in a final water level elevation equal to the target value. This incorporates spill without requiring additional decision variables, which is important since spill flow is often engaged to improve downstream water quality.

In an effort to maintain minimum flows along the river, the maximum number of consecutive hours z allowed without power generation is defined by Equation III.3. The USACE Nashville District implements this rule for water quality purposes as well.

Equation III.4 limits the hourly rate of change in the number of active turbines, with c being is the maximum number of turbine units that can become active or go inactive each hour. Since Old Hickory reservoir exists on a navigable waterway with lock systems, this constraint assists in minimizing fluctuations in the surface elevation and adverse impacts on water level stability.

Equation III.5 attempts to reduce oscillations in the turbine operations over time. This constraint is formulated with logic that states that, except in cases of ramping turbines up or down, the number of active turbines must be fixed for at least three hours consecutively before changing. Reducing oscillations is desired to minimize equipment wear.

Equation III.6 defines the maximum number of turbines at the hydropower facility, a. It is assumed that all turbines operate at the same turbine power rating, r, and that the number of active turbines is selected from a set of integer options.

The Nashville District monitors DO levels in the Old Hickory dam, which is directly upstream of

the metropolitan Nashville area and has historically proven to be a strong indicator of water quality system-wide (*U.S. Army Corps of Engineers*, 1998). Maintaining cool discharge temperatures is also important as the Cumberland River serves as a source of cooling water for TVA's thermal power plants both upstream and downstream of Old Hickory dam. Equations III.7 and III.8 define lower constraints on discharge DO and temperature, respectively, where o_l and t_l are lower limits and o_i and t_i are DO and temperature estimates at time i. These equations can be modified to account for maximum constraints as well. Discharge water quality over the operating period is computed by:

$$O(\vec{x}) = (o_1 \;\; o_2 \;\; \cdots \;\; o_n) \tag{III.10}$$

$$T(\vec{x}) = (t_1 \;\; t_2 \;\; \cdots \;\; t_n) \tag{III.11}$$

where $O(\vec{x})$ is a function estimating discharge DO concentration and $T(\vec{x})$ is a function estimating discharge temperature. In this application, $O(\vec{x})$ and $T(\vec{x})$ are ANN models predicting the water quality estimations of a simulation model. S, the set of timesteps with total dam discharge flow not equal to zero, is defined by:

$$S = \left\{ i \mid (Q_i^T + Q_i^S) \neq 0 \right\} \tag{III.12}$$

where Q_i^T is the turbine discharge and Q_i^S is the spill discharge at time i. $|S|$ is the size of set S. Dividing by $|S|$ accounts for the fact that at times when there is no release from the turbines or spillway, discharge water quality is undefined. This approach also makes it easier to compare population members which are not fully-feasible with respect to water quality by having a single metric for comparison. Equations III.7 and III.8 require the average hourly constraint violation to be less than or equal to zero; since the constraint violation can never be negative, the average hourly constraint violation is equal to zero.

III.4 Methodology

A GA-based decision support tool was developed to determine optimal turbine operations for a single hydropower reservoir, with inclusion of point release water quality constraints informed by a high-fidelity simulation model. The overall approach, illustrated in Figure I.2, integrates a system of water quality and hydrodynamic models into an optimization framework by use of a

reduced model. This model is formulated as an ANN of the nonlinear autoregressive network with exogenous inputs (NARX) form, and is trained using model simulation outputs. The computational expense of prediction is considerably reduced from that of the original model, thereby allowing for a great number of function evaluations required during optimization. An hourly timescale over a 10-day horizon was employed, reflective of actual operator planning routines; however, this approach could be applied over longer horizons on a less-refined timescale for seasonal or yearly planning. Longer horizon studies would be sensitive to accuracy of inflow and meteorological forecasts.

CE-QUAL-W2, a two-dimensional high-fidelity hydrodynamic and WQM, was used as the original simulation model. CE-QUAL-W2 has successfully been used to simulate rivers, lakes, reservoirs, and estuaries since 1975 (*Martin*, 1988; *Adams et al.*, 1997; *Saito et al.*, 2001; *Bowen and Hieronymus*, 2003; *Kuo et al.*, 2006; *Chung and Oh*, 2006; *Debele et al.*, 2008; *Afshar et al.*, 2011), with the ability to model physical, chemical, and biological processes including temperature, DO, nutrients, algae, and sediments (*Cole and Wells*, 2007). The spatial grid is user-defined and laterally averaged, making it well-suited for modeling long narrow water bodies such as the Cumberland River system controlled reservoirs. The temporal resolution is determined by time stepping routines which limit numerical instability.

The reduced model is represented by a NARX network, a form of ANN. ANNs are flexible tools for function approximation composed of neurons assembled into a multi-layer architecture, and have been used for numerous complex problems (*Cheng and Titterington*, 1994), including as emulators in reservoir operations problems (*Raman and Chandramouli*, 1996; *Saad et al.*, 1994). *Solomatine and Avila Torres* (1996) used ANNs within an optimization routine to meet water depth and power generation targets, but the spatial and temporal resolution were coarse and the optimization formulation highly simplified. *Aguilar et al.* (2014) built a water quality forecasting surrogate model using a tree-based approach as an alternative to ANNs, acknowledging a likelihood for error propagation. They did not integrate the reduced model within a decision-making process.

Construction of ANNs consists of two steps: (i) specifying the architecture and (ii) training the network. Model architecture is generally determined by trial-and-error (*Razavi et al.*, 2012a), and is specified through several parameters, including number of hidden layers, number of neurons in each hidden layer, and form of transfer function. As in all modeling approaches, the smallest architecture with an acceptably low error should be used to minimize computational expense, both

during training and prediction. Once the architecture is defined, model weights are determined through a training process like back-propagation (*Simpson et al.*, 2001).

Several surrogate model forms were initially tested. Linear regression, Gaussian process, radial basis function, and Shepard's Method were unable to emulate CE-QUAL-W2's highly nonlinear and dynamic water quality predictions (*Shaw et al.*, 2013). The NARX model form was selected for its ability to approximate time-dependent functions that are dependent upon a large number of inputs using training data derived from high-fidelity simulation model runs. NARX training, visualization, and prediction tools are available in the MATLAB® Neural Network Toolbox (R2016a, The MathWorks Inc., Natick, Massachusetts, United States). This model relates past values of the same series in the following way:

$$y(t) = f\big(y(t-n_{y,1}), y(t-n_{y,2}), \ldots, y(t-n_{y,last}), u(t-n_{u,1}), u(t-n_{u,2}), \ldots, u(t-n_{u,last})\big) \quad \text{(III.13)}$$

where y is/are the variable(s) of interest, u is/are the exogenous variable(s), and f is a nonlinear function mapped by a multilayer perceptron (*Lin et al.*, 1996). The model is a function of feedback delays defined by the set n_y and input delays defined by the set n_u. NARX models are trained using a family of CE-QUAL-W2 simulation results, obtained by combining different CE-QUAL-W2 input scenarios. Training is performed using a Levenberg-Marquardt backpropagation optimization algorithm, considered to be one of the most computationally efficient ANN training methods (*Razavi et al.*, 2012a). Once trained, a NARX model emulates CE-QUAL-W2's predictive ability for new scenarios without the need for additional CE-QUAL-W2 simulations. MATLAB® codes utilized to create NARX surrogate models are provided in Appendix C.

GA optimization was selected due to its ability to identify global optima for problems with nonlinearities and discontinuities, as are present in objective and constraint functions in many hydropower optimization operations (*Esat and Hall*, 1994; *Oliveira and Loucks*, 1997; *Wardlaw and Sharif*, 1999; *Labadie*, 2004; *Ahmed and Sarma*, 2005; *Suiadee and Tingsanchali*, 2007), including optimization of systems in combination with surface WQMs (*Kerachian and Karamouz*, 2007; *Dhar and Datta*, 2008). GAs represent a family of heuristic algorithms based on the mechanics of genetics and natural selection, employing a variety of methods to transition from one generation

population to the next, including inheritance, mutation, selection, and crossover. In GA applications, stopping criteria as well as other algorithm parameters are typically tuned through trial-and-error (*Reed et al.*, 2000). GAs are not mathematically guaranteed to find globally optimal solutions, but studies have shown their improved performance in terms of avoiding local optima over LP and NLP for complex applications (*Azamathulla et al.*, 2008; *Aly and Peralta*, 1999; *Wardlaw and Bhaktikul*, 2004).

Dhar and Datta (2008) linked CE-QUAL-W2 model with a GA to determine optimal reservoir operation policy with the aim of maintaining water quality downstream of a reservoir release, concluding that development of parallel code or integration of metamodels, such as ANNs, could reduce computational time and increase the feasibility of solving larger, more complex reservoir system operations problems. In the study described, water quality processes are integrated using NARX models, which can be viewed as "black box" approximators. The optimization routine seeks to determine the active turbine pattern on an hourly timestep to maximize power production or power value subject to constraints on discharge water quality, water level elevation, zero-generation hourly limits, limits on rate of change in turbines, and turbine unit availability. The objective and constraint functions are structured so that they can be modified to meet the needs of other reservoirs in a multi-reservoir, linked system.

The optimization routine was constructed using the GA functionality available in the MATLAB® Optimization Toolbox (R2016a, The MathWorks Inc., Natick, Massachusetts, United States), and the MATLAB® codes used for the hydropower optimization process described here are provided in Appendix D. This process (Figure III.3) begins with defining reservoir characteristics: tailwater rating curve, storage elevation curve, number of turbine units, turbine rating in MWs, and turbine discharge curve. A turbine discharge curve provides a relationship between turbine release, head difference, and turbine rating in MW. At a fixed turbine rating, the turbine discharge curve allows one to compute turbine release flowrates as a function of the number of turbines active, upstream water surface elevation, and tailwater elevation (computed using the tailwater rating curve). A CE-QUAL-W2 model folder is also provided with measured and forecasted input files updated to reflect the current year.

Optimization settings include optimization start date (JDAY, or Julian day), operating period length (days), midnight water surface elevation targets (meters), maximum change in active turbine

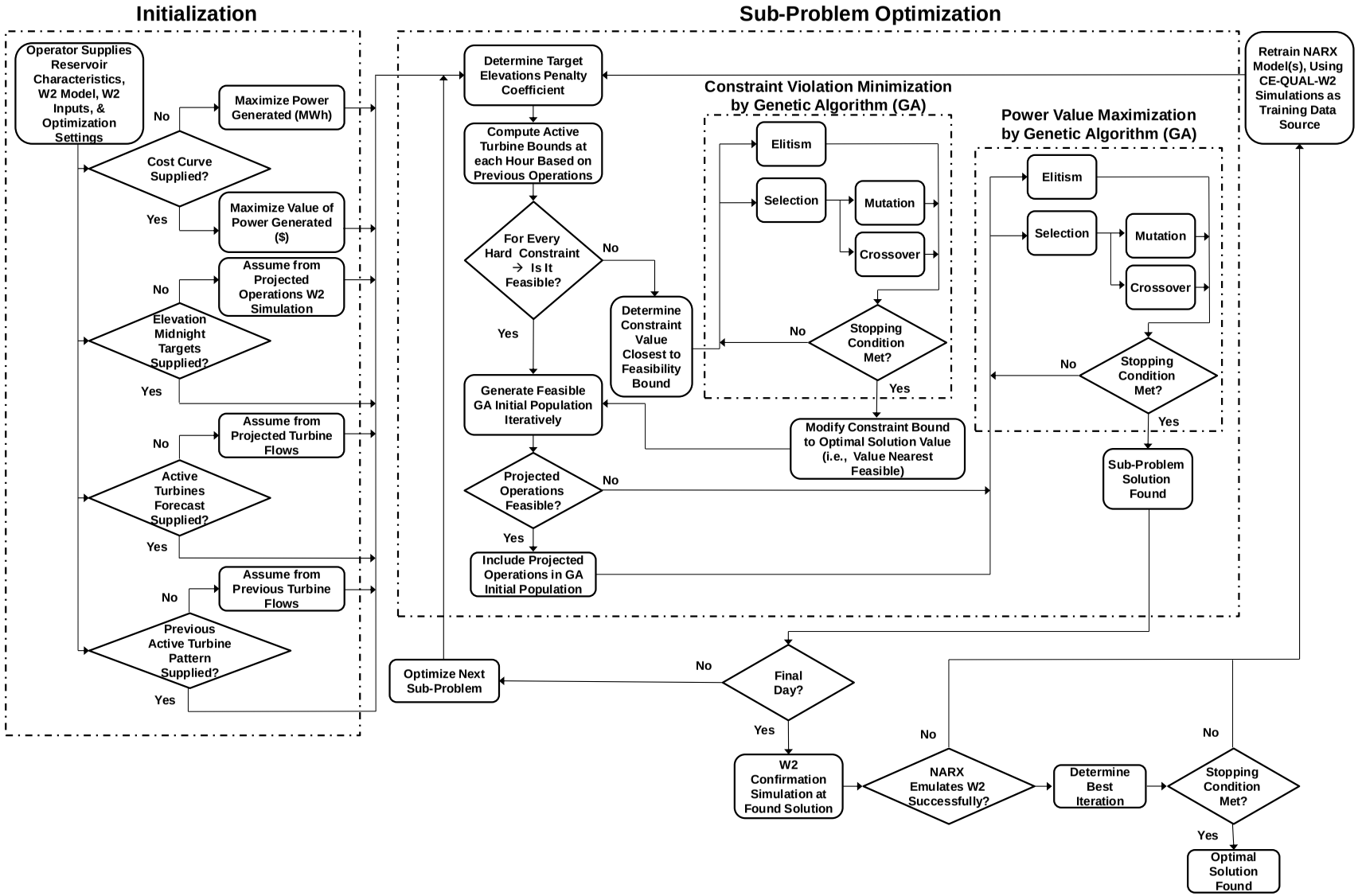


Figure III.3: Schematic of optimization methodology.

units (units/hour), maximum hours with zero power generation, daily cost curve, and elevation and water quality constraint limits. Scenarios may exist where elevation or water quality constraints are not feasible, independent of release decisions. For these constraints, a hard constraint feasibility estimate is performed prior to power value optimization. If no feasible solution can be found for a particular constraint, the constraint bound is relaxed to the value found nearest to the constraint limit. Power value optimization can proceed in scenarios with no fully feasible solution by allowing the algorithm to prioritize these constraints over the objective of power generation. An initial population of potential solutions satisfying all hard constraints is produced and supplied to GA at the onset of each daily sub-problem. These potential solutions are found using logical decision-making and random selection, starting with first hour turbine setting and progressing through the last hour for each potential solution for the sub-problem. If the projected turbine operations are feasible they are added to the initial population set.

The GA employs creation and mutation functions which produce populations consisting of integer values for the decision variables. The optimal solution is identified by the GA and iterating forward until a stopping condition is satisfied, with each daily sub-problem solved in succession. After optimal operations have been determined over the planning period, a CE-QUAL-W2 validation simulation provides means for comparison to the surrogate WQM predictions. Following each optimization iteration, the best iteration is determined by the tiered logic system described below. When the best iteration ceases to change over two iterations, the stopping condition is satisfied and the algorithm terminates.

After the series of daily sub-problems is solved over one iteration, a CE-QUAL-W2 confirmation simulation is performed at the identified optimal release operations to ensure the surrogate model sufficiently emulates the CE-QUAL-W2 model. If the confirmation simulation and NARX predictions acceptably agree, the solution is accepted. Otherwise, NARX models are retrained and updated using two CE-QUAL-W2 simulations as training data. These two simulations consist of (i) the CE-QUAL-W2 confirmation simulation, and (ii) a simulation with the confirmation turbine and spill discharges swapped. This provides diversity in the spill and turbine exogenous inputs, and assists the surrogate model in emulating the water quality outcomes from each release point. NARX models are retrained five times and the resulting model with the lowest cross-validation error is chosen, which provides enhanced training data for improved prediction of the optimal solution.

Following each iteration, two CE-QUAL-W2 simulations (confirmation and confirmation with releases swapped) are added to the training data set; therefore, following the first iteration (which uses a robust training data set described below) each training data set consists of $2 \cdot (iteration - 1)$ CE-QUAL-W2 simulations.

The algorithm's stopping condition is based upon the "best iteration" index at the end of each iteration. If any water quality constraint is "active", meaning not fully satisfied, the absolute mean error (AME) between the NARX and CE-QUAL-W2 water quality predictions is checked. If the AME is greater than 0.5 °C for temperature or 0.5 mg/L for DO (AME thresholds lower than acceptable levels given by *Cole and Wells* (2007)), the iterations solution is not acceptable and the best iteration is set to the previously found best iteration, or null in the case of no acceptable solution found thus far. If the AME is acceptable and the best iteration is null thus far, the current iteration is the best iteration. If a best iteration has been determined already, the water quality violation from the constraint limit is compared between the current iteration and the previously found best iteration. If the current iteration achieves a smaller water quality violation, it becomes the new best iteration. If there are no "active" water quality constraints (i.e., these constraints are fully satisfied), then the best iteration is based on the objective function valuation, which represents the power value. The power value of each iteration is compared to the power value of the best iteration found thus far, and if the new solution results in greater power value, it then becomes the new best iteration.

III.5 Experimental Setup

The USACE Nashville District provided operations data, field measurements, and CE-QUAL-W2 version 3.5 (*Cole and Wells*, 2007) models. CE-QUAL-W2 models were calibrated and validated for the case study reservoir for prediction of water level, temperature, and DO. Temperature and DO predictions were compared to measured values at the dam releases and available in-stream vertical profiles. Visualization and plotting during this process were performed using the AGPM-2D v3.5 post-processor for CE-QUAL-W2 (Loginetics, Inc.).

Calibration and validation time series results and water quality profiles are provided in Appendix A as Figures A.1 through A.6. Calibration and validation error metrics are summarized in Table III.1. Here, we consider CE-QUAL-W2 model calibration or validation acceptable when the

AME values are less than 1 °C for temperature and 1.5 mg/L for DO (*Cole and Wells*, 2007). Error metrics are within this threshold with the exception of Old Hickory in-stream temperature profiles for both years, likely due to only having daily temperature values available for the mainstem inflow. Old Hickory reservoir's main inflows consist of releases from two upstream dams, both of which are stratified in the summer and have outlet structures at multiple depths. The temperature and other water quality characteristics of the upstream dams' discharges are strongly impacted by release decisions, which much like Old Hickory reservoir are adjusted by operators on a short timescale. Consequently, the water quality of upstream releases is not adequately captured by a single measurement each day, thereby resulting in larger water quality prediction errors at profile locations in the upstream half of the reservoir.

Additionally, the original developers of the Old Hickory model separated side bank storage volume from mainstem conveyance volume by use of a separate branch and a series of weirs connecting the storage branch to the mainstem. While this may improve hydrodynamics modeling, this methodology does not properly represent the water quality phenomenon of the system. This makes a particular impact in the forebay of the reservoir, where the additional storage branch (Branch 10) enters the mainstem (Branch 1) as shown in Figure III.4. While the model is not constructed as desired, CE-QUAL-W2 emulation by surrogate model and integration within an optimization scheme is demonstrated using the Old Hickory model regardless of model structure and accuracy. The focus here is transition from high-fidelity simulation to reduced surrogate model, not transition from the true system to high-fidelity simulation model.

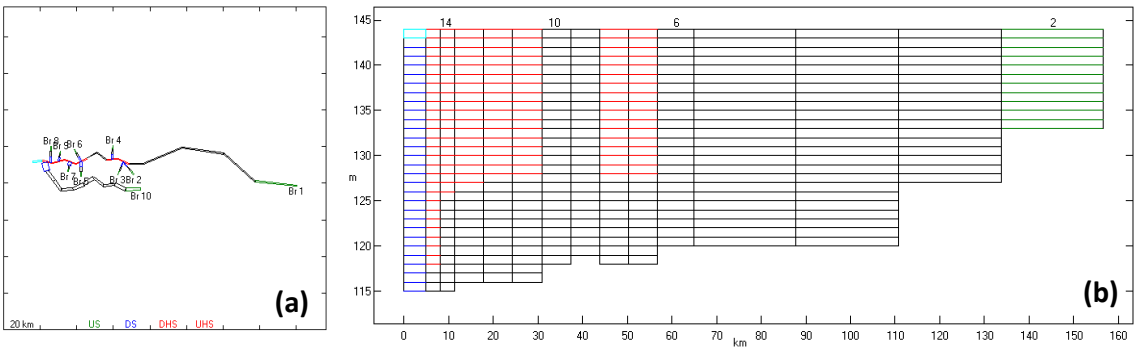


Figure III.4: Bathymetry of Old Hickory reservoir CE-QUAL-W2 model, showing (a) plan view of all branches and (b) elevation view of the mainstem, Branch 1 (created using AGPM-2D v3.5 post-processor for CE-QUAL-W2 by Loginetics, Inc.).

Table III.1: Summary of Old Hickory CE-QUAL-W2 model calibration and validation results.

	Calibration	Validation
Year	1988	2005
Computational Time (minutes)	9	9
Elevation AME [a] (meters)	0.025	0.053
Dam Releases:		
Temperature AME [a] (°C)	0.963	0.617
DO AME [a] (mg/L)	1.010	1.196
In-stream Profiles:		
Temperature AME [a] (°C)	2.076	1.350
DO AME [a] (mg/L)	0.943	0.716

[a] Errors are presented as absolute mean error (AME). In-stream profile measurements of temperature and DO were collected at 8 locations on 7 dates in the calibration year (1988) and at 7 locations on 2 dates in the validation year (2005).

The Old Hickory tailwater is considered the point of compliance and monitoring for water quality by dam operators; therefore, ANN models were trained to emulate the hourly discharge temperature and DO predictions of the CE-QUAL-W2 model. Based on observations made during CE-QUAL-W2 model calibration and validation, the discharge temperature and DO at Old Hickory are sensitive to only the two most dominant upstream inflows: Branch 1 (the mainstem) and Tributary 2 (Caney Fork, and the Center Hill dam discharge). Flowrates, temperatures, and DO concentrations for these two inflows were included in an initial exogenous input set. Additionally, meteorological data and operational data (spill and turbine flowrates) were included. Using the 2005 Old Hickory CE-QUAL-W2 model inputs and outputs, correlation tests were performed to narrow the set of exogenous inputs to the main driving factors for discharge temperature and to estimate the appropriate sets of input and feedback delays. Examples of correlation plots for discharge temperature are shown in Figure III.5 for demonstration. Exogenous inputs with low correlations were removed from the set. For the narrowed exogenous variable set, correlations with discharge temperature and DO were maximized in the vicinities of 0, 1, and 12 hour delays; hence, the input delay set was assigned to these values. Lagged autocorrelation testing of the discharge temperature and DO output time series show decreasing correlation over time, meaning a single feedback of 1 is appropriate. The resulting sets of exogenous variables for temperature and DO NARX models are given in Table III.2. The number of hidden layers and neurons in each layer were assigned to

the default values of 1 and 10, respectively, following sensitivity testing that revealed an increase in these values yielded little to no improvement in prediction ability at considerable computational expense.

Training data for Old Hickory NARX WQMs was generated by combining dominant inflows, outflows, and meteorological data time series. For each input type, three variations were considered. Meteorological conditions consisted of the 2005 (average year), 2006 (wet year), and 2007 (dry year) values. Inflow temperatures and DO concentrations consisted of the values from 2005 and the 2005 values were increased and decreased by 5%. Inflows were not varied, but outflows were varied to create heavy spill and heavy turbine scenarios. The heavy spill scenario was created by allocating 20% of the 2005 turbine outflow to the spill gates, and the heavy turbine scenario was created by allocating 20% of the 2005 spill outflow to the turbine structure outflow. Spill and turbine scenarios were not combined exhaustively, but instead were paired to maintain an equivalent total outflow to maintain water balance stability in the CE-QUAL-W2 simulations. This process creates a surrogate model which can be used to explore the trade-off between releases through the turbines and spill gates. An exhaustive combination of all variables, with the exception of the paired spill and turbine inputs as explained, resulted in a total of 729 CE-QUAL-W2 model simulations.

Seventy percent of the simulations were provided to the training algorithm and the remaining thirty percent saved for final validation. To minimize the impact of substantial oscillatory noise found in some CE-QUAL-W2 simulation results, the water quality predictions were smoothed using a 24-hour moving average process prior to training. A smoothing approach was selected in order to avoid removing runs from the design of experiments set; with the understanding that the

Table III.2: Exogenous variables lists for Old Hickory discharge NARX models.

Discharge Temperature	Discharge DO
Branch 1 Inflow	Branch 1 Inflow
Branch 1 Temperature	Branch 1 Temperature
Tributary 2 Temperature	Branch 1 DO
Air Temperature	Tributary 2 Temperature
Dew Point	Tributary 2 DO
Turbine Flow	Air Temperature
Spill Flow	Dew Point
	Turbine Flow
	Spill Flow

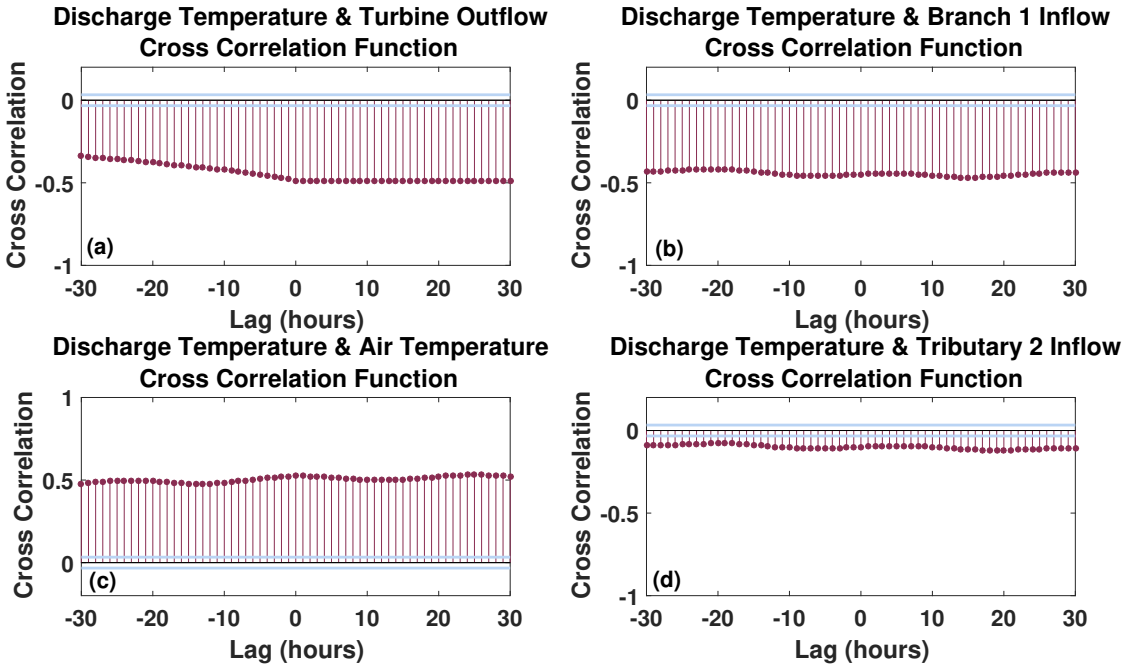


Figure III.5: Old Hickory discharge temperature lagged cross correlation test examples for (a) turbine outflow, (b) branch 1 inflow, (c) air temperature, and (d) tributary 2 inflow with 95% confidence bounds. Inputs shown in (a), (b), and (c) are considered correlated with discharge temperature and are included in the NARX model exogenous variables, while input (d) is not.

initial set of NARX models provides somewhat "smoothed" predictions due to the wide range of conditions in the training data set; and due to the fact that the NARX models are later updated in a retraining step within the optimization process, which is based upon non-smoothed CE-QUAL-W2 outputs. The training algorithm randomly divides its portion of data between training (70%), validation (15%), and test (15%) subsets. The training subset is used to compute gradients and update network weights and biases, the validation subset for computing errors and determining when to halt the training routine, and the test subset for confirming an appropriate division of data by comparing when the test subset and validation subset errors reach their minimums. Figure III.6 provides a visual demonstration of the random data division, with each box representing a CE-QUAL-W2 simulation.

Because the models are trained using an optimization algorithm that incorporates a random process, temperature and DO networks were each trained five times. After five networks were constructed and bias correction performed, an interior point constrained nonlinear optimization algorithm was employed to compute network weights (which sum to 1) that minimize the validation set error. After the first weight set was computed, any networks with a weight less than 25% of the maximum weight were removed and the weights recomputed for the smaller set of NARX models. This removes inferior networks from the set while still maintaining a "family" of networks that may provide better global predictions than a single trained network. In this application, the temperature surrogate model consists of 4 weighted NARX models and the DO surrogate model consists of 4 weighted NARX models.

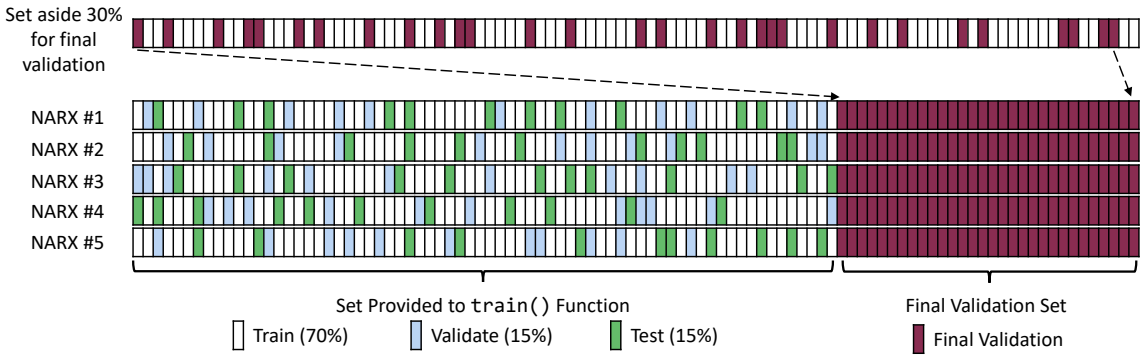


Figure III.6: Data division demonstration for NARX model training. Each box represents 1% of the total set of CE-QUAL-W2 simulations resulting from design of experiments.

III.6 Results

NARX models were trained to simulate hourly summer (May-September) discharge water quality using the family of CE-QUAL-W2 simulations described earlier, and validation errors computed. Shown in Figure III.7, training and validation errors have similar distributions suggesting no occurrence of overfitting. Examples of NARX model predictions compared to the 24-hour moving average smoothed CE-QUAL-W2 outcomes for Old Hickory reservoir are given in Figure III.8. The NARX surrogate model predictions closely follow the seasonal trends produced by CE-QUAL-W2, but are unable to fully replicate "peaks and valleys." The initial surrogate training data set consists of many exogenous variable and release scenario combinations, producing a robust model capable of providing general solutions for a variety of scenarios at the expense of refined predictions. Missing these extreme values could provide incorrect solutions in the region of optimization constraints; therefore, solution confirmation by CE-QUAL-W2 and surrogate model updating (as shown in Figure III.3) are vital steps for refining surrogate water quality predictions.

The success rate of a heuristic optimization algorithm is highly dependent on the problem to be solved and algorithm settings (*Reed et al.*, 2000). For GAs, computational time and accuracy are often at odds and depend on population size. It is beneficial to determine the population size where little accuracy is gained from larger populations. Researchers have attempted to determine heuristics for setting population size based on the number of problem decision variables (i.e., the variable space dimension) (*Reed et al.*, 2000; *Gotshall et al.*, 2002), but there is little consensus.

Population sizes were determined for both GA optimization steps shown in Figure III.3: the pre-screening constraint violation minimizer and power value maximizer. First the minimum DO constraint was set to 10 mg/L. For the 24-hour period of August 3, 2005, this constraint bound is unobtainable so the constraint violation minimizer step is activated. Various population sizes were tested, with 10 optimization trials conducted for each size. Figure III.9 displays the resulting optimal solution values (i.e., minimum DO constraint violations) found as well as computational times. The optimal solutions found appear to be logarithmically related to population, while computational time is linearly related. There is little to no improvement for population sizes greater than 360, so this value was chosen for the water quality pre-screening optimizer population size. The DO constraint was then relaxed to the obtainable value of 5 mg/L and the process was repeated, maximizing power

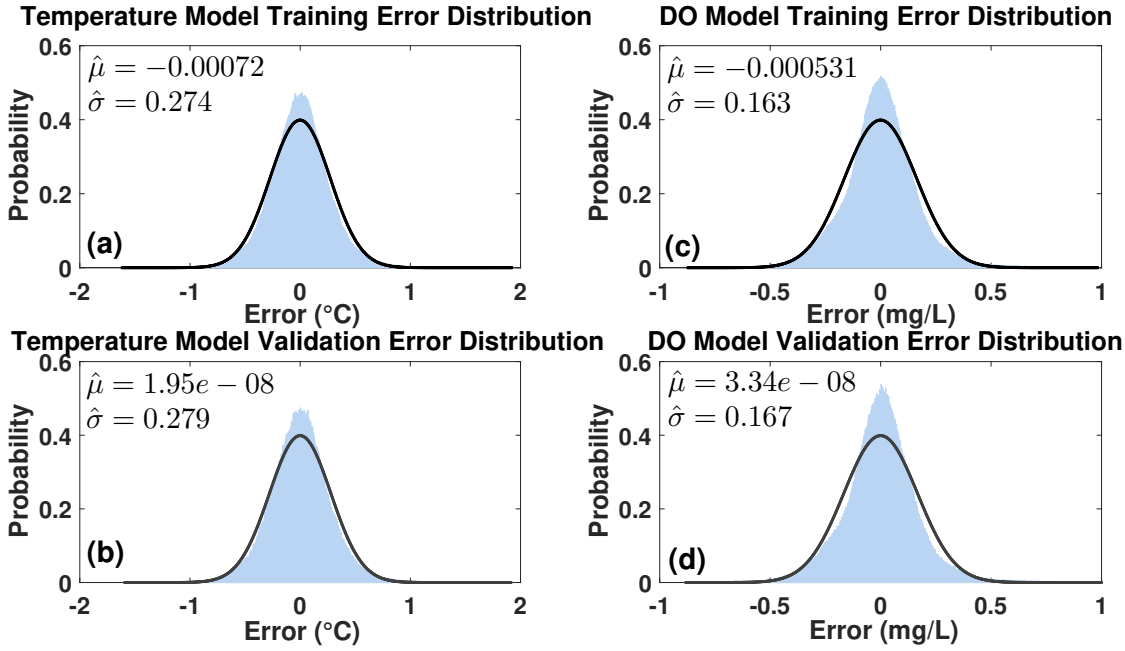


Figure III.7: Old Hickory NARX model distributions of hourly prediction errors for (a) temperature training, (b) temperature validation, (c) DO training, and (d) DO validation sets. Normal distribution fits are shown by the curve.

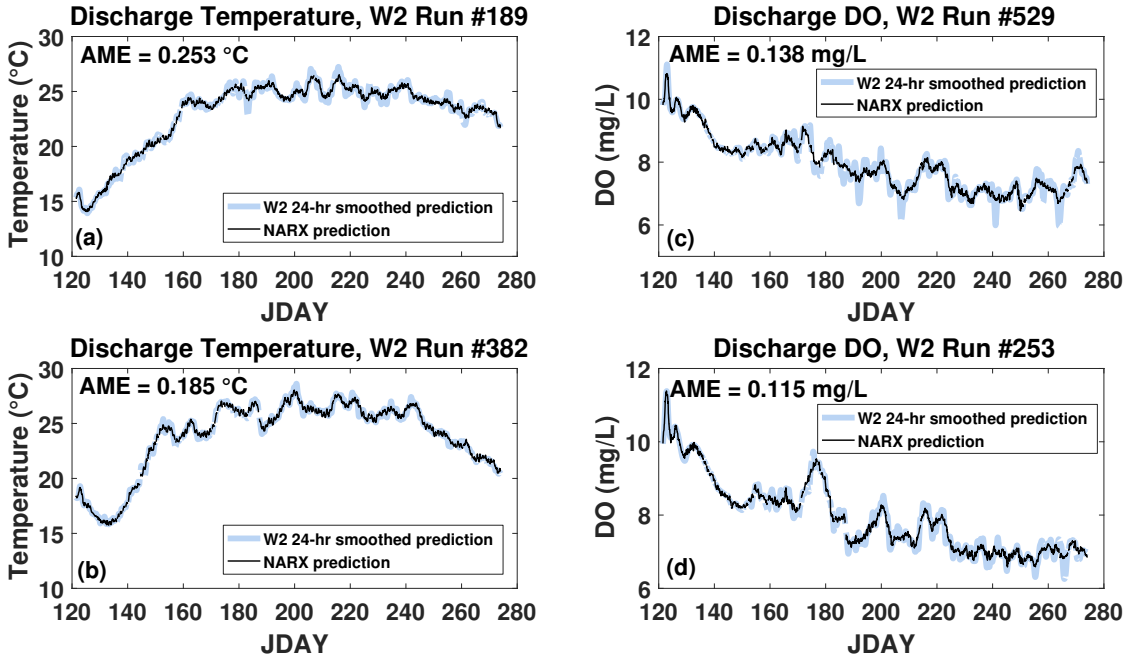


Figure III.8: Examples of validation simulation results for (a-b) Old Hickory discharge temperature and (c-d) Old Hickory discharge DO. Discontinuities in the curves represent times with neither spill nor turbine discharge present. CE-QUAL-W2 outcomes shown here and used in initial NARX training are smoothed on a 24-hour moving average.

value over the same 24-hour period using various GA population sizes. These results are shown in Figure III.10, and a population size of 480 was selected. All other GA parameter settings were determined by trial-and-error and are provided in Table III.3.

The optimization methodology is demonstrated on Old Hickory reservoir over the 10 day operating period from midnight 3 August through midnight 13 August 2005 (Julian days 215-225). This represents a period in the summer when the reservoir is vertically stratified and water quality issues appear in the reservoir and tailwater. In order to demonstrate the effectiveness of this tool for improving water quality and the impact that high-fidelity WQM incorporation can have on optimal power generation solutions, two experiments were performed. First, the relationship between maintaining several stages of constraints on DO and the resulting energy production was explored. Second, reservoir operations were optimized under constraints on both discharge temperature and DO. Computations were performed on a server equipped with 64-bit Windows Server 2008 R2 Enterprise and two 3.10 GHz AMD® Opteron™ 4334 triple core processors. As stated earlier, GA solvers are capable of, but not mathematically guaranteed, to find globally optimal solutions; therefore, comparisons to historical operations are provided to show improved performance of solutions found by GA.

III.6.1 Experiment 1: Trade-Offs Between Water Quality and Energy Production

Optimization constraint values were set to those listed in Table III.4, with the addition of a lower constraint on discharge DO (o_l). Operations were optimized under a series of values for this constraint, ranging from $o_l = 5$ mg/L to $o_l = 8$ mg/L. While this experiment could be formulated as a multi-objective optimization problem, the purpose of the developed methodology is for implementation for a system with known regulatory water quality limits, not for determination of a trade-off point between discharge water quality and power production, so the additional computational expense required to solve a multi-objective problem is not beneficial to the intended usage. This presentation intends to demonstrate how the algorithm returns results that agree with the standard practice of incorporating additional spill release to reduce negative water quality outcomes. During the constraint feasibility pre-screen step, surface elevation and discharge DO constraints were prioritized in that order, respectively. Target elevations were set to match the elevation pattern from

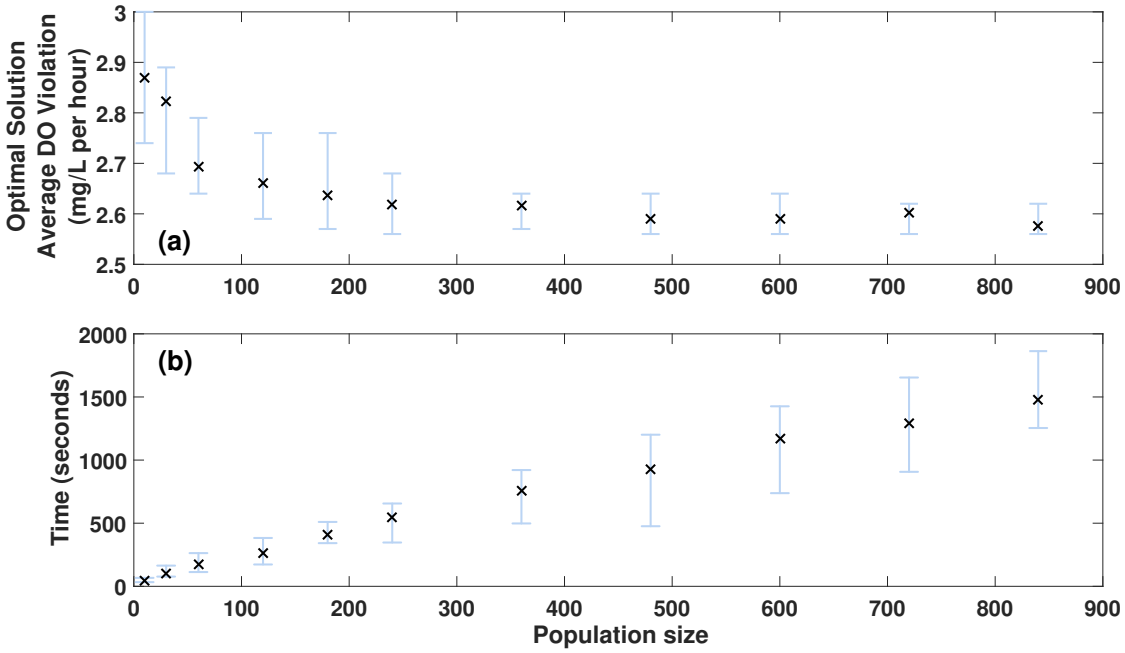


Figure III.9: Results of population size parameter tuning for DO constraint violation minimization optimization routine, showing (a) Optimal solutions found, and (b) Optimization time. Error bars represent the range of solutions for the 10 evaluations made per population size.

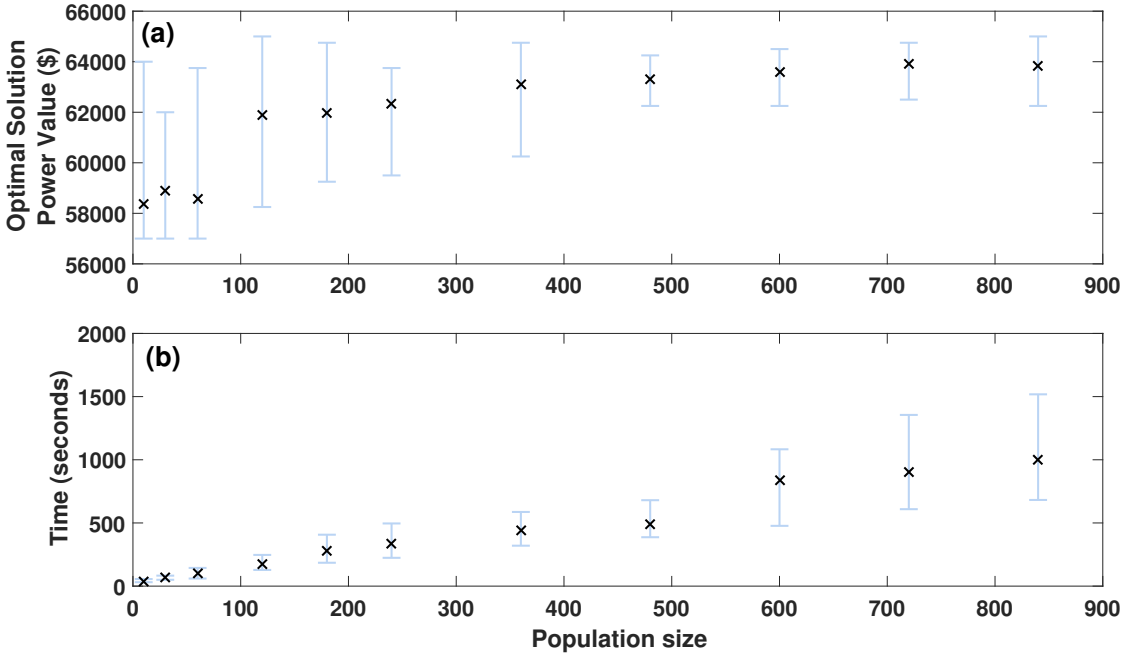


Figure III.10: Results of population size parameter tuning for power value maximization optimization routine, showing (a) Optimal solutions found, and (b) Optimization time. Error bars represent the range of solutions for the 10 evaluations made per population size.

Table III.3: Optimization parameter settings.

	Constraint Violation Minimization	Power Value Maximization
Population size	360	480
Objective function tolerance	10^{-8}	
Constraint function tolerance	10^{-8}	
Crossover function	single point	
Crossover fraction	0.85	
Mutation function	integer Gaussian	
Mutation fraction	0.15	
Creation function	integer uniform	
Selection function	stochastic uniform	
Elite count	ceiling(0.05*population size)	
Stopping criteria:		
Maximum number of generations	50	
Maximum stalled generations (no more than)	2	3
Fitness limit	0	not applicable

recorded operations over this time period.

Table III.5 summarizes the 4 optimization trials performed. During this 10-day period in 2005, recorded operations resulted in 10,450 MWh produced with a value of $812,750 using the assumed cost curve. For lower DO constraint limits of 5 mg/L and 6 mg/L, greater power values were achieved by the optimization routine. As the DO constraint becomes more restrictive, computational time increases and the value of the power generated decreases. Additionally, the DO constraint is not fully satisfied during the entire planning period for the last two cases.

Figure III.11 shows the cumulative turbine and spill releases at the optimal operations for each constraint level. Additional spill is required to maintain the desired DO concentration level when

Table III.4: Optimization constraint values.

Turbine power rating, r (MW)	25
Number of turbines, a	4
Power pool elevation upper bound, p_u (meters)	135.636
Power pool elevation lower bound, p_l (meters)	134.722
Maximum zero-generation hours, z	6
Rate of change of active turbines, c (turbines/hour)	1
Minimum discharge DO concentration, o_l (mg/L)	varies
Minimum discharge temperature, t_l (°C)	varies

Table III.5: Summary of Experiment 1 and Experiment 2 results.

	DO Constraint	Temperature Constraint	Iterations Required	Time (minutes)[a]	Mean Hourly DO Violation (mg/L)	Mean Hourly Temperature Violation (°C)	Energy Produced (MWh)	Generation Value ($) [b]
Exp. 1:								
	$\geq$ 5 mg/L	—	3	190.8	0	—	10050	$868250
	$\geq$ 6 mg/L	—	4	254.3	0	—	10100	$866500
	$\geq$ 7 mg/L	—	16	997.3	0.035	—	8825	$730000
	$\geq$ 8 mg/L	—	14	1436.1	0.344	—	2600	$171500
Exp. 2:								
	$\geq$ 7 mg/L	$\geq$ 25 °C	13	1915.9	0.005	0.071	4300	$316000

[a] Time to complete optimization of series of sub-problems, not including CE-QUAL-W2 simulation or NARX retraining time.
[b] Generation value determined using assumed cost curve shown in Figure III.2.

the DO constraint threshold is greater. In the case when $o_l = 8$ mg/L, this results in more release by spill than by turbine.

III.6.2 Experiment 2: Simultaneous Constraints on Temperature and DO

Optimization constraint values were set to those listed in Table III.4, with the addition of lower constraints on discharge DO ($o_l = 7$ mg/L) and temperature ($t_l = 25$ °C). These constraints represent potential requirements for a downstream sensitive aquatic species, as seen in the past elsewhere in the Cumberland River system (*Andrews*, 2014). During the constraint feasibility pre-screen step, constraints were prioritized in the following order: surface elevation, discharge DO, and discharge temperature. Figure III.12 illustrates the resulting flowrates, elevations, and discharge water quality predictions at the identified optimal solution. The surrogate water quality predictions cannot replicate CE-QUAL-W2 predictions with zero error, but the temperature and DO surrogate models successfully captured overall trends and provided improved predictions at "peaks and valleys" over those seen in the robust model (see Figure III.8) due to retraining the model using improved training data found by the optimizer. Additional results are detailed in Table III.5. Employing the assumed cost curve, the power value of the optimized solution over the 10 day period is $316,000, as compared to the projected (or actual) operations value of $812,750 due to the introduction of spill release in order to meet water quality constraints.

III.7 Discussion

Water quality prediction computational time through the chosen operating period was reduced from approximately 6 minutes to 2 seconds per operations scenario by use of a NARX ANN surrogate model rather than CE-QUAL-W2. Optimization computational time increases as feasible space shrinks due to constraints, and additional iterations are necessary for algorithm convergence for stricter water quality limits; however, for all experiments shown there are considerable computational cost savings as compared to expense should CE-QUAL-W2 be directly embedded within the framework. For perspective, Experiment 2 required 313,423 objective and constraint function-pair evaluations per iteration on average. This depends on the optimization problem characteristics, not the form of the simulation model embedded within. The optimization problem demonstrated has a

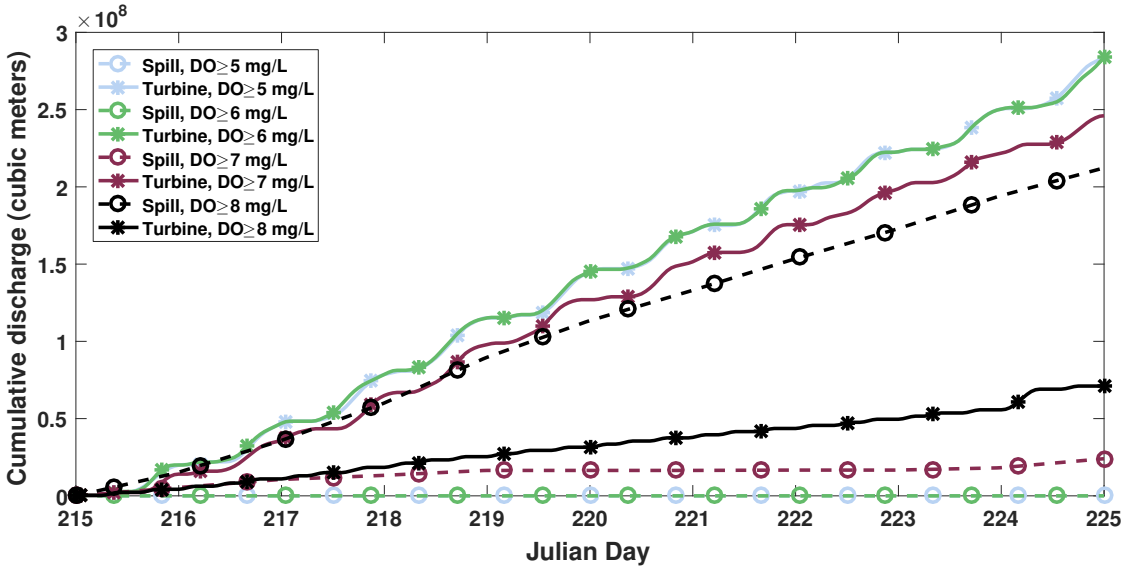


Figure III.11: Cumulative spill and turbine discharges over 10-day planning period for various minimum discharge DO constraint levels.

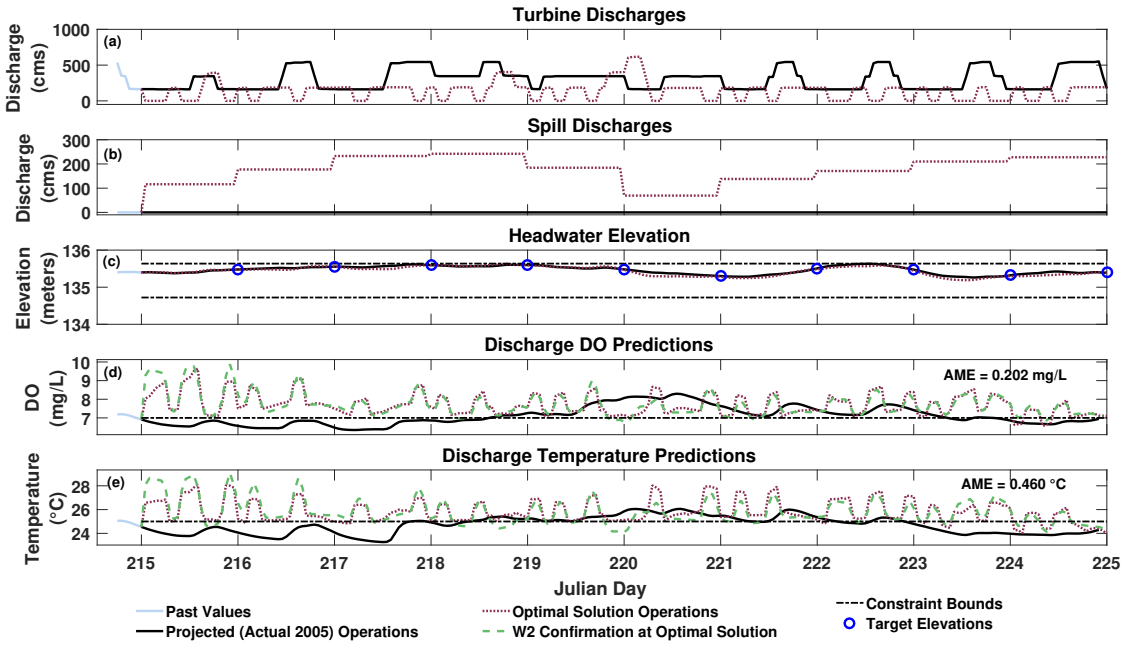


Figure III.12: Experiment 2 results for optimization of Old Hickory reservoir operations for a 10-day planning period: (a) Turbine discharge flowrates, (b) Spill discharge flowrates, (c) Headwater elevations, (d) Discharge DO predictions, and (e) Discharge temperature predictions. AME values represent absolute mean error between the NARX and CE-QUAL-W2 model predictions at the optimal solution.

large number of decision variables, is highly-constrained, and is highly nonlinear with many discontinuities; this means a greater number of function evaluations are required in order to have confidence in the GA's outcomes. Therefore, because the number of function evaluations required during GA optimization is considerably greater than the number of CE-QUAL-W2 simulations required for initial NARX training, the surrogate-enabled framework provides computational savings overall despite the necessary initial simulations and training. Further, completing 313,423 water quality predictions using CE-QUAL-W2 in parallel on the 6-core machine employed here would alone require over seven months, as compared to the 40 hours in total spent for the iterative surrogate-enabled optimization routine. The surrogate models are not perfect emulators of CE-QUAL-W2, which is why the overall surrogate-embedded framework is iterative, has retraining steps between iterations, and includes final confirmation by CE-QUAL-W2. Based on the large number of function-pair evaluations required to solve each optimization iteration, these additional steps add considerably less time than a single, non-iterative optimization approach with CE-QUAL-W2 embedded.

This routine requires several computing steps prior to optimization, including CE-QUAL-W2 model construction, calibration, and validation; design and implementation of CE-QUAL-W2 experiments to inform the surrogate model; and NARX architecture design, model training, and validation. CE-QUAL-W2 construction, calibration, and validation should be performed by an experienced modeler with knowledge of the river system. With careful implementation and data management, the design of experiments and NARX model training can be performed as an automated process. NARX architecture design can also be automated but should be supervised to ensure reasonable performance.

The relationship between spill and turbine releases and tailwater quality demonstrated by the results is in agreement with current Nashville District operator experience. During periods of water quality stress, a portion of discharges are diverted from the turbine release to the spill release to alleviate this stress. Old Hickory reservoir's operators currently make this determination based on past operator experience, and the exact amount of spill necessary in a specific situation to result in water quality compliance is unknown. In the Old Hickory case study, too little spill release results in suboptimal water quality outcomes and too much spill release results in unnecessary loss of potential hydropower production. The optimization methodology returns optimal turbine and spill release for scheduling on an operations timescale, reducing potential for downstream water quality

noncompliance and unnecessary loss of potential energy production.

III.8 Conclusions

This study demonstrated development and application of a novel method to optimize the value of hydropower production under a variety of operational constraints, including constraints on tailwater water quality, for hourly operations over a 10 day planning period for a USACE reservoir with turbine and gate control structures. The high-fidelity CE-QUAL-W2 model was employed to generate data for training NARX ANN models for prediction of discharge temperature and DO as a function of exogenous inputs, including upstream inflows, meteorological data, and structure releases. NARX models trained using an initial set of 729 CE-QUAL-W2 simulations were employed initially, GA optimization performed, and when necessary the NARX models were retrained using a CE-QUAL-W2 simulation at the discovered optimal solution, and optimization repeated. The retraining step is important in cases when the GA explores regions of the decision space not captured in the original training set, which is likely to occur in complex applications. Surrogate validity outside of the training region is difficult to evaluate and should be further researched (*Castelletti et al.*, 2012).

This methodology could be applied to other water quality constituents of concern such as total dissolved gas, phosphorus, nitrogen, or suspended sediments. Water quality at a single monitoring location is the focus here, but the process could be adapted to address water quality at additional point locations or to incorporate a metric for average water quality based on high-fidelity simulation outputs. This type of application would be valuable for assessing the impacts of river operations at water withdrawal locations for thermal and water treatment plants as well as known locations of protected species. Additionally, this approach can be applied over longer horizons on a less-refined timescale for seasonal or yearly planning; however, accuracy of inflow and meteorological forecasts must be considered for longer-term applications. For reservoirs with storage facilities, the problem could be reformulated with the end of day water level constraints as decision variables in a bilevel optimization problem; however, this adds computational expense. Efforts are currently underway to expand this methodology to a system of multiple controlled reservoirs. Future efforts include exploring additional means for improving constraint handling (*Ilich and Simonovic*, 2001), ANN

retraining (*Yan and Minsker*, 2006), and overall computational efficiency.

Chapter IV

ADAPTIVE NEURAL NETWORKS FOR EFFICIENT WATER QUALITY-CONSTRAINED HYDROPOWER OPTIMIZATION

IV.1 Introduction

Hydroelectric power generation serves as both a renewable energy source and a flexible power supplement for baseload generation (i.e., fossil and nuclear power production) during times of peak demand (*U.S. Department of Energy*, 2016b). Hydropower is expected to account for 27% of the anticipated growth in worldwide renewables production and 1.7% of the growth in U.S. renewables production through the year 2040 (*U.S. Department of Energy*, 2016a). This growth in power production must be achieved while fulfilling other reservoir objectives and constraints. Hydropower facilities and their impounded backwater serve many roles, including power production, navigation, recreation, water supply, and flood control. Hydropower operations can have environmental impacts, particularly due to releasing water on a peaking schedule in order to supply electricity to the grid during periods of high demand (*Jager and Smith*, 2008). Additionally, reservoir thermal stratification (i.e., when surface layers are warmer than deep layers, thereby reducing or eliminating vertical mixing) can be exacerbated by hydropower release decisions (*Dortch*, 1997).

This chapter demonstrates an approach for optimizing operating schemes, with a focus on efficiently determining hydropower outflow allocations while treating water quality impacts as operational constraints.This is accomplished by embedding an ANN WQM, a surrogate of a complex WQM, within a GA optimization framework and adaptively training the ANN model within the GA. Offline or static training alone, performed prior to the optimization run, results in poor accuracy for problems with complex search spaces; the broad sampling of training points may not produce accurate solutions in local regions and lead to false optima (*Yan and Minsker*, 2006).

This issue compounds when the feasible space is bounded by a set of constraints, as shown here. In many real-world constrained optimization problems, optimal solutions lie along constraint boundaries. When constraints depend on an approximation model and the true optimal solution lies

along the constraint boundary, offline training alone can produce solutions that are infeasible. Adaptive surrogate model updating within an optimization process balances exploration of the decision space and exploitation in regions with suspected optima. As the optimizer proceeds towards convergence, the surrogate model is updated to improve prediction quality in the region being searched. Building upon *Shaw et al.* (2017), we demonstrate how adaptively updating a surrogate WQM embedded within a population-based hydropower optimization routine improves solution quality. We know of no prior work that employs adaptive ANNs for constraint formulation within a GA routine, let alone for a hydropower optimization application.

IV.2 Adaptive Linked Neural Network-Genetic Algorithms

In offline ANN training, a set of potential model inputs is typically randomly generated and simulated by the original model that is to be approximated. The outputs of these runs are used to train an ANN approximator, which is then employed to solve for optima. This approach can perform well for simple problems (*Zou et al.*, 2007) with appropriate sampling. High-fidelity simulators typically model complicated relationships with nonlinearities, discontinuities, and local minima, making it difficult to develop an offline sampling plan. For problems requiring high-fidelity models, there is a need to employ surrogate models to solve within computational budgets, and offline ANN training alone is not likely to produce satisfactory optimization results. For these applications, an approach in which the ANN is updated with new information during its application within an optimization routine is necessary.

Yan and Minsker (2006) developed an adaptive ANN-GA approach and applied it to a groundwater remediation design optimizer. The full WQM was the linked multi-layer two-dimensional flow and transport model MODFLOW-MT3DMS. They implemented a caching system to improve performance by using the true WQM outcomes as the fitness values for population members previously sampled by the full WQM, and used the ANN for fitness value estimation otherwise. This reduced the number of calls to the ANN while improving GA performance. The authors concluded that the adaptive ANN with caching approach saved more than 85 percent of the full WQM evaluations required by the GA if solved without use of ANN surrogate models, while returning comparable quality solutions. This approach was later modified to account for sampling noise (*Yan*

and Minsker, 2011).

Zou et al. (2007) demonstrated an adaptive linked ANN-GA method to calibrate a computationally-expensive eutrophication model, where an ANN is used in place of the simulation model within the error-minimizing fitness function. For each final GA population, candidate solutions were grouped into clusters. The best solutions from each cluster formed a new set of simulations to be performed using the full WQM and then used for ANN updating. The authors note that this approach improved ANN capability for a particular desired usage rather than overall generalization (i.e., the goal of offline, one-step ANN training).

IV.3 Case Study Description

Here, we demonstrate the methodology by solving for optimal operations at Old Hickory reservoir; Old Hickory operations are described in detail in Chapter III and *U.S. Army Corps of Engineers* (1998). During the warm summer months, temperature and DO concentrations are highly stratified both longitudinally (along the direction of river flow) and vertically directly upstream of outlet structures. To better predict stratification conditions, the Nashville District uses the high-fidelity CE-QUAL-W2 model to simulate water quality throughout the reservoir as well as at reservoir discharge locations.

Outlet structures include tainter gates for spill flow and 4 Kaplan hydropower turbine units, each with a capacity of 25 MW. Release projections are made typically on an hourly or finer timescale 10 days in advance, and then updated daily. These projections consist of the number of active turbines to be used over time, as well as projected spill volumes. Spill releases are used when heavy precipitation is expected and operators are planning for flood conditions, and also as a means to improve discharge DO concentrations by incorporating oxygenated spill water when flow through the turbines has a low DO concentration.

IV.4 Optimization Problem Formulation

The adaptive optimization approach is demonstrated using the hydropower optimization problem defined in Chapter III, which has the objective of generating maximum power value subject to several operational constraints. USACE operations forecasting plans for the Cumberland River

system are typically generated over a 10 day period and updated daily with a focus on the next day's operations (*U.S. Army Corps of Engineers*, 1998); the optimization scenario thus covers 1 day, or 24 hours. In order to demonstrate the adaptive ANN training approach, a single water quality constraint was applied (representing a lower bound on DO) and no constraint on temperature was considered.

IV.5 Methodology

Given unlimited computing resources, Equation III.7 would be solved using a high-fidelity WQM; however, high-fidelity models are computationally expensive, and therefore ill-suited to be used within optimization routines. Considering this limitation, water quality estimates are determined by a surrogate model, which is trained using the original model simulation outputs. The high-fidelity WQM here is CE-QUAL-W2, and the surrogate model is formulated as an ANN of the NARX form.

GA optimization is a flexible method that is capable of handling nonlinearities and discontinuities, as well as quasi-black box functions including ANNs as present in the objective and constraint functions noted above. The GA functionality in the MATLAB® Optimization Toolbox (R2016a, The MathWorks Inc., Natick, Massachusetts, United States) was used in this application. GA operators consist of elitism (where the best population members are passed directly to the next generation) and score-weighted selection for creating mutation and crossover children. For nonlinear constrained optimization problems, penalty and augmented Lagrangian methods attempt to evolve populations toward the feasible space when determining candidate solution fitness; here, an Augmented Lagrangian GA (ALGA) approach is used (*Conn et al.*, 1997).

An approach for incorporating an adaptive ANN-based constraint on water quality within a GA-based hydropower optimization process for determining hourly turbine releases is shown in Figure IV.1. First, an initial population of potential solutions that satisfy all hard constraints with the exception of water quality was created. Using a problem-specific creation function that accounts for constraint equations III.2-III.6, the initial population of operating scenarios is constructed using logical decision-making and random selection, starting with the first hour's turbine setting and progressing to the final hour. Using historical hourly operations data from 1987-2015, transition probabilities were determined for ramping up, ramping down, and maintaining turbine levels given

the previous hour's active turbines.

The initial population is divided into K clusters by the k-means method (*MacQueen*, 1967), which increases the diversity of training data sampling points. A member from each cluster is chosen to create a set of K training members. Additionally, if the operator's anticipated turbine operating pattern satisfies the constraints it is included in the initial population set; this manuscript uses past operations for demonstration, so they are used here in the place of anticipated operations. The initial NARX model is constructed by simulating the selected training members and the anticipated operations with the full CE-QUAL-W2 model and training the surrogate using the resulting set of outputs. The initial offline training builds a surrogate with broad predictive coverage over the design space, but poor detailed predictive capability. A cache of full CE-QUAL-W2 model outputs is updated every time the CE-QUAL-W2 model is called. In future generations if a previously-sampled population member is present, then the water quality estimations are provided by the cache rather than the NARX model. While NARX model estimation errors will never be fully eliminated, the caching step eliminates error at points where WQM results are known.

With the initial population assigned and offline surrogate training complete, the process enters into a GA phase stepping forward one generation. The feasible population member with the best fitness value is saved as x_{gen}. If the new population appears to contain no feasible solutions, then the member with the smallest constraint violation is chosen. If necessary, x_{gen} is simulated using the full CE-QUAL-W2 model, and if the stopping condition is not met an additional selection of A population members is chosen for additional sampling with CE-QUAL-W2. The surrogate model is updated using the expanded training data set. In an effort to improve population diversity, a random immigrants step is employed in which a portion of the population is replaced with new randomly generated new population members.

The following subsections further describe the resampling for ANN updating and the random immigrants replacement steps.

IV.5.1 Resampling for ANN Adaptation

The surrogate WQM's range of predictive power depends on the set of training data it is built upon. Even a very large randomly-generated training data set can yield a surrogate model that can-

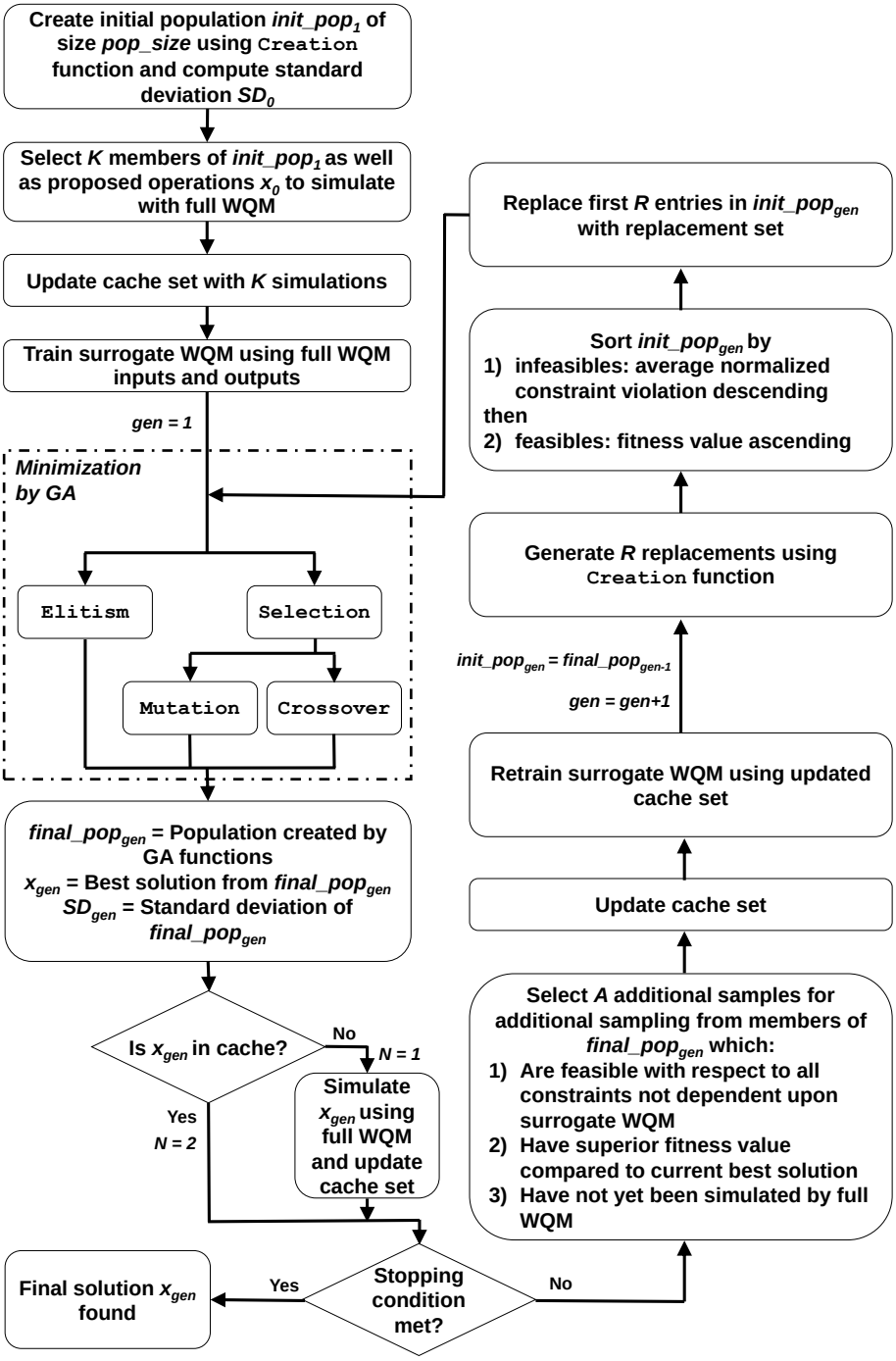


Figure IV.1: Framework for adaptively-trained ANN water quality constraint within GA-based hydropower optimization routine.

not produce reliable predictions in the region of optima. The locations of these optima are unknown prior to optimization, so an approach to choose additional training points within GA optimization is employed here. As the GA progresses, the population of potential solutions converges to a set of solutions with improving fitness and estimated feasibility. Additionally, as the surrogate model improves from generation to generation, the estimated feasibility error reduces. For these reasons, generating training data from these improving populations is more beneficial than by random selection.

After the traditional GA step and x_{gen} simulation with the full CE-QUAL-W2 model, an additional A samples from the population are simulated by CE-QUAL-W2. This not only provides additional training data to the ANN, but also eliminates prediction errors at these points going forward. The members of $final\_pop_{gen}$ considered for additional sampling are (i) feasible with respect to all constraints that are not dependent upon the surrogate WQM, (ii) have superior fitness values than the previous best feasible solution found, and (iii) have not yet been simulated using CE-QUAL-W2. These members are sorted primarily by the ANN-dependent constraint violation and secondarily by fitness value, both ascending. The highest ranked members are then selected (i.e., those approximated first as most feasible, and secondarily of best fitness) for simulation.

IV.5.2 Random Immigrants Replacement

Dynamic optimization problems (DOPs) are those in which the problem (i.e., the decision variables or the objective or constraint functions) changes during the solution-solving process (*Tinos and Yang*, 2007). For DOPs, intermediate potential solutions may no longer be effective going forward. One method for approaching DOPs is to restart the optimizer under the new conditions, which is computationally inefficient. In cases like the one shown here, changes in the problem are related to the trajectory of the optimizer, and there are techniques which use prior solutions to move forward under a problem's new conditions. Methods such as hypermutation and random immigrants replacement aim to avoid premature GA convergence by improving population diversity, thereby improving the algorithm's effectiveness for solving DOPs. Hypermutation is triggered when changes in the DOP are detected based on current population members; however, the current population may not represent the search space where changes are occurring, in which case hyper-

mutation can fail (*Grefenstette*, 1992). Alternatively, random immigrants is a method in which a portion of the population is routinely replaced with new members, inspired by immigrants entering a biological population. Studies have found the random immigrants approach to be favorable for solving problems whose response surface (i.e., objective and/or constraint function outputs) changes dynamically during searching (*Grefenstette*, 1992; *Tinos and Yang*, 2007).

The optimization formulation here exhibits such dynamic changes, since water quality predictions change each time the surrogate model is updated. The replacement rate R is the percentage of the population members to be replaced in each generation. In each generation, the optimizer replaces the least desirable population members. The algorithm ranks population members by weighted average constraint violation by normalizing the violation of each constraint across all population members, averaging across all constraints, and then ranking from least to most feasible. Ranking fully feasible members is also dependent upon fitness, with best fitness value ranking last. The optimizer then replaces the earliest-ranked (i.e., the least desirable) R percentage of the current population with new members generated by the creation function used to generate the initial population.

IV.6 Experimental Setup

The Nashville District USACE provided a CE-QUAL-W2 version 3.5 model for Old Hickory reservoir. This model underwent calibration and validation steps for prediction of water level, temperature, and DO (*Shaw et al.*, 2017). Sensitivity testing using the model determined an appropriate NARX model architecture for predicting tailwater DO, as this location is considered the water quality point of compliance and monitoring by dam operators. Selection of NARX model architecture, including number of neurons, layers, delays, and exogenous variables set, is described in *Shaw et al.* (2017).

Using the problem formulation described earlier, operations at Old Hickory reservoir were optimized on an hourly timestep from midnight 3 August through midnight 4 August 2005 (Julian days 215-216). This date was chosen because 2005 was the validation year during CE-QUAL-W2 model development, so operations and water quality data were available. Additionally, during this period in late summer the reservoir is vertically stratified and water quality issues influence operations. Old

Hickory has a power rating r of 25 MW for each of its four turbines. The employed cost curve $C(i)$ (see *Shaw et al.* (2017)) was created from historical operating patterns. Lower and upper bounds on water levels were based on the USACE guide curve and set to $p_l = 134.722$ m and $p_u = 135.636$ m, respectively. The maximum number of consecutive hours allowable without generation is $z = 6$, and the turbine rate change limit is $c = 1$ turbine/hour. The minimum DO concentration at the dam discharge is set to $o_l = 7$ mg/L; the true operations on this day in 2005 resulted in DO concentrations below 7 mg/L, so this setting provides an adequate demonstration of how the methodology successfully discovers the feasible space under a demanding constraint limit.

The problem was solved by four approaches: (i) without random immigrants replacement or adaptive sampling (beyond simulating the best solution x_{gen} after each generation), (ii) with replacement but without additional sampling, (iii) with additional sampling but without replacement, and (iv) full adaptive framework shown in Figure IV.1 including additional sampling and random immigrants replacement.

Surrogate retraining is performed in all four scenarios; this means the surrogate model changes between each generation, but retraining is influenced by the introduction of additional training data beyond x_{gen} in cases 3 and 4. Based on previous work, the GA population size, pop_size, was set to 480. Additional GA settings are provided in Table IV.1. In cases 2 and 4, the replacement rate $R = 0.2$ was chosen. In cases 3 and 4, the number of additional samples simulated by the full CE-QUAL-W2 model is four for generations where x_{gen} does not require simulation and three for generations where x_{gen} requires simulation by CE-QUAL-W2, resulting in a total of four CE-QUAL-W2 simulations per generation.

IV.7 Results

Because the methodology employs random number generation, each experiment was performed eight times, with each of the four approaches tested using the same eight random number generator seeds. The power values (first term in Equation III.1) of the resulting best feasible solutions are provided in Table IV.2. Figure IV.2 shows the power value means and ranges, as well as the means and ranges of the ANN function call and CE-QUAL-W2 simulation counts for each case. All trials returned a solution in the feasible space. As expected, Case 1 performed the worst, with all eight

Table IV.1: GA and overall framework settings.

Population size (*pop_size*)	480
Objective function tolerance	10^{-20}
Constraint function tolerance	10^{-20}
Creation function	logical decision-making with constraint consideration and random selection
Elite count	ceiling(0.05*population size)
Crossover function	single point
Crossover fraction	0.95
Mutation function	integer Gaussian
Mutation fraction	0.05
Selection function	stochastic uniform
Max generations (stopping condition)	50
Initial training set size (K)	10
Additional samples for ANN training (A)	3 or 4
Replacement rate (R)	0.2

trials converging on local minima. The addition of either random immigrants to the population pools or adaptive additional sampling improved solutions. Implementing random immigrants and additional sampling together yielded the best results, improving solutions by 8.5% on average over Case 1. Cases 3 and 4 have a similar range of results, but on average Case 4 performed best. Further analysis of the Case 4 trials revealed decreasing solution improvement as the optimizer proceeds (Figure IV.3). This means later generations provide smaller gains in power value than provided by earlier generations.

Random immigrants replacement aims to counter premature homogenization of the population.

Table IV.2: Power values for best feasible solutions found by the four approaches in eight trials.

Random Number Generator Seed	Case 1: Without Replacement or Additional Sampling	Case 2: With Replacement & Without Additional Sampling	Case 3: Without Replacement & With Additional Sampling	Case 4: With Replacement & With Additional Sampling
1	42000	43750	43500	45250
2	41500	43750	47500	44250
3	40500	44250	47750	47750
4	43750	43750	43750	47750
5	45250	46500	47500	47750
6	43750	47500	43750	43750
7	42500	44000	44250	47500
8	43000	41000	44250	47500

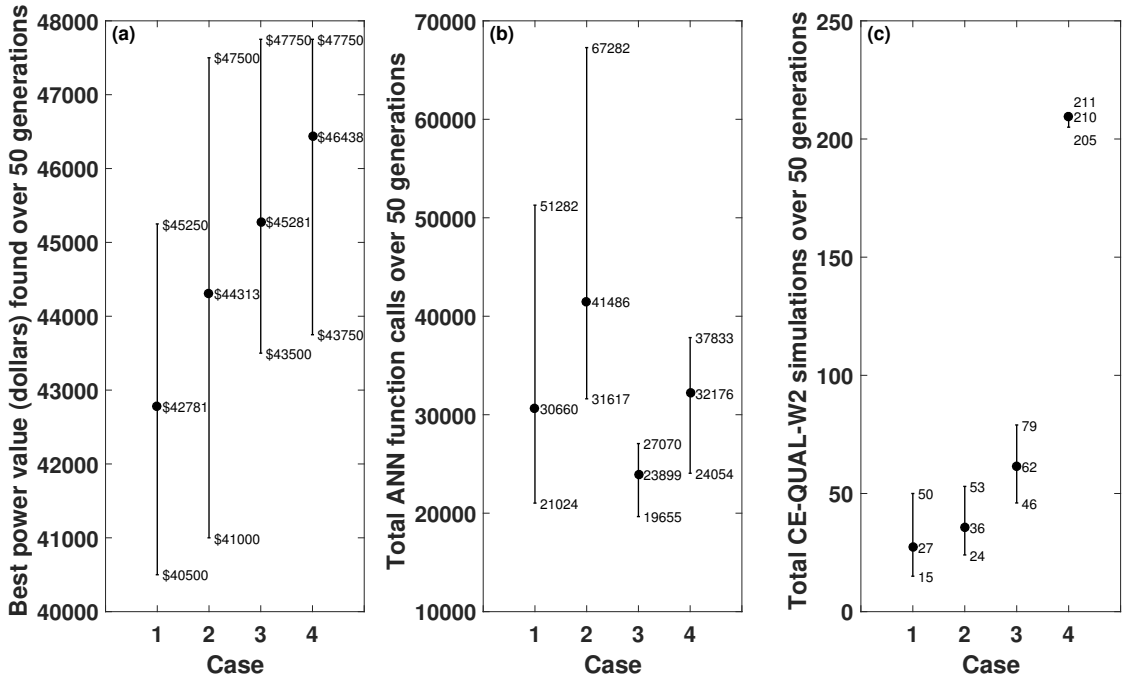


Figure IV.2: Means and ranges for (a) power values of the best feasible solutions found, (b) total ANN function calls, and (c) total CE-QUAL-W2 simulations for the four tested cases.

The average standard deviation of the decision variable makeup in each generation, SD_{gen}, is a metric which can demonstrate replacement's impact on population diversity. Average standard deviations are calculated by scaling decision variables to a $[-1.0, 1.0]$ range, calculating the standard deviation of each variable, and averaging these values. This is computed after the GA minimization step in each generation. The population standard deviations at each generation for each case, averaged over the eight simulations, is shown in Figure IV.4. Standard deviations are maintained at higher levels for the two approaches which included random immigrants replacement. In all four cases, standard deviations reach approximate minimums around generation 25. Because the problem is dynamically changing via the water quality constraint, standard deviations do not converge to zero as they would for a static optimization problem.

The caching step eliminates prediction errors at points which have been evaluated by the full simulation model by returning the full simulator (rather than the ANN) water quality predictions during future constraint function calls. This step also lessens the number of calls to the ANN function as optimization progresses. The percentage of the population intersecting with the cached

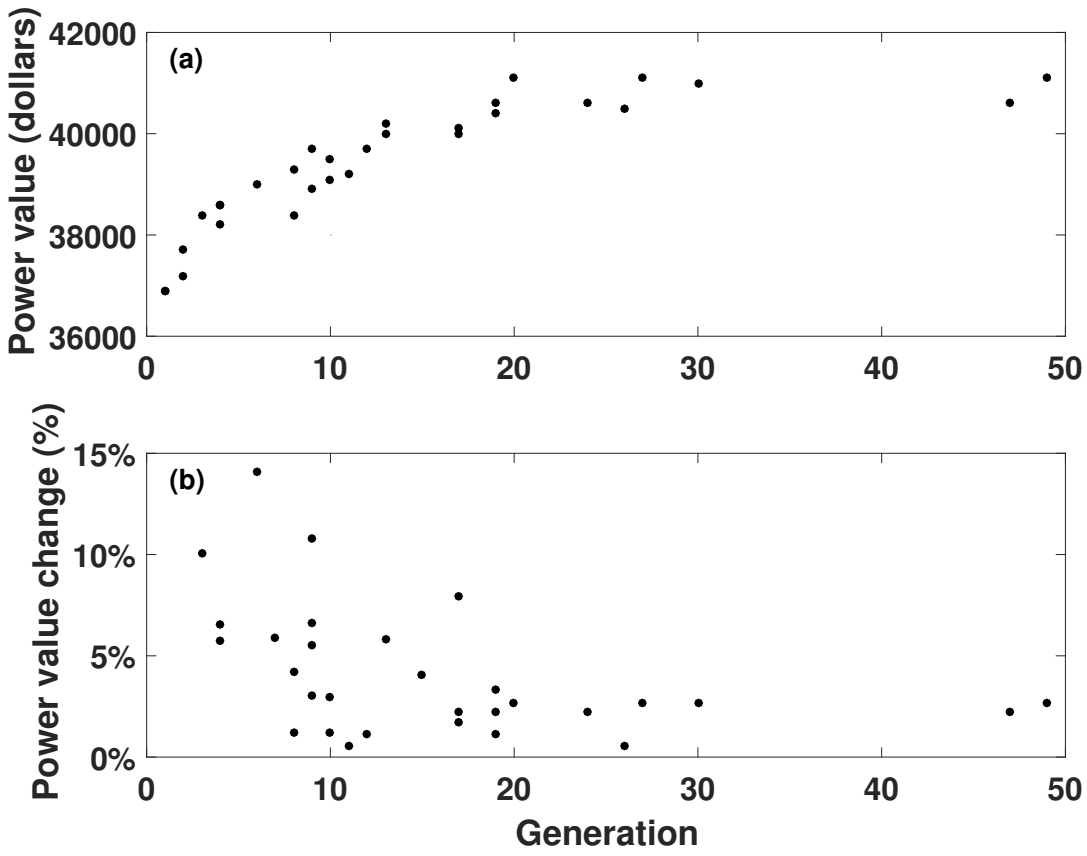


Figure IV.3: Generation number versus (a) power values for newly-discovered incumbent solutions and (b) percentage change in incumbent solution power value for the Case 4 trials.

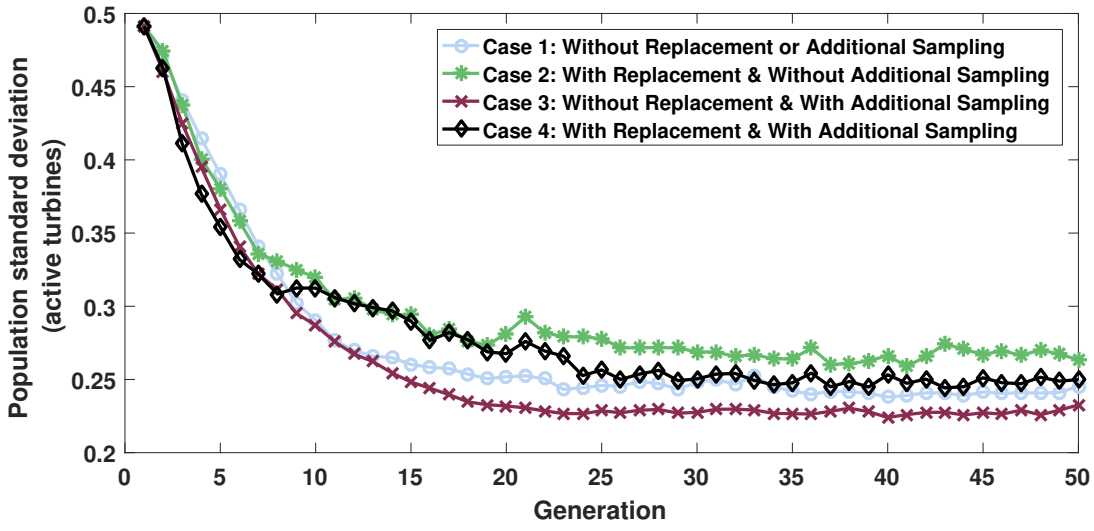


Figure IV.4: Population average standard deviations for the four tested cases.

set at each generation, averaged over the eight trials, is shown in Figure IV.5. This metric reflects on the convergence of the GA, with populations becoming less diverse as optimization progresses.

IV.8 Discussion

Figure IV.2 indicates the modified optimization methodologies (Cases 2, 3, and 4) provide solutions of superior fitness value compared to those provided by the unmodified optimizer (Case 1). This reveals turbine release patterns which provide additional hydropower revenue without forcing discharge flow DO concentrations below the minimum allowable. These refined optimal release schedules are a function of water quality and depend upon many factors, and are therefore likely unknown to hydropower decision-makers without optimization.

As seen in Figure IV.5, the test cases without replacement ultimately converge to populations with a high percentage of cached (i.e. previously simulated using CE-QUAL-W2) points, while the cases with replacement converge to populations with a lower proportion of previously simulated members. This difference is greatest for the case with additional sampling (Case 2). Additionally, the cases with additional sampling exhibit smoother growth in percentage of the population in the cache, while the cases without additional sampling exhibit occasional drops and less steady growth. The approaches providing more samples to the training data set result in ANN surrogate models with smoother changes in prediction values from generation to generation, meaning the makeup of the population from generation to generation is not dramatically altered due to adjustments to water quality constraint values. For the approaches that only provide additional training data when a new suspected optimal feasible solution is discovered and checked by the full CE-QUAL-W2 model, the training data supply is more sparse and is updated less frequently, which can cause more extreme adjustments to the surrogate model when new data is supplied.

The two cases with additional training data sampling provided the highest quality solutions overall. This comes with the computational drawback of additional calls to CE-QUAL-W2, as seen in Figure IV.2. Including the simulations used for initial NARX training, the four cases required on average 27, 36, 62, and 210 CE-QUAL-W2 simulations. There is a less-clear relationship in terms of required ANN function calls. While results indicate that Case 4 has the highest likelihood of returning a superior solution, the combined approach also has a greater computational burden in terms

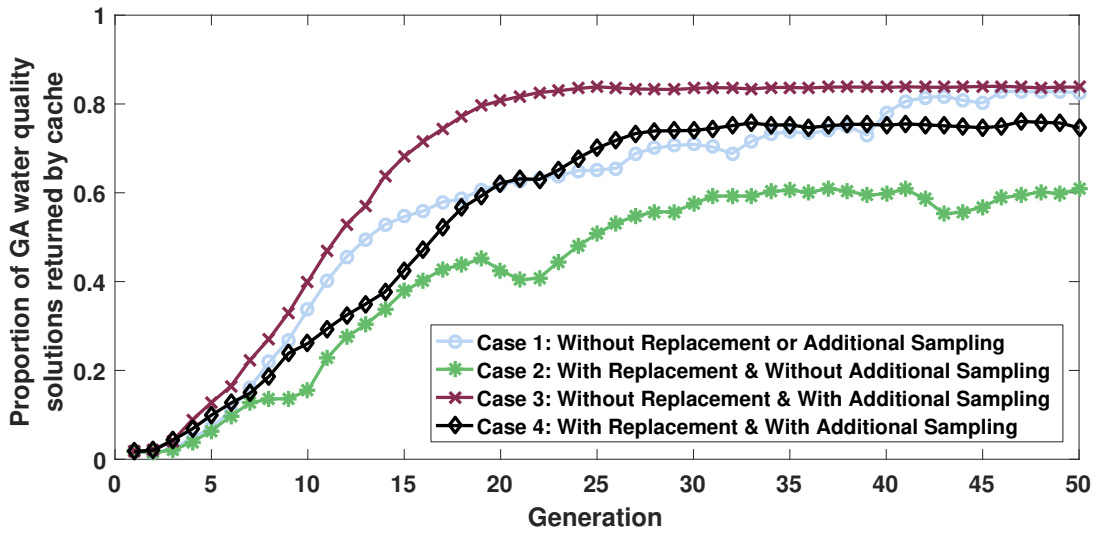


Figure IV.5: Averaged proportions of GA water quality solutions found within cache at each GA generation for the four test cases.

of CE-QUAL-W2 simulations. Translating function and simulation calls to execution time depends on problem specifics and computational resources. For this problem, ANN water quality surrogate predictions and CE-QUAL-W2 simulations each required approximately 0.08 and 55 seconds, respectively, when solved in parallel batches on a 64-bit Windows 10 computer equipped with a 3.40 GHz Intel® quad core processor. Adaptive resampling with the original simulation model, even to a limited extent, can therefore result in a notable computational expense increase. Additionally, results indicate that later optimization generations provide less solution improvement than earlier generations, so halting the optimizer early could reduce computational burden with limited impact on solution quality. These factors should be considered when determining which approach is most appropriate for future applications. As always, the modeler must consider the potential trade-off between computational expense and the benefits provided by improved solution quality.

This demonstration yielded a potential average improvement in power value of 8.5% or $3,657 for a single 24-hour interval during a period of low discharge DO concentrations. Figures A.1 and A.4 in Appendix A show a period of approximately 100 days in which discharge DO levels were measured in the vicinity of the regulatory limit of 5 mg/L. Assuming the $3,657 additional power value provided daily is realized over the 100 day period, the algorithm modifications have the potential to improve power value by approximately $360,000 while maintaining water quality

in the Old Hickory tailwater.

Using a surrogate model in place of the full CE-QUAL-W2 model delivered computational savings. The ALGA method for constrained GA optimization requires a series of subproblems to be solved within each generation; therefore, more than pop_size calls to the objective and constraint functions are made. In each generation the GA minimizer step required approximately 1450 constraint function evaluations. Over 50 generations (the chosen stopping condition), each solution required approximately 72,500 constraint evaluations. Without using a high performance computing cluster, this many evaluations of the full CE-QUAL-W2 model (which in this case takes 1-3 minutes to evaluate on a desktop computer) would not be feasible for real-world operations planning.

IV.9 Conclusions

We demonstrated an approach for solving a constrained optimization problem with a dynamically-changing constraint formulated as a ANN model, a surrogate of an expensive simulation model. The surrogate model replaces a full simulation model to reduce computational expense. Because the ANN model is not an exact emulator, prediction errors can lead the optimizer to converge on infeasible solutions. To counteract this, two approaches were tested. The first approach, random immigrants replacement, involves injecting new members within each population. This is an easily-implemented technique for increasing population diversity, which is of particular importance for DOPs. The second approach improves surrogate model prediction quality in a way that is influenced by the optimization trajectory. Additional training data samples are routinely chosen from GA populations and simulated with the full simulation model, improving surrogate performance in regions of suspected optimality.

These approaches were used to solve a high-constrained hourly operations planning problem for a single, multipurpose reservoir with hydropower capabilities. The objective was to maximize the value of power generated, while satisfying numerous constraints including a constraint on tailwater DO. DO predictions were generated by a ANN model trained to emulate the CE-QUAL-W2 hydrodynamics and WQM. Of the approach combinations tested, combining random immigrants replacement and adaptive additional sampling produced superior fitness values, and when used individually improved results over trials where neither approach was used.

Prior work in the area of adaptive model updating within optimization relies on surrogate model forms which provide statistical information (for example, GPs as used in *Bichon et al.* (2013)). Black-box emulators like ANNs do not produce the statistical information necessary to use such techniques, so a population-based resampling approach was described here. The algorithm modifications shown here could prove useful for solving any optimization problem where a population-based optimizer is appropriate, a constraint depends on an black-box inexact emulator of an expensive simulation model, and there is a need for emulator construction and/or training to be influenced by outcomes from the optimization process itself. Additional research on the level of additional sampling necessary for improved results is needed. Developing a non-problem-specific heuristic for this would be greatly beneficial when exploring additional applications for the framework shown here.

Peaking hydropower operations have been known to negatively impact river systems. The modified optimization methodology provides solutions of superior fitness value compared to those from the optimizer without the two modified features. This reveals an even greater potential for additional hydropower generation at times of peak demand than shown in *Shaw et al.* (2017), translating to additional revenue generation, without having an adverse impact on water quality. Hydropower producers are often required to make tradeoffs between power generation and water quality objectives (*Loftis et al.*, 1985). The results seen here indicate that an approach such as this is capable of discovering release patterns which improve both power generation revenue and water quality simultaneously.

Chapter V

SENSITIVITY ANALYSIS FOR INFORMED WATER QUALITY-CONSTRAINED HYDROPOWER SYSTEM OPTIMIZATION

V.1 Introduction

Previous chapters focused on optimizing hydropower production for a single reservoir subject to a variety of constraints, including constraints on water quality informed by a high-fidelity simulation model. It is often the case that reservoirs with hydropower capabilities are not operated in isolation, but are part of a larger water management system, including other hydropower-producing waterbodies. Operations that optimize power production at individual reservoirs may not provide system-wide maximal power. Upper reservoir power production also depends on lower reservoir pool levels, which are influenced by lower reservoir operations. Including water quality considerations further complicates this, as downstream water quality is driven by upstream releases.

With the advantages of flexibility and global searching, GAs have a general disadvantage of high computational expense, which increases with larger problem size. GAs do not scale well; as the number of decision variables increases, the search space becomes exponentially larger. With additional constraints, the problem may require a large number of function evaluations to find the feasible space, let alone a globally optimal solution. A GA optimizer is the foundation of the reservoir optimization routine discussed earlier. Real-time dam operations optimization for a river system with multiple hydropower facilities represents a highly-constrained large-scale problem. To efficiently implement the methodology developed for a single reservoir on a larger system of reservoirs, an approach for reducing computational expense while expanding the problem size should be explored.

One approach to counteract expanding problem size is problem segmentation. This involves breaking a large-scale problem into segments and optimizing them individually; the optimization results can then be used to solve a reduced network-level optimization problem (*Hegazy and Rashedi*, 2013). The challenge with this approach in this application is the dependencies of downstream

reservoir water quality on the releases from upstream reservoirs, downstream reservoir water availability for power production on upstream reservoir releases, and upstream reservoir head differential on tailwater elevations, which may fluctuate based on downstream reservoir pool levels. In order to fully optimize a system of reservoirs with water quality constraints, reservoirs should not be assumed to operate in isolation, without the feedforward impacts of water quality or the feedforward and feedback impacts of water balance on hydropower production. However, if it can be shown that varying operations within reasonable bounds at individual reservoirs has little or minimal impact on water quality or balance at other reservoirs in the system, a segmented approach could be a viable method for expanding the water quality-constrained optimization approach here to larger systems of reservoirs.

An extensive body of literature exists examining river and lake hydrodynamic and water quality sensitivity to changes or uncertainties in model inputs or structure. In most studies, researchers modify model structure such as resolution or dimension (*Muñoz-Carpena et al.*, 2007; *Blumensaat et al.*, 2014), calibration parameters such as kinetics rates or oxygen demands (*Spear and Hornberger*, 1980; *Reichert and Vanrolleghem*, 2001; *Sincock et al.*, 2003; *Rangel-Peraza et al.*, 2016; *Cheng et al.*, 2018), or boundary conditions such as hydrological or meteorological conditions (*Henderson-Sellers*, 1988; *Reichert and Vanrolleghem*, 2001; *van der Linden et al.*, 2015; *Rangel-Peraza et al.*, 2016) and then observe changes to model outputs. Some sensitivity analyses are intended to alert WQM users of potential impacts of uncertainties and how they may propagate through to model predictions (e.g., *Blumensaat et al.* (2014)). Other studies use sensitivity analyses to explore waterbody response to extreme boundary conditions, such as climate change scenarios (e.g., *van der Linden et al.* (2015)).

Solutions to optimization problems are sensitive to many factors, including objective and constraint functions (*Padula et al.*, 2006) and decision variable choice (*Gramacy et al.*, 2013). Constraint function uncertainty (in this case, driven by uncertain boundary conditions) is the main interest of this application. It is possible to assess solution sensitivity to linear constraints by studying marginal values and "right hand side" and coefficient ranges (*Bisschop*, 2018), but these techniques are not valid for highly nonlinear and black box functional forms. Quantifying model output uncertainties requires first identifying and characterizing all sources of uncertainty (*Eslick et al.*, 2014). Here, the uncertainty source of interest is the neighboring hydropower facility operations that for-

mulate boundary conditions. Defining this uncertainty with ranges and probabilities is not possible as the operations are human-driven, based on a large set of operational constraints, and in the case of a fully-optimized system of reservoirs also depend on their own potentially-uncertain boundary conditions.

Here, we look toward expanding the prior Chapters' work to a system of reservoirs by performing a necessary exploration of the feedforward water quality relationship between two reservoirs connected in series. The sensitivity of release water quality at the downstream reservoir due to changes in the upstream boundary condition (i.e., upstream dam operations) is examined. Because the optimization routine is designed to be used for hourly operational planning over a typical planning period, the sensitivity analysis is focused on short-term fluctuations in water quality due to changing operations, not seasonal effects. The purpose of the boundary condition sensitivity analysis is to develop a computationally efficient method for optimizing a system of reservoirs in which individual reservoirs can be handled individually and optimized in parallel. Therefore, a straightforward bracketing approach testing a range of boundary conditions, without the effort of defining uncertainty conditions, is selected.

V.2 Case Study Description

The sensitivity analysis was conducted on Old Hickory and Cordell Hull reservoirs on the Cumberland River system. These two run-of-river projects are linked in series, with Cordell Hull located upstream and Old Hickory downstream (see Figures I.3, III.1, and V.1). Both have total hydropower capacities of 100 MW; while Old Hickory's capacity is spread across four 25 MW turbines, Cordell Hull is equipped with three 33.3 MW turbines. Similar to Old Hickory reservoir, Cordell Hull is equipped to allow releases through a spillway, typically used for flood control and water quality mitigation purposes. Both reservoirs are operated on a peaking pattern, with generation greatest at times of high power demand.

The hydrodynamics and water quality behaviors of both reservoirs were modeled in 2D using CE-QUAL-W2, which is well-suited for riverine waterbodies such as these. The Old Hickory modeling efforts were described earlier in Chapter III. As with Old Hickory reservoir, Cordell Hull reservoir's CE-QUAL-W2 model was upgraded to version 3.5, calibrated, and validated. Calibra-

tion and validation time series results and water quality profiles are provided in Appendix B as Figures B.1 through B.6. Calibration and validation error metrics are summarized in Table V.1.

V.3 Methodology and Experimental Setup

This methodology explores the dependency of Old Hickory reservoir water quality on Cordell Hull reservoir releases. The chosen testing period is the same 10-day planning period utilized in Chapter III, and the sensitivity analysis tested temperature and DO sensitives separately as discussed below. We utilized CE-QUAL-W2 models for the two reservoirs to determine water quality changes as a result of changes to the operating pattern. In this case study, the outflow rates and water quality constituent concentrations of Cordell Hull become the mainstem inflow rates and water quality constituent concentrations for Old Hickory reservoir downstream. Figure V.1 shows the bathymetries of the two reservoirs and indicates the locations of withdrawal structures, consisting of turbine and spillway release points.

The interaction between reservoirs in terms of water quality is a feedforward relationship. Releases from upstream reservoirs are transported downstream. Water quality feedback may need to be considered for applications in an estuarine setting or when pumped storage hydropower is present, but constituents have no means of transport from downstream reservoir to upstream reservoir in traditional river systems like the Cumberland River.

To test the impact of Cordell Hull's operations on Old Hickory reservoir's tailwater water quality, we performed a series of CE-QUAL-W2 simulations in which we modified Cordell Hull's withdrawal patterns over the planning period used in Chapter III, JDAY 215-225 during the year 2005. Target water elevations define the overall water volume passed through the dam prior to optimization; we constructed the experiments defined here with this in mind. Over this period there was no recorded spill flow out of Cordell Hull dam, and the first test simulation (CH-1) performed diverted the turbine flow to spill flow, resulting in all flow passing through the spillway. The second test simulation (CH-2) converted the hourly peaking turbine flow pattern to a daily average flow through the turbines. The third test simulation (CH-3) went further, by setting turbine releases at Cordell Hull to a fixed flowrate over the full 10-day planning period. In contrast, the fourth test simulation (CH-4) exaggerates the turbine discharge peaking pattern from the actual 2005 operations. Table

Table V.1: Summary of Cordell Hull CE-QUAL-W2 model calibration and validation results.

	Calibration	Validation
Year	2000	2005
Computational Time (minutes)	15	17
Elevation AME [a] (meters)	0.045	0.032
Dam Releases:		
Temperature AME [a] (°C)	0.658	0.745
DO AME [a] (mg/L)	1.245	1.298
In-stream Profiles:		
Temperature AME [a] (°C)	0.938	0.866
DO AME [a] (mg/L)	1.096	1.102

[a] Errors are presented as absolute mean error (AME). In-stream profile measurements of temperature and DO were collected at 9 locations on 2 dates in the calibration year (2000) and at 9 locations on 5 dates in the validation year (2005).

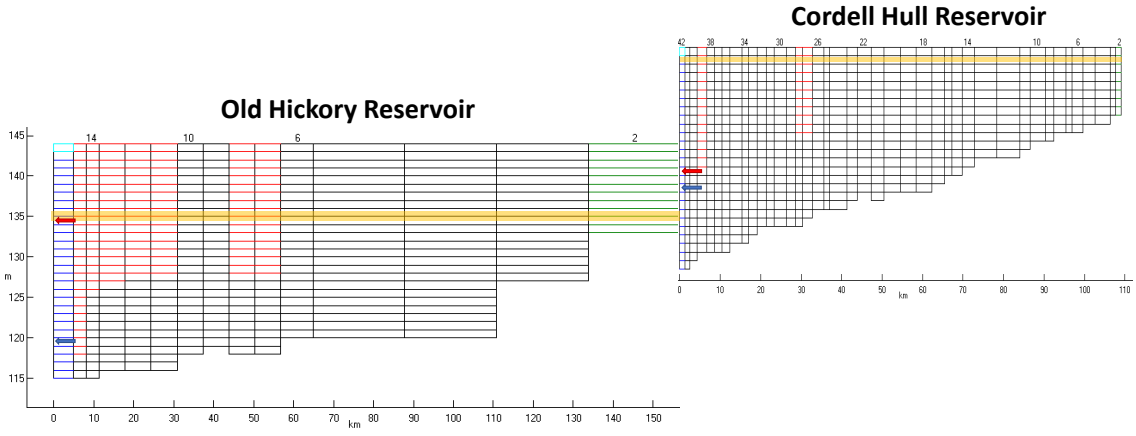


Figure V.1: Bathymetries of the mainstem sections of Cordell Hull and Old Hickory reservoirs, with turbine (red) and spill (blue) release elevations indicated by arrows and summer power pool storage zones shown in yellow.

V.2 summarizes the Cordell Hull release scenarios tested, and Figure V.2 provides the Cordell Hull actual turbine and spillway discharges over the defined planning period (CH-0), as well as the four modified flow regimes that were tested.

Cordell Hull's experimental outflows and discharge temperatures and DO concentrations then replaced the Old Hickory mainstem inflows, resulting in changes in Old Hickory tailwater temperature and DO concentrations. This allows for analysis of how changing operations at a singular dam propagates water quality changes downstream. We performed this twice, first assuming Old Hickory actual outflows from 2005, and then using the Old Hickory outflows from Experiment 2 in Chapter III, with simultaneous constraints on DO and temperature (see Subsection III.6.2 for additional information).

V.4 Results

Figure V.3 provides the CE-QUAL-W2 discharge temperatures at both reservoirs resulting from the experimental Cordell Hull release scenarios. Scenario CH-1, in which the only outflow modification was diverting the turbine flow to spillway flow, exhibited the smallest change from CH-0. The extreme peaking scenario (CH-4) exhibited the second smallest change, and the scenarios with daily (CH-2) and full 10-day averaged (CH-3) flows resulted in the greatest differences. The maximum difference at any time at Cordell Hull's release is approximately 0.5 °C, and at Old Hickory's release is 0.6 °C.

Figure V.4 provides discharge DO concentrations at both reservoirs resulting from the experimental Cordell Hull release scenarios. The temperature results echo the same general patterns exhibited by the DO results. Scenario CH-1 exhibited the smallest change from CH-0, CH-4 exhibited the second smallest change, and the smoothed scenarios CH-2 and CH-3 resulted in the greatest

Table V.2: Cordell Hull release scenarios used in sensitivity analysis.

Name	Description
CH-0	2005 actual turbine and spillway discharges (all flow released through turbines)
CH-1	CH-0 discharges swapped (all flow released through spillway)
CH-2	CH-0 hourly peaking turbine flow pattern converted to a daily average flowrate
CH-3	CH-0 hourly peaking turbine flow pattern converted to a 10-day average flowrate
CH-4	CH-0 turbine discharge peaking pattern exaggerated

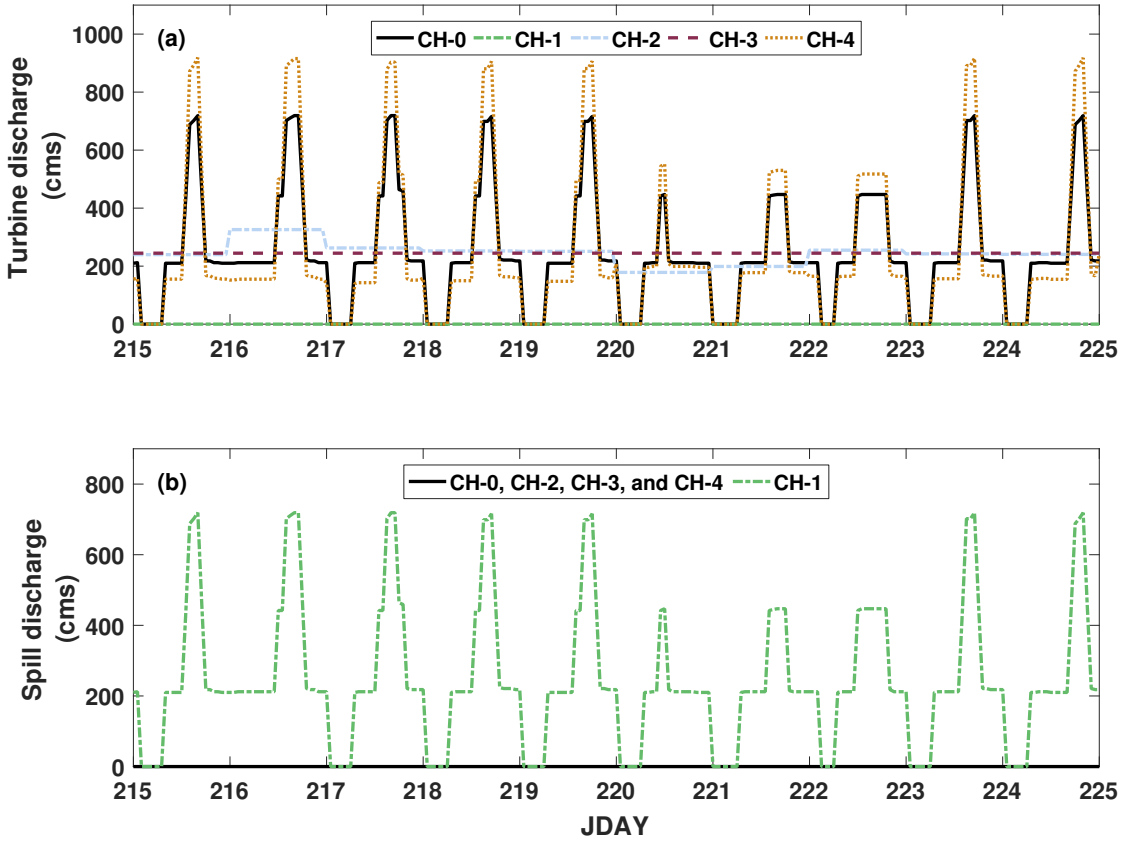


Figure V.2: Cordell Hull baseline (CH-0) and experimental (CH-1, CH-2, CH-3, and CH-4) turbine and spill releases over the 10-day planning period.

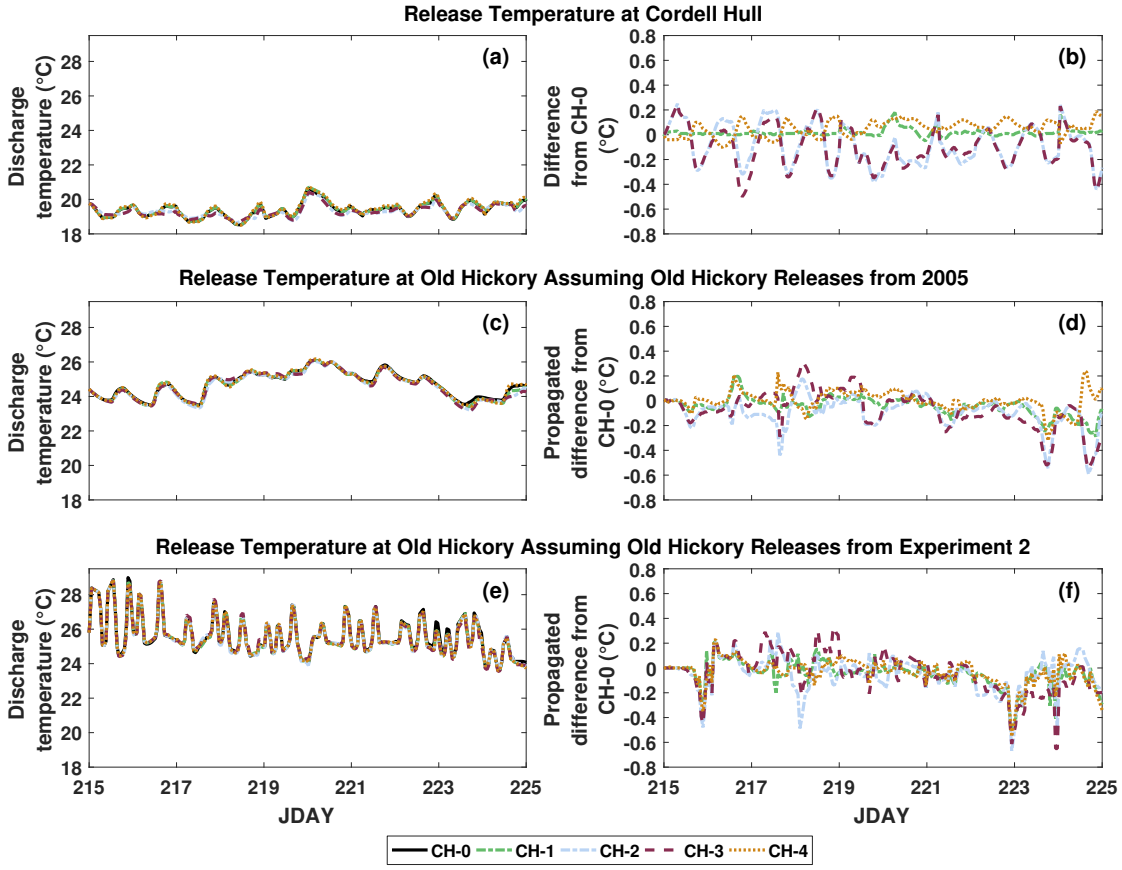


Figure V.3: Cordell Hull and Old Hickory baseline (CH-0) and experimental (CH-1, CH-2, CH-3, and CH-4) discharge temperatures and differences from baseline temperatures.

differences. The maximum difference at any time at Cordell Hull's release is approximately 0.6 mg/L, and at Old Hickory's release is 0.3 mg/L.

For each experiment, the water quality prediction AME over the 10-day operating period due to changes in Cordell Hull releases was computed, as provided in Table V.3. Because of the system's feedforward water quality relationship, Old Hickory releases impact Old Hickory release water quality, while Cordell Hull releases impact release water quality at both reservoirs.

V.5 Discussion

Discharge water quality at the downstream reservoir Old Hickory does exhibit some sensitivity to operations at the upstream Cordell Hull reservoir, although this sensitivity is small. For both temperature and DO, the fluctuations caused by changing Cordell Hull operations are greater at the Cordell Hull discharge and dampened further downstream at the Old Hickory discharge, as expected. The approximate upper two thirds of Old Hickory's 97.3 miles of impounded backwater is well-mixed, even during the late summer when the lower end of the reservoir becomes vertically stratified. The stratified zone, which drives Old Hickory reservoir's discharge water quality, is resistant to mixing due to low density water stored at the surface and high density water stored deeper in the forebay. If stratification is present, minor fluctuations in water quality upstream are not sufficient to offset density gradients in the forebay and induce mixing. Although minor fluctuations are seen in Figure V.3 and Figure V.4, Old Hickory discharge water quality during this time period is relatively stable regardless of Cordell Hull operations.

Table V.3: Cordell Hull and Old Hickory release temperature and DO concentration differences between experimental Cordell Hull release scenarios and 2005 (CH-0) releases, computed as AME.

Flow Release Pattern		Temperature (°C)		DO (mg/L)	
Cordell Hull (CH)	Old Hickory (OH)	CH	OH	CH	OH
CH-1	2005	0.017	0.062	0.026	0.025
	Exp 2		0.077		0.048
CH-2	2005	0.167	0.110	0.109	0.042
	Exp 2		0.103		0.072
CH-3	2005	0.167	0.123	0.129	0.065
	Exp 2		0.128		0.090
CH-4	2005	0.073	0.057	0.049	0.033
	Exp 2		0.075		0.045

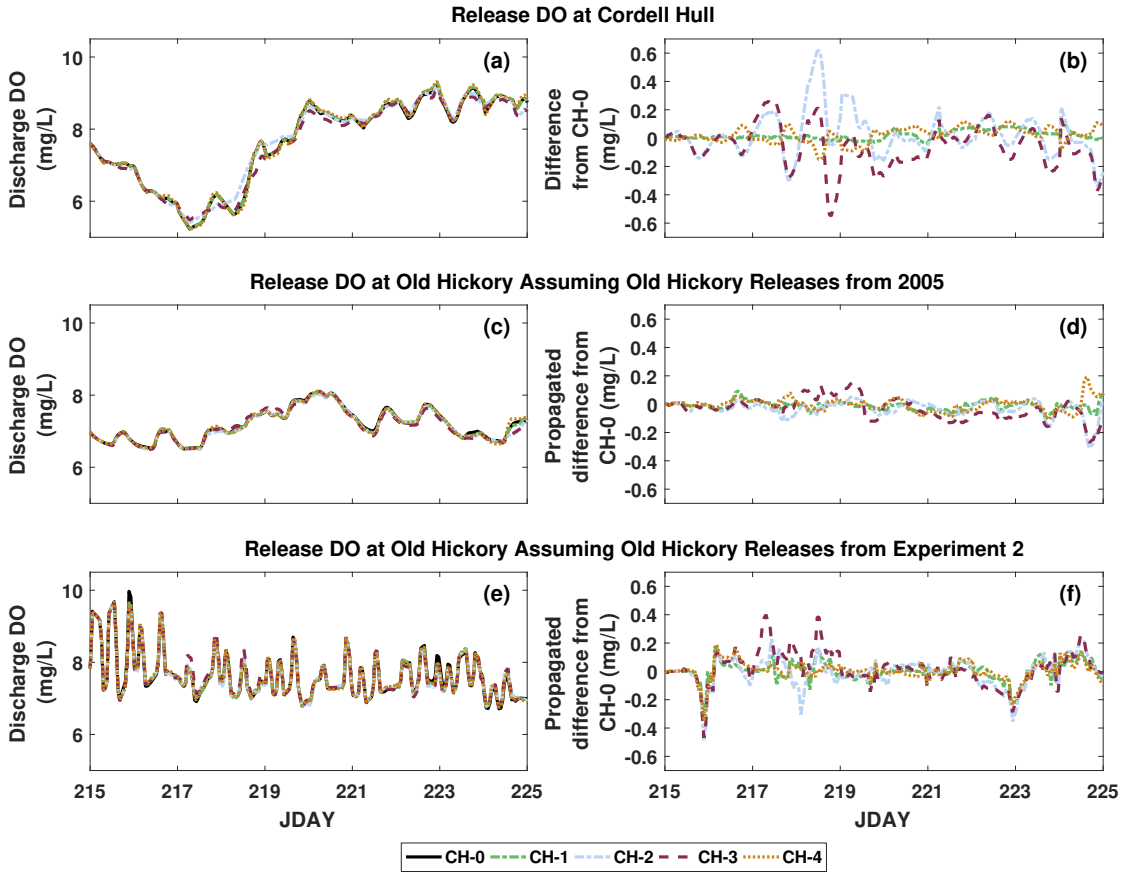


Figure V.4: Cordell Hull and Old Hickory baseline (CH-0) and experimental (CH-1, CH-2, CH-3, and CH-4) discharge DO concentrations and differences from baseline DO concentrations.

The scenario in which all Cordell Hull outlet flow is diverted from the turbines to the spillway resulted in the smallest alteration to water quality downstream. This is due to the small elevation difference between spill and turbine release structures, as seen in Figure V.1. Regardless of which structure at Cordell Hull is used for releases, water is drawn from the same approximate depth and stratification has little impact on release water quality. This may not be the case when upstream reservoirs are constructed with release structures located further apart. For example, a scenario in which the release structure of Old Hickory, whose turbine withdrawal point is 15 meters below the spillway withdrawal, is located at the upstream reservoir would likely be a system in which downstream water quality is much more sensitive to upstream structure release choice than the system used for sensitivity analysis here.

The tested Cordell Hull releases are not all realistic examples of hydropower release patterns. For example, hydropower typically operates on a peaking pattern to supply power during peak demand periods, so scenarios CH-2 and CH-3 will likely never occur. The sensitivity analysis aims to provide insight into the potential for water quality prediction errors due to changes in boundary conditions. The conditions tested, though somewhat unrealistic, represent "extreme conditions" with regard to peaking pattern severity. Water quality sensitivity appears to be small when assessed using the extreme conditions, so actual boundary conditions (i.e., hydropower and spill releases) will likely produce even smaller errors. In other words, the sensitivity analysis approach uses extreme boundary conditions as a means for determining an upper limit on expected water quality prediction errors due to changes in upstream release decisions.

This sensitivity analysis aims to inform hourly optimization of the case study reservoirs subject to one or more constraints on Old Hickory reservoir water quality; for example, Experiment 2 applies simultaneous lower bounds on DO concentrations and temperature of the Old Hickory discharge. Subplot (e) in Figures V.3 and V.4 provide the error for Old Hickory discharge water quality, assuming the Chapter III Experiment 2 Old Hickory discharges and the various experimental upstream Cordell Hull release patterns; however, since water quality predictions constitute a constraint on operations, errors away from constraint boundaries are of little interest. Chapter III Experiment 2 applied lower bounds on discharge DO ($o_l = 7$ mg/L) and temperature ($t_l = 25$ °C), and Figures V.5 and V.6 compare Old Hickory discharge water quality values using Cordell Hull baseline compared to experimental operations. The best solution found for Experiment 2 was not fully feasible with

respect to DO or temperature constraints over the full 10-day period, which is indicated by values less than o_l and t_l present along the temperature and DO response to CH-0 axes. Focusing on the constraint boundaries, quadrants two and four represent areas of concern, as they contain predictions which shifted across the constraint limit as a result of differences in the upstream boundary condition. Simply put, prediction differences here cause infeasible timepoints to be falsely determined as feasible (and vice versa) due to differences in upstream reservoir discharge. Although CH-1 and CH-4 overall provide water quality outcomes more similar to CH-0 than do CH-2 and CH-3, the differences overall are minimal and there are no additional trends visible near the constraint boundary. These results indicate that Old Hickory reservoir discharges are fairly independent from the operating pattern at the upstream reservoir over this time period. Therefore, a segmented approach for optimizing the Cordell Hull-Old Hickory linked system, in which reservoirs are optimized independently with assumed boundary conditions, will likely result in minimal errors in downstream water quality predictions.

V.6 Conclusions

For two reservoirs with hydropower capabilities linked in series, we assessed the sensitivity of the downstream reservoir's discharge water quality in response to the upstream reservoir's discharge pattern. Determining independence between these variables could enable expanded application of the previously-developed optimization routine (detailed in Chapter III) from single reservoirs to reservoir systems. Here, we used the linked Cordell Hull-Old Hickory system to demonstrate a method for analyzing downstream water quality dependency on upstream release scheduling over a typical 10-day operating period. Assuming a fixed volume of water is passed through the upstream Cordell Hull reservoir, these results indicate minor impacts on downstream water quality predictions. For the demonstrated problem formulation with defined lower bounds on temperature and DO, prediction errors caused by differences in upstream boundary condition indicate minimal impact on potential solutions to an optimization procedure, where water quality constraints are defined by these predictions.

This study analyzed the downstream propagation of water quality changes in a system of two reservoirs linked in series. We fixed the total volume of flow released from the upstream reservoir

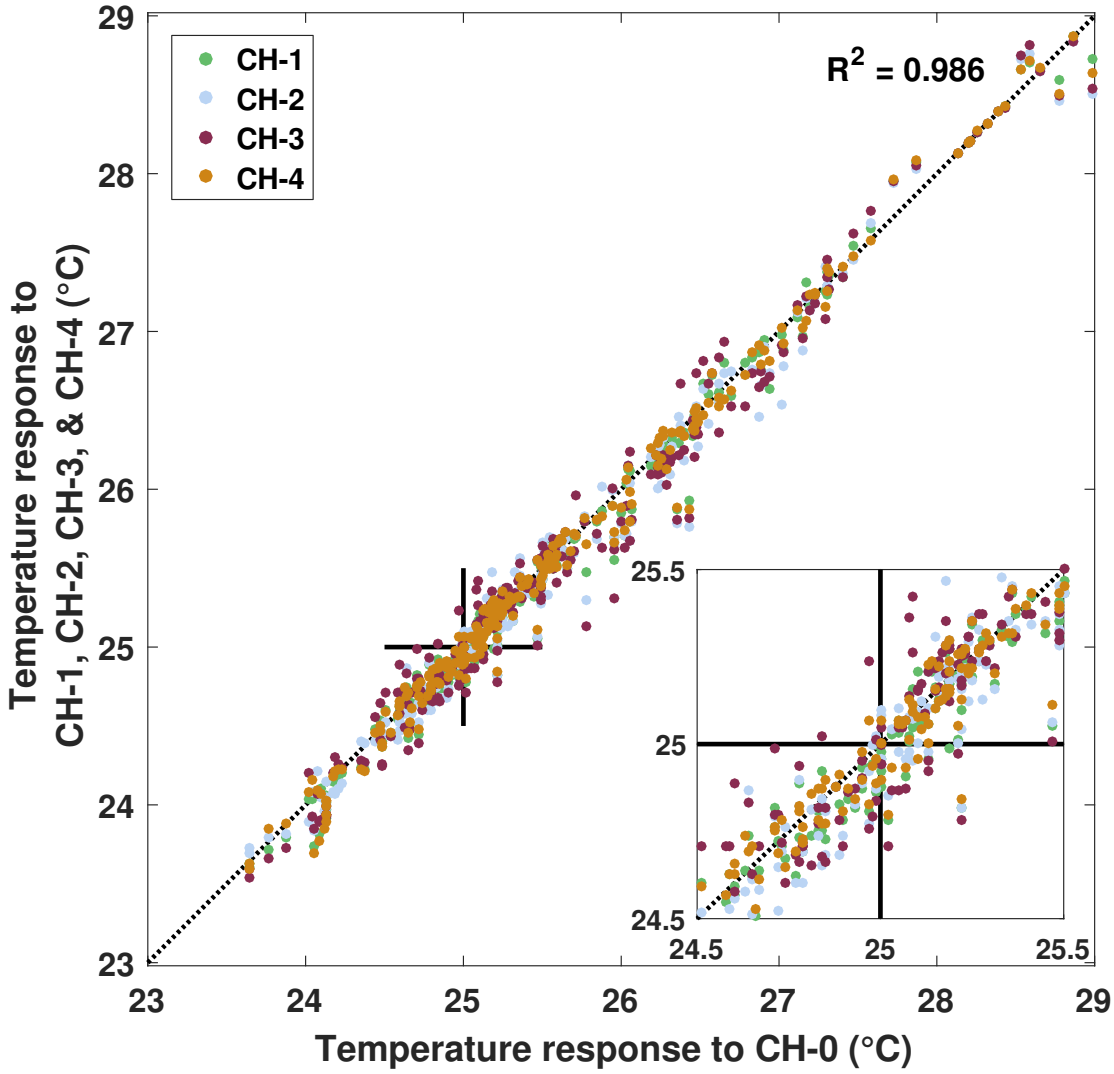


Figure V.5: Old Hickory release temperatures at all timepoints in 10-day planning period assuming operations found in Chapter III Experiment 2, assuming Cordell Hull baseline releases (CH-0) along the x-axis and experimental releases (CH-1, CH-2, CH-3, and CH-4) along the y-axis. Horizontal and vertical lines represent constraint boundaries.

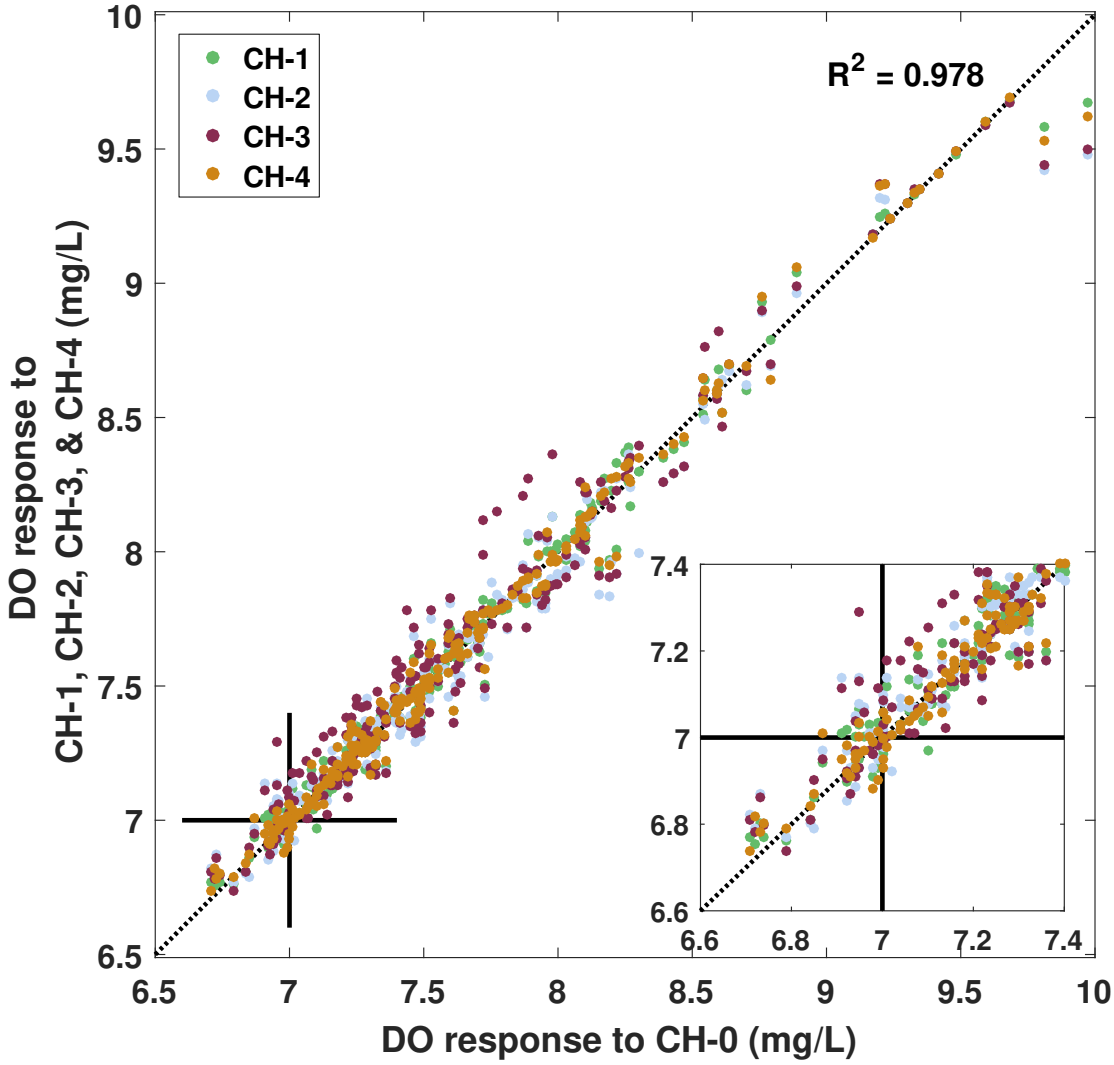


Figure V.6: Old Hickory release DO concentrations at all timepoints in 10-day planning period assuming operations found in Chapter III Experiment 2, assuming Cordell Hull baseline releases (CH-0) along the x-axis and experimental releases (CH-1, CH-2, CH-3, and CH-4) along the y-axis. Horizontal and vertical lines represent constraint boundaries.

over each day or the planning period as a whole and modified the time series of upstream reservoir discharges by adjusting peaking intensity. This reflects the typical decision-making process at these reservoirs for short-term planning, in which total release volumes are defined first and then operations are determined on a refined timestep in order to best meet constraints and objectives. While changes to the water balance of the two reservoirs here are likely limited, this is still an important consideration when determining the feasibility of using a segmented optimization approach. Power generation is a function of the headwater and tailwater head difference, so even small errors in headwater or tailwater elevation predictions at specific instances in time over the operating period could result in errors in power production estimates, which drive the direction of an optimizer seeking to maximize power generation. Future work should expand this sensitivity analysis to explore the impacts of boundary condition flow differences on water balance (and therefore hydropower production).

We formulated this sensitivity analysis around the current case study system with the aim of optimizing operations over a concise 10-day period, with water quality considerations solely at a tailwater location. Other waterbody systems may have different concerns, such as an interest in water quality at specific locations within the waterbody itself, including spawning grounds of sensitive species, water utility withdrawal points, or thermal power plant withdrawal and/or discharge points. The sensitivity analysis methodology demonstrated is easily applied to such scenarios.

Chapter VI

CONCLUSIONS AND FUTURE WORK

VI.1 Conclusions

In this work, we described and demonstrated an approach for computing globally optimal power generation schemes for a hydropower reservoir using high-fidelity WQMs, surrogate modeling techniques, and multidimensional optimization methods. By combining these methods, we were able to include high-fidelity water quality constraints within dam release decision-making on an operational timescale. We applied the approach to a single multipurpose reservoir with hydropower capabilities and used the surrogate-enabled optimizer to explore the trade-offs between spillway and hydropower flow releases. We then explored methods to improve optimization solution quality. Finally, we investigated the sensitivity of downstream water quality on upstream boundary conditions to better inform future applications of the approach to a larger system of reservoirs.

We introduced the overall optimization methodology and case study reservoir in Chapter III. Old Hickory reservoir, located on the Cumberland River and operated by the USACE Nashville District, is a run-of-river hydropower facility with downstream water quality concerns. The reservoir is modeled using the high-fidelity hydrodynamics and water quality model CE-QUAL-W2, but the model is not currently employed for decision-making due to computational expense. The CE-QUAL-W2 model generated data for training NARX ANN surrogate models which predict discharge temperature and DO as a function of exogenous inputs, including upstream inflows, meteorological data, and structure releases. Validation tests revealed that the ANN model form successfully emulates the dynamic water quality simulator. We utilized the ANN model within a genetic algorithm optimization approach to maximize hydropower generation subject to constraints on dam operations and water quality. The model successfully reproduced high-fidelity reservoir information while enabling 6.8 and 6.6 percent increases in hydropower production value relative to actual operations for DO limits of 5 and 6 mg/L, respectively, while witnessing an expected decrease in power generation at more restrictive DO constraints. Exploration of simultaneous temperature and DO constraints

revealed capability to address multiple water quality constraints at specified locations. The reduced computational requirements of the new modeling approach provides decision support for reservoir operations scheduling while maintaining high-fidelity hydrodynamic and water quality information as part of the optimization decision support routines.

Chapter IV focused on the optimizer itself, exploring modifications to the optimization algorithm in an effort to improve solution quality. Because the ANN surrogate model is not an exact emulator, prediction errors can lead the optimizer to converge on infeasible solutions. To counteract this, two approaches were tested. The first approach, random immigrants replacement, is a technique to improve GA population diversity by injecting new members within each population. Improving population diversity is of particular importance for DOPs. The second approach involved soliciting additional surrogate model training data adaptively mid-optimization. Additional training data samples were chosen from GA populations and simulated with the full simulation model, improving surrogate performance in regions of suspected optimality. We merged these two approaches within the optimization methodology introduced in Chapter III in order to optimize Old Hickory reservoir operations over 24 hours with a constraint on minimum release DO concentrations. Combining random immigrants replacement and adaptive additional sampling produced superior fitness values, and when used individually improved results over trials where neither approach was used.

Chapter V looked toward expanding this work to a system of reservoirs by performing a necessary exploration of the feedbacks exhibited between two reservoirs connected in series. For two reservoirs with hydropower capabilities linked in series, we assessed the sensitivity of the downstream reservoir's discharge water quality in response to the upstream reservoir's discharge pattern. Determining independence between these variables could enable expanded application of the previously-developed optimization routine (detailed in Chapter III) from single reservoirs to reservoir systems. Here, we used the linked Cordell Hull-Old Hickory system to demonstrate a method for analyzing downstream water quality dependency on upstream release scheduling over a typical 10-day operating period. Assuming a fixed volume of water is passed through the upstream Cordell Hull reservoir, these results indicate minor impacts on downstream water quality predictions. For the demonstrated problem formulation with defined lower bounds on temperature and DO, prediction errors caused by differences in upstream boundary condition indicate minimal impact on potential solutions to an optimization procedure, where water quality constraints are defined

by these predictions.

VI.2 Future Work

This work provides an initial demonstration of how a high-fidelity WQM can be integrated within a hydropower operations decision support tool in order to couple water quality with hydropower generation decision-making. We developed this approach using two Cumberland River mainstem reservoirs as prototypes, and made methodology development assumptions with this system in mind. These include assumptions that turbines operate at rated capacity, turbines are dispatched hourly, and spill is adjusted daily, as well as the water quality compliance point assumption and the target elevation storage assumption. In order for this approach to be applied to other systems, these assumptions will need to reconsidered for appropriateness.

Here, the optimized mainstem hydropower reservoir has little power pool storage and flood control storage. Pool elevations are relatively fixed in this case, so the optimizer focuses on reallocating a predetermined volume of release water over the planning period between two release structures. The overall seasonal water allocation plan for the basin largely determines stratification, which drives the water quality characteristics of these releases. Pool levels at tributary reservoirs are more flexible, and tributary reservoir operations strongly impact stratification downstream based on the timing and supply of cool water through the warm, dry season. Additionally, tributary projects on the Cumberland River have greater power capacities than projects on the mainstem. It would be beneficial to apply this optimization methodology to tributary reservoirs, as well as to develop an approach for seasonal planning optimization to be informed by high-fidelity water quality simulators.

This work assumes that spillway aeration has a negligible influence on tailwater DO, and that discharge DO concentrations result from the simple mixing of turbine and spill releases computed by the equation:

$$DO_{mix} = \frac{Q_{spill} \cdot DO_{spill} + Q_{turbines} \cdot DO_{turbines}}{Q_{spill} + Q_{turbines}} \tag{VI.1}$$

where DO_{spill} and $DO_{turbines}$ are concentrations and Q_{spill} and $Q_{turbines}$ are flowrates. However, releases over the spillway are subject to aeration including oxygenation. Assuming that this flow is at

saturation concentration is appropriate in some cases, but in other cases supersaturation may occur (*Wolff et al.*, 2013). By neglecting spillway aeration, the solutions found here are conservative in regards to meeting a lower bound constraint on DO, but conversely this may cause the optimizer to bypass solutions with higher power generation potential. Applying this optimization methodology to a system with constraints on TDG for aquatic species health requires spillway aeration to be considered, as gas entrainment primarily occurs during times of high spill (*Witt et al.*, 2017). Therefore, future work should incorporate spillway aeration as an additional process following release through the dam structure.

Another potential area of study is applying the WQM-informed optimizer to reservoir water quality mitigation device design, including forebay and turbine aeration installations. Many studies have employed WQMs to analyze site-specific mitigation techniques (*Bartholow et al.*, 2001; *Saito et al.*, 2001; *Caliskan and Elci*, 2009; *Castelletti et al.*, 2010; *Singleton et al.*, 2013), but these studies tend to consider water quality changes due to mitigation action and neglect to explore how water quality improvements impact reservoir operations. When designing devices like forebay and turbine aerators, expenses including construction, operations, and maintenance costs are considered, and impacts on optimal hydropower generation potential should also be considered. Determining optimal generation potential under various conditions and mitigation device designs requires integrating optimization and high-fidelity water quality predictions, and the methodology demonstrated here serves as a foundation for these types of studies.

Chapter IV focused on the optimizer itself, exploring modifications to the optimization algorithm in an effort to improve solution quality. Prior work in the area of adaptive model updating within optimization relies on surrogate model forms which provide statistical information (for example, GPs as used in *Bichon et al.* (2013)). Black-box emulators like ANNs do not produce the statistical information necessary to use such techniques, so a population-based resampling approach was described here. The algorithm modifications demonstrated in Chapter IV could prove useful for solving any optimization problem where a population-based optimizer is appropriate, a constraint depends on a black-box inexact emulator of an expensive simulation model, and there is a need for emulator construction and/or training to be influenced by outcomes from the optimization process itself. Further research on the level of additional sampling necessary for improved results is needed, and a non-problem-specific heuristic defining appropriate additional sampling levels is necessary

for exploring new applications for the framework shown here. An approach for quantifying ANN surrogate model error during optimization would be a valuable addition to the methodology, as this could be used to further inform the resampling step as well as provide the user with a metric for assessing confidence in the provided solution.

Looking forward, expanding this methodology to efficiently optimize a system of reservoirs would provide a beneficial tool for hydropower operations. The optimization routine here, built on a GA, is not well-suited for unlimited problem size expansion. Researchers should explore techniques for applying the general approach shown here to larger problems. We performed the water quality propagation sensitivity analysis in Chapter V with the idea of potentially optimizing a larger system of reservoirs by segmenting it into smaller problems to be solved in parallel. Before exploring this, the sensitivity of assumed upstream and downstream boundary conditions on water balance, and therefore hydropower production estimation, needs to be assessed.

In summary, the proposed improvements to the model framework presented herein would provide a powerful tool for activities including mitigation technology design, tributary reservoir operations planning, and reservoir system release decision-making. Bringing together high-fidelity water quality predictions and global optimization methods strengthens capabilities to regulate water quality while maximizing power production in controlled waterways.

Appendix A

OLD HICKORY RESERVOIR CE-QUAL-W2 MODEL CALIBRATION AND VALIDATION FIGURES

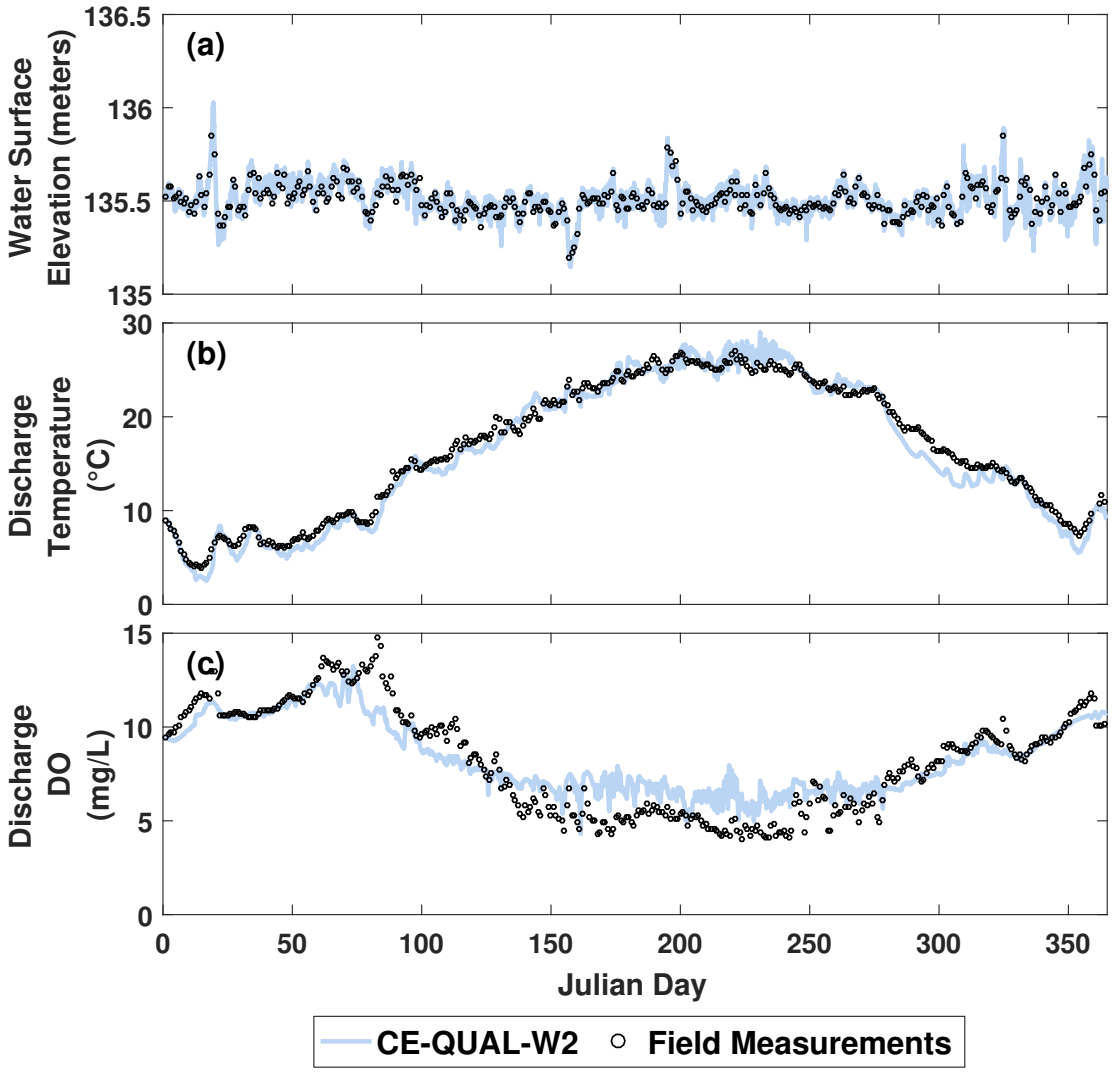


Figure A.1: Old Hickory CE-QUAL-W2 model calibration timeseries outcomes for the year 1988: (a) water surface elevation, (b) discharge temperature, and (c) discharge DO.

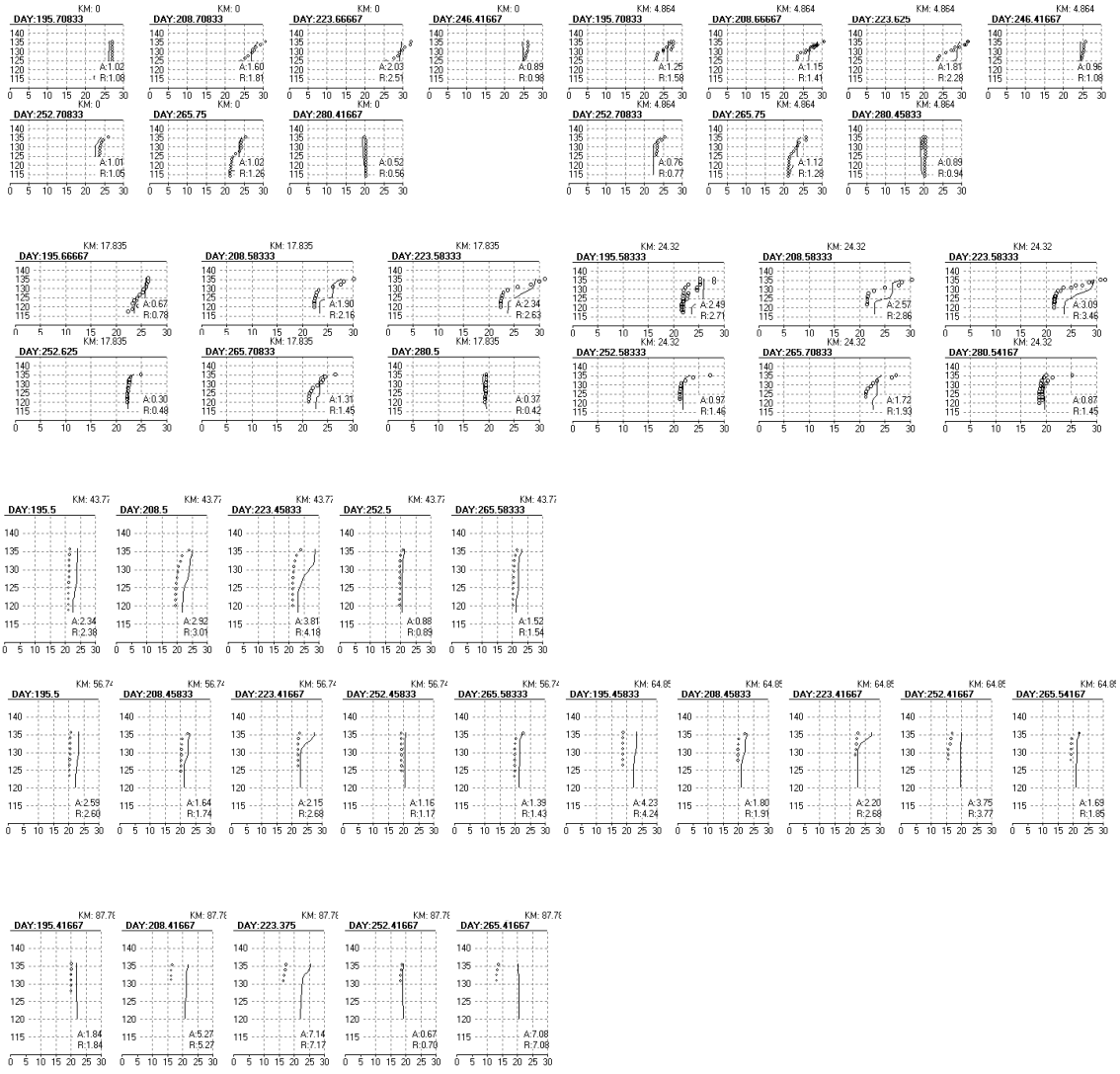


Figure A.2: Old Hickory CE-QUAL-W2 model calibration temperature profiles for the year 1988 (created using AGPM-2D v3.5 post-processor for CE-QUAL-W2 by Loginetics, Inc.). Profile measurements were collected on 7 dates at 8 locations.

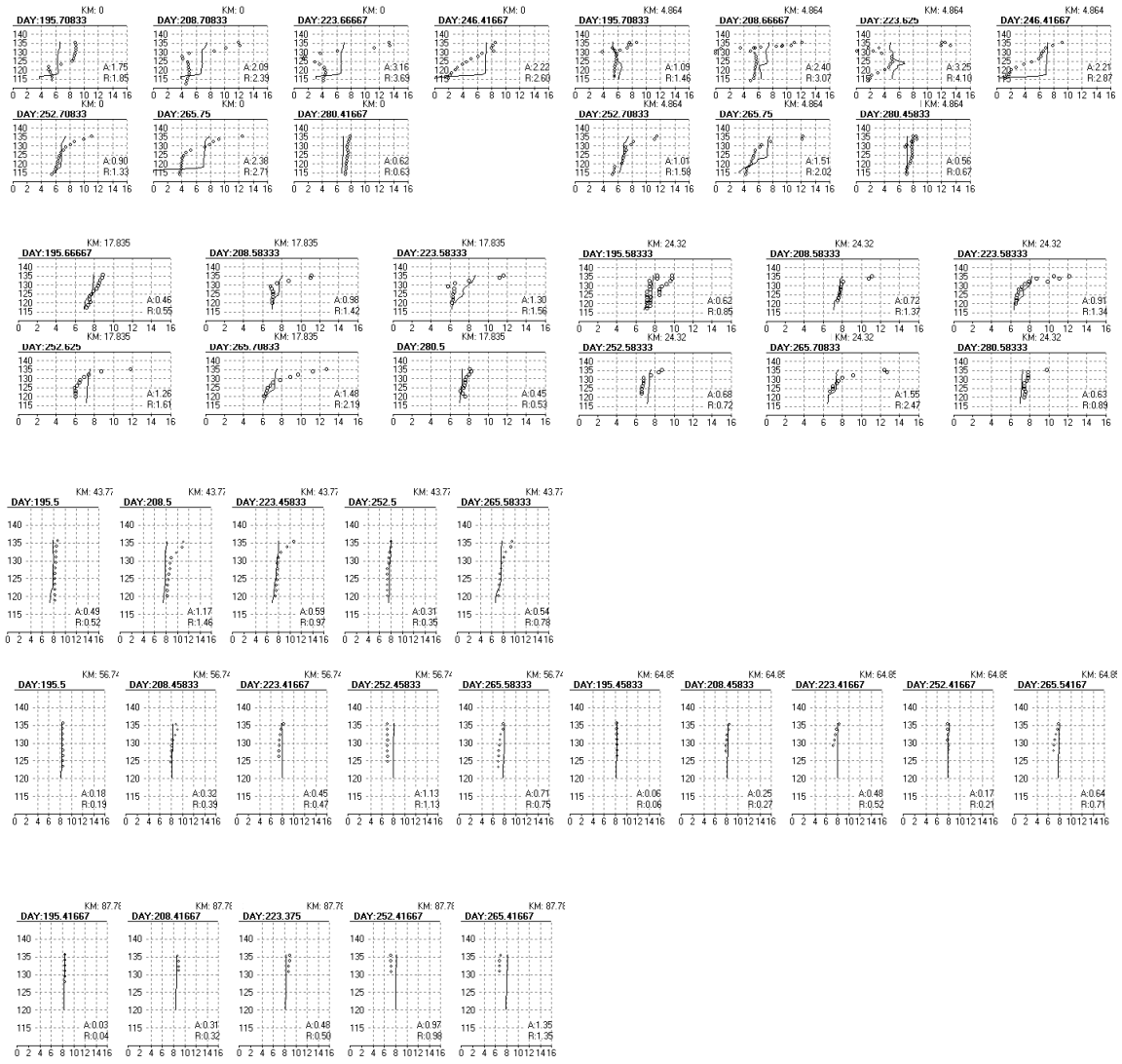


Figure A.3: Old Hickory CE-QUAL-W2 model calibration DO profiles for the year 1988 (created using AGPM-2D v3.5 post-processor for CE-QUAL-W2 by Loginetics, Inc.). Profile measurements were collected on 7 dates at 8 locations.

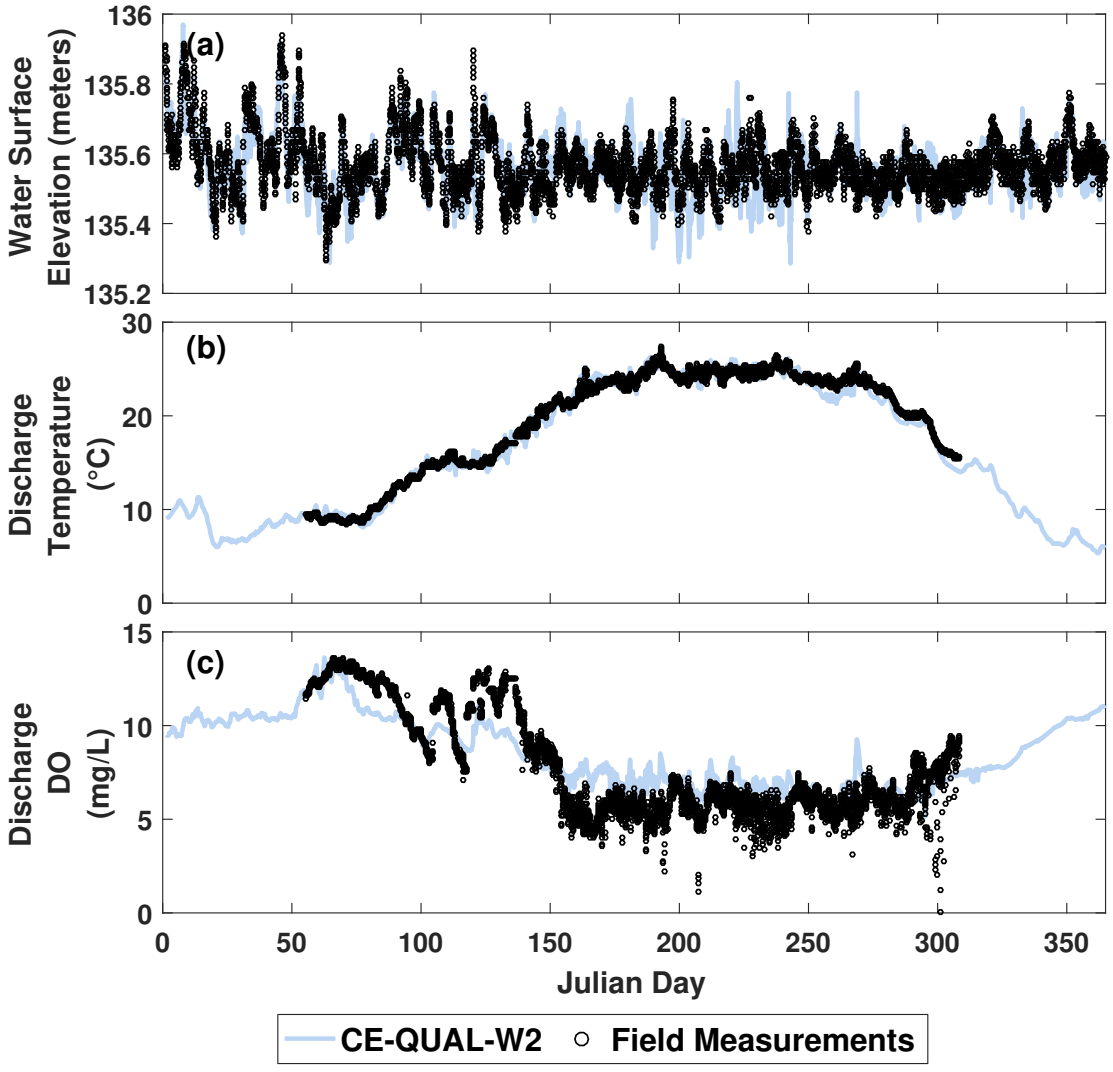


Figure A.4: Old Hickory CE-QUAL-W2 model validation timeseries outcomes for the year 2005: (a) water surface elevation, (b) discharge temperature, and (c) discharge DO.

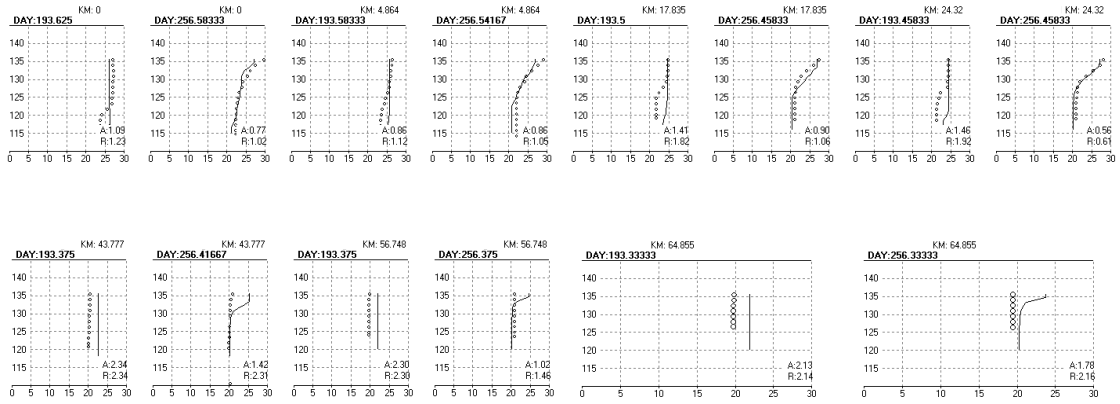


Figure A.5: Old Hickory CE-QUAL-W2 model validation temperature profiles for the year 2005 (created using AGPM-2D v3.5 post-processor for CE-QUAL-W2 by Loginetics, Inc.). Profile measurements were collected on 2 dates at 7 locations.

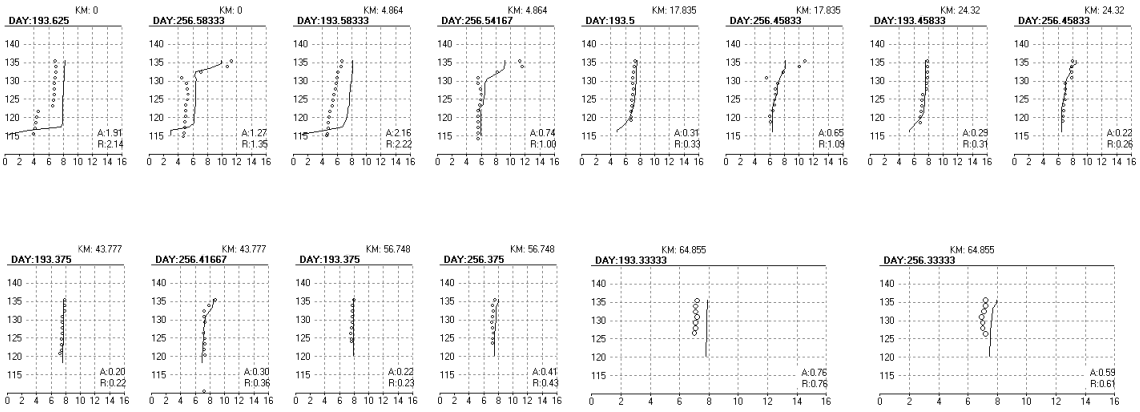


Figure A.6: Old Hickory CE-QUAL-W2 model validation DO profiles for the year 2005 (created using AGPM-2D v3.5 post-processor for CE-QUAL-W2 by Loginetics, Inc.). Profile measurements were collected on 2 dates at 7 locations.

Appendix B

CORDELL HULL RESERVOIR CE-QUAL-W2 MODEL CALIBRATION AND VALIDATION FIGURES

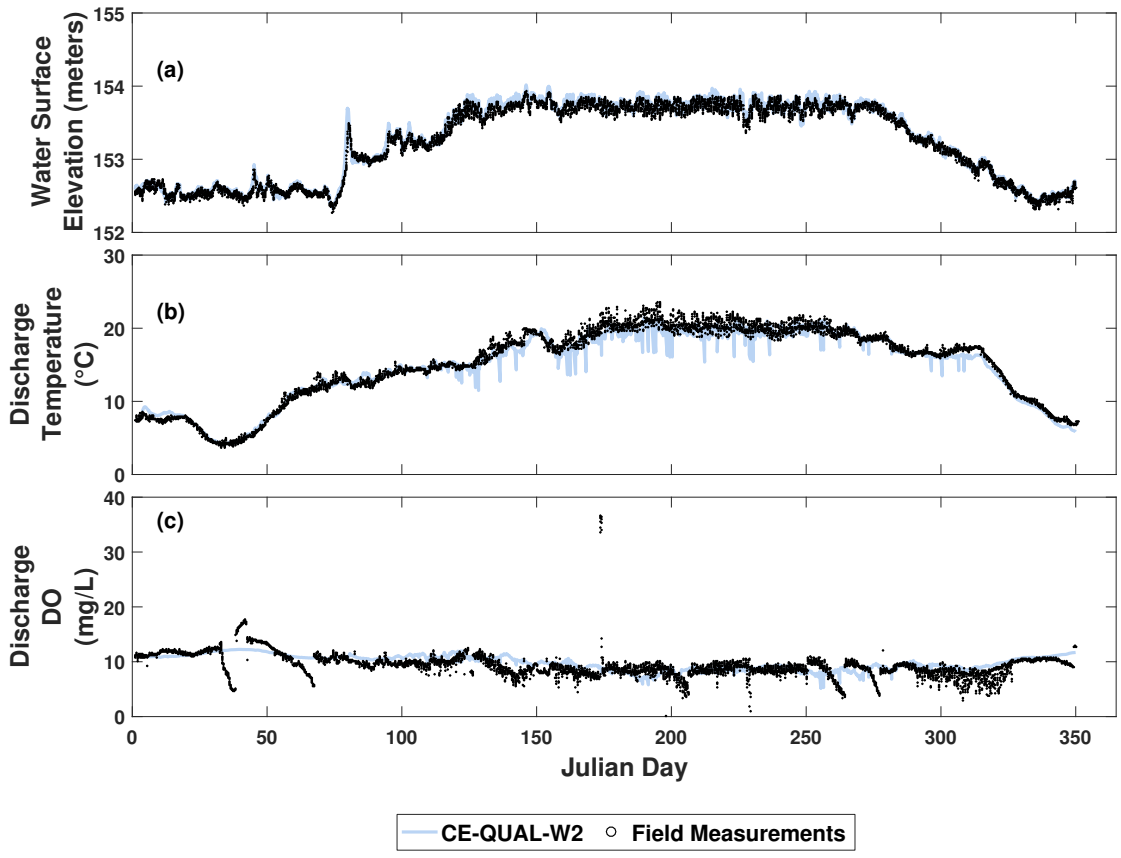


Figure B.1: Cordell Hull CE-QUAL-W2 model calibration timeseries outcomes for the year 2000: (a) water surface elevation, (b) discharge temperature, and (c) discharge DO.

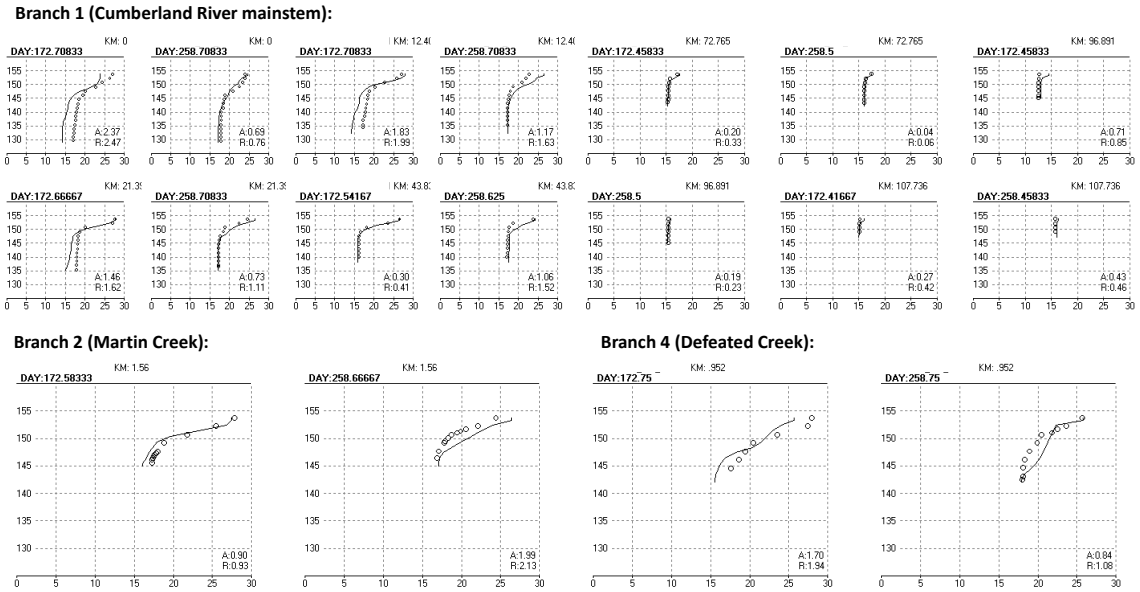


Figure B.2: Cordell Hull CE-QUAL-W2 model calibration temperature profiles for the year 2000 (created using AGPM-2D v3.5 post-processor for CE-QUAL-W2 by Loginetics, Inc.). Profile measurements were collected on 2 dates at 9 locations.

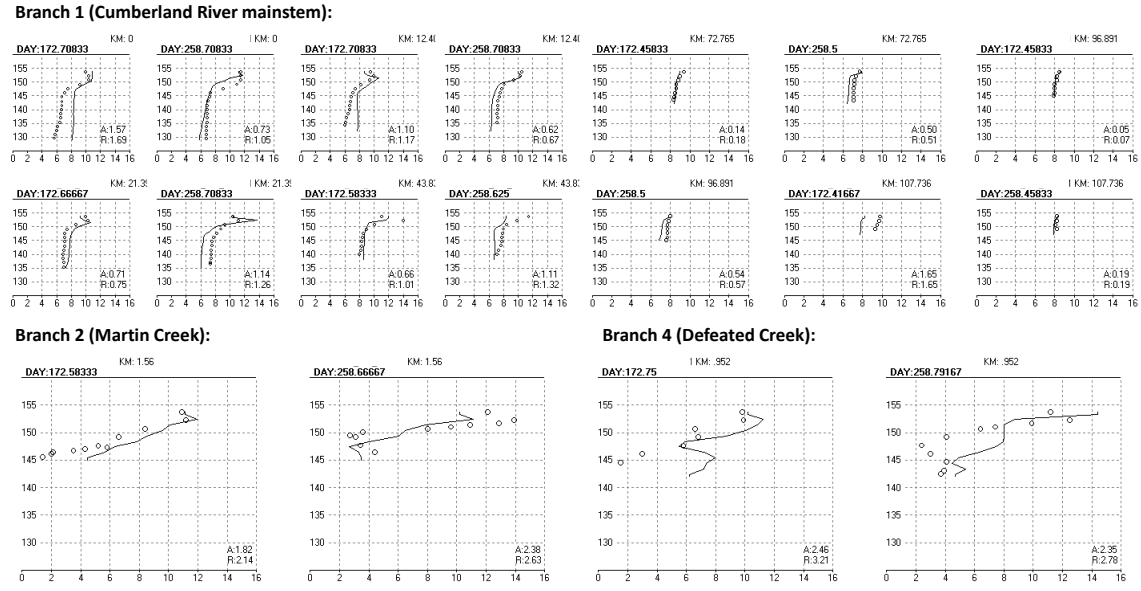


Figure B.3: Cordell Hull CE-QUAL-W2 model calibration DO profiles for the year 2000 (created using AGPM-2D v3.5 post-processor for CE-QUAL-W2 by Loginetics, Inc.). Profile measurements were collected on 2 dates at 9 locations.

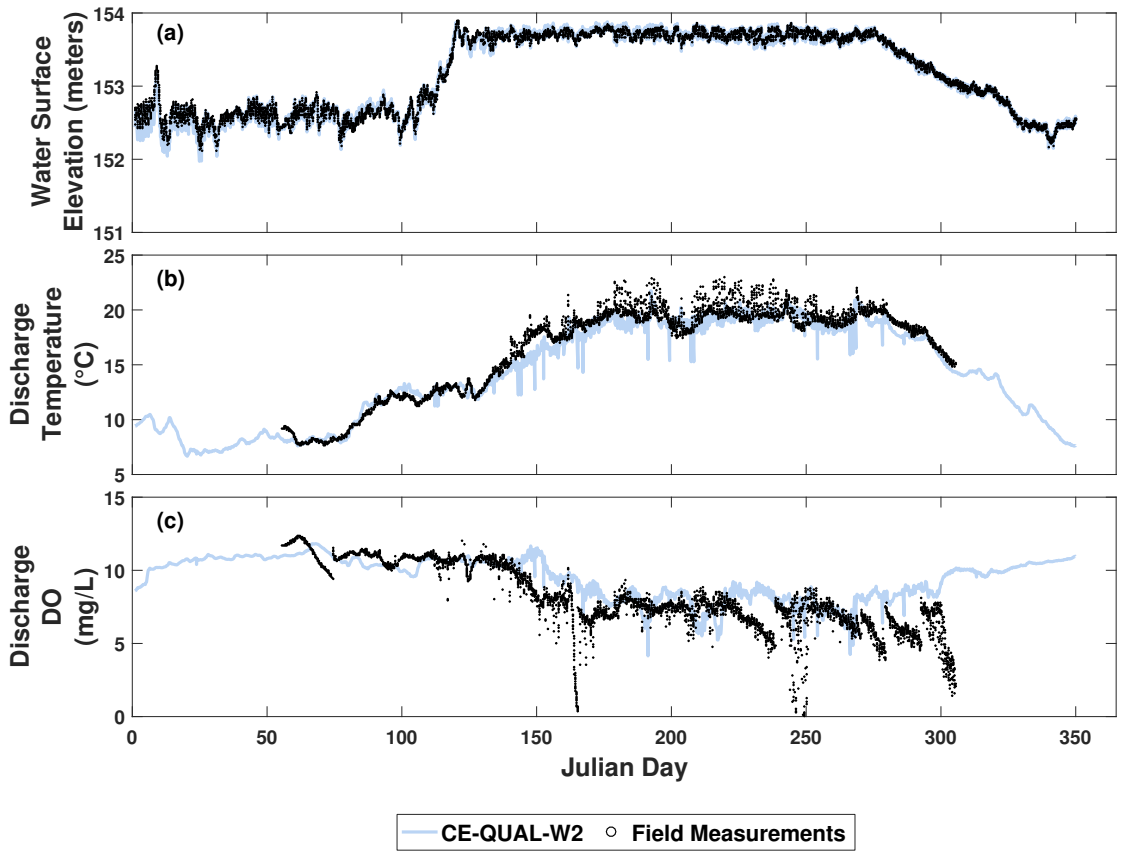


Figure B.4: Cordell Hull CE-QUAL-W2 model validation timeseries outcomes for the year 2005: (a) water surface elevation, (b) discharge temperature, and (c) discharge DO.

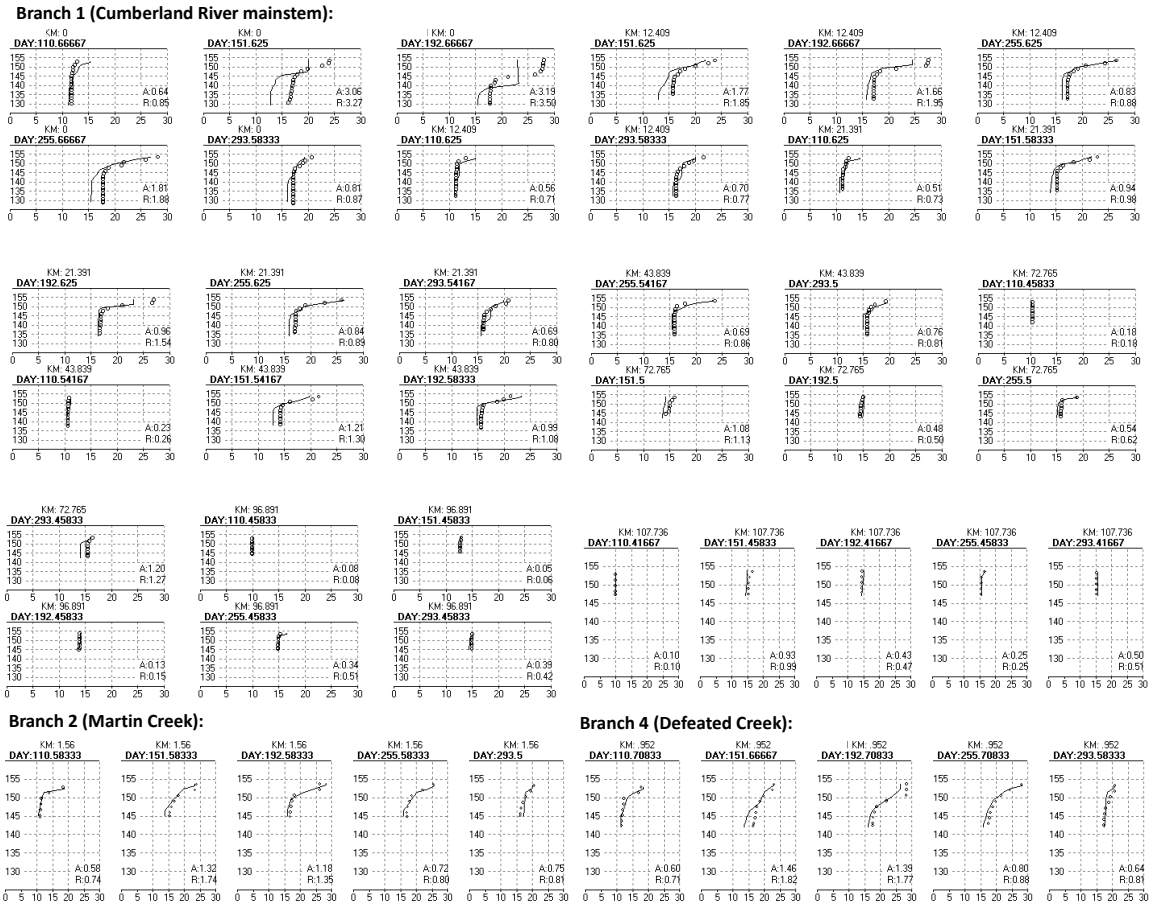


Figure B.5: Cordell Hull CE-QUAL-W2 model validation temperature profiles for the year 2005 (created using AGPM-2D v3.5 post-processor for CE-QUAL-W2 by Loginetics, Inc.). Profile measurements were collected on 5 dates at 9 locations.

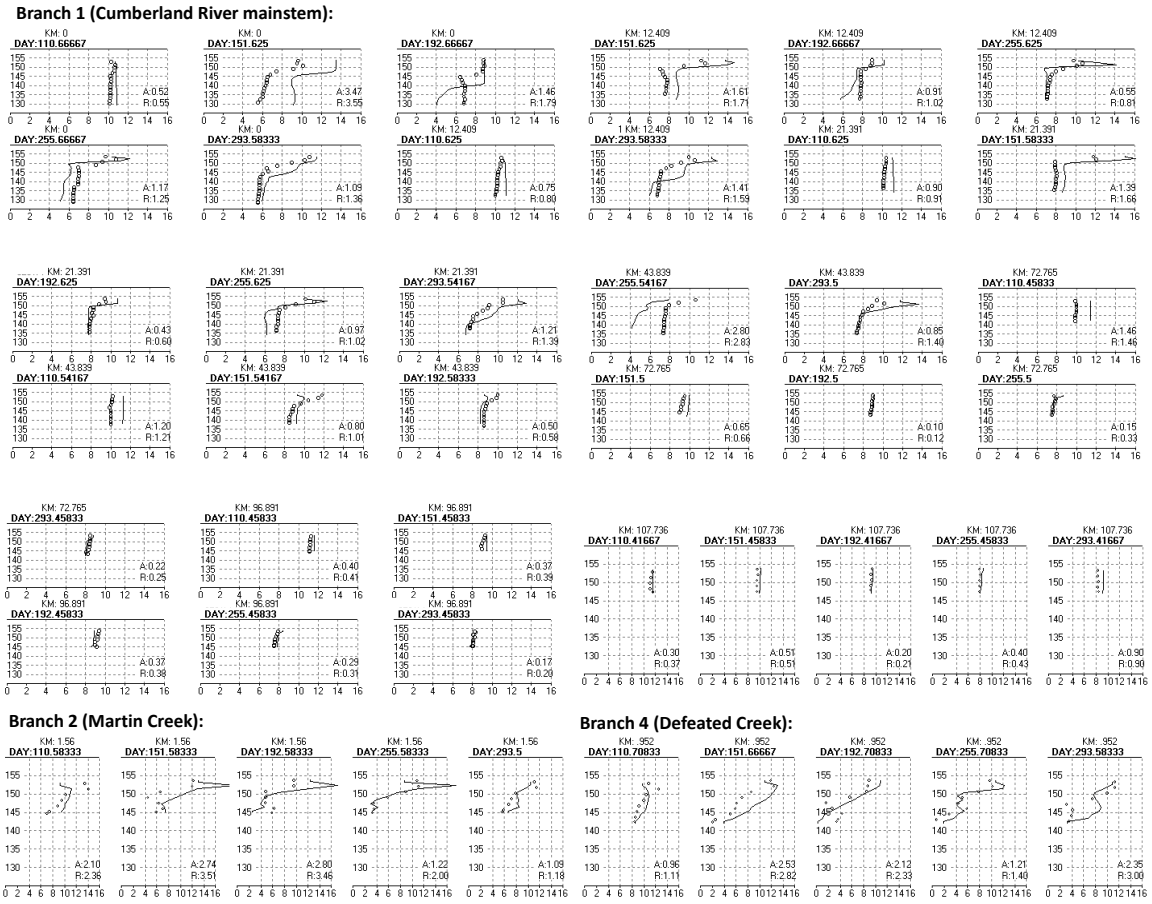


Figure B.6: Cordell Hull CE-QUAL-W2 model validation DO profiles for the year 2005 (created using AGPM-2D v3.5 post-processor for CE-QUAL-W2 by Loginetics, Inc.). Profile measurements were collected on 5 dates at 9 locations.

Appendix C

MATLAB® CODE FOR NARX MODEL TRAINING

The following code is used to train a family of NARX WQMs for temperature and DO for Old Hickory reservoir, and performed similarly for Cordell Hull reservoir. Training data must be provided in comma separated values (CSV) format for each intput and output variable, with all simulations combined in a single file.

1_Train_NARX_for_discharge_temp_DO.m

```
%% Discharge temp ANN - tweaked for OHL 2005 model
%Has sections for both temperature and DO neural nets

%% (1) - Load all data files
%Data files are csv for each input/output, with the first column being run
    ↪ number, the second column being JDAY, and following columns with data

clearvars
d=dir('DATA_FOR_TRAINING/*.csv');
for i=1:length(d)
   Dstr_max_structure(i).name=d(i).name;
   tic
   Dstr_max_structure(i).matrix=...
      csvread(['DATA_FOR_TRAINING/' d(i).name]);
   toc
end
clearvars d i ans
%Vector of Run IDs, where -1 is base case
RunIDs=unique(Dstr_max_structure(1).matrix(:,1));

%% (2) - Bring in data from each run

%Find index for each input in Dstr_max_structure
indexes.dischargeDO=...
   find(strcmp('dischargeDO.csv',{Dstr_max_structure.name})==1);
indexes.dischargeTemp=...
   find(strcmp('dischargeTemp.csv',{Dstr_max_structure.name})==1);
indexes.exogBR1DO=...
   find(strcmp('exogBR1DO.csv',{Dstr_max_structure.name})==1);
indexes.exogBR1Q=...
   find(strcmp('exogBR1Q.csv',{Dstr_max_structure.name})==1);
indexes.exogBR1T=...
   find(strcmp('exogBR1T.csv',{Dstr_max_structure.name})==1);
indexes.exogDODT=...
   find(strcmp('exogDODT.csv',{Dstr_max_structure.name})==1);
indexes.exogMET=...
   find(strcmp('exogMETBig.csv',{Dstr_max_structure.name})==1);
indexes.exogQDT=...
   find(strcmp('exogQDT.csv',{Dstr_max_structure.name})==1);
indexes.exogTDT=...
   find(strcmp('exogTDT.csv',{Dstr_max_structure.name})==1);
indexes.exogTR2DO=...
   find(strcmp('exogTR2DO.csv',{Dstr_max_structure.name})==1);
```

```
indexes.exogTR2Q=...
   find(strcmp('exogTR2Q.csv',{Dstr_max_structure.name})==1);
indexes.exogTR2T=...
   find(strcmp('exogTR2T.csv',{Dstr_max_structure.name})==1);
indexes.exogTurbSpill=...
   find(strcmp('exogTurbSpill.csv',{Dstr_max_structure.name})==1);

%Loop through all Run IDs
for i=1:size(RunIDs)
   RunID=RunIDs(i); fprintf([num2str(RunID), ' \n'])
   %Discharge DO
   [r,c]=find(Dstr_max_structure(indexes.dischargeDO).matrix(:,1)==RunID);
   Discharge.DO{i}=...
      unique(sortrows(Dstr_max_structure(indexes.dischargeDO).matrix(r,2:end)),'
          ↪ rows');
   %Discharge Temp
   [r,c]=find(Dstr_max_structure(indexes.dischargeTemp).matrix(:,1)==RunID);
   Discharge.temp{i}=...
      unique(sortrows(Dstr_max_structure(indexes.dischargeTemp).matrix(r,2:end))
          ↪ ,'rows');
   %BR1 Q, T, DO
   [r,c]=find(Dstr_max_structure(indexes.exogBR1Q).matrix(:,1)==RunID);
   Exog.BR1Q{i}=...
      unique(sortrows(Dstr_max_structure(indexes.exogBR1Q).matrix(r,2:end)),'
          ↪ rows');
   [r,c]=find(Dstr_max_structure(indexes.exogBR1T).matrix(:,1)==RunID);
   Exog.BR1T{i}=...
      unique(sortrows(Dstr_max_structure(indexes.exogBR1T).matrix(r,2:end)),'
          ↪ rows');
   [r,c]=find(Dstr_max_structure(indexes.exogBR1DO).matrix(:,1)==RunID);
   Exog.BR1DO{i}=...
      unique(sortrows(Dstr_max_structure(indexes.exogBR1DO).matrix(r,2:end)),'
          ↪ rows');
   %TR2 Q, T, DO
   [r,c]=find(Dstr_max_structure(indexes.exogTR2Q).matrix(:,1)==RunID);
   Exog.TR2Q{i}=...
      unique(sortrows(Dstr_max_structure(indexes.exogTR2Q).matrix(r,2:end)),'
          ↪ rows');
   [r,c]=find(Dstr_max_structure(indexes.exogTR2T).matrix(:,1)==RunID);
   Exog.TR2T{i}=...
      unique(sortrows(Dstr_max_structure(indexes.exogTR2T).matrix(r,2:end)),'
          ↪ rows');
   [r,c]=find(Dstr_max_structure(indexes.exogTR2DO).matrix(:,1)==RunID);
   Exog.TR2DO{i}=...
      unique(sortrows(Dstr_max_structure(indexes.exogTR2DO).matrix(r,2:end)),'
          ↪ rows');
   %Met
   [r,c]=find(Dstr_max_structure(indexes.exogMET).matrix(:,1)==RunID);
   Exog.met{i}=...
      unique(sortrows(Dstr_max_structure(indexes.exogMET).matrix(r,3:end)),'rows
          ↪ '); %skip col 2, which contains year right now
   %Turb, Spill
   [r,c]=find(Dstr_max_structure(indexes.exogTurbSpill).matrix(:,1)==RunID);
   Exog.turb_spill{i}=...
      unique(sortrows(Dstr_max_structure(indexes.exogTurbSpill).matrix(r,2:end))
          ↪ ,'rows');
   %QDT Q, T, DO
   [r,c]=find(Dstr_max_structure(indexes.exogQDT).matrix(:,1)==RunID);
   Exog.QDT{i}=...
      unique(sortrows(Dstr_max_structure(indexes.exogQDT).matrix(r,2:end)),'rows
          ↪ ');
   [r,c]=find(Dstr_max_structure(indexes.exogTDT).matrix(:,1)==RunID);
   Exog.TDT{i}=...
      unique(sortrows(Dstr_max_structure(indexes.exogTDT).matrix(r,2:end)),'rows
```

```
            ↪ ');
96      [r,c]=find(Dstr_max_structure(indexes.exogDODT).matrix(:,1)==RunID);
97      Exog.DODT{i}=...
98         unique(sortrows(Dstr_max_structure(indexes.exogDODT).matrix(r,2:end)),'
               ↪ rows');
99
100 end
101 clearvars ans c r i RunID indexes Dstr_max_structure
102
103 %% (3) - Define timestep and get raw data at these times using correct
        ↪ interpolation setting
104
105 timesteps=[121:(1/24):274]';
106 clearvars Inputs Output Inputs_seq Output_seq Discharge.temp_no0s Discharge.
        ↪ DO_no0s
107
108 %Make temperature Inputs and Outputs
109 for i=1:size(RunIDs)
110    fprintf([num2str(RunIDs(i)), ' \n'])
111    Inputs.discharge_temp{i}=[];
112    %BR1Q, BR1T - interpolation OFF
113    for ii=1:size(timesteps,1)
114       index1(ii)=find(Exog.BR1Q{i}(:,1)<=timesteps(ii),1,'last');
115       index2(ii)=find(Exog.BR1T{i}(:,1)<=timesteps(ii),1,'last');
116    end
117    Inputs.discharge_temp{i}(:,1:2)=[Exog.BR1Q{i}(index1,2) Exog.BR1T{i}(index2
           ↪ ,2)];
118    clearvars ii index1 index2
119    %TR2Q, TR2T, - interpolation ON
120    Inputs.discharge_temp{i}(:,end+1)=interp1(Exog.TR2Q{i}(:,1),Exog.TR2Q{i}(:,2)
           ↪ ,timesteps);
121    Inputs.discharge_temp{i}(:,end+1)=interp1(Exog.TR2T{i}(:,1),Exog.TR2T{i}(:,2)
           ↪ ,timesteps);
122    %Met - interpolation ON
123    Inputs.discharge_temp{i}(:,end+1:end+5)=interp1(Exog.met{i}(:,1),Exog.met{i
           ↪ }(:,2:end),timesteps);
124    %Turb & spill - interpolation ON
125    Inputs.discharge_temp{i}(:,end+1:end+2)=interp1(Exog.turb_spill{i}(:,1),Exog.
           ↪ turb_spill{i}(:,2:end),timesteps);
126
127    %Discharge temps output
128    %Option 1 - interpolate to remove timepoints with no discharge (temp=0)
129    % --> Use this for testing correlations (discontinuities mess this up)
130    index=find(Discharge.temp{i}(:,2)~=0);
131    %Remove rows with zeros (no discharge)
132    Discharge.temp_no0s{i}=Discharge.temp{i}(index,:);
133    %Smooth data
134    Discharge.temp_no0s_smooth{i}(:,1)=Discharge.temp_no0s{i}(:,1);
135    Discharge.temp_no0s_smooth{i}(:,2)=smooth(Discharge.temp_no0s{i}(:,1),
           ↪ Discharge.temp_no0s{i}(:,2),24);
136    discharge_temp_no0s{i}(:,1)=interp1(Discharge.temp_no0s_smooth{i}(:,1),
           ↪ Discharge.temp_no0s_smooth{i}(:,2),timesteps);
137    clearvars index
138    clearvars xlims ylims xrange yrange
139    index=find(Discharge.temp{i}(:,2)==0);
140    discharge_with_nans(:,1)=Discharge.temp{i}(:,1);
141    discharge_with_nans(:,2)=interp1(Discharge.temp_no0s_smooth{i}(:,1),Discharge
           ↪ .temp_no0s_smooth{i}(:,2),Discharge.temp{i}(:,1));
142    discharge_with_nans(index,2)=nan;
143    Output.discharge_temp{i}(:,1)=interp1(discharge_with_nans(:,1),
           ↪ discharge_with_nans(:,2:end),timesteps);
144    clearvars index discharge_with_nans
145
146    %Sensitive inputs seem to be BR1Q, BR1T, TR2T, 1st 2 cols in met,
```

```
147     %turb, spill
148     Inputs.discharge_temp{i}=Inputs.discharge_temp{i}(:,[1:2 4:6 10:11]);
149
150     %Convert to cells
151     Inputs_seq.discharge_temp{i} = con2seq(Inputs.discharge_temp{i}');
152     Output_seq.discharge_temp{i} = con2seq(Output.discharge_temp{i}');
153
154 end
155
156 %Make DO Inputs and Outputs
157 for i=1:size(RunIDs)
158     fprintf([num2str(RunIDs(i)), ' \n'])
159     Inputs.discharge_DO{i}=[];
160     %BR1Q, BR1T, BR1DO - interpolation OFF
161     for ii=1:size(timesteps,1)
162         index1(ii)=find(Exog.BR1Q{i}(:,1)<=timesteps(ii),1,'last');
163         index2(ii)=find(Exog.BR1T{i}(:,1)<=timesteps(ii),1,'last');
164         index3(ii)=find(Exog.BR1DO{i}(:,1)<=timesteps(ii),1,'last');
165     end
166     Inputs.discharge_DO{i}(:,1:3)=[Exog.BR1Q{i}(index1,2) ...
167         Exog.BR1T{i}(index2,2) Exog.BR1DO{i}(index3,2)];
168     clearvars ii index1 index2 index3 index4
169     %TR2Q, TR2T, TR2DO - interpolation ON
170     Inputs.discharge_DO{i}(:,end+1)=interp1(Exog.TR2Q{i}(:,1),Exog.TR2Q{i}(:,2),
            ↪ timesteps);
171     Inputs.discharge_DO{i}(:,end+1)=interp1(Exog.TR2T{i}(:,1),Exog.TR2T{i}(:,2),
            ↪ timesteps);
172     Inputs.discharge_DO{i}(:,end+1)=interp1(Exog.TR2DO{i}(:,1),Exog.TR2DO{i}(:,2)
            ↪ ,timesteps);
173     %Met - interpolation ON
174     Inputs.discharge_DO{i}(:,end+1:end+5)=interp1(Exog.met{i}(:,1),Exog.met{i
            ↪ }(:,2:end),timesteps);
175     %Turb & spill - interpolation ON
176     Inputs.discharge_DO{i}(:,end+1:end+2)=interp1(Exog.turb_spill{i}(:,1),Exog.
            ↪ turb_spill{i}(:,2:end),timesteps);
177
178     %Discharge DO output
179     index=find(Discharge.DO{i}(:,2)~=0);
180     %Remove rows with zeros (no discharge)
181     Discharge.DO_no0s{i}=Discharge.DO{i}(index,:);
182     %Smooth data
183     Discharge.DO_no0s_smooth{i}(:,1)=Discharge.DO_no0s{i}(:,1);
184     Discharge.DO_no0s_smooth{i}(:,2)=smooth(Discharge.DO_no0s{i}(:,1),Discharge.
            ↪ DO_no0s{i}(:,2),24);
185     discharge_DO_no0s{i}(:,1)=interp1(Discharge.DO_no0s_smooth{i}(:,1),Discharge.
            ↪ DO_no0s_smooth{i}(:,2),timesteps);
186     clearvars index
187     index=find(Discharge.DO{i}(:,2)==0);
188     discharge_with_nans(:,1)=Discharge.DO{i}(:,1);
189     discharge_with_nans(:,2)=interp1(Discharge.DO_no0s_smooth{i}(:,1),Discharge.
            ↪ DO_no0s_smooth{i}(:,2),Discharge.DO{i}(:,1));
190     discharge_with_nans(index,2)=nan;
191     Output.discharge_DO{i}(:,1)=interp1(discharge_with_nans(:,1),
            ↪ discharge_with_nans(:,2:end),timesteps);
192     clearvars index discharge_with_nans
193
194     %Sensitive inputs seem to be BR1Q, BR1T, BR1DO, TR2T, TR2DO, 1st 2 cols in
            ↪ met, turb, spill
195     Inputs.discharge_DO{i}=Inputs.discharge_DO{i}(:,[1:3 5:6 7:8 12:13]);
196
197     %Convert to cells
198     Inputs_seq.discharge_DO{i} = con2seq(Inputs.discharge_DO{i}');
199     Output_seq.discharge_DO{i} = con2seq(Output.discharge_DO{i}');
200
```

```
end
clearvars i

%% Check for input delays and correlations
clearvars Temp_correlations DO_correlations
for i=1:size(RunIDs)
   for ii=1:size(Inputs.discharge_temp{i},2)
      %Temp
      figure
      crosscorr(Inputs.discharge_temp{i}(:,ii),discharge_temp_no0s{i},30)
      [r,lags]=xcorr(Inputs.discharge_temp{i}(:,ii)-mean(Inputs.discharge_temp{i
          ↪ }(:,ii)),discharge_temp_no0s{i}-mean(discharge_temp_no0s{i}),30,'
          ↪ coeff');
      [~,b]=max(abs(r));
      Temp_correlations{ii}(i,:)=[r(b) lags(b)];
   end
end
for i=1:size(RunIDs)
   for ii=1:size(Inputs.discharge_DO{i},2)
      %DO
         figure
         crosscorr(Inputs.discharge_DO{i}(:,ii),discharge_DO_no0s{i},30)
      [r,lags]=xcorr(Inputs.discharge_DO{i}(:,ii)-mean(Inputs.discharge_DO{i}(:,
          ↪ ii)),discharge_DO_no0s{i}-mean(discharge_DO_no0s{i}),30,'coeff');
      [~,b]=max(abs(r));
      DO_correlations{ii}(i,:)=[r(b) lags(b)];
   end
end
clearvars b r lags

%% (4) - Define training and validation sets and combine into cell arrays

%Define validation & training sets
validation_indexes=sort(randsample(size(RunIDs,1),round(.3*size(RunIDs,1)),'
    ↪ false'));
training_indexes=setdiff(1:size(RunIDs,1),validation_indexes)';

%Combine them all into single Input and Output cell arrays
%Training set
tic
Inputs_seq_mul.discharge_temp_train=catsamples(Inputs_seq.discharge_temp{
    ↪ training_indexes},'pad');
Output_seq_mul.discharge_temp_train=catsamples(Output_seq.discharge_temp{
    ↪ training_indexes},'pad');
Inputs_seq_mul.discharge_DO_train=catsamples(Inputs_seq.discharge_DO{
    ↪ training_indexes},'pad');
Output_seq_mul.discharge_DO_train=catsamples(Output_seq.discharge_DO{
    ↪ training_indexes},'pad');
toc
%Validation set
tic
Inputs_seq_mul.discharge_temp_valid=catsamples(Inputs_seq.discharge_temp{
    ↪ validation_indexes},'pad');
Output_seq_mul.discharge_temp_valid=catsamples(Output_seq.discharge_temp{
    ↪ validation_indexes},'pad');
Inputs_seq_mul.discharge_DO_valid=catsamples(Inputs_seq.discharge_DO{
    ↪ validation_indexes},'pad');
Output_seq_mul.discharge_DO_valid=catsamples(Output_seq.discharge_DO{
    ↪ validation_indexes},'pad');
toc

%% (5) - Train temp model
clearvars ohl_temp_narx
clearvars ame_temp_training ame_temp_validation ameavg_temp_training
```

```
    ↪ ameavg_temp_validation
savename='ohl_temp_narx_20160906';
for i=1:5
   fprintf(['Training model #', num2str(i), '\n'])
   inputDelays = [0 1 12];
   feedbackDelays = [1];
   hiddenNeurons=[10];
   narx_net = narxnet(inputDelays,feedbackDelays,hiddenNeurons);
   % For a list of all data division functions type: help nndivide
   narx_net.divideFcn = 'dividerand';
   % The property DIVIDEMODE set to TIMESTEP means that targets are divided
   % into training, validation and test sets according to timesteps.
   % For a list of data division modes type: help nntype_data_division_mode
   narx_net.divideMode = 'time'; % Divide up every value
   narx_net.divideParam.trainRatio = 70/100;
   narx_net.divideParam.valRatio = 15/100;
   narx_net.divideParam.testRatio = 15/100;
   narx_net.trainParam.min_grad = 1e-10;
   narx_net.trainFcn = 'trainlm';
   narx_net.trainParam.showWindow=0;
   narx_net.trainParam.showCommandLine=1;
   narx_net.trainParam.show=100;
   [Xs,Xi,Ai,Ts] = preparets(narx_net,Inputs_seq_mul.discharge_temp_train,{},
       ↪ ...
      Output_seq_mul.discharge_temp_train);
   tic
   [narx_net,tr]=train(narx_net,Xs,Ts,Xi,Ai,'UseParallel','yes');
   ohl_temp_narx.train_time{i}(1,1)=toc;
   tic
   %Convert to closed loop
   narx_net_closed = closeloop(narx_net);
   narx_net_closed.trainParam.mu_max=1e14;
   narx_net_closed.TrainParam.epochs=3000;
   %Continue training as a closed loop - as suggested here: http://www.mathworks
       ↪ .com/matlabcentral/answers/89070-narx-model-training-in-the-neural-
       ↪ network-tool-box
   [Xs,Xi,Ai,Ts] = preparets(narx_net_closed,Inputs_seq_mul.discharge_temp_train
       ↪ ,{}, ...
      Output_seq_mul.discharge_temp_train);
   [narx_net_closed,tr] = train(narx_net_closed,Xs,Ts,Xi,Ai,'UseParallel','yes')
       ↪ ;
   ohl_temp_narx.train_time{i}(1,2)=toc;

   %% (6) - Save it all in one stucture, for input in optimization problem
   ohl_temp_narx.Inputs=Inputs.discharge_temp;
   ohl_temp_narx.Output=Output.discharge_temp;
   ohl_temp_narx.Discharge_temp_no0s=Discharge.temp_no0s; %save interpolated set
       ↪  in case starting condition is at NaN entry
   ohl_temp_narx.Discharge_temp_no0s_smooth=Discharge.temp_no0s_smooth; %save
       ↪ interpolated set in case starting condition is at NaN entry
   ohl_temp_narx.turb_column=6;
   ohl_temp_narx.spill_column=7;
   ohl_temp_narx.inputDelays=inputDelays;
   ohl_temp_narx.feedbackDelays=feedbackDelays;
   ohl_temp_narx.input_variables={'QIN_BR1','TIN_BR1','TTR_TR2',...
      'MET_WB1','MET_WB1','QOT_BR1_T','QOT_BR1_S';...
      1,1,1,1,2,1,1};
   ohl_temp_narx.narx_net_closed{i}=narx_net_closed;
   save(savename, 'ohl_temp_narx')

   %% (7) - Predict full time series
   for run=validation_indexes'
      u=Inputs_seq.discharge_temp{run};
      y=Output_seq.discharge_temp{run};
```

```
309         y1 = y(1:size(timesteps,1));
310         u1 = u(1:size(timesteps,1));
311         [p1,Pi1,Ai1,t1] = preparets(ohl_temp_narx.narx_net_closed{i},u1,{},y1);
312         yp1 = ohl_temp_narx.narx_net_closed{i}(p1,Pi1,Ai1);
313         %Remove plotting for indexes where discharge=0
314         t1=cell2mat(t1);
315         yp1=cell2mat(yp1);
316         start=(max([ohl_temp_narx.inputDelays';...
317            ohl_temp_narx.feedbackDelays'])+1);
318         indexes=find(isnan(Output.discharge_temp{run}(start:end,end)));
319         t1(1,indexes)=nan;
320         yp1(1,indexes)=nan;
321         ame_temp_validation{i}(run)=nanmean(abs(t1-yp1));
322      end
323
324      for run=training_indexes'
325         u=Inputs_seq.discharge_temp{run};
326         y=Output_seq.discharge_temp{run};
327         y1 = y(1:size(timesteps,1));
328         u1 = u(1:size(timesteps,1));
329         [p1,Pi1,Ai1,t1] = preparets(ohl_temp_narx.narx_net_closed{i},u1,{},y1);
330         yp1 = ohl_temp_narx.narx_net_closed{i}(p1,Pi1,Ai1);
331         %Remove plotting for indexes where discharge=0
332         t1=cell2mat(t1);
333         yp1=cell2mat(yp1);
334         start=(max([ohl_temp_narx.inputDelays';...
335            ohl_temp_narx.feedbackDelays'])+1);
336         indexes=find(isnan(Output.discharge_temp{run}(start:end,end)));
337         t1(1,indexes)=nan;
338         yp1(1,indexes)=nan;
339         ame_temp_training{i}(run)=nanmean(abs(t1-yp1));
340      end
341
342      ameavg_temp_training{i}=sum(ame_temp_training{i})./sum(ame_temp_training{i
         ↪ }~=0);
343      ameavg_temp_validation{i}=sum(ame_temp_validation{i})./sum(
         ↪ ame_temp_validation{i}~=0);
344      save(savename, 'ohl_temp_narx','ameavg_temp_training','ameavg_temp_validation
         ↪ ',...
345         'ame_temp_training','ame_temp_validation');
346   end
347
348   %% (8) - Train DO model
349   clearvars ohl_DO_narx
350   clearvars ame_DO_training ame_DO_validation ameavg_DO_training
       ↪ ameavg_DO_validation
351   savename='ohl_DO_narx_20160906';
352   for i=1:5
353      fprintf(['Training model #', num2str(i), '\n'])
354      inputDelays = [0 1 12];
355      feedbackDelays = [1];
356      hiddenNeurons=[10];
357      narx_net = narxnet(inputDelays,feedbackDelays,hiddenNeurons);
358      % For a list of all data division functions type: help nndivide
359      narx_net.divideFcn = 'dividerand';
360      % The property DIVIDEMODE set to TIMESTEP means that targets are divided
361      % into training, validation and test sets according to timesteps.
362      % For a list of data division modes type: help nntype_data_division_mode
363      narx_net.divideMode = 'time'; % Divide up every value
364      narx_net.divideParam.trainRatio = 70/100;
365      narx_net.divideParam.valRatio = 15/100;
366      narx_net.divideParam.testRatio = 15/100;
367      narx_net.trainParam.min_grad = 1e-10;
368      narx_net.trainFcn = 'trainlm';
```

```
    narx_net.trainParam.showWindow=0;
    narx_net.trainParam.showCommandLine=1;
    narx_net.trainParam.show=100;
    [Xs,Xi,Ai,Ts] = preparets(narx_net,Inputs_seq_mul.discharge_DO_train,{}, ...
       Output_seq_mul.discharge_DO_train);

    tic
    [narx_net,tr]=train(narx_net,Xs,Ts,Xi,Ai,'UseParallel','yes');
    ohl_DO_narx.train_time{i}(1,1)=toc;
    tic
    %Convert to closed loop
    narx_net_closed = closeloop(narx_net);
    narx_net_closed.trainParam.mu_max=1e12;
    narx_net_closed.TrainParam.epochs=3000;
    %Continue training as a closed loop - as suggested here: http://www.mathworks
        ↪ .com/matlabcentral/answers/89070-narx-model-training-in-the-neural-
        ↪ network-tool-box
    [Xs,Xi,Ai,Ts] = preparets(narx_net_closed,Inputs_seq_mul.discharge_DO_train
        ↪ ,{}, ...
       Output_seq_mul.discharge_DO_train);
    [narx_net_closed,tr] = train(narx_net_closed,Xs,Ts,Xi,Ai,'UseParallel','yes')
        ↪ ;
    ohl_DO_narx.train_time{i}(1,2)=toc;

    %% (9) - Save it all in one stucture, for input in optimization problem
    ohl_DO_narx.Inputs=Inputs.discharge_DO;
    ohl_DO_narx.Output=Output.discharge_DO;
    ohl_DO_narx.Discharge_DO_no0s=Discharge.DO_no0s; %save interpolated set in
        ↪ case starting condition is at NaN entry
    ohl_DO_narx.turb_column=8;
    ohl_DO_narx.spill_column=9;
    ohl_DO_narx.inputDelays=inputDelays;
    ohl_DO_narx.feedbackDelays=feedbackDelays;
    ohl_DO_narx.input_variables={'QIN_BR1','TIN_BR1','CIN_BR1',...
       'TTR_TR2','CTR_TR2','MET_WB1','MET_WB1','QOT_BR1_T','QOT_BR1_S';...
       1,1,1,1,1,1,2,1,1};
    ohl_DO_narx.narx_net_closed{i}=narx_net_closed;
    save(savename, 'ohl_DO_narx')

    %% (10) - Predict full time series
    for run=validation_indexes'
       u=Inputs_seq.discharge_DO{run};
       y=Output_seq.discharge_DO{run};
       y1 = y(1:size(timesteps,1));
       u1 = u(1:size(timesteps,1));
       [p1,Pi1,Ai1,t1] = preparets(ohl_DO_narx.narx_net_closed{i},u1,{},y1);
       yp1 = ohl_DO_narx.narx_net_closed{i}(p1,Pi1,Ai1);
       %Remove plotting for indexes where discharge=0
       t1=cell2mat(t1);
       yp1=cell2mat(yp1);
       start=(max([ohl_DO_narx.inputDelays';...
          ohl_DO_narx.feedbackDelays'])+1);
       indexes=find(isnan(Output.discharge_DO{run}(start:end,end)));
       t1(1,indexes)=nan;
       yp1(1,indexes)=nan;
       ame_DO_validation{i}(run)=nanmean(abs(t1-yp1));
    end

    for run=training_indexes'
       u=Inputs_seq.discharge_DO{run};
       y=Output_seq.discharge_DO{run};
       y1 = y(1:size(timesteps,1));
       u1 = u(1:size(timesteps,1));
       [p1,Pi1,Ai1,t1] = preparets(ohl_DO_narx.narx_net_closed{i},u1,{},y1);
```

```
      yp1 = ohl_DO_narx.narx_net_closed{i}(p1,Pi1,Ai1);
      %Remove plotting for indexes where discharge=0
      t1=cell2mat(t1);
      yp1=cell2mat(yp1);
      start=(max([ohl_DO_narx.inputDelays';...
         ohl_DO_narx.feedbackDelays'])+1);
      indexes=find(isnan(Output.discharge_DO{run}(start:end,end)));
      t1(1,indexes)=nan;
      yp1(1,indexes)=nan;
      ame_DO_training{i}(run)=nanmean(abs(t1-yp1));
   end

   ameavg_DO_training{i}=sum(ame_DO_training{i})./sum(ame_DO_training{i}~=0);
   ameavg_DO_validation{i}=sum(ame_DO_validation{i})./sum(ame_DO_validation{i
      ↪ }~=0);
   save(savename, 'ohl_DO_narx','ameavg_DO_training','ameavg_DO_validation',...
      'ame_DO_training','ame_DO_validation');
end
```

2a_Compute_weights_for_DO_model.m

```
clearvars yp1 t1 residuals
for i=1:size(ohl_DO_narx.narx_net_closed,2)
   fprintf(['NARX model #', num2str(i), '\n'])
   for run=validation_indexes'
      u=Inputs_seq.discharge_DO{run};
      y=Output_seq.discharge_DO{run};
      y1 = y(1:size(timesteps,1));
      u1 = u(1:size(timesteps,1));
      [p1,Pi1,Ai1,t1{run}] = preparets(ohl_DO_narx.narx_net_closed{i},u1,{},y1);
      yp1{run}(i,:) = ohl_DO_narx.narx_net_closed{i}(p1,Pi1,Ai1);
   end
end
for run=validation_indexes'
   %Remove plotting for indexes where discharge=0
   t1{run}=cell2mat(t1{run});
   yp1{run}=cell2mat(yp1{run});
   start=(max([ohl_DO_narx.inputDelays';...
      ohl_DO_narx.feedbackDelays'])+1);
   indexes=find(isnan(Output.discharge_DO{run}(start:end,end)));
   t1{run}(1,indexes)=nan;
   yp1{run}(:,indexes)=nan;
end
for i=1:size(ohl_DO_narx.narx_net_closed,2)
   for run=validation_indexes'
      residuals(i,run)=nanmean(yp1{run}(i,:)-t1{run});
   end
end
for i=1:size(ohl_DO_narx.narx_net_closed,2)
   count=1;
   for j=validation_indexes(:)'
   residuals_validationonly(i,count)=residuals(i,j);
   count=count+1;
   end
end
clearvars j i
ohl_DO_narx.bias=mean(residuals_validationonly')';
for i=1:size(ohl_DO_narx.narx_net_closed,2)
   fprintf(['NARX model #', num2str(i), '\n'])
   for run=validation_indexes'
      yp1{run}(i,:)=yp1{run}(i,:)-ohl_DO_narx.bias(i);
      mean_of_square_errors{i}(run)=nanmean((t1{run}-yp1{run}(i,:)).^2);
   end
```

```
end
for i=1:size(ohl_DO_narx.narx_net_closed,2)
   count=1;
   for j=validation_indexes(:)'
      mse_validationonly(i,count)=mean_of_square_errors{i}(j);
      count=count+1;
   end
end
clearvars j i

%% Optimize weights
init_weights=ones(1,size(ohl_DO_narx.narx_net_closed,2))*(1/size(ohl_DO_narx.
    ↪ narx_net_closed,2));
Aeq=ones(1,size(ohl_DO_narx.narx_net_closed,2));
beq=1;
lb=zeros(1,size(ohl_DO_narx.narx_net_closed,2));
ub=ones(1,size(ohl_DO_narx.narx_net_closed,2));
options=optimset('Display','iter-detailed');
FitnessFunction=@(weights) optimal_weights(weights,validation_indexes,t1,yp1);
[weights,avg_mse]=fmincon(FitnessFunction,init_weights,[],[],Aeq,beq,lb,ub,[],
    ↪ options);
weights=weights';
%Remove the networks with weights <25% the max weight
lb(find(weights/max(weights)<(1/4)))=0;
ub(find(weights/max(weights)<(1/4)))=0;
[weights,avg_mse]=fmincon(FitnessFunction,init_weights,[],[],Aeq,beq,lb,ub,[],
    ↪ options);
weights=weights';
%Save weights, bias, and networks into final stucture
indexes=find(weights~=0);
ohl_DO_narx.weights=weights(indexes);
ohl_DO_narx.bias=ohl_DO_narx.bias(indexes);
for i=1:size(indexes,1)
   ohl_DO_narx.narx_net_closed3{i}=ohl_DO_narx.narx_net_closed{indexes(i)};
end
ohl_DO_narx.narx_net_closed=ohl_DO_narx.narx_net_closed3;
ohl_DO_narx=rmfield(ohl_DO_narx,'narx_net_closed3');
```

2b_Compute_weights_for_temp_model.m

```
clearvars yp1 t1 residuals
for i=1:size(ohl_temp_narx.narx_net_closed,2)
   fprintf(['NARX model #', num2str(i), '\n'])
   for run=validation_indexes'
      u=Inputs_seq.discharge_temp{run};
      y=Output_seq.discharge_temp{run};
      y1 = y(1:size(timesteps,1));
      u1 = u(1:size(timesteps,1));
      [p1,Pi1,Ai1,t1{run}] = preparets(ohl_temp_narx.narx_net_closed{i},u1,{},y1
          ↪ );
      yp1{run}(i,:) = ohl_temp_narx.narx_net_closed{i}(p1,Pi1,Ai1);
   end
end
for run=validation_indexes'
   %Remove plotting for indexes where discharge=0
   t1{run}=cell2mat(t1{run});
   yp1{run}=cell2mat(yp1{run});
   start=(max([ohl_temp_narx.inputDelays';...
      ohl_temp_narx.feedbackDelays'])+1);
   indexes=find(isnan(Output.discharge_temp{run}(start:end,end)));
   t1{run}(1,indexes)=nan;
   yp1{run}(:,indexes)=nan;
end
```

```
for i=1:size(ohl_temp_narx.narx_net_closed,2)
   for run=validation_indexes'
      residuals(i,run)=nanmean(yp1{run}(i,:)-t1{run});
   end
end
for i=1:size(ohl_temp_narx.narx_net_closed,2)
   count=1;
   for j=validation_indexes(:)'
   residuals_validationonly(i,count)=residuals(i,j);
   count=count+1;
   end
end
clearvars j i
ohl_temp_narx.bias=mean(residuals_validationonly')';
for i=1:size(ohl_temp_narx.narx_net_closed,2)
   fprintf(['NARX model #', num2str(i), '\n'])
   for run=validation_indexes'
      yp1{run}(i,:)=yp1{run}(i,:)-ohl_temp_narx.bias(i);
      mean_of_square_errors{i}(run)=nanmean((t1{run}-yp1{run}(i,:)).^2);
   end
end
for i=1:size(ohl_temp_narx.narx_net_closed,2)
   count=1;
   for j=validation_indexes(:)'
      mse_validationonly(i,count)=mean_of_square_errors{i}(j);
      count=count+1;
   end
end
clearvars j i

%% Optimize weights
init_weights=ones(1,size(ohl_temp_narx.narx_net_closed,2))*(1/size(ohl_temp_narx
    ↪ .narx_net_closed,2));
Aeq=ones(1,size(ohl_temp_narx.narx_net_closed,2));
beq=1;
lb=zeros(1,size(ohl_temp_narx.narx_net_closed,2));
ub=ones(1,size(ohl_temp_narx.narx_net_closed,2));
options=optimset('Display','iter-detailed');
FitnessFunction=@(weights) optimal_weights(weights,validation_indexes,t1,yp1);
[weights,avg_mse]=fmincon(FitnessFunction,init_weights,[],[],Aeq,beq,lb,ub,[],
    ↪ options);
weights=weights';
%Remove the networks with weights <25% the max weight
lb(find(weights/max(weights)<(1/4)))=0;
ub(find(weights/max(weights)<(1/4)))=0;
[weights,avg_mse]=fmincon(FitnessFunction,init_weights,[],[],Aeq,beq,lb,ub,[],
    ↪ options);
weights=weights';
%Save weights, bias, and networks into final stucture
indexes=find(weights~=0);
ohl_temp_narx.weights=weights(indexes);
ohl_temp_narx.bias=ohl_temp_narx.bias(indexes);
for i=1:size(indexes,1)
   ohl_temp_narx.narx_net_closed3{i}=ohl_temp_narx.narx_net_closed{indexes(i)};
end
ohl_temp_narx.narx_net_closed=ohl_temp_narx.narx_net_closed3;
ohl_temp_narx=rmfield(ohl_temp_narx,'narx_net_closed3');
```

optimal_weights.m

```
function avg_mse=optimal_weights(weights,validation_indexes,t1,yp1)

weights=weights';
```

```
4
5  for run=validation_indexes(:)'
6     weighted_mse(run)=nanmean((t1{run}-sum(bsxfun(@times,weights,yp1{run}))).^2);
7     weightedNARXame(run)=nanmean(abs(t1{run}-sum(bsxfun(@times,weights,yp1{run}))
          ↪ ));
8  end
9
10 count=1;
11 for j=validation_indexes(:)'
12    weighted_mse_validation(count)=weighted_mse(j);
13    count=count+1;
14 end
15 clearvars j i
16
17 avg_mse=mean(weighted_mse_validation,2);
```

Appendix D

MATLAB® CODE FOR HYDROPOWER OPTIMIZATION UNDER WATER QUALITY CONSTRAINTS

The following code can be used to optimize multiple reservoirs linked in series on an hourly timestep over multiple days, as described in Chapter III. Each day is optimized individually, creating a series of daily sub-problems. A configuration file defines general optimization settings and the layout of waterbodies, and each waterbody has an additional configuration file defining reservoir characteristics and constraints. The base file of the optimizer is main.m. The user must supply:

1. An already-trained water quality NARX surrogate model in order to use water quality constraints.
2. A CE-QUAL-W2 base folder for each reservoir.
3. Each CE-QUAL-W2 input and output file reconfigured as individual CSV files.
4. A CSV file defining inflow and withdrawal interpolation settings as determined from the CE-QUAL-W2 configuration file.

config.json

```
{
      "jdayStart": "215",
      "OperatingPeriod": "10",
      "LogFile": "results/results_log.txt",
      "NumberOfWaterbodies": "1",
      "wb1config": "config_OHL.json"
}
```

config_OHL.json

```
{
      "Name": "Old Hickory",
      "WaterSurfaceElevationInitial": "",
      "DischargeDOInitial": "",
      "DischargeTempInitial": "",
      "WaterSurfaceElevationMin": "134.722",
      "WaterSurfaceElevationMax": "135.636",
      "DischargeDOMin": "6",
      "DischargeDOMax": "",
      "DischargeTempMin": "",
      "DischargeTempMax": "",
      "MaxHourlyChangeInTurbineUnit": "1",
      "MaxHoursWithZeroGeneration": "6",
      "NumberOfTurbineUnits": "4",
```

```
      "MWRatingPerTurbineUnit": "25",
      "TurbineDischargeCurve": "OHL/testfiles/turbine_discharge_curve_25MW.txt",
      "StorageElevationCurve": "OHL/testfiles/storage_elevation.txt",
      "TailWaterRatingCurve": "OHL/testfiles/tailwater_rating.txt",
      "DailyCostCurve": "OHL/testfiles/cost_curve2.txt",
      "TrainedDONeuralNetworkFile": "OHL/testfiles/ohl_DO_narx_20160906.mat",
      "TrainedTempNeuralNetworkFile": "OHL/testfiles/ohl_temp_narx_20160906.mat
          ↪ ",
      "WaterSurfaceElevationTargets": "",
      "optimizationDir": "OHL/testfiles/optimization215/",
      "ForecastTurbinePattern": "OHL/testfiles/forecast_turbine_pattern215.txt",
      "PreviousTurbinePattern": "OHL/testfiles/previous_turbine_pattern215.txt",
      "w2inputDir": "OHL/testfiles/w2input215/",
      "TurbSpillOrder": "1",
      "MainstemBR1Qin": "qin_br1.npt",
      "MainstemBR1Tin": "tin_br1_2005.npt",
      "MainstemBR1Cin": "cin_br1_2005.npt"
}
```

main.m

```
function main(configfile)

%% Startup: Empty vars, setup paths, check input, init config
clearvars -except configfile

% add path to 'lib' folder
if (~isdeployed)
   addpath('./lib');
end

% load general config
config=loadjson('config.json');
%Load config for each waterbody, as defined in general config
for wb=1:str2double(config.NumberOfWaterbodies)
   CFG{wb}=loadjson(eval(['config.wb' num2str(wb) 'config']));
end

% create logger
L = log4m.getLogger('optimization_run.log');

%% Load in data and set constraints and system specs

%TOTAL time period to optimize on
start_date=str2double(config.jdayStart);
frequency=1/24;
days_forward=str2double(config.OperatingPeriod);
t=[start_date:frequency:start_date+1];
%GA population sizes
ga_pop_size=480*size(CFG,2); %max(240,size(CFG,2)*(size(t,2)-1)*10);
feasiblilitycheck_ga_pop_size=360*size(CFG,2);

for wb=1:size(CFG,2)
   %Number of turbines - 4 for OHL
   no_of_units{wb}=str2double(CFG{wb}.NumberOfTurbineUnits);
   %Operating level, MW
   MW_rating{wb}=str2double(CFG{wb}.MWRatingPerTurbineUnit);
      %Previous elevations
   elevtemp{wb}=dlmread(strcat(CFG{wb}.optimizationDir,filesep,'ELWS.csv'),','
       ↪ ,1,0);
   %Elevation constraints - general
   ELWS_limit{wb}(1)=str2double(CFG{wb}.WaterSurfaceElevationMin);
   ELWS_limit{wb}(2)=str2double(CFG{wb}.WaterSurfaceElevationMax);
```

```
   %Max hourly unit change constraint
   if ~isempty(CFG{wb}.MaxHourlyChangeInTurbineUnit)
      max_hrly_unit_change{wb}=str2double(CFG{wb}.MaxHourlyChangeInTurbineUnit);
   else
      max_hrly_unit_change{wb}=[];
   end
   %Zero generation hourly limit - can't go longer than this with no turb flow
   if ~isempty(CFG{wb}.MaxHoursWithZeroGeneration)
      zero_gen_limit{wb}=str2double(CFG{wb}.MaxHoursWithZeroGeneration);
   else
      zero_gen_limit{wb}=[];
   end
   %DO discharge NARX model
   if isempty(CFG{wb}.TrainedDONeuralNetworkFile)
         WQ{wb}.DO_narx=[];
      else
         WQ{wb}.DO_narx=load(CFG{wb}.TrainedDONeuralNetworkFile);
         fn=fieldnames(WQ{wb}.DO_narx); WQ{wb}.DO_narx=WQ{wb}.DO_narx.(fn{1})
             ↪ ; clearvars fn
      end
   WQ{wb}.DO_limit(1)=str2double(CFG{wb}.DischargeDOMin);
   WQ{wb}.DO_limit(2)=str2double(CFG{wb}.DischargeDOMax);
   WQ{wb}.DO_slack=0;
   %Temperature discharge NARX model
   if isempty(CFG{wb}.TrainedTempNeuralNetworkFile)
         WQ{wb}.Temp_narx=[];
      else
         WQ{wb}.Temp_narx=load(CFG{wb}.TrainedTempNeuralNetworkFile);
         fn=fieldnames(WQ{wb}.Temp_narx); WQ{wb}.Temp_narx=WQ{wb}.Temp_narx.(
             ↪ fn{1});
         clearvars fn
      end
      WQ{wb}.Temp_limit(1)=str2double(CFG{wb}.DischargeTempMin);
      WQ{wb}.Temp_limit(2)=str2double(CFG{wb}.DischargeTempMax);
   WQ{wb}.Temp_slack=0;
   %Cost curve
   if isempty(CFG{wb}.DailyCostCurve)
      cost_curve_MW{wb}=[0 1];
   else
      cost_curve_MW{wb}=dlmread(CFG{wb}.DailyCostCurve,' ',1,0);
   end
   %Turbine discharge curve - meters, cms at MW_rating
   turbine_discharge{wb}=dlmread(CFG{wb}.TurbineDischargeCurve,' ',1,0);
      %Find initial elevation
   ic_elev_first{wb}=interp1(elevtemp{wb}(:,1),elevtemp{wb}(:,2),start_date);
   %Build the variable Q, which includes all flows for water balance,
       ↪ interpolation settings, tw curve both tabular discharge vs. tw and tw
       ↪ as f(twprev,discharge)), se curve, and other WQ inputs needed for NARX
       ↪  predictions
   Q{wb}=buildQ(CFG{wb}.optimizationDir);
   Q{wb}.tw_curve_cms_m=dlmread(CFG{wb}.TailWaterRatingCurve,' ',1,0);
   Q{wb}.SE_meters_m3=dlmread(CFG{wb}.StorageElevationCurve,' ',1,0);
      %Save a copy of Q as original projected values - Q will update during
          ↪ optimziation
      Qprojected=Q;
end

t_all=[start_date:frequency:start_date+days_forward];
t_all_round=roundn(t_all,-2);
tprev=[t(1)-max(cell2mat(zero_gen_limit(:)))*frequency:frequency:t(1)];
tprev_round=roundn(tprev,-2);
for wb=1:size(CFG,2)
      %Forecast turbine pattern (if supplied)
      if isempty(CFG{wb}.ForecastTurbinePattern)
```

```
100             L.warn('INITIALIZATION',['No reservoir ', num2str(wb), ' forecast
                    ↪ turbine pattern provided - assuming from turbine flows in W2
                    ↪ QOT file.'])
101             x0_all(wb,:)=actual_turb_ops(t_all_round,Qprojected{wb},elevtemp{wb
                    ↪ },turbine_discharge{wb},...
102                   no_of_units{wb});
103       else
104             forecastturbpattern=dlmread(CFG{wb}.ForecastTurbinePattern,'\t',1,0)
                    ↪ ;
105             for i=1:size(t_all_round,2)-1
106                   index=find(forecastturbpattern(:,1)<=t_all_round(i+1));
107                   x0_all(wb,i)=forecastturbpattern(index(end),2);
108             end
109             clearvars i forecastturbpattern index
110       end
111       %Previous turbine pattern for the year (if supplied)
112       if isempty(CFG{wb}.ForecastTurbinePattern)
113             L.warn('INITIALIZATION',['No reservoir ', num2str(wb), ' previous
                    ↪ turbine pattern provided - assuming from turbine flows in W2
                    ↪ QOT file.'])
114             xprev{wb}=actual_turb_ops(tprev_round,Qprojected{wb},elevtemp{wb},
                    ↪ turbine_discharge{wb},no_of_units{wb});
115       else
116             prevturbpattern=dlmread(CFG{wb}.PreviousTurbinePattern,'\t',1,0);
117             for i=1:size(tprev_round,2)
118                   index=find(prevturbpattern(:,1)<=tprev_round(i));
119                   xprev{wb}(i)=prevturbpattern(index(end),2);
120             end
121             clearvars i prevturbpattern index
122    end
123       %Target elevations (soft constraint)
124    if isempty(CFG{wb}.WaterSurfaceElevationTargets)
125       L.warn('INITIALIZATION',['No reservoir ', num2str(wb),' ELWS targets
              ↪ provided - assuming targets from projected operations W2 simulation
              ↪ .'])
126       [~,~,HWs_x0,~,~]=activeunits_to_discharges(x0_all(wb,:),t_all,...
127                   frequency,Qprojected{wb},ic_elev_first{wb},...
128          turbine_discharge{wb},[],[],[]);
129             ELWS_targets{wb}(:,1)=[start_date+1:1:start_date+days_forward]';
130             ELWS_targets{wb}(:,2)=interp1(t_all,HWs_x0,...
131                   [start_date+1:1:start_date+days_forward])';
132       if isnan(ELWS_targets{wb}(end,2))
133          ELWS_targets{wb}(end,2)=elevtemp{wb}(end,2);
134       end
135    else
136       ELWS_targets{wb}=dlmread(CFG{wb}.WaterSurfaceElevationTargets,'\t',1,0);
137    end
138       ELWS_targets{wb}(:,2)=min(ELWS_targets{wb}(:,2),ELWS_limit{wb}(2));
139       ELWS_targets{wb}(:,2)=max(ELWS_targets{wb}(:,2),ELWS_limit{wb}(1));
140       clearvars HWs_x0
141 end
142 clearvars wb t_all_round t_prev_round elevtemp
143 %Soft penalty coeff for deviation from final target elevation
144 elev_soft_penalty_coeff_constant=[1e3 5e2];
145 %Water quality and elevation constraint rounding setting (10=tenths place, 100=
        ↪ hundredths place, etc.)
146 elev_constraint_rounding=100;
147 wq_constraint_rounding=100;
148 %Assign priority ranking for constraints on elev, DO, and temp, starting with
        ↪ highest priority first. This is used during the prescreen to see if
        ↪ constraints are even feasible
149 ranking={'elev','do','temp'};
150 %Penalty tolerance
151 tolerance=10^-8;
```

```
152
153 retraining='Y';
154 iter=0; best_iter=[];
155
156 fileID=fopen(config.LogFile,'w');
157 fprintf(fileID,'%12s %12s %12s %12s %12s %12s %12s','Iter','Fcn_Evals','Time(s)'
      ↪ ,'Proj_MWh','Tot_MWh','Proj_Dollars','Tot_Dollars');
158 for wb=1:size(CFG,2)
159    fprintf(fileID,' %12s %12s',['Wb' num2str(wb) '_MWh'],['Wb' num2str(wb) '
         ↪ _dollars']);
160 end
161 for wb=1:size(CFG,2)
162    fprintf(fileID,' %12s %12s %15s %15s %15s %15s',...
163       ['Wb' num2str(wb) '_T_AME'],['Wb' num2str(wb) '_DO_AME'],...
164       ['Wb' num2str(wb) '_NN_T_slack'],['Wb' num2str(wb) '_NN_DO_slack'],...
165       ['Wb' num2str(wb) '_W2_T_slack'],['Wb' num2str(wb) '_W2_DO_slack']);
166 end
167 fprintf(fileID,'%12s\r\n','Best_Iter');
168 fclose(fileID);
169
170 while retraining=='Y'
171    iter=iter+1;
172    %Run optimization over planning period
173    tic; optimization_routine; timing=toc;
174    %Run W2 validation check
175    runW2validation;
176    %Plot results and save to files
177    close all; ga_results_plotting_nobanding
178    h = get(0,'children'); h=sort(h);
179    for wb=1:length(h)
180       str=['results/' datestr(clock,'yyyy-mm-dd-HHMM') '_iter' num2str(iter) '
            ↪ _wb' num2str(wb) '_' num2str(round(y_dollars_total(2)))];
181       savefig(h(wb),str)
182    end
183    %Print to results log file
184    fileID=fopen(config.LogFile,'a');
185    results.dollars(iter)=y_dollars_total(2);
186    fprintf(fileID,'%12.0f %12.0f %12.0f %12.0f %12.0f %12.0f %12.0f',...
187       iter,function_evals,timing,y_MWh_total(1),y_MWh_total(2),...
188       y_dollars_total(1),y_dollars_total(2));
189    for wb=1:size(CFG,2)
190       fprintf(fileID,' %12.0f %12.0f',y_MWh(wb,2),y_dollars(wb,2));
191    end
192    for wb=1:size(CFG,2)
193       results.AME(iter,wb*2-1:wb*2)=[AME{wb}.T,AME{wb}.DO];
194       results.slacks(iter,wb*2-1:wb*2)=[slacks{wb}.T.W2,slacks{wb}.DO.W2];
195       fprintf(fileID,' %12.3f %12.3f %15.3f %15.3f %15.3f %15.3f',...
196          AME{wb}.T,AME{wb}.DO,slacks{wb}.T.NN,slacks{wb}.DO.NN,...
197          slacks{wb}.T.W2,slacks{wb}.DO.W2);
198    end
199    clearvars slacks ans data_start objfuncvalues Output_no0s Outputprev h wb Ax1
         ↪  Ax2 Ax3 H h1 h2 h3 h5 h6 h7 legend1 output nVar maxdelay wb xlims
         ↪ xrange ylims yrange
200
201    %Determine if termination criteria is reached
202    if any(results.AME(iter,:)>0.5)
203       if isempty(best_iter)
204          best_iter(iter)=nan;
205       else
206          best_iter(iter)=best_iter(iter-1);
207       end
208    else
209       if isempty(best_iter) | isnan(best_iter(iter-1))
210          best_iter(iter)=iter;
```

```
        else
            if all((results.slacks(iter,:)-results.slacks(best_iter(iter-1),:))<=0)
                best_iter(iter)=iter;
            else
                best_iter(iter)=best_iter(iter-1);
            end
        end
    end
    fprintf(fileID,'%12.0f\r\n',best_iter(iter));
    fclose(fileID);
    if size(best_iter,2)>=2
        if best_iter(iter)==best_iter(iter-1)
            retraining='N';
        end
    end

    %Ask for user's input on how well the NARX predictions look and if they need to retrain the models
    if retraining=='Y'
        for wb=1:size(CFG,2)
            fprintf(['NARX_RETRAIN: Retraining NARX models for waterbody ' num2str(wb) '.\n']);
            NARX_retrain;
        end
    else
        str=['results/' datestr(clock,'yyyy-mm-dd-HHMM') '_iter' num2str(iter) '_' num2str(round(y_dollars_total(2)))];
        save(str)
        clearvars str
    end
end
L.info('OPTIMIZATION','Optimization over operating period complete.')
cumulative_discharge_plot;
```

optimization_routine.m

```
%% Optimize over days_forward

day=1;
if ~exist('plot_data','dir')
    mkdir('plot_data');
end
clearvars xprev tprev
for wb=1:size(CFG,2)
    x_final{wb}=[];
    %Previous turbine pattern for the year (if supplied)
    if isempty(CFG{wb}.ForecastTurbinePattern)
        xprev{wb}=actual_turb_ops(tprev_round,Qprojected{wb},elevtemp{wb},turbine_discharge{wb},no_of_units{wb});
    else
        prevturbpattern=dlmread(CFG{wb}.PreviousTurbinePattern,'\t',1,0);
        for i=1:size(tprev_round,2)
            index=find(prevturbpattern(:,1)<=tprev_round(i));
            xprev{wb}(i)=prevturbpattern(index(end),2);
        end
        clearvars i prevturbpattern index
    end
end
clearvars wb
tprev=[t_all(1)-max(cell2mat(zero_gen_limit(:)))*frequency:frequency:t_all(1)];
xprev_ic=xprev; tprev_ic=tprev;

while day<=days_forward
```

```
27
28    %For each day, determine if elevation, DO , and temp constraints are even
          ↪ feasible (in priority order). If not found feasible, then bounds
          ↪ defined earlier by the config files are modified. Then problem is
          ↪ optimized for maximize power (or power value)
29
30    L.info('OPTIMIZATION', ['OPTIMIZING DAY ', num2str(day)]);
31
32    WQ_subproblem{day}=WQ;
33    ELWS_limit_subproblem{day}=ELWS_limit;
34
35    %Optimization timeperiod
36    t=[start_date+day-1:frequency:start_date+day];
37
38    %Set initial condition elevation
39    for wb=1:size(CFG,2)
40       if day==1
41          ic_elev{wb}=ic_elev_first{wb};
42          if ic_elev_first{wb}<ELWS_limit_subproblem{day}{wb}(1)
43             L.warn('INITIALIZATION', ['Reservoir ', num2str(wb),' initial
                   ↪ elevation of ' cell2mat(ic_elev_first{wb}) ' m is less than
                   ↪ ELWS lower limit (firm constraint). Expanding ELWS limits to
                   ↪ continue with optimization.']);
44             ELWS_limit_subproblem{day}{wb}(1)=ic_elev_first{wb};
45          elseif ic_elev_first{wb}>ELWS_limit_subproblem{day}{wb}(2)
46             L.warn('INITIALIZATION', ['Reservoir ', num2str(wb),' initial
                   ↪ elevation of ' cell2mat(ic_elev_first{wb}) ' m is greater
                   ↪ than ELWS upper limit (firm constraint). Expanding ELWS
                   ↪ limits to continue with optimization.']);
47             ELWS_limit_subproblem{day}{wb}(2)=ic_elev_first{wb};
48          end
49       else
50          ic_elev{wb}=HWs{wb}(end);
51       end
52    end
53
54    for wb=1:size(CFG,2)
55       %Determine x0, actual turbine operations, to seed initial population
56       x0(wb,:)=x0_all(wb,(day-1)*(1/frequency)+1:day*(1/frequency));
57       [~, y_dollars1]=power_value(x0(wb,:),t,cost_curve_MW{wb},...
58          MW_rating{wb});
59       elev_soft_penalty_coeff{day}(wb)=interp1(ELWS_limit_subproblem{day}{wb}(:)
             ↪ ,...
60          elev_soft_penalty_coeff_constant,...
61          interp1(ELWS_targets{wb}(:,1),ELWS_targets{wb}(:,2),start_date+day),...
62          'linear','extrap')*y_dollars1; %$/m with cost curve, MWh/m with all cc
                ↪ =1
63       clearvars y_dollars1
64
65       %Find possible values for x(1) (based on previous zero_gen_limit turbs)
66       options=[0:no_of_units{wb}];
67       % (1) Eliminate options based on change in active unit violations
68       if ~isnan(max_hrly_unit_change{wb})
69          auvoptions=[xprev{wb}(end)-max_hrly_unit_change{wb}:...
70             xprev{wb}(end)+max_hrly_unit_change{wb}];
71          options=intersect(options,auvoptions);
72       end
73       % (2) Non-integer constraint (assumed in selection algorithm)
74       % (3) Eliminate options based on zero generation hourly limit
75       if ~isnan(zero_gen_limit{wb})
76          if sum(xprev{wb}(end-zero_gen_limit{wb}+1:end))==0
77             zghloptions=[1:no_of_units{wb}]; %if previous zero_gen_limit hrs had
                   ↪  zero total flow, must have flow next hr
78             options=intersect(options,zghloptions);
```

```
79          end
80       end
81       % (4) Eliminate options that violate oscillations constraint - violates
             ↪ whenever the number of turbines increases and then decreases within
             ↪  2 hours, or vice versa
82       allopt=[0:no_of_units{wb}];
83       if xprev{wb}(end-1)<xprev{wb}(end) %if prev turbs increasing
84          oscoptions=allopt(allopt>=xprev{wb}(end));
85          options=intersect(options,oscoptions);
86       elseif xprev{wb}(end-1)==xprev{wb}(end) %need 3 hrs btwn ramping up and
             ↪ down
87          if xprev{wb}(end-2)<xprev{wb}(end-1) %ramping up
88             oscoptions=allopt(allopt>=xprev{wb}(end));
89             options=intersect(options,oscoptions);
90          elseif xprev{wb}(end-2)>xprev{wb}(end-1) %ramping down
91             oscoptions=allopt(allopt<=xprev{wb}(end));
92             options=intersect(options,oscoptions);
93          elseif xprev{wb}(end-2)==xprev{wb}(end-1)
94             %do nothing -->3 consecutive hours between ramping up and down
                   ↪ satisfied
95          end
96       elseif xprev{wb}(end-1)>xprev{wb}(end) %if prev turbs decreasing
97          oscoptions=allopt(allopt<=xprev{wb}(end));
98          options=intersect(options,oscoptions);
99       end
100      x1_options{wb}=options;
101      if isempty(x1_options{wb})
102         L.fatal('OPTIMIZATION','Based on previous turbine pattern, there is no
               ↪ feasible first hour turbine level.');
103         return
104      end
105      clearvars tprev options auvoptions zghloptions allopt oscoptions
106   end
107   clearvars wb
108
109   %Determine if elevation, DO, and temp constraints are feasible (based on
         ↪ ranking order) and adjust bounds in this order if necessary
110   L.info('OPTIMIZATION','Check constraint feasibilities and adjust if needed.')
         ↪ ;
111   feasible_option1=[];
112   [WQ_subproblem{day},ELWS_limit_subproblem{day},funccount(day,1),...
113      feasible_option1]=check_feasibilities(ranking,x1_options,...
114      feasiblilitycheck_ga_pop_size,frequency,Q,ic_elev,...
115      no_of_units,t,max_hrly_unit_change,zero_gen_limit,...
116      turbine_discharge,ELWS_limit,WQ,xprev,ELWS_targets,...
117      elev_constraint_rounding,wq_constraint_rounding,tolerance);
118   if ~isempty(feasible_option1)
119      c=penalty_fcn(feasible_option1,t,frequency,Q,ic_elev,...
120         turbine_discharge,ELWS_limit_subproblem{day},...
121         max_hrly_unit_change,WQ_subproblem{day},zero_gen_limit,xprev,...
122         ELWS_targets,tolerance);
123      funccount(day,1)=funccount(day,1)+size(feasible_option1,1);
124      feasible_option1=feasible_option1(find(all(c<=eps,2)),:);
125      clearvars c
126   end
127
128   %Create initial population that satisfies all constraints
129   L.info('OPTIMIZATION','Finding initial population to seed genetic algorithm.'
         ↪ );
130   [feasible_options,objfuncvalues,~,funccount(day,2)]=...
131      create_feasible_initpop(ga_pop_size,feasible_option1,...
132      x1_options,frequency,Q,ic_elev,MW_rating,no_of_units,t,...
133      max_hrly_unit_change,zero_gen_limit,turbine_discharge,...
134      ELWS_limit_subproblem{day},WQ_subproblem{day},cost_curve_MW,xprev,...
```

```
135         elev_soft_penalty_coeff{day},ELWS_targets,tolerance);
136     if isempty(feasible_options) & isempty(feasible_option1)
137         L.info('OPTIMIZATION','No feasible solutions found during initialization \
                ↪ n');
138         return
139     end
140     [objfuncvalues,b]=sort(objfuncvalues,'descend');
141     feasible_options=feasible_options(b,:);
142     clearvars objfcn feasible_option1 b
143     %Check if x0 is feasible - include it if it is
144     y=penalty_fcn(reshape(x0',1,[]),t,frequency,Q,ic_elev,...
145         turbine_discharge,ELWS_limit_subproblem{day},max_hrly_unit_change,...
146         WQ_subproblem{day},zero_gen_limit,xprev,ELWS_targets,tolerance);
147     best_fvals(day,1)=obj_fcn(reshape(x0',1,[]),t,cost_curve_MW,MW_rating,...
148         elev_soft_penalty_coeff{day},ELWS_targets,...
149         frequency,Q,ic_elev,turbine_discharge);
150     funccount(day,2)=funccount(day,2)+1;
151     %Check to see if any values in x0>no_of_units
152     over_no_of_units=0;
153     for wb=1:size(CFG,2)
154         if any(x0(wb,:)>no_of_units{wb}) over_no_of_units=1; end
155     end
156     if ~all(y==0) | over_no_of_units==1
157         L.info('OPTIMIZATION','x0 is not feasible with respect to previous optimal
                ↪  solution.');
158         best_fvals(day,2)=max(objfuncvalues);
159         %Diversity measurement
160         diversity(day,1)=std(objfuncvalues);
161     else
162         L.info('OPTIMIZATION','x0 is feasible with respect to previous optimal
                ↪ solution.');
163         if size(feasible_options,1)==ga_pop_size*3
164             feasible_options=[reshape(x0',1,[]);feasible_options(1:end-1,:)];
165             objfuncvalues=[best_fvals(day,1); objfuncvalues(1:end-1,:)];
166         else
167             feasible_options=[reshape(x0',1,[]);feasible_options];
168             objfuncvalues=[best_fvals(day,1); objfuncvalues];
169         end
170         best_fvals(day,2)=max(objfuncvalues);
171         %Diversity measurement
172         diversity(day,1)=std(objfuncvalues);
173     end
174     clearvars over_no_of_units
175
176     %GA setup
177     %If feasible_options<GA pop size, fill in a larger matrix with repeating
            ↪ values to create a full initial population
178     if size(feasible_options,1)<ga_pop_size
179         feasible_options=repmat(feasible_options,ceil(ga_pop_size/size(
                ↪ feasible_options,1)),1);
180         feasible_options=feasible_options(1:ga_pop_size,:);
181     else
182         feasible_options=feasible_options(1:ga_pop_size,:);
183     end
184     clearvars y x count
185     %Set optimization algorithm options
186     FitnessFunction = @(x) -obj_fcn(x,t,cost_curve_MW,...
187         MW_rating,elev_soft_penalty_coeff{day},...
188         ELWS_targets,frequency,Q,ic_elev,...
189         turbine_discharge);
190     mycon= @(x) penalty_fcn(x,t,frequency,Q,ic_elev,...
191         turbine_discharge,ELWS_limit_subproblem{day},...
192         max_hrly_unit_change,WQ_subproblem{day},zero_gen_limit,...
193         xprev,ELWS_targets,tolerance);
```

```
194     opt = gaoptimset(...
195        'Display','iter','Vectorized','on','Generations',50, ...
196        'PopulationSize',ga_pop_size,...
197        'EliteCount',ceil(0.05*ga_pop_size),...
198        'InitialPopulation',feasible_options,...
199        'StallGenLimit',2,'TolFun',tolerance,'TolCon',tolerance,...
200        'CrossoverFcn',@crossoversinglepoint,'CrossoverFraction',0.85,...
201        'CreationFcn',@int_pop,'MutationFcn',@int_mutation);
202     nVar = size(CFG,2)*(size(t,2)-1);
203     %Set dv lower and upper bounds, narrowed considering max_hrly_unit_change,
            ↪ for both reservoirs
204     for wb=1:size(CFG,2)
205        lb(wb,:)=0*ones(1,size(t,2)-1); lb(wb,1)=x1_options{wb}(1);
206        for i=2:no_of_units{wb}
207           lb(wb,i)=lb(wb,i-1)-max_hrly_unit_change{wb};
208        end
209        lb(wb,:)=max(0,lb(wb,:));
210        ub(wb,:)=no_of_units{wb}*ones(1,size(t,2)-1);
211        ub(wb,1)=x1_options{wb}(end);
212        for i=2:no_of_units{wb}
213           ub(wb,i)=ub(wb,i-1)+max_hrly_unit_change{wb};
214        end
215        ub(wb,:)=min(no_of_units{wb},ub(wb,:));
216        clearvars i
217     end
218     lb=reshape(lb',1,[]); ub=reshape(ub',1,[]);
219
220     %Run GA
221     L.info('OPTIMIZATION','Begin running genetic algorithm.');
222     [x,fval,~,output,~,~]=ga(FitnessFunction,nVar,[],[],[],[],lb,ub,...
223        mycon,[],opt);
224     funccount(day,3)=output.funccount*2; %multiply by 2 to cover penalty & obj
            ↪ functions
225     best_fvals(day,3)=-fval;
226
227     %Split up rows of x to separate reservoirs
228     for wb=1:size(CFG,2)
229        x_final{wb}=[x_final{wb} ...
230           x(:,wb*(size(t,2)-1)-(size(t,2)-2):wb*(size(t,2)-1))];
231     end
232     clearvars wb fval x lb ub FitnessFunction opt mycon feasible_options
233
234     %Update elevations and discharges/inflows in Q before going on to next day
235     Q=updateQ(Q,CFG,x_final,t,frequency,ic_elev,turbine_discharge,...
236        WQ_subproblem{day},ELWS_targets);
237     %Generate csv data files for plotting
238     if day~=days_forward
239        for wb=1:size(CFG,2)
240           [~,~,HWs{wb},~,~] = ...
241              activeunits_to_discharges(x_final{wb},...
242              t_all(1:1+day*(1/frequency)),frequency,Q{wb},...
243              ic_elev_first{wb},turbine_discharge{wb},ELWS_targets{wb},...
244              [],[]);
245           %don't need to supply mainstem_inflows because it's already been
                  ↪ updated in Q{wb}
246        end
247        day=day+1;
248        for wb=1:size(CFG,2)
249           xprev{wb}=[xprev_ic{wb} x_final{wb}];
250        end
251     else
252        day=day+1;
253     end
254  end
```

```
day=day-1;

%Sum funccount
function_evals=sum(sum(funccount));
clearvars funccount

%Compute total y_dollars
clearvars elev_soft_penalty_coeff
for wb=1:size(CFG,2)
   [y_MWh(wb,1), y_dollars(wb,1)]=power_value(x0_all(wb,1:day*(1/frequency)),
       ↪ t_all(1:1+day*(1/frequency)),cost_curve_MW{wb},...
      MW_rating{wb});
   elev_soft_penalty_coeff{wb}=interp1(ELWS_limit{wb}(:)',...
      elev_soft_penalty_coeff_constant,ELWS_targets{wb}(day),...
      'linear','extrap')*y_dollars(wb,1); %$/m with cost curve, MWh/m with all
          ↪ cc=1
   [y_MWh(wb,2), y_dollars(wb,2)]=power_value(x_final{wb},t_all(1:1+day*(1/
       ↪ frequency)),cost_curve_MW{wb},...
      MW_rating{wb});
end
y_MWh_total=sum(y_MWh(1:size(CFG,2),:),1);
y_dollars_total=sum(y_dollars(1:size(CFG,2),:),1);

%Compute average WQ constraint violation for each wb
for wb=1:size(CFG,2)
   slacks{wb}.DO.NN=[]; slacks{wb}.T.NN=[];
   for i=1:size(WQ_subproblem,2)
   slacks{wb}.DO.NN(i)=WQ_subproblem{i}{wb}.DO_slack;
   slacks{wb}.T.NN(i)=WQ_subproblem{i}{wb}.Temp_slack;
   end
   slacks{wb}.DO.NN=mean(slacks{wb}.DO.NN);
   slacks{wb}.T.NN=mean(slacks{wb}.T.NN);
end
clearvars wb i
```

activeunits_to_discharges.m

```
function [turb_discharges,spill_discharges,HWs,TWs,Storage] = ...
   activeunits_to_discharges(x,t,frequency,Q,ic_elev,...
   turbine_discharge,ELWS_targets,mainstem_inflows_t,mainstem_inflows_Q)

% Calculates discharges and HWs and TWs from time series of number of
% active units
%
% Inputs:
% x - hourly turbine time series (as rows for vectorizing!), integers
% between 0 and no_of_units
% t  time series of JDAY values
% frequency - frequency of predictions (hourly=1/24)
% Q - all other inflows and outflows, interpolation settings,
% storage-elev curve, and tailwater curve (all in meters)
% ic_elev - initial condition (meters)
% turbine_discharge - turbine discharge curve at fixed MW level, with
% col 1 in meters and col 2 in cms
% ELWS_targets - 2 column matrix with JDAY in col1 and elevation target
% in col2
% mainstem_inflows_t - vector of JDAY values that correspond to
% mainstem_inflows_Q
% mainstem_inflows_Q - if applicable (wb~=1), rows of incoming flows from
% upstream reservoir correlated to times in mainstem_inflows_t
% Outputs:
% turb_discharges  turbine discharge time series in cms
% spill_discharges - spill discharge in cms
```

```
% HWs - headwater time series in m
% TWs - tailwater time series in m
% Storage - storage time series in cubic meters

JDAY_initial=t(1);

%Number of x scenarios being tested
n=size(x,1);

if n<1
   fprintf('Active units to discharges code --> x is empty!')
   return
end

%Initial condition
clearvars HWs Storage turb_discharges TWs
HWs(1,1:n)=ic_elev;
Storage(1:n,1)=interp1(Q.SE_meters_m3(:,1),Q.SE_meters_m3(:,2),HWs(1,1));
index1=find(Q.QOT_BR1_T(:,1)<=JDAY_initial);
index2=find(Q.QOT_BR1_S(:,1)<=JDAY_initial);
turb_discharges(1:n,1)=Q.QOT_BR1_T(index1(end),2);
tot_discharge=Q.QOT_BR1_T(index1(end),2)+Q.QOT_BR1_S(index2(end),2);
TWs(1:n,1)=interp1(Q.tw_curve_cms_m(:,1),Q.tw_curve_cms_m(:,2), ...
   tot_discharge);
clearvars index1 index2 tot_discharge

%Compute discharge (cms) per unit at first timestep using prev hr HW and TW
head=HWs(1,:)'-TWs(:,1);
unit_discharges=interp1(turbine_discharge(:,1),turbine_discharge(:,2), ...
   head);
unit_discharges(head>=turbine_discharge(end,1))=turbine_discharge(end,2);
unit_discharges(head<=turbine_discharge(1,1))=turbine_discharge(1,2);
turb_discharges(1:n,2)=unit_discharges.*x(:,1);
clearvars head unit_discharges

%Compute HW elevs for every scenario
for i=2:size(t,2)
   elevation=HWs(i-1,:);
   turbs=turb_discharges(:,i-1:i);
   if isempty(ELWS_targets) %If testing projected operations
      HWs(i-1:i,:)=Elevation_massbalance_vectorized(turbs,[],...
         t(i-1),t(i),frequency,Q,elevation,mainstem_inflows_t,...
         mainstem_inflows_Q);
   else %If testing new operations, assuming no spill flow here
      HWs(i-1:i,:)=Elevation_massbalance_vectorized(turbs,...
         zeros(size(turbs)),t(i-1),t(i),frequency,Q,elevation,...
         mainstem_inflows_t,mainstem_inflows_Q);
   end
   clearvars elevation turbs
   %Compute storage and TWs
   %If too full and overtops SE curve (or drains and empties), linearly
      ↪ extrapolate
   Storage(:,i)=interp1(Q.SE_meters_m3(:,1),Q.SE_meters_m3(:,2),...
      HWs(i,:)','linear','extrap');
   if isempty(ELWS_targets) %if testing projected operations
      index2=find(Q.QOT_BR1_S(:,1)<=t(i));
      tot_discharge=turb_discharges(:,i)+Q.QOT_BR1_S(index2(end),2);
      clearvars index2
   else %if testing new operations, assuming no spill flow here
      tot_discharge=turb_discharges(:,i)+0; %assume no spill
   end
   TWs(:,i)=interp1(Q.tw_curve_cms_m(:,1),Q.tw_curve_cms_m(:,2), ...
      tot_discharge,'linear','extrap');
   clearvars tot_discharge
```

```
90     %Compute total turbine flowrate
91     if i~=size(t,2)
92        head=HWs(i,:)'-TWs(:,i);
93        %Compute turbine flow based on head, with catches at bounds of turbine
              ↪ discharge curve
94        unit_discharges=interp1(turbine_discharge(:,1), ...
95           turbine_discharge(:,2),head);
96        unit_discharges(head>=turbine_discharge(end,1))=...
97           turbine_discharge(end,2);
98        unit_discharges(head<=turbine_discharge(1,1))=...
99           turbine_discharge(1,2);
100       turb_discharges(:,i+1)=unit_discharges.*x(:,i);
101       clearvars head unit_discharges
102    end
103 end
104 clearvars i ii
105
106 %If testing new operations (i.e. ELWS_targets is not empty), continue on and
        ↪ compute spill
107 if ~isempty(ELWS_targets)
108    %Check for cases when the final HW elev is greater than target
109    ELWS_goal=interp1(ELWS_targets(:,1),ELWS_targets(:,2),t(end));
110    volume_to_spill=max(0,...
111       interp1(Q.SE_meters_m3(:,1),Q.SE_meters_m3(:,2),HWs(end,:))...
112       -interp1(Q.SE_meters_m3(:,1),Q.SE_meters_m3(:,2),ELWS_goal));
113    spill_discharges=0.95*((volume_to_spill/((t(end)-t(1))*24*60*60)))';
114
115    %Compute HWs again for situations with spill added to lower to ELWS target
116    [a,~]=find(spill_discharges~=0);
117    if ~isempty(a)
118       stop=0;
119       while stop==0
120          for i=2:size(t,2)
121             elevation=HWs(i-1,a);
122             turbs=turb_discharges(a,i-1:i);
123             if isempty(mainstem_inflows_Q)
124                HWs(i-1:i,a)=Elevation_massbalance_vectorized(turbs,...
125                   [spill_discharges(a) spill_discharges(a)],...
126                   t(i-1),t(i),frequency,Q,elevation,mainstem_inflows_t,...
127                   mainstem_inflows_Q);
128             else
129                HWs(i-1:i,a)=Elevation_massbalance_vectorized(turbs,...
130                   [spill_discharges(a) spill_discharges(a)],...
131                   t(i-1),t(i),frequency,Q,elevation,mainstem_inflows_t,...
132                   mainstem_inflows_Q(a,:));
133             end
134             clearvars elevation turbs
135             %Compute storage and TWs
136             Storage(a,i)=interp1(Q.SE_meters_m3(:,1),Q.SE_meters_m3(:,2),...
137                HWs(i,a)');
138             tot_discharge=turb_discharges(a,i)+spill_discharges(a); %now assume
                    ↪ we have the spill we calculated above
139             TWs(a,i)=interp1(Q.tw_curve_cms_m(:,1),Q.tw_curve_cms_m(:,2), ...
140                tot_discharge);
141             clearvars tot_discharge
142             %Compute total turbine flowrate
143             if i~=size(t,2)
144                head=HWs(i,a)'-TWs(a,i);
145                %Compute turbine flow based on head, with catches at bounds of
                       ↪ turbine discharge curve
146                unit_discharges=interp1(turbine_discharge(:,1), ...
147                   turbine_discharge(:,2),head);
148                unit_discharges(head>=turbine_discharge(end,1))=...
149                   turbine_discharge(end,2);
```

```
150                 unit_discharges(head<=turbine_discharge(1,1))=...
151                    turbine_discharge(1,2);
152                 turb_discharges(a,i+1)=unit_discharges.*x(a,i);
153                 clearvars head unit_discharges
154              end
155           end
156           %Check end elevations again and adjust spill and iterate (if necessary)
157           volume_to_spill=max(0,...
158              interp1(Q.SE_meters_m3(:,1),Q.SE_meters_m3(:,2),HWs(end,:))...
159              -interp1(Q.SE_meters_m3(:,1),Q.SE_meters_m3(:,2),ELWS_goal));
160           spill_discharges2=spill_discharges+0.95*((volume_to_spill/((t(end)-t(1)
                 ↪ )*24*60*60)))';
161           diffspill=spill_discharges2-spill_discharges;
162           if all(round(diffspill,3)==0)
163              stop=1;
164           end
165           spill_discharges=spill_discharges2; clearvars spill_discharges2
166        end
167        clearvars i ii stop diffspill
168        %Recompute HWs and TWs with final spillrate
169        for i=2:size(t,2)
170           elevation=HWs(i-1,a);
171           turbs=turb_discharges(a,i-1:i);
172           if isempty(mainstem_inflows_Q)
173                 HWs(i-1:i,a)=Elevation_massbalance_vectorized(turbs,...
174                    [spill_discharges(a) spill_discharges(a)],...
175                    t(i-1),t(i),frequency,Q,elevation,mainstem_inflows_t,...
176                    mainstem_inflows_Q);
177              else
178                 HWs(i-1:i,a)=Elevation_massbalance_vectorized(turbs,...
179                    [spill_discharges(a) spill_discharges(a)],...
180                    t(i-1),t(i),frequency,Q,elevation,mainstem_inflows_t,...
181                    mainstem_inflows_Q(a,:));
182              end
183           clearvars elevation turbs
184           %Compute storage and TWs
185           %If too full and overtops SE curve (or drains and empties), linearly
                 ↪ extrapolate
186           Storage(a,i)=interp1(Q.SE_meters_m3(:,1),Q.SE_meters_m3(:,2),...
187              HWs(i,a)',’linear’,’extrap’);
188           tot_discharge=turb_discharges(a,i)+spill_discharges(a); %now assume we
                 ↪ have the spill we calculated above
189           TWs(a,i)=interp1(Q.tw_curve_cms_m(:,1),Q.tw_curve_cms_m(:,2), ...
190              tot_discharge);
191           clearvars tot_discharge
192           %Compute total turbine flowrate
193           if i~=size(t,2)
194              head=HWs(i,a)'-TWs(a,i);
195              %Compute turbine flow based on head, with catches at bounds of
                    ↪ turbine discharge curve
196              unit_discharges=interp1(turbine_discharge(:,1), ...
197                 turbine_discharge(:,2),head);
198              unit_discharges(head>=turbine_discharge(end,1))=...
199                 turbine_discharge(end,2);
200              unit_discharges(head<=turbine_discharge(1,1))=...
201                 turbine_discharge(1,2);
202              turb_discharges(a,i+1)=unit_discharges.*x(a,i);
203              clearvars head unit_discharges
204           end
205        end
206        clearvars i ii
207     end
208  else
209     spill_discharges=zeros(n,1);
```

```
end


HWs=HWs';%change back to rows to match all the other outputs (computed as cols
    ↪ to make vectorizing Elevation_massbalance_vectorized easier)
```

buildQ.m

```
function Q=buildQ(directory)

% Builds the variable Q, used for the water balance
%
% Inputs:
% directory - directory of csv files needed to build Q
% Outputs:
% Q - all other inflows and outflows, interpolation settings,
% storage-elev curve, and tailwater curve (all in meters)

clearvars Q

%Load in interpolation file (can't use csvread due to strings)
C=importdata(strcat(directory, 'interpolation.csv'),',');
for i=1:size(C,1)
   Q.interpolation(i,:) = strsplit(C{i,1},',');
end
clearvars i C

%Load in data files from optimization directory folder
d=dir(strcat(directory, '*.csv'));
for i=1:length(d)
   if ~strcmp(d(i).name,'interpolation.csv') & d(i).bytes~=0
      Dstr_max_structure(i).name=d(i).name;
      Dstr_max_structure(i).matrix=csvread(strcat(directory, d(i).name));
      [~,name,~] = fileparts(Dstr_max_structure(i).name);
      %Make sure that each matrix has 2 rows (avoid interpolation errors)
      if size(Dstr_max_structure(i).matrix,1)<2
         Dstr_max_structure(i).matrix(end+1,1)=366;
         Dstr_max_structure(i).matrix(end,2)=...
            Dstr_max_structure(i).matrix(1,2);
      end
      Q.(sprintf(name))=Dstr_max_structure(i).matrix;
   end
end
clearvars d i name Dstr_max_structure
```

check_feasibilities.m

```
function [WQ_adjusted,ELWS_limit_adjusted,funccount,feasible_options]=
    ↪ check_feasibilities(ranking,...
   x1_options,ga_pop_size,frequency,Q,ic_elev,no_of_units,t,max_hrly_unit_change
       ↪ ,...
   zero_gen_limit,turbine_discharge,ELWS_limit,WQ,xprev,ELWS_targets,...
   elev_constraint_rounding,wq_constraint_rounding,tolerance)

% Checks the feasibility of constraints (elev, do, temp) in the priority
% order defined by the user, and adjusting constraints as necessary
%
% Inputs:
% ranking - assign priority ranking for constraints on elev, DO, and temp,
    ↪ starting
% with highest priority first
% x1_options - options for the turbine setting at the first hour
```

```
% ga_pop_size - population size
% frequency - frequency of predictions (hourly=1/24)
% Q - all other inflows and outflows, interpolation settings,
% storage-elev curve, and tailwater curve
% ic_elev - initial condition (meters)
% no_of_units - max number of turbines (4 for OHL)
% t  time series of JDAY values
% max_hrly_unit_change - max number of units that can be changed per hour
% (1 for OHL)
% zero_gen_limit - Zero generation hourly limit (can't go longer than
% this with no turb flow)
% turbine_discharge - turbine discharge curve at fixed MW level, with
% col 1 in meters and col 2 in cms
% ELWS_limit - min and max elevation limits for constraints, in meters
% WQ - structure containing water quality constraints and NARX models
% DO_narx - structure containing everything needed to make DO discharge
% predictions, including:
% turb_colum - column in exogenous variables with turb flows
% spill_column - column in exogenous variables with spill flows
% times - JDAY values used in training (not used)
% inputDelays - delays for exogenous inputs
% feedbackDelays - delays for prediction feedbacks
% input_variables - 2 row cell containing variable names in first
% row and column number in second. For example, 'MET_WB1'
% contains multiple columns of data but only some may be used
% for NARX predictions
% bias - bias for each trained neural network
% weights - weights for each trained neural network (sum to 1)
% narx_net_closed - neural networks
% DO_limit - lower and upper DO limits (NaN means it doesn't exist)
% DO_slack - relaxation from DO_limit (either upper or lower -
% doesn't make sense to have both)
% Temp_narx - structure containing everything needed to make temp discharge
% predictions, including:
% turb_colum - column in exogenous variables with turb flows
% spill_column - column in exogenous variables with spill flows
% times - JDAY values used in training (not used)
% inputDelays - delays for exogenous inputs
% feedbackDelays - delays for prediction feedbacks
% input_variables - 2 row cell containing variable names in first
% row and column number in second. For example, 'MET_WB1'
% contains multiple columns of data but only some may be used
% for NARX predictions
% bias - bias for each trained neural network
% weights - weights for each trained neural network (sum to 1)
% narx_net_closed - neural networks
% Temp_limit - lower and upper temp limits (NaN means it doesn't exist)
% Temp_slack - relaxation from Temp_limit (either upper or lower -
% doesn't make sense to have both)
% xprev - vector of previous active turbine levels
% ELWS_targets - 2 column matrix with JDAY in col1 and elevation target
% in col2
% elev_constraint_rounding - rounding setting (10=tenths place,
% 100=hundredths place, etc.)
% wq_constraint_rounding - rounding setting (10=tenths place,
% 100=hundredths place, etc.)
% tolerance - penalty tolerance
% Outputs:
% WQ_adjusted  updated WQ structure (same structure as WQ, with updated
% constraints if necessary)
% ELWS_limit_adjusted - updated elevation limits (if necessary)
% funccount - total number of function evaluations (both obj and penalty)
% feasible_options - save any solutions that are totally feasible to feed
% into initial population creation function next
```

```
77
78  funccount=0; generations=0;
79  exitflag=[];
80
81  %% Create 500 potential solutions feasible wrt constraints #1-3
82
83  %Weights
84  for wb=1:size(x1_options,2)
85  for i=2:no_of_units{wb}+1
86     weights{wb}{i}(1)=no_of_units{wb};
87     for ii=2:i
88        weights{wb}{i}(ii)=weights{wb}{i}(ii-1)*.1;
89     end
90  end
91  end
92  clearvars i ii wb
93
94  %First, generate a few solutions quickly and test feasibility. If any are
        ↪ feasible, terminate this function with changes to WQ or elevation
        ↪ constraints
95  setsize=[10 2*ga_pop_size];
96  for z=1:size(setsize,2)
97     for wb=1:size(x1_options,2)
98        raw_options{wb}{z}=nan(setsize(z),size(t,2)-1);
99        if size(x1_options{wb},2)==1 %only 1 option left
100          raw_options{wb}{z}(:,1)=x1_options{wb};
101       else
102          if z==1
103             raw_options{wb}{z}(:,1)=randsample(x1_options{wb},setsize(z),true);
104          elseif z==2
105             raw_options{wb}{z}(:,1)=randsample(x1_options{wb},setsize(z),true,
                   ↪ weights{wb}{size(x1_options{wb},2)});
106          end
107       end
108       for i=1:size(raw_options{wb}{z},1)
109          for j=2:size(t,2)-1
110             %Variable consisting of xprev and turbine pattern through j-1
111             pattern=[xprev{wb} raw_options{wb}{z}(i,1:j-1)];
112             %First start with all available options, then eliminate infeasible
                   ↪ ones based on turbines from 1:j-1
113             options=[0:no_of_units{wb}];
114             % (1) Eliminate options based on change in active unit violations
115             if ~isnan(max_hrly_unit_change{wb})
116                auvoptions=[pattern(end)-max_hrly_unit_change{wb}: ...
117                   pattern(end)+max_hrly_unit_change{wb}];
118                options=intersect(options,auvoptions);
119             end
120             % (2) Non-integer constraint (assumed in selection algorithm)
121             % (3) Eliminate options based on zero generation hourly limit
122             if ~isnan(zero_gen_limit{wb})
123                if sum(pattern(end-zero_gen_limit{wb}+1:end))==0
124                   zghloptions=[1:no_of_units{wb}]; %if previous zero_gen_limit
                         ↪ hrs had zero total flow, must have flow next hr
125                   options=intersect(options,zghloptions);
126                end
127             end
128             % (4) Eliminate options that violate oscillations constraint -
                   ↪ violates whenever the number of turbines increases and then
                   ↪ decreases within 3 hours, or vice versa
129             allopt=[0:no_of_units{wb}];
130             if pattern(end-1)<pattern(end) %if prev turbs increasing
131                osoptions=allopt(allopt>=pattern(end));
132                options=intersect(options,osoptions);
133             elseif pattern(end-1)==pattern(end) %need 3 hrs btwn ramping up and
```

```
                ↪ down
134             if pattern(end-2)<pattern(end-1) %ramping up
135                oscoptions=allopt(allopt>=pattern(end));
136                options=intersect(options,oscoptions);
137             elseif pattern(end-2)>pattern(end-1) %ramping down
138                oscoptions=allopt(allopt<=pattern(end));
139                options=intersect(options,oscoptions);
140             elseif pattern(end-2)==pattern(end-1)
141                %do nothing -->3 consecutive hours between ramping up and down
                       ↪ satisfied
142             end
143          elseif pattern(end-1)>pattern(end) %if prev turbs decreasing
144             oscoptions=allopt(allopt<=pattern(end));
145             options=intersect(options,oscoptions);
146          end
147          %Out of the available options left, pick the next turbine setting
148          if size(options,2)==1 %only 1 option left
149             raw_options{wb}{z}(i,j)=options;
150          else
151             if z==1
152                raw_options{wb}{z}(i,j)=randsample(options,1,true);
153             elseif z==2
154                raw_options{wb}{z}(i,j)=randsample(options,1,true,weights{wb}{
                       ↪ size(options,2)});
155             end
156          end
157       end
158    end
159 end
160
161 %Convert raw_options cells to long vectors containing all reservoirs per row
162 raw_options2{z}=[];
163 for wb=1:size(x1_options,2)
164    raw_options2{z}=[raw_options2{z} raw_options{wb}{z}];
165 end
166
167 %Check feasibilities if first small set
168 if z==1
169    [c,~]=penalty_fcn(raw_options2{z},t,frequency,Q,ic_elev,...
170       turbine_discharge,ELWS_limit,max_hrly_unit_change, ...
171       WQ,zero_gen_limit,xprev,ELWS_targets,tolerance);
172    funccount=funccount+size(raw_options2{z},1);
173    feasibles=raw_options2{z}(find(all(c<=eps,2)),:);
174    if ~isempty(feasibles)
175       fprintf('All constraints are feasible. \n');
176       WQ_adjusted=WQ; ELWS_limit_adjusted=ELWS_limit;
177       feasible_options=feasibles;
178       return
179    end
180 end
181 end
182 feasible_options2=[];
183 for z=1:size(setsize,2)
184    feasible_options2=[feasible_options2; raw_options2{z}];
185 end
186 feasible_options=feasible_options2; feasible_options_raw=feasible_options;
187 clearvars z i a j feasibles feasible_options2
188
189 %% Optimize each constraint in priority order and terminate at 0. Otherwise,
       ↪ modify the constraint bounds
190
191 for wb=1:size(x1_options,2)
192    ELWS_limit_adjusted{wb}=nan(size(ELWS_limit{wb}));
193    WQ_adjusted{wb}.DO_limit=nan(size(WQ{wb}.DO_limit));
```

```
194     WQ_adjusted{wb}.Temp_limit=nan(size(WQ{wb}.Temp_limit));
195     WQ_adjusted{wb}.DO_narx=WQ{wb}.DO_narx;
196     WQ_adjusted{wb}.Temp_narx=WQ{wb}.Temp_narx;
197     WQ_adjusted{wb}.DO_slack=WQ{wb}.DO_slack;
198     WQ_adjusted{wb}.Temp_slack=WQ{wb}.Temp_slack;
199 end
200 skip=0;
201
202 for wb=1:size(x1_options,2)
203     for i=1:size(ranking,2)
204         if strcmp(ranking{i},'elev') & (~isnan(ELWS_limit{wb}(1)) | ~isnan(
                ↪ ELWS_limit{wb}(2)))
205             fprintf(['Checking reservoir #', num2str(wb),' elevation constraint
                    ↪ feasibility. \n']);
206         elseif strcmp(ranking{i},'do') & (~isnan(WQ{wb}.DO_limit(1)) | ~isnan(WQ{
                ↪ wb}.DO_limit(2)))
207             fprintf(['Checking reservoir #', num2str(wb),' DO constraint
                    ↪ feasibility. \n']);
208         elseif strcmp(ranking{i},'temp') & (~isnan(WQ{wb}.Temp_limit(1)) | ~isnan(
                ↪ WQ{wb}.Temp_limit(2)))
209             fprintf(['Checking reservoir #', num2str(wb),' temperature constraint
                    ↪ feasibility. \n']);
210         end
211
212         %Check lower limit then upper limit. In each step, check maximum violation
                ↪  and then mean value (for temp & DO, not elevation)
213         for a=1:2
214             if a==1 level='lower'; elseif a==2 level='upper'; end
215
216             if strcmp(ranking{i},'elev') & ~isnan(ELWS_limit{wb}(a))
217                 skip=0;
218             elseif strcmp(ranking{i},'do') & ~isnan(WQ{wb}.DO_limit(a))
219                 skip=0;
220             elseif strcmp(ranking{i},'temp') & ~isnan(WQ{wb}.Temp_limit(a))
221                 skip=0;
222             else
223                 skip=1; %if there is no constraint being added here, no need to
                        ↪ check feasibility!
224             end
225
226             if skip==0
227                 clearvars FitnessFunction mycon opt
228
229                 %(1) Test the maximum constraint violation first
230
231                 %Set penalty function first to make sure it doesn't include the
                        ↪ constraint that is being optimized, but all constraints
                        ↪ before that one
232                 mycon= @(x) penalty_fcn(x,t,frequency,Q,ic_elev,...
233                     turbine_discharge,ELWS_limit_adjusted,max_hrly_unit_change,...
234                     WQ_adjusted,zero_gen_limit,xprev,ELWS_targets,tolerance);
235                 %Load in the relevant constraints
236                 if strcmp(ranking{i},'elev')
237                     ELWS_limit_adjusted{wb}(a)=ELWS_limit{wb}(a);
238                 elseif strcmp(ranking{i},'do')
239                     WQ_adjusted{wb}.DO_limit(a)=WQ{wb}.DO_limit(a);
240                     WQ_adjusted{wb}.DO_slack=WQ{wb}.DO_slack;
241                 elseif strcmp(ranking{i},'temp')
242                     WQ_adjusted{wb}.Temp_limit(a)=WQ{wb}.Temp_limit(a);
243                     WQ_adjusted{wb}.Temp_slack=WQ{wb}.Temp_slack;
244                 end
245                 %Set objective function
246                 if strcmp(ranking{i},'elev') & ~isnan(ELWS_limit_adjusted{wb}(a))
247                     FitnessFunction = @(x) obj_fcn_elev(x,t,frequency,Q,ic_elev,...
```

```
248                 turbine_discharge,ELWS_limit_adjusted{wb},ELWS_targets,level,
                        ↪ wb);
249             elseif strcmp(ranking{i},'do') & ~isnan(WQ_adjusted{wb}.DO_limit(a))
250                FitnessFunction = @(x) obj_fcn_do(x,t,frequency,Q,ic_elev,...
251                   turbine_discharge,WQ_adjusted,ELWS_targets,level,wb);
252             elseif strcmp(ranking{i},'temp') & ~isnan(WQ_adjusted{wb}.Temp_limit
                    ↪ (a))
253                FitnessFunction = @(x) obj_fcn_temp(x,t,frequency,Q,ic_elev,...
254                   turbine_discharge,WQ_adjusted,ELWS_targets,level,wb);
255             end
256             %Check feasibility
257             if any(FitnessFunction(feasible_options(1:min(size(feasible_options
                    ↪ ,1),setsize(1)),:))==0)
258                fval=0; funccount=funccount+size(feasible_options,1);
259                pop=feasible_options;
260             else
261                %If feasible_options<GA pop size, fill in a larger matrix with
                       ↪ repeating values to create a full initial population
262                if size(feasible_options,1)<ga_pop_size
263                   feasible_options=repmat(feasible_options,ceil(ga_pop_size/size
                          ↪ (feasible_options,1)),1);
264                   feasible_options=feasible_options(1:ga_pop_size,:);
265                end
266                %GA settings
267                opt = gaoptimset(...
268                   'Display','iter','Vectorized','on','Generations',50, ...
269                   'PopulationSize',ga_pop_size,...
270                   'InitialPopulation',feasible_options(1:ga_pop_size,:),...
271                   'StallGenLimit',1,'TolFun',tolerance,'TolCon',tolerance,...
272                   'CrossoverFcn',@crossoversinglepoint,'CrossoverFraction'
                          ↪ ,0.85,...
273                   'EliteCount',ceil(.05*ga_pop_size),...
274                   'CreationFcn',@int_pop,'MutationFcn',@int_mutation,'
                          ↪ FitnessLimit',0);
275                nVar = size(x1_options,2)*(size(t,2)-1);
276                %Set dv lower and upper bounds, narrowed considering
                       ↪ max_hrly_unit_change
277                clearvars lb ub
278                for wb2=1:size(x1_options,2)
279                   lb(wb2,:)=0*ones(1,size(t,2)-1); lb(wb2,1)=x1_options{wb2}(1);
280                   for ii=2:no_of_units{wb2}
281                      lb(wb2,ii)=lb(wb2,ii-1)-max_hrly_unit_change{wb2};
282                   end
283                   lb(wb2,:)=max(0,lb(wb2,:));
284                   ub(wb2,:)=no_of_units{wb2}*ones(1,size(t,2)-1);
285                   ub(wb2,1)=x1_options{wb2}(end);
286                   for ii=2:no_of_units{wb2}
287                      ub(wb2,ii)=ub(wb2,ii-1)+max_hrly_unit_change{wb2};
288                   end
289                   ub(wb2,:)=min(no_of_units{wb2},ub(wb2,:));
290                   clearvars ii
291                end
292                clearvars wb2
293                lb=reshape(lb',1,[]); ub=reshape(ub',1,[]);
294                %Run GA
295                [~,fval,~,output,pop,~]=ga(FitnessFunction,nVar,[],[],[],[],lb,ub
                       ↪ ,...
296                   mycon,[],opt);
297                funccount=funccount+output.funccount*2; %multiply by 2 to cover
                       ↪ penalty & obj functions
298                generations=output.generations;
299             end
300             %Adjust constraint limits if necessary
301             if fval~=0
```

```
                if level=='lower'
                   plusminus=-1;
                elseif level=='upper'
                   plusminus=1;
                end
                if strcmp(ranking{i},'elev')
                   fprintf(['Adjusting reservoir #', num2str(wb),' ', level, '
                       ↪ elevation constraint. \n']);
                   ELWS_limit_adjusted{wb}(a)=ELWS_limit{wb}(a)...
                      +plusminus*ceil(elev_constraint_rounding*fval)/
                          ↪ elev_constraint_rounding;
                   if ~isempty(pop)
                      pop=[pop; feasible_options_raw]; pop=unique(pop,'rows');
                      c=mycon(pop); pop=pop(all(c<=tolerance,2),:);
                      o=FitnessFunction(pop);
                      feasible_options=pop(find(o==min(o)),:);
                   end
                elseif strcmp(ranking{i},'do')
                   fprintf(['Adjusting reservoir #', num2str(wb),' ', level, ' DO
                       ↪  slack constraint. \n']);
                   WQ_adjusted{wb}.DO_slack(a)=ceil(wq_constraint_rounding*fval)/
                       ↪ wq_constraint_rounding;
                   if ~isempty(pop)
                      pop=[pop; feasible_options_raw]; pop=unique(pop,'rows');
                      c=mycon(pop); pop=pop(all(c<=tolerance,2),:);
                      o=FitnessFunction(pop);
                      feasible_options=pop(find(o==min(o)),:);
                   end
                elseif strcmp(ranking{i},'temp')
                   fprintf(['Adjusting reservoir #', num2str(wb),' ', level, '
                       ↪ temperature slack constraint. \n']);
                   WQ_adjusted{wb}.Temp_slack(a)=ceil(wq_constraint_rounding*fval
                       ↪ )/wq_constraint_rounding;
                   if ~isempty(pop)
                      pop=[pop; feasible_options_raw]; pop=unique(pop,'rows');
                      c=mycon(pop); pop=pop(all(c<=tolerance,2),:);
                      o=FitnessFunction(pop);
                      feasible_options=pop(find(o==min(o)),:);
                   end
                end
             else
                pop=[pop; feasible_options_raw]; pop=unique(pop,'rows','stable');
                c=mycon(pop); pop=pop(all(c<=tolerance,2),:);
                o=FitnessFunction(pop);
                feasible_options=pop(find(o==min(o)),:);
             end
             clearvars plusminus output
          end
       end
    end
end
clearvars i a
WQ_adjusted{wb}.DO_slack=sum(WQ_adjusted{wb}.DO_slack,2);
WQ_adjusted{wb}.Temp_slack=sum(WQ_adjusted{wb}.Temp_slack,2);
```

cost_curve.m

```
function price = cost_curve(t,output_MW,cost_curve_MW)

% Calculates elevation predictions under various turbine outflow conditions
%
% Inputs:
% t  time series of JDAY values
```

```
% output_MW - MW at each timepoint (step function)
% cost_curve_MW  2 row matrix to create step function, with 1st row
% being hours and 2nd row $/MW-hr values
% Outputs:
% price  total price in $ of generation pattern

timepoint_dollars=nan(size(t,2),2);
if size(cost_curve_MW,1)==1
   timepoint_dollars(:,1)=cost_curve_MW(1,2);
   timepoint_dollars(:,2)=1;
else
   for i=1:size(t,2)
      a=round((t(i)-floor(t(i)))*24-cost_curve_MW(1,:));
      a=a(a>=0);
      [c index] = min(a);
      timepoint_dollars(i,1)=cost_curve_MW(2,index);
      timepoint_dollars(i,2)=index;
   end
end

price=nan(size(output_MW));
for i=1:size(t,2)-1
   if timepoint_dollars(i,2)==timepoint_dollars(i+1,2)
      price(:,i)=output_MW(:,i)*timepoint_dollars(i,1)*(t(i+1)-t(i))*24;
   else
      if timepoint_dollars(i+1,2)>=timepoint_dollars(i,2)
         price(:,i)=0;
         for ii=timepoint_dollars(i,2):timepoint_dollars(i+1,2)-1
            price(:,i)=price(:,i)+(cost_curve_MW(1,ii+1)-(t(i)-...
               floor(t(i)))*24)*output_MW(:,i)*cost_curve_MW(2,ii);
         end
         if i+1>size(cost_curve_MW,2)
            price(:,i)=price(:,i)+((t(i+1)-floor(t(i+1)))*24-...
               cost_curve_MW(1,timepoint_dollars(i+1,2)))*...
               output_MW(:,i)*cost_curve_MW(2,1);
         else
            price(:,i)=price(:,i)+((t(i+1)-floor(t(i+1)))*24-...
               cost_curve_MW(1,timepoint_dollars(i+1,2)))*...
               output_MW(:,i)*cost_curve_MW(2,i+1);
         end
      %if we've passed midnight into next day...
      elseif timepoint_dollars(i+1,2)<timepoint_dollars(i,2)
         price(:,i)=0;
         for ii=timepoint_dollars(i,2):size(cost_curve_MW,2)
            price(:,i)=price(:,i)+(24-(t(i)-floor(t(i)))*24)*...
               output_MW(:,i)*cost_curve_MW(2,ii);
         end
         for ii=1:timepoint_dollars(i+1,2)-1
            price(:,i)=price(:,i)+(cost_curve_MW(1,ii+1)-(t(i)-...
               floor(t(i)))*24)*output_MW(:,i)*cost_curve_MW(2,ii);
         end
         if i+1>size(cost_curve_MW,2)
            price(:,i)=price(:,i)+((t(i+1)-floor(t(i+1)))*24-...
               cost_curve_MW(1,timepoint_dollars(i+1,2)))*...
               output_MW(:,i)*cost_curve_MW(2,1);
         else
            price(:,i)=price(:,i)+((t(i+1)-floor(t(i+1)))*24-...
               cost_curve_MW(1,timepoint_dollars(i+1,2)))*...
               output_MW(:,i)*cost_curve_MW(2,i+1);
         end
      end
   end
end

```

```
price=sum(price')';

end
```

create_feasible_initpop.m

```
function [feasible_options,y,c,funccount]=create_feasible_initpop(no_of_solns ...
    feasible_options,x1_options,frequency,Q,ic_elev,MW_rating,no_of_units,t,...
    max_hrly_unit_change,zero_gen_limit,turbine_discharge,ELWS_limit,...
    WQ,cost_curve_MW,xprev,elev_soft_penalty_coeff,...
    ELWS_targets,tolerance)

% Generate and save lots of solutions that are feasible in terms of:
% (1) Change in active unit violations
% (2) Non-integer constraint (assumed in this seletion algorithm)
% (3) Zero generation hourly limit
% (4) Oscillations constraint
%
% Inputs:
% no_of_solns - the number of feasible solutions we want to find
% feasible_options - feasible solutions already found during constraint
% prescreening
% x1_options - feasible options for first value of x, between 0 and
% no_of_units
% frequency - frequency of predictions (hourly=1/24)
% Q - all other inflows and outflows, interpolation settings,
% storage-elev curve, and tailwater curve (all in meters)
% ic_elev - initial elevation condition (m)
% MW_rating - the fixed MW level of turbine_discharge_curve (25 MW for
% OHL)
% no_of_units - max number of available turbine units
% t  time series of JDAY values
% max_hrly_unit_change - max number of units that can be changed per hour
% (1 for OHL)
% zero_gen_limit - Zero generation hourly limit (can't go longer than
% this with no turb flow)
% turbine_discharge - turbine discharge curve at fixed MW level, with
% col 1 in meters and col 2 in cms
% ELWS_limit - min and max elevation limits for constraints, in meters
% WQ - structure containing water quality constraints and NARX models
% DO_narx - structure containing everything needed to make DO discharge
% predictions, including:
% turb_colum - column in exogenous variables with turb flows
% spill_column - column in exogenous variables with spill flows
% times - JDAY values used in training (not used)
% inputDelays - delays for exogenous inputs
% feedbackDelays - delays for prediction feedbacks
% input_variables - 2 row cell containing variable names in first
% row and column number in second. For example, 'MET_WB1'
% contains multiple columns of data but only some may be used
% for NARX predictions
% bias - bias for each trained neural network
% weights - weights for each trained neural network (sum to 1)
% narx_net_closed - neural networks
% DO_limit - lower and upper DO limits (NaN means it doesn't exist)
% DO_slack - relaxation from DO_limit (either upper or lower -
% doesn't make sense to have both)
% Temp_narx - structure containing everything needed to make temp discharge
% predictions, including:
% turb_colum - column in exogenous variables with turb flows
% spill_column - column in exogenous variables with spill flows
% times - JDAY values used in training (not used)
```

```
% inputDelays - delays for exogenous inputs
% feedbackDelays - delays for prediction feedbacks
% input_variables - 2 row cell containing variable names in first
% row and column number in second. For example, 'MET_WB1'
% contains multiple columns of data but only some may be used
% for NARX predictions
% bias - bias for each trained neural network
% weights - weights for each trained neural network (sum to 1)
% narx_net_closed - neural networks
% Temp_limit - lower and upper temp limits (NaN means it doesn't exist)
% Temp_slack - relaxation from Temp_limit (either upper or lower -
% doesn't make sense to have both)
% cost_curve_MW  2 row matrix to create step function, with 1st row
% being hours and 2nd row $/MW-hr values
% xprev - vector of previous active turbine levels
% elev_soft_penalty_coeff - penalty coefficient for soft ending elev soft
% constraint
% ELWS_targets - target elevations for end of time period
% tolerance - penalty tolerance
% Outputs:
% feasible_options  feasible potential solutions for GA initialization
% y - objective function solutions for feasible_options
% c - constraint violations
% funccount - number of paired function evaluations

%Start with upstream reservoir (wb=1), find feasible operations, and
%compute associated discharge flows for each. Then use those flows as
%upstream inflow for next wb, find feasible operations, and compute
%associated discharge flows. Etc...

c=[];
n=size(feasible_options,1);
funccount=0;

count=1;
while size(feasible_options,1)<no_of_solns

   if count==1
      %Starting set size
      setsize=no_of_solns;
   elseif count==2
      %Modify set size as a function of how many feasible solns found so far (
          ↪ maximum is 30*setsize)
      setsize=min(5*(setsize),round((setsize/(size(feasible_options,1)-n))*...
         (no_of_solns-(size(feasible_options,1)-n))));
   else
      %If still not enough solns found, should be close so try 50 at a time
      setsize=50;
   end

   for wb=1:size(x1_options,2)
      raw_options{wb}=nan(setsize,size(t,2)-1);
      if size(x1_options{wb},2)==1 %only 1 option left
         raw_options{wb}(:,1)=x1_options{wb};
      else
         raw_options{wb}(:,1)=randsample(x1_options{wb},setsize,true);
      end
      for i=1:setsize
         for j=2:size(t,2)-1
            %Variable consisting of xprev and turbine pattern through j-1
            pattern=[xprev{wb} raw_options{wb}(i,1:j-1)];
            %First start with all available options, then eliminate infeasible
                ↪ ones based on turbines from 1:j-1
            options=[0:no_of_units{wb}];
```

```
            % (1) Eliminate options based on change in active unit violations
            if ~isnan(max_hrly_unit_change{wb})
               auvoptions=[pattern(end)-max_hrly_unit_change{wb}: ...
                  pattern(end)+max_hrly_unit_change{wb}];
               options=intersect(options,auvoptions);
            end
            % (2) Non-integer constraint (assumed in selection algorithm)
            % (3) Eliminate options based on zero generation hourly limit
            if ~isnan(zero_gen_limit{wb})
               if sum(pattern(end-zero_gen_limit{wb}+1:end))==0
                  zghloptions=[1:no_of_units{wb}]; %if previous zero_gen_limit
                     ↪ hrs had
                  %zero total flow, must have flow next hr
                  options=intersect(options,zghloptions);
               end
            end
            % (4) Eliminate options that violate oscillations constraint -
                ↪ violates whenever the number of turbines increases and then
                ↪ decreases within 3 hours, or vice versa
            allopt=[0:no_of_units{wb}];
            if pattern(end-1)<pattern(end) %if prev turbs increasing
               oscoptions=allopt(allopt>=pattern(end));
               options=intersect(options,oscoptions);
            elseif pattern(end-1)==pattern(end) %need 3 hrs btwn ramping up and
                ↪ down
               if pattern(end-2)<pattern(end-1) %ramping up
                  oscoptions=allopt(allopt>=pattern(end));
                  options=intersect(options,oscoptions);
               elseif pattern(end-2)>pattern(end-1) %ramping down
                  oscoptions=allopt(allopt<=pattern(end));
                  options=intersect(options,oscoptions);
               elseif pattern(end-2)==pattern(end-1)
                  %do nothing -->3 consecutive hours between ramping up and down
                     ↪  satisfied
               end
            elseif pattern(end-1)>pattern(end) %if prev turbs decreasing
               oscoptions=allopt(allopt<=pattern(end));
               options=intersect(options,oscoptions);
            end
            %Out of the available options left, pick the next turbine setting
            if size(options,2)==1 %only 1 option left
               raw_options{wb}(i,j)=options;
            else
               raw_options{wb}(i,j)=randsample(options,1,true);
            end
         end
      end
   end

   %Convert raw_options cells to long vectors containing all reservoirs per row
   raw_options2=[];
   for wb=1:size(x1_options,2)
      raw_options2=[raw_options2 raw_options{wb}];
   end

   %Check feasibility
   [c_new,~]=penalty_fcn(raw_options2,t,frequency,Q,ic_elev,...
      turbine_discharge,ELWS_limit,max_hrly_unit_change,WQ,...
      zero_gen_limit,xprev,ELWS_targets,tolerance);
   funccount=funccount+size(raw_options2,1);
   c=c_new;

   raw_options3=raw_options2(all(c_new<=tolerance,2),:);
   feasible_options=[feasible_options; raw_options3];
```

```
   fprintf(['Feasible options found: ',...
      num2str(size(feasible_options,1)), '\n']);
   if count==2 & isempty(feasible_options)
      y=[]; return
   elseif count==5 & ~isempty(feasible_options)
      y=obj_fcn(feasible_options,t,cost_curve_MW,MW_rating,...
         elev_soft_penalty_coeff,ELWS_targets,frequency,Q,ic_elev,...
         turbine_discharge);
      funccount=funccount+size(feasible_options,1);
      [y,b]=sort(y,'descend');
      feasible_options=feasible_options(b,:);
      return
   else
      count=count+1;
   end
end

%Pick the best no_of_solns from feasible_options
y=obj_fcn(feasible_options,t,cost_curve_MW,MW_rating,...
   elev_soft_penalty_coeff,ELWS_targets,frequency,Q,ic_elev,...
   turbine_discharge);
funccount=funccount+size(feasible_options,1);
[y,b]=sort(y,'descend');
feasible_options=feasible_options(b,:);
```

Elevation_massbalance_vectorized.m

```
function Predictions=Elevation_massbalance_vectorized(turb_discharges, ...
   spill_discharges,JDAY_initial,JDAY_end,frequency,Q,elevation,...
   mainstem_inflows_t,mainstem_inflows_Q)

% Calculates elevation predictions under various turbine outflow conditions
%
% Inputs:
% turb_discharges  turbine discharge time series to test (rows)
% spill_discharges - spill discharge time series
% JDAY_initial  start JDAY (initial condition)
% JDAY_end  end JDAY
% frequency - prediction frequency (ex: 0.25=1/4 day=6 hours)
% Q - all other inflows and outflows, interpolation settings, and
% storage-elev curve
% elevation - initial elevation at JDAY_initial
% mainstem_inflows_t - vector of JDAY values that correspond to
% mainstem_inflows_Q
% mainstem_inflows_Q - if applicable (wb~=1), rows of incoming flows from
% upstream reservoir correlated to times in mainstem_inflows_t
% Outputs:
% Predictions  vector of elevation predictions

n=round((JDAY_end-JDAY_initial)/frequency);
m=size(turb_discharges,1);
Predictions=nan(n+1,m);
Storage=nan(n+1,m);
deltav=nan(n+1,m);

%Initial condition
Predictions(1,:)=elevation;
%If too full and overtops SE curve (or drains and empties), linearly extrapolate
Storage(1,:)=interp1(Q.SE_meters_m3(:,1),Q.SE_meters_m3(:,2),...
   Predictions(1,:),'linear','extrap');

%Run the model
q1=[];
```

```
q2=[];
for time=2:n+1
   volin=0;
   volout=0;
   turbout=0;
   spillout=0;
   volin_BR1=0;
   %Loop through all inflows and outflows (except turbine out)
   for i=1:size(Q.interpolation,2)
      %VOLUMES IN
      %Inflow w/out interpolation
      if (isequal(char(Q.interpolation(2,i)),'inflow') | ...
            isequal(char(Q.interpolation(2,i)),'dist')) & ...
            isequal(char(Q.interpolation(3,i)),'OFF')
         %If the inflow is the mainstem, check if there is data in
            ↪ mainstem_inflows and use that instead of Q
         if isequal(char(Q.interpolation(1,i)),'QIN_BR1') & ...
               ~isempty(mainstem_inflows_Q)
            index1=find(mainstem_inflows_t<=...
               JDAY_initial+(time-2)*frequency,1,'last');
            index2=find(mainstem_inflows_t<=...
               JDAY_initial+(time-1)*frequency,1,'last');
            q1=mainstem_inflows_Q(:,index1);
            if index1==index2
               volin_BR1=volin_BR1+q1*frequency*24*60*60;
            else
               volin_BR1=volin_BR1+q1*...
                  (mainstem_inflows_t(1,index1+1)-(JDAY_initial+...
                  (time-2)*frequency))*24*60*60;
               for ii=index1+1:index2-1
                  q1=mainstem_inflows_Q(:,ii);
                  volin_BR1=volin_BR1+q1*...
                     (mainstem_inflows_t(1,ii+1)-...
                     mainstem_inflows_t(1,ii))*24*60*60;
               end
               q1=mainstem_inflows_Q(:,index2);
               volin_BR1=volin_BR1+q1*...
                  ((JDAY_initial+(time-1)*frequency)-...
                  mainstem_inflows_t(1,index2))*24*60*60;
            end
         else
            flow=Q.(Q.interpolation{1,i});
            index1=find(flow(:,1)<=...
               JDAY_initial+(time-2)*frequency,1,'last');
            index2=find(flow(:,1)<=...
               JDAY_initial+(time-1)*frequency,1,'last');
            q1=flow(index1,2); %flowrate at beginning of timestep
            if index1==index2
               volin=volin+q1*frequency*24*60*60;
            else
               volin=volin+q1*(flow(index1+1,1)-(JDAY_initial+...
                  (time-2)*frequency))*24*60*60;
               for ii=index1+1:index2-1
                  q1=flow(ii,2);
                  volin=volin+q1*(flow(ii+1,1)-flow(ii,1))*24*60*60;
               end
               q1=flow(index2,2);
               volin=volin+q1*((JDAY_initial+(time-1)*frequency)-...
                  flow(index2,1))*24*60*60;
            end
         end
      end
      %Inflow w/ interpolation
      if (isequal(char(Q.interpolation(2,i)),'inflow') | ...
```

```
100             isequal(char(Q.interpolation(2,i)),'dist')) & ...
101             isequal(char(Q.interpolation(3,i)),'ON')
102         %If the inflow is the mainstem, check if there is data in
                ↪ mainstem_inflows and use that instead of Q
103         if isequal(char(Q.interpolation(1,i)),'QIN_BR1') & ...
104                ~isempty(mainstem_inflows_Q)
105             index1=find(mainstem_inflows_t<=...
106                JDAY_initial+(time-2)*frequency,1,'last');
107             index2=find(mainstem_inflows_t<=...
108                JDAY_initial+(time-1)*frequency,1,'last');
109             q1=interp1(mainstem_inflows_t',...
110                mainstem_inflows_Q(:,index1)',JDAY_initial+(time-2)*...
111                frequency);
112             %if JDAY_initial+(time-2)*frequency=timesteps(1), interp1 outputs
                    ↪ nan
113             q1(isnan(q1))=mainstem_inflows_Q(isnan(q1),1);
114             if index1==index2
115                q2=interp1(mainstem_inflows_t',mainstem_inflows_Q',...
116                   JDAY_initial+(time-1)*...
117                   frequency);
118                volin_BR1=volin_BR1+.5*(q1+q2)*...
119                   ((JDAY_initial+(time-1)*frequency)-...
120                   (JDAY_initial+(time-2)*frequency))*24*60*60;
121             else
122                q2=mainstem_inflows_Q(:,index1+1)';
123                volin_BR1=volin_BR1+.5*(q1+q2)*...
124                   (mainstem_inflows_t(1,index1+1)-...
125                   (JDAY_initial+(time-2)*frequency))*24*60*60;
126                for ii=index1+2:index2
127                   q1=q2; %start flowrate is equal to previous end flowrate
128                   q2=mainstem_inflows_Q(:,ii);
129                   volin_BR1=volin_BR1+.5*(q1+q2)*...
130                      (mainstem_inflows_t(1,ii)-...
131                      mainstem_inflows_t(1,ii-1))*24*60*60;
132                end
133                q1=q2;
134                q2=interp1(mainstem_inflows_t',mainstem_inflows_Q',...
135                   JDAY_initial+(time-1)*frequency);
136                if ~any(isnan(q2)) %may have some rounding issues, causing it to
                       ↪ go past JDAY_end?
137                   volin_BR1=volin_BR1+.5*(q1+q2)*...
138                      ((JDAY_initial+(time-1)*frequency)-...
139                      mainstem_inflows_t(1,index2))*24*60*60;
140                end
141             end
142         else
143             flow=Q.(Q.interpolation{1,i});
144             index1=find(flow(:,1)<=...
145                JDAY_initial+(time-2)*frequency,1,'last');
146             index2=find(flow(:,1)<=...
147                JDAY_initial+(time-1)*frequency,1,'last');
148             q1=interp1(flow(:,1),flow(:,2),JDAY_initial+...
149                (time-2)*frequency); %flowrate at beginning of timestep
150             if index1==index2
151                q2=interp1(flow(:,1),flow(:,2),JDAY_initial+...
152                   (time-1)*frequency);
153                volin=volin+.5*(q1+q2)*...
154                   ((JDAY_initial+(time-1)*frequency)-...
155                   (JDAY_initial+(time-2)*frequency))*24*60*60;
156             else
157                q2=flow(index1+1,2);
158                volin=volin+.5*(q1+q2)*...
159                   (flow(index1+1,1)-...
160                   (JDAY_initial+(time-2)*frequency))*24*60*60;
```

```
161                 for ii=index1+2:index2
162                     q1=q2;%start flowrate is equal to previous end flowrate
163                     q2=flow(ii,2);
164                     volin=volin+.5*(q1+q2)*(flow(ii,1)-flow(ii-1,1))...
165                         *24*60*60;
166                 end
167                 q1=q2;
168                 q2=interp1(flow(:,1),flow(:,2),...
169                     JDAY_initial+(time-1)*frequency);
170                 volin=volin+.5*(q1+q2)*((JDAY_initial+(time-1)*...
171                     frequency)-flow(index2,1))*24*60*60;
172             end
173         end
174     end
175     %VOLUMES OUT
176     %Outflow w/out interpolation - EXCEPT TURB (and spill if it's defined in
        ↪ the outputs, otherwise take values from Q)
177     if (isequal(char(Q.interpolation(2,i)),'outflow') | ...
178             (isequal(char(Q.interpolation(2,i)),'outflow_spill') & isempty(
                ↪ spill_discharges)) | ...
179             isequal(char(Q.interpolation(2,i)),'qwd')) & ...
180             isequal(char(Q.interpolation(3,i)),'OFF')
181         flow=Q.(Q.interpolation{1,i});
182         index1=find(flow(:,1)<=...
183             JDAY_initial+(time-2)*frequency,1,'last');
184         index2=find(flow(:,1)<=...
185             JDAY_initial+(time-1)*frequency,1,'last');
186         q1=flow(index1,2); %flowrate at beginning of timestep
187         if index1==index2
188             volout=volout+q1*frequency*24*60*60;
189         else
190             volout=volout+q1*(flow(index1+1,1)-(JDAY_initial+...
191                 (time-2)*frequency))*24*60*60;
192             for ii=index1+1:index2-1
193                 q1=flow(ii,2);
194                 volout=volout+q1*(flow(ii+1,1)-flow(ii,1))*24*60*60;
195             end
196             q1=flow(index2,2);
197             volout=volout+q1*((JDAY_initial+(time-1)*frequency)-...
198                 flow(index2,1))*24*60*60;
199         end
200     end
201     %Outflow w/ interpolation - EXCEPT TURB (and spill if it's defined in the
        ↪ outputs, otherwise take values from Q)
202     if (isequal(char(Q.interpolation(2,i)),'outflow') | ...
203             (isequal(char(Q.interpolation(2,i)),'outflow_spill') & isempty(
                ↪ spill_discharges)) | ...
204             isequal(char(Q.interpolation(2,i)),'qwd')) & ...
205             isequal(char(Q.interpolation(3,i)),'ON')
206         flow=Q.(Q.interpolation{1,i});
207         index1=find(flow(:,1)<=...
208             JDAY_initial+(time-2)*frequency,1,'last');
209         index2=find(flow(:,1)<=...
210             JDAY_initial+(time-1)*frequency,1,'last');
211         q1=interp1(flow(:,1),flow(:,2),JDAY_initial+(time-2)*...
212             frequency); %flowrate at beginning of timestep
213         if index1==index2
214             q2=interp1(flow(:,1),flow(:,2),JDAY_initial+(time-1)*...
215                 frequency);
216             volout=volout+.5*(q1+q2)*((JDAY_initial+(time-1)*...
217                 frequency)-(JDAY_initial+(time-2)*frequency))*24*60*60;
218         else
219             q2=flow(index1+1,2);
220             volout=volout+.5*(q1+q2)*(flow(index1+1,1)-...
```

```
            (JDAY_initial+(time-2)*frequency))*24*60*60;
         for ii=index1+2:index2
            q1=q2;%start flowrate is equal to previous end flowrate
            q2=flow(ii,2);
            volout=volout+.5*(q1+q2)*(flow(ii,1)-flow(ii-1,1))*...
               24*60*60;
         end
         q1=q2;
         q2=interp1(flow(:,1),flow(:,2),JDAY_initial+(time-1)*...
            frequency);
         volout=volout+.5*(q1+q2)*((JDAY_initial+(time-1)*...
            frequency)-flow(index2,1))*24*60*60;
      end
   end
end

%Turbine outflow using turb_discharges
q1=[]; q2=[];
timesteps=[JDAY_initial:frequency:JDAY_end];
%find turbine interpolation setting (b)
[a,b]=find(strcmp(Q.interpolation,'outflow_turb'));
if isequal(char(Q.interpolation(3,b)),'OFF')
   index1=find(timesteps(1,:)<=...
      JDAY_initial+(time-2)*frequency,1,'last');
   index2=find(timesteps(1,:)<=...
      JDAY_initial+(time-1)*frequency,1,'last');
   q1=turb_discharges(:,index1); %flowrates at beginning of timestep
   if index1==index2
      turbout=turbout+q1*frequency*24*60*60;
   else
      turbout=turbout+q1*(timesteps(1,index1+1)-(JDAY_initial+...
         (time-2)*frequency))*24*60*60;
      for ii=index1+1:index2-1
         q1=turb_discharges(:,ii);
         turbout=turbout+q1*(timesteps(1,ii+1)-timesteps(1,ii))*...
            24*60*60;
      end
      q1=turb_discharges(:,index2);
      turbout=turbout+q1*((JDAY_initial+(time-1)*frequency)-...
         timesteps(1,index2))*24*60*60;
   end
elseif isequal(char(Q.interpolation(3,b)),'ON')
   index1=find(timesteps(1,:)<=...
      JDAY_initial+(time-2)*frequency,1,'last');
   index2=find(timesteps(1,:)<=...
      JDAY_initial+(time-1)*frequency,1,'last');
   q1=interp1(timesteps',turb_discharges',JDAY_initial+(time-2)*...
      frequency);
   %if JDAY_initial+(time-2)*frequency=timesteps(1), interp1 outputs nan
   q1(isnan(q1))=turb_discharges(isnan(q1),1);
   if index1==index2
      q2=interp1(timesteps',turb_discharges',...
         JDAY_initial+(time-1)*frequency);
      turbout=turbout+.5*(q1+q2)*((JDAY_initial+(time-1)*...
         frequency)-(JDAY_initial+(time-2)*frequency))*24*60*60;
   else
      q2=turb_discharges(:,index1+1)';
      turbout=turbout+.5*(q1+q2)*(timesteps(1,index1+1)-...
         (JDAY_initial+(time-2)*frequency))*24*60*60;
      for ii=index1+2:index2
         q1=q2; %start flowrate is equal to previous end flowrate
         q2=turb_discharges(:,ii);
         turbout=turbout+.5*(q1+q2)*(timesteps(1,ii)-...
            timesteps(1,ii-1))*24*60*60;
```

```
285          end
286          q1=q2;
287          q2=interp1(timesteps',turb_discharges',JDAY_initial+...
288             (time-1)*frequency);
289          if ~any(isnan(q2)) %may have some rounding issues, causing it to go
                 ↪ past JDAY_end?
290             turbout=turbout+.5*(q1+q2)*((JDAY_initial+(time-1)*...
291                frequency)-timesteps(1,index2))*24*60*60;
292          end
293       end
294    end
295
296     %Spill outflow from spill_discharges
297    if ~isempty(spill_discharges)
298       q1=[]; q2=[];
299       timesteps=[JDAY_initial:frequency:JDAY_end];
300       %find spill interpolation setting (b)
301       [a,b]=find(strcmp(Q.interpolation,'outflow_spill'));
302       if isequal(char(Q.interpolation(3,b)),'OFF')
303          index1=find(timesteps(1,:)<=...
304             JDAY_initial+(time-2)*frequency,1,'last');
305          index2=find(timesteps(1,:)<=...
306             JDAY_initial+(time-1)*frequency,1,'last');
307          q1=spill_discharges(:,index1); %flowrates at beginning of timestep
308          if index1==index2
309             spillout=spillout+q1*frequency*24*60*60;
310          else
311             spillout=spillout+q1*(timesteps(1,index1+1)-(JDAY_initial+...
312                (time-2)*frequency))*24*60*60;
313             for ii=index1+1:index2-1
314                q1=spill_discharges(:,ii);
315                spillout=spillout+q1*(timesteps(1,ii+1)-timesteps(1,ii))*...
316                   24*60*60;
317             end
318             q1=spill_discharges(:,index2);
319             spillout=spillout+q1*((JDAY_initial+(time-1)*frequency)-...
320                timesteps(1,index2))*24*60*60;
321          end
322       elseif isequal(char(Q.interpolation(3,b)),'ON')
323          index1=find(timesteps(1,:)<=...
324             JDAY_initial+(time-2)*frequency,1,'last');
325          index2=find(timesteps(1,:)<=...
326             JDAY_initial+(time-1)*frequency,1,'last');
327          q1=interp1(timesteps',spill_discharges',JDAY_initial+(time-2)*...
328             frequency);
329          q1(isnan(q1))=spill_discharges(isnan(q1),1);
330          if index1==index2
331             q2=interp1(timesteps',spill_discharges',JDAY_initial+(time-1)*...
332                frequency);
333             spillout=spillout+.5*(q1+q2)*((JDAY_initial+(time-1)*...
334                frequency)-(JDAY_initial+(time-2)*frequency))*24*60*60;
335          else
336             q2=spill_discharges(:,index1+1)';
337             spillout=spillout+.5*(q1+q2)*(timesteps(1,index1+1)-...
338                (JDAY_initial+(time-2)*frequency))*24*60*60;
339             for ii=index1+2:index2
340                q1=q2; %start flowrate is equal to previous end flowrate
341                q2=spill_discharges(:,ii);
342                spillout=spillout+.5*(q1+q2)*(timesteps(1,ii)-...
343                   timesteps(1,ii-1))*24*60*60;
344             end
345             q1=q2;
346             q2=interp1(timesteps',spill_discharges',JDAY_initial+...
347                (time-1)*frequency);
```

```
            %may have some rounding issues, causing it to go past JDAY_end?
            if ~any(isnan(q2))
               spillout=spillout+.5*(q1+q2)*((JDAY_initial+(time-1)*...
                  frequency)-timesteps(1,index2))*24*60*60;
            end
         end
      end
   end

   deltav(time-1,:)=volin-volout-turbout-spillout+volin_BR1';
   Storage(time,:)=Storage(time-1,:)+deltav(time-1,:);
   %If too full and overtops SE curve (or drains and empties), linearly
       ↪ extrapolate
   pred=interp1(Q.SE_meters_m3(:,2),Q.SE_meters_m3(:,1),...
      Storage(time,:),'linear','extrap');
   Predictions(time,:)=pred;
end
```

ga_results_plotting_nobanding.m

```
%% plot_data

% L.info('OPTIMIZATION','Generating plotting data in plot_data folder.')
t_all=[start_date:frequency:start_date+days_forward];
for wb=1:size(CFG,2)

maxdelay=max([WQ{wb}.DO_narx.inputDelays' WQ{wb}.DO_narx.feedbackDelays']);
data_start=frequency*(maxdelay-1);
figure('units','normalized','outerposition',[0 0 1 1])
% Title
annotation('textbox',...
   [0.357741573033708 0.952787192414743 0.325808054820903
       ↪ 0.0410246887733755],...
   'String',{[CFG{wb}.Name ' Reservoir Optimization Results']},...
   'FontWeight','bold',...
   'FontSize',16,...
   'EdgeColor',[0.941176470588235 0.941176470588235 0.941176470588235],...
   'HorizontalAlignment','center');

%% Subplot 1: Turbine discharge patterns as active units
subplot(12,2,[1 3 5])
Ax1=plot(tprev_ic,xprev_ic{wb},'k',...
   t_all(1:1+day*(1/frequency)),[xprev_ic{wb}(end) x0_all(wb,1:day*(1/frequency)
       ↪ )],'b',...
   t_all(1:1+day*(1/frequency)),[xprev_ic{wb}(end) x_final{wb}],':r',...
   'LineWidth',2);
xlabel('Julian Day'); xlim([t_all(1)-data_start t_all(1+day*24)]);
set(gca,'YTick',0:1:no_of_units{wb});
ylabel('Active turbine units')
title('Active Turbine Units')
ylim([0 max([xprev_ic{wb}(end) x0_all(wb,1:day*(1/frequency)) x_final{wb}])])
ylims=get(gca,'ylim'); xlims=get(gca,'xlim'); xrange=xlims(2)-xlims(1); yrange=
    ↪ ylims(2)-ylims(1);
text(xlims(1)+0.025*xrange,ylims(1)+0.9*yrange,'(a)','FontSize',12);

%% Subplot 2: Turbine discharge patterns as flowrate
turb_discharges_x0{wb}=interp1(Qprojected{wb}.QOT_BR1_T(:,1),Qprojected{wb}.
    ↪ QOT_BR1_T(:,2),t_all(1:1+day*(1/frequency)));
turb_discharges{wb}=interp1(Q{wb}.QOT_BR1_T(:,1),Q{wb}.QOT_BR1_T(:,2),t_all(1:1+
    ↪ day*(1/frequency)));
turb_discharges_prev{wb}=interp1(Q{wb}.QOT_BR1_T(:,1),Q{wb}.QOT_BR1_T(:,2),
    ↪ tprev_ic);
subplot(12,2,[9 11 13])
```

```
Ax2=plot(tprev_ic,turb_discharges_prev{wb},'k',...
   t_all(1:1+day*(1/frequency)),[turb_discharges_prev{wb}(end)
      ↪ turb_discharges_x0{wb}(2:end)],'b',...
   t_all(1:1+day*(1/frequency)),[turb_discharges_prev{wb}(end) turb_discharges{
      ↪ wb}(2:end)],':r','LineWidth',2);
xlabel('Julian Day'); xlim([t_all(1)-data_start t_all(1+day*(1/frequency))]);
ylabel('Turbine discharge, cms')
title('Turbine Discharges')
ylims=get(gca,'ylim'); xlims=get(gca,'xlim'); xrange=xlims(2)-xlims(1); yrange=
   ↪ ylims(2)-ylims(1);
text(xlims(1)+0.025*xrange,ylims(1)+0.9*yrange,'(b)','FontSize',12);

%% Subplot 3: Spill discharge patterns as flowrate
spill_discharges_x0{wb}=interp1(Qprojected{wb}.QOT_BR1_S(:,1),Qprojected{wb}.
   ↪ QOT_BR1_S(:,2),t_all(1:1+day*(1/frequency)));
spill_discharges{wb}=interp1(Q{wb}.QOT_BR1_S(:,1),Q{wb}.QOT_BR1_S(:,2),t_all
   ↪ (1:1+day*(1/frequency)));
spill_discharges_prev{wb}=interp1(Qprojected{wb}.QOT_BR1_S(:,1),Qprojected{wb}.
   ↪ QOT_BR1_S(:,2),tprev_ic);
subplot(12,2,[17 19 21])
Ax2=plot(tprev_ic,spill_discharges_prev{wb},'k',...
   t_all(1:1+day*(1/frequency)),[spill_discharges_prev{wb}(end)
      ↪ spill_discharges_x0{wb}(2:end)],'b',...
   t_all(1:1+day*(1/frequency)),[spill_discharges_prev{wb}(end) spill_discharges
      ↪ {wb}(2:end)],':r','LineWidth',2);
xlabel('Julian Day'); xlim([t_all(1)-data_start t_all(1+day*(1/frequency))]);
ylabel('Spill discharge, cms')
title('Spill Discharges')
if all([spill_discharges_prev{wb}(end) spill_discharges_x0{wb} spill_discharges{
   ↪ wb}]==0)
   ylim([0 1])
end
ylims=get(gca,'ylim'); xlims=get(gca,'xlim'); xrange=xlims(2)-xlims(1); yrange=
   ↪ ylims(2)-ylims(1);
text(xlims(1)+0.025*xrange,ylims(1)+0.9*yrange,'(c)','FontSize',12);

%% Subplot 4: Headwater elevations
[~,~,HWs_x0{wb},~,~]=activeunits_to_discharges(x0_all(wb,1:day*(1/frequency)),
   ↪ t_all(1:1+day*(1/frequency)),...
                  frequency,Qprojected{wb},ic_elev_first{wb},...
         turbine_discharge{wb},[],[],[]);
HWs_prev{wb}=interp1(Q{wb}.ELWS(:,1),Q{wb}.ELWS(:,2),tprev_ic);
HWs{wb}=interp1(Q{wb}.ELWS(:,1),Q{wb}.ELWS(:,2),t_all(1:1+day*(1/frequency)));
subplot(12,2,[2 4 6])
Ax3=plot(tprev_ic,HWs_prev{wb},'k',...
   t_all(1:1+day*(1/frequency)),HWs_x0{wb},'b',...
   t_all(1:1+day*(1/frequency)),HWs{wb},':r','LineWidth',2);
hold on;
h5=plot([t_all(1) t_all(1+day*(1/frequency))],...
   [ELWS_limit{wb}(1) ELWS_limit{wb}(1)],':k',...
   'LineWidth',1.5);
plot([t_all(1) t_all(1+day*(1/frequency))],...
   [ELWS_limit{wb}(2) ELWS_limit{wb}(2)],':k',...
   'LineWidth',1.5)
h6=scatter(ELWS_targets{wb}(:,1),ELWS_targets{wb}(:,2));
hold off;
xlabel('Julian Day'); xlim([t_all(1)-data_start t_all(1+day*(1/frequency))]);
ylabel('Elevation, m')
title('Headwater Elevation')
ylims=get(gca,'ylim'); xlims=get(gca,'xlim'); xrange=xlims(2)-xlims(1); yrange=
   ↪ ylims(2)-ylims(1);
text(xlims(1)+0.025*xrange,ylims(1)+0.9*yrange,'(d)','FontSize',12);

%% Subplot 5: Discharge DO
```

```
DO_pred_x0{wb}=interp1(Qprojected{wb}.CWO(Qprojected{wb}.CWO(:,2)~=0,1),
    ↪ Qprojected{wb}.CWO(Qprojected{wb}.CWO(:,2)~=0,2),t_all(2:1+day*(1/
    ↪ frequency)));
DO_pred{wb}=interp1(Q{wb}.CWO(:,1),Q{wb}.CWO(:,2),t_all(2:1+day*(1/frequency)));
flowout_x0=turb_discharges_x0{wb}(2:end)+spill_discharges_x0{wb}(2:end);
flowout=turb_discharges{wb}(2:end)+spill_discharges{wb}(2:end);
DO_pred_x0{wb}(flowout_x0==0)=nan; DO_pred{wb}(flowout==0)=nan;
Output_no0s{wb}=interp1(Qprojected{wb}.CWO(find(Qprojected{wb}.CWO(:,2)~=0),1)
    ↪ ,...
   Qprojected{wb}.CWO(find(Qprojected{wb}.CWO(:,2)~=0),2),...
   [t_all(1)-data_start:frequency:t_all(1)])';
if interp1(Qprojected{wb}.QOT_BR1_T(:,1),Qprojected{wb}.QOT_BR1_T(:,2),...
      tprev_ic(end))==0 & ...
      interp1(Qprojected{wb}.QOT_BR1_S(:,1),Qprojected{wb}.QOT_BR1_S(:,2),...
      tprev_ic(end))==0
   DOinitcon{wb}=nan;
else
   DOinitcon{wb}=Output_no0s{wb}(end);
end
Outputprev{wb}=interp1([t_all(1)-data_start:frequency:t_all(1)],Output_no0s{wb
    ↪ },...
   tprev_ic);
j=find(interp1(Qprojected{wb}.QOT_BR1_T(:,1),Qprojected{wb}.QOT_BR1_T(:,2),...
   tprev_ic)==0 & ...
   interp1(Qprojected{wb}.QOT_BR1_S(:,1),Qprojected{wb}.QOT_BR1_S(:,2),...
   tprev_ic)==0);
Outputprev{wb}(j)=nan; clearvars j
subplot(12,2,[10 12 14])
h1=plot(tprev_ic,Outputprev{wb},'k','LineWidth',2);
hold on;
h2=plot(t_all(1:1+day*(1/frequency)),[DOinitcon{wb} DO_pred_x0{wb}],'b','
    ↪ LineWidth',2);
h3=plot(t_all(1:1+day*(1/frequency)),[DOinitcon{wb} DO_pred{wb}],':r','LineWidth
    ↪ ',2);
index=~isnan(W2validation{wb}.DO(:,2)); index2=isnan([DOinitcon{wb} DO_pred{wb
    ↪ }]);
%Remove rows with zeros (no discharge)
W2_no0s=W2validation{wb}.DO(index,:);
%Smooth data
W2_no0s_smooth(:,1)=W2_no0s(:,1); W2_no0s_smooth(:,2)=smooth(W2_no0s(:,1),
    ↪ W2_no0s(:,2),1);
W2_no0s_smooth2(:,1)=t_all(1:1+day*(1/frequency))';
W2_no0s_smooth2(:,2)=interp1(W2_no0s_smooth(:,1),W2_no0s_smooth(:,2),t_all(1:1+
    ↪ day*(1/frequency))');
W2_no0s_smooth2(index2,2)=nan;
h7=plot([t_all(1); W2_no0s_smooth2(W2_no0s_smooth2(:,1)>t_all(1),1)],...
   [DOinitcon{wb}; W2_no0s_smooth2(W2_no0s_smooth2(:,1)>t_all(1),2)],'g','
       ↪ LineWidth',2);
if ~isnan(WQ{wb}.DO_limit(1))
   h5=plot([t_all(1) t_all(1+day*(1/frequency))],[WQ{wb}.DO_limit(1) WQ{wb}.
       ↪ DO_limit(1)],':k',...
      'LineWidth',1.5);
elseif ~isnan(WQ{wb}.DO_limit(2))
   plot([t_all(1) t_all(1+day*(1/frequency))],[WQ{wb}.DO_limit(2) WQ{wb}.
       ↪ DO_limit(2)],':k',...
      'LineWidth',1.5);
end
xlabel('Julian Day'); xlim([t_all(1)-data_start t_all(1+day*(1/frequency))]);
ylabel('DO, mg/L');
ylim([min([min(W2_no0s_smooth2(W2_no0s_smooth2(:,1)>t_all(1),2)) min(DO_pred{wb
    ↪ }) min(DO_pred_x0{wb}) Output_no0s{wb}' WQ{wb}.DO_limit(1) WQ{wb}.
    ↪ DO_limit(2)])-.25...
   max([max(W2_no0s_smooth2(W2_no0s_smooth2(:,1)>t_all(1),2)) max(DO_pred{wb})
       ↪ max(DO_pred_x0{wb}) Output_no0s{wb}' WQ{wb}.DO_limit(1) WQ{wb}.
```

```
       ↪ DO_limit(2)])+.25]);
title('Discharge DO Predictions')
ylims=get(gca,'ylim'); xlims=get(gca,'xlim'); xrange=xlims(2)-xlims(1); yrange=
    ↪ ylims(2)-ylims(1);
text(xlims(1)+0.025*xrange,ylims(1)+0.9*yrange,'(e)','FontSize',12);
AME{wb}.DO=nanmean(abs(W2_no0s_smooth2(W2_no0s_smooth2(:,1)>t_all(1),2)-DO_pred{
    ↪ wb}'));
str=['AME = ', sprintf('%5.3f',AME{wb}.DO), ' mg/L'];
text(xlims(1)+0.025*xrange,ylims(1)+0.1*yrange,str,'FontSize',12);
%Compute WQ average slack using W2 results
slack_compute=W2_no0s_smooth2(W2_no0s_smooth2(:,1)>t_all(1),2);
non_nan_count=sum(~isnan(slack_compute),1);
if ~isnan(WQ{wb}.DO_limit(1))
   slacks{wb}.DO.W2=sum(-min(0,slack_compute-WQ{wb}.DO_limit(1)),1)./
       ↪ non_nan_count;
elseif ~isnan(WQ{wb}.DO_limit(2))
   slacks{wb}.DO.W2=sum(-min(0,slack_compute-WQ{wb}.DO_limit(2)),1)./
       ↪ non_nan_count;
else
   slacks{wb}.DO.W2=0;
end
clearvars W2_no0s_smooth index index2 W2_no0s str slack_compute non_nan_count

%% Subplot 5: Discharge Temp
Temp_pred_x0{wb}=interp1(Qprojected{wb}.TWO(Qprojected{wb}.TWO(:,2)~=0,1),
    ↪ Qprojected{wb}.TWO(Qprojected{wb}.TWO(:,2)~=0,2),t_all(2:1+day*(1/
    ↪ frequency)));
Temp_pred{wb}=interp1(Q{wb}.TWO(:,1),Q{wb}.TWO(:,2),t_all(2:1+day*(1/frequency))
    ↪ );
Temp_pred_x0{wb}(flowout_x0==0)=nan; Temp_pred{wb}(flowout==0)=nan;
clearvars flowout_x0
Output_no0s{wb}=interp1(Qprojected{wb}.TWO(find(Qprojected{wb}.TWO(:,2)~=0),1)
    ↪ ,...
   Qprojected{wb}.TWO(find(Qprojected{wb}.TWO(:,2)~=0),2),...
   [t_all(1)-data_start:frequency:t_all(1)])';
if interp1(Qprojected{wb}.QOT_BR1_T(:,1),Qprojected{wb}.QOT_BR1_T(:,2),...
      tprev_ic(end))==0 & ...
      interp1(Qprojected{wb}.QOT_BR1_S(:,1),Qprojected{wb}.QOT_BR1_S(:,2),...
      tprev_ic(end))==0
   Tempinitcon{wb}=nan;
else
   Tempinitcon{wb}=Output_no0s{wb}(end);
end
Outputprev{wb}=interp1([t_all(1)-data_start:frequency:t_all(1)],Output_no0s{wb
    ↪ },...
   tprev_ic);
j=find(interp1(Qprojected{wb}.QOT_BR1_T(:,1),Qprojected{wb}.QOT_BR1_T(:,2),...
   tprev_ic)==0 & ...
   interp1(Qprojected{wb}.QOT_BR1_S(:,1),Qprojected{wb}.QOT_BR1_S(:,2),...
   tprev_ic)==0);
Outputprev{wb}(j)=nan; clearvars j
subplot(12,2,[18 20 22])
h1=plot(tprev_ic,Outputprev{wb},'k','LineWidth',2);
hold on;
h2=plot(t_all(1:1+day*(1/frequency)),[Tempinitcon{wb} Temp_pred_x0{wb}],'b','
    ↪ LineWidth',2);
h3=plot(t_all(1:1+day*(1/frequency)),[Tempinitcon{wb} Temp_pred{wb}],':r','
    ↪ LineWidth',2);
index=~isnan(W2validation{wb}.T(:,2)); index2=isnan([Tempinitcon{wb} Temp_pred{
    ↪ wb}]);
%Remove rows with zeros (no discharge)
W2_no0s=W2validation{wb}.T(index,:);
%Smooth data
W2_no0s_smooth(:,1)=W2_no0s(:,1); W2_no0s_smooth(:,2)=smooth(W2_no0s(:,1),
```

```
    ↪ W2_no0s(:,2),1);
W2_no0s_smooth2(:,1)=t_all(1:1+day*(1/frequency))';
W2_no0s_smooth2(:,2)=interp1(W2_no0s_smooth(:,1),W2_no0s_smooth(:,2),t_all(1:1+
    ↪ day*(1/frequency))');
W2_no0s_smooth2(index2,2)=nan;
h7=plot([t_all(1); W2_no0s_smooth2(W2_no0s_smooth2(:,1)>t_all(1),1)],...
   [Tempinitcon{wb}; W2_no0s_smooth2(W2_no0s_smooth2(:,1)>t_all(1),2)],'g','
       ↪ LineWidth',2);
if ~isnan(WQ{wb}.Temp_limit(1))
   h5=plot([t_all(1) t_all(1+day*(1/frequency))],[WQ{wb}.Temp_limit(1) WQ{wb}.
       ↪ Temp_limit(1)],':k',...
      'LineWidth',1.5);
elseif ~isnan(WQ{wb}.Temp_limit(2))
   h5=plot([t_all(1) t_all(1+day*(1/frequency))],[WQ{wb}.Temp_limit(2) WQ{wb}.
       ↪ Temp_limit(2)],':k',...
      'LineWidth',1.5);
end
xlabel('Julian Day'); xlim([t_all(1)-data_start t_all(1+day*(1/frequency))]);
ylabel('Temperature, C');
ylim([min([min(W2_no0s_smooth2(W2_no0s_smooth2(:,1)>t_all(1),2)) min(Temp_pred{
    ↪ wb}) min(Temp_pred_x0{wb}) Output_no0s{wb}' WQ{wb}.Temp_limit(1) WQ{wb}.
    ↪ Temp_limit(2)])-.25...
   max([max(W2_no0s_smooth2(W2_no0s_smooth2(:,1)>t_all(1),2)) max(Temp_pred{wb})
       ↪  max(Temp_pred_x0{wb}) Output_no0s{wb}' WQ{wb}.Temp_limit(1) WQ{wb}.
       ↪ Temp_limit(2)])+.25]);
title('Discharge Temperature Predictions')
ylims=get(gca,'ylim'); xlims=get(gca,'xlim'); xrange=xlims(2)-xlims(1); yrange=
    ↪ ylims(2)-ylims(1);
text(xlims(1)+0.025*xrange,ylims(1)+0.9*yrange,'(f)','FontSize',12);
AME{wb}.T=nanmean(abs(W2_no0s_smooth2(W2_no0s_smooth2(:,1)>t_all(1),2)-Temp_pred
    ↪ {wb}'));
str=['AME = ', sprintf('%5.3f',AME{wb}.T), ' C'];
text(xlims(1)+0.025*xrange,ylims(1)+0.1*yrange,str,'FontSize',12);
%Compute WQ average slack using W2 results
slack_compute=W2_no0s_smooth2(W2_no0s_smooth2(:,1)>t_all(1),2);
non_nan_count=sum(~isnan(slack_compute),1);
if ~isnan(WQ{wb}.Temp_limit(1))
   slacks{wb}.T.W2=sum(-min(0,slack_compute-WQ{wb}.Temp_limit(1)),1)./
       ↪ non_nan_count;
elseif ~isnan(WQ{wb}.Temp_limit(2))
   slacks{wb}.T.W2=sum(-min(0,slack_compute-WQ{wb}.Temp_limit(2)),1)./
       ↪ non_nan_count;
else
   slacks{wb}.T.W2=0;
end
clearvars W2_no0s_smooth W2_no0s_smooth2 index index2 W2_no0s flowout str
    ↪ slack_compute non_nan_count

legend1=legend([h1 h2 h3 h7 h5 h6],'Past Values',...
   'Projected Operations',...
   'Optimal Solution',...
   'W2 Validation at Optimal Solution',...
   'Constraint Bounds',...
   'Target Elevations');
set(legend1,...
   'Position',[0.39086885358981 0.0131729985010991 0.256670797003518
       ↪ 0.119367775250152],...
   'FontSize',10);

end
```

int_mutation.m

```
function mutationChildren = int_mutation(parents,options,GenomeLength, ...
   FitnessFcn,state,thisScore,thisPopulation)

% Mutation function to generate childrens satisfying the range and integer
% constraints on decision variables.

shrink = .01;
scale = 1;
scale = scale - shrink * scale * state.Generation/options.Generations;
range = options.PopInitRange;
lower = range(1,:);
upper = range(2,:);
scale = scale * (upper - lower);
mutationPop = length(parents);
% The use of ROUND function will make sure that childrens are integers.
mutationChildren = repmat(lower,mutationPop,1) + ...
   round(repmat(scale,mutationPop,1) .* rand(mutationPop,GenomeLength));
% End of mutation function
```

int_pop.m

```
function Population = int_pop(GenomeLength,FitnessFcn,options)

totalpopulation = sum(options.PopulationSize);
range = options.PopInitRange;
lower= range(1,:);
span = range(2,:) - lower;
% The use of ROUND function will make sure that individuals are integers.
Population = repmat(lower,totalpopulation,1) + ...
   round(repmat(span,totalpopulation,1).*...
   rand(totalpopulation,GenomeLength));
% End of creation function
```

narx_predictions.m

```
function pred=narx_predictions(NARX_model,frequency,t,Q,x,...
   turb_discharges,spill_discharges,mainstem_inflows,previous_Output,flag)

% Calculates WQ predictions using a trained family of NARX models
%
% Inputs:
% NARX_model - structure containing everything needed to make WQ
% discharge predictions, including:
% turb_colum - column in exogenous variables with turb flows
% spill_column - column in exogenous variables with spill flows
% inputDelays - delays for exogenous inputs
% feedbackDelays - delays for prediction feedbacks
% input_variables - 2 row cell containing variable names in first
% row and column number in second. For example, 'MET_WB1'
% contains multiple columns of data but only some may be used
% for NARX predictions
% bias - bias for each trained neural network
% weights - weights for each trained neural network (sum to 1)
% narx_net_closed - neural networks
% frequency - frequency of predictions (hourly=1/24)
% t  time series of JDAY values
% Q - all other inflows and outflows, interpolation settings,
% storage-elev curve, and tailwater curve
% x - hourly turbine time series (as rows for vectorizing!), integers
% between 0 and no_of_units
% turb_discharges - matrix the same size as x that includes the turbine
% discharge flowrates over the time t
```

```
% spill_discharges - spill discharge flowrates
% mainstem_inflows - structure containing Q, T, and DO with time series
% data from previous days' optimal solution
% previous_Output - the time series of previous outputs of the
% constituent being predicted by NARX model
% flag - 'do' if predicting DO, to check to make sure not <0
% Outputs:
% pred  vector of NARX model predictions for water quality, with NaN
% values anywhere turb+spill=0

if isempty(mainstem_inflows)
   mainstem_inflows.Q=[];
   mainstem_inflows.T=[];
   mainstem_inflows.DO=[];
end
if exist('mainstem_inflows', 'var') && isfield(mainstem_inflows, 'Q')
   if isempty(mainstem_inflows.Q) mainstem_inflows.Q=[]; end
else
   mainstem_inflows.Q=[];
end
if exist('mainstem_inflows', 'var') && isfield(mainstem_inflows, 'T')
   if isempty(mainstem_inflows.T) mainstem_inflows.T=[]; end
else
   mainstem_inflows.T=[];
end
if exist('mainstem_inflows', 'var') && isfield(mainstem_inflows, 'DO')
   if isempty(mainstem_inflows.DO) mainstem_inflows.DO=[]; end
else
   mainstem_inflows.DO=[];
end

maxdelay=max([NARX_model.inputDelays'; NARX_model.feedbackDelays']);
data_start=frequency*(maxdelay-1);
timesteps=[t(1)-data_start:frequency:t];
Output_no0s=interp1(previous_Output(find(previous_Output(:,2)~=0),1),...
   previous_Output(find(previous_Output(:,2)~=0),2),timesteps)';
clearvars timesteps
y1=con2seq([Output_no0s' nan(1,size(x,2))]);
timesteps2=[t(1)-data_start:frequency:t t(2:end)];
Inputs=nan(size(timesteps2,2),size(NARX_model.input_variables,2));
index_QIN_BR1=[]; index_TIN_BR1=[]; index_CIN_BR1=[];
for i=1:size(NARX_model.input_variables,2)
   %If mainstem_inflows are provided and the variable is BR1 Q, T, or DO
   if ~isempty(mainstem_inflows.Q) & ...
         isequal(NARX_model.input_variables{1,i},'QIN_BR1')
      index_QIN_BR1=i;
   end
   if ~isempty(mainstem_inflows.T) & ...
         isequal(NARX_model.input_variables{1,i},'TIN_BR1')
      index_TIN_BR1=i;
   end
   if ~isempty(mainstem_inflows.DO) & ...
         isequal(NARX_model.input_variables{1,i},'CIN_BR1')
      index_CIN_BR1=i;
   end
   Inputs(:,i)=interp1(Q.(sprintf(NARX_model.input_variables{1,i}))(:,1),...
      Q.(sprintf(NARX_model.input_variables{1,i}))(:,NARX_model.input_variables
         ↪ {2,i}+1),...
      timesteps2);
end
clearvars i timesteps2
pred=nan(size(x,1),size(x,2));
for i=1:size(x,1) %attempt to vectorize this part later
   %Update mainstem_inflows, if necessary
```

```
    if ~isempty(index_QIN_BR1)
       Inputs(size(Inputs,1)-size(mainstem_inflows.Q,2)+1:...
          size(Inputs,1),index_QIN_BR1)=mainstem_inflows.Q(i,:)';
    end
    if ~isempty(index_TIN_BR1)
       Inputs(size(Inputs,1)-size(mainstem_inflows.T,2)+1:...
          size(Inputs,1),index_TIN_BR1)=mainstem_inflows.T(i,:)';
    end
    if ~isempty(index_CIN_BR1)
       Inputs(size(Inputs,1)-size(mainstem_inflows.DO,2)+1:...
          size(Inputs,1),index_CIN_BR1)=mainstem_inflows.DO(i,:)';
    end
    %Update turbine outflow and spill outflow columns, if necessary
    if ~isempty(turb_discharges)
       Inputs(size(Inputs,1)-size(turb_discharges,2)+...
          1:size(Inputs,1),NARX_model.turb_column)=...
          turb_discharges(i,:)';
    end
    if ~isempty(spill_discharges)
    Inputs(size(Inputs,1)-size(turb_discharges,2)+...
       1:size(Inputs,1),NARX_model.spill_column)=...
       spill_discharges(i);
    end
    u1 = con2seq(Inputs');
    if size(NARX_model.narx_net_closed,2)==1
       if iscell(NARX_model.narx_net_closed)
          [p1,Pi1,Ai1,t1]=preparets(NARX_model.narx_net_closed{:},u1,{},y1);
          yp1(1,:)=NARX_model.narx_net_closed{:}(p1,Pi1,Ai1);
       else
          [p1,Pi1,Ai1,t1]=preparets(NARX_model.narx_net_closed,u1,{},y1);
          yp1(1,:)=NARX_model.narx_net_closed(p1,Pi1,Ai1);
       end
    else
       for j=1:size(NARX_model.narx_net_closed,2)
          [p1,Pi1,Ai1,t1]=preparets(NARX_model.narx_net_closed{j},u1,{},y1);
          yp1(j,:)=NARX_model.narx_net_closed{j}(p1,Pi1,Ai1);
       end
    end
    yp1=cell2mat(yp1);
    if size(NARX_model.weights,1)==1
       yp1=yp1-NARX_model.bias;
       pred(i,:)=yp1;
    else
       yp1=bsxfun(@minus,yp1,NARX_model.bias);
       pred(i,:)=sum(bsxfun(@times,NARX_model.weights,yp1));
    end
    clearvars yp1
end
clearvars i j
if strcmp(flag,'do')
   pred=max(0,pred); %can't have negative concentrations of DO
end
for i=1:size(x,1)
   j=[];
   if ~isempty(spill_discharges)
      if all(spill_discharges(i)==0)
         j=find(x(i,:)==0);
      else
         if size(spill_discharges(i,:),2)==1 %if solving subproblem
            j=[];
         else
            j=find(turb_discharges(i,2:end)==0 & spill_discharges(i,2:end)==0);
               ↪ %if solving final solution over all subproblems
         end
```

```
      end
   else
         j=find(x(i,:)==0 & interp1(Q.QOT_BR1_S(:,1),Q.QOT_BR1_S(:,2),t(2:end))
             ↪ ==0);
   end
   pred(i,j)=nan;
end
clearvars i j
```

NARX_retrain.m

```
%Retrain temperature and DO NARX models for wb
%For each iteration, add the new W2 validation run data to the training data set
    ↪ , and then retrain. This means the training set grows with each iteration
    ↪ .

%% DO validation run
timesteps=[t_all(1)-max(WQ{wb}.DO_narx.inputDelays)/24:(1/24):t_all(end)]';
vars=WQ{wb}.DO_narx.input_variables;
Inputs{wb}.discharge_DO{iter}=[]; count=0;
for i=1:size(vars,2)
   count=count+1;
      if strfind(char(vars(1,i)),'TIN')
      flow_variable=strrep(char(vars(1,i)),'TIN','QIN');
   elseif strfind(char(vars(1,i)),'CIN')
      flow_variable=strrep(char(vars(1,i)),'CIN','QIN');
   elseif strfind(char(vars(1,i)),'TTR')
      flow_variable=strrep(char(vars(1,i)),'TTR','QTR');
   elseif strfind(char(vars(1,i)),'CTR')
      flow_variable=strrep(char(vars(1,i)),'CTR','QTR');
   else
      flow_variable=char(vars(1,i));
   end
   if ~strcmp(char(vars(1,i)),'MET_WB1') %assume interpolation for MET data
      for ii=1:size(Q{wb}.interpolation,2)
         if strcmp(char(Q{wb}.interpolation(1,ii)),flow_variable)
            break
         end
      end
      if strcmp(char(Q{wb}.interpolation(3,ii)),'ON')
         Inputs{wb}.discharge_DO{iter}(:,i)=interp1(Q{wb}.(vars{1,i})(:,1),...
            Q{wb}.(vars{1,i})(:,vars{2,i}+1),timesteps);
      elseif strcmp(char(Q{wb}.interpolation(3,ii)),'OFF')
         for iii=1:size(timesteps,1)
            index=find(Q{wb}.(vars{1,i})(:,1)<=timesteps(ii),1,'last');
            Inputs{wb}.discharge_DO{iter}(iii,i)=Q{wb}.(vars{1,i})(index,vars{2,
                ↪ i}+1);
         end
      end
   else
      Inputs{wb}.discharge_DO{iter}(:,i)=interp1(Q{wb}.(vars{1,i})(:,1),...
         Q{wb}.(vars{1,i})(:,vars{2,i}+1),timesteps);
   end
end
%Smooth output data (zeros already removed in W2validation)
Discharge.DO_no0s=W2validation{wb}.DO;
Discharge.DO_no0s_smooth(:,1)=Discharge.DO_no0s(:,1);
Discharge.DO_no0s_smooth(:,2)=smooth(Discharge.DO_no0s(:,1),Discharge.DO_no0s
    ↪ (:,2),1);
Output{wb}.discharge_DO{iter}(:,1)=interp1(Discharge.DO_no0s_smooth(:,1),
    ↪ Discharge.DO_no0s_smooth(:,2),timesteps);
%Convert to cells
Inputs_seq{wb}.discharge_DO{iter*2-1} = con2seq(Inputs{wb}.discharge_DO{iter}');
```

```
Output_seq{wb}.discharge_DO{iter*2-1} = con2seq(Output{wb}.discharge_DO{iter}');
clearvars i ii iii flow_variable

%% DO validation run with turb and spill flipped
[~,a]= find(cellfun(@(s) ~isempty(strfind('QOT_BR1_T', s)), vars)==1);
[~,b]= find(cellfun(@(s) ~isempty(strfind('QOT_BR1_S', s)), vars)==1);
timesteps_flip=[t_all(1):(1/24):t_all(end)]';
Inputs{wb}.discharge_DO_flip{iter}=Inputs{wb}.discharge_DO{iter};
turb=Inputs{wb}.discharge_DO{iter}(:,a); spill=Inputs{wb}.discharge_DO{iter}(:,b);
Inputs{wb}.discharge_DO_flip{iter}(size(timesteps,1)-size(timesteps_flip,1)+1:end,a)=spill(size(timesteps,1)-size(timesteps_flip,1)+1:end);
Inputs{wb}.discharge_DO_flip{iter}(size(timesteps,1)-size(timesteps_flip,1)+1:end,b)=turb(size(timesteps,1)-size(timesteps_flip,1)+1:end);
%Smooth output data (zeros already removed in W2validation)
Discharge.DO_no0s=W2validation_flip{wb}.DO;
Discharge.DO_no0s_smooth(:,1)=Discharge.DO_no0s(:,1);
Discharge.DO_no0s_smooth(:,2)=smooth(Discharge.DO_no0s(:,1),Discharge.DO_no0s(:,2),1);
Output{wb}.discharge_DO_flip{iter}(:,1)=interp1(Discharge.DO_no0s_smooth(:,1),Discharge.DO_no0s_smooth(:,2),timesteps);
%Convert to cells
Inputs_seq{wb}.discharge_DO{iter*2} = con2seq(Inputs{wb}.discharge_DO_flip{iter}');
Output_seq{wb}.discharge_DO{iter*2} = con2seq(Output{wb}.discharge_DO_flip{iter}');
clearvars vars Discharge turb spill a b

%% DO training
%Combine them all into single Input and Output cell arrays
Inputs_seq_mul{wb}.discharge_DO=catsamples(Inputs_seq{wb}.discharge_DO{:},'pad');
Output_seq_mul{wb}.discharge_DO=catsamples(Output_seq{wb}.discharge_DO{:},'pad');

%Train DO model - start with best DO model from before (greatest weight)
fprintf(['Training 5 DO models and picking the best \n'])
for i=1:5
   inputDelays = [0 1 12]; %[10:14]?
   feedbackDelays = [1];
   hiddenNeurons=[10];
   narx_net{i} = narxnet(inputDelays,feedbackDelays,hiddenNeurons);
   narx_net{i}.divideFcn = 'dividerand';
   % The property DIVIDEMODE set to TIMESTEP means that targets are divided
   % into training, validation and test sets according to timesteps.
   % For a list of data division modes type: help nntype_data_division_mode
   narx_net{i}.divideMode = 'time'; % Divide up every value
   narx_net{i}.divideParam.trainRatio = 70/100;
   narx_net{i}.divideParam.valRatio = 15/100;
   narx_net{i}.divideParam.testRatio = 15/100;
   narx_net{i}.trainParam.min_grad = 1e-10;
   narx_net{i}.trainFcn = 'trainlm';
   narx_net{i}.trainParam.showWindow=0;
   narx_net{i}.trainParam.showCommandLine=1;
   [Xs,Xi,Ai,Ts]=preparets(narx_net{i},Inputs_seq_mul{wb}.discharge_DO,{}, ...
      Output_seq_mul{wb}.discharge_DO);
   [narx_net{i},~]=train(narx_net{i},Xs,Ts,Xi,Ai,'UseParallel','yes');
   narx_net_closed{i} = closeloop(narx_net{i});
   narx_net_closed{i}.trainParam.mu_max=1e14;
   [Xs,Xi,Ai,Ts]=preparets(narx_net_closed{i},Inputs_seq_mul{wb}.discharge_DO,{}, ...
      Output_seq_mul{wb}.discharge_DO);
   [narx_net_closed{i},tr{i}]=train(narx_net_closed{i},Xs,Ts,Xi,Ai,'UseParallel','yes');
```

```
end
for i=1:5 tr2(i)=tr{i}.best_perf; end
[~,b]=min(tr2); WQ{wb}.DO_narx.narx_net_closed=narx_net_closed{b};
yp1= WQ{wb}.DO_narx.narx_net_closed(Xs,Xi,Ai);
%Calculate bias & standard dev using only predictions at test timepoints
bias=cell2mat(yp1(tr{b}.testInd))-cell2mat(Ts(tr{b}.testInd)); bias=nanmean(bias
    ↪ );
allerrors=(cell2mat(yp1(tr{b}.testInd))-bias)-cell2mat(Ts(tr{b}.testInd));
allerrors=allerrors(~isnan(allerrors));
[~,sigmahat] = normfit(allerrors);
WQ{wb}.DO_narx.bias=bias;
WQ{wb}.DO_narx.weights=1;
WQ{wb}.DO_narx.inputDelays=inputDelays;
WQ{wb}.DO_narx.std_dev=sigmahat;
WQ{wb}.DO_narx.Inputs=Inputs{wb}.discharge_DO;
WQ{wb}.DO_narx.Output=Output{wb}.discharge_DO;
if isfield(WQ{wb}.DO_narx,'train_time')
   WQ{wb}.DO_narx=rmfield(WQ{wb}.DO_narx,{'train_time'});
end
if isfield(WQ{wb}.DO_narx,'Discharge_DO_no0s')
   WQ{wb}.DO_narx=rmfield(WQ{wb}.DO_narx,{'Discharge_DO_no0s'});
end
clearvars b Xs Xi Ai Ts tr tr2 b yp1 TS bias narx_net_closed narx_net muhat
    ↪ sigmahat

%% Temp validation run
timesteps=[t_all(1)-max(WQ{wb}.Temp_narx.inputDelays)/24:(1/24):t_all(end)]';
vars=WQ{wb}.Temp_narx.input_variables;
Inputs{wb}.discharge_Temp{iter}=[];
for i=1:size(vars,2)
   if strfind(char(vars(1,i)),'TIN')
      flow_variable=strrep(char(vars(1,i)),'TIN','QIN');
   elseif strfind(char(vars(1,i)),'CIN')
      flow_variable=strrep(char(vars(1,i)),'CIN','QIN');
   elseif strfind(char(vars(1,i)),'TTR')
      flow_variable=strrep(char(vars(1,i)),'TTR','QTR');
   elseif strfind(char(vars(1,i)),'CTR')
      flow_variable=strrep(char(vars(1,i)),'CTR','QTR');
   else
      flow_variable=char(vars(1,i));
   end
   if ~strcmp(char(vars(1,i)),'MET_WB1') %assume interpolation for MET data
      for ii=1:size(Q{wb}.interpolation,2)
         if strcmp(char(Q{wb}.interpolation(1,ii)),flow_variable)
            break
         end
      end
      if strcmp(char(Q{wb}.interpolation(3,ii)),'ON')
         Inputs{wb}.discharge_Temp{iter}(:,i)=interp1(Q{wb}.(vars{1,i})(:,1),...
            Q{wb}.(vars{1,i})(:,vars{2,i}+1),timesteps);
      elseif strcmp(char(Q{wb}.interpolation(3,ii)),'OFF')
         for iii=1:size(timesteps,1)
            index=find(Q{wb}.(vars{1,i})(:,1)<=timesteps(ii),1,'last');
            Inputs{wb}.discharge_Temp{iter}(iii,i)=Q{wb}.(vars{1,i})(index,vars
                ↪ {2,i}+1);
         end
      end
   else
      Inputs{wb}.discharge_Temp{iter}(:,i)=interp1(Q{wb}.(vars{1,i})(:,1),...
         Q{wb}.(vars{1,i})(:,vars{2,i}+1),timesteps);
   end
end
%Smooth output data (zeros already removed in W2validation)
Discharge.Temp_no0s=[W2validation{wb}.T(:,1) interp1(W2validation{wb}.T(~isnan(
```

```
    ↪ W2validation{wb}.T(:,2)),1),...
162    W2validation{wb}.T(~isnan(W2validation{wb}.T(:,2)),2),W2validation{wb}.T(:,1)
           ↪ )];
163 Discharge.Temp_no0s_smooth(:,1)=Discharge.Temp_no0s(:,1);
164 Discharge.Temp_no0s_smooth(:,2)=smooth(Discharge.Temp_no0s(:,1),Discharge.
    ↪ Temp_no0s(:,2),1);
165 Output{wb}.discharge_Temp{iter}(:,1)=interp1(Discharge.Temp_no0s_smooth(:,1),
    ↪ Discharge.Temp_no0s_smooth(:,2),timesteps);
166 %Convert to cells
167 Inputs_seq{wb}.discharge_Temp{iter*2-1} = con2seq(Inputs{wb}.discharge_Temp{iter
    ↪ }');
168 Output_seq{wb}.discharge_Temp{iter*2-1} = con2seq(Output{wb}.discharge_Temp{iter
    ↪ }');
169 clearvars i ii iii flow_variable
170
171 %% Temp validation run with turb and spill flipped
172 [~,a]= find(cellfun(@(s) ~isempty(strfind('QOT_BR1_T', s)), vars)==1);
173 [~,b]= find(cellfun(@(s) ~isempty(strfind('QOT_BR1_S', s)), vars)==1);
174 timesteps_flip=[t_all(1):(1/24):t_all(end)]';
175 Inputs{wb}.discharge_Temp_flip{iter}=Inputs{wb}.discharge_Temp{iter};
176 turb=Inputs{wb}.discharge_Temp{iter}(:,a); spill=Inputs{wb}.discharge_Temp{iter
    ↪ }(:,b);
177 Inputs{wb}.discharge_Temp_flip{iter}(size(timesteps,1)-size(timesteps_flip,1)+1:
    ↪ end,a)=spill(size(timesteps,1)-size(timesteps_flip,1)+1:end);
178 Inputs{wb}.discharge_Temp_flip{iter}(size(timesteps,1)-size(timesteps_flip,1)+1:
    ↪ end,b)=turb(size(timesteps,1)-size(timesteps_flip,1)+1:end);
179 %Smooth output data (zeros already removed in W2validation)
180 Discharge.Temp_no0s=[W2validation_flip{wb}.T(:,1) interp1(W2validation_flip{wb}.
    ↪ T(~isnan(W2validation_flip{wb}.T(:,2)),1),...
181    W2validation_flip{wb}.T(~isnan(W2validation_flip{wb}.T(:,2)),2),
           ↪ W2validation_flip{wb}.T(:,1))];
182 Discharge.Temp_no0s_smooth(:,1)=Discharge.Temp_no0s(:,1);
183 Discharge.Temp_no0s_smooth(:,2)=smooth(Discharge.Temp_no0s(:,1),Discharge.
    ↪ Temp_no0s(:,2),1);
184 Output{wb}.discharge_Temp_flip{iter}(:,1)=interp1(Discharge.Temp_no0s_smooth
    ↪ (:,1),Discharge.Temp_no0s_smooth(:,2),timesteps);
185 %Convert to cells
186 Inputs_seq{wb}.discharge_Temp{iter*2} = con2seq(Inputs{wb}.discharge_Temp_flip{
    ↪ iter}');
187 Output_seq{wb}.discharge_Temp{iter*2} = con2seq(Output{wb}.discharge_Temp_flip{
    ↪ iter}');
188 clearvars vars Discharge turb spill a b
189
190 %% Temp training
191 %Combine them all into single Input and Output cell arrays
192 Inputs_seq_mul{wb}.discharge_Temp=catsamples(Inputs_seq{wb}.discharge_Temp{:},'
    ↪ pad');
193 Output_seq_mul{wb}.discharge_Temp=catsamples(Output_seq{wb}.discharge_Temp{:},'
    ↪ pad');
194 clearvars vars i Discharge
195
196 %Train temp model - start with best DO model from before (greatest weight)
197 fprintf(['Training 5 temperature models and picking the best \n'])
198 for i=1:5
199    inputDelays = [0 1 12]; %[10:14]?
200    feedbackDelays = [1];
201    hiddenNeurons=[10];
202    narx_net{i} = narxnet(inputDelays,feedbackDelays,hiddenNeurons);
203    narx_net{i}.divideFcn = 'dividerand';
204    % The property DIVIDEMODE set to TIMESTEP means that targets are divided
205    % into training, validation and test sets according to timesteps.
206    % For a list of data division modes type: help nntype_data_division_mode
207    narx_net{i}.divideMode = 'time'; % Divide up every value
208    narx_net{i}.divideParam.trainRatio = 70/100;
```

```
    narx_net{i}.divideParam.valRatio = 15/100;
    narx_net{i}.divideParam.testRatio = 15/100;
    narx_net{i}.trainParam.min_grad = 1e-10;
    narx_net{i}.trainFcn = 'trainlm';
    narx_net{i}.trainParam.showWindow=0;
    narx_net{i}.trainParam.showCommandLine=1;
    [Xs,Xi,Ai,Ts]=preparets(narx_net{i},Inputs_seq_mul{wb}.discharge_Temp,{}, ...
       Output_seq_mul{wb}.discharge_Temp);
    [narx_net{i},~]=train(narx_net{i},Xs,Ts,Xi,Ai,'UseParallel','yes');
    narx_net_closed{i} = closeloop(narx_net{i});
    narx_net_closed{i}.trainParam.mu_max=1e14;
    [Xs,Xi,Ai,Ts]=preparets(narx_net_closed{i},Inputs_seq_mul{wb}.discharge_Temp
        ↪ ,{}, ...
       Output_seq_mul{wb}.discharge_Temp);
    [narx_net_closed{i},tr{i}]=train(narx_net_closed{i},Xs,Ts,Xi,Ai,'UseParallel'
        ↪ ,'yes');
end
for i=1:5 tr2(i)=tr{i}.best_perf; end
[~,b]=min(tr2); WQ{wb}.Temp_narx.narx_net_closed=narx_net_closed{b};
yp1= WQ{wb}.Temp_narx.narx_net_closed(Xs,Xi,Ai);
%Calculate bias & standard dev using only predictions at test timepoints
bias=cell2mat(yp1(tr{b}.testInd))-cell2mat(Ts(tr{b}.testInd)); bias=nanmean(bias
    ↪ );
allerrors=(cell2mat(yp1(tr{b}.testInd))-bias)-cell2mat(Ts(tr{b}.testInd));
allerrors=allerrors(~isnan(allerrors));
[~,sigmahat] = normfit(allerrors);
WQ{wb}.Temp_narx.bias=bias;
WQ{wb}.Temp_narx.weights=1;
WQ{wb}.Temp_narx.inputDelays=inputDelays;
WQ{wb}.Temp_narx.std_dev=sigmahat;
WQ{wb}.Temp_narx.Inputs=Inputs{wb}.discharge_Temp;
WQ{wb}.Temp_narx.Output=Output{wb}.discharge_Temp;
if isfield(WQ{wb}.Temp_narx,'train_time')
   WQ{wb}.Temp_narx=rmfield(WQ{wb}.Temp_narx,{'train_time'});
end
if isfield(WQ{wb}.Temp_narx,'Discharge_temp_no0s')
   WQ{wb}.Temp_narx=rmfield(WQ{wb}.Temp_narx,{'Discharge_temp_no0s'});
end
clearvars b Xs Xi Ai Ts tr tr2 yp1 TS bias narx_net_closed narx_net muhat
    ↪ sigmahat

clearvars timesteps timesteps_flip
```

obj_fcn.m

```
function y=obj_fcn(x_allwb,t,cost_curve_MW,MW_rating,...
   elev_soft_penalty_coeff,ELWS_targets,frequency,Q,ic_elev,...
   turbine_discharge)

% Calculates value of generation pattern over time t
%
% Inputs:
% x_allwb - hourly turbine time series (as rows for vectorizing!),
% integers between 0 and no_of_units for all waterbodies
% t  time series of JDAY values
% cost_curve_MW  2 row matrix to create step function, with 1st row
% being hours and 2nd row $/MW-hr values
% MW_rating - the fixed MW level of turbine_discharge_curve (25 MW for
% OHL)
% elev_soft_penalty_coeff - penalty coefficient for soft ending elev soft
% constraint
% ELWS_targets - target elevations for end of time period
% frequency - frequency of predictions (hourly=1/24)
```

```
% Q - all other inflows and outflows, interpolation settings,
% storage-elev curve, and tailwater curve (all in meters)
% ic_elev - initial elevation condition (m)
% turbine_discharge - turbine discharge curve at fixed MW level, with
% col 1 in meters and col 2 in cms
% Outputs:
% y  total price in $ of generation pattern

y=zeros(size(x_allwb,1),1);

%Split up rows of x to separate reservoirs
for wb=1:size(MW_rating,2)
   x{wb}=x_allwb(:,wb*(size(t,2)-1)-(size(t,2)-2):wb*(size(t,2)-1));
end
clearvars wb

for wb=1:size(MW_rating,2)

   %Calculate turbine output over 10 days
   %Multiply each turbine output by number of turbines online
   output_MW{wb}=x{wb}*MW_rating{wb}; %MW

   %Calculate total power output
   y_MWh{wb}=sum(output_MW{wb}')';
   %Calculate weighted price output
   y_dollars{wb}=cost_curve(t,output_MW{wb},cost_curve_MW{wb}');

   %Calculate deviation from ELWS_target and subtract/add penalty
   if wb==1
      %Preallocate mainstem_inflows for following wbs
      mainstem_inflows=cell(1:size(MW_rating,2));
      for i=1:size(MW_rating,2)
         mainstem_inflows{i}.t=[];
         mainstem_inflows{i}.Q=[];
      end
      clearvars i
      [turb_discharges{wb},spill_discharges{wb},HWs{wb},~,~] = ...
         activeunits_to_discharges(x{wb},t,frequency,...
         Q{wb},ic_elev{wb},turbine_discharge{wb},ELWS_targets{wb},...
         [],[]);
   else
      [turb_discharges{wb},spill_discharges{wb},HWs{wb},~,~] = ...
         activeunits_to_discharges(x{wb},t,frequency,...
         Q{wb},ic_elev{wb},turbine_discharge{wb},ELWS_targets{wb},...
         mainstem_inflows{wb}.t,mainstem_inflows{wb}.Q);
   end

   %ELWS end goal
   ELWS_goal{wb}=interp1(ELWS_targets{wb}(:,1),ELWS_targets{wb}(:,2),t(end));
   ELWS_error{wb}=HWs{wb}(:,end)-ELWS_goal{wb};
   ELWS_deduction{wb}=(ELWS_error{wb}.^2)*elev_soft_penalty_coeff(wb);

   y=y+y_dollars{wb}-ELWS_deduction{wb};

   %If we haven't reached the last reservoir, update mainstem_inflows
   if wb~=size(ic_elev,2)
      mainstem_inflows{wb+1}.t=t;
      mainstem_inflows{wb+1}.Q=bsxfun(@plus,turb_discharges{wb},spill_discharges
          ↪ {wb});
   end

end
```

obj_fcn_do.m

```
function y=obj_fcn_do(x_allwb,t,frequency,Q,ic_elev,...
   turbine_discharge,WQ,ELWS_targets,level,waterbody)

% Objective function to minimize DO constraint violation
%
% Inputs:
% x_allwb - hourly turbine time series (as rows for vectorizing!),
% integers between 0 and no_of_units for all waterbodies
% t  time series of JDAY values
% frequency - frequency of predictions (hourly=1/24)
% Q - all other inflows and outflows, interpolation settings,
% storage-elev curve, and tailwater curve (all in meters)
% ic_elev - initial elevation condition (m)
% turbine_discharge - turbine discharge curve at fixed MW level, with
% col 1 in meters and col 2 in cms
% WQ - structure containing water quality constraints and NARX models
% DO_narx - structure containing everything needed to make DO discharge
% predictions, including:
% turb_colum - column in exogenous variables with turb flows
% spill_column - column in exogenous variables with spill flows
% times - JDAY values used in training (not used)
% inputDelays - delays for exogenous inputs
% feedbackDelays - delays for prediction feedbacks
% input_variables - 2 row cell containing variable names in first
% row and column number in second. For example, 'MET_WB1'
% contains multiple columns of data but only some may be used
% for NARX predictions
% bias - bias for each trained neural network
% weights - weights for each trained neural network (sum to 1)
% narx_net_closed - neural networks
% DO_limit - lower and upper DO limits (NaN means it doesn't exist)
% DO_slack - relaxation from DO_limit (either upper or lower -
% doesn't make sense to have both)
% Temp_narx - structure containing everything needed to make temp discharge
% predictions, including:
% turb_colum - column in exogenous variables with turb flows
% spill_column - column in exogenous variables with spill flows
% times - JDAY values used in training (not used)
% inputDelays - delays for exogenous inputs
% feedbackDelays - delays for prediction feedbacks
% input_variables - 2 row cell containing variable names in first
% row and column number in second. For example, 'MET_WB1'
% contains multiple columns of data but only some may be used
% for NARX predictions
% bias - bias for each trained neural network
% weights - weights for each trained neural network (sum to 1)
% narx_net_closed - neural networks
% Temp_limit - lower and upper temp limits (NaN means it doesn't exist)
% Temp_slack - relaxation from Temp_limit (either upper or lower -
% doesn't make sense to have both)
% ELWS_targets - 2 column matrix with JDAY in col1 and elevation target
% in col2
% level - 'upper' or 'lower'
% waterbody - which waterbody we're checking the discharge DO for
% Outputs:
% y  DO constraint violation for each scenario in x

%Split up rows of x to separate reservoirs
for wb=1:waterbody
   x{wb}=x_allwb(:,wb*(size(t,2)-1)-(size(t,2)-2):wb*(size(t,2)-1));
end
clearvars wb
```

```
63
64  %Calculate headwater elevs for constraints
65  for wb=1:waterbody
66     %Calculate headwater elevs for constraints
67     if wb==1
68        mainstem_inflows{wb}.t=[];
69        mainstem_inflows{wb}.Q=[];
70        [turb_discharges{wb},spill_discharges{wb},HWs{wb},~,~] = ...
71           activeunits_to_discharges(x{wb},t,frequency,...
72           Q{wb},ic_elev{wb},turbine_discharge{wb},ELWS_targets{wb},...
73           [],[]);
74     else
75        [turb_discharges{wb},spill_discharges{wb},HWs{wb},~,~] = ...
76           activeunits_to_discharges(x{wb},t,frequency,...
77           Q{wb},ic_elev{wb},turbine_discharge{wb},ELWS_targets{wb},...
78           mainstem_inflows{wb}.t,mainstem_inflows{wb}.Q);
79     end
80     %If we haven't reached the last reservoir, update mainstem_inflows.Q (include
          ↪ both turbine + spill incoming!) and mainstem_inflows.t
81     if wb~=size(ic_elev,2)
82        mainstem_inflows{wb+1}.Q=...
83           bsxfun(@plus,turb_discharges{wb},spill_discharges{wb});
84        mainstem_inflows{wb+1}.t=t;
85     end
86  end
87
88  for wb=1:waterbody
89
90     if wb~=1
91        mainstem_inflows_temp{wb}.t=mainstem_inflows{wb}.t;
92        mainstem_inflows_temp{wb}.Q=mainstem_inflows{wb}.Q;
93        mainstem_inflows_temp{wb}.T=mainstem_inflows{wb}.T;
94        mainstem_inflows_temp{wb}.DO=mainstem_inflows{wb}.DO;
95        %Remove Nan values and interpolate for T and DO
96        for i=1:size(x{wb},1)
97           extrap_index=~isnan(mainstem_inflows_temp{wb}.T(i,:));
98           [~,c]=find(extrap_index==1); extrap_index=c(end);
99           mainstem_inflows_temp{wb}.T(i,:)=...
100             interp1(mainstem_inflows_temp{wb}.t(1,~isnan(mainstem_inflows_temp{
                    ↪ wb}.T(i,:))),...
101             mainstem_inflows_temp{wb}.T(i,~isnan(mainstem_inflows_temp{wb}.T(i
                    ↪ ,:))),...
102             mainstem_inflows_temp{wb}.t,'linear',...
103             mainstem_inflows_temp{wb}.T(i,extrap_index));
104          mainstem_inflows_temp{wb}.DO(i,:)=...
105             interp1(mainstem_inflows_temp{wb}.t(1,~isnan(mainstem_inflows_temp{
                    ↪ wb}.DO(i,:))),...
106             mainstem_inflows_temp{wb}.DO(i,~isnan(mainstem_inflows_temp{wb}.DO(i
                    ↪ ,:))),...
107             mainstem_inflows_temp{wb}.t,'linear',...
108             mainstem_inflows_temp{wb}.DO(i,extrap_index));
109          clearvars extrap_index c
110       end
111       clearvars i
112    end
113
114    %Discharge Temp estimation, to update incoming mainstem temp for next
          ↪ waterbody discharge DO estimation
115    Temp_narx=WQ{wb}.Temp_narx;
116    if wb==1 & waterbody~=1
117       Temp_pred{wb}=...
118          narx_predictions(Temp_narx,frequency,t,Q{wb},x{wb},...
119          turb_discharges{wb},spill_discharges{wb},[],...
120          Q{wb}.TWO,'temp');
```

```
    elseif wb~=1 & wb~=waterbody
       Temp_pred{wb}=narx_predictions(Temp_narx,frequency,t,Q{wb},x{wb},...
          turb_discharges{wb},spill_discharges{wb},...
          mainstem_inflows_temp{wb},Q{wb}.TWO,'temp');
    end
    %If we haven't reached the last reservoir, update mainstem_inflows.T
    if wb~=waterbody
       mainstem_inflows{wb+1}.T(1:size(x{wb},1),1)=...
          interp1(Q{wb}.TWO(:,1),Q{wb}.TWO(:,2),t(1));
       mainstem_inflows{wb+1}.T(:,2:size(Temp_pred{wb},2)+1)=...
          Temp_pred{wb};
    end

    %Now move on to DO....
    DO_narx=WQ{wb}.DO_narx; DO_limit=WQ{wb}.DO_limit;
    if wb==1
       DO_pred{wb}=narx_predictions(DO_narx,frequency,t,Q{wb},x{wb},...
          turb_discharges{wb},spill_discharges{wb},[],...
          Q{wb}.CWO,'do');
    else
       DO_pred{wb}=narx_predictions(DO_narx,frequency,t,Q{wb},x{wb},...
          turb_discharges{wb},spill_discharges{wb},...
          mainstem_inflows_temp{wb},Q{wb}.CWO,'do');
    end
    %If we haven't reached the last reservoir, update mainstem_inflows.DO
    if wb~=waterbody
       mainstem_inflows{wb+1}.DO(1:size(x{wb},1),1)=...
          interp1(Q{wb}.CWO(:,1),Q{wb}.CWO(:,2),t(1));
       mainstem_inflows{wb+1}.DO(:,2:size(DO_pred{wb},2)+1)=...
          DO_pred{wb};
    else
       non_nan_count=sum(~isnan(DO_pred{wb}),2);
       if strcmp(level,'lower')
          %DO violations - lower
          if isnan(DO_limit(1))
             DO_violations=zeros(size(DO_pred{wb},1),1);
          else
             DO_violations=sum(-min(0,DO_pred{wb}-DO_limit(1)),2)./non_nan_count;
          end
       elseif strcmp(level,'upper')
          %DO violations - upper
          if isnan(DO_limit(2))
             DO_violations=zeros(size(DO_pred{wb},1),1);
          else
             DO_violations=sum(max(0,DO_pred{wb}-DO_limit(2)),2)./non_nan_count;
          end
       end

       y=max(DO_violations,[],2);
    end
end
```

obj_fcn_elev.m

```
function y=obj_fcn_elev(x_allwb,t,frequency,Q,ic_elev,...
   turbine_discharge,ELWS_limit,ELWS_targets,level,waterbody)

% Objective function to minimize elevation constraint violation
%
% Inputs:
% x_allwb - hourly turbine time series (as rows for vectorizing!),
% integers between 0 and no_of_units for all waterbodies
% t  time series of JDAY values
```

```
% frequency - frequency of predictions (hourly=1/24)
% Q - all other inflows and outflows, interpolation settings,
% storage-elev curve, and tailwater curve (all in meters)
% ic_elev - initial elevation condition (m)
% turbine_discharge - turbine discharge curve at fixed MW level, with
% col 1 in meters and col 2 in cms
% ELWS_limit - min and max elevation limits for constraints, in meters
% ELWS_targets - 2 column matrix with JDAY in col1 and elevation target
% in col2
% level - 'upper' or 'lower'
% waterbody - which waterbody we're checking elevation for
% Outputs:
% y  elevation constraint violation for each scenario in x

%Split up rows of x to separate reservoirs
for wb=1:waterbody
   x{wb}=x_allwb(:,wb*(size(t,2)-1)-(size(t,2)-2):wb*(size(t,2)-1));
end
clearvars wb

for wb=1:waterbody
   %Calculate headwater elevs for constraints
   if wb==1
      mainstem_inflows{wb}.t=[];
      mainstem_inflows{wb}.Q=[];
      [turb_discharges{wb},spill_discharges{wb},HWs{wb},~,~] = ...
         activeunits_to_discharges(x{wb},t,frequency,...
         Q{wb},ic_elev{wb},turbine_discharge{wb},ELWS_targets{wb},...
         [],[]);
   else
      [turb_discharges{wb},spill_discharges{wb},HWs{wb},~,~] = ...
         activeunits_to_discharges(x{wb},t,frequency,...
         Q{wb},ic_elev{wb},turbine_discharge{wb},ELWS_targets{wb},...
         mainstem_inflows{wb}.t,mainstem_inflows{wb}.Q);
   end
   %If we haven't reached the last reservoir, update mainstem_inflows.Q (include
      ↪ both turbine + spill incoming!) and mainstem_inflows.t
   if wb~=size(ic_elev,2)
      mainstem_inflows{wb+1}.Q=...
         bsxfun(@plus,turb_discharges{wb},spill_discharges{wb});
      mainstem_inflows{wb+1}.t=t;
   end
end

%Inequality constraints:
if strcmp(level,'lower')
   %Elevation violations - lower
   if isnan(ELWS_limit(1))
      deductions=zeros(size(HWs{waterbody}(:,1:end)));
   else
      deductions=-min(0,HWs{waterbody}(:,1:end)-ELWS_limit(1));
   end
elseif strcmp(level,'upper')
   %Elevation violations - upper
   if isnan(ELWS_limit(2))
      deductions=zeros(size(HWs{waterbody}(:,1:end)));
   else
      deductions=max(0,HWs{waterbody}(:,1:end)-ELWS_limit(2));
   end
end

y=max(deductions,[],2);
```

obj_fcn_penalty_dollars.m

```
function [penalty,dollars,ELWS_error2]=obj_fcn_penalty_dollars(x_allwb,t,
    ↪ cost_curve_MW,MW_rating,...
   elev_soft_penalty_coeff,ELWS_targets,frequency,Q,ic_elev,...
   turbine_discharge)

% Calculates value of generation pattern over time t
%
% Inputs:
% x_allwb - hourly turbine time series (as rows for vectorizing!),
% integers between 0 and no_of_units for all waterbodies
% t  time series of JDAY values
% cost_curve_MW  2 row matrix to create step function, with 1st row
% being hours and 2nd row $/MW-hr values
% MW_rating - the fixed MW level of turbine_discharge_curve (25 MW for
% OHL)
% elev_soft_penalty_coeff - penalty coefficient for soft ending elev soft
% constraint
% ELWS_targets - target elevations for end of time period
% frequency - frequency of predictions (hourly=1/24)
% Q - all other inflows and outflows, interpolation settings,
% storage-elev curve, and tailwater curve (all in meters)
% ic_elev - initial elevation condition (m)
% turbine_discharge - turbine discharge curve at fixed MW level, with
% col 1 in meters and col 2 in cms
% Outputs:
% penalty - penalty amount
% dollars - total price in $ of generation pattern
% ELWS_error2  how far elevation is from target


%Split up rows of x to separate reservoirs
for wb=1:size(MW_rating,2)
   x{wb}=x_allwb(:,wb*(size(t,2)-1)-(size(t,2)-2):wb*(size(t,2)-1));
end
clearvars wb

for wb=1:size(MW_rating,2)

   %Calculate turbine output over 10 days
   %Multiply each turbine output by number of turbines online
   output_MW{wb}=x{wb}*MW_rating{wb}; %MW

   %Calculate total power output
   y_MWh{wb}=sum(output_MW{wb}')';
   %Calculate weighted price output
   y_dollars{wb}=cost_curve(t,output_MW{wb},cost_curve_MW{wb}');

   %Calculate deviation from ELWS_target and subtract/add penalty
   if wb==1
      %Preallocate mainstem_inflows for following wbs
      mainstem_inflows=cell(1:size(MW_rating,2));
      for i=1:size(MW_rating,2)
         mainstem_inflows{i}.t=[];
         mainstem_inflows{i}.Q=[];
      end
      clearvars i
      [turb_discharges{wb},spill_discharges{wb},HWs{wb},~,~] = ...
         activeunits_to_discharges(x{wb},t,frequency,...
         Q{wb},ic_elev{wb},turbine_discharge{wb},ELWS_targets{wb},...
         [],[]);
   else
      [turb_discharges{wb},spill_discharges{wb},HWs{wb},~,~] = ...
```

```
        activeunits_to_discharges(x{wb},t,frequency,...
        Q{wb},ic_elev{wb},turbine_discharge{wb},ELWS_targets{wb},...
        mainstem_inflows{wb}.t,mainstem_inflows{wb}.Q);
   end

   %ELWS end goal
   if size(ELWS_targets{wb},1)>1
      ELWS_goal{wb}=interp1(ELWS_targets{wb}(:,1),ELWS_targets{wb}(:,2),t(end));
   else
      ELWS_goal{wb}=ELWS_targets{wb}(:,2);
   end
   ELWS_error{wb}=HWs{wb}(:,end)-ELWS_goal{wb};
   ELWS_error2=ELWS_error{wb}; ELWS_error{wb}(ELWS_error{wb}>0)=0;
   ELWS_deduction{wb}=(ELWS_error{wb}.^2)*elev_soft_penalty_coeff(wb);

   %If we haven't reached the last reservoir, update mainstem_inflows
   if wb~=size(ic_elev,2)
      mainstem_inflows{wb+1}.t=t;
      mainstem_inflows{wb+1}.Q=bsxfun(@plus,turb_discharges{wb},spill_discharges
          ↪ {wb});
   end

end

penalty=0; dollars=0;
for wb=1:size(MW_rating,2)
   dollars=dollars+y_dollars{wb};
   penalty=penalty+ELWS_deduction{wb};
end
```

obj_fcn_temp.m

```
function y=obj_fcn_temp(x_allwb,t,frequency,Q,ic_elev,...
   turbine_discharge,WQ,ELWS_targets,level,waterbody)

% Objective function to minimize temp constraint violation
%
% Inputs:
% x - hourly turbine time series (as rows for vectorizing!), integers
% between 0 and no_of_units
% t  time series of JDAY values
% frequency - frequency of predictions (hourly=1/24)
% Q - all other inflows and outflows, interpolation settings,
% storage-elev curve, and tailwater curve (all in meters)
% ic_elev - initial elevation condition (m)
% turbine_discharge - turbine discharge curve at fixed MW level, with
% col 1 in meters and col 2 in cms
% WQ - structure containing water quality constraints and NARX models
% DO_narx - structure containing everything needed to make DO discharge
% predictions, including:
% turb_colum - column in exogenous variables with turb flows
% spill_column - column in exogenous variables with spill flows
% times - JDAY values used in training (not used)
% inputDelays - delays for exogenous inputs
% feedbackDelays - delays for prediction feedbacks
% input_variables - 2 row cell containing variable names in first
% row and column number in second. For example, 'MET_WB1'
% contains multiple columns of data but only some may be used
% for NARX predictions
% bias - bias for each trained neural network
% weights - weights for each trained neural network (sum to 1)
% narx_net_closed - neural networks
% DO_limit - lower and upper DO limits (NaN means it doesn't exist)
```

```
% DO_slack - relaxation from DO_limit (either upper or lower -
% doesn't make sense to have both)
% Temp_narx - structure containing everything needed to make temp discharge
% predictions, including:
% turb_colum - column in exogenous variables with turb flows
% spill_column - column in exogenous variables with spill flows
% times - JDAY values used in training (not used)
% inputDelays - delays for exogenous inputs
% feedbackDelays - delays for prediction feedbacks
% input_variables - 2 row cell containing variable names in first
% row and column number in second. For example, 'MET_WB1'
% contains multiple columns of data but only some may be used
% for NARX predictions
% bias - bias for each trained neural network
% weights - weights for each trained neural network (sum to 1)
% narx_net_closed - neural networks
% Temp_limit - lower and upper temp limits (NaN means it doesn't exist)
% Temp_slack - relaxation from Temp_limit (either upper or lower -
% doesn't make sense to have both)
% ELWS_targets - 2 column matrix with JDAY in col1 and elevation target
% in col2
% level - 'upper' or 'lower'
% waterbody - which waterbody we're checking the discharge temp for
% Outputs:
% y  temp constraint violation for each scenario in x

%Split up rows of x to separate reservoirs
for wb=1:waterbody
   x{wb}=x_allwb(:,wb*(size(t,2)-1)-(size(t,2)-2):wb*(size(t,2)-1));
end
clearvars wb

%Calculate headwater elevs for constraints
for wb=1:waterbody
   %Calculate headwater elevs for constraints
   if wb==1
      mainstem_inflows{wb}.t=[];
      mainstem_inflows{wb}.Q=[];
      [turb_discharges{wb},spill_discharges{wb},HWs{wb},~,~] = ...
         activeunits_to_discharges(x{wb},t,frequency,...
         Q{wb},ic_elev{wb},turbine_discharge{wb},ELWS_targets{wb},...
         [],[]);
   else
      [turb_discharges{wb},spill_discharges{wb},HWs{wb},~,~] = ...
         activeunits_to_discharges(x{wb},t,frequency,...
         Q{wb},ic_elev{wb},turbine_discharge{wb},ELWS_targets{wb},...
         mainstem_inflows{wb}.t,mainstem_inflows{wb}.Q);
   end
   %If we haven't reached the last reservoir, update mainstem_inflows.Q (include
      ↪ both turbine + spill incoming!) and mainstem_inflows.t
   if wb~=size(ic_elev,2)
      mainstem_inflows{wb+1}.Q=...
         bsxfun(@plus,turb_discharges{wb},spill_discharges{wb});
      mainstem_inflows{wb+1}.t=t;
   end
end


for wb=1:waterbody

   if wb~=1
      mainstem_inflows_temp{wb}.t=mainstem_inflows{wb}.t;
      mainstem_inflows_temp{wb}.Q=mainstem_inflows{wb}.Q;
      mainstem_inflows_temp{wb}.T=mainstem_inflows{wb}.T;
```

```
95        %Remove Nan values and interpolate for T
96        for i=1:size(x{wb},1)
97           extrap_index=~isnan(mainstem_inflows_temp{wb}.T(i,:));
98           [~,c]=find(extrap_index==1); extrap_index=c(end);
99           mainstem_inflows_temp{wb}.T(i,:)=...
100             interp1(mainstem_inflows_temp{wb}.t(1,~isnan(mainstem_inflows_temp{
                    ↪ wb}.T(i,:))),...
101             mainstem_inflows_temp{wb}.T(i,~isnan(mainstem_inflows_temp{wb}.T(i
                    ↪ ,:))),...
102             mainstem_inflows_temp{wb}.t,'linear',... %'extrap');
103             mainstem_inflows_temp{wb}.T(i,extrap_index));
104          clearvars extrap_index c
105       end
106       clearvars i
107    end
108
109    %Discharge Temp estimation
110    Temp_narx=WQ{wb}.Temp_narx; Temp_limit=WQ{wb}.Temp_limit;
111    if wb==1
112       Temp_pred{wb}=narx_predictions(Temp_narx,frequency,t,Q{wb},x{wb},...
113          turb_discharges{wb},spill_discharges{wb},[],...
114          Q{wb}.TWO,'temp');
115    else
116       Temp_pred{wb}=narx_predictions(Temp_narx,frequency,t,Q{wb},x{wb},...
117          turb_discharges{wb},spill_discharges{wb},...
118          mainstem_inflows_temp{wb},Q{wb}.TWO,'temp');
119    end
120    %If we haven't reached the last reservoir, update mainstem_inflows.T
121    if wb~=waterbody
122       mainstem_inflows{wb+1}.T(1:size(x{wb},1),1)=...
123          interp1(Q{wb}.TWO(:,1),Q{wb}.TWO(:,2),t(1));
124       mainstem_inflows{wb+1}.T(:,2:size(Temp_pred{wb},2)+1)=...
125          Temp_pred{wb};
126    else
127       non_nan_count=sum(~isnan(Temp_pred{wb}),2);
128       if strcmp(level,'lower')
129          %Temp violations - lower
130          if isnan(Temp_limit(1))
131             Temp_violations=zeros(size(Temp_pred{wb},1),1);
132          else
133             Temp_violations=sum(-min(0,Temp_pred{wb}-Temp_limit(1)),2)./
                    ↪ non_nan_count;
134          end
135       elseif strcmp(level,'upper')
136          %Temp violations - upper
137          if isnan(Temp_limit(2))
138             Temp_violations=zeros(size(Temp_pred{wb},1),1);
139          else
140             Temp_violations=sum(max(0,Temp_pred{wb}-Temp_limit(2)),2)./
                    ↪ non_nan_count;
141          end
142       end
143
144       y=max(Temp_violations,[],2);
145    end
146 end
```

penalty_fcn.m

```
1  function [c_all,ceq]=penalty_fcn(x_allwb,t,frequency,Q,ic_elev,...
2     turbine_discharge,ELWS_limit,max_hrly_unit_change,...
3     WQ,zero_gen_limit,xprev,ELWS_targets,tolerance)
4
```

```
% Calculates penalty violations, starting with the least expensive
% computations and continuing on to the more expensive computations for
% runs that are found to be feasible thus far
%
% Inputs:
% x_allwb - hourly turbine time series (as rows for vectorizing!),
% integers between 0 and no_of_units for all waterbodies
% t  time series of JDAY values
% frequency - frequency of predictions (hourly=1/24)
% Q - all other inflows and outflows, interpolation settings,
% storage-elev curve, and tailwater curve
% ic_elev - initial condition (meters)
% turbine_discharge - turbine discharge curve at fixed MW level, with
% col 1 in meters and col 2 in cms
% ELWS_limit - min and max elevation limits for constraints, in meters
% max_hrly_unit_change - max number of units that can be changed per hour
% (1 for OHL)
% WQ - structure containing water quality constraints and NARX models
% DO_narx - structure containing everything needed to make DO discharge
% predictions, including:
% turb_colum - column in exogenous variables with turb flows
% spill_column - column in exogenous variables with spill flows
% times - JDAY values used in training (not used)
% inputDelays - delays for exogenous inputs
% feedbackDelays - delays for prediction feedbacks
% input_variables - 2 row cell containing variable names in first
% row and column number in second. For example, 'MET_WB1'
% contains multiple columns of data but only some may be used
% for NARX predictions
% bias - bias for each trained neural network
% weights - weights for each trained neural network (sum to 1)
% narx_net_closed - neural networks
% DO_limit - lower and upper DO limits (NaN means it doesn't exist)
% DO_slack - relaxation from DO_limit (either upper or lower -
% doesn't make sense to have both)
% Temp_narx - structure containing everything needed to make temp discharge
% predictions, including:
% turb_colum - column in exogenous variables with turb flows
% spill_column - column in exogenous variables with spill flows
% times - JDAY values used in training (not used)
% inputDelays - delays for exogenous inputs
% feedbackDelays - delays for prediction feedbacks
% input_variables - 2 row cell containing variable names in first
% row and column number in second. For example, 'MET_WB1'
% contains multiple columns of data but only some may be used
% for NARX predictions
% bias - bias for each trained neural network
% weights - weights for each trained neural network (sum to 1)
% narx_net_closed - neural networks
% Temp_limit - lower and upper temp limits (NaN means it doesn't exist)
% Temp_slack - relaxation from Temp_limit (either upper or lower -
% doesn't make sense to have both)
% zero_gen_limit - Zero generation hourly limit (can't go longer than
% this with no turb flow)
% xprev - vector of previous active turbine levels
% ELWS_targets - 2 column matrix with JDAY in col1 and elevation target
% in col2
% tolerance - penalty tolerance
% Outputs:
% c_all  inequality constraint output (n/a, so 0)
% ceq - equality constraint output (=0 for feasible solution)

%Equality constraint
ceq=[];
```

```
69
70  %Preallocate memory
71  x{1,size(ic_elev,2)}=[];
72  xall{1,size(ic_elev,2)}=[];
73  turb_discharges{1,size(ic_elev,2)}=[];
74  HWs{1,size(ic_elev,2)}=[];
75  c_all=zeros(size(x{1},1),size(ic_elev,2)*(3+(1+size(x{1},2))*2+2+2+2+2));
76
77  zeroRows_empty=0;
78  zeroRows0=[1:size(x_allwb,1)]';
79
80  for wb=1:size(ic_elev,2)
81     %Split up rows of x to separate reservoirs
82     x{wb}=x_allwb(:,wb*(size(t,2)-1)-(size(t,2)-2):wb*(size(t,2)-1));
83     %Preallocate c, with columns representing: (1) change in active unit
           ↪ violations, (2) zero gen hourly limit, (3) oscillations constraint,
           ↪ (4:28) ELWS lower violations, (29:53) ELWS upper violations, (54:77)
           ↪ hrly DO upper violations, (78:101) hrly DO lower violations, (102)
           ↪ mean DO upper violation, (103) mean DO lower violations, (104:127)
           ↪ hrly temp upper violations, (128:151) hrly temp lower violations,
           ↪ (152) mean temp upper violation, and (153) mean temp lower violations
84     c{wb}=zeros(size(x{1},1),3+(1+size(x{1},2))*2+2+2+2+2);
85  end
86  clearvars wb
87
88
89
90  for wb=1:size(ic_elev,2)
91
92     c{wb}(setdiff([1:size(x{wb},1)],zeroRows0),:)=1;
93
94     %Check if all entries in x are infeasible due to previous reservoirs, and if
           ↪ so set the rest of c==1 and go to end
95     if zeroRows_empty==1
96        c{wb}(:)=1;
97     else
98
99        %Break up WQ structure into separate variables
100       DO_narx=WQ{wb}.DO_narx; DO_limit=WQ{wb}.DO_limit; DO_slack=WQ{wb}.DO_slack
              ↪ ;
101       Temp_narx=WQ{wb}.Temp_narx; Temp_limit=WQ{wb}.Temp_limit; Temp_slack=WQ{wb
              ↪ }.Temp_slack;
102
103       %Stitch together xprev & x to check for feasibility wrt active unit viols,
              ↪  zero generation hrly limit, and oscillations
104       xall{wb}=[repmat(xprev{wb},size(x{wb},1),1) x{wb}];
105
106       %Change in active unit violations
107       if isempty(max_hrly_unit_change{wb})
108          delta_sum=zeros(size(zeroRows0,1),1);
109       else
110          delta=abs(round(xall{wb}(zeroRows0,2:end))-...
111             round(xall{wb}(zeroRows0,1:end-1)));
112          index=find(delta<=max_hrly_unit_change{wb});
113          delta(index)=0;
114          delta_sum=sum(delta')';
115       end
116
117       %Zero generation hourly limit - can't go longer with no turb flow
118       if isempty(zero_gen_limit{wb})
119          zero_gen_viols_sum=zeros(size(zeroRows0,1),1);
120       else
121          zero_gen_viols=zeros(size(zeroRows0,1),size(xall{wb},2)-...
122             zero_gen_limit{wb}-1);
```

```
123             x_trans=xall{wb}(zeroRows0,:)';
124             for i=1:size(x_trans,1)-zero_gen_limit{wb}
125                 a=sum(x_trans(i:i+zero_gen_limit{wb},:))';
126                 zero_gen_viols(:,i)=(a==0);
127             end
128             clearvars i
129             zero_gen_viols_sum=sum(zero_gen_viols')';
130         end
131
132         %Oscillations constraint - violates whenever the number of turbines
                ↪ increases and then decreases within 3 hours, or vice versa
133         osc_violations=zeros(size(zeroRows0,1),size(xall{wb},2)-2);
134         xall_osc=xall{wb}(zeroRows0,:);
135         for ii=1:size(xall_osc,1) %loop through each member of population
136             for i=1:size(xall_osc,2)-2; %loop forward through time
137                 if xall_osc(ii,i+1)>xall_osc(ii,i) & ...
138                         xall_osc(ii,i+2)<xall_osc(ii,i+1)
139                     osc_violations(ii,i)=1;
140                 elseif xall_osc(ii,i+1)<xall_osc(ii,i) & ...
141                         xall_osc(ii,i+2)>xall_osc(ii,i+1)
142                     osc_violations(ii,i)=1;
143                 elseif i~=1
144                     if xall_osc(ii,i)==xall_osc(ii,i+1) %need 3 hrs btwn ramping up
                            ↪ and down
145                         if xall_osc(ii,i-1)<xall_osc(ii,i) & ...
146                                 xall_osc(ii,i+1)>xall_osc(ii,i+2) %ramping up & back
                                    ↪ down too quickly
147                             osc_violations(ii,i)=1;
148                         elseif xall_osc(ii,i-1)>xall_osc(ii,i) & ...
149                                 xall_osc(ii,i+1)<xall_osc(ii,i+2) %ramping down & back
                                    ↪ up too quickly
150                             osc_violations(ii,i)=1;
151                         end
152                     end
153                 end
154             end
155         end
156         clearvars i ii xall_osc
157         osc_violations_sum=sum(osc_violations')';
158
159         %Compile least expensive constraints
160         c{wb}(zeroRows0,1:3)=...
161             [delta_sum zero_gen_viols_sum osc_violations_sum];
162
163         clearvars zeroRows1 zeroRows2 zeroRows3 zeroRows4 x_zeroRows1 x_zeroRows2
                ↪ x_zeroRows3 x_zeroRows4
164         x_zeroRows1=[];
165         x_zeroRows2=[];
166         x_zeroRows3=[];
167         x_zeroRows4=[];
168         %Only compute expensive constraints if all others pass
169         zeroRows1=find(all(c{wb}<=tolerance,2));
170         x_zeroRows1=x{wb}(zeroRows1,:);
171         if isempty(x_zeroRows1)
172             c{wb}(:,4:end)=1;
173             zeroRows_empty=1;
174         end
175
176         if zeroRows_empty~=1
177
178             %Calculate headwater elevs for constraints
179             if wb==1
180                 %Preallocate mainstem_inflows for following wbs
181                 mainstem_inflows=cell(1:size(ic_elev,2));
```

```
182             for i=1:size(ic_elev,2)
183                mainstem_inflows{i}.t=[];
184                mainstem_inflows{i}.Q=[];
185                mainstem_inflows{i}.T=[];
186                mainstem_inflows{i}.DO=[];
187             end
188             clearvars i
189             [turb_discharges{wb},spill_discharges{wb},HWs{wb},~,~] = ...
190                activeunits_to_discharges(x_zeroRows1,t,...
191                frequency,Q{wb},ic_elev{wb},turbine_discharge{wb},...
192                ELWS_targets{wb},[],[]);
193          else
194             [turb_discharges{wb},spill_discharges{wb},HWs{wb},~,~] = ...
195                activeunits_to_discharges(x_zeroRows1,t,...
196                frequency,Q{wb},ic_elev{wb},turbine_discharge{wb},...
197                ELWS_targets{wb},mainstem_inflows{wb}.t,...
198                mainstem_inflows{wb}.Q(zeroRows1,:));
199          end
200          %If we haven't reached the last reservoir, update mainstem_inflows.Q (
                 ↪ include both turbine + spill incoming!)
201          if wb~=size(ic_elev,2)
202             mainstem_inflows{wb+1}.Q(zeroRows1,:)=...
203             bsxfun(@plus,turb_discharges{wb},spill_discharges{wb});
204          end
205          %Inequality constraints:
206          %Elevation violations - lower
207          if isnan(ELWS_limit{wb}(1))
208             deductions1=zeros(size(HWs{wb}(:,1:end)));
209          else
210             deductions1=-min(0,HWs{wb}(:,1:end)-ELWS_limit{wb}(1));
211          end
212          %Elevation violations - upper
213          if isnan(ELWS_limit{wb}(2))
214             deductions2=zeros(size(HWs{wb}(:,1:end)));
215          else
216             deductions2=max(0,HWs{wb}(:,1:end)-ELWS_limit{wb}(2));
217          end
218
219          c{wb}(setdiff([1:size(x{wb},1)],zeroRows1),4:end)=1;
220          c{wb}(zeroRows1,4:3+(1+size(x{wb},2))*2)=[deductions1 deductions2];
221
222          zeroRows2=find(all(c{wb}<=tolerance,2));
223          x_zeroRows2=x{wb}(zeroRows2,:);
224          if isempty(x_zeroRows2)
225             c{wb}(:,3+(1+size(x{wb},2))*2+1:end)=1;
226             zeroRows_empty=1;
227          end
228
229          turb_discharges2=zeros(size(x{wb},1),size(x{wb},2)+1);
230          turb_discharges2(zeroRows1,:)=turb_discharges{wb};
231          %-->need to reset this with zero rows back in
232          turb_discharges{wb}=turb_discharges2;
233          spill_discharges2=zeros(size(x{wb},1),1);
234          spill_discharges2(zeroRows1,:)=spill_discharges{wb};
235          %-->need to reset this with zero rows back in
236          spill_discharges{wb}=spill_discharges2;
237          clearvars spill_discharges2 turb_discharges2
238       end
239
240       %Continue on and calculate discharge DO if still feasible, if DO_narx is
             ↪ provided and a limit exists
241       if zeroRows_empty~=1 & ~isempty(DO_narx) & (wb~=size(ic_elev,2) | any(
             ↪ DO_limit))
242
```

```
243         %Discharge DO constraint
244         if wb==1
245            DO_pred{wb}=narx_predictions(DO_narx,...
246               frequency,t,Q{wb},x_zeroRows2,...
247               turb_discharges{wb}(zeroRows2,:),...
248               spill_discharges{wb}(zeroRows2),[],...
249               Q{wb}.CWO,'do');
250         else
251            mainstem_inflows_zeroRows2{wb}.Q=...
252               mainstem_inflows{wb}.Q(zeroRows2,:);
253            mainstem_inflows_zeroRows2{wb}.T=...
254               mainstem_inflows{wb}.T(zeroRows2,:);
255            mainstem_inflows_zeroRows2{wb}.DO=...
256               mainstem_inflows{wb}.DO(zeroRows2,:);
257            DO_pred{wb}=narx_predictions(DO_narx,...
258               frequency,t,Q{wb},x_zeroRows2,...
259               turb_discharges{wb}(zeroRows2,:),...
260               spill_discharges{wb}(zeroRows2),...
261               mainstem_inflows_zeroRows2{wb},Q{wb}.CWO,'do');
262         end
263         %If we haven't reached the last reservoir, update mainstem_inflows.DO
264         if wb~=size(ic_elev,2)
265            mainstem_inflows{wb+1}.DO(zeroRows2,1)=...
266               interp1(Q{wb}.CWO(:,1),Q{wb}.CWO(:,2),t(1));
267            mainstem_inflows{wb+1}.DO(zeroRows2,2:size(DO_pred{wb},2)+1)=...
268               DO_pred{wb};
269         end
270         non_nan_count=sum(~isnan(DO_pred{wb}),2);
271         %DO violations - lower
272         if isnan(DO_limit(1))
273            DO_violations1=zeros(size(DO_pred{wb},1),1);
274         else
275            DO_violations1=sum(-min(0,DO_pred{wb}-DO_limit(1)),2)./non_nan_count
                ↪ ;
276         end
277         %DO violations - upper
278         if isnan(DO_limit(2))
279            DO_violations2=zeros(size(DO_pred{wb},1),1);
280         else
281            DO_violations2=sum(max(0,DO_pred{wb}-DO_limit(2)),2)./non_nan_count;
282         end
283         DO_violations=[max(0,DO_violations1-DO_slack) max(0,DO_violations2-
              ↪ DO_slack)];
284
285         c{wb}(setdiff([1:size(x{wb},1)],zeroRows2),3+(1+size(x{wb},2))*2+1:end)
              ↪ =1;
286         c{wb}(zeroRows2,3+(1+size(x{wb},2))*2+1:3+(1+size(x{wb},2))*2+2)=
              ↪ DO_violations;
287         clearvars DO_violations1 DO_violations2 Last_values
288
289         zeroRows3=find(all(c{wb}<=tolerance,2));
290         x_zeroRows3=x{wb}(zeroRows3,:);
291         DO_pred{wb}(zeroRows2,:)=DO_pred{wb};
292         DO_pred{wb}=DO_pred{wb}(zeroRows3,:);
293         if isempty(x_zeroRows3)
294            c{wb}(:,3+(1+size(x{wb},2))*2+2+1:end)=1;
295            zeroRows_empty=1;
296         end
297
298      end
299
300      %Continue on and calculate discharge temp if still feasible
301      if zeroRows_empty~=1 & ~isempty(Temp_narx) & (wb~=size(ic_elev,2) | any(
           ↪ Temp_limit))
```

```
302             zeroRows4=find(all(c{wb}<=tolerance,2));
303             x_zeroRows4=x{wb}(zeroRows4,:);
304             if isempty(x_zeroRows4)
305                c{wb}(:,3+(1+size(x{wb},2))*2+2+2+1:end)=1;
306                zeroRows_empty=1;
307             end
308
309             if zeroRows_empty~=1
310                %Discharge Temp constraint
311                if wb==1
312                   Temp_pred{wb}=...
313                      narx_predictions(Temp_narx,...
314                      frequency,t,Q{wb},x_zeroRows4,...
315                      turb_discharges{wb}(zeroRows4,:),...
316                      spill_discharges{wb}(zeroRows4,:),[],...
317                      Q{wb}.TWO,'temp');
318                else
319                   mainstem_inflows_zeroRows4{wb}.Q=...
320                      mainstem_inflows{wb}.Q(zeroRows4,:);
321                   mainstem_inflows_zeroRows4{wb}.T=...
322                      mainstem_inflows{wb}.T(zeroRows4,:);
323                   mainstem_inflows_zeroRows4{wb}.DO=...
324                      mainstem_inflows{wb}.DO(zeroRows4,:);
325                   Temp_pred{wb}=...
326                      narx_predictions(Temp_narx,...
327                      frequency,t,Q{wb},x_zeroRows4,...
328                      turb_discharges{wb}(zeroRows4,:),...
329                      spill_discharges{wb}(zeroRows4,:),...
330                      mainstem_inflows_zeroRows4{wb},...
331                      Q{wb}.TWO,'temp');
332                end
333                %If we haven't reached the last reservoir, update mainstem_inflows.T
334                if wb~=size(ic_elev,2)
335                   mainstem_inflows{wb+1}.T(zeroRows3,1)=...
336                      interp1(Q{wb}.TWO(:,1),Q{wb}.TWO(:,2),t(1));
337                   mainstem_inflows{wb+1}.T(zeroRows3,2:size(Temp_pred{wb},2)+1)=...
338                      Temp_pred{wb};
339                end
340                non_nan_count=sum(~isnan(Temp_pred{wb}),2);
341                %Temp violations - lower
342                if isnan(Temp_limit(1))
343                   Temp_violations1=zeros(size(Temp_pred{wb},1),1);
344                else
345                   Temp_violations1=sum(-min(0,Temp_pred{wb}-Temp_limit(1)),2)./
                       ↪ non_nan_count;
346                end
347                %Temp violations - upper
348                if isnan(Temp_limit(2))
349                   Temp_violations2=zeros(size(Temp_pred{wb},1),1);
350                else
351                   Temp_violations2=sum(max(0,Temp_pred{wb}-Temp_limit(2)),2)./
                       ↪ non_nan_count;
352                end
353                Temp_violations=[max(0,Temp_violations1-Temp_slack) max(0,
                    ↪ Temp_violations2-Temp_slack)];
354
355                c{wb}(setdiff([1:size(x{wb},1)],zeroRows4),3+(1+size(x{wb},2))
                    ↪ *2+2+2+1:end)=1;
356                c{wb}(zeroRows4,3+(1+size(x{wb},2))*2+2+2+1:3+(1+size(x{wb},2))
                    ↪ *2+2+2+2)=Temp_violations;
357
358                zeroRows5=find(all(c{wb}<=tolerance,2));
359                x_zeroRows5=x{wb}(zeroRows5,:);
360                Temp_pred{wb}(zeroRows4,:)=Temp_pred{wb};
```

```
            Temp_pred{wb}=Temp_pred{wb}(zeroRows5,:);
            if isempty(x_zeroRows5)
               c{wb}(:,3+(1+size(x{wb},2))*2+2+2+2+1:end)=1;
               zeroRows_empty=1;
            end

         end
      end
   end
   %If we haven't reached the last reservoir, update mainstem_inflows.t, remove NaN from mainstem_inflows.T and mainstem_inflows.DO, and update zeroRows0
   if wb~=size(ic_elev,2) & zeroRows_empty~=1
      mainstem_inflows{wb+1}.t=t;
      %Remove Nan values and interpolate for T and DO
      for i=1:size(mainstem_inflows{wb+1}.T,1)
         extrap_index=~isnan(mainstem_inflows{wb+1}.T(i,:));
         [~,column]=find(extrap_index==1); extrap_index=column(end);
         mainstem_inflows{wb+1}.T(i,:)=...
            interp1(t(1,~isnan(mainstem_inflows{wb+1}.T(i,:))),...
            mainstem_inflows{wb+1}.T(i,~isnan(mainstem_inflows{wb+1}.T(i,:))),...
            t,'linear',mainstem_inflows{wb+1}.T(i,extrap_index));
         mainstem_inflows{wb+1}.DO(i,:)=...
            interp1(t(1,~isnan(mainstem_inflows{wb+1}.DO(i,:))),...
            mainstem_inflows{wb+1}.DO(i,~isnan(mainstem_inflows{wb+1}.DO(i,:))),...
            t,'linear',mainstem_inflows{wb+1}.DO(i,extrap_index));
         clearvars extrap_index column
      end
      zeroRows0=find(all(c{wb}<=tolerance,2));
   end

end

%Update c_all with the values from c{wb}
c_all=[c{:}];
```

power_value.m

```
function [y_MWh, y_dollars]=power_value(x,t,cost_curve_MW,MW_rating)

% Calculates value of generation pattern over time t

% Inputs:
% x - hourly turbine time series (as rows for vectorizing!), integers
% between 0 and no_of_units
% t  time series of JDAY values
% cost_curve_MW  2 row matrix to create step function, with 1st row
% being hours and 2nd row $/MW-hr values
% MW_rating - the fixed MW level of turbine_discharge_curve (25 MW for
% OHL)
% Outputs:
% y_MWh - total MWh produced
% y_dollars  total price in $ of generation pattern

%Multiply each turbine output by number of turbines online
output_MW=x*MW_rating; %MW

%Calculate total power output
y_MWh=sum(output_MW')';
%Calculate price output
y_dollars= cost_curve(t, output_MW, cost_curve_MW');
```

```

clearvars t output_MW i
```

runW2validation.m

```

for wb=1:size(CFG,2)
   %% Run W2 validation and bring the resulting two and cwo values back
   fprintf(['Running W2 validation simulation for reservoir #', num2str(wb),'. \n']);
   %Copy W2 folder into new directory in results
   copyfile(CFG{wb}.w2inputDir,['results/w2_iter' num2str(iter) '_wb' num2str(wb)])

   %Open control file and modify TMEND
   fid=fopen(['results/w2_iter' num2str(iter) '_wb' num2str(wb) '/w2_con.npt']);
   i=1; A{i}=fgetl(fid);
   while ischar(A{i}) i=i+1; A{i}=fgetl(fid); end
   fclose(fid); A{28}(22:24)=num2str(t_all(end));
   fid=fopen(['results/w2_iter' num2str(iter) '_wb' num2str(wb) '/w2_con.npt'],'w');
   for i=1:numel(A)
      fprintf(fid,'%s\r\n', A{i});
      if A{i+1}==-1
         break
      end
   end
   fclose(fid); clearvars A i fid

   %If wb~=1, update BR1 Qin, Tin, and DOin
   if wb~=1
      %BR1 Qin
      fid=fopen(['results/w2_iter' num2str(iter) '_wb' num2str(wb) '/' CFG{wb}.MainstemBR1Qin]);
      i=1; A{i}=fgetl(fid);
      while ischar(A{i})
         i=i+1; A{i}=fgetl(fid);
         if i>3
            if str2double(A{i}(1:8))>=t_all(1)
               A(end)=[]; break
            end
         end
      end
      fclose(fid);
      for i=1:size(replacements{wb-1},1)
         A{numel(A)+1}=sprintf('%8.3f%8.3f',...
            [replacements{wb-1}(i,1) sum(replacements{wb-1}(i,2:end),2)]);
      end
      fid=fopen(['results/w2_iter' num2str(iter) '_wb' num2str(wb) '/' CFG{wb}.MainstemBR1Qin],'w');
      for i=1:numel(A)
         fprintf(fid,'%s\r\n', A{i});
      end
      fclose(fid); clearvars A i fid
      %BR1 Tin
      fid=fopen(['results/w2_iter' num2str(iter) '_wb' num2str(wb) '/' CFG{wb}.MainstemBR1Tin]);
      for i=1:3
         A{i}=fgetl(fid);
      end
      fclose(fid);
      temps=W2validation{wb-1}.T(~isnan(W2validation{wb-1}.T(:,2)),:);
      for i=1:size(temps,1)
```

```
53          A{i+3}=sprintf('%8.3f%8.3f', temps(i,:));
54       end
55       fid=fopen(['results/w2_iter' num2str(iter) '_wb' num2str(wb) '/' CFG{wb}.
             ↪ MainstemBR1Tin],'w');
56       for i=1:numel(A)
57          fprintf(fid,'%s\r\n', A{i});
58       end
59       fclose(fid); clearvars A i fid temps
60       %BR1 DOin
61       fid=fopen(['results/w2_iter' num2str(iter) '_wb' num2str(wb) '/' CFG{wb}.
             ↪ MainstemBR1Cin]);
62       for i=1:3
63          A{i}=fgetl(fid);
64       end
65       fclose(fid);
66       fid=fopen(['results/w2_iter' num2str(iter) '_wb' num2str(wb) '/' CFG{wb}.
             ↪ MainstemBR1Cin]);
67       C=textscan(fid,[repmat('%8f', 1, 50) '%*[^\n]'],10^8,...
68          'headerLines',3,'collectoutput', true); %50 & 10^8 are arbitrary big
                ↪ numbers
69       C{1}(:,isnan(C{1}(1,:)))=[]; C{1}=C{1}(C{1}(:,1)<=t_all(end),:);
70       dos=W2validation{wb-1}.DO(~isnan(W2validation{wb-1}.DO(:,2)),:);
71       flag=0;
72       for i=1:size(C{1},1)
73          r(i)=interp1(dos(:,1),dos(:,2),C{1}(i,1));
74          if ~isnan(r(i))
75             C{1}(i,end)=r(i);
76          elseif isnan(r(i)) & C{1}(i,1)>dos(end,1) & flag==0
77             a=dos(end,2); flag=1;
78             C{1}(i,end)=a;
79          end
80       end
81       for i=1:size(C{1},1)
82          A{i+3}=sprintf('%8.3f%8.3f%8.3f%8.3f%8.3f%8.3f%8.3f%8.3f%8.3f', C{1}(i
                ↪ ,:));
83       end
84       fclose(fid);
85       fid=fopen(['results/w2_iter' num2str(iter) '_wb' num2str(wb) '/' CFG{wb}.
             ↪ MainstemBR1Cin],'w');
86       for i=1:numel(A)
87          fprintf(fid,'%s\r\n', A{i});
88       end
89       fclose(fid); clearvars A fid C i r dos flag a
90    end
91    %Open qot_br1.npt and modify turb and spill columns
92    fid=fopen(['results/w2_iter' num2str(iter) '_wb' num2str(wb) '/qot_br1.npt'])
          ↪ ;
93    i=1; A{i}=fgetl(fid);
94    while ischar(A{i})
95       i=i+1; A{i}=fgetl(fid);
96       if i>3
97          if str2double(A{i}(1:8))>=t_all(1)
98             A(end)=[]; break
99          end
100      end
101   end
102   fclose(fid);
103   if strcmp(CFG{wb}.TurbSpillOrder,'1')
104      replacements{wb}=[Q{wb}.QOT_BR1_T(Q{wb}.QOT_BR1_T(:,1)>=t_all(1),:) ...
105         Q{wb}.QOT_BR1_S(Q{wb}.QOT_BR1_S(:,1)>=t_all(1),2)];
106   elseif strcmp(CFG{wb}.TurbSpillOrder,'0')
107      replacements{wb}=[Q{wb}.QOT_BR1_S(Q{wb}.QOT_BR1_S(:,1)>=t_all(1),:) ...
108         Q{wb}.QOT_BR1_T(Q{wb}.QOT_BR1_T(:,1)>=t_all(1),2)];
109   end
```

```
110     for i=1:size(replacements{wb},1)
111        A{numel(A)+1}=sprintf('%8.3f%8.3f%8.3f', replacements{wb}(i,:));
112     end
113     fid=fopen(['results/w2_iter' num2str(iter) '_wb' num2str(wb) '/qot_br1.npt'],
            ↪ 'w');
114     for i=1:numel(A)
115        fprintf(fid,'%s\r\n', A{i});
116     end
117     fclose(fid); clearvars A i fid
118
119     %Run executable w2.exe
120     str=['results/w2_iter' num2str(iter) '_wb' num2str(wb)];
121     cd(str)
122     [~,~]=system('w2.exe');
123     cd ../..
124     clearvars str
125
126     %Read in results from two and cwo files (assume DO is last col in cwo)
127     W2validation{wb}.T=[]; W2validation{wb}.DO=[];
128     d=dir(['results/w2_iter' num2str(iter) '_wb' num2str(wb) '/two*.opt']);
129     fid=fopen(['results/w2_iter' num2str(iter) '_wb' num2str(wb) '/' d(end).name
            ↪ ]);
130     C=textscan(fid,[repmat('%8f', 1, 50) '%*[^\n]'],10^8,...
131        'headerLines',3,'collectoutput', true); %50 & 10^8 are arbitrary big
               ↪ numbers
132     W2validation{wb}.T=C{1}; W2validation{wb}.T(:,isnan(W2validation{wb}.T(1,:)))
            ↪ =[];
133     fclose(fid);
134     d=dir(['results/w2_iter' num2str(iter) '_wb' num2str(wb) '/cwo*.opt']);
135     fid=fopen(['results/w2_iter' num2str(iter) '_wb' num2str(wb) '/' d(end).name
            ↪ ]);
136     C=textscan(fid,[repmat('%8f', 1, 50) '%*[^\n]'],10^8,...
137        'headerLines',3,'collectoutput', true); %50 & 10^8 are arbitrary big
               ↪ numbers
138     W2validation{wb}.DO=C{1}; W2validation{wb}.DO(:,isnan(W2validation{wb}.DO
            ↪ (1,:)))=[];
139     W2validation{wb}.DO=[W2validation{wb}.DO(:,1) W2validation{wb}.DO(:,end)];
140     fclose(fid);
141     clearvars d C fid
142     %Reset 0 values to nan
143     W2validation{wb}.T(W2validation{wb}.T(:,2)==0,2)=nan;
144     W2validation{wb}.DO(W2validation{wb}.DO(:,2)==0,2)=nan;
145
146     %% Run another W2 simulation, swapping turb and spill, for NARX training data
            ↪  diversity
147     fprintf(['Running W2 simulation for reservoir #', num2str(wb),', swapping
            ↪ turb and spill for NARX training data diversity. \n']);
148     %Copy W2 folder into new directory in results
149     copyfile(['results/w2_iter' num2str(iter) '_wb' num2str(wb)],['results/
            ↪ w2_iter' num2str(iter) '_wb' num2str(wb) '_flip']);
150     %Open qot_br1.npt and modify turb and spill columns
151     fid=fopen(['results/w2_iter' num2str(iter) '_wb' num2str(wb) '_flip/qot_br1.
            ↪ npt']);
152     i=1; A{i}=fgetl(fid);
153     while ischar(A{i})
154        i=i+1; A{i}=fgetl(fid);
155        if i>3
156           if str2double(A{i}(1:8))>=t_all(1)
157              A(end)=[]; break
158           end
159        end
160     end
161     fclose(fid);
162     if strcmp(CFG{wb}.TurbSpillOrder,'1') %THIS PART IS SWAPPED FROM ABOVE
```

```
      replacements{wb}=[Q{wb}.QOT_BR1_S(Q{wb}.QOT_BR1_S(:,1)>=t_all(1),:) ...
         Q{wb}.QOT_BR1_T(Q{wb}.QOT_BR1_T(:,1)>=t_all(1),2)];
   elseif strcmp(CFG{wb}.TurbSpillOrder,'0')
      replacements{wb}=[Q{wb}.QOT_BR1_T(Q{wb}.QOT_BR1_T(:,1)>=t_all(1),:) ...
         Q{wb}.QOT_BR1_S(Q{wb}.QOT_BR1_S(:,1)>=t_all(1),2)];
   end
   for i=1:size(replacements{wb},1)
      A{numel(A)+1}=sprintf('%8.3f%8.3f%8.3f', replacements{wb}(i,:));
   end
   fid=fopen(['results/w2_iter' num2str(iter) '_wb' num2str(wb) '_flip/qot_br1.
       ↪ npt'],'w');
   for i=1:numel(A)
      fprintf(fid,'%s\r\n', A{i});
   end
   fclose(fid); clearvars A i fid
   %Run executable w2.exe
   str=['results/w2_iter' num2str(iter) '_wb' num2str(wb) '_flip'];
   cd(str)
   [~,~]=system('w2.exe');
   cd ../..
   clearvars str
   %Read in results from two and cwo files (assume DO is last col in cwo)
   W2validation_flip{wb}.T=[]; W2validation_flip{wb}.DO=[];
   d=dir(['results/w2_iter' num2str(iter) '_wb' num2str(wb) '_flip/two*.opt']);
   fid=fopen(['results/w2_iter' num2str(iter) '_wb' num2str(wb) '_flip/' d(end).
       ↪ name]);
   C=textscan(fid,[repmat('%8f', 1, 50) '%*[^\n]'],10^8,...
      'headerLines',3,'collectoutput', true); %50 & 10^8 are arbitrary big
          ↪ numbers
   W2validation_flip{wb}.T=C{1}; W2validation_flip{wb}.T(:,isnan(
       ↪ W2validation_flip{wb}.T(1,:)))=[];
   fclose(fid);
   d=dir(['results/w2_iter' num2str(iter) '_wb' num2str(wb) '_flip/cwo*.opt']);
   fid=fopen(['results/w2_iter' num2str(iter) '_wb' num2str(wb) '_flip/' d(end).
       ↪ name]);
   C=textscan(fid,[repmat('%8f', 1, 50) '%*[^\n]'],10^8,...
      'headerLines',3,'collectoutput', true); %50 & 10^8 are arbitrary big
          ↪ numbers
   W2validation_flip{wb}.DO=C{1}; W2validation_flip{wb}.DO(:,isnan(
       ↪ W2validation_flip{wb}.DO(1,:)))=[];
   W2validation_flip{wb}.DO=[W2validation_flip{wb}.DO(:,1) W2validation_flip{wb
       ↪ }.DO(:,end)];
   fclose(fid);
   clearvars d C fid
   %Reset 0 values to nan
   W2validation_flip{wb}.T(W2validation_flip{wb}.T(:,2)==0,2)=nan;
   W2validation_flip{wb}.DO(W2validation_flip{wb}.DO(:,2)==0,2)=nan;
end

clearvars replacements
```

updateQ.m

```
function Q=updateQ(Q,CFG,x_final,t,frequency,ic_elev,turbine_discharge,...
   WQ,ELWS_targets)

% Updates the structure Q with ELWS, discharge flows, and discharge WQ
% based on previous days optimized
%
% Inputs:
% Q - all other inflows and outflows, interpolation settings, and
% storage-elev curve
% CFG - structure containing field values from config files
```

```
% x_final - vector containing timeseries of active turbine levels for all
% waterbodies
% t  time series of JDAY values
% frequency - prediction frequency (ex: 0.25=1/4 day=6 hours)
% ic_elev - initial elevation condition (meters)
% turbine_discharge - turbine discharge curve at fixed MW level, with
% col 1 in meters and col 2 in cms
% WQ - structure containing water quality constraints and NARX models
% DO_narx - structure containing everything needed to make DO discharge
% predictions, including:
% turb_colum - column in exogenous variables with turb flows
% spill_column - column in exogenous variables with spill flows
% times - JDAY values used in training (not used)
% inputDelays - delays for exogenous inputs
% feedbackDelays - delays for prediction feedbacks
% input_variables - 2 row cell containing variable names in first
% row and column number in second. For example, 'MET_WB1'
% contains multiple columns of data but only some may be used
% for NARX predictions
% bias - bias for each trained neural network
% weights - weights for each trained neural network (sum to 1)
% narx_net_closed - neural networks
% DO_limit - lower and upper DO limits (NaN means it doesn't exist)
% DO_slack - relaxation from DO_limit (either upper or lower -
% doesn't make sense to have both)
% Temp_narx - structure containing everything needed to make temp discharge
% predictions, including:
% turb_colum - column in exogenous variables with turb flows
% spill_column - column in exogenous variables with spill flows
% times - JDAY values used in training (not used)
% inputDelays - delays for exogenous inputs
% feedbackDelays - delays for prediction feedbacks
% input_variables - 2 row cell containing variable names in first
% row and column number in second. For example, 'MET_WB1'
% contains multiple columns of data but only some may be used
% for NARX predictions
% bias - bias for each trained neural network
% weights - weights for each trained neural network (sum to 1)
% narx_net_closed - neural networks
% Temp_limit - lower and upper temp limits (NaN means it doesn't exist)
% Temp_slack - relaxation from Temp_limit (either upper or lower -
% doesn't make sense to have both)
% ELWS_targets - 2 column matrix with JDAY in col1 and elevation target
% in col2
% Outputs:
% Q - all other inflows and outflows, interpolation settings,
% storage-elev curve, and tailwater curve (all in meters)

for wb=1:size(CFG,2)
   clearvars incoming_flow
   %If wb==1, update ELWS, QOT_BR1_T, CWO, TWO
   %If wb~=1, update ELWS, QOT_BR1_T, CWO, TWO, QIN_BR1, CIN_BR1, TIN_BR1 (CWO &
      ↪ TWO may not update for last reservoir if NARX models aren't provided)
   x=x_final{wb}(size(x_final{wb},2)-size(t,2)+2:end);
   if wb==1
      [turb_discharges{wb},spill_discharges{wb},HWs{wb},~,~] = ...
         activeunits_to_discharges(x,t,frequency,...
         Q{wb},ic_elev{wb},turbine_discharge{wb},ELWS_targets{wb},...
         [],[]);
      Q{wb}.ELWS=[Q{wb}.ELWS(Q{wb}.ELWS(:,1)<t(1),:); t' HWs{wb}'];
      Q{wb}.QOT_BR1_T=[Q{wb}.QOT_BR1_T(Q{wb}.QOT_BR1_T(:,1)<t(1),:);...
         t' turb_discharges{wb}'];
      Q{wb}.QOT_BR1_S=[Q{wb}.QOT_BR1_S(Q{wb}.QOT_BR1_S(:,1)<t(1),:);...
         t' ones(size(t,2),1)*spill_discharges{wb}];
```

```
74        DO_pred{wb}=narx_predictions(WQ{wb}.DO_narx,frequency,t,Q{wb},x,...
75           turb_discharges{wb},spill_discharges{wb},[],Q{wb}.CWO,'do');
76        Temp_pred{wb}=narx_predictions(WQ{wb}.Temp_narx,frequency,t,Q{wb},x,...
77           turb_discharges{wb},spill_discharges{wb},[],Q{wb}.TWO,'temp');
78        %Remove NaNs from DO_pred and Temp_pred!
79        outgoing_DO{wb}=[t(2:end)' DO_pred{wb}'];
80        outgoing_DO{wb}=outgoing_DO{wb}(~isnan(outgoing_DO{wb}(:,2)),:);
81        outgoing_Temp{wb}=[t(2:end)' Temp_pred{wb}'];
82        outgoing_Temp{wb}=outgoing_Temp{wb}(~isnan(outgoing_Temp{wb}(:,2)),:);
83        %If last values in WQ predictions are NaN, need to add last row to
              ↪ outgoing_DO and outgoing_Temp
84        if outgoing_Temp{wb}(end,1)<t(end)
85           outgoing_Temp{wb}=[outgoing_Temp{wb}; t(end) outgoing_Temp{wb}(end,2)];
86           outgoing_DO{wb}=[outgoing_DO{wb}; t(end) outgoing_DO{wb}(end,2)];
87        end
88        Q{wb}.CWO=[Q{wb}.CWO(Q{wb}.CWO(:,1)<t(2),:); outgoing_DO{wb}];
89        Q{wb}.TWO=[Q{wb}.TWO(Q{wb}.TWO(:,1)<t(2),:); outgoing_Temp{wb}];
90     else
91        incoming_flow=turb_discharges{wb-1}+spill_discharges{wb-1};
92        [turb_discharges{wb},spill_discharges{wb},HWs{wb},~,~] = ...
93           activeunits_to_discharges(x,t,frequency,...
94           Q{wb},ic_elev{wb},turbine_discharge{wb},ELWS_targets{wb},...
95           t,incoming_flow);
96        Q{wb}.ELWS=[Q{wb}.ELWS(Q{wb}.ELWS(:,1)<t(1),:); t' HWs{wb}'];
97        Q{wb}.QOT_BR1_T=[Q{wb}.QOT_BR1_T(Q{wb}.QOT_BR1_T(:,1)<t(1),:);...
98           t' turb_discharges{wb}'];
99        Q{wb}.QOT_BR1_S=[Q{wb}.QOT_BR1_S(Q{wb}.QOT_BR1_S(:,1)<t(1),:);...
100          t' ones(size(t,2),1)*spill_discharges{wb}];
101       %Qin contains both spill and turbine
102       Q{wb}.QIN_BR1=[Q{wb}.QIN_BR1(Q{wb}.QIN_BR1(:,1)<t(1),:);...
103          t' incoming_flow'];
104       Q{wb}.CIN_BR1=[Q{wb}.CIN_BR1(Q{wb}.CIN_BR1(:,1)<t(2),:);...
105          outgoing_DO{wb-1}];
106       Q{wb}.TIN_BR1=[Q{wb}.TIN_BR1(Q{wb}.TIN_BR1(:,1)<t(2),:);...
107          outgoing_Temp{wb-1}];
108       %May not have WQ calculations for final reservoir's discharge (depends on
              ↪ problem definition) so check for these
109       if ~isempty(WQ{wb}.DO_narx)
110          DO_pred{wb}=narx_predictions(WQ{wb}.DO_narx,frequency,t,...
111             Q{wb},x,turb_discharges{wb},spill_discharges{wb},[],Q{wb}.CWO,'do');
112          %Remove NaNs from DO_pred and Temp_pred!
113          outgoing_DO{wb}=[t(2:end)' DO_pred{wb}'];
114          outgoing_DO{wb}=outgoing_DO{wb}(~isnan(outgoing_DO{wb}(:,2)),:);
115          %If last values in WQ predictions are NaN, need to add last row to
                 ↪ outgoing_DO and outgoing_Temp
116          if outgoing_DO{wb}(end,1)<t(end)
117             outgoing_DO{wb}=[outgoing_DO{wb}; t(end) outgoing_DO{wb}(end,2)];
118          end
119          Q{wb}.CWO=[Q{wb}.CWO(Q{wb}.CWO(:,1)<t(1),:); outgoing_DO{wb}];
120       end
121       if ~isempty(WQ{wb}.Temp_narx)
122          Temp_pred{wb}=narx_predictions(WQ{wb}.Temp_narx,frequency,t,...
123             Q{wb},x,turb_discharges{wb},spill_discharges{wb},[],Q{wb}.TWO,'temp'
                    ↪ );
124          %Remove NaNs from DO_pred and Temp_pred!
125          outgoing_Temp{wb}=[t(2:end)' Temp_pred{wb}'];
126          outgoing_Temp{wb}=...
127             outgoing_Temp{wb}(~isnan(outgoing_Temp{wb}(:,2)),:);
128          %If last values in WQ predictions are NaN, need to add last row to
                 ↪ outgoing_DO and outgoing_Temp
129          if outgoing_Temp{wb}(end,1)<t(end)
130             outgoing_Temp{wb}=[outgoing_Temp{wb}; t(end) outgoing_Temp{wb}(end
                    ↪ ,2)];
131          end
```

```
132             Q{wb}.TWO=[Q{wb}.TWO(Q{wb}.TWO(:,1)<t(1),:); outgoing_Temp{wb}];
133         end
134     end
135 end
136 
137 clearvars outgoing_DO outgoing_Temp
```

Appendix E

MATLAB® CODE FOR HYDROPOWER OPTIMIZATION UNDER WATER QUALITY CONSTRAINTS MODIFIED FOR RANDOM IMMIGRANTS REPLACEMENT AND ADAPTIVE ADDITIONAL SAMPLING

This appendix contains code that is in addition to or modified from that which is provided in Appendix D in order to create the replacement and adaptive additional sampling functionalities described in Chapter IV. As written, it is not equipped to handle multiple waterbodies or multiday problems.

config.json

```
{
      "jdayStart": "215",
      "OperatingPeriod": "1",
      "OptimizeDayByDay": "0",
      "LogFile": "results/results_log.txt",
      "NumberOfWaterbodies": "1",
      "wb1config": "config_OHL.json",
      "GAPopSizeMultiplierStart": "480",
      "FeasibilityCheckPopSizeMultiplierStart": "480",
      "GAGenerationsEarlyStoppingStart": "1",
      "RandomNumberGeneratorSeed": "7",
      "TrainingSetSize": "4",
      "InitialTrainingSetSize": "10",
      "ReplacementOnOff": "ON",
      "AdditionalSamplingOnOff": "OFF"
}
```

config_OHL.json

```
{
      "Name": "Old Hickory",
      "WaterSurfaceElevationInitial": "",
      "DischargeDOInitial": "",
      "DischargeTempInitial": "",
      "WaterSurfaceElevationMin": "134.722",
      "WaterSurfaceElevationMax": "135.636",
      "DischargeDOMin": "7",
      "DischargeDOMax": "",
      "DischargeTempMin": "",
      "DischargeTempMax": "",
      "MaxHourlyChangeInTurbineUnit": "1",
      "MaxHoursWithZeroGeneration": "6",
      "NumberOfTurbineUnits": "4",
      "MWRatingPerTurbineUnit": "25",
      "TurbineDischargeCurve": "OHL/testfiles/turbine_discharge_curve_25MW.txt",
      "StorageElevationCurve": "OHL/testfiles/storage_elevation.txt",
      "TailWaterRatingCurve": "OHL/testfiles/tailwater_rating.txt",
```

```
        "DailyCostCurve": "OHL/testfiles/cost_curve2.txt",
        "TrainedDONeuralNetworkFile": "OHL/testfiles/ohl_DO_narx_20160906.mat",
        "TrainedTempNeuralNetworkFile": "OHL/testfiles/ohl_temp_narx_20160906.mat
            ↪ ",
        "WaterSurfaceElevationTargets": "",
        "optimizationDir": "OHL/testfiles/optimization215/",
        "ForecastTurbinePattern": "OHL/testfiles/forecast_turbine_pattern215.txt",
        "PreviousTurbinePattern": "OHL/testfiles/previous_turbine_pattern215.txt",
        "w2inputDir": "OHL/testfiles/w2input215/",
        "TurbSpillOrder": "1",
        "MainstemBR1Qin": "qin_br1.npt",
        "MainstemBR1Tin": "tin_br1_2005.npt",
        "MainstemBR1Cin": "cin_br1_2005.npt",
        "TransitionMatrix": "OHL/testfiles/transition_matrix.txt"
}
```

main.m

```
function main(configfile)

initialization;
tic; initial_NARX_model_generation; toc

while retraining=='Y'
   iter=iter+1;
   %Run optimization over planning period
   fprintf(['Running 2-step optimization to minimize WQ constraint violations,
       ↪ then maximize power value. \n']);
   opttiming=tic; optimization_routine; timing(1)=toc(opttiming); clearvars
       ↪ opttiming

   close all; ga_results_plotting_nobanding
   h = get(0,'children'); h=sort(h);
   for wb=1:length(h)
      str=['results/' datestr(clock,'yyyy-mm-dd-HHMM') '_iter' num2str(iter) '
          ↪ _wb' num2str(wb) '_' num2str(round(y_dollars_total(2)))];
      savefig(h(wb),str)
   end
   %Retrain NARX models
   retraintiming=tic; NARX_retrain_trpt; timing(3)=toc(retraintiming);
   clearvars trainingpop retraintiming

   %Print to results log file
   fileID=fopen(config.LogFile,'a');
   results.dollars(iter)=y_dollars_total(2);
   fprintf(fileID,'%12.0f %16.0f %12.0f %18.0f %18.0f %14.0f %14.0f %14.0f %14.0
       ↪ f %12.3f %12.3f %12.0f %12.0f %12.0f %12.0f',...
      iter,feasiblilitycheck_ga_pop_size,ga_pop_size,training_ss_clusters,
          ↪ training_ss_nearby,...
      funccount_tot,funccount_cache,funccount_ga_tot,funccount_ga_cache,SD(iter
          ↪ +1),replacement_rate,y_MWh_total(1),y_MWh_total(2),...
      y_dollars_total(1),y_dollars_total(2));
   for wb=1:size(CFG,2)
      fprintf(fileID,' %12.0f %12.0f',y_MWh(wb,2),y_dollars(wb,2));
   end
   for wb=1:size(CFG,2)
      results.AME(iter,wb*2-1:wb*2)=[AME{wb}.T,AME{wb}.DO];
      results.slacks(iter,wb*2-1:wb*2)=[slacks{wb}.T.W2,slacks{wb}.DO.W2];
      fprintf(fileID,' %12.3f %12.3f %16.3f %16.3f %15.3f %15.3f %15.3f %15.3f'
          ↪ ,...
         AME{wb}.T,AME{wb}.DO,AME_trpt.T_avg(iter),AME_trpt.DO_avg(iter),...
         slacks{wb}.T.NN,slacks{wb}.DO.NN,...
         slacks{wb}.T.W2,slacks{wb}.DO.W2);
```

```
    end
    clearvars slacks ans data_start objfuncvalues Output_no0s Outputprev h wb Ax1 Ax2 Ax3 H h1 h2 h3 h5 h6 h7 legend1 output nVar maxdelay wb xlims xrange ylims yrange

    %Adjust ga_pop_size, if best solution found is the same as the best solution from the last iteration
    if iter==1
       ga_pop_size_1=ga_pop_size;
    else
       if all(x_final_all{iter}==x_final_all{best_iter(iter-1)})
          ga_pop_size=round(min(4800,ga_pop_size*ga_pop_size_expand));
       else
           ga_pop_size=round(max(ga_pop_size_1,ga_pop_size/ga_pop_size_expand));
       end
    end

    %Determine if we've met stopping point, when best soln has not changed in 3 iter and DO and/or temp validation checks at best soln are below 0.5 AME
    fprintf(fileID,' %12.0f %13s %13s',best_iter(iter),x_in_initpop,x_in_prevpop);
    if iter==50 retraining='N'; end

    if iter==1
       fprintf(fileID,' %12.0f',nan);
    else
       %2-norm between current solution and best iteration solution
       two_norm(iter)=norm(x_final_all{iter}(:)-x_final_all{best_iter(iter-1)}(:));
       fprintf(fileID,' %12.3f',two_norm(iter));
    end

    %Optimization timing(1) includes W2 runs and NARX retraining, so subtract those out
    timing(1)=timing(1)-timing(2);
    fprintf(fileID,'%12.3f %12.3f %12.3f\r\n',timing(1),timing(2),...
       timing(3)); fclose(fileID);

end
save('results/end.mat');

fprintf('OPTIMIZATION','Optimization over operating period complete.')
cumulative_discharge_plot;
```

initialization.m

```
%Initialization

%% Startup: Empty vars, setup paths, check input, init config, random # init
clearvars -except configfile transition_matrix

%Start parallel pool
gcp;

% add path to 'lib' folder
if (~isdeployed)
   addpath('./lib');
end

% load general config
config=loadjson('config.json');
%Load config for each waterbody, as defined in general config
```

```
for wb=1:str2double(config.NumberOfWaterbodies)
   CFG{wb}=loadjson(eval(['config.wb' num2str(wb) 'config']));
end

% create logger
L = log4m.getLogger('optimization_run.log');

%% Load in data and set constraints and system specs

%Do replacement and/or additional W2 sampling steps?
ReplacementOnOff=config.ReplacementOnOff;
AdditionalSamplingOnOff=config.AdditionalSamplingOnOff;

transition_matrix=[];
%TOTAL time period to optimize on
start_date=str2double(config.jdayStart);
frequency=1/24;
days_forward=str2double(config.OperatingPeriod);
t=[start_date:frequency:start_date+1];
%Optimize day by day (1), or all in one step (0)
Optimize_day_by_day=str2double(config.OptimizeDayByDay);
%GA population sizes
ga_pop_size=str2double(config.GAPopSizeMultiplierStart)*size(CFG,2);
feasiblilitycheck_ga_pop_size=str2double(config.
    ↪ FeasibilityCheckPopSizeMultiplierStart)*size(CFG,2);
GAgenerations=str2double(config.GAGenerationsEarlyStoppingStart);
%Random number generator seed
rng(str2double(config.RandomNumberGeneratorSeed))
%Training set size (number of kmeans clusters, and number of samples near
    ↪ optimal)
training_ss=str2double(config.TrainingSetSize);
training_ss_clusters=training_ss; training_ss_nearby=0; %initial values
Initialtrainingsetsize=str2double(config.InitialTrainingSetSize);
%Other variables from config files
for wb=1:size(CFG,2)
   %Transition matrix for markov chain
   if ~isempty(CFG{wb}.TransitionMatrix)
      transition_matrix{wb}=dlmread(CFG{wb}.TransitionMatrix);
   else
      transition_matrix{wb}=[];
   end
   %Number of turbines - 4 for OHL
   no_of_units{wb}=str2double(CFG{wb}.NumberOfTurbineUnits);
   %Operating level, MW
   MW_rating{wb}=str2double(CFG{wb}.MWRatingPerTurbineUnit);
      %Previous elevations
   elevtemp{wb}=dlmread(strcat(CFG{wb}.optimizationDir,filesep,'ELWS.csv'),','
       ↪ ,1,0);
   %Elevation constraints - general
   ELWS_limit{wb}(1)=str2double(CFG{wb}.WaterSurfaceElevationMin);
   ELWS_limit{wb}(2)=str2double(CFG{wb}.WaterSurfaceElevationMax);
   %Max hourly unit change constraint
   if ~isempty(CFG{wb}.MaxHourlyChangeInTurbineUnit)
      max_hrly_unit_change{wb}=str2double(CFG{wb}.MaxHourlyChangeInTurbineUnit);
   else
      max_hrly_unit_change{wb}=[];
   end
   %Zero generation hourly limit - can't go longer than this with no turb flow
   if ~isempty(CFG{wb}.MaxHoursWithZeroGeneration)
      zero_gen_limit{wb}=str2double(CFG{wb}.MaxHoursWithZeroGeneration);
   else
      zero_gen_limit{wb}=[];
   end
   %DO discharge NARX model
```

```
78    if isempty(CFG{wb}.TrainedDONeuralNetworkFile)
79              WQ{wb}.DO_narx=[];
80       else
81              WQ{wb}.DO_narx=load(CFG{wb}.TrainedDONeuralNetworkFile);
82              fn=fieldnames(WQ{wb}.DO_narx); WQ{wb}.DO_narx=WQ{wb}.DO_narx.(fn{1})
                    ↪ ; clearvars fn
83       end
84    WQ{wb}.DO_limit(1)=str2double(CFG{wb}.DischargeDOMin);
85    WQ{wb}.DO_limit(2)=str2double(CFG{wb}.DischargeDOMax);
86    WQ{wb}.DO_slack=0;
87    %Temperature discharge NARX model
88    if isempty(CFG{wb}.TrainedTempNeuralNetworkFile)
89              WQ{wb}.Temp_narx=[];
90       else
91              WQ{wb}.Temp_narx=load(CFG{wb}.TrainedTempNeuralNetworkFile);
92              fn=fieldnames(WQ{wb}.Temp_narx); WQ{wb}.Temp_narx=WQ{wb}.Temp_narx.(
                    ↪ fn{1});
93              clearvars fn
94       end
95       WQ{wb}.Temp_limit(1)=str2double(CFG{wb}.DischargeTempMin);
96       WQ{wb}.Temp_limit(2)=str2double(CFG{wb}.DischargeTempMax);
97    WQ{wb}.Temp_slack=0;
98    %Cost curve
99    if isempty(CFG{wb}.DailyCostCurve)
100      cost_curve_MW{wb}=[0 1];
101   else
102      cost_curve_MW{wb}=dlmread(CFG{wb}.DailyCostCurve,' ',1,0);
103   end
104   %Turbine discharge curve - meters, cms at MW_rating
105   turbine_discharge{wb}=dlmread(CFG{wb}.TurbineDischargeCurve,' ',1,0);
106      %Find initial elevation
107   ic_elev_first{wb}=interp1(elevtemp{wb}(:,1),elevtemp{wb}(:,2),start_date);
108   %Build the variable Q, which includes all flows for water balance,
          ↪ interpolation settings, tw curve both tabular discharge vs. tw and tw
          ↪ as f(twprev,discharge)), se curve, and other WQ inputs needed for NARX
          ↪  predictions
109   Q{wb}=buildQ(CFG{wb}.optimizationDir);
110   Q{wb}.tw_curve_cms_m=dlmread(CFG{wb}.TailWaterRatingCurve,' ',1,0);
111   Q{wb}.SE_meters_m3=dlmread(CFG{wb}.StorageElevationCurve,' ',1,0);
112      %Save a copy of Q as original projected values - Q will update during
             ↪ optimization
113      Qprojected=Q;
114   end
115
116   %Set up flag for when to do validation checks and retraining of NARX models
117   for wb=1:size(CFG,2)
118         WQ{wb}.DO_valid_check=0; WQ{wb}.Temp_valid_check=0;
119   end
120   for wb=1:size(CFG,2)
121      if any(~isnan(WQ{wb}.DO_limit))
122         WQ{wb}.DO_valid_check=1;
123         if wb~=1
124            for i=1:size(WQ{wb}.DO_narx.input_variables,2)
125               if WQ{wb}.DO_narx.input_variables{1,i}=='TIN_BR1'
126                  WQ{wb-1}.Temp_valid_check=1;
127               end
128               if WQ{wb}.DO_narx.input_variables{1,i}=='CIN_BR1'
129                  WQ{wb-1}.DO_valid_check=1;
130               end
131            end
132         end
133      end
134   end
135   for wb=1:size(CFG,2)
```

```
136     if any(~isnan(WQ{wb}.Temp_limit))
137        WQ{wb}.Temp_valid_check=1;
138     end
139     if wb~=1
140        for i=1:size(WQ{wb}.Temp_narx.input_variables,2)
141           if WQ{wb}.Temp_narx.input_variables{1,i}=='TIN_BR1'
142              WQ{wb-1}.Temp_valid_check=1;
143           end
144           if WQ{wb}.Temp_narx.input_variables{1,i}=='CIN_BR1'
145              WQ{wb-1}.DO_valid_check=1;
146           end
147        end
148     end
149 end
150 clearvars i
151
152 %Set feasibility_check to start algorithm checking constraint feasibility
153 feasibility_check=1;
154 feasible_soln_found=0;
155
156 %Set up time variables, determine forecast and past turbine patterns
157 t_all=[start_date:frequency:start_date+days_forward];
158 t_all_round=roundn(t_all,-2);
159 tprev=[t(1)-max(cell2mat(zero_gen_limit(:)))*frequency:frequency:t(1)];
160 tprev_round=roundn(tprev,-2);
161 for wb=1:size(CFG,2)
162       %Forecast turbine pattern (if supplied)
163       if isempty(CFG{wb}.ForecastTurbinePattern)
164             L.warn('INITIALIZATION',['No reservoir ', num2str(wb), ' forecast
                  ↪ turbine pattern provided - assuming from turbine flows in W2
                  ↪ QOT file.'])
165             x0_all(wb,:)=actual_turb_ops(t_all_round,Qprojected{wb},elevtemp{wb
                  ↪ },turbine_discharge{wb},...
166                no_of_units{wb});
167       else
168             forecastturbpattern=dlmread(CFG{wb}.ForecastTurbinePattern,'\t',1,0)
                  ↪ ;
169             for i=1:size(t_all_round,2)-1
170                index=find(forecastturbpattern(:,1)<=t_all_round(i+1));
171                x0_all(wb,i)=forecastturbpattern(index(end),2);
172             end
173             clearvars i forecastturbpattern index
174    end
175       %Previous turbine pattern for the year (if supplied)
176       if isempty(CFG{wb}.ForecastTurbinePattern)
177             L.warn('INITIALIZATION',['No reservoir ', num2str(wb), ' previous
                  ↪ turbine pattern provided - assuming from turbine flows in W2
                  ↪ QOT file.'])
178             xprev{wb}=actual_turb_ops(tprev_round,Qprojected{wb},elevtemp{wb},
                  ↪ turbine_discharge{wb},no_of_units{wb});
179       else
180             prevturbpattern=dlmread(CFG{wb}.PreviousTurbinePattern,'\t',1,0);
181             for i=1:size(tprev_round,2)
182                index=find(prevturbpattern(:,1)<=tprev_round(i));
183                xprev{wb}(i)=prevturbpattern(index(end),2);
184             end
185             clearvars i prevturbpattern index
186    end
187 end
188
189 %% Do W2 run with outflows consistent with x0_all (supplied W2 folder isn't
     ↪ guaranteed to have flows corresponding to x0 operations)
190 for wb=1:size(CFG,2)
191    [turb_discharges,spill_discharges,~,~,~]=activeunits_to_discharges(x0_all(wb
```

```
    ↪ ,:),t_all,...
192     frequency,Qprojected{wb},ic_elev_first{wb},...
193     turbine_discharge{wb},[],[],[]);
194  Q{wb}.QOT_BR1_T=[Q{wb}.QOT_BR1_T(Q{wb}.QOT_BR1_T(:,1)<t(1),:);...
195     t_all' turb_discharges'];
196  if Optimize_day_by_day==1
197     Q{wb}.QOT_BR1_S=[Q{wb}.QOT_BR1_S(Q{wb}.QOT_BR1_S(:,1)<t(1),:);...
198        t_all' ones(size(t_all,2),1)*spill_discharges];
199  else
200     for ii=1:size(spill_discharges,2)
201        spill_values(1,(1/frequency)*(ii-1)+1:(1/frequency)*(ii)+1)=...
202           spill_discharges(1,ii);
203     end
204     Q{wb}.QOT_BR1_S=[Q{wb}.QOT_BR1_S(Q{wb}.QOT_BR1_S(:,1)<t(1),:);...
205        t_all' spill_values'];
206     clearvars ii spill_values
207  end
208  if ~exist(['results/w2_iter0_wb' num2str(wb)]) %If folder already exists in
    ↪ the results folder from previous testing, don't have to rerun W2 here
209     copyfile(CFG{wb}.w2inputDir,['results/w2_iter0_wb' num2str(wb)])
210     %Open control file and modify TMEND
211     fid=fopen(['results/w2_iter0_wb' num2str(wb) '/w2_con.npt']);
212     i=1; A{i}=fgetl(fid);
213     while ischar(A{i}) i=i+1; A{i}=fgetl(fid); end
214     fclose(fid); A{28}(22:24)=num2str(t_all(end));
215     fid=fopen(['results/w2_iter0_wb' num2str(wb) '/w2_con.npt'],'w');
216     for i=1:numel(A)
217        fprintf(fid,'%s\r\n', A{i});
218        if A{i+1}==-1
219           break
220        end
221     end
222     fclose(fid); clearvars A i fid
223
224     %If wb~=1, update BR1 Qin, Tin, and DOin
225     if wb~=1
226        %BR1 Qin
227        fid=fopen(['results/w2_iter0_wb' num2str(wb) '/' CFG{wb}.MainstemBR1Qin
    ↪ ]);
228        i=1; A{i}=fgetl(fid);
229        while ischar(A{i})
230           i=i+1; A{i}=fgetl(fid);
231           if i>3
232              if str2double(A{i}(1:8))>=t_all(1)
233                 A(end)=[]; break
234              end
235           end
236        end
237        fclose(fid);
238        for i=1:size(replacements{wb-1},1)
239           A{numel(A)+1}=sprintf('%8.3f%8.3f',...
240              [replacements{wb-1}(i,1) sum(replacements{wb-1}(i,2:end),2)]);
241        end
242        fid=fopen(['results/w2_iter0_wb' num2str(wb) '/' CFG{wb}.MainstemBR1Qin
    ↪ ],'w');
243        for i=1:numel(A)
244           fprintf(fid,'%s\r\n', A{i});
245        end
246        fclose(fid); clearvars A i fid
247        %BR1 Tin
248        fid=fopen(['results/w2_iter0_wb' num2str(wb) '/' CFG{wb}.MainstemBR1Tin
    ↪ ]);
249        for i=1:3
250           A{i}=fgetl(fid);
```

```
251             end
252             fclose(fid);
253             temps=W2validation{wb-1}.T(~isnan(W2validation{wb-1}.T(:,2)),:);
254             for i=1:size(temps,1)
255                A{i+3}=sprintf('%8.3f%8.3f', temps(i,:));
256             end
257             fid=fopen(['results/w2_iter0_wb' num2str(wb) '/' CFG{wb}.MainstemBR1Tin
                    ↪ ],'w');
258             for i=1:numel(A)
259                fprintf(fid,'%s\r\n', A{i});
260             end
261             fclose(fid); clearvars A i fid temps
262             %BR1 DOin
263             fid=fopen(['results/w2_iter0_wb' num2str(wb) '/' CFG{wb}.MainstemBR1Cin
                    ↪ ]);
264             for i=1:3
265                A{i}=fgetl(fid);
266             end
267             fclose(fid);
268             fid=fopen(['results/w2_iter0_wb' num2str(wb) '/' CFG{wb}.MainstemBR1Cin
                    ↪ ]);
269             C=textscan(fid,[repmat('%8f', 1, 50) '%*[^\n]'],10^8,...
270                'headerLines',3,'collectoutput', true); %50 & 10^8 are arbitrary big
                       ↪ numbers
271             C{1}(:,isnan(C{1}(1,:)))=[]; C{1}=C{1}(C{1}(:,1)<=t_all(end),:);
272             dos=W2validation{wb-1}.DO(~isnan(W2validation{wb-1}.DO(:,2)),:);
273             flag=0;
274             for i=1:size(C{1},1)
275                r(i)=interp1(dos(:,1),dos(:,2),C{1}(i,1));
276                if ~isnan(r(i))
277                   C{1}(i,end)=r(i);
278                elseif isnan(r(i)) & C{1}(i,1)>dos(end,1) & flag==0
279                   a=dos(end,2); flag=1;
280                   C{1}(i,end)=a;
281                end
282             end
283             for i=1:size(C{1},1)
284                A{i+3}=sprintf('%8.3f%8.3f%8.3f%8.3f%8.3f%8.3f%8.3f%8.3f%8.3f', C
                       ↪ {1}(i,:));
285             end
286             fclose(fid);
287             fid=fopen(['results/w2_iter0_wb' num2str(wb) '/' CFG{wb}.MainstemBR1Cin
                    ↪ ],'w');
288             for i=1:numel(A)
289                fprintf(fid,'%s\r\n', A{i});
290             end
291             fclose(fid); clearvars A fid C i r dos flag a
292          end
293          %Open qot_br1.npt and modify turb and spill columns
294          fid=fopen(['results/w2_iter0_wb' num2str(wb) '/qot_br1.npt']);
295          i=1; A{i}=fgetl(fid);
296          while ischar(A{i})
297             i=i+1; A{i}=fgetl(fid);
298             if i>3
299                if str2double(A{i}(1:8))>=t_all(1)
300                   A(end)=[]; break
301                end
302             end
303          end
304          fclose(fid);
305          if strcmp(CFG{wb}.TurbSpillOrder,'1')
306             replacements{wb}=[Q{wb}.QOT_BR1_T(Q{wb}.QOT_BR1_T(:,1)>=t_all(1),:) ...
307                Q{wb}.QOT_BR1_S(Q{wb}.QOT_BR1_S(:,1)>=t_all(1),2)];
308          elseif strcmp(CFG{wb}.TurbSpillOrder,'0')
```

```
309             replacements{wb}=[Q{wb}.QOT_BR1_S(Q{wb}.QOT_BR1_S(:,1)>=t_all(1),:) ...
310                Q{wb}.QOT_BR1_T(Q{wb}.QOT_BR1_T(:,1)>=t_all(1),2)];
311          end
312          for i=1:size(replacements{wb},1)
313             A{numel(A)+1}=sprintf('%8.3f%8.3f%8.3f', replacements{wb}(i,:));
314          end
315          fid=fopen(['results/w2_iter0_wb' num2str(wb) '/qot_br1.npt'],'w');
316          for i=1:numel(A)
317             fprintf(fid,'%s\r\n', A{i});
318          end
319          fclose(fid); clearvars A i fid
320
321          %Run executable w2.exe
322          fprintf(['Running W2 for wb' num2str(wb) ' with outflows consistent with
                 ↪ projected turbine operations. \n']);
323          str=['results/w2_iter0_wb' num2str(wb)];
324          cd(str)
325          [~,~]=system('w2.exe');
326          cd ../..
327       end
328       %Read in TWO, CWO, and ELWS from W2 run
329       d=dir(['results/w2_iter0_wb' num2str(wb) '/two*.opt']);
330       fid=fopen(['results/w2_iter0_wb' num2str(wb) '/' d(end).name]);
331       C=textscan(fid,[repmat('%8f', 1, 50) '%*[^\n]'],10^8,...
332          'headerLines',3,'collectoutput', true); %50 & 10^8 are arbitrary big
                 ↪ number
333       Q{wb}.TWO=C{1}; Q{wb}.TWO(:,isnan(Q{wb}.TWO(1,:)))=[];
334       fclose(fid);
335       d=dir(['results/w2_iter0_wb' num2str(wb) '/cwo*.opt']);
336       fid=fopen(['results/w2_iter0_wb' num2str(wb) '/' d(end).name]);
337       C=textscan(fid,[repmat('%8f', 1, 50) '%*[^\n]'],10^8,...
338          'headerLines',3,'collectoutput', true); %50 & 10^8 are arbitrary big
                 ↪ numbers
339       Q{wb}.CWO=C{1}; Q{wb}.CWO(:,isnan(Q{wb}.CWO(1,:)))=[];
340       Q{wb}.CWO=[Q{wb}.CWO(:,1) Q{wb}.CWO(:,end)];
341       fclose(fid);
342       d=dir(['results/w2_iter0_wb' num2str(wb) '/tsr*.opt']);
343       fid=fopen(['results/w2_iter0_wb' num2str(wb) '/' d(end).name]);
344       C=textscan(fid,[repmat('%8f', 1, 50) '%*[^\n]'],10^8,...
345          'headerLines',12,'collectoutput', true); %50 & 10^8 are arbitrary big
                 ↪ numbers
346       Q{wb}.ELWS=C{1}; Q{wb}.ELWS(:,isnan(Q{wb}.ELWS(1,:)))=[];
347       Q{wb}.ELWS=[Q{wb}.ELWS(:,1) Q{wb}.ELWS(:,3)];
348       fclose(fid);
349
350       Qprojected=Q; clearvars str turb_discharges spill_discharges C fid d
351    end
352    clearvars replacements
353
354    %% Compute target elevations
355    for wb=1:size(CFG,2)
356       %Target elevations (soft constraint)
357       [~,~,HWs_x0(wb,:),~,~]=activeunits_to_discharges(x0_all(wb,:),t_all,...
358          frequency,Qprojected{wb},ic_elev_first{wb},...
359          turbine_discharge{wb},[],[],[]);
360       if isempty(CFG{wb}.WaterSurfaceElevationTargets)
361          L.warn('INITIALIZATION',['No reservoir ', num2str(wb),' ELWS targets
                 ↪ provided - assuming targets from projected operations W2 simulation
                 ↪ .'])
362          ELWS_targets{wb}(:,1)=[start_date+1:1:start_date+days_forward]';
363          ELWS_targets{wb}(:,2)=interp1(t_all,HWs_x0(wb,:),...
364             [start_date+1:1:start_date+days_forward])';
365          if isnan(ELWS_targets{wb}(end,2))
366             ELWS_targets{wb}(end,2)=elevtemp{wb}(end,2);
```

```
      end
   else
      ELWS_targets{wb}=dlmread(CFG{wb}.WaterSurfaceElevationTargets,'\t',1,0);
   end
      ELWS_targets{wb}(:,2)=min(ELWS_targets{wb}(:,2),ELWS_limit{wb}(2));
      ELWS_targets{wb}(:,2)=max(ELWS_targets{wb}(:,2),ELWS_limit{wb}(1));
end
clearvars wb t_all_round t_prev_round elevtemp x0_all_fix
%Soft penalty coeff for deviation from final target elevation
elev_soft_penalty_coeff_constant=[1e3 5e2];
%Water quality and elevation constraint rounding setting (10=tenths place, 100=
    ↪ hundredths place, etc.)
elev_constraint_rounding=100;
wq_constraint_rounding=100;
%Assign priority ranking for constraints on elev, DO, and temp, starting with
    ↪ highest priority first. This is used during the prescreen to see if
    ↪ constraints are even feasible
ranking={'elev','do','temp'};
%Penalty tolerance
tolerance=10^-20;

%% Initialize cache of solutions - only use if Optimize day by day is off and
    ↪ there is 1 waterbody
cache.t=t_all;
cache.x=[]; cache.HWs=[]; cache.DO=[]; cache.T=[];
cache.x=x0_all(:,:); cache.flag={'x0'}; cache.HWs=HWs_x0(1:end);
cache.DO=interp1(Qprojected{1}.CWO(Qprojected{1}.CWO(:,2)~=0,1),Qprojected{1}.
    ↪ CWO(Qprojected{1}.CWO(:,2)~=0,2),t_all(2:end));
%Fill in Nans at the end
a=cache.DO(~isnan(cache.DO)); cache.DO(isnan(cache.DO))=a(end);
turbs=interp1(Qprojected{1}.QOT_BR1_T(:,1),Qprojected{1}.QOT_BR1_T(:,2),t_all);
spills=interp1(Qprojected{1}.QOT_BR1_S(:,1),Qprojected{1}.QOT_BR1_S(:,2),t_all);
flowout_x0=turbs(2:end)+spills(2:end);
cache.DO(flowout_x0==0)=nan;
cache.T=interp1(Qprojected{1}.TWO(Qprojected{1}.TWO(:,2)~=0,1),Qprojected{1}.TWO
    ↪ (Qprojected{1}.TWO(:,2)~=0,2),t_all(2:end));
%Fill in Nans at the end
a=cache.T(~isnan(cache.T)); cache.T(isnan(cache.T))=a(end);
cache.T(flowout_x0==0)=nan;
clearvars a flowout_x0 turbs spills HWs_x0

%% Save projected operations data in Input and Output for future NARX training
if ~exist('Inputs')
   wb=1;
   Inputs{wb}.discharge_DO=[];
   Inputs{wb}.discharge_Temp=[];
end

% DO inputs and output
for wb=1:size(CFG,2)
   if WQ{wb}.DO_valid_check==1
      index=size(Inputs{wb}.discharge_DO,2);
      timesteps=[t_all(1)-max(WQ{wb}.DO_narx.inputDelays)/24:(1/24):t_all(end)
          ↪ ]';
      vars=WQ{wb}.DO_narx.input_variables;
      Inputs{wb}.discharge_DO{index+1}=[];
      for i=1:size(vars,2)
         if strfind(char(vars(1,i)),'TIN')
            flow_variable=strrep(char(vars(1,i)),'TIN','QIN');
         elseif strfind(char(vars(1,i)),'CIN')
            flow_variable=strrep(char(vars(1,i)),'CIN','QIN');
         elseif strfind(char(vars(1,i)),'TTR')
            flow_variable=strrep(char(vars(1,i)),'TTR','QTR');
         elseif strfind(char(vars(1,i)),'CTR')
```

```
424             flow_variable=strrep(char(vars(1,i)),'CTR','QTR');
425          else
426             flow_variable=char(vars(1,i));
427          end
428          if ~strcmp(char(vars(1,i)),'MET_WB1') %assume interpolation for MET
                 ↪ data
429             for ii=1:size(Q{wb}.interpolation,2)
430                if strcmp(char(Q{wb}.interpolation(1,ii)),flow_variable)
431                   break
432                end
433             end
434             if strcmp(char(Q{wb}.interpolation(3,ii)),'ON')
435                Inputs{wb}.discharge_DO{index+1}(:,i)=interp1(Q{wb}.(vars{1,i})
                       ↪ (:,1),...
436                   Q{wb}.(vars{1,i})(:,vars{2,i}+1),timesteps);
437             elseif strcmp(char(Q{wb}.interpolation(3,ii)),'OFF')
438                for iii=1:size(timesteps,1)
439                   index2=find(Q{wb}.(vars{1,i})(:,1)<=timesteps(ii),1,'last');
440                   Inputs{wb}.discharge_DO{index+1}(iii,i)=Q{wb}.(vars{1,i})(
                          ↪ index2,vars{2,i}+1);
441                end
442             end
443          else
444             Inputs{wb}.discharge_DO{index+1}(:,i)=interp1(Q{wb}.(vars{1,i})(:,1)
                    ↪ ,...
445                Q{wb}.(vars{1,i})(:,vars{2,i}+1),timesteps);
446          end
447       end
448       DO_noNAN=interp1(Qprojected{wb}.CWO(Qprojected{wb}.CWO(:,2)~=0,1),...
449          Qprojected{wb}.CWO(Qprojected{wb}.CWO(:,2)~=0,2),timesteps);
450       %Fill in Nans at the end
451       a=DO_noNAN(~isnan(DO_noNAN)); DO_noNAN(isnan(DO_noNAN))=a(end);
452       turbs=interp1(Qprojected{wb}.QOT_BR1_T(:,1),Qprojected{wb}.QOT_BR1_T(:,2),
              ↪ timesteps);
453       spills=interp1(Qprojected{wb}.QOT_BR1_S(:,1),Qprojected{wb}.QOT_BR1_S(:,2)
              ↪ ,timesteps);
454       flowout=turbs+spills; DO_noNAN(flowout==0)=nan;
455
456       %Output data
457       Output{wb}.discharge_DO{index+1}(:,1)=DO_noNAN;
458
459       for i=1:size(Inputs{wb}.discharge_DO,2)
460          %Convert to cells
461          Inputs_seq{wb}.discharge_DO{i}=con2seq(Inputs{wb}.discharge_DO{i}');
462          Output_seq{wb}.discharge_DO{i}=con2seq(Output{wb}.discharge_DO{i}');
463       end
464       clearvars i ii iii flow_variable index a DO_noNAN turbs spills flowout
              ↪ index2 vars timesteps
465       %Combine them all into single Input and Output cell arrays
466       Inputs_seq_mul{wb}.discharge_DO=catsamples(Inputs_seq{wb}.discharge_DO{:},
              ↪ 'pad');
467       Output_seq_mul{wb}.discharge_DO=catsamples(Output_seq{wb}.discharge_DO{:},
              ↪ 'pad');
468       clearvars b Xs Xi Ai Ts tr tr2 b yp1 TS bias narx_net_closed narx_net
              ↪ muhat sigmahat
469
470    end
471 end
472
473
474 % Temp inuts and output
475 for wb=1:size(CFG,2)
476     if WQ{wb}.Temp_valid_check==1
477       index=size(Inputs{wb}.discharge_Temp,2);
```

```
478         timesteps=[t_all(1)-max(WQ{wb}.Temp_narx.inputDelays)/24:(1/24):t_all(end)
                ↪ ]';
479         vars=WQ{wb}.Temp_narx.input_variables;
480         Inputs{wb}.discharge_Temp{index+1}=[];
481         for i=1:size(vars,2)
482            if strfind(char(vars(1,i)),'TIN')
483               flow_variable=strrep(char(vars(1,i)),'TIN','QIN');
484            elseif strfind(char(vars(1,i)),'CIN')
485               flow_variable=strrep(char(vars(1,i)),'CIN','QIN');
486            elseif strfind(char(vars(1,i)),'TTR')
487               flow_variable=strrep(char(vars(1,i)),'TTR','QTR');
488            elseif strfind(char(vars(1,i)),'CTR')
489               flow_variable=strrep(char(vars(1,i)),'CTR','QTR');
490            else
491               flow_variable=char(vars(1,i));
492            end
493            if ~strcmp(char(vars(1,i)),'MET_WB1') %assume interpolation for MET
                   ↪ data
494               for ii=1:size(Q{wb}.interpolation,2)
495                  if strcmp(char(Q{wb}.interpolation(1,ii)),flow_variable)
496                     break
497                  end
498               end
499               if strcmp(char(Q{wb}.interpolation(3,ii)),'ON')
500                  Inputs{wb}.discharge_Temp{index+1}(:,i)=interp1(Q{wb}.(vars{1,i})
                       ↪ (:,1),...
501                     Q{wb}.(vars{1,i})(:,vars{2,i}+1),timesteps);
502               elseif strcmp(char(Q{wb}.interpolation(3,ii)),'OFF')
503                  for iii=1:size(timesteps,1)
504                     index2=find(Q{wb}.(vars{1,i})(:,1)<=timesteps(ii),1,'last');
505                     Inputs{wb}.discharge_Temp{index+1}(iii,i)=Q{wb}.(vars{1,i})(
                          ↪ index2,vars{2,i}+1);
506                  end
507               end
508            else
509               Inputs{wb}.discharge_Temp{index+1}(:,i)=interp1(Q{wb}.(vars{1,i})
                       ↪ (:,1),...
510                  Q{wb}.(vars{1,i})(:,vars{2,i}+1),timesteps);
511            end
512         end
513         T_noNAN=interp1(Qprojected{wb}.TWO(Qprojected{wb}.TWO(:,2)~=0,1),...
514            Qprojected{wb}.TWO(Qprojected{wb}.TWO(:,2)~=0,2),timesteps);
515         %Fill in Nans at the end
516         a=T_noNAN(~isnan(T_noNAN)); T_noNAN(isnan(T_noNAN))=a(end);
517         turbs=interp1(Qprojected{wb}.QOT_BR1_T(:,1),Qprojected{wb}.QOT_BR1_T(:,2),
                ↪ timesteps);
518         spills=interp1(Qprojected{wb}.QOT_BR1_S(:,1),Qprojected{wb}.QOT_BR1_S(:,2)
                ↪ ,timesteps);
519         flowout=turbs+spills; T_noNAN(flowout==0)=nan;
520
521         %Output data
522         Output{wb}.discharge_Temp{index+1}(:,1)=T_noNAN;
523
524         for i=1:size(Inputs{wb}.discharge_Temp,2)
525            %Convert to cells
526            Inputs_seq{wb}.discharge_Temp{i}=con2seq(Inputs{wb}.discharge_Temp{i}')
                   ↪ ;
527            Output_seq{wb}.discharge_Temp{i}=con2seq(Output{wb}.discharge_Temp{i}')
                   ↪ ;
528         end
529         clearvars i ii iii flow_variable index a T_noNAN turbs spills flowout
                ↪ index2 vars timesteps
530
531         %Combine them all into single Input and Output cell arrays
```

```
        Inputs_seq_mul{wb}.discharge_Temp=catsamples(Inputs_seq{wb}.discharge_Temp
            ↪ {:},'pad');
        Output_seq_mul{wb}.discharge_Temp=catsamples(Output_seq{wb}.discharge_Temp
            ↪ {:},'pad');
        clearvars b Xs Xi Ai Ts tr tr2 yp1 TS bias narx_net_closed narx_net muhat
            ↪ sigmahat
        clearvars timesteps

    end
end

retraining='Y';
iter=0; best_iter=[];

%Build log file
if ~exist('results','dir')
   mkdir('results');
end
fileID=fopen(config.LogFile,'w');
fprintf(fileID,'%12s %16s %12s %18s %18s %14s %14s %14s %14s %12s %12s %12s %12s
    ↪  %12s %12s',...
'Iter','Feas_GA_pop_size','GA_pop_size','Train_SS_Clusters','Train_SS_Nearby'
    ↪ ,...
'Feval_Tot','Feval_Cache','Feval_GAtot','Feval_GAcache','Pop_stdev','ReplaceRate
    ↪ ','Proj_MWh','Tot_MWh','Proj_Dollars','Tot_Dollars');
for wb=1:size(CFG,2)
   fprintf(fileID,' %12s %12s',['Wb' num2str(wb) '_MWh'],['Wb' num2str(wb) '
      ↪ _dollars']);
end
for wb=1:size(CFG,2)
   fprintf(fileID,' %12s %12s %16s %16s %15s %15s %15s %15s',...
      ['Wb' num2str(wb) '_T_AME'],['Wb' num2str(wb) '_DO_AME'],...
      ['Wb' num2str(wb) '_T_trpt_AME'],['Wb' num2str(wb) '_DO_trpt_AME'],...
      ['Wb' num2str(wb) '_NN_T_slack'],['Wb' num2str(wb) '_NN_DO_slack'],...
      ['Wb' num2str(wb) '_W2_T_slack'],['Wb' num2str(wb) '_W2_DO_slack']);
end
fprintf(fileID,' %12s %13s %13s %12s','Best_Iter', 'x_in_initpop', 'x_in_prevpop
    ↪ ', '2-norm');
fprintf(fileID,' %12s %12s %12s\r\n','Opt_time(s)','W2_time(s)','Trn_time(s)');
fclose(fileID); clearvars fileID ans wb
```

optimization_routine.m

```
%% Optimize over days_forward

day=1; stop=0;
global funccount_cache_global funccount_tot_global
funccount_cache_global=0; funccount_tot_global=0;
if ~exist('plot_data','dir')
   mkdir('plot_data');
end
clearvars xprev tprev
for wb=1:size(CFG,2)
   x_final{wb}=[];
   %Previous turbine pattern for the year (if supplied)
   if isempty(CFG{wb}.ForecastTurbinePattern)
      xprev{wb}=actual_turb_ops(tprev_round,Qprojected{wb},elevtemp{wb},
          ↪ turbine_discharge{wb},no_of_units{wb});
   else
      prevturbpattern=dlmread(CFG{wb}.PreviousTurbinePattern,'\t',1,0);
      for i=1:size(tprev_round,2)
         index=find(prevturbpattern(:,1)<=tprev_round(i));
         xprev{wb}(i)=prevturbpattern(index(end),2);
```

```
20          end
21          clearvars i prevturbpattern index
22       end
23    end
24    clearvars wb
25    tprev=[t_all(1)-max(cell2mat(zero_gen_limit(:)))*frequency:frequency:t_all(1)];
26    xprev_ic=xprev; tprev_ic=tprev;
27
28    while stop==0
29
30       %For each day, determine if elevation, DO , and temp constraints are even
             ↪ feasible (in priority order). If not found feasible, then bounds
             ↪ defined earlier by the config files are modified. Then problem is
             ↪ optimized for maximize power (or power value)
31
32       fprintf(['OPTIMIZATION: OPTIMIZING DAY ', num2str(day), ' \n']);
33
34       WQ_subproblem{day}=WQ;
35       ELWS_limit_subproblem{day}=ELWS_limit;
36
37       %Optimization timeperiod
38       if Optimize_day_by_day==1
39          t=[start_date+day-1:frequency:start_date+day];
40       else
41          t=t_all;
42       end
43
44       %Set initial condition elevation
45       for wb=1:size(CFG,2)
46          if day==1
47             ic_elev{wb}=ic_elev_first{wb};
48             if ic_elev_first{wb}<ELWS_limit_subproblem{day}{wb}(1)
49                fprintf(['INITIALIZATION: Reservoir ', num2str(wb),' initial
                       ↪ elevation of ', num2str(ic_elev_first{wb}), ' m is less than
                       ↪ ELWS lower limit (firm constraint). Expanding ELWS limits to
                       ↪ continue with optimization. \n']);
50                ELWS_limit_subproblem{day}{wb}(1)=ic_elev_first{wb};
51             elseif ic_elev_first{wb}>ELWS_limit_subproblem{day}{wb}(2)
52                fprintf(['INITIALIZATION: Reservoir ', num2str(wb),' initial
                       ↪ elevation of ' num2str(ic_elev_first{wb}) ' m is greater than
                       ↪  ELWS upper limit (firm constraint). Expanding ELWS limits to
                       ↪  continue with optimization. \n']);
53                ELWS_limit_subproblem{day}{wb}(2)=ic_elev_first{wb};
54             end
55          else
56             ic_elev{wb}=interp1(Q{wb}.ELWS(:,1),Q{wb}.ELWS(:,2),t(1));
57          end
58       end
59
60       for wb=1:size(CFG,2)
61          %Determine x0, actual turbine operations, to seed initial population
62          if Optimize_day_by_day==1
63             x0(wb,:)=x0_all(wb,(day-1)*(1/frequency)+1:day*(1/frequency));
64          else
65             x0(wb,:)=x0_all(wb,:);
66          end
67          [~, y_dollars1]=power_value(x0(wb,:),t,cost_curve_MW{wb},...
68             MW_rating{wb});
69          if size(ELWS_targets{wb}(:,1),1)==1
70             elev_soft_penalty_coeff{day}(wb)=interp1(ELWS_limit_subproblem{day}{wb
                    ↪ }(:),...
71                elev_soft_penalty_coeff_constant,...
72                ELWS_targets{wb}(:,2),'linear','extrap')*y_dollars1; %$/m with cost
                    ↪ curve, MWh/m with all cc=1
```

```
73        else
74           elev_soft_penalty_coeff{day}(wb)=interp1(ELWS_limit_subproblem{day}{wb
                 ↪ }(:),...
75              elev_soft_penalty_coeff_constant,...
76              interp1(ELWS_targets{wb}(:,1),ELWS_targets{wb}(:,2),start_date+day)
                    ↪ ,...
77              'linear','extrap')*y_dollars1; %$/m with cost curve, MWh/m with all
                    ↪ cc=1
78        end
79        clearvars y_dollars1
80
81        if (iter==1 && Optimize_day_by_day==0) || Optimize_day_by_day==1
82           %Find possible values for x(1) (based on previous zero_gen_limit turbs)
83           options=[0:no_of_units{wb}];
84           % (1) Eliminate options based on change in active unit violations
85           if ~isnan(max_hrly_unit_change{wb})
86              auvoptions=[xprev{wb}(end)-max_hrly_unit_change{wb}:...
87                 xprev{wb}(end)+max_hrly_unit_change{wb}];
88              options=intersect(options,auvoptions);
89           end
90           % (2) Non-integer constraint (assumed in selection algorithm)
91           % (3) Eliminate options based on zero generation hourly limit
92           if ~isnan(zero_gen_limit{wb})
93              if sum(xprev{wb}(end-zero_gen_limit{wb}+1:end))==0
94                 zghloptions=[1:no_of_units{wb}]; %if previous zero_gen_limit hrs
                       ↪ had zero total flow, must have flow next hr
95                 options=intersect(options,zghloptions);
96              end
97           end
98           % (4) Eliminate options that violate oscillations constraint - violates
                 ↪  whenever the number of turbines increases and then decreases
                 ↪ within 2 hours, or vice versa
99           allopt=[0:no_of_units{wb}];
100          if xprev{wb}(end-1)<xprev{wb}(end) %if prev turbs increasing
101             osceptions=allopt(allopt>=xprev{wb}(end));
102             options=intersect(options,osceptions);
103          elseif xprev{wb}(end-1)==xprev{wb}(end) %need 3 hrs btwn ramping up and
                    ↪  down
104             if xprev{wb}(end-2)<xprev{wb}(end-1) %ramping up
105                osceptions=allopt(allopt>=xprev{wb}(end));
106                options=intersect(options,osceptions);
107             elseif xprev{wb}(end-2)>xprev{wb}(end-1) %ramping down
108                osceptions=allopt(allopt<=xprev{wb}(end));
109                options=intersect(options,osceptions);
110             elseif xprev{wb}(end-2)==xprev{wb}(end-1)
111                %do nothing -->3 consecutive hours between ramping up and down
                       ↪ satisfied
112             end
113          elseif xprev{wb}(end-1)>xprev{wb}(end) %if prev turbs decreasing
114             osceptions=allopt(allopt<=xprev{wb}(end));
115             options=intersect(options,osceptions);
116          end
117          x1_options{wb}=options;
118          if isempty(x1_options{wb})
119             fprintf('OPTIMIZATION: Based on previous turbine pattern, there is
                    ↪ no feasible first hour turbine level. \n');
120             return
121          end
122          clearvars tprev options auvoptions zghloptions allopt osceptions
123       end
124    end
125    clearvars wb
126
127    %Determine if elevation, DO, and temp constraints are feasible (based on
```

```
        ↪ ranking order) and adjust bounds in this order if necessary
128     fprintf('OPTIMIZATION: Check constraint feasibilities and adjust if needed. \
            ↪ n');
129     if iter==1
130        y=penalty_fcn(trainingpop,t,frequency,Q,ic_elev,...
131           turbine_discharge,ELWS_limit_subproblem{day},max_hrly_unit_change,...
132           WQ_subproblem{day},zero_gen_limit,xprev,ELWS_targets,tolerance,cache,
                  ↪ Optimize_day_by_day);
133        y=sum(y,2); feasible_option1=trainingpop(y==0,:); clearvars y
134        %Check if x0 is feasible - include it if it is
135        y=penalty_fcn(reshape(x0',1,[]),t,frequency,Q,ic_elev,...
136           turbine_discharge,ELWS_limit_subproblem{day},max_hrly_unit_change,...
137           WQ_subproblem{day},zero_gen_limit,xprev,ELWS_targets,tolerance,cache,
                  ↪ Optimize_day_by_day);
138        best_fvals(day,1)=obj_fcn(reshape(x0',1,[]),t,cost_curve_MW,MW_rating,...
139           elev_soft_penalty_coeff{day},ELWS_targets,...
140           frequency,Q,ic_elev,turbine_discharge,cache,Optimize_day_by_day);
141        %Check to see if any values in x0>no_of_units
142        over_no_of_units=0;
143        for wb=1:size(CFG,2)
144           if any(x0(wb,:)>no_of_units{wb}) over_no_of_units=1; end
145        end
146        if ~all(y==0) || over_no_of_units==1
147           fprintf('OPTIMIZATION: x0 is not feasible with respect to previous
                  ↪ optimal solution. \n');
148        else
149           fprintf('OPTIMIZATION: x0 is feasible with respect to previous optimal
                  ↪ solution. \n');
150           feasible_option1=[reshape(x0',1,[]);feasible_option1];
151        end
152        clearvars over_no_of_units
153     end
154     funccount_tot(day,1)=funccount_tot_global;
155     funccount_cache(day,1)=funccount_cache_global;
156     funccount_cache_global=0; funccount_tot_global=0; %reset to 0 to restart
            ↪ count
157
158     %Create initial population if iter=1. Otherwise, start with prev gen
            ↪ population and replace a percentage of the population (rank by
            ↪ weighted avg constraint violation and pick the worst ones) with newly
            ↪ generated points.
159     fprintf('OPTIMIZATION: Finding initial population to seed genetic algorithm.
            ↪ \n');
160     if iter==1
161        feasible_options=pop0; replacement_rate=0;
162     else
163        feasible_options=population{iter-1}; replacement_rate=0;
164        if strcmp(ReplacementOnOff,'ON')
165           %Rank pop members by weighted avg constraint violation (use modified
                  ↪ penalty function that computes all constraints)
166           violations=penalty_fcn_inf(population{iter-1},t,frequency,Q,ic_elev,...
167              turbine_discharge,ELWS_limit_subproblem{day},...
168              max_hrly_unit_change,WQ_subproblem{day},zero_gen_limit,...
169              xprev,ELWS_targets,tolerance,cache,Optimize_day_by_day);
170           %Normalize each column and average across, then rank population members
                  ↪  from worst (least feasible) to best (feasible). Then amongst
                  ↪ feasible pop members, rank by fval
171           normc=violations(:,:); normc2=[];
172           for i=1:size(normc,2)
173              if ~all(normc(:,i)==normc(1,i)) normc2=[normc2 normc(:,i)]; end
174           end
175           mindata = min(normc2); maxdata = max(normc2);
176           normc2 = bsxfun(@rdivide, bsxfun(@minus, normc2, mindata), maxdata -
                  ↪ mindata);
```

```
177         meanc=mean(normc2,2); [meanc,b]=sort(meanc,'descend');
178         %Set the replacement rate for the next generation
179         replacement_rate=0.2;
180         replacement_size=round(ga_pop_size*replacement_rate);
181         if replacement_size>sum(meanc>0)
182            %Rank by fval
183            bb=b(meanc==0); a=FitnessFunction(population{iter-1}(bb,:));
184            [~,bbb]=sort(a,'descend'); b(meanc==0)=bb(bbb);
185            clearvars a bb bbb
186         end
187         %Generate new replacement pop members
188         WQ_r=WQ_subproblem{day}; wb=1;
189         WQ_r{wb}.DO_limit=nan(size(WQ{wb}.DO_limit)); WQ_r{wb}.Temp_limit=nan(
              ↪ size(WQ{wb}.Temp_limit));
190         [replacements]=...
191            create_replacements(replacement_size,[],...
192            x1_options,frequency,Q,ic_elev,MW_rating,no_of_units,t,...
193            max_hrly_unit_change,zero_gen_limit,turbine_discharge,...
194            ELWS_limit_subproblem{day},WQ_r,cost_curve_MW,xprev,...
195            elev_soft_penalty_coeff{day},ELWS_targets,tolerance,cache,
                 ↪ Optimize_day_by_day,...
196            transition_matrix);
197         %Sub out the replacement pop members
198         feasible_options(b(1:replacement_size),:)=replacements;
199      end
200   end
201   funccount_tot(day,2)=funccount_tot_global;
202   funccount_cache(day,2)=funccount_cache_global;
203   funccount_cache_global=0; funccount_tot_global=0; %reset to 0 to restart
        ↪ count
204   if isempty(feasible_options)
205      fprintf('OPTIMIZATION: No feasible solutions found during initialization \
           ↪ n');
206      return
207   end
208   clearvars objfcn feasible_option1 b c normc i normc2 mindata maxdata meanc b
        ↪ replacements
209
210   %Set optimization algorithm options
211   FitnessFunction = @(x) -obj_fcn(x,t,cost_curve_MW,...
212      MW_rating,elev_soft_penalty_coeff{day},...
213      ELWS_targets,frequency,Q,ic_elev,...
214      turbine_discharge,cache,Optimize_day_by_day);
215   mycon= @(x) penalty_fcn(x,t,frequency,Q,ic_elev,...
216      turbine_discharge,ELWS_limit_subproblem{day},...
217      max_hrly_unit_change,WQ_subproblem{day},zero_gen_limit,...
218      xprev,ELWS_targets,tolerance,cache,Optimize_day_by_day);
219   opt = gaoptimset(...
220      'Display','iter','Vectorized','on','Generations',GAgenerations, ...
221      'PopulationSize',ga_pop_size,...
222      'EliteCount',ceil(0.05*ga_pop_size),...
223      'InitialPopulation',feasible_options,...
224      'StallGenLimit',2,'TolFun',tolerance,'TolCon',tolerance,...
225      'CrossoverFcn',@crossoversinglepoint,'CrossoverFraction',.95,...
226      'CreationFcn',@int_pop,'MutationFcn',@int_mutation,...
227      'InitialPenalty',10^10);
228   nVar = size(CFG,2)*(size(t,2)-1);
229   %Set dv lower and upper bounds, narrowed considering max_hrly_unit_change,
        ↪ for both reservoirs
230   for wb=1:size(CFG,2)
231      lb(wb,:)=0*ones(1,size(t,2)-1); lb(wb,1)=x1_options{wb}(1);
232      for i=2:no_of_units{wb}
233         lb(wb,i)=lb(wb,i-1)-max_hrly_unit_change{wb};
234      end
```

```
235         lb(wb,:)=max(0,lb(wb,:));
236         ub(wb,:)=no_of_units{wb}*ones(1,size(t,2)-1);
237         ub(wb,1)=x1_options{wb}(end);
238         for i=2:no_of_units{wb}
239             ub(wb,i)=ub(wb,i-1)+max_hrly_unit_change{wb};
240         end
241         ub(wb,:)=min(no_of_units{wb},ub(wb,:));
242         clearvars i
243     end
244     lb=reshape(lb',1,[]); ub=reshape(ub',1,[]);
245
246     %Run GA
247     fprintf('OPTIMIZATION: Begin running genetic algorithm. \n');
248     [x,fval,~,~,population{iter},scores]=ga(FitnessFunction,nVar,[],[],[],[],lb,
          ↪ ub,...
249         mycon,[],opt);
250     %Was x in feasible_options?
251     x_in_initpop='NO'; x_in_prevpop='NO';
252     if ismember(x,feasible_options,'rows')
253         fprintf('x was in feasible_options \n');
254         x_in_initpop='YES';
255     end
256     if iter==1 x_in_prevpop='n/a';
257     else
258         if ismember(x,population{iter-1},'rows')
259             fprintf('x was in prev pop \n');
260             x_in_prevpop='YES';
261         end
262     end
263     funccount_tot(day,3)=funccount_tot_global;
264     funccount_cache(day,3)=funccount_cache_global;
265     funccount_cache_global=0; funccount_tot_global=0; %reset to 0 to restart
          ↪ count
266     best_fvals(day,3)=-fval;
267
268     %Calculate stdev of population (scale first to [-1,1]) - EXPAND TO
          ↪ MULTIRESERVOIR LATER
269     if Optimize_day_by_day==0
270         pop_scaled=(2*population{iter}/no_of_units{1})-1;
271         for variables=1:size(t,2)-1 standarddevs(variables)=std(pop_scaled(:,
              ↪ variables)); end
272         SD(iter+1)=mean(standarddevs);
273         if iter==1
274             pop_scaled=(2*feasible_options/no_of_units{1})-1;
275             for variables=1:size(t,2)-1 standarddevs(variables)=std(pop_scaled(:,
                  ↪ variables)); end
276             SD(1)=mean(standarddevs);
277         end
278         clearvars variables pop_scaled standarddevs
279     end
280     if iter==1 SSD(iter)=SD(iter); end
281     SSD(iter+1)=SSD(iter)+0.5*(SD(iter+1)-SSD(iter));
282
283
284     %Split up rows of x to separate reservoirs
285     for wb=1:size(CFG,2)
286         x_final{wb}=[x_final{wb} ...
287             x(:,wb*(size(t,2)-1)-(size(t,2)-2):wb*(size(t,2)-1))];
288     end
289     for wb=1:size(CFG,2)
290         x_final_all{iter}(wb,:)=x_final{wb};
291     end
292     clearvars wb fval lb ub opt feasible_options
293
```

```
294     %Update elevations and discharges/inflows in Q before going on to next
295     %day
296     Q=updateQ(Q,CFG,x_final,t,frequency,ic_elev,turbine_discharge,...
297        WQ_subproblem{day},xprev,ELWS_targets,cache,Optimize_day_by_day);
298
299     %Compute total y_dollars
300     clearvars elev_soft_penalty_coeff
301     for wb=1:size(CFG,2)
302        if Optimize_day_by_day==1
303           if iter==1
304              [y_MWh(wb,1), y_dollars(wb,1)]=power_value(x0_all(wb,1:day*(1/
                     ↪ frequency)),t_all(1:1+day*(1/frequency)),cost_curve_MW{wb
                     ↪ },...
305                 MW_rating{wb});
306              elev_soft_penalty_coeff{wb}=interp1(ELWS_limit{wb}(:)',...
307                 elev_soft_penalty_coeff_constant,ELWS_targets{wb}(day),...
308                 'linear','extrap')*y_dollars(wb,1); %$/m with cost curve, MWh/m
                        ↪ with all cc=1
309           end
310           [y_MWh(wb,2), y_dollars(wb,2)]=power_value(x_final{wb},t_all(1:1+day
                  ↪ *(1/frequency)),cost_curve_MW{wb},...
311              MW_rating{wb});
312        else
313           if iter==1
314              [y_MWh(wb,1), y_dollars(wb,1)]=power_value(x0_all,t_all,
                     ↪ cost_curve_MW{wb},...
315                 MW_rating{wb});
316              elev_soft_penalty_coeff{wb}=interp1(ELWS_limit{wb}(:)',...
317                 elev_soft_penalty_coeff_constant,t_all(end),...
318                 'linear','extrap')*y_dollars(wb,1); %$/m with cost curve, MWh/m
                        ↪ with all cc=1
319           end
320           [y_MWh(wb,2), y_dollars(wb,2)]=power_value(x_final{wb},t_all,
                  ↪ cost_curve_MW{wb},...
321              MW_rating{wb});
322        end
323     end
324     y_MWh_total=sum(y_MWh(1:size(CFG,2),:),1);
325     y_dollars_total=sum(y_dollars(1:size(CFG,2),:),1);
326
327     %Will run W2 for best x from optimization (if it hasn't already been sampled)
           ↪  and then compute performance (AME)
328     trainingpop=[]; cache_size_pre=size(cache.x,1);wb=1;
329     if ~ismember(x,cache.x,'rows')
330        trainingpop(1,:)=x; correction=0;
331        cache.flag{size(cache.flag,1)+1,1}={'bestx'};
332     else
333        correction=1;
334     end
335
336     if strcmp(AdditionalSamplingOnOff,'ON')
337        training_ss_clusters=0;
338        if correction==1
339           training_ss_nearby=4;
340        else
341           training_ss_nearby=3;
342        end
343     else
344        training_ss_clusters=0; training_ss_nearby=0;
345     end
346     training_ss_clusters_reset=training_ss_clusters; training_ss_nearby_reset=
           ↪ training_ss_nearby;
347     %kmeans clustering on population set and pick one from each cluster to run
           ↪ through W2
```

```
348     if (training_ss_clusters+training_ss_nearby)>0
349        ii=1; pop=population{iter}; wb=1;
350        %Remove points that aren't feasible wrt constraints other than WQ
351        violations=mycon(pop); violations2=sum(violations(:,1:53),2);
352        ia=find(violations2==0); %find the pop members feasible wrt all
               ↪ constraints except WQ
353        pop=pop(ia,:);
354        if feasible_soln_found==0
355           [bb,b]=sortrows([violations(ia,54) FitnessFunction(pop)],[1 2]);
356        else
357           pop=pop(FitnessFunction(pop)<FitnessFunction(x_final_all{best_iter(iter
                   ↪ -1)}),:);
358           if ~isempty(pop)
359              violations=mycon(pop);
360              [bb,b]=sortrows([violations(:,54) FitnessFunction(pop)],[1 2]);
361           end
362        end
363        if ~isempty(pop) pop=pop(b,:); end
364        clearvars ia violations violations2 b bb
365        if training_ss_clusters>0
366           for a=1:500
367              [idx(:,a),~,~,D{a}]=kmeans(pop,training_ss_clusters);
368              B=unique(idx(:,a));
369              group_var(a)=var(histc(idx(:,a),B));
370           end
371           %Pick the cluster that minimizes the max group size (i.e., results in
                   ↪ fairly even distribution)
372           [~,a]=min(group_var); idx=idx(:,a); D=D{a}; clearvars a B group_var
373        end
374        if training_ss_nearby>0
375           e=[1:size(pop,1)]';
376        end
377        for i=2:(training_ss_clusters+training_ss_nearby)+1
378           if any(i==2:(1+training_ss_nearby))
379              if ~isempty(trainingpop)
380                 while isempty(e) || ismember(pop(e(ii),:),cache.x,'rows') || ...
381                       ismember(pop(e(ii),:),trainingpop,'rows')
382                    if (ii+1)>size(e,1) fprintf('No new points to sample. \n')
383                       training_ss_nearby_reset=training_ss_nearby_reset-1;
384                       break
385                    else ii=ii+1; end
386                 end
387              else
388                 while isempty(e) || ismember(pop(e(ii),:),cache.x,'rows')
389                    if (ii+1)>size(e,1) fprintf('No new points to sample. \n')
390                       training_ss_nearby_reset=training_ss_nearby_reset-1;
391                       break
392                    else ii=ii+1; end
393                 end
394              end
395              if (ii+1)>size(e,1) %do nothing
396              else trainingpop(size(trainingpop,1)+1,:)=pop(e(ii),:); ii=ii+1;
                       ↪ cache.flag{size(cache.flag,1)+1,1}={'nearby'}; end
397           elseif ~isempty(pop)
398              b=find(idx==i-1-training_ss_nearby);
399              %Pick randomly from each cluster
400              a=randsample(b,1);
401              if ~isempty(trainingpop)
402                 while ismember(pop(a,:),cache.x,'rows') || ...
403                       ismember(pop(a,:),trainingpop,'rows')
404                    b=setdiff(b,a);
405                    if isempty(b) a=[]; fprintf('No new points to sample. \n')
406                       training_ss_clusters_reset=training_ss_clusters_reset-1;
407                       break
```

```
                else a=randsample(b,1); end
            end
        else
            while ismember(pop(a,:),cache.x,'rows')
                b=setdiff(b,a);
                if isempty(b) a=[]; fprintf('No new points to sample. \n')
                    training_ss_clusters_reset=training_ss_clusters_reset-1;
                    break
                else a=randsample(b,1); end
            end
        end
        if ~isempty(a)
            trainingpop(size(trainingpop,1)+1,:)=pop(a,:); cache.flag{size(
                ↪ cache.flag,1)+1,1}={'cluster'};
        end
    end
  end
end
training_ss_clusters=training_ss_clusters_reset; training_ss_nearby=
    ↪ training_ss_nearby_reset;
clearvars training_ss_clusters_reset training_ss_nearby_reset

%Create Qtrainingpop for each trainingpop entry (QOT_BR1_T, QOT_BR1_S, ELWS,
    ↪ CWO, TWO)
if size(trainingpop,1)>0
    for i=1:size(trainingpop,1)
        xtr{1}=trainingpop(i,:);
        Qtrainingpop{i}=updateQ(Q,CFG,xtr,t,frequency,ic_elev,turbine_discharge
            ↪ ,...
            WQ_subproblem{day},xprev,ELWS_targets,cache,Optimize_day_by_day);
    end
end
%Run each row in trainingpop through W2 (only works for 1-day, 1-wb problems
    ↪ for now), and update cache with these values as well
timing(2)=0;
if size(trainingpop,1)>0
    w2timing=tic;
    for trindex=1:size(trainingpop,1)
        wb=1;
        if correction==0 && trindex==1
            fprintf(['Running W2 validation simulation for reservoir #', num2str
                ↪ (wb),'. \n']);
            directory=['results/w2_iter' num2str(iter) '_wb' num2str(wb)];
        else
            if size(trainingpop,1)>(training_ss_clusters+training_ss_nearby)
                fprintf(['Running training point ' num2str(trindex-1) ' for
                    ↪ reservoir #', num2str(wb),'. \n']);
                directory=['results/w2_iter' num2str(iter) '_trpt' num2str(
                    ↪ trindex-1) '_wb' num2str(wb)];
            else
                fprintf(['Running training point ' num2str(trindex) ' for
                    ↪ reservoir #', num2str(wb),'. \n']);
                directory=['results/w2_iter' num2str(iter) '_trpt' num2str(
                    ↪ trindex) '_wb' num2str(wb)];
            end
        end
        runW2trainingpop;
    end
    while istaskrunning('w2.exe') end %is w2 still running? if so, hold on
    system('taskkill /F /IM cmd.exe'); cache_size_pre=size(cache.x,1);
    for trindex=1:size(trainingpop,1)
        wb=1;
        if correction==0 && trindex==1
            directory=['results/w2_iter' num2str(iter) '_wb' num2str(wb)];
```

```
462             else
463                 if size(trainingpop,1)>(training_ss_clusters+training_ss_nearby)
464                     directory=['results/w2_iter' num2str(iter) '_trpt' num2str(
                            ↪ trindex-1) '_wb' num2str(wb)];
465                 else
466                     directory=['results/w2_iter' num2str(iter) '_trpt' num2str(
                            ↪ trindex) '_wb' num2str(wb)];
467                 end
468             end
469             runW2trainingpop_part2;
470         end
471         timing(2)=toc(w2timing);
472         %Compute AME for each of these new training points using current NARX
                ↪ surrogate model
473         for trindex=1:size(trainingpop,1);
474             x_trpt=trainingpop(trindex,:); wb=1; compute_AME_trpt;
475             AME_trpt.DO{iter}(trindex)=nanmean(abs(cache.DO(b,:)-DO_pred));
476             AME_trpt.T{iter}(trindex)=nanmean(abs(cache.T(b,:)-T_pred));
477             AME_trpt.DO_error{iter}(trindex,:)=cache.DO(b,:)-DO_pred;
478             AME_trpt.T_error{iter}(trindex,:)=cache.T(b,:)-T_pred;
479         end
480         %Compute AME for each old training point, for comparison against new
                ↪ training points
481         for index=1:cache_size_pre
482             x_trpt=cache.x(index,:); wb=1; compute_AME_trpt;
483             AME_trpt.DO_old{iter}(index)=nanmean(abs(cache.DO(b,:)-DO_pred));
484             AME_trpt.T_old{iter}(index)=nanmean(abs(cache.T(b,:)-T_pred));
485             AME_trpt.DO_old_error{iter}(index,:)=cache.DO(b,:)-DO_pred;
486             AME_trpt.T_old_error{iter}(index,:)=cache.T(b,:)-T_pred;
487         end
488         %Compute averages
489         AME_trpt.DO_avg(iter)=mean(AME_trpt.DO{iter});
490         AME_trpt.T_avg(iter)=mean(AME_trpt.T{iter});
491         AME_trpt.DO_old_avg(iter)=mean(AME_trpt.DO_old{iter});
492         AME_trpt.T_old_avg(iter)=mean(AME_trpt.T_old{iter});
493     else
494         AME_trpt.T_avg(iter)=NaN; AME_trpt.DO_avg(iter)=NaN;
495     end
496
497     %Save the AME for the best solution found this generation
498     wb=1; x_trpt=x_final_all{iter}; compute_AME_trpt;
499     AME{wb}.DO=nanmean(abs(cache.DO(b,:)-DO_pred));
500     AME{wb}.T=nanmean(abs(cache.T(b,:)-T_pred));
501
502     %Determine the index in cache corresponding to the best solution from last
            ↪ generation
503     [~,b]=ismember(x_final_all{iter},cache.x,'rows');
504     %Compute WQ average slack using W2 results
505     slack_compute=cache.DO(b,:)';
506     non_nan_count=sum(~isnan(slack_compute),1);
507     if ~isnan(WQ{wb}.DO_limit(1))
508         slacks{wb}.DO.W2=sum(-min(0,[slack_compute-WQ{wb}.DO_limit(1)]),1)./
                ↪ non_nan_count;
509     elseif ~isnan(WQ{wb}.DO_limit(2))
510         slacks{wb}.DO.W2=sum(-min(0,[slack_compute-WQ{wb}.DO_limit(2)]),1)./
                ↪ non_nan_count;
511     else
512         slacks{wb}.DO.W2=0;
513     end
514     %Compute WQ average slack using NN results
515     slack_compute=DO_pred';
516     non_nan_count=sum(~isnan(slack_compute),1);
517     if ~isnan(WQ{wb}.DO_limit(1))
518         slacks{wb}.DO.NN=sum(-min(0,[slack_compute-WQ{wb}.DO_limit(1)]),1)./
```

```
        ↪ non_nan_count;
519     elseif ~isnan(WQ{wb}.DO_limit(2))
520        slacks{wb}.DO.NN=sum(-min(0,[slack_compute-WQ{wb}.DO_limit(2)]),1)./
               ↪ non_nan_count;
521     else
522        slacks{wb}.DO.NN=0;
523     end
524     %Compute WQ average slack using W2 results
525     slack_compute=cache.T(b,:)';
526     non_nan_count=sum(~isnan(slack_compute),1);
527     if ~isnan(WQ{wb}.Temp_limit(1))
528        slacks{wb}.T.W2=sum(-min(0,[slack_compute-WQ{wb}.Temp_limit(1)]),1)./
               ↪ non_nan_count;
529     elseif ~isnan(WQ{wb}.Temp_limit(2))
530        slacks{wb}.T.W2=sum(-min(0,[slack_compute-WQ{wb}.Temp_limit(2)]),1)./
               ↪ non_nan_count;
531     else
532        slacks{wb}.T.W2=0;
533     end
534     %Compute WQ average slack using NN results
535     slack_compute=T_pred';
536     non_nan_count=sum(~isnan(slack_compute),1);
537     if ~isnan(WQ{wb}.Temp_limit(1))
538        slacks{wb}.T.NN=sum(-min(0,[slack_compute-WQ{wb}.Temp_limit(1)]),1)./
               ↪ non_nan_count;
539     elseif ~isnan(WQ{wb}.Temp_limit(2))
540        slacks{wb}.T.NN=sum(-min(0,[slack_compute-WQ{wb}.Temp_limit(2)]),1)./
               ↪ non_nan_count;
541     else
542        slacks{wb}.T.NN=0;
543     end
544     clearvars W2_no0s_smooth index2 W2_no0s str slack_compute non_nan_count b
545     for wb=1:size(CFG,2)
546        results.AME(iter,wb*2-1:wb*2)=[AME{wb}.T,AME{wb}.DO];
547        results.slacks(iter,wb*2-1:wb*2)=[slacks{wb}.T.W2,slacks{wb}.DO.W2];
548     end
549     clearvars turb_discharges spill_discharges b s z zz zzz distances
           ↪ distance_mins start_index w2runstiming bestsolniter index pop DO_pred
           ↪ T_pred w2timing trindex xtr idx f i a b D wb D2 correction directory
           ↪ distance_to_soln ii e d
550
551     %Determine best iteration
552     results.dollars(iter)=y_dollars_total(2);
553     if isempty(best_iter)
554        best_iter(iter)=iter;
555     else
556        if all((results.slacks(iter,:)-results.slacks(best_iter(iter-1),:))<=0)
557           if all((results.slacks(iter,:)-results.slacks(best_iter(iter-1),:))==0)
558              if (results.dollars(iter)-results.dollars(best_iter(iter-1)))>0
559                 best_iter(iter)=iter;
560              else
561                 best_iter(iter)=best_iter(iter-1);
562              end
563           else
564              best_iter(iter)=iter; feasible_soln_found=1;
565           end
566        else
567           best_iter(iter)=best_iter(iter-1);
568        end
569     end
570
571     if Optimize_day_by_day==0
572        stop=1;
573     else
```

```
574         if day~=days_forward
575             day=day+1;
576             for wb=1:size(CFG,2)
577                 xprev{wb}=[xprev_ic{wb} x_final{wb}];
578             end
579         else
580             stop=1;
581         end
582     end
583 
584     funccount_tot(day,4)=funccount_tot_global;
585     funccount_cache(day,4)=funccount_cache_global;
586     funccount_cache_global=0; funccount_tot_global=0; %reset to 0 to restart
        ↪ count
587 
588 end
589 
590 %Sum funccount_tot
591 funccount_ga_tot=funccount_tot(day,3);
592 funccount_ga_cache=funccount_cache(day,3);
593 funccount_tot=sum(sum(funccount_tot));
594 funccount_cache=sum(sum(funccount_cache));
595 clear global funccount_cache_global funccount_tot_global
596 
597 %Compute total y_dollars
598 clearvars elev_soft_penalty_coeff
599 for wb=1:size(CFG,2)
600     if Optimize_day_by_day==1
601         if iter==1
602             [y_MWh(wb,1), y_dollars(wb,1)]=power_value(x0_all(wb,1:day*(1/frequency
                ↪ )),t_all(1:1+day*(1/frequency)),cost_curve_MW{wb},...
603                 MW_rating{wb});
604             elev_soft_penalty_coeff{wb}=interp1(ELWS_limit{wb}(:)',...
605                 elev_soft_penalty_coeff_constant,ELWS_targets{wb}(day),...
606                 'linear','extrap')*y_dollars(wb,1); %$/m with cost curve, MWh/m with
                    ↪  all cc=1
607         end
608         [y_MWh(wb,2), y_dollars(wb,2)]=power_value(x_final{wb},t_all(1:1+day*(1/
            ↪ frequency)),cost_curve_MW{wb},...
609             MW_rating{wb});
610     else
611         if iter==1
612             [y_MWh(wb,1), y_dollars(wb,1)]=power_value(x0_all,t_all,cost_curve_MW{
                ↪ wb},...
613                 MW_rating{wb});
614             elev_soft_penalty_coeff{wb}=interp1(ELWS_limit{wb}(:)',...
615                 elev_soft_penalty_coeff_constant,t_all(end),...
616                 'linear','extrap')*y_dollars(wb,1); %$/m with cost curve, MWh/m with
                    ↪  all cc=1
617         end
618         [y_MWh(wb,2), y_dollars(wb,2)]=power_value(x_final{wb},t_all,cost_curve_MW
            ↪ {wb},...
619             MW_rating{wb});
620     end
621 end
622 y_MWh_total=sum(y_MWh(1:size(CFG,2),:),1);
623 y_dollars_total=sum(y_dollars(1:size(CFG,2),:),1);
624 clearvars wb
```

activeunits_to_discharges.m

```
1 function [turb_discharges,spill_discharges,HWs,TWs,Storage] = ...
2     activeunits_to_discharges(x,t,frequency,Q,ic_elev,...
```

```
   turbine_discharge,ELWS_targets,mainstem_inflows_t,mainstem_inflows_Q,...
   Optimize_day_by_day)

% Calculates discharges and HWs and TWs from time series of number of
% active units
%
% Inputs:
% x - hourly turbine time series (as rows for vectorizing!), integers
% between 0 and no_of_units
% t  time series of JDAY values
% frequency - frequency of predictions (hourly=1/24)
% Q - all other inflows and outflows, interpolation settings,
% storage-elev curve, and tailwater curve (all in meters)
% ic_elev - initial condition (meters)
% turbine_discharge - turbine discharge curve at fixed MW level, with
% col 1 in meters and col 2 in cms
% ELWS_targets - 2 column matrix with JDAY in col1 and elevation target
% in col2. Leave empty if want to backcalculate spill
% mainstem_inflows_t - vector of JDAY values that correspond to
% mainstem_inflows_Q
% mainstem_inflows_Q - if applicable (wb~=1), rows of incoming flows from
% upstream reservoir correlated to times in mainstem_inflows_t
% Optimize_day_by_day - 1 if optimizing daily, 0 if optimizing all together
% Outputs:
% turb_discharges  turbine discharge time series in cms
% spill_discharges - spill discharge in cms
% HWs - headwater time series in m
% TWs - tailwater time series in m
% Storage - storage time series in cubic meters

if isempty(x)
   turb_discharges=[]; spill_discharges=[]; HWs=[]; TWs=[]; Storage=[];
else

JDAY_initial=t(1);

%Number of x scenarios being tested
n=size(x,1);

if n<1
   fprintf('Active units to discharges code --> x is empty!')
   return
end

%Initial condition
clearvars HWs Storage turb_discharges TWs
HWs(1,1:n)=ic_elev;
Storage(1:n,1)=interp1(Q.SE_meters_m3(:,1),Q.SE_meters_m3(:,2),HWs(1,1));
index1=find(Q.QOT_BR1_T(:,1)<=JDAY_initial);
index2=find(Q.QOT_BR1_S(:,1)<=JDAY_initial);
turb_discharges(1:n,1)=Q.QOT_BR1_T(index1(end),2);
tot_discharge=Q.QOT_BR1_T(index1(end),2)+Q.QOT_BR1_S(index2(end),2);
TWs(1:n,1)=interp1(Q.tw_curve_cms_m(:,1),Q.tw_curve_cms_m(:,2), ...
   tot_discharge);
clearvars index1 index2 tot_discharge

%Compute discharge (cms) per unit at first timestep using prev hr HW and TW
head=HWs(1,:)'-TWs(:,1);
unit_discharges=interp1(turbine_discharge(:,1),turbine_discharge(:,2), ...
   head);
unit_discharges(head>=turbine_discharge(end,1))=turbine_discharge(end,2);
unit_discharges(head<=turbine_discharge(1,1))=turbine_discharge(1,2);
turb_discharges(1:n,2)=unit_discharges.*x(:,1);
clearvars head unit_discharges
```

```
67
68  %Compute HW elevs for every scenario
69  for i=2:size(t,2)
70     elevation=HWs(i-1,:);
71     turbs=turb_discharges(:,i-1:i);
72     if isempty(ELWS_targets) %If testing projected operations
73        HWs(i-1:i,:)=Elevation_massbalance_vectorized(turbs,[],...
74           t(i-1),t(i),frequency,Q,elevation,mainstem_inflows_t,...
75           mainstem_inflows_Q);
76     else %If testing new operations, assuming no spill flow here
77        HWs(i-1:i,:)=Elevation_massbalance_vectorized(turbs,...
78           zeros(size(turbs)),t(i-1),t(i),frequency,Q,elevation,...
79           mainstem_inflows_t,mainstem_inflows_Q);
80     end
81     clearvars elevation turbs
82     %Compute storage and TWs
83     %If too full and overtops SE curve (or drains and empties), linearly
          ↪ extrapolate
84     Storage(:,i)=interp1(Q.SE_meters_m3(:,1),Q.SE_meters_m3(:,2),...
85        HWs(i,:)',’linear’,’extrap’);
86     if isempty(ELWS_targets) %if testing projected operations
87        index2=find(Q.QOT_BR1_S(:,1)<=t(i));
88        tot_discharge=turb_discharges(:,i)+Q.QOT_BR1_S(index2(end),2);
89        clearvars index2
90     else %if testing new operations, assuming no spill flow here
91        tot_discharge=turb_discharges(:,i)+0; %assume no spill
92     end
93     TWs(:,i)=interp1(Q.tw_curve_cms_m(:,1),Q.tw_curve_cms_m(:,2), ...
94        tot_discharge,’linear’,’extrap’);
95     clearvars tot_discharge
96     %Compute total turbine flowrate
97     if i~=size(t,2)
98        head=HWs(i,:)’-TWs(:,i);
99        %Compute turbine flow based on head, with catches at bounds of turbine
             ↪ discharge curve
100       unit_discharges=interp1(turbine_discharge(:,1), ...
101          turbine_discharge(:,2),head);
102       unit_discharges(head>=turbine_discharge(end,1))=...
103          turbine_discharge(end,2);
104       unit_discharges(head<=turbine_discharge(1,1))=...
105          turbine_discharge(1,2);
106       turb_discharges(:,i+1)=unit_discharges.*x(:,i);
107       clearvars head unit_discharges
108    end
109 end
110 clearvars i ii
111
112 %If testing new operations (i.e. ELWS_targets is not empty), continue on and
       ↪ compute spill
113 if ~isempty(ELWS_targets)
114    if Optimize_day_by_day==1 %optimize each day in series
115       %Check for cases when the final HW elev is greater than target
116       if size(ELWS_targets(:,1),1)==1
117          ELWS_goal=ELWS_targets(:,2);
118       else
119          ELWS_goal=interp1(ELWS_targets(:,1),ELWS_targets(:,2),t(end));
120       end
121       volume_to_spill=max(0,...
122          interp1(Q.SE_meters_m3(:,1),Q.SE_meters_m3(:,2),HWs(end,:))...
123          -interp1(Q.SE_meters_m3(:,1),Q.SE_meters_m3(:,2),ELWS_goal));
124       spill_discharges=0.95*((volume_to_spill/((t(end)-t(1))*24*60*60)))';
125
126       %Compute HWs again for situations with spill added to lower to ELWS
127       %target
```

```
128         [a,~]=find(spill_discharges~=0);
129         if ~isempty(a)
130             stop=0;
131             while stop==0
132                 for i=2:size(t,2)
133                     elevation=HWs(i-1,a);
134                     turbs=turb_discharges(a,i-1:i);
135                     if isempty(mainstem_inflows_Q)
136                         HWs(i-1:i,a)=Elevation_massbalance_vectorized(turbs,...
137                             [spill_discharges(a) spill_discharges(a)],...
138                             t(i-1),t(i),frequency,Q,elevation,mainstem_inflows_t,...
139                             mainstem_inflows_Q);
140                     else
141                         HWs(i-1:i,a)=Elevation_massbalance_vectorized(turbs,...
142                             [spill_discharges(a) spill_discharges(a)],...
143                             t(i-1),t(i),frequency,Q,elevation,mainstem_inflows_t,...
144                             mainstem_inflows_Q(a,:));
145                     end
146                     clearvars elevation turbs
147                     %Compute storage and TWs
148                     Storage(a,i)=interp1(Q.SE_meters_m3(:,1),Q.SE_meters_m3(:,2),...
149                         HWs(i,a)');
150                     tot_discharge=turb_discharges(a,i)+spill_discharges(a); %now
                            ↪ assume we have the spill we calculated above
151                     TWs(a,i)=interp1(Q.tw_curve_cms_m(:,1),Q.tw_curve_cms_m(:,2), ...
152                         tot_discharge);
153                     clearvars tot_discharge
154                     %Compute total turbine flowrate
155                     if i~=size(t,2)
156                         head=HWs(i,a)'-TWs(a,i);
157                         %Compute turbine flow based on head, with catches at bounds of
                                ↪  turbine discharge curve
158                         unit_discharges=interp1(turbine_discharge(:,1), ...
159                             turbine_discharge(:,2),head);
160                         unit_discharges(head>=turbine_discharge(end,1))=...
161                             turbine_discharge(end,2);
162                         unit_discharges(head<=turbine_discharge(1,1))=...
163                             turbine_discharge(1,2);
164                         turb_discharges(a,i+1)=unit_discharges.*x(a,i);
165                         clearvars head unit_discharges
166                     end
167                 end
168                 %Check end elevations again and adjust spill and iterate (if
                        ↪ necessary)
169                 volume_to_spill=interp1(Q.SE_meters_m3(:,1),Q.SE_meters_m3(:,2),HWs(
                        ↪ end,:))...
170                     -interp1(Q.SE_meters_m3(:,1),Q.SE_meters_m3(:,2),ELWS_goal);
171                 volume_to_spill(setdiff([1:size(volume_to_spill,2)],a))=0;
172                 spill_discharges2=spill_discharges+0.95*((volume_to_spill/((t(end)-t
                        ↪ (1))*24*60*60)))';
173                 diffspill=spill_discharges2-spill_discharges;
174                 if all(round(diffspill,3)==0)
175                     stop=1;
176                 end
177                 spill_discharges=spill_discharges2; clearvars spill_discharges2
178             end
179             clearvars i ii stop diffspill
180             %Recompute HWs and TWs with final spillrate
181             for i=2:size(t,2)
182                 elevation=HWs(i-1,a);
183                 turbs=turb_discharges(a,i-1:i);
184                 if isempty(mainstem_inflows_Q)
185                     HWs(i-1:i,a)=Elevation_massbalance_vectorized(turbs,...
186                         [spill_discharges(a) spill_discharges(a)],...
```

```
187                     t(i-1),t(i),frequency,Q,elevation,mainstem_inflows_t,...
188                     mainstem_inflows_Q);
189                 else
190                     HWs(i-1:i,a)=Elevation_massbalance_vectorized(turbs,...
191                         [spill_discharges(a) spill_discharges(a)],...
192                         t(i-1),t(i),frequency,Q,elevation,mainstem_inflows_t,...
193                         mainstem_inflows_Q(a,:));
194                 end
195                 clearvars elevation turbs
196                 %Compute storage and TWs
197                 %If too full and overtops SE curve (or drains and empties), linearly
                        ↪ extrapolate
198                 Storage(a,i)=interp1(Q.SE_meters_m3(:,1),Q.SE_meters_m3(:,2),...
199                     HWs(i,a)',’linear’,’extrap’);
200                 tot_discharge=turb_discharges(a,i)+spill_discharges(a); %now assume
                        ↪ we have the spill we calculated above
201                 TWs(a,i)=interp1(Q.tw_curve_cms_m(:,1),Q.tw_curve_cms_m(:,2), ...
202                     tot_discharge);
203                 clearvars tot_discharge
204                 %Compute total turbine flowrate
205                 if i~=size(t,2)
206                     head=HWs(i,a)'-TWs(a,i);
207                     %Compute turbine flow based on head, with catches at bounds of
                            ↪ turbine discharge curve
208                     unit_discharges=interp1(turbine_discharge(:,1), ...
209                         turbine_discharge(:,2),head);
210                     unit_discharges(head>=turbine_discharge(end,1))=...
211                         turbine_discharge(end,2);
212                     unit_discharges(head<=turbine_discharge(1,1))=...
213                         turbine_discharge(1,2);
214                     turb_discharges(a,i+1)=unit_discharges.*x(a,i);
215                     clearvars head unit_discharges
216                 end
217             end
218             clearvars i ii
219         end
220     else %optimize all days in 1 optimizer
221         for target=1:size(ELWS_targets,1) %loop through each target
222             if target==1 JDAY_initial=t(1); else JDAY_initial=ELWS_targets(target
                        ↪ -1,1); end
223             ELWS_goal_time=ELWS_targets(target,1);
224             ELWS_goal=ELWS_targets(target,2);
225             for i=1:size(HWs,2)
226                 inital_HWs_computed(i)=interp1(t,HWs(:,i),ELWS_goal_time);
227             end
228             clearvars i
229             volume_to_spill=max(0,...
230                 interp1(Q.SE_meters_m3(:,1),Q.SE_meters_m3(:,2),inital_HWs_computed)
                        ↪ ...
231                 -interp1(Q.SE_meters_m3(:,1),Q.SE_meters_m3(:,2),ELWS_goal));
232             spill_discharges(:,target)=0.95*((volume_to_spill/((ELWS_goal_time-
                        ↪ JDAY_initial)*24*60*60)))';
233             clearvars initial_HWs_computed
234
235             %Compute HWs again for situations with spill added to lower to ELWS
                        ↪ target
236             [a,~]=find(spill_discharges(:,target)~=0);
237             if ~isempty(a)
238                 stop=0;
239                 while stop==0
240                     for i=(1/frequency)*(target-1)+2:(1/frequency)*(target)+1
241                         elevation=HWs(i-1,a);
242                         turbs=turb_discharges(a,i-1:i);
243                         if isempty(mainstem_inflows_Q)
```

```
                    HWs(i-1:i,a)=Elevation_massbalance_vectorized(turbs,...
                        [spill_discharges(a,target) spill_discharges(a,target)
                            ↪ ],...
                        t(i-1),t(i),frequency,Q,elevation,mainstem_inflows_t,...
                        mainstem_inflows_Q);
                else
                    HWs(i-1:i,a)=Elevation_massbalance_vectorized(turbs,...
                        [spill_discharges(a,target) spill_discharges(a,target)
                            ↪ ],...
                        t(i-1),t(i),frequency,Q,elevation,mainstem_inflows_t,...
                        mainstem_inflows_Q(a,:));
                end
                clearvars elevation turbs
                %Compute storage and TWs
                Storage(a,i)=interp1(Q.SE_meters_m3(:,1),Q.SE_meters_m3(:,2)
                    ↪ ,...
                    HWs(i,a)');
                tot_discharge=turb_discharges(a,i)+spill_discharges(a,target);
                    ↪  %now assume we have the spill we calculated above
                TWs(a,i)=interp1(Q.tw_curve_cms_m(:,1),Q.tw_curve_cms_m(:,2),
                    ↪ ...
                    tot_discharge);
                clearvars tot_discharge
                %Compute total turbine flowrate
                if i~=(1/frequency)*(target)+1
                    head=HWs(i,a)'-TWs(a,i);
                    %Compute turbine flow based on head, with catches at bounds
                        ↪  of turbine discharge curve
                    unit_discharges=interp1(turbine_discharge(:,1), ...
                        turbine_discharge(:,2),head);
                    unit_discharges(head>=turbine_discharge(end,1))=...
                        turbine_discharge(end,2);
                    unit_discharges(head<=turbine_discharge(1,1))=...
                        turbine_discharge(1,2);
                    turb_discharges(a,i+1)=unit_discharges.*x(a,i);
                    clearvars head unit_discharges
                end
            end
            %Check end elevations again and adjust spill and iterate (if
                ↪ necessary)
            for i=1:size(HWs,2)
                HWs_computed_again(i)=interp1(t,HWs(:,i),ELWS_goal_time);
            end
            clearvars i
            volume_to_spill=interp1(Q.SE_meters_m3(:,1),Q.SE_meters_m3(:,2),
                ↪ HWs_computed_again)...
                -interp1(Q.SE_meters_m3(:,1),Q.SE_meters_m3(:,2),ELWS_goal);
            volume_to_spill(setdiff([1:size(volume_to_spill,2)],a))=0;
            spill_discharges2=spill_discharges(:,target)+0.95*((
                ↪ volume_to_spill/((ELWS_goal_time-JDAY_initial)*24*60*60)))
                ↪ ';
            diffspill=spill_discharges2-spill_discharges(:,target);
            if all(round(diffspill,3)==0)
                stop=1;
            end
             %if overshoot and spills go negative, set to 0.5*previous spill
                ↪ guess
            spill_discharges2(spill_discharges2<0)=0.5*spill_discharges(
                ↪ spill_discharges2<0,target);
            spill_discharges(:,target)=spill_discharges2; clearvars
                ↪ spill_discharges2 HWs_computed_again
        end
        clearvars i ii stop diffspill volume_to_spill
        %Recompute the target day HWs and TWs with final spillrate
```

```
295             for i=(1/frequency)*(target-1)+2:(1/frequency)*(target)+1
296                 elevation=HWs(i-1,a);
297                 turbs=turb_discharges(a,i-1:i);
298                 if isempty(mainstem_inflows_Q)
299                     HWs(i-1:i,a)=Elevation_massbalance_vectorized(turbs,...
300                         [spill_discharges(a,target) spill_discharges(a,target)],...
301                         t(i-1),t(i),frequency,Q,elevation,mainstem_inflows_t,...
302                         mainstem_inflows_Q);
303                 else
304                     HWs(i-1:i,a)=Elevation_massbalance_vectorized(turbs,...
305                         [spill_discharges(a,target) spill_discharges(a,target)],...
306                         t(i-1),t(i),frequency,Q,elevation,mainstem_inflows_t,...
307                         mainstem_inflows_Q(a,:));
308                 end
309                 clearvars elevation turbs
310                 %Compute storage and TWs
311                 %If too full and overtops SE curve (or drains and empties),
                        ↪ linearly extrapolate
312                 Storage(a,i)=interp1(Q.SE_meters_m3(:,1),Q.SE_meters_m3(:,2),...
313                     HWs(i,a)',’linear’,’extrap’);
314                 tot_discharge=turb_discharges(a,i)+spill_discharges(a,target); %
                        ↪ now assume we have the spill we calculated above
315                 TWs(a,i)=interp1(Q.tw_curve_cms_m(:,1),Q.tw_curve_cms_m(:,2), ...
316                     tot_discharge);
317                 clearvars tot_discharge
318                 %Compute total turbine flowrate
319                 if i~=size(t,2)
320                     head=HWs(i,a)'-TWs(a,i);
321                     %Compute turbine flow based on head, with catches at bounds of
                            ↪ turbine discharge curve
322                     unit_discharges=interp1(turbine_discharge(:,1), ...
323                         turbine_discharge(:,2),head);
324                     unit_discharges(head>=turbine_discharge(end,1))=...
325                         turbine_discharge(end,2);
326                     unit_discharges(head<=turbine_discharge(1,1))=...
327                         turbine_discharge(1,2);
328                     turb_discharges(a,i+1)=unit_discharges.*x(a,i);
329                     clearvars head unit_discharges
330                 end
331             end
332             clearvars i ii
333             %Now update HW elevs for the subsequent days, starting with the new
                    ↪ final HW elev of the target day
334             for i=(1/frequency)*(target)+1:size(t,2)
335                 elevation=HWs(i-1,a);
336                 turbs=turb_discharges(a,i-1:i);
337                 %assuming no spill flow here
338                 HWs(i-1:i,a)=Elevation_massbalance_vectorized(turbs,...
339                     zeros(size(turbs)),t(i-1),t(i),frequency,Q,elevation,...
340                     mainstem_inflows_t,mainstem_inflows_Q);
341                 clearvars elevation turbs
342                 %Compute storage and TWs
343                 %If too full and overtops SE curve (or drains and empties),
                        ↪ linearly extrapolate
344                 Storage(a,i)=interp1(Q.SE_meters_m3(:,1),Q.SE_meters_m3(:,2),...
345                     HWs(i,a)',’linear’,’extrap’);
346                 tot_discharge=turb_discharges(a,i)+0; %assume no spill
347                 TWs(a,i)=interp1(Q.tw_curve_cms_m(:,1),Q.tw_curve_cms_m(:,2), ...
348                     tot_discharge,’linear’,’extrap’);
349                 clearvars tot_discharge
350                 %Compute total turbine flowrate
351                 if i~=size(t,2)
352                     head=HWs(i,a)'-TWs(a,i);
353                     %Compute turbine flow based on head, with catches at bounds of
```

```
                    ↪ turbine discharge curve
354                 unit_discharges=interp1(turbine_discharge(:,1), ...
355                    turbine_discharge(:,2),head);
356                 unit_discharges(head>=turbine_discharge(end,1))=...
357                    turbine_discharge(end,2);
358                 unit_discharges(head<=turbine_discharge(1,1))=...
359                    turbine_discharge(1,2);
360                 turb_discharges(a,i+1)=unit_discharges.*x(a,i);
361                 clearvars head unit_discharges
362              end
363           end
364        end
365     end
366     clearvars target
367  end
368 else
369    spill_discharges=zeros(n,1);
370 end
371
372
373 HWs=HWs';%change back to rows to match all the other outputs (computed as
374        %cols to make vectorizing Elevation_massbalance_vectorized easier)
375
376 end
```

check_feasibilities.m

```
1  function [WQ_adjusted,ELWS_limit_adjusted,funccount,feasible_options,
       ↪ feasibility_check]=check_feasibilities(ranking,...
2     feasible_option1,x1_options,ga_pop_size,frequency,Q,ic_elev,no_of_units,t,
          ↪ max_hrly_unit_change,...
3     zero_gen_limit,turbine_discharge,ELWS_limit,WQ,xprev,ELWS_targets,...
4     elev_constraint_rounding,wq_constraint_rounding,tolerance,cache,
          ↪ Optimize_day_by_day,...
5        transition_matrix,Feasibilitygenerations)
6
7  % Checks the feasibility of constraints (elev, do, temp) in the priority
8  % order defined by the user, and adjusting constraints as necessary
9  %
10 % Inputs:
11 % ranking - assign priority ranking for constraints on elev, DO, and temp,
       ↪ starting
12 % with highest priority first
13 % x1_options - options for the turbine setting at the first hour
14 % ga_pop_size - population size
15 % frequency - frequency of predictions (hourly=1/24)
16 % Q - all other inflows and outflows, interpolation settings,
17 % storage-elev curve, and tailwater curve
18 % ic_elev - initial condition (meters)
19 % no_of_units - max number of turbines (4 for OHL)
20 % t  time series of JDAY values
21 % max_hrly_unit_change - max number of units that can be changed per hour
22 % (1 for OHL)
23 % zero_gen_limit - Zero generation hourly limit (can't go longer than
24 % this with no turb flow)
25 % turbine_discharge - turbine discharge curve at fixed MW level, with
26 % col 1 in meters and col 2 in cms
27 % ELWS_limit - min and max elevation limits for constraints, in meters
28 % WQ - structure containing water quality constraints and NARX models
29 % DO_narx - structure containing everything needed to make DO discharge
30 % predictions, including:
31 % turb_colum - column in exogenous variables with turb flows
32 % spill_column - column in exogenous variables with spill flows
```

```
% times - JDAY values used in training (not used)
% inputDelays - delays for exogenous inputs
% feedbackDelays - delays for prediction feedbacks
% input_variables - 2 row cell containing variable names in first
% row and column number in second. For example, 'MET_WB1'
% contains multiple columns of data but only some may be used
% for NARX predictions
% bias - bias for each trained neural network
% weights - weights for each trained neural network (sum to 1)
% narx_net_closed - neural networks
% DO_limit - lower and upper DO limits (NaN means it doesn't exist)
% DO_slack - relaxation from DO_limit (either upper or lower -
% doesn't make sense to have both)
% Temp_narx - structure containing everything needed to make temp discharge
% predictions, including:
% turb_colum - column in exogenous variables with turb flows
% spill_column - column in exogenous variables with spill flows
% times - JDAY values used in training (not used)
% inputDelays - delays for exogenous inputs
% feedbackDelays - delays for prediction feedbacks
% input_variables - 2 row cell containing variable names in first
% row and column number in second. For example, 'MET_WB1'
% contains multiple columns of data but only some may be used
% for NARX predictions
% bias - bias for each trained neural network
% weights - weights for each trained neural network (sum to 1)
% narx_net_closed - neural networks
% Temp_limit - lower and upper temp limits (NaN means it doesn't exist)
% Temp_slack - relaxation from Temp_limit (either upper or lower -
% doesn't make sense to have both)
% xprev - vector of previous active turbine levels
% ELWS_targets - 2 column matrix with JDAY in col1 and elevation target
% in col2
% elev_constraint_rounding - rounding setting (10=tenths place,
% 100=hundredths place, etc.)
% wq_constraint_rounding - rounding setting (10=tenths place,
% 100=hundredths place, etc.)
% tolerance - penalty tolerance
% cache - water quality predictions provided by W2 simulations
% Optimize_day_by_day - 1 if optimizing daily, 0 if optimizing all together
% transition_matrix - transition probabilities for turbine ramping up and down
% Feasibilitygenerations - max generations for GA feasibility check
% Outputs:
% WQ_adjusted  updated WQ structure (same structure as WQ, with updated
% constraints if necessary)
% ELWS_limit_adjusted - updated elevation limits (if necessary)
% funccount - total number of function evaluations (both obj and penalty)
% feasible_options - save any solutions that are totally feasible to feed
% into initial population creation function next
% feasibility_check - 0 if no constraints need adjusted, 1 if no fully
% feasible solution is found and constraints are adjusted

funccount=0; generations=0;
exitflag=[]; feasibility_check=0;

%First check the cache members to see if any of them are feasible
[c,~]=penalty_fcn(cache.x,t,frequency,Q,ic_elev,...
   turbine_discharge,ELWS_limit,max_hrly_unit_change, ...
   WQ,zero_gen_limit,xprev,ELWS_targets,tolerance,cache,Optimize_day_by_day);
funccount=funccount+size(cache.x,1);
feasibles=cache.x(find(all(c<=eps,2)),:);
if ~isempty(feasibles)
   fprintf('All constraints are feasible. \n');
   WQ_adjusted=WQ; ELWS_limit_adjusted=ELWS_limit;
```

```
97     feasible_options=feasibles;
98     return
99  end
100
101
102 %% Create 500 potential solutions feasible wrt constraints #1-3
103
104 %Weights
105 for wb=1:size(x1_options,2)
106     for i=2:no_of_units{wb}+1
107         weights{wb}{i}(1)=no_of_units{wb};
108         for ii=2:i
109             weights{wb}{i}(ii)=weights{wb}{i}(ii-1)*.1;
110         end
111     end
112 end
113 clearvars i ii wb
114
115 %First, generate a few solutions quickly and test feasibility. If any are
116 %feasible, terminate this function with changes to WQ or elevation
117 %constraints
118 setsize=[max(10,size(feasible_option1,1)) 2*ga_pop_size];
119 for z=1:size(setsize,2)
120     for wb=1:size(x1_options,2)
121         raw_options{wb}{z}=nan(setsize(z),size(t,2)-1);
122         if size(x1_options{wb},2)==1 %only 1 option left
123             raw_options{wb}{z}(:,1)=x1_options{wb};
124         else
125             if z==1
126                 raw_options{wb}{z}(1,1)=x1_options{wb}(end);%scenario with max
                        ↪ turbines
127                  raw_options{wb}{z}(2,1)=x1_options{wb}(1);%scenario with max spill
128                 raw_options{wb}{z}(3:end,1)=randsample(x1_options{wb},setsize(z)-2,
                        ↪ true);
129             elseif z==2
130                 raw_options{wb}{z}(:,1)=randsample(x1_options{wb},setsize(z),true,
                        ↪ weights{wb}{size(x1_options{wb},2)});
131             end
132         end
133         for i=1:size(raw_options{wb}{z},1)
134             for j=2:size(t,2)-1
135                 %Variable consisting of xprev and turbine pattern through j-1
136                 pattern=[xprev{wb} raw_options{wb}{z}(i,1:j-1)];
137                 %First start with all available options, then eliminate infeasible
                        ↪ ones based on turbines from 1:j-1
138                 options=[0:no_of_units{wb}];
139                 % (1) Eliminate options based on change in active unit violations
140                 if ~isnan(max_hrly_unit_change{wb})
141                     auvoptions=[pattern(end)-max_hrly_unit_change{wb}: ...
142                         pattern(end)+max_hrly_unit_change{wb}];
143                     options=intersect(options,auvoptions);
144                 end
145                 % (2) Non-integer constraint (assumed in selection algorithm)
146                 % (3) Eliminate options based on zero generation hourly limit
147                 if ~isnan(zero_gen_limit{wb})
148                     if sum(pattern(end-zero_gen_limit{wb}+1:end))==0
149                         zghloptions=[1:no_of_units{wb}]; %if previous zero_gen_limit
                                ↪ hrs had zero total flow, must have flow next hr
150                         options=intersect(options,zghloptions);
151                     end
152                 end
153                 % (4) Eliminate options that violate oscillations constraint -
                        ↪ violates whenever the number of turbines increases and then
                        ↪ decreases within 3 hours, or vice versa
```

```
154             allopt=[0:no_of_units{wb}];
155             if pattern(end-1)<pattern(end) %if prev turbs increasing
156                oscoptions=allopt(allopt>=pattern(end));
157                options=intersect(options,oscoptions);
158             elseif pattern(end-1)==pattern(end) %need 3 hrs btwn ramping up and
                    ↪ down
159                if pattern(end-2)<pattern(end-1) %ramping up
160                   oscoptions=allopt(allopt>=pattern(end));
161                   options=intersect(options,oscoptions);
162                elseif pattern(end-2)>pattern(end-1) %ramping down
163                   oscoptions=allopt(allopt<=pattern(end));
164                   options=intersect(options,oscoptions);
165                elseif pattern(end-2)==pattern(end-1)
166                   %do nothing -->3 consecutive hours between ramping up and down
                          ↪  satisfied
167                end
168             elseif pattern(end-1)>pattern(end) %if prev turbs decreasing
169                oscoptions=allopt(allopt<=pattern(end));
170                options=intersect(options,oscoptions);
171             end
172             %Out of the available options left, pick the next turbine setting
173             if size(options,2)==1 %only 1 option left
174                raw_options{wb}{z}(i,j)=options;
175             else
176                if z==1
177                   if i==1 %scenario with max turbines
178                      raw_options{wb}{z}(i,j)=options(end);
179                   elseif i==2 %scenario with max spill
180                      raw_options{wb}{z}(i,j)=options(1);
181                   elseif i==3 %scenario with fairly level turbines (minimal
                          ↪ change)
182                      if mod(size(options,2),2)==0 %is even
183                         raw_options{wb}{z}(i,j)=options(round((size(options,2)
                                ↪ /2)+.5+randsample([0.1 -0.1],1)));
184                      else
185                         raw_options{wb}{z}(i,j)=options(round((size(options,2)
                                ↪ /2)+randsample([0.1 -0.1],1)));
186                      end
187                   else
188                      raw_options{wb}{z}(i,j)=randsample(options,1,true);
189                   end
190                elseif z==2
191                   raw_options{wb}{z}(i,j)=randsample(options,1,true,weights{wb}{
                          ↪ size(options,2)});
192                end
193             end
194          end
195       end
196    end
197
198    %Convert raw_options cells to long vectors containing all reservoirs
199    %per row
200    raw_options2{z}=[];
201    for wb=1:size(x1_options,2)
202       raw_options2{z}=[raw_options2{z} raw_options{wb}{z}];
203    end
204    if z==1
205       raw_options2{z}=[raw_options2{z}; feasible_option1];
206    end
207    [raw_options2{z},~,~]=unique(raw_options2{z},'rows');
208
209    %Check feasibilities of first small set
210    if z==1
211       [c,~]=penalty_fcn(raw_options2{z},t,frequency,Q,ic_elev,...
```

```
212          turbine_discharge,ELWS_limit,max_hrly_unit_change, ...
213          WQ,zero_gen_limit,xprev,ELWS_targets,tolerance,cache,
                 ↪ Optimize_day_by_day);
214       funccount=funccount+size(raw_options2{z},1);
215       feasibles=raw_options2{z}(find(all(c<=eps,2)),:);
216       if ~isempty(feasibles)
217          fprintf('All constraints are feasible. \n');
218          WQ_adjusted=WQ; ELWS_limit_adjusted=ELWS_limit;
219          feasible_options=feasibles;
220          return
221       end
222    end
223 end
224 feasible_options2=[];
225 for z=1:size(setsize,2)
226    feasible_options2=[feasible_options2; raw_options2{z}];
227 end
228 [feasible_options2,~,~]=unique(feasible_options2,'rows');
229 feasible_options=feasible_options2; feasible_options_raw=feasible_options;
230 clearvars z i a j feasibles feasible_options2
231
232 %% Optimize each constraint in priority order and terminate at 0. Otherwise,
        ↪ modify the constraint bounds
233
234 for wb=1:size(x1_options,2)
235    ELWS_limit_adjusted{wb}=nan(size(ELWS_limit{wb}));
236    WQ_adjusted{wb}.DO_limit=nan(size(WQ{wb}.DO_limit));
237    WQ_adjusted{wb}.Temp_limit=nan(size(WQ{wb}.Temp_limit));
238    WQ_adjusted{wb}.DO_narx=WQ{wb}.DO_narx;
239    WQ_adjusted{wb}.Temp_narx=WQ{wb}.Temp_narx;
240    WQ_adjusted{wb}.DO_slack=WQ{wb}.DO_slack;
241    WQ_adjusted{wb}.Temp_slack=WQ{wb}.Temp_slack;
242 end
243 skip=0;
244
245 for wb=1:size(x1_options,2)
246    for i=1:size(ranking,2)
247       if strcmp(ranking{i},'elev') & (~isnan(ELWS_limit{wb}(1)) | ~isnan(
              ↪ ELWS_limit{wb}(2)))
248          fprintf(['Checking reservoir #', num2str(wb),' elevation constraint
                 ↪ feasibility. \n']);
249       elseif strcmp(ranking{i},'do') & (~isnan(WQ{wb}.DO_limit(1)) | ~isnan(WQ{
              ↪ wb}.DO_limit(2)))
250          fprintf(['Checking reservoir #', num2str(wb),' DO constraint
                 ↪ feasibility. \n']);
251       elseif strcmp(ranking{i},'temp') & (~isnan(WQ{wb}.Temp_limit(1)) | ~isnan(
              ↪ WQ{wb}.Temp_limit(2)))
252          fprintf(['Checking reservoir #', num2str(wb),' temperature constraint
                 ↪ feasibility. \n']);
253       end
254
255       %Check lower limit then upper limit. In each step, check maximum violation
              ↪  and then mean value (for temp & DO, not elevation)
256       for a=1:2
257          if a==1 level='lower'; elseif a==2 level='upper'; end
258
259          if strcmp(ranking{i},'elev') & ~isnan(ELWS_limit{wb}(a))
260             skip=0;
261          elseif strcmp(ranking{i},'do') & ~isnan(WQ{wb}.DO_limit(a))
262             skip=0;
263          elseif strcmp(ranking{i},'temp') & ~isnan(WQ{wb}.Temp_limit(a))
264             skip=0;
265          else
266             skip=1; %if there is no constraint being added here, no need to
```

```
            ↪ check feasibility!
        end

        if skip==0
            clearvars FitnessFunction mycon opt

            %(1) Test the maximum constraint violation first

            %Set penalty function first to make sure it doesn't include the
                ↪ constraint that is being optimized, but all constraints
                ↪ before that one
            mycon= @(x) penalty_fcn(x,t,frequency,Q,ic_elev,...
                turbine_discharge,ELWS_limit_adjusted,max_hrly_unit_change,...
                WQ_adjusted,zero_gen_limit,xprev,ELWS_targets,tolerance,cache,
                    ↪ Optimize_day_by_day);
            %Load in the relevant constraints
            if strcmp(ranking{i},'elev')
                ELWS_limit_adjusted{wb}(a)=ELWS_limit{wb}(a);
            elseif strcmp(ranking{i},'do')
                WQ_adjusted{wb}.DO_limit(a)=WQ{wb}.DO_limit(a);
                WQ_adjusted{wb}.DO_slack=WQ{wb}.DO_slack;
            elseif strcmp(ranking{i},'temp')
                WQ_adjusted{wb}.Temp_limit(a)=WQ{wb}.Temp_limit(a);
                WQ_adjusted{wb}.Temp_slack=WQ{wb}.Temp_slack;
            end
            %Set objective function
            if strcmp(ranking{i},'elev') & ~isnan(ELWS_limit_adjusted{wb}(a))
                FitnessFunction = @(x) obj_fcn_elev(x,t,frequency,Q,ic_elev,...
                    turbine_discharge,ELWS_limit_adjusted{wb},xprev,ELWS_targets,
                        ↪ level,wb,cache,Optimize_day_by_day);
            elseif strcmp(ranking{i},'do') & ~isnan(WQ_adjusted{wb}.DO_limit(a))
                FitnessFunction = @(x) obj_fcn_do(x,t,frequency,Q,ic_elev,...
                    turbine_discharge,WQ_adjusted,xprev,ELWS_targets,level,wb,
                        ↪ cache,Optimize_day_by_day);
            elseif strcmp(ranking{i},'temp') & ~isnan(WQ_adjusted{wb}.Temp_limit
                ↪ (a))
                FitnessFunction = @(x) obj_fcn_temp(x,t,frequency,Q,ic_elev,...
                    turbine_discharge,WQ_adjusted,xprev,ELWS_targets,level,wb,
                        ↪ cache,Optimize_day_by_day);
            end
            %Check feasibility
            if any(FitnessFunction(feasible_options(1:min(size(feasible_options
                ↪ ,1),setsize(1)),:))==0)
                fval=0; funccount=funccount+size(feasible_options,1);
                pop=feasible_options;
            else
                %If feasible_options<GA pop size, fill in a larger matrix with
                    ↪ repeating values to create a full initial population
                if size(feasible_options,1)<ga_pop_size
                    feasible_options=repmat(feasible_options,ceil(ga_pop_size/size
                        ↪ (feasible_options,1)),1);
                    feasible_options=feasible_options(1:ga_pop_size,:);
                end
                %GA settings
                opt = gaoptimset(...
                    'Display','iter','Vectorized','on','Generations',
                        ↪ Feasibilitygenerations, ...
                    'PopulationSize',ga_pop_size,...
                    'InitialPopulation',feasible_options(1:ga_pop_size,:),...
                    'StallGenLimit',1,'TolFun',tolerance,'TolCon',tolerance,...
                    'CrossoverFcn',@crossoversinglepoint,'CrossoverFraction'
                        ↪ ,0.85,...
                    'EliteCount',ceil(.05*ga_pop_size),...
                    'CreationFcn',@int_pop,'MutationFcn',@int_mutation,'
```

```
                    ↪ FitnessLimit',0);
318             nVar = size(x1_options,2)*(size(t,2)-1);
319             %Set dv lower and upper bounds, narrowed considering
                    ↪ max_hrly_unit_change
320             clearvars lb ub
321             for wb2=1:size(x1_options,2)
322                 lb(wb2,:)=0*ones(1,size(t,2)-1); lb(wb2,1)=x1_options{wb2}(1);
323                 for ii=2:no_of_units{wb2}
324                     lb(wb2,ii)=lb(wb2,ii-1)-max_hrly_unit_change{wb2};
325                 end
326                 lb(wb2,:)=max(0,lb(wb2,:));
327                 ub(wb2,:)=no_of_units{wb2}*ones(1,size(t,2)-1);
328                 ub(wb2,1)=x1_options{wb2}(end);
329                 for ii=2:no_of_units{wb2}
330                     ub(wb2,ii)=ub(wb2,ii-1)+max_hrly_unit_change{wb2};
331                 end
332                 ub(wb2,:)=min(no_of_units{wb2},ub(wb2,:));
333                 clearvars ii
334             end
335             clearvars wb2
336             lb=reshape(lb',1,[]); ub=reshape(ub',1,[]);
337             %Run GA
338             [~,fval,~,output,pop,~]=ga(FitnessFunction,nVar,[],[],[],[],lb,ub
                    ↪ ,...
339                 mycon,[],opt);
340             funccount=funccount+output.funccount*2; %multiply by 2 to cover
                    ↪ penalty & obj functions
341             generations=output.generations;
342         end
343         %Adjust constraint limits if necessary
344         if fval~=0
345             if level=='lower'
346                 plusminus=-1;
347             elseif level=='upper'
348                 plusminus=1;
349             end
350             if strcmp(ranking{i},'elev')
351                 fprintf(['Adjusting reservoir #', num2str(wb),' ', level, '
                        ↪ elevation constraint. \n']);
352                 ELWS_limit_adjusted{wb}(a)=ELWS_limit{wb}(a)...
353                     +plusminus*ceil(elev_constraint_rounding*fval)/
                            ↪ elev_constraint_rounding;
354                 feasibility_check=1;
355                 if ~isempty(pop)
356                     pop=[pop; feasible_options_raw]; pop=unique(pop,'rows');
357                     c=mycon(pop); pop=pop(all(c<=tolerance,2),:);
358                     o=FitnessFunction(pop);
359                     feasible_options=pop(find(o==min(o)),:);
360                 end
361             elseif strcmp(ranking{i},'do')
362                 fprintf(['Adjusting reservoir #', num2str(wb),' ', level, ' DO
                        ↪  slack constraint. \n']);
363                 WQ_adjusted{wb}.DO_slack(a)=ceil(wq_constraint_rounding*fval)/
                        ↪ wq_constraint_rounding;
364                 feasibility_check=1;
365                 if ~isempty(pop)
366                     pop=[pop; feasible_options_raw]; pop=unique(pop,'rows');
367                     c=mycon(pop); pop=pop(all(c<=tolerance,2),:);
368                     o=FitnessFunction(pop);
369                     feasible_options=pop(find(o==min(o)),:);
370                 end
371             elseif strcmp(ranking{i},'temp')
372                 fprintf(['Adjusting reservoir #', num2str(wb),' ', level, '
                        ↪ temperature slack constraint. \n']);
```

```
                    WQ_adjusted{wb}.Temp_slack(a)=ceil(wq_constraint_rounding*fval
                        ↪ )/wq_constraint_rounding;
                    feasibility_check=1;
                    if ~isempty(pop)
                       pop=[pop; feasible_options_raw]; pop=unique(pop,'rows');
                       c=mycon(pop); pop=pop(all(c<=tolerance,2),:);
                       o=FitnessFunction(pop);
                       feasible_options=pop(find(o==min(o)),:);
                    end
                 end
              else
                 pop=[pop; feasible_options_raw]; pop=unique(pop,'rows','stable');
                 c=mycon(pop); pop=pop(all(c<=tolerance,2),:);
                 o=FitnessFunction(pop);
                 feasible_options=pop(find(o==min(o)),:);
              end
              clearvars plusminus output
           end
        end
     end
end
clearvars i a
WQ_adjusted{wb}.DO_slack=sum(WQ_adjusted{wb}.DO_slack,2);
WQ_adjusted{wb}.Temp_slack=sum(WQ_adjusted{wb}.Temp_slack,2);
[feasible_options,~,~]=unique(feasible_options,'rows');
```

compute_AME_trpt.m

```
%Find index in cache where the trainingpop point is
[~,b]=ismember(x_trpt,cache.x,'rows');

%Compute DO and temp predictions
[turb_discharges,spill_discharges,~,~,~]=...
   activeunits_to_discharges(x_trpt,t,frequency,Q{wb},ic_elev{wb},...
   turbine_discharge{wb},ELWS_targets{wb},[],[],Optimize_day_by_day);
DO_pred=narx_predictions(WQ{wb}.DO_narx,frequency,t,Q{wb},...
   x_trpt,turb_discharges,spill_discharges,[],Q{wb}.CWO,...
   'do',Optimize_day_by_day);
T_pred=narx_predictions(WQ{wb}.Temp_narx,frequency,t,Q{wb},...
   x_trpt,turb_discharges,spill_discharges,[],Q{wb}.TWO,...
   'temp',Optimize_day_by_day);

clearvars turb_discharges spill_discharges x_trpt
```

create_feasible_initpop.m

```
function [feasible_options,y,c,funccount]=create_feasible_initpop(no_of_solns
    ↪ ,...
   feasible_options,x1_options,frequency,Q,ic_elev,MW_rating,no_of_units,t,...
   max_hrly_unit_change,zero_gen_limit,turbine_discharge,ELWS_limit,...
   WQ,cost_curve_MW,xprev,elev_soft_penalty_coeff,...
   ELWS_targets,tolerance,cache,Optimize_day_by_day,transition_matrix,...
   initial_NARX_training_pop)

% Generate and save lots of solutions that are feasible in terms of:
% (1) Change in active unit violations
% (2) Non-integer constraint (assumed in this seletion algorithm)
% (3) Zero generation hourly limit
% (4) Oscillations constraint
% If can't find enough feasible solutions, the rest of the population is
% filled in with near-feasible soultions
%
```

```
% Inputs:
% no_of_solns - the number of feasible solutions we want to find
% feasible_options - feasible solutions already found during constraint
% prescreening
% x1_options - feasible options for first value of x, between 0 and
% no_of_units
% frequency - frequency of predictions (hourly=1/24)
% Q - all other inflows and outflows, interpolation settings,
% storage-elev curve, and tailwater curve (all in meters)
% ic_elev - initial elevation condition (m)
% MW_rating - the fixed MW level of turbine_discharge_curve (25 MW for
% OHL)
% no_of_units - max number of available turbine units
% t  time series of JDAY values
% max_hrly_unit_change - max number of units that can be changed per hour
% (1 for OHL)
% zero_gen_limit - Zero generation hourly limit (can't go longer than
% this with no turb flow)
% turbine_discharge - turbine discharge curve at fixed MW level, with
% col 1 in meters and col 2 in cms
% ELWS_limit - min and max elevation limits for constraints, in meters
% WQ - structure containing water quality constraints and NARX models
% DO_narx - structure containing everything needed to make DO discharge
% predictions, including:
% turb_colum - column in exogenous variables with turb flows
% spill_column - column in exogenous variables with spill flows
% times - JDAY values used in training (not used)
% inputDelays - delays for exogenous inputs
% feedbackDelays - delays for prediction feedbacks
% input_variables - 2 row cell containing variable names in first
% row and column number in second. For example, 'MET_WB1'
% contains multiple columns of data but only some may be used
% for NARX predictions
% bias - bias for each trained neural network
% weights - weights for each trained neural network (sum to 1)
% narx_net_closed - neural networks
% DO_limit - lower and upper DO limits (NaN means it doesn't exist)
% DO_slack - relaxation from DO_limit (either upper or lower -
% doesn't make sense to have both)
% Temp_narx - structure containing everything needed to make temp discharge
% predictions, including:
% turb_colum - column in exogenous variables with turb flows
% spill_column - column in exogenous variables with spill flows
% times - JDAY values used in training (not used)
% inputDelays - delays for exogenous inputs
% feedbackDelays - delays for prediction feedbacks
% input_variables - 2 row cell containing variable names in first
% row and column number in second. For example, 'MET_WB1'
% contains multiple columns of data but only some may be used
% for NARX predictions
% bias - bias for each trained neural network
% weights - weights for each trained neural network (sum to 1)
% narx_net_closed - neural networks
% Temp_limit - lower and upper temp limits (NaN means it doesn't exist)
% Temp_slack - relaxation from Temp_limit (either upper or lower -
% doesn't make sense to have both)
% cost_curve_MW  2 row matrix to create step function, with 1st row
% being hours and 2nd row $/MW-hr values
% xprev - vector of previous active turbine levels
% elev_soft_penalty_coeff - penalty coefficient for soft ending elev soft
% constraint
% ELWS_targets - target elevations for end of time period
% tolerance - penalty tolerance
% cache - water quality predictions provided by W2 simulations
```

```
% Optimize_day_by_day - 1 if optimizing daily, 0 if optimizing all together
% transition_matrix - transition probabilities for turbine ramping up and down
% initial_NARX_training_pop - 1 if creating initial population for NARX training
% Outputs:
% feasible_options  feasible potential solutions for GA initialization
% y - objective function solutions for feasible_options
% c - constraint violations
% funccount - number of paired function evaluations

%Start with upstream reservoir (wb=1), find feasible operations, and compute associated discharge flows for each. Then use those flows as upstream inflow for next wb, find feasible operations, and compute associated discharge flows. Etc...

c=[]; infeasibles.x=[]; infeasibles.c=[];
n=size(feasible_options,1);
funccount=0;

count=1;
while size(feasible_options,1)<no_of_solns

   if count==1
      %Starting set size
      setsize=no_of_solns;
   elseif count==2
      %Modify set size as a function of how many feasible solns found so far (maximum is 30*setsize)
      setsize=min(5*(setsize),round((setsize/(size(feasible_options,1)-n))*...
         (no_of_solns-(size(feasible_options,1)-n))));
   else
      %If still not enough solns found, should be close so try 50 at a time
      setsize=50;
   end

   for wb=1:size(x1_options,2)
      raw_options{wb}=nan(setsize,size(t,2)-1);
      if size(x1_options{wb},2)==1 %only 1 option left
         raw_options{wb}(:,1)=x1_options{wb};
      else
         if ~isempty(initial_NARX_training_pop) %if it's the initial sample, make sure to include a min and a max outflow
            raw_options{wb}(1,1)=x1_options{wb}(end);%scenario with max turbines
            raw_options{wb}(2,1)=x1_options{wb}(1);%scenario with max spill
            raw_options{wb}(3:end,1)=randsample(x1_options{wb},setsize-2,true,...
               transition_matrix{wb}(xprev{wb}(end)+1,x1_options{wb}+1));
         else
            raw_options{wb}(:,1)=randsample(x1_options{wb},setsize,true,...
               transition_matrix{wb}(xprev{wb}(end)+1,x1_options{wb}+1));
         end
      end
      for i=1:setsize
         for j=2:size(t,2)-1
            %Variable consisting of xprev and turbine pattern through j-1
            pattern=[xprev{wb} raw_options{wb}(i,1:j-1)];
            %First start with all available options, then eliminate infeasible ones based on turbines from 1:j-1
            options=[0:no_of_units{wb}];
            % (1) Eliminate options based on change in active unit violations
            if ~isnan(max_hrly_unit_change{wb})
               auvoptions=[pattern(end)-max_hrly_unit_change{wb}: ...
                  pattern(end)+max_hrly_unit_change{wb}];
               options=intersect(options,auvoptions);
            end
```

```
137             % (2) Non-integer constraint (assumed in selection algorithm)
138             % (3) Eliminate options based on zero generation hourly limit
139             if ~isnan(zero_gen_limit{wb})
140                 if sum(pattern(end-zero_gen_limit{wb}+1:end))==0
141                     zghloptions=[1:no_of_units{wb}]; %if previous zero_gen_limit
                        ↪ hrs had zero total flow, must have flow next hr
142                     options=intersect(options,zghloptions);
143                 end
144             end
145             % (4) Eliminate options that violate oscillations constraint -
                    ↪ violates whenever the number of turbines increases and then
                    ↪ decreases within 3 hours, or vice versa
146             allopt=[0:no_of_units{wb}];
147             if pattern(end-1)<pattern(end) %if prev turbs increasing
148                 oscoptions=allopt(allopt>=pattern(end));
149                 options=intersect(options,oscoptions);
150             elseif pattern(end-1)==pattern(end) %need 3 hrs btwn ramping up and
                    ↪ down
151                 if pattern(end-2)<pattern(end-1) %ramping up
152                     oscoptions=allopt(allopt>=pattern(end));
153                     options=intersect(options,oscoptions);
154                 elseif pattern(end-2)>pattern(end-1) %ramping down
155                     oscoptions=allopt(allopt<=pattern(end));
156                     options=intersect(options,oscoptions);
157                 elseif pattern(end-2)==pattern(end-1)
158                     %do nothing -->3 consecutive hours between ramping up and down
                        ↪  satisfied
159                 end
160             elseif pattern(end-1)>pattern(end) %if prev turbs decreasing
161                 oscoptions=allopt(allopt<=pattern(end));
162                 options=intersect(options,oscoptions);
163             end
164             %Out of the available options left, pick the next turbine setting
165             if size(options,2)==1 %only 1 option left
166                 raw_options{wb}(i,j)=options;
167             else
168                 if ~isempty(initial_NARX_training_pop) %if it's the initial
                        ↪ sample, make sure to include a min and a max outflow
169                     if i==1 %scenario with max turbines
170                         raw_options{wb}(i,j)=options(end);
171                     elseif i==2 %scenario with max spill
172                         raw_options{wb}(i,j)=options(1);
173                     else
174                         raw_options{wb}(i,j)=randsample(options,1,true,...
175                             transition_matrix{wb}(raw_options{wb}(i,j-1)+1,options
                                ↪ +1));
176                     end
177                 else
178                     raw_options{wb}(i,j)=randsample(options,1,true,...
179                         transition_matrix{wb}(raw_options{wb}(i,j-1)+1,options+1));
180                 end
181             end
182         end
183     end
184 end
185
186 %Convert raw_options cells to long vectors containing all reservoirs per row
187 raw_options2=[];
188 for wb=1:size(x1_options,2)
189     raw_options2=[raw_options2 raw_options{wb}];
190 end
191
192 %Check feasibility
193 [c_new,~]=penalty_fcn(raw_options2,t,frequency,Q,ic_elev,...
```

```
194         turbine_discharge,ELWS_limit,max_hrly_unit_change,WQ,...
195         zero_gen_limit,xprev,ELWS_targets,tolerance,cache,Optimize_day_by_day);
196     funccount=funccount+size(raw_options2,1);
197     c=c_new;
198 
199     raw_options3=raw_options2(all(c_new<=tolerance,2),:);
200     feasible_options=[feasible_options; raw_options3];
201     %Save the infeasible options in case needed later
202     infeasibles.x=[infeasibles.x; raw_options2(any(c_new>tolerance,2),:)];
203     infeasibles.c=[infeasibles.c; c_new(any(c_new>tolerance,2),:)];
204     %Remove duplicates
205     feasible_options=unique(feasible_options,'rows');
206     fprintf(['Feasible options found: ',...
207         num2str(size(feasible_options,1)), '\n']);
208     if count==2 & isempty(feasible_options)
209         y=[]; return
210     elseif count==5 & ~isempty(feasible_options)
211         break
212     else
213         count=count+1;
214     end
215 end
216 
217 if isempty(initial_NARX_training_pop)
218     %Pick the best no_of_solns from feasible_options
219     y=obj_fcn(feasible_options,t,cost_curve_MW,MW_rating,...
220         elev_soft_penalty_coeff,ELWS_targets,frequency,Q,ic_elev,...
221         turbine_discharge,cache,Optimize_day_by_day);
222     funccount=funccount+size(feasible_options,1);
223     [y,b]=sort(y,'descend');
224     feasible_options=feasible_options(b,:);
225 else
226     y=[];
227     if size(feasible_options,1)>no_of_solns
228         picks=randsample(size(feasible_options,1),no_of_solns);
229         feasible_options=feasible_options(picks,:);
230     end
231 end
232 
233 % If haven't found enough feasible options, fill in the rest of the pop with
        ↪ near-feasibles ONLY WORKS FOR 1 WB PROBLEMS FOR NOW
234 if size(feasible_options,1)<no_of_solns
235     relevant_indexes=[];
236     for wb=1:size(x1_options,2)
237         if ~isnan(ELWS_limit{wb}(1))
238             relevant_indexes=[relevant_indexes 4:3+(1+(size(t,2)-1))*1];
239         end
240         if ~isnan(ELWS_limit{wb}(2))
241             relevant_indexes=[relevant_indexes 3+(1+(size(t,2)-1))+1:3+(1+(size(t
                ↪ ,2)-1))*2];
242         end
243         if ~isnan(WQ{wb}.DO_limit(1))
244             relevant_indexes=[relevant_indexes 3+(1+(size(t,2)-1))*2+1];
245         end
246         if ~isnan(WQ{wb}.DO_limit(2))
247             relevant_indexes=[relevant_indexes 3+(1+(size(t,2)-1))*2+2];
248         end
249         if ~isnan(WQ{wb}.Temp_limit(1))
250             relevant_indexes=[relevant_indexes 3+(1+(size(t,2)-1))*2+2+1];
251         end
252         if ~isnan(WQ{wb}.Temp_limit(2))
253             relevant_indexes=[relevant_indexes 3+(1+(size(t,2)-1))*2+2+2];
254         end
255     end
```

```
256     %Remove duplicates
257     [infeasibles.x,ia,~]=unique(infeasibles.x,'rows');
258     infeasibles.c=infeasibles.c(ia,:);
259     %Normalize the relevant index cols
260     normc=infeasibles.c(:,relevant_indexes); normc2=[];
261     for i=1:size(normc,2)
262        if ~all(normc(:,i)==normc(1,i)) normc2=[normc2 normc(:,i)]; end
263     end
264     mindata = min(normc2); maxdata = max(normc2);
265     normc2 = bsxfun(@rdivide, bsxfun(@minus, normc2, mindata), maxdata - mindata)
          ↪ ;
266     meanc=mean(normc2,2); [meanc,b]=sort(meanc,'ascend');
267     feasible_options=[feasible_options; ...
268        infeasibles.x(b(1:no_of_solns-size(feasible_options,1)),:)];
269  end
```

create_replacements.m

```
1   function [feasible_options]=create_replacements(no_of_solns,...
2      feasible_options,x1_options,frequency,Q,ic_elev,MW_rating,no_of_units,t,...
3      max_hrly_unit_change,zero_gen_limit,turbine_discharge,ELWS_limit,...
4      WQ,cost_curve_MW,xprev,elev_soft_penalty_coeff,...
5      ELWS_targets,tolerance,cache,Optimize_day_by_day,transition_matrix)
6
7   % Generate and save lots of solutions that are feasible in terms of:
8   % (1) Change in active unit violations
9   % (2) Non-integer constraint (assumed in this seletion algorithm)
10  % (3) Zero generation hourly limit
11  % (4) Oscillations constraint
12  % If can't find enough feasible solutions, the rest of the population is
13  % filled in with near-feasible soultions
14  %
15  % Inputs:
16  % no_of_solns - the number of feasible solutions we want to find
17  % feasible_options - feasible solutions already found during constraint
18  % prescreening
19  % x1_options - feasible options for first value of x, between 0 and
20  % no_of_units
21  % frequency - frequency of predictions (hourly=1/24)
22  % Q - all other inflows and outflows, interpolation settings,
23  % storage-elev curve, and tailwater curve (all in meters)
24  % ic_elev - initial elevation condition (m)
25  % MW_rating - the fixed MW level of turbine_discharge_curve (25 MW for
26  % OHL)
27  % no_of_units - max number of available turbine units
28  % t  time series of JDAY values
29  % max_hrly_unit_change - max number of units that can be changed per hour
30  % (1 for OHL)
31  % zero_gen_limit - Zero generation hourly limit (can't go longer than
32  % this with no turb flow)
33  % turbine_discharge - turbine discharge curve at fixed MW level, with
34  % col 1 in meters and col 2 in cms
35  % ELWS_limit - min and max elevation limits for constraints, in meters
36  % WQ - structure containing water quality constraints and NARX models
37  % DO_narx - structure containing everything needed to make DO discharge
38  % predictions, including:
39  % turb_colum - column in exogenous variables with turb flows
40  % spill_column - column in exogenous variables with spill flows
41  % times - JDAY values used in training (not used)
42  % inputDelays - delays for exogenous inputs
43  % feedbackDelays - delays for prediction feedbacks
44  % input_variables - 2 row cell containing variable names in first
45  % row and column number in second. For example, 'MET_WB1'
```

```
% contains multiple columns of data but only some may be used
% for NARX predictions
% bias - bias for each trained neural network
% weights - weights for each trained neural network (sum to 1)
% narx_net_closed - neural networks
% DO_limit - lower and upper DO limits (NaN means it doesn't exist)
% DO_slack - relaxation from DO_limit (either upper or lower -
% doesn't make sense to have both)
% Temp_narx - structure containing everything needed to make temp discharge
% predictions, including:
% turb_colum - column in exogenous variables with turb flows
% spill_column - column in exogenous variables with spill flows
% times - JDAY values used in training (not used)
% inputDelays - delays for exogenous inputs
% feedbackDelays - delays for prediction feedbacks
% input_variables - 2 row cell containing variable names in first
% row and column number in second. For example, 'MET_WB1'
% contains multiple columns of data but only some may be used
% for NARX predictions
% bias - bias for each trained neural network
% weights - weights for each trained neural network (sum to 1)
% narx_net_closed - neural networks
% Temp_limit - lower and upper temp limits (NaN means it doesn't exist)
% Temp_slack - relaxation from Temp_limit (either upper or lower -
% doesn't make sense to have both)
% cost_curve_MW  2 row matrix to create step function, with 1st row
% being hours and 2nd row $/MW-hr values
% xprev - vector of previous active turbine levels
% elev_soft_penalty_coeff - penalty coefficient for soft ending elev soft
% constraint
% ELWS_targets - target elevations for end of time period
% tolerance - penalty tolerance
% cache - water quality predictions provided by W2 simulations
% Optimize_day_by_day - 1 if optimizing daily, 0 if optimizing all together
% transition_matrix - transition probabilities for turbine ramping up and down
% Outputs:
% feasible_options  feasible potential solutions for GA initialization
% y - objective function solutions for feasible_options
% c - constraint violations
% funccount - number of paired function evaluations

%Start with upstream reservoir (wb=1), find feasible operations, and compute
    ↪ associated discharge flows for each. Then use those flows as upstream
    ↪ inflow for next wb, find feasible operations, and compute associated
    ↪ discharge flows. Etc...

c=[]; infeasibles.x=[]; infeasibles.c=[];
n=size(feasible_options,1);
funccount=0;

%Weights
for wb=1:size(x1_options,2)
   for i=2:no_of_units{wb}+1
      weights{wb}{i}(1)=no_of_units{wb};
      for ii=2:i
         weights{wb}{i}(ii)=weights{wb}{i}(ii-1)*.5;
      end
   end
end
clearvars i ii wb

count=1;
while size(feasible_options,1)<no_of_solns

```

```
107     if count==1
108        %Starting set size
109        setsize=no_of_solns;
110     elseif count==2
111        %Modify set size as a function of how many feasible solns found so far (
               ↪ maximum is 30*setsize)
112        setsize=min(5*(setsize),round((setsize/(size(feasible_options,1)-n))*...
113           (no_of_solns-(size(feasible_options,1)-n))));
114     else
115        %If still not enough solns found, should be close so try 50 at a time
116        setsize=50;
117     end
118
119     for wb=1:size(x1_options,2)
120        raw_options{wb}=nan(setsize,size(t,2)-1);
121        if size(x1_options{wb},2)==1 %only 1 option left
122           raw_options{wb}(:,1)=x1_options{wb};
123        else
124           raw_options{wb}(:,1)=randsample(x1_options{wb},setsize,true,...
125              transition_matrix{wb}(xprev{wb}(end)+1,x1_options{wb}+1));
126        end
127        for i=1:setsize
128           for j=2:size(t,2)-1
129              %Variable consisting of xprev and turbine pattern through j-1
130              pattern=[xprev{wb} raw_options{wb}(i,1:j-1)];
131              %First start with all available options, then eliminate infeasible
                     ↪ ones based on turbines from 1:j-1
132              options=[0:no_of_units{wb}];
133              % (1) Eliminate options based on change in active unit violations
134              if ~isnan(max_hrly_unit_change{wb})
135                 auvoptions=[pattern(end)-max_hrly_unit_change{wb}: ...
136                    pattern(end)+max_hrly_unit_change{wb}];
137                 options=intersect(options,auvoptions);
138              end
139              % (2) Non-integer constraint (assumed in selection algorithm)
140              % (3) Eliminate options based on zero generation hourly limit
141              if ~isnan(zero_gen_limit{wb})
142                 if sum(pattern(end-zero_gen_limit{wb}+1:end))==0
143                    zghloptions=[1:no_of_units{wb}]; %if previous zero_gen_limit
                           ↪ hrs had zero total flow, must have flow next hr
144                    options=intersect(options,zghloptions);
145                 end
146              end
147              % (4) Eliminate options that violate oscillations constraint -
                     ↪ violates whenever the number of turbines increases and then
                     ↪ decreases within 3 hours, or vice versa
148              allopt=[0:no_of_units{wb}];
149              if pattern(end-1)<pattern(end) %if prev turbs increasing
150                 oscoptions=allopt(allopt>=pattern(end));
151                 options=intersect(options,oscoptions);
152              elseif pattern(end-1)==pattern(end) %need 3 hrs btwn ramping up and
                     ↪ down
153                 if pattern(end-2)<pattern(end-1) %ramping up
154                    oscoptions=allopt(allopt>=pattern(end));
155                    options=intersect(options,oscoptions);
156                 elseif pattern(end-2)>pattern(end-1) %ramping down
157                    oscoptions=allopt(allopt<=pattern(end));
158                    options=intersect(options,oscoptions);
159                 elseif pattern(end-2)==pattern(end-1)
160                    %do nothing -->3 consecutive hours between ramping up and down
                           ↪  satisfied
161                 end
162              elseif pattern(end-1)>pattern(end) %if prev turbs decreasing
163                 oscoptions=allopt(allopt<=pattern(end));
```

```
                options=intersect(options,oscoptions);
            end
            %Out of the available options left, pick the next turbine setting
            if size(options,2)==1 %only 1 option left
                raw_options{wb}(i,j)=options;
            else
                raw_options{wb}(i,j)=randsample(options,1,true,...
                    transition_matrix{wb}(raw_options{wb}(i,j-1)+1,options+1));
            end
        end
    end
end

%Convert raw_options cells to long vectors containing all reservoirs per row
raw_options2=[];
for wb=1:size(x1_options,2)
    raw_options2=[raw_options2 raw_options{wb}];
end

%Check feasibility (all but WQ)
[c_new,~]=penalty_fcn(raw_options2,t,frequency,Q,ic_elev,...
    turbine_discharge,ELWS_limit,max_hrly_unit_change,WQ,...
    zero_gen_limit,xprev,ELWS_targets,tolerance,cache,Optimize_day_by_day);
funccount=funccount+size(raw_options2,1);
raw_options3=raw_options2(all(c_new<=tolerance,2),:);
feasible_options=[feasible_options; raw_options3];

%Remove duplicates
feasible_options=unique(feasible_options,'rows');
fprintf(['Feasible options found: ',...
    num2str(size(feasible_options,1)), '\n']);
if count==2 & isempty(feasible_options)
    y=[]; return
elseif count==5 & ~isempty(feasible_options)
    break
else
    count=count+1;
end
end

if size(feasible_options,1)>no_of_solns
    picks=randsample(size(feasible_options,1),no_of_solns);
    feasible_options=feasible_options(picks,:);
end
```

ga_results_plotting_nobanding.m

```
%% plot_data

%Determine the index in cache corresponding to the best solution from last
    ↪ generation
[~,index]=ismember(x_final_all{iter},cache.x,'rows');

t_all=[start_date:frequency:start_date+days_forward];
if Optimize_day_by_day==0
    day=days_forward;
end
for wb=1:size(CFG,2)

    maxdelay=max([WQ{wb}.DO_narx.inputDelays'; WQ{wb}.DO_narx.feedbackDelays']);
    data_start=frequency*(maxdelay-1);
    figure('units','normalized','outerposition',[0 0 1 1])
    % Title
```

```
    annotation('textbox',...
        [0.357741573033708 0.952787192414743 0.325808054820903
            ↪ 0.0410246887733755],...
        'String',{[CFG{wb}.Name ' Reservoir Optimization Results']},...
        'FontWeight','bold',...
        'FontSize',16,...
        'EdgeColor',[0.941176470588235 0.941176470588235 0.941176470588235],...
        'HorizontalAlignment','center');

    %% Subplot 1: Turbine discharge patterns as active units
    subplot(12,2,[1 3 5])
    Ax1=plot(tprev_ic,xprev_ic{wb},'k',...
        t_all(1:1+day*(1/frequency)),[xprev_ic{wb}(end) x0_all(wb,1:day*(1/
            ↪ frequency))],'b',...
        t_all(1:1+day*(1/frequency)),[xprev_ic{wb}(end) x_final{wb}],':r',...
        'LineWidth',2);
    xlabel('Julian Day'); xlim([t_all(1)-data_start t_all(1+day*24)]);
    set(gca,'YTick',0:1:no_of_units{wb});
    ylabel('Active turbine units')
    title('Active Turbine Units')
    ylim([0 max([xprev_ic{wb}(end) x0_all(wb,1:day*(1/frequency)) x_final{wb}])])
    ylims=get(gca,'ylim'); xlims=get(gca,'xlim'); xrange=xlims(2)-xlims(1);
        ↪ yrange=ylims(2)-ylims(1);
    text(xlims(1)+0.025*xrange,ylims(1)+0.9*yrange,'(a)','FontSize',12);

    %% Subplot 2: Turbine discharge patterns as flowrate
    turb_discharges_x0{wb}=interp1(Qprojected{wb}.QOT_BR1_T(:,1),Qprojected{wb}.
        ↪ QOT_BR1_T(:,2),t_all(1:1+day*(1/frequency)));
    [turb_discharges{wb},spill_discharges{wb},~,~,~]=...
        activeunits_to_discharges(x_final{wb},t,frequency,...
        Q{wb},ic_elev{wb},turbine_discharge{wb},ELWS_targets{wb},...
        [],[],Optimize_day_by_day);
    turb_discharges_prev{wb}=interp1(Q{wb}.QOT_BR1_T(:,1),Q{wb}.QOT_BR1_T(:,2),
        ↪ tprev_ic);
    subplot(12,2,[9 11 13])
    Ax2=plot(tprev_ic,turb_discharges_prev{wb},'k',...
        t_all(1:1+day*(1/frequency)),[turb_discharges_prev{wb}(end)
            ↪ turb_discharges_x0{wb}(2:end)],'b',...
        t_all(1:1+day*(1/frequency)),[turb_discharges_prev{wb}(end)
            ↪ turb_discharges{wb}(2:end)],':r','LineWidth',2);
    xlabel('Julian Day'); xlim([t_all(1)-data_start t_all(1+day*(1/frequency))]);
    ylabel('Turbine discharge, cms')
    title('Turbine Discharges')
    ylims=get(gca,'ylim'); xlims=get(gca,'xlim'); xrange=xlims(2)-xlims(1);
        ↪ yrange=ylims(2)-ylims(1);
    text(xlims(1)+0.025*xrange,ylims(1)+0.9*yrange,'(b)','FontSize',12);

    %% Subplot 3: Spill discharge patterns as flowrate
    spill_discharges_x0{wb}=interp1(Qprojected{wb}.QOT_BR1_S(:,1),Qprojected{wb}.
        ↪ QOT_BR1_S(:,2),t_all(1:1+day*(1/frequency)));
    if size(spill_discharges{wb},2)==1
        spill_discharges{wb}=ones(1,size(t_all(1:1+day*(1/frequency)),2))*
            ↪ spill_discharges{wb};
    else
        spill_discharges{wb}=interp1([start_date:1:start_date+days_forward-1],
            ↪ spill_discharges{wb},t_all(1:1+day*(1/frequency)));
    end
    spill_discharges_prev{wb}=interp1(Qprojected{wb}.QOT_BR1_S(:,1),Qprojected{wb
        ↪ }.QOT_BR1_S(:,2),tprev_ic);
    subplot(12,2,[17 19 21])
    Ax2=plot(tprev_ic,spill_discharges_prev{wb},'k',...
        t_all(1:1+day*(1/frequency)),[spill_discharges_prev{wb}(end)
            ↪ spill_discharges_x0{wb}(2:end)],'b',...
        t_all(1:1+day*(1/frequency)),[spill_discharges_prev{wb}(end)
```

```
        ↪ spill_discharges{wb}(2:end)],':r','LineWidth',2);
xlabel('Julian Day'); xlim([t_all(1)-data_start t_all(1+day*(1/frequency))]);
ylabel('Spill discharge, cms')
title('Spill Discharges')
if all([spill_discharges_prev{wb}(end) spill_discharges_x0{wb}
    ↪ spill_discharges{wb}]==0)
   ylim([0 1])
end
ylims=get(gca,'ylim'); xlims=get(gca,'xlim'); xrange=xlims(2)-xlims(1);
    ↪ yrange=ylims(2)-ylims(1);
text(xlims(1)+0.025*xrange,ylims(1)+0.9*yrange,'(c)','FontSize',12);

%% Subplot 4: Headwater elevations
clearvars HWs_x0
[~,~,HWs_x0{wb},~,~]=activeunits_to_discharges(x0_all(wb,1:day*(1/frequency))
    ↪ ,t_all(1:1+day*(1/frequency)),...
   frequency,Qprojected{wb},ic_elev_first{wb},...
   turbine_discharge{wb},[],[],[],Optimize_day_by_day);
HWs_prev{wb}=interp1(Q{wb}.ELWS(:,1),Q{wb}.ELWS(:,2),tprev_ic);
HWs{wb}=cache.HWs(index,:);
subplot(12,2,[2 4 6])
Ax3=plot(tprev_ic,HWs_prev{wb},'k',...
   t_all(1:1+day*(1/frequency)),HWs_x0{wb},'b',...
   t_all(1:1+day*(1/frequency)),HWs{wb},':r','LineWidth',2);
hold on;
h5=plot([t_all(1) t_all(1+day*(1/frequency))],...
   [ELWS_limit{wb}(1) ELWS_limit{wb}(1)],':k',...
   'LineWidth',1.5);
plot([t_all(1) t_all(1+day*(1/frequency))],...
   [ELWS_limit{wb}(2) ELWS_limit{wb}(2)],':k',...
   'LineWidth',1.5)
if Optimize_day_by_day==0
   h6=scatter(ELWS_targets{wb}(end,1),ELWS_targets{wb}(end,2));
else
   h6=scatter(ELWS_targets{wb}(:,1),ELWS_targets{wb}(:,2));
end
hold off;
xlabel('Julian Day'); xlim([t_all(1)-data_start t_all(1+day*(1/frequency))]);
ylabel('Elevation, m')
title('Headwater Elevation')
ylims=get(gca,'ylim'); xlims=get(gca,'xlim'); xrange=xlims(2)-xlims(1);
    ↪ yrange=ylims(2)-ylims(1);
text(xlims(1)+0.025*xrange,ylims(1)+0.9*yrange,'(d)','FontSize',12);

%% Subplot 5: Discharge DO
DO_pred_x0{wb}=interp1(Qprojected{wb}.CWO(Qprojected{wb}.CWO(:,2)~=0,1),
    ↪ Qprojected{wb}.CWO(Qprojected{wb}.CWO(:,2)~=0,2),t_all(2:1+day*(1/
    ↪ frequency)));
[turb_discharges2,spill_discharges2,~,~,~]=...
   activeunits_to_discharges(x_final{wb},t,frequency,Q{wb},ic_elev{wb},...
   turbine_discharge{wb},ELWS_targets{wb},[],[],Optimize_day_by_day);
DO_pred{wb}=narx_predictions(WQ{wb}.DO_narx,frequency,t,Q{wb},...
   x_final{wb},turb_discharges2,spill_discharges2,[],Q{wb}.CWO,...
   'do',Optimize_day_by_day);
flowout_x0=turb_discharges_x0{wb}(2:end)+spill_discharges_x0{wb}(2:end);
flowout=turb_discharges{wb}(2:end)+spill_discharges{wb}(2:end);
DO_pred_x0{wb}(flowout_x0==0)=nan; DO_pred{wb}(flowout==0)=nan;
Output_no0s{wb}=interp1(Qprojected{wb}.CWO(find(Qprojected{wb}.CWO(:,2)~=0)
    ↪ ,1),...
   Qprojected{wb}.CWO(find(Qprojected{wb}.CWO(:,2)~=0),2),...
   [t_all(1)-data_start:frequency:t_all(1)])';
if interp1(Qprojected{wb}.QOT_BR1_T(:,1),Qprojected{wb}.QOT_BR1_T(:,2),...
      tprev_ic(end))==0 & ...
      interp1(Qprojected{wb}.QOT_BR1_S(:,1),Qprojected{wb}.QOT_BR1_S(:,2),...
```

```
123         tprev_ic(end))==0
124       DOinitcon{wb}=nan;
125    else
126       DOinitcon{wb}=Output_no0s{wb}(end);
127    end
128    Outputprev{wb}=interp1([t_all(1)-data_start:frequency:t_all(1)],Output_no0s{
           ↪ wb},...
129       tprev_ic);
130    j=find(interp1(Qprojected{wb}.QOT_BR1_T(:,1),Qprojected{wb}.QOT_BR1_T(:,2)
           ↪ ,...
131       tprev_ic)==0 & ...
132       interp1(Qprojected{wb}.QOT_BR1_S(:,1),Qprojected{wb}.QOT_BR1_S(:,2),...
133       tprev_ic)==0);
134    Outputprev{wb}(j)=nan; clearvars j
135    subplot(12,2,[10 12 14])
136    h1=plot(tprev_ic,Outputprev{wb},'k','LineWidth',2);
137    hold on;
138    h2=plot(t_all(1:1+day*(1/frequency)),[DOinitcon{wb} DO_pred_x0{wb}],'b','
           ↪ LineWidth',2);
139    h3=plot(t_all(1:1+day*(1/frequency)),[DOinitcon{wb} DO_pred{wb}],':r','
           ↪ LineWidth',2);
140    h7=plot(cache.t,[DOinitcon{wb} cache.DO(index,:)],'g','LineWidth',2);
141    if ~isnan(WQ{wb}.DO_limit(1))
142       h5=plot([t_all(1) t_all(1+day*(1/frequency))],[WQ{wb}.DO_limit(1) WQ{wb}.
              ↪ DO_limit(1)],':k',...
143          'LineWidth',1.5);
144    elseif ~isnan(WQ{wb}.DO_limit(2))
145       plot([t_all(1) t_all(1+day*(1/frequency))],[WQ{wb}.DO_limit(2) WQ{wb}.
              ↪ DO_limit(2)],':k',...
146          'LineWidth',1.5);
147    end
148    xlabel('Julian Day'); xlim([t_all(1)-data_start t_all(1+day*(1/frequency))]);
149    ylabel('DO, mg/L');
150    ylim([min([DOinitcon{wb} cache.DO(index,:) min(DO_pred{wb}) min(DO_pred_x0{wb
           ↪ }) Output_no0s{wb}' WQ{wb}.DO_limit(1) WQ{wb}.DO_limit(2)])-.25...
151       max([DOinitcon{wb} cache.DO(index,:) max(DO_pred{wb}) max(DO_pred_x0{wb})
              ↪ Output_no0s{wb}' WQ{wb}.DO_limit(1) WQ{wb}.DO_limit(2)])+.25]);
152    title('Discharge DO Predictions')
153    ylims=get(gca,'ylim'); xlims=get(gca,'xlim'); xrange=xlims(2)-xlims(1);
           ↪ yrange=ylims(2)-ylims(1);
154    text(xlims(1)+0.025*xrange,ylims(1)+0.9*yrange,'(e)','FontSize',12);
155    str=['AME = ', sprintf('%5.3f',AME{wb}.DO), ' mg/L'];
156    text(xlims(1)+0.025*xrange,ylims(1)+0.1*yrange,str,'FontSize',12);
157    clearvars W2_no0s_smooth index2 W2_no0s str slack_compute non_nan_count
158
159    %% Subplot 5: Discharge Temp
160    Temp_pred_x0{wb}=interp1(Qprojected{wb}.TWO(Qprojected{wb}.TWO(:,2)~=0,1),
           ↪ Qprojected{wb}.TWO(Qprojected{wb}.TWO(:,2)~=0,2),t_all(2:1+day*(1/
           ↪ frequency)));
161    Temp_pred{wb}=narx_predictions(WQ{wb}.Temp_narx,frequency,t,Q{wb},...
162       x_final{wb},turb_discharges2,spill_discharges2,[],Q{wb}.TWO,...
163       'temp',Optimize_day_by_day);
164    Temp_pred_x0{wb}(flowout_x0==0)=nan; Temp_pred{wb}(flowout==0)=nan;
165    clearvars flowout_x0
166    Output_no0s{wb}=interp1(Qprojected{wb}.TWO(find(Qprojected{wb}.TWO(:,2)~=0)
           ↪ ,1),...
167       Qprojected{wb}.TWO(find(Qprojected{wb}.TWO(:,2)~=0),2),...
168       [t_all(1)-data_start:frequency:t_all(1)])';
169    if interp1(Qprojected{wb}.QOT_BR1_T(:,1),Qprojected{wb}.QOT_BR1_T(:,2),...
170          tprev_ic(end))==0 & ...
171          interp1(Qprojected{wb}.QOT_BR1_S(:,1),Qprojected{wb}.QOT_BR1_S(:,2),...
172          tprev_ic(end))==0
173       Tempinitcon{wb}=nan;
174    else
```

```
        Tempinitcon{wb}=Output_no0s{wb}(end);
    end
    Outputprev{wb}=interp1([t_all(1)-data_start:frequency:t_all(1)],Output_no0s{wb},...
        tprev_ic);
    j=find(interp1(Qprojected{wb}.QOT_BR1_T(:,1),Qprojected{wb}.QOT_BR1_T(:,2),...
        tprev_ic)==0 & ...
        interp1(Qprojected{wb}.QOT_BR1_S(:,1),Qprojected{wb}.QOT_BR1_S(:,2),...
        tprev_ic)==0);
    Outputprev{wb}(j)=nan; clearvars j
    subplot(12,2,[18 20 22])
    h1=plot(tprev_ic,Outputprev{wb},'k','LineWidth',2);
    hold on;
    h2=plot(t_all(1:1+day*(1/frequency)),[Tempinitcon{wb} Temp_pred_x0{wb}],'b','LineWidth',2);
    h3=plot(t_all(1:1+day*(1/frequency)),[Tempinitcon{wb} Temp_pred{wb}],':r','LineWidth',2);
    h7=plot(cache.t,[Tempinitcon{wb} cache.T(index,:)],'g','LineWidth',2);
    if ~isnan(WQ{wb}.Temp_limit(1))
        h5=plot([t_all(1) t_all(1+day*(1/frequency))],[WQ{wb}.Temp_limit(1) WQ{wb}.Temp_limit(1)],':k',...
        'LineWidth',1.5);
    elseif ~isnan(WQ{wb}.Temp_limit(2))
        h5=plot([t_all(1) t_all(1+day*(1/frequency))],[WQ{wb}.Temp_limit(2) WQ{wb}.Temp_limit(2)],':k',...
        'LineWidth',1.5);
    end
    xlabel('Julian Day'); xlim([t_all(1)-data_start t_all(1+day*(1/frequency))]);
    ylabel('Temperature, C');
    ylim([min([Tempinitcon{wb} cache.T(index,:) min(Temp_pred{wb}) min(Temp_pred_x0{wb}) Output_no0s{wb}' WQ{wb}.Temp_limit(1) WQ{wb}.Temp_limit(2)])-.25...
        max([Tempinitcon{wb} cache.T(index,:) max(Temp_pred{wb}) max(Temp_pred_x0{wb}) Output_no0s{wb}' WQ{wb}.Temp_limit(1) WQ{wb}.Temp_limit(2)])+.25]);
    title('Discharge Temperature Predictions')
    ylims=get(gca,'ylim'); xlims=get(gca,'xlim'); xrange=xlims(2)-xlims(1); yrange=ylims(2)-ylims(1);
    text(xlims(1)+0.025*xrange,ylims(1)+0.9*yrange,'(f)','FontSize',12);
    str=['AME = ', sprintf('%5.3f',AME{wb}.T), ' C'];
    text(xlims(1)+0.025*xrange,ylims(1)+0.1*yrange,str,'FontSize',12);
    clearvars W2_no0s_smooth W2_no0s_smooth2 index2 W2_no0s flowout str slack_compute non_nan_count

    legend1=legend([h1 h2 h3 h7 h5 h6],'Past Values',...
        'Projected Operations',...
        'Optimal Solution',...
        'W2 Validation at Optimal Solution',...
        'Constraint Bounds',...
        'Target Elevations');
    set(legend1,...
        'Position',[0.39086885358981 0.0131729985010991 0.256670797003518 0.119367775250152],...
        'FontSize',10);

end
```

initial_NARX_model_generation.m

```
%Initial NARX model generation

wb=1;
```

```
global funccount_cache_global funccount_tot_global
funccount_cache_global=0; funccount_tot_global=0;
clearvars xprev tprev
x_final{wb}=[];
%Previous turbine pattern for the year (if supplied)
if isempty(CFG{wb}.ForecastTurbinePattern)
   xprev{wb}=actual_turb_ops(tprev_round,Qprojected{wb},elevtemp{wb},
       ↪ turbine_discharge{wb},no_of_units{wb});
else
   prevturbpattern=dlmread(CFG{wb}.PreviousTurbinePattern,'\t',1,0);
   for i=1:size(tprev_round,2)
      index=find(prevturbpattern(:,1)<=tprev_round(i));
      xprev{wb}(i)=prevturbpattern(index(end),2);
   end
   clearvars i prevturbpattern index
end
tprev=[t_all(1)-max(cell2mat(zero_gen_limit(:)))*frequency:frequency:t_all(1)];
xprev_ic=xprev; tprev_ic=tprev;

WQ_initial=WQ;
WQ_initial{wb}.DO_limit=nan(size(WQ{wb}.DO_limit)); WQ_initial{wb}.Temp_limit=
    ↪ nan(size(WQ{wb}.Temp_limit));

%Optimization timeperiod
if Optimize_day_by_day==1
   t=[start_date+day-1:frequency:start_date+day];
else
   t=t_all;
end

%Set initial condition elevation
for wb=1:size(CFG,2)
   ic_elev{wb}=ic_elev_first{wb};
   if ic_elev_first{wb}<ELWS_limit{wb}(1)
      L.warn('INITIALIZATION', ['Reservoir ', num2str(wb),' initial elevation of
          ↪  ' cell2mat(ic_elev_first{wb}) ' m is less than ELWS lower limit (
          ↪ firm constraint). Expanding ELWS limits to continue with
          ↪ optimization.']);
      ELWS_limit{wb}(1)=ic_elev_first{wb};
   elseif ic_elev_first{wb}>ELWS_limit{wb}(2)
      L.warn('INITIALIZATION', ['Reservoir ', num2str(wb),' initial elevation of
          ↪  ' cell2mat(ic_elev_first{wb}) ' m is greater than ELWS upper limit
          ↪  (firm constraint). Expanding ELWS limits to continue with
          ↪ optimization.']);
      ELWS_limit{wb}(2)=ic_elev_first{wb};
   end
end
%Find possible values for x(1) (based on previous zero_gen_limit turbs)
options=[0:no_of_units{wb}];
% (1) Eliminate options based on change in active unit violations
if ~isnan(max_hrly_unit_change{wb})
   auvoptions=[xprev{wb}(end)-max_hrly_unit_change{wb}:...
      xprev{wb}(end)+max_hrly_unit_change{wb}];
   options=intersect(options,auvoptions);
end
% (2) Non-integer constraint (assumed in selection algorithm)
% (3) Eliminate options based on zero generation hourly limit
if ~isnan(zero_gen_limit{wb})
   if sum(xprev{wb}(end-zero_gen_limit{wb}+1:end))==0
      zghloptions=[1:no_of_units{wb}]; %if previous zero_gen_limit hrs had zero
          ↪ total flow, must have flow next hr
      options=intersect(options,zghloptions);
   end
end
```

```
% (4) Eliminate options that violate oscillations constraint - violates whenever the number of turbines increases and then decreases within 2 hours, or vice versa
allopt=[0:no_of_units{wb}];
if xprev{wb}(end-1)<xprev{wb}(end) %if prev turbs increasing
   oscoptions=allopt(allopt>=xprev{wb}(end));
   options=intersect(options,oscoptions);
elseif xprev{wb}(end-1)==xprev{wb}(end) %need 3 hrs btwn ramping up and down
   if xprev{wb}(end-2)<xprev{wb}(end-1) %ramping up
      oscoptions=allopt(allopt>=xprev{wb}(end));
      options=intersect(options,oscoptions);
   elseif xprev{wb}(end-2)>xprev{wb}(end-1) %ramping down
      oscoptions=allopt(allopt<=xprev{wb}(end));
      options=intersect(options,oscoptions);
   elseif xprev{wb}(end-2)==xprev{wb}(end-1)
      %do nothing -->3 consecutive hours between ramping up and down satisfied
   end
elseif xprev{wb}(end-1)>xprev{wb}(end) %if prev turbs decreasing
   oscoptions=allopt(allopt<=xprev{wb}(end));
   options=intersect(options,oscoptions);
end
x1_options{wb}=options;
if isempty(x1_options{wb})
   L.fatal('OPTIMIZATION','Based on previous turbine pattern, there is no feasible first hour turbine level.');
   return
end
clearvars tprev options auvoptions zghloptions allopt oscoptions
[pop0,~,~,~]=...
   create_feasible_initpop(ga_pop_size,[],...
   x1_options,frequency,Q,ic_elev,MW_rating,no_of_units,t,...
   max_hrly_unit_change,zero_gen_limit,turbine_discharge,...
   ELWS_limit,WQ_initial,cost_curve_MW,xprev,...
   [],ELWS_targets,tolerance,cache,Optimize_day_by_day,...
   transition_matrix,1);
clearvars WQ_initial
%Take initial pop pop0 and pick out Initialtrainingsetsize number of scenarios to run through W2 using kmeans clusters
wb=1; day=1; ELWS_limit_subproblem{day}=ELWS_limit;
%Determine x0, actual turbine operations, to seed initial population
if Optimize_day_by_day==1
   x0(wb,:)=x0_all(wb,(day-1)*(1/frequency)+1:day*(1/frequency));
else
   x0(wb,:)=x0_all(wb,:);
end
[~, y_dollars1]=power_value(x0(wb,:),t,cost_curve_MW{wb},...
   MW_rating{wb});
if size(ELWS_targets{wb}(:,1),1)==1
   elev_soft_penalty_coeff{day}(wb)=interp1(ELWS_limit_subproblem{day}{wb}(:),...
      elev_soft_penalty_coeff_constant,...
      ELWS_targets{wb}(:,2),'linear','extrap')*y_dollars1; %$/m with cost curve, MWh/m with all cc=1
else
   elev_soft_penalty_coeff{day}(wb)=interp1(ELWS_limit_subproblem{day}{wb}(:),...
      elev_soft_penalty_coeff_constant,...
      interp1(ELWS_targets{wb}(:,1),ELWS_targets{wb}(:,2),start_date+day),...
      'linear','extrap')*y_dollars1; %$/m with cost curve, MWh/m with all cc=1
end
FitnessFunction = @(x) -obj_fcn(x,t,cost_curve_MW,...
   MW_rating,elev_soft_penalty_coeff{day},...
   ELWS_targets,frequency,Q,ic_elev,...
   turbine_discharge,cache,Optimize_day_by_day);
```

```
[~,b]=sort(FitnessFunction(pop0),'ascend'); pop0=pop0(b,:);
for a=1:500
   [idx(:,a),~,~,D{a}]=kmeans(pop0,Initialtrainingsetsize);
   B=unique(idx(:,a));
   group_var(a)=var(histc(idx(:,a),B));
end
%Pick the cluster that minimizes the max group size (i.e., results in fairly even distribution)
[~,a]=min(group_var); idx=idx(:,a); D=D{a};
for i=1:Initialtrainingsetsize
   %Pick random one from each cluster
   b=find(idx==i); a=randsample(b,1); init_train_set(i,:)=pop0(a,:);
end
clearvars a b B group_var i

%Create Qtrainingpop for each feasible_options entry (QOT_BR1_T, QOT_BR1_S, ELWS, CWO, TWO)
for i=1:Initialtrainingsetsize
   xtr{1}=init_train_set(i,:);
   Qtrainingpop{i}=updateQ(Q,CFG,xtr,t,frequency,ic_elev,turbine_discharge,...
      WQ,xprev,ELWS_targets,cache,Optimize_day_by_day);
end
%Run each row in feasible_options through W2 (only works for 1-day, 1-wb problems for now), and update cache with these values as well
for trindex=1:Initialtrainingsetsize
   fprintf(['Running initial training point ' num2str(trindex) ' for reservoir #', num2str(wb),'. \n']);
   directory=['results/w2_iter0_trpt' num2str(trindex) '_wb' num2str(wb)];
   runW2trainingpop;
end
while istaskrunning('w2.exe') end %is w2 still running? if so, hold on
system('taskkill /F /IM cmd.exe'); cache_size_pre=size(cache.x,1);
for trindex=1:Initialtrainingsetsize
   directory=['results/w2_iter0_trpt' num2str(trindex) '_wb' num2str(wb)];
   trainingpop=init_train_set; runW2trainingpop_part2;
end
clearvars s z zz zzz distances distance_mins start_index w2runstiming bestsolniter index pop b DO_pred T_pred w2timing trindex xtr idx f i a b D wb D2 correction directory distance_to_soln ii e d

for i=1:Initialtrainingsetsize
   cache.flag{size(cache.flag,1)+1,1}={'initial'};
end

NARX_retrain_trpt;
clearvars Qtrainingpop
```

narx_predictions.m

```
function pred=narx_predictions(NARX_model,frequency,t,Q,x,...
   turb_discharges,spill_discharges,mainstem_inflows,previous_Output,flag,...
   Optimize_day_by_day)

% Calculates WQ predictions using a trained family of NARX models
%
% Inputs:
% NARX_model - structure containing everything needed to make WQ
% discharge predictions, including:
% turb_colum - column in exogenous variables with turb flows
% spill_column - column in exogenous variables with spill flows
% inputDelays - delays for exogenous inputs
% feedbackDelays - delays for prediction feedbacks
% input_variables - 2 row cell containing variable names in first
```

```
% row and column number in second. For example, 'MET_WB1'
% contains multiple columns of data but only some may be used
% for NARX predictions
% bias - bias for each trained neural network
% weights - weights for each trained neural network (sum to 1)
% narx_net_closed - neural networks
% frequency - frequency of predictions (hourly=1/24)
% t  time series of JDAY values
% Q - all other inflows and outflows, interpolation settings,
% storage-elev curve, and tailwater curve
% x - hourly turbine time series (as rows for vectorizing!), integers
% between 0 and no_of_units
% turb_discharges - matrix the same size as x that includes the turbine
% discharge flowrates over the time t
% spill_discharges - spill discharge flowrates
% mainstem_inflows - structure containing Q, T, and DO with time series
% data from previous days' optimal solution
% previous_Output - the time series of previous outputs of the
% constitiuent being predicted by NARX model
% flag - 'do' if predicting DO, to check to make sure not <0
% Optimize_day_by_day - 1 if optimizing daily, 0 if optimizing all together
% Outputs:
% pred  vector of NARX model predictions for water quality, with NaN
% values anywhere turb+spill=0

if isempty(x)
   pred=[];
else

if isempty(mainstem_inflows)
   mainstem_inflows.Q=[];
   mainstem_inflows.T=[];
   mainstem_inflows.DO=[];
end
if exist('mainstem_inflows', 'var') && isfield(mainstem_inflows, 'Q')
   if isempty(mainstem_inflows.Q) mainstem_inflows.Q=[]; end
else
   mainstem_inflows.Q=[];
end
if exist('mainstem_inflows', 'var') && isfield(mainstem_inflows, 'T')
   if isempty(mainstem_inflows.T) mainstem_inflows.T=[]; end
else
   mainstem_inflows.T=[];
end
if exist('mainstem_inflows', 'var') && isfield(mainstem_inflows, 'DO')
   if isempty(mainstem_inflows.DO) mainstem_inflows.DO=[]; end
else
   mainstem_inflows.DO=[];
end

maxdelay=max([NARX_model.inputDelays'; NARX_model.feedbackDelays']);
data_start=frequency*(maxdelay-1);
timesteps=[t(1)-data_start:frequency:t];
Output_no0s=interp1(previous_Output(find(previous_Output(:,2)~=0),1),...
   previous_Output(find(previous_Output(:,2)~=0),2),timesteps)';
clearvars timesteps
y1=con2seq([Output_no0s' nan(1,size(x,2))]);
timesteps2=[t(1)-data_start:frequency:t t(2:end)];
Inputs=nan(size(timesteps2,2),size(NARX_model.input_variables,2));
index_QIN_BR1=[]; index_TIN_BR1=[]; index_CIN_BR1=[];
for i=1:size(NARX_model.input_variables,2)
   %If mainstem_inflows are provided and the variable is BR1 Q, T, or DO
   if ~isempty(mainstem_inflows.Q) & ...
         isequal(NARX_model.input_variables{1,i},'QIN_BR1')
```

```
79          index_QIN_BR1=i;
80      end
81      if ~isempty(mainstem_inflows.T) & ...
82              isequal(NARX_model.input_variables{1,i},'TIN_BR1')
83          index_TIN_BR1=i;
84      end
85      if ~isempty(mainstem_inflows.DO) & ...
86              isequal(NARX_model.input_variables{1,i},'CIN_BR1')
87          index_CIN_BR1=i;
88      end
89      Inputs(:,i)=interp1(Q.(sprintf(NARX_model.input_variables{1,i}))(:,1),...
90          Q.(sprintf(NARX_model.input_variables{1,i}))(:,NARX_model.input_variables
                ↪ {2,i}+1),...
91          timesteps2);
92  end
93  clearvars i timesteps2
94  pred=nan(size(x,1),size(x,2));
95  for i=1:size(x,1) %attempt to vectorize this part later
96      %Update mainstem_inflows, if necessary
97      if ~isempty(index_QIN_BR1)
98          Inputs(size(Inputs,1)-size(mainstem_inflows.Q,2)+1:...
99              size(Inputs,1),index_QIN_BR1)=mainstem_inflows.Q(i,:)';
100     end
101     if ~isempty(index_TIN_BR1)
102         Inputs(size(Inputs,1)-size(mainstem_inflows.T,2)+1:...
103             size(Inputs,1),index_TIN_BR1)=mainstem_inflows.T(i,:)';
104     end
105     if ~isempty(index_CIN_BR1)
106         Inputs(size(Inputs,1)-size(mainstem_inflows.DO,2)+1:...
107             size(Inputs,1),index_CIN_BR1)=mainstem_inflows.DO(i,:)';
108     end
109     %Update turbine outflow and spill outflow columns, if necessary
110     if ~isempty(turb_discharges)
111         Inputs(size(Inputs,1)-size(turb_discharges,2)+...
112             1:size(Inputs,1),NARX_model.turb_column)=...
113             turb_discharges(i,:)';
114     end
115     if ~isempty(spill_discharges)
116         if Optimize_day_by_day==1 %optimize each day sequentially
117             Inputs(size(Inputs,1)-size(turb_discharges,2)+...
118                 1:size(Inputs,1),NARX_model.spill_column)=...
119                 spill_discharges(i);
120         else %optimize all days together, so each col in spill_discharges is each
                ↪ day
121             for ii=1:size(spill_discharges,2)
122                 spill_values(i,(1/frequency)*(ii-1)+1:(1/frequency)*(ii)+1)=...
123                     spill_discharges(i,ii);
124             end
125             Inputs(size(Inputs,1)-size(turb_discharges,2)+1:...
126                 size(Inputs,1),NARX_model.spill_column)=...
127                 spill_values(i,:)';
128             clearvars ii
129         end
130     end
131     u1 = con2seq(Inputs');
132     if size(NARX_model.narx_net_closed,2)==1
133         if iscell(NARX_model.narx_net_closed)
134             [p1,Pi1,Ai1,t1]=preparets(NARX_model.narx_net_closed{:},u1,{},y1);
135             yp1(1,:)=NARX_model.narx_net_closed{:}(p1,Pi1,Ai1);
136         else
137             [p1,Pi1,Ai1,t1]=preparets(NARX_model.narx_net_closed,u1,{},y1);
138             yp1(1,:)=NARX_model.narx_net_closed(p1,Pi1,Ai1);
139         end
140     else
```

```
      for j=1:size(NARX_model.narx_net_closed,2)
         [p1,Pi1,Ai1,t1]=preparets(NARX_model.narx_net_closed{j},u1,{},y1);
         yp1(j,:)=NARX_model.narx_net_closed{j}(p1,Pi1,Ai1);
      end
   end
   yp1=cell2mat(yp1);
   if size(NARX_model.weights,1)==1
      yp1=yp1-NARX_model.bias;
      pred(i,:)=yp1;
   else
      yp1=bsxfun(@minus,yp1,NARX_model.bias);
      pred(i,:)=sum(bsxfun(@times,NARX_model.weights,yp1));
   end
   clearvars yp1
end
clearvars i j
if strcmp(flag,'do')
   pred=max(0,pred); %can't have negative concentrations of DO
end
for i=1:size(x,1)
   j=[];
   if ~isempty(spill_discharges)
      if Optimize_day_by_day==1 %optimize each day sequentially
         if all(spill_discharges(i)==0)
            j=find(x(i,:)==0);
         else
            if size(spill_discharges(i,:),2)==1 %if solving subproblem
               j=[];
            else
               j=find(turb_discharges(i,2:end)==0 & spill_discharges(i,2:end)
                   ↪ ==0); %if solving final solution over all subproblems
            end
         end
      else %optimize all days together, so each col in spill_discharges is each
          ↪ day
         j=find(turb_discharges(i,2:end)==0 & spill_values(i,2:end)==0);
      end
   else
      j=find(x(i,:)==0 & interp1(Q.QOT_BR1_S(:,1),Q.QOT_BR1_S(:,2),t(2:end))==0)
          ↪ ;
   end
   pred(i,j)=nan;
end
clearvars i j spill_values

end
```

NARX_retrain_trpt.m

```
%Retrain temperature and DO NARX models for wb
%For each iteration, add the new W2 validation run data to the training data set
    ↪ , and then retrain. This means the training set grows with each iteration
    ↪ .

wb=1; %Assume 1 wb system for now

if ~exist('Inputs')
   Inputs{wb}.discharge_DO=[];
   Inputs{wb}.discharge_Temp=[];
end

%% DO validation run
if WQ{wb}.DO_valid_check==1
```

```
13    if size(trainingpop,1)>0
14       for trindex=1:size(trainingpop,1)
15          index=size(Inputs{wb}.discharge_DO,2);
16          timesteps=[t_all(1)-max(WQ{wb}.DO_narx.inputDelays)/24:(1/24):t_all(end
                ↪ )]';
17          vars=WQ{wb}.DO_narx.input_variables;
18          Inputs{wb}.discharge_DO{index+1}=[];
19          for i=1:size(vars,2)
20             if strfind(char(vars(1,i)),'TIN')
21                flow_variable=strrep(char(vars(1,i)),'TIN','QIN');
22             elseif strfind(char(vars(1,i)),'CIN')
23                flow_variable=strrep(char(vars(1,i)),'CIN','QIN');
24             elseif strfind(char(vars(1,i)),'TTR')
25                flow_variable=strrep(char(vars(1,i)),'TTR','QTR');
26             elseif strfind(char(vars(1,i)),'CTR')
27                flow_variable=strrep(char(vars(1,i)),'CTR','QTR');
28             else
29                flow_variable=char(vars(1,i));
30             end
31             if ~strcmp(char(vars(1,i)),'MET_WB1') %assume interpolation for MET
                   ↪ data
32                for ii=1:size(Qtrainingpop{trindex}{wb}.interpolation,2)
33                   if strcmp(char(Qtrainingpop{trindex}{wb}.interpolation(1,ii)),
                         ↪ flow_variable)
34                      break
35                   end
36                end
37                if strcmp(char(Qtrainingpop{trindex}{wb}.interpolation(3,ii)),'ON
                      ↪ ')
38                   Inputs{wb}.discharge_DO{index+1}(:,i)=interp1(Qtrainingpop{
                         ↪ trindex}{wb}.(vars{1,i})(:,1),...
39                      Qtrainingpop{trindex}{wb}.(vars{1,i})(:,vars{2,i}+1),
                            ↪ timesteps);
40                elseif strcmp(char(Qtrainingpop{trindex}{wb}.interpolation(3,ii))
                      ↪ ,'OFF')
41                   for iii=1:size(timesteps,1)
42                      index2=find(Qtrainingpop{trindex}{wb}.(vars{1,i})(:,1)<=
                            ↪ timesteps(ii),1,'last');
43                      Inputs{wb}.discharge_DO{index+1}(iii,i)=Qtrainingpop{
                            ↪ trindex}{wb}.(vars{1,i})(index2,vars{2,i}+1);
44                   end
45                end
46             else
47                Inputs{wb}.discharge_DO{index+1}(:,i)=interp1(Qtrainingpop{
                      ↪ trindex}{wb}.(vars{1,i})(:,1),...
48                   Qtrainingpop{trindex}{wb}.(vars{1,i})(:,vars{2,i}+1),timesteps
                         ↪ );
49             end
50          end
51          DO_noNAN=interp1(DO{trindex}(~isnan(DO{trindex}(:,2)),1),...
52             DO{trindex}(~isnan(DO{trindex}(:,2)),2),timesteps);
53          %Fill in Nans at the end
54          a=DO_noNAN(~isnan(DO_noNAN)); DO_noNAN(isnan(DO_noNAN))=a(end);
55          turbs=interp1(Qtrainingpop{trindex}{wb}.QOT_BR1_T(:,1),Qtrainingpop{
                ↪ trindex}{wb}.QOT_BR1_T(:,2),timesteps);
56          spills=interp1(Qtrainingpop{trindex}{wb}.QOT_BR1_S(:,1),Qtrainingpop{
                ↪ trindex}{wb}.QOT_BR1_S(:,2),timesteps);
57          flowout=turbs+spills; DO_noNAN(flowout==0)=nan;
58
59          %Output data
60          Output{wb}.discharge_DO{index+1}(:,1)=DO_noNAN;
61       end
62    end
63    for i=1:size(Inputs{wb}.discharge_DO,2)
```

```
64        %Convert to cells
65        Inputs_seq{wb}.discharge_DO{i}=con2seq(Inputs{wb}.discharge_DO{i}');
66        Output_seq{wb}.discharge_DO{i}=con2seq(Output{wb}.discharge_DO{i}');
67     end
68     clearvars i ii iii flow_variable index a DO_noNAN turbs spills flowout index2
           ↪ vars timesteps
69
70     %Combine them all into single Input and Output cell arrays
71     Inputs_seq_mul{wb}.discharge_DO=catsamples(Inputs_seq{wb}.discharge_DO{:},'
           ↪ pad');
72     Output_seq_mul{wb}.discharge_DO=catsamples(Output_seq{wb}.discharge_DO{:},'
           ↪ pad');
73
74     %Train DO model - start with best DO model from before (greatest weight)
75     fprintf(['Training 5 DO models and picking the best \n'])
76     for i=1:5
77        inputDelays = [0 1 12];
78        feedbackDelays = [1];
79        hiddenNeurons=[10];
80        narx_net{i} = narxnet(inputDelays,feedbackDelays,hiddenNeurons);
81        narx_net{i}.divideFcn = 'dividerand';
82        % The property DIVIDEMODE set to TIMESTEP means that targets are divided
83        % into training, validation and test sets according to timesteps.
84        % For a list of data division modes type: help nntype_data_division_mode
85        narx_net{i}.divideMode = 'time'; % Divide up every value
86        narx_net{i}.divideParam.trainRatio = 70/100;
87        narx_net{i}.divideParam.valRatio = 15/100;
88        narx_net{i}.divideParam.testRatio = 15/100;
89        narx_net{i}.trainParam.min_grad = 1e-10;
90        narx_net{i}.trainFcn = 'trainlm';
91        narx_net{i}.trainParam.showWindow=0;
92        narx_net{i}.trainParam.showCommandLine=1;
93        [Xs,Xi,Ai,Ts]=preparets(narx_net{i},Inputs_seq_mul{wb}.discharge_DO,{},
              ↪ ...
94           Output_seq_mul{wb}.discharge_DO);
95        [narx_net{i},~]=train(narx_net{i},Xs,Ts,Xi,Ai,'UseParallel','yes');
96        narx_net_closed{i} = closeloop(narx_net{i});
97        narx_net_closed{i}.trainParam.mu_max=1e14;
98        [Xs,Xi,Ai,Ts]=preparets(narx_net_closed{i},Inputs_seq_mul{wb}.discharge_DO
              ↪ ,{}, ...
99           Output_seq_mul{wb}.discharge_DO);
100       [narx_net_closed{i},tr{i}]=train(narx_net_closed{i},Xs,Ts,Xi,Ai,'
              ↪ UseParallel','yes');
101    end
102    for i=1:5 tr2(i)=tr{i}.best_perf; end
103    [~,b]=min(tr2); WQ{wb}.DO_narx.narx_net_closed=narx_net_closed{b};
104    yp1= WQ{wb}.DO_narx.narx_net_closed(Xs,Xi,Ai);
105    %Calculate bias & standard dev using only predictions at test timepoints
106    bias=cell2mat(yp1(tr{b}.testInd))-cell2mat(Ts(tr{b}.testInd)); bias=nanmean(
           ↪ bias);
107    allerrors=(cell2mat(yp1(tr{b}.testInd))-bias)-cell2mat(Ts(tr{b}.testInd));
108    allerrors=allerrors(~isnan(allerrors));
109    [~,sigmahat] = normfit(allerrors);
110    WQ{wb}.DO_narx.bias=bias;
111    WQ{wb}.DO_narx.weights=1;
112    WQ{wb}.DO_narx.inputDelays=inputDelays;
113    WQ{wb}.DO_narx.std_dev=sigmahat;
114    WQ{wb}.DO_narx.Inputs=Inputs{wb}.discharge_DO;
115    WQ{wb}.DO_narx.Output=Output{wb}.discharge_DO;
116    if isfield(WQ{wb}.DO_narx,'train_time')
117       WQ{wb}.DO_narx=rmfield(WQ{wb}.DO_narx,{'train_time'});
118    end
119    if isfield(WQ{wb}.DO_narx,'Discharge_DO_no0s')
120       WQ{wb}.DO_narx=rmfield(WQ{wb}.DO_narx,{'Discharge_DO_no0s'});
```

```
121     end
122     clearvars b Xs Xi Ai Ts tr tr2 b yp1 TS bias narx_net_closed narx_net muhat
            ↪ sigmahat
123 end
124
125 %% Temp validation run
126 if WQ{wb}.Temp_valid_check==1
127     if size(trainingpop,1)>0
128         for trindex=1:size(trainingpop,1)
129             index=size(Inputs{wb}.discharge_Temp,2);
130             timesteps=[t_all(1)-max(WQ{wb}.Temp_narx.inputDelays)/24:(1/24):t_all(
                    ↪ end)]';
131             vars=WQ{wb}.Temp_narx.input_variables;
132             Inputs{wb}.discharge_Temp{index+1}=[];
133             for i=1:size(vars,2)
134                 if strfind(char(vars(1,i)),'TIN')
135                     flow_variable=strrep(char(vars(1,i)),'TIN','QIN');
136                 elseif strfind(char(vars(1,i)),'CIN')
137                     flow_variable=strrep(char(vars(1,i)),'CIN','QIN');
138                 elseif strfind(char(vars(1,i)),'TTR')
139                     flow_variable=strrep(char(vars(1,i)),'TTR','QTR');
140                 elseif strfind(char(vars(1,i)),'CTR')
141                     flow_variable=strrep(char(vars(1,i)),'CTR','QTR');
142                 else
143                     flow_variable=char(vars(1,i));
144                 end
145                 if ~strcmp(char(vars(1,i)),'MET_WB1') %assume interpolation for MET
                        ↪ data
146                     for ii=1:size(Qtrainingpop{trindex}{wb}.interpolation,2)
147                         if strcmp(char(Qtrainingpop{trindex}{wb}.interpolation(1,ii)),
                                ↪ flow_variable)
148                             break
149                         end
150                     end
151                     if strcmp(char(Qtrainingpop{trindex}{wb}.interpolation(3,ii)),'ON
                            ↪ ')
152                         Inputs{wb}.discharge_Temp{index+1}(:,i)=interp1(Qtrainingpop{
                                ↪ trindex}{wb}.(vars{1,i})(:,1),...
153                             Qtrainingpop{trindex}{wb}.(vars{1,i})(:,vars{2,i}+1),
                                ↪ timesteps);
154                     elseif strcmp(char(Qtrainingpop{trindex}{wb}.interpolation(3,ii))
                            ↪ ,'OFF')
155                         for iii=1:size(timesteps,1)
156                             index2=find(Qtrainingpop{trindex}{wb}.(vars{1,i})(:,1)<=
                                    ↪ timesteps(ii),1,'last');
157                             Inputs{wb}.discharge_Temp{index+1}(iii,i)=Qtrainingpop{
                                    ↪ trindex}{wb}.(vars{1,i})(index2,vars{2,i}+1);
158                         end
159                     end
160                 else
161                     Inputs{wb}.discharge_Temp{index+1}(:,i)=interp1(Qtrainingpop{
                            ↪ trindex}{wb}.(vars{1,i})(:,1),...
162                         Qtrainingpop{trindex}{wb}.(vars{1,i})(:,vars{2,i}+1),timesteps
                            ↪ );
163                 end
164             end
165             T_noNAN=interp1(T{trindex}(~isnan(T{trindex}(:,2)),1),...
166                 T{trindex}(~isnan(T{trindex}(:,2)),2),timesteps);
167             %Fill in Nans at the end
168             a=T_noNAN(~isnan(T_noNAN)); T_noNAN(isnan(T_noNAN))=a(end);
169             turbs=interp1(Qtrainingpop{trindex}{wb}.QOT_BR1_T(:,1),Qtrainingpop{
                    ↪ trindex}{wb}.QOT_BR1_T(:,2),timesteps);
170             spills=interp1(Qtrainingpop{trindex}{wb}.QOT_BR1_S(:,1),Qtrainingpop{
                    ↪ trindex}{wb}.QOT_BR1_S(:,2),timesteps);
```

```
171         flowout=turbs+spills; T_noNAN(flowout==0)=nan;
172
173         %Output data
174         Output{wb}.discharge_Temp{index+1}(:,1)=T_noNAN;
175       end
176     end
177     for i=1:size(Inputs{wb}.discharge_Temp,2)
178       %Convert to cells
179       Inputs_seq{wb}.discharge_Temp{i}=con2seq(Inputs{wb}.discharge_Temp{i}');
180       Output_seq{wb}.discharge_Temp{i}=con2seq(Output{wb}.discharge_Temp{i}');
181     end
182     clearvars i ii iii flow_variable index a T_noNAN turbs spills flowout index2
          ↪ vars timesteps
183
184     %Combine them all into single Input and Output cell arrays
185     Inputs_seq_mul{wb}.discharge_Temp=catsamples(Inputs_seq{wb}.discharge_Temp
          ↪ {:},'pad');
186     Output_seq_mul{wb}.discharge_Temp=catsamples(Output_seq{wb}.discharge_Temp
          ↪ {:},'pad');
187
188     %Train temp model - start with best DO model from before (greatest weight)
189     fprintf(['Training 5 temperature models and picking the best \n'])
190     for i=1:5
191       inputDelays = [0 1 12];
192       feedbackDelays = [1];
193       hiddenNeurons=[10];
194       narx_net{i} = narxnet(inputDelays,feedbackDelays,hiddenNeurons);
195       narx_net{i}.divideFcn = 'dividerand';
196       % The property DIVIDEMODE set to TIMESTEP means that targets are divided
197       % into training, validation and test sets according to timesteps.
198       % For a list of data division modes type: help nntype_data_division_mode
199       narx_net{i}.divideMode = 'time'; % Divide up every value
200       narx_net{i}.divideParam.trainRatio = 70/100;
201       narx_net{i}.divideParam.valRatio = 15/100;
202       narx_net{i}.divideParam.testRatio = 15/100;
203       narx_net{i}.trainParam.min_grad = 1e-10;
204       narx_net{i}.trainFcn = 'trainlm';
205       narx_net{i}.trainParam.showWindow=0;
206       narx_net{i}.trainParam.showCommandLine=1;
207       [Xs,Xi,Ai,Ts]=preparets(narx_net{i},Inputs_seq_mul{wb}.discharge_Temp,{},
            ↪ ...
208         Output_seq_mul{wb}.discharge_Temp);
209       [narx_net{i},~]=train(narx_net{i},Xs,Ts,Xi,Ai,'UseParallel','yes');
210       narx_net_closed{i} = closeloop(narx_net{i});
211       narx_net_closed{i}.trainParam.mu_max=1e14;
212       [Xs,Xi,Ai,Ts]=preparets(narx_net_closed{i},Inputs_seq_mul{wb}.
            ↪ discharge_Temp,{}, ...
213         Output_seq_mul{wb}.discharge_Temp);
214       [narx_net_closed{i},tr{i}]=train(narx_net_closed{i},Xs,Ts,Xi,Ai,'
            ↪ UseParallel','yes');
215     end
216     for i=1:5 tr2(i)=tr{i}.best_perf; end
217     [~,b]=min(tr2); WQ{wb}.Temp_narx.narx_net_closed=narx_net_closed{b};
218     yp1= WQ{wb}.Temp_narx.narx_net_closed(Xs,Xi,Ai);
219     %Calculate bias & standard dev using only predictions at test timepoints
220     bias=cell2mat(yp1(tr{b}.testInd))-cell2mat(Ts(tr{b}.testInd)); bias=nanmean(
          ↪ bias);
221     allerrors=(cell2mat(yp1(tr{b}.testInd))-bias)-cell2mat(Ts(tr{b}.testInd));
222     allerrors=allerrors(~isnan(allerrors));
223     [~,sigmahat] = normfit(allerrors);
224     WQ{wb}.Temp_narx.bias=bias;
225     WQ{wb}.Temp_narx.weights=1;
226     WQ{wb}.Temp_narx.inputDelays=inputDelays;
227     WQ{wb}.Temp_narx.std_dev=sigmahat;
```

```
    WQ{wb}.Temp_narx.Inputs=Inputs{wb}.discharge_Temp;
    WQ{wb}.Temp_narx.Output=Output{wb}.discharge_Temp;
    if isfield(WQ{wb}.Temp_narx,'train_time')
        WQ{wb}.Temp_narx=rmfield(WQ{wb}.Temp_narx,{'train_time'});
    end
    if isfield(WQ{wb}.Temp_narx,'Discharge_temp_no0s')
        WQ{wb}.Temp_narx=rmfield(WQ{wb}.Temp_narx,{'Discharge_temp_no0s'});
    end
    clearvars b Xs Xi Ai Ts tr tr2 yp1 TS bias narx_net_closed narx_net muhat
        ↪ sigmahat
end
clearvars timesteps
```

obj_fcn.m

```
function y=obj_fcn(x_allwb,t,cost_curve_MW,MW_rating,...
    elev_soft_penalty_coeff,ELWS_targets,frequency,Q,ic_elev,...
    turbine_discharge,cache,Optimize_day_by_day)

% Calculates value of generation pattern over time t
%
% Inputs:
% x_allwb - hourly turbine time series (as rows for vectorizing!),
% integers between 0 and no_of_units for all waterbodies
% t  time series of JDAY values
% cost_curve_MW  2 row matrix to create step function, with 1st row
% being hours and 2nd row $/MW-hr values
% MW_rating - the fixed MW level of turbine_discharge_curve (25 MW for
% OHL)
% elev_soft_penalty_coeff - penalty coefficient for soft ending elev soft
% constraint
% ELWS_targets - target elevations for end of time period
% frequency - frequency of predictions (hourly=1/24)
% Q - all other inflows and outflows, interpolation settings,
% storage-elev curve, and tailwater curve (all in meters)
% ic_elev - initial elevation condition (m)
% turbine_discharge - turbine discharge curve at fixed MW level, with
% col 1 in meters and col 2 in cms
% cache - water quality predictions provided by W2 simulations
% Optimize_day_by_day - 1 if optimizing daily, 0 if optimizing all together
% Outputs:
% y  total price in $ of generation pattern

x_allwb=round(x_allwb);

y=zeros(size(x_allwb,1),1);

%Split up rows of x to separate reservoirs
for wb=1:size(MW_rating,2)
    x{wb}=x_allwb(:,wb*(size(t,2)-1)-(size(t,2)-2):wb*(size(t,2)-1));
end
clearvars wb

for wb=1:size(MW_rating,2)

    %Calculate turbine output over 10 days
    %Multiply each turbine output by number of turbines online
    output_MW{wb}=x{wb}*MW_rating{wb}; %MW

    %Calculate total power output
    y_MWh{wb}=sum(output_MW{wb}')';
    %Calculate weighted price output
    y_dollars{wb}=cost_curve(t,output_MW{wb},cost_curve_MW{wb}');
```

```
49
50      %Calculate deviation from ELWS_target and subtract/add penalty
51      if wb==1
52         %Preallocate mainstem_inflows for following wbs
53         mainstem_inflows=cell(1:size(MW_rating,2));
54         for i=1:size(MW_rating,2)
55            mainstem_inflows{i}.t=[];
56            mainstem_inflows{i}.Q=[];
57         end
58         clearvars i
59         [turb_discharges{wb},spill_discharges{wb},HWs{wb},~,~] = ...
60            activeunits_to_discharges(x{wb},t,frequency,...
61            Q{wb},ic_elev{wb},turbine_discharge{wb},ELWS_targets{wb},...
62            [],[],Optimize_day_by_day);
63      else
64         [turb_discharges{wb},spill_discharges{wb},HWs{wb},~,~] = ...
65            activeunits_to_discharges(x{wb},t,frequency,...
66            Q{wb},ic_elev{wb},turbine_discharge{wb},ELWS_targets{wb},...
67            mainstem_inflows{wb}.t,mainstem_inflows{wb}.Q,Optimize_day_by_day);
68      end
69
70      %ELWS end goal
71      if size(ELWS_targets{wb}(:,1),1)==1
72         ELWS_goal{wb}=ELWS_targets{wb}(:,2);
73      else
74         ELWS_goal{wb}=interp1(ELWS_targets{wb}(:,1),ELWS_targets{wb}(:,2),t(end));
75      end
76      ELWS_error{wb}=HWs{wb}(:,end)-ELWS_goal{wb};
77      ELWS_deduction{wb}=(ELWS_error{wb}.^2)*elev_soft_penalty_coeff(wb);
78
79      y=y+y_dollars{wb}-ELWS_deduction{wb};
80
81      %If we haven't reached the last reservoir, update mainstem_inflows
82      if wb~=size(ic_elev,2)
83         mainstem_inflows{wb+1}.t=t;
84         mainstem_inflows{wb+1}.Q=bsxfun(@plus,turb_discharges{wb},spill_discharges
               ↪ {wb});
85      end
86
87   end
```

obj_fcn_do.m

```
1   function y=obj_fcn_do(x_allwb,t,frequency,Q,ic_elev,...
2      turbine_discharge,WQ,xprev,ELWS_targets,level,waterbody,cache,
           ↪ Optimize_day_by_day)
3
4   % Objective function to minimize DO constraint violation
5   %
6   % Inputs:
7   % x_allwb - hourly turbine time series (as rows for vectorizing!),
8   % integers between 0 and no_of_units for all waterbodies
9   % t  time series of JDAY values
10  % frequency - frequency of predictions (hourly=1/24)
11  % Q - all other inflows and outflows, interpolation settings,
12  % storage-elev curve, and tailwater curve (all in meters)
13  % ic_elev - initial elevation condition (m)
14  % turbine_discharge - turbine discharge curve at fixed MW level, with
15  % col 1 in meters and col 2 in cms
16  % WQ - structure containing water quality constraints and NARX models
17  % DO_narx - structure containing everything needed to make DO discharge
18  % predictions, including:
19  % turb_colum - column in exogenous variables with turb flows
```

```
% spill_column - column in exogenous variables with spill flows
% times - JDAY values used in training (not used)
% inputDelays - delays for exogenous inputs
% feedbackDelays - delays for prediction feedbacks
% input_variables - 2 row cell containing variable names in first
% row and column number in second. For example, 'MET_WB1'
% contains multiple columns of data but only some may be used
% for NARX predictions
% bias - bias for each trained neural network
% weights - weights for each trained neural network (sum to 1)
% narx_net_closed - neural networks
% DO_limit - lower and upper DO limits (NaN means it doesn't exist)
% DO_slack - relaxation from DO_limit (either upper or lower -
% doesn't make sense to have both)
% Temp_narx - structure containing everything needed to make temp discharge
% predictions, including:
% turb_colum - column in exogenous variables with turb flows
% spill_column - column in exogenous variables with spill flows
% times - JDAY values used in training (not used)
% inputDelays - delays for exogenous inputs
% feedbackDelays - delays for prediction feedbacks
% input_variables - 2 row cell containing variable names in first
% row and column number in second. For example, 'MET_WB1'
% contains multiple columns of data but only some may be used
% for NARX predictions
% bias - bias for each trained neural network
% weights - weights for each trained neural network (sum to 1)
% narx_net_closed - neural networks
% Temp_limit - lower and upper temp limits (NaN means it doesn't exist)
% Temp_slack - relaxation from Temp_limit (either upper or lower -
% doesn't make sense to have both)
% ELWS_targets - 2 column matrix with JDAY in col1 and elevation target
% in col2
% level - 'upper' or 'lower'
% waterbody - which waterbody we're checking the discharge DO for
% cache - water quality predictions provided by W2 simulations
% Optimize_day_by_day - 1 if optimizing daily, 0 if optimizing all together
% Outputs:
% y  DO constraint violation for each scenario in x

%If using the cache, get list of cache indices here
[~,~,tib]=intersect(t,cache.t);
if Optimize_day_by_day==0 & size(ic_elev,2)==1 & ~isempty(cache.x)
   [ia,ib]=ismember(x_allwb,cache.x,'rows');
else
   index=find(cache.t==t(1)); %last index for previous operations
   if index==1 %first day
      [ia,ib]=ismember(x_allwb,cache.x(:,index:index+23),'rows');
   else
      [ia,ib]=ismember([repmat(xprev{1}(size(xprev{1},2)-tib(1)+2:end),...
         size(x_allwb,1),1) x_allwb],cache.x(:,1:index+23),'rows'); %fix later
            ↪ to solve multi waterbody problems
   end
end
ia=find(ia==1); ib=ib(ib~=0);
if ~isempty(ia) fprintf(['Cached points here: ', num2str(ib'), '\n']); end

%Split up rows of x to separate reservoirs
for wb=1:waterbody
   x{wb}=x_allwb(:,wb*(size(t,2)-1)-(size(t,2)-2):wb*(size(t,2)-1));
end
clearvars wb

%Calculate headwater elevs for constraints
```

```
for wb=1:waterbody
   %Calculate headwater elevs for constraints
   if wb==1
      mainstem_inflows{wb}.t=[];
      mainstem_inflows{wb}.Q=[];
      %Check to see if any cached rows can be skipped by elev calcs
      if ~isempty(ia)
         [HWcalcrows,b]=setdiff(1:size(x_allwb,1),ia);
         x_HWcalcrows=x{wb}(HWcalcrows,:);
         [turb_discharges{wb}(b,:),spill_discharges{wb}(b,:),HWs{wb}(b,:),~,~] =
             ↪  ...
            activeunits_to_discharges(x_HWcalcrows,t,...
            frequency,Q{wb},ic_elev{wb},turbine_discharge{wb},...
            ELWS_targets{wb},[],[],Optimize_day_by_day);
         HWs{wb}(setdiff(1:size(x_allwb,1),b),:)=cache.HWs(ib,tib);
      else
      [turb_discharges{wb},spill_discharges{wb},HWs{wb},~,~] = ...
         activeunits_to_discharges(x{wb},t,frequency,...
         Q{wb},ic_elev{wb},turbine_discharge{wb},ELWS_targets{wb},...
         [],[],Optimize_day_by_day);
      end
   else
      [turb_discharges{wb},spill_discharges{wb},HWs{wb},~,~] = ...
         activeunits_to_discharges(x{wb},t,frequency,...
         Q{wb},ic_elev{wb},turbine_discharge{wb},ELWS_targets{wb},...
         mainstem_inflows{wb}.t,mainstem_inflows{wb}.Q,Optimize_day_by_day);
   end
   %If we haven't reached the last reservoir, update mainstem_inflows.Q (include
       ↪  both turbine + spill incoming!) and mainstem_inflows.t
   if wb~=size(ic_elev,2)
      mainstem_inflows{wb+1}.Q=...
         bsxfun(@plus,turb_discharges{wb},spill_discharges{wb});
      mainstem_inflows{wb+1}.t=t;
   end
end

for wb=1:waterbody

   if wb~=1
      mainstem_inflows_temp{wb}.t=mainstem_inflows{wb}.t;
      mainstem_inflows_temp{wb}.Q=mainstem_inflows{wb}.Q;
      mainstem_inflows_temp{wb}.T=mainstem_inflows{wb}.T;
      mainstem_inflows_temp{wb}.DO=mainstem_inflows{wb}.DO;
      %Remove Nan values and interpolate for T and DO
      for i=1:size(x{wb},1)
         extrap_index=~isnan(mainstem_inflows_temp{wb}.T(i,:));
         [~,c]=find(extrap_index==1); extrap_index=c(end);
         mainstem_inflows_temp{wb}.T(i,:)=...
            interp1(mainstem_inflows_temp{wb}.t(1,~isnan(mainstem_inflows_temp{
                ↪ wb}.T(i,:))),...
            mainstem_inflows_temp{wb}.T(i,~isnan(mainstem_inflows_temp{wb}.T(i
                ↪ ,:))),...
            mainstem_inflows_temp{wb}.t,'linear',...
            mainstem_inflows_temp{wb}.T(i,extrap_index));
         mainstem_inflows_temp{wb}.DO(i,:)=...
            interp1(mainstem_inflows_temp{wb}.t(1,~isnan(mainstem_inflows_temp{
                ↪ wb}.DO(i,:))),...
            mainstem_inflows_temp{wb}.DO(i,~isnan(mainstem_inflows_temp{wb}.DO(i
                ↪ ,:))),...
            mainstem_inflows_temp{wb}.t,'linear',...
            mainstem_inflows_temp{wb}.DO(i,extrap_index));
         clearvars extrap_index c
      end
      clearvars i
```

```
141     end
142
143     %Discharge Temp estimation, to update incoming mainstem temp for next
            ↪ waterbody discharge DO estimation
144     Temp_narx=WQ{wb}.Temp_narx;
145     if wb==1 & waterbody~=1 %don't need to search cache for incoming temp,
            ↪ because cache is only set up for 1 wb problems
146        Temp_pred{wb}=...
147           narx_predictions(Temp_narx,frequency,t,Q{wb},x{wb},...
148           turb_discharges{wb},spill_discharges{wb},[],...
149           Q{wb}.TWO,'temp',Optimize_day_by_day);
150     elseif wb~=1 & wb~=waterbody
151        Temp_pred{wb}=narx_predictions(Temp_narx,frequency,t,Q{wb},x{wb},...
152           turb_discharges{wb},spill_discharges{wb},...
153           mainstem_inflows_temp{wb},Q{wb}.TWO,'temp',Optimize_day_by_day);
154     end
155     %If we haven't reached the last reservoir, update mainstem_inflows.T
156     if wb~=waterbody
157        mainstem_inflows{wb+1}.T(1:size(x{wb},1),1)=...
158           interp1(Q{wb}.TWO(:,1),Q{wb}.TWO(:,2),t(1));
159        mainstem_inflows{wb+1}.T(:,2:size(Temp_pred{wb},2)+1)=...
160           Temp_pred{wb};
161     end
162
163     %Now move on to DO....
164     DO_narx=WQ{wb}.DO_narx; DO_limit=WQ{wb}.DO_limit;
165     if wb==1
166        if ~isempty(ia)
167           [DOcalcrows,b]=setdiff(1:size(x_allwb,1),ia);
168           x_DOcalcrows=x{wb}(DOcalcrows,:);
169           DO_pred{wb}(b,:)=...
170              narx_predictions(DO_narx,...
171              frequency,t,Q{wb},x_DOcalcrows,...
172              turb_discharges{wb}(DOcalcrows,:),...
173              spill_discharges{wb}(DOcalcrows,:),[],...
174              Q{wb}.CWO,'do',Optimize_day_by_day);
175           DO_pred{wb}(ia,:)=cache.DO(ib,tib(1:end-1));
176           clearvars DOcalcrows x_DOcalcrows b
177        else
178           DO_pred{wb}=narx_predictions(DO_narx,frequency,t,Q{wb},x{wb},...
179              turb_discharges{wb},spill_discharges{wb},[],...
180              Q{wb}.CWO,'do',Optimize_day_by_day);
181        end
182     else
183        DO_pred{wb}=narx_predictions(DO_narx,frequency,t,Q{wb},x{wb},...
184           turb_discharges{wb},spill_discharges{wb},...
185           mainstem_inflows_temp{wb},Q{wb}.CWO,'do',Optimize_day_by_day);
186     end
187     %If we haven't reached the last reservoir, update mainstem_inflows.DO
188     if wb~=waterbody
189        mainstem_inflows{wb+1}.DO(1:size(x{wb},1),1)=...
190           interp1(Q{wb}.CWO(:,1),Q{wb}.CWO(:,2),t(1));
191        mainstem_inflows{wb+1}.DO(:,2:size(DO_pred{wb},2)+1)=...
192           DO_pred{wb};
193     else
194        non_nan_count=sum(~isnan(DO_pred{wb}),2);
195        if strcmp(level,'lower')
196           %DO violations - lower
197           if isnan(DO_limit(1))
198              DO_violations=zeros(size(DO_pred{wb},1),1);
199           else
200              DO_violations=sum(-min(0,DO_pred{wb}-DO_limit(1)),2)./non_nan_count;
201           end
202        elseif strcmp(level,'upper')
```

```
         %DO violations - upper
         if isnan(DO_limit(2))
            DO_violations=zeros(size(DO_pred{wb},1),1);
         else
            DO_violations=sum(max(0,DO_pred{wb}-DO_limit(2)),2)./non_nan_count;
         end
      end

      y=max(DO_violations,[],2);
   end
end
```

obj_fcn_elev.m

```
function y=obj_fcn_elev(x_allwb,t,frequency,Q,ic_elev,...
   turbine_discharge,ELWS_limit,xprev,ELWS_targets,level,waterbody,cache,...
   Optimize_day_by_day)

% Objective function to minimize elevation constraint violation
%
% Inputs:
% x_allwb - hourly turbine time series (as rows for vectorizing!),
% integers between 0 and no_of_units for all waterbodies
% t  time series of JDAY values
% frequency - frequency of predictions (hourly=1/24)
% Q - all other inflows and outflows, interpolation settings,
% storage-elev curve, and tailwater curve (all in meters)
% ic_elev - initial elevation condition (m)
% turbine_discharge - turbine discharge curve at fixed MW level, with
% col 1 in meters and col 2 in cms
% ELWS_limit - min and max elevation limits for constraints, in meters
% ELWS_targets - 2 column matrix with JDAY in col1 and elevation target
% in col2
% level - 'upper' or 'lower'
% waterbody - which waterbody we're checking elevation for
% cache - water quality predictions provided by W2 simulations
% Optimize_day_by_day - 1 if optimizing daily, 0 if optimizing all together
% Outputs:
% y  elevation constraint violation for each scenario in x

%If using the cache, get list of cache indices here
[~,~,tib]=intersect(t,cache.t);
if Optimize_day_by_day==0 & size(ic_elev,2)==1 & ~isempty(cache.x)
   [ia,ib]=ismember(x_allwb,cache.x,'rows');
else
   index=find(cache.t==t(1)); %last index for previous operations
   if index==1 %first day
      [ia,ib]=ismember(x_allwb,cache.x(:,index:index+23),'rows');
   else
      [ia,ib]=ismember([repmat(xprev{1}(size(xprev{1},2)-tib(1)+2:end),...
         size(x_allwb,1),1) x_allwb],cache.x(:,1:index+23),'rows'); %fix later
            ↪ to solve multi waterbody problems
   end
end
ia=find(ia==1); ib=ib(ib~=0);
if ~isempty(ia) fprintf(['Cached points here: ', num2str(ib'), '\n']); end

%Split up rows of x to separate reservoirs
for wb=1:waterbody
   x{wb}=x_allwb(:,wb*(size(t,2)-1)-(size(t,2)-2):wb*(size(t,2)-1));
end
clearvars wb

```

```
for wb=1:waterbody
   %Calculate headwater elevs for constraints
   if wb==1
      mainstem_inflows{wb}.t=[];
      mainstem_inflows{wb}.Q=[];
       %Check to see if any cached rows can be skipped by elev calcs
      if ~isempty(ia)
         [HWcalcrows,b]=setdiff(1:size(x_allwb,1),ia);
         x_HWcalcrows=x{wb}(HWcalcrows,:);
         [turb_discharges{wb}(b,:),spill_discharges{wb}(b,:),HWs{wb}(b,:),~,~] =
             ↪  ...
            activeunits_to_discharges(x_HWcalcrows,t,...
            frequency,Q{wb},ic_elev{wb},turbine_discharge{wb},...
            ELWS_targets{wb},[],[],Optimize_day_by_day);
         HWs{wb}(setdiff(1:size(x_allwb,1),b),:)=cache.HWs(ib,tib);
      else
         [turb_discharges{wb},spill_discharges{wb},HWs{wb},~,~] = ...
            activeunits_to_discharges(x{wb},t,frequency,...
            Q{wb},ic_elev{wb},turbine_discharge{wb},ELWS_targets{wb},...
            [],[],Optimize_day_by_day);
      end
   else
      [turb_discharges{wb},spill_discharges{wb},HWs{wb},~,~] = ...
         activeunits_to_discharges(x{wb},t,frequency,...
         Q{wb},ic_elev{wb},turbine_discharge{wb},ELWS_targets{wb},...
         mainstem_inflows{wb}.t,mainstem_inflows{wb}.Q,Optimize_day_by_day);
   end
   %If we haven't reached the last reservoir, update mainstem_inflows.Q (include
       ↪  both turbine + spill incoming!) and mainstem_inflows.t
   if wb~=size(ic_elev,2)
      mainstem_inflows{wb+1}.Q=...
         bsxfun(@plus,turb_discharges{wb},spill_discharges{wb});
      mainstem_inflows{wb+1}.t=t;
   end
end

%Inequality constraints:
if strcmp(level,'lower')
   %Elevation violations - lower
   if isnan(ELWS_limit(1))
      deductions=zeros(size(HWs{waterbody}(:,1:end)));
   else
      deductions=-min(0,HWs{waterbody}(:,1:end)-ELWS_limit(1));
   end
elseif strcmp(level,'upper')
   %Elevation violations - upper
   if isnan(ELWS_limit(2))
      deductions=zeros(size(HWs{waterbody}(:,1:end)));
   else
      deductions=max(0,HWs{waterbody}(:,1:end)-ELWS_limit(2));
   end
end

y=max(deductions,[],2);
```

obj_fcn_temp.m

```
function y=obj_fcn_temp(x_allwb,t,frequency,Q,ic_elev,...
   turbine_discharge,WQ,xprev,ELWS_targets,level,waterbody,cache,
       ↪ Optimize_day_by_day)

% Objective function to minimize temp constraint violation
%
```

```
% Inputs:
% x - hourly turbine time series (as rows for vectorizing!), integers
% between 0 and no_of_units
% t  time series of JDAY values
% frequency - frequency of predictions (hourly=1/24)
% Q - all other inflows and outflows, interpolation settings,
% storage-elev curve, and tailwater curve (all in meters)
% ic_elev - initial elevation condition (m)
% turbine_discharge - turbine discharge curve at fixed MW level, with
% col 1 in meters and col 2 in cms
% WQ - structure containing water quality constraints and NARX models
% DO_narx - structure containing everything needed to make DO discharge
% predictions, including:
% turb_colum - column in exogenous variables with turb flows
% spill_column - column in exogenous variables with spill flows
% times - JDAY values used in training (not used)
% inputDelays - delays for exogenous inputs
% feedbackDelays - delays for prediction feedbacks
% input_variables - 2 row cell containing variable names in first
% row and column number in second. For example, 'MET_WB1'
% contains multiple columns of data but only some may be used
% for NARX predictions
% bias - bias for each trained neural network
% weights - weights for each trained neural network (sum to 1)
% narx_net_closed - neural networks
% DO_limit - lower and upper DO limits (NaN means it doesn't exist)
% DO_slack - relaxation from DO_limit (either upper or lower -
% doesn't make sense to have both)
% Temp_narx - structure containing everything needed to make temp discharge
% predictions, including:
% turb_colum - column in exogenous variables with turb flows
% spill_column - column in exogenous variables with spill flows
% times - JDAY values used in training (not used)
% inputDelays - delays for exogenous inputs
% feedbackDelays - delays for prediction feedbacks
% input_variables - 2 row cell containing variable names in first
% row and column number in second. For example, 'MET_WB1'
% contains multiple columns of data but only some may be used
% for NARX predictions
% bias - bias for each trained neural network
% weights - weights for each trained neural network (sum to 1)
% narx_net_closed - neural networks
% Temp_limit - lower and upper temp limits (NaN means it doesn't exist)
% Temp_slack - relaxation from Temp_limit (either upper or lower -
% doesn't make sense to have both)
% ELWS_targets - 2 column matrix with JDAY in col1 and elevation target
% in col2
% level - 'upper' or 'lower'
% waterbody - which waterbody we're checking the discharge temp for
% cache - water quality predictions provided by W2 simulations
% Optimize_day_by_day - 1 if optimizing daily, 0 if optimizing all together
% Outputs:
% y  temp constraint violation for each scenario in x

%If using the cache, get list of cache indices here
[~,~,tib]=intersect(t,cache.t);
if Optimize_day_by_day==0 & size(ic_elev,2)==1 & ~isempty(cache.x)
   [ia,ib]=ismember(x_allwb,cache.x,'rows');
else
   index=find(cache.t==t(1)); %last index for previous operations
   if index==1 %first day
      [ia,ib]=ismember(x_allwb,cache.x(:,index:index+23),'rows');
   else
      [ia,ib]=ismember([repmat(xprev{1}(size(xprev{1},2)-tib(1)+2:end),...
```

```
        size(x_allwb,1),1) x_allwb],cache.x(:,1:index+23),'rows'); %fix later
            ↪ to solve multi waterbody problems
   end
end
ia=find(ia==1); ib=ib(ib~=0);
if ~isempty(ia) fprintf(['Cached points here: ', num2str(ib'), '\n']); end

%Split up rows of x to separate reservoirs
for wb=1:waterbody
   x{wb}=x_allwb(:,wb*(size(t,2)-1)-(size(t,2)-2):wb*(size(t,2)-1));
end
clearvars wb

%Calculate headwater elevs for constraints
for wb=1:waterbody
   %Calculate headwater elevs for constraints
   if wb==1
      mainstem_inflows{wb}.t=[];
      mainstem_inflows{wb}.Q=[];
      %Check to see if any cached rows can be skipped by elev calcs
      if ~isempty(ia)
         [HWcalcrows,b]=setdiff(1:size(x_allwb,1),ia);
         x_HWcalcrows=x{wb}(HWcalcrows,:);
         [turb_discharges{wb}(b,:),spill_discharges{wb}(b,:),HWs{wb}(b,:),~,~] =
             ↪  ...
            activeunits_to_discharges(x_HWcalcrows,t,...
            frequency,Q{wb},ic_elev{wb},turbine_discharge{wb},...
            ELWS_targets{wb},[],[],Optimize_day_by_day);
         HWs{wb}(setdiff(1:size(x_allwb,1),b),:)=cache.HWs(ib,tib);
      else
         [turb_discharges{wb},spill_discharges{wb},HWs{wb},~,~] = ...
            activeunits_to_discharges(x{wb},t,frequency,...
            Q{wb},ic_elev{wb},turbine_discharge{wb},ELWS_targets{wb},...
            [],[],Optimize_day_by_day);
      end
   else
      [turb_discharges{wb},spill_discharges{wb},HWs{wb},~,~] = ...
         activeunits_to_discharges(x{wb},t,frequency,...
         Q{wb},ic_elev{wb},turbine_discharge{wb},ELWS_targets{wb},...
         mainstem_inflows{wb}.t,mainstem_inflows{wb}.Q,Optimize_day_by_day);
   end
   %If we haven't reached the last reservoir, update mainstem_inflows.Q (include
       ↪  both turbine + spill incoming!) and mainstem_inflows.t
   if wb~=size(ic_elev,2)
      mainstem_inflows{wb+1}.Q=...
         bsxfun(@plus,turb_discharges{wb},spill_discharges{wb});
      mainstem_inflows{wb+1}.t=t;
   end
end


for wb=1:waterbody

   if wb~=1
      mainstem_inflows_temp{wb}.t=mainstem_inflows{wb}.t;
      mainstem_inflows_temp{wb}.Q=mainstem_inflows{wb}.Q;
      mainstem_inflows_temp{wb}.T=mainstem_inflows{wb}.T;
      %Remove Nan values and interpolate for T
      for i=1:size(x{wb},1)
         extrap_index=~isnan(mainstem_inflows_temp{wb}.T(i,:));
         [~,c]=find(extrap_index==1); extrap_index=c(end);
         mainstem_inflows_temp{wb}.T(i,:)=...
            interp1(mainstem_inflows_temp{wb}.t(1,~isnan(mainstem_inflows_temp{
                ↪ wb}.T(i,:))),...
```

```
            mainstem_inflows_temp{wb}.T(i,~isnan(mainstem_inflows_temp{wb}.T(i
                ↪ ,:))),...
            mainstem_inflows_temp{wb}.t,'linear',...
            mainstem_inflows_temp{wb}.T(i,extrap_index));
         clearvars extrap_index c
      end
      clearvars i
   end

   %Discharge Temp estimation
   Temp_narx=WQ{wb}.Temp_narx; Temp_limit=WQ{wb}.Temp_limit;
   if wb==1
      if ~isempty(ia)
         [Tcalcrows,b]=setdiff(1:size(x_allwb,1),ia);
         x_Tcalcrows=x{wb}(Tcalcrows,:);
         Temp_pred{wb}(b,:)=...
            narx_predictions(Temp_narx,...
            frequency,t,Q{wb},x_Tcalcrows,...
            turb_discharges{wb}(Tcalcrows,:),...
            spill_discharges{wb}(Tcalcrows,:),[],...
            Q{wb}.TWO,'temp',Optimize_day_by_day);
         Temp_pred{wb}(ia,:)=cache.T(ib,tib(1:end-1));
         clearvars Tcalcrows x_Tcalcrows b
      else
         Temp_pred{wb}=narx_predictions(Temp_narx,frequency,t,Q{wb},x{wb},...
            turb_discharges{wb},spill_discharges{wb},[],...
            Q{wb}.TWO,'temp',Optimize_day_by_day);
      end
   else
      Temp_pred{wb}=narx_predictions(Temp_narx,frequency,t,Q{wb},x{wb},...
         turb_discharges{wb},spill_discharges{wb},...
         mainstem_inflows_temp{wb},Q{wb}.TWO,'temp',Optimize_day_by_day);
   end
   %If we haven't reached the last reservoir, update mainstem_inflows.T
   if wb~=waterbody
      mainstem_inflows{wb+1}.T(1:size(x{wb},1),1)=...
         interp1(Q{wb}.TWO(:,1),Q{wb}.TWO(:,2),t(1));
      mainstem_inflows{wb+1}.T(:,2:size(Temp_pred{wb},2)+1)=...
         Temp_pred{wb};
   else
      non_nan_count=sum(~isnan(Temp_pred{wb}),2);
      if strcmp(level,'lower')
         %Temp violations - lower
         if isnan(Temp_limit(1))
            Temp_violations=zeros(size(Temp_pred{wb},1),1);
         else
            Temp_violations=sum(-min(0,Temp_pred{wb}-Temp_limit(1)),2)./
                ↪ non_nan_count;
         end
      elseif strcmp(level,'upper')
         %Temp violations - upper
         if isnan(Temp_limit(2))
            Temp_violations=zeros(size(Temp_pred{wb},1),1);
         else
            Temp_violations=sum(max(0,Temp_pred{wb}-Temp_limit(2)),2)./
                ↪ non_nan_count;
         end
      end

      y=max(Temp_violations,[],2);
   end
end
```

penalty_fcn.m

```
function [c_all,ceq]=penalty_fcn(x_allwb,t,frequency,Q,ic_elev,...
   turbine_discharge,ELWS_limit,max_hrly_unit_change,...
   WQ,zero_gen_limit,xprev,ELWS_targets,tolerance,cache,Optimize_day_by_day)

% Calculates penalty violations, starting with the least expensive
% computations and continuing on to the more expensive computations for
% runs that are found to be feasible thus far
%
% Inputs:
% x_allwb - hourly turbine time series (as rows for vectorizing!),
% integers between 0 and no_of_units for all waterbodies
% t  time series of JDAY values
% frequency - frequency of predictions (hourly=1/24)
% Q - all other inflows and outflows, interpolation settings,
% storage-elev curve, and tailwater curve
% ic_elev - initial condition (meters)
% turbine_discharge - turbine discharge curve at fixed MW level, with
% col 1 in meters and col 2 in cms
% ELWS_limit - min and max elevation limits for constraints, in meters
% max_hrly_unit_change - max number of units that can be changed per hour
% (1 for OHL)
% WQ - structure containing water quality constraints and NARX models
% DO_narx - structure containing everything needed to make DO discharge
% predictions, including:
% turb_colum - column in exogenous variables with turb flows
% spill_column - column in exogenous variables with spill flows
% times - JDAY values used in training (not used)
% inputDelays - delays for exogenous inputs
% feedbackDelays - delays for prediction feedbacks
% input_variables - 2 row cell containing variable names in first
% row and column number in second. For example, 'MET_WB1'
% contains multiple columns of data but only some may be used
% for NARX predictions
% bias - bias for each trained neural network
% weights - weights for each trained neural network (sum to 1)
% narx_net_closed - neural networks
% DO_limit - lower and upper DO limits (NaN means it doesn't exist)
% DO_slack - relaxation from DO_limit (either upper or lower -
% doesn't make sense to have both)
% Temp_narx - structure containing everything needed to make temp discharge
% predictions, including:
% turb_colum - column in exogenous variables with turb flows
% spill_column - column in exogenous variables with spill flows
% times - JDAY values used in training (not used)
% inputDelays - delays for exogenous inputs
% feedbackDelays - delays for prediction feedbacks
% input_variables - 2 row cell containing variable names in first
% row and column number in second. For example, 'MET_WB1'
% contains multiple columns of data but only some may be used
% for NARX predictions
% bias - bias for each trained neural network
% weights - weights for each trained neural network (sum to 1)
% narx_net_closed - neural networks
% Temp_limit - lower and upper temp limits (NaN means it doesn't exist)
% Temp_slack - relaxation from Temp_limit (either upper or lower -
% doesn't make sense to have both)
% zero_gen_limit - Zero generation hourly limit (can't go longer than
% this with no turb flow)
% xprev - vector of previous active turbine levels
% ELWS_targets - 2 column matrix with JDAY in col1 and elevation target
% in col2
% tolerance - penalty tolerance
```

```
% cache - water quality predictions provided by W2 simulations
% Optimize_day_by_day - 1 if optimizing daily, 0 if optimizing all together
% Outputs:
% c_all  inequality constraint output (n/a, so 0)
% ceq - equality constraint output (=0 for feasible solution)

%Name global variables to be used for function counts
global funccount_cache_global funccount_tot_global
funccount_tot_global=funccount_tot_global+size(x_allwb,1);

x_allwb=round(x_allwb);

%Equality constraint
ceq=[];

%Preallocate memory
x{1,size(ic_elev,2)}=[];
xall{1,size(ic_elev,2)}=[];
turb_discharges{1,size(ic_elev,2)}=[];
HWs{1,size(ic_elev,2)}=[];
c_all=zeros(size(x{1},1),size(ic_elev,2)*(3+(1+size(x{1},2))*2+2+2));

%If using the cache, get list of cache indices here
if ~isempty(cache)
   [~,~,tib]=intersect(t,cache.t);
   if Optimize_day_by_day==0 & size(ic_elev,2)==1 & ~isempty(cache.x)
      [ia,ib]=ismember(x_allwb,cache.x,'rows');
   else
      index=find(cache.t==t(1)); %last index for previous operations
      if index==1 %first day
         [ia,ib]=ismember(x_allwb,cache.x(:,index:index+23),'rows');
      else
         [ia,ib]=ismember([repmat(xprev{1}(size(xprev{1},2)-tib(1)+2:end),...
            size(x_allwb,1),1) x_allwb],cache.x(:,1:index+23),'rows'); %fix
                ↪ later to solve multi waterbody problems
      end
   end
   ia=find(ia==1); ib=ib(ib~=0);
   funccount_cache_global=funccount_cache_global+size(ia,1);
   if ~isempty(ia) fprintf(['Cached points here: ', num2str(ib'), '\n']); end
else
   ia=[]; ib=[];
end

zeroRows_empty=0;
zeroRows0=[1:size(x_allwb,1)]';

for wb=1:size(ic_elev,2)
   %Split up rows of x to separate reservoirs
   x{wb}=x_allwb(:,wb*(size(t,2)-1)-(size(t,2)-2):wb*(size(t,2)-1));
   %Preallocate c, with columns representing: (1) change in active unit
       ↪ violations, (2) zero gen hourly limit, (3) oscillations constraint,
       ↪ (4:28) ELWS lower violations, (29:53) ELWS upper violations, (54:55)
       ↪ mean lower and upper DO violations, (56:57) mean temp lower and upper
       ↪ violations
   c{wb}=zeros(size(x{1},1),3+(1+size(x{1},2))*2+2+2);
end
clearvars wb

for wb=1:size(ic_elev,2)

   c{wb}(setdiff([1:size(x{wb},1)],zeroRows0),:)=1;

   %Check if all entries in x are infeasible due to previous reservoirs, and if
```

```
        ↪ so set the rest of c==1 and go to end
122     if zeroRows_empty==1
123        c{wb}(:)=1;
124     else
125
126        %Break up WQ structure into separate variables
127        DO_narx=WQ{wb}.DO_narx; DO_limit=WQ{wb}.DO_limit; DO_slack=WQ{wb}.DO_slack
               ↪ ;
128        Temp_narx=WQ{wb}.Temp_narx; Temp_limit=WQ{wb}.Temp_limit; Temp_slack=WQ{wb
               ↪ }.Temp_slack;
129
130        %Stitch together xprev & x to check for feasibility wrt active unit viols,
               ↪  zero generation hrly limit, and oscillations
131        xall{wb}=[repmat(xprev{wb},size(x{wb},1),1) x{wb}];
132
133        %Change in active unit violations
134        if isempty(max_hrly_unit_change{wb})
135           delta_sum=zeros(size(zeroRows0,1),1);
136        else
137           delta=abs(round(xall{wb}(zeroRows0,2:end))-...
138              round(xall{wb}(zeroRows0,1:end-1)));
139           index=find(delta<=max_hrly_unit_change{wb});
140           delta(index)=0;
141           delta_sum=sum(delta')';
142        end
143
144        %Zero generation hourly limit - can't go longer with no turb flow
145        if isempty(zero_gen_limit{wb})
146           zero_gen_viols_sum=zeros(size(zeroRows0,1),1);
147        else
148           zero_gen_viols=zeros(size(zeroRows0,1),size(xall{wb},2)-...
149              zero_gen_limit{wb}-1);
150           x_trans=xall{wb}(zeroRows0,:)';
151           for i=1:size(x_trans,1)-zero_gen_limit{wb}
152              a=sum(x_trans(i:i+zero_gen_limit{wb},:))';
153              zero_gen_viols(:,i)=(a==0);
154           end
155           clearvars i
156           zero_gen_viols_sum=sum(zero_gen_viols')';
157        end
158
159        %Oscillations constraint - violates whenever the number of turbines
               ↪ increases and then decreases within 3 hours, or vice versa
160        osc_violations=zeros(size(zeroRows0,1),size(xall{wb},2)-2);
161        xall_osc=xall{wb}(zeroRows0,:);
162        for ii=1:size(xall_osc,1) %loop through each member of population
163           for i=1:size(xall_osc,2)-2; %loop forward through time
164              if xall_osc(ii,i+1)>xall_osc(ii,i) & ...
165                    xall_osc(ii,i+2)<xall_osc(ii,i+1)
166                 osc_violations(ii,i)=1;
167              elseif xall_osc(ii,i+1)<xall_osc(ii,i) & ...
168                    xall_osc(ii,i+2)>xall_osc(ii,i+1)
169                 osc_violations(ii,i)=1;
170              elseif i~=1
171                 if xall_osc(ii,i)==xall_osc(ii,i+1) %need 3 hrs btwn ramping up
                       ↪ and down
172                    if xall_osc(ii,i-1)<xall_osc(ii,i) & ...
173                          xall_osc(ii,i+1)>xall_osc(ii,i+2) %ramping up & back
                             ↪ down too quickly
174                       osc_violations(ii,i)=1;
175                    elseif xall_osc(ii,i-1)>xall_osc(ii,i) & ...
176                          xall_osc(ii,i+1)<xall_osc(ii,i+2) %ramping down & back
                             ↪ up too quickly
177                       osc_violations(ii,i)=1;
```

```
                end
            end
        end
    end
end
clearvars i ii xall_osc
osc_violations_sum=sum(osc_violations')';

%Compile least expensive constraints
c{wb}(zeroRows0,1:3)=...
    [delta_sum zero_gen_viols_sum osc_violations_sum];

clearvars zeroRows1 zeroRows2 zeroRows3 zeroRows4 x_zeroRows1 x_zeroRows2
    ↪ x_zeroRows3 x_zeroRows4
x_zeroRows1=[];
x_zeroRows2=[];
x_zeroRows3=[];
x_zeroRows4=[];
%Only compute expensive constraints if all others pass
zeroRows1=find(all(c{wb}<=tolerance,2));
x_zeroRows1=x{wb}(zeroRows1,:);
if isempty(x_zeroRows1)
    c{wb}(:,4:end)=1;
    zeroRows_empty=1;
end

if zeroRows_empty~=1

    %Calculate headwater elevs for constraints
    if wb==1
        %Preallocate mainstem_inflows for following wbs
        mainstem_inflows=cell(1:size(ic_elev,2));
        for i=1:size(ic_elev,2)
            mainstem_inflows{i}.t=[];
            mainstem_inflows{i}.Q=[];
            mainstem_inflows{i}.T=[];
            mainstem_inflows{i}.DO=[];
        end
        clearvars i
        %Check to see if any cached rows can be skipped by elev calcs
        if ~isempty(ia)
            [HWcalcrows,b]=setdiff(zeroRows1,ia);
            x_HWcalcrows=x{wb}(HWcalcrows,:);
            [turb_discharges{wb}(b,:),spill_discharges{wb}(b,:),HWs{wb}(b,:)
                ↪ ,~,~] = ...
                activeunits_to_discharges(x_HWcalcrows,t,...
                frequency,Q{wb},ic_elev{wb},turbine_discharge{wb},...
                ELWS_targets{wb},[],[],Optimize_day_by_day);
            [~,bb]=ismember(x_zeroRows1,cache.x,'rows'); bb=nonzeros(bb);
            HWs{wb}(setdiff(1:size(zeroRows1,1),b),:)=...
                cache.HWs(bb,tib);
        else
            [turb_discharges{wb},spill_discharges{wb},HWs{wb},~,~] = ...
                activeunits_to_discharges(x_zeroRows1,t,...
                frequency,Q{wb},ic_elev{wb},turbine_discharge{wb},...
                ELWS_targets{wb},[],[],Optimize_day_by_day);
        end
    else
        [turb_discharges{wb},spill_discharges{wb},HWs{wb},~,~] = ...
            activeunits_to_discharges(x_zeroRows1,t,...
            frequency,Q{wb},ic_elev{wb},turbine_discharge{wb},...
            ELWS_targets{wb},mainstem_inflows{wb}.t,...
            mainstem_inflows{wb}.Q(zeroRows1,:),Optimize_day_by_day);
    end
```

```
240         %If we haven't reached the last reservoir, update mainstem_inflows.Q (
                ↪ include both turbine + spill incoming!)
241         if wb~=size(ic_elev,2)
242            mainstem_inflows{wb+1}.Q(zeroRows1,:)=...
243               bsxfun(@plus,turb_discharges{wb},spill_discharges{wb});
244         end
245         %Inequality constraints:
246         %Elevation violations - lower
247         if isnan(ELWS_limit{wb}(1))
248            deductions1=zeros(size(HWs{wb}(:,1:end)));
249         else
250            deductions1=-min(0,HWs{wb}(:,1:end)-ELWS_limit{wb}(1));
251         end
252         %Elevation violations - upper
253         if isnan(ELWS_limit{wb}(2))
254            deductions2=zeros(size(HWs{wb}(:,1:end)));
255         else
256            deductions2=max(0,HWs{wb}(:,1:end)-ELWS_limit{wb}(2));
257         end
258
259         c{wb}(setdiff([1:size(x{wb},1)],zeroRows1),4:end)=1;
260         c{wb}(zeroRows1,4:3+(1+size(x{wb},2))*2)=[deductions1 deductions2];
261
262         zeroRows2=find(all(c{wb}<=tolerance,2));
263         x_zeroRows2=x{wb}(zeroRows2,:);
264         if isempty(x_zeroRows2)
265            c{wb}(:,3+(1+size(x{wb},2))*2+1:end)=1;
266            zeroRows_empty=1;
267         end
268
269         turb_discharges2=zeros(size(x{wb},1),size(x{wb},2)+1);
270         spill_discharges2=zeros(size(spill_discharges{wb}));
271         if ~isempty(ia)
272            turb_discharges2(HWcalcrows,:)=turb_discharges{wb}(b,:);
273            spill_discharges2(HWcalcrows,:)=spill_discharges{wb}(b,:);
274         else
275            turb_discharges2(zeroRows1,:)=turb_discharges{wb};
276            spill_discharges2(zeroRows1,:)=spill_discharges{wb};
277         end
278         %-->need to reset this with zero rows back in
279         turb_discharges{wb}=turb_discharges2;
280         spill_discharges{wb}=spill_discharges2;
281         clearvars spill_discharges2 turb_discharges2 x_HWcalcrows HWcalcrows
282      end
283
284      %Continue on and calculate discharge DO if still feasible, if DO_narx is
             ↪ provided and a limit exists
285      if zeroRows_empty~=1 & ~isempty(DO_narx) & (wb~=size(ic_elev,2) | any(
             ↪ DO_limit))
286
287         %Discharge DO constraint
288         if wb==1
289            %Check to see if any cached rows can be skipped by DO calcs
290            if ~isempty(ia)
291               [DOcalcrows,b]=setdiff(zeroRows2,ia);
292               x_DOcalcrows=x{wb}(DOcalcrows,:);
293               DO_pred{wb}(b,:)=narx_predictions(DO_narx,...
294                  frequency,t,Q{wb},x_DOcalcrows,...
295                  turb_discharges{wb}(DOcalcrows,:),...
296                  spill_discharges{wb}(DOcalcrows,:),[],...
297                  Q{wb}.CWO,'do',Optimize_day_by_day);
298               [~,bb]=ismember(x_zeroRows2,cache.x,'rows'); bb=nonzeros(bb);
299               DO_pred{wb}(setdiff(1:size(zeroRows2,1),b),:)=...
300                  cache.DO(bb,tib(1:end-1));
```

```
301                 clearvars DOcalcrows x_DOcalcrows b
302             else
303                 DO_pred{wb}=narx_predictions(DO_narx,...
304                     frequency,t,Q{wb},x_zeroRows2,...
305                     turb_discharges{wb}(zeroRows2,:),...
306                     spill_discharges{wb}(zeroRows2,:),[],...
307                     Q{wb}.CWO,'do',Optimize_day_by_day);
308             end
309         else
310             mainstem_inflows_zeroRows2{wb}.Q=...
311                 mainstem_inflows{wb}.Q(zeroRows2,:);
312             mainstem_inflows_zeroRows2{wb}.T=...
313                 mainstem_inflows{wb}.T(zeroRows2,:);
314             mainstem_inflows_zeroRows2{wb}.DO=...
315                 mainstem_inflows{wb}.DO(zeroRows2,:);
316             DO_pred{wb}=narx_predictions(DO_narx,...
317                 frequency,t,Q{wb},x_zeroRows2,...
318                 turb_discharges{wb}(zeroRows2,:),...
319                 spill_discharges{wb}(zeroRows2,:),...
320                 mainstem_inflows_zeroRows2{wb},Q{wb}.CWO,'do',Optimize_day_by_day
                        ↪ );
321         end
322         %If we haven't reached the last reservoir, update mainstem_inflows.DO
323         if wb~=size(ic_elev,2)
324             mainstem_inflows{wb+1}.DO(zeroRows2,1)=...
325                 interp1(Q{wb}.CWO(:,1),Q{wb}.CWO(:,2),t(1));
326             mainstem_inflows{wb+1}.DO(zeroRows2,2:size(DO_pred{wb},2)+1)=...
327                 DO_pred{wb};
328         end
329         non_nan_count=sum(~isnan(DO_pred{wb}),2);
330         %DO violations - lower
331         if isnan(DO_limit(1))
332             DO_violations1=zeros(size(DO_pred{wb},1),1);
333         else
334             DO_violations1=sum(-min(0,DO_pred{wb}-DO_limit(1)),2)./non_nan_count
                        ↪ ;
335         end
336         %DO violations - upper
337         if isnan(DO_limit(2))
338             DO_violations2=zeros(size(DO_pred{wb},1),1);
339         else
340             DO_violations2=sum(max(0,DO_pred{wb}-DO_limit(2)),2)./non_nan_count;
341         end
342         DO_violations=[max(0,DO_violations1-DO_slack) max(0,DO_violations2-
                ↪ DO_slack)];
343 
344         c{wb}(setdiff([1:size(x{wb},1)],zeroRows2),3+(1+size(x{wb},2))*2+1:end)
                ↪ =1;
345         c{wb}(zeroRows2,3+(1+size(x{wb},2))*2+1:3+(1+size(x{wb},2))*2+2)=
                ↪ DO_violations;
346         clearvars DO_violations1 DO_violations2 Last_values
347 
348         zeroRows3=find(all(c{wb}<=tolerance,2));
349         x_zeroRows3=x{wb}(zeroRows3,:);
350         DO_pred{wb}(zeroRows2,:)=DO_pred{wb};
351         DO_pred{wb}=DO_pred{wb}(zeroRows3,:);
352         if isempty(x_zeroRows3)
353             c{wb}(:,3+(1+size(x{wb},2))*2+2+1:end)=1;
354             zeroRows_empty=1;
355         end
356 
357     end
358 
359     %Continue on and calculate discharge temp if still feasible
```

```
360         if zeroRows_empty~=1 & ~isempty(Temp_narx) & (wb~=size(ic_elev,2) | any(
                ↪ Temp_limit))
361             zeroRows4=find(all(c{wb}<=tolerance,2));
362             x_zeroRows4=x{wb}(zeroRows4,:);
363             if isempty(x_zeroRows4)
364                 c{wb}(:,3+(1+size(x{wb},2))*2+2+1:end)=1;
365                 zeroRows_empty=1;
366             end
367 
368             if zeroRows_empty~=1
369                 %Discharge Temp constraint
370                 if wb==1
371                     %Check to see if any cached rows can be skipped by temp calcs
372                     if ~isempty(ia)
373                         [Tcalcrows,b]=setdiff(zeroRows4,ia);
374                         x_Tcalcrows=x{wb}(Tcalcrows,:);
375                         Temp_pred{wb}(b,:)=...
376                             narx_predictions(Temp_narx,...
377                             frequency,t,Q{wb},x_Tcalcrows,...
378                             turb_discharges{wb}(Tcalcrows,:),...
379                             spill_discharges{wb}(Tcalcrows,:),[],...
380                             Q{wb}.TWO,'temp',Optimize_day_by_day);
381                         [~,bb]=ismember(x_zeroRows4,cache.x,'rows'); bb=nonzeros(bb);
382                         Temp_pred{wb}(setdiff(1:size(zeroRows4,1),b),:)=...
383                             cache.T(bb,tib(1:end-1));
384                         clearvars Tcalcrows x_Tcalcrows b
385                     else
386                         Temp_pred{wb}=...
387                             narx_predictions(Temp_narx,...
388                             frequency,t,Q{wb},x_zeroRows4,...
389                             turb_discharges{wb}(zeroRows4,:),...
390                             spill_discharges{wb}(zeroRows4,:),[],...
391                             Q{wb}.TWO,'temp',Optimize_day_by_day);
392                     end
393                 else
394                     mainstem_inflows_zeroRows4{wb}.Q=...
395                         mainstem_inflows{wb}.Q(zeroRows4,:);
396                     mainstem_inflows_zeroRows4{wb}.T=...
397                         mainstem_inflows{wb}.T(zeroRows4,:);
398                     mainstem_inflows_zeroRows4{wb}.DO=...
399                         mainstem_inflows{wb}.DO(zeroRows4,:);
400                     Temp_pred{wb}=...
401                         narx_predictions(Temp_narx,...
402                         frequency,t,Q{wb},x_zeroRows4,...
403                         turb_discharges{wb}(zeroRows4,:),...
404                         spill_discharges{wb}(zeroRows4,:),...
405                         mainstem_inflows_zeroRows4{wb},...
406                         Q{wb}.TWO,'temp',Optimize_day_by_day);
407                 end
408                 %If we haven't reached the last reservoir, update mainstem_inflows.T
409                 if wb~=size(ic_elev,2)
410                     mainstem_inflows{wb+1}.T(zeroRows3,1)=...
411                         interp1(Q{wb}.TWO(:,1),Q{wb}.TWO(:,2),t(1));
412                     mainstem_inflows{wb+1}.T(zeroRows3,2:size(Temp_pred{wb},2)+1)=...
413                         Temp_pred{wb};
414                 end
415                 non_nan_count=sum(~isnan(Temp_pred{wb}),2);
416                 %Temp violations - lower
417                 if isnan(Temp_limit(1))
418                     Temp_violations1=zeros(size(Temp_pred{wb},1),1);
419                 else
420                     Temp_violations1=sum(-min(0,Temp_pred{wb}-Temp_limit(1)),2)./
                            ↪ non_nan_count;
421                 end
```

```
            %Temp violations - upper
            if isnan(Temp_limit(2))
               Temp_violations2=zeros(size(Temp_pred{wb},1),1);
            else
               Temp_violations2=sum(max(0,Temp_pred{wb}-Temp_limit(2)),2)./
                   ↪ non_nan_count;
            end
            Temp_violations=[max(0,Temp_violations1-Temp_slack) max(0,
                ↪ Temp_violations2-Temp_slack)];

            c{wb}(setdiff([1:size(x{wb},1)],zeroRows4),3+(1+size(x{wb},2))
                ↪ *2+2+1:end)=1;
            c{wb}(zeroRows4,3+(1+size(x{wb},2))*2+2+1:3+(1+size(x{wb},2))*2+2+2)
                ↪ =Temp_violations;

            zeroRows5=find(all(c{wb}<=tolerance,2));
            x_zeroRows5=x{wb}(zeroRows5,:);
            Temp_pred{wb}(zeroRows4,:)=Temp_pred{wb};
            Temp_pred{wb}=Temp_pred{wb}(zeroRows5,:);
            if isempty(x_zeroRows5)
               zeroRows_empty=1;
            end

         end
      end
   end
   %If we haven't reached the last reservoir, update mainstem_inflows.t, remove
       ↪ NaN from mainstem_inflows.T and mainstem_inflows.DO, and update
       ↪ zeroRows0
   if wb~=size(ic_elev,2) & zeroRows_empty~=1
      mainstem_inflows{wb+1}.t=t;
      %Remove Nan values and interpolate for T and DO
      for i=1:size(mainstem_inflows{wb+1}.T,1)
         extrap_index=~isnan(mainstem_inflows{wb+1}.T(i,:));
         [~,column]=find(extrap_index==1); extrap_index=column(end);
         mainstem_inflows{wb+1}.T(i,:)=...
            interp1(t(1,~isnan(mainstem_inflows{wb+1}.T(i,:))),...
            mainstem_inflows{wb+1}.T(i,~isnan(mainstem_inflows{wb+1}.T(i,:)))
                ↪ ,...
            t,'linear',mainstem_inflows{wb+1}.T(i,extrap_index));
         mainstem_inflows{wb+1}.DO(i,:)=...
            interp1(t(1,~isnan(mainstem_inflows{wb+1}.DO(i,:))),...
            mainstem_inflows{wb+1}.DO(i,~isnan(mainstem_inflows{wb+1}.DO(i,:)))
                ↪ ,...
            t,'linear',mainstem_inflows{wb+1}.DO(i,extrap_index));
         clearvars extrap_index column
      end
      zeroRows0=find(all(c{wb}<=tolerance,2));
   end

end

%Update c_all with the values from c{wb}
c_all=[c{:}];
```

penalty_fcn._inf.m

```
function [c_all,ceq]=penalty_fcn_inf(x_allwb,t,frequency,Q,ic_elev,...
   turbine_discharge,ELWS_limit,max_hrly_unit_change,...
   WQ,zero_gen_limit,xprev,ELWS_targets,tolerance,cache,Optimize_day_by_day)

% modified penalty function that computes all constraints
%
```

```
% Calculates penalty violations, starting with the least expensive
% computations and continuing on to the more expensive computations for
% runs that are found to be feasible thus far
%
% Inputs:
% x_allwb - hourly turbine time series (as rows for vectorizing!),
% integers between 0 and no_of_units for all waterbodies
% t  time series of JDAY values
% frequency - frequency of predictions (hourly=1/24)
% Q - all other inflows and outflows, interpolation settings,
% storage-elev curve, and tailwater curve
% ic_elev - initial condition (meters)
% turbine_discharge - turbine discharge curve at fixed MW level, with
% col 1 in meters and col 2 in cms
% ELWS_limit - min and max elevation limits for constraints, in meters
% max_hrly_unit_change - max number of units that can be changed per hour
% (1 for OHL)
% WQ - structure containing water quality constraints and NARX models
% DO_narx - structure containing everything needed to make DO discharge
% predictions, including:
% turb_colum - column in exogenous variables with turb flows
% spill_column - column in exogenous variables with spill flows
% times - JDAY values used in training (not used)
% inputDelays - delays for exogenous inputs
% feedbackDelays - delays for prediction feedbacks
% input_variables - 2 row cell containing variable names in first
% row and column number in second. For example, 'MET_WB1'
% contains multiple columns of data but only some may be used
% for NARX predictions
% bias - bias for each trained neural network
% weights - weights for each trained neural network (sum to 1)
% narx_net_closed - neural networks
% DO_limit - lower and upper DO limits (NaN means it doesn't exist)
% DO_slack - relaxation from DO_limit (either upper or lower -
% doesn't make sense to have both)
% Temp_narx - structure containing everything needed to make temp discharge
% predictions, including:
% turb_colum - column in exogenous variables with turb flows
% spill_column - column in exogenous variables with spill flows
% times - JDAY values used in training (not used)
% inputDelays - delays for exogenous inputs
% feedbackDelays - delays for prediction feedbacks
% input_variables - 2 row cell containing variable names in first
% row and column number in second. For example, 'MET_WB1'
% contains multiple columns of data but only some may be used
% for NARX predictions
% bias - bias for each trained neural network
% weights - weights for each trained neural network (sum to 1)
% narx_net_closed - neural networks
% Temp_limit - lower and upper temp limits (NaN means it doesn't exist)
% Temp_slack - relaxation from Temp_limit (either upper or lower -
% doesn't make sense to have both)
% zero_gen_limit - Zero generation hourly limit (can't go longer than
% this with no turb flow)
% xprev - vector of previous active turbine levels
% ELWS_targets - 2 column matrix with JDAY in col1 and elevation target
% in col2
% tolerance - penalty tolerance
% cache - water quality predictions provided by W2 simulations
% Optimize_day_by_day - 1 if optimizing daily, 0 if optimizing all together
% Outputs:
% c_all  inequality constraint output (n/a, so 0)
% ceq - equality constraint output (=0 for feasible solution)

```

```
%Name global variables to be used for function counts
global funccount_cache_global funccount_tot_global
funccount_tot_global=funccount_tot_global+size(x_allwb,1);

x_allwb=round(x_allwb);

%Equality constraint
ceq=[];

%Preallocate memory
x{1,size(ic_elev,2)}=[];
xall{1,size(ic_elev,2)}=[];
turb_discharges{1,size(ic_elev,2)}=[];
HWs{1,size(ic_elev,2)}=[];
c_all=zeros(size(x{1},1),size(ic_elev,2)*(3+(1+size(x{1},2))*2+2+2));

%If using the cache, get list of cache indices here
if ~isempty(cache)
   [~,~,tib]=intersect(t,cache.t);
   if Optimize_day_by_day==0 & size(ic_elev,2)==1 & ~isempty(cache.x)
      [ia,ib]=ismember(x_allwb,cache.x,'rows');
   else
      index=find(cache.t==t(1)); %last index for previous operations
      if index==1 %first day
         [ia,ib]=ismember(x_allwb,cache.x(:,index:index+23),'rows');
      else
         [ia,ib]=ismember([repmat(xprev{1}(size(xprev{1},2)-tib(1)+2:end),...
            size(x_allwb,1),1) x_allwb],cache.x(:,1:index+23),'rows'); %fix
                ↪ later to solve multi waterbody problems
      end
   end
   ia=find(ia==1); ib=ib(ib~=0);
   funccount_cache_global=funccount_cache_global+size(ia,1);
   if ~isempty(ia) fprintf(['Cached points here: ', num2str(ib'), '\n']); end
else
   ia=[]; ib=[];
end

zeroRows_empty=0;
zeroRows0=[1:size(x_allwb,1)]';

for wb=1:size(ic_elev,2)
   %Split up rows of x to separate reservoirs
   x{wb}=x_allwb(:,wb*(size(t,2)-1)-(size(t,2)-2):wb*(size(t,2)-1));
   %Preallocate c, with columns representing: (1) change in active unit
       ↪ violations, (2) zero gen hourly limit, (3) oscillations constraint,
       ↪ (4:28) ELWS lower violations, (29:53) ELWS upper violations, (54:55)
       ↪ mean lower and upper DO violations, (56:57) mean temp lower and upper
       ↪ violations
   c{wb}=zeros(size(x{1},1),3+(1+size(x{1},2))*2+2+2);
end
clearvars wb

for wb=1:size(ic_elev,2)

   c{wb}(setdiff([1:size(x{wb},1)],zeroRows0),:)=Inf;

   %Check if all entries in x are infeasible due to previous reservoirs, and if
       ↪ so set the rest of c==1 and go to end
   if zeroRows_empty==1
      c{wb}(:)=Inf;
   else

      %Break up WQ structure into separate variables
```

```
129         DO_narx=WQ{wb}.DO_narx; DO_limit=WQ{wb}.DO_limit; DO_slack=WQ{wb}.DO_slack
                ↪ ;
130         Temp_narx=WQ{wb}.Temp_narx; Temp_limit=WQ{wb}.Temp_limit; Temp_slack=WQ{wb
                ↪ }.Temp_slack;
131
132         %Stitch together xprev & x to check for feasibility wrt active unit viols,
                ↪  zero generation hrly limit, and oscillations
133         xall{wb}=[repmat(xprev{wb},size(x{wb},1),1) x{wb}];
134
135         %Change in active unit violations
136         if isempty(max_hrly_unit_change{wb})
137            delta_sum=zeros(size(zeroRows0,1),1);
138         else
139            delta=abs(round(xall{wb}(zeroRows0,2:end))-...
140               round(xall{wb}(zeroRows0,1:end-1)));
141            index=find(delta<=max_hrly_unit_change{wb});
142            delta(index)=0;
143            delta_sum=sum(delta')';
144         end
145
146         %Zero generation hourly limit - can't go longer with no turb flow
147         if isempty(zero_gen_limit{wb})
148            zero_gen_viols_sum=zeros(size(zeroRows0,1),1);
149         else
150            zero_gen_viols=zeros(size(zeroRows0,1),size(xall{wb},2)-...
151               zero_gen_limit{wb}-1);
152            x_trans=xall{wb}(zeroRows0,:)';
153            for i=1:size(x_trans,1)-zero_gen_limit{wb}
154               a=sum(x_trans(i:i+zero_gen_limit{wb},:))';
155               zero_gen_viols(:,i)=(a==0);
156            end
157            clearvars i
158            zero_gen_viols_sum=sum(zero_gen_viols')';
159         end
160
161         %Oscillations constraint - violates whenever the number of turbines
                ↪ increases and then decreases within 3 hours, or vice versa
162         osc_violations=zeros(size(zeroRows0,1),size(xall{wb},2)-2);
163         xall_osc=xall{wb}(zeroRows0,:);
164         for ii=1:size(xall_osc,1) %loop through each member of population
165            for i=1:size(xall_osc,2)-2; %loop forward through time
166               if xall_osc(ii,i+1)>xall_osc(ii,i) & ...
167                     xall_osc(ii,i+2)<xall_osc(ii,i+1)
168                  osc_violations(ii,i)=1;
169               elseif xall_osc(ii,i+1)<xall_osc(ii,i) & ...
170                     xall_osc(ii,i+2)>xall_osc(ii,i+1)
171                  osc_violations(ii,i)=1;
172               elseif i~=1
173                  if xall_osc(ii,i)==xall_osc(ii,i+1) %need 3 hrs btwn ramping up
                        ↪ and down
174                     if xall_osc(ii,i-1)<xall_osc(ii,i) & ...
175                           xall_osc(ii,i+1)>xall_osc(ii,i+2) %ramping up & back
                              ↪ down too quickly
176                        osc_violations(ii,i)=1;
177                     elseif xall_osc(ii,i-1)>xall_osc(ii,i) & ...
178                           xall_osc(ii,i+1)<xall_osc(ii,i+2) %ramping down & back
                              ↪ up too quickly
179                        osc_violations(ii,i)=1;
180                     end
181                  end
182               end
183            end
184         end
185         clearvars i ii xall_osc
```

```
186         osc_violations_sum=sum(osc_violations')';
187
188         %Compile least expensive constraints
189         c{wb}(zeroRows0,1:3)=...
190            [delta_sum zero_gen_viols_sum osc_violations_sum];
191
192         clearvars zeroRows1 zeroRows2 zeroRows3 zeroRows4 x_zeroRows1 x_zeroRows2
               ↪ x_zeroRows3 x_zeroRows4
193         x_zeroRows1=[];
194         x_zeroRows2=[];
195         x_zeroRows3=[];
196         x_zeroRows4=[];
197         %Only compute expensive constraints if all others pass
198         zeroRows1=zeroRows0;
199         x_zeroRows1=x{wb}(zeroRows1,:);
200         if isempty(x_zeroRows1)
201            c{wb}(:,4:end)=Inf;
202            zeroRows_empty=1;
203         end
204
205         if zeroRows_empty~=1
206
207            %Calculate headwater elevs for constraints
208            if wb==1
209               %Preallocate mainstem_inflows for following wbs
210               mainstem_inflows=cell(1:size(ic_elev,2));
211               for i=1:size(ic_elev,2)
212                  mainstem_inflows{i}.t=[];
213                  mainstem_inflows{i}.Q=[];
214                  mainstem_inflows{i}.T=[];
215                  mainstem_inflows{i}.DO=[];
216               end
217               clearvars i
218               %Check to see if any cached rows can be skipped by elev calcs
219               if ~isempty(ia)
220                  [HWcalcrows,b]=setdiff(zeroRows1,ia);
221                  x_HWcalcrows=x{wb}(HWcalcrows,:);
222                  [turb_discharges{wb}(b,:),spill_discharges{wb}(b,:),HWs{wb}(b,:)
                        ↪ ,~,~] = ...
223                     activeunits_to_discharges(x_HWcalcrows,t,...
224                     frequency,Q{wb},ic_elev{wb},turbine_discharge{wb},...
225                     ELWS_targets{wb},[],[],Optimize_day_by_day);
226                  [~,bb]=ismember(x_zeroRows1,cache.x,'rows'); bb=nonzeros(bb);
227                  HWs{wb}(setdiff(1:size(zeroRows1,1),b),:)=...
228                     cache.HWs(bb,tib);
229               else
230                  [turb_discharges{wb},spill_discharges{wb},HWs{wb},~,~] = ...
231                     activeunits_to_discharges(x_zeroRows1,t,...
232                     frequency,Q{wb},ic_elev{wb},turbine_discharge{wb},...
233                     ELWS_targets{wb},[],[],Optimize_day_by_day);
234               end
235            else
236               [turb_discharges{wb},spill_discharges{wb},HWs{wb},~,~] = ...
237                  activeunits_to_discharges(x_zeroRows1,t,...
238                  frequency,Q{wb},ic_elev{wb},turbine_discharge{wb},...
239                  ELWS_targets{wb},mainstem_inflows{wb}.t,...
240                  mainstem_inflows{wb}.Q(zeroRows1,:),Optimize_day_by_day);
241            end
242            %If we haven't reached the last reservoir, update mainstem_inflows.Q (
                  ↪ include both turbine + spill incoming!)
243            if wb~=size(ic_elev,2)
244               mainstem_inflows{wb+1}.Q(zeroRows1,:)=...
245                  bsxfun(@plus,turb_discharges{wb},spill_discharges{wb});
246            end
```

```
247             %Inequality constraints:
248             %Elevation violations - lower
249             if isnan(ELWS_limit{wb}(1))
250                deductions1=zeros(size(HWs{wb}(:,1:end)));
251             else
252                deductions1=-min(0,HWs{wb}(:,1:end)-ELWS_limit{wb}(1));
253             end
254             %Elevation violations - upper
255             if isnan(ELWS_limit{wb}(2))
256                deductions2=zeros(size(HWs{wb}(:,1:end)));
257             else
258                deductions2=max(0,HWs{wb}(:,1:end)-ELWS_limit{wb}(2));
259             end
260
261             c{wb}(setdiff([1:size(x{wb},1)],zeroRows1),4:end)=Inf;
262             c{wb}(zeroRows1,4:3+(1+size(x{wb},2))*2)=[deductions1 deductions2];
263
264             zeroRows2=zeroRows0;
265             x_zeroRows2=x{wb}(zeroRows2,:);
266             if isempty(x_zeroRows2)
267                c{wb}(:,3+(1+size(x{wb},2))*2+1:end)=Inf;
268                zeroRows_empty=1;
269             end
270
271             turb_discharges2=zeros(size(x{wb},1),size(x{wb},2)+1);
272             spill_discharges2=zeros(size(spill_discharges{wb}));
273             if ~isempty(ia)
274                turb_discharges2(HWcalcrows,:)=turb_discharges{wb}(b,:);
275                spill_discharges2(HWcalcrows,:)=spill_discharges{wb}(b,:);
276             else
277                turb_discharges2(zeroRows1,:)=turb_discharges{wb};
278                spill_discharges2(zeroRows1,:)=spill_discharges{wb};
279             end
280             %-->need to reset this with zero rows back in
281             turb_discharges{wb}=turb_discharges2;
282             spill_discharges{wb}=spill_discharges2;
283             clearvars spill_discharges2 turb_discharges2 x_HWcalcrows HWcalcrows
284          end
285
286          %Continue on and calculate discharge DO if still feasible, if DO_narx is
                 ↪ provided and a limit exists
287          if zeroRows_empty~=1 & ~isempty(DO_narx) & (wb~=size(ic_elev,2) | any(
                 ↪ DO_limit))
288
289             %Discharge DO constraint
290             if wb==1
291                %Check to see if any cached rows can be skipped by DO calcs
292                if ~isempty(ia)
293                   [DOcalcrows,b]=setdiff(zeroRows2,ia);
294                   x_DOcalcrows=x{wb}(DOcalcrows,:);
295                   DO_pred{wb}(b,:)=narx_predictions(DO_narx,...
296                      frequency,t,Q{wb},x_DOcalcrows,...
297                      turb_discharges{wb}(DOcalcrows,:),...
298                      spill_discharges{wb}(DOcalcrows,:),[],...
299                      Q{wb}.CWO,'do',Optimize_day_by_day);
300                   [~,bb]=ismember(x_zeroRows2,cache.x,'rows'); bb=nonzeros(bb);
301                   DO_pred{wb}(setdiff(1:size(zeroRows2,1),b),:)=...
302                      cache.DO(bb,tib(1:end-1));
303                   clearvars DOcalcrows x_DOcalcrows b
304                else
305                   DO_pred{wb}=narx_predictions(DO_narx,...
306                      frequency,t,Q{wb},x_zeroRows2,...
307                      turb_discharges{wb}(zeroRows2,:),...
308                      spill_discharges{wb}(zeroRows2,:),[],...
```

```
                Q{wb}.CWO,'do',Optimize_day_by_day);
            end
        else
            mainstem_inflows_zeroRows2{wb}.Q=...
                mainstem_inflows{wb}.Q(zeroRows2,:);
            mainstem_inflows_zeroRows2{wb}.T=...
                mainstem_inflows{wb}.T(zeroRows2,:);
            mainstem_inflows_zeroRows2{wb}.DO=...
                mainstem_inflows{wb}.DO(zeroRows2,:);
            DO_pred{wb}=narx_predictions(DO_narx,...
                frequency,t,Q{wb},x_zeroRows2,...
                turb_discharges{wb}(zeroRows2,:),...
                spill_discharges{wb}(zeroRows2,:),...
                mainstem_inflows_zeroRows2{wb},Q{wb}.CWO,'do',Optimize_day_by_day
                    ↪ );
        end
        %If we haven't reached the last reservoir, update mainstem_inflows.DO
        if wb~=size(ic_elev,2)
            mainstem_inflows{wb+1}.DO(zeroRows2,1)=...
                interp1(Q{wb}.CWO(:,1),Q{wb}.CWO(:,2),t(1));
            mainstem_inflows{wb+1}.DO(zeroRows2,2:size(DO_pred{wb},2)+1)=...
                DO_pred{wb};
        end
        non_nan_count=sum(~isnan(DO_pred{wb}),2);
        %DO violations - lower
        if isnan(DO_limit(1))
            DO_violations1=zeros(size(DO_pred{wb},1),1);
        else
            DO_violations1=sum(-min(0,DO_pred{wb}-DO_limit(1)),2)./non_nan_count
                ↪ ;
        end
        %DO violations - upper
        if isnan(DO_limit(2))
            DO_violations2=zeros(size(DO_pred{wb},1),1);
        else
            DO_violations2=sum(max(0,DO_pred{wb}-DO_limit(2)),2)./non_nan_count;
        end
        DO_violations=[max(0,DO_violations1-DO_slack) max(0,DO_violations2-
            ↪ DO_slack)];

        c{wb}(setdiff([1:size(x{wb},1)],zeroRows2),3+(1+size(x{wb},2))*2+1:end)
            ↪ =Inf;
        c{wb}(zeroRows2,3+(1+size(x{wb},2))*2+1:3+(1+size(x{wb},2))*2+2)=
            ↪ DO_violations;
        clearvars DO_violations1 DO_violations2 Last_values

        zeroRows3=zeroRows0;
        x_zeroRows3=x{wb}(zeroRows3,:);
        DO_pred{wb}(zeroRows2,:)=DO_pred{wb};
        DO_pred{wb}=DO_pred{wb}(zeroRows3,:);
        if isempty(x_zeroRows3)
            c{wb}(:,3+(1+size(x{wb},2))*2+2+1:end)=Inf;
            zeroRows_empty=1;
        end

    end

    %Continue on and calculate discharge temp if still feasible
    if zeroRows_empty~=1 & ~isempty(Temp_narx) & (wb~=size(ic_elev,2) | any(
        ↪ Temp_limit))
        zeroRows4=zeroRows0;
        x_zeroRows4=x{wb}(zeroRows4,:);
        if isempty(x_zeroRows4)
            c{wb}(:,3+(1+size(x{wb},2))*2+2+1:end)=Inf;
```

```
367             zeroRows_empty=1;
368         end
369
370         if zeroRows_empty~=1
371             %Discharge Temp constraint
372             if wb==1
373                 %Check to see if any cached rows can be skipped by temp calcs
374                 if ~isempty(ia)
375                     [Tcalcrows,b]=setdiff(zeroRows4,ia);
376                     x_Tcalcrows=x{wb}(Tcalcrows,:);
377                     Temp_pred{wb}(b,:)=...
378                         narx_predictions(Temp_narx,...
379                         frequency,t,Q{wb},x_Tcalcrows,...
380                         turb_discharges{wb}(Tcalcrows,:),...
381                         spill_discharges{wb}(Tcalcrows,:),[],...
382                         Q{wb}.TWO,'temp',Optimize_day_by_day);
383                     [~,bb]=ismember(x_zeroRows4,cache.x,'rows'); bb=nonzeros(bb);
384                     Temp_pred{wb}(setdiff(1:size(zeroRows4,1),b),:)=...
385                         cache.T(bb,tib(1:end-1));
386                     clearvars Tcalcrows x_Tcalcrows b
387                 else
388                     Temp_pred{wb}=...
389                         narx_predictions(Temp_narx,...
390                         frequency,t,Q{wb},x_zeroRows4,...
391                         turb_discharges{wb}(zeroRows4,:),...
392                         spill_discharges{wb}(zeroRows4,:),[],...
393                         Q{wb}.TWO,'temp',Optimize_day_by_day);
394                 end
395             else
396                 mainstem_inflows_zeroRows4{wb}.Q=...
397                     mainstem_inflows{wb}.Q(zeroRows4,:);
398                 mainstem_inflows_zeroRows4{wb}.T=...
399                     mainstem_inflows{wb}.T(zeroRows4,:);
400                 mainstem_inflows_zeroRows4{wb}.DO=...
401                     mainstem_inflows{wb}.DO(zeroRows4,:);
402                 Temp_pred{wb}=...
403                     narx_predictions(Temp_narx,...
404                     frequency,t,Q{wb},x_zeroRows4,...
405                     turb_discharges{wb}(zeroRows4,:),...
406                     spill_discharges{wb}(zeroRows4,:),...
407                     mainstem_inflows_zeroRows4{wb},...
408                     Q{wb}.TWO,'temp',Optimize_day_by_day);
409             end
410             %If we haven't reached the last reservoir, update mainstem_inflows.T
411             if wb~=size(ic_elev,2)
412                 mainstem_inflows{wb+1}.T(zeroRows3,1)=...
413                     interp1(Q{wb}.TWO(:,1),Q{wb}.TWO(:,2),t(1));
414                 mainstem_inflows{wb+1}.T(zeroRows3,2:size(Temp_pred{wb},2)+1)=...
415                     Temp_pred{wb};
416             end
417             non_nan_count=sum(~isnan(Temp_pred{wb}),2);
418             %Temp violations - lower
419             if isnan(Temp_limit(1))
420                 Temp_violations1=zeros(size(Temp_pred{wb},1),1);
421             else
422                 Temp_violations1=sum(-min(0,Temp_pred{wb}-Temp_limit(1)),2)./
                        ↪ non_nan_count;
423             end
424             %Temp violations - upper
425             if isnan(Temp_limit(2))
426                 Temp_violations2=zeros(size(Temp_pred{wb},1),1);
427             else
428                 Temp_violations2=sum(max(0,Temp_pred{wb}-Temp_limit(2)),2)./
                        ↪ non_nan_count;
```

```
            end
            Temp_violations=[max(0,Temp_violations1-Temp_slack) max(0,
                ↪ Temp_violations2-Temp_slack)];

            c{wb}(setdiff([1:size(x{wb},1)],zeroRows4),3+(1+size(x{wb},2))
                ↪ *2+2+1:end)=Inf;
            c{wb}(zeroRows4,3+(1+size(x{wb},2))*2+2+1:3+(1+size(x{wb},2))*2+2+2)
                ↪ =Temp_violations;

            zeroRows5=zeroRows0;
            x_zeroRows5=x{wb}(zeroRows5,:);
            Temp_pred{wb}(zeroRows4,:)=Temp_pred{wb};
            Temp_pred{wb}=Temp_pred{wb}(zeroRows5,:);
            if isempty(x_zeroRows5)
               zeroRows_empty=1;
            end

         end
      end
   end
   %If we haven't reached the last reservoir, update mainstem_inflows.t, remove
       ↪ NaN from mainstem_inflows.T and mainstem_inflows.DO, and update
       ↪ zeroRows0
   if wb~=size(ic_elev,2) & zeroRows_empty~=1
      mainstem_inflows{wb+1}.t=t;
      %Remove Nan values and interpolate for T and DO
      for i=1:size(mainstem_inflows{wb+1}.T,1)
         extrap_index=~isnan(mainstem_inflows{wb+1}.T(i,:));
         [~,column]=find(extrap_index==1); extrap_index=column(end);
         mainstem_inflows{wb+1}.T(i,:)=...
            interp1(t(1,~isnan(mainstem_inflows{wb+1}.T(i,:))),...
            mainstem_inflows{wb+1}.T(i,~isnan(mainstem_inflows{wb+1}.T(i,:)))
                ↪ ,...
            t,'linear',mainstem_inflows{wb+1}.T(i,extrap_index));
         mainstem_inflows{wb+1}.DO(i,:)=...
            interp1(t(1,~isnan(mainstem_inflows{wb+1}.DO(i,:))),...
            mainstem_inflows{wb+1}.DO(i,~isnan(mainstem_inflows{wb+1}.DO(i,:)))
                ↪ ,...
            t,'linear',mainstem_inflows{wb+1}.DO(i,extrap_index));
         clearvars extrap_index column
      end
      zeroRows0=find(all(c{wb}<=tolerance,2));
   end

end

%Update c_all with the values from c{wb}
c_all=[c{:}];
```

runW2trainingpop.m

```
copyfile(CFG{wb}.w2inputDir,directory)

%Open control file and modify TMEND
fid=fopen([directory '/w2_con.npt']);
i=1; A{i}=fgetl(fid);
while ischar(A{i}) i=i+1; A{i}=fgetl(fid); end
fclose(fid); A{28}(22:24)=num2str(t_all(end));
fid=fopen([directory '/w2_con.npt'],'w');
for i=1:numel(A)
   fprintf(fid,'%s\r\n', A{i});
   if A{i+1}==-1
      break
```

```
   end
end
fclose(fid); clearvars A i fid

%Open qot_br1.npt and modify turb and spill columns
fid=fopen([directory '/qot_br1.npt']);
i=1; A{i}=fgetl(fid);
while ischar(A{i})
   i=i+1; A{i}=fgetl(fid);
   if i>3
      if str2double(A{i}(1:8))>=t_all(1)
         A(end)=[]; break
      end
   end
end
fclose(fid);
if strcmp(CFG{wb}.TurbSpillOrder,'1')
   replacements{wb}=[Qtrainingpop{trindex}{wb}.QOT_BR1_T(Qtrainingpop{trindex}{
       ↪ wb}.QOT_BR1_T(:,1)>=t_all(1),:) ...
      Qtrainingpop{trindex}{wb}.QOT_BR1_S(Qtrainingpop{trindex}{wb}.QOT_BR1_S
          ↪ (:,1)>=t_all(1),2)];
elseif strcmp(CFG{wb}.TurbSpillOrder,'0')
   replacements{wb}=[Qtrainingpop{trindex}{wb}.QOT_BR1_S(Qtrainingpop{trindex}{
       ↪ wb}.QOT_BR1_S(:,1)>=t_all(1),:) ...
      Qtrainingpop{trindex}{wb}.QOT_BR1_T(Qtrainingpop{trindex}{wb}.QOT_BR1_T
          ↪ (:,1)>=t_all(1),2)];
end
for i=1:size(replacements{wb},1)
   A{numel(A)+1}=sprintf('%8.3f%8.3f%8.3f', replacements{wb}(i,:));
end
fid=fopen([directory '/qot_br1.npt'],'w');
for i=1:numel(A)
   fprintf(fid,'%s\r\n', A{i});
end
fclose(fid); clearvars A i fid

%Run executable w2.exe
cd(directory)
clearvars binarydecimalguide
[~,~]=system(['w2.exe &']); %the & means execute in the background
cd ../..

clearvars a ia ib DO_noNAN T_noNAN flowout turbs spills HWs
```

runW2trainingpop_part2.m

```
cd(directory)
delete('w2.exe'); delete('pre.exe');
cd ../..

%Read in results from two and cwo files (assume DO is last col in cwo)
T{trindex}=[]; DO{trindex}=[];
d=dir([directory '/two*.opt']);
fid=fopen([directory '/' d(end).name]);
C=textscan(fid,[repmat('%8f', 1, 50) '%*[^\n]'],10^8,...
   'headerLines',3,'collectoutput', true); %50 & 10^8 are arbitrary big numbers
T{trindex}=C{1}; T{trindex}(:,isnan(T{trindex}(1,:)))=[];
fclose(fid);
d=dir([directory '/cwo*.opt']);
fid=fopen([directory '/' d(end).name]);
C=textscan(fid,[repmat('%8f', 1, 50) '%*[^\n]'],10^8,...
   'headerLines',3,'collectoutput', true); %50 & 10^8 are arbitrary big numbers
DO{trindex}=C{1}; DO{trindex}(:,isnan(DO{trindex}(1,:)))=[];
```

```
DO{trindex}=[DO{trindex}(:,1) DO{trindex}(:,end)];
fclose(fid);
clearvars d C fid
%Reset 0 values to nan
T{trindex}(T{trindex}(:,2)==0,2)=nan;
DO{trindex}(DO{trindex}(:,2)==0,2)=nan;

%% Update cache
if size(CFG,2)==1
   cache.x=[cache.x; trainingpop(trindex,:)];
   [~,~,HWs,~,~]=activeunits_to_discharges(trainingpop(trindex,:),...
      t,frequency,Qtrainingpop{trindex}{1},ic_elev{1},...
      turbine_discharge{1},ELWS_targets{1},[],[],Optimize_day_by_day);
   cache.HWs=[cache.HWs; HWs];
   DO_noNAN=interp1(DO{trindex}(~isnan(DO{trindex}(:,2)),1),...
      DO{trindex}(~isnan(DO{trindex}(:,2)),2),t_all(2:end));
   T_noNAN=interp1(T{trindex}(~isnan(T{trindex}(:,2)),1),...
      T{trindex}(~isnan(T{trindex}(:,2)),2),t_all(2:end));
   %Fill in Nans at the end
   a=DO_noNAN(~isnan(DO_noNAN)); DO_noNAN(isnan(DO_noNAN))=a(end);
   a=T_noNAN(~isnan(T_noNAN)); T_noNAN(isnan(T_noNAN))=a(end);
   turbs=interp1(Qtrainingpop{trindex}{1}.QOT_BR1_T(:,1),Qtrainingpop{trindex
      ↪ }{1}.QOT_BR1_T(:,2),t_all);
   spills=interp1(Qtrainingpop{trindex}{1}.QOT_BR1_S(:,1),Qtrainingpop{trindex
      ↪ }{1}.QOT_BR1_S(:,2),t_all);
   flowout=turbs(2:end)+spills(2:end);
   DO_noNAN(flowout==0)=nan; T_noNAN(flowout==0)=nan;
   cache.DO=[cache.DO; DO_noNAN]; cache.T=[cache.T; T_noNAN];
end
clearvars a ia ib DO_noNAN T_noNAN flowout turbs spills HWs directory
```

update_cache.m

```
% Add solution and W2 outputs to cache

% if Optimize_day_by_day==0 & size(CFG,2)==1
if size(CFG,2)==1
   if ~isempty(cache.x)
   %If using the cache, get list of cache indices here
      [~,ia,ib]=intersect(x_final_all{end},cache.x,'rows');
   else
      ia=[];
   end
   if isempty(ia)
      cache.x=[cache.x; x_final_all{end}];
      cache.HWs=[cache.HWs; HWs{wb}];
      DO_noNAN=interp1(W2validation{1}.DO(~isnan(W2validation{1}.DO(:,2)),1),...
         W2validation{1}.DO(~isnan(W2validation{1}.DO(:,2)),2),t_all(2:end));
      T_noNAN=interp1(W2validation{1}.T(~isnan(W2validation{1}.T(:,2)),1),...
         W2validation{1}.T(~isnan(W2validation{1}.T(:,2)),2),t_all(2:end));
      %Fill in Nans at the end
      a=DO_noNAN(~isnan(DO_noNAN)); DO_noNAN(isnan(DO_noNAN))=a(end);
      a=T_noNAN(~isnan(T_noNAN)); T_noNAN(isnan(T_noNAN))=a(end);
      turbs=interp1(Q{1}.QOT_BR1_T(:,1),Q{1}.QOT_BR1_T(:,2),t_all);
      spills=interp1(Q{1}.QOT_BR1_S(:,1),Q{1}.QOT_BR1_S(:,2),t_all);
      flowout=turbs(2:end)+spills(2:end);
      DO_noNAN(flowout==0)=nan; T_noNAN(flowout==0)=nan;
      cache.DO=[cache.DO; DO_noNAN]; cache.T=[cache.T; T_noNAN];
   end
end
clearvars a ia ib DO_noNAN T_noNAN flowout turbs spills
```

updateQ.m

```
function Q=updateQ(Q,CFG,x_final,t,frequency,ic_elev,turbine_discharge,...
   WQ,xprev,ELWS_targets,cache,Optimize_day_by_day)

% Updates the structure Q with ELWS, discharge flows, and discharge WQ
% based on previous days optimized
%
% Inputs:
% Q - all other inflows and outflows, interpolation settings, and
% storage-elev curve
% CFG - structure containing field values from config files
% x_final - vector containing timeseries of active turbine levels for all
% waterbodies
% t  time series of JDAY values
% frequency - prediction frequency (ex: 0.25=1/4 day=6 hours)
% ic_elev - initial elevation condition (meters)
% turbine_discharge - turbine discharge curve at fixed MW level, with
% col 1 in meters and col 2 in cms
% WQ - structure containing water quality constraints and NARX models
% DO_narx - structure containing everything needed to make DO discharge
% predictions, including:
% turb_colum - column in exogenous variables with turb flows
% spill_column - column in exogenous variables with spill flows
% times - JDAY values used in training (not used)
% inputDelays - delays for exogenous inputs
% feedbackDelays - delays for prediction feedbacks
% input_variables - 2 row cell containing variable names in first
% row and column number in second. For example, 'MET_WB1'
% contains multiple columns of data but only some may be used
% for NARX predictions
% bias - bias for each trained neural network
% weights - weights for each trained neural network (sum to 1)
% narx_net_closed - neural networks
% DO_limit - lower and upper DO limits (NaN means it doesn't exist)
% DO_slack - relaxation from DO_limit (either upper or lower -
% doesn't make sense to have both)
% Temp_narx - structure containing everything needed to make temp discharge
% predictions, including:
% turb_colum - column in exogenous variables with turb flows
% spill_column - column in exogenous variables with spill flows
% times - JDAY values used in training (not used)
% inputDelays - delays for exogenous inputs
% feedbackDelays - delays for prediction feedbacks
% input_variables - 2 row cell containing variable names in first
% row and column number in second. For example, 'MET_WB1'
% contains multiple columns of data but only some may be used
% for NARX predictions
% bias - bias for each trained neural network
% weights - weights for each trained neural network (sum to 1)
% narx_net_closed - neural networks
% Temp_limit - lower and upper temp limits (NaN means it doesn't exist)
% Temp_slack - relaxation from Temp_limit (either upper or lower -
% doesn't make sense to have both)
% ELWS_targets - 2 column matrix with JDAY in col1 and elevation target
% in col2
% cache - water quality predictions provided by W2 simulations
% Optimize_day_by_day - 1 if optimizing daily, 0 if optimizing all together
% Outputs:
% Q - all other inflows and outflows, interpolation settings,
% storage-elev curve, and tailwater curve (all in meters)

for wb=1:size(CFG,2)
   clearvars incoming_flow
```

```
63    %If wb==1, update ELWS, QOT_BR1_T, CWO, TWO
64    %If wb~=1, update ELWS, QOT_BR1_T, CWO, TWO, QIN_BR1, CIN_BR1, TIN_BR1 (CWO &
          ↪ TWO may not update for last reservoir if NARX models aren't provided)
65    x=x_final{wb}(size(x_final{wb},2)-size(t,2)+2:end);
66    if wb==1
67       if isempty(cache)
68          ia=[]; ib=[];
69       else
70          %If using the cache, get list of cache indices here
71          [~,~,tib]=intersect(t,cache.t);
72          if Optimize_day_by_day==0 & size(CFG,2)==1 & ~isempty(cache.x)
73             [ia,ib]=ismember(x,cache.x,'rows');
74          else
75             index=find(cache.t==t(1)); %last index for previous operations
76             [ia,ib]=ismember(x,cache.x(:,index:index+23),'rows'); %fix later to
                   ↪ solve multi waterbody problems
77          end
78          ia=find(ia==1); ib=ib(ib~=0);
79       end
80
81       [turb_discharges{wb},spill_discharges{wb},HWs{wb},~,~] = ...
82          activeunits_to_discharges(x,t,frequency,...
83          Q{wb},ic_elev{wb},turbine_discharge{wb},ELWS_targets{wb},...
84          [],[],Optimize_day_by_day);
85       %Check to see if HWs is cached, and replace if it is
86       if ~isempty(ia)
87          HWs{wb}=cache.HWs(ib,tib);
88       end
89       Q{wb}.ELWS=[Q{wb}.ELWS(Q{wb}.ELWS(:,1)<t(1),:); t' HWs{wb}'];
90       Q{wb}.QOT_BR1_T=[Q{wb}.QOT_BR1_T(Q{wb}.QOT_BR1_T(:,1)<t(1),:);...
91          t' turb_discharges{wb}'];
92       if Optimize_day_by_day==1
93          Q{wb}.QOT_BR1_S=[Q{wb}.QOT_BR1_S(Q{wb}.QOT_BR1_S(:,1)<t(1),:);...
94             t' ones(size(t,2),1)*spill_discharges{wb}];
95       else
96          for ii=1:size(spill_discharges{wb},2)
97             spill_values(1,(1/frequency)*(ii-1)+1:(1/frequency)*(ii)+1)=...
98                spill_discharges{wb}(1,ii);
99          end
100         Q{wb}.QOT_BR1_S=[Q{wb}.QOT_BR1_S(Q{wb}.QOT_BR1_S(:,1)<t(1),:);...
101            t' spill_values'];
102         clearvars ii spill_values
103      end
104      if isempty(ia)
105         DO_pred{wb}=narx_predictions(WQ{wb}.DO_narx,frequency,t,Q{wb},x,...
106            turb_discharges{wb},spill_discharges{wb},[],Q{wb}.CWO,'do',
                   ↪ Optimize_day_by_day);
107         Temp_pred{wb}=narx_predictions(WQ{wb}.Temp_narx,frequency,t,Q{wb},x,...
108            turb_discharges{wb},spill_discharges{wb},[],Q{wb}.TWO,'temp',
                   ↪ Optimize_day_by_day);
109      else
110         DO_pred{wb}=cache.DO(ib,tib(1:end-1)); Temp_pred{wb}=cache.T(ib,tib(1:
               ↪ end-1));
111      end
112      %Remove NaNs from DO_pred and Temp_pred!
113      outgoing_DO{wb}=[t(2:end)' DO_pred{wb}'];
114      outgoing_DO{wb}=outgoing_DO{wb}(~isnan(outgoing_DO{wb}(:,2)),:);
115      outgoing_Temp{wb}=[t(2:end)' Temp_pred{wb}'];
116      outgoing_Temp{wb}=outgoing_Temp{wb}(~isnan(outgoing_Temp{wb}(:,2)),:);
117      %If last values in WQ predictions are NaN, need to add last row to
             ↪ outgoing_DO and outgoing_Temp
118      if outgoing_Temp{wb}(end,1)<t(end)
119         outgoing_Temp{wb}=[outgoing_Temp{wb}; t(end) outgoing_Temp{wb}(end,2)];
120         outgoing_DO{wb}=[outgoing_DO{wb}; t(end) outgoing_DO{wb}(end,2)];
```

```
121         end
122         Q{wb}.CWO=[Q{wb}.CWO(Q{wb}.CWO(:,1)<t(2),:); outgoing_DO{wb}];
123         Q{wb}.TWO=[Q{wb}.TWO(Q{wb}.TWO(:,1)<t(2),:); outgoing_Temp{wb}];
124      else
125         incoming_flow=turb_discharges{wb-1}+spill_discharges{wb-1};
126         [turb_discharges{wb},spill_discharges{wb},HWs{wb},~,~] = ...
127            activeunits_to_discharges(x,t,frequency,...
128            Q{wb},ic_elev{wb},turbine_discharge{wb},ELWS_targets{wb},...
129            t,incoming_flow,Optimize_day_by_day);
130         Q{wb}.ELWS=[Q{wb}.ELWS(Q{wb}.ELWS(:,1)<t(1),:); t' HWs{wb}'];
131         Q{wb}.QOT_BR1_T=[Q{wb}.QOT_BR1_T(Q{wb}.QOT_BR1_T(:,1)<t(1),:);...
132            t' turb_discharges{wb}'];
133         Q{wb}.QOT_BR1_S=[Q{wb}.QOT_BR1_S(Q{wb}.QOT_BR1_S(:,1)<t(1),:);...
134            t' ones(size(t,2),1)*spill_discharges{wb}];
135         %Qin contains both spill and turbine
136         Q{wb}.QIN_BR1=[Q{wb}.QIN_BR1(Q{wb}.QIN_BR1(:,1)<t(1),:);...
137            t' incoming_flow'];
138         Q{wb}.CIN_BR1=[Q{wb}.CIN_BR1(Q{wb}.CIN_BR1(:,1)<t(2),:);...
139            outgoing_DO{wb-1}];
140         Q{wb}.TIN_BR1=[Q{wb}.TIN_BR1(Q{wb}.TIN_BR1(:,1)<t(2),:);...
141            outgoing_Temp{wb-1}];
142         %May not have WQ calculations for final reservoir's discharge (depends on
                ↪ problem definition) so check for these
143         if ~isempty(WQ{wb}.DO_narx)
144            DO_pred{wb}=narx_predictions(WQ{wb}.DO_narx,frequency,t,...
145               Q{wb},x,turb_discharges{wb},spill_discharges{wb},[],Q{wb}.CWO,'do');
146            %Remove NaNs from DO_pred and Temp_pred!
147            outgoing_DO{wb}=[t(2:end)' DO_pred{wb}'];
148            outgoing_DO{wb}=outgoing_DO{wb}(~isnan(outgoing_DO{wb}(:,2)),:);
149            %If last values in WQ predictions are NaN, need to add last row to
                   ↪ outgoing_DO and outgoing_Temp
150            if outgoing_DO{wb}(end,1)<t(end)
151               outgoing_DO{wb}=[outgoing_DO{wb}; t(end) outgoing_DO{wb}(end,2)];
152            end
153            Q{wb}.CWO=[Q{wb}.CWO(Q{wb}.CWO(:,1)<t(1),:); outgoing_DO{wb}];
154         end
155         if ~isempty(WQ{wb}.Temp_narx)
156            Temp_pred{wb}=narx_predictions(WQ{wb}.Temp_narx,frequency,t,...
157               Q{wb},x,turb_discharges{wb},spill_discharges{wb},[],Q{wb}.TWO,'temp'
                      ↪ );
158            %Remove NaNs from DO_pred and Temp_pred!
159            outgoing_Temp{wb}=[t(2:end)' Temp_pred{wb}'];
160            outgoing_Temp{wb}=...
161               outgoing_Temp{wb}(~isnan(outgoing_Temp{wb}(:,2)),:);
162            %If last values in WQ predictions are NaN, need to add last row to
                   ↪ outgoing_DO and outgoing_Temp
163            if outgoing_Temp{wb}(end,1)<t(end)
164               outgoing_Temp{wb}=[outgoing_Temp{wb}; t(end) outgoing_Temp{wb}(end
                      ↪ ,2)];
165            end
166            Q{wb}.TWO=[Q{wb}.TWO(Q{wb}.TWO(:,1)<t(1),:); outgoing_Temp{wb}];
167         end
168      end
169   end
170
171   clearvars outgoing_DO outgoing_Temp
```

REFERENCES

Adams, W. R., E. L. Thackston, and R. E. Speece (1997), Modeling CSO impacts from Nashville using EPA's demonstration approach, *Journal of Environmental Engineering*, *123*(2), 126–133, doi:10.1061/(ASCE)0733-9372(1997)123:2(126).

Afshar, A., H. Kazemi, and M. Saadatpour (2011), Particle swarm optimization for automatic calibration of large scale water quality model (CE-QUAL-W2): Application to Karkheh Reservoir, Iran, *Water Resources Management*, *25*(10), 2613–2632, doi:10.1007/s11269-011-9829-7.

Aguilar, J., S. Van Andel, M. Werner, and D. P. Solomatine (2014), Hydrodynamic and water quality surrogate modeling for reservoir operation, in *11th International Conference on Hydroinformatics*, New York City, New York, Aug 1, 2014.

Ahmed, J. A., and A. K. Sarma (2005), Genetic algorithm for optimal operating policy of a multipurpose reservoir, *Water Resources Management*, *19*(2), 145–161, doi:10.1007/s11269-005-2704-7.

ALGLIB (2014), Inverse distance weighting interpolation/fitting, ⟨http://www.alglib.net/interpolation/inversedistanceweighting.php⟩, accessed January 21 2014.

Alley, W. M. (1986), Regression approximations for transport model constraint sets in combined aquifer simulation-optimization studies, *Water Resources Research*, *22*(4), 581–586, doi:10.1029/Wr022i004p00581.

Aly, A. H., and P. C. Peralta (1999), Comparison of a genetic algorithm and mathematical programming to the design of groundwater cleanup systems, *Water Resources Research*, *35*(8), 2415–2425, doi:10.1029/1998wr900128.

Anderson, M. A. (2010), Influence of pumped-storage hydroelectric plant operation on a shallow polymictic lake: Predictions from 3-D hydrodynamic modeling, *Lake and Reservoir Management*, *26*(1), 1–13, doi:10.1080/10402380903479102.

Andrews, V. L. (2014), FWS 2008-B-0075; Final Biological Opinion on the Wolf Creek Dam/Lake Cumberland Return to Historical Pool Level Operations, Russell County, Kentucky, *Report*, Kentucky Ecological Services Field Office, Fish and Wildlife Service, U.S. Department of the Interior, March 24, 2014.

Annear, R. L., and S. A. Wells (2002), The Bull Run River-Reservoir system model, in *2nd Federal Interagency Hydrologic Modeling Conference*, Las Vegas, Nevada, July 28–August 1, 2002.

Arnold, E., P. Tatjewski, and P. Wolochowicz (1994), Two methods for large-scale nonlinear optimization and their comparison on a case-study of hydropower optimization, *Journal of Optimization Theory and Applications*, *81*(2), 221–248, doi:10.1007/Bf02191662.

Azamathulla, H. M., F. C. Wu, A. Ab Ghani, S. M. Narulkar, N. A. Zakaria, and C. K. Chang (2008), Comparison between genetic algorithm and linear programming approach for real time operation, *Journal of Hydro-Environment Research*, *2*(3), 172–181, doi:10.1016/j.jher.2008.10.001.

Barros, M. T. L., F. T. C. Tsai, S. L. Yang, J. E. G. Lopes, and W. W. G. Yeh (2003), Optimization of large-scale hydropower system operations, *Journal of Water Resources Planning and Management*, *129*(3), 178–188, doi:10.1061/(ASCE)0733-9496(2003)129:3(178).

Bartholow, J., R. B. Hanna, L. Saito, D. Lieberman, and M. Horn (2001), Simulated limnological effects of the Shasta Lake temperature control device, *Environmental Management*, *27*(4), 609–626, doi:10.1007/S0026702324.

Basudhar, A., C. Dribusch, S. Lacaze, and S. Missoum (2012), Constrained efficient global optimization with support vector machines, *Structural and Multidisciplinary Optimization*, *46*(2), 201–221, doi:10.1007/s00158-011-0745-5.

Batick, B. M. (2011), Modeling temperature and dissolved oxygen in the Cheatham Reservoir with CE-QUAL-W2, Masters thesis, Department of Environmental Engineering, Vanderbilt University.

Berger, C. J., and S. A. Wells (2008), Modeling the effects of macrophytes on hydrodynamics, *Journal of Environmental Engineering*, *134*(9), 778–788, doi:10.1061/(ASCE)0733-9372(2008) 134:9(778).

Bichon, B. J., M. S. Eldred, S. Mahadevan, and J. M. McFarland (2013), Efficient global surrogate modeling for reliability-based design optimization, *Journal of Mechanical Design*, *135*(1), doi: 10.1115/1.4022999.

Biddle, S. H. (2001), Optimizing the TVA reservoir system using RiverWare, in *World Water and Environmental Resources Congress*, Orlando, FL, May 20–24, 2001.

Bisschop, J. (2018), *AIMMS Optimization Modelling*, 306 pp.

Blanning, R. W. (1975), Construction and implementation of metamodels, *Simulation*, *24*(6), 177–184, doi:10.1177/003754977502400606.

Bliznyuk, N., D. Ruppert, C. Shoemaker, R. Regis, S. Wild, and P. Mugunthan (2008), Bayesian calibration and uncertainty analysis for computationally expensive models using optimization and radial basis function approximation, *Journal of Computational and Graphical Statistics*, *17*(2), 270–294, doi:10.1198/106186008x320681.

Bloss, S., R. Lehfeldt, and J. C. Patterson (1988), Modeling turbulent transport in stratified estuary, *Journal of Hydraulic Engineering*, *114*(9), 1115–1133.

Blumensaat, F., J. Seydel, P. Krebs, and P. A. Vanrolleghem (2014), Model structure sensitivity of river water quality models for urban drainage impact assessment, in *7th International Congress on Environmental Modelling and Software*, San Diego, CA, June 15–19, 2014.

Bonalumi, M., F. S. Anselmetti, A. Wuest, and M. Schmid (2012), Modeling of temperature and turbidity in a natural lake and a reservoir connected by pumped-storage operations, *Water Resources Research*, *48*(8), doi:10.1029/2012wr011844.

Bos, M. F. M. (2011), The morphological effects of sediment diversions on the Lower Mississippi River, Masters thesis, Department of Hydraulic Engineering, Delft University of Technology, Delft, Netherlands.

Boukouvala, F., and M. G. Ierapetritou (2013), Surrogate-based optimization of expensive flowsheet modeling for continuous pharmaceutical manufacturing, *Journal of Pharmaceutical Innovation*, *8*(2), 131–145, doi:10.1007/s12247-013-9154-1.

Bowen, J. D., and J. W. Hieronymus (2003), A CE-QUAL-W2 model of Neuse Estuary for total maximum daily load development, *Journal of Water Resources Planning and Management*, *129*(4), 283–294, doi:10.1061/(ASCE)0733-9496(2003)129:4(283).

Box, G. E. P., and K. B. Wilson (1951), On the experimental attainment of optimum conditions, *Journal of the Royal Statistical Society Series*, *13*, 1–45.

Broad, D. R., G. C. Dandy, and H. R. Maier (2005), Water distribution system optimization using metamodels, *Journal of Water Resources Planning and Management*, *131*(3), 172–180, doi:10.1061/(ASCE)0733-9496(2005)131:3(172).

Brown, L. C., and T. O. Barnwell (1987), The Enhanced Stream Water Quality Models QUAL2E and QUAL2E-UNCAS: Documentation and User Manual, *Report EPA/600/3-87/007*, Environmental Research Laboratory, Office of Research and Development, U.S. Environmental Protection Agency.

Caliskan, A., and S. Elci (2009), Effects of selective withdrawal on hydrodynamics of a stratified reservoir, *Water Resources Management*, *23*(7), 1257–1273, doi:10.1007/s11269-008-9325-x.

Camp, J. V. S. (2009), Design and implementation of an advanced spill management information system for surface waters, PhD dissertation, Department of Environmental Engineering, Vanderbilt University, Nashville, TN.

Castelletti, A., D. de Rigo, A. E. Rizzoli, R. Soncini-Sessa, and E. Weber (2007), Neuro-dynamic programming for designing water reservoir network management policies, *Control Engineering Practice*, *15*(8), 1031–1038, doi:10.1016/j.conengprac.2006.02.011.

Castelletti, A., F. Pianosi, R. Soncini-Sessa, and J. P. Antenucci (2010), A multiobjective response surface approach for improved water quality planning in lakes and reservoirs, *Water Resources Research*, *46*, doi:10.1029/2009wr008389.

Castelletti, A., S. Galelli, M. Ratto, R. Soncini-Sessa, and P. C. Young (2012), A general framework for dynamic emulation modelling in environmental problems, *Environmental Modelling & Software*, *34*, 5–18, doi:10.1016/j.envsoft.2012.01.002.

Castelletti, A., H. Yajima, M. Giuliani, R. Soncini-Sessa, and E. Weber (2014), Planning the optimal operation of a multioutlet water reservoir with water quality and quantity targets, *Journal of Water Resources Planning and Management*, *140*(4), 496–510, doi:10.1061/(Asce)Wr.1943-5452.0000348.

Center for Advanced Decision Support for Water and Environmental Systems (CADSWES) (2015), RiverWare, ⟨http://cadswes.colorado.edu/creative-works/riverware⟩, accessed October 12 2015.

Chang, L. C., and F. J. Chang (2001), Intelligent control for modelling of real-time reservoir operation, *Hydrological Processes*, *15*(9), 1621–1634, doi:10.1002/Hyp.226.

Chapra, S. C. (1997), *Surface Water-Quality Modeling*, 844 pp., McGraw-Hill, New York.

Chaves, P., and T. Kojiri (2007), Conceptual fuzzy neural network model for water quality simulation, *Hydrological Processes*, *21*(5), 634–646, doi:10.1002/Hyp.6279.

Chen, D., A. S. Leon, N. L. Gibson, and P. Hosseini (2016), Dimension reduction of decision variables for multireservoir operation: A spectral optimization model, *Water Resources Research*, *52*(1), 36–51, doi:10.1002/2015wr017756.

Chen, L. L., C. Liao, W. B. Lin, L. Chang, and X. M. Zhong (2012), Hybrid-surrogate-model-based efficient global optimization for high-dimensional antenna design, *Progress in Electromagnetics Research-Pier*, *124*, 85–100, doi:10.2528/Pier11121203.

Cheng, B., and D. M. Titterington (1994), Neural networks: A review from a statistical perspective, *Statistical Science*, *9*(1), 2–30, doi:10.1214/ss/1177010638.

Cheng, C. T., W. C. Wang, D. M. Xu, and K. W. Chau (2008), Optimizing hydropower reservoir operation using hybrid genetic algorithm and chaos, *Water Resources Management*, *22*(7), 895–909, doi:10.1007/s11269-007-9200-1.

Cheng, Y., Y. Li, F. Ji, and Y. Wang (2018), Global sensitivity analysis of a water quality model in the Three Gorges Reservoir, *Water*, *10*(2), 153, doi:10.3390/w10020153.

Chiu, Y. C., L. C. Chang, and F. J. Chang (2007), Using a hybrid genetic algorithm-simulated annealing algorithm for fuzzy programming of reservoir operation, *Hydrological Processes*, *21*(23), 3162–3172, doi:10.1002/Hyp.6539.

Cho, J. H., and S. R. Ha (2010), Parameter optimization of the QUAL2K model for a multiple-reach river using an influence coefficient algorithm, *Science of the Total Environment*, *408*(8), 1985–1991, doi:10.1016/j.scitotenv.2010.01.025.

Choi, J. H., S. A. Jeong, and S. S. Park (2007), Longitudinal-vertical hydrodynamic and turbidity simulations for prediction of dam reconstruction effects in Asian monsoon area, *Journal of the American Water Resources Association*, *43*(6), 1444–1454, doi:10.1111/j.1752-1688.2007.00120.x.

Chung, S. W., and R. R. Gu (2009), Prediction of the fate and transport processes of atrazine in a reservoir, *Environmental Management*, *44*(1), 46–61, doi:10.1007/s00267-009-9312-x.

Chung, S. W., and J. K. Oh (2006), Calibration of CE-QUAL-W2 for a monomictic reservoir in a monsoon climate area, *Water Science and Technology*, *54*(11-12), 29–37, doi:10.2166/Wst.2006.841.

Cole, T. M., and S. A. Wells (2007), *CE-QUAL-W2: A Two-Dimensional, Laterally Averaged, Hydrodynamic and Water Quality Model, Version 3.5 User Manual*, 681 pp., Department of Civil and Environmental Engineering, Portland State University.

Conn, A. R., N. Gould, and P. L. Toint (1997), A globally convergent Lagrangian barrier algorithm for optimization with general inequality constraints and simple bounds, *Mathematics of Computation*, *66*(217), 261–&, doi:10.1090/S0025-5718-97-00777-1.

Cooper, G. S., R. C. Peralta, and J. J. Kaluarachchi (1998), Optimizing separate phase light hydrocarbon recovery from contaminated unconfined aquifers, *Advances in Water Resources*, *21*(5), 339–350, doi:10.1016/S0309-1708(97)00005-5.

Covich, A. (1993), Water and ecosystems, in *Water in Crisis*, edited by P. Gleick, pp. 40–55, Oxford University Press, New York.

Cox, B. A. (2003), A review of currently available in-stream water-quality models and their applicability for simulating dissolved oxygen in lowland rivers, *Science of the Total Environment*, *314*, 335–377, doi:10.1016/S0048-9697(03)00063-9.

Crawley, P. D., and G. C. Dandy (1993), Optimal operation of multiple-reservoir system, *Journal of Water Resources Planning and Management*, *119*(1), 1–17, doi:10.1061/(ASCE) 0733-9496(1993)119:1(1).

Cristea, N., and G. J. Pelletier (2005), Wenatchee River Temperature Total Maximum Daily Load Study, *Report 05-03-011*, Environmental Assessment Program, Washington State Department of Ecology.

Crowder, D. W., and P. Diplas (2006), Applying spatial hydraulic principles to quantify stream habitat, *River Research and Applications*, *22*(1), 79–89, doi:10.1002/rra.893.

Dai, T. W., and J. W. Labadie (2001), River basin network model for integrated water quantity/quality management, *Journal of Water Resources Planning and Management*, *127*(5), 295–305, doi: 10.1061/(ASCE)0733-9496(2001)127:5(295).

Dariane, A. B., and Z. Farahmandfar (2013), A comparative study of marriage in honey bees optimisation (MBO) algorithm in multi-reservoir system optimisation, *Water SA*, *39*(2), 327–334, doi:10.4314/Wsa.V39i2.17.

de Azevedo, L. G. T., T. K. Gates, D. G. Fontane, J. W. Labadie, and R. L. Porto (2000), Integration of water quantity and quality in strategic river basin planning, *Journal of Water Resources Planning and Management*, *126*(2), 85–97.

Debele, B., R. Srinivasan, and J.-Y. Parlange (2008), Coupling upland watershed and downstream waterbody hydrodynamic and water quality models (SWAT and CE-QUAL-W2) for better water resources management in complex river basins, *Environmental Modeling & Assessment*, *13*(1), 135–153, doi:10.1007/s10666-006-9075-1.

Deliman, P. N., and J. A. Gerald (2002), Application of the two-dimensional hydrothermal and water quality model, CE-QUAL-W2, to the Chesapeake Bay - Conowingo Reservoir, *Lake and Reservoir Management*, *18*(1), 10–19, doi:10.1080/07438140209353925.

Deltares (2015), Delft3D Functional Specifications, *Report Version 2.20*.

Dhar, A., and B. Datta (2008), Optimal operation of reservoirs for downstream water quality control using linked simulation optimization, *Hydrological Processes*, *22*(6), 842–853, doi:10.1002/Hyp. 6651.

di Pierro, F., S. T. Khu, D. Savic, and L. Berardi (2009), Efficient multi-objective optimal design of water distribution networks on a budget of simulations using hybrid algorithms, *Environmental Modelling & Software*, *24*(2), 202–213, doi:10.1016/j.envsoft.2008.06.008.

Dissanayake, D. M. P. K., R. Ranasinghe, and J. A. Roelvink (2012), The morphological response of large tidal inlet/basin systems to relative sea level rise, *Climatic Change*, *113*(2), 253–276, doi:10.1007/s10584-012-0402-z.

Donnell, B. P., J. V. Letter, W. H. McAnally, and W. A. Thomas (2006), Users Guide for RMA2 WES Version 4.5, *Report*, U.S. Army Engineer Research and Development Center, Waterways Experiment Station, Coastal and Hydraulics Laboratory.

Dorigo, M., and T. Stützle (2004), *Ant Colony Optimization*, 305 pp., MIT Press, Cambridge, Mass.

Dortch, M. S. (1997), Water quality considerations in reservoir management, *Journal of Contemporary Water Resources and Education*, pp. 32–42.

Draper, A. J., M. W. Jenkins, K. W. Kirby, J. R. Lund, and R. E. Howitt (2003), Economic-engineering optimization for California water management, *Journal of Water Resources Planning and Management*, *129*(3), 155–164, doi:10.1061/(ASCE)0733-9496(2003)129:3(155).

Draper, A. J., A. Munevar, S. K. Arora, E. Reyes, N. L. Parker, F. I. Chung, and L. E. Peterson (2004), CalSim: Generalized model for reservoir system analysis, *Journal of Water Resources Planning and Management*, *130*(6), 480–489, doi:10.1061/(ASCE)0733-9496(2004)130:6(480).

Eberhart, R. C., and J. Kennedy (1995), A new optimizer using particle swarm theory, in *6th Symposium on Micro Machine and Human Science*, pp. 39–43, IEEE, Nagoya, Japan, October 4-6, 1995.

Edmonds, D. A., and R. L. Slingerland (2008), Stability of delta distributary networks and their bifurcations, *Water Resources Research*, *44*(9), doi:10.1029/2008wr006992.

Ejaz, M. S., and R. C. Peralta (1995), Modeling for optimal management of agricultural and domestic waste-water loading to streams, *Water Resources Research*, *31*(4), 1087–1096, doi:10.1029/94wr02980.

El-Awar, F. A., J. W. Labadie, and T. B. M. J. Ouarda (1998), Stochastic differential dynamic programming for multi-reservoir system control, *Stochastic Hydrology and Hydraulics*, *12*(4), 247–266, doi:10.1007/s004770050020.

El Serafy, G. Y. H., and A. E. Mynett (2008), Improving the operational forecasting system of the stratified flow in Osaka Bay using an ensemble Kalman filter-based steady state Kalman filter, *Water Resources Research*, *44*(6), doi:10.1029/2006wr005412.

Elsayed, K., D. Vucinic, R. d'Ippolito, and C. Lacor (2012), Comparison between RBF and kriging surrogates in design optimization of high dimensional problems, in *3rd International Conference on Engineering Optimization*, Rio de Janeiro, Brazil, July 1–5, 2012.

Esat, V., and M. J. Hall (1994), Water resources system optimization using genetic algorithms, in *First International Conference on Hydroinformatics*, vol. 1, pp. 225–231, Delft, Netherlands, September 19–23, 1994.

Eschenbach, E. A., T. H. Magee, E. Zagona, M. Goranflo, and R. Shane (2001), Goal programming decision support system for multiobjective operation of reservoir systems, *Journal of Water Resources Planning and Management*, *127*(2), 108–120, doi:10.1061/(ASCE)0733-9496(2001)127:2(108).

Eslick, J. C., B. Ng, Q. W. Gao, C. H. Tong, N. V. Sahinidis, and D. C. Miller (2014), A framework for optimization and quantification of uncertainty and sensitivity for developing carbon capture systems, *Energy Procedia*, *63*, 1055–1063, doi:10.1016/j.egypro.2014.11.113.

Faber, B. A., and J. J. Harou (2006), Multiobjective optimization with HEC Res-PRM - Application to the Upper Mississippi Reservoir System, in *Operating Reservoirs in Changing Conditions: Proceedings of the Operations Management 2006 Conference*, pp. 215–224, American Society of Civil Engineers, Sacramento, CA, August 14-16, 2006.

Fasshauer, G. E. (2007), *Meshfree Approximation Methods with MATLAB*, Interdisciplinary Mathematical Sciences, 500 pp., World Scientific, Singapore; Hackensack, N.J.

Fen, C. S., C. C. Chan, and H. C. Cheng (2009), Assessing a response surface-based optimization approach for soil vapor extraction system design, *Journal of Water Resources Planning and Management*, *135*(3), 198–207, doi:10.1061/(ASCE)0733-9496(2009)135:3(198).

Ferreira, A. R., and R. S. V. Teegavarapu (2012), Optimal and adaptive operation of a hydropower system with unit commitment and water quality constraints, *Water Resources Management*, *26*(3), 707–732, doi:10.1007/s11269-011-9940-9.

Finardi, E. C., and M. R. Scuzziato (2013), Hydro unit commitment and loading problem for day-ahead operation planning problem, *International Journal of Electrical Power & Energy Systems*, *44*(1), 7–16, doi:10.1016/j.ijepes.2012.07.023.

Finardi, E. C., E. L. da Silva, and C. Sagastizabal (2005), Solving the unit commitment problem of hydropower plants via lagrangian relaxation and sequential quadratic programming, *Computational & Applied Mathematics*, *24*(3), 317–341.

Fontane, D. G., T. K. Gates, and E. Moncada (1997), Planning reservoir operations with imprecise objectives, *Journal of Water Resources Planning and Management*, *123*(3), 154–162, doi:10.1061/(ASCE)0733-9496(1997)123:3(154).

Forrester, A. I. J., and A. J. Keane (2009), Recent advances in surrogate-based optimization, *Progress in Aerospace Sciences*, *45*(1-3), 50–79, doi:10.1016/j.paerosci.2008.11.001.

Forrester, A. I. J., A. Sóbester, and A. J. Keane (2008), *Engineering Design via Surrogate Modelling: A Practical Guide*, 210 pp., J. Wiley, Chichester, West Sussex, England; Hoboken, NJ.

Franke, R., and G. Nielson (1980), Smooth interpolation of large sets of scattered data, *International Journal for Numerical Methods in Engineering*, *15*(11), 1691–1704, doi:10.1002/nme.1620151110.

Friedl, G., and A. Wuest (2002), Disrupting biogeochemical cycles - Consequences of damming, *Aquatic Sciences*, *64*(1), 55–65, doi:10.1007/S00027-002-8054-0.

Garvey, E., J. E. Tobiason, M. Hayes, E. Wolfram, D. A. Reckhow, and J. W. Male (1998), Coliform transport in a pristine reservoir: Modeling and field studies, *Water Science and Technology*, *37*(2), 137–144, doi:10.1016/S0273-1223(98)00048-1.

Gastelum, J. R., and C. Cullom (2013), Application of the Colorado River Simulation System model to evaluate water shortage conditions in the Central Arizona Project, *Water Resources Management*, *27*(7), 2369–2389, doi:10.1007/s11269-013-0292-5.

Georgakakos, A. P., H. M. Yao, and Y. Q. Yu (1997), Control models for hydroelectric energy optimization, *Water Resources Research*, *33*(10), 2367–2379, doi:10.1029/97wr01714.

Gorelick, S. M., C. I. Voss, P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright (1984), Aquifer reclamation design: The use of contaminant transport simulation combined with nonlinear-programming, *Water Resources Research*, *20*(4), 415–427, doi:10.1029/Wr020i004p00415.

Gotshall, S., B. Rylander, V. Esat, and M. J. Hall (2002), Optimal population size and the genetic algorithm, in *World Scientific and Engineering Academy and Society (WSEAS) International Conference on Soft Computing, Optimization, Simulation, and Manufacturing Systems (SOSM 2002)*, Cancun, Mexico, May 12–16, 2002.

Graf, W. L. (2005), Geomorphology and American dams: The scientific, social, and economic context, *Geomorphology*, *71*(1-2), 3–26, doi:10.1016/j.geomorph.2004.05.005.

Gramacy, R. B., M. Taddy, and S. M. Wild (2013), Variable selection and sensitivity analysis using dynamic trees, with an application to computer code performance tuning, *Annals of Applied Statistics*, *7*(1), 51–80, doi:10.1214/12-Aoas590.

Grefenstette, J. J. (1992), Genetic algorithms for changing environments, *Parallel Problem Solving from Nature 2*, pp. 137–144.

Grenney, W. J., M. C. Teuscher, and L. S. Dixon (1978), Characteristics of the solution algorithms for the QUAL II river model, *Journal (Water Pollution Control Federation)*, *50*(1), 151–157.

Grygier, J. C., and J. R. Stedinger (1985), Algorithms for optimizing hydropower system operation, *Water Resources Research*, *21*(1), 1–10, doi:10.1029/Wr021i001p00001.

Gunduz, O., S. Soyupak, and C. Yurteri (1998), Development of water quality management strategies for the proposed Isikli reservoir, *Water Science and Technology*, *37*(2), 369–376, doi:10.1016/S0273-1223(98)00045-6.

Gutmann, H. M. (2001), A radial basis function method for global optimization, *Journal of Global Optimization*, *19*(3), 201–227, doi:10.1023/A:1011255519438.

Haddad, O. B., A. Afshar, and M. A. Marino (2006), Honey-bees mating optimization (HBMO) algorithm: A new heuristic approach for water resources optimization, *Water Resources Management*, *20*(5), 661–680, doi:10.1007/s11269-005-9001-3.

Hall, W. A., W. S. Butcher, and A. Esogbue (1968), Optimization of operation of a multiple-purpose reservoir by dynamic programming, *Water Resources Research*, *4*(3), 471–477, doi:10.1029/Wr004i003p00471.

Hamrick, J. M. (1996), User's Manual for the Environmental Fluid Dynamics Computer Code, *Report No. 331*, Department of Physical Sciences, School of Marine Science, Virginia Institute of Marine Science, The College of William and Mary.

Hayes, D. F., J. W. Labadie, T. G. Sanders, and J. K. Brown (1998), Enhancing water quality in hydropower system operations, *Water Resources Research*, *34*(3), 471–483, doi:10.1029/97wr03038.

Hegazy, T., and R. Rashedi (2013), Large-scale asset renewal optimization using genetic algorithms plus segmentation, *Journal of Computing in Civil Engineering*, *27*(4), 419–426, doi:10.1061/(Asce)Cp.1943-5487.0000249.

Henderson-Sellers, B. (1988), Sensitivity of thermal stratification models to changing boundary conditions, *Applied Mathematical Modelling*, *12*(1), 31–43, doi:10.1016/0307-904x(88)90021-2.

Hiew, K. (1987), Optimization algorithms for large-scale multireservoir hydropower systems, PhD dissertation, Department of Civil Engineering, Colorado State University, Fort Collins, CO.

Higgins, J. M., and W. G. Brock (1999), Overview of reservoir release improvements at 20 TVA dams, *Journal of Energy Engineering*, *125*(1), 1–17, doi:10.1061/(ASCE)0733-9402(1999)125:1(1).

Holland, J. H. (1975), *Adaptation in Natural and Artificial Systems : An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*, 183 pp., University of Michigan Press, Ann Arbor.

Horowitz, B., L. J. D. Guimaraes, V. Dantas, and S. M. B. Afonso (2010), A concurrent efficient global optimization algorithm applied to polymer injection strategies, *Journal of Petroleum Science and Engineering*, *71*(3-4), 195–204, doi:10.1016/j.petrol.2010.02.002.

Huang, D., T. T. Allen, W. I. Notz, and R. A. Miller (2006), Sequential kriging optimization using multiple-fidelity evaluations, *Structural and Multidisciplinary Optimization*, *32*(5), 369–382, doi:10.1007/s00158-005-0587-0.

Huang, Y. T., and L. Liu (2010), Multiobjective water quality model calibration using a hybrid genetic algorithm and neural network-based approach, *Journal of Environmental Engineering*, *136*(10), 1020–1031, doi:10.1061/(ASCE)Ee.1943-7870.0000237.

Ilich, N., and S. P. Simonovic (2001), An evolution program for non-linear transportation problems, *Journal of Heuristics*, *7*(2), 145–168, doi:10.1023/A:1009609820093.

Jager, H. I., and B. T. Smith (2008), Sustainable reservoir operation: Can we generate hydropower and preserve ecosystem values?, *River Research and Applications*, *24*(3), 340–352, doi:10.1002/Rra.1069.

Jalali, M. R., A. Afshar, and M. A. Marino (2007), Multi-colony ant algorithm for continuous multi-reservoir operation optimization problem, *Water Resources Management*, *21*(9), 1429–1447, doi:10.1007/s11269-006-9092-5.

James, R. T., J. Martin, T. Wool, and P. F. Wang (1997), A sediment resuspension and water quality model of Lake Okeechobee, *Journal of the American Water Resources Association*, *33*(3), 661–680, doi:10.1111/j.1752-1688.1997.tb03540.x.

Jenkins, M. W., J. R. Lund, R. E. Howitt, A. J. Draper, S. M. Msangi, S. K. Tanaka, R. S. Ritzema, and G. F. Marques (2004), Optimization of California's water supply system: Results and insights, *Journal of Water Resources Planning and Management*, *130*(4), 271–280, doi:10.1061/(ASCE)0733-9496(2004)130:4(271).

Ji, Z. G., J. H. Hamrick, and J. Pagenkopf (2002), Sediment and metals modeling in shallow river, *Journal of Environmental Engineering*, *128*(2), 105–119, doi:10.1061/(Asce)0733-9372(2002)128:2(105).

Jin, K. R., Z. G. Ji, and J. H. Hamrick (2002a), Modeling winter circulation in Lake Okeechobee, Florida, *Journal of Waterway Port Coastal and Ocean Engineering*, *128*(3), 114–125, doi:10.1061/(Asce)0733-950x(2002)128;3(114).

Jin, Y. (2005), A comprehensive survey of fitness approximation in evolutionary computation, *Soft Computing*, *9*(1), 3–12, doi:10.1007/s00500-003-0328-5.

Jin, Y. C., M. Olhofer, and B. Sendhoff (2002b), A framework for evolutionary optimization with approximate fitness functions, *IEEE Transactions on Evolutionary Computation*, *6*(5), 481–494, doi:10.1109/Tevc.2002.800884.

Johnson, V. M., and L. L. Rogers (2000), Accuracy of neural network approximators in simulation-optimization, *Journal of Water Resources Planning and Management*, *126*(2), 48–56, doi:10.1061/(ASCE)0733-9496(2000)126:2(48).

Jones, D. R. (2001), A taxonomy of global optimization methods based on response surfaces, *Journal of Global Optimization*, *21*(4), 345–383, doi:10.1023/A:1012771025575.

Jones, D. R., M. Schonlau, and W. J. Welch (1998), Efficient global optimization of expensive black-box functions, *Journal of Global Optimization*, *13*(4), 455–492, doi:10.1023/A:1008306431147.

Kacikoc, M., and M. Beyhan (2014), Hydrodynamic and water quality modeling of Lake Egirdir, *Clean-Soil Air Water*, *42*(11), 1573–1582, doi:10.1002/clen.201300455.

Kannel, P. R., S. Lee, Y. S. Lee, S. R. Kanel, and G. J. Pelletier (2007), Application of automated QUAL2Kw for water quality modeling and management in the Bagmati River, Nepal, *Ecological Modelling*, *202*(3-4), 503–517, doi:10.1016/j.ecolmodel.2006.12.033.

Kannel, P. R., S. R. Kanel, S. Lee, Y. S. Lee, and T. Y. Gan (2011), A review of public domain water quality models for simulating dissolved oxygen in rivers and streams, *Environmental Modeling & Assessment*, *16*(2), 183–204, doi:10.1007/s10666-010-9235-1.

Karamouz, M., B. Zahraie, and S. Araghinejad (2005), Decision support system for monthly operation of hydropower reservoirs: A case study, *Journal of Computing in Civil Engineering*, *19*(2), 194–207, doi:10.1061/(ASCE)0887-3801(2005)19:2(194).

Kennedy, R. H., and R. F. Gaugush (1988), Assessment of water quality in Corps of Engineers reservoirs, *Lake and Reservoir Management*, *4*(2), 253–260.

Kerachian, R., and M. Karamouz (2007), A stochastic conflict resolution model for water quality management in reservoir-river systems, *Advances in Water Resources*, *30*(4), 866–882, doi:10.1016/j.advwatres.2006.07.005.

Khu, S. T., and M. G. F. Werner (2003), Reduction of Monte-Carlo simulation runs for uncertainty estimation in hydrological modelling, *Hydrology and Earth System Sciences*, *7*(5), 680–692.

Khu, S. T., D. Savic, Y. Liu, and H. Madsen (2004), A fast evolutionary-based meta-modelling approach for the calibration of a rainfall-runoff model, in *First Biennial Meeting of the International Environmental Modelling Software Society*, Osnabruck, Germany.

Kirkpatrick, S., C. D. Gelatt, and M. P. Vecchi (1983), Optimization by simulated annealing, *Science*, *220*(4598), 671–680, doi:10.1126/science.220.4598.671.

Knowles, J. (2006), Parego: A hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems, *IEEE Transactions on Evolutionary Computation*, *10*(1), 50–66, doi:10.1109/Tevc.2005.851274.

Krige, D. G. (1951), A statistical approach to some basic mine valuation problems on the witwatersrand, *Journal of the Chemical, Metallurgical and Mining Engineering Society of South Africa*, *52*(6), 119–139.

Kumar, D. N., and M. J. Reddy (2006), Ant colony optimization for multi-purpose reservoir operation, *Water Resources Management*, *20*(6), 879–898, doi:10.1007/s11269-005-9012-0.

Kumar, D. N., and M. J. Reddy (2007), Multipurpose reservoir operation using particle swarm optimization, *Journal of Water Resources Planning and Management*, *133*(3), 192–201, doi:10.1061/(ASCE)0733-9496(2007)133:3(192).

Kuo, J. T., W. C. Liu, R. T. Lin, W. S. Lung, M. D. Yang, C. P. Yang, and S. C. Chu (2003), Water quality modeling for the Feitsui Reservoir in northern Taiwan, *Journal of the American Water Resources Association*, *39*(3), 671–687, doi:10.1111/j.1752-1688.2003.tb03684.x.

Kuo, J. T., W. S. Lung, C. P. Yang, W. C. Liu, M. D. Yang, and T. S. Tang (2006), Eutrophication modelling of reservoirs in Taiwan, *Environmental Modelling & Software*, *21*(6), 829–844, doi:10.1016/j.envsoft.2005.03.006.

Kurup, R. G., D. P. Hamilton, and R. L. Phillips (2000), Comparison of two 2-dimensional, laterally averaged hydrodynamic model applications to the Swan River Estuary, *Mathematics and Computers in Simulation*, *51*(6), 627–638, doi:10.1016/S0378-4754(99)00146-9.

Labadie, J. W. (2004), Optimal operation of multireservoir systems: State-of-the-art review, *Journal of Water Resources Planning and Management*, *130*(2), 93–111, doi:10.1061/(ASCE)0733-9496(2004)130:2(93).

Labadie, J. W., and R. Larson (2007), *MODSIM 8.1: River Basin Management Decision Support System: User Manual and Documentation*, 123 pp., Department of Civil and Environmental Engineering, Colorado State University, Fort Collins, CO.

Lee, C., and G. Foster (2013), Assessing the potential of reservoir outflow management to reduce sedimentation using continuous turbidity monitoring and reservoir modelling, *Hydrological Processes*, *27*(10), 1426–1439, doi:10.1002/Hyp.9284.

Lee, J. H. W., and B. Qu (2004), Hydrodynamic tracking of the massive spring 1998 red tide in Hong Kong, *Journal of Environmental Engineering*, *130*(5), 535–550, doi:10.1061/(Asce)0733-9372(2004)130:5(535).

Lee, Y., S.-K. Kim, and I. H. Ko (2006), Two-stage stochastic linear programming model for coordinated multi-reservoir operation, in *Operations Management Conference 2006*, ASCE, Sacramento, California, August 14–16, 2006.

Lefkoff, L. J., and S. M. Gorelick (1990), Simulating physical processes and economic-behavior in saline, irrigated agriculture - model development, *Water Resources Research*, *26*(7), 1359–1369, doi:10.1029/Wr026i007p01359.

Letter, J. V., G. L. Brown, and B. P. Donnell (2011), Users Guide for RMA4 WES Version 4.5, *Report*, U.S. Army Engineer Research and Development Center, Waterways Experiment Station, Coastal and Hydraulics Laboratory.

Li, X. G., and X. Wei (2008), An improved genetic algorithm-simulated annealing hybrid algorithm for the optimization of multiple reservoirs, *Water Resources Management*, *22*(8), 1031–1049, doi:10.1007/s11269-007-9209-5.

Lin, T. N., B. G. Horne, P. Tino, and C. L. Giles (1996), Learning long-term dependencies in NARX recurrent neural networks, *IEEE Transactions on Neural Networks*, *7*(6), 1329–1338.

Liong, S. Y., S. T. Khu, and W. T. Chan (2001), Derivation of Pareto front with genetic algorithm and neural network, *Journal of Hydrologic Engineering*, *6*(1), 52–61, doi:10.1061/(ASCE)1084-0699(2001)6:1(52).

Liu, X. Y., S. L. Guo, P. Liu, L. Chen, and X. A. Li (2011), Deriving optimal refill rules for multi-purpose reservoir operation, *Water Resources Management*, *25*(2), 431–448, doi:10.1007/s11269-010-9707-8.

Loftis, B., J. W. Labadie, and D. G. Fontane (1985), Optimal operation of a system of lakes for quality and quantity, in *Computer Applications in Water Resources*, pp. 693–702, Buffalo, New York, June 10–12, 1985.

Lund, J. R., and I. Ferreira (1996), Operating rule optimization for Missouri River reservoir system, *Journal of Water Resources Planning and Management*, *122*(4), 287–295, doi:10.1061/(ASCE)0733-9496(1996)122:4(287).

Lung, W. S., and S. Bai (2003), A water quality model for the Patuxent Estuary: Current conditions and predictions under changing land-use scenarios, *Estuaries*, *26*(2A), 267–279, doi:10.1007/Bf02695966.

MacQueen, J. (1967), Some methods for classification and analysis of multivariate observations, in *Fifth Berkeley Symposium on Mathematical Statistics and Probability*, vol. 1, pp. 281–297, University of California Press, Berkeley, CA, June 21–July 18, 1965.

Madadgar, S., and A. Afshar (2009), An improved continuous ant algorithm for optimization of water resources problems, *Water Resources Management*, *23*(10), 2119–2139, doi:10.1007/s11269-008-9373-2.

Magee, T. H. (2015), System optimization of operations with TDG, in *Enhancing Water Quality in Hydropower System Operations: A Workshop to Review Advances, Challenges, and Identify Opportunities*, Nashville, TN, August 19, 2015.

Martin, J. L. (1988), Application of two-dimensional water-quality model, *Journal of Environmental Engineering*, *114*(2), 317–336, doi:10.1061/(ASCE)0733-9372(1988)114:2(317).

Martin, Q. W. (1983), Optimal operation of multiple reservoir systems, *Journal of Water Resources Planning and Management*, *109*(1), 58–74.

Martin, Q. W. (1995), Optimal reservoir control for hydropower on Colorado River, Texas, *Journal of Water Resources Planning and Management*, *121*(6), 438–446, doi:10.1061/(Asce)0733-9496(1995)121:6(438).

McCartney, M. (2009), Living with dams: Managing the environmental impacts, *Water Policy*, *11*(1), 121–139, doi:10.2166/Wp.2009.108.

Micchelli, C. A. (1986), Interpolation of scattered data - distance matrices and conditionally positive definite functions, *Constructive Approximation*, *2*(1), 11–22, doi:10.1007/Bf01893414.

Mobley, M. H., and W. G. Brock (1995), Widespread oxygen bubbles to improve reservoir releases, *Lake and Reservoir Management*, *11*(2), 231–234, doi:10.1080/07438149509354204.

Mooij, W. M., D. Trolle, E. Jeppesen, G. Arhonditsis, P. V. Belolipetsky, D. B. R. Chitamwebwa, A. G. Degermendzhy, D. L. DeAngelis, L. N. D. Domis, A. S. Downing, J. A. Elliott, C. R. Fragoso, U. Gaedke, S. N. Genova, R. D. Gulati, L. Hakanson, D. P. Hamilton, M. R. Hipsey, J. 't Hoen, S. Hulsmann, F. H. Los, V. Makler-Pick, T. Petzoldt, I. G. Prokopkin, K. Rinke, S. A. Schep, K. Tominaga, A. A. Van Dam, E. H. Van Nes, S. A. Wells, and J. H. Janse (2010), Challenges and opportunities for integrating lake ecosystem modelling approaches, *Aquatic Ecology*, *44*(3), 633–667, doi:10.1007/s10452-010-9339-3.

Mousavi, S. J., K. S. Moghaddam, and A. Seifi (2004), Application of an interior-point algorithm for optimization of a large-scale reservoir system, *Water Resources Management*, *18*(6), 519–540, doi:10.1007/s11269-004-1075-9.

Mugunthan, P., and C. A. Shoemaker (2006), Assessing the impacts of parameter uncertainty for computationally expensive groundwater models, *Water Resources Research*, *42*(10), doi:10.1029/2005wr004640.

Mugunthan, P., C. A. Shoemaker, and R. G. Regis (2005), Comparison of function approximation, heuristic, and derivative-based methods for automatic calibration of computationally expensive groundwater bioremediation models, *Water Resources Research*, *41*(11), 1–17, doi:10.1029/2005wr004134.

Muñoz-Carpena, R., Z. Zajac, and Y. M. Kuo (2007), Global sensitivity and uncertainty analyses of the water quality model VFSMOD-W, *Transactions of the American Society of Agricultural and Biological Engineers*, *50*(5), 1719–1732.

Naresh, R., and J. Sharma (2002), Short term hydro scheduling using two-phase neural network, *International Journal of Electrical Power & Energy Systems*, *24*(7), 583–590, doi:10.1016/S0142-0615(01)00069-2.

Needham, J. T., D. W. Watkins, J. R. Lund, and S. K. Nanda (2000), Linear programming for flood control in the Iowa and Des Moines Rivers, *Journal of Water Resources Planning and Management*, *126*(3), 118–127, doi:10.1061/(ASCE)0733-9496(2000)126:3(118).

Neelakantan, T. R., and N. V. Pundarikanthan (1999), Hedging rule optimisation for water supply reservoirs system, *Water Resources Management*, *13*(6), 409–426, doi:10.1023/A:1008157316584.

Nestler, J. M., R. A. Goodwin, T. M. Cole, D. Degan, and D. Dennerline (2002), Simulating movement patterns of blueback herring in a stratified southern impoundment, *Transactions of the American Fisheries Society*, *131*(1), 55–69, doi:10.1577/1548-8659(2002)131⟨0055:Smpobh⟩2.0.Co;2.

Nikolaidis, N. P., A. P. Karageorgis, V. Kapsimalis, G. Marconis, P. Drakopoulou, H. Kontoyiannis, E. Krasakopoulou, A. Pavlidou, and K. Pagou (2006), Circulation and nutrient modeling of Thermaikos Gulf, Greece, *Journal of Marine Systems*, *60*(1-2), 51–62, doi:10.1016/j.jmarsys.2005.11.007.

Norton, G. E., and A. Bradford (2009), Comparison of two stream temperature models and evaluation of potential management alternatives for the Speed River, Southern Ontario, *Journal of Environmental Management*, *90*(2), 866–878, doi:10.1016/j.jenvman.2008.02.002.

O'Connor, D. J. (1960), Oxygen balance of an estuary, *Journal of the Sanitary Engineering Division*, *86*(SA3), 35–55.

O'Hagan, A. (2006), Bayesian analysis of computer code outputs: A tutorial, *Reliability Engineering & System Safety*, *91*(10-11), 1290–1300, doi:10.1016/j.ress.2005.11.025.

Oliveira, R., and D. P. Loucks (1997), Operating rules for multireservoir systems, *Water Resources Research*, *33*(4), 839–852, doi:10.1029/96wr03745.

Ostfeld, A., and S. Salomons (2005), A hybrid genetic-instance based learning algorithm for CE-QUAL-W2 calibration, *Journal of Hydrology*, *310*(1-4), 122–142, doi:10.1016/j.jhydrol.2004.12.004.

Padula, S. L., C. R. Gumbert, and W. Li (2006), Aerospace applications of optimization under uncertainty, *Optimization and Engineering*, *7*(3), 317–328, doi:10.1007/s11081-006-9974-7.

Park, J. Y., and S. J. Kim (2014), Potential impacts of climate change on the reliability of water and hydropower supply from a multipurpose dam in South Korea, *Journal of the American Water Resources Association*, *50*(5), 1273–1288, doi:10.1111/jawr.12190.

Park, S. S., and Y. S. Lee (2002), A water quality modeling study of the Nakdong River, Korea, *Ecological Modelling*, *152*(1), 65–75, doi:10.1016/S0304-3800(01)00489-6.

Piman, T., T. A. Cochrane, M. E. Arias, A. Green, and N. D. Dat (2013), Assessment of flow changes from hydropower development and operations in Sekong, Sesan, and Srepok Rivers of the Mekong Basin, *Journal of Water Resources Planning and Management*, *139*(6), 723–732, doi:10.1061/(Asce)Wr.1943-5452.0000286.

Poff, N. L., and J. C. Schmidt (2016), How dams can go with the flow, *Science*, *353*(6304), 1099–1100, doi:10.1126/science.aah4926.

Ponnambalam, K., A. Vannelli, and T. E. Unny (1989), An application of Karmarkar's interior-point linear-programming algorithm for multi-reservoir operations optimization, *Stochastic Hydrology and Hydraulics*, *3*(1), 17–29, doi:10.1007/Bf01543425.

Ponweiser, W., T. Wagner, D. Biermann, and M. Vincze (2008), Multiobjective optimization on a limited budget of evaluations using model-assisted s-metric selection, in *Parallel Problem Solving from Nature PPSN X*, *Lecture Notes in Computer Science*, vol. 5199, edited by G. Rudolph, T. Jansen, S. Lucas, C. Poloni, and N. Beume, book section 78, pp. 784–794, Springer Berlin Heidelberg, doi:10.1007/978-3-540-87700-4_78.

Portland State University (2007), CE-QUAL-W2 Hydrodynamic and Water Quality Model, Application by Country, Water Quality Research Group, ⟨http://www.ce.pdx.edu/w2/applications_new.html⟩, accessed December 12 2013.

Price, R. E., and E. B. Meyer (1992), Water Quality Management for Reservoirs and Tailwaters: Report 2, Operational and Structural Water Quality Techniques, *Report E-89-1*, U.S. Army Engineer Waterways Experiment Station, Vicksburg, MS.

Queipo, N. V., A. J. Verde, J. Canelon, and S. Pintos (2002), Efficient global optimization for hydraulic fracturing treatment design, *Journal of Petroleum Science and Engineering*, *35*(3-4), 151–166, doi:10.1016/S0920-4105(02)00237-1.

Queipo, N. V., R. T. Haftka, W. Shyy, T. Goel, R. Vaidyanathan, and P. K. Tucker (2005), Surrogate-based analysis and optimization, *Progress in Aerospace Sciences*, *41*(1), 1–28, doi:10.1016/j.paerosci.2005.02.001.

Raman, H., and V. Chandramouli (1996), Deriving a general operating policy for reservoirs using neural network, *Journal of Water Resources Planning and Management*, *122*(5), 342–347, doi:10.1061/(ASCE)0733-9496(1996)122:5(342).

Rangel-Peraza, J. G., J. De Anda, F. A. Gonzalez-Farias, and M. Rode (2016), Sensitivity and uncertainty analysis on water quality modelling of Aguamilpa reservoir, *Journal of Limnology*, *75*, 81–92, doi:10.4081/jlimnol.2016.1391.

Rani, D., and M. M. Moreira (2010), Simulation-optimization modeling: A survey and potential application in reservoir systems operation, *Water Resources Management*, *24*(6), 1107–1138, doi:10.1007/s11269-009-9488-0.

Razavi, S., B. A. Tolson, and D. H. Burn (2012a), Review of surrogate modeling in water resources, *Water Resources Research*, *48*(7), doi:10.1029/2011wr011527.

Razavi, S., B. A. Tolson, and D. H. Burn (2012b), Numerical assessment of metamodelling strategies in computationally intensive optimization, *Environmental Modelling & Software*, *34*, 67–86, doi:10.1016/j.envsoft.2011.09.010.

Reed, P., B. Minsker, and D. E. Goldberg (2000), Designing a competent simple genetic algorithm for search and optimization, *Water Resources Research*, *36*(12), 3757–3761, doi:10.1029/2000wr900231.

Regis, R. G., and C. A. Shoemaker (2004), Local function approximation in evolutionary algorithms for the optimization of costly functions, *IEEE Transactions on Evolutionary Computation*, *8*(5), 490–505, doi:10.1109/Tevc.2004.835247.

Reichert, P., and P. Vanrolleghem (2001), Identifiability and uncertainty analysis of the River Water Quality Model No. 1 (RWQM1), *Water Science and Technology*, *43*(7), 329–338.

Reis, J., T. B. Culver, M. McCartney, J. Lautze, and S. Kibret (2011), Water resources implications of integrating malaria control into the operation of an Ethiopian dam, *Water Resources Research*, *47*, doi:10.1029/2010wr010166.

Renka, R. J. (1988), Multivariate interpolation of large sets of scattered data, *ACM Transactions on Mathematical Software*, *14*(2), 139–148, doi:10.1145/45054.45055.

Rizzo, D. M., and D. E. Dougherty (1994), Characterization of aquifer properties using artificial neural networks - neural kriging, *Water Resources Research*, *30*(2), 483–497, doi:10.1029/93wr02477.

Rygwelski, K. R., W. L. Richardson, and D. D. Endicott (1999), A screening-level model evaluation of atrazine in the Lake Michigan basin, *Journal of Great Lakes Research*, *25*(1), 94–106.

Saad, M., A. Turgeon, P. Bigras, and R. Duquette (1994), Learning disaggregation technique for the operation of long-term hydroelectric power-systems, *Water Resources Research*, *30*(11), 3195–3202, doi:10.1029/94wr01731.

Saad, M., P. Bigras, A. Turgeon, and R. Duquette (1996), Fuzzy learning decomposition for the scheduling of hydroelectric power systems, *Water Resources Research*, *32*(1), 179–186, doi:10.1029/95wr02971.

Saadatpour, M., and A. Afshar (2013), Multi objective simulation-optimization approach in pollution spill response management model in reservoirs, *Water Resources Management*, *27*(6), 1851–1865, doi:10.1007/s11269-012-0230-y.

Saito, L., B. M. Johnson, J. Bartholow, and R. B. Hanna (2001), Assessing ecosystem effects of reservoir operations using food web-energy transfer and water quality models, *Ecosystems*, *4*(2), 105–125, doi:10.1007/s100210000062.

Sale, M. J., E. D. Brill, and E. E. Herricks (1982), An approach to optimizing reservoir operation for downstream aquatic resources, *Water Resources Research*, *18*(4), 705–712, doi:10.1029/Wr018i004p00705.

Saloranta, T. M. (2006), Highlighting the model code selection and application process in policy-relevant water quality modelling, *Ecological Modelling*, *194*(1-3), 316–327, doi:10.1016/j.ecolmodel.2005.10.031.

Seifi, A., and K. W. Hipel (2001), Interior-point method for reservoir operation with stochastic inflows, *Journal of Water Resources Planning and Management*, *127*(1), 48–57, doi:10.1061/(ASCE)0733-9496(2001)127:1(48).

Shaw, A. R., H. Smith Sawyer, E. J. LeBoeuf, and M. P. McDonald (2013), Surrogate model development for a large-scale controlled reservoir system, in *Hydrovision International*, Denver, CO, July 23–26, 2013.

Shaw, A. R., H. Smith Sawyer, E. J. LeBoeuf, and M. P. McDonald (2015), High-fidelity reservoir water quality model emulation by artificial neural network, in *Hydrovision International*, Portland, OR, July 14–17, 2015.

Shaw, A. R., H. Smith Sawyer, E. J. LeBoeuf, and M. P. McDonald (2016), Generation optimization for linked riverine reservoir systems with constraints on water quality, in *Hydrovision International*, Minneapolis, MN, July 26–29, 2016.

Shaw, A. R., H. S. Sawyer, E. J. LeBoeuf, M. P. McDonald, and B. Hadjerioua (2017), Hydropower optimization using artificial neural network surrogate models of a high-fidelity hydrodynamics and water quality model, *Water Resources Research*, *53*(11), 9444–9461, doi:10.1002/2017wr021039.

Shepard, D. (1968), A two-dimensional interpolation function for irregularly-spaced data, in *23rd ACM National Conference (ACM '68)*, pp. 517–524.

Shinkyu, J., and S. Obayashi (2005), Efficient global optimization (EGO) for multi-objective problem and data mining, in *2005 IEEE Congress on Evolutionary Computation*, vol. 3, pp. 2138–2145, doi:10.1109/CEC.2005.1554959, September 2–5, 2005.

Shirangi, E., R. Kerachian, and M. S. Bajestan (2008), A simplified model for reservoir operation considering the water quality issues: Application of the Young conflict resolution theory, *Environmental Monitoring and Assessment*, *146*(1-3), 77–89, doi:10.1007/s10661-007-0061-0.

Shoemaker, C. A., R. G. Regis, and R. C. Fleming (2007), Watershed calibration using multistart local optimization and evolutionary optimization with radial basis function approximation, *Hydrological Sciences Journal-Journal Des Sciences Hydrologiques*, *52*(3), 450–465, doi: 10.1623/hysj.52.3.450.

Shourian, M., S. J. Mousavi, and A. Tahershamsi (2008), Basin-wide water resources planning by integrating PSO algorithm and MODSIM, *Water Resources Management*, *22*(10), 1347–1366, doi:10.1007/s11269-007-9229-1.

Shrestha, D. L., N. Kayastha, and D. P. Solomatine (2009), A novel approach to parameter uncertainty analysis of hydrological models using neural networks, *Hydrology and Earth System Sciences*, *13*(7), 1235–1248.

Simpson, T. W., J. D. Peplinski, P. N. Koch, and J. K. Allen (2001), Metamodels for computer-based engineering design: Survey and recommendations, *Engineering with Computers*, *17*(2), 129–150, doi:10.1007/Pl00007198.

Sincock, A. M., H. S. Wheater, and P. G. Whitehead (2003), Calibration and sensitivity analysis of a river water quality model under unsteady flow conditions, *Journal of Hydrology*, *277*(3-4), 214–229, doi:10.1016/S0022-1694(03)00127-6.

Singleton, V. L., B. Jacob, M. T. Feeney, and J. C. Little (2013), Modeling a proposed quarry reservoir for raw water storage in Atlanta, Georgia, *Journal of Environmental Engineering*, *139*(1), 70–78, doi:10.1061/(ASCE)Ee.1943-7870.0000582.

Smith Sawyer, H., A. R. Shaw, E. J. LeBoeuf, and M. P. McDonald (2013), A novel approach to optimization of power production in a large-scale controlled reservoir system, in *Hydrovision International*, Denver, CO, July 23–26, 2013.

Sóbester, A. (2003), Enhancements to global design optimization techniques, PhD dissertation, Department of Mechanical Engineering, University of Southampton, Southampton, UK.

Sóbester, A., S. J. Leary, and A. J. Keane (2005), On the design of optimization strategies based on global response surface approximation models, *Journal of Global Optimization*, *33*(1), 31–59, doi:10.1007/s10898-004-6733-1.

Solomatine, D. P., and L. A. Avila Torres (1996), Neural network approximation of a hydrodynamic model in optimizing reservoir operation, in *2nd International Conference on Hydroinformatics*, pp. 201–206, Zurich, Switzerland, Sept 1996.

Spear, R. C., and G. M. Hornberger (1980), Eutrophication in Peel Inlet - II. Identification of critical uncertainties via generalized sensitivity analysis, *Water Research*, *14*(1), 43–49, doi:10.1016/0043-1354(80)90040-8.

Srdjevic, B., Y. D. P. Medeiros, and A. S. Faria (2004), An objective multi-criteria evaluation of water management scenarios, *Water Resources Management*, *18*(1), 35–54, doi:10.1023/B:Warm.0000015348.88832.52.

Srivastava, A., K. Hacker, K. Lewis, and T. W. Simpson (2004), A method for using legacy data for metamodel-based design of large-scale systems, *Structural and Multidisciplinary Optimization*, *28*(2-3), 146–155, doi:10.1007/s00158-004-0438-4.

Stansbury, J., and D. M. Admiraal (2004), Modeling to evaluate macrophyte induced impacts to dissolved oxygen in a tailwater reservoir, *Journal of the American Water Resources Association*, *40*(6), 1483–1497, doi:10.1111/j.1752-1688.2004.tb01600.x.

Stedinger, J. R., B. F. Sule, and D. P. Loucks (1984), Stochastic dynamic-programming models for reservoir operation optimization, *Water Resources Research*, *20*(11), 1499–1505, doi:10.1029/Wr020i011p01499.

Stewart, G., R. Anderson, and E. Wohl (2005), Two-dimensional modelling of habitat suitability as a function of discharge on two Colorado rivers, *River Research and Applications*, *21*(10), 1061–1074, doi:10.1002/rra.868.

Streeter, H. W., and E. B. Phelps (1925), A study of the pollution and natural purification of the Ohio River, *Report*, United States Public Health Service, U.S. Department of Health, Education, and Welfare.

Suiadee, W., and T. Tingsanchali (2007), A combined simulation-genetic algorithm optimization model for optimal rule curves of a reservoir: A case study of the Nam Oon Irrigation Project, Thailand, *Hydrological Processes*, *21*(23), 3211–3225, doi:10.1002/Hyp.6528.

Sulis, A., and G. M. Sechi (2013), Comparison of generic simulation models for water resource systems, *Environmental Modelling & Software*, *40*, 214–225, doi:10.1016/j.envsoft.2012.09.012.

Sullivan, A. B., H. I. Jager, and R. Myers (2003), Modeling white sturgeon movement in a reservoir: The effect of water quality and sturgeon density, *Ecological Modelling*, *167*(1-2), 97–114, doi:10.1016/S0304-3800(03)00169-8.

Tamura, S., and M. Tateishi (1997), Capabilities of a four-layered feedforward neural network: Four layers versus three, *IEEE Transactions on Neural Networks*, *8*(2), 251–255, doi:10.1109/72.557662.

Teegavarapu, R. S. V., and S. P. Simonovic (2000), Short-term operation model for coupled hydropower reservoirs, *Journal of Water Resources Planning and Management*, *126*(2), 98–106, doi:10.1061/(ASCE)0733-9496(2000)126:2(98).

Teegavarapu, R. S. V., and S. P. Simonovic (2002), Optimal operation of reservoir systems using simulated annealing, *Water Resources Management*, *16*(5), 401–428, doi:10.1023/A:1021993222371.

Teegavarapu, R. S. V., A. R. Ferreira, and S. P. Simonovic (2013), Fuzzy multiobjective models for optimal operation of a hydropower system, *Water Resources Research*, *49*(6), 3180–3193, doi:10.1002/Wrcr.20224.

Tejada-Guibert, J. A., J. R. Stedinger, and K. Staschus (1990), Optimization of the value of CVP's hydropower production, *Journal of Water Resources Planning and Management*, *116*(1), 52–70.

Thacker, W. I., J. W. Zhang, L. T. Watson, J. B. Birch, M. A. Iyer, and M. W. Berry (2010), Algorithm 905: SHEPPACK: Modified Shepard algorithm for interpolation of scattered multivariate data, *ACM Transactions on Mathematical Software*, *37*(3), doi:10.1145/1824801.1824812.

The Organisation for Economic Co-operation and Development (OECD) (2018), Members and Partners, ⟨http://www.oecd.org/about/membersandpartners⟩, accessed February 21 2018.

Thomann, R. V. (1963), Mathematical model for dissolved oxygen, *Journal of the Sanitary Engineering Division*, *89*(5), 1–32.

Tinos, R., and S. X. Yang (2007), A self-organizing random immigrants genetic algorithm for dynamic optimization problems, *Genetic Programming and Evolvable Machines*, *8*(3), 255–286, doi:10.1007/s10710-007-9024-z.

Tospornsampan, J., I. Kita, M. Ishii, and Y. Kitamura (2005), Optimization of a multiple reservoir system using a simulated annealing-A case study in the Mae Klong system, Thailand, *Paddy and Water Environment*, *3*(3), 137–147, doi:10.1007/s10333-005-0010-x.

Tufford, D. L., and H. N. McKellar (1999), Spatial and temporal hydrodynamic and water quality modeling analysis of a large reservoir on the South Carolina (USA) coastal plain, *Ecological Modelling*, *114*(2-3), 137–173, doi:10.1016/S0304-3800(98)00122-7.

Turner, D. F., G. J. Pelletier, and B. Kasper (2009), Dissolved oxygen and pH modeling of a periphyton dominated, nutrient enriched river, *Journal of Environmental Engineering*, *135*(8), 645–652, doi:10.1061/(Asce)0733-9372(2009)135:8(645).

Unver, O. I., and L. W. Mays (1990), Model for real-time optimal flood control operation of a reservoir system, *Water Resources Management*, *4*(1), 21–46, doi:10.1007/BF00429923.

U.S. Army Corps of Engineers (1998), Cumberland River Basin Master Water Control Plan, *Report*, Nashville District.

U.S. Army Corps of Engineers (2003a), HEC-PRM Prescriptive Reservoir Model User's Manual, *Report CPD-95*, Hydrologic Engineering Center.

U.S. Army Corps of Engineers (2003b), Application of the HEC Prescriptive Reservoir Model in the Columbia River System, *Report TP-146*, Hydrologic Engineering Center.

U.S. Army Corps of Engineers (2013a), National Inventory of Dams, NID National, ⟨http://geo.usace.army.mil/pgis/f?p=397:5:0::NO⟩, accessed December 5 2013.

U.S. Army Corps of Engineers (2013b), HEC-ResSim Reservoir System Simulation User's Manual, Version 3.1, *Report CPD-82*, Hydrologic Engineering Center.

U.S. Department of Energy (2015), 2014 Hydropower Market Report, *Report*, Wind and Water Technologies Office, Energy Efficiency and Renewable Energy (EERE), U.S. Department of Energy.

U.S. Department of Energy (2016a), International Energy Outlook 2016, *Report DOE/EIA-0484(2016)*, Energy Information Administration (EIA), U.S. Department of Energy.

U.S. Department of Energy (2016b), Hydropower Vision: A New Chapter for America's 1st Renewable Electricity Source, *Report DOE/GO-102016-4869*.

U.S. Environmental Protection Agency (2016), Cooling Water Intakes, ⟨https://www.epa.gov/cooling-water-intakes⟩, accessed October 18 2016.

Valerio, A., H. Rajaram, and E. Zagona (2010), Incorporating groundwater-surface water interaction into river management models, *Ground Water*, *48*(5), 661–673, doi:10.1111/j.1745-6584.2010.00702.x.

van der Linden, L., R. I. Daly, and M. D. Burch (2015), Suitability of a coupled hydrodynamic water quality model to predict changes in water quality from altered meteorological boundary conditions, *Water*, *7*(1), 348–361, doi:10.3390/w7010348.

Viana, F. A. C., and R. T. Haftka (2008), Using multiple surrogates for metamodeling, in *7th ASMO-UK/ISSMO International Conference on Engineering Design Optimization*, Bath, UK, July 7–8, 2008.

Viana, F. A. C., V. Picheny, and R. T. Haftka (2010), Using cross validation to design conservative surrogates, *AIAA Journal*, *48*(10), 2286–2298, doi:10.2514/1.J050327.

Viana, F. A. C., T. W. Simpson, V. Balabanov, and V. Toropov (2014), Metamodeling in multidisciplinary design optimization: How far have we really come?, *AIAA Journal*, *52*(4), 670–690, doi:10.2514/1.J052375.

Wang, J. Z., H. B. Yin, and F. Chung (2011), Isolated and integrated effects of sea level rise, seasonal runoff shifts, and annual runoff volume on California's largest water supply, *Journal of Hydrology*, *405*(1-2), 83–92, doi:10.1016/j.jhydrol.2011.05.012.

Wang, X. X., and W. H. Yang (2008), Modelling potential impacts of coalbed methane development on stream water quality in an American watershed, *Hydrological Processes*, *22*(1), 87–103, doi:10.1002/Hyp.6647.

Wang, Y. C., J. Yoshitani, and K. Fukami (2005), Stochastic multiobjective optimization of reservoirs in parallel, *Hydrological Processes*, *19*(18), 3551–3567, doi:10.1002/Hyp.5845.

Wardlaw, R., and K. Bhaktikul (2004), Comparison of genetic algorithm and linear programming approaches for lateral canal scheduling, *Journal of Irrigation and Drainage Engineering*, *130*(4), 311–317, doi:10.1061/(Asce)0733-9437(2004)130:4(311).

Wardlaw, R., and M. Sharif (1999), Evaluation of genetic algorithms for optimal reservoir system operation, *Journal of Water Resources Planning and Management*, *125*(1), 25–33, doi:10.1061/(Asce)0733-9496(1999)125:1(25).

Watkins, D. W., and D. A. Moser (2006), Economic-based optimization of Panama Canal system operations, *Journal of Water Resources Planning and Management*, *132*(6), 503–512, doi:10.1061/(ASCE)0733-9496(2006)132:6(503).

Witt, A., T. Magee, K. Stewart, B. Hadjerioua, D. Neumann, E. Zagona, and M. Politano (2017), Development and implementation of an optimization model for hydropower and total dissolved gas in the Mid-Columbia River System, *Journal of Water Resources Planning and Management*, *143*(10), doi:10.1061/(Asce)Wr.1943-5452.0000827.

Wolff, P. J., C. W. Almquist, R. J. Dorman, G. E. Hauser, D. F. McGinnis, M. H. Mobley, R. J. Ruane, and A. Sawyer (2013), Identifying the least cost approach for aeration strategies of new small hydro projects, in *Hydrovision International*, Denver, CO, July 23–26, 2013.

Wool, T. A., R. B. Ambrose, J. L. Martin, and E. A. Comer (2002), *Water Quality Analysis Simulation Program (WASP), User's Manual, Version 6.0*, 267 pp., USEPA Environmental Research Laboratory.

Wool, T. A., S. R. Davie, and H. N. Rodriguez (2003), Development of three-dimensional hydrodynamic and water quality models to support total maximum daily load decision process for the Neuse River Estuary, North Carolina, *Journal of Water Resources Planning and Management*, *129*(4), 295–306, doi:10.1061/(Asce)0733-9496(2003)129:4(295).

Wurbs, R. A. (2005), Comparative evaluation of generalized river/reservoir system models, *Report TR-282*, Texas Water Resouces Institute, Texas A&M University.

Xia, M., P. M. Craig, B. Schaeffer, A. Stoddard, Z. J. Liu, M. C. Peng, H. Y. Zhang, C. M. Wallen, N. Bailey, and J. Mandrup-Poulsenl (2010), Influence of physical forcing on bottom-water dissolved oxygen within Caloosahatchee River Estuary, Florida, *Journal of Environmental Engineering*, *136*(10), 1032–1044, doi:10.1061/(Asce)Ee.1943-7870.0000239.

Xu, Z., A. N. Godrej, and T. J. Grizzard (2007), The hydrological calibration and validation of a complexly-linked watershed-reservoir model for the Occoquan watershed, Virginia, *Journal of Hydrology*, *345*(3-4), 167–183, doi:10.1016/j.jhydrol.2007.07.015.

Xu, Z. X., H. L. Yin, and Y. J. Yao (2008), Prediction of water quality of Huangpu River using a tidal river network model, *Environmental Engineering Science*, *25*(10), 1463–1475, doi:10.1089/ees.2007.0219.

Yan, S. Q., and B. Minsker (2006), Optimal groundwater remediation design using an adaptive neural network genetic algorithm, *Water Resources Research*, *42*(5), doi:10.1029/2005wr004303.

Yan, S. Q., and B. Minsker (2011), Applying dynamic surrogate models in noisy genetic algorithms to optimize groundwater remediation designs, *Journal of Water Resources Planning and Management*, *137*(3), 284–292, doi:10.1061/(ASCE)Wr.1943-5452.0000106.

Yi, J., J. W. Labadie, and S. Stitt (2003), Dynamic optimal unit commitment and loading in hydropower systems, *Journal of Water Resources Planning and Management*, *129*(5), 388–398, doi:10.1061/(ASCE)0733-9496(2003)129:5(388).

Yurtal, R., G. Seckin, and M. Ardiclioglu (2005), Hydropower optimization for the Lower Seyhan System in Turkey using dynamic programming, *Water International*, *30*(4), 522–529.

Zagona, E. A., T. J. Fulp, R. Shane, Y. Magee, and H. M. Goranflo (2001), RiverWare: A generalized tool for complex reservoir system modeling, *Journal of the American Water Resources Association*, *37*(4), 913–929, doi:10.1111/j.1752-1688.2001.tb05522.x.

Zhang, R., J. Z. Zhou, S. Ouyang, X. M. Wang, and H. F. Zhang (2013), Optimal operation of multi-reservoir system by multi-elite guide particle swarm optimization, *International Journal of Electrical Power & Energy Systems*, *48*, 58–68, doi:10.1016/j.ijepes.2012.11.031.

Zhang, X. S., R. Srinivasan, and M. Van Liew (2009), Approximating SWAT model using artificial neural network and support vector machine, *Journal of the American Water Resources Association*, *45*(2), 460–474, doi:10.1111/j.1752-1688.2009.00302.x.

Zhao, T. T. G., J. S. Zhao, and D. W. Yang (2014), Improved dynamic programming for hydropower reservoir operation, *Journal of Water Resources Planning and Management*, *140*(3), 365–374, doi:10.1061/(Asce)Wr.1943-5452.0000343.

Ziaei, M., L. T. Shui, and E. Goodarzi (2012), Optimization and simulation modelling for operation of the Zayandeh Rud Reservoir, *Water International*, *37*(3), 305–318, doi:10.1080/02508060.2012.688189.

Zou, R., W. S. Lung, and J. Wu (2007), An adaptive neural network embedded genetic algorithm approach for inverse water quality modeling, *Water Resources Research*, *43*(8), doi:10.1029/2006wr005158.