



## EXECUTIVE SUMMARY

This effort is part of the larger Missouri River Recovery modeling project. The project involves the creation of a detailed suite of models for the Missouri River basin that can eventually aid in the evaluation of scenarios reflecting a wide-range of hydrologic conditions. The objective of this element of the project was to develop a computer reservoir model capable of simulating the system operation for the flow of record for assessment of base conditions on the mainstem portion of the Missouri River. To accomplish the objective, the computer model HEC-ResSim was utilized to simulate operations at the six mainstem dams on the Missouri River. Prior to model creation, much effort was spent developing required input data such as local inflow, evaporation, and dam and reservoir physical parameters; all parameters are explained in detail within this report.

The Missouri River system is operated for eight congressionally authorized purposes. The system is unique and contains six reservoirs with dramatic differences in storage distribution and long river reaches in between. There are four downstream target points for which releases are planned, the farthest having an approximate travel time of six days from the closest reservoir. In normal operation, system releases are typically planned from downstream to upstream. For these and other reasons including limitations in ResSim's standard features for tandem reservoir system and downstream control operations, modeling such an involved river system required the development of complex scripted rules. Rather than attempt to capture every historic operation, strategic modeling goals were established which outlined tasks critical to meeting this project element's objective.

The current model performs many functions well and meets most of the established goals. Met goals include:

- Achieving system balance at the start of the navigation season
- Evacuating annual runoff while satisfying downstream flow requirements (environmental, water supply, water quality, navigation, etc.)
- Evacuating flood waters while mitigating damaging flows
- Meeting special operations at Fort Randall and Big Bend

# TABLE OF CONTENTS

Executive Summary .....	i
Table of Contents .....	ii
List of Figures .....	vi
List of Tables .....	xii
Acronyms .....	xiv
1 Introduction.....	1-1
2 Background .....	2-1
2.1 Basin Description .....	2-1
2.2 Mainstem Projects .....	2-3
2.2.1 Fort Peck .....	2-5
2.2.2 Garrison.....	2-6
2.2.3 Oahe.....	2-6
2.2.4 Big Bend.....	2-7
2.2.5 Fort Randall .....	2-7
2.2.6 Gavins Point .....	2-7
2.3 Missouri River Mainstem System Operations.....	2-7
2.3.1 System Regulation.....	2-7
2.3.1.1 Overview.....	2-7
2.3.1.2 Intrasystem Regulation - General.....	2-9
2.3.1.3 Seasonal Intrasystem Regulation Patterns .....	2-10
2.3.1.4 Short-Term Intrasystem Adjustments.....	2-11
2.3.1.5 Project Release Limits .....	2-12
2.3.2 Recurring Operational Considerations .....	2-14
2.3.2.1 Flood Control .....	2-14
2.3.2.2 Water Requirements Below Gavins Point .....	2-15
2.3.2.3 Water Requirements Above Gavins Point.....	2-19
3 Data.....	3-20
3.1 GIS Data.....	3-20
3.2 Observed Data.....	3-24
3.2.1 USGS .....	3-26
3.2.2 MRBWM .....	3-26
3.2.3 UMRSSFFS .....	3-26

3.2.4	DRM .....	3-26
3.2.5	USBR Depletions .....	3-26
3.3	Routing Parameters .....	3-29
3.4	Local Flows .....	3-29
3.5	Evaporation .....	3-33
3.5.1	Evaporation Rates Pre-1967 .....	3-34
3.5.2	Evaporation Rates Post-1966 .....	3-35
3.5.3	Inflow Loss due to Evaporation Computations .....	3-39
3.6	Reservoir Data .....	3-40
3.6.1	Elevation-Area-Capacity Curves .....	3-40
3.6.2	Spillway Flow .....	3-40
3.6.3	Outlet Works Flow .....	3-44
3.6.4	Powerhouse Flow .....	3-46
3.6.5	Power Generation Capacity .....	3-49
3.6.6	Tailwater .....	3-52
3.6.7	Power Efficiencies .....	3-55
4	ResSim Modeling .....	4-59
4.1	ResSim Program Overview .....	4-59
4.2	Modeling Extents .....	4-59
4.3	Modeling Strategy .....	4-60
4.3.1	Overview .....	4-60
4.3.2	Rules .....	4-63
4.3.3	Downstream Model vs System Model .....	4-72
4.3.4	Scripted Rules in the Downstream Model .....	4-72
4.3.4.1	WS, NAV, and Flood Targets Scripted Rule .....	4-72
4.3.4.2	Flood Evacuation Scripted Rule .....	4-73
4.3.4.3	Steady Release Scripted Rule .....	4-74
4.3.4.4	Service Level State Variable .....	4-74
4.3.4.1	Service Level x1k State Variable .....	4-81
4.3.4.2	Spring Pulse Scripted Rule .....	4-82
4.3.5	Scripted Rules in the System Model .....	4-83
4.3.5.1	GAPT Control Scripted Rule .....	4-83
4.3.5.2	Reservoir Balancing Scripted Rule .....	4-85

4.3.5.3	Water Supply Scripted Rule .....	4-88
5	Evaluation of Model Performance .....	5-89
5.1	Calibration Comparison .....	5-89
5.2	Pool Probability Comparison.....	5-96
5.3	Release and Flow Probability Comparison.....	5-103
5.1	Decision Comparisons .....	5-114
5.1.1	Downstream Model.....	5-114
5.1.1.1	Service Level .....	5-114
5.1.1.2	Navigation Targets.....	5-120
5.1.1.3	Steady Release .....	5-122
5.1.1.4	Navigation End Date.....	5-126
5.1.1.5	Winter Release .....	5-129
5.1.1.6	Flood Targets .....	5-133
5.1.1.7	Water Supply Requirements .....	5-136
5.1.2	System Model.....	5-145
5.1.2.1	System Balancing .....	5-145
5.1.2.2	Guide Curve Operations .....	5-148
5.1.2.3	Upstream Water Supply Requirements.....	5-153
6	References .....	6-156
7	Appendix A – Pertinent Data.....	7-1
8	Appendix B – Routing Parameter Determination Summary.....	8-1
8.1	Coefficient Routing.....	8-1
8.2	Straddle-Stagger Routing.....	8-2
8.3	Muskingum Routing .....	8-4
8.4	Muskingum-Cunge Routing.....	8-5
8.5	Modified Puls Routing .....	8-7
8.6	Results.....	8-8
8.6.1	Modified Puls, Muskingum-Cunge, Muskingum, and Straddle-Stagger Routing Method Comparison Plots.....	8-12
8.6.2	Straddle-Stagger Routing Results.....	8-21
8.6.3	Straddle-Stagger vs. Coefficient Routing Plots.....	8-39
9	Appendix C – Elevation-Area-Capacity Curves .....	9-1
10	Appendix D – ResSim Models Flow-Chart .....	10-1

11	Appendix E – Service Level State Variable .....	11-1
12	Appendix F – Service Level x1k State Variable.....	12-1
13	Appendix G – Spring Pulse Scripted Rule.....	13-1
14	Appendix H – Flood Evacuation Scripted Rule.....	14-1
15	Appendix I – Steady Release Scripted Rule.....	15-1
16	Appendix J – GAP Control Scripted Rule.....	16-1
17	Appendix K – Reservoir Balancing Scripted Rule.....	17-1

## LIST OF FIGURES

Figure 2-1: Storage capacity of Corps reservoirs. ....	2-2
Figure 2-2: Missouri River Basin .....	2-2
Figure 2-3: System storage zones.....	2-4
Figure 2-4: Profile of mainstem System and storage capacities .....	2-5
Figure 2-5: Water control calendar of events.....	2-9
Figure 3-1: ResSim stream alignment and computation points.....	3-21
Figure 3-2: GIS study map. ....	3-23
Figure 3-3: RUNE unadjusted local flow for POR. ....	3-30
Figure 3-4: Oahe DRM daily evaporation. ....	3-38
Figure 3-5: Oahe DRM monthly evaporation. ....	3-38
Figure 3-6: Annual DRM evaporation at the mainstem projects.....	3-39
Figure 3-7: FTPK spillway capacity curve.....	3-41
Figure 3-8: GARR spillway capacity curve. ....	3-41
Figure 3-9: OAHE spillway capacity curve.....	3-42
Figure 3-10: BEND spillway capacity curve.....	3-42
Figure 3-11: FTRA spillway capacity curve. ....	3-43
Figure 3-12: GAPT spillway capacity curve. ....	3-43
Figure 3-13: GARR outlet works capacity curve. ....	3-44
Figure 3-14: OAHE outlet works capacity curve. ....	3-45
Figure 3-15: FTRA outlet works capacity curve. ....	3-45
Figure 3-16: FTPK powerhouse discharge capacity. ....	3-46
Figure 3-17: GARR powerhouse discharge capacity.....	3-47
Figure 3-18: OAHE powerhouse discharge capacity. ....	3-47
Figure 3-19: FTRA powerhouse discharge capacity.....	3-48
Figure 3-20: GAPT powerhouse discharge capacity.....	3-48
Figure 3-21: FTPK power generation capacity. ....	3-49
Figure 3-22: GARR power generation capacity. ....	3-49
Figure 3-23: OAHE power generation capacity. ....	3-50
Figure 3-24: BEND power generation capacity.....	3-50
Figure 3-25: FTRA power generation capacity. ....	3-51
Figure 3-26: GAPT power generation capacity.....	3-51
Figure 3-27: FTPK tailwater. ....	3-52
Figure 3-28: GARR tailwater. ....	3-53
Figure 3-29: OAHE tailwater. ....	3-53
Figure 3-30: BEND tailwater.....	3-54
Figure 3-31: FTRA tailwater. ....	3-54
Figure 3-32: GAPT tailwater.....	3-55
Figure 3-33: FTPK powerplant efficiency.....	3-56
Figure 3-34: GARR powerplant efficiency. ....	3-56
Figure 3-35: OAHE powerplant efficiency.....	3-57
Figure 3-36: BEND powerplant efficiency.....	3-57
Figure 3-37: FTRA powerplant efficiency. ....	3-58
Figure 3-38: GAPT powerplant efficiency.....	3-58

Figure 4-1: Missouri River Mainstem System ResSim Network.....	4-60
Figure 4-2: Service Level Determination Chart (from Plate VI-1 in Master Manual).....	4-77
Figure 4-3: Sample DRM forecast text file.....	4-79
Figure 4-4: Plot of release schedules for FTPK used in the reservoir balancing script.....	4-87
Figure 4-5: Plot of release schedules for GARR used in the reservoir balancing script. ....	4-88
Figure 5-1: ResSim versus historic operations at Fort Peck Dam 1998-2012.....	5-90
Figure 5-2: ResSim versus historic operations at Garrison Dam 1998-2012. ....	5-91
Figure 5-3: ResSim versus historic operations at Oahe Dam 1998-2012. ....	5-92
Figure 5-4: ResSim versus historic operations at Big Bend Dam 1998-2012.....	5-93
Figure 5-5: ResSim versus historic operations at Fort Randall Dam 1998-2012.....	5-94
Figure 5-6: ResSim versus historic operations at Gavins Point Dam 1998-2012.....	5-95
Figure 5-7: Fort Peck Lake pool probability curve for years 1998-2012.....	5-97
Figure 5-8: Lake Sakakawea pool probability curve for years 1998-2012.....	5-98
Figure 5-9: Lake Oahe pool probability curves for years 1998-2012.....	5-99
Figure 5-10: Lake Sharpe pool probability curves for years 1998-2012.....	5-100
Figure 5-11: Lake Francis Case pool probability curves for years 1998-2012. ....	5-101
Figure 5-12: Lewis and Clark Lake pool probability curves for years 1998-2012. ....	5-102
Figure 5-13: Fort Peck Lake release probability curves for years 1998-2012. ....	5-104
Figure 5-14: Lake Sakakawea release probability curves for years 1998-2012. ....	5-105
Figure 5-15: Lake Oahe release probability curves for years 1998-2012.....	5-106
Figure 5-16: Lake Sharpe release probability curves for years 1998-2012. ....	5-107
Figure 5-17: Lake Francis Case release probability curves for years 1998-2012.....	5-108
Figure 5-18: Lewis and Clark Lake release probability for years 1998-2012.....	5-109
Figure 5-19: Sioux City, IA flow probability curves for years 1998-2012. ....	5-110
Figure 5-20: Omaha, NE flow probability curves for years 1998-2012.....	5-111
Figure 5-21: Nebraska City, NE flow probability curves for years 1998-2012. ....	5-112
Figure 5-22: Kansas City, MO flow probability curves for years 1998-2012.....	5-113
Figure 5-23: Plot of service level and system storage in 1946.....	5-115
Figure 5-24: Plot of service level and system storage in 2002.....	5-117
Figure 5-25: Plot of service level and system storage in 1981.....	5-119
Figure 5-26: Control points plot of target discharges and simulated discharges at each of the target locations.....	5-121
Figure 5-27: Plot of forecasted runoff and steady release in 1980.....	5-123
Figure 5-28: Plot of System releases in 1962.....	5-124
Figure 5-29: Plot of System release in 1956.....	5-125
Figure 5-30: Plot of the navigation end date in 1955. ....	5-127
Figure 5-31: Plot of the navigation end date in 1956. ....	5-128
Figure 5-32: Plot of winter releases in 1956-1957. ....	5-130
Figure 5-33: Plot of winter releases in 1943-1944. ....	5-131
Figure 5-34: Plot of Gavins Point releases and System storage in 1969-1970. ....	5-132
Figure 5-35: Plot of target locations with full-service flood target reductions in 1953. Gray hatched areas are the navigation requirements that Gavins Point is operating for and the gray dotted line is the navigation targets based on service level. ....	5-134

Figure 5-36: Plot of target locations with full-service and minimum-service flood target reductions in 1953. Gray hatched areas are the navigation requirements that Gavins Point is operating for and the gray dotted line is the navigation targets based on service level.....	5-135
Figure 5-37: Plot of Gavins Point releases and System storage in 1990 operating for water supply. ....	5-138
Figure 5-38: Plot of System releases and System storage in 1936 operating for 18,000 cfs water supply. ....	5-139
Figure 5-39: Plot of target locations operating for 18,000 cfs water supply in 1936. ....	5-140
Figure 5-40: Plot of System releases and System storage in 1936 operating for 9,000 cfs water supply. ....	5-141
Figure 5-41: Plot of target locations operating for 9,000 cfs water supply in 1936. ....	5-142
Figure 5-42: Plot of System releases and System storage in 1936 operating for 12,000 cfs water supply. ....	5-143
Figure 5-43: Plot of target locations operating for 12,000 cfs water supply in 1936. ....	5-144
Figure 5-44: Plot of Fort Peck's, Garrison's, and Oahe's percent of occupied carryover storage and reservoir storages in 1949-1950. The top plot shows Fort Peck's current percent occupied carryover storage (blue), Garrison's current percent occupied carryover storage (red), Oahe's current percent occupied carryover storage (green), and the combined forecasted percent occupied carryover storage. The bottom three plots show each reservoir's target storage and current storage.....	5-146
Figure 5-45: Plot of Fort Peck's, Garrison's, and Oahe's percent of occupied carryover storage and reservoir storages in 1964-1965. The top plot shows Fort Peck's current percent occupied carryover storage (blue), Garrison's current percent occupied carryover storage (red), Oahe's current percent occupied carryover storage (green), and the combined forecasted percent occupied carryover storage. The bottom three plots show each reservoir's target storage and current storage.....	5-147
Figure 5-46: Plot of Big Bend guide curve operations in 1962. ....	5-149
Figure 5-47: Plot of Fort Randall guide curve operations with a navigation end date of December 1, 1968.....	5-150
Figure 5-48: Plot of Fort Randall guide curve operations with a navigation end date of November 1, 1932.....	5-151
Figure 5-49: Plot of Gavins Point's guide curve operations in 1972.....	5-152
Figure 5-50: Plot of Fort Peck's releases operating for water supply in 1938. ....	5-154
Figure 5-51: Plot of Garrison's releases operating for water supply in December 1961.....	5-155
Figure 8-1: Straddle-Stagger and Muskingum HMS basin schematic.....	8-4
Figure 8-2: Muskingum-Cunge and Modified Puls HMS basin schematic.....	8-7
Figure 8-3: Straddle-Stagger composite routing HMS basin schematic.....	8-10
Figure 8-4: SUX-OMA 2011 event.....	8-12
Figure 8-5: SUX-OMA 1997 event.....	8-13
Figure 8-6: SUX-OMA 1993 Event. ....	8-14
Figure 8-7: OMA-NCNE 2011 Event. ....	8-15
Figure 8-8: OMA-NCNE 1952 Event. ....	8-16
Figure 8-9: OMA-NCNE 1944 Event. ....	8-17
Figure 8-10: NCNE-RUNE 2011 Event. ....	8-18

Figure 8-11: NCNE-RUNE 1993 Event. .... 8-19

Figure 8-12: NCNE-RUNE 1984 Event. .... 8-20

Figure 8-13: Straddle-Stagger routing results for RBMT-FTPK during 2011. Red data are the Straddle-Stagger data and black data are the observed data..... 8-21

Figure 8-14: Straddle-Stagger routing results for FTPK-WPMT during 2011. Red data are the Straddle-Stagger data and black data are the observed data..... 8-22

Figure 8-15: Straddle-Stagger routing results for WPMT-CLMT during 2011. Red data are the Straddle-Stagger data and black data are the observed data..... 8-23

Figure 8-16 Straddle-Stagger routing results for CLMT-GARR during 2011. Red data are the Straddle-Stagger data and black data are the observed data..... 8-24

Figure 8-17: Straddle-Stagger routing results for GARR-BIS during 2011. Red data are the Straddle-Stagger data and black data are the observed data..... 8-25

Figure 8-18: Straddle-Stagger routing results for BIS-OAHE during 2011. Red data are the Straddle-Stagger data and black data are the observed data..... 8-26

Figure 8-19: Straddle-Stagger routing results for OAHE-BEND during 2011. Red data are the Straddle-Stagger data and black data are the observed data..... 8-27

Figure 8-20: Straddle-Stagger routing results for BEND-FTRA during 2011. Red data are the Straddle-Stagger data and black data are the observed data..... 8-28

Figure 8-21: Straddle-Stagger routing results for FTRA-GAPT during 2011. Red data are the Straddle-Stagger data and black data are the observed data..... 8-29

Figure 8-22: Straddle-Stagger routing results for GAPT-SUX during 2011. Red data are the Straddle-Stagger data and black data are the observed data..... 8-30

Figure 8-23: Straddle-Stagger routing results for SUX-OMA during 2011. Red data are the Straddle-Stagger data and black data are the observed data..... 8-31

Figure 8-24: Straddle-Stagger routing results for OMA-NCNE during 2011. Red data are the Straddle-Stagger data and black data are the observed data..... 8-32

Figure 8-25: Straddle-Stagger routing results for NCNE-RUNE during 2011. Red data are the Straddle-Stagger data and black data are the observed data..... 8-33

Figure 8-26: Straddle-Stagger routing results for RUNE-STJ during 2011. Red data are the Straddle-Stagger data and black data are the observed data..... 8-34

Figure 8-27: Straddle-Stagger routing results for STJ-MKC during 2011. Red data are the Straddle-Stagger data and black data are the observed data..... 8-35

Figure 8-28: Straddle-Stagger routing results for MKC-WVMO during 2011. Red data are the Straddle-Stagger data and black data are the observed data..... 8-36

Figure 8-29: Straddle-Stagger routing results for WVMO-BNMO during 2011. Red data are the Straddle-Stagger data and black data are the observed data..... 8-37

Figure 8-30: Straddle-Stagger routing results for BNMO-HEMO during 2011. Red data are the Straddle-Stagger data and black data are the observed data..... 8-38

Figure 8-31: Straddle-Stagger vs. Coefficient routing results for RBMT-FTPK during 2011. Red data are the Straddle-Stagger data, blue data are the Coefficient Routing data, and black data are the observed data. .... 8-39

Figure 8-32: Straddle-Stagger vs. Coefficient routing results for FTPK-WPMT during 2011. Red data are the Straddle-Stagger data, blue data are the Coefficient Routing data, and black data are the observed data. .... 8-40

Figure 8-33: Straddle-Stagger vs. Coefficient routing results for WPMT-CLMT during 2011. Red data are the Straddle-Stagger data, blue data are the Coefficient Routing data, and black data are the observed data. .... 8-41

Figure 8-34: Straddle-Stagger vs. Coefficient routing results for CLMT-GARR during 2011. Red data are the Straddle-Stagger data, blue data are the Coefficient Routing data, and black data are the observed data. .... 8-42

Figure 8-35: Straddle-Stagger vs. Coefficient routing results for GARR-BIS during 2011. Red data are the Straddle-Stagger data, blue data are the Coefficient Routing data, and black data are the observed data. .... 8-43

Figure 8-36: Straddle-Stagger vs. Coefficient routing results for BIS-OAHE during 2011. Red data are the Straddle-Stagger data, blue data are the Coefficient Routing data, and black data are the observed data. .... 8-44

Figure 8-37: Straddle-Stagger vs. Coefficient routing results for OAHE-BEND during 2011. Red data are the Straddle-Stagger data, blue data are the Coefficient Routing data, and black data are the observed data. .... 8-45

Figure 8-38: Straddle-Stagger vs. Coefficient routing results for BEND-FTRA during 2011. Red data are the Straddle-Stagger data, blue data are the Coefficient Routing data, and black data are the observed data. .... 8-46

Figure 8-39: Straddle-Stagger vs. Coefficient routing results for FTRA-GAPT during 2011. Red data are the Straddle-Stagger data, blue data are the Coefficient Routing data, and black data are the observed data. .... 8-47

Figure 8-40: Straddle-Stagger vs. Coefficient routing results for GAPT-SUX during 2011. Red data are the Straddle-Stagger data, blue data are the Coefficient Routing data, and black data are the observed data. .... 8-48

Figure 8-41: Straddle-Stagger vs. Coefficient routing results for SUX-OMA during 2011. Red data are the Straddle-Stagger data, blue data are the Coefficient Routing data, and black data are the observed data. .... 8-49

Figure 8-42: Straddle-Stagger vs. Coefficient routing results for OMA-NCNE during 2011. Red data are the Straddle-Stagger data, blue data are the Coefficient Routing data, and black data are the observed data. .... 8-50

Figure 8-43: Straddle-Stagger vs. Coefficient routing results for NCNE-RUNE during 2011. Red data are the Straddle-Stagger data, blue data are the Coefficient Routing data, and black data are the observed data. .... 8-51

Figure 8-44: Straddle-Stagger vs. Coefficient routing results for RUNE-STJ during 2011. Red data are the Straddle-Stagger data, blue data are the Coefficient Routing data, and black data are the observed data. .... 8-52

Figure 8-45: Straddle-Stagger vs. Coefficient routing results for STJ-MKC during 2011. Red data are the Straddle-Stagger data, blue data are the Coefficient Routing data, and black data are the observed data. .... 8-53

Figure 8-46: Straddle-Stagger vs. Coefficient routing results for MKC-WVMO during 2011. Red data are the Straddle-Stagger data, blue data are the Coefficient Routing data, and black data are the observed data. .... 8-54

Figure 8-47: Straddle-Stagger vs. Coefficient routing results for WVMO-BNMO during 2011. Red data are the Straddle-Stagger data, blue data are the Coefficient Routing data, and black data are the observed data. .... 8-55

Figure 8-48: Straddle-Stagger vs. Coefficient routing results for BNMO-HEMO during 2011. Red data are the Straddle-Stagger data, blue data are the Coefficient Routing data, and black data are the observed data. .... 8-56

## LIST OF TABLES

Table 3-1: Mainstem Gage ID and Locations. ....	3-22
Table 3-2: Sources for POR data set construction. ....	3-25
Table 3-3: HUC8 depletion adjustment factors based on DCP drainage area. ....	3-28
Table 3-4: Coefficient method final routing parameters. ....	3-29
Table 3-5: Local flow adjustments summary. ....	3-32
Table 3-6: Routing coefficient adjustments used in local flow determination. ....	3-33
Table 3-7: Normal annual lake and net evaporation from TP-37. ....	3-34
Table 3-8: Annual net evaporation during 1930-1941.....	3-35
Table 3-9: Monthly Distribution of DRM Evaporation Prior to 1967.....	3-35
Table 3-10: Normal monthly pan evaporation in inches.....	3-36
Table 3-11: Normal monthly pan to lake evaporation coefficients.....	3-37
Table 3-12: Normal monthly lake evaporation. ....	3-37
Table 4-1: Fort Peck operations/rule stack in the System model. ....	4-63
Table 4-2: Garrison operations/rule stack in the System model.....	4-64
Table 4-3: Oahe operations/rule stack in the System model.....	4-65
Table 4-4: Big Bend operations/rule stack in the System model. ....	4-65
Table 4-5: Fort Randall operations/rule stack in the System model.....	4-66
Table 4-6: Fort Randall guide curve. ....	4-67
Table 4-7: Gavins Point operations/rule stack in the System model. ....	4-67
Table 4-8: Gavins Point guide curve. ....	4-67
Table 4-9: Gavins Point operations/rule stack in the Downstream model. ....	4-68
Table 4-10: Downstream flood targets. Summarized from Tables VII-7 and VII-8 in the Master Manual (U.S. Army Corps of Engineers, 2006).....	4-72
Table 4-11: Gavins Point releases needed to meet target flows from Plate 3 in the AOP.....	4-75
Table 4-12: Service Level Script Output State Variables. ....	4-76
Table 4-13: Navigation target discharges related to service level. Summarized from Table VII-1 in the Master Manual (U.S. Army Corps of Engineers, 2006). ....	4-82
Table 4-14: Navigation season at each target location. Summarized from Section 7-03.4.1 in the Master Manual. ....	4-82
Table 5-1: Service level requirements. Summarized from Table VII-2 in the Master Manual (U.S. Army Corps of Engineers, 2006). ....	5-114
Table 5-2: Navigation end date or season length criteria. Summarized from Table VII-3 in the Master Manual (U.S. Army Corps of Engineers, 2006). ....	5-126
Table 5-3: Winter release from Gavins Point criteria. Summarized from Table VII-4 in the Master Manual (U.S. Army Corps of Engineers, 2006). ....	5-129
Table 5-4: Water supply dates and discharges summarized from Sections 7-03.6.1 and 7-11.3.5 of the Master Manual (U.S. Army Corps of Engineers, 2006). ....	5-136
Table 8-1: Routing methods in hydrologic HEC programs. ....	8-1
Table 8-2: DRM coefficient routing parameters. ....	8-2
Table 8-3: Calibrated Straddle-Stagger and corresponding Coefficient routing parameters. ....	8-3
Table 8-4: Calibrated Musking routing parameters. ....	8-5
Table 8-5: Muskingum-Cunge routing parameters. ....	8-6
Table 8-6: Modified Puls routing storage-discharge curves. ....	8-8

Table 8-7: Final routing parameters. .... 8-11  
Table 9-1: Fort Peck E-A-C curves..... 9-1  
Table 9-2: Garrison E-A-C curves. .... 9-2  
Table 9-3: Oahe E-A-C curves. .... 9-3  
Table 9-4: Big Bend E-A-C curves. .... 9-4  
Table 9-5: Fort Randall E-A-C curves. .... 9-5  
Table 9-6: Gavins Point E-A-C curves..... 9-6

## **ACRONYMS**

AOP.....	Annual Operating Plan
CMA.....	Centered Moving Average
CR.....	Coefficient Routing
CRREL.....	Cold Regions Research and Engineering Laboratory
DCP.....	Data Collection Platform
DRM .....	Daily Routing Model
DSSVue.....	Data Storage System (by HEC)
E-A-C.....	Elevation-Area-Capacity
ESOP.....	Emergency Systems Operation Plan
FEMA.....	Federal Emergency Management Agency
FTT .....	Flow to Target
GIS .....	Geographic Information System
HEC.....	Hydrologic Engineering Center
HMS.....	Hydrologic Modeling Software (by HEC)
WAT.....	Watershed Analysis Tool (by HEC)
MAF.....	Million acre-feet
MO.....	Missouri
MR.....	Missouri River
MRADS.....	Mass Random Access Data Storage
MRBWM.....	Missouri River Basin Water Management Division (previously RCC)
MS.....	Mississippi
MVS.....	Mississippi Valley Division St Louis District
NOAA CPC.....	National Oceanic and Atmospheric Administration Climate Prediction .. Center
NRCS.....	Natural Resources Conservation Service
NWK.....	Northwest Division Kansas City District

NWO..... Northwest Division Omaha District  
NWS ..... National Weather Service  
POR..... Period of Record  
RAS ..... River Analysis System  
RCC..... Reservoir Control Center  
ResSim.....Reservoir Simulation Software (by HEC)  
RM..... River Mile  
SR..... Steady Release  
SS..... Straddle-Stagger Routing  
UMRSFFS ..... Upper Mississippi River System Flow Frequency Study  
USACE..... United States Army Corps of Engineers  
USBR..... United States Bureau of Reclamation  
USGS ..... United States Geological Survey  
WAPA ..... Western Area Power Administration Home  
WCM..... Water Control Manual

# **1 INTRODUCTION**

The Missouri River mainstem ResSim model was created as a base model for planning studies which could be used in the future to simulate and analyze broadscale watershed alternatives. The objective of this ResSim model is to simulate system operation for the flow of record for assessment of base conditions on the Missouri River. Alternatives to be modeled have not been defined yet. Alternatives are likely to reflect the desires of basin stakeholders as well as local and federal agencies. The need for Missouri River ResSim modeling has been discussed in conjunction with various federal studies for many years now. While the exact end results are yet unknown, the ResSim model was constructed to be adaptable while still adequately simulating Missouri River System overall operations.

## **2 BACKGROUND**

### **2.1 BASIN DESCRIPTION**

The Missouri River is 2,341 miles long and drains one sixth of the United States encompassing 529,350 square miles. The Missouri River reservoir system, which became fully operational in 1967, consists of six Corps dams with a total storage capacity of 72.4 million acre-feet (MAF) and carry-over storage of 38.5 MAF of water, which makes it the largest reservoir system in North America. Figure 2-1 shows how the Missouri River System reservoirs compare to other USACE reservoirs in the United States. The system is operated to serve eight congressionally authorized project purposes of flood control, navigation, irrigation, hydropower, water supply, water quality, recreation, and fish and wildlife. Runoff from above the mainstem reservoir system dams is stored in the six reservoirs where it serves project purposes. Water is released from the mainstem reservoir system as directed by the system's Master Manual (U.S. Army Corps of Engineers, 2006). Figure 2-2 shows the Missouri River Basin.

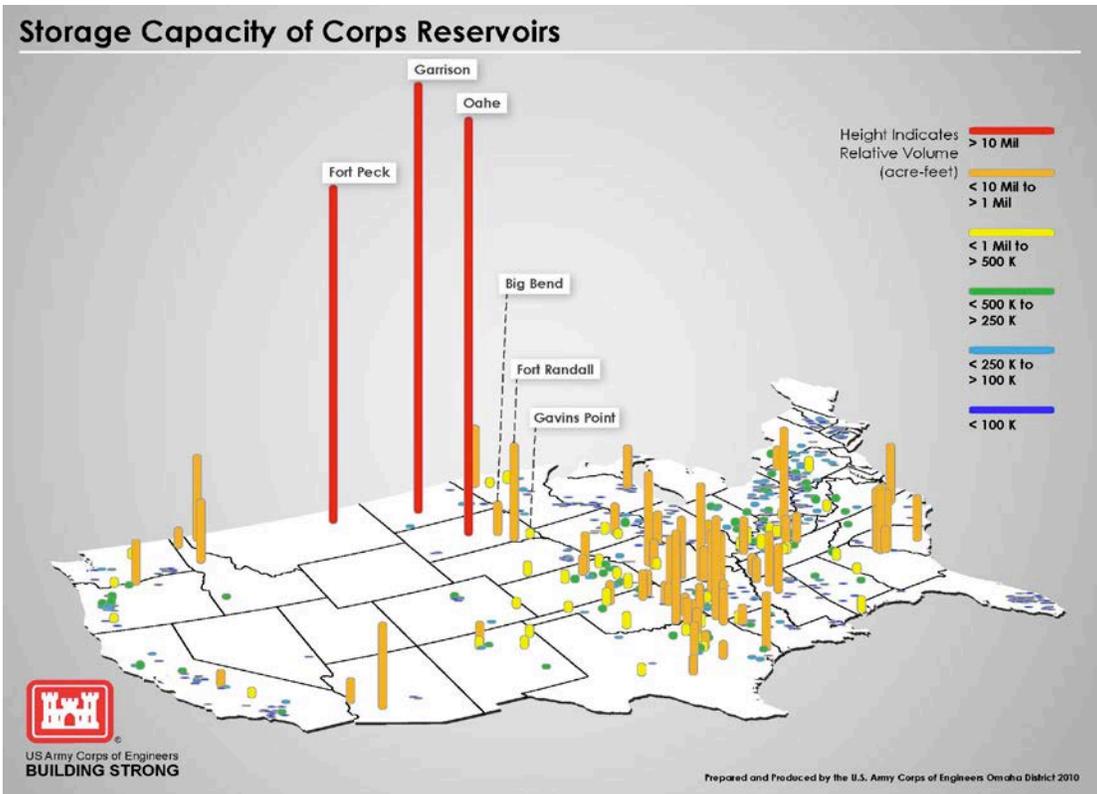


Figure 2-1: Storage capacity of Corps reservoirs.

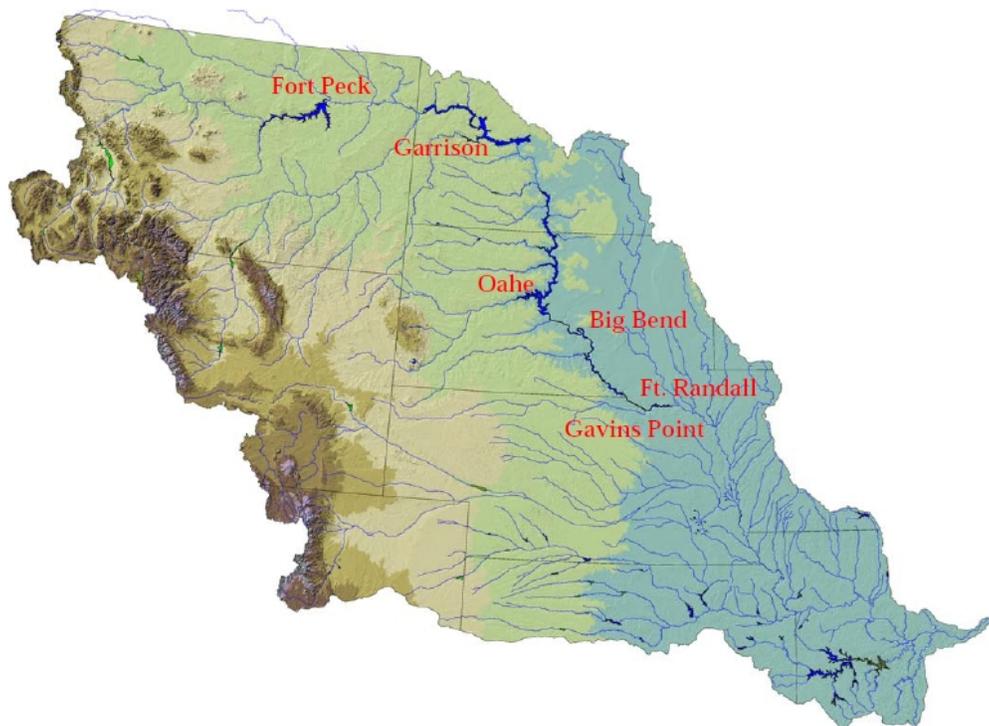


Figure 2-2: Missouri River Basin

## 2.2 MAINSTEM PROJECTS

The six Corps dams spanning the Missouri River control runoff from approximately half of the basin. Those six dams, from the upper three giants of Fort Peck (FTPK) in eastern Montana, Garrison (GARR) in central North Dakota and Oahe (OAHE) in central South Dakota, to the lower three smaller reservoirs of Big Bend (BEND) and Fort Randall (FTRA) in South Dakota, and Gavins Point (GAPT) along the Nebraska-South Dakota border, comprise the largest system of reservoirs in the United States. Four of the System reservoirs were named by the Congress: Lake Sakakawea (Garrison Dam); Lake Sharpe (Big Bend Dam); Lake Francis Case (Fort Randall Dam); and Lewis and Clark Lake (Gavins Point Dam). The reservoirs have a combined capacity of over 72.4 MAF. The System storage capacity is divided into four unique storage zones for regulation purposes, as shown in Figure 2-3. The Permanent Pool Zones are intended to remain permanently filled with water to ensure the maintenance of minimum power heads, minimum irrigation diversion levels, and minimum reservoir elevations for water supply, recreation, and fish and wildlife purposes. The Carryover Multiple Use Zones are intermediate zones that provides a storage reserve for irrigation, navigation, power production, water supply, recreation, and fish and wildlife during extended droughts. The Annual Flood Control and Multiple Use Zones provide storage for the annual capture and retention of normal and flood runoff and for annual multiple-purpose regulation of this impounded water. The Exclusive Flood Control Zones are reserved exclusively for regulation of the largest of floods and are generally empty. Figure 2-4 shows a profile of the mainstem projects, including the elevations of the projects and locations in river miles above the mouth of the Missouri River near St. Louis and also displays the relative proportion of storage capacity in each of the projects.

# Missouri River Mainstem System Storage Zones and Allocations

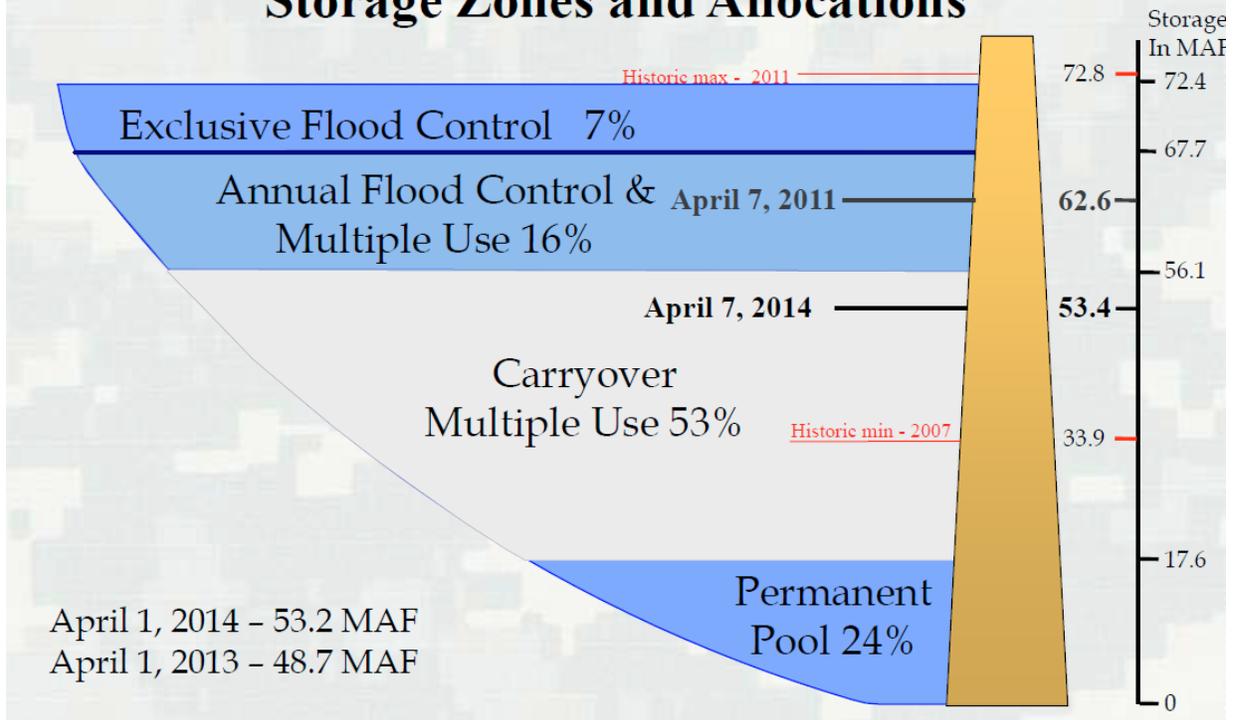
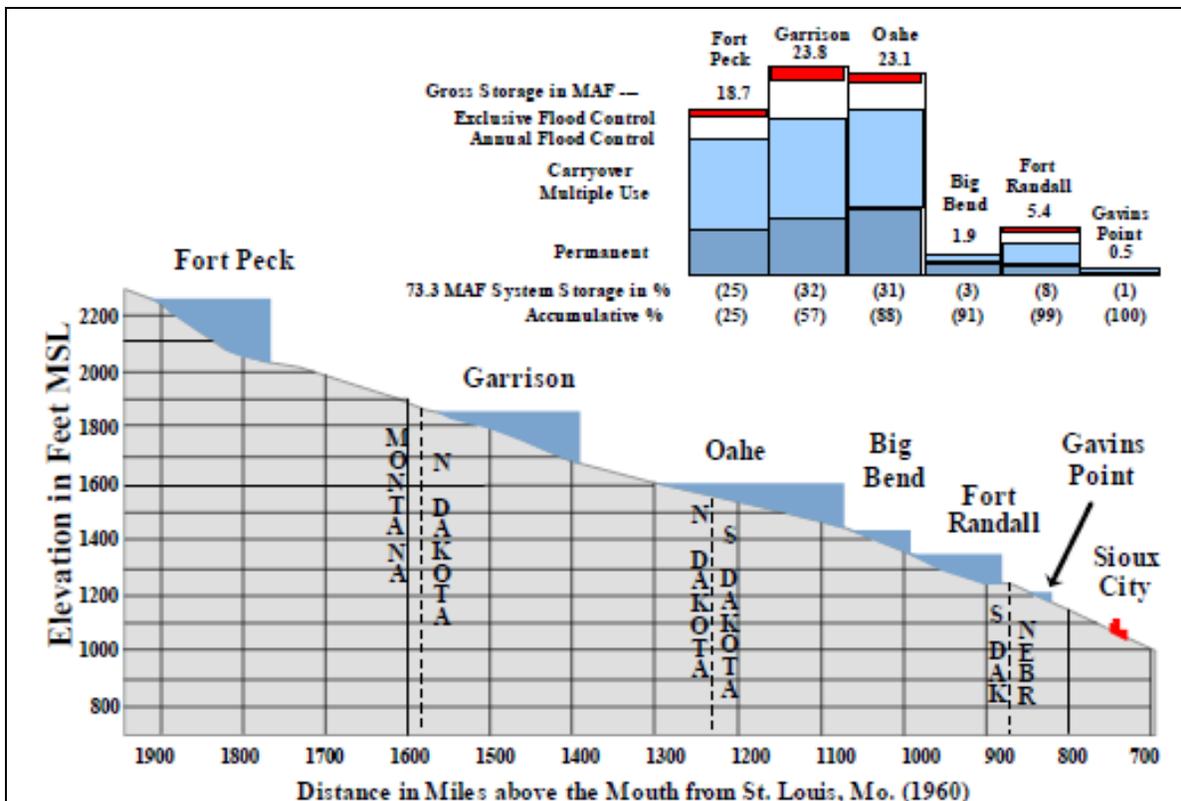


Figure 2-3: System storage zones.



**Figure 2-4: Profile of mainstem System and storage capacities**

The storage capacity of the six individual reservoirs ranges from over 23 MAF at Garrison and Oahe, to less than 0.5 MAF at Gavins Point as shown in Figure 2-4. The System is also unique in the fact that 88 percent of the combined storage capacity is in the upper three reservoirs of Fort Peck, Garrison, and Oahe. As a result, these three projects experience the bulk of the impacts during periods of very high runoff or extended drought. The lower three projects, Big Bend, Fort Randall, and Gavins Point, are regulated in much the same manner year after year regardless of the runoff conditions.

The individual projects are described briefly in the following sections, from upstream to downstream. Individual project descriptions were taken from the *Water Control Manual (WCM) Master Manual (Volume 1)* (U.S. Army Corps of Engineers, 2006). More detailed information on each project can be found in the *Water Control Manual (WCM)* for that specific project (Volumes 2-7). Other pertinent data for all projects are presented in the Summary of Engineering Data shown in Appendix A – Pertinent Data.

### 2.2.1 Fort Peck

Fort Peck Dam is located on the Missouri River at river mile (RM) 1772 in northeastern Montana, 17 miles southeast of Glasgow, Montana and 9 miles south of Nashua. Construction of the Fort Peck project was initiated in 1933 and embankment closure was made in 1937. The Fort Peck Dam embankment is nearly 4 miles long (excluding the spillway) and rises over 250 feet above the original streambed. Fort Peck Dam remains the largest dam embankment in the

United States (126 million cubic yards of fill), the second largest volume embankment in the world, and the largest “hydraulic fill” dam in the world. Fort Peck Lake is the third largest Corps reservoir in the United States. When full, the reservoir is 134 miles long. The concrete spillway is over 1 mile long. In 1943, the first unit of the power installation went on line, and the third unit became operational in 1951, completing construction of the first powerplant. Construction of a second powerplant began in the late 1950’s and the two units of this plant became operational in 1961. The Permanent Pool Zone (inactive storage) of the reservoir was initially filled (elevation 2150) in April 1942 and the Carryover Multiple Use Zone (elevation 2234) first filled in 1947, 5 years later. Drought conditions during the late 1950’s, combined with withdrawals to provide water for the initial fill of other System projects, resulted in a drawdown of the reservoir level to elevation 2167.4 in early 1956, followed by a generally slow increase in pool elevation. The Carryover Multiple Use Zone was finally refilled in July 1964. Generally, it has remained filled from that time with the exception of the droughts of 1987 to 1993 and 1999 to 2010. Exclusive flood control storage space was first used in 1969, and then again in 1970, 1975, 1976, 1978, 1979, 1996, 1997, and 2011.

### **2.2.2 Garrison**

Garrison Dam is located in central North Dakota on the Missouri River at RM 1390, about 75 river miles northwest of Bismarck, North Dakota and 11 miles south of the town of Garrison, North Dakota. Construction of the project was initiated in 1946, closure was made in April 1953, and the navigation and flood control functions of the project were placed in operation in 1955. Garrison Dam is currently the fifth largest earthen dam in the world. The first power unit of the project went on line in January 1956, followed by the second and third units in March and August of the same year. Power units 4 and 5 were placed in operation in October 1960. Lake Sakakawea first reached its minimum operating level in late 1955. Due to the drought conditions it was not until 10 years later, in 1965, that the Carryover Multiple Use Zone was first filled. Generally, it remained filled from that time through 2002, except for the two drought periods to date. Exclusive flood control storage space was used in 1969, 1975, 1995, 1997, 2010 and 2011. Lake Sakakawea is the largest Corps reservoir. When full, the reservoir is 178 miles long and up to 6 miles wide. The reservoir contains almost a third of the total storage capacity of the System, nearly 24 MAF, which is enough water to cover the State of North Dakota to a depth of 6 inches.

### **2.2.3 Oahe**

The Oahe Dam is located on the Missouri River at RM 1072, 6 miles northwest of Pierre, South Dakota. Construction of Oahe Dam was initiated in September 1948. Closure of the dam was completed in 1958, and deliberate accumulation of storage was begun in late 1961, just before the first power unit came on line in April 1962. The last of the seven power units became operational in July 1966. Permanent Pool storage space in Lake Oahe was first filled in 1962 and the Carryover Multiple Use Zone was filled in 1967. Generally, the Carryover Multiple Use Zone remained filled from that time through 2002, except for seasonal drawdowns in the interest of increased winter power generation and the two drought periods to date. The Exclusive Flood Control Zone in Lake Oahe was used in 1975, 1984, 1986, 1995, 1996, 1997, 1999, 2010 and

2011. Lake Oahe is the second largest Corps reservoir, with just over 23 MAF of storage capability. When full, the reservoir is 231 miles long, with 2,250 miles of shoreline.

### **2.2.4 Big Bend**

Big Bend Dam is located on the Missouri River at RM 987, near Fort Thompson, South Dakota and about 20 miles upstream from Chamberlain, South Dakota. Lake Sharpe extends 80 miles upstream to the vicinity of the Oahe Dam. The project is basically a run-of-the-river power development with regulation of flows limited almost entirely to daily and weekly power operations. Construction began in 1959, with closure in July 1963. The first power unit was placed on line in October 1964, and the last of the eight units began operation during July 1966. Since full operation began, the reservoir has been held very near the normal operating level of elevation 1420.

### **2.2.5 Fort Randall**

Fort Randall Dam is located on the Missouri River at RM 880, about 6 miles south of Lake Andes, South Dakota. Lake Frances Case extends to Big Bend Dam. Construction of the project was initiated in August 1946, closure was made in July 1952, initial power generation began in March 1954, and the project reached an essentially complete status in January 1956, when the eighth and final unit of the 320,000-kilowatt installation came into service. The reservoir filling was initiated in January 1953 and reached the minimum operating pool elevation of 1320 feet on November 24, 1953.

### **2.2.6 Gavins Point**

Gavins Point Dam is located on the Missouri River at RM 811 on the Nebraska-South Dakota border, 4 miles west of Yankton, South Dakota. Lewis and Clark Lake extends 37 miles to the vicinity of Niobrara, Nebraska. Construction was initiated in 1952, and closure was made in July 1955, with initial power generation beginning in September 1956. The third and final unit of the 100,000-kilowatt installation came into service in January 1957. Total project power generation has since been revised to 132,000 kilowatts.

## **2.3 MISSOURI RIVER MAINSTEM SYSTEM OPERATIONS**

The Missouri River mainstem system is very large and complex. The following sections summarize the mainstem System operation components as described in *Missouri River Mainstem Reservoir System: System Description and Regulation* (U.S. Army Corps of Engineers, 2007), but do not provide every detail of the Master Manual and individual project WCM's.

### **2.3.1 System Regulation**

#### **2.3.1.1 Overview**

The System is regulated to serve the congressionally authorized purposes of flood control, navigation, hydropower, irrigation, water supply, water quality control, recreation, and fish and wildlife. Overall System regulation follows the “water control plan” presented in the Master

Manual. Each of the six System dams also has an individual water control manual that presents more detailed information on its regulation. System regulation is in many ways a repetitive annual cycle. Most of the year's water supply is produced by runoff from winter snows and spring and summer rains which increase System storage. After reaching a peak, usually during July, System storage declines until late in the winter when the cycle begins anew. A similar pattern may be found in releases from the System, with the higher releases from mid-March to late-November, followed by low rates of winter discharge from late-November until mid-March, after which the cycle repeats.

The Water Control Calendar of Events, shown in Figure 2-5, displays the time sequence of many of these cyclic events. The water control plan is designed to achieve the multipurpose objectives of the System given these cyclical events. The two primary high-risk flood seasons shown are the plains snowmelt season, which extends from late February through April, and the mountain snowmelt period, which extends from May through July. Runoff during both of these periods may be augmented by rainfall. The winter ice-jam flood period extends from mid-December through February. The highest average power generation period extends from mid-April to mid-October, with high peaking loads during the winter heating season (mid-December to mid-February) and the summer air conditioning season (mid-June to mid-August). The normal 8-month navigation season extends from April 1st through November 30th during which time System releases are scheduled, in combination with downstream tributary flows, to meet downstream target flows. Winter releases after the close of navigation season are much lower, and vary depending on the need to conserve or evacuate System storage while managing downstream river stages for water supply given ice conditions. Minimum release restrictions and pool fluctuations for fish spawning management generally occur from April through June. Gavins Point spring pulses, which are designed to cue spawning of the endangered pallid sturgeon, have been provided in March and May of some years. Nesting of the two Federally protected bird species, the endangered interior least tern and the threatened piping plover, occurs from early May through mid-August.

Generally speaking, the System has three seasons per year. The Navigation season typically runs Apr. 1 to Dec. 1. The winter season goes from Dec. 1 to March 1. The open water non-navigation season includes the Mar. 1 to April 1 timeframe and may also include time between a shortened navigation season and the winter season.

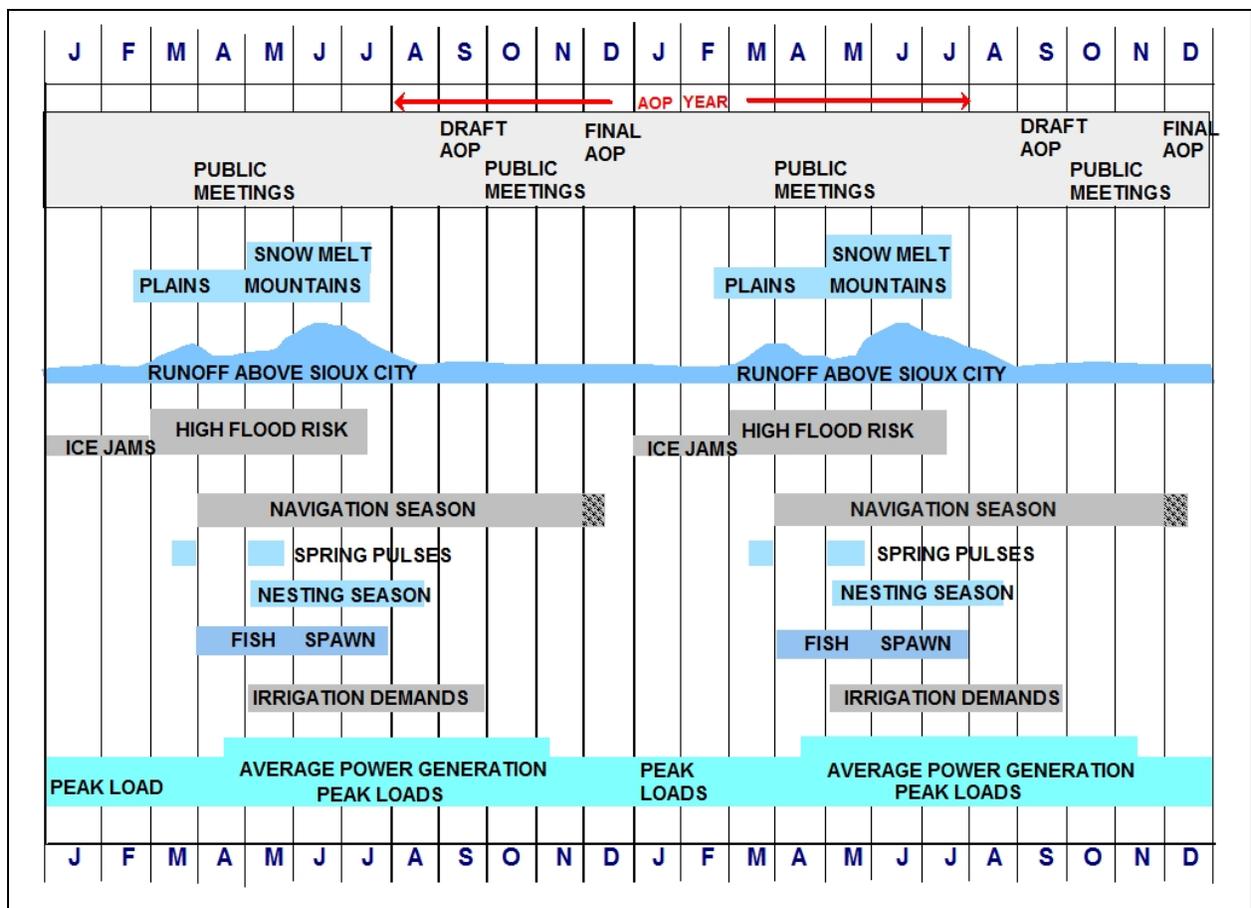


Figure 2-5: Water control calendar of events.

**2.3.1.2 Intrasystem Regulation - General**

Much of the flexibility of the System is derived from intrasystem regulation, or the transfer of water from one reservoir to another. This is due to the fact that System releases necessary to support downstream water requirements are defined within a relatively narrow range and inflow to the System is subject to only very minor regulatory control by upstream tributary reservoirs.

Intrasystem regulation is an important tool in the management of water in the System to meet the authorized purposes. It is used to regulate individual reservoir levels in the System to balance or unbalance the water in storage at each project, to smooth the annual System regulation by anticipating unusual snowmelt runoff, to maintain the seasonal capability of the hydropower system, and to improve conditions for the reservoir fish spawn and recruitment. It also can be used to maintain stages on the open river reaches between projects at desirable levels. Intrasystem adjustments may also be used to meet emergencies, including the protection of human health and safety, protection of significant historic and cultural properties, or to meet the provisions of applicable laws including the Endangered Species Act. These adjustments are made to the extent reasonably possible after evaluating impacts to other System uses, are generally short term in nature, and continue only until the issue is resolved.

The presence of large reservoirs in the System increases intrasystem regulation flexibility. A small reservoir such as Gavins Point with storage of less than one-half million acre-feet can only tolerate a large difference between inflow and release for less than a day. Big Bend is in this category as well. To a lesser extent, so is Fort Randall, although its carryover-multiple use and annual flood control and multiple use storage of nearly 3 MAF make possible significant storage transfers and flow differentials extending a month or more. But it is the upper three large reservoirs of Fort Peck, Garrison, and Oahe, with their combined 37.3 MAF of carryover multiple-use storage plus an additional 10.1 MAF of annual flood control multiple-use storage, that provide the flexibility to adjust intrasystem regulation to better serve authorized purposes.

### **2.3.1.3 Seasonal Intrasystem Regulation Patterns**

Factors that influence intrasystem regulation may vary widely from year to year; however, regulation of the System generally follows a regular seasonal pattern. Some of these factors, such as the amount of System storage and the magnitude and distribution of inflow received during the year, can affect the timing and magnitude of individual System project releases. The levels of each of the six System reservoirs are checked on a daily basis and compared to the water control plan and the AOP. Adjustments to the amount of water transferred between reservoirs are made when necessary to achieve the desired volume of water in each project and to maximize power generation.

#### *2.3.1.3.1 Summer Release Patterns*

Intrasystem regulation to meet the needs of power generation follows a regular seasonal cycle. Releases from Gavins Point are generally at their highest during the navigation season when downstream flow requirements are highest. Since Gavins Point reservoir is small, these releases must be backed up with similar magnitude releases from Fort Randall, and Fort Randall, in turn, requires similar support flows from Oahe via Big Bend. Here the chain can be interrupted; Oahe is large enough to support high releases for extended periods without high inflows. Generation at Fort Peck and Garrison are held to lower levels during the summer to allow more winter hydropower production unless the evacuation of water accumulated in the flood control zones or the desire to balance or unbalance storage among the upper three projects becomes an overriding consideration.

#### *2.3.1.3.2 Winter Release Patterns*

With the onset of the non-navigation season, conditions are reversed. Gavins Point releases drop to about one-third to slightly greater than half of summer levels and the chain reaction proceeds upstream, curtailing daily average discharges from Fort Randall, Big Bend, and Oahe. At this time, Fort Peck and Garrison daily releases are usually maintained at relatively high levels (within the limits imposed by downstream ice cover) to partially compensate for the reduction of generation downstream where high winter releases could result in significant flood damages in urban areas when the formation of ice impedes the flow.

#### 2.3.1.3.3 *Drawdown of Fort Randall Reservoir*

An additional means of partially compensating for the reduced hydropower generation associated with the lower winter release rate from Gavins Point is the autumn drawdown of Fort Randall reservoir. In this regulation, releases from Oahe and Big Bend are reduced several weeks before the close of the navigation season. This leaves Fort Randall with the task of supplying a large portion of downstream flow requirements for the remainder of the navigation season, a process that results in evacuation of a portion of its carryover storage space. This vacated carryover storage space, or recapture storage capacity, is then refilled with higher releases from Oahe and Big Bend during the non-navigation season, allowing winter releases from the upstream projects to substantially exceed those from Fort Randall.

Fort Randall reservoir is normally drawn down to 1337.5 feet msl, which provides about 1,200,000 acre-feet of recapture storage capacity. During severe drought periods, the flexibility exists to draw down to elevation 1320.0 feet msl. This provides an additional recapture space of about 800,000 acre-feet and increases the average winter energy generation about 150 million kilowatt hours (kWh).

#### 2.3.1.3.4 *Recapture at Oahe*

While not as significant (in terms of pool level fluctuation) as the drawdown and recapture regulation plan at Fort Randall reservoir, a similar recapture regulation plan at Oahe is coordinated with upstream Garrison and Fort Peck releases to significantly increase the amount of winter energy generation. During the 4-month winter period, Garrison releases normally can be expected to be at least 1 MAF more than Oahe releases. Recapture of these upstream releases generally results in a rise of about 5 feet or greater in Oahe reservoir elevation during the winter months, depending on the current storage level and whether the upper three reservoirs are intentionally unbalanced.

#### 2.3.1.3.5 *Balancing/Unbalancing the Upper Three Reservoirs*

In the past, the volume of water stored in each of the upper three reservoirs was balanced at the beginning of March of every year. However, intentionally unbalancing the water stored in the upper three reservoirs can benefit the reservoir fisheries and increase tern and plover habitat. Unfortunately, drought conditions and other reasons have prevented implementation of reservoir unbalancing.

#### **2.3.1.4 Short-Term Intrasystem Adjustments**

The interaction among projects described above, repeated as it is year after year, might make intrasystem regulation appear to be a routine and rigid procedure. However, routine regulation is often disrupted by the short-term extremes of nature. Heavy rains may raise river stages near the flood level, necessitating a release reduction at one project and a corresponding increase at others. Very hot or very cold weather may create sharp increases in the demand for power. Inflows for a week or for a season may concentrate disproportionately in one segment of the System, causing abrupt shifts in regulating objectives. In addition, short-term intrasystem adjustments are occasionally required to meet emergencies, including the protection of human health and safety, protection of significant historic and cultural properties, or to meet the provisions of applicable laws, including the Endangered Species Act. These adjustments are

made to the extent possible after evaluating impacts to other System uses, are generally short term in nature, and continue only until the issue is resolved.

### **2.3.1.5 Project Release Limits**

Limitations imposed upon System regulation (maximums and minimums) are related not only to System or individual project storage, which is varied in accordance with the flood control restrictions previously given and the requirements for active storage pools, but also to releases.

#### *2.3.1.5.1 Maximum Rates – Summer*

During the summer, releases at all projects other than Gavins Point are normally within the powerplant discharge capacity, the river channel downstream usually being more than adequate to carry such releases. Discharges from all projects will usually be made through the powerplant. At times, support for the downstream navigation or spawning cue flows may require releases from Gavins Point in excess of powerplant capacity. At all projects, special regulation considerations may require releases bypassing the powerplants but usually for only relatively short periods of time. Unusually large inflows during any particular year may require significant releases beyond those through the powerplants at any or all projects to evacuate flood waters and thereby maintain the future flood control capability of the System.

#### *2.3.1.5.2 Maximum Rates – Winter*

Releases are more restricted during the winter period. An ice cover can be expected to form over major portions of the Missouri River every winter and occasionally as far downstream as the river's mouth. During and after formation, this ice cover significantly reduces the discharge capacity of the river channel. In addition, during periods of ice formation and subsequent breakup, a substantial risk of ice jam formation and associated flooding exists. The maximum allowable winter releases are those that will not significantly increase the probability of flooding or intensify potential flooding during periods of ice cover. In the upper Missouri River, releases may be limited during periods of ice formation and then gradually increased once a stable ice cover is in place. Once formed, the ice cover can be expected to remain through the winter. Below Sioux City, ice formation or ice breakup can occur repeatedly throughout the season and may also jeopardize downstream navigation structures such as dikes and revetments. Since the travel time of any release from the System to areas of vulnerability is much longer than the time for which reliable forecasts of such events can be made, it is necessary to schedule winter System releases at a conservative level. During periods of normal or below water supply, winter releases from Gavins Point range from less than 9,000 cubic feet per second (cfs) to 17,000 cfs. During years with low winter releases, ice formation can result in significant stage reductions on the lower river; therefore, it is often prudent to increase System releases prior to the onset of river ice forming or even during a significant jam to maintain adequate stages at water intakes. Experience during recent years indicates that increasing System releases speeds the recovery of the river to more normal stages and assures that the downstream water intakes are operational sooner or affected less by the icing conditions. The maximum daily winter release from Gavins Point usually ranges between 12,000 and 25,000 cfs. With an excess water supply and evacuation of flood control storage space as a primary consideration, an average Gavins Point release rate of between 25,000 and 30,000 cfs is scheduled. The extent and location of river ice cover is important in determining the release rate. Experience

accumulated during past winters indicates that at times it may be necessary to reduce System releases below these levels when bankfull to slightly above bankfull stages occur in the Nebraska City to St. Joseph reach of the Missouri River.

No daily release limitations exist at Big Bend, where discharges are made almost directly into the downstream reservoir area. The maximum ice-covered channel capacity below Fort Peck and Garrison are estimated to be about 15,000 and 27,000 cfs, respectively, except during ice formation. Releases are limited to lower levels while the river initially freezes because the ice cover is usually rough and jagged, which creates a less efficient channel and causes river stages to increase. Releases are increased once the ice cover and streambed have stabilized and both have smoothed sufficiently to accommodate increased releases without increasing river stages. Winter releases from Fort Randall are generally 1,000 to 2,000 cfs lower than those from Gavins Point, but during periods of ice formation may be scheduled at or slightly higher than Gavins Point releases to prevent rapid declines in the Gavins Point pool elevation. At Oahe, peak hourly releases may be constrained to prevent urban flooding in the Pierre and Fort Pierre areas if severe ice conditions develop below the project.

#### 2.3.1.5.3 *Minimum Releases*

There are no minimum daily flow requirements from Oahe or Big Bend except that, to the extent possible, weekend releases from Oahe are typically held above 3,000 cfs during the daytime hours of the recreation season in the interest of downstream fishing and boating. In addition, during periods of ice formation a one-unit minimum may be imposed at Oahe to prevent ice formation in the channel directly below the dam. Minimum daily releases from Fort Peck and Fort Randall are typically maintained during the fish spawning seasons. Fort Peck also has a year-round instantaneous minimum release of 3,000 cfs for the trout fishery below the dam, though to the extent possible, releases are maintained above 4,000 cfs. During periods of high inflow below the project, releases may be scheduled below 3,000 cfs for flood damage reduction, but these instances are rare. Minimum daily releases at Fort Peck, Garrison, Fort Randall, and Gavins Point are established as those necessary to supply water quality control and downstream water intake requirements, which generally also furnish more than an adequate quantity of water for irrigation withdrawals below the reservoirs. At Garrison a minimum average daily release of 9,000 cfs has been established as a guide to provide for downstream intakes. Access problems have been experienced at municipal, industrial, powerplant, and irrigation intakes along the length of the river due to channel degradation, inadequate intake screens, sandbar formation, winter ice formation, or relatively high elevation of the intakes. Temporary increases above the open-water minimum release rates may be made to the extent reasonably possible to allow intake owners to take remedial action.

#### 2.3.1.5.4 *Hourly Fluctuation of Release Rates*

At all projects except Gavins Point, hourly release rates may vary widely as necessary to meet fluctuating power loads. Changes in release rates at Gavins Point are subject to limitations to restrict stage fluctuations downstream. Minimum hourly release restrictions are applicable at Fort Peck and Garrison due to downstream intakes. A uniform peaking release pattern has been established during the summer months at Garrison and Fort Randall for endangered birds

nesting along the river below the projects, and may be reinstated at Fort Peck if nesting patterns deem it necessary.

### ***2.3.2 Recurring Operational Considerations***

#### **2.3.2.1 Flood Control**

Flood control is the only authorized project purpose that requires the availability of empty storage space rather than impounded water. Actual flood events are generally unpredictable; therefore, detailed routing of specific major flood flows is accomplished when floods occur. There is a recurring pattern of high-risk flood periods during each year: a season when snowmelt, ice jams, and protracted heavy rains will almost surely occur with or without generating consequent floods; and a season when these situations are most unlikely and the flood threat is correspondingly low. The high-risk flood season begins about March 1st and extends through the summer. As a consequence, regulation of the System throughout the fall and winter months is predicated on the achievement of a March 1st System storage level at or below the base of the annual flood control zone. Exceptions to this will occur due to the availability of replacement flood control storage in major upstream tributary reservoirs.

Due to release limitations imposed by the formation of a downstream ice cover, a major portion of the required flood control space in the System must be evacuated prior to the winter season. Gavins Point winter releases exceeding 25,000 cfs are not normally scheduled. In general, individual System projects will also be scheduled to be near or below their respective base of annual flood control by March 1st. Some departure is possible due to the availability of upstream tributary flood control storage space, intrasystem unbalancing of the upper three reservoirs to benefit the reservoir fishery and the protected species, and/or recognition of the relative ease by which the water in storage may be transferred downstream to other projects in the System even during the flood season.

During all but excessively dry years, water stored in the reservoirs will increase during the March-July period. The base of exclusive flood control defines the maximum level of storage that will be accumulated for purposes other than flood control. Water stored in the annual flood control and multiple-use zones will normally be released through the powerplant of each of the individual projects except when evacuation of this zone prior to the winter season necessitates higher flow rates requiring flood control outlet tunnel or spillway releases. When the exclusive flood control zone in a particular reservoir is encroached upon, the control of subsequent flood inflows becomes the paramount factor. During such periods, releases may substantially exceed the powerplant release capacity with the evacuation rate of any project dependent upon existing flood conditions, the potential for further inflows, and conditions of other reservoirs in the System. Maximum release rates at such times are based upon the Master Manual flood control criteria and the flood control status of the System. Detailed information regarding the adjustment of service levels for flood control evacuation and downstream flood control constraints can be found in Chapter 7 of the Master Manual.

Below Fort Peck, minor downstream flooding will occur when open-water flows exceed 35,000 cfs. Open-water channel capacity below each of the other reservoirs was approximated at 100,000 cfs or more at the time the reservoirs were constructed (1950's). In

addition, releases may need to be reduced to less than the immediate downstream channel capacity due to uncontrolled actual and potential tributary flows below each project, particularly below Gavins Point, Garrison, and Fort Randall.

Guidance from the Daily Routing Model (DRM) and discussions with the Missouri River Basin Water Management (MRBWM) team suggested channel capacities listed in the rules tables in Section 4.3.2.

### **2.3.2.2 Water Requirements Below Gavins Point**

Just as the water supply and upstream uses must be evaluated each year to determine the net supply into the System, so must System release rates be established. This is the only means of regulating the System storage, since the weather and its resultant effects are not subject to control. Daily releases from Gavins Point, commonly referred to as the System releases, fall into two classes. Open-water releases, generally in the range of 21,000 to 35,000 cfs, are made in support of Missouri River navigation and other downstream uses. In years with above-normal water supply or extended periods of downstream flooding, the navigation releases are increased to the extent necessary to evacuate the flood control storage space by the succeeding March, with due consideration of reduced channel capacities during the winter ice-cover period. System releases during the non-navigation season generally range from 9,000 to 30,000 cfs, and are made for water supply, water quality control, power production, and flood evacuation purposes.

#### *2.3.2.2.1 Navigation Season Requirements*

The Missouri River navigation channel extends for 734.8 miles from near Sioux City, Iowa (River Mile 732.3) to the mouth (River Mile 0) near St. Louis, Missouri. Navigation on the Missouri River is limited to the normal ice-free season with a full-length season normally extending from April 1st through November 30th at the mouth. To permit a viable navigation industry during the ice-free months, it is desirable to maintain navigable flows throughout this 8-month period. During past navigation seasons in years of adequate water supply, 10-day extensions either at the beginning or end of this normal season have been scheduled, downstream river ice conditions permitting.

Construction of the navigation works was declared complete in September 1981. In years with adequate water supply, System releases are scheduled to provide adequate flows for navigation at the target locations of Sioux City, Omaha, Nebraska City, and Kansas City, if navigation is occurring on the reaches associated with those targets. If navigation is not occurring in one or more upstream reaches, flows may be allowed to fall below the respective targets, depending on the needs of other authorized project purposes at the time. The target flows increase in a downstream direction because of the increased flow requirements needed to maintain corresponding navigation channel widths and flow depths with naturally increasing channel dimensions. The assignment of target flows is based upon available water supply that, when combined with winter releases needed to ensure water supply requirements and winter hydropower demand, obligates all of the available water supply during a normal year. These target flows may need to be evaluated and adjusted periodically to ensure compatibility between available water supply and current navigation channel conditions.

#### 2.3.2.2.2 *Navigation Service Level and Season Length*

As described in the Master Manual, flow support for navigation and other downstream purposes is defined based on service level. A “full-service” level of 35,000 cfs results in target flows of 31,000 cfs at Sioux City and Omaha, 37,000 cfs at Nebraska City and 41,000 cfs at Kansas City. Similarly, a “minimum-service” level of 29,000 cfs results in target flow values of 6,000 cfs less than the full- service levels.

Day-by-day regulation of the System to support navigation requires forecasts of inflow to the various river reaches below the System. These daily forecasts, along with anticipated navigation traffic or the absence of traffic in the various river reaches, are used to determine the target location (Sioux City, Omaha, Nebraska City, or Kansas City). After determining the target location, releases from the System are adjusted so that, in combination with the forecast tributary inflows, the resultant flow will meet the target flow at the control location. During periods when the target location is Kansas City, navigation flow support can also be provided from three Kansas basin reservoirs (Tuttle Creek, Milford, and Perry) since those projects are authorized to support Missouri River navigation. This regulation conserves water in the System and may also minimize incidental take of the Endangered Species Act (ESA)-protected species.

Regulation experience has shown that the full-service target flows will be adequate to maintain the designed 9-by-300-foot channel with a minimum of groundings and little or no emergency dredging. Slightly greater flows are required at the mouth (approximately 45,000 cfs) but tributary flows below Kansas City are usually adequate to provide the needed incremental flows.

Selection of the appropriate service level is based on the actual volume of System storage on March 15th and July 1st of each year. During years when flood evacuation is required, the service level is calculated monthly, or more frequently if required, to facilitate a smooth transition in System release adjustments.

The water control plan calls for suspension of Missouri River navigation if System storage is at or below 31 MAF on March 15th of any year. It should be noted that the occurrence of System storage at or below 31 MAF would likely coincide with a national drought emergency.

Assuming the System storage is above 31 MAF on March 15th, a navigation season will be supported. The System storage check for navigation season length is made on July 1st of each year. A full 8-month navigation season will be provided if System storage is 51.5 MAF or above on July 1st, unless the navigation season is extended to evacuate flood control storage. However, if System storage falls below 51.5 MAF on July 1st, a shortened navigation season will be provided to conserve water.

The System release required to meet minimum- and full-service target flows varies by month in response to downstream tributary flows. In general, higher releases are needed to meet flow targets during years with below normal runoff in the upper basin than during years with higher upper basin runoff. The target location early in the season is generally at Sioux City with adequate tributary flows meeting the other downstream flow targets. Tributary flows normally

decrease during the summer and the target location moves from Sioux City to Nebraska City, and then to Kansas City as the runoff season progresses. This requires higher releases from the System as the season progresses through summer. Often the target location moves upstream during the fall as downstream tributary flows traditionally increase. With normal inflows below the System, Sioux City flows will average about 35,000 cfs over the entire 8-month navigation season during periods when full-service navigation targets are utilized for System regulation.

#### 2.3.2.2.3 *Release Patterns during Nesting Season*

In general, releases from Gavins Point are adjusted as needed to meet target flow levels on the lower Missouri River, taking advantage of downstream tributary runoff. However, during the nesting season of the endangered interior least tern (tern) and the threatened piping plover (plover), care must be taken to avoid impacts to nesting areas. These two bird species are listed as threatened and endangered under the ESA and are protected under that Act. Several scenarios have been used in past years to regulate the System during the nesting season. Under the Steady-Release (SR) scenario, when the birds begin to initiate nesting activities in early to mid-May, the release from Gavins Point is set to the level expected to be required to meet downstream flow targets through August and maintained at that level until the end of the nesting season. This regulation results in releases that exceed the amount necessary to meet downstream flow targets during the early portion of the nesting season, and may result in targets being missed if basin conditions are drier than expected during the summer.

Gavins Point releases under the Flow-to-Target (FTT) scenario are adjusted as needed throughout the nesting season to meet downstream flow targets and would typically result in increasing releases as the nesting season progresses. This is due to reduced tributary inflows downstream as the summer heat builds, evaporation increases, and precipitation wanes. Increasing releases as the nesting season progresses has the potential to inundate nests and chicks on low-lying emergent sandbar habitat. Compared to the SR scenario, this scenario conserves more water in the System, which keeps the reservoirs at the upper three System projects at relatively higher levels. However, this scenario also increases the risk of inundating nests. The FTT scenario also ensures that targets on the lower river are met throughout the nesting season.

A third scenario for Gavins Point releases combines features of the other two options. This scenario, called the Steady Release - Flow-to-Target (SR-FTT) scenario, sets Gavins Point releases at an initial steady rate and then allows releases to be adjusted upward or downward during the nesting season to meet downstream flow targets, if necessary. Depending on the rate of the initial steady release, this regulation makes a larger amount of habitat available early in the nesting season and saves additional water in the upper three reservoirs when compared to the SR scenario. The SR-FTT scenario also reduces the potential for flooding nests when compared to the FTT scenario. The SR-FTT regulation also provides certainty for downstream users that releases could be increased if needed to meet Missouri River flow targets.

Under each of these regulation scenarios, releases from Gavins Point may be increased every third day to encourage terns and plovers to build their nests on higher habitat so that the nests would not be inundated later when higher releases are required to meet the regulation objectives of the System. This pattern of increasing releases every third day is referred to as “cycling”. Cycling is generally not used during years when System storage is high but has been used during extended drought, when water conservation is of primary importance. Cycling is suspended when endangered and threatened chicks hatch to reduce the risk of stranding chicks on low-lying sandbars. Unfledged chicks can be lost if stranded on low-lying sandbars that are subsequently totally inundated. Cycling of Gavins Point releases when releases are reduced for downstream flood control during the protected bird species nesting season has also been used to keep birds nesting at sufficiently high elevations to maintain room for release increases when downstream flooding has subsided. The daily variation in releases is normally limited to 8,000 cfs to minimize adverse affects on downstream river users and fish.

#### 2.3.2.2.4 *Non-navigation Season Requirements*

When releases are not being made for downstream flow support during navigation season, other factors, including water quality control and water supply, are used to establish the System release rates. System project release levels necessary to meet downstream water supply purposes generally exceed the minimum release levels necessary to meet minimum downstream water quality requirements. The minimum daily flow requirements established for water supply are designed to prevent operational problems at municipal and thermal powerplant intakes.

In years of excess inflows and storage, several options are utilized to evacuate flood control storage, including an extension of the navigation season, increased winter releases, and the provision of summer and fall releases above full service. Because releases above full service increase the risk of downstream flooding, the first option normally utilized is up to a 10-day extension of the navigation season. This increases the service to navigation by providing a longer season and to hydropower by increasing the amount of winter energy generation. If additional evacuation is required, winter releases are increased to evacuate flood control storage, and finally, the summer and fall service levels are increased. Increasing winter releases slightly increases the risk of minor ice-induced flooding; however, open-water flooding during the summer and fall has a higher flood damage potential because of the value of the agricultural crops on the floodplain at that time of year. Moderate increases above full-service requirements during the open-water summer and fall season can be beneficial to the navigation and power purposes.

With normal or below normal inflows and storage, conservation measures may be implemented that reduce navigation and hydropower releases during the open-water season based on System storage, and may provide less than full-service navigation flows and season lengths of less than 8 months as described previously. Winter System releases are also reduced as a drought conservation measure. The winter System release rate is determined based on a September 1st System storage check. This release rate in combination with average downstream tributary flows is normally sufficient to meet downstream water supply intake

requirements, but may be adjusted based on tributary flows and the potential for ice formation. In an extended drought, System releases from Gavins Point may be reduced to a level that results in only the minimum flows necessary for downstream water intake or water quality requirements. Based on typical downstream tributary flow contributions, the minimum releases are 9,000 cfs during the non-summer open-water season (March-April and September- November), 18,000 cfs during the summer open-water season (May-August), and 12,000 cfs during the winter period (December-February). These minimum releases for downstream water intakes are average values; actual releases may vary significantly from the listed values.

### **2.3.2.3 Water Requirements Above Gavins Point**

#### *2.3.2.3.1 Water Supply*

The minimum releases established for water supply are designed to prevent operational problems at municipal and thermal powerplant intakes at numerous locations along the Missouri River.

At Fort Peck, a minimum daily average release of 3,000 cfs is satisfactory for municipal water supply; however, instantaneous releases of no less than 4,000 cfs are normally scheduled for the coldwater fishery directly below the dam. During periods of high inflow below the project, releases may be scheduled below 3,000 cfs for flood damage reduction, but these instances are rare. To the extent possible, releases are maintained above 6,000 cfs during the irrigation season.

At Garrison, it is desirable to maintain minimum average daily releases of at least 9,000 cfs during the open-water season and the ice-cover season to provide sufficient river depths for continued operation of municipal, irrigation, and powerplant water intakes in North Dakota. In this reach of the river, as well as that below Fort Peck, fluctuations in release levels at times require the resetting of irrigation pumping facilities to maintain access to available water or to prevent inundation of pumps.

Mean daily releases of 1,000 cfs are considered to be adequate to meet all water supply requirements between Fort Randall and Gavins Point.

#### *2.3.2.3.2 Power Production*

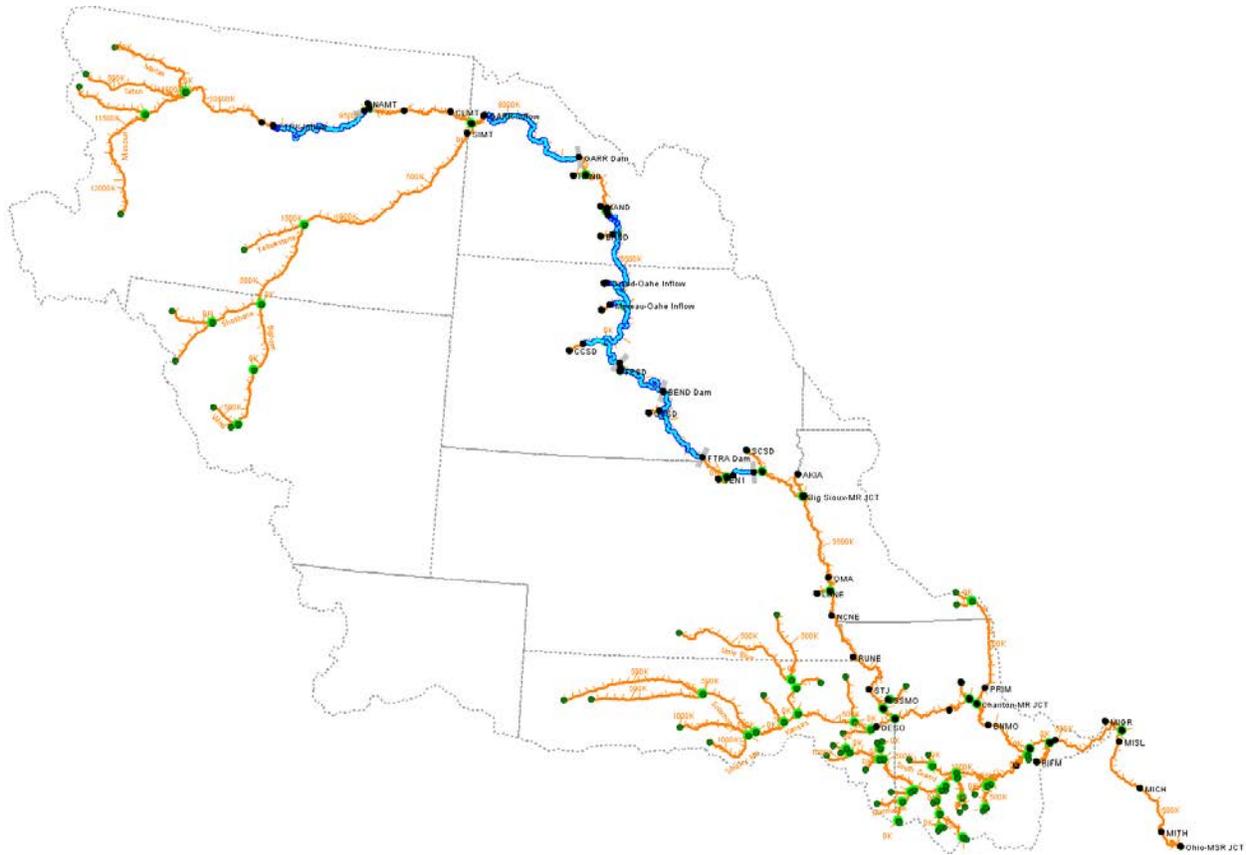
Since the completion of the power production facilities at the System projects, virtually all project releases have been made through the respective powerplants. When releases are exceptionally high due to flood control storage evacuation, spillway releases are necessary at Gavins Point and Fort Randall and on rare occasions at Fort Peck and Garrison. The six System dams support 36 hydropower units with a combined plant capacity of 2,501 megawatts (MW) of potential power generation. These units provide an average of 10 million megawatt-hours (MWh) of energy per year. WAPA markets hydroelectric energy and capacity from the System. Firm energy is marketed on both an annual and a seasonal basis, recognizing the seasonal pattern of releases made for navigation and required for flood control. During the navigation season, releases from the four uppermost reservoirs are varied in an effort to generate the greatest amount of energy at the times the power loads are the greatest. During

the winter period, the most critical with respect to maintaining load requirements, releases from Fort Peck and Garrison are scheduled at relatively high rates to compensate for reduced power production at the downstream powerplants. The fall drawdown at Fort Randall makes available space for recapture of winter power releases from upstream reservoirs.

### **3 DATA**

#### **3.1 GIS DATA**

GIS (ArcGIS 10) was used to create the ResSim stream alignment, computation points and reservoirs, and background maps (see Figure 3-1). The stream alignment was created from multiple sources. The Missouri and Mississippi Rivers were created by converting river mile point files (gathered from NWK and MVS, respectively) to polylines. Inflow tributaries were primarily digitized from background imagery from the mainstem to the first upstream tributary gage; however, the upper Missouri, Yellowstone, Kansas, Chariton, and Osage Rivers used a combination of source data and were extended using a more coarse river shapefile provided by MRBWM. The Kansas, Chariton, and Osage Rivers were primarily created by NWK. Stream alignments for ResSim were imported directly from the polylines shapefiles.



**Figure 3-1: ResSim stream alignment and computation points.**

ResSim model computation points were created for all gages, junctions, dams, and reservoir inflows in GIS. Missouri and Mississippi River stream gages and dams were located and placed by river mile. Confluence points were created based on the intersection of stream alignments. Mainstem reservoir inflow points were approximated based on information regarding the length of the reservoirs from the Master Manual, with a couple exceptions: BEND-Inflow was moved to 1 mi downstream of Bad River-MR Junction, and FTRA-Inflow moved to 1 mi downstream of BEND. The computation points were imported directly into the Watershed interface using HEC-WAT since ResSim does not currently have that tool. Table 3-1 provides the primary gages on the mainstem Missouri River, their identifying abbreviation, and their location. Figure 3-2 provides a GIS overview of the ResSim project study area.

**Table 3-1: Mainstem Gage ID and Locations.**

<b>DCP ID</b>	<b>Location</b>
HEMO	Missouri River at Hermann, MO
BNMO	Missouri River at Boonville, MO
WVMO	Missouri River at Waverly, MO
MKC	Missouri River at Kansas City, MO
STJ	Missouri River at St. Joseph, MO
RUNE	Missouri River at Rulo, NE
NCNE	Missouri River at Nebraska City, NE
OMA	Missouri River at Omaha, NE
SUX	Missouri River at Sioux City, IA
GAPT	Missouri River at Gavins Point
FTRA	Missouri River at Fort Randall
BEND	Missouri River at Big Bend
OAHE	Missouri River at Oahe
BIS	Missouri River at Bismarck, ND
GARR	Missouri River at Garrison
CLMT	Missouri River at Culbertson, MT
WPMT	Missouri River at Wolf Point, MT
FTPK	Missouri River at Fort Peck
RBMT	Missouri River at Landusky (Robinson Bridge), MT

# NWO ResSim Mainstem Model Computation Point and Stream Alignment Planning

## 04-10-2012

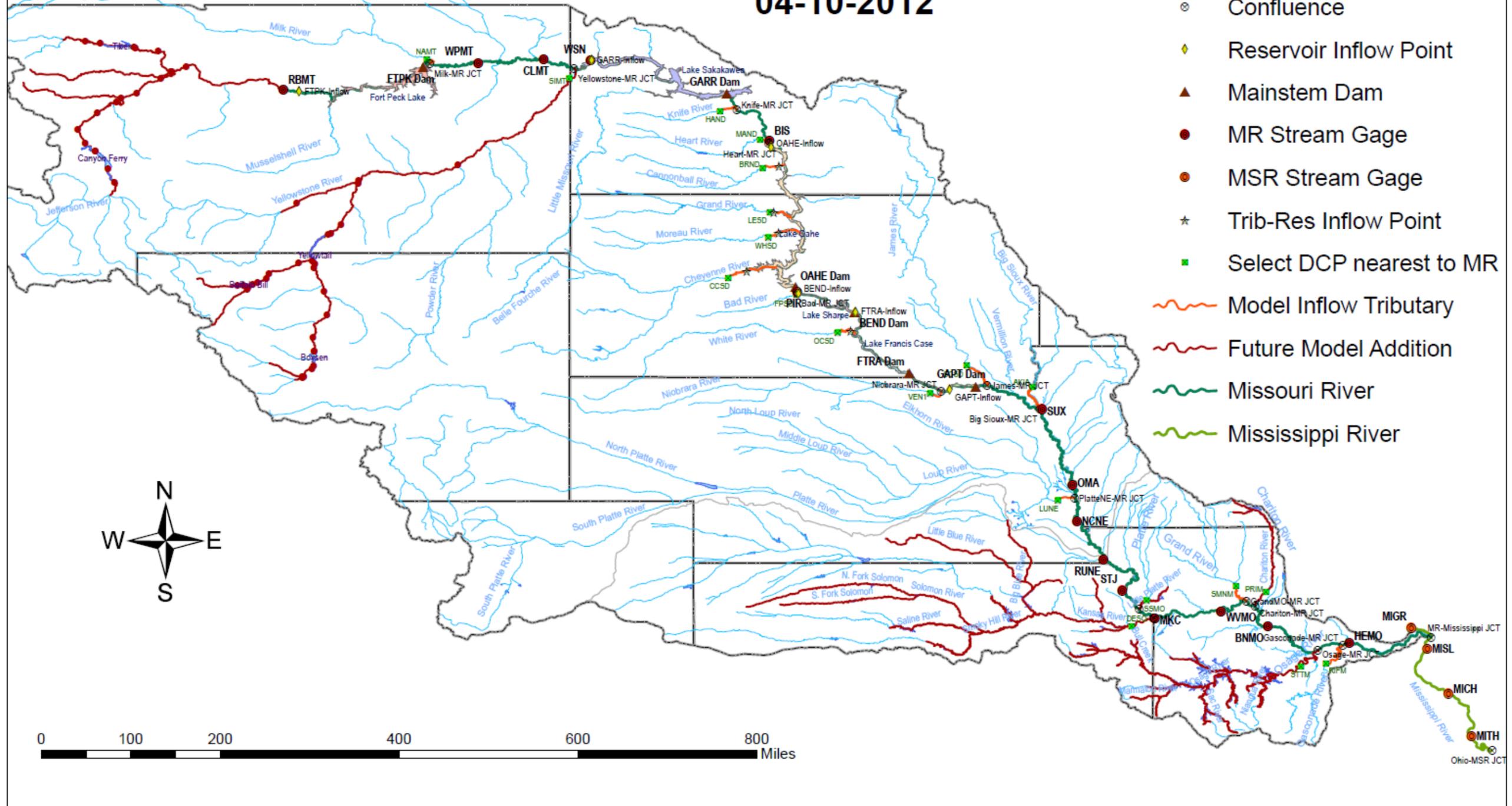


Figure 3-2: GIS study map.

### **3.2 OBSERVED DATA**

To create a complete discharge data set for the entire period of record (POR) from 1898-2011, several different sources of data (defined in the following sections) were necessary. USGS gage data and observed inflow/outflow data from MRBWM (MRRPPCS-REV data) were used when available. To fill in missing POR data upstream of Sioux City (SUX), data from the DRM was utilized. Two different DRM simulations were used: “No dams and no current depletions”, and “Observed”. The DRM data had to be used for the upstream reaches since the UMRSFFS data did not extend upstream of SUX prior to the completion of the reservoirs. The DRM “No dams and no current depletions” data was used at reservoir or gage locations prior to completion of dams or the start of USGS gages at the current or upstream locations. After 1 or more reservoirs or gages were completed at an upstream location, the DRM “Observed” data was used until the reservoir or gage at that location was complete. For all locations at SUX and downstream, the UMRSFFS “observed” data was used. It was felt the UMRSFFS data are more reliable than the DRM data, since the UMRSFFS data precisely match observed USGS gage data after the gages came online. The DRM “Observed” data do not match the USGS gage data quite as well, but were the only available data at the reservoir locations and gages upstream of SUX prior to system completion.

Final POR data for use in ResSim is stored in the DSS file “Input\_Data.dss”. The data used for the final POR construction at each gage is shown in Table 3-2.

**Table 3-2: Sources for POR data set construction.**

<b>Gage</b>	<b>Time Period</b>	<b>Source</b>
RBMT	1898-2011	Upper Missouri ResSim
FTPK	1898-1937	DRM, no dams no depletions
	1938-2011	MRBWM
WPMT	1898-1928	DRM, no dams no depletions
	1929-2011	USGS
CLMT	1898-1938	DRM, no dams no depletions
	1938-1941	DRM, observed
	1941-1952	USGS
	1952-1958	DRM, observed
GARR	1958-2011	USGS
	1898-1938	DRM, no dams no depletions
	1938-1954	DRM, observed
BIS	1954-2011	MRBWM
	1898-1927	DRM, no dams no depletions
OAHE	1928-2011	USGS
	1898-1929	DRM, no dams no depletions
	1930-1959	DRM, observed
PIR	1959-2011	MRBWM
	1933-1965	USGS
BEND	1898-1929	DRM, no dams no depletions
	1930-1963	DRM, observed
	1963-2011	MRBWM
FTRA	1898-1929	DRM, no dams no depletions
	1930-1952	DRM, observed
	1953-2011	MRBWM
GAPT	1898-1929	DRM, no dams no depletions
	1929-1955	DRM, observed
	1955-2011	MRBWM
SUX	1898-1928	UMRSFFS
	1929-1931	USGS
	1932-1938	UMRSFFS
	1938-2011	USGS
OMA	1898-1928	UMRSFFS
	1928-2011	USGS
NCNE	1898-1929	UMRSFFS
	1929-2011	USGS
RUNE	1898-1949	UMRSFFS
	1949-2011	USGS
STJ	1898-1928	UMRSFFS
	1928-2011	USGS
MKC	1898-1928	UMRSFFS
	1928-2011	USGS
WVMO	1898-1928	UMRSFFS
	1928-1977	USGS
	1977-1978	UMRSFFS
	1978-2011	USGS
BNMO	1898-1925	UMRSFFS
	1925-2011	USGS
HEMO	1898-1928	UMRSFFS
	1928-2011	USGS
MISL	1898-2011	USGS

### **3.2.1 USGS**

USGS data were considered the most accurate data option and were used for flow data at all locations and time periods for which it was available. USGS data was imported from the USGS website using HEC-DSSVue.

### **3.2.2 MRBWM**

The Missouri River Basin Water Management (MRBWM) Division data were used for reservoir inflow, outflow, storage, and energy data at each reservoir location after that reservoir was online. This was the best data available at reservoir locations, since the USGS does not calculate or gage reservoir inflows and outflows for these locations. Observed storage and energy data were not used by the ResSim model computations as an input data set, but were only used for model accuracy verification. MRBWM was previously called the Reservoir Control Center (RCC), and this old label may still appear on data file sets in this project.

### **3.2.3 UMRSFFS**

The *Upper Mississippi River System Flow Frequency Study* (UMRSFFS) was completed in 2004 by a task force consisting of team members from USACE, USGS, NWS, USBR, NRCS, FEMA, and the states of Minnesota, Wisconsin, Iowa, Illinois, Missouri, Kansas, and Nebraska. The study used unsteady flow models to update the flow frequencies for the Illinois River, the Upper Mississippi River mainstem, and the Missouri River below Gavins Point Dam. The daily flow data from this study were used in the ResSim model for locations downstream of Gavins Point dam when USGS data were not available. More information on modeling associated with the UMRSFFS can be found in the study report.

### **3.2.4 DRM**

The Daily Routing Model (DRM) was completed by MRBWM in 1997. It is used for long range forecasting and is generally only run by MRBWM once a year. DRM data was the only data available for most locations upstream of Gavins Point prior to the completion of the reservoirs. These flow data were used for locations and time periods when USGS, MRBWM, and UMRSFFS data were unavailable. More detail on the assumptions and operation of the DRM program can be found in the *DRM User's Manual (USACE, 1997)* and the *Mainstem Master Manual (2006)*.

### **3.2.5 USBR Depletions**

USBR provided estimates for irrigated agriculture, public surface water supply, USBR reservoir holdouts, and basin transfer depletions at an 8-digit Hydrologic Unit (HUC8) scale within the Missouri River basin. These estimates were split into two periods of water use: Historic Condition and Present Condition. Historic Condition depletions were categorized as an estimated amount of water removed from the system based on historical water usage; Present Condition depletions were categorized as an estimated amount of water that would have been removed from the system based on present water usage. The USBR HUC8 depletions were used to calculate local and total depletions at the DCP locations. Total depletions were the sum of all depletions upstream of a DCP location and local depletions were the incremental

depletions that occurred between two DCP locations. Since the USBR depletions were based on a HUC8 resolution and some of the DCP locations did not lie on a HUC8 boundary, ArcGIS was used to determine what percentage of a HUC8, in terms of drainage area, contributed to the DCP location. This percentage of drainage area was then used to factor the depletions of the HUC8 that contained the DCP to estimate the amount of HUC8 depletions that should be included at a DCP location to ensure depletions were not counted twice. Table 3-3 lists the factors associated with the DCP locations and corresponding HUC8's.

The USBR depletions, when added to the local flow datasets, produced very large negative inflows. These large negative inflows have a substantial impact on reservoir operations and can cause issues with other models. At the time of this report, a draft document was being completed that assessed the USBR depletions and how to apply them to local flow datasets in the Missouri River Basin.

**Table 3-3: HUC8 depletion adjustment factors based on DCP drainage area.**

DCP ID	Adjusted HUC8	Local Drainage Area (sq mi)	Total Drainage Area (sq mi)	Depletion Adjustment Factor	Note
RBMT	10040104	40,694	40,694	<b>0.16</b>	
FTPK	10040104	16,676	57,370	<b>0.84</b>	
WPMT	10060001	24,694	82,064	<b>0.79</b>	
CLMT	10060001	10,086	92,150	<b>0.21</b>	Contains a closed basin, which was removed from total drainage area
	10060005	10,086	92,150	<b>0.61</b>	
GARR	10060005	87,757	179,907	<b>0.39</b>	
BIS	#N/A	5,030	184,937	<b>#N/A</b>	Poor watershed delineation using 30m DEM. Approximately all of HUC8 10130101 is upstream of DCP
OAHE	#N/A	56,277	241,214	<b>#N/A</b>	USGS gage was downstream of dam
BEND	10140101	5,801	247,015	<b>0.26</b>	
FTRA	10140101	14,288	261,303	<b>0.74</b>	
GAPT	10170101	16,238	277,541	<b>0.59</b>	
SUX	10170101	33,908	311,449	<b>0.41</b>	
	10230001	33,908	311,449	<b>0.01</b>	
OMA	10230001	8,617	320,066	<b>0.99</b>	
	10230006	8,617	320,066	<b>0.34</b>	
NCNE	10230006	86,870	406,936	<b>0.66</b>	
	10240001	86,870	406,936	<b>0.72</b>	
RUNE	10240001	4,961	411,897	<b>0.28</b>	
	10240005	4,961	411,897	<b>0.51</b>	
STJ	10240005	4,677	416,574	<b>0.49</b>	USGS drainage area is incorrect. Use listed drainage area
	10240011	4,677	416,574	<b>0.12</b>	
MKC	10240011	63,452	480,026	<b>0.88</b>	
	10300101	63,452	480,026	<b>0.00</b>	
WVMO	10300101	1,972	481,998	<b>0.73</b>	
BNMO	10300101	14,600	496,598	<b>0.27</b>	
	10300102	14,600	496,598	<b>0.08</b>	
HEMO	10300102	22,181	518,780	<b>0.92</b>	
	10300200	22,181	518,780	<b>0.27</b>	

### 3.3 ROUTING PARAMETERS

Various hydrologic routing methods were analyzed to determine the most appropriate methods and parameters. The process was very involved and complex, and therefore, is explained in complete detail in Appendix B – Routing Parameter Determination Summary. The final routing parameters using the coefficient routing method used in the model are shown in Table 3-4.

**Table 3-4: Coefficient method final routing parameters.**

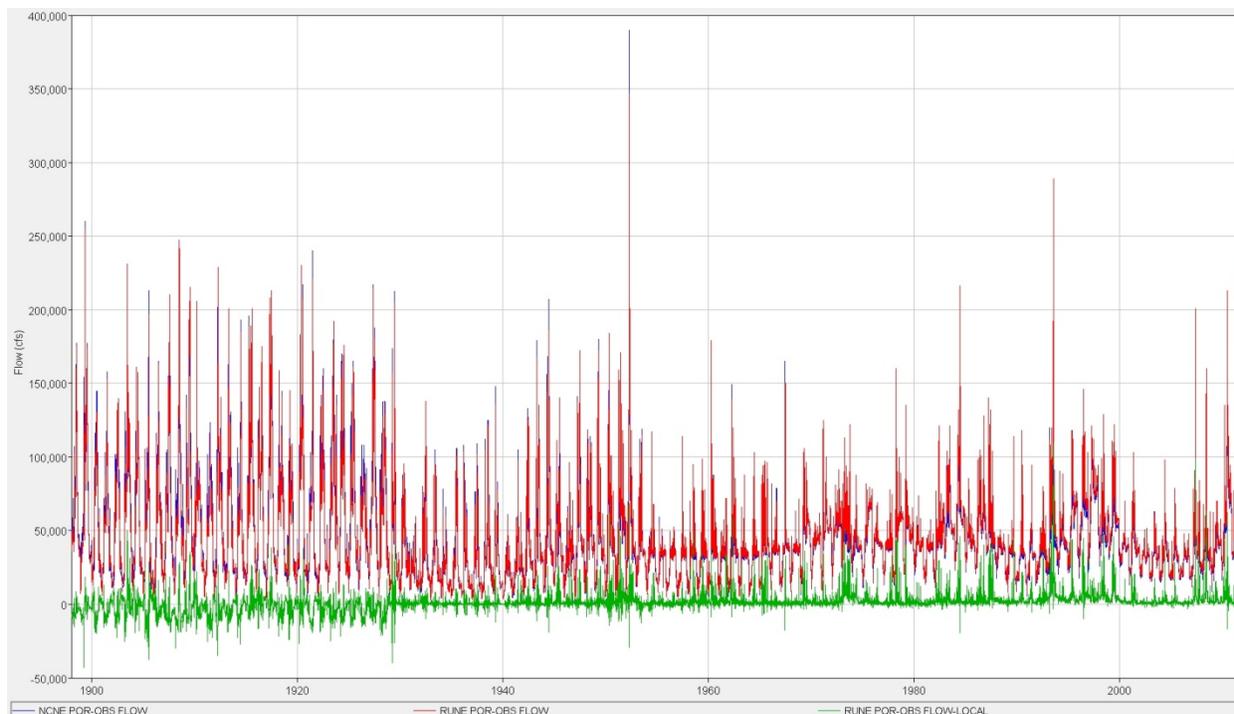
Reach	A1 (d)	A2 (d-1)	A3 (d-2)
RBMT_FTPK	0.000	1.000	0.000
FTP_K_WPMT	0.000	1.000	0.000
WPMT_CLMT	0.000	1.000	0.000
CLMT_GARR	0.000	0.000	1.000
GARR_BIS	0.000	1.000	0.000
BIS_OAHE	0.000	0.000	1.000
OAHE_BEND	0.766	0.234	0.000
BEND_FTRA	0.647	0.353	0.000
FTRA_GAPT	0.005	0.637	0.358
GAPT_SUX	0.175	0.538	0.287
SUX_OMA	0.168	0.722	0.110
OMA_NCNE	0.588	0.412	0.000
NCNE_RUNE	0.588	0.412	0.000
RUNE_STJ	0.775	0.225	0.000
STJ_MKC	0.426	0.449	0.125
MKC_WVMO	0.476	0.524	0.000
WVMO_BNMO	0.354	0.618	0.028
BNMO_HEMO	0.381	0.434	0.185
HEMO_MISL	0.222	0.778	0.000

### 3.4 LOCAL FLOWS

Local flows at each Missouri River gage and reservoir location were computed by subtracting the routed upstream flow from the observed flow at the downstream gage or reservoir. Some of the computed local flows have one or more days with large negative discharges. One reason for this is the compatibility of using multiple different data sources. Another possible reason for the large negative flow is routing parameters that don't fit that particular event the best. However, the routing parameters used are the parameters that were tested and worked best for recent events. The routing parameter determination is described in more detail in Appendix B – Routing Parameter Determination Summary. Examples of locations and periods of slightly unusual local flows are discussed further below.

When using UMRSSFS data to compute local flows, the resulting local flows sometimes look slightly more irregular than the local flows produced using only the USGS data. These local flows are felt to be reasonable estimations for a time period when no USGS gage data was

available, but should still be mentioned. The local flows at RUNE are an example of this. UMRSFFS data were used upstream at NCNE and routed to RUNE, then subtracted from the UMRSFFS data at RUNE. The USGS gage at NCNE came online in 1929, which can be easily seen in the local flows (green) on the graph in Figure 3-3.



**Figure 3-3: RUNE unadjusted local flow for POR.**

The resulting local flow irregularities are not due to improper routing or timing. The irregularities are due mainly to the datasets (UMRSFFS) used. Since the only other data available for use before gages/reservoirs were online is the DRM data, it is recommended that the UMRSFFS data still be used and the irregular local flows be cautiously accepted. The UMRSFFS data are thought to be more reliable than the DRM data, since the DRM data were based on estimated monthly flows with an average daily distribution while the UMRSFFS was based on daily stages converted to flows from a stage-discharge relationship based on flow measurements.

While the local flows may not look as ideal as the local flows calculated using only observed USGS data, the local flows are still believed to be reasonable. The data sets used to calculate the local flows are the best available data, and are felt to be more reliable than synthetic data based on statistics and basin characteristics.

Calculated local flows include many instances of negative inflows. ResSim is capable of incorporating negative flows when performing basic routing calculations; however, flow travel times can exceed six days to the most downstream target at Kansas City, and accounting for large negative flows could produce unrealistic reservoir releases. Some of the negative flows are realistic where large withdrawals of water from the basin (especially upper portions) actually occurred. However, calculated negative local flows are often due to gage measurement errors

or assumptions with routing parameters. In these cases, the negative flows are not realistic. To correct this problem, the local flow data sets were adjusted using three different methods to reduce the large negative local inflows.

The first method used a 3-day centered moving average (CMA) in order to smooth all local inflow data sets. For certain events and historic time periods where alternate routing parameters could reduce large negative inflows (such as flood events), different routing coefficients were used. If the large negative local inflows were small in duration and could not be corrected by alternate routing coefficients, the large negative flows were zeroed out and the volume was redistributed over the surrounding month (15 days on either side of the largest negative local inflow). If large negative local inflows occurred frequently in a data set and alternate routing parameters could not correct for this, up to a 31-day CMA was used. A 31-day CMA was only used for periods prior to 1930 (with UMRSSFFS data). The local flow datasets were further smoothed to limit negative inflows to 2,000 cfs since withdrawals greater than that were attributed to routing errors and not considered realistic. Table 3-5 summarizes all the changes made to every data set. Year dates are from January 1<sup>st</sup> of the first year listed through December 31<sup>st</sup> of the last year listed, unless otherwise specified in the notes.

Table 3-6 summarizes the alternate routing coefficients used for different time periods. If no routing coefficients are listed in the table, no changes to routing parameters were made at that location. The first line of routing coefficients for each location is the original coefficients used (and the coefficients used for POR and future modeling in ResSim).

**Table 3-5: Local flow adjustments summary.**

Gage	Time Period	Adjustment	Notes
FTPK	1898-1928	13-Day CMA	Jun 1908 zeroed and redistributed.
	1929-2011	3-Day CMA	
WPMT	1898-2011	3-Day CMA	
CLMT	1898-2011	3-Day CMA	
GARR	1898-2011	3-Day CMA	
BIS	1898-1928	3-Day CMA	15 Mar 1928 - 15 Oct 1929 Routing adjusted 1930-1953. Nov 1955 GARR outflow corrected (MRBWM entered data incorrectly).
	1928-1929	31-Day CMA	
	1929-2011	3-Day CMA	
OAHE	1898-1928	3-Day CMA	15 Mar 1928 - 15 Oct 1929 Routing adjusted 1930-1953.
	1928-1929	31-Day CMA	
	1929-2011	3-Day CMA	
BEND	1898-2011	13-Day CMA	Very cyclic with 3-day cma. Used 13-day for smoothing.
FTRA	1898-1929	3-Day CMA	Different routings did not improve. DRM data is issue, but only data available.
	1930-1956	13-Day CMA	
	1957-2011	3-Day CMA	
GAPT	1898-1929	3-Day CMA	Different routings did not improve. DRM data is issue, but only data available.
	1930-1953	13-Day CMA	
	1954-2011	3-Day CMA	
SUX	1898-1929	31-Day CMA	Apr 1943 & Jun 1944 zeroed and redistributed. Different routings did not improve.
	1930-2011	3-Day CMA	
OMA	1898-1929	31-Day CMA	Different routings did not improve. Didn't want to average over longer period than this. Routing adjusted Mar-Apr 1952, 31 Mar-20 Apr 1943, & Apr 1944. Jun-Jul 1944 zeroed and redistributed.
	1930-2011	3-Day CMA	
NCNE	1898-1929	31-Day CMA	Mar-Apr 1943, Apr 1950, & Mar-Apr 1952 zeroed and redistributed. Different routings did not improve.
	1930-2011	3-Day CMA	
RUNE	1898-1929	31-Day CMA	Routing adjusted Apr 1952. May-Jun 1944 & Jun-Jul 2011 zeroed and redistributed, different routings did not improve.
	1930-2011	3-Day CMA	
STJ	1898-1928	31-Day CMA	Still mass neg, but don't want to ave over longer period than this. Routing adjusted 1898-1928, Apr 1952, Jun 2010, & Jun-Jul 2011.
	1929-2011	3-Day CMA	
MKC	1898-1928	31-Day CMA	Apr 1952 & May-Jul 2011 zeroed and redistributed. Different routings did not improve.
	1929-2011	3-Day CMA	
WVMO	1898-1914	3-Day CMA	Routing adjusted 1898-1914. Different routings did not improve. Routing adjusted Jul 1951.
	1915-1929	31-Day CMA	
	1930-2011	3-Day CMA	
BNMO	1898-1929	31-Day CMA	Still -50,000 cfs in one area, but don't want to ave over longer period than this. Routing adjusted Jul 1951, Jul-Aug 1981, & Jul-Aug 1993.
	1930-2011	3-Day CMA	
HEMO	1898-2011	3-Day CMA	Jul 1993 & May 2007 zeroed and redistributed.

**Table 3-6: Routing coefficient adjustments used in local flow determination.**

Reach Name	Routing Coefficients						
	A1 (d)	A2 (d-1)	A3 (d-2)	A4 (d-3)	A5 (d-4)	A6 (d-5)	A7 (d-6)
FTPK_GARR							
GARR_OAHE							
OAHE_BEND							
BEND_FTRA							
FTRA_GAPT							
GAPT_SUX							
SUX_OMA	0.16794	0.72176	0.1103	0	0	0	0
Mar-Apr 1952	0.05	0.1	0.15	0.4	0.15	0.1	0.05
31 Mar - 20 Apr 1943 & Apr 1944	0	0.05	0.15	0.6	0.15	0.05	0
OMA_NCNE							
NCNE_RUNE	0.58837	0.41163	0	0	0	0	0
Apr 1952	0.35	0.33	0.32	0	0	0	0
RUNE_STJ	0.77547	0.22453	0	0	0	0	0
1898-1928, Apr 1952, Jun 2010, & Jun-Jul 2011	0.2	0.5	0.3	0	0	0	0
STJ_MKC	0.42647	0.44863	0.1249	0	0	0	0
MKC_WVMO	0.47605	0.52395	0	0	0	0	0
1898-1914	0	0.2	0.8	0	0	0	0
July 1951	0.1	0.9	0	0	0	0	0
WVMO_BNMO	0.3542	0.61748	0.02832	0	0	0	0
Jul 1951, Jul-Aug 1981, & Jul-Aug 1993	0	0.3542	0.61748	0.02832	0	0	0
BNMO_HEMO							
HEMO_STL							
FTPK_WPMT							
WPMT_CLMT							
CLMT_WSN							
BIS_OAHE							
GARR_BIS							

The adjusted local inflow data sets were stored in the file "Input\_Data.dss". They have a part F pathname of "HISTORIC: CALCULATED".

### 3.5 EVAPORATION

Best available evaporation data for the Missouri River POR were obtained from the Daily Routing Model (DRM). Assessment of the data was performed prior to incorporation into the ResSim model. There was a significant change in how evaporation data was measured between 1966 and 1967. MRBWM Division recommended using the DRM net evaporation data, which is total evaporation minus precipitation. The DRM evaporation data is consistent with the data used by MRBWM for their water management tasks, and corresponds well with

the other data used from MRBWM in this study (all reservoir data). Evaporation data from the DRM was computed by two different processes. Prior to 1967 the evaporation was computed using the data from the Long Range Study model (LRS). Beginning in 1967, historic lake evaporation values were used in determining evaporation in the DRM model by using a ratio of the computed reservoirs areas to the areas determined from historic elevations.

### 3.5.1 Evaporation Rates Pre-1967

Monthly normal annual lake evaporation rates (in inches) for each of the mainstem reservoirs were developed using the Weather Bureau Technical Paper No. 37 (TP-37). Using Plate 2 of TP-37, the normal annual lake evaporation amounts were obtained and are listed in Table 3-7. Initially, Gavins Point evaporation losses were considered insignificant due to the small surface area and no annual lake evaporation was listed.

It was assumed that the evaporation through the entire period of record with the exception of the 1930-1941 drought period would approach the normal. Evaporation was reduced by the average annual precipitation over each of the reservoirs to obtain the normal annual net evaporation and is listed in Table 3-7. The net evaporation listed in Table 3-7 for Gavins Point Dam was taken from the LRS model input.

**Table 3-7: Normal annual lake and net evaporation from TP-37.**

Project	Normal Annual Lake Evaporation (inches/year)	Normal Annual Net Evaporation (inches/year)
Fort Peck	39.0	27.6
Garrison	35.0	21.1
Oahe	35.5	19.3
Big Bend	37.0	18.1
Fort Randall	37.5	13.4
Gavin Point	n/a	14.0

Note : Net Evaporation = Normal annual lake evaporation – Normal annual precipitation

During the 1930-1941 drought period, annual evaporation was computed for each year using the following procedures. The period 1942-1962 was considered a normal period and by utilizing the normal annual evaporation depths obtained from the Weather Bureau Technical Paper with temperature, humidity and wind data at or near each of the reservoirs during the 1942-1962 period, an appropriate constant was derived for each project to solve the empirical equation:

$$\text{Evaporation} = K (\text{wind}) (\text{water surface vapor pressure} - \text{vapor pressure of air})$$

It was assumed that the water surface temperature would, over the course of a year, equal the observed air temperature. The constant derived above, in combination with the observed wind, humidity and temperature data, was then used for computing the annual evaporation from each project for each of the drought years. Annual net evaporation was determined by subtracting out observed precipitation at each of the projects and is listed in Table 3-8.

**Table 3-8: Annual net evaporation during 1930-1941.**

Year	Annual Net Evaporation Data (inches/year)					
	FTPK	GARR	OAHE	BEND	FTRA	GAPT
1898-1928	27.6	21.5	19.3	13.4	18.1	14.0
1929	27.6	21.5	19.3	25.4	18.1	24.0
1930	26.2	29.0	23.2	25.7	28.7	21.0
1931	34.8	26.5	33.5	35.9	38.9	30.0
1932	27.1	24.2	29.6	33.4	35.0	33.0
1933	37.0	38.0	46.1	43.4	45.0	42.0
1934	41.0	44.9	56.3	49.6	55.4	48.0
1935	34.0	17.4	31.6	27.0	37.9	18.0
1936	57.6	50.3	63.7	48.7	60.7	37.2
1937	41.3	29.8	38.9	34.0	44.6	24.0
1938	22.7	21.8	38.3	28.2	37.3	21.6
1939	37.8	24.0	39.4	37.0	41.4	30.0
1940	25.2	20.9	35.0	40.3	43.9	36.0
1941	26.6	7.0	23.6	22.3	27.4	18.0
1942-1966	27.6	21.5	19.3	13.4	18.1	14.0

Note: Drought period from 1930 to 1941. Evaporation for BEND and GAPT looks suspect in 1929.

From an analysis of available surface temperature and precipitation records of the mainstem reservoirs the seasonal distribution of the annual net evaporation was estimated and is listed in Table 3-9.

**Table 3-9: Monthly Distribution of DRM Evaporation Prior to 1967.**

Month	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Percent	0.0	0.0	0.0	0.0	7.0	5.0	19.0	20.0	19.0	13.0	12.0	5.0

The initial evaporation input to the ResSim model for each dam from 1898 to 1966 is equal to the annual evaporation times the monthly distribution ratio divided by the number of days in the month.

### **3.5.2 Evaporation Rates Post-1966**

Data after 1966 was based on measurements taken at each of the projects. The data are based on pan evaporation except during the winter periods when it was estimated. It should be noted that some projects have quit measuring pan evaporation during the spring, summer and fall and have instead started using estimated values year round.

For the data based on pan evaporation, observed pan measurements were taken daily and then factored by an average monthly pan coefficient to come up with the lake evaporation. During periods when pan evaporation readings were not taken, the evaporation is considered to be a constant for each particular month and normal monthly representative pan evaporation values

were used. These are taken from the 1973 report titled, *Missouri River Main Stem Reservoir System, Reservoir Evaporation Estimates, MRD-RCC Technical Report JE-73*, and are listed in Table 3-10. Both the measured and normal monthly pan evaporation are factored by the normal pan to lake evaporation coefficient which were also taken from the June 1973 report and are listed in Table 3-11. Table 3-12 lists the normal monthly lake evaporation based on the normal monthly pan evaporation (Table 3-10) multiplied by the normal monthly pan to lake coefficient (Table 3-11).

**Table 3-10: Normal monthly pan evaporation in inches.**

Month	Fort Peck	Garrison	Oahe	Big Bend	Fort Randall	Gavins Point
January	0.62	0.51	1.02	0.80	1.02	0.74
February	0.74	0.58	1.14	0.98	1.16	0.91
March	1.68	1.42	2.24	1.97	2.31	1.91
April	3.50	2.79	4.70	4.48	4.27	4.19
May	6.96	6.35	7.80	7.83	6.74	7.30
June	8.05	7.07	8.51	8.47	7.54	8.30
July	10.45	8.97	10.74	10.85	9.00	9.64
August	10.22	8.56	10.44	10.31	8.13	8.41
September	5.97	6.63	7.25	7.26	5.07	5.57
October	4.03	4.07	4.92	4.06	4.42	4.46
November	1.96	1.38	2.25	1.83	2.34	1.79
December	0.83	0.70	1.19	1.04	1.24	0.87
<b>Annual</b>	<b>55.01</b>	<b>48.03</b>	<b>62.20</b>	<b>59.88</b>	<b>53.24</b>	<b>54.09</b>

Note: Taken from Figure 9 of the *Missouri River Main Stem Reservoir System, Reservoir Evaporation Estimates, MRD-RCC Technical Report JE-73*. Based on available pan data from 1963-1972. During months pan data were not available, pan depths were computed by mass-transfer equation assuming pan water temperature to be equivalent to air temperature. **Values shown for Oahe and Big Bend are believed to be unrepresentative. Representative pan data requires an adjustment factor of 0.80 for Oahe and 0.90 for Big Bend.** See Report JE-73 for more detail.

**Table 3-11: Normal monthly pan to lake evaporation coefficients.**

Month	Fort Peck	Garrison	Oahe	Big Bend	Fort Randall	Gavins Point
January	1.28	0.70	0.73	0.63	0.70	0.70
February	0.70	0.70	0.56	0.63	0.70	0.70
March	0.60	0.70	0.49	0.54	0.63	0.62
April	0.11	0.14	0.13	0.47	0.19	0.53
May	0.22	0.20	0.16	0.35	0.32	0.53
June	0.32	0.21	0.18	0.39	0.37	0.53
July	0.39	0.26	0.22	0.53	0.42	0.56
August	0.64	0.64	0.50	0.70	0.78	0.70
September	1.21	1.13	0.89	0.82	1.31	0.93
October	1.32	1.44	1.19	1.05	1.42	0.97
November	2.57	3.74	2.22	1.52	1.62	1.59
December	4.22	5.04	3.42	1.36	1.39	1.57

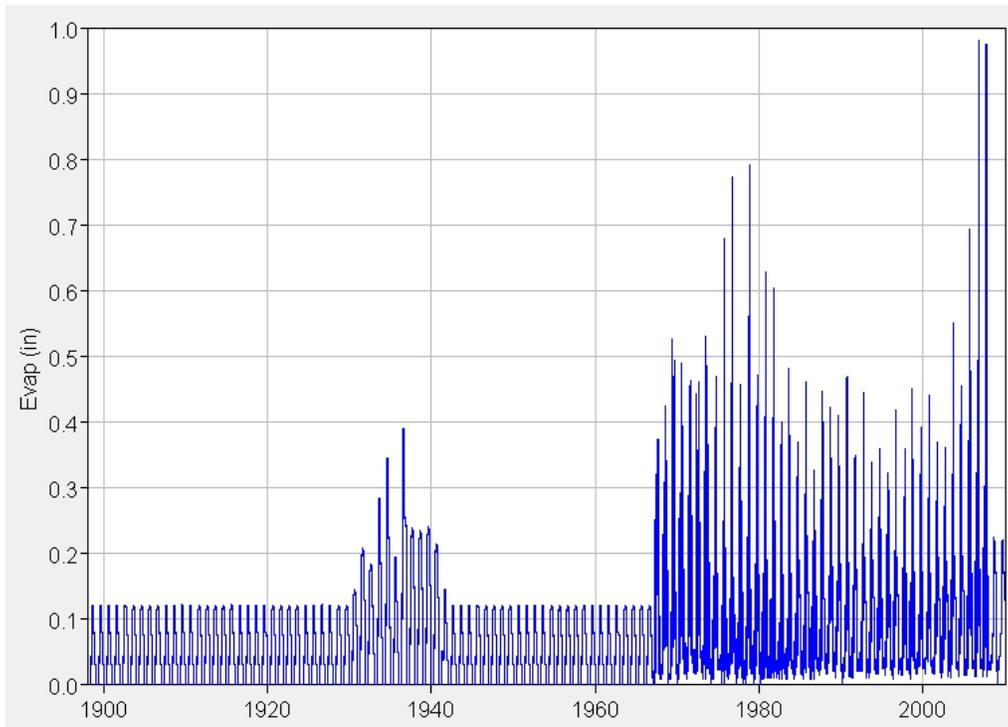
Note: Taken from Figure 11 of the *Missouri River Main Stem Reservoir System, Reservoir Evaporation Estimates, MRD-RCC Technical Report JE-73*

**Table 3-12: Normal monthly lake evaporation.**

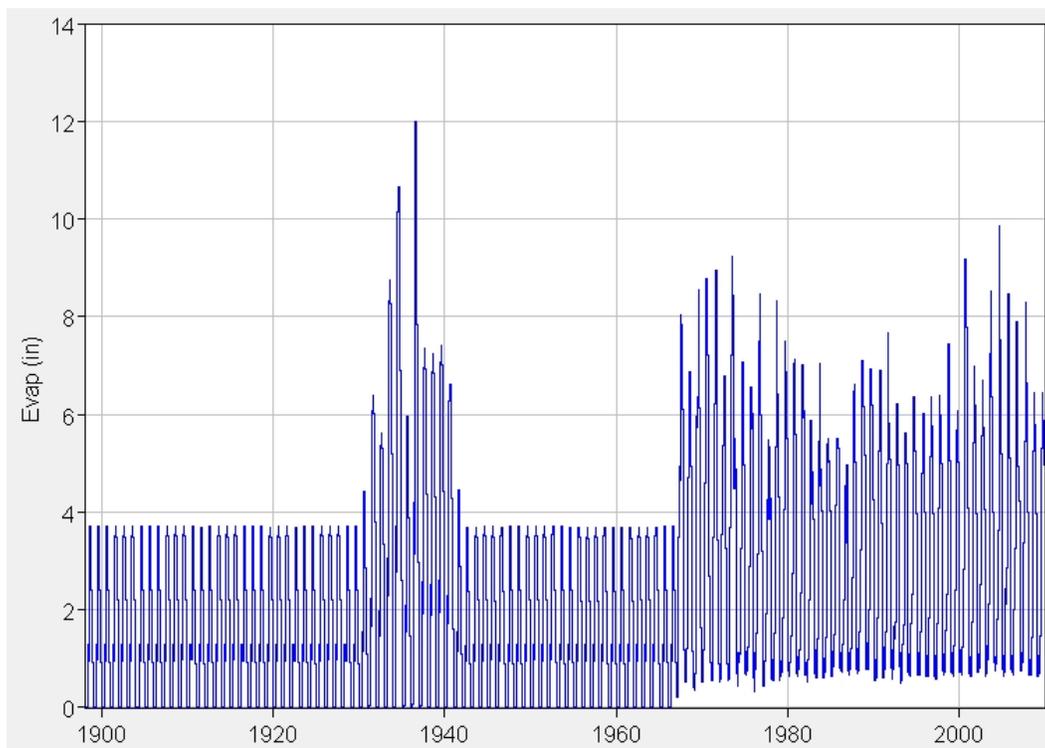
Month	Fort Peck	Garrison	Oahe	Big Bend	Fort Randall	Gavins Point
January	0.79	0.36	0.74	0.50	0.71	0.52
February	0.52	0.41	0.64	0.62	0.81	0.64
March	1.01	0.99	1.10	1.06	1.46	1.18
April	0.38	0.39	0.61	2.11	0.81	2.22
May	1.53	1.27	1.25	2.74	2.16	3.87
June	2.58	1.48	1.53	3.30	2.79	4.40
July	4.08	2.33	2.36	5.75	3.78	5.41
August	6.54	5.48	5.22	7.22	6.34	5.89
September	7.22	6.36	6.45	5.97	6.64	5.18
October	5.32	5.86	5.85	4.26	6.26	4.33
November	5.04	5.16	5.00	2.78	3.79	2.85
December	3.50	3.53	4.07	1.41	1.72	1.37
<b>Annual</b>	<b>38.51</b>	<b>33.62</b>	<b>34.82</b>	<b>37.72</b>	<b>37.27</b>	<b>37.85</b>

Note: Taken from Figure 12 of the *Missouri River Main Stem Reservoir System, Reservoir Evaporation Estimates, MRD-RCC Technical Report JE-73*.

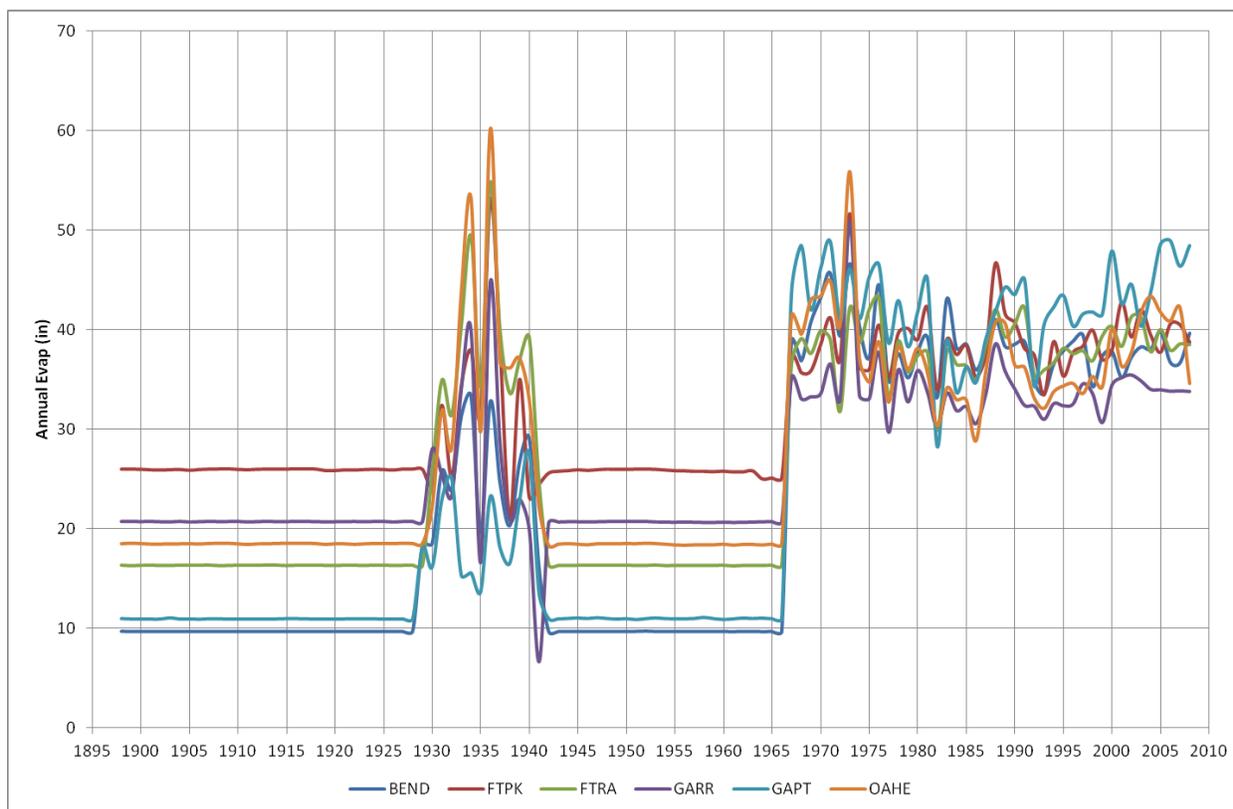
Daily and monthly evaporation outputs from the DRM for 1898-2011 were obtained in text format and input into HEC-DSS. The DRM output was converted to inches per day or month using the DRM output elevations to determine the corresponding area from the DRM input elevation-area file. Figure 3-4 and Figure 3-5 below show a sample daily and monthly evaporation plot for Lake Oahe using DRM data. Figure 3-6 provides a plot of annual evaporation based on the DRM data.



**Figure 3-4: Oahe DRM daily evaporation.**



**Figure 3-5: Oahe DRM monthly evaporation.**



**Figure 3-6: Annual DRM evaporation at the mainstem projects.**

### **3.5.3 Inflow Loss due to Evaporation Computations**

The computations for calculating inflow loss due to evaporation (flow-evap) in the DRM model are essentially the same for all six dams. If the year is pre-1967, the DRM uses the information from the Long Range Study (LRS) model. The flow-evap is computed by multiplying the factor (feet of evaporation per day) described in Section 3.5.1 by the difference in the reservoir area less the channel area, as was done in the LRS evaporation computations. The flow-evap from the portion of the reservoir occupied by the original channel area was assumed to be included in the depletion computations. If the computation year is post-1966 then the flow-evap is computed by multiplying the historic evaporation by the ratio of DRM computed area to historic area. The evaporation values in the MRBWM database do not consider channel area, and it was not used in the computations post-1966.

The computations for calculating evaporation in the ResSim model do not differentiate the two periods of evaporation data and are computed differently than the DRM. ResSim flow-evap computations compute a daily evaporation flow with units of cfs by converting the daily evaporation depth (data discussed in Sections 3.5.1 and 3.5.2) to feet and then multiplying the depth by the reservoir area, which varies with the pool elevation.

Although there are computational differences, it was decided that the DRM evaporation data was acceptable for the ResSim model; however, the data will be updated in the future to ensure that the evaporation data used in ResSim is a homogeneous dataset.

## **3.6 RESERVOIR DATA**

The following sections summarize the physical reservoir data used at the mainstem reservoirs in the ResSim model. A sequential release allocation in ResSim was applied to all the mainstem projects with priority release given first to the Powerplant, second to Outlet Works (if applicable), and third to spillway flow.

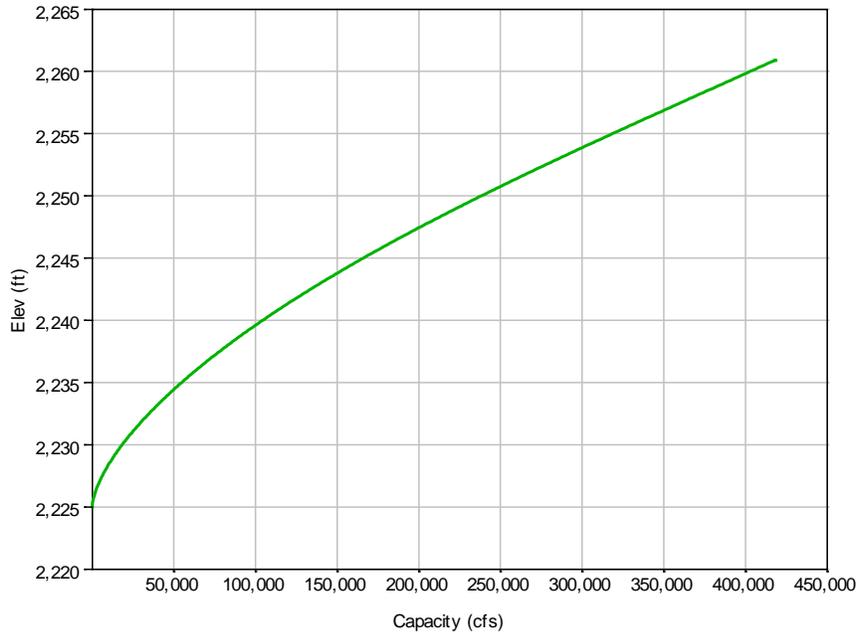
### ***3.6.1 Elevation-Area-Capacity Curves***

Elevation-Area-Capacity (E-A-C) curves for each mainstem reservoir were obtained from surveys performed by the channel sedimentation section of NWO. These data were entered into the ResSim model in 1 ft increments but are summarized in 10 ft increments and list in Appendix C – Elevation-Area-Capacity Curves. The following list summarizes the survey data used at each location:

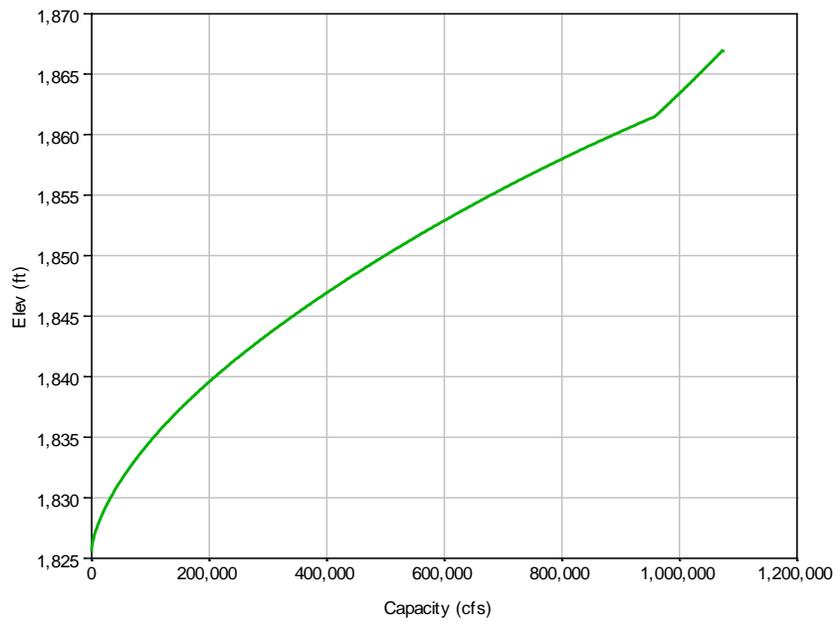
- FTPK from 2007 survey
- GARR from 2011 survey
- OAHE from 2010 survey
- BEND from 2012 survey
- FTRA from 2011 survey
- GAPT from 2011 survey

### ***3.6.2 Spillway Flow***

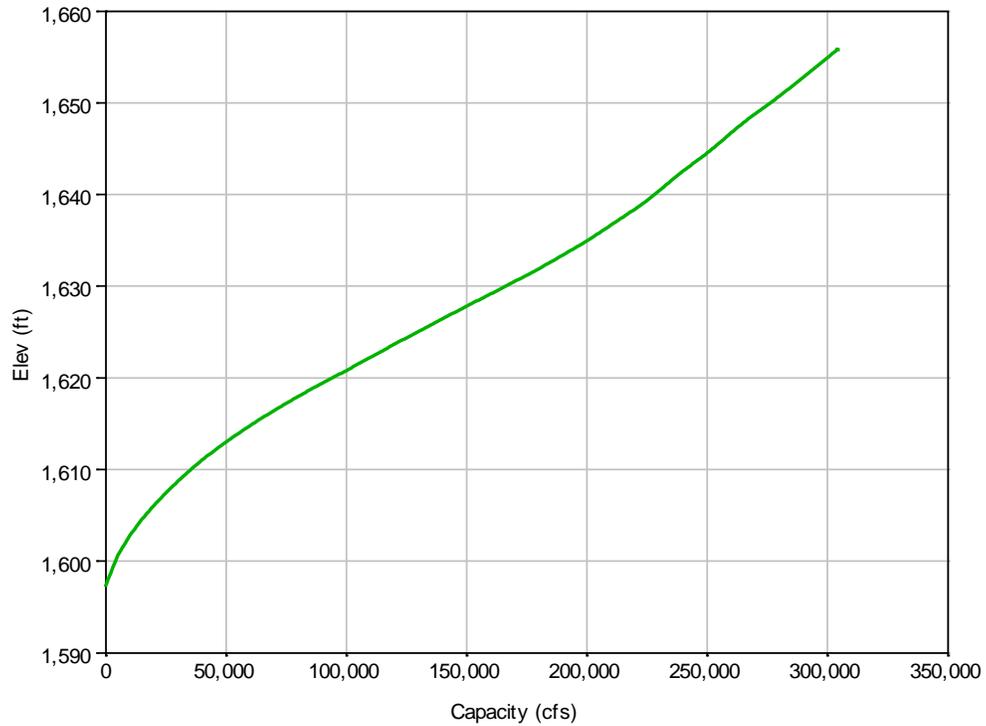
Spillway rating curves were obtained from the Hydraulics Section of NWO. These curves were not used for the Master Manual update, but they are the latest and "best" available as of May 2012. The following figures display the spillway rating curves used for each project in the model. It should be noted that these curves represent the maximum capacity with all gates fully utilized.



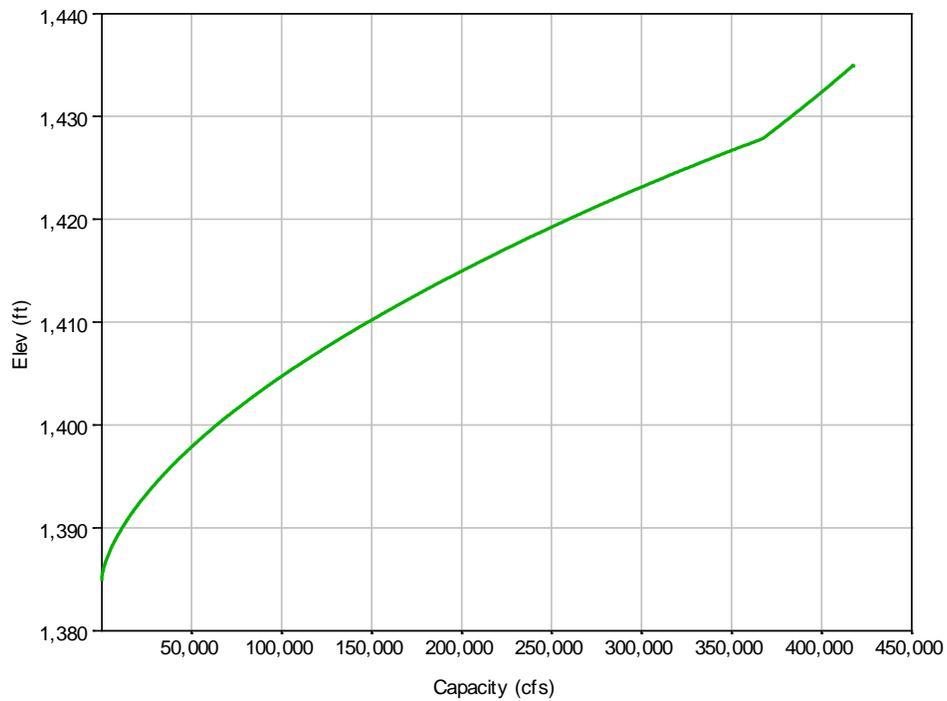
**Figure 3-7: FTPK spillway capacity curve.**



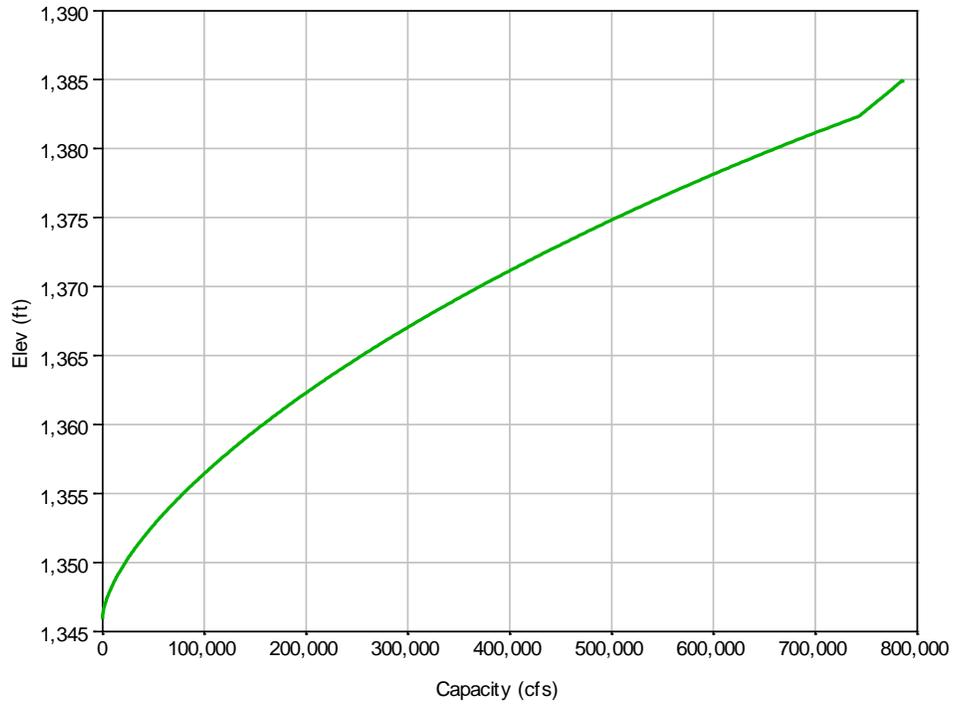
**Figure 3-8: GARR spillway capacity curve.**



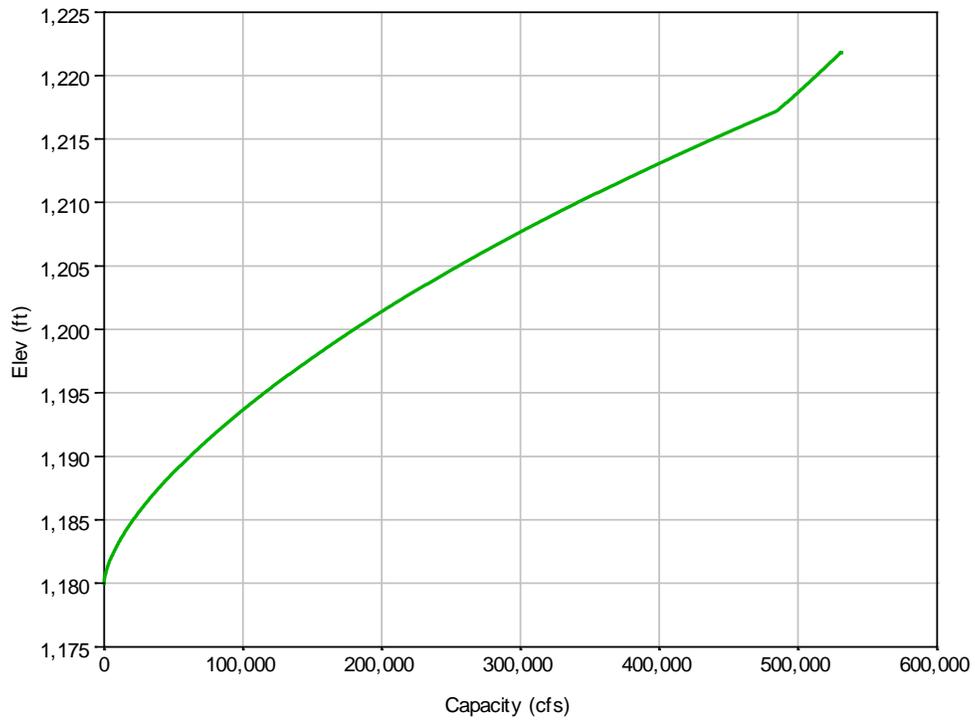
**Figure 3-9: OAHE spillway capacity curve.**



**Figure 3-10: BEND spillway capacity curve.**



**Figure 3-11: FTRA spillway capacity curve.**



**Figure 3-12: GAPT spillway capacity curve.**

### 3.6.3 Outlet Works Flow

Outlet works flow is considered discharge through the dam not made by the powerhouse or the spillway but released through the flood control tunnels. Curves are provided below for GARR, OAHE, and FTRA. FTPK does have outlet works but they are not to be operated unless absolutely necessary because of problems with the ring gates. GARR and OAHE outlet works discharge were estimated from the Mar 2004 Master Manual curves. BEND and GAPT have no flood control tunnel outlet works. FTRA curve was the best available as of May 2012 and was provided by the Hydraulics Section (NWO). The elevation-discharge curve for elevations lower than 1320 were estimated based on a second order polynomial equation of the upper curve.

The following figures display the outlet works rating curves for GARR, OAHE, and FTRA.

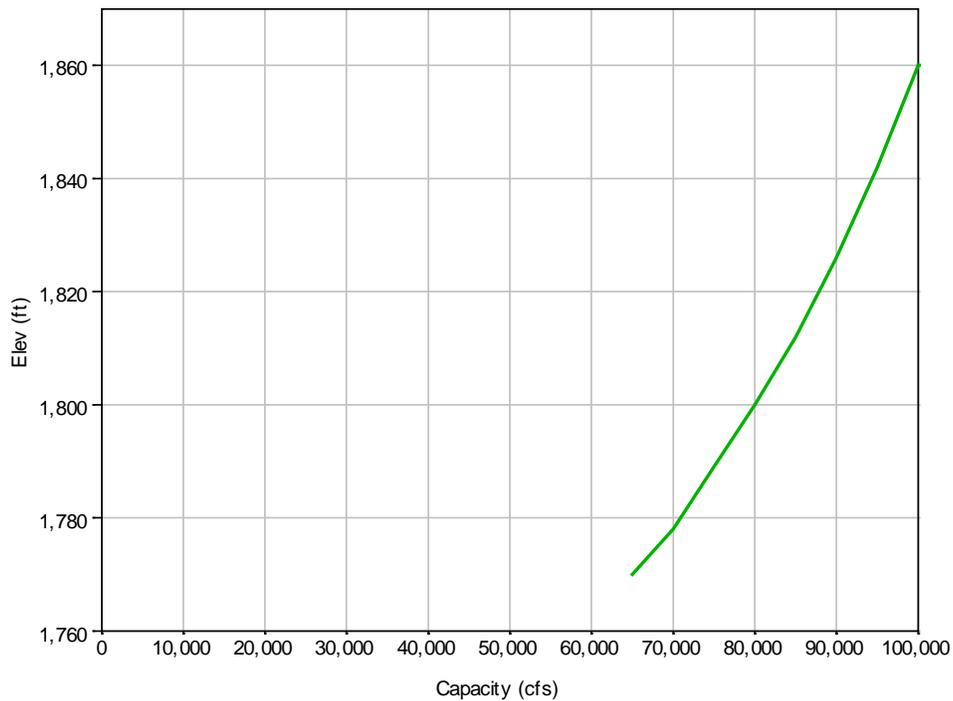
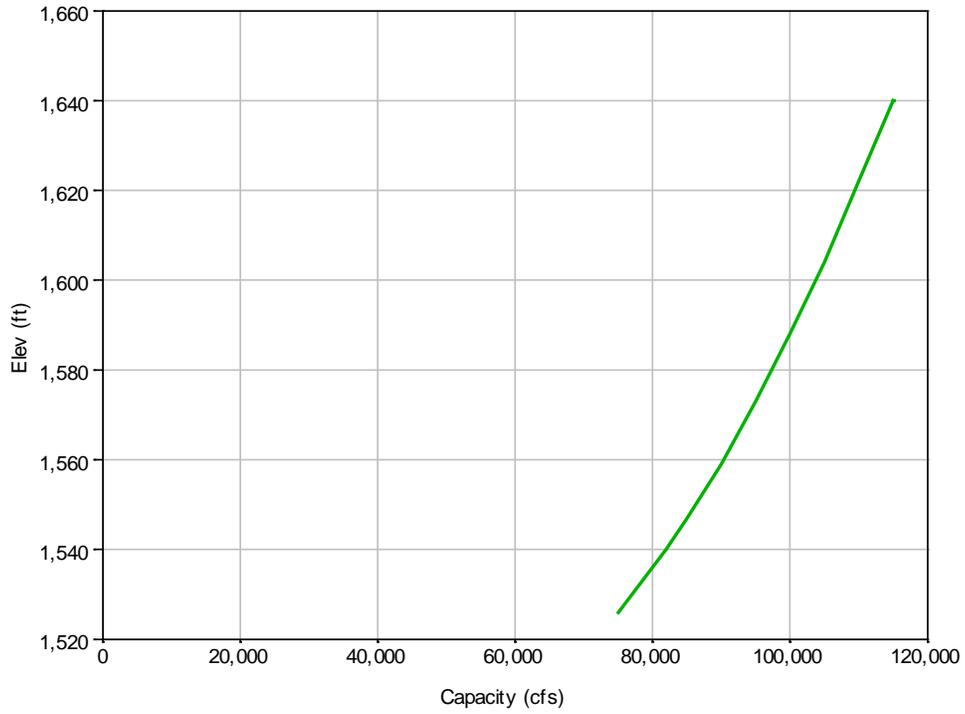
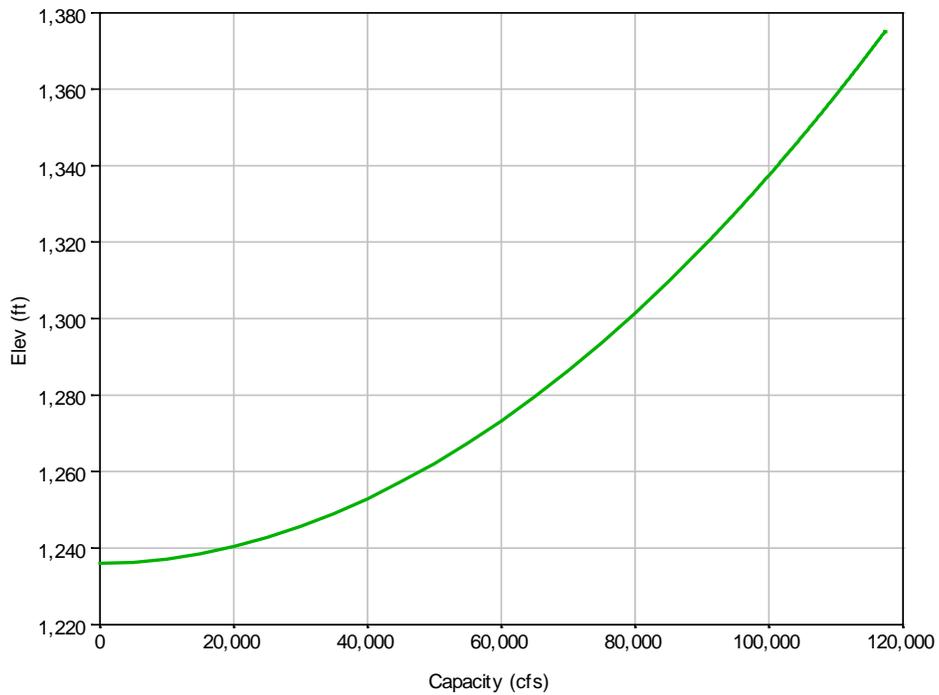


Figure 3-13: GARR outlet works capacity curve.



**Figure 3-14: OAHE outlet works capacity curve.**



**Figure 3-15: FTRA outlet works capacity curve.**

### 3.6.4 Powerhouse Flow

Release capacity through the powerhouses was determined based on the reservoir “Powerplant Characteristics” curves in the Master Manual and a “PowerplantCharacteristics” spreadsheet from MRBWM in May 2012, with the exception of BEND. BEND required additional examination since flow capacity depends partially on the downstream Lake Francis Case water surface elevation. Maximum release through the powerhouse at BEND was entered as an operational Rule within ResSim stating that maximum release is a function of Lake Sharpe (BEND) tailwater current value.

It should be noted that all project powerhouse discharge capacity curves had to be adjusted slightly in ResSim from their actual Master Manual curves. This affected only the portion of the upper end of the curve when capacity began to decrease with elevation. This was adjusted in ResSim so that the curve remained at a constant discharge as the elevation increased. It had to be changed in ResSim because ResSim adjusted the capacity flows, but wrongfully would not release the remaining desired flow through the spillways/outlet works.

The following figures display the powerhouse discharge capacity for the other 5 projects.

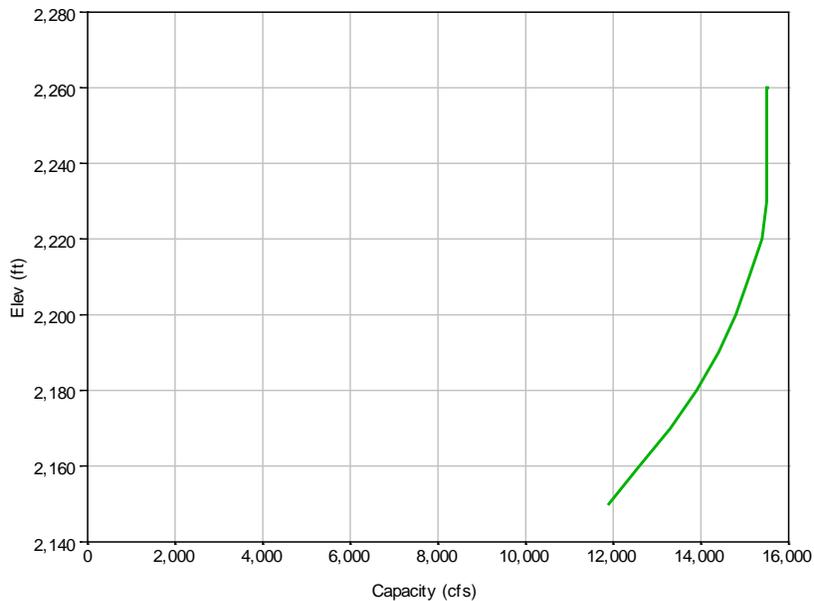
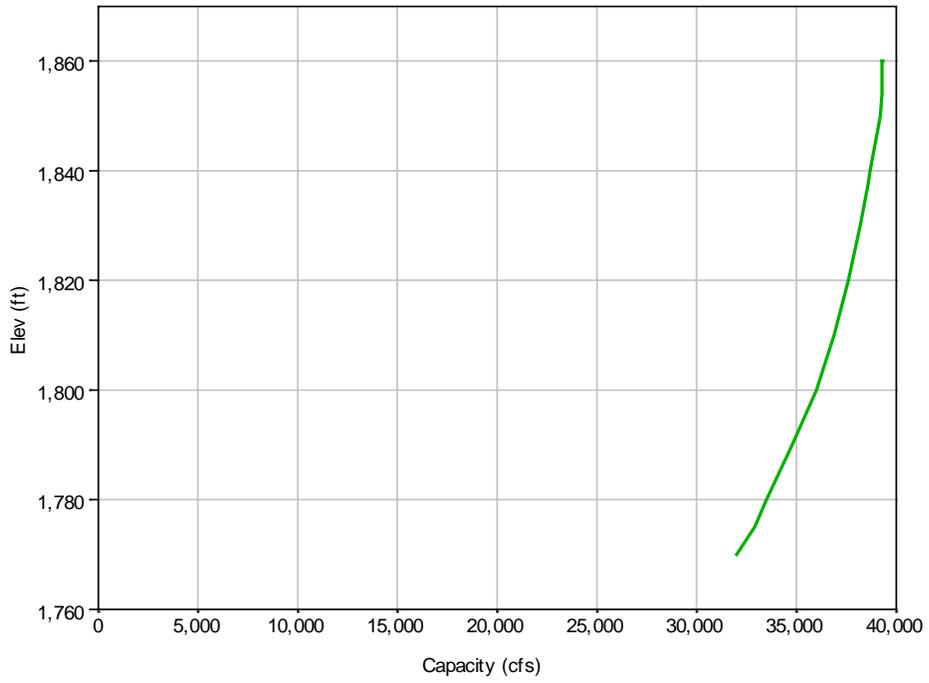
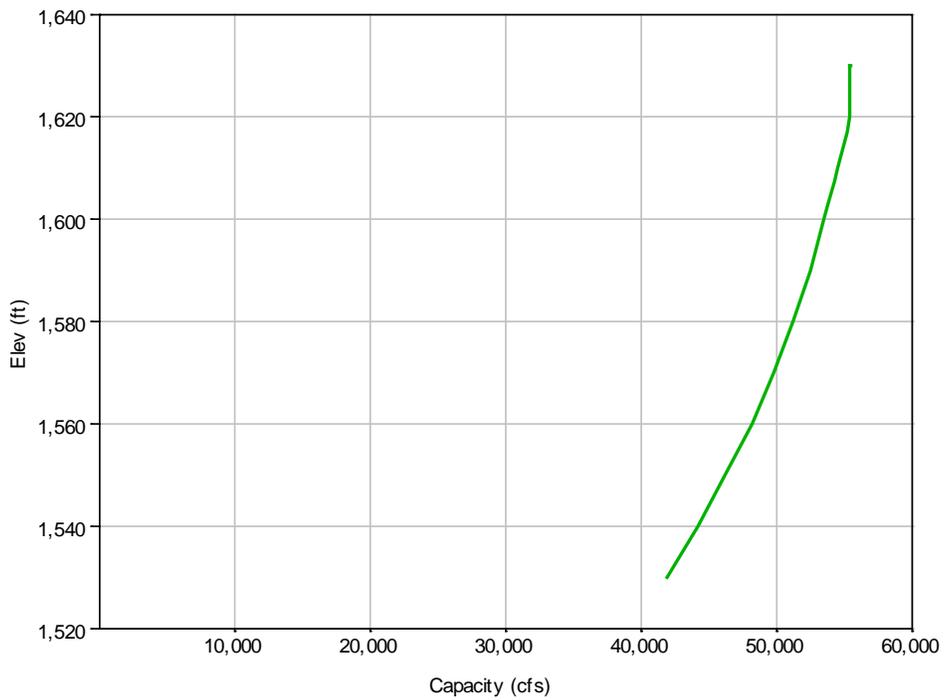


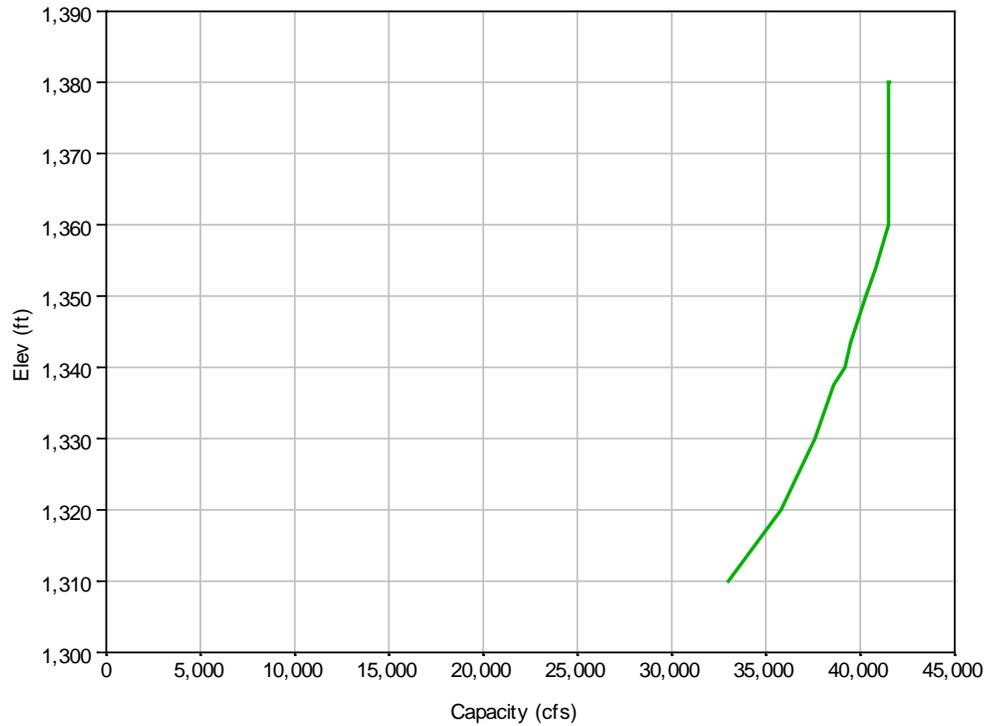
Figure 3-16: FPK powerhouse discharge capacity.



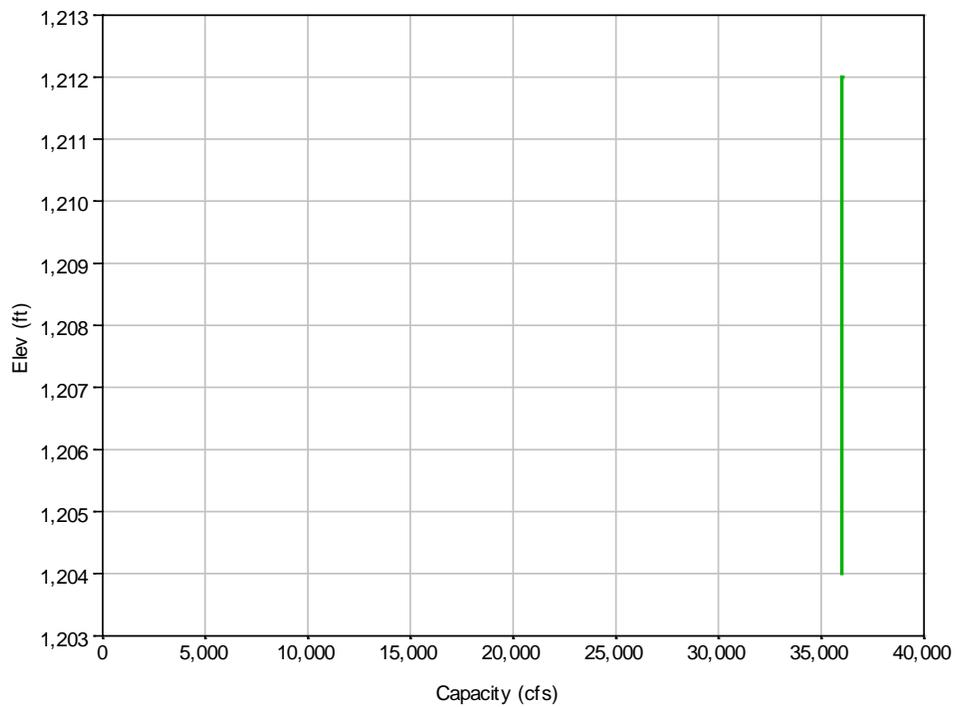
**Figure 3-17: GARR powerhouse discharge capacity.**



**Figure 3-18: OAHE powerhouse discharge capacity.**



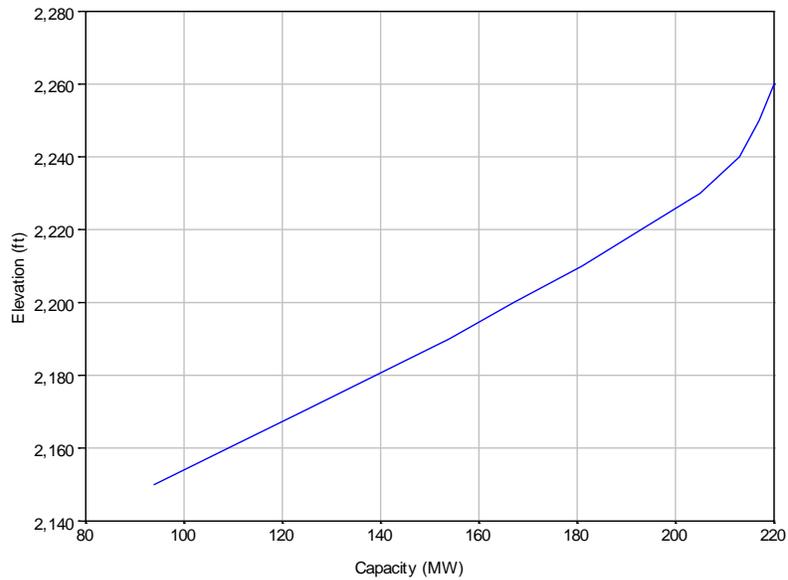
**Figure 3-19: FTRA powerhouse discharge capacity.**



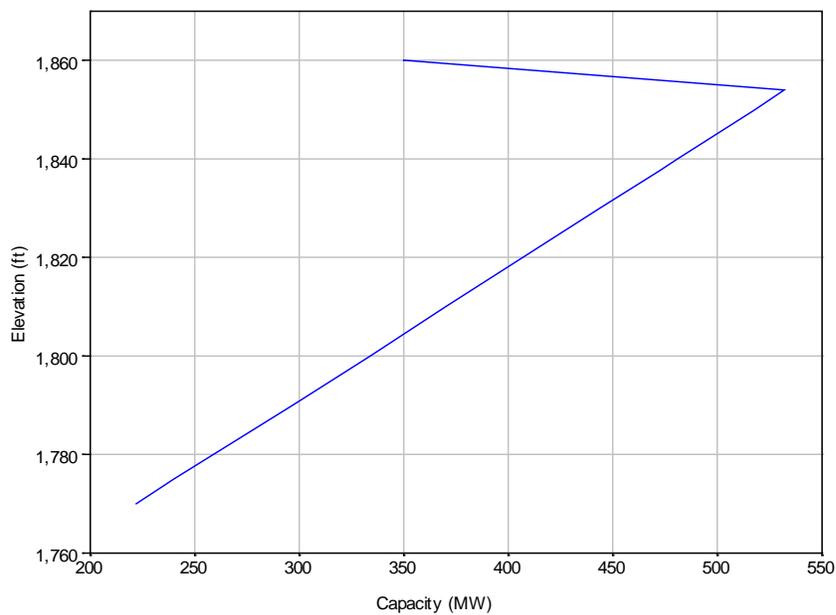
**Figure 3-20: GAPT powerhouse discharge capacity.**

### 3.6.5 Power Generation Capacity

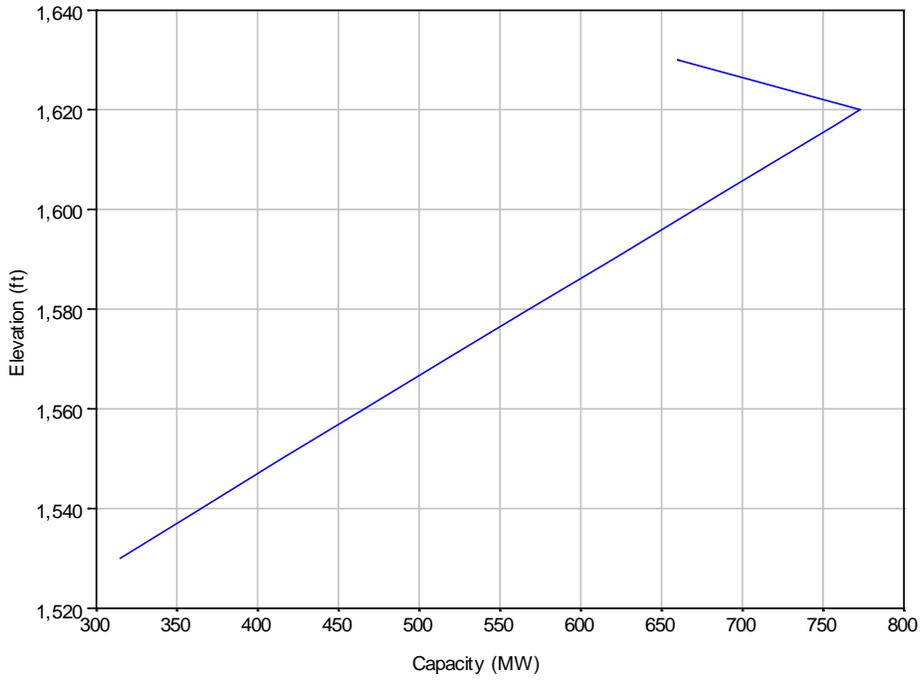
Power generation capacity curves were determined based on the reservoir “Powerplant Characteristics” curves in the Master Manual. The following figures display the power generation capacity of the projects.



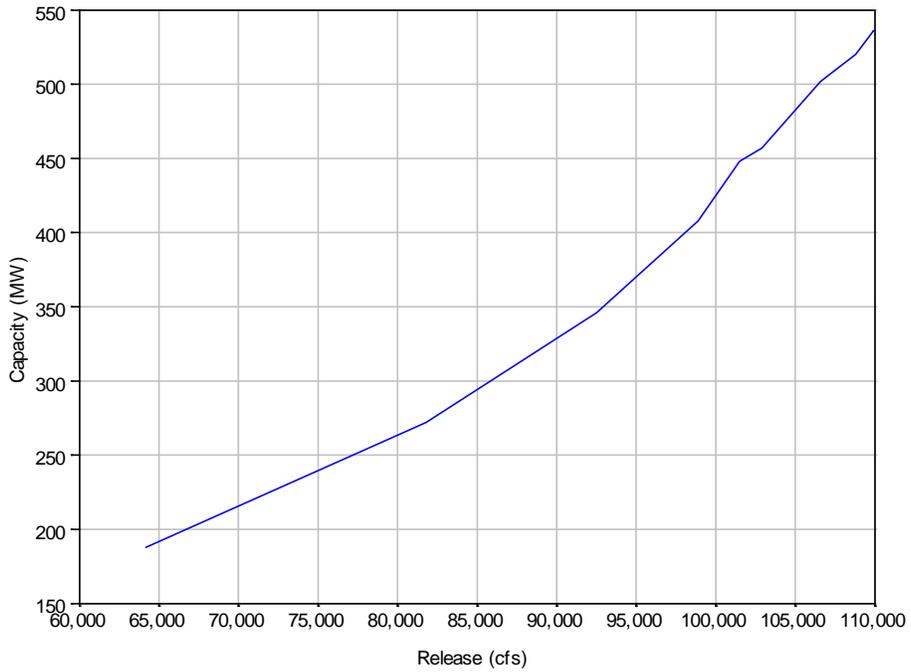
**Figure 3-21: FTPK power generation capacity.**



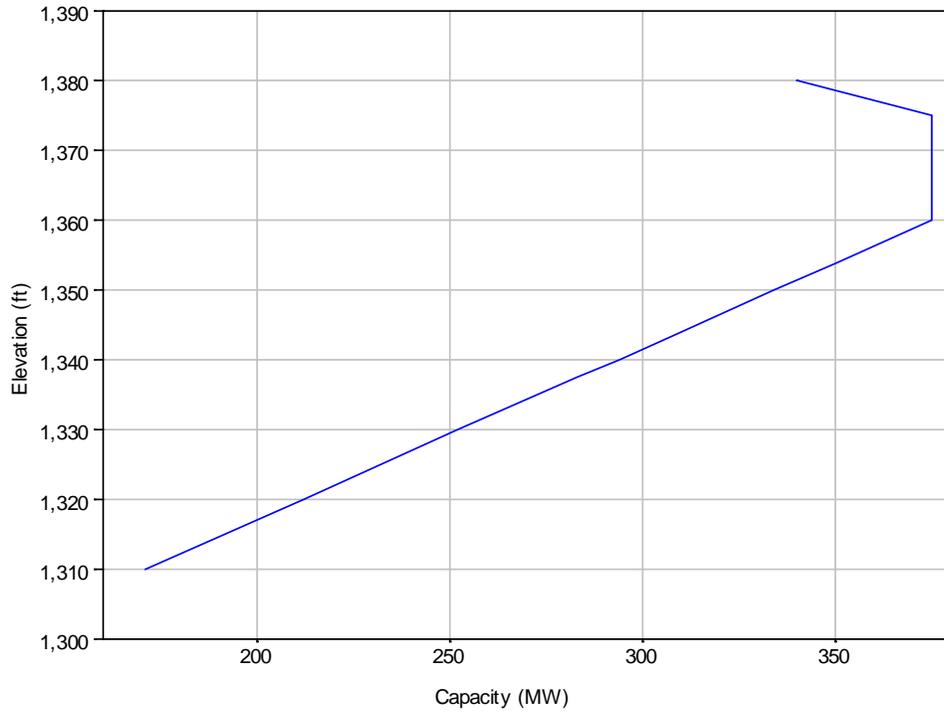
**Figure 3-22: GARR power generation capacity.**



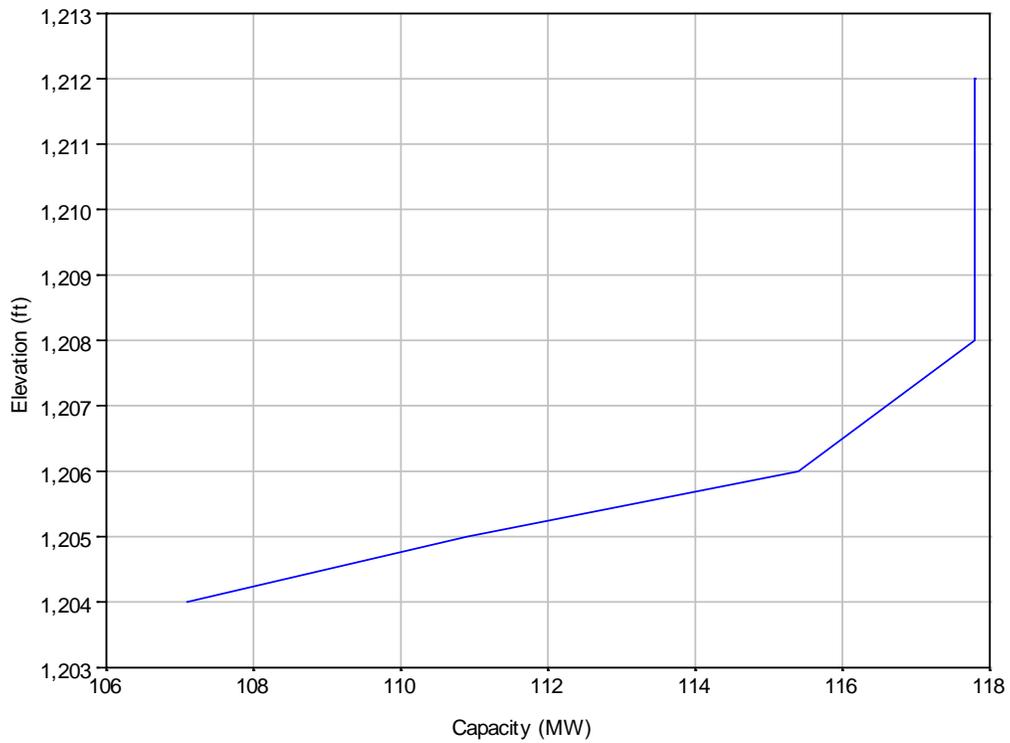
**Figure 3-23: OAHE power generation capacity.**



**Figure 3-24: BEND power generation capacity.**



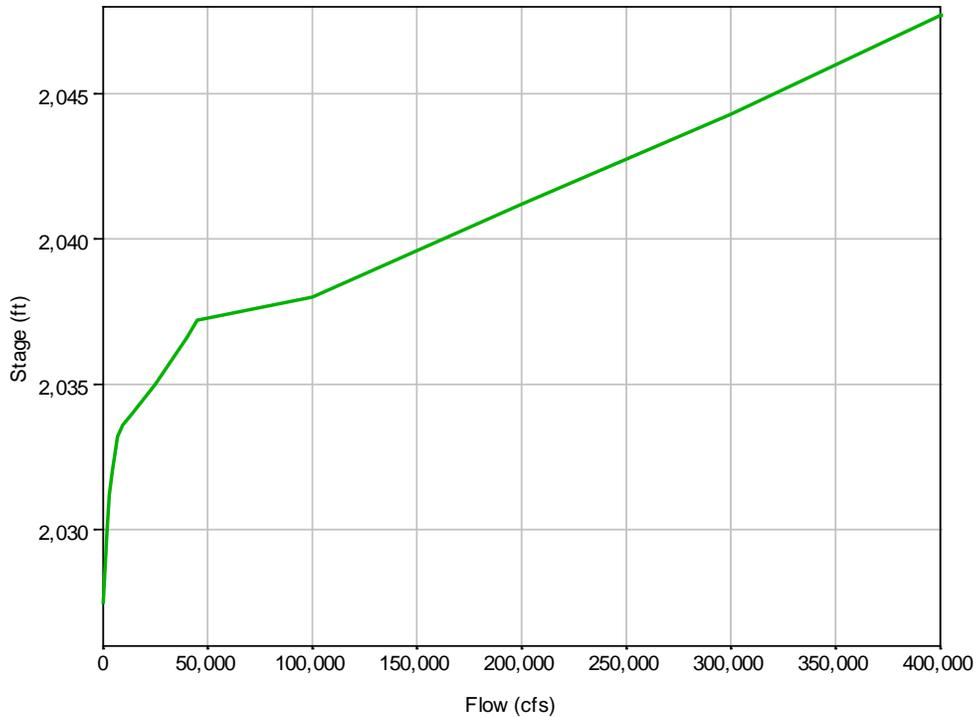
**Figure 3-25: FTRA power generation capacity.**



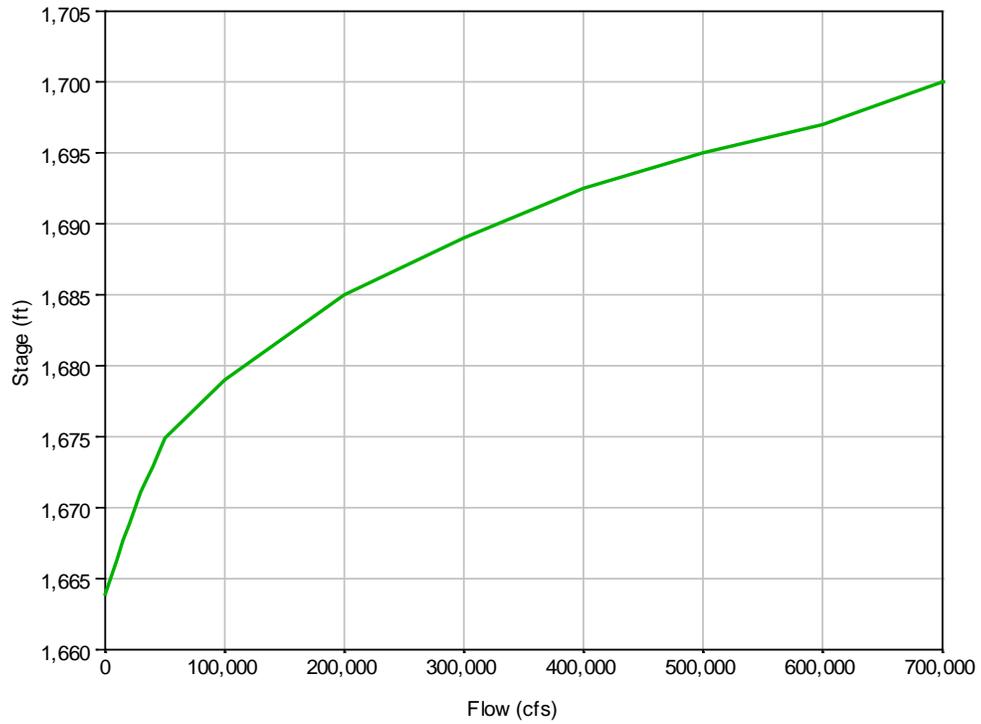
**Figure 3-26: GAPT power generation capacity.**

### 3.6.6 Tailwater

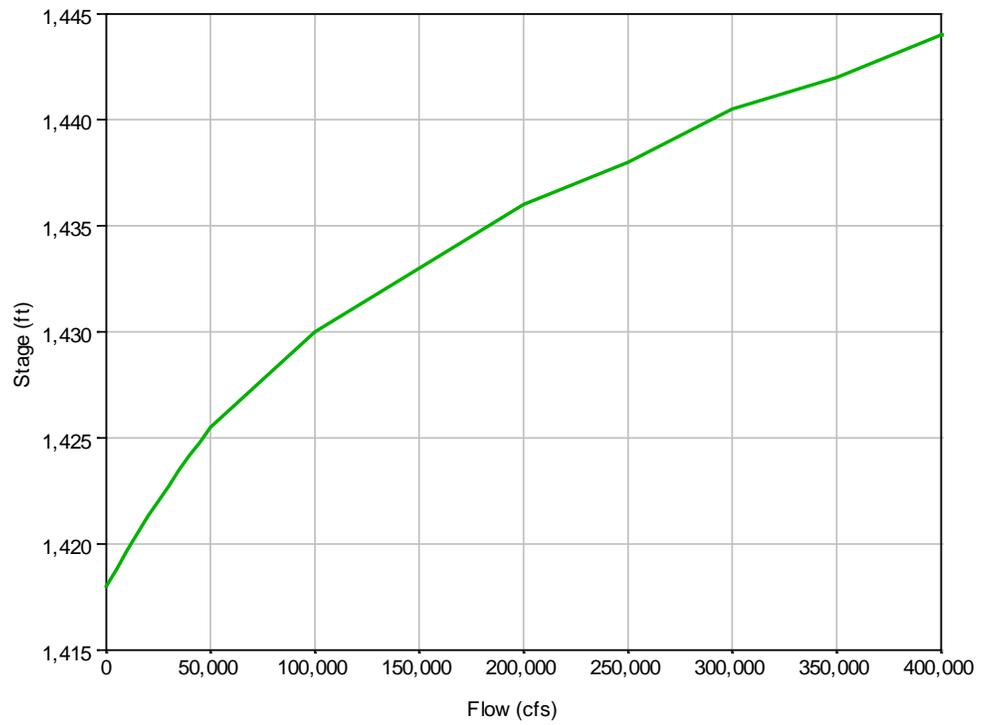
Tailwater below the projects was determined primarily using the 1992 Emergency System Operation Plan (ESOP) reports in combination with the tailwater rating curves from the Master Manual. Tailwater within ResSim was created at the same hierarchy level as the release outlets because the curves were developed assuming total outflow. The following figures display tailwater curves for all projects. It should be noted that the tailwater at Oahe is the highest elevation from downstream Lake Sharpe and the rating curve. Likewise, the tailwater at Big Bend is the highest elevation from downstream Lake Francis Case and the rating curve.



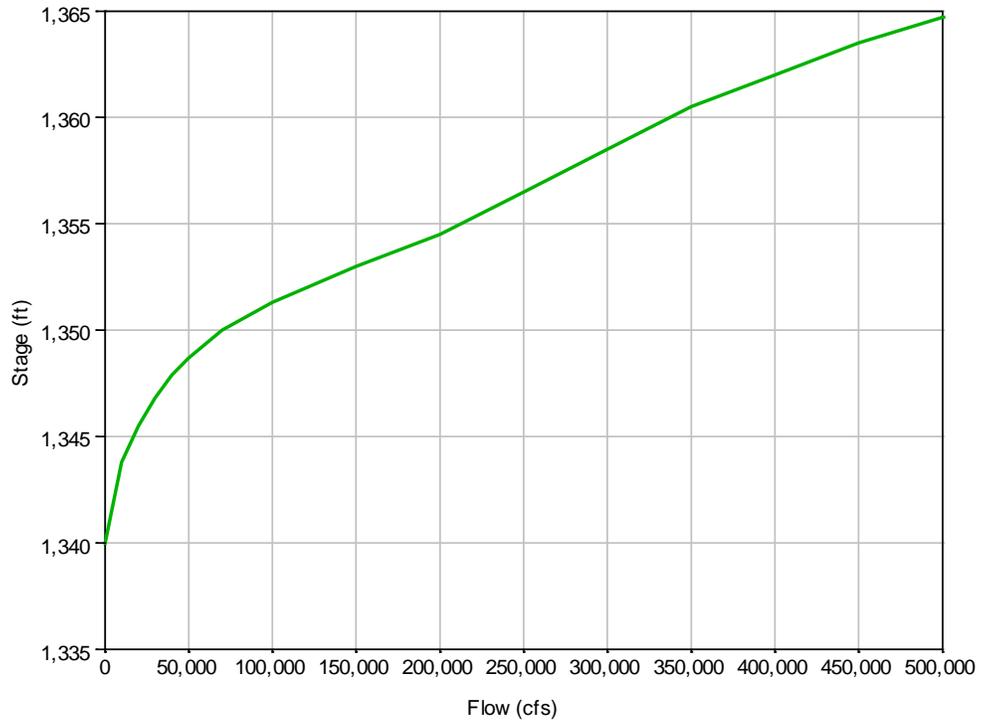
**Figure 3-27: FTPK tailwater.**



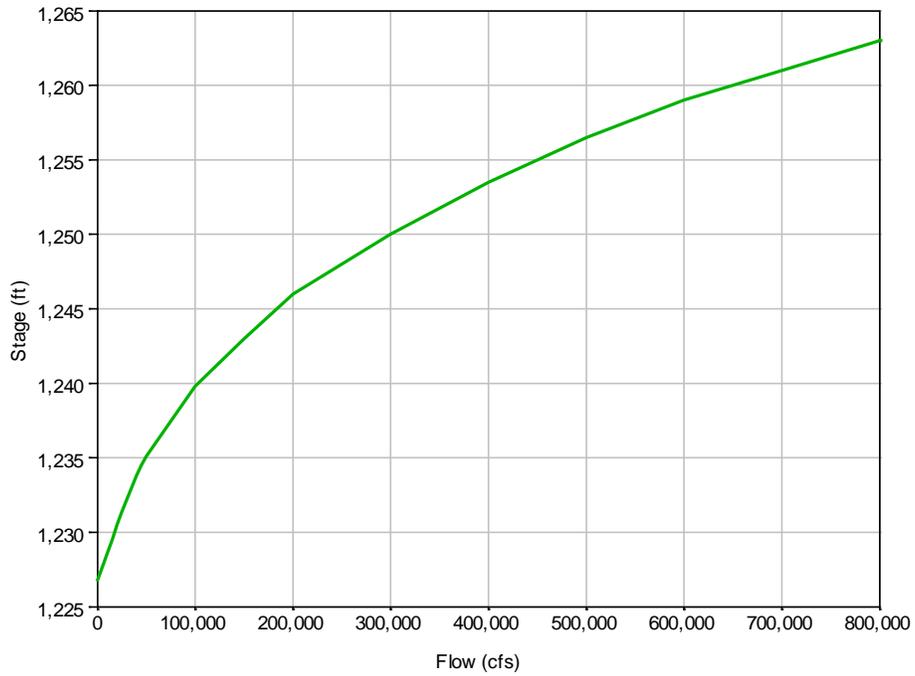
**Figure 3-28: GARR tailwater.**



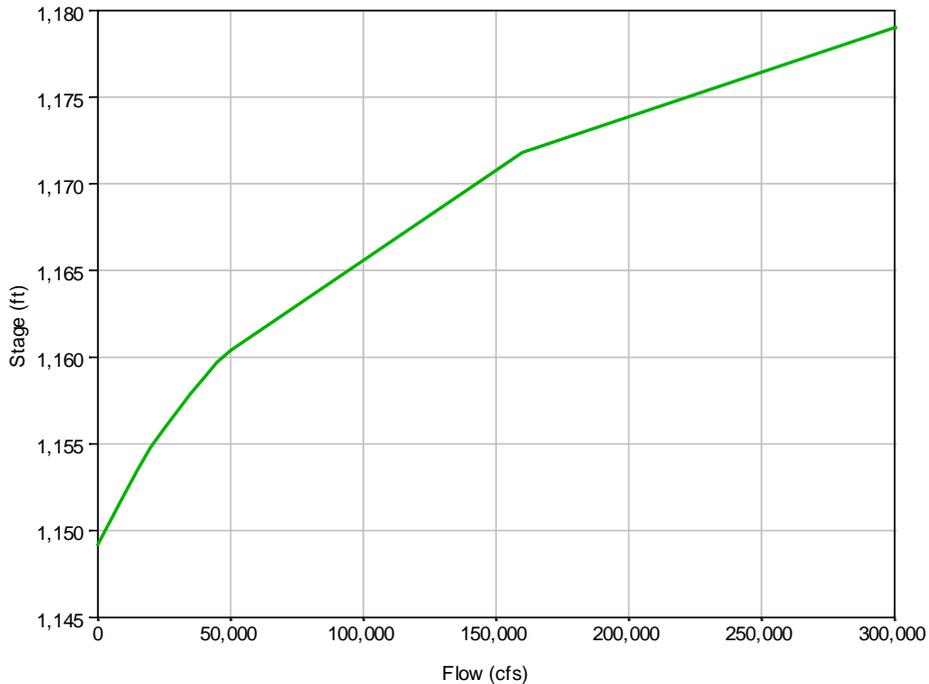
**Figure 3-29: OAHE tailwater.**



**Figure 3-30: BEND tailwater.**



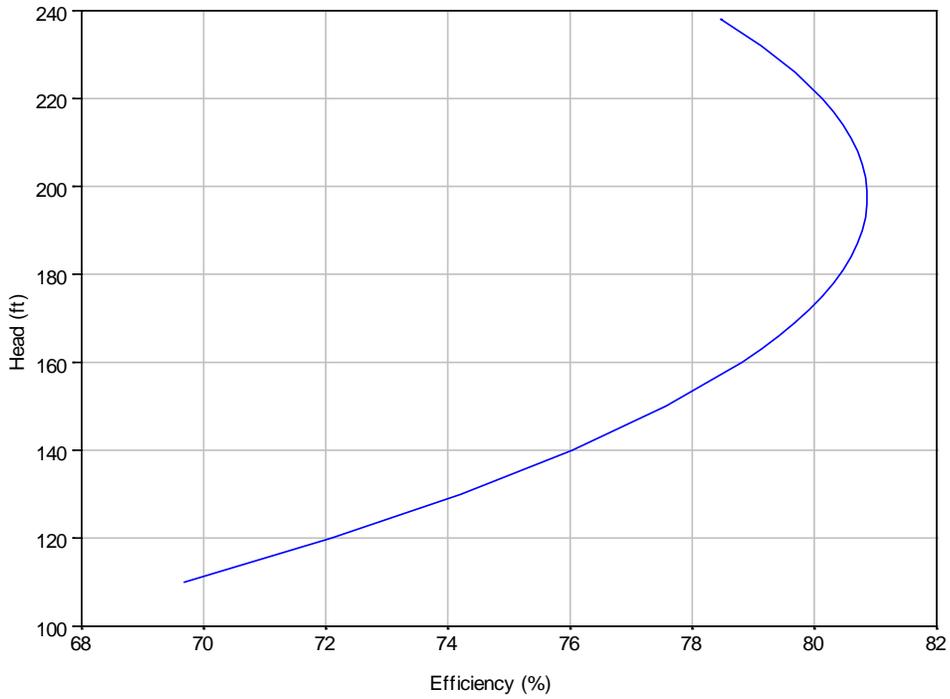
**Figure 3-31: FTRA tailwater.**



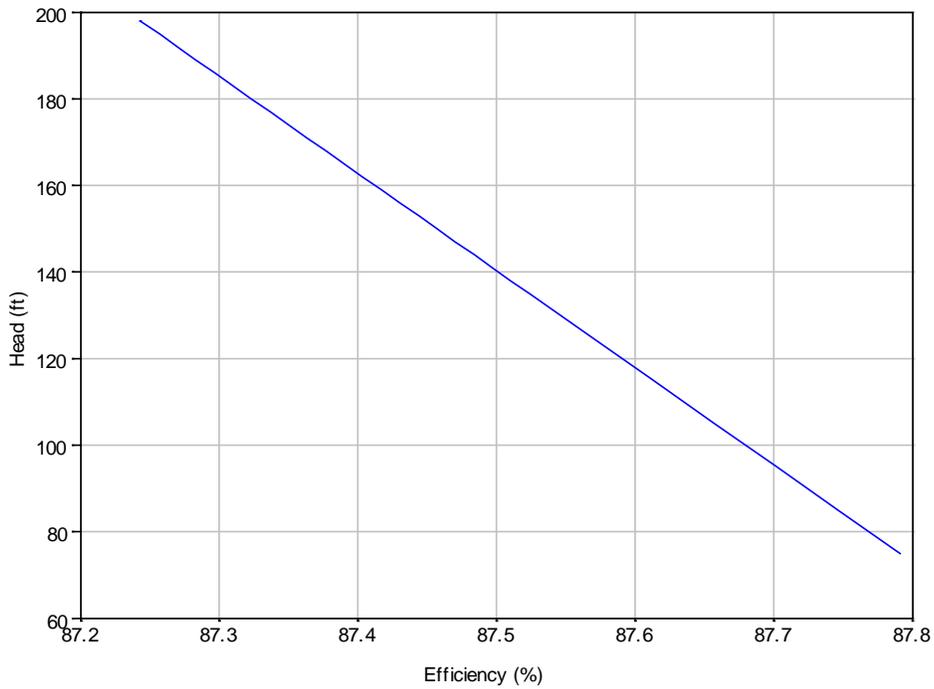
**Figure 3-32: GAPT tailwater.**

### **3.6.7 Power Efficiencies**

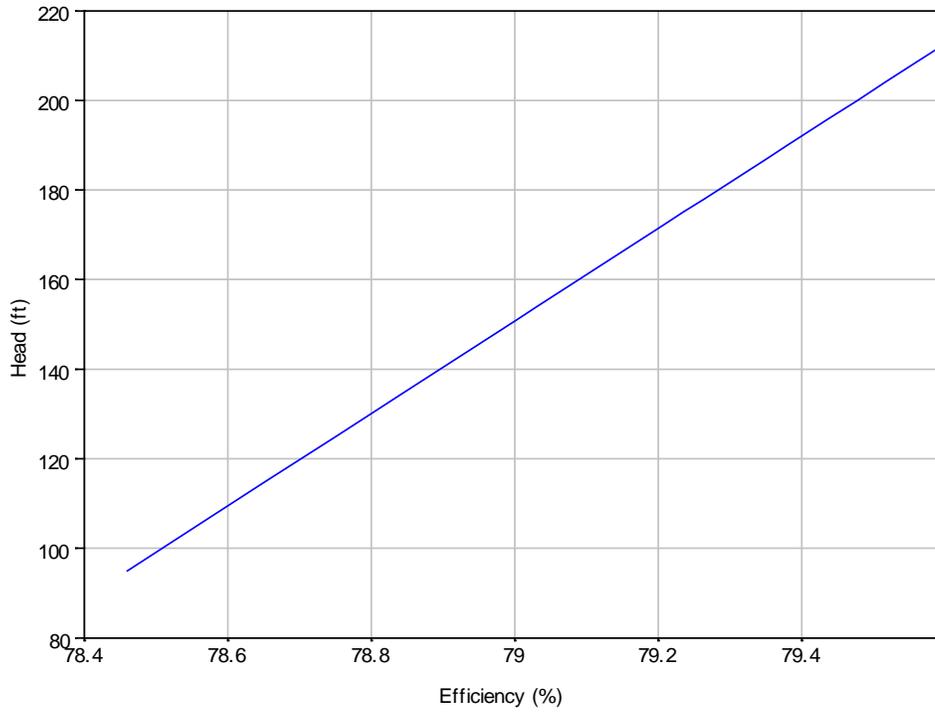
Reservoir efficiency in ResSim can be entered as a constant or a function of reservoir elevation, release, or operating head. MRBWM performed analysis on observed data and developed relationships for FTPK, BEND, FTRA, and GAPT. It was determined the relationships with efficiency as a function of differential head based on observed data, mostly post-1997, provided the best correlation. The curves were then extended to the minimum and maximum potential head, based on the equations for the lines of the 2<sup>nd</sup> order polynomials. For the extreme ends of the curves, some calculated efficiencies were less than what has been historically observed. For these differential heads, the minimum observed efficiency was used. GARR and OAHE required additional analysis due to the irregular nature of the observed data. The curves were also extended to the minimum and maximum potential head, but based on a linear fit rather than a 2<sup>nd</sup> order polynomial fit. The final efficiency curves for the projects are shown in the following figures.



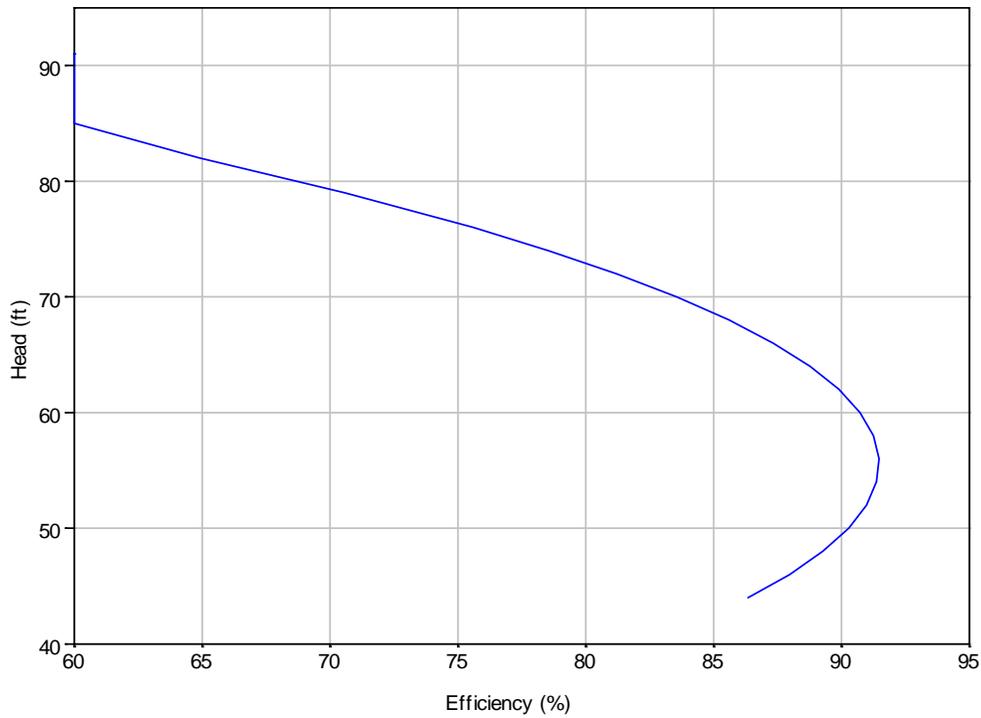
**Figure 3-33: FPK powerplant efficiency.**



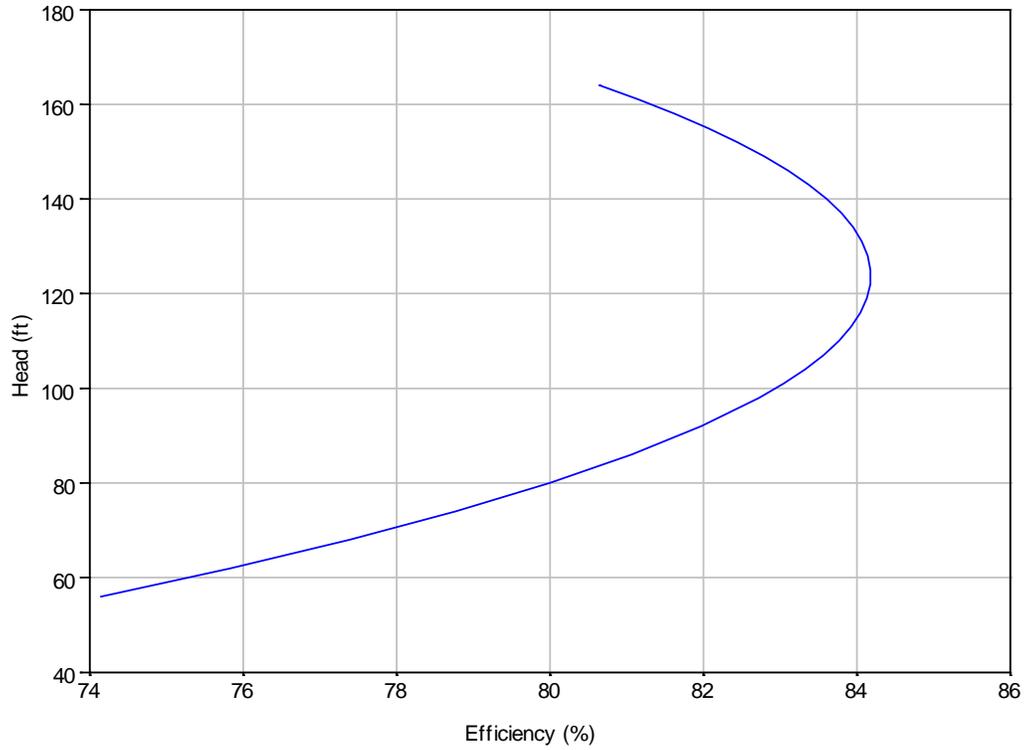
**Figure 3-34: GARR powerplant efficiency.**



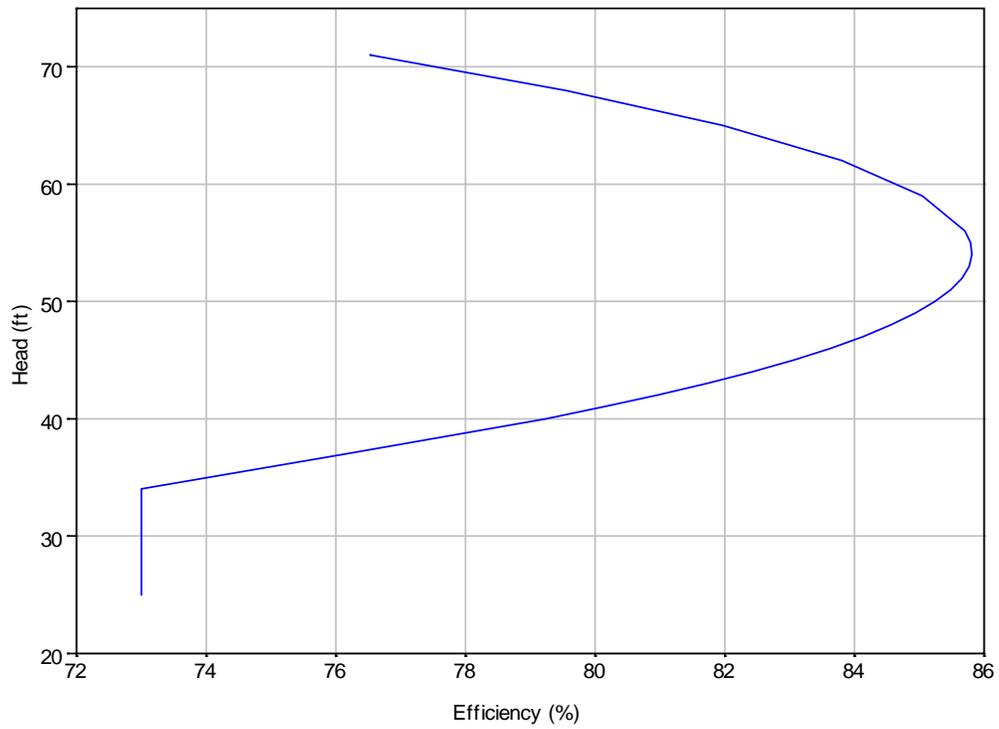
**Figure 3-35: OAHE powerplant efficiency.**



**Figure 3-36: BEND powerplant efficiency.**



**Figure 3-37: FTRA powerplant efficiency.**



**Figure 3-38: GAPT powerplant efficiency.**

## **4 RESSIM MODELING**

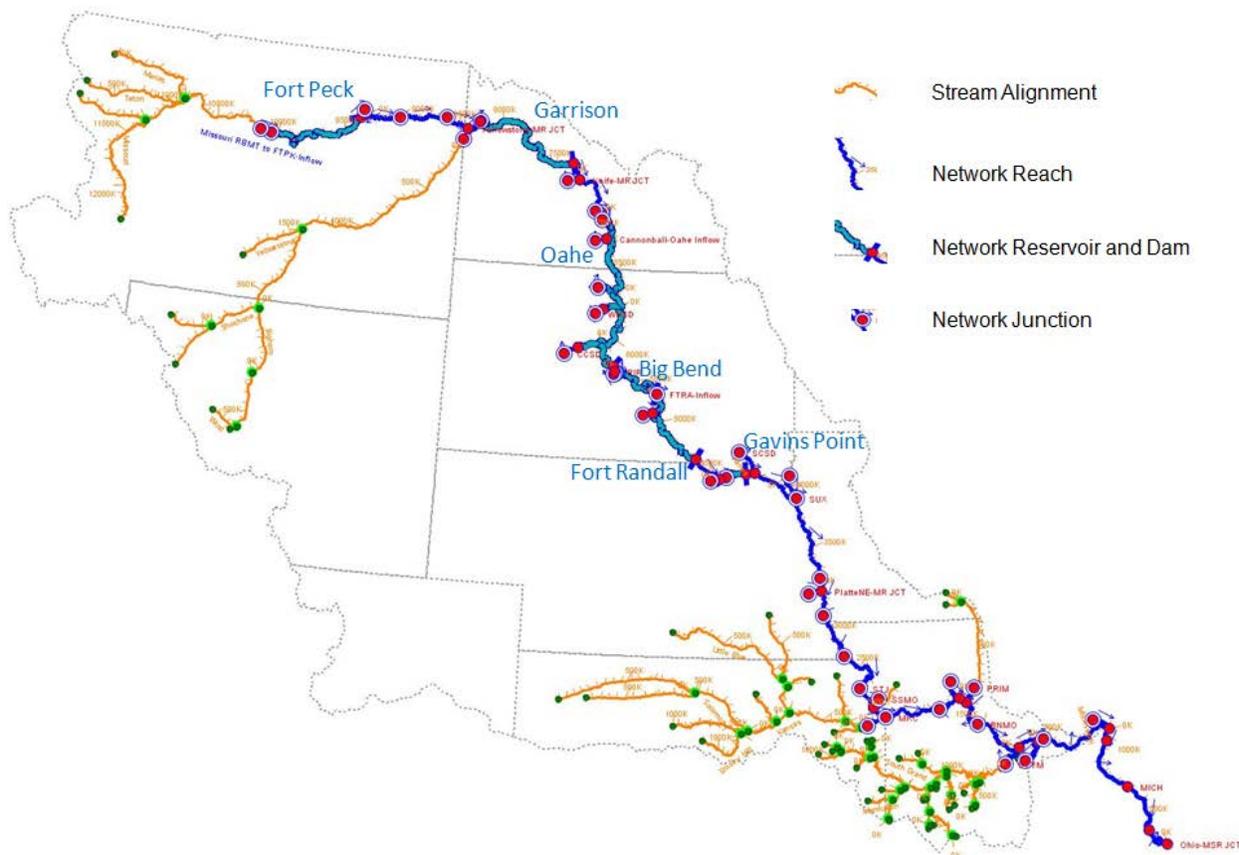
### **4.1 RESSIM PROGRAM OVERVIEW**

HEC-ResSim is a reservoir operations model developed by the USACE Hydrologic Engineering Center (HEC). The model incorporates user defined rules with other conditions (i.e. inflow, pool elevation, and downstream flows) to determine reservoir outflow. The model also performs downstream hydrologic channel routing. Water managers, water control manuals, and other documentation all can help in determining the rules necessary to simulate a reservoir within the model.

The model was simulated using a daily time interval. ResSim version 3.2, build 3.2.0.1221R, from April 2013 was used for modeling in this project.

### **4.2 MODELING EXTENTS**

A ResSim stream alignment provides the framework for what streams can be modeled in desired networks. The stream alignment created for the Missouri River Mainstem model (Figure 3-1) extends from the headwaters of the Missouri River down to the Ohio-Mississippi River confluence and includes major tributaries such as the Yellowstone, Kansas, Chariton, and Osage Rivers as well as segments of minor tributaries. The Network created for the mainstem model (Figure 4-1) begins at the RBMT gage located just upstream of Fort Peck on the Missouri River and terminates at the Mississippi-Ohio River confluence. It does not include the upper Missouri, Yellowstone, Kansas, Chariton, or Osage River Basins available from the stream alignment. These basins are being developed in ResSim externally, but having a common stream alignment is expected to facilitate combining the models in the future. At the downstream end, the model is only functional to the Hermann; MO gage (HEMO), as this is the last point which local flow is added into the System. No data for the Mississippi River was collected or included for modeling, but the stream itself was included in the Network for potential future alternative analyses.



**Figure 4-1: Missouri River Mainstem System ResSim Network.**

### 4.3 MODELING STRATEGY

#### 4.3.1 Overview

The objective of this ResSim model is to simulate system operation for the period of record for assessment of base conditions on the Missouri River. Early efforts focused on data and model planning and development. Later efforts emphasized model operations development and validation of model performance.

Missouri River System operation is extremely complex. Section 2.3 described the Mainstem System operation as outlined in the Master Manual. Various characteristics make it unique such as the physical arrangement of the reservoirs with the upper three holding most of the storage and the travel times between the reservoirs. Also the combined storage capacity of all six System reservoirs is 72.4 MAF, about three times the annual runoff into the System above Sioux City. This high ratio of storage capacity to runoff lends an unusual degree of flexibility to the regulation of the multipurpose reservoir system. In contrast, the ratio of reservoir storage capacity to annual runoff in the Columbia and Ohio River basins is 1:5, approximately one acre-foot of storage for each five acre-feet of annual runoff.

The concept of using a traditional guide curve is not applicable for the mainstem reservoirs, especially in the upper three reservoirs where their enormous storage provides flexibility for operation (see Section 2.3.1.3). All the reservoirs are in series, and releases must be planned systematically. Normal regulation releases are first determined from the most downstream dam, GAPT. These releases take into account the downstream flow targets (Sioux City--SUX, Omaha--OMA, Nebraska City--NCNE, and Kansas City—MKC), the incremental local flows between the flow targets, and the routing time it takes for water to reach the targets. Once a daily release has been determined at GAPT, releases are planned at the next upstream reservoir, again accounting for local flows, routing, and special reservoir conditions. The progression of release decisions moves upstream through Fort Peck. In essence, the upstream three reservoirs supply the volume needed at the most downstream Gavins Point to meet target flows. Big Bend passes water from Oahe and provides peaking power. Fort Randall follows a seasonal guide curve that lowers pool elevations in the fall to allow for higher winter hydropower releases from the upstream projects.

Releases from Gavins Point are generally highest during the navigation season when downstream flow requirements are highest. Since Gavins Point reservoir is small, these releases must be backed up with similar magnitude releases from Fort Randall, and Fort Randall requires similar support flows from Oahe via Big Bend. Here the chain can be interrupted; Oahe has enough storage to support high releases for extended periods without high inflows.

Because of the unique nature of the System, considerable time was spent deciding how it could best be modeled using HEC-ResSim. Face-to-face meetings between NWO, MRBWM, and HEC were required to discuss strategies to put historic operations into HEC-ResSim. The ResSim model does not capture every detail stated in the Master Manual and cannot capture special operations not outlined in the Master Manual; rather focus was turned to major decision-making stimuli. Major ResSim goals were established and included the following:

- System Regulation
  - Achieve system storage balance and storage level at the base of the Flood Control and Multiple Use Zone prior to March 15<sup>th</sup> assessment
  - Operate the upper three dams (FTPK, GARR, and OAHE) to pass sufficient volume through to Gavins Point so that it can meet downstream targets.
  - Meet Fort Randall's fall drawdown criteria and general guide curve elevation targets.
- Recurring Operational Considerations
  - Perform scheduled system checks on March 15 and July 1 to assess system storage and forecasted runoff which would determine the service level for navigation and flood evacuation operations. The July 1 check would also determine the navigation season length.
  - Perform additional weekly system storage and forecasted runoff checks during flood storage evacuation between May 1 and July 1. This would allow increasing releases for unforecasted runoff. For example, if the system was in flood storage evacuation and an assessment was performed May 1, flow targets would be set for the entire month. If multiple large rainfall events were to occur a week later,

- without additional system checks, releases would remain constant and pool levels could rise drastically.
- Keep releases below channel capacities when possible to reduce flooding.
  - Perform system storage check September 1 to determine GAPT winter releases. Also determine if a 10 day navigation season extension is warranted.
  - During navigation season, make releases from Gavins Point sufficient to meet navigation flow targets downstream (FTT scenario). Also provide a secondary option to operate some form of the SR-FTT scenario.
  - Establish and meet water supply/water quality/environmental flows.

It was desired to model these concepts and operations using the standard rules in ResSim whenever possible. Rate of change, minimum releases, maximum channel capacities, and induced surcharge curves were all entered as standard rules. Summaries of the rule stacks are provided in Section 4.3.2.

It was realized a scripted state variable approach would be required to apply the service level concept to determine downstream flow targets during the navigation season. Once downstream targets had been established, ResSim could theoretically be used to determine releases all the reservoirs to meet those targets and maintain proper pool elevations. The standard features in ResSim, however, could not accomplish this task because of ResSim's computation progression. During a simulation, ResSim begins computations at the most upstream reservoir and progresses downstream, but the operations of the Missouri River reservoirs are reversed. Releases are set at the furthest downstream reservoir and progress upstream. Because ResSim computes releases from upstream reservoirs first and the furthest downstream reservoir, GAPT, has little storage, ResSim would not compute the necessary releases from the upper 3 reservoirs to supply GAPT with enough volume to keep its pool elevation above the inactive pool while also releasing enough water to meet downstream flow requirements. If a reservoir's pool elevation reaches its inactive zone during a ResSim simulation, ResSim will override all rules and lower releases until the pool elevation rises above the inactive zone. This frequently occurred at GAPT, which caused the model to consistently miss downstream flow targets.

To overcome the limitations of the standard features of ResSim, two modeling decisions were made. First, two models would be used to properly operate the system. A Downstream model was created that consisted of one reservoir with a capacity equal to the system capacity located at GAPT and all of the downstream gage locations. This model would be responsible for assessing the system storage and making releases for downstream operations: service level, navigation season length, flood constraints, water supply, etc. A System model would then set GAPT's releases equal to the releases computed from the Downstream model and determine the releases for the other five reservoirs. The second decision was to use scripts to operate five of the reservoirs: FTRA, BEND, OAHE, GARR, and FTPK. Utilizing scripts to set releases at these reservoirs allows ResSim to first set releases at FTRA and progress upstream. The rule stacks for both models are described in Section 4.3.2, and the scripts are described in Sections 4.3.4 and 4.3.4.

### 4.3.2 Rules

Summaries of the rule stacks used in HEC-ResSim for each of the mainstem reservoirs and explanations of reservoir guide curves and special operations are provided in the following Table 4-1 through Table 4-7. Criteria for the rules were identified from the Master Manual, Daily Routing Model (DRM), and observations of historic operations. The rules appear as they were ordered within the model.

**Table 4-1: Fort Peck operations/rule stack in the System model.**

Pool Zone	Rule Name	Description
Surcharge (Guide Curve)	Service Level <sup>3</sup>	Runs the ServiceLevel state variable which computes the service level, navigation season length, system storage, etc. No decisions are based on this state variable in the System model, but it is still computed because it produces variables such as system storage. Placed at FTPK because ResSim calculates from upstream to downstream.
	ServiceLevel_x1k <sup>3</sup>	Runs the ServiceLevel_x1K state variable. Calculates navigation target discharges at the downstream target locations based on the service level. No decisions are based on this state variable in the System model, but it is still computed because the ServiceLevel state variable is computed. Placed at FTPK because ResSim calculates from upstream to downstream.
	GAPT Control <sup>1, 2</sup>	Scripted rule to estimate OAHE, BEND, and FTRA releases using forecasted GAPT release values calculated from Missouri River Downstream model with all reservoir storage included in GAPT. This script first calculates specified releases for FTRA based on GAPT releases calculated from the Downstream model that allows GAPT to remain at guide curve. Then it calculates OAHE releases based on FTRA releases that allows FTRA to remain at guide curve. Finally, it calculates BEND releases based on its guide curve and the inflow into the reservoir.
	Reservoir Balancing <sup>1, 2</sup>	Scripted rule to set releases at FTPK and GARR that will unbalance/balance storage in FTPK, GARR, and OAHE. This calculation is performed on the 1st and 15th of each month during normal runoff and every 7 days during flooding. The script accounts for forecasted inflow into FTPK, GARR, and OAHE.
	Water Supply <sup>4</sup>	Scripted rule that ensures minimum discharges are met at each gauge location downstream of FTPK and GARR
Exclusive Flood Control	ROC Limit for Flood - Increase	Limits discharge rate of change increase to 375 cfs/hour (9,000 cfs per day).
	ROC Limit for Flood - Decrease	Limits discharge rate of change decrease to 500 cfs/hour (12,000 cfs per day).
	Service Level <sup>3</sup>	See description in the Surcharge Zone
	ServiceLevel_x1k <sup>3</sup>	See description in the Surcharge Zone
	GAPT Control <sup>1, 2</sup>	Scripted rule: see description in the Surcharge Zone
	Reservoir Balancing <sup>1, 2</sup>	Scripted rule: see description in the Surcharge Zone
	Water Supply <sup>4</sup>	Scripted rule that ensures minimum discharges are met at each gauge location downstream of FTPK and GARR
Flood Control & Multiple Use	ROC Limit for Normal - Increase	Limits discharge rate of change increase to 250 cfs/hour (6,000 cfs per day).
	ROC Limit for Normal - Decrease	Limits discharge rate of change decrease to 125 cfs/hour (3,000

		cfs per day).
	Service Level <sup>3</sup>	See description in the Surcharge Zone
	ServiceLevel_x1k <sup>3</sup>	See description in the Surcharge Zone
	GAPT Control <sup>1, 2</sup>	Scripted rule: see description in the Surcharge Zone
	Reservoir Balancing <sup>1, 2</sup>	Scripted rule: see description in the Surcharge Zone
	Water Supply <sup>4</sup>	Scripted rule that ensures minimum discharges are met at each gauge location downstream of FTPK and GARR
Carryover Multiple Use	ROC Limit for Normal - Increase	See description in the Flood Control & Multiple Use Zone
	ROC Limit for Normal - Decrease	See description in the Flood Control & Multiple Use Zone
	Service Level <sup>3</sup>	See description in the Surcharge Zone
	ServiceLevel_x1k <sup>3</sup>	See description in the Surcharge Zone
	GAPT Control <sup>1, 2</sup>	Scripted rule: see description in the Surcharge Zone
	Reservoir Balancing <sup>1, 2</sup>	Scripted rule: see description in the Surcharge Zone
	Water Supply <sup>4</sup>	Scripted rule that ensures minimum discharges are met at each gauge location downstream of FTPK and GARR
Permanent	ROC Limit for Normal - Increase	See description in the Flood Control & Multiple Use Zone
	ROC Limit for Normal - Decrease	See description in the Flood Control & Multiple Use Zone
	Service Level <sup>3</sup>	See description in the Surcharge Zone
	ServiceLevel_x1k <sup>3</sup>	See description in the Surcharge Zone
	GAPT Control <sup>1, 2</sup>	Scripted rule: see description in the Surcharge Zone
	Reservoir Balancing <sup>1, 2</sup>	Scripted rule: see description in the Surcharge Zone
	Water Supply <sup>4</sup>	Scripted rule that ensures minimum discharges are met at each gauge location downstream of FTPK and GARR
No Storage (Inactive)	--	No rules apply to the No Storage or Inactive Zone

Note: Future/hypothetical extreme flood events may have discharges higher than the 2011 event, at which time, the scripted rules will have to be revisited to ensure the reservoirs are operated correctly

<sup>1</sup>Seasonal intrasystem operation

<sup>2</sup>Flood control operation

<sup>3</sup>Water requirement below Gavins Point operations

<sup>4</sup>Water requirement above Gavins Point operation

**Table 4-2: Garrison operations/rule stack in the System model.**

Pool Zone	Rule Name	Description
Surcharge (Guide Curve)	Reservoir Balancing	Relates GARR release to Reservoir Balancing scripted rule (see FTPK rules) release at FTPK on a 1 to 1 basis.
Exclusive Flood Control	ROC Limit for Flood - Increase	Limits discharge rate of change increase to 375 cfs/hr (9,000 cfs/day).
	ROC Limit for Flood - Decrease	Limits discharge rate of change decrease to 500 cfs/hr (12,000 cfs/day).
	Reservoir Balancing <sup>1, 2</sup>	See description in the Surcharge Zone
Flood Control & Multiple Use	ROC Limit for Normal - Increase	Limits discharge rate of change increase to 250 cfs/hr (6,000 cfs/day).
	ROC Limit for Normal - Decrease	Limits discharge rate of change decrease to 125 cfs/hr (3,000 cfs/day).
	Reservoir Balancing <sup>1, 2</sup>	See description in the Surcharge Zone
Carryover Multiple Use	ROC Limit for Normal - Increase	See description in the Flood Control & Multiple Use Zone
	ROC Limit for Normal - Decrease	See description in the Flood Control & Multiple Use Zone

	Reservoir Balancing <sup>1,2</sup>	See description in the Surchage Zone
Permanent	ROC Limit for Normal - Increase	See description in the Flood Control & Multiple Use Zone
	ROC Limit for Normal - Decrease	See description in the Flood Control & Multiple Use Zone
	Reservoir Balancing <sup>1,2</sup>	See description in the Surchage Zone
No Storage (Inactive)	--	No rules apply to the No Storage or Inactive Zone

<sup>1</sup>Seasonal intrasystem operation

<sup>2</sup>Flood control operation

<sup>3</sup>Water requirement below Gavins Point operations

<sup>4</sup>Water requirement above Gavins Point operation

**Table 4-3: Oahe operations/rule stack in the System model.**

Pool Zone	Rule Name	Description
Surchage (Guide Curve)	Specified Release <sup>1,2</sup>	Relates OAHE release to GAPT Control scripted rule (see FTPK rules) release at FTPK on a 1 to 1 basis.
Exclusive Flood Control	Power Release - Min <sup>4</sup>	Limits min discharge to 1,000 cfs for sustained power flow.
	ROC Limit for Normal - Increase	Limits discharge rate of change increase to 833 cfs/hr (20,000 cfs/day).
	ROC Limit for Normal - Decrease	Limits discharge rate of change increase to 833 cfs/hr (20,000 cfs/day).
	Specified Release <sup>1,2</sup>	See description in the Surchage Zone
Flood Control & Multiple Use	Power Release - Min <sup>4</sup>	See description in the Exclusive Flood Control Zone
	ROC Limit for Normal - Increase	See description in the Exclusive Flood Control Zone
	ROC Limit for Normal - Decrease	See description in the Exclusive Flood Control Zone
	Specified Release <sup>1,2</sup>	See description in the Surchage Zone
Carryover Multiple Use	Power Release - Min <sup>4</sup>	See description in the Exclusive Flood Control Zone
	ROC Limit for Normal - Increase	See description in the Exclusive Flood Control Zone
	ROC Limit for Normal - Decrease	See description in the Exclusive Flood Control Zone
	Specified Release <sup>1,2</sup>	See description in the Surchage Zone
Permanent	Power Release - Min <sup>4</sup>	See description in the Exclusive Flood Control Zone
	ROC Limit for Normal - Increase	See description in the Exclusive Flood Control Zone
	ROC Limit for Normal - Decrease	See description in the Exclusive Flood Control Zone
	Specified Release <sup>1,2</sup>	See description in the Surchage Zone
No Storage (Inactive)	--	No rules apply to the No Storage or Inactive Zone

<sup>1</sup>Seasonal intrasystem operation

<sup>2</sup>Flood control operation

<sup>3</sup>Water requirement below Gavins Point operations

<sup>4</sup>Water requirement above Gavins Point operation

**Table 4-4: Big Bend operations/rule stack in the System model.**

Pool Zone	Rule Name	Description
Top of Dam Zone	Power Release – Tailwater <sup>4</sup>	Determine max powerplant release since at BEND, powerplant max release is a function of downstream Lake Francis Case pool elevation
	Specified Release <sup>1,2</sup>	Relates BEND release to GAPT Control scripted rule (see FTPK rules) release at FTPK on a 1 to 1 basis.

Exclusive Flood Control	Power Release – Tailwater <sup>4</sup>	See description in the Top of Dam Zone
	Specified Release <sup>1, 2</sup>	See description in the Top of Dam Zone
Flood Control & Multiple Use	Power Release – Tailwater <sup>4</sup>	See description in the Top of Dam Zone
	Specified Release <sup>1, 2</sup>	See description in the Top of Dam Zone
Guide Curve (Guide Curve)	Power Release – Tailwater <sup>4</sup>	See description in the Top of Dam Zone
	Specified Release <sup>1, 2</sup>	See description in the Top of Dam Zone
Permanent	Power Release – Tailwater <sup>4</sup>	See description in the Top of Dam Zone
	Specified Release <sup>1, 2</sup>	See description in the Top of Dam Zone
No Storage (Inactive)	--	No rules apply to the No Storage or Inactive Zone

<sup>1</sup>Seasonal intrasystem operation

<sup>2</sup>Flood control operation

<sup>3</sup>Water requirement below Gavins Point operations

<sup>4</sup>Water requirement above Gavins Point operation

**Table 4-5: Fort Randall operations/rule stack in the System model.**

Pool Zone	Rule Name	Description
Surcharge	Specified Release <sup>1, 2</sup>	Relates FTRA release to GAPT Control scripted rule (see FTPK rules) release at FTPK on a 1 to 1 basis.
Exclusive Flood Control	ROC Limit for Flood - Increase	Limits discharge rate of change increase to 500 cfs/hour (12,000 cfs per day).
	ROC Limit for Flood - Decrease	Limits discharge rate of change decrease to 708 cfs/hour (17,000 cfs per day).
	Specified Release <sup>1, 2</sup>	See description in the Surcharge Zone
Flood Control & Multiple Use	ROC Limit for Normal - Increase	Limits discharge rate of change increase to 500 cfs/hour (12,000 cfs per day).
	ROC Limit for Normal - Decrease	Limits discharge rate of change decrease to 500 cfs/hour (12,000 cfs per day).
	Specified Release <sup>1, 2</sup>	See description in the Surcharge Zone
Guide Curve (Guide Curve)	ROC Limit for Normal - Increase	See description in the Flood Control & Multiple Use Zone
	ROC Limit for Normal - Decrease	See description in the Flood Control & Multiple Use Zone
	Specified Release <sup>1, 2</sup>	See description in the Surcharge Zone
Permanent	ROC Limit for Normal - Increase	See description in the Flood Control & Multiple Use Zone
	ROC Limit for Normal - Decrease	See description in the Flood Control & Multiple Use Zone
	Specified Release <sup>1, 2</sup>	See description in the Surcharge Zone
No Storage (Inactive)	--	No rules apply to the No Storage or Inactive Zone

<sup>1</sup>Seasonal intrasystem operation

<sup>2</sup>Flood control operation

<sup>3</sup>Water requirement below Gavins Point operations

<sup>4</sup>Water requirement above Gavins Point operation

Fort Randall's operation includes an annual drawdown during the autumn. Guidance from MRBWM suggested a drop from elevation 1355 ft to 1337.5 from Labor Day to the end of the navigation season. The pool will refill during the winter power season and be near the base of flood control and multiple use on March 1 (elevation 1350 ft.). Table 4-6 below summarizes the guide curve used in ResSim. Linear interpolation is applied between dates.

**Table 4-6: Fort Randall guide curve.**

Date	Elevation (ft)
1-Jan	1341.8
1-Mar	1350.0
1-Apr	1355.0
1-Sep	1355.0
1-Dec	1337.5

**Table 4-7: Gavins Point operations/rule stack in the System model.**

Pool Zone	Rule Name	Description
Surcharge	Forecast Release <sup>2, 3</sup>	Relates GAPT release to GAPT Control scripted rule (see FTPK rules) release at FTPK on a 1 to 1 basis.
Exclusive Flood Control	Forecast Release <sup>2, 3</sup>	See description in the Surcharge Zone
Flood Control & Multiple Use	Forecast Release <sup>2, 3</sup>	See description in the Surcharge Zone
Guide Curve (Guide Curve)	Forecast Release <sup>2, 3</sup>	See description in the Surcharge Zone
Permanent	Forecast Release <sup>2, 3</sup>	See description in the Surcharge Zone
No Storage (Inactive)	--	No rules apply to the No Storage or Inactive Zone

<sup>1</sup>Seasonal intrasystem operation

<sup>2</sup>Flood control operation

<sup>3</sup>Water requirement below Gavins Point operations

<sup>4</sup>Water requirement above Gavins Point operation

**Table 4-8: Gavins Point guide curve.**

Date	Elevation (ft)
1-Jan	1207.0
1-Feb	1206.0
1-Aug	1206.0
1-Sep	1207.0
1-Dec	1207.0

**Table 4-9: Gavins Point operations/rule stack in the Downstream model.**

Pool Zone	Rule Name	Description
Surcharge	ROC Limit for Flood - Increase	Limits discharge rate of change increase to 450 cfs/hour (10,800 cfs per day).
	ROC Limit for Flood – Decrease	Limits discharge rate of change increase to 600 cfs/hour (14,400 cfs per day).
	Rate of Change If Block	This block of rules specifies the rate of change for System releases during normal and flood operations
	If(High Release)	To enter this block, the system release needs to be greater than 50,000 cfs
	ROC Limit for Flood - Increase	Limits discharge rate of change increase to 450 cfs/hour (10,800 cfs per day).
	ROC Limit for Flood – Decrease	Limits discharge rate of change increase to 600 cfs/hour (14,400 cfs per day).
	Else()	To enter this block, the system release needs to be <= 50,000 cfs
	ROC Limit for Normal - Increase	Limits discharge rate of change increase to 125 cfs/hour (3,000 cfs per day).
	ROC Limit for Normal – Decrease	Limits discharge rate of change increase to 125 cfs/hour (3,000 cfs per day).
	Flood Evacuation If Block	This block of rules specifies system releases when flood storage needs to be evacuated
	If(High Spring, Summer, and Fall Release)	To enter this block, the date is 15Mar-01Dec and the service level is >= 35,000 cfs.
	Flood Evacuation <sup>2</sup>	Scripted rule that forecasts the system storage on the following 01Mar. If the system storage > 56.1 MAF, minimum releases are increased
	Else If(High Winter Release)	To enter this block, the system storage >= 56.1 MAF and the date is 01Jan – 21Mar or 02Dec – 31Dec
	High Winter Releases <sup>2</sup>	Sets a higher winter release between 20,000 – 27,000 cfs to continue evacuation of flood storage.
	Else(Reduce High Winter Release)	To enter this block, the system storage < 56.1 MAF and the date is 01Jan – 21Mar or 02Dec – 31Dec
	Cut Winter Release <sup>2</sup>	Cuts the high winter release if system storage drops below 56.1 MAF
	WS, NAV, and Flood Targets <sup>2, 3</sup>	Scripted rule that sets minimum System releases in order to meet minimum water supply requirements, navigation target discharges, and cuts releases for downstream flooding.
	Steady Release If Block	This block of rules sets the steady release during the summer
	If (Steady Release)	To enter this block, the date is 15Mar – 15Jul and there is no downstream flooding
	Steady Release <sup>3</sup>	Scripted rule that sets minimum System release to the steady release value
Service Level <sup>3</sup>	Runs the ServiceLevel state variable which computes the service level, navigation season length, system storage, etc.	
ServiceLevel_x1k <sup>3</sup>	Runs the ServiceLevel_x1K state variable. Calculates navigation target discharges at the downstream target locations based on the service level.	

<sup>1</sup>Seasonal intrasystem operation

<sup>2</sup>Flood control operation

<sup>3</sup>Water requirement below Gavins Point operations

<sup>4</sup>Water requirement above Gavins Point operation

Pool Zone	Rule Name	Description
Exclusive Flood Control	ROC Limit for Flood - Increase	Limits discharge rate of change increase to 450 cfs/hour (10,800 cfs per day).
	ROC Limit for Flood – Decrease	Limits discharge rate of change increase to 600 cfs/hour (14,400 cfs per day).
	Rate of Change If Block	This block of rules specifies the rate of change for System releases during normal and flood operations
	If(High Release)	To enter this block, the system release needs to be greater than 50,000 cfs
	ROC Limit for Flood - Increase	Limits discharge rate of change increase to 450 cfs/hour (10,800 cfs per day).
	ROC Limit for Flood – Decrease	Limits discharge rate of change increase to 600 cfs/hour (14,400 cfs per day).
	Else()	To enter this block, the system release needs to be <= 50,000 cfs
	ROC Limit for Normal - Increase	Limits discharge rate of change increase to 125 cfs/hour (3,000 cfs per day).
	ROC Limit for Normal – Decrease	Limits discharge rate of change increase to 125 cfs/hour (3,000 cfs per day).
	Flood Evacuation If Block	This block of rules specifies system releases when flood storage needs to be evacuated
	If(High Spring, Summer, and Fall Release)	To enter this block, the date is 15Mar-01Dec and the service level is >= 35,000 cfs.
	Flood Evacuation <sup>2</sup>	Scripted rule that forecasts the system storage on the following 01Mar. If the system storage > 56.1 MAF, minimum releases are increased
	Else If(High Winter Release)	To enter this block, the system storage >= 56.1 MAF and the date is 01Jan – 21Mar or 02Dec – 31Dec
	High Winter Releases <sup>2</sup>	Sets a higher winter release between 20,000 – 27,000 cfs to continue evacuation of flood storage.
	Else(Reduce High Winter Release)	To enter this block, the system storage < 56.1 MAF and the date is 01Jan – 21Mar or 02Dec – 31Dec
	Cut Winter Release <sup>2</sup>	Cuts the high winter release if system storage drops below 56.1 MAF
	WS, NAV, and Flood Targets <sup>2, 3</sup>	Scripted rule that sets minimum System releases in order to meet minimum water supply requirements, navigation target discharges, and cuts releases for downstream flooding.
	Steady Release If Block	This block of rules sets the steady release during the summer
	If (Steady Release)	To enter this block, the date is 15Mar – 15Jul and there is no downstream flooding
	Steady Release <sup>3</sup>	Scripted rule that sets minimum System release to the steady release value
Service Level <sup>3</sup>	Runs the ServiceLevel state variable which computes the service level, navigation season length, system storage, etc.	
ServiceLevel_x1k <sup>3</sup>	Runs the ServiceLevel_x1K state variable. Calculates navigation target discharges at the downstream target locations based on the service level.	

<sup>1</sup>Seasonal intrasystem operation

<sup>2</sup>Flood control operation

<sup>3</sup>Water requirement below Gavins Point operations

<sup>4</sup>Water requirement above Gavins Point operation

Pool Zone	Rule Name	Description
Flood Control and Multiple Use	Spring Pulse <sup>3</sup>	Scripted rule that sets the spring pulse releases
	ROC Limit for Flood - Increase	Limits discharge rate of change increase to 450 cfs/hour (10,800 cfs per day).
	ROC Limit for Flood – Decrease	Limits discharge rate of change increase to 600 cfs/hour (14,400 cfs per day).
	Rate of Change If Block	This block of rules specifies the rate of change for System releases during normal and flood operations
	If(High Release)	To enter this block, the system release needs to be greater than 50,000 cfs
	ROC Limit for Flood - Increase	Limits discharge rate of change increase to 450 cfs/hour (10,800 cfs per day).
	ROC Limit for Flood – Decrease	Limits discharge rate of change increase to 600 cfs/hour (14,400 cfs per day).
	Else()	To enter this block, the system release needs to be <= 50,000 cfs
	ROC Limit for Normal - Increase	Limits discharge rate of change increase to 125 cfs/hour (3,000 cfs per day).
	ROC Limit for Normal – Decrease	Limits discharge rate of change increase to 125 cfs/hour (3,000 cfs per day).
	Flood Evacuation If Block	This block of rules specifies system releases when flood storage needs to be evacuated
	If(High Spring, Summer, and Fall Release)	To enter this block, the date is 15Mar-01Dec and the service level is >= 35,000 cfs.
	Flood Evacuation <sup>2</sup>	Scripted rule that forecasts the system storage on the following 01Mar. If the system storage > 56.1 MAF, minimum releases are increased
	Else If(High Winter Release)	To enter this block, the system storage >= 56.1 MAF and the date is 01Jan – 21Mar or 02Dec – 31Dec
	High Winter Releases <sup>2</sup>	Sets a higher winter release between 20,000 – 27,000 cfs to continue evacuation of flood storage.
	Else(Reduce High Winter Release)	To enter this block, the system storage < 56.1 MAF and the date is 01Jan – 21Mar or 02Dec – 31Dec
	Cut Winter Release <sup>2</sup>	Cuts the high winter release if system storage drops below 56.1 MAF
	WS, NAV, and Flood Targets <sup>2, 3</sup>	Scripted rule that sets minimum System releases in order to meet minimum water supply requirements, navigation target discharges, and cuts releases for downstream flooding.
	Steady Release If Block	This block of rules sets the steady release during the summer
	If (Steady Release)	To enter this block, the date is 15Mar – 15Jul and there is no downstream flooding
	Steady Release <sup>3</sup>	Scripted rule that sets minimum System release to the steady release value
	Service Level <sup>3</sup>	Runs the ServiceLevel state variable which computes the service level, navigation season length, system storage, etc.
	ServiceLevel_x1k <sup>3</sup>	Runs the ServiceLevel_x1K state variable. Calculates navigation target discharges at the downstream target locations based on the service level.

<sup>1</sup>Seasonal intrasystem operation

<sup>2</sup>Flood control operation

<sup>3</sup>Water requirement below Gavins Point operations

<sup>4</sup>Water requirement above Gavins Point operation

Pool Zone	Rule Name	Description
Carryover and Multiple Use	Spring Pulse <sup>3</sup>	Scripted rule that sets the spring pulse releases
	ROC Limit for Flood - Increase	Limits discharge rate of change increase to 450 cfs/hour (10,800 cfs per day).
	ROC Limit for Flood – Decrease	Limits discharge rate of change increase to 600 cfs/hour (14,400 cfs per day).
	Rate of Change If Block	This block of rules specifies the rate of change for System releases during normal and flood operations
	If(High Release)	To enter this block, the system release needs to be greater than 50,000 cfs
	ROC Limit for Flood - Increase	Limits discharge rate of change increase to 450 cfs/hour (10,800 cfs per day).
	ROC Limit for Flood – Decrease	Limits discharge rate of change increase to 600 cfs/hour (14,400 cfs per day).
	Else()	To enter this block, the system release needs to be <= 50,000 cfs
	ROC Limit for Normal - Increase	Limits discharge rate of change increase to 125 cfs/hour (3,000 cfs per day).
	ROC Limit for Normal – Decrease	Limits discharge rate of change increase to 125 cfs/hour (3,000 cfs per day).
	Flood Evacuation If Block	This block of rules specifies system releases when flood storage needs to be evacuated
	If(High Spring, Summer, and Fall Release)	To enter this block, the date is 15Mar-01Dec and the service level is >= 35,000 cfs.
	Flood Evacuation <sup>2</sup>	Scripted rule that forecasts the system storage on the following 01Mar. If the system storage > 56.1 MAF, minimum releases are increased
	Else If(High Winter Release)	To enter this block, the system storage >= 56.1 MAF and the date is 01Jan – 21Mar or 02Dec – 31Dec
	High Winter Releases <sup>2</sup>	Sets a higher winter release between 20,000 – 27,000 cfs to continue evacuation of flood storage.
	Else(Reduce High Winter Release)	To enter this block, the system storage < 56.1 MAF and the date is 01Jan – 21Mar or 02Dec – 31Dec
	Cut Winter Release <sup>2</sup>	Cuts the high winter release if system storage drops below 56.1 MAF
	WS, NAV, and Flood Targets <sup>2, 3</sup>	Scripted rule that sets minimum System releases in order to meet minimum water supply requirements, navigation target discharges, and cuts releases for downstream flooding.
	Steady Release If Block	This block of rules sets the steady release during the summer
	If (Steady Release)	To enter this block, the date is 15Mar – 15Jul and there is no downstream flooding
	Steady Release <sup>3</sup>	Scripted rule that sets minimum System release to the steady release value
Service Level <sup>3</sup>	Runs the ServiceLevel state variable which computes the service level, navigation season length, system storage, etc.	
ServiceLevel_x1k <sup>3</sup>	Runs the ServiceLevel_x1K state variable. Calculates navigation target discharges at the downstream target locations based on the service level.	
Permanent (Inactive)	--	No rules apply to the No Storage or Inactive Zone

<sup>1</sup>Seasonal intrasystem operation

<sup>2</sup>Flood control operation

<sup>3</sup>Water requirement below Gavins Point operations

<sup>4</sup>Water requirement above Gavins Point operation

### **4.3.3 Downstream Model vs System Model**

Two models are used to simulate the Missouri River main stem reservoir operations: a Downstream and System model. The Downstream model assumes that storage from all six reservoirs is located within one reservoir at GAPT. This is done because the Missouri River main stem reservoir operations are performed in a downstream to upstream manner where GAPT releases are first set, and then all releases from the upper reservoirs are set so enough storage remains in GAPT to meet required releases and maintain desired pool elevations. Therefore, the Downstream model contains all the rules needed for downstream operations: service level, navigation season length, flood constraints, water supply, etc, and calculates GAPT releases for the period-of-record. Once GAPT releases have been calculated for the period-of-record, the System model sets releases for the other five reservoirs upstream of GAPT to ensure that enough storage remains in GAPT to meet GAPT releases calculated from the Downstream model.

### **4.3.4 Scripted Rules in the Downstream Model**

#### **4.3.4.1 WS, NAV, and Flood Targets Scripted Rule**

The WS, NAV, and Flood Targets scripted rule is responsible for ensuring GAPT's releases meet navigation target discharges, water supply requirements, and reducing GAPT's releases to reduce downstream flooding. If there is a navigation season, which is determined on 01 Mar, navigation requirements begin to be assessed on 15 Mar when a 10-day forecast is created by routing Gavins Point releases and local inflows downstream and checking navigation target discharges at Sioux City, Omaha, Nebraska City, and Kansas City; Gavins Point releases are increased by 500 cfs until each target discharge is met. Navigation target discharges are discussed in more detail in Section 4.3.4.1.

Once a release is set for navigation, flood targets are assessed using the same 10-day forecast created for navigation releases and checking if the forecasted discharges exceed the current flood targets at Omaha, Nebraska City, and Kansas City. If the flood targets are exceeded, Gavins Point releases are decreased to minimize flooding while still balancing service to navigation at the other locations. Although ResSim contains standard flood control features that allow reservoirs to set releases such that downstream flood target discharges are not exceeded, ResSim's standard flood control features do not allow for the dual check of minimizing flooding while still meeting the navigation requirements.

**Table 4-10: Downstream flood targets. Summarized from Tables VII-7 and VII-8 in the Master Manual (U.S. Army Corps of Engineers, 2006).**

	Flood Targets	
	Full-Service (1 <sup>st</sup> Level)	Minimum-Service (2 <sup>nd</sup> Level)
Omaha	Target Discharge + 10,000 cfs	Target Discharge + 15,000 cfs
Nebraska City	Target Discharge + 10,000 cfs	Target Discharge + 20,000 cfs
Kansas City	Target Discharge + 30,000 cfs	Target Discharge + 60,000 cfs

The scripted rule first calculates the flood target at each of the three target locations: Omaha, Nebraska City, and Kansas City, using the criteria in Table 4-10. If the lower of the two flood targets, full-service flood target, is exceeded, the navigation target discharges are set to full-service. Then Gavins Point releases are decreased by 500 cfs until either discharges at the three flood target locations are less than the full-service flood targets or releases reach a minimum required to meet the full-service navigation target discharges at Sioux City, Omaha, Nebraska City, and Kansas City. If the higher of the two flood targets, minimum-service flood target, is exceeded, the navigation target discharges are set to minimum-service. Gavins Point releases are then decreased by 500 cfs until either discharges at the three flood target locations are less than the minimum-service flood targets or releases reach a minimum required to meet the minimum-service navigation target discharges at Sioux City, Omaha, Nebraska City, and Kansas City. If flood targets require releases less than 9,000 cfs during March – November, the flood target logic will be overwritten and a minimum release of 9,000 cfs will be specified. If none of the flood target discharges are exceeded, normal navigation releases will occur. A more verbose explanation of the flood targets is seen in Section 7-04.15 in the Master Manual (U.S. Army Corps of Engineers, 2006).

#### **4.3.4.2 Flood Evacuation Scripted Rule**

The flood evacuation scripted rule is used in conjunction with Plate VI-1 in the Master Manual to evacuate flood storage. Plate VI-1 sets the service level when System storage has reached thresholds and higher releases are needed to evacuate the stored water. This is discussed in more detail in Section 4.3.4.4. Even with higher service levels established by Plate VI-1, Gavins Point releases can still be reduced to help with downstream flooding. If the System does not have enough flood storage available to help with downstream flooding, reducing Gavins Point releases may be detrimental to System operation; the flood evacuation scripted rule provides higher minimum releases during such scenarios. On the 1<sup>st</sup> and 15<sup>th</sup> of each month the forecasted reservoir system water supply and average Gavins Point releases are used to calculate an expected system storage for the following March 1<sup>st</sup>. If system storage is above the base of the multiple use and annual flood control zone, minimum releases are increased to ensure Gavins Point releases are not reduced and the current year's flood storage is evacuated before the beginning of the next runoff season. These release minimums increase as the forecasted system storage in the multiple use and annual flood control zone increases.

The remaining year runoff forecast is a combination of the POR forecasted Mar-Jul runoff and average runoff from Aug through the following Mar. This provides the forecasted inflow into the reservoir system at the current timestep. Average Gavins Point releases from the current date through the following Mar are subtracted from the forecasted inflow to calculate a change in storage over that period. The current reservoir system storage is then added to the change in storage to determine the forecasted system reservoir storage for March 1<sup>st</sup>. If the forecasted March 1<sup>st</sup> system storage is above the base of the multiple use and annual flood control zone, Gavins Point minimum releases are increased for the remainder of the year. If a small volume of water needs evacuated, the model will release the water in October and November. As the volume increases, Gavins Point minimum releases will be increased first in September, then August, then July and lastly June and May.

#### **4.3.4.3 Steady Release Scripted Rule**

The Steady Release scripted rule sets Gavins Point release to the Steady Release state variable value. The Steady Release state variable is computed by the Service Level scripted rule described in Section 4.3.1.4. The steady release is effective from May15-Jul15, the current estimate of endangered species nesting. If releases above the steady release are required to meet navigation targets, the script will set a new steady release and hold that release throughout the remainder of the period. If a flood event occurs, the Steady Release scripted rule will allow releases to be reduced to releases specified by the WS, NAV, and Flood Targets scripted rule. Following the flood event, the Steady Release scripted rule will increase flows back to the peak Steady Release set during that year.

#### **4.3.4.4 Service Level State Variable**

##### *4.3.4.4.1 Overview*

The Service Level state variable script is a key component to the ResSim model. When run, it produces numerous state variables which are used for various release decisions for both FTT and SR-FTT release scenarios. A detailed description is located below in Section 4.3.4.4.2 as well as a full text version in Appendix E – Service Level State Variable.

The service level script primarily creates state variables that are used to determine releases from Gavins Point Dam during the navigation and winter release seasons. It brings in data from outside sources such as historic forecasted runoff data, historical quantile flows and releases, and Plate VI-1 (Figure 4-2) from the Master Manual to make calculations. Much of the script looks at defining a “service level” for the system. Service level determines flow targets at four locations downstream of Gavins Point. A “full-service” level of 35,000 cfs results in target flows of 31,000 (-4,000) cfs at Sioux City and Omaha, 37,000 (+2,000) cfs at Nebraska City and 41,000 (+6,000) cfs at Kansas City. Similarly, a “minimum-service” level of 29,000 cfs results in target flow values of 6,000 cfs less than the full-service levels. Storage evacuation service levels are those above 35,000 cfs that set higher targets downstream and aid in evacuating flood water.

The first of three scheduled System assessments is performed on March 15 to determine the initial service level. The service level depends on water supply and is estimated from Plate VI-1 (Figure 4-2) from the Master Manual. Water supply for the script is determined from the actual current system storage and the forecasted remaining calendar year runoff volume above Gavins Point Dam. This service level sets the downstream targets until July 1<sup>st</sup> with exceptions. If the service level is in the storage evacuation category, additional service level checks are performed so that targets can be updated to release more appropriate volumes of water. Another exception occurs if it is desired to use a SR-FTT scenario. In this case, FTT is practiced until the nesting season begins on May 15. Gavins Point then releases at an initial steady rate through July 15. If cycling is desired, releases from Gavins Point will cycle 6,000 cfs every other day from May 5 until June 1<sup>st</sup> then will return to a steady release for the month of June. The steady release rate is determined from the “Gavins Point Releases Needed to Meet Target Flows” section of Plate 3 in the AOP, which is summarized in Table 4-11. The idea is to release a rate sufficient to meet target requirements during the driest part of the summer. This

prevents the endangered birds from nesting on sandbars early in the season that would get inundated later when Gavins Point releases are increased to meet flow targets.

**Table 4-11: Gavins Point releases needed to meet target flows from Plate 3 in the AOP.**

1950 to 1996 Data (kcfs)								
	Median, Upper Quartile, Upper Decile Runoff							
	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov
Full Service	26.7	28.0	27.9	31.6	33.2	32.6	32.0	31.1
Minimum Service	20.7	22.0	21.9	25.6	27.2	26.6	26.0	25.1
	Lower Quartile, Lower Decile Runoff							
	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov
Full Service	29.8	31.3	31.2	34.3	34.0	33.5	33.1	31.2
Minimum Service	23.8	25.3	25.2	28.3	28.0	27.5	27.1	25.2

Intermittent service level checks are performed weekly between May 1 and July 1 when in storage evacuation service level. This allows the model to better adjust releases during flooding. For example, the forecast text file may anticipate a certain flood evacuation service level on May 1, but large rainfall events in the upcoming weeks would necessitate greater releases.

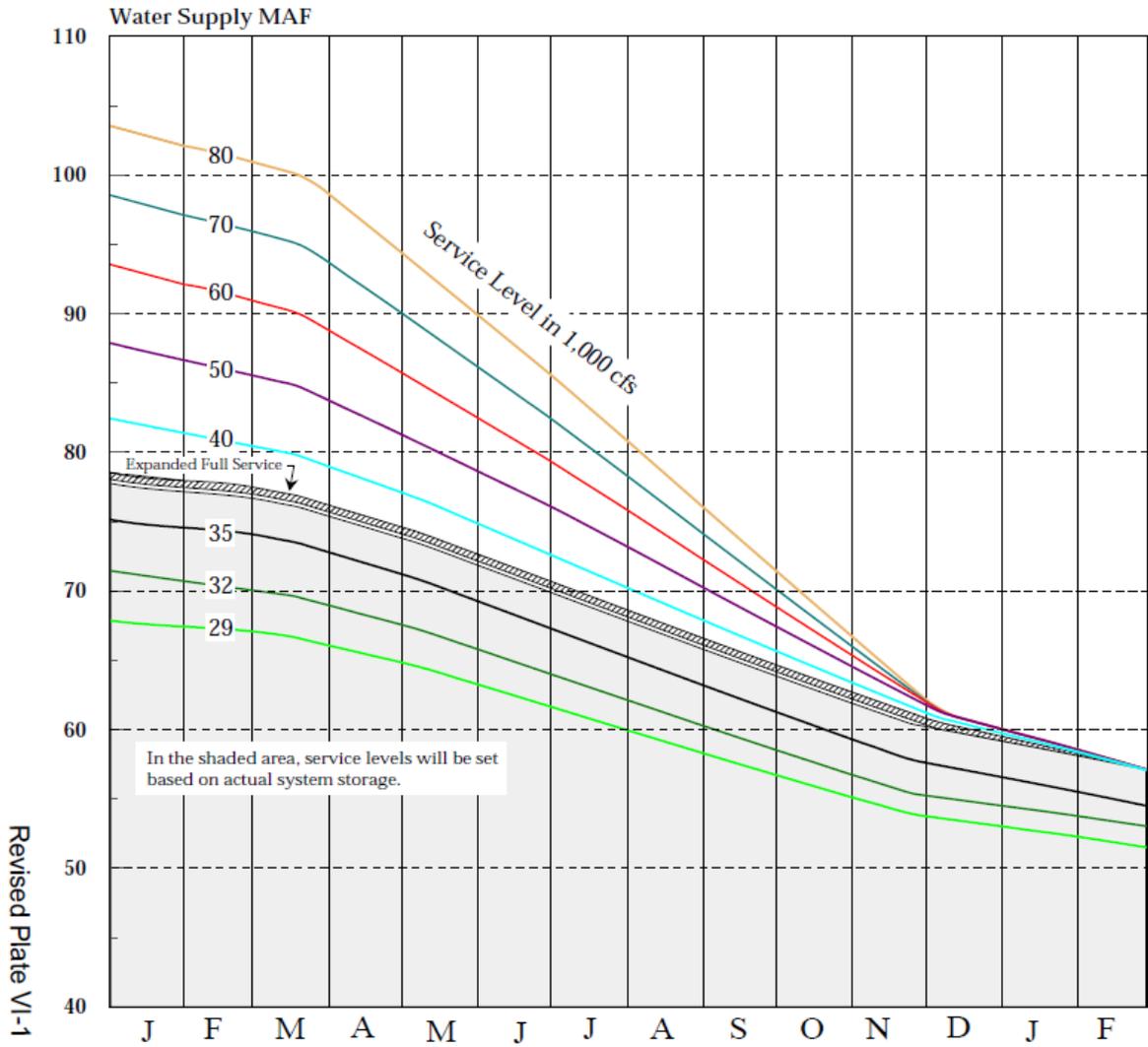
The second of three scheduled assessments is performed on July 1. Like the March 15 assessment, service level for the remaining navigation season is determined using Figure 4-2. Also, based on current system storage, the navigation season length is determined at this time. A full season would imply a closure date of December 1 near the mouth at St. Louis.

The final system assessment is performed September 1. At this time, it is decided based on current system storage whether a 10 day extension to the navigation season is warranted (postponing closure date to December 10). This is not specified in the Master Manual, but has been done in practice in the past. This is done to evacuate more storage from the reservoirs prior to the main ice season. Also on September 1, the current system storage is used to determine the average winter releases from Gavins Point.

Table 4-12 below summarizes the state variables produced in the Service Level state variable.

**Table 4-12: Service Level Script Output State Variables.**

<b>State Variables</b>	<b>Brief Description</b>
ServiceLevel	Determined from Plate VI-1 in the Master Manual. Sets the minimum flow targets at SUX, OMA, NCNE, and MKC during the navigation season. Value in cfs.
GAPTSteadyRelease	When running the SR-FTT scenario without cycling, this is the desired release out of Gavins Point from May 5 to July 15.
GAPTCycledRelease	When running the SR-FTT scenario with cycling, this is the desired release out of Gavins Point from May 5 to July 15.
GAPTWinterRelease	Determined from the Sept. 1 assessment, this is the desired release out of Gavins Point from Dec. 1 (or Dec. 10 during an extended season) to Mar. 1
NavigationEndDate	The last day for navigation season in a given year at the Mouth.
MainstemStorage	Total storage from six mainstem reservoirs.
FcstRunoff	Forecasted remainder calendar year runoff. Combines forecast from given year text file and historic runoff.
WaterSupply	Summation of FcstRunoff and MainstemStorage. Used to determine ServiceLevel.



**Notes:**

1. Water supply consists of the accumulation of the following:
  - a. Actual system storage
  - b. Forecast remaining calendar year runoff volume (1949 basin development level) above Gavins Point Dam.
  - c. Departure of total tributary storage from base level. (See text.)
  
2. Expanded full service consists of the following:
  - a. Maintenance of 35,000 cfs service level through the navigation season.
  - b. Extension of the navigation season for up to 10 days beyond the normal closing date of 1 December at the mouth of the Missouri River.
  - c. Winter releases averaging 20,000 cfs from Gavins Point.
  
3. The relationship between the service level and target flow is as given in the table below:

**Target Flows - 1,000 cfs**

Service Level	Sioux City & Omaha	Nebraska City	Kansas City
29.0 <sub>1/</sub>	25.0	31.0	35.0
35.0 <sub>2/</sub>	31.0	37.0	41.0
40.0 <sub>3/</sub>	36.0	42.0	46.0
50.0 <sub>3/</sub>	46.0	52.0	56.0

<sub>1/</sub> Minimum service level  
<sub>2/</sub> Full service level  
<sub>3/</sub> Storage evacuation service level

**Missouri River  
 Mainstem Reservoir System  
 Service Level**  
U. S. ARMY ENGINEER DIVISION, NORTHWESTERN  
 CORPS OF ENGINEERS, OMAHA, NEBRASKA  
 MARCH 2006

**Figure 4-2: Service Level Determination Chart (from Plate VI-1 in Master Manual).**

#### 4.3.4.4.2 *Detailed Description*

(Refer to Appendix E – Service Level State Variable for actual script)

The Service Level state variable sets values of state variables which can then be used by rules and if-then-else blocks to specify flow constraints. It is initiated in the most upstream reservoir, Fort Peck, through a dummy rule which calls upon the state variable, “ServiceLevel”, as a minimum flow rule (actually sets minimum flow to zero, no matter what the service level is). It is initiated here so that all output state variables written by the script are available to all reservoirs since ResSim computes from upstream to downstream.

The state variable script option in ResSim is divided into three parts: the initialization section, the main computation section, and the cleanup section. The initialization section is only computed once at the beginning of the ResSim simulation, and should contain as much of the jython code as possible to reduce simulation run time. This is where new functions and variables are often defined. The main section of the state variable script is called or computed at every iteration of each time step during the simulation. The cleanup section is only called or computed once at the end of the simulation, and is typically where jython code for user-defined post-processing outputs (graphs, tables, etc) is located.

The state variable script to determine navigation service level in ResSim produces numerous state variables as output. These state variables are often used in other rules in ResSim to control reservoir operations. The state variables produced by the service level jython script are: GAPTWinterRelease, MainstemStorage, NavigationEndDate, ServiceLevel, FcstRunoff, WaterSupply, GAPSTeadyRelease, and GAPTCycledRelease. Most of these state variables are self explanatory, but each is explained in further detail in the following paragraphs.

#### Initialization

The initialization section of the service level script defines various variables and functions and brings in external data which will later be used in the main computation section. The initialization begins by importing various modules needed throughout the script. Next, numerous variables are defined such as the forecast months, minimum and maximum service level values, Julian dates, assessment dates, quantiles and reservoir project names. Then historical runoff data from historic quantile runoff analysis (used in AOP studies by MRBWM) is read in as a string and reformatted into a dictionary with lists of monthly runoff at each project for different percentiles. The service level is then set to “No\_Service” before the service level information from the Master Manual, Revised Plate VI-1 shown in Figure 4-2, is read into ResSim. An error message is returned if any pathnames are incorrect in the DSS file or data is missing. Next, the DRM forecast runoff information (in zipped text files) is read in for the simulation duration and processed (unzipped, formatted, checked for missing years, etc). If no error messages are generated due to missing years of data, the initialization section of the script moves on to process the forecast runoff into totals for the forecast month through July for each project. A sample DRM forecast text file is shown in Figure 4-3.

**Figure 4-3: Sample DRM forecast text file.**

FORECAST		1967 KA-F					
FC_Mth	Month	FTPK	GARR	OAHE	FTRA	GAPT	SUX
Jan	01-1967	323	236	22	28	86	28
Jan	02-1967	361	328	72	54	97	68
Jan	03-1967	566	922	466	210	138	253
Jan	04-1967	656	1162	492	155	124	249
Jan	05-1967	1121	1292	300	169	167	219
Jan	06-1967	1968	3206	469	147	152	241
Jan	07-1967	815	1934	173	30	121	145
Feb	02-1967	386	309	75	45	91	78
Feb	03-1967	765	906	448	189	148	251
Feb	04-1967	663	1210	415	142	149	248
Feb	05-1967	1076	1265	321	148	167	220
Feb	06-1967	1919	3295	448	148	154	230
Feb	07-1967	825	1982	155	39	123	148
Mar	03-1967	597	911	488	196	152	235
Mar	04-1967	651	1127	454	140	147	303
Mar	05-1967	1054	1178	319	152	171	218
Mar	06-1967	1917	3133	472	135	152	237
Mar	07-1967	879	1871	158	39	122	149
Apr	04-1967	655	1271	445	143	130	215
Apr	05-1967	1012	1272	310	150	157	204
Apr	06-1967	2033	3203	453	141	163	233
Apr	07-1967	820	2039	158	34	126	144
May	05-1967	1036	1403	296	150	138	182
May	06-1967	1995	3638	449	126	158	211
May	07-1967	826	2225	153	39	122	135
Jun	06-1967	2225	3736	501	250	103	175
Jun	07-1967	782	2251	158	49	113	121
Jul	07-1967	1260	2801	257	129	155	175

The following summary of the forecast runoff text files used in the script was provided by NWD MRBWM:

“The long-range runoff forecast is presented as the Calendar Year Runoff forecast. This forecast is developed shortly after the beginning of each calendar year and is updated at the beginning of each month to show the actual runoff for historic months of that year and updated forecasts for the remaining months of the year. This forecast presents monthly inflows in MAF from five incremental drainage areas, as defined by the individual System projects, plus the incremental drainage area between Gavins Point Dam and Sioux City. Due to their close proximity, the Big Bend and Fort Randall drainage areas are combined. Summations are provided for the total Missouri River reach above Gavins Point Dam and for the total Missouri River reach above Sioux City. The runoff forecast is adjusted as data becomes available to the common level of basin development, which has been selected as 1949. The 1949 development year is the most recent year that is not affected, to a great extent, by water resource development in the Missouri River basin. By adjusting runoffs to this common level of development, a consistent historical runoff data set has been created by river reach.”

“The forecast of monthly inflows accounts for the three primary runoff components - mountain snowpack, plains snowpack and rainfall. The MRBWM has developed an analytical technique to forecast snowpack runoff from mountainous regions. The mountain snowpack runoff is captured by Fort Peck and Garrison during the May-June-July period. Snow accumulated over the plains area is frequently a major contributor to System inflows

during March and April. To date, few reliable procedures for making accurate quantitative volume runoff forecasts for plains snowmelt are available that consider basin soil conditions during successive wet and/or dry years. However, the MRBWM is actively working with HEC and CRREL, as well as other governmental entities, to improve existing plains snowmelt techniques. Runoff from rainfall events, which mostly occurs between March and October, is particularly difficult to determine more than a few days in advance. The MRBWM utilizes long-term 3-month precipitation outlooks from the NOAA CPC as well as drought monitor maps and various soil moisture models to qualitatively factor those into the long-term runoff forecast.”

“Records of runoff forecasts were available from 1971 to present. At the time of development, 1971-1997 records were used to correlate runoff forecasts to actual runoff. The correlation coefficients were applied to historic runoff values to obtain forecast files for years prior to 1971.”

Next, the script moves on to define several functions for later use in the main computation section of the script. There are functions used to get forecasted system runoff (historicQuantilesTotal and getForecast). First, the runoff forecasts are classified into the appropriate historic quantiles. Once the forecast quantile has been determined, the historic August to December runoff for the same quantile is added to get an expected remainder calendar year runoff for each project. The project totals are summed to determine the total system forecast runoff, “totalFcstRunoff”.

Two utility functions, getInterp and julianDay, are then defined to be able to interpolate between values and to retrieve a Julian day based on a calendar day, respectively.

Next, a very important function, assessSystemState, is defined which is used to obtain the service level at desired dates, determine the season length, and determine the winter release from Gavins Point. The function first verifies if it is an assessment date in the simulation period. If it is an assessment date, numerous variables (such as ServiceLevel, NavSeasonEnd, CurStor, etc) are initialized. The evacuation threshold is also checked. The script then performs the assessment appropriate for the date and sets the service level. The February 1 assessment date is not currently set to return a service level, but can be changed in the future if this assessment date is desired for use. If the assessment date is July 1, the navigation season end date is determined based on the current storage at the time of the assessment. Last, on September 1, the winter releases at Gavins Point (gaptAvg) are set based on the current storage.

The next defined function within the initialization section is the plotStateVariables. This function can create a graph which plots the GAPTWinterRelease, MainstemStorage, NavigationEndDate, ServiceLevel, FcstRunoff, and WaterSupply state variables for the entire simulation duration.

The last part of the initialization section stores all the information needed for use later in the script into a dictionary for quick reference.

## Main Computation

The main computation section of the script is computed for each iteration of each day in the ResSim Simulation. When it runs, it begins by grabbing all the information stored in the dictionary in the initialization section of the script. It then computes the system storage. Next the navigation season end date and winter releases are unset on March 1. The script then calls on the large function, `assessSystemState` which sets service level at indicated dates, determines the season length based on the July 1 assessment, determines if a 10 day extension is warranted, and sets Gavins Point winter release. Additional state variables are defined on March 15<sup>th</sup> which set Gavins Point Releases for the SR-FTT scenario for both cycled and non cycled releases (`GAPTCycledRelease` and `GAPTSteadyRelease`). While looping through every time step in the simulation, the winter releases and navigation season end dates are un-set after they end each season, and the service level is reset to `No_Service` after the navigation season end date.

## Cleanup

The final section of the script, the State Variable Cleanup Section, contains only a few lines of code. This script calls the script from the initialization section to plot the state variables. This script is currently commented out, since the ResSim model has a separate version of this script stored elsewhere. The current ResSim model has buttons in the panel to plot the State Variables and the Downstream Control Points. Additional plot scripts can be added to the panel by going to Tools – Script Editor, from the Simulation Module after a specific simulation has been opened.

### **4.3.4.1 Service Level x1k State Variable**

The Service Level x1k state variable is responsible for setting navigation target discharges based on the service level computed in the Service Level state variable. This state variable first checks if there will be a navigation season based on the service level calculated in the Service Level state variable. If the service level is undefined on March 15 (no navigation season due to low System storage), the `DryYear` state variable is set to 1.0; if the service level has a value on March 15 (there will be a navigation season), the `DryYear` state variable is set to 0.0.

Next the Service Level x1k state variable sets the navigation target discharges using the criteria in Table 4-13. The target discharges for Sioux City, Omaha, Nebraska City, and Kansas City are saved to `SL_SUX`, `SL_OMA`, `SL_NCNE`, and `SL_MKC` slave state variables, respectively. These slave state variables are then used in other scripted rules to ensure navigation requirements are met. The target discharges are only written to the slave state variables during the navigation season at each target location, which are summarized in Table 4-14. The navigation end dates are based on the `NavigationEndDate` state variable computed in the Service Level state variable.

**Table 4-13: Navigation target discharges related to service level. Summarized from Table VII-1 in the Master Manual (U.S. Army Corps of Engineers, 2006).**

Target Location	Flow Target Discharge Deviation from Service Level
Sioux City	- 4,000 cfs
Omaha	- 4,000 cfs
Nebraska City	+ 2,000 cfs
Kansas City	+ 6,000 cfs

**Table 4-14: Navigation season at each target location. Summarized from Section 7-03.4.1 in the Master Manual.**

Target Location	Opening Date	Closing Date
Sioux City	March 23	Nav End Date – 9 days (November 22 <sup>**</sup> )
Omaha	March 25	Nav End Date – 7 days (November 24 <sup>**</sup> )
Nebraska City*	March 26	Nav End Date – 6 days (November 25 <sup>**</sup> )
Kansas City	March 28	Nav End Date – 4 days (November 27 <sup>**</sup> )
Mouth	April 1	Nav End Date (December 1 <sup>**</sup> )

\*There is no navigation start or end dates specified in the Master Manual for Nebraska City. For modeling purposes, they were assumed to be 1 day after Omaha's start and end dates.

\*\*Example dates listed are for a normal 8-month navigation season.

#### **4.3.4.2 Spring Pulse Scripted Rule**

The Spring Pulse scripted rule is used to simulate the bimodal spring pulse during March and May, which is used to benefit the pallid sturgeon. The first part of the spring pulse script checks the system storage on March 1 and May 1. If the system storage is less than 40.0 MAF on March 1, the March pulse is cancelled. If there is enough storage on March 1 for the March pulse to occur, the script begins checking Gavins Point releases when releases are increased for the navigation season. The March pulse is initiated when downstream locations first meet their flow targets for the navigation season and Gavins Point releases stop increasing. The maximum March pulse is 5,000 cfs minus inflow from the James River. The inflow from the James River is determined by lagging the discharge measured at Scotland, SD by 1 day. Once the maximum pulse is determined, it is added to the normal Gavins Point release. The March pulse release is held constant for 2 days and then releases are reduced 1,000 cfs per day until Gavins Point releases reach required navigation releases.

Two checks occur after the magnitude of the March pulse has been determined. The first check sets the maximum March pulse release to 35,000 cfs if the release would have originally exceeded 35,000 cfs. The second check looks at downstream flood limits. The March pulse releases are routed downstream to Omaha, Nebraska City, and Kansas City. If the March pulse

releases cause flows at OMA to exceed 41,000 cfs, or flows at NCNE exceed 47,000 cfs, or flows at MKC exceed 71,000 cfs, the pulse is reduced by 500 cfs until a pulse magnitude is calculated that no longer exceeds the downstream flood limits or the pulse is canceled because any pulse will exceed the flood limits.

On May 1, the system storage is checked again and if the system storage is below 40.0 MAF, the May pulse is cancelled. If there is sufficient system storage for the May pulse, the pulse is initiated on May 1. The maximum May pulse is first prorated by the system storage; the maximum May pulse is set to 16,000 cfs if the system storage is greater than or equal to 54.5 MAF. If the system storage is less than 54.5 MAF, the maximum May pulse is linearly interpolated to 12,000 cfs when the system storage is 40.0 MAF. The maximum May pulse is prorated a second time based on forecasted runoff. If the remaining forecasted annual runoff is a median runoff, no change is made to the first prorated maximum May pulse. An additional 4,000 cfs can be added to first prorated maximum May pulse if the remaining forecasted runoff is upper quartile. An additional 4,000 cfs can be subtracted from the first prorated maximum May pulse if the remaining forecasted runoff is lower quartile.

Once the maximum May pulse has been prorated, the first 3 days of releases for the May pulse, May 1-3, are increased by 1/3 of the maximum May pulse. The May pulse is held at its peak for 2 days, May 3-4, before releases are reduced by 15% of the maximum May pulse for the next 2 days, May 5-6. Over the next 8 days, May 7-14, the remaining 80% of the maximum May pulse is reduced equally per day until releases reach normal navigation releases.

With the maximum May pulse set, the releases are routed downstream and the flood limits are checked at Omaha, Nebraska City, and Kansas City. If flood limits are exceeded at any of those locations, the maximum May pulse is reduced by 500 cfs and the releases pattern is adjusted and the downstream flood limits are checked again. This process is continued until a May pulse that does not induce flooding is reached. If flooding will occur with any magnitude of pulse, the May pulse is cancelled.

#### ***4.3.5 Scripted Rules in the System Model***

##### **4.3.5.1 GAPT Control Scripted Rule**

The GAPT Control scripted rule is a multi-faceted script that determines the releases for Oahe, Big Bend, and Fort Randall. The first step in the script is to set Fort Randall's guide curve during the fall. Fort Randall's pool begins to drawdown on Sep 1 from 1355.0 ft to 1337.5 ft on the the final day of the navigation season. The scripted rule linearly interpolates the guide curve elevations and overwrite the current guide elevation values which are based on a navigation end date of December 1.

The scripted rule then checks Fort Randall's pool elevation. If Fort Randall's pool is greater than the top of the flood and multiple use zone and Gavins Point's release is less than 35,000 cfs, Gavins Point's release is increased by 15,000 cfs. By increasing Gavins Point's releases, Fort Randalls' releases will increase because more water is required to keep Gavins Point at its guide curve.

Fort Randall's releases are calculated through an iterative process of setting three days of releases, routing the releases to Gavins Point, adding the local inflows for the same three-day period, subtracting the known Gavins Point releases, and finally computing the storage at the end of the three-day period. The computed storage is compared to the guide curve storage for Gavins Point. If the computed storage is more than 100 acre-ft lower than the guide curve storage, all three days of Fort Randall releases are initially increased by 10,000 cfs until the computed storage is greater than the guide curve storage. Now that the computed storage is greater than the guide curve storage more, all three days of Fort Randall releases are decreased by 1/3 of the previous release change (i.e. releases are decreased by  $10,000 / 3 = 3,333$  cfs) until the computed storage is less than the guide curve storage. This process is repeated until three days of Fort Randall releases produce a Gavins Point storage that is within 100 acre-ft of the guide curve storage. Normally, this process is repeated three days later for the next three days of releases. Three days of releases were used for Fort Randall's releases because there are three routing coefficients for the Fort Randall to Gavins Point reach. If releases are changed every day when using coefficient routing, it is difficult to consistently keep Gavins Point at its guide curve storage because the operational range is small. However, setting releases every three days ensures that Gavins Point will converge to its guide curve storage every three days. During the iterations, Gavins Point's pool elevation is monitored; if Gavins Point's pool exceeds the guide curve by  $\pm 1.0$  ft, Fort Randall's releases are recomputed the following timestep instead of waiting for 3 days to be recomputed. This was done to reduce the large fluctuations in Gavins Point's pool that can occur due to the coefficient routing.

Once Fort Randall's releases have been calculated, the scripted rule calculates Oahe's releases to keep Fort Randall at guide curve. Calculating Oahe's releases was done differently than Fort Randall's because Big Bend is between Oahe and Fort Randall. Oahe's releases are routed to Big Bend and Big Bend's local inflows are added to the routed releases. Since Big Bend is a run of the river project, it is assumed that Big Bend will release inflow. Big Bend's releases are then routed to Fort Randall and Fort Randall's local flows are added to the routed releases. This process is repeated if the error between the computed storage and guide curve storage is greater than 100 acre-ft. Calculation of Oahe's releases occurs every 4 days there are a total of 4 routing coefficients between Oahe and Fort Randall. Oahe's pool elevation is also monitored; if Oahe's pool elevation exceeds 1616.0 ft, flood evacuation releases begin. The flood evacuation releases attempt to keep Oahe's pool elevation below flood constraints while also releasing only through the power house. If higher releases are required to keep Oahe below 1619.5 ft, releases through the flood tunnels occur. Fort Randall's storage will exceed its guide curve storage while Oahe is evacuating flood storage, which is consistent with current operations: Fort Randall's limited flood storage is utilized during extreme events when Oahe's flood storage is nearly full.

Big Bend's releases are computed last since it is run of the river project and does not require iterations. If Big Bend's pool elevation is  $\pm 0.3$  ft from the guide curve elevation, its releases will be equal to the inflow. If Big Bend's pool elevation is greater than 0.3 ft above the guide curve elevation, its releases will increase to lower its pool elevation to the guide curve elevation. If Big Bend's pool elevation is greater than 0.3 ft below the guide curve elevation, its releases will decrease to raise its pool elevation to the guide curve elevation.

After Fort Randall's, Big Bends, and Oahe's releases have been computed for either guide curve operation or flood evacuation, one last check is done for Fort Randall and Gavins Point. It is possible for Fort Randall's pool elevation to reach the top of exclusive control zone during extreme events. If the pool elevation reaches the top of that zone, ResSim will ignore all rules and operate in a "save the dam" manner by releasing inflow. This can cause Gavins Point's pool elevation to rapidly increase or decrease and hit the top of its exclusive flood control zone, which causes large variation in its releases as it switches to a "save the dam" operation. Two additional checks were added to help Fort Randall and Gavins Point avoid hitting the top of their respective exclusive flood control zones. If Fort Randall's pool elevation exceeds 1374.7 ft (0.3 ft below the top of its exclusive flood control zone), it will ignore the releases for Gavins Point's guide curve and release 5,000 cfs more than the inflow. If Gavins Point's pool elevation exceeds 1210.0 ft (2.0 ft below the top of its exclusive flood control zone), it will ignore the releases computed during the Downstream model's simulation and increase its specified release by 1,500 cfs.

All differences between Gavins Point's releases computed in the System model and the Downstream model are recorded during this scripted rule. If there is a difference between the releases, fall and winter releases in the System model will be adjusted to account for the change in volume leaving the system.

#### **4.3.5.2 Reservoir Balancing Scripted Rule**

The reservoir balancing scripted rule sets releases at Fort Peck and Garrison as well as balances the carryover storage for Fort Peck, Garrison, and Oahe. Releases from Fort Peck and Garrison typically follow an annual pattern: high releases during the summer, low releases in the fall, and medium releases in the winter. Figure 4-4 and Figure 4-5 shows the detailed flow patterns for both Fort Peck and Garrison.

On the 1<sup>st</sup> and 15<sup>th</sup> of each month, a forecast is completed for the percentage of total carryover storage that will be utilized at Fort Peck, Garrison, and Oahe on March 1 of the following year based on runoff forecasts for all six reservoirs and known GAPT releases. This forecast may be completed on the 7<sup>th</sup> and 21<sup>st</sup> of each month if pool elevations at Fort Peck, Garrison, or Oahe exceed their flood elevations of 2244.0 ft, 1845.0 ft, and 1616.0 ft, respectively. A target storage is calculated for Fort Peck, Garrison, and Oahe based on the percentage of the system carryover storage (i.e. if the forecasted percentage of total system storage is 90%, the target storage at Fort Peck, Garrison, and Oahe would be equal to 90% of their respective carryover storages.). The release pattern at Fort Peck is adjusted up or down so that Fort Peck reaches its target storage by Mar 1 of the following year. It is possible that Fort Peck will not reach its target because there are minimum and maximum releases at Fort Peck that could prevent Fort Peck's releases from going as high or low as needed to reach its target. Once Fort Peck's releases are set for the remainder of the year, Garrison's release pattern is adjusted up or down so that Garrison reaches its target storage by March 1 of the following year.

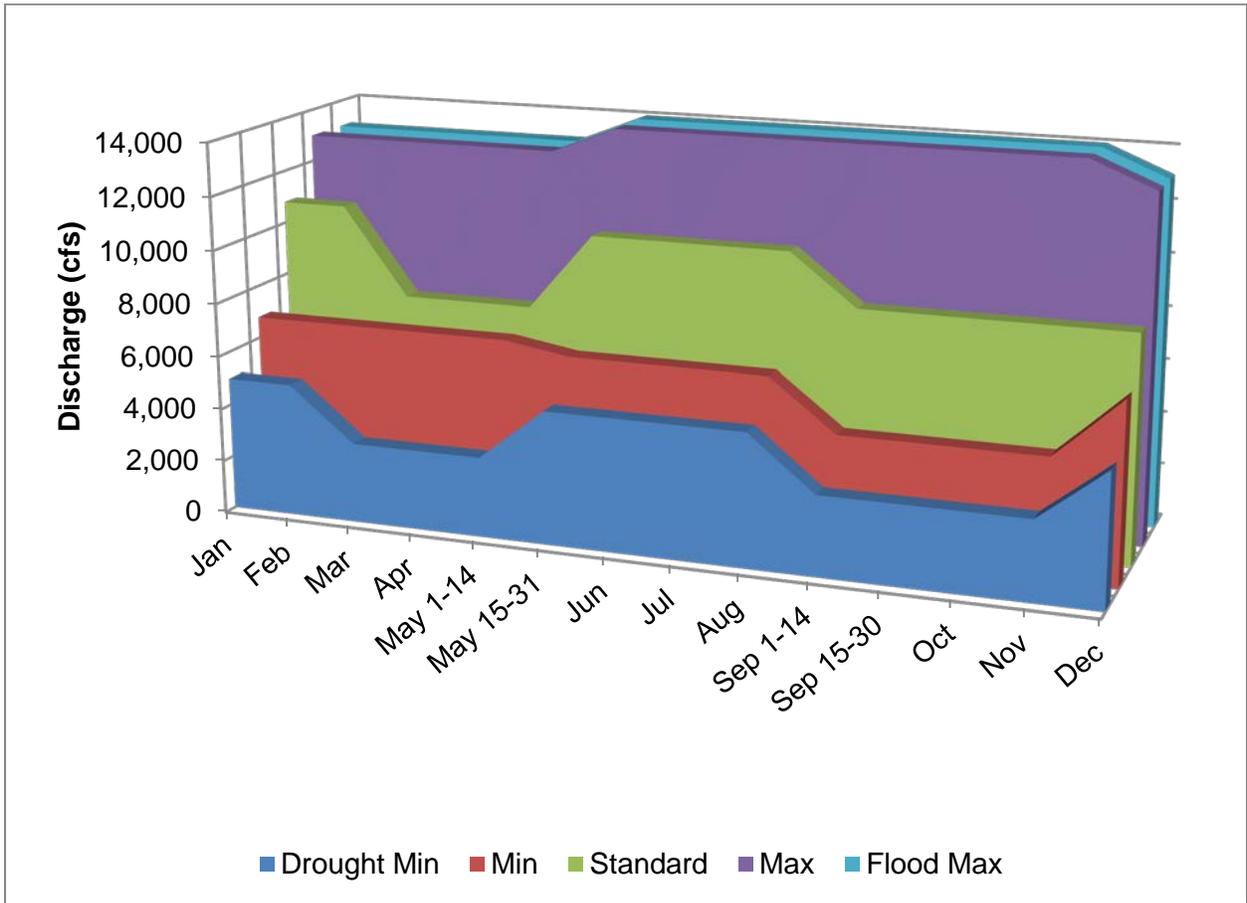
Minimum and maximum releases at Fort Peck and Garrison are scaled depending on the pool elevation to allow for drought conservation releases. At Fort Peck, normal minimum releases when the pool is at the top of the carryover storage zone are linearly interpolated down to the

drought minimum releases when the pool is 35 ft above the permanent pool. Normal maximum releases when the pool is at the top of carryover storage zone are linearly interpolated up to the flood maximum releases when the pool reaches the top of the exclusive flood control zone. At Garrison, normal minimum releases when the pool is at the top of the carryover storage zone are linearly interpolated down to the drought minimum releases when the pool is 10 ft above the permanent pool. Normal maximum releases when the pool is at the top of carryover storage zone are linearly interpolated up to the flood maximum releases when the pool reaches the top of the exclusive flood control zone.

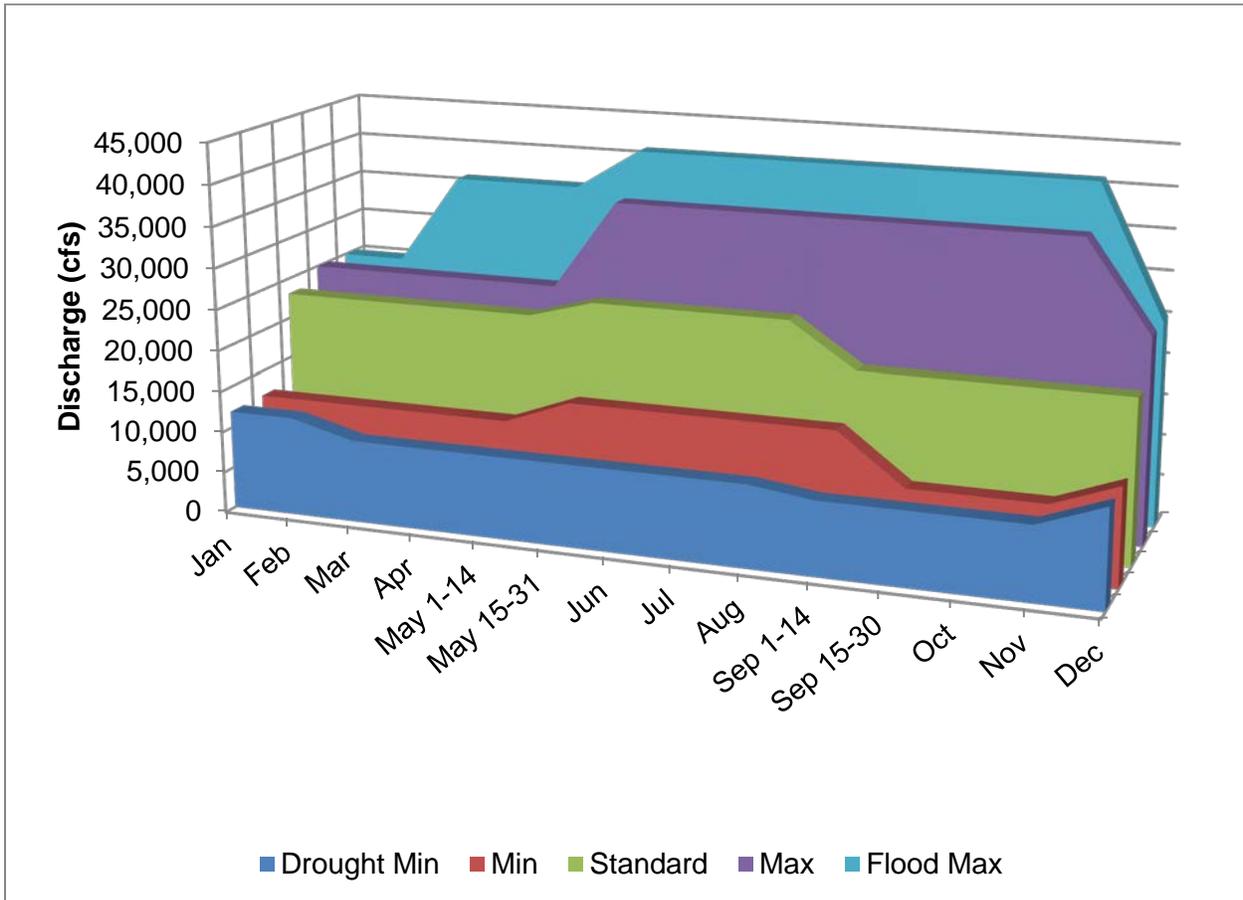
Oahe is allowed to float throughout the year based on Garrison and Gavins Point releases; however, there are several checks to ensure Oahe does not get too unbalanced or its pool elevation does not get too high compared to Garrison. If Oahe's March 1 forecasted storage is greater than its target storage, half of the difference between Oahe's forecasted March 1 storage is added to Garrison's target storage and half is added to Fort Peck's target storage. Garrison's releases will be scaled up or down if the difference between Garrison's and Oahe's current percent of carryover storage is  $\pm 5.0$  percent. Fort Peck's releases will also be scaled up or down if the difference between Fort Peck's and Garrison's current percent of carryover storage is  $\pm 5$  percent.

Adjustments are also made to Fort Peck's and Garrison's releases for tern and plover operations. Summer releases are typically held constant if there is no immediate concerns with the upper three reservoirs. Garrison's releases are held constant from May 15 – Sep 15 if Garrison's pool elevation is between 1775.0 ft and 1850.0 ft and Oahe's pool elevation is greater than 1545.0 ft. Fort Peck's releases are held constant from May 15 – Sep 15 if Fort Peck's pool elevation is between 2195.0 ft and 2246.0 ft and Garrison's pool elevation is greater than 1775.0 ft.

Figure 4-4: Plot of release schedules for FTPK used in the reservoir balancing script.



**Figure 4-5: Plot of release schedules for GARR used in the reservoir balancing script.**



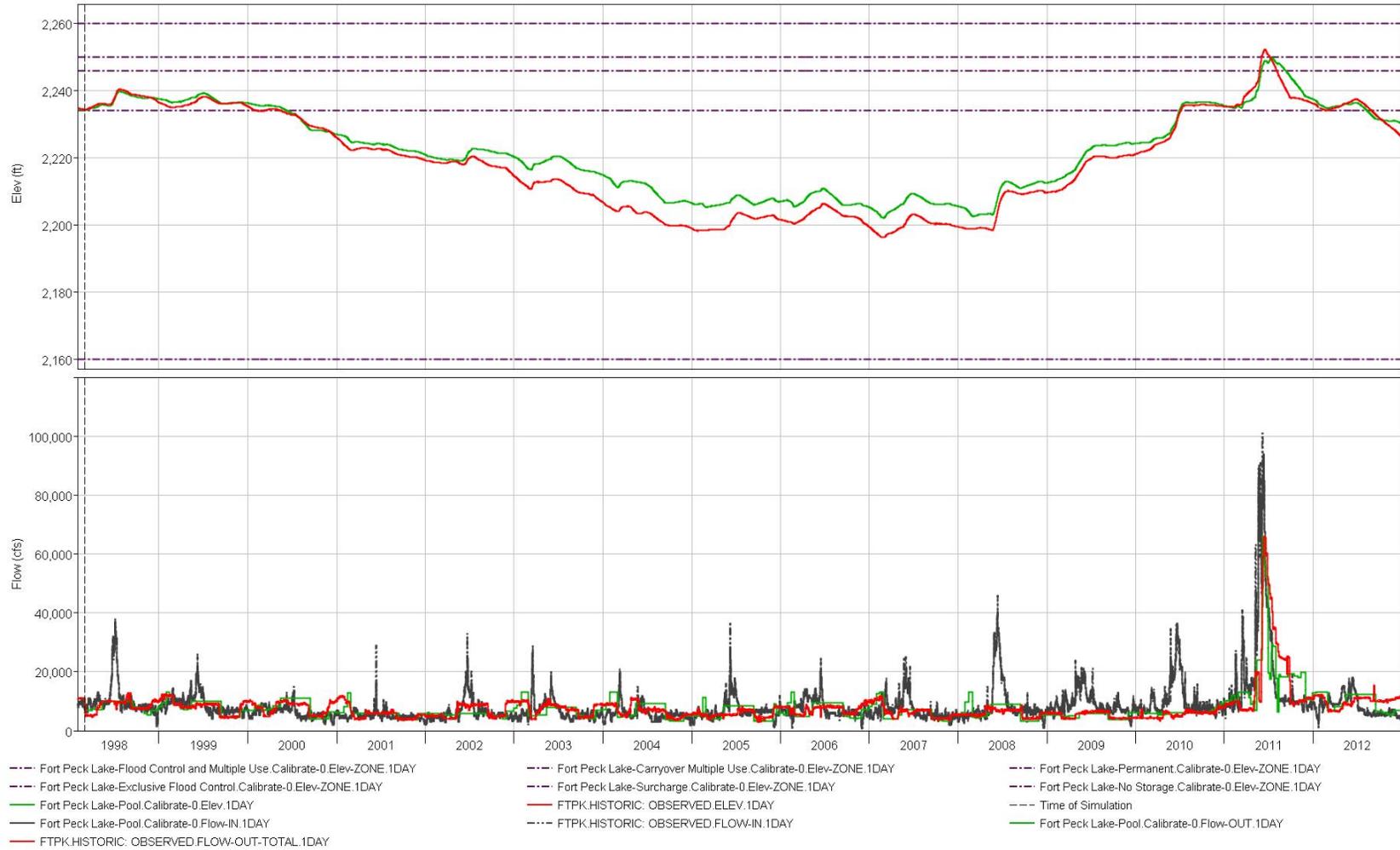
#### 4.3.5.3 Water Supply Scripted Rule

The upstream water supply scripted rule uses the releases at Fort Peck and Garrison that are calculated in the reservoir balancing scripted rule and checks downstream water supply requirements. Fort Peck's releases are routed downstream to Wolf Point and Culbertson. If 3,000 cfs is not observed at both locations during March – May 14 and September – November, Fort Peck's releases are increased until that criteria is met. If 5,000 cfs is not observed at both locations during May 15 – August and December – February, Fort Peck's releases are increased until that criteria is met. Then Garrison's releases are routed downstream to Bismarck. If 10,000 cfs is not observed at Bismarck during March – August, Garrison's release are increased until that criteria is met. If 9,000 cfs is not observed at Bismarck during September – November, Garrison's release are increased until that criteria is met. If 12,000 cfs is not observed at Bismarck during December – February, Garrison's release are increased until that criteria is met.

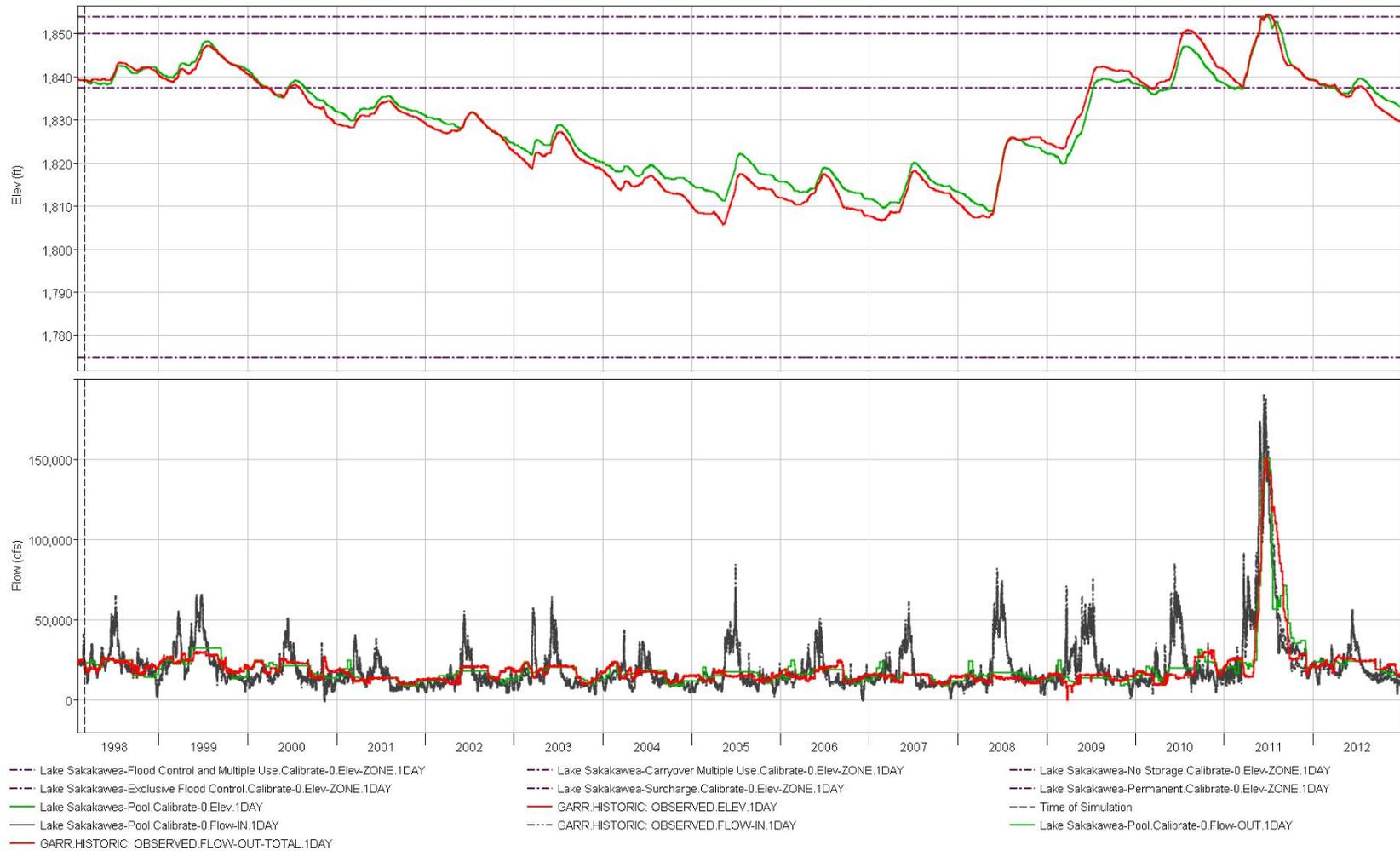
## **5 EVALUATION OF MODEL PERFORMANCE**

### **5.1 CALIBRATION COMPARISON**

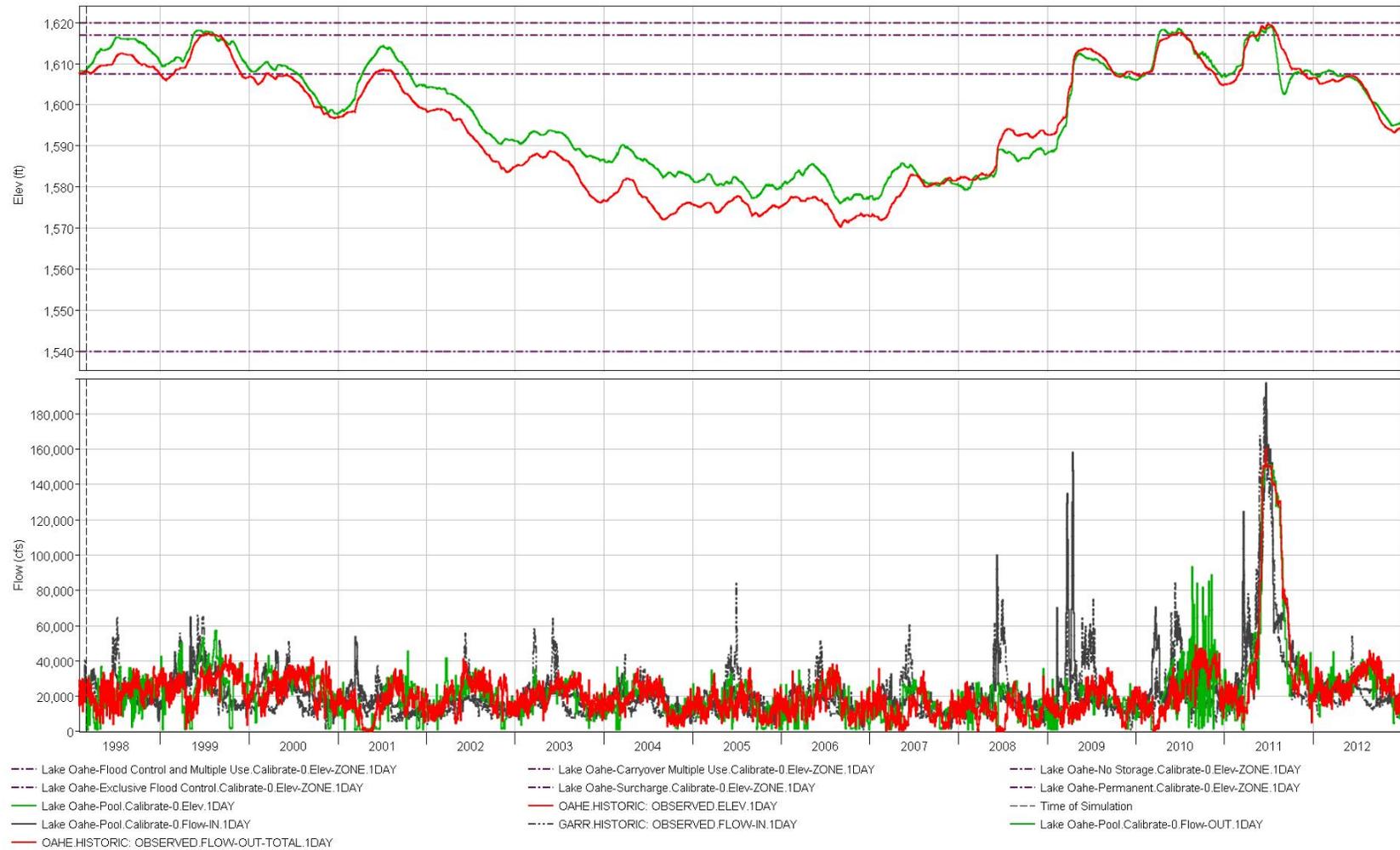
Calibration of the model was based on the period of Mar 1, 1998 – Dec 31, 2012. This period was selected as the calibration period because it contains a large drought period, multiple flood events and occurs in the recent history where water management policies in the observed period closely match what is used in the model. Figure 5-1 through Figure 5-6 display the ResSim operations versus historic operations for the calibration period.



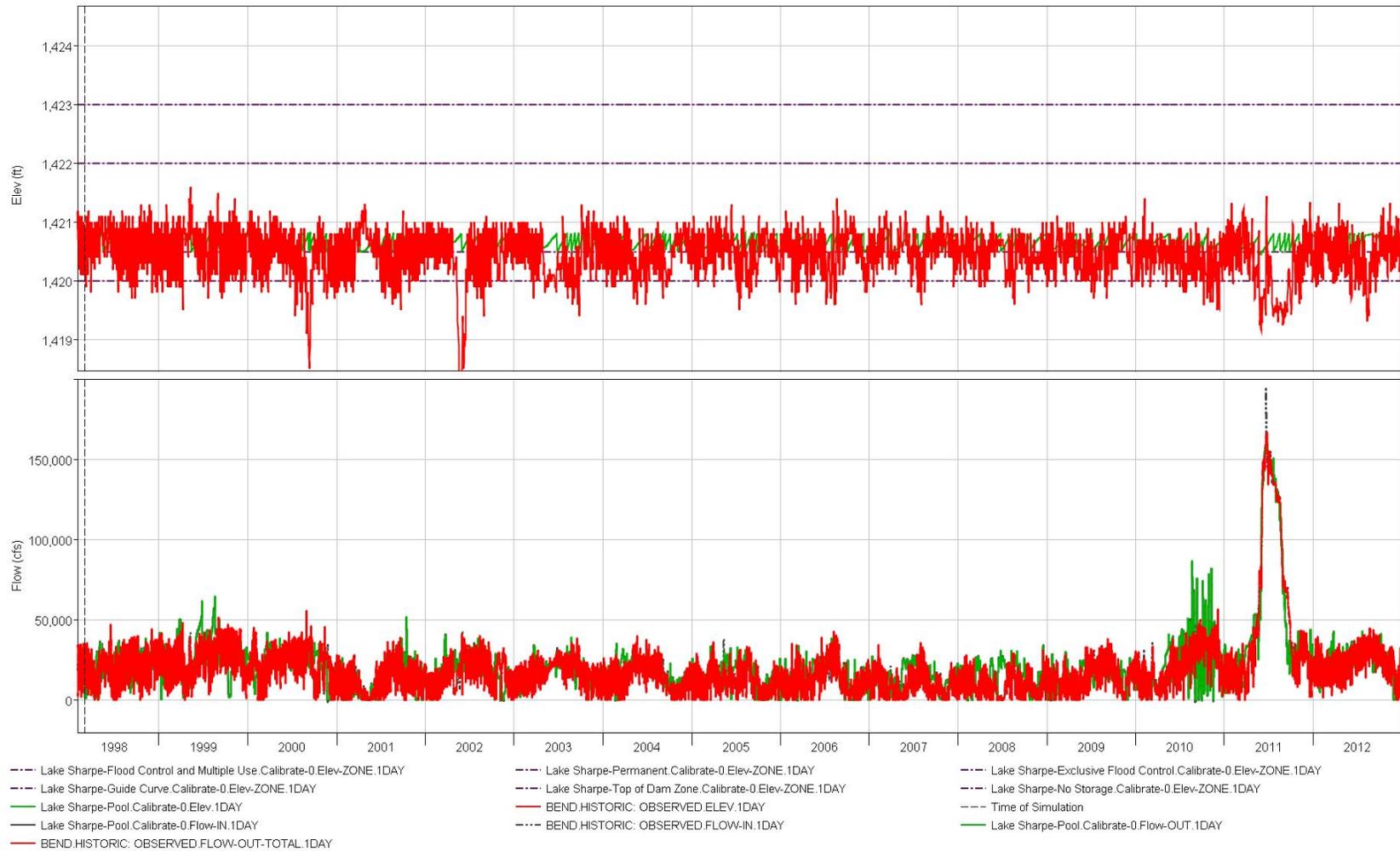
**Figure 5-1: ResSim versus historic operations at Fort Peck Dam 1998-2012.**



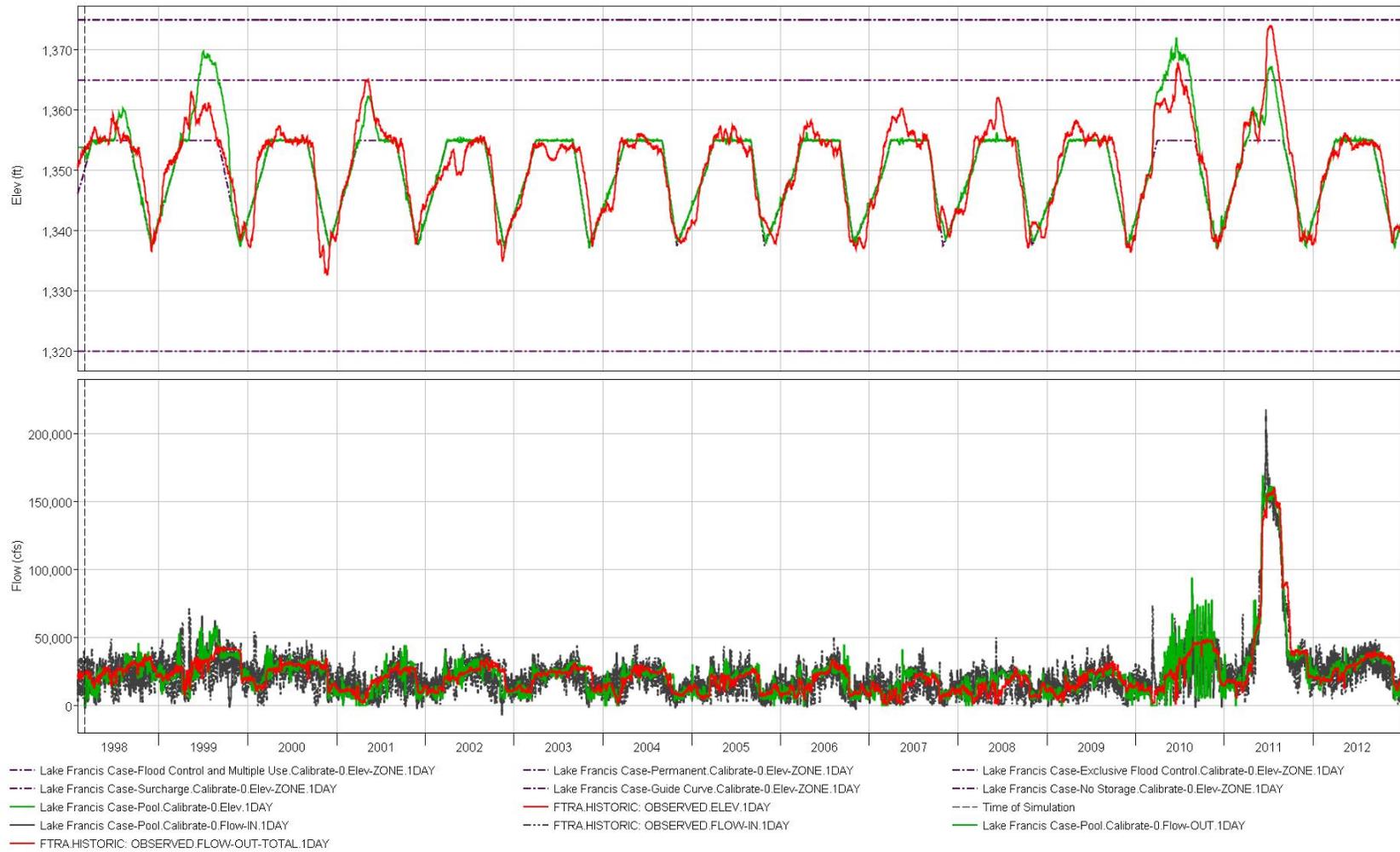
**Figure 5-2: ResSim versus historic operations at Garrison Dam 1998-2012.**



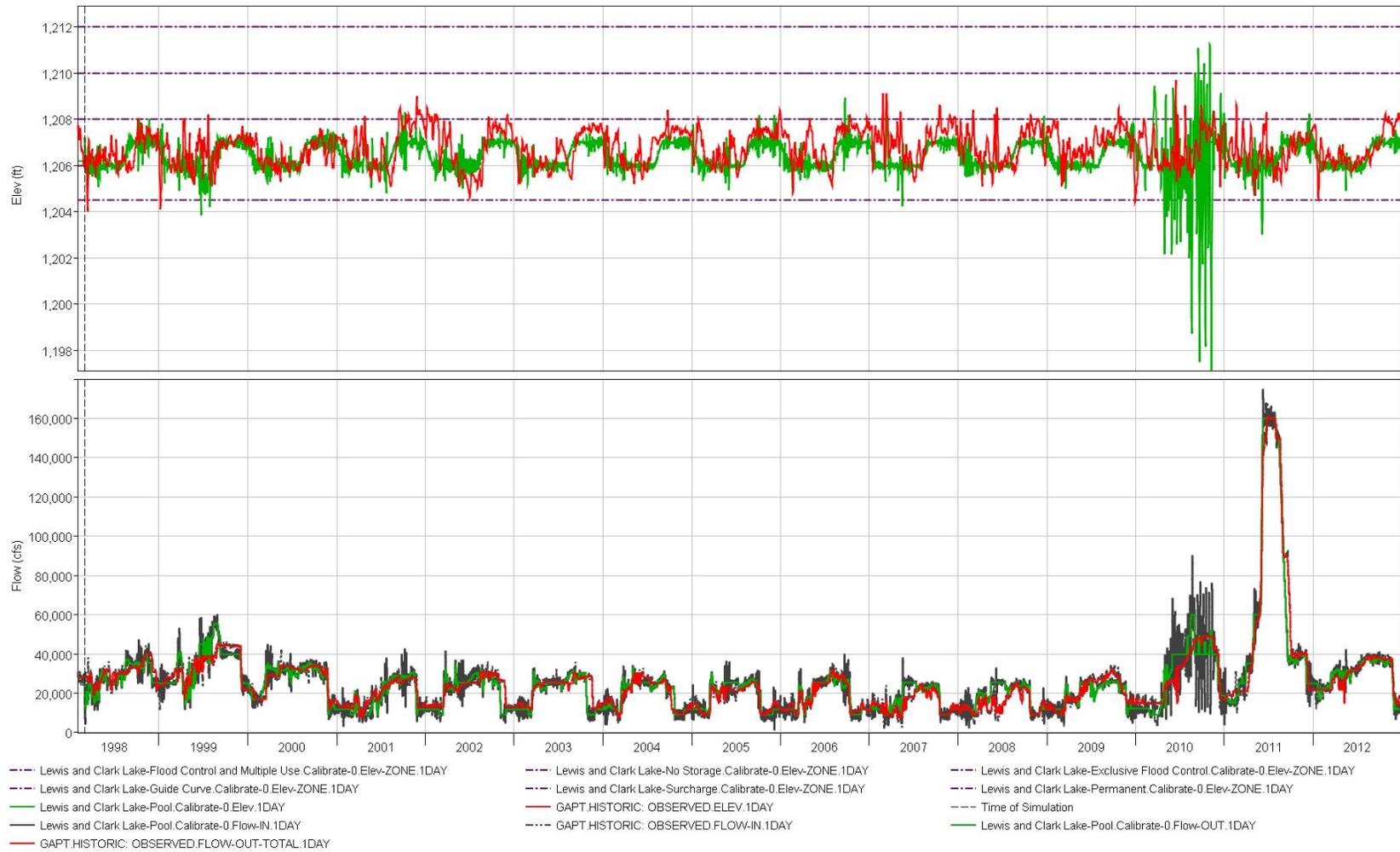
**Figure 5-3: ResSim versus historic operations at Oahe Dam 1998-2012.**



**Figure 5-4: ResSim versus historic operations at Big Bend Dam 1998-2012.**



**Figure 5-5: ResSim versus historic operations at Fort Randall Dam 1998-2012.**



**Figure 5-6: ResSim versus historic operations at Gavins Point Dam 1998-2012.**

The above plots show a variety of information, but the most valuable pieces that explain the system operation are the pool elevations at Fort Peck, Garrison and Oahe and the releases from Gavins Point.

Gavins Point releases follow the general trend of releases to meet navigation and other release targets as well as evacuate excessive floodwaters when necessary. Gavins Point missed navigation targets only when the computed local flow forecasts had significant error. Local flow forecasts were included to make the operations more realistic since errors in downstream forecasting is part of the realtime operations.

The pool elevations at the upper three projects follows the historic trends and maintain the proper balance of storage in all three projects. These elevations also validate Gavins Point's annual release volume, as the bulk of the system inflow originates above the upper three reservoirs.

The objective of this ResSim model is to simulate system operation for the flow of record for assessment of base conditions on the Missouri River. It should be noted that the ResSim model will never fully be able to operate the reservoirs to exactly match historic operations because of changes in operation over time, changes in basin depletions and other water development within the basin, and special short term operations that have deviated from the water control manuals. The end product should be representative to a reasonable degree of historic operations and ensuring that all major operational decisions occur correctly. There are instances where special operations have occurred due to situational occurrences, political requests, environmental, and changes in policy. Such instances that have required deviation from normal operations include but are not limited to:

- Dam maintenance
- Near real-time adjustments to adjust max flows for endangered species
- Navigation flow target adjustments to account for barge traffic (or lack thereof)
- Release adjustments to mitigate ice impacts
- Others

## **5.2 POOL PROBABILITY COMPARISON**

Comparisons of 1998-2012 pool probability plots are shown below; including data from earlier in the period-of-record did not allow for an accurate comparison because system operations have changed throughout the period-of-record. The plots show the observed and modeled pool probability data. The pool elevations at the upper three projects follows the historic trends and maintains the proper balance of storage in all three projects. It should be noted that since only 13 years of data was used to produce these probability plots, the probabilities associated with the pool elevations should only be used for comparison of modeled data to observed data.

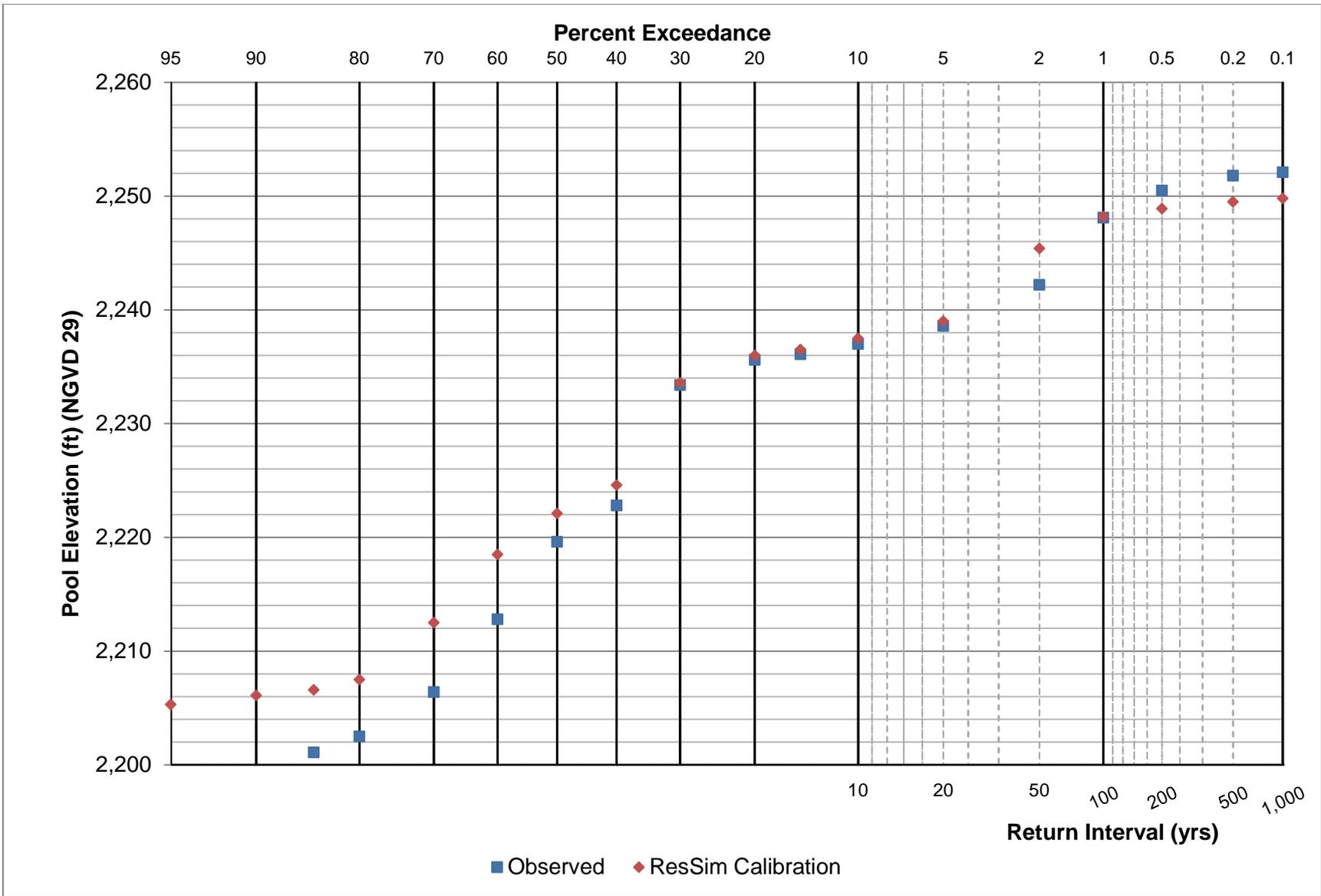


Figure 5-7: Fort Peck Lake pool probability curve for years 1998-2012.

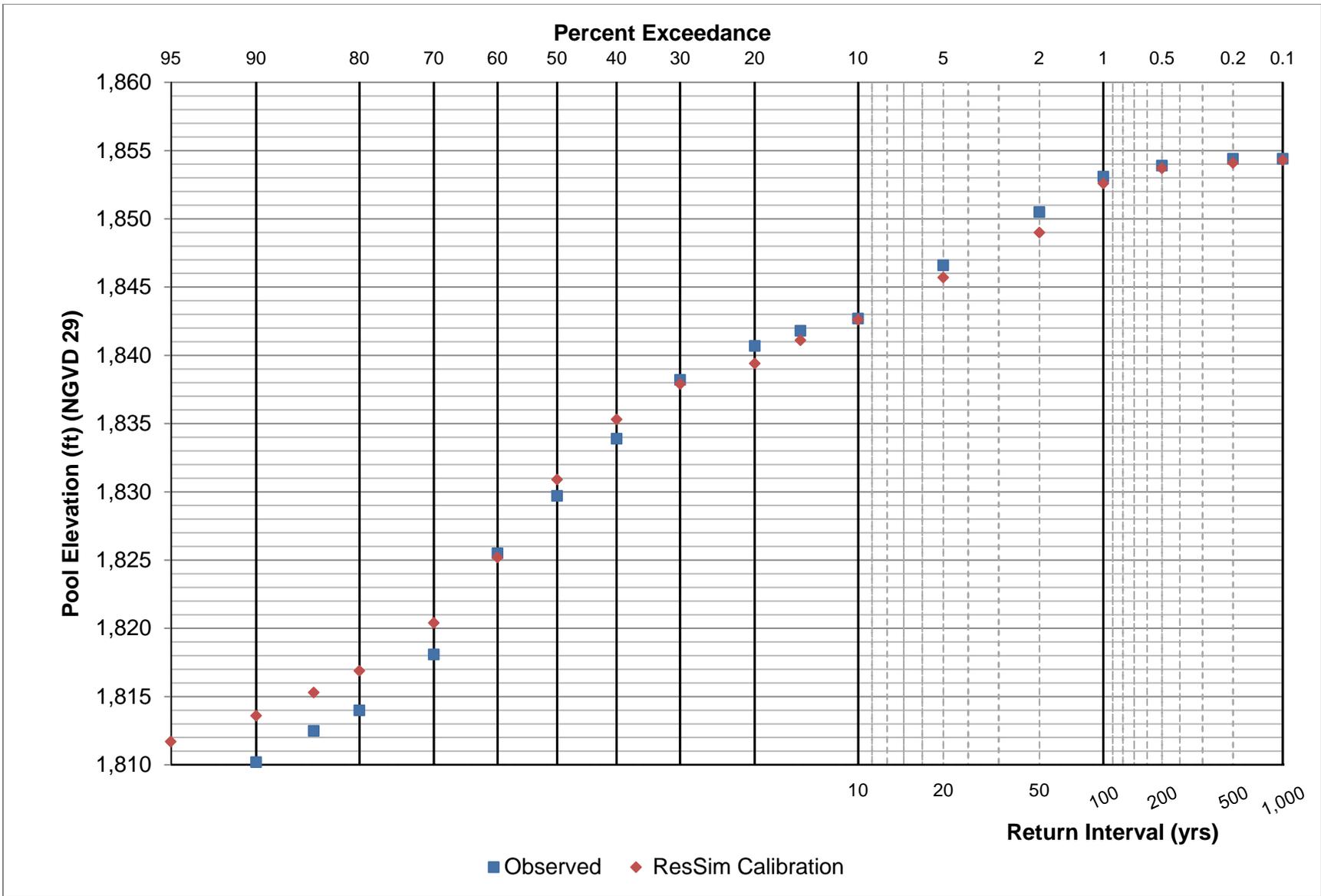


Figure 5-8: Lake Sakakawea pool probability curve for years 1998-2012.

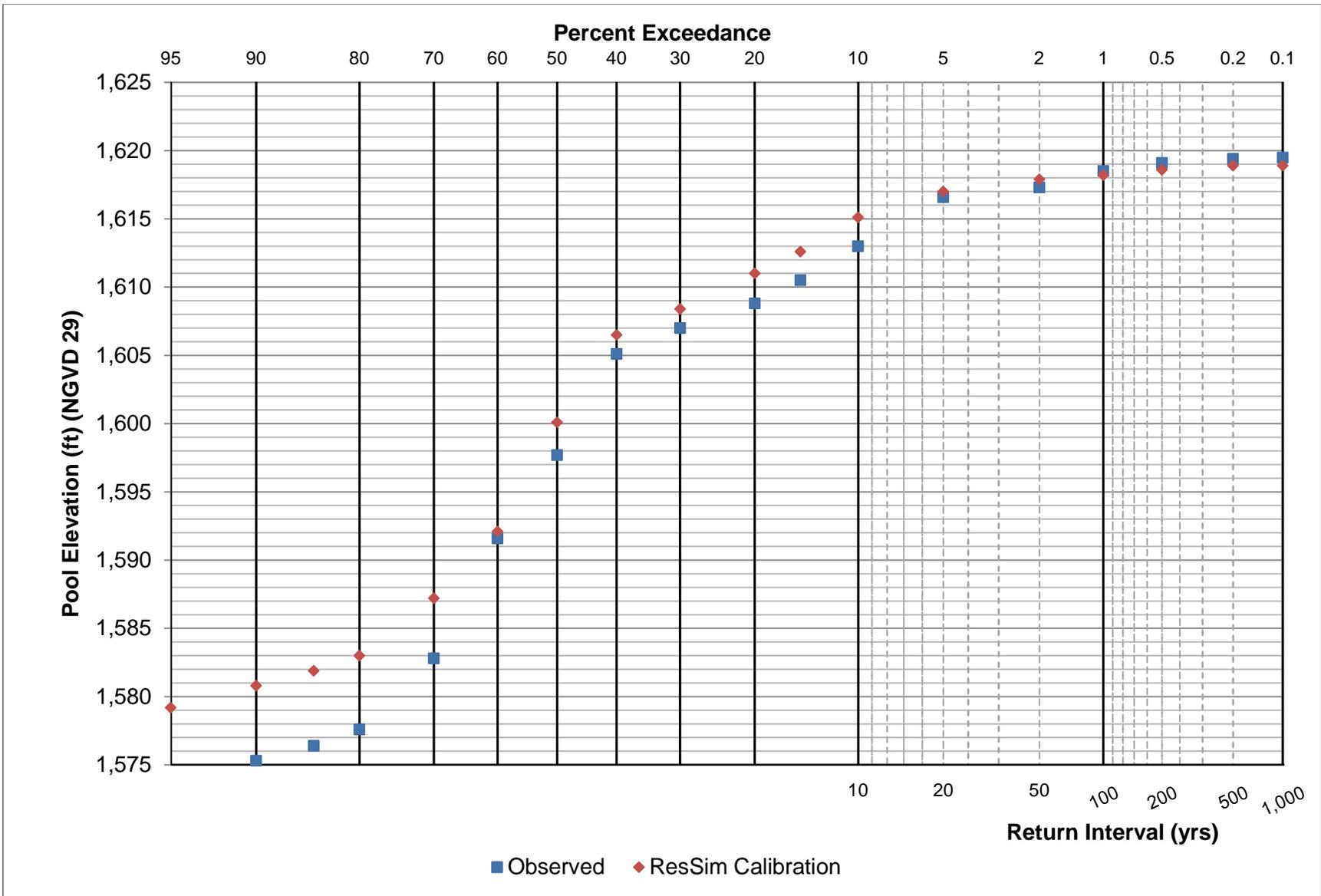


Figure 5-9: Lake Oahe pool probability curves for years 1998-2012.

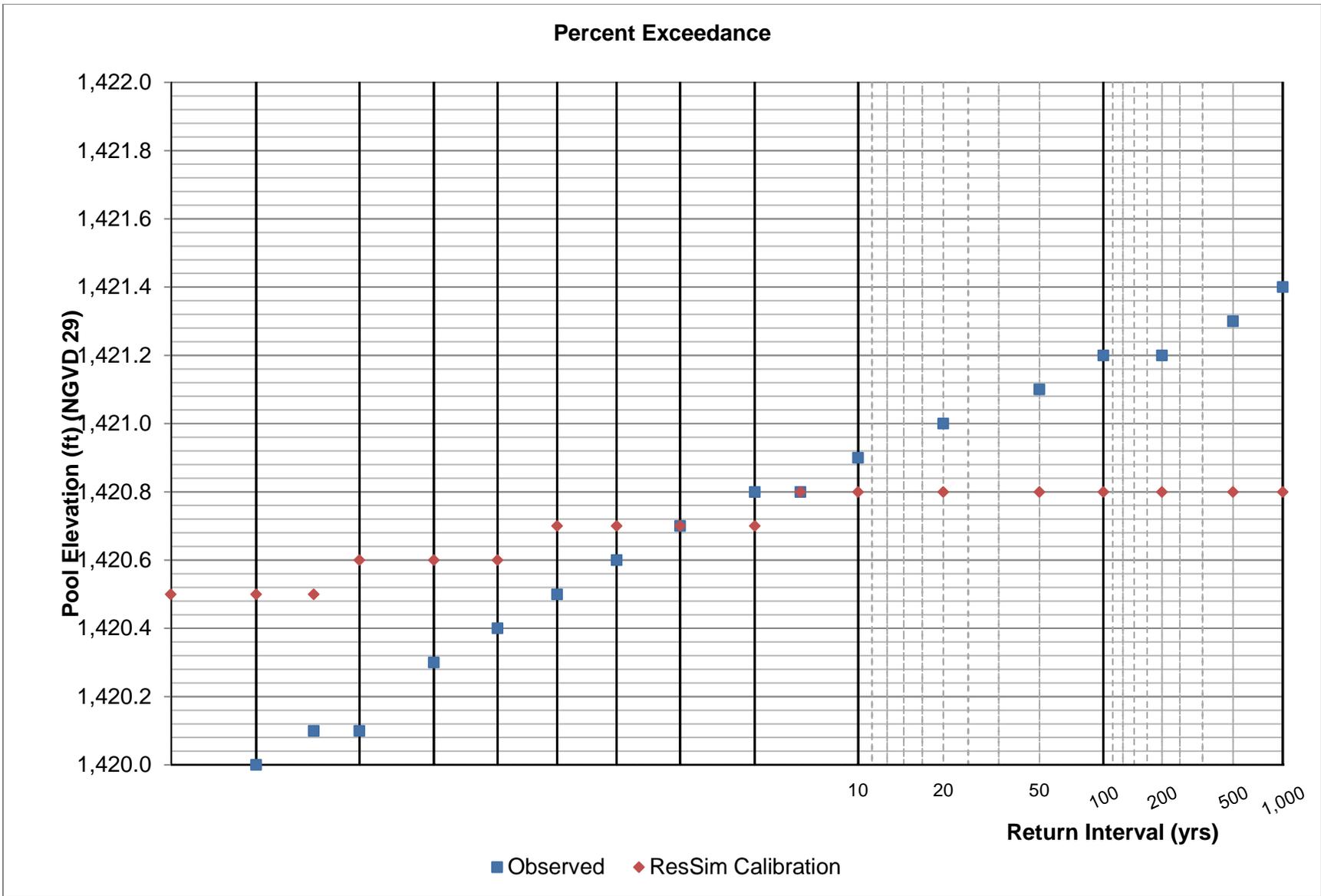


Figure 5-10: Lake Sharpe pool probability curves for years 1998-2012.

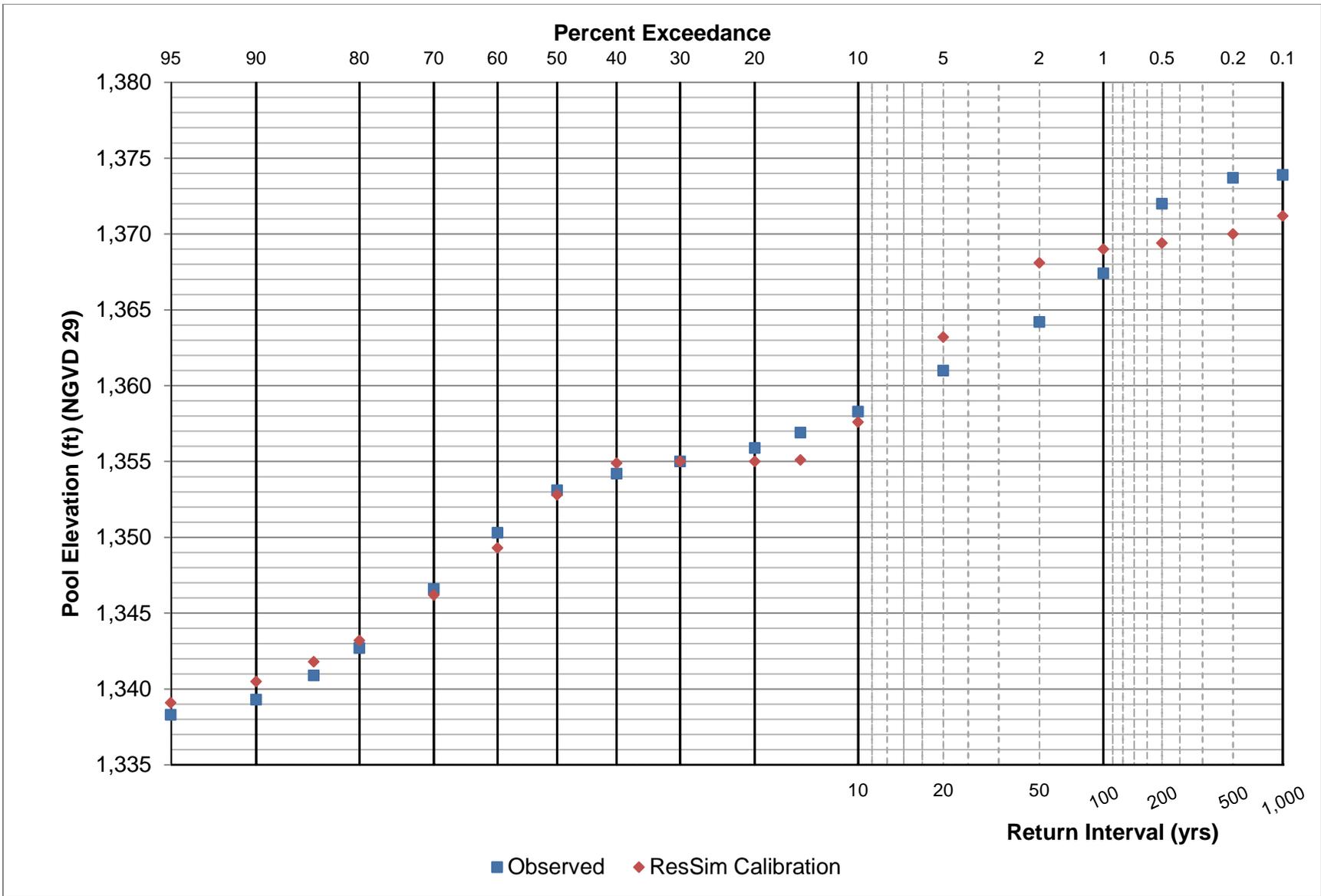


Figure 5-11: Lake Francis Case pool probability curves for years 1998-2012.

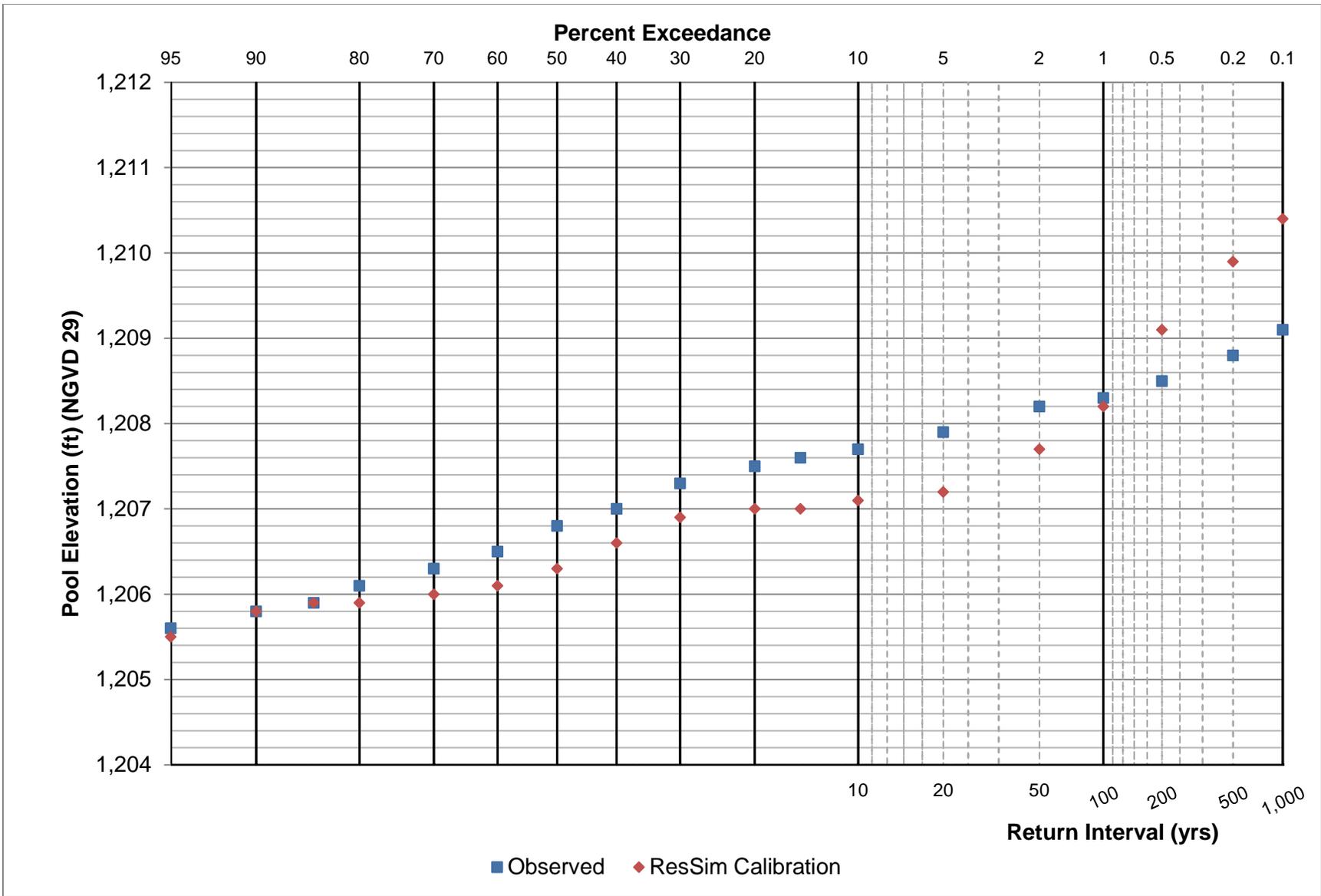


Figure 5-12: Lewis and Clark Lake pool probability curves for years 1998-2012.

### **5.3 RELEASE AND FLOW PROBABILITY COMPARISON**

Comparisons of 1998-2012 release and flow plots are shown below; including data from earlier in the period-of-record did not allow for an accurate comparison because system operations have changed throughout the period-of-record. The plots show the observed and modeled release probability data. The releases at the upper three projects follows the historic trends and maintains the proper balance of storage in all three projects. It should be noted that since only 13 years of data was used to produce these plots, the probabilities associated with the releases should only be used for comparison of modeled data to observed data.

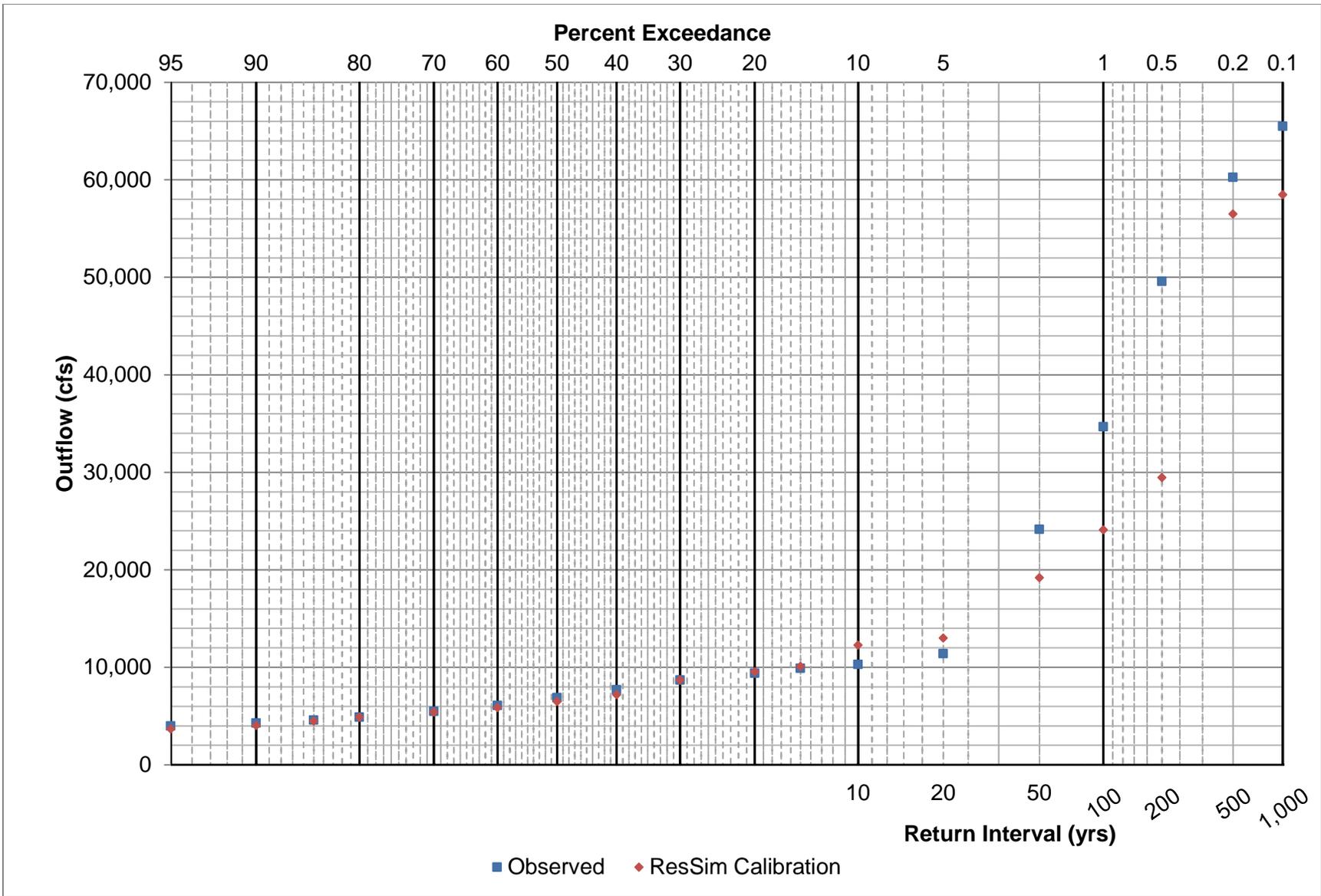


Figure 5-13: Fort Peck Lake release probability curves for years 1998-2012.

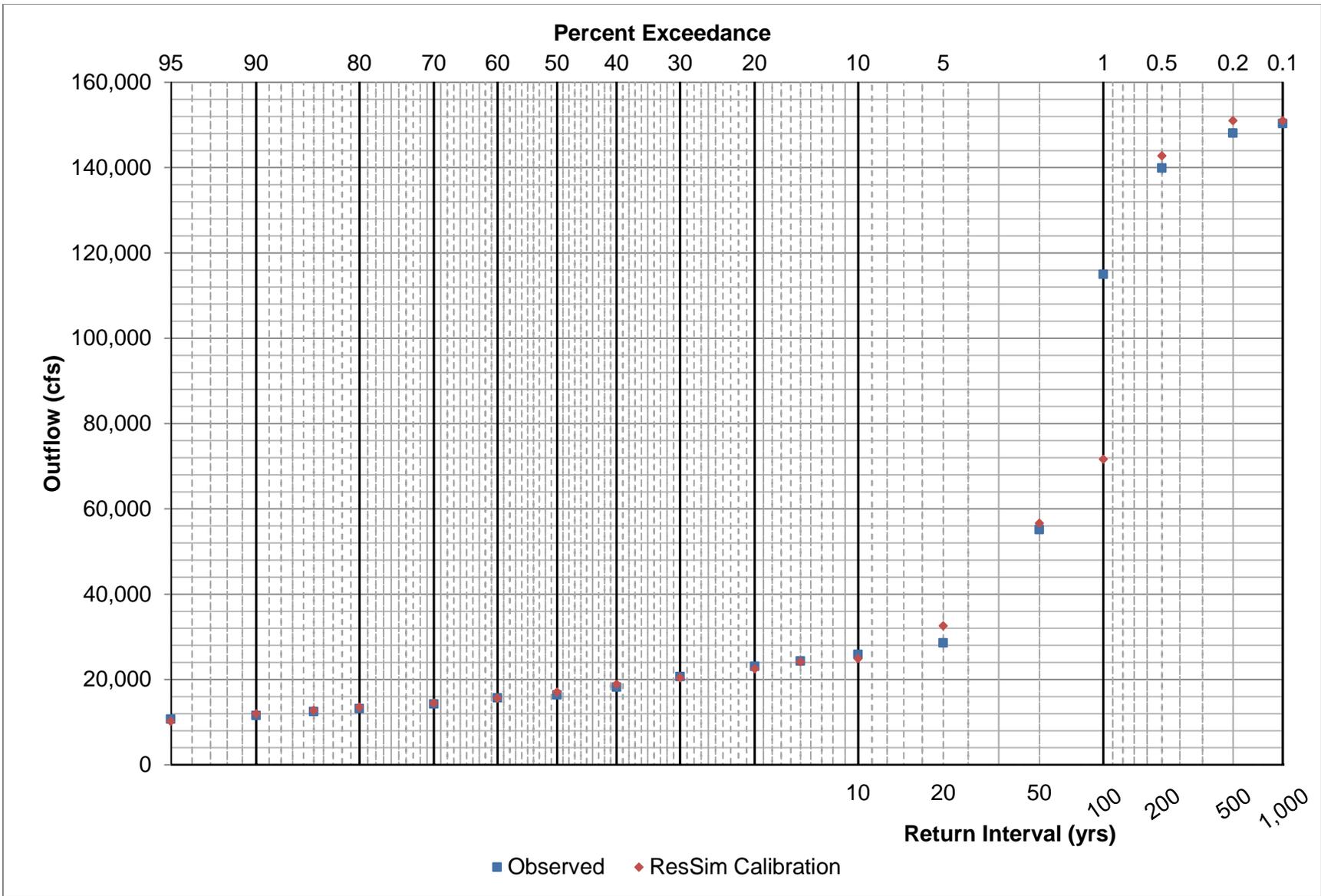


Figure 5-14: Lake Sakakawea release probability curves for years 1998-2012.

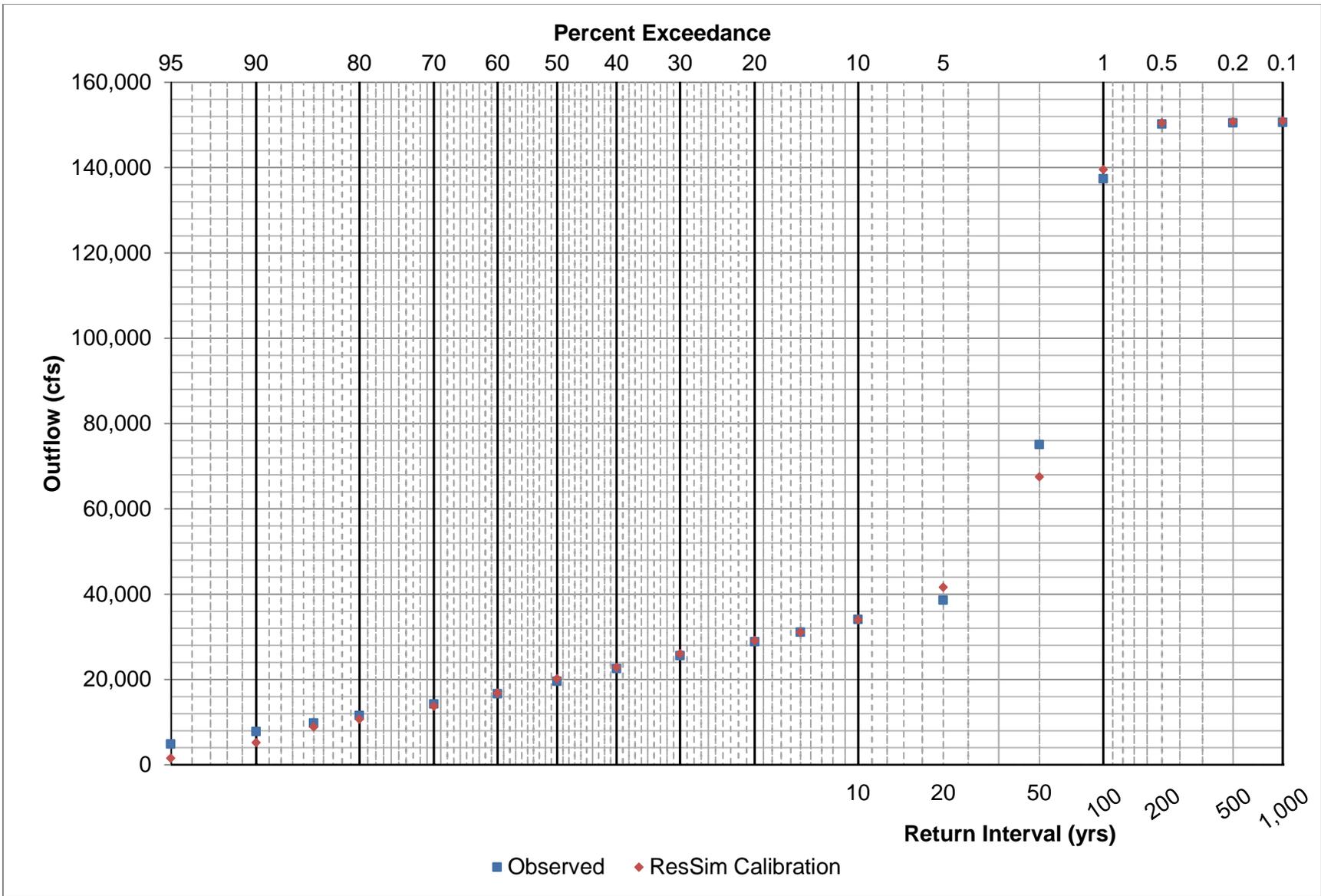
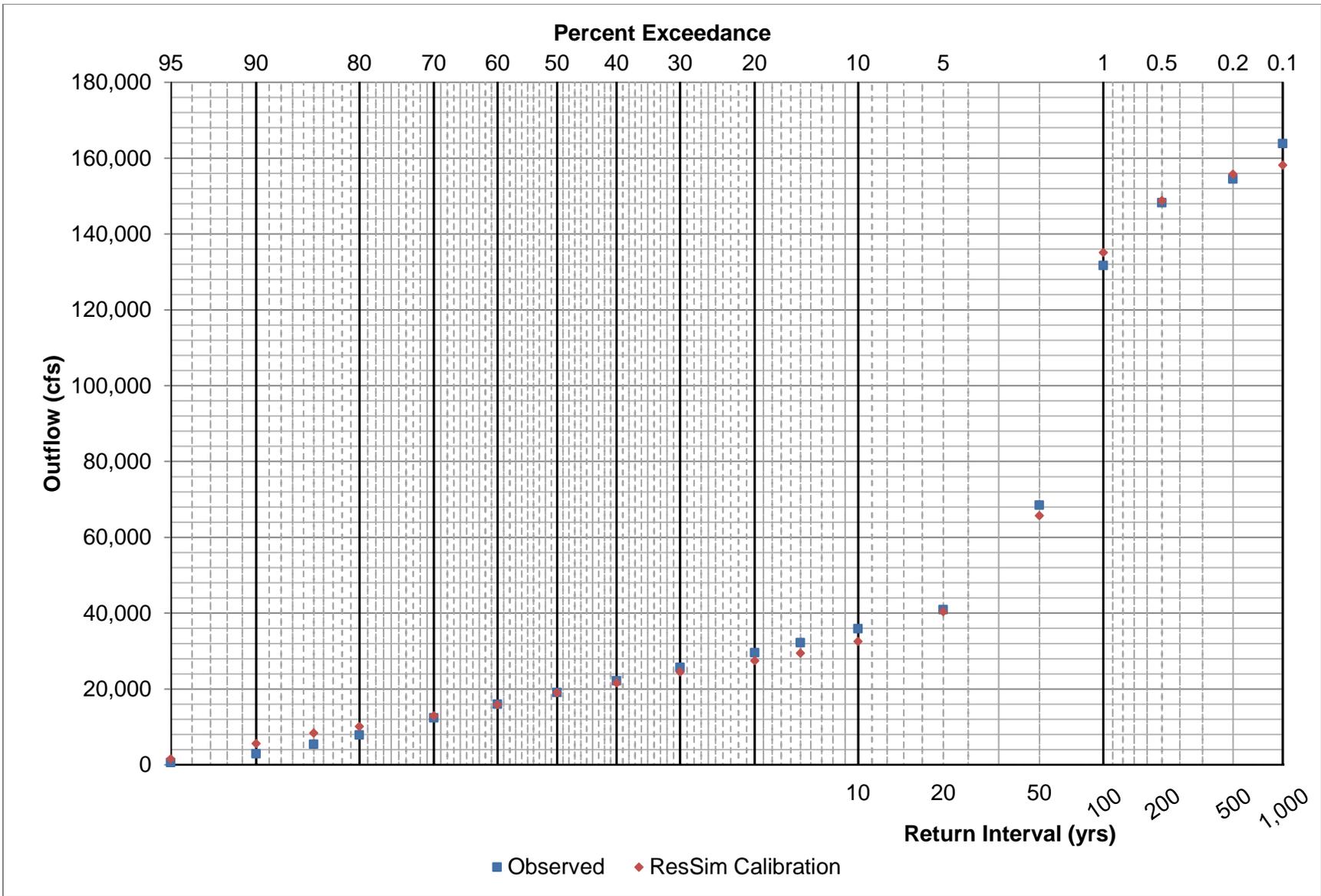


Figure 5-15: Lake Oahe release probability curves for years 1998-2012.



**Figure 5-16: Lake Sharpe release probability curves for years 1998-2012.**

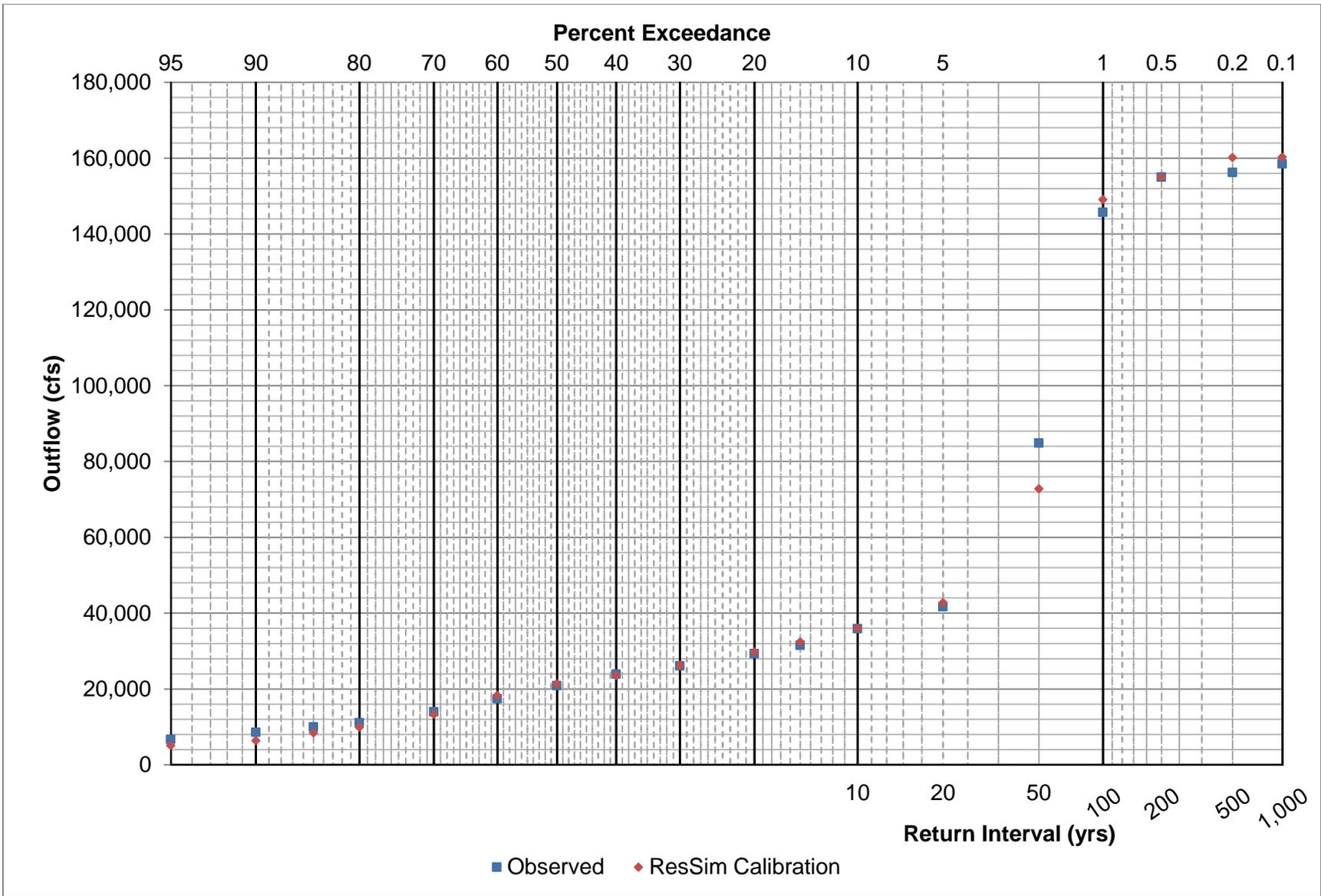


Figure 5-17: Lake Francis Case release probability curves for years 1998-2012.

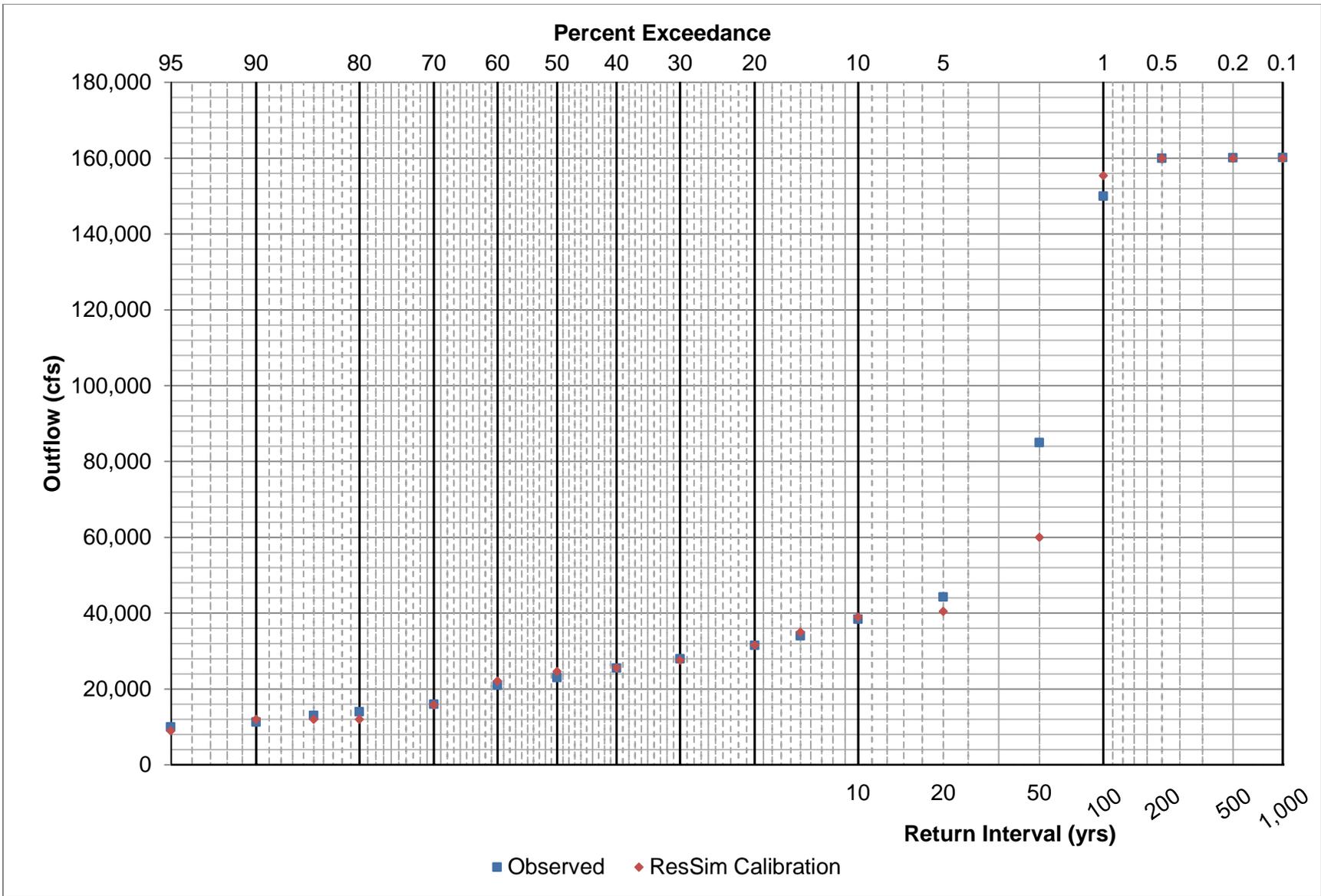


Figure 5-18: Lewis and Clark Lake release probability for years 1998-2012.

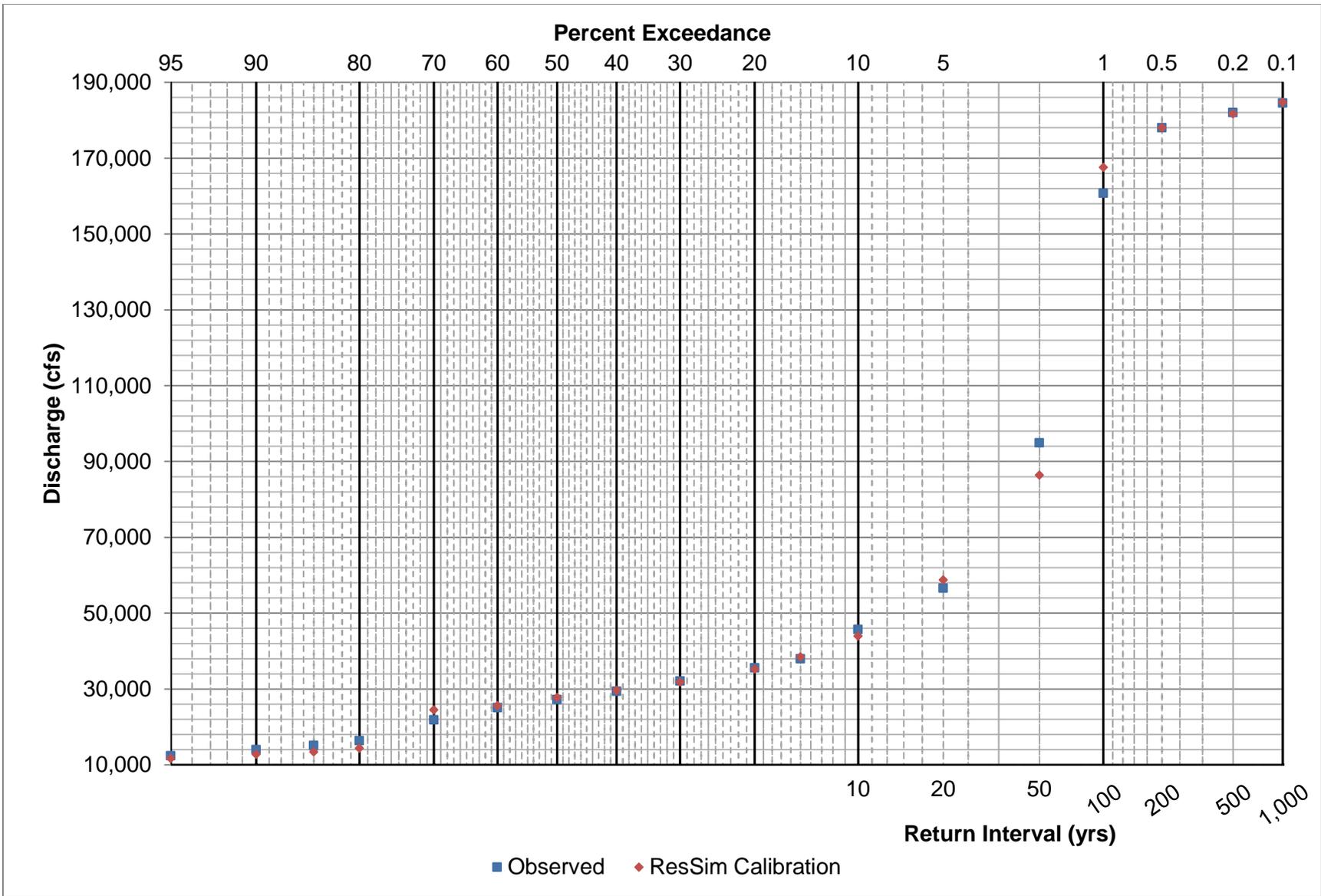


Figure 5-19: Sioux City, IA flow probability curves for years 1998-2012.

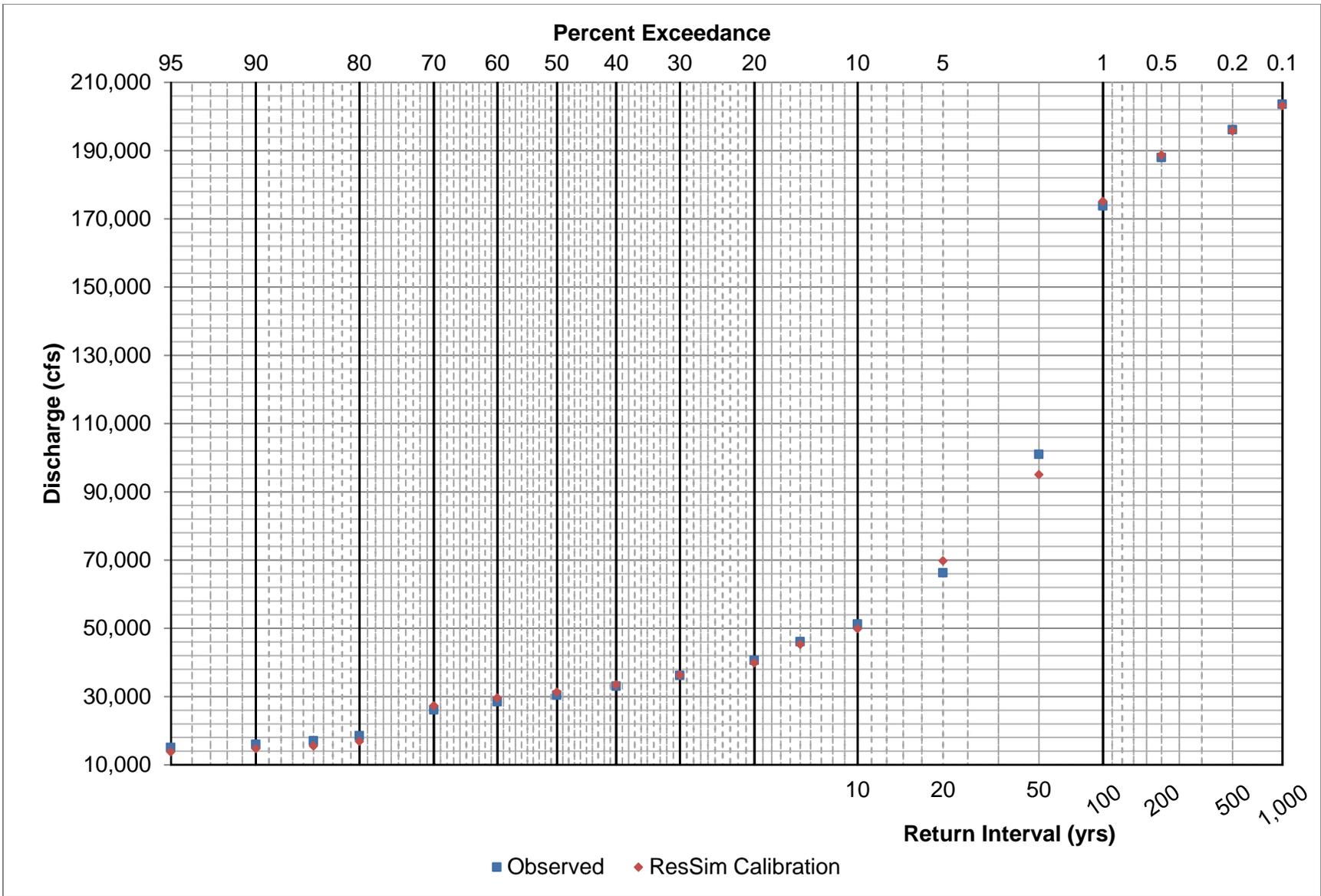


Figure 5-20: Omaha, NE flow probability curves for years 1998-2012.

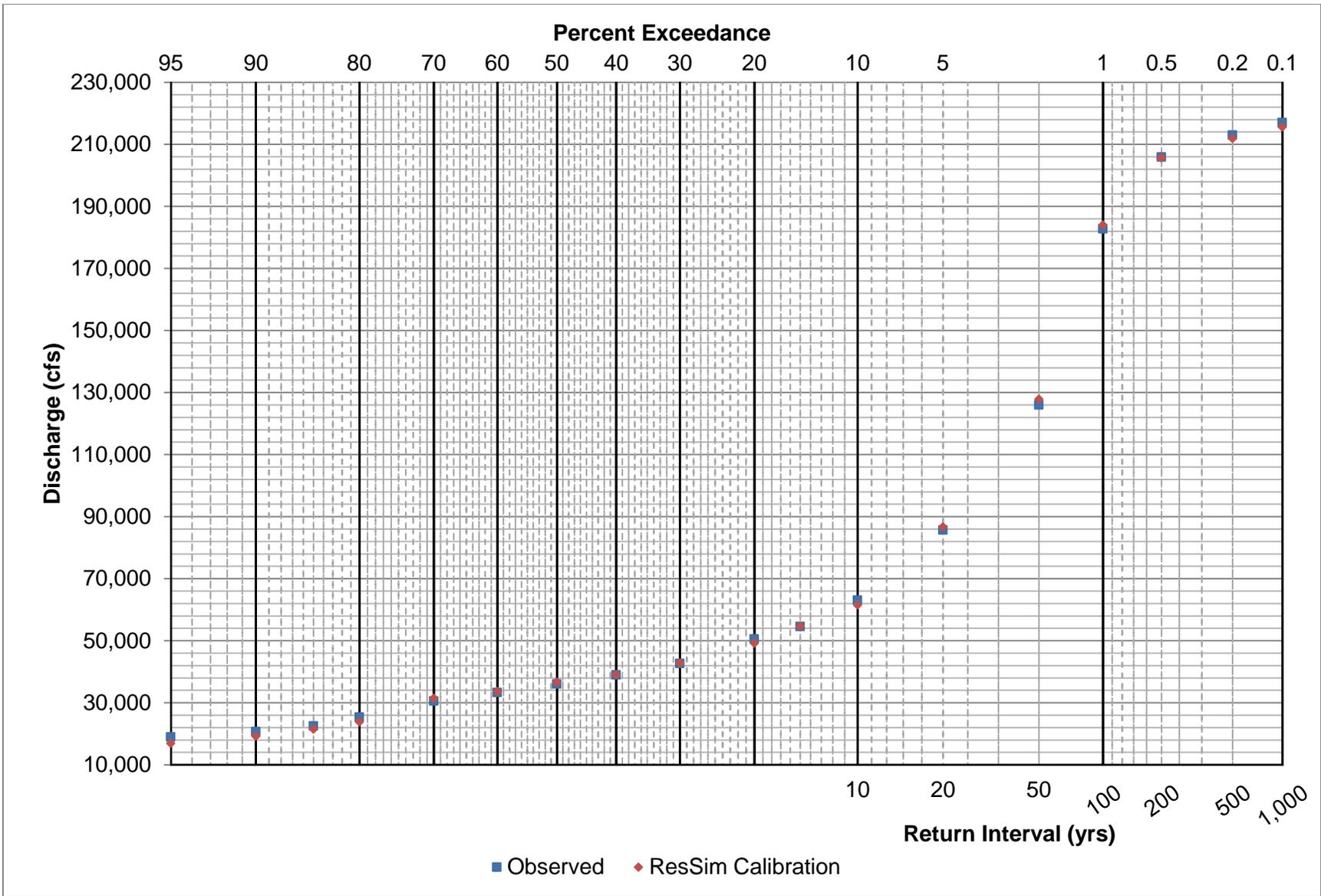


Figure 5-21: Nebraska City, NE flow probability curves for years 1998-2012.

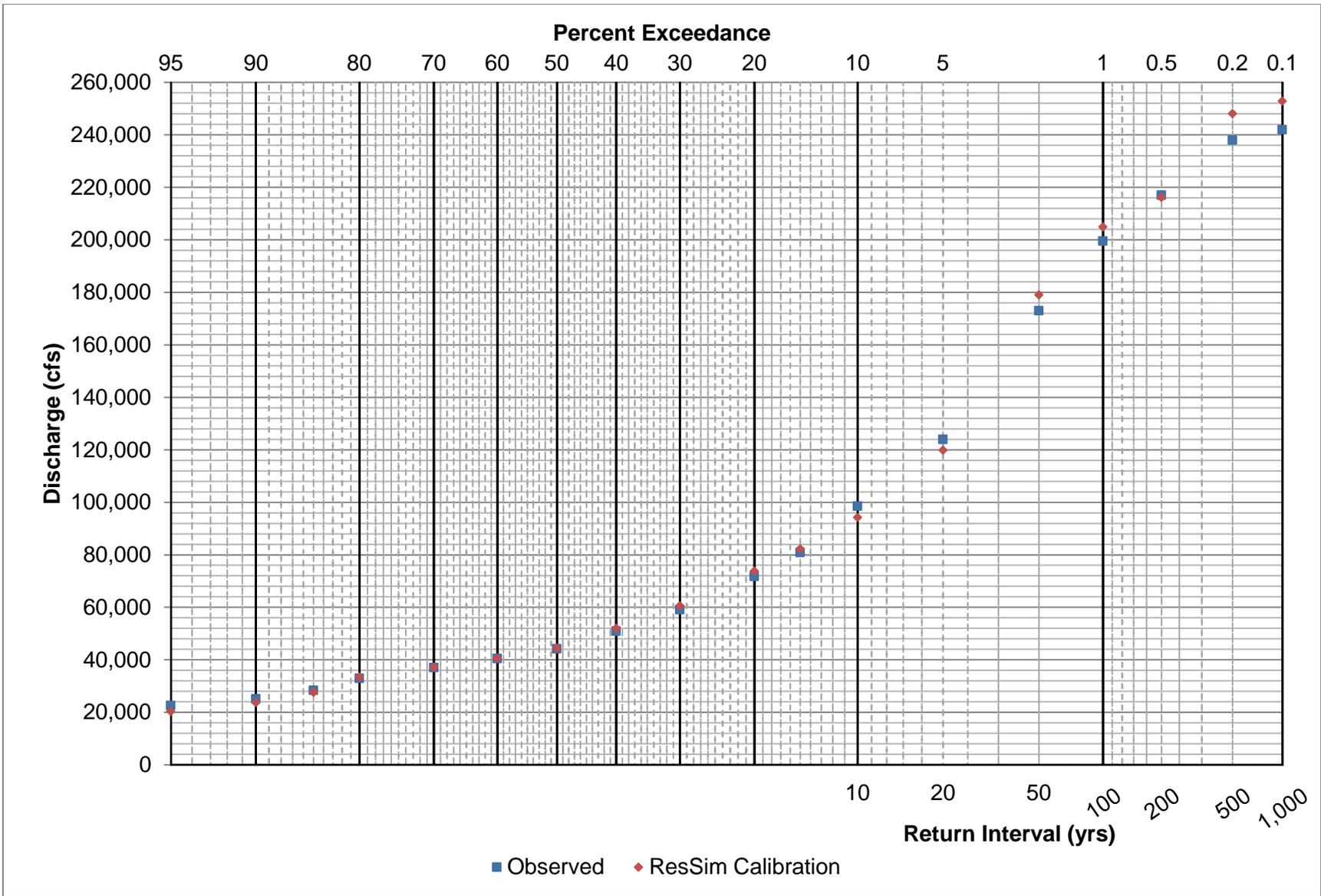


Figure 5-22: Kansas City, MO flow probability curves for years 1998-2012.

## 5.1 DECISION COMPARISONS

Since operations change throughout the period-of-record and a model will never be able to perfectly reproduce real-time decisions, a better estimate of model performance was checking operational decisions based on the simulated information. To check this, several key operational decisions were assessed during the period-of-record, 1930-2012. In the Downstream model, service level, meeting navigation target discharges, meeting water supply requirements, navigation end date, winter release, flood targets, and steady release decisions were examined. In the System model, balancing storage in the upper three reservoirs, meeting water supply requirements between reservoir, and meeting guide curve operations at Big Bend, Fort Randall, and Gavins Point Dams were examined. Each of these operational decisions were described in more detail in Section 2.3.

### 5.1.1 Downstream Model

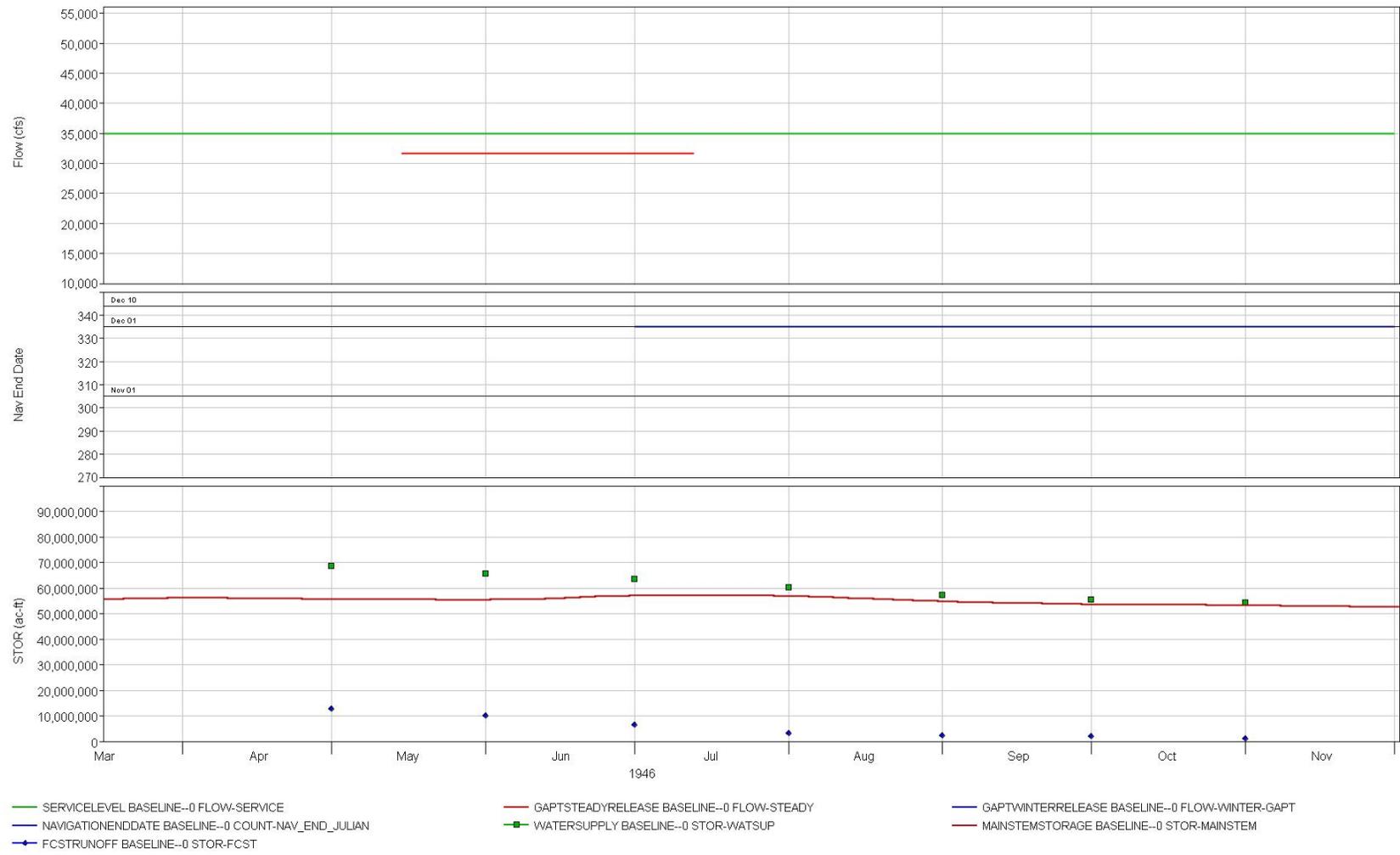
#### 5.1.1.1 Service Level

Normally, service level is calculated two times per year: March 15 and July 1. The March 15 storage check determines if there will be a navigation season and if there is a navigation season, the service level for the first half of the navigation season. Table 5-1 lists the service level requirements, which are based on water in system storage. For System storages between the values listed in Table 5-1, the service level is linearly interpolated.

**Table 5-1: Service level requirements. Summarized from Table VII-2 in the Master Manual (U.S. Army Corps of Engineers, 2006).**

Date	Service Level (cfs)	Water in System Storage (MAF)
March 15	35,000 cfs (full-service)	54.5 or more
March 15	29,000 cfs (minimum-service)	31.0 – 49.0
March 15	No service	31.0 or less
July 1	35,000 cfs (full-service)	57.0 or more
July 1	29,000 cfs (minimum-service)	50.5 or less

Several years were checked for service level calculations to ensure that years with full-service, minimum-service, no service, and interpolated service levels were correctly calculated on March 15 and July 1. In 1946, the simulated System storage on March 15 was approximately 55.7 MAF, which is greater than the minimum System storage for full-service of 54.5 MAF. The model correctly set the service level to 35,000 cfs or full-service for the first half of the season. The simulated System storage on July 1 was approximately 57.04 MAF, which is greater than the System storage for full-service of 57.0 MAF. The model correctly set the service level to 35,000 cfs or full-service for the remainder of the navigation season. Figure 5-23 shows a summary plot of the state variables calculated in the Service Level state variable for 1946, which includes the System storage and the service level.



**Figure 5-23: Plot of service level and system storage in 1946.**

In 2002, the simulated System storage on March 15 was approximately 48.9 MAF, which is less than the upper limit for minimum-service of 49.0 MAF. The model correctly set the service level to 29,000 cfs or minimum-service for the first half of the season. The simulated System storage on July 1 was approximately 48.8 MAF, which is less than the upper limit for minimum-service of 50.5 MAF. The model correctly set the service level to 29,000 cfs or minimum-service for the remainder of the navigation season. Figure 5-24 shows a summary plot of the state variables calculated in the Service Level state variable for 2002, which includes the System storage and the service level.

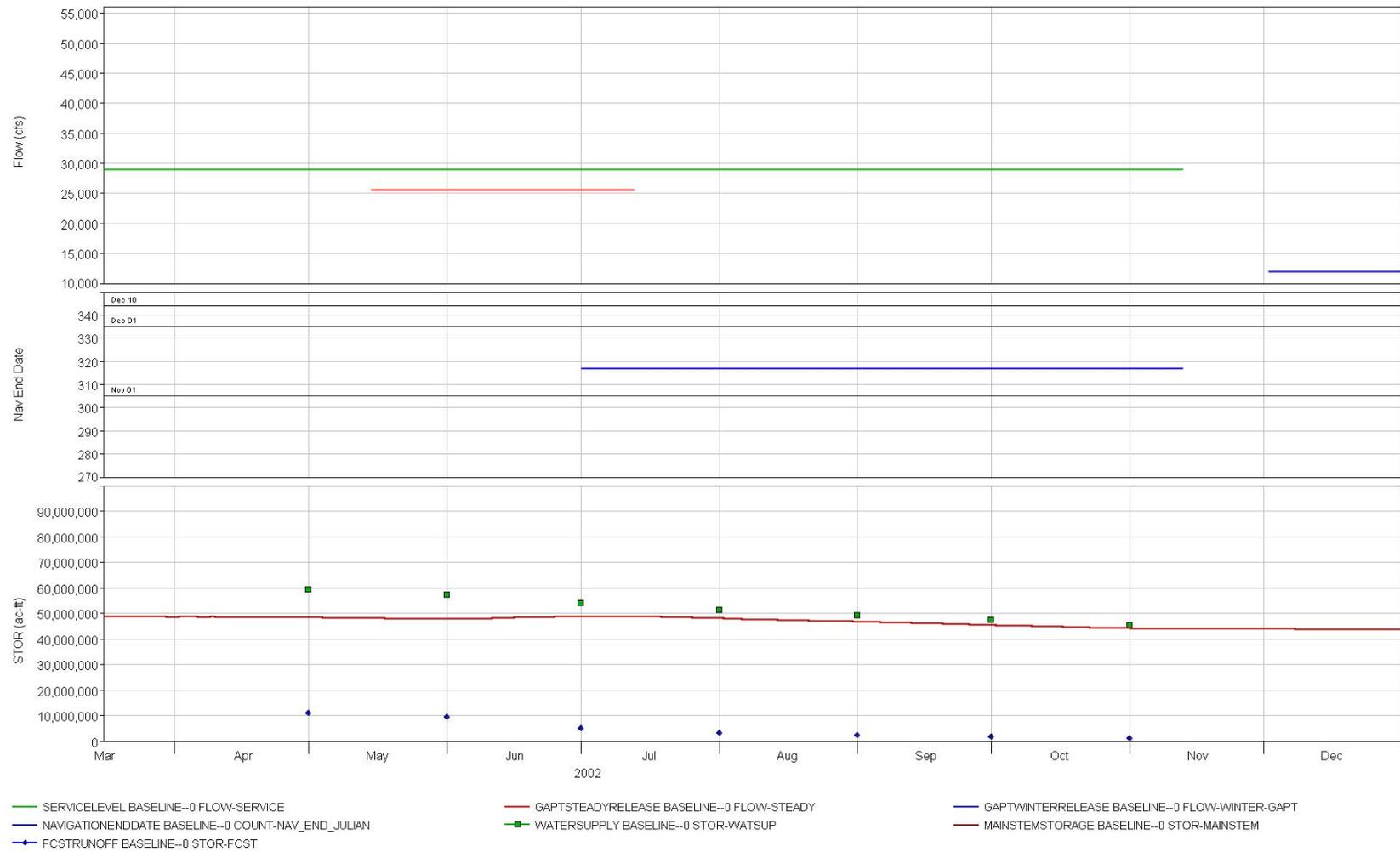


Figure 5-24: Plot of service level and system storage in 2002.

In 1981, the simulated System storage on March 15 was 52,551,398 acre-ft, which is between the full-service limit of 54.5 MAF and upper limit for minimum-service of 49.0 MAF. Interpolating between those limits, the model correctly set the service level to 32,874 cfs for the first half of the season. The simulated System storage on July 1 was 54,404,890 acre-ft, which is between the full-service limit of 57.0 MAF and upper limit for minimum-service of 50.5 MAF. Interpolating between those limits, the model correctly set the service level to 32,605 cfs or minimum-service for the remainder of the navigation season. Figure 5-25 shows a summary plot of the state variables calculated in the Service Level state variable for 2002, which includes the System storage and the service level.

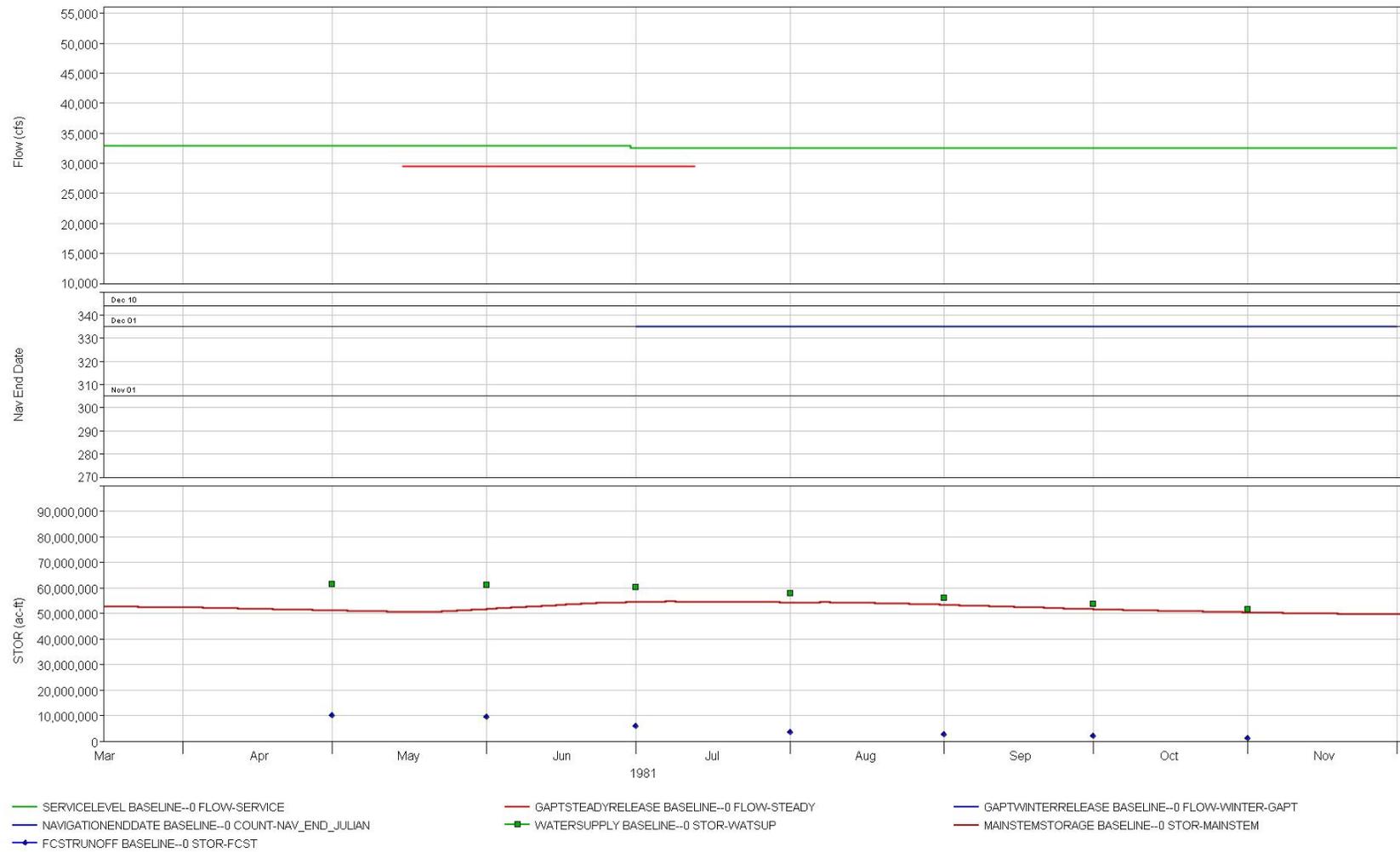
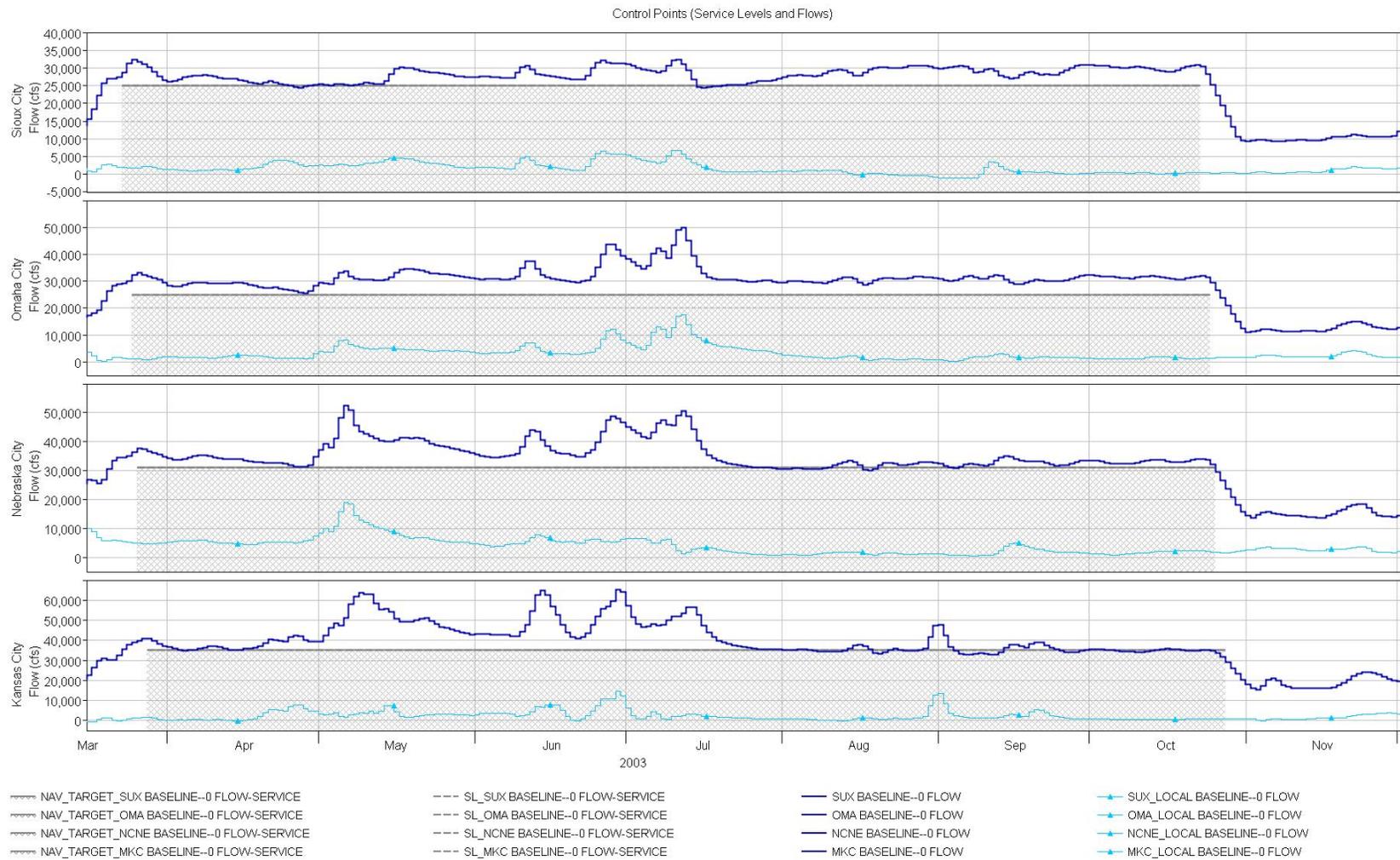


Figure 5-25: Plot of service level and system storage in 1981.

### 5.1.1.2 Navigation Targets

Meeting navigation discharges at the four target locations is a critical piece of the Missouri River operations. During non-flood periods, navigation target discharges are calculated by adjusting the service level with the information in Table 4-13. Gavins Point releases are set so that the navigation target discharges are met at each target location. The hatched area in Figure 5-26 represents the target discharges at Sioux City, Omaha, Nebraska City, and Kansas City. On the plot, the beginning and end of the hatched area represents the beginning and end of the navigation season at each location. In perfect operations, the discharges would never be lower than the target discharges; however, this is not reasonable because travel times between Gavins Point and the target locations require the use of local inflow forecasts. These forecasts can be incorrect and discharges may drop below the target discharge for short periods. The times when navigation targets are missed usually coincide with local flows at the governing location receding quicker than the forecasted local inflow. The governing location is the target location that requires Gavins Point releases to be adjusted because local inflows are not high enough to meet the navigation target with current Gavins Point releases; this location is identified as the location where the discharge is equal to the navigation target discharge (see Figure 5-26).

In 2003, the service level is 29,000 cfs or minimum service, which results in navigation target discharges of 25,000 cfs at Sioux City and Omaha, 31,000 cfs at Nebraska City, and 35,000 cfs at Kansas City. The model correctly calculates and operates for the navigation targets; the days a navigation target discharge is not met are attributed to times when the local inflow forecasts did not accurately capture the hydrograph of the local inflows. An example of this occurs on September 1 when Kansas City local inflows begin to recede more quickly than the forecasted local inflows causing discharges at Kansas City to fall below the navigation target discharge by September 5. Due to the travel time between Gavins Point and Kansas City, it takes a couple of more days before discharges return to the navigation target discharge.



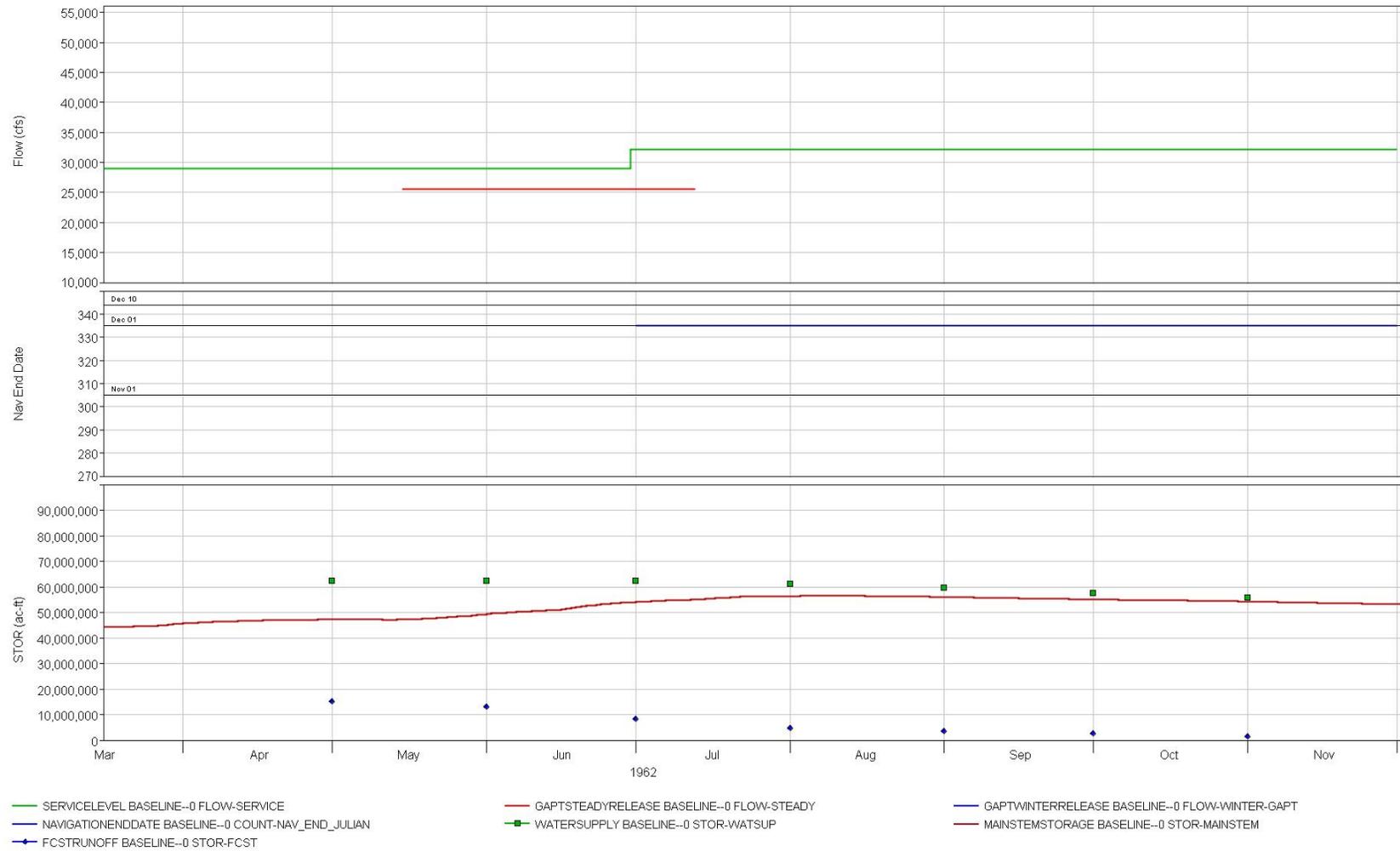
**Figure 5-26: Control points plot of target discharges and simulated discharges at each of the target locations.**

### **5.1.1.3 Steady Release**

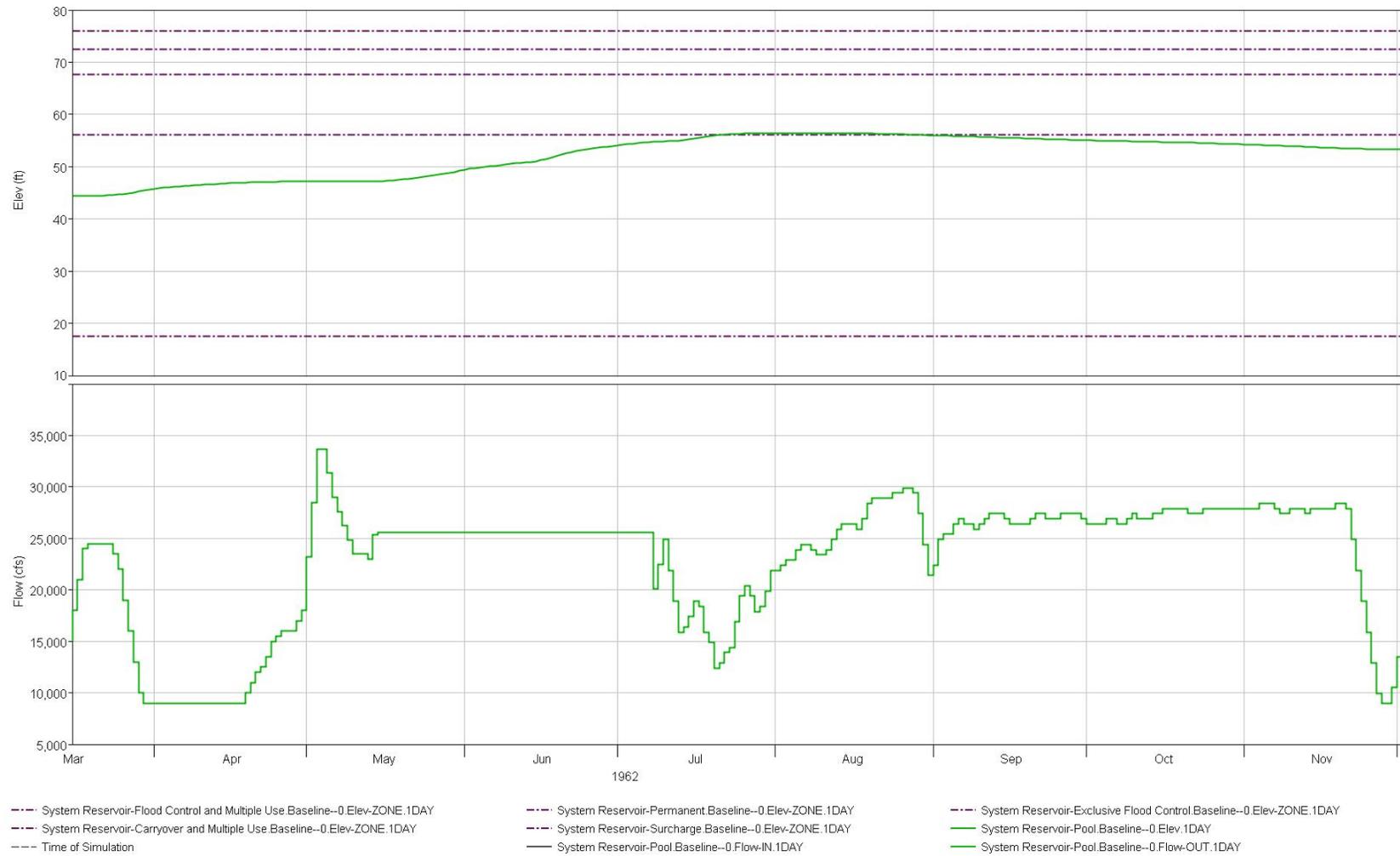
The System operates for a steady release between May 15 and July 15 when the least terns and piping plovers are nesting. The steady release during this period is based on the System release typically needed to ensure full-service or minimum-service for a below median forecasted runoff or a median or greater forecasted runoff during the month of July. In 1962, the service level on March 15 is 29,000 cfs or minimum-service; the March 15 forecasted runoff for March-July is 18.0 MAF, which is greater than the median forecasted runoff for March-July of 16.6 MAF. Using the data in Table 4-11, the steady release for minimum-service and a forecasted runoff greater than the median runoff is 25,600 cfs, which the model correctly calculates and applies for releases between May 15 and July 15 shown in Figure 5-28.

Another part of the steady release operations is ensuring releases are reduced when downstream flood targets are exceeded. In July of 1962, downstream flood control targets are exceeded and Gavins Point releases need to be reduced to alleviate downstream flooding. The model correctly reduces Gavins Point's releases for downstream flooding and ignores the steady release criteria.

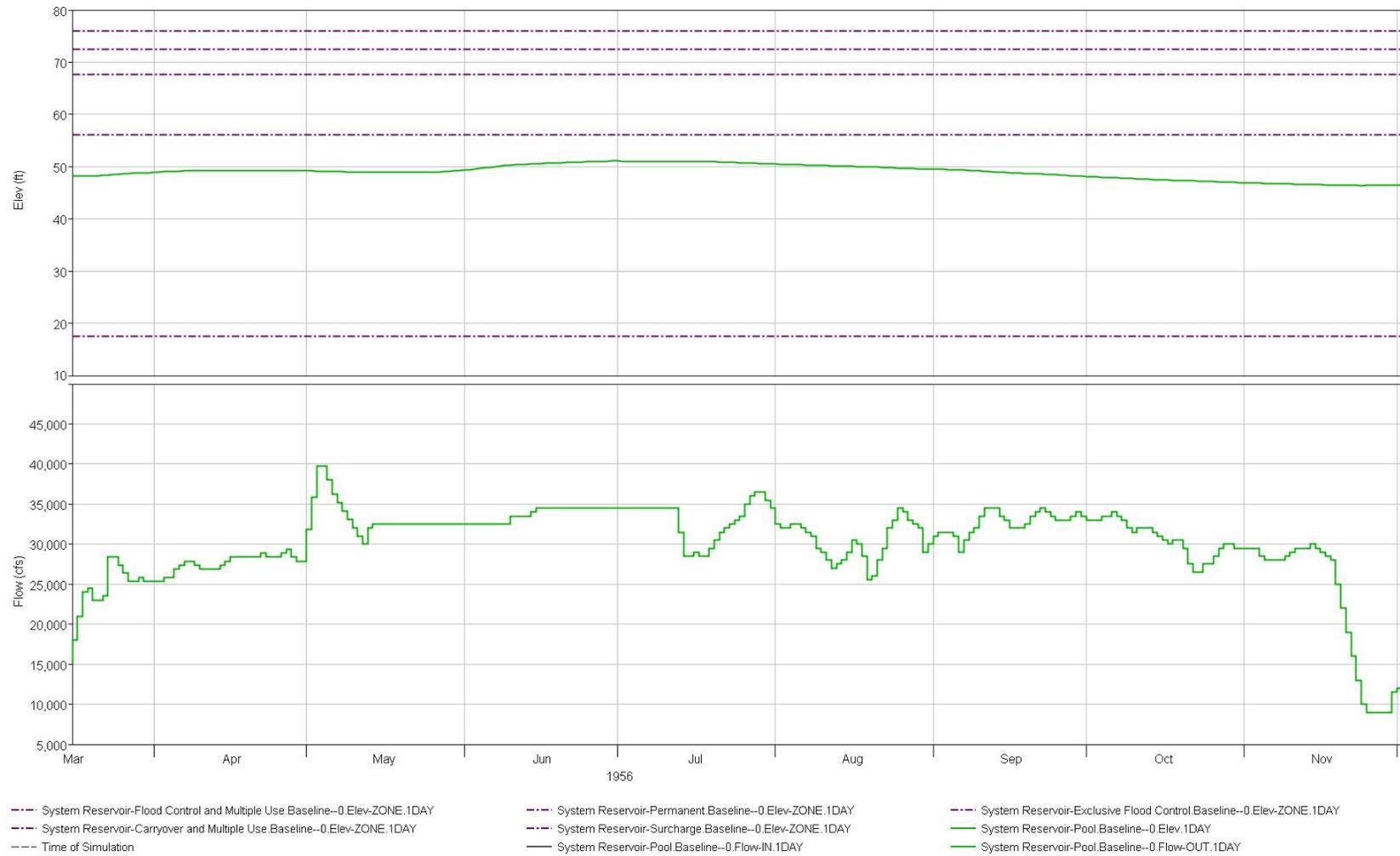
The final part of the steady release operations is ensuring the steady release meets all navigation targets. If it does not, increase the steady release and then hold the steady release at the higher required steady release. In 1956, a steady release of 25,600 cfs is calculated based on a forecasted runoff greater than the median and a service level of 29,000 cfs or minimum-service. If a 25,600 cfs steady release is used, navigation target discharges will not be met because local inflows are lower than forecasted; therefore, the model increases the steady release discharge until navigation targets will initially be met resulting in a steady release of 32,500 cfs on May 15. During June, the steady release is increased again to 33,500 cfs and finally to 34,500 cfs. Each time the steady release is increased, the model will hold the new steady release through the remainder of the steady release period or until it calculates that it needs to increase for navigation requirements. Figure 5-29 shows the increases in the steady release described above.



**Figure 5-27: Plot of forecasted runoff and steady release in 1980.**



**Figure 5-28: Plot of System releases in 1962.**



**Figure 5-29: Plot of System release in 1956.**

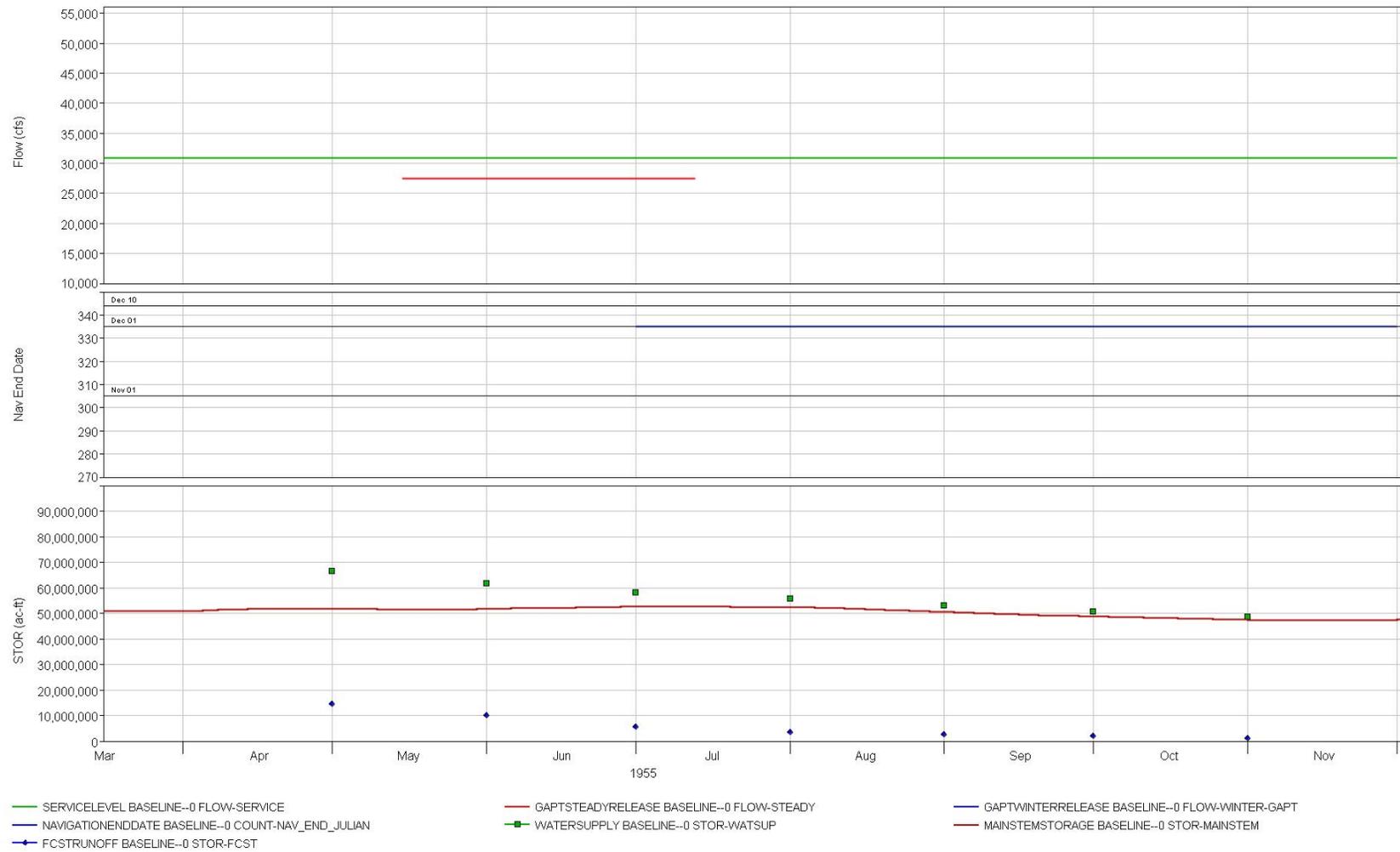
#### 5.1.1.4 Navigation End Date

The end of the navigation season or season length is determined on System storage on July 1 and the criteria is summarized in Table 5-2. If the System storage is between any of the two thresholds, the end date will be linearly interpolated.

**Table 5-2: Navigation end date or season length criteria. Summarized from Table VII-3 in the Master Manual (U.S. Army Corps of Engineers, 2006).**

Date	System Storage (MAF)	Season Closure Date at Mouth of the Missouri River
March 15	31.0 or less	No season
July 1	51.5 or more	December 1 – 8-month season
July 1	46.8 through 41.0	November 1 – 7-month season
July 1	36.5 or less	October 1 – 6-month season

In 1955, the System storage on July 1 is approximately 52.6 MAF. This is above the System storage amount for a full 8-month season, so the end date is December 1 and the season length is 8-months. The model correctly calculates the end date for the navigation season and sets the end date to December 1. In Figure 5-30, the end date is December 1 shown in Julian days. In 1956 the System storage on July is 51,062,195 acre-ft, which is between the thresholds for an 8-month season and a 7-month season. The model correctly interpolates the end date and sets the end date to Julian day 332 or November 28 as shown in Figure 5-31.



**Figure 5-30: Plot of the navigation end date in 1955.**

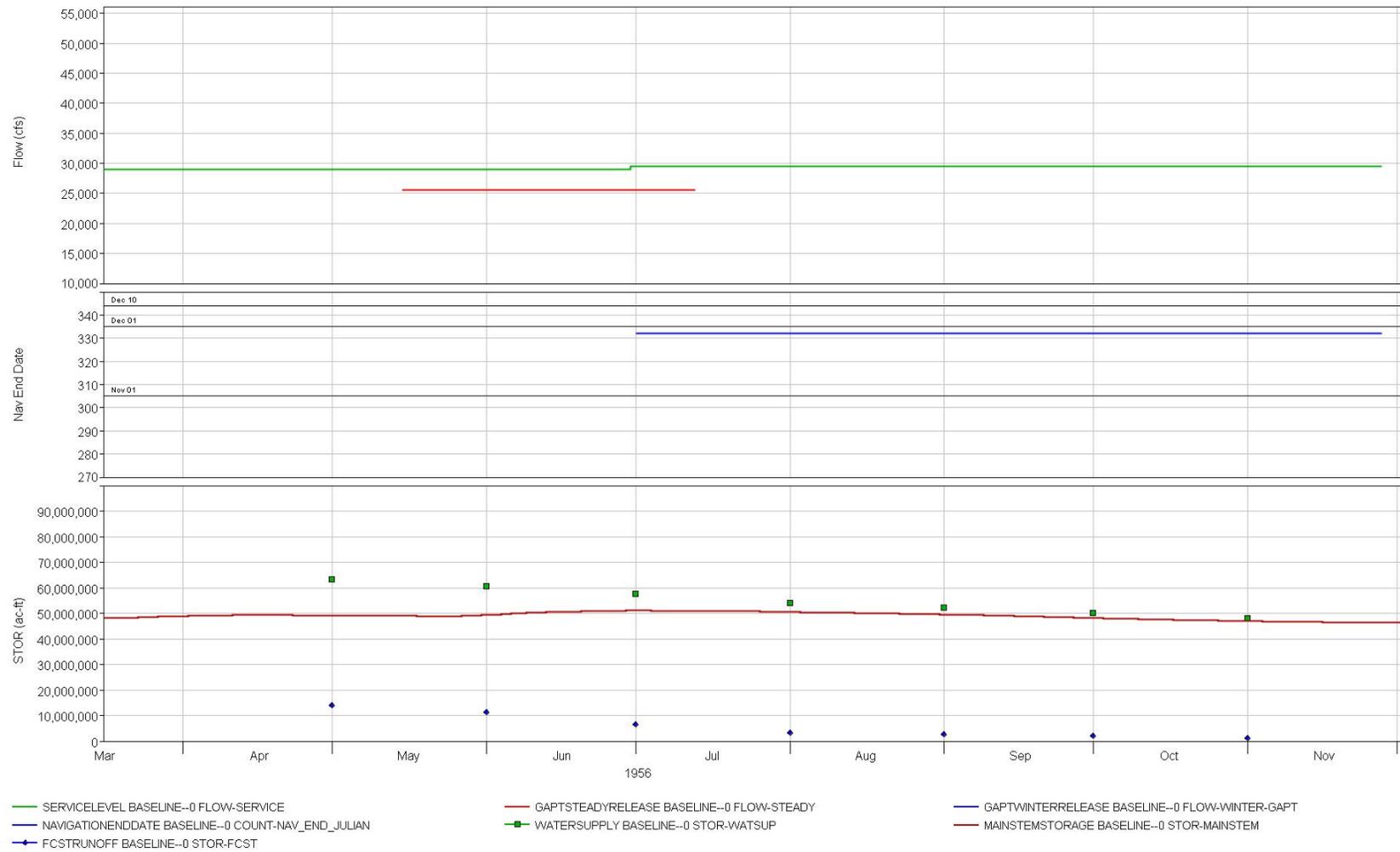


Figure 5-31: Plot of the navigation end date in 1956.

### 5.1.1.5 Winter Release

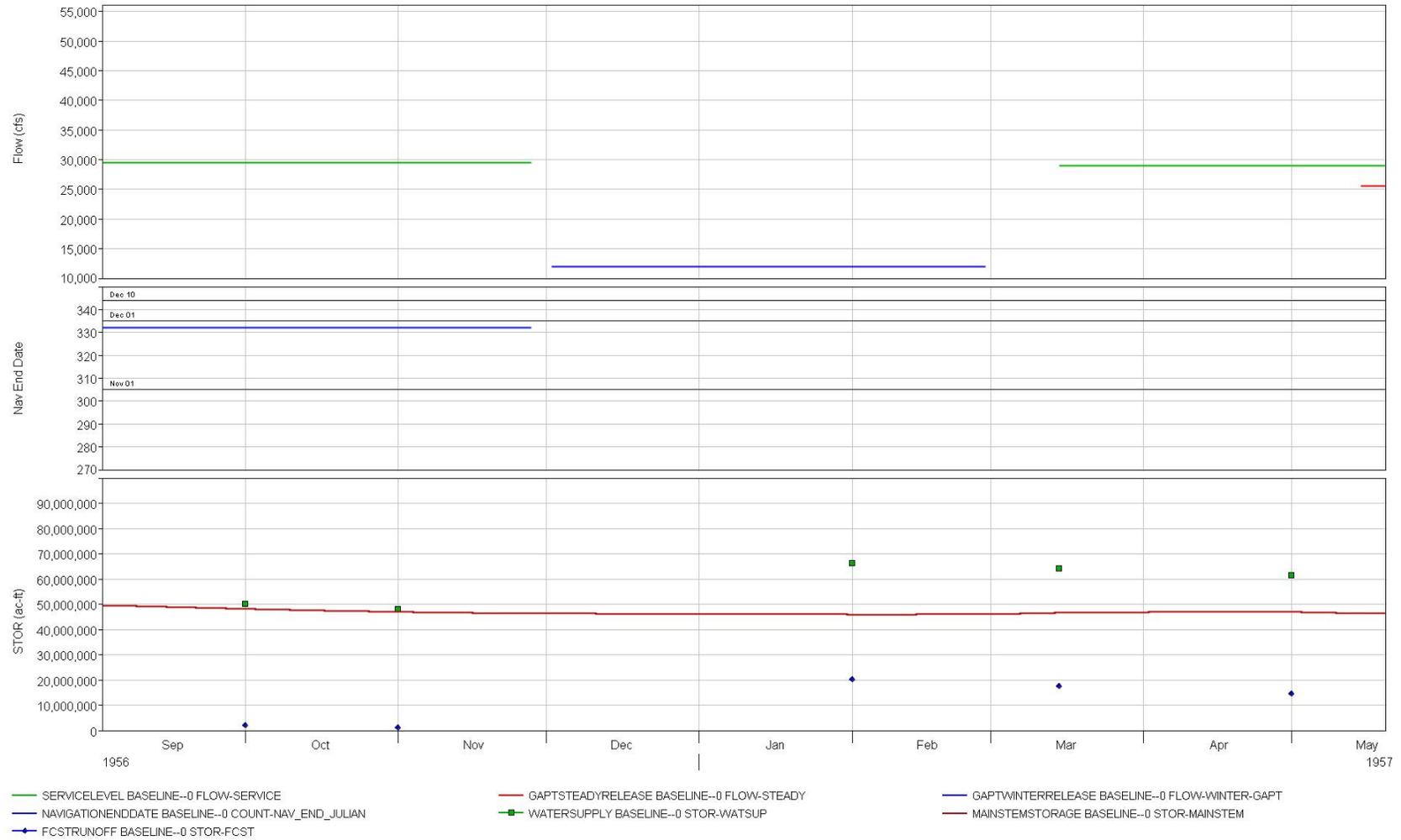
The winter release from Gavins Point is based on the System storage on September 1; the criteria are summarized in Table 5-3. If the system storage is between the thresholds for a 17,000 cfs winter release and a 12,000 cfs winter release, the winter release is linearly interpolated.

**Table 5-3: Winter release from Gavins Point criteria. Summarized from Table VII-4 in the Master Manual (U.S. Army Corps of Engineers, 2006).**

September 1 System Storage (MAF)	Average Winter Release from Gavins Point (cfs)
58.0 or more	17,000
55.0 or less	12,000

On September 1, 1956, the System storage is approximately 49.5 MAF, which is below the threshold for a 12,000 cfs winter release. The model correctly calculates the winter release and applies it during the winter release period. On September 1, 1943, the System storage is 57,795,955 acre-ft, which is between the thresholds for a 17,000 cfs and 12,000 cfs winter release. The model correctly interpolates the winter release and applies a release of 13,327 cfs during the winter release period. These results are shown in Figure 5-32 and Figure 5-33.

Winter releases may need to be increased during years when flood storage is not evacuated to a level where a normal winter releases will evacuate the remaining flood storage to reach a System storage of 56.1 MAF on March 1. During years such as this, the winter releases are greater than the calculated winter release until all of the flood storage has been evacuated. An example of this is shown in Figure 5-34 where the System storage is still too high for all of the flood storage to be evacuated utilizing the calculated winter release, so the winter release is approximately 25,000 cfs during December until the System storage reaches 56.1 MAF at which point the winter release drops to the normal winter release of 17,000 cfs. The winter release increases again in late February when the System storage exceeds 56.1 MAF to ensure that all of the annual flood control storage is available on March 1. Higher than normal winter releases are explained in more detail in Section 7-03.5.2 of the Master Manual (U.S. Army Corps of Engineers, 2006).



**Figure 5-32: Plot of winter releases in 1956-1957.**

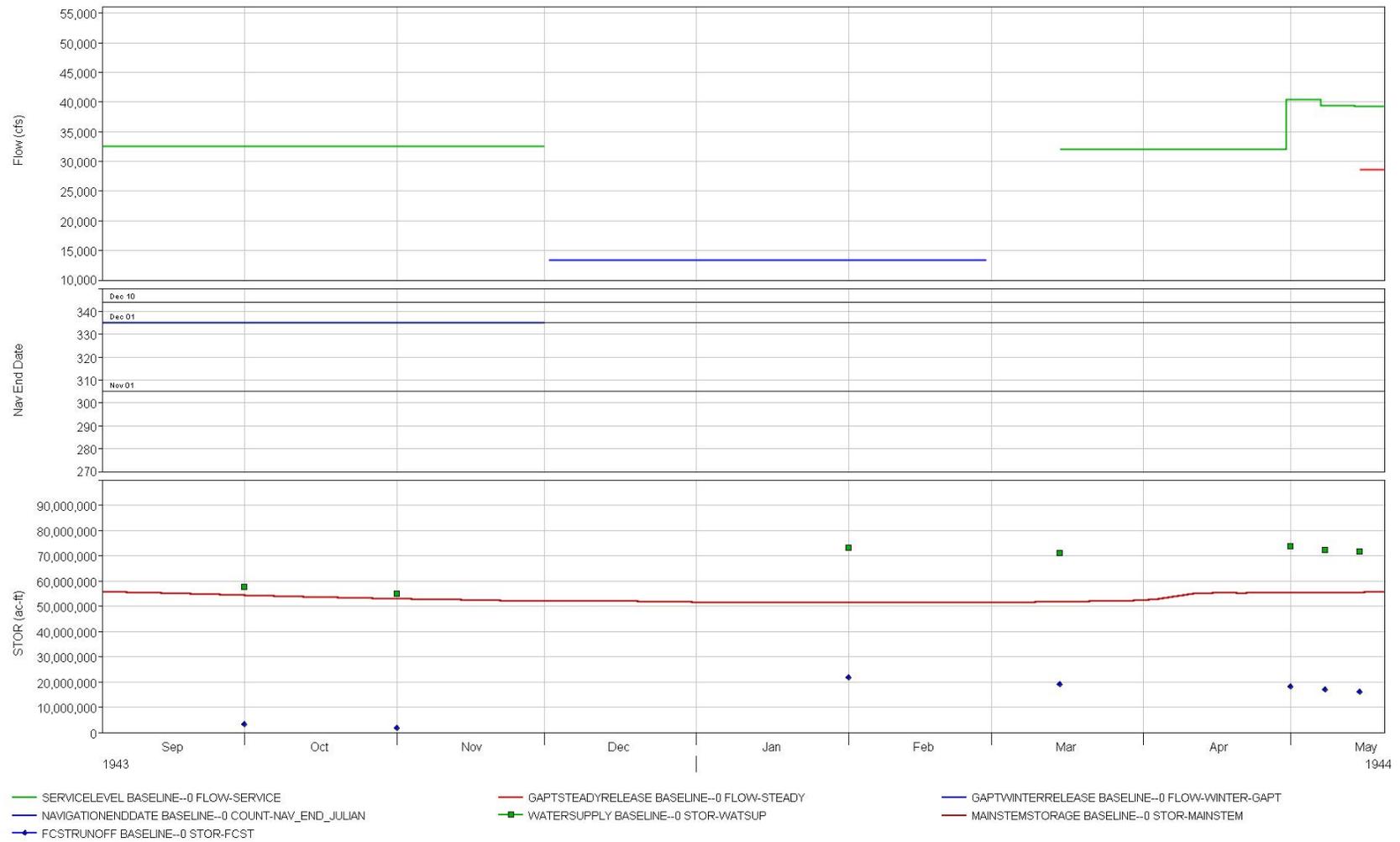
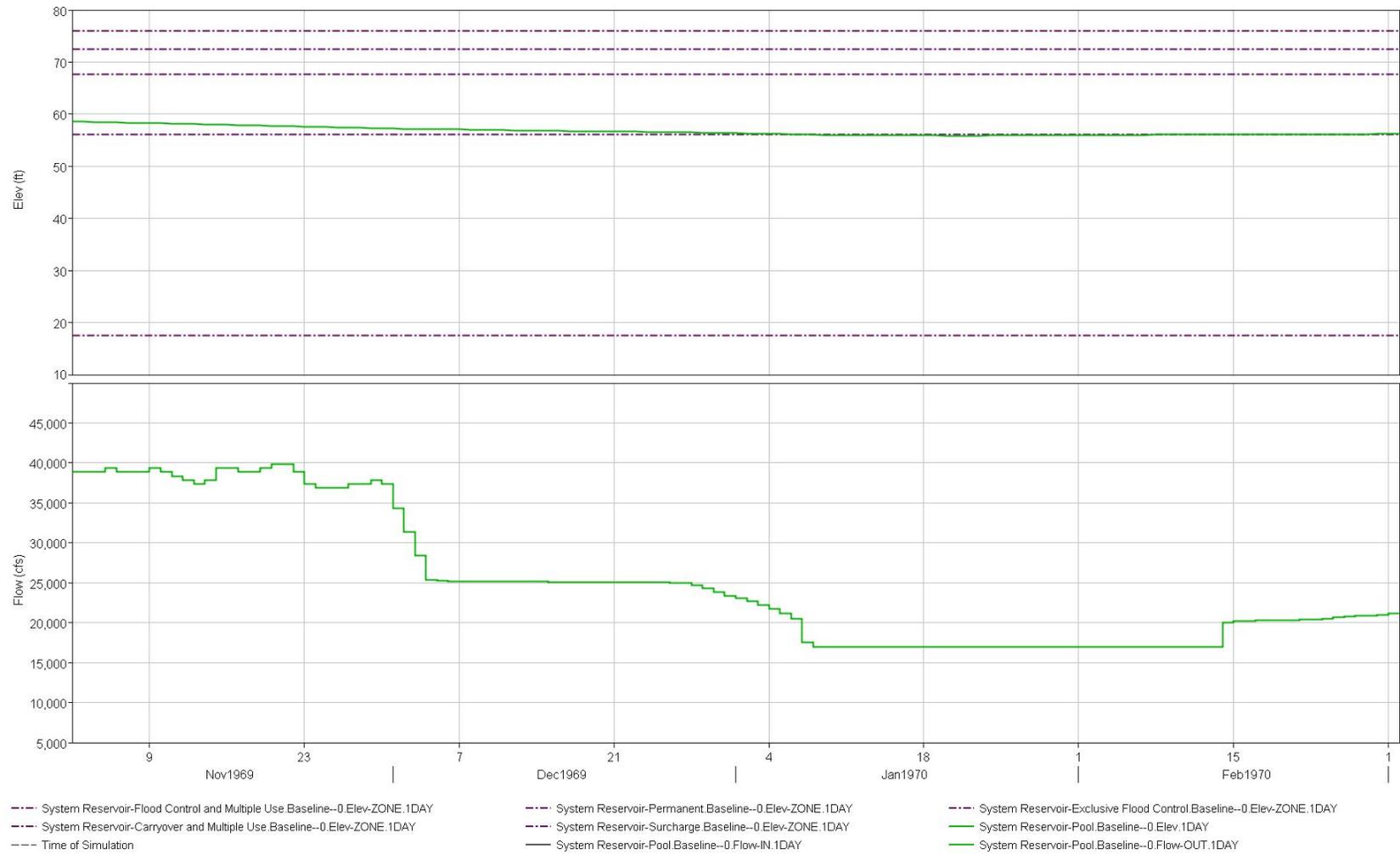


Figure 5-33: Plot of winter releases in 1943-1944.



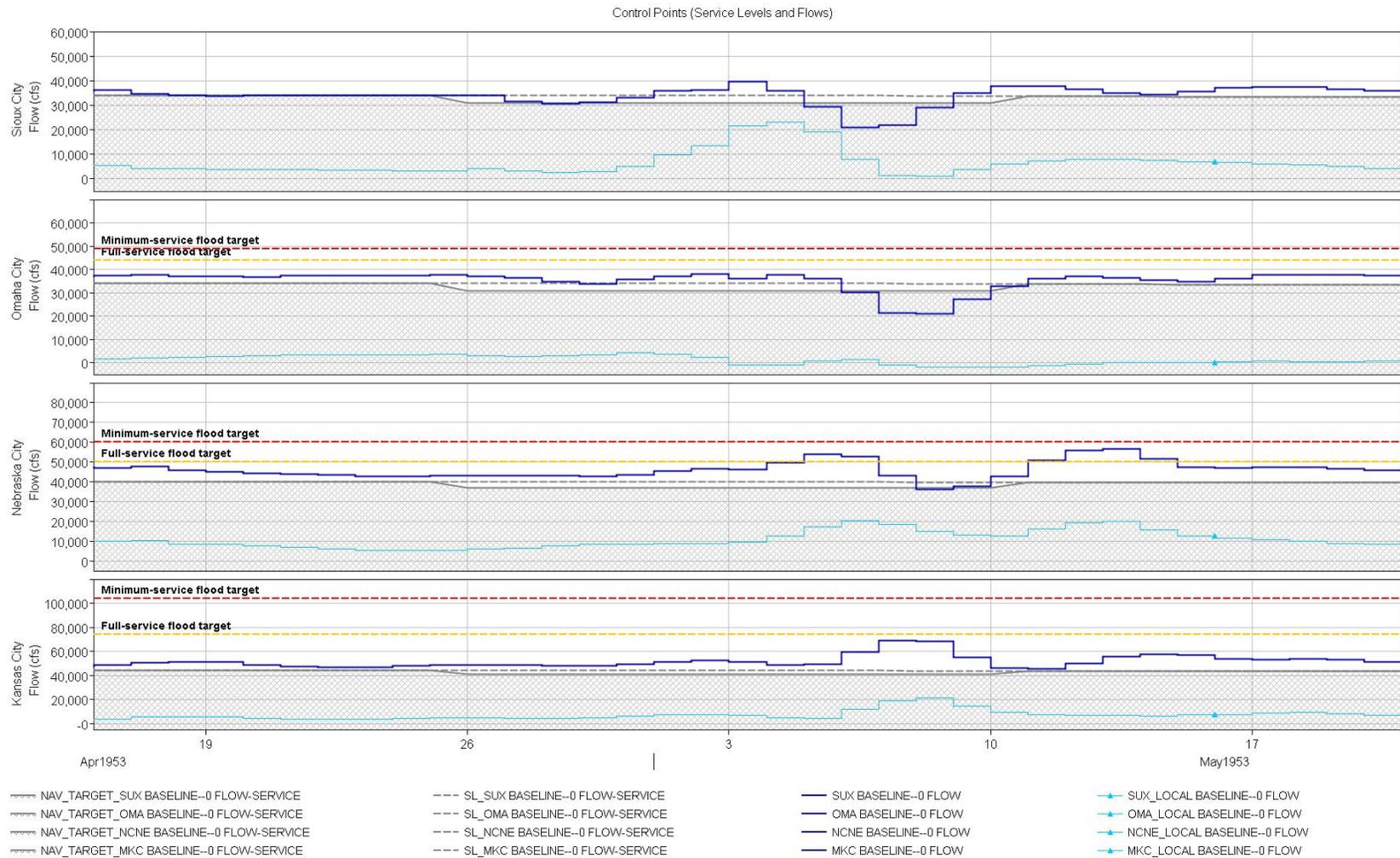
**Figure 5-34: Plot of Gavins Point releases and System storage in 1969-1970.**

#### 5.1.1.6 Flood Targets

One of the main functions of the Missouri River main stem dams is to provide flood control. Flood control on the Missouri River is more complicated than most systems because it needs to be balanced with navigation requirements. Gavins Point releases are reduced to alleviate downstream flooding but only to an extent that navigation targets are still met. Real-time operation can slightly differ from this during extreme flood events because parts of the river may be closed to navigation, which means that Gavins Point releases can be reduced further if certain navigation targets do not need to be met. For modeling purposes, it was assumed that the river was never closed for navigation during the period-of-record.

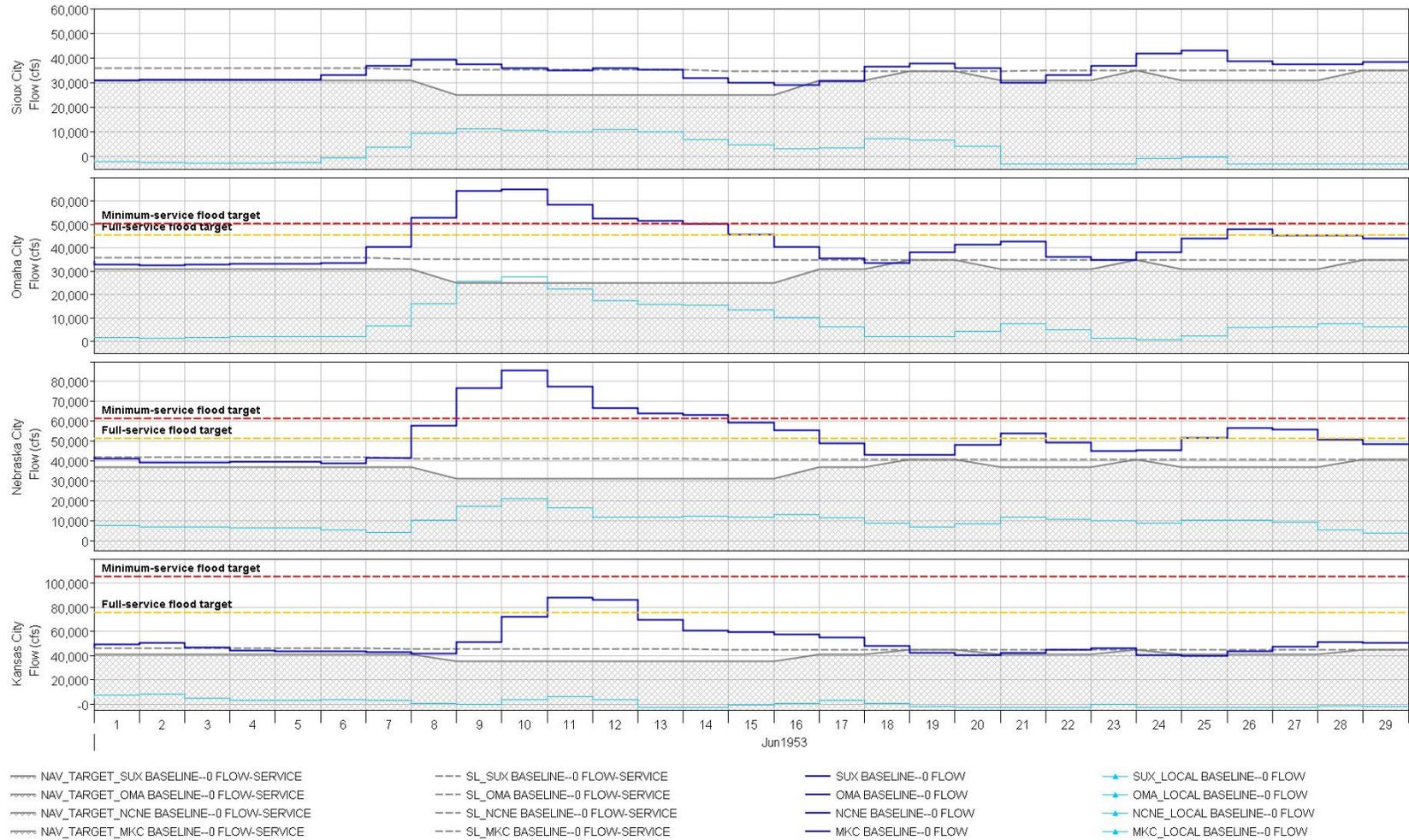
The flood target criteria is based on a two tier system and are summarized in Table 4-10. The first tier, full-service flood targets, is for smaller magnitude flooding and example of the model correctly reducing Gavins Point releases for full-service flood targets is shown in Figure 5-35. On April 26, 1953, the target discharge at Nebraska City is 39,978 cfs and the full-service flood target is 49,978 cfs. Discharges at Nebraska City are forecasted to exceed the full-service flood target of 49,978 cfs during the next 10 days, so navigation targets are set to full-service and Gavins Point releases are then decreased to alleviate downstream flooding. However, Gavins Point releases could not be reduced enough to prevent the discharges at Nebraska City from exceeding the full-service flood target and still meet full-service navigation requirements at Sioux City. This is shown on April 26-29 when discharges at Sioux City drop to near 31,000 cfs (full-service) while discharges at Nebraska City still exceed the full-service flood target on May 5-6. By May 7, the downstream flooding has subsided and navigation targets return to the navigation targets based on service level.

The second tier, minimum-service flood targets, is for larger magnitude flooding and example of the model correctly reducing Gavins Point releases for minimum-service flood targets is shown in . On May 1, 1953, the target discharges at Omaha and Nebraska City are 35,163 cfs and 41,163 cfs, respectively, and the full-service flood targets are 45,163 cfs and 51,163 cfs, respectively. Discharges at Omaha and Nebraska City are forecasted to exceed the full-service flood targets during the next 10 days, so navigation targets are set to full-service and Gavins Point releases are then decreased to alleviate downstream flooding. However, Gavins Point releases could not be reduced enough to prevent the discharges at Omaha and Nebraska City from exceeding the full-service flood targets and still meet full-service navigation requirements at Sioux City. This is shown in Figure 5-36 on May 1-7 when discharges at Sioux City drop to near 31,000 cfs (full-service) while discharges at Omaha and Nebraska City still exceed the full-service flood target on May 8. Also occurring on May 8, local inflows at Omaha and Nebraska City increase and cause discharges at those location to exceed their minimum-service flood targets of 50,163 cfs and 61,163 cfs, respectively. Once this occurs, the model correctly reduces navigation targets to minimum-service targets and Gavins Point releases are reduced further in an attempt to minimize downstream flooding while still meeting minimum-service at all target locations.



**Figure 5-35: Plot of target locations with full-service flood target reductions in 1953. Gray hatched areas are the navigation requirements that Gavins Point is operating for and the gray dotted line is the navigation targets based on service level.**

Control Points (Service Levels and Flows)



**Figure 5-36: Plot of target locations with full-service and minimum-service flood target reductions in 1953. Gray hatched areas are the navigation requirements that Gavins Point is operating for and the gray dotted line is the navigation targets based on service level.**

### 5.1.1.7 Water Supply Requirements

Water supply requirements are always met during navigation seasons, so the only time a release is made for water supply will be during winter release periods or during years of shortened or no navigation seasons. Section 7-11.3.5 of the Master Manual contains the criteria that the model's water supply requirements are based on:

“When the water-in-storage in the System is at normal or higher levels, releases for the navigation and power production purposes and to evacuate flood control storage during the navigation season and winter period will normally be at levels that are deemed to be sufficient for the downstream water supply needs. During extended droughts, Gavins Point Dam releases are reduced. Some intakes require more than 9,000 cfs (minimum release required in the early 1990's) during the open-water season for effective operation. These intakes should be modified as soon as possible to ensure that they can remain operational as the Corps continues to pursue lowering the Gavins Point Dam release in the non-navigation months during drought periods to this rate. A winter Gavins Point Dam minimum release rate of 12,000 cfs has been established as the guide in meeting downstream water supply requirements during this period. Intakes typically have higher requirements during the winter period because of the effects of river ice in reducing the capacity of their intakes. If Gavins Point Dam release rates are reduced below 12,000 cfs for water conservation, continued surveillance of these intakes will be required, and, if appropriate, additional releases may be required to assure adequate water levels for uninterrupted intake operation. During the critical and more difficult winter period, release rates may be adjusted according to river icing conditions to assure that the water supply service is provided downstream. During drought years when System storage is low enough to reduce or eliminate the navigation season, a Gavins Point Dam release of 18,000 cfs has been established as meeting the summer water supply requirement. Intake owners should modify their intakes as soon as possible if a summer Gavins Point Dam release rate of 18,000 cfs will not be adequate to meet their needs.”

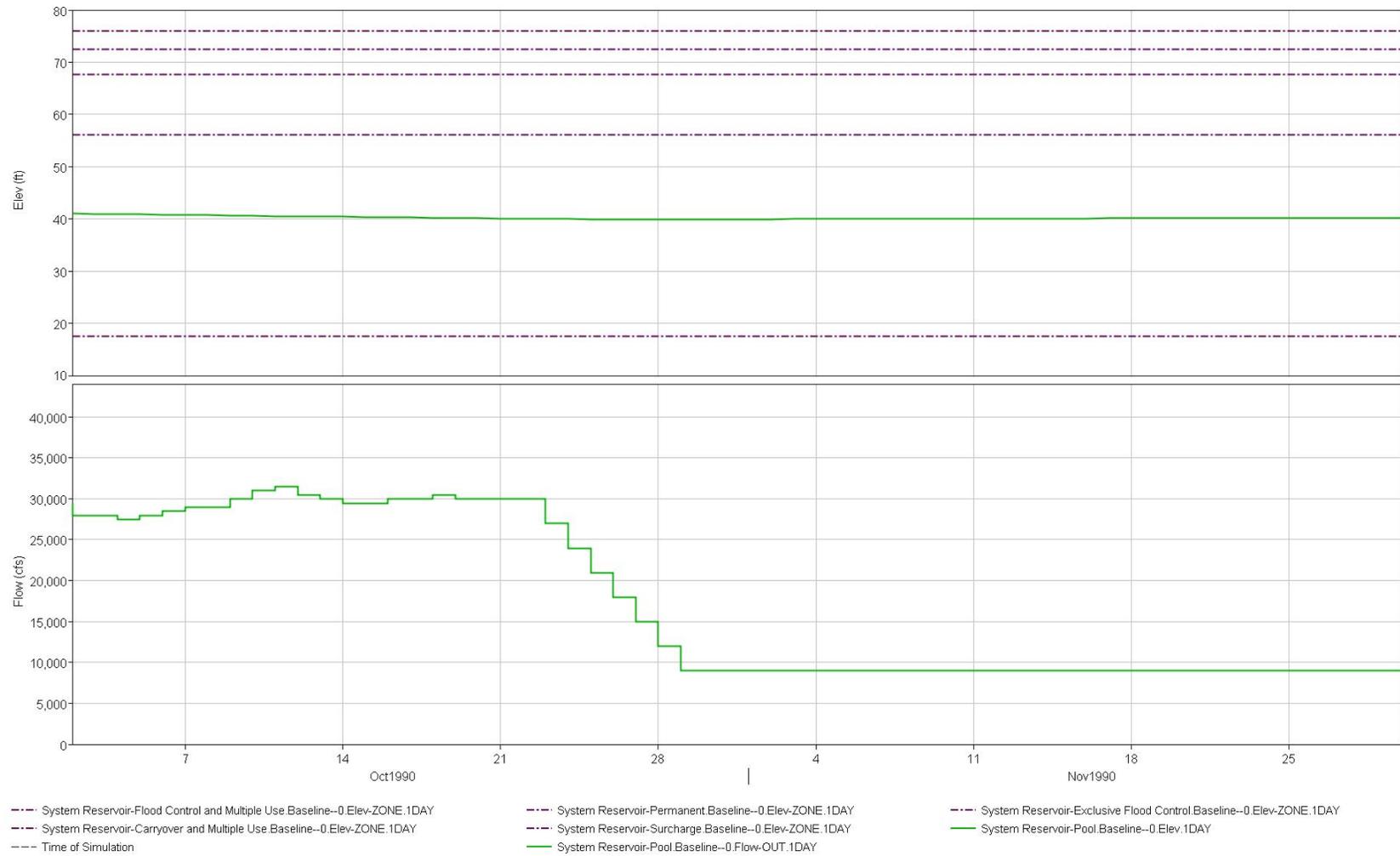
Real-time operations allows for some flexibility in meeting water supply needs during droughts, such as increasing releases temporarily to ensure water supply intakes have access to the river. However, the model does not have this flexibility. Strict dates and discharges were used in the model for water supply operations. The dates are listed in Section 7-03.6.1 of the Master Manual and are summarized in Table 5-4.

**Table 5-4: Water supply dates and discharges summarized from Sections 7-03.6.1 and 7-11.3.5 of the Master Manual (U.S. Army Corps of Engineers, 2006).**

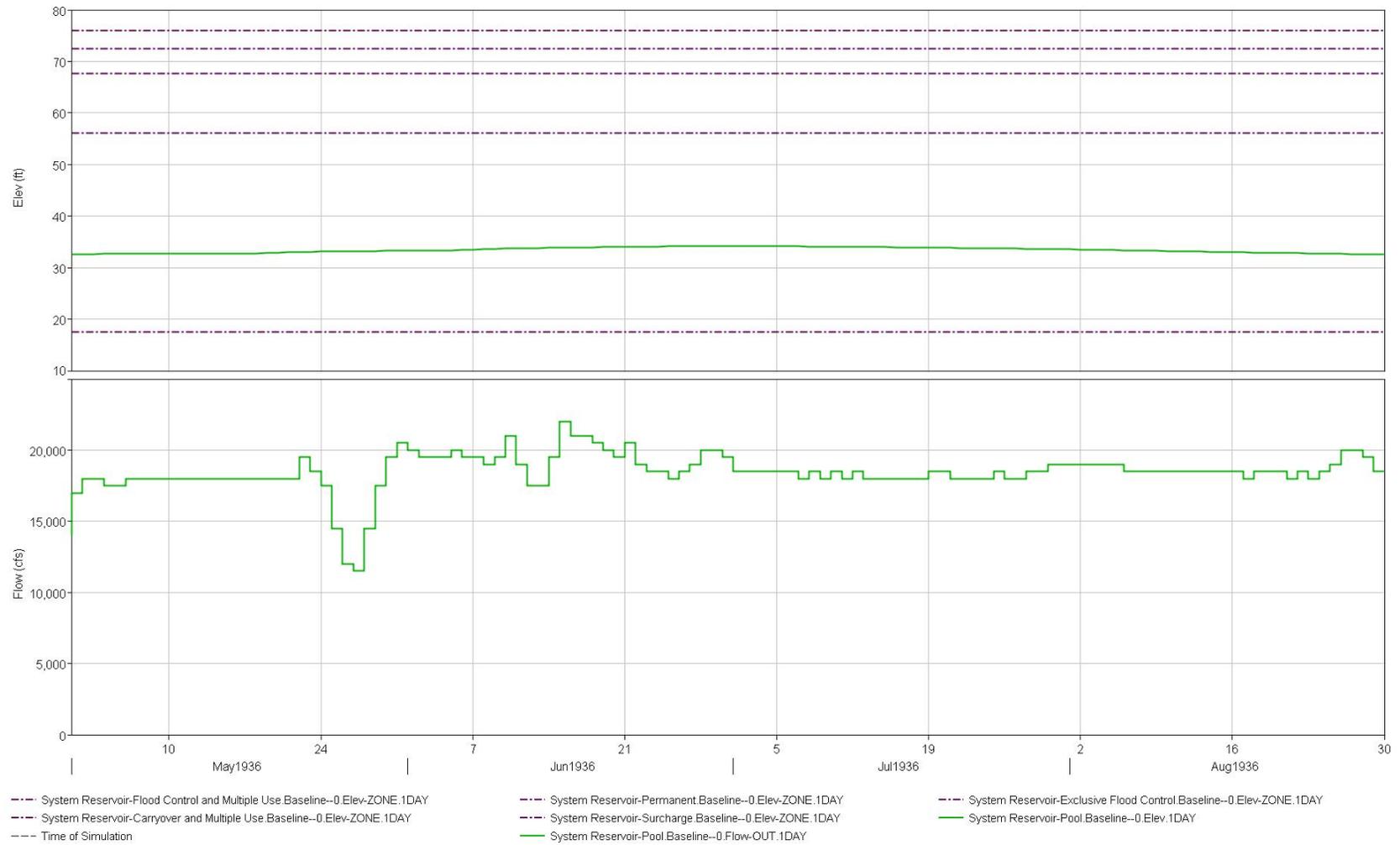
Date	Minimum Gavins Point Release for Water Supply (cfs)
March-April, September-November	9,000
May-August	18,000
December-February	12,000

In 1990, there was a shortened navigation season ending on November 1. Between November 1 and December 1, the system was operating for water supply requirements; the model correctly reduced Gavins Point's releases to 9,000 cfs at the end of the navigation season until December 1, as shown in Figure 5-37. The model has added logic for water supply that treats the minimum Gavins Point releases for water supply as minimum discharges at three locations: Omaha, Nebraska City, and Kansas City. This helps to alleviate water supply impacts that would occur due to the increased depletions used in the model to create a simulation representative of the current basin development. With this added logic, the model could specify Gavins Point releases below 9,000 cfs if local inflows were sufficient to meet water supply requirements at the three locations, which would cause water supply problems for cities between Gavins Point and Omaha. If this is the case, the model will specify a release of at least 9,000 cfs during March-November and at least 12,000 cfs during December-February.

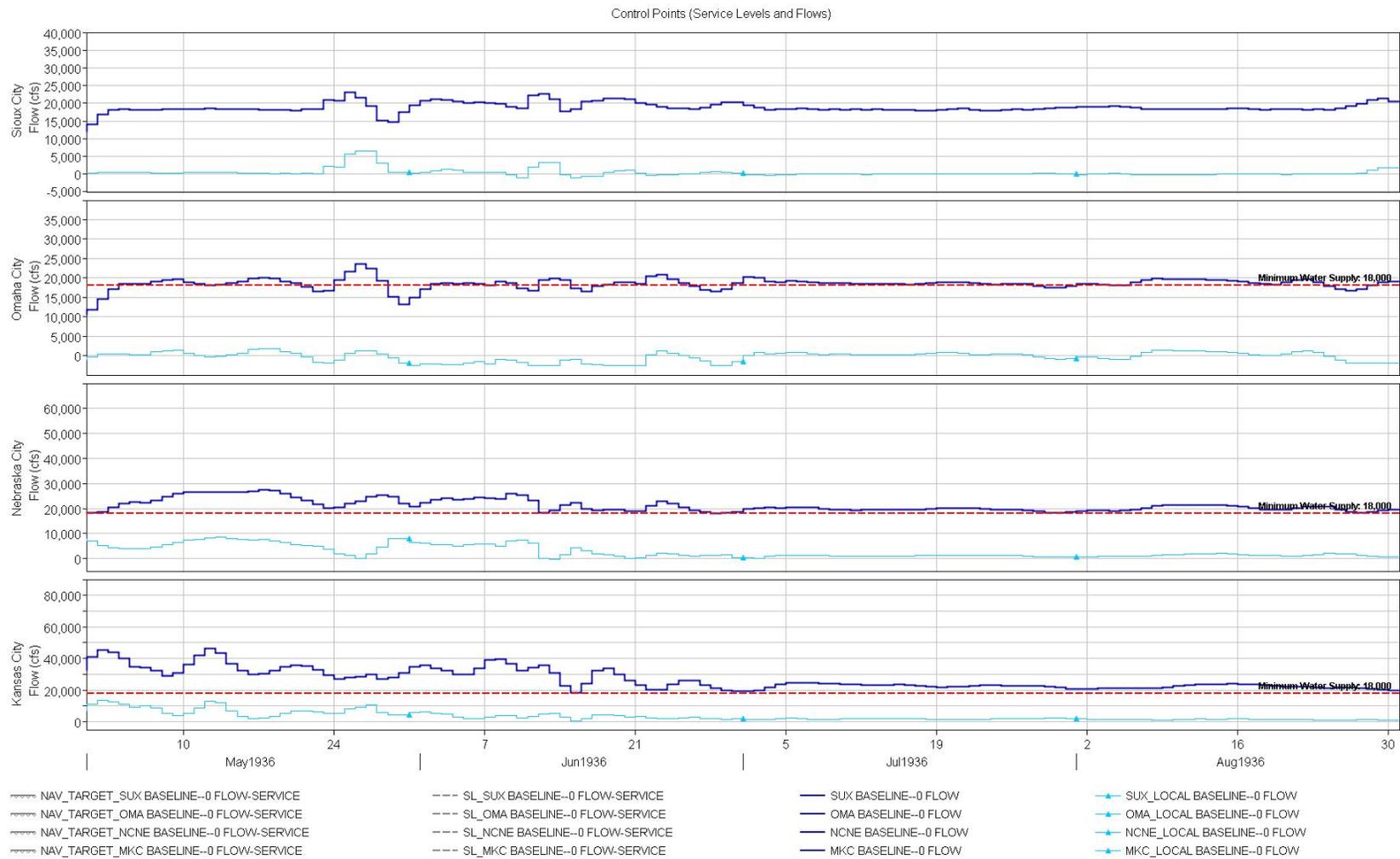
In 1936, the System is not operating for navigation because System storage is too low; therefore, the System is operating for water supply throughout the entire year. During May-August, the model correctly assesses downstream conditions and sets Gavins Point's releases to ensure a minimum of 18,000 cfs is observed at Omaha, Nebraska City, and Kansas City as shown in Figure 5-38 and Figure 5-39. During September-November, the model correctly assesses downstream conditions and sets Gavins Point's release to ensure a minimum of 9,000 cfs is observed at Omaha, Nebraska City, and Kansas City. If Gavins Point's required release is less than 9,000 cfs to meet minimum discharges of 9,000 cfs at the three locations, Gavins Point's release is set to 9,000 cfs. This is shown in September 1 – October 30 in Figure 5-40 when Gavins Point's releases are a constant 9,000 cfs because downstream local inflows are sufficient to provide a minimum of 9,000 cfs at the three locations. In October 31 – November 30, downstream local inflows are not sufficient for water supply, so Gavins Point releases are increased above the minimum of 9,000 cfs to ensure a minimum of 9,000 cfs is observed at the three locations as shown in Figure 5-41. During December-February, the model correctly assesses downstream conditions and sets Gavins Point's release to ensure a minimum of 12,000 cfs is observed at Omaha, Nebraska City, and Kansas City. At times when local inflows with a Gavins Point release of 12,000 cfs are sufficient to meet minimum discharges at the three locations, Figure 5-42 shows a constant 12,000 cfs Gavins Point release. At times when local inflows with a 12,000 cfs release are not sufficient to meet minimum discharges at the three locations, Figure 5-42 shows Gavins Point releases are increased to ensure a minimum discharge and Figure 5-43 shows discharges at the three locations maintaining 12,000 cfs except when local inflow forecasts are incorrect.



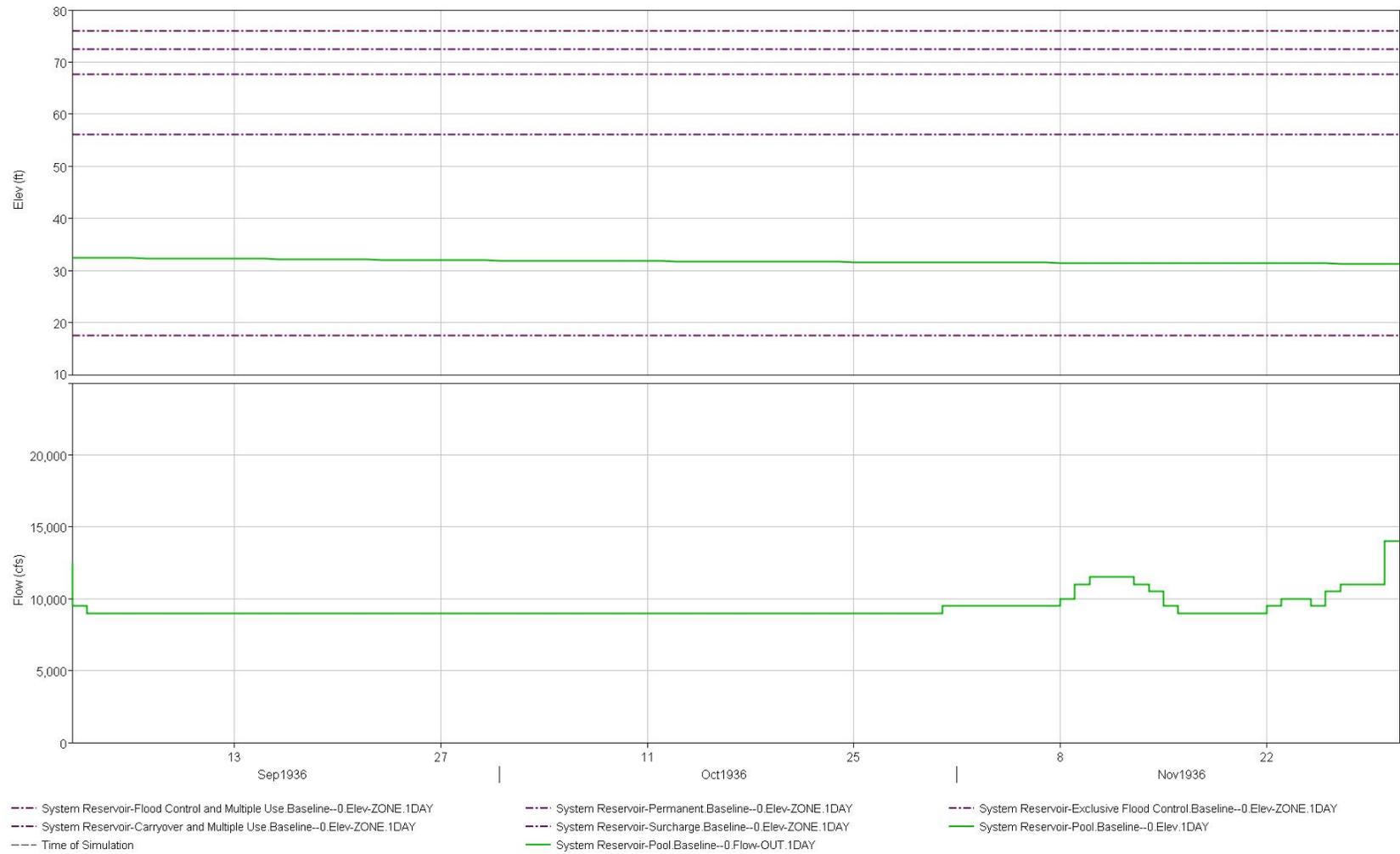
**Figure 5-37: Plot of Gavins Point releases and System storage in 1990 operating for water supply.**



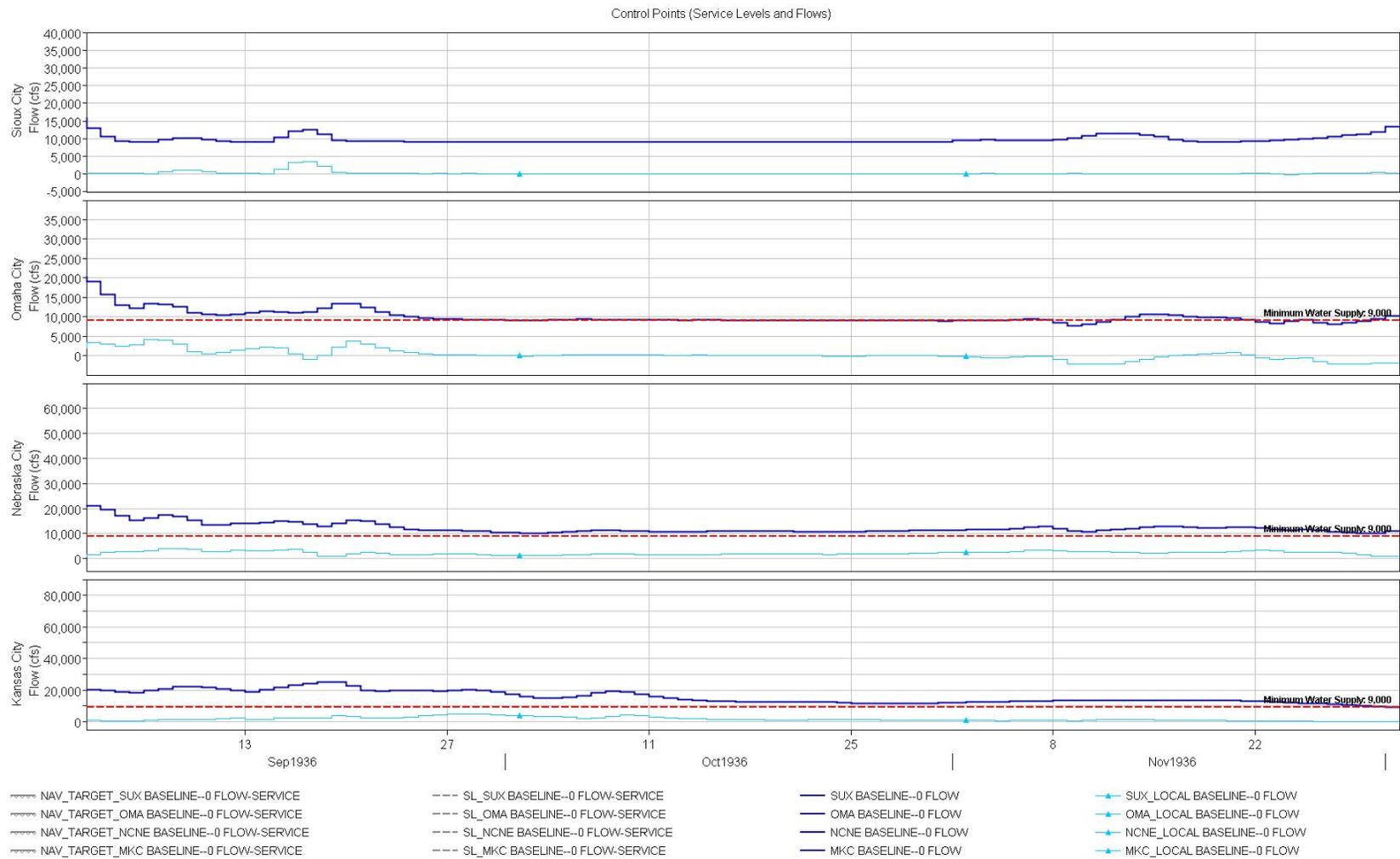
**Figure 5-38: Plot of System releases and System storage in 1936 operating for 18,000 cfs water supply.**



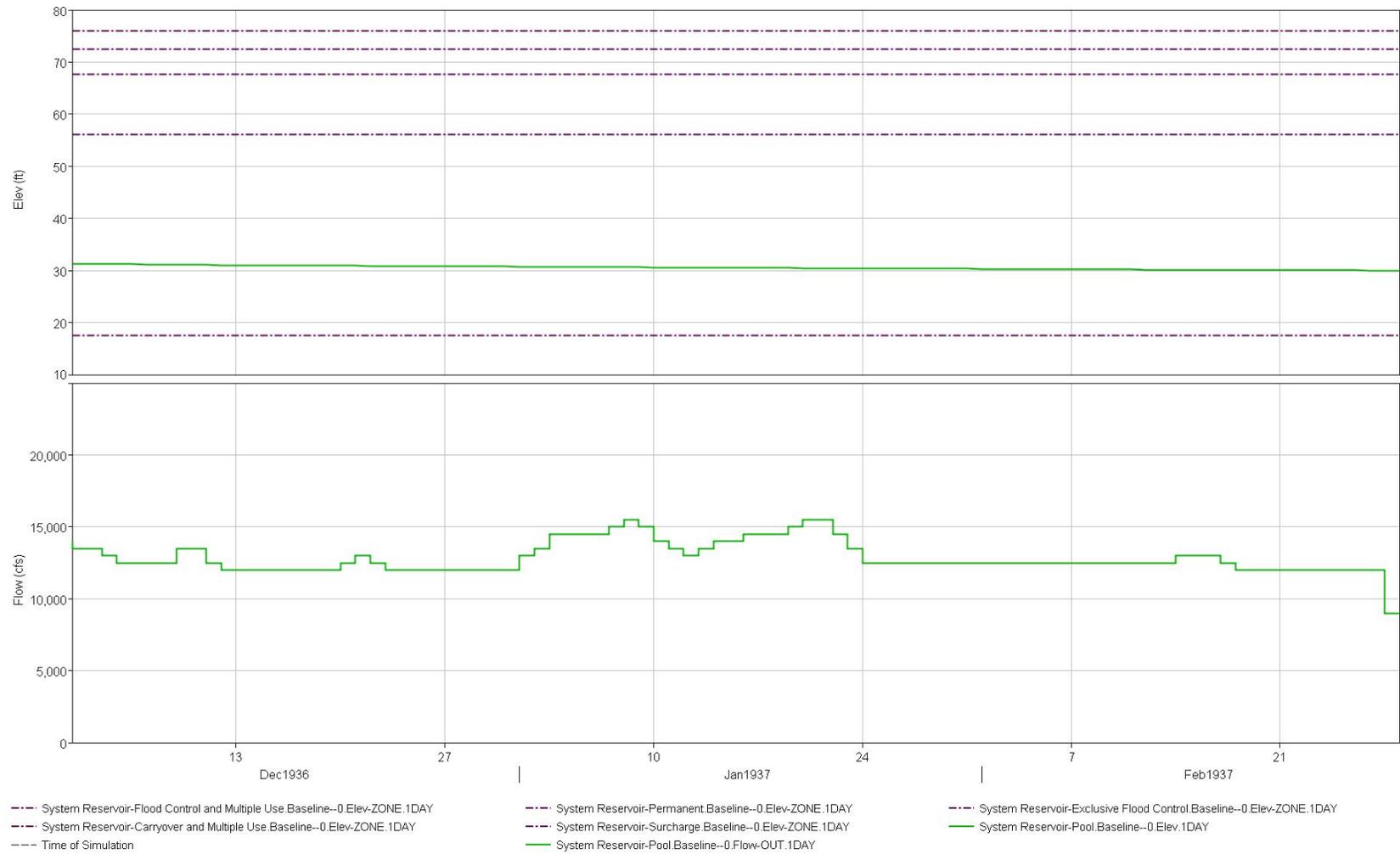
**Figure 5-39: Plot of target locations operating for 18,000 cfs water supply in 1936.**



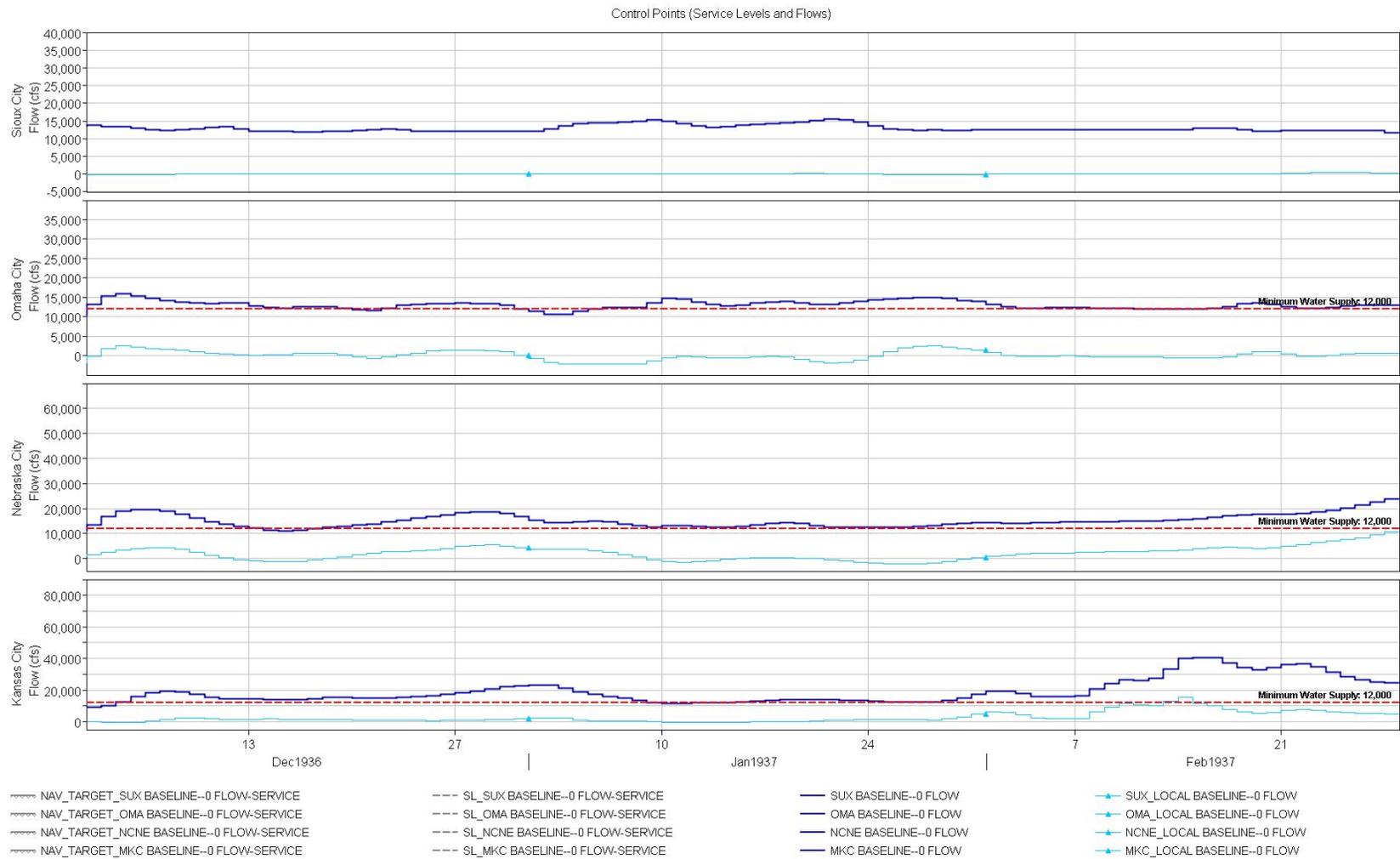
**Figure 5-40: Plot of System releases and System storage in 1936 operating for 9,000 cfs water supply.**



**Figure 5-41: Plot of target locations operating for 9,000 cfs water supply in 1936.**



**Figure 5-42: Plot of System releases and System storage in 1936 operating for 12,000 cfs water supply.**

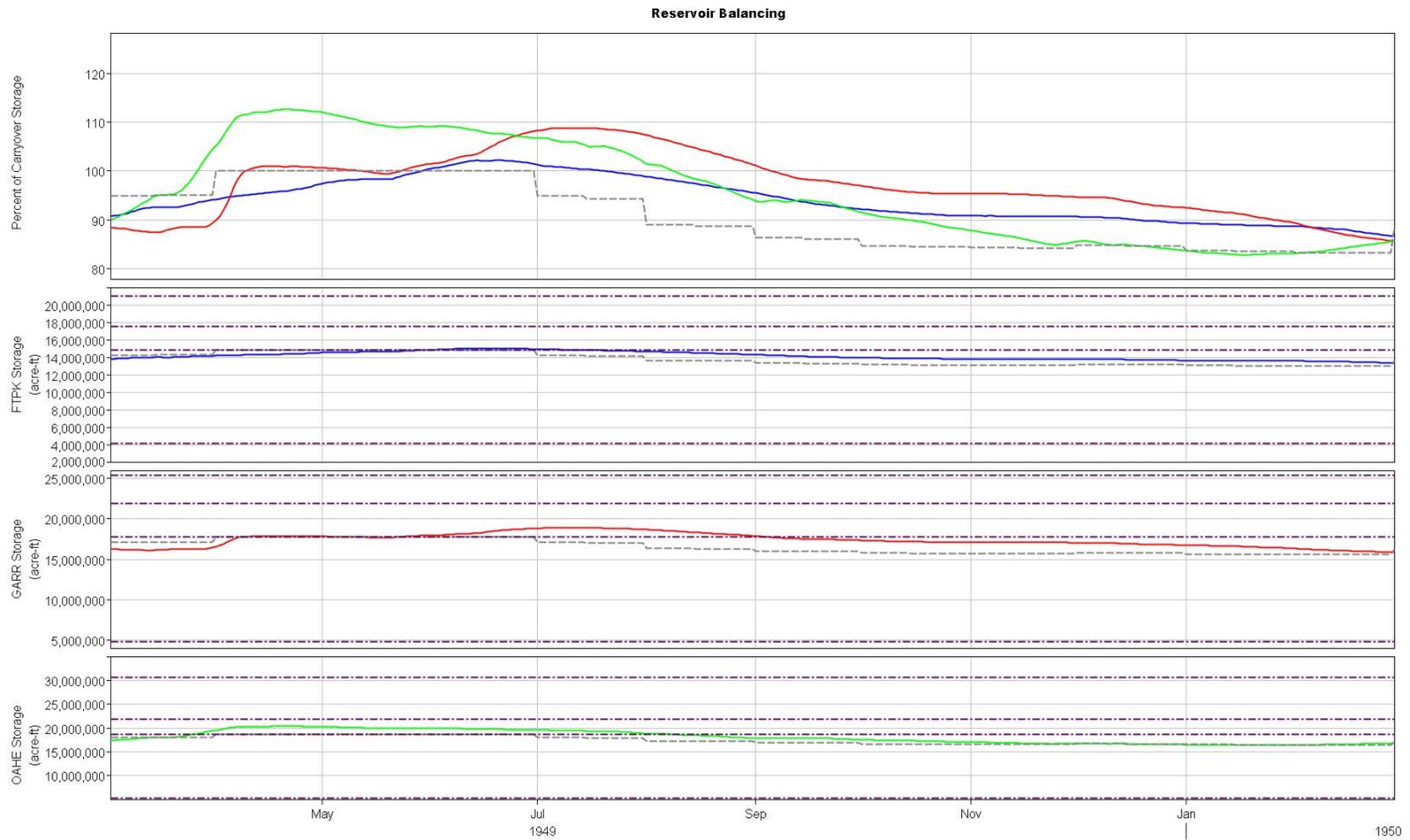


**Figure 5-43: Plot of target locations operating for 12,000 cfs water supply in 1936.**

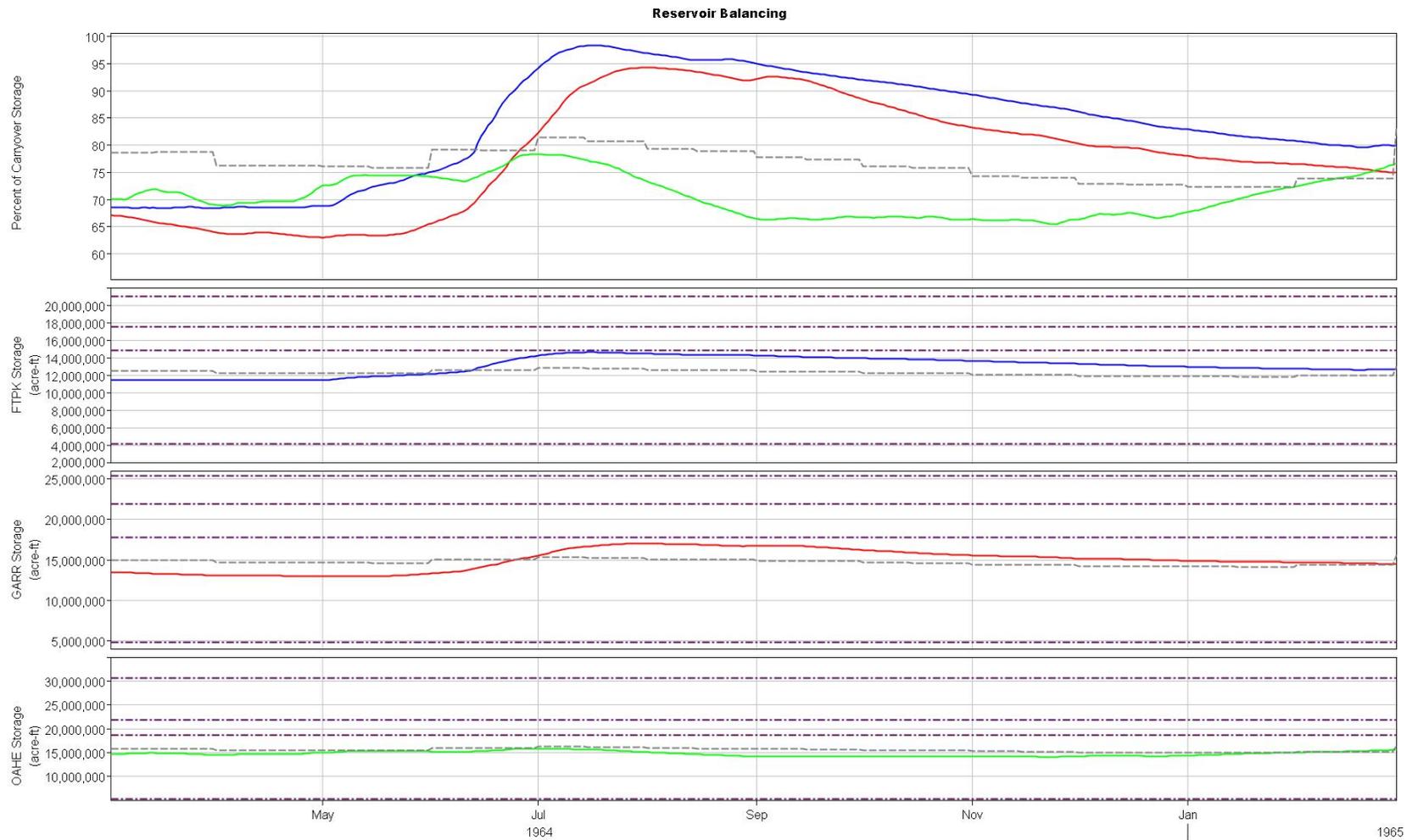
## **5.1.2 System Model**

### **5.1.2.1 System Balancing**

System balancing consists of balancing the carryover storage in the upper three reservoirs: Fort Peck, Garrison, and Oahe. Based on forecasted System releases, System inflows, and current System storage, Fort Peck's and Garrison's releases are adjusted so the percent of Fort Peck's occupied carryover storage is equal to the percent of Garrison's occupied carryover storage and is equal to the percent of Oahe's occupied carryover storage. To achieve balanced System storage, Fort Peck's and Garrison's releases are adjusted up or down while Oahe is allowed to float, meaning that Oahe's storage will move up or down to allow Fort Peck and Garrison to reach their target storages. There are limits to these adjustments; Fort Peck and Garrison follow general release patterns with minimum and maximum releases, so it is possible that Fort Peck and Garrison will not be able to reach their target storages by March 1. Oahe's ability to float is also limited. If Oahe's percent of carryover storage becomes more than 5 percent different than Garrison's, Garrison's releases will then be adjusted to keep Oahe's storage within a reasonable difference. Throughout the period of record, the model correctly attempts to balance System storage based on forecasted System releases, System inflows, and current System storage. Figure 5-44 shows how each reservoir's percent of occupied carryover storage converges to the target storages. Although the percent of occupied carryover storage for each reservoir do not reach the target, the System storage is balanced by March 1. Figure 5-45 shows how minimum and maximum releases at Fort Peck and Garrison can prevent a balanced System. Fort Peck is unable to release enough water during 1964-1965 to lower its storage and increase Garrison's storage so that the System is balanced by March 1.



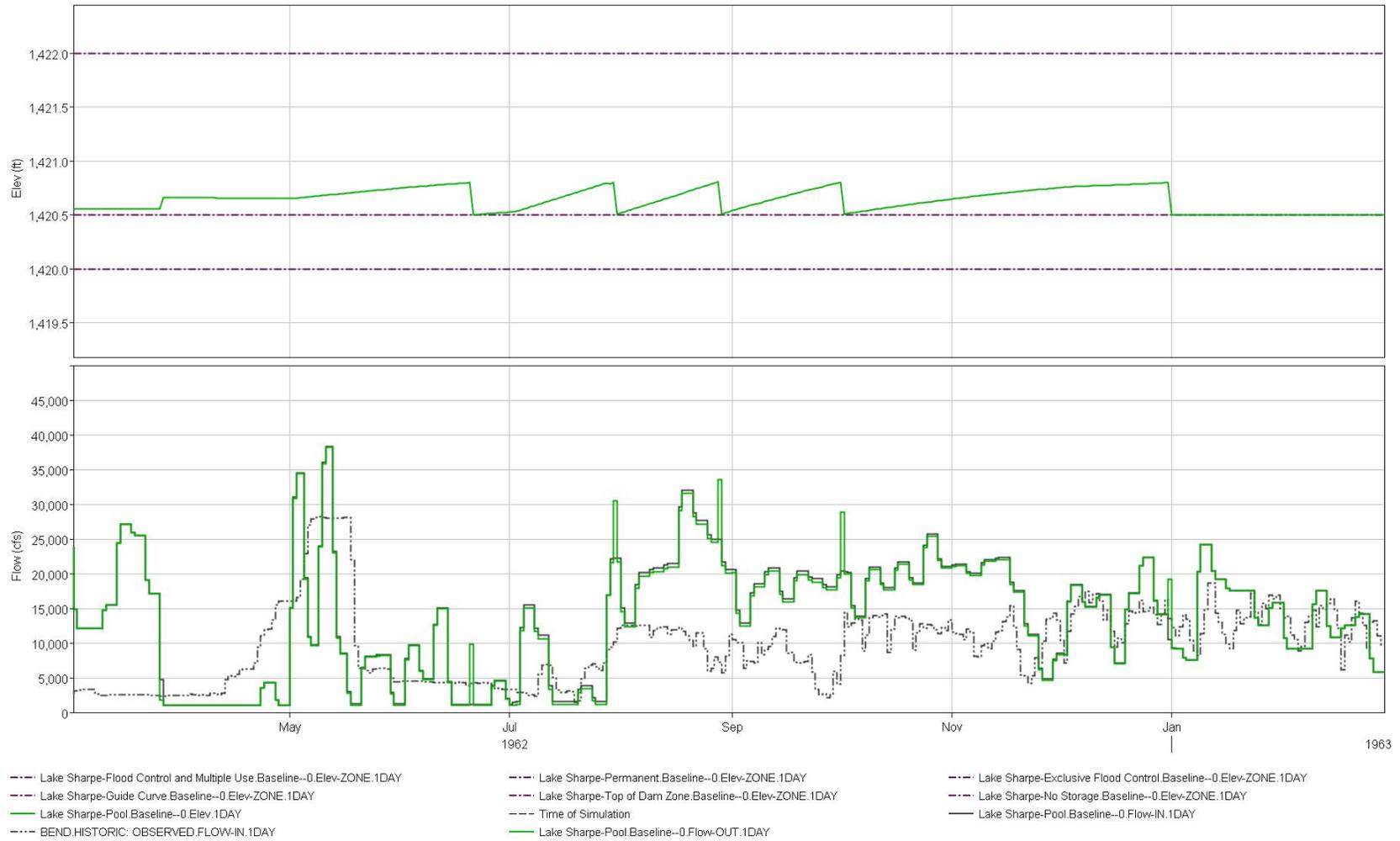
**Figure 5-44: Plot of Fort Peck’s, Garrison’s, and Oahe’s percent of occupied carryover storage and reservoir storages in 1949-1950. The top plot shows Fort Peck’s current percent occupied carryover storage (blue), Garrison’s current percent occupied carryover storage (red), Oahe’s current percent occupied carryover storage (green), and the combined forecasted percent occupied carryover storage. The bottom three plots show each reservoir’s target storage and current storage.**



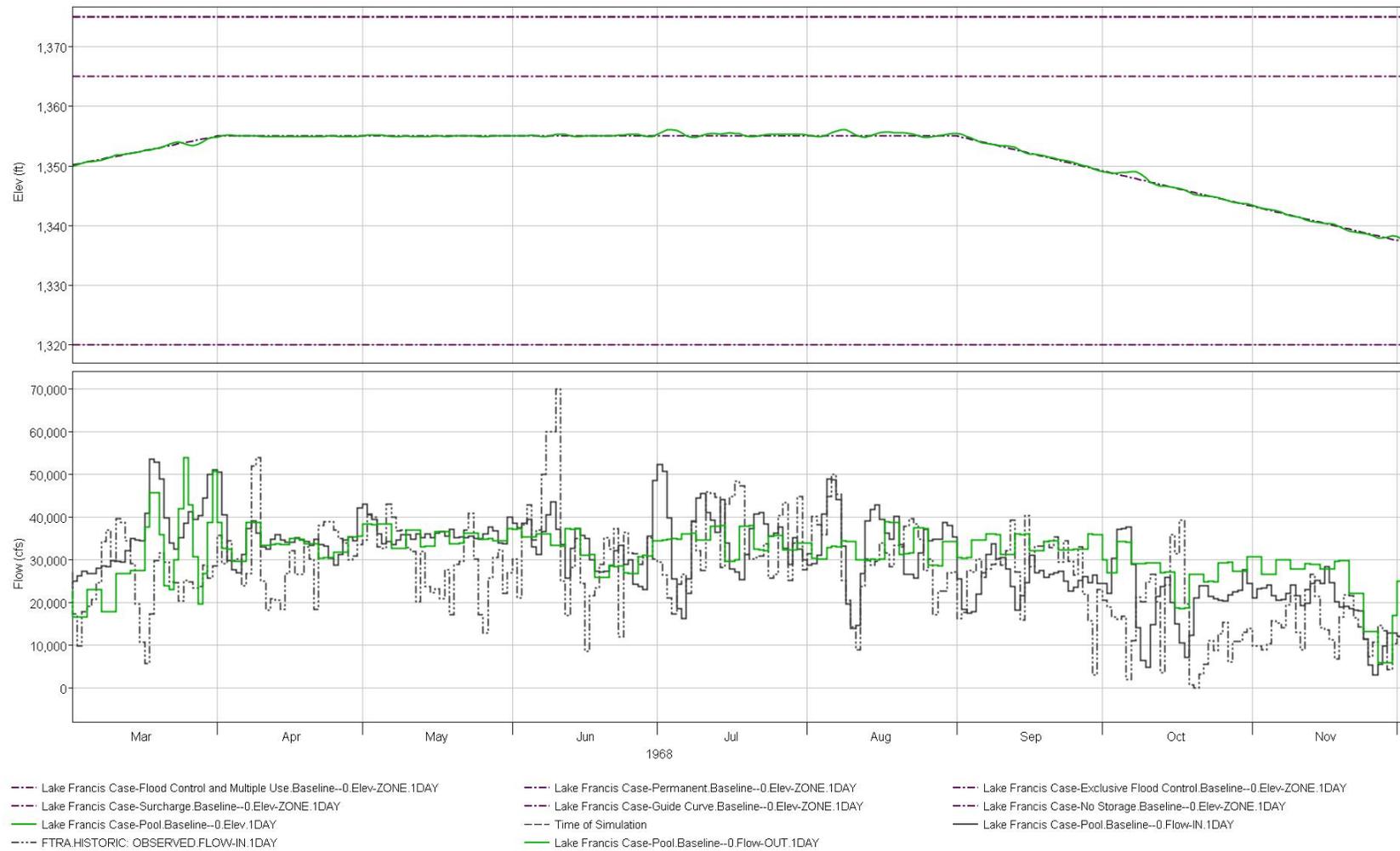
**Figure 5-45: Plot of Fort Peck’s, Garrison’s, and Oahe’s percent of occupied carryover storage and reservoir storages in 1964-1965. The top plot shows Fort Peck’s current percent occupied carryover storage (blue), Garrison’s current percent occupied carryover storage (red), Oahe’s current percent occupied carryover storage (green), and the combined forecasted percent occupied carryover storage. The bottom three plots show each reservoir’s target storage and current storage.**

### **5.1.2.2 Guide Curve Operations**

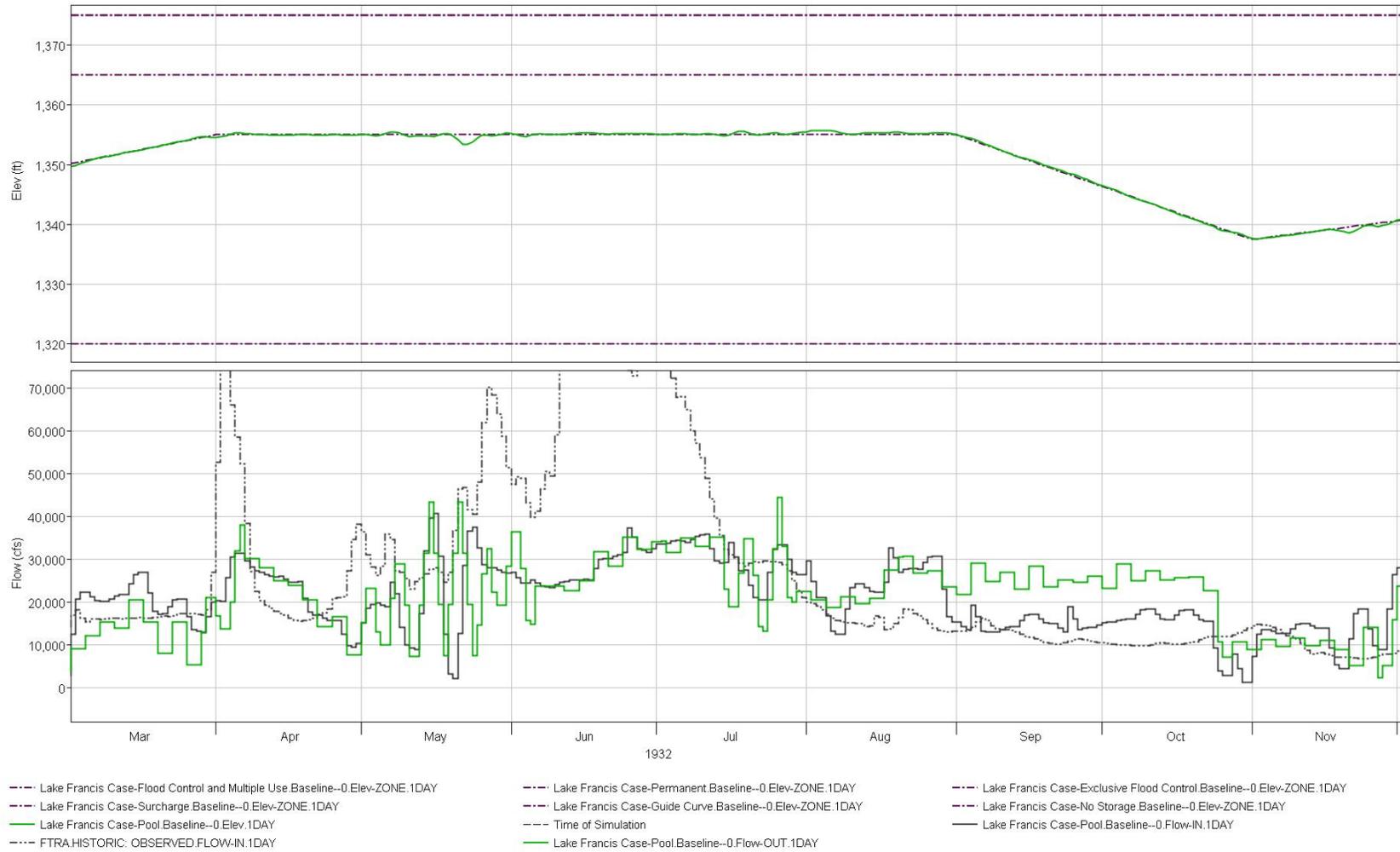
Guide curve operations occur at Big Bend, Fort Randall, and Gavins Point. Big Bend's pool elevation typically fluctuates between 1421.0 and 1420.0, so for modeling purposes, the guide curve is set at 1420.5 ft. Since this is a "run of the river" reservoir, the releases will be similar to inflows as the dam operates to keep a constant pool elevation of 1420.5 ft year round. Figure 5-46 shows the guide curve operations at Big Bend. Fort Randall's guide curve operations are more complex than Big Bend. Fort Randall's pool elevation begins at 1350.0 ft on March 1 and rises to 1355.0 ft by April 1; its pool elevation is held at 1355.0 ft through August. On September 1, Fort Randall begins to draw down its pool reaching a pool elevation of 1337.5 ft on the last day of the navigation season. Figure 5-47 shows Fort Randall's guide curve operations with a navigation end date of December 1 and Figure 5-48 shows Fort Randall's guide operations with a navigation end date of November 1. Gavins Point's guide curve operations keeps the pool elevation within a small band. On March 1, Gavins Point's pool elevation is 1206.0 ft and remains at 1206.0 ft during the summer. On August 1, Gavins Point's pool begins to rise so that it reaches elevation 1207.0 ft by September 1. Gavins Point's pool elevations remains at 1207.0 ft through December when its pool begins to decrease so it reaches an elevation of 1206.0 ft by February 1. Figure 5-49 shows Gavins Point's guide curve operations in 1972.



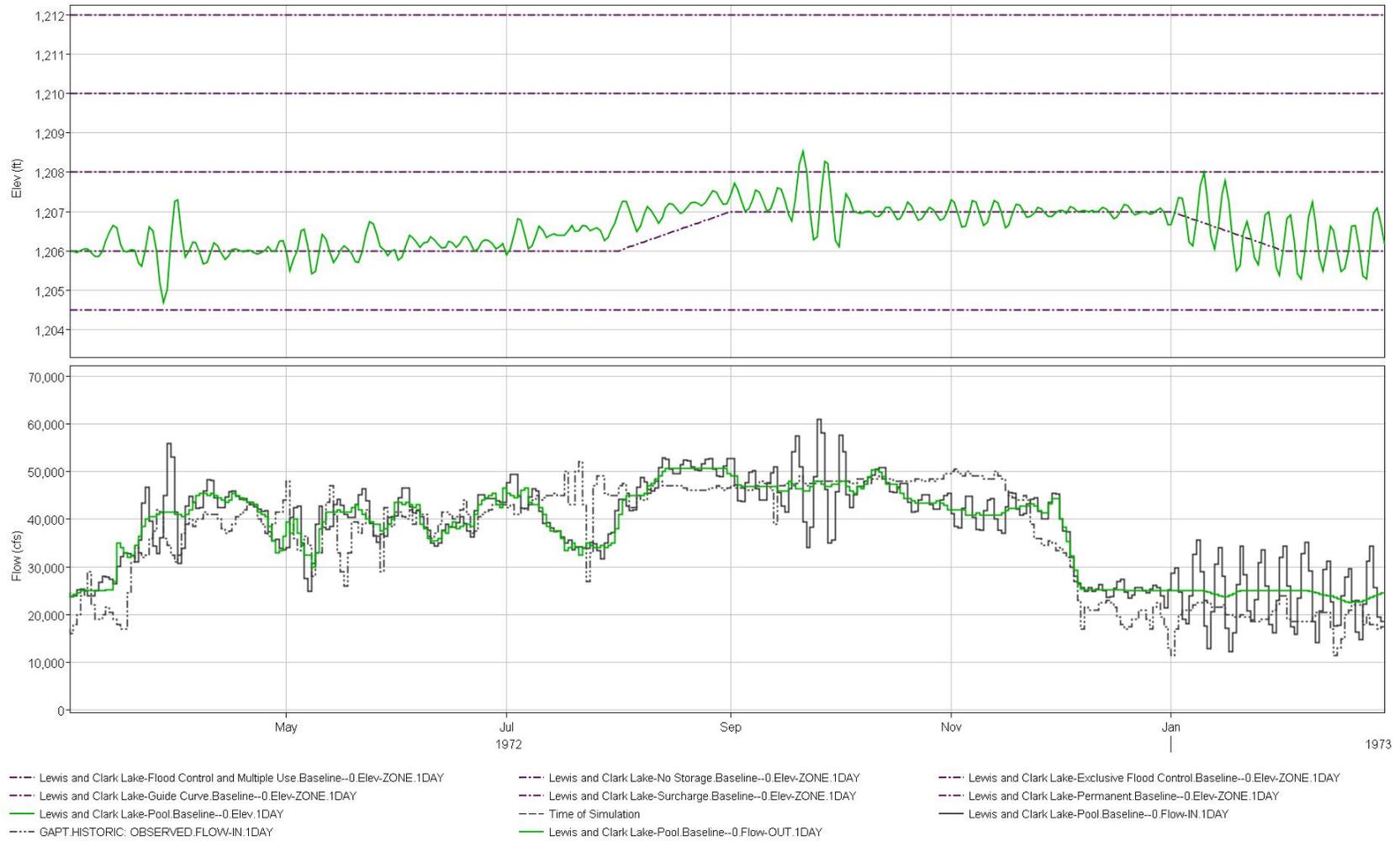
**Figure 5-46: Plot of Big Bend guide curve operations in 1962.**



**Figure 5-47: Plot of Fort Randall guide curve operations with a navigation end date of December 1, 1968.**



**Figure 5-48: Plot of Fort Randall guide curve operations with a navigation end date of November 1, 1932.**

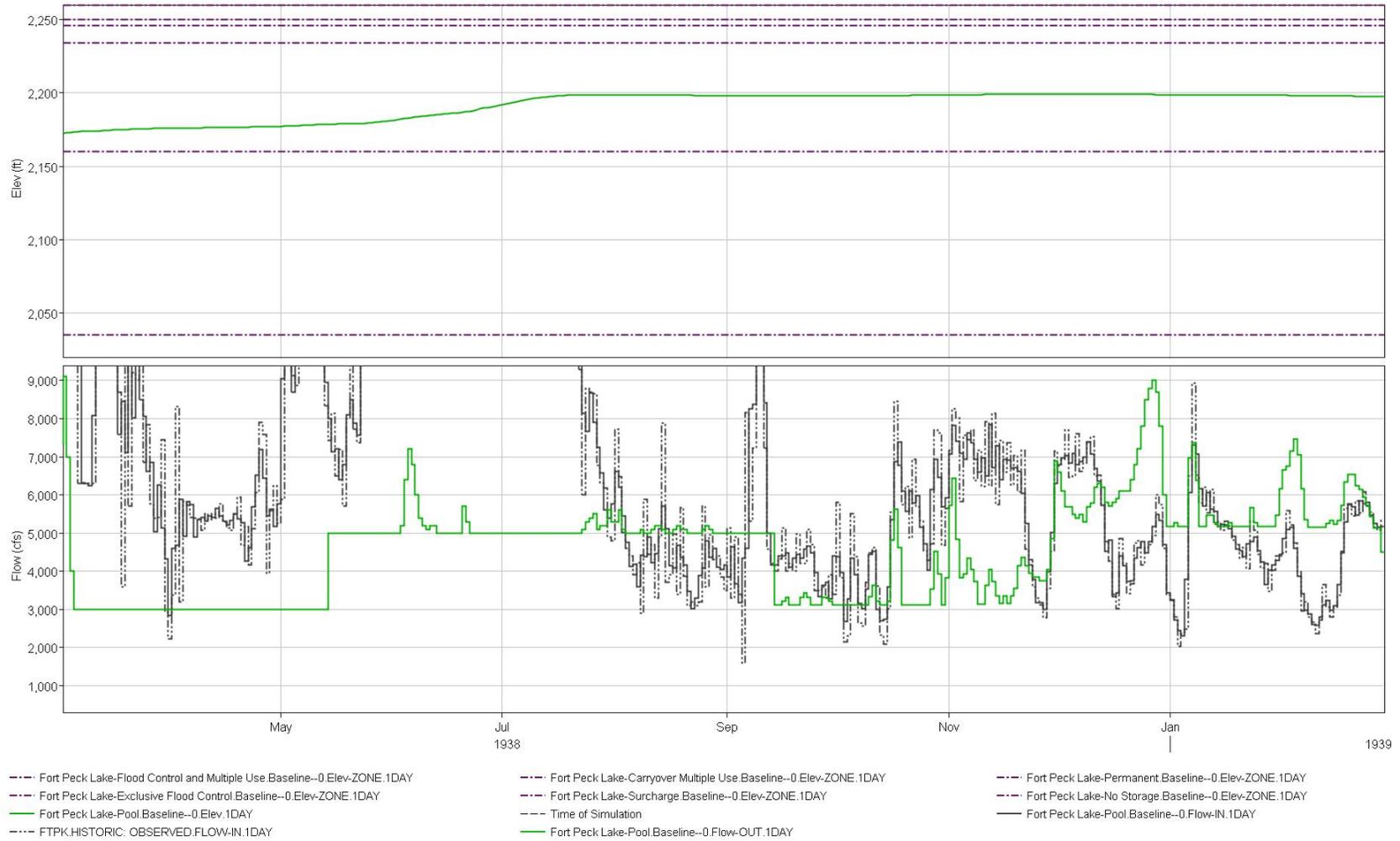


**Figure 5-49: Plot of Gavins Point's guide curve operations in 1972.**

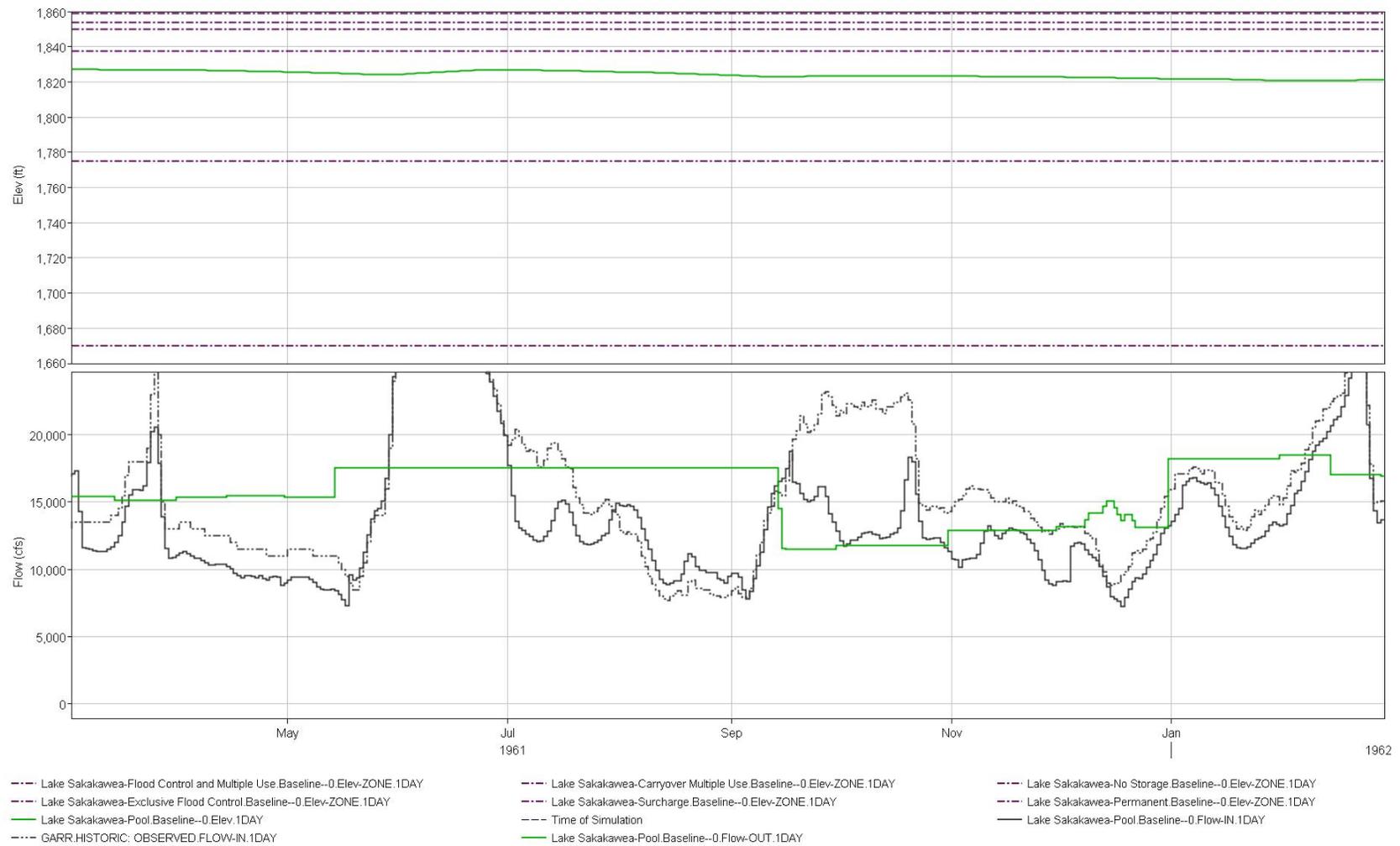
### **5.1.2.3 Upstream Water Supply Requirements**

The upstream water supply scripted rule uses the releases at Fort Peck and Garrison that are calculated in the reservoir balancing scripted rule and checks downstream water supply requirements. Fort Peck's releases are routed downstream to Wolf Point and Culbertson. If 3,000 cfs is not observed at both locations during March – May 14 and September – November, Fort Peck's releases are increased until that criteria is met. If 5,000 cfs is not observed at both locations during May 15 – August and December – February, Fort Peck's releases are increased until that criteria is met. Figure 5-50 shows how Fort Peck's releases are increased above releases specified in the reservoir balancing scripted rule (3,000 cfs and 5,000 cfs) for water supply requirements at Wolf Point and Culbertson in 1938-1939.

Then Garrison's releases are routed downstream to Bismarck. If 10,000 cfs is not observed at Bismarck during March – August, Garrison's release are increased until that criteria is met. If 9,000 cfs is not observed at Bismarck during September – November, Garrison's release are increased until that criteria is met. If 12,000 cfs is not observed at Bismarck during December – February, Garrison's release are increased until that criteria is met. Figure 5-51 shows how Garrison's releases are increased above releases specified in the reservoir balancing scripted rule (13,200 cfs) for water supply requirements at Bismarck in December of 1961.



**Figure 5-50: Plot of Fort Peck's releases operating for water supply in 1938.**



**Figure 5-51: Plot of Garrison’s releases operating for water supply in December 1961.**

## 6 REFERENCES

- Hydrologic Engineering Center. (2007). *HEC-ResSim Reservoir System Simulation User's Manual*. Davis, CA: U.S. Army Corps of Engineers.
- National Weather Service. (1982). *NOAA Technical Report NWS 34 - Mean Monthly, Seasonal, and Annual Pan Evaporation for the United States*. Washington, D.C.: National Weather Service.
- U.S. Army Corps of Engineers. (1987). *EM 1110-2-3600 Management of Water Control Systems*. Washington D.C.: U.S. Army Corps of Engineers.
- U.S. Army Corps of Engineers. (2003). *Upper Mississippi River System Flow Frequency Study - Hydrology and Hydraulics Appendix F*. Omaha, NE: U.S. Army Corps of Engineers.
- U.S. Army Corps of Engineers. (2006). *Missouri River Mainstem Reservoir System Master Water Control Manual*. Omaha, NE: USACE.
- U.S. Army Corps of Engineers. (2007). *Missouri River Mainstem Reservoir System: System Description and Regulation*. Omaha, NE: USACE.
- U.S. Army Corps of Engineers. (2010). *Yellowtail Dam Reallocation Study*. Omaha, NE: U.S. Army Corps of Engineers.
- U.S. Army Corps of Engineers. (2012). *Upper Missouri River Basin Study*. Omaha, NE: U.S. Army Corps of Engineers.
- U.S. Bureau of Reclamation. (2012, 12 21). *Buffalo Bill Dam*. Retrieved 03 12, 2013, from U.S. Bureau of Reclamation: [http://www.usbr.gov/projects/Facility.jsp?fac\\_Name=Buffalo%20Bill%20Dam](http://www.usbr.gov/projects/Facility.jsp?fac_Name=Buffalo%20Bill%20Dam)
- U.S. Bureau of Reclamation. (1999). *Standing Operating Procedures - Volume 1 Canyon Ferry Dam and Reservoir*. Billings, MT: U.S. Bureau of Reclamation.
- U.S. Bureau of Reclamation. (2000). *Standing Operating Procedures - Tiber Dam and Lake Elwell*. Billings, MT: U.S. Bureau of Reclamation.
- U.S. Bureau of Reclamation. (2000). *Standing Operating Procedures - Volume 1 Yellowtail Dam and Yellowtail Afterbay Dam*. Billings, MT: U.S. Bureau of Reclamation.
- U.S. Bureau of Reclamation. (2003). *Standing Operating Procedures - Boysen Dam, Reservoir, and Powerplant*. Billings, MT: U.S. Bureau of Reclamation.
- U.S. Bureau of Reclamation. (2012, 10 3). *Buffalo Bill Reservoir Allocations*. Retrieved 03 12, 2013, from U.S. Bureau of Reclamation: [http://www.usbr.gov/gp/aop/resaloc/buffalo\\_bill.pdf](http://www.usbr.gov/gp/aop/resaloc/buffalo_bill.pdf)

U.S. Geological Survey. (1982). *Guidlines for Determining Flood Flow Frequency Bulletin 19B*. Reston, VA: U.S. Geological Survey.

Western Regional Climate Center. (2008, 12 31). *Period of Record Monthly Climate Summary Yellowtail Dam, Montana*. Retrieved 06 29, 2011, from Western Regional Climate Center: <http://www.wrcc.dri.edu/cgi-bin/cliMAIN.pl?mt9240>

## 7 APPENDIX A – PERTINENT DATA

**Summary of Engineering Data - Missouri River Main Stem System**

<b>Item No.</b>		<b>Fort Peck Dam - Fort Peck Lake</b>	<b>Garrison Dam - Lake Sakakawea</b>	<b>Oahe Dam - Lake Oahe</b>	<b>Big Bend Dam - Lake Sharpe</b>	<b>Fort Randall Dam - Lake Francis Case</b>	<b>Gavins Point Dam - Lewis &amp; Clark Lake</b>	<b>Total</b>	<b>Item No.</b>	<b>Remarks</b>	
1	Location of Dam	Near Glasgow, Montana	Near Garrison, ND	Near Pierre, SD	21 miles upstream Chamberlain, SD	Near Lake Andes, SD	Near Yankton, SD		1	(1) Includes 4,280 square miles of non-contributing areas.	
2	River Mile - 1960 Mileage	Mile 1771.5	Mile 1389.9	Mile 1072.3	Mile 987.4	Mile 880.0	Mile 811.1		2		
3	Total & incremental drainage areas in square miles	57,500	181,400      123,900	243,490      62,090	249,330      5,840	263,480 (1)      14,150	279,480 (1)      16,000		3		
4	Approximate length of full reservoir (in valley miles)	134, ending near Zortman, MT	178, ending near Trenton, ND	231, ending near Bismarck, ND	80, ending near Pierre, SD	107, ending at Big Bend Dam	25, ending near Niobrara, NE	755 miles	4		
5	Shoreline in miles (3)	1520 (elevation 2234)	1340 (elevation 1837.5)	2250 (elevation 1607.5)	200 (elevation 1420)	540 (elevation 1350)	90 (elevation 1204.5)	5,940 miles	5		
6	Average total & incremental inflow in cfs	10,200	25,600      15,400	28,900      3,300	28,900	30,000      1,100	32,000      2,000		6		
7	Max. discharge of record near damsite in cfs	137,000 (June 1953)	348,000 (April 1952)	440,000 (April 1952)	440,000 (April 1952)	447,000 (April 1952)	480,000 (April 1952)		7		
8	Construction started - calendar yr.	1933	1946	1948	1959	1946	1952		8		
9	In operation (4) cal. yr.	1940	1955	1962	1964	1953	1955		9		
	<u>Dam and Embankment</u>										
10	Top of dam elevation in feet msl	2280.5	1875	1660	1440	1395	1234		10	(5) Damming height is height from low water to maximum operating pool. Maximum height is from average streambed to top of dam.	
11	Length of dam in feet	21,026 (excluding spillway)	11,300 (including spillway)	9,300 (excluding spillway)	10,570 (including spillway)	10,700 (including spillway)	8,700 (including spillway)	71,596	11		
12	Damming height in feet (5)	220	180	200	78	140	45	863 feet	12		
13	Maximum height in feet (5)	250.5	210	245	95	165	74		13		
14	Max. base width, total & w/o berms in feet	3500, 2700	3400, 2050	3500, 1500	1200, 700	4300, 1250	850, 450		14		
15	Abutment formations ( under dam & embankment)	Bearpaw shale and glacial fill	Fort Union clay shale	Pierre shale	Pierre shale & Niobrara chalk	Niobrara chalk	Niobrara chalk & Carlile shale		15		
16	Type of fill	Hydraulic & rolled earth fill	Rolled earth filled	Rolled earth fill & shale berms	Rolled earth, shale, chalk fill	Rolled earth fill & chalk berms	Rolled earth & chalk fill		16		
17	Fill quantity, cubic yards	125,628,000	66,500,000	55,000,000 & 37,000,000	17,000,000	28,000,000 & 22,000,000	7,000,000	358,128,000 cu. yds	17		
18	Volume of concrete (cubic yards)	1,200,000	1,500,000	1,045,000	540,000	961,000	308,000	5,554,000 cu. yds.	18		
19	Date of Closure	24 June 1937	15 April 1953	3 August 1958	24 July 1963	20 July 1952	31 July 1955		19		
	<u>Spillway Data</u>										
20	Location	Right bank - remote	Left bank - adjacent	Right bank - remote	Left bank - adjacent	Left bank - adjacent	Right bank - adjacent		20		(8) Length from upstream face of outlet or to spiral case.
21	Crest elevation in feet msl	2225	1825	1596.5	1385	1346	1180		21		
22	Width (including piers) in feet	820 gated	1336 gated	456 gated	376 gated	1000 gated	664 gated		22		(9) Based on 8th year (1961) of drought drawdown (From study 8-83-1985).
23	No., size and types of gates	16 - 40' x 25' vertical lift gates	28 - 40' x 29' Tainter	8 - 50' x 23.5' Tainter	8 - 40' x 38' Tainter	21 - 40' x 29' Tainter	14 - 40' x 30' Tainter		23		
24	Design discharge capacity, cfs	275,000 at elev 2253.3	827,000 at elev 1858.5	304,000 at elev 1644.4	390,000 at elev 1433.6	633,000 at elev 1379.8	584,000 at elev 1221.4		24		
25	Discharge capacity at maximum operating pool in cfs	230,000	660,000	80,000	270,000	508,000	345,000		25	(10) Affected by level of Lake	

	<u>Reservoir Data (6)</u>																Francis case. Applicable to
26	Max. operating pool elev & area	2250 msl	245,000 acres	1854 msl	383,000 acres	1620 msl	386,000 acres	1423 msl	61,000 acres	1375 msl	102,000 acres	1210 msl	29,000 acres	1,207,000 acres	26		pool at elevation 1350.
27	Max. normal op pool elev & area	2246 msl	240,000 acres	1850 msl	365,000 acres	1617 msl	362,000 acres	1422 msl	60,000 acres	1365 msl	94,000 acres	1208 msl	25,000 acres	1,146,000 acres	27		
28	Base flood control elev & area	2234 msl	211,000 acres	1837.5 msl	308,000 acres	1607.5 msl	311,000 acres	1420 msl	57,000 acres	1350 msl	76,000 acres	1204.5 msl	21,000 acres	985,000 acres	28	(11)	Spillway crest.
29	Min. op. pool elev. & area	2160 msl	89,000 acres	1775 msl	125,000 acres	1540 msl	115,000 acres	1415 msl	50,000 acres	1320 msl	36,000 acres	1204.5 msl	21,000 acres	437,000 acres	29	(12)	1967-2013 Average
	<u>Storage allocation &amp; capacity</u>																(13)
30	Exclusive flood control	2250-2246	971,000 a.f.	1854-1850	1,495,000 a.f.	1620-1617	1,107,000 a.f.	1423-1422	61,000 a.f.	1375-1365	986,000 a.f.	1210-1208	54,000 a.f.	4,673,000 a.f.	30		Source: Annual Report on Civil Works Activities of the Corps of Engineers. Extract
31	Flood control & multiple use	2246-2234	2,704,000 a.f.	1850-1837.5	4,211,000 a.f.	1617-1607.5	3,208,000 a.f.	1422-1420	118,000 a.f.	1365-1350	1,306,000 a.f.	1208-1204.5	79,000 a.f.	11,625,000 a.f.	31		Report Fiscal Year 1999.
32	Carryover multiple use	2234-2160	10,700,000 a.f.	1837.5-1775	12,951,000 a.f.	1607.5-1540	13,353,000 a.f.			1350-1320	1,532,000 a.f.			38,536,000 a.f.	32		
33	Permanent	2160-2030	4,088,000 a.f.	1775-1673	4,794,000 a.f.	1540-1415	5,315,000 a.f.	1420-1345	1,631,000 a.f.	1320-1240	1,469,000 a.f.	1204.5-1160	295,000 a.f.	17,582,000 a.f.	33		
34	Gross	2250-2030	18,463,000 a.f.	1854-1673	23,451,000 a.f.	1620-1415	22,983,000 a.f.	1423-1345	1,810,000 a.f.	1375-1240	5,293,000 a.f.	1210-1160	428,000 a.f.	72,416,000 a.f.	34		
35	Reservoir filling initiated	November 1937		December 1953		August 1958		November 1963		January 1953		August 1955			35		
36	Initially reached min. operating pool	27 May 1942		7 August 1955		3 April 1962		25 March 1964		24 November 1953		22 December 1955			36		
37	Estimated annual sediment inflow	17,200 a.f.	1070 yrs.	21,600 a.f.	1080 yrs.	14,100 a.f.	1630	3,400 a.f.	530	15,800 a.f.	340 yrs.	2,500 a.f.	170 yrs.	73,900 a.f.	37		
	<u>Outlet Works Data</u>																
38	Location	Right bank		Right Bank		Right Bank		None (7)		Left Bank		None (7)			38		
39	Number and size of conduits	2 - 24' 8" diameter (nos. 3 & 4)		1 - 26' dia. and 2 - 22' dia.		6 - 19.75' dia. upstream, 18.25' dia. Downstream				4 - 22' diameter					39		
40	Length of conduits in feet (8)	No. 3 - 6,615, No. 4 - 7,240		1529		3496 to 3659				1013					40		
41	No., size, and type of service gates	1 - 28' dia. cylindrical gate 6 ports, 7.6' x 8.5' high (net opening) in each control shaft		1 - 18' x 24.5' Tainter gate per conduit for fine regulation		1 - 13' x 22' per conduit, vertical lift, 4 cable suspension and 2 hydraulic suspension (fine regulation)				2 - 11' x 23' per conduit, vertical lift, cable suspension					41		
42	Entrance invert elevation (msl)	2095		1672		1425		1385 (11)		1229		1180 (11)			42		
43	Avg. discharge capacity per conduit & total	Elev. 2250 22,500 cfs - 45,000 cfs		Elev. 1854 30,400 cfs - 98,000 cfs		Elev. 1620 18,500 cfs - 111,000 cfs				Elev 1375 32,000 cfs - 128,000 cfs					43		
44	Present tailwater elevation (ft msl)	2032-2036 5,000 - 35,000 cfs		1669-1677 15,000- 60,000 cfs		1422-1427 20,000-55,000 cfs		1351-1355(10) 25,000-100,000 cfs		1228-1237 10,000-60,000 cfs		1153-1161 15,000-60,000 cfs			44		

<u>Power Facilities and Data</u>									
45	Avg. gross head avail in feet (14)	194	161	174	70	117	48	764 feet	45
46	Number and size of conduits	No. 1-24'8" dia., No. 2-22'4" dia.	5 - 29' dia., 25' penstocks	7 - 24' dia., imbedded penstocks	None: direct intake	8 - 28' dia., 22' penstocks	None: direct intake		46
47	Length of conduits in feet (8)	No. 1 - 5,653, No. 2 - 6,355	1829	From 3,280 to 4,005		1074		55,083	47
48	Surge tanks	PH#1: 3-40' dia., PH#2: 2-65' dia.	65' dia. - 2 per penstock	70' dia., 2 per penstock	None	59' dia, 2 per alternate penstock	None		48
49	No., type and speed of turbines	5 Francis, PH#1-2: 128.5 rpm, 1-164 rpm, PH#2-2: 128.6 rpm	5 Francis, 90 rpm	7 Francis, 100 rpm	8 Fixed blade, 81.8 rpm	8 Francis, 85.7 rpm	3 Kaplan, 75 rpm	36 units	49
50	Disch. cap. at rated head in cfs	PH#1, units 1&3 170', 2-140' 8,800 cfs, PH#2-4&5 170'-7, 200 cfs	150' 41,000 cfs	185' 54,000 cfs	67' 103,000 cfs	112' 44,500 cfs	48' 36,000 cfs		50
51	Generator nameplate rating in kW	1&3: 43,500; 2: 18,250; 4&5: 40,000	3 - 121,600, 2 - 109,250	112,290	3 - 67,276, 5 - 58,500	40,000	44,100		51
52	Plant capacity in kW	185,250	583,300	786,030	494,320	320,000	132,300	2,501,200 kw	52
53	Dependable capacity in kW (9)	181,000	388,000	534,000	497,000	293,000	74,000	1,967,000 kw	53
54	Avg annual energy, million kWh (12)	1,046	2,251	2,625	981	1,726	725	9,354 million kWh	54
55	Initial generation, first and last unit	July 1943 - June 1961	January 1956 - October 1960	April 1962 - June 1963	October 1964 - July 1966	March 1954 - January 1956	September 1956 - January 1957	July 1943 - July 1966	55
56	Estimated cost September 1999 Completed project (13)	\$158,428,000	\$305,274,000	\$346,521,000	\$107,498,000	\$199,066,000	\$49,617,000	\$1,166,404,000	56

Corps of Engineers, U.S. Army  
Compiled by  
Missouri River Division  
August 2014

## 8 APPENDIX B – ROUTING PARAMETER DETERMINATION SUMMARY

To determine routing parameters for use in the HEC-ResSim model more quickly, a HEC-HMS routing model was setup to test four different routing methods, and an HEC-ResSim model was used to test the Coefficient Routing Method. HEC-DSSVue has limited routing capabilities and was not used to route flows. All three HEC hydrologic modeling software programs have different available routing methods; Table 8-1 below is a summary of the routing methods available in each program.

**Table 8-1: Routing methods in hydrologic HEC programs.**

HEC-ResSim	HEC-HMS	HEC-DSSVue
Coefficient Routing	N/A	N/A
N/A	Kinematic Wave	N/A
N/A	Lag	N/A
Modified Puls	Modified Puls	Modified Puls
Muskingum	Muskingum	Muskingum
Muskingum-Cunge 8-pt	Muskingum-Cunge 8-pt	N/A
Muskingum-Cunge		
Prismatic	N/A	N/A
SSARR	N/A	N/A
N/A	Straddle-Stagger	Straddle-Stagger
Working R&D	N/A	N/A
Variable Lag & K	N/A	N/A

### 8.1 COEFFICIENT ROUTING

The Coefficient Routing parameters from the USACE Daily Routing Model (DRM) were used to help determine initial routing parameters for some of the methods. The Coefficient Routing parameters in the DRM were based on statistical discharge correlations from 1/1/1967 to 12/31/1994. The routing parameters from the DRM are shown in Table 8-2 on the following page. The A0 value, or intercept, is zero for all reaches in the DRM because that model already included local flow and only translation was necessary. A1 through A4 are coefficients, and must add to 1 for each reach. A1 is the coefficient to be applied to today's (d) flow. A2 is the coefficient to be applied to yesterday's (d-1) flow, or the flow lagged by 1 day. A3 is the coefficient to be applied to the flow from 2 days ago (d-2), or the flow lagged by 2 days. Since HMS does not have Coefficient Routing or any comparable method, the DRM Coefficient Routing parameters were tested using HEC-ResSim.

**Table 8-2: DRM coefficient routing parameters.**

Reach	A0	A1 (d)	A2 (d-1)	A3 (d-2)	A4 (d-3)
FTPK_GARR	0	0.237	0.444	0.319	0
GARR_OAHE	0	0.057	0.503	0.44	0
OAHE_BEND	0	0.766	0.234	0	0
BEND_FTRA	0	0.647	0.353	0	0
FTRA_GAPT	0	0.005	0.637	0.358	0
GAPT_SUX	0	0.17532	0.53734	0.28734	0
SUX_OMA	0	0.16794	0.72176	0.1103	0
OMA_NCNE	0	0.5879	0.4121	0	0
NCNE_RUNE	0	0.58837	0.41163	0	0
RUNE_STJ	0	0.77547	0.22453	0	0
STJ_MKC	0	0.42647	0.44863	0.1249	0
MKC_WVMO	0	0.47605	0.52395	0	0
WVMO_BNMO	0	0.3542	0.61748	0.02832	0
BNMO_HEMO	0	0.38146	0.43382	0.18472	0
HEMO_STL	0	0.22208	0.77792	0	0
FTPK_WPMT	0	0.10283	0.65925	0.23792	0
WPMT_CLMT	0	0.18943	0.55198	0.25858	0
CLMT_WSN	0	0.0847	0.41119	0.50411	0
GARR_BIS	0	0.05704	0.50308	0.43988	0

## 8.2 STRADDLE-STAGGER ROUTING

The four routing methods tested in HMS were Straddle-Stagger, Muskingum, Muskingum-Cunge, and Modified Puls. Each method was calibrated based on major events during the period of record (POR). The POR was January 1, 1898 to October 1, 2011. For each reach during calibration, the two major event years of 1952 and 2011 were examined. At least two other major peak years noticed in each reach comparable to those events were also examined. After calibrating the routing parameters, the four methods were compared to each other and the observed POR. Priority was given to the more recent events during calibration and routing method comparison, since the final model will require routing parameters that are representative of current conditions.

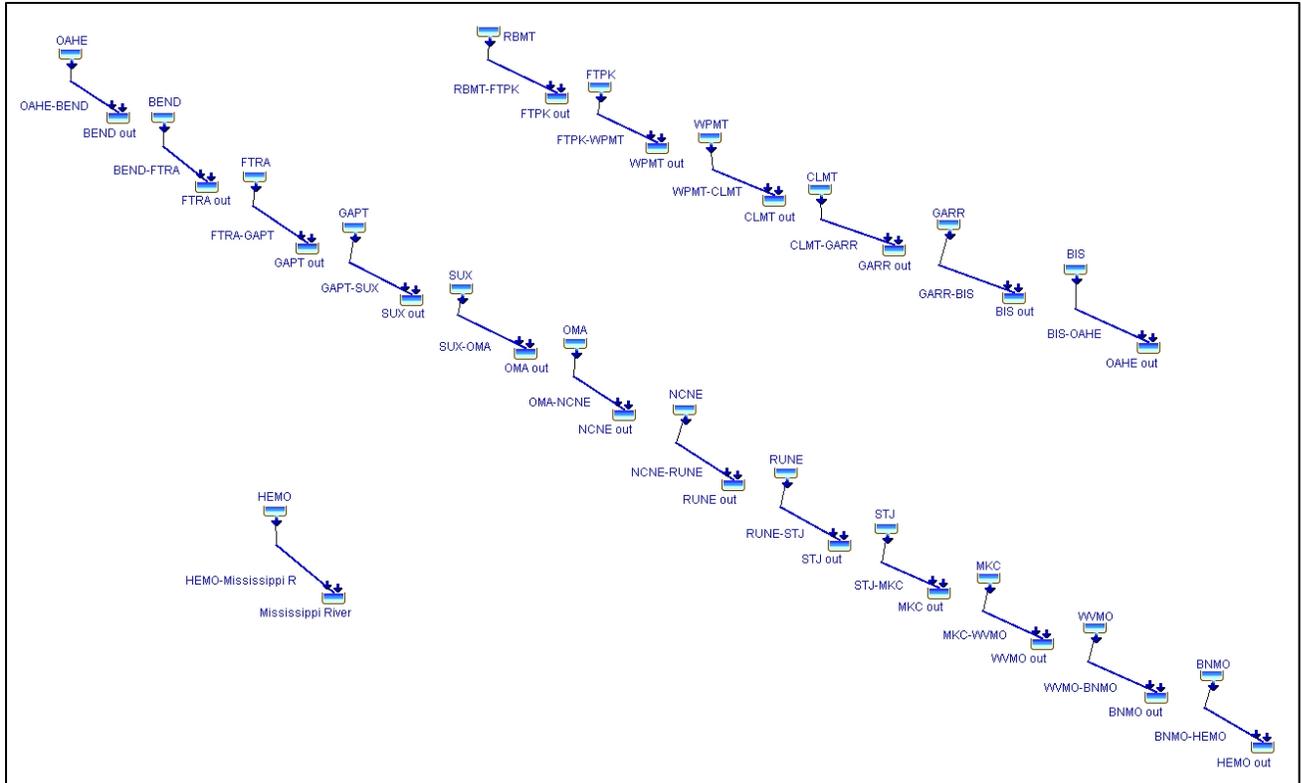
Straddle-Stagger is a progressive average-lag routing method in which equal weight is applied to each day's flow for the straddle duration. For the Straddle-Stagger method, the initial lag (Stagger) was determined by the day with the highest coefficient from the DRM routing method for each reach. For example: The A1 (d) column for the RUNE-STJ reach had the highest coefficient for that reach, so a zero day lag was used initially for that reach. The initial duration (Straddle) was determined by the equation:

$$\text{Straddle} = \text{Stagger} + 1 \text{ day.}$$

The straddle value is the number of days that the flow is averaged over. For example, a 1-day stagger with a 1-day straddle would apply the total weight of 1.0 to the d-1 timestep. A 1-day stagger with a 2-day straddle applies equal weights of 0.5 to the d-1 and d-2 timesteps. A 1-day stagger with a 3-day straddle applies equal weights of 0.33 to the d, d-1, and d-2 timesteps. The lag and durations were varied for some reaches during calibration. The duration cannot be less than the lag. The Straddle-Stagger method in HMS has hourly input values. However, only whole day increments were used since the computation and data input time-step of the final ResSim model will be daily. The calibrated Straddle-Stagger routing parameters, along with the equivalent coefficient routing parameters, are shown in Table 8-3. The HMS basin schematic used for Straddle-Stagger routing is shown in Figure 8-1.

**Table 8-3: Calibrated Straddle-Stagger and corresponding Coefficient routing parameters.**

Reach	Lag (day)	Duration (day)	A0	A1 (d)	A2 (d-1)	A3 (d-2)	A4 (d-3)
RBMT_FTPK	1	1	0	0	1	0	0
FTP_K_WPMT	1	2	0	0	0.5	0.5	0
WPMT_CLMT	1	1	0	0	1	0	0
CLMT_GARR	1	2	0	0	0.5	0.5	0
GARR_BIS	1	2	0	0	0.5	0.5	0
BIS_OAHE	0	0	0	1	0	0	0
OAHE_BEND	0	0	0	1	0	0	0
BEND_FTRA	0	0	0	1	0	0	0
FTRA_GAPT	1	2	0	0	0.5	0.5	0
GAPT_SUX	1	2	0	0	0.5	0.5	0
SUX_OMA	1	2	0	0	0.5	0.5	0
OMA_NCNE	1	1	0	0	1	0	0
NCNE_RUNE	1	2	0	0	0.5	0.5	0
RUNE_STJ	0	0	0	1	0	0	0
STJ_MKC	1	2	0	0	0.5	0.5	0
MKC_WVMO	1	2	0	0	0.5	0.5	0
WVMO_BNMO	1	2	0	0	0.5	0.5	0
BNMO_HEMO	1	2	0	0	0.5	0.5	0
HEMO_MR-Mississippi	1	2	0	0	0.5	0.5	0



**Figure 8-1: Straddle-Stagger and Muskingum HMS basin schematic.**

### 8.3 MUSKINGUM ROUTING

For the Muskingum Routing method, the final calibrated lag values from the Straddle-Stagger method were used as the initial Lag (K) values. However, HMS does not allow Muskingum routing reaches with K values of zero. To force the model to compute, zeros were replaced with lags of 1 day. With the exception of the zero lag routing reaches, the initial lag values used for the Muskingum routing reaches produced results that matched the timing of the observed events.

The number of steps, or subreaches, in the Muskingum Routing method is approximated by the equation:

$$\# \text{ Subreaches} = K/\Delta t,$$

Where K is the lag in days and  $\Delta t$  is the computation interval in days. Since the computation time-step that will be used in the final ResSim model is 1 day, and most reaches have a lag of 1 day, only 1 subreach is required.

The Muskingum Routing X parameter is a coefficient determined or verified during calibration. The value X can vary anywhere between zero and 0.5. According to the HMS Technical Manual, X is typically near zero for channels with mild slopes and lots of overbank flow. An X coefficient of zero produces hydrograph results that are considerably smoother and flatter than the Straddle-Stagger routing results. The X coefficient is typically near 0.5 for well-defined

channels with steeper slopes and minimal out of bank flows. An X coefficient of 0.5 produces the most peaked hydrograph flows possible with the Muskingum routing method, and results similar to the Straddle-Stagger routing method. With these guidelines in mind, X values closer to 0.5 seem most logical for the Missouri River main channel. However, five different X values were tested on all reaches using the Muskingum routing method: 0.1, 0.2, 0.3, 0.4, and 0.5. These results were compared to the observed events during calibration. The final Muskingum Routing parameters selected are shown in Table 8-4. The HMS basin schematic for the Muskingum routing was the same as it was for the Straddle-Stagger routing (Figure 1). It should be noted that reaches most accurately modeled with a zero day lag cannot be modeled using Muskingum Routing, and are denoted in Table 8-4 with an “N/A.” If Muskingum Routing were selected as the final routing method, these reaches should be modeled in ResSim using null, or no, routing.

**Table 8-4: Calibrated Musking routing parameters.**

Reach	Muskingum Final		
	K (hr)	X	Subreaches
RBMT_FTPK	24	0.38	1
FTP_K_WPMT	24	0.45	1
WPMT_CLMT	24	0.5	1
CLMT_GARR	24	0.28	1
GARR_BIS	24	0.3	1
BIS_OAHE	N/A	N/A	N/A
OAHE_BEND	N/A	N/A	N/A
BEND_FTRA	N/A	N/A	N/A
FTRA_GAPT	24	0.38	1
GAPT_SUX	24	0.38	1
SUX_OMA	24	0.3	1
OMA_NCNE	24	0.45	1
NCNE_RUNE	24	0.4	1
RUNE_STJ	N/A	N/A	N/A
STJ_MKC	24	0.4	1
MKC_WVMO	24	0.4	1
WVMO_BNMO	24	0.28	1
BNMO_HEMO	24	0.4	1
HEMO_MR-Mississippi	24	0.4	1

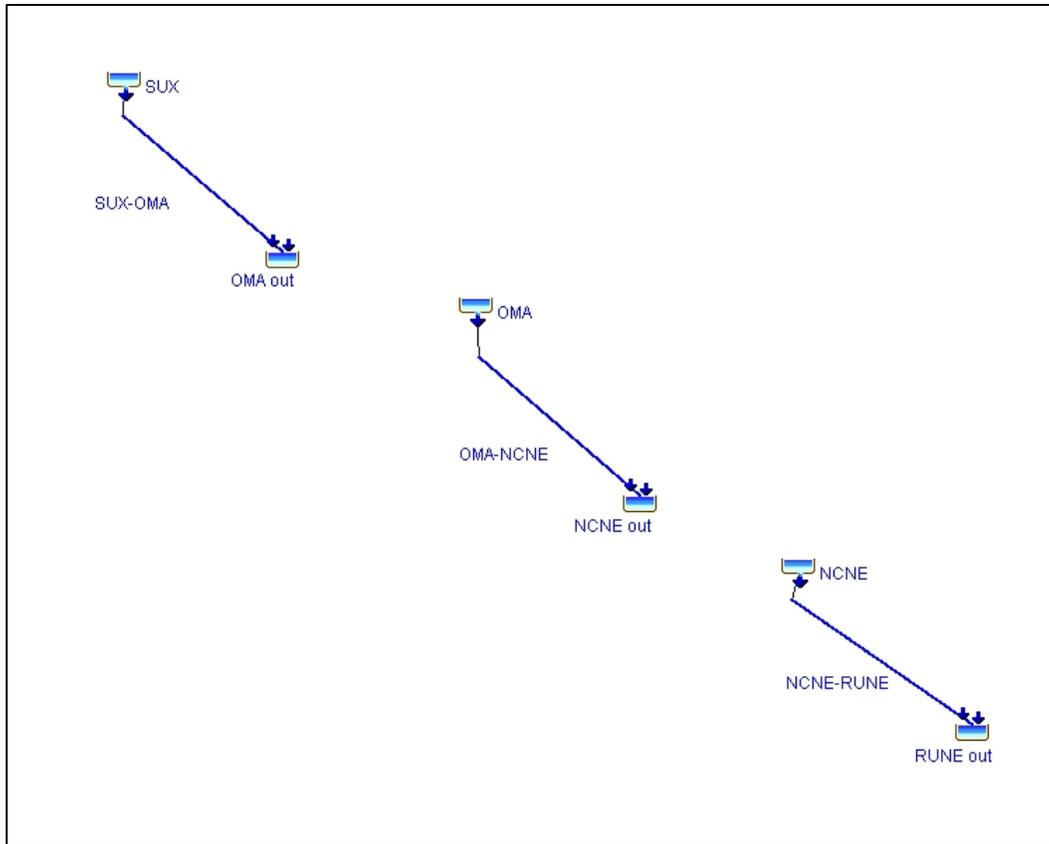
#### 8.4 MUSKINGUM-CUNGE ROUTING

For the Muskingum-Cunge and Modified Puls Routing methods, only reaches downstream of Sioux City and upstream of Rulo were modeled. These routing methods require cross-sections, Manning’s n values and storage-discharge curves, best obtained from existing calibrated HEC-RAS models. The Omaha District currently has RAS models for these reaches only.

For Muskingum-Cunge routing, the 8-point cross section was selected. Cross sections in the RAS model were much more complex and had to be reduced to 8-point cross sections while conserving the total flow area. Each hydraulic modeling reach in the RAS model also had many cross sections. One representative cross section had to be selected for each hydrologic modeling reach in the HMS model. This was done by calculating the average cross section flow area for each reach and selecting a cross section with the corresponding flow area that was not located in the immediate vicinity of a bridge/road. The average main channel, left overbank, and right overbank Manning's n values were determined from the cross-sections in each RAS reach. The lengths and slopes for each Muskingum-Cunge routing reach were also obtained from the RAS model. The Muskingum-Cunge routing parameters were not changed during calibration since the parameters from the RAS model had already been calibrated and the HMS results closely matched the observed events. The final Muskingum-Cunge routing parameters are shown in Table 8-5. The HMS basin schematic used for Muskingum-Cunge and Modified Puls routing is shown in Figure 8-2.

**Table 8-5: Muskingum-Cunge routing parameters.**

<b>Reach</b>	<b>Length (ft)</b>	<b>Slope (ft/ft)</b>	<b>Manning's n</b>	<b>Left n</b>	<b>Right n</b>
SUX_OMA	609363	0.000172	0.023	0.056	0.058
OMA_NCNE	276081	0.000171	0.025	0.065	0.059
NCNE_RUNE	339739	0.000206	0.025	0.055	0.058



**Figure 8-2: Muskingum-Cunge and Modified Puls HMS basin schematic.**

## 8.5 MODIFIED PULS ROUTING

The Modified Puls Routing method in HMS requires storage-discharge curves and the number of subreaches for each reach as input parameters. Storage-discharge curves from a calibrated RAS model had previously been used in an HMS model during 2011 flood forecasting. The reaches used in the 2011 flood forecasting HMS model were shorter and had to be combined for use in this HMS Routing model. The Sioux City to Decatur, Decatur to Blair, and Blair to Omaha storage-discharge curves were combined to create the SUX-OMA storage-discharge curve. The Omaha to Plattsmouth and Plattsmouth to Nebraska City storage-discharge curves were combined into the OMA-NCNE storage-discharge curve. The Nebraska City to Brownville and Brownville to Rulo storage-discharge curves were combined into the NCNE-RUNE storage-discharge curve. The number of subreaches in each reach is determined using the same procedure as the Muskingum Routing method, and had previously been determined to be 1 subreach for each of these three reaches. The storage-discharge curves were not modified during calibration, since the curves had been obtained from a calibrated RAS model. The storage-discharge curves are shown in Table 8-6.

**Table 8-6: Modified Puls routing storage-discharge curves.**

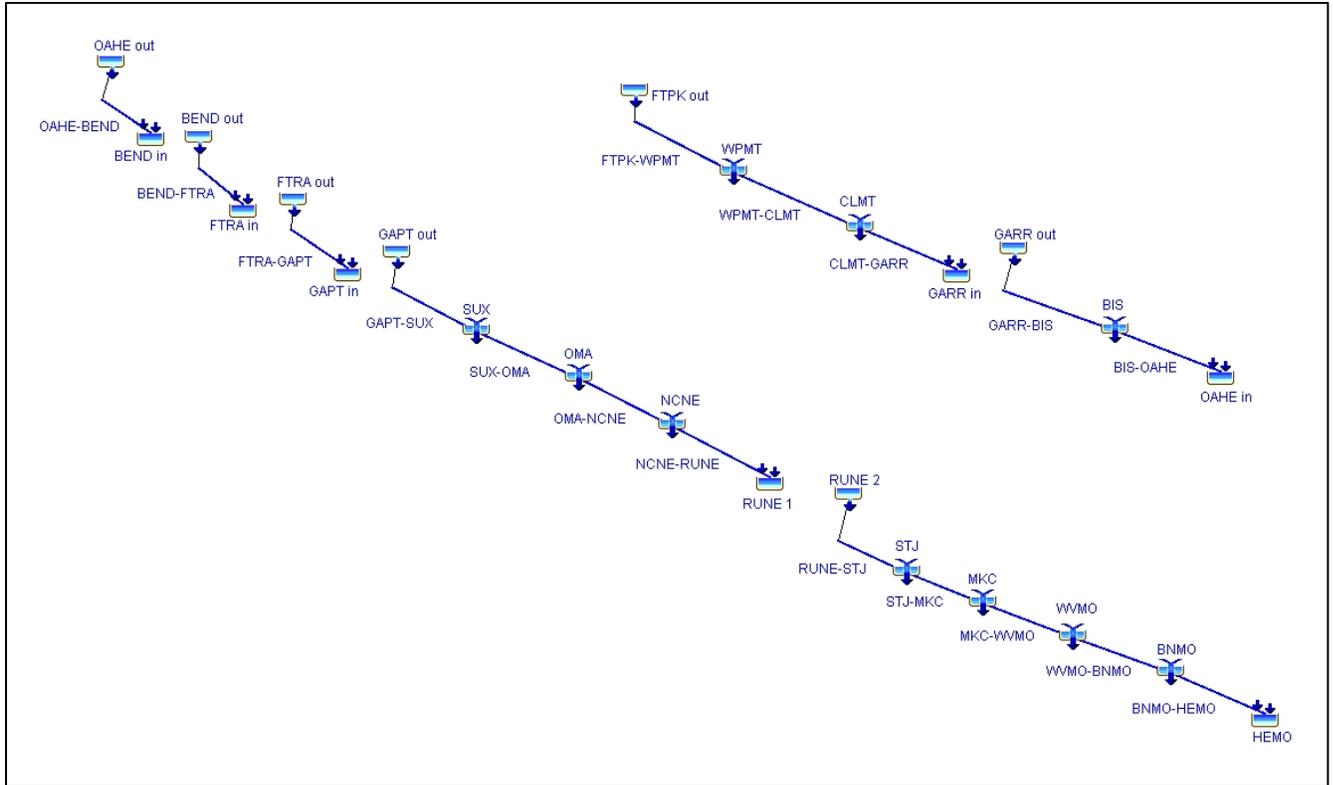
SUX-OMA		OMA-NCNE		NCNE-RUNE	
Storage (ac-ft)	Discharge (cfs)	Storage (ac-ft)	Discharge (cfs)	Storage (ac-ft)	Discharge (cfs)
0	0	0	0	0	0
167,208	50,000	70,843	50,000	121,963	50,000
211,082	60,000	80,652	60,000	182,006	60,000
276,447	70,000	92,197	70,000	272,336	70,000
368,533	80,000	107,949	80,000	362,845	80,000
490,166	90,000	128,270	90,000	450,979	90,000
647,438	100,000	151,478	100,000	535,653	100,000
843,532	110,000	176,494	110,000	617,771	110,000
1,061,784	120,000	198,216	120,000	662,627	120,000
1,289,901	130,000	218,697	130,000	706,682	130,000
1,544,669	140,000	237,981	140,000	758,253	140,000
1,806,854	150,000	258,650	150,000	800,555	150,000
2,091,718	160,000	278,438	160,000	838,997	160,000
2,382,440	170,000	300,742	170,000	887,171	170,000
2,696,430	180,000	319,279	180,000	924,584	180,000
2,995,799	190,000	337,718	190,000	965,073	190,000
3,277,063	200,000	354,551	200,000	1,002,559	200,000
3,491,316	210,000	372,652	210,000	1,037,098	210,000
3,675,907	220,000	388,961	220,000	1,069,014	220,000
3,823,026	230,000	404,992	230,000	1,101,622	230,000
3,966,847	240,000	420,696	240,000	1,135,661	240,000
4,094,193	250,000	436,594	250,000	1,168,479	250,000
4,214,866	260,000	452,352	260,000	1,202,194	260,000
4,322,512	270,000	467,764	270,000	1,236,466	270,000

## 8.6 RESULTS

Of the four routing methods tested using HMS, which does not include the coefficient method, the Straddle-Stagger Routing method was best. The Straddle-Stagger routing results closely approximate the timing of the observed events. The resulting peak flows do not always match the observed event peak flows, but this is mainly because the incremental local or ungaged flow between the upstream and downstream gages has not been factored into the model at this point. The Muskingum Routing results are very similar to the Straddle-Stagger Routing results, and also approximate the timing of the observed events fairly well. However, the Straddle-Stagger method produces better results in a couple locations. The Straddle-Stagger routing method is also less complicated and should be better understood by all users of the final model, since various sources and previous models have attempted to determine the lag or travel times between mainstem reservoirs and reaches. The Muskingum-Cunge Routing results approximated the observed events fairly well and were very similar to the Straddle-Stagger results also. However, the Straddle-Stagger results approximated some events more closely

than the Muskingum-Cunge results. The Muskingum-Cunge results also had slightly delayed timing for some events compared to the observed data. The Modified Puls Routing results do not approximate the timing of the observed events as closely as the other routing methods. The hydrographs produced by this routing method are considerably flatter and delayed compared to the observed events. Comparison hydrograph results for the 3 reaches that tested all four routing methods are shown in the Figures in Section 8.6.1 for select events. The black dotted lines are the observed events (Flow-Observed), the blue lines are the Modified Puls routing (Mod Puls), the purple lines are the Muskingum-Cunge routing (Musk Cunge), the green lines are the Muskingum routing (Musk-Final), and the dashed red lines are the Straddle-Stagger routing (SS-Final).

A composite HMS routing model using the final Straddle-Stagger routing parameters was constructed to test the overall timing of the routing method. One continuous routing model could not be constructed due to the effect of reservoir routing at upstream locations. The timing of peaks for inflow hydrographs is often different than the timing of the peaks for outflow hydrographs at reservoirs. For this reason, the model was broken up at reservoir locations. The reach from GAPT to HEMO was also broken up at RUNE, to better observe the timing effects of the routing parameters. When the GAPT outflow hydrograph is routed all the way to HEMO without any additional flow added between those locations, the difference in modeled and observed discharge is so great that it becomes difficult to locate and compare the timing of the peaks. For this reason, the observed RUNE flow was routed downstream to HEMO instead. After reviewing the results of the composite HMS routing model, none of the Straddle-Stagger Routing parameters were changed. The timing produced by the previously determined parameters was considered acceptable. The composite routing HMS basin schematic is shown in Figure 8-3. Section 8.6.2 contains Straddle-Stagger routing result hydrographs versus observed hydrographs for the 2011 event for each reach. Results for the complete POR are stored in HEC-DSSVue and are best viewed there.



**Figure 8-3: Straddle-Stagger composite routing HMS basin schematic.**

A simplified routing model similar to the HMS model was constructed in ResSim to compare the DRM Coefficient routing parameters to the final Straddle-Stagger routing parameters. The structure of the ResSim model is identical to the structure of the HMS model shown in Figure 8-1. Four reaches that will be used in the final ResSim model do not have Coefficient routing parameters defined in the DRM: RBMT-FTPK, CLMT-GARR, and BIS-OAHE. These reaches use the final Straddle-Stagger routing parameters converted to the Coefficient routing method, and are identical to the Straddle-Stagger results in Section 8.6.2. The Coefficient routing results compared to the Straddle-Stagger routing results and the observed discharges for all other reaches during the 2011 event are shown in Section 8.6.3. Coefficient routing results are in blue, Straddle-Stagger routing results are in red, and the observed discharges are in black. After comparing the two methods, the Coefficient routing method was selected as the final method for use in the ResSim model. For the majority of the reaches, the Coefficient routing results and the Straddle-Stagger routing results are nearly identical. However, the timing of the Coefficient routing results is slightly better on a few reaches (NCNE-RUNE, STJ-MKC, and MKC-WVMO). The final routing parameters for use in the ResSim model are shown in Table 8-7.

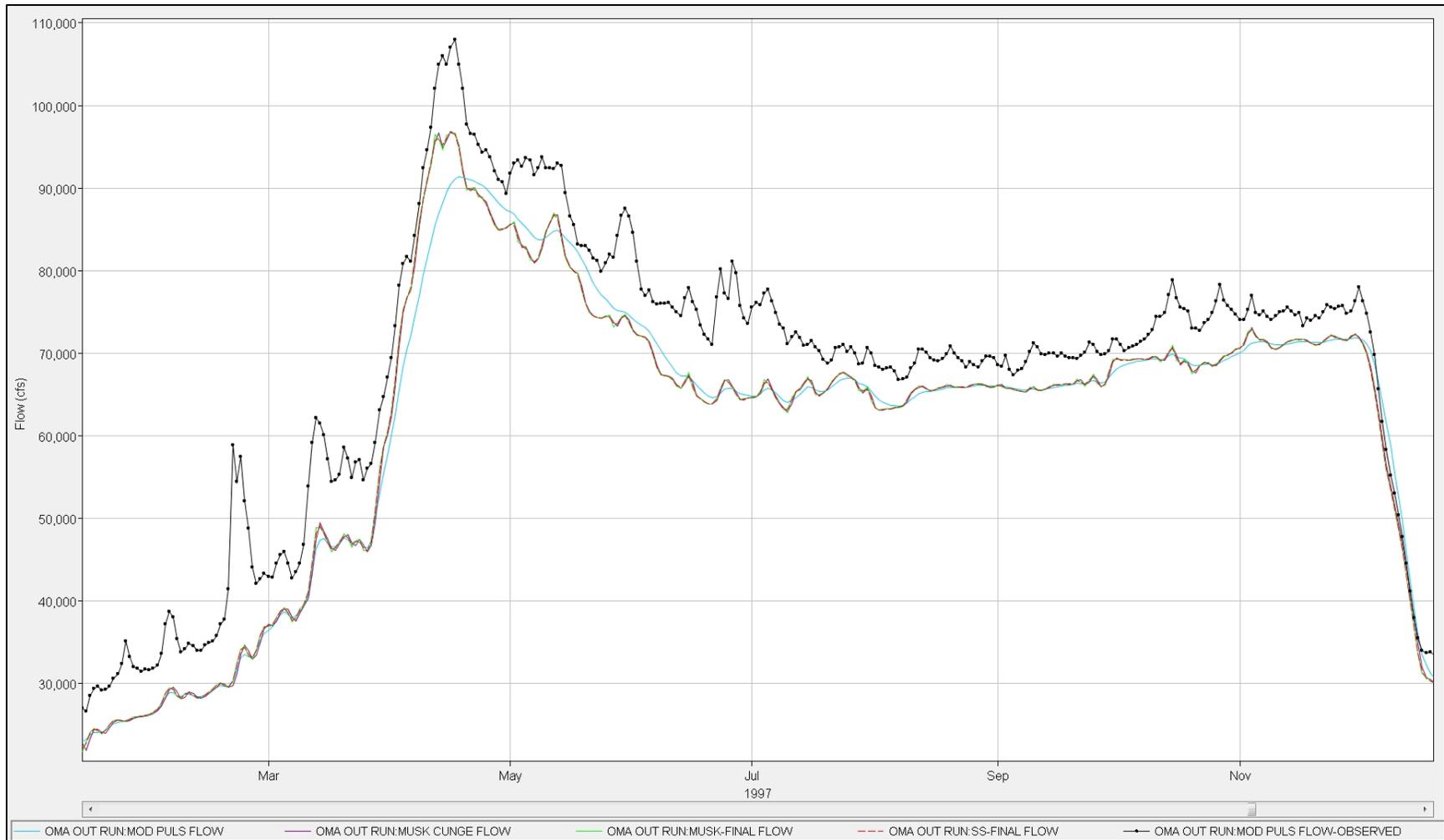
**Table 8-7: Final routing parameters.**

<b>Reach</b>	<b>A1 (d)</b>	<b>A2 (d-1)</b>	<b>A3 (d-2)</b>
RBMT_FTPK	0	1	0
FTPK_WPMT	0.10283	0.65925	0.23792
WPMT_CLMT	0.18943	0.55198	0.25858
CLMT_GARR	0	0.5	0.5
GARR_BIS	0.05704	0.50308	0.43988
BIS_OAHE	1	0	0
OAHE_BEND	0.766	0.234	0
BEND_FTRA	0.647	0.353	0
FTRA_GAPT	0.005	0.637	0.358
GAPT_SUX	0.17532	0.53734	0.28734
SUX_OMA	0.16794	0.72176	0.1103
OMA_NCNE	0.5879	0.4121	0
NCNE_RUNE	0.58837	0.41163	0
RUNE_STJ	0.77547	0.22453	0
STJ_MKC	0.42647	0.44863	0.1249
MKC_WVMO	0.47605	0.52395	0
WVMO_BNMO	0.3542	0.61748	0.02832
BNMO_HEMO	0.38146	0.43382	0.18472
HEMO_MISL	0.22208	0.77792	0

**8.6.1 Modified Puls, Muskingum-Cunge, Muskingum, and Straddle-Stagger Routing Method Comparison Plots**



**Figure 8-4: SUX-OMA 2011 event.**



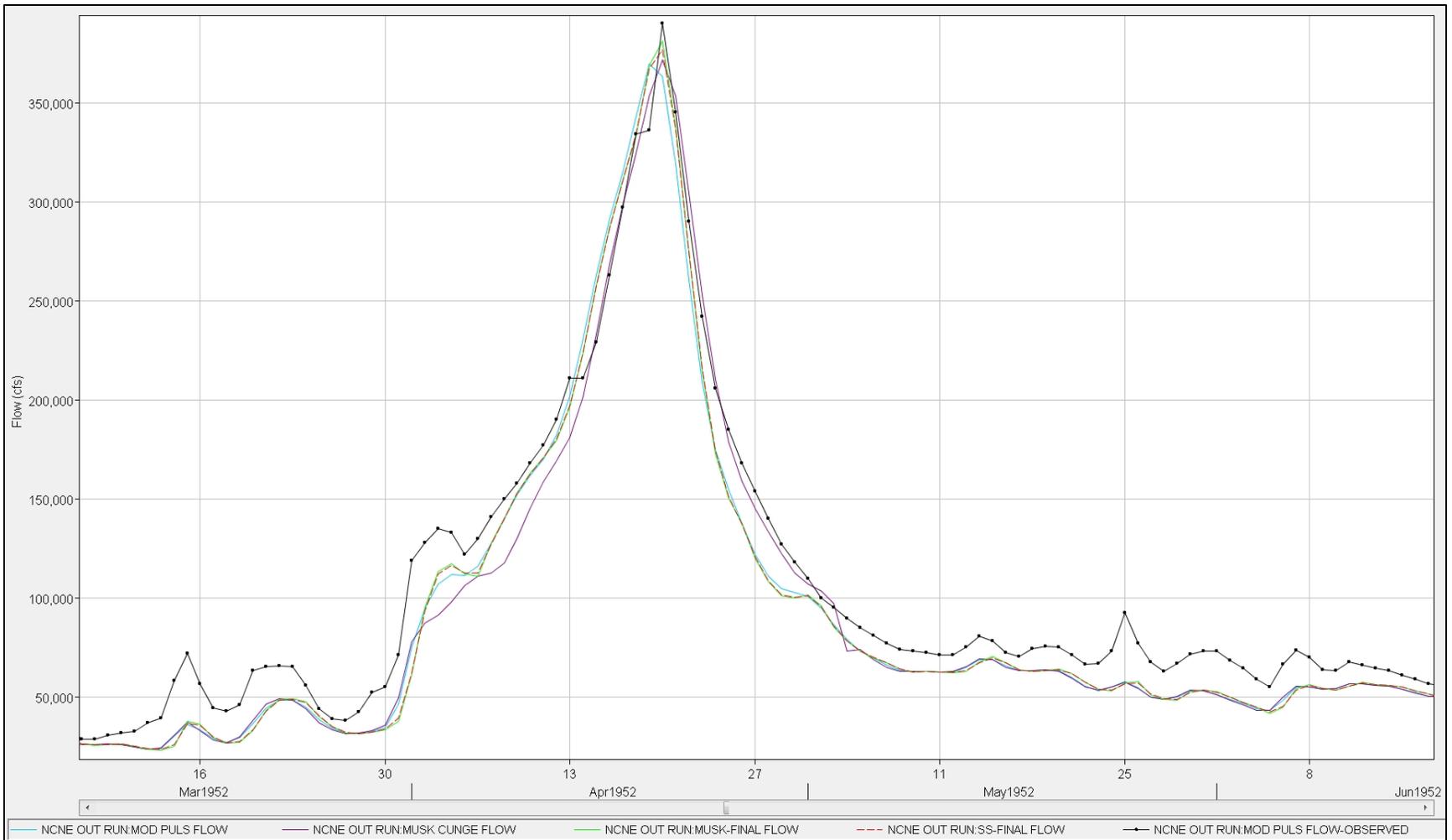
**Figure 8-5: SUX-OMA 1997 event.**



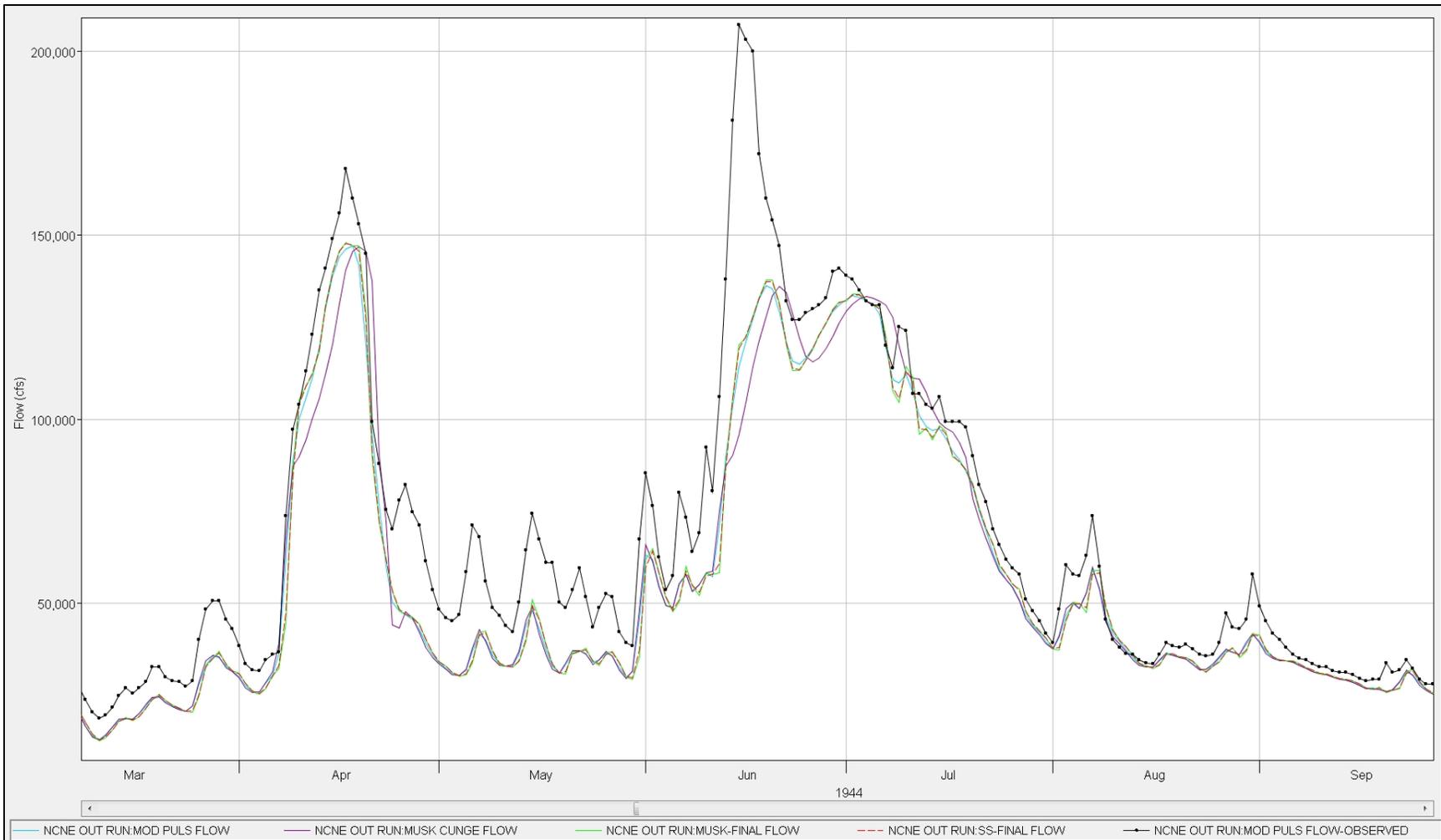
**Figure 8-6: SUX-OMA 1993 Event.**



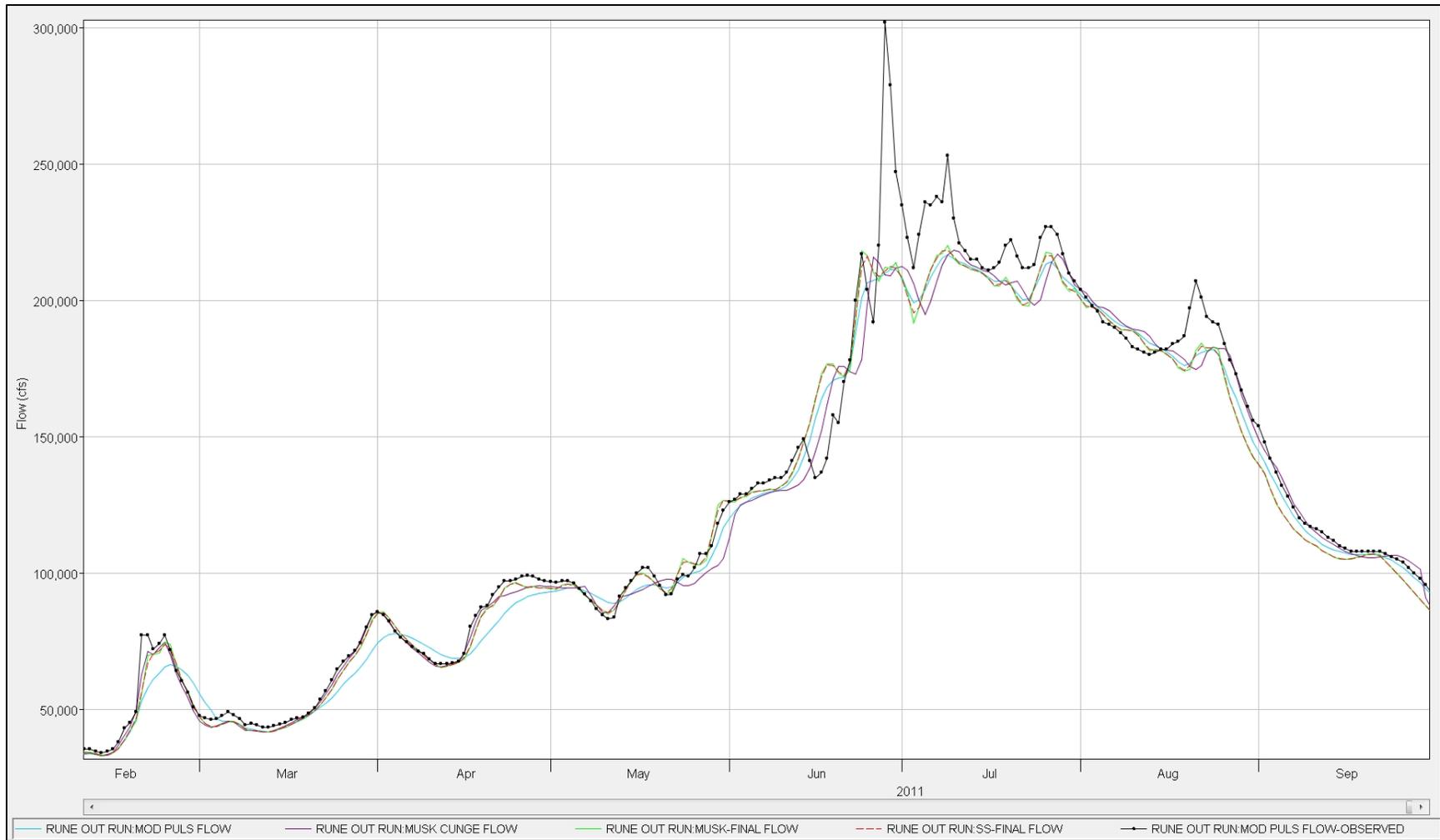
**Figure 8-7: OMA-NCNE 2011 Event.**



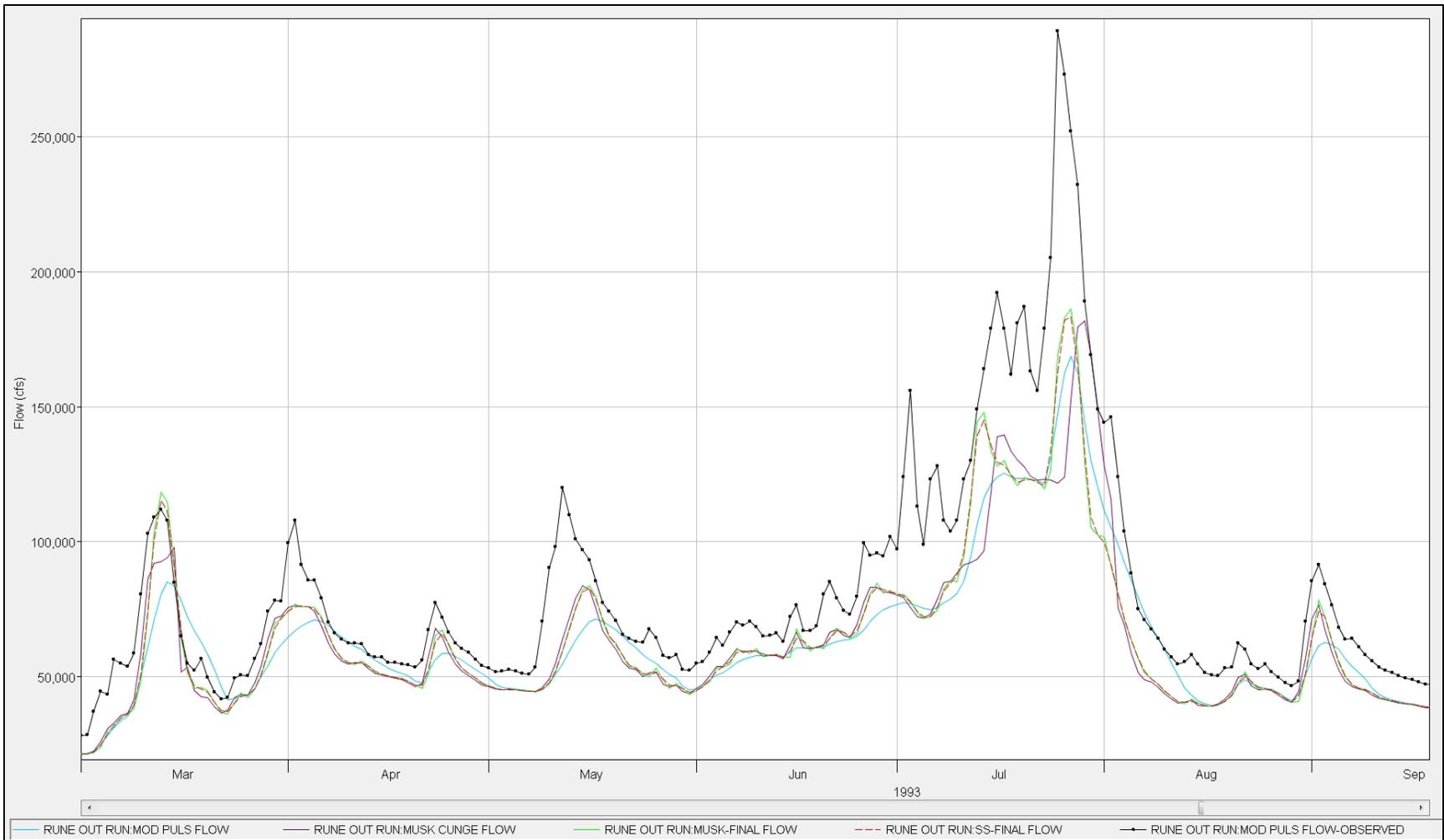
**Figure 8-8: OMA-NCNE 1952 Event.**



**Figure 8-9: OMA-NCNE 1944 Event.**



**Figure 8-10: NCNE-RUNE 2011 Event.**



**Figure 8-11: NCNE-RUNE 1993 Event.**

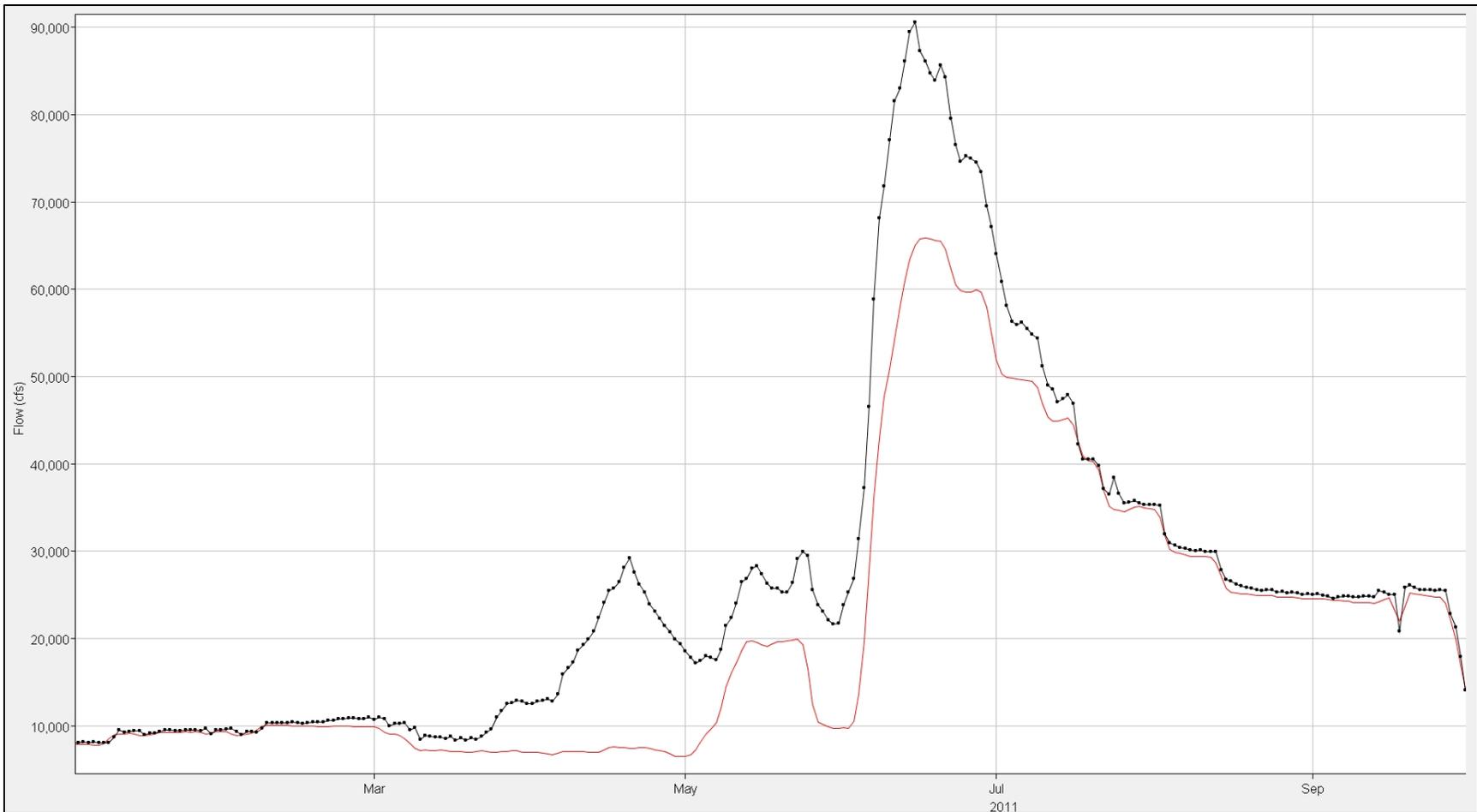


**Figure 8-12: NCNE-RUNE 1984 Event.**

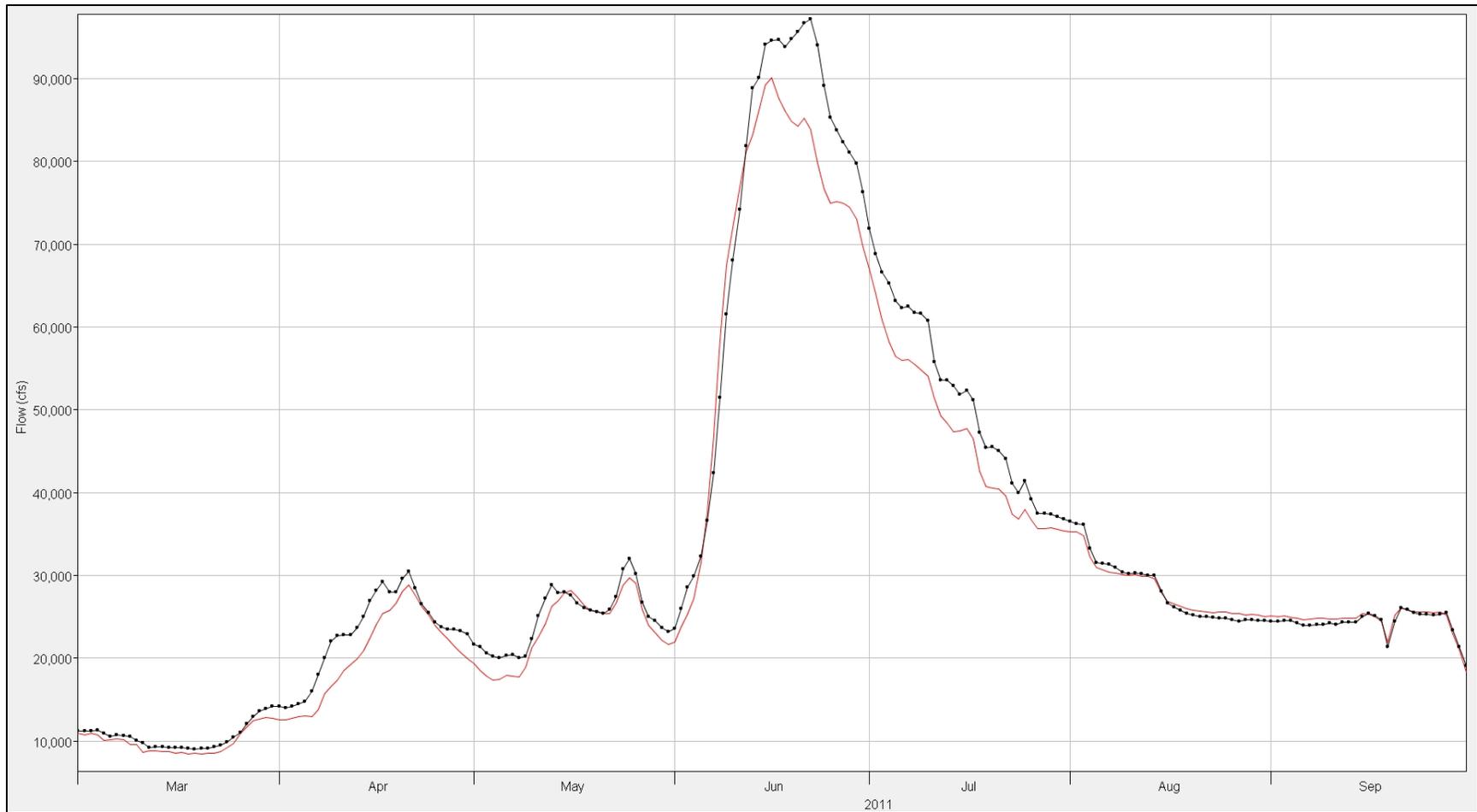
### 8.6.2 Straddle-Stagger Routing Results



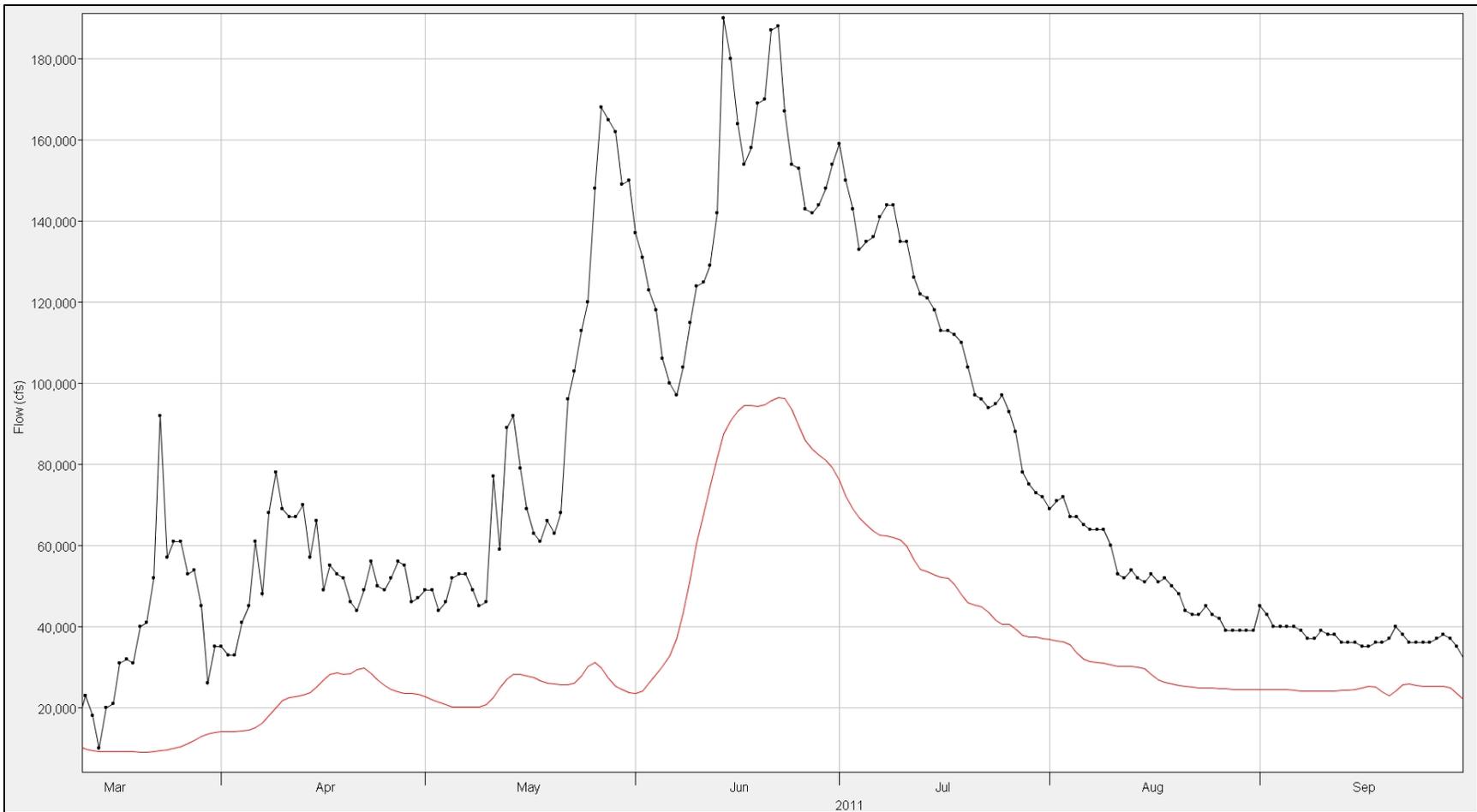
**Figure 8-13: Straddle-Stagger routing results for RBMT-FTPK during 2011. Red data are the Straddle-Stagger data and black data are the observed data.**



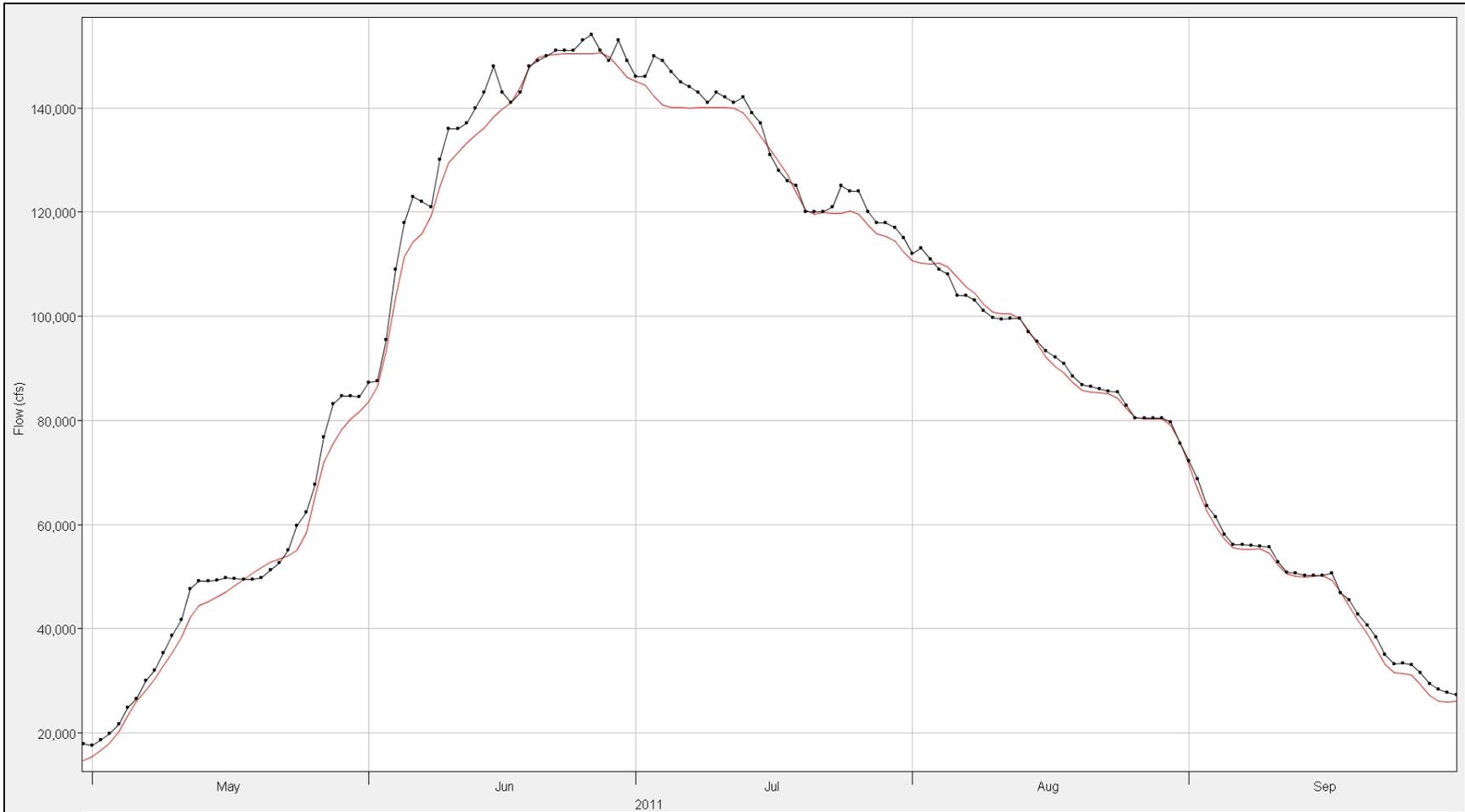
**Figure 8-14: Straddle-Stagger routing results for FTPK-WPMT during 2011. Red data are the Straddle-Stagger data and black data are the observed data.**



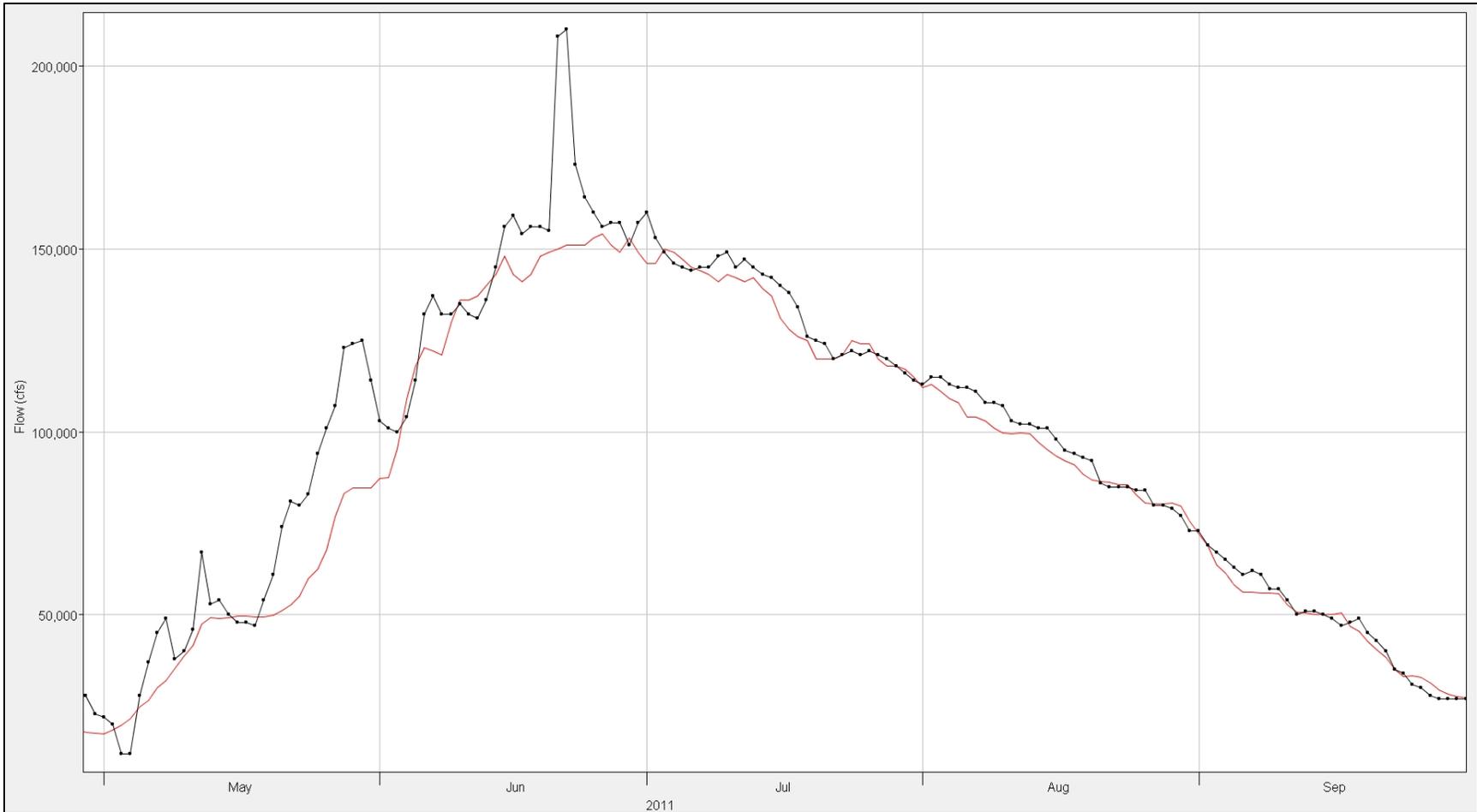
**Figure 8-15: Straddle-Stagger routing results for WPMT-CLMT during 2011. Red data are the Straddle-Stagger data and black data are the observed data.**



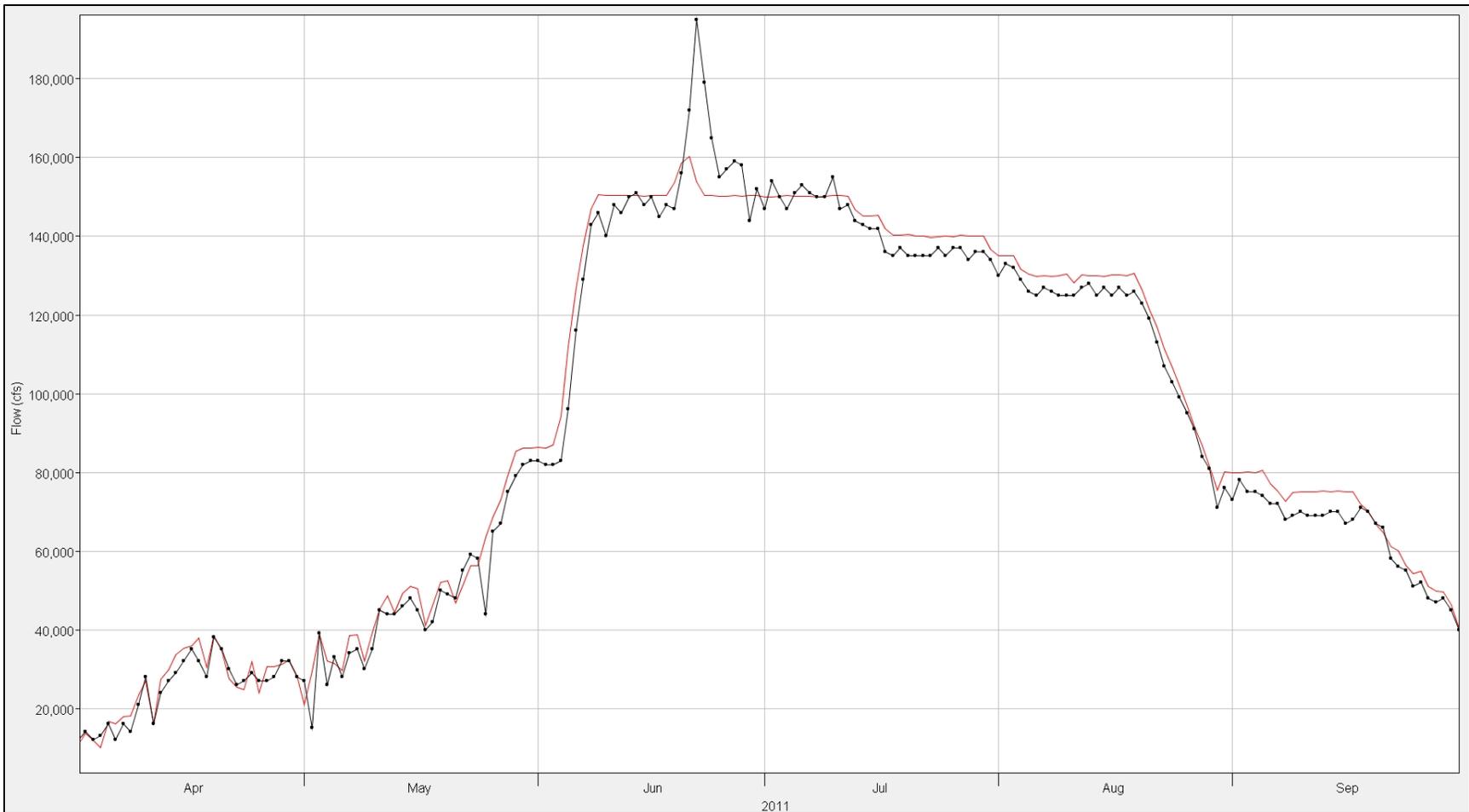
**Figure 8-16 Straddle-Stagger routing results for CLMT-GARR during 2011. Red data are the Straddle-Stagger data and black data are the observed data.**



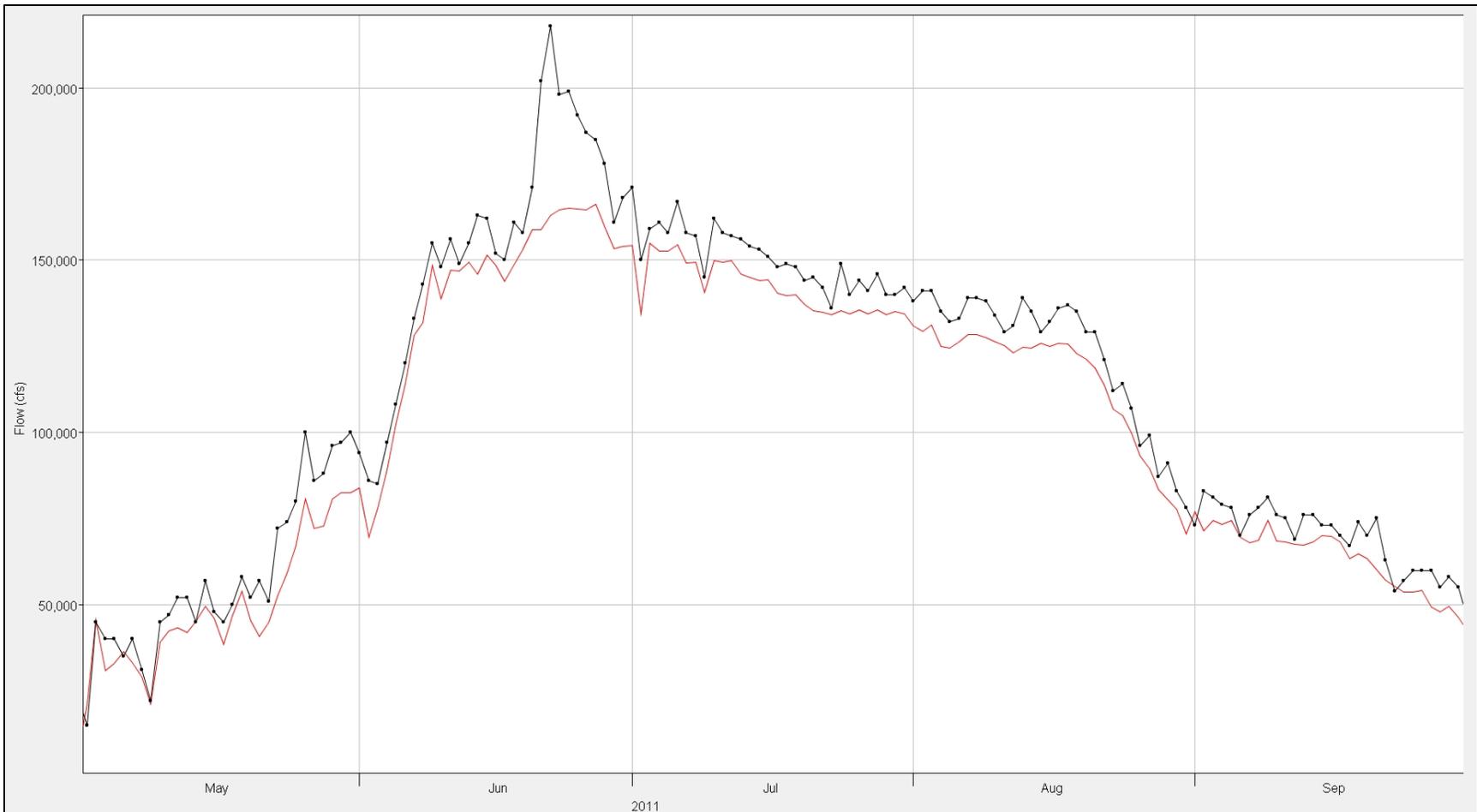
**Figure 8-17: Straddle-Stagger routing results for GARR-BIS during 2011. Red data are the Straddle-Stagger data and black data are the observed data.**



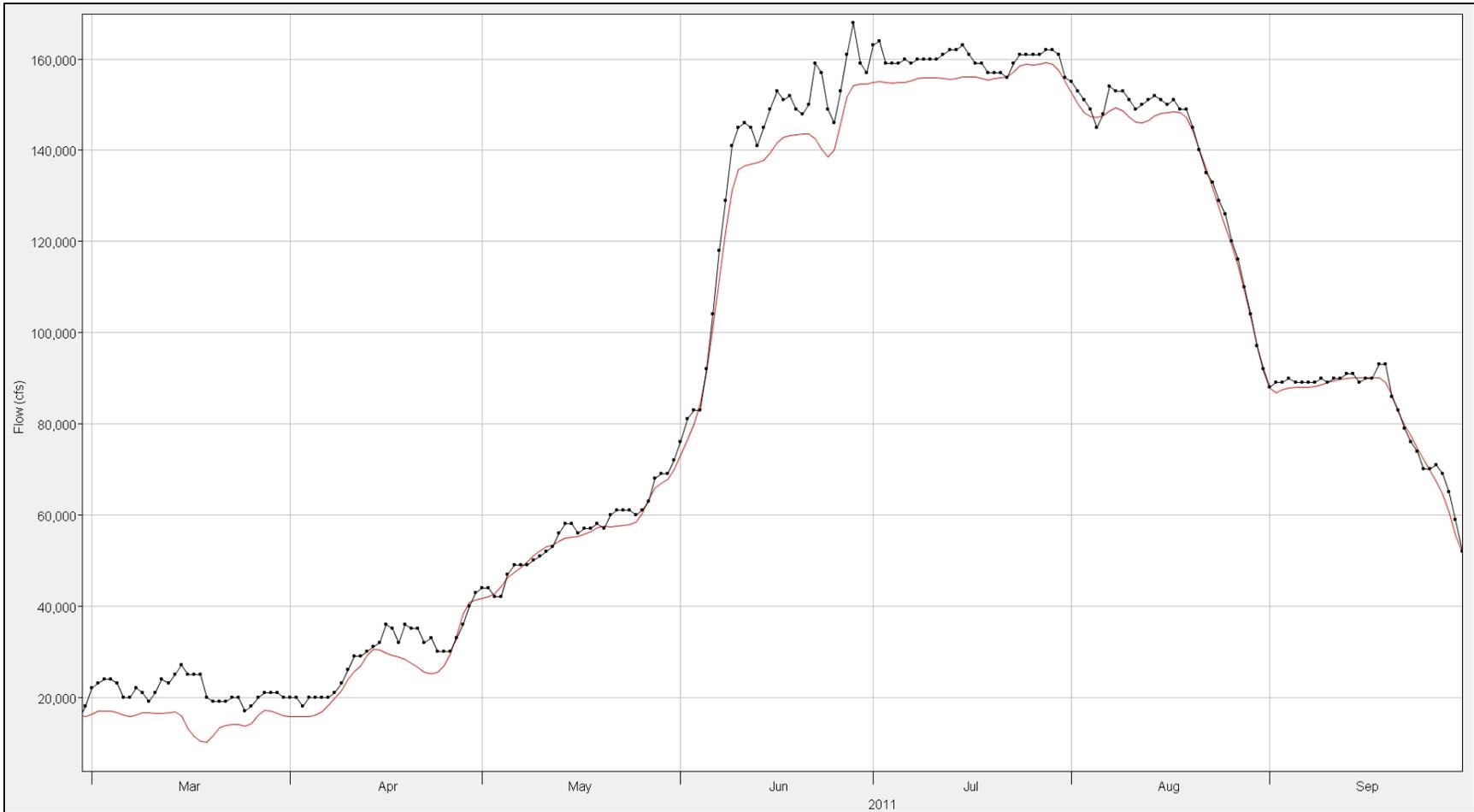
**Figure 8-18: Straddle-Stagger routing results for BIS-OAHE during 2011. Red data are the Straddle-Stagger data and black data are the observed data.**



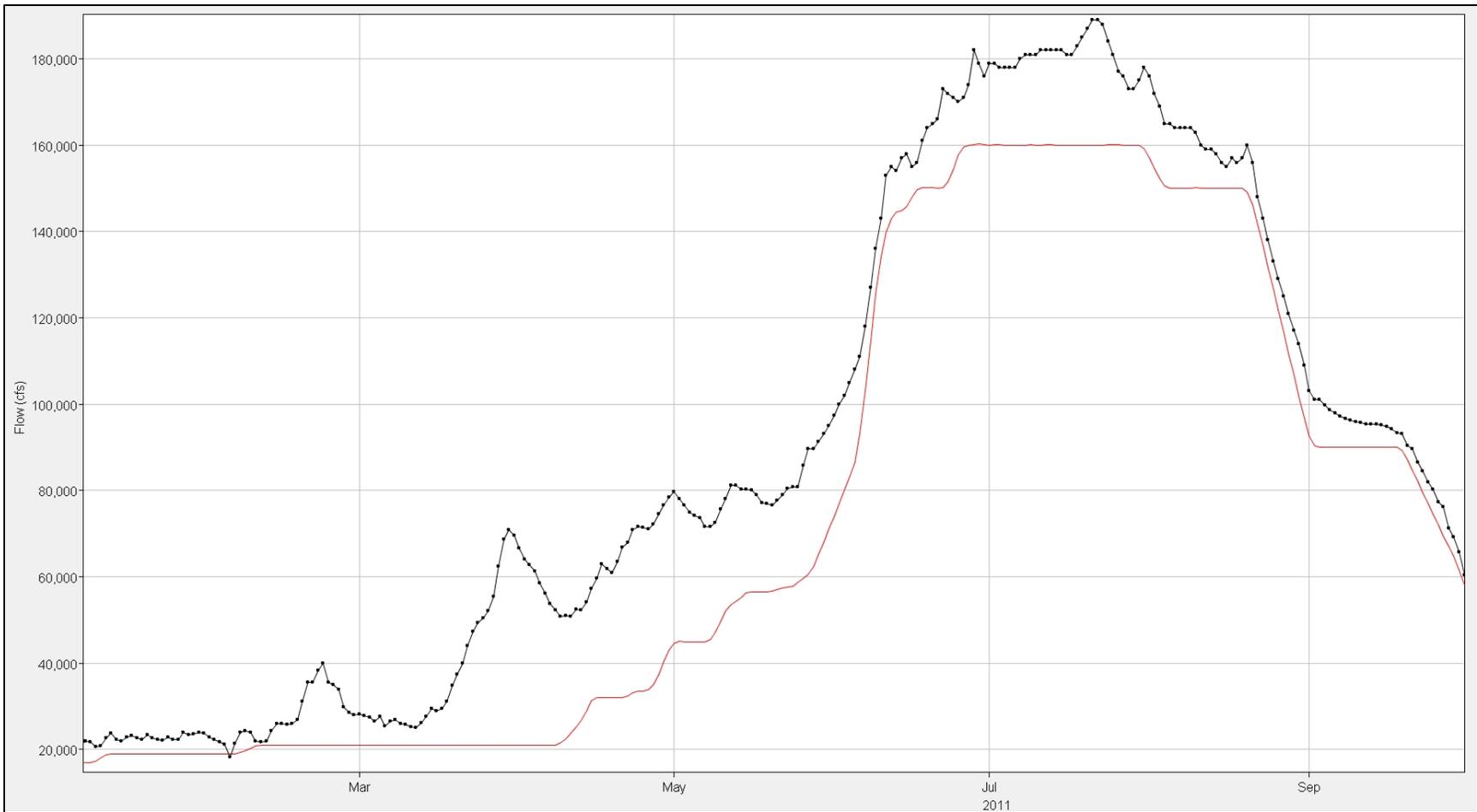
**Figure 8-19: Straddle-Stagger routing results for OAHE-BEND during 2011. Red data are the Straddle-Stagger data and black data are the observed data.**



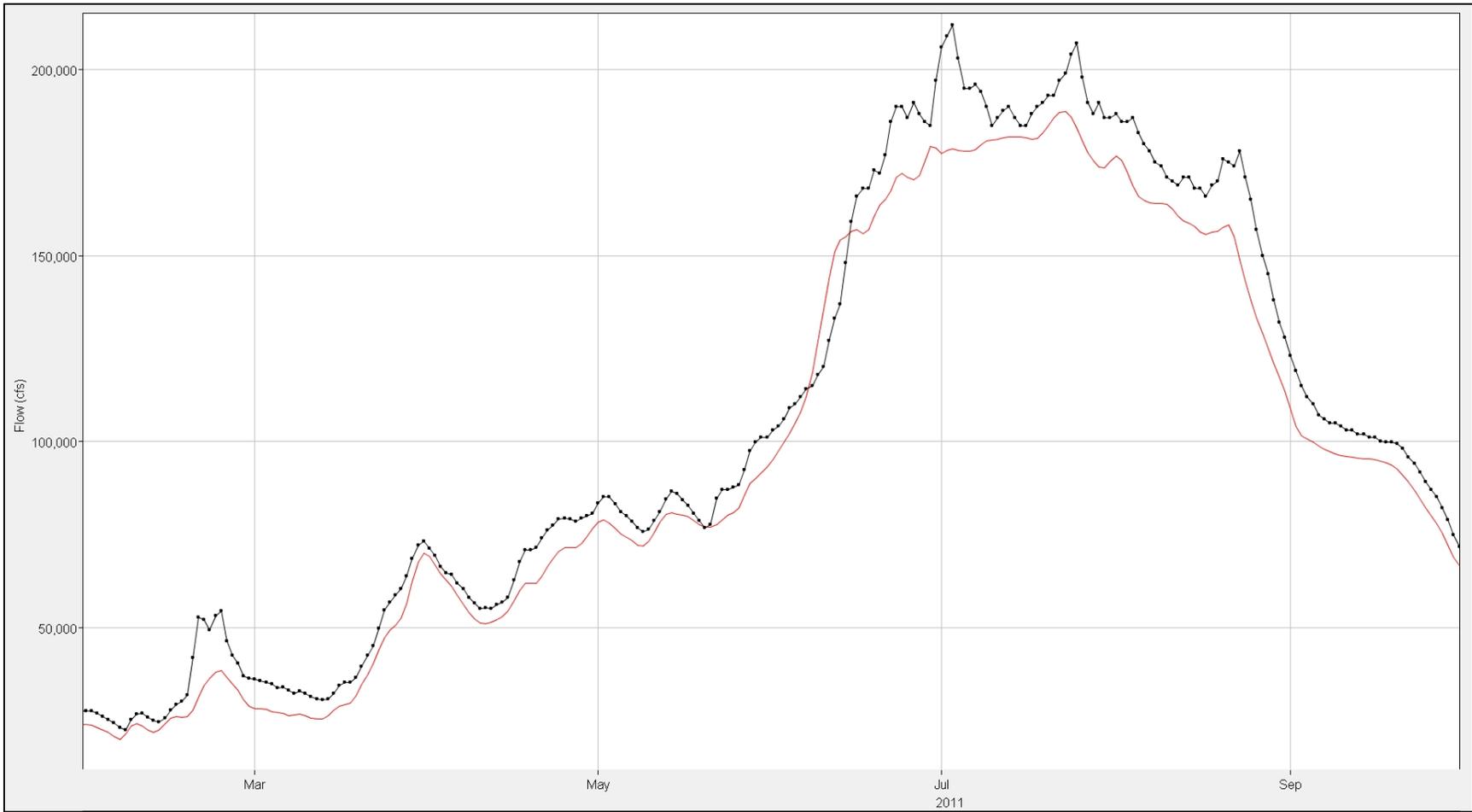
**Figure 8-20: Straddle-Stagger routing results for BEND-FTRA during 2011. Red data are the Straddle-Stagger data and black data are the observed data.**



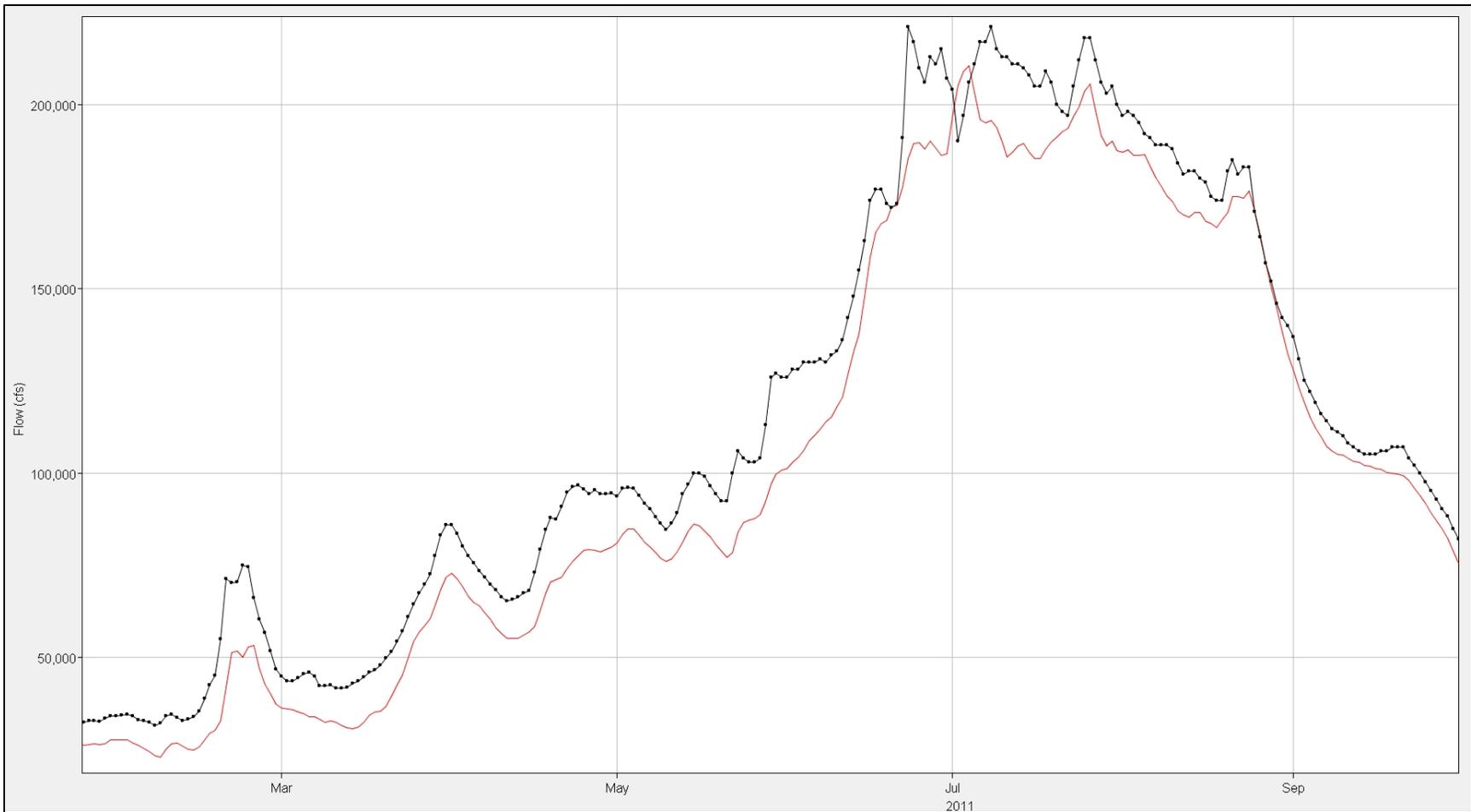
**Figure 8-21: Straddle-Stagger routing results for FTRA-GAPT during 2011. Red data are the Straddle-Stagger data and black data are the observed data.**



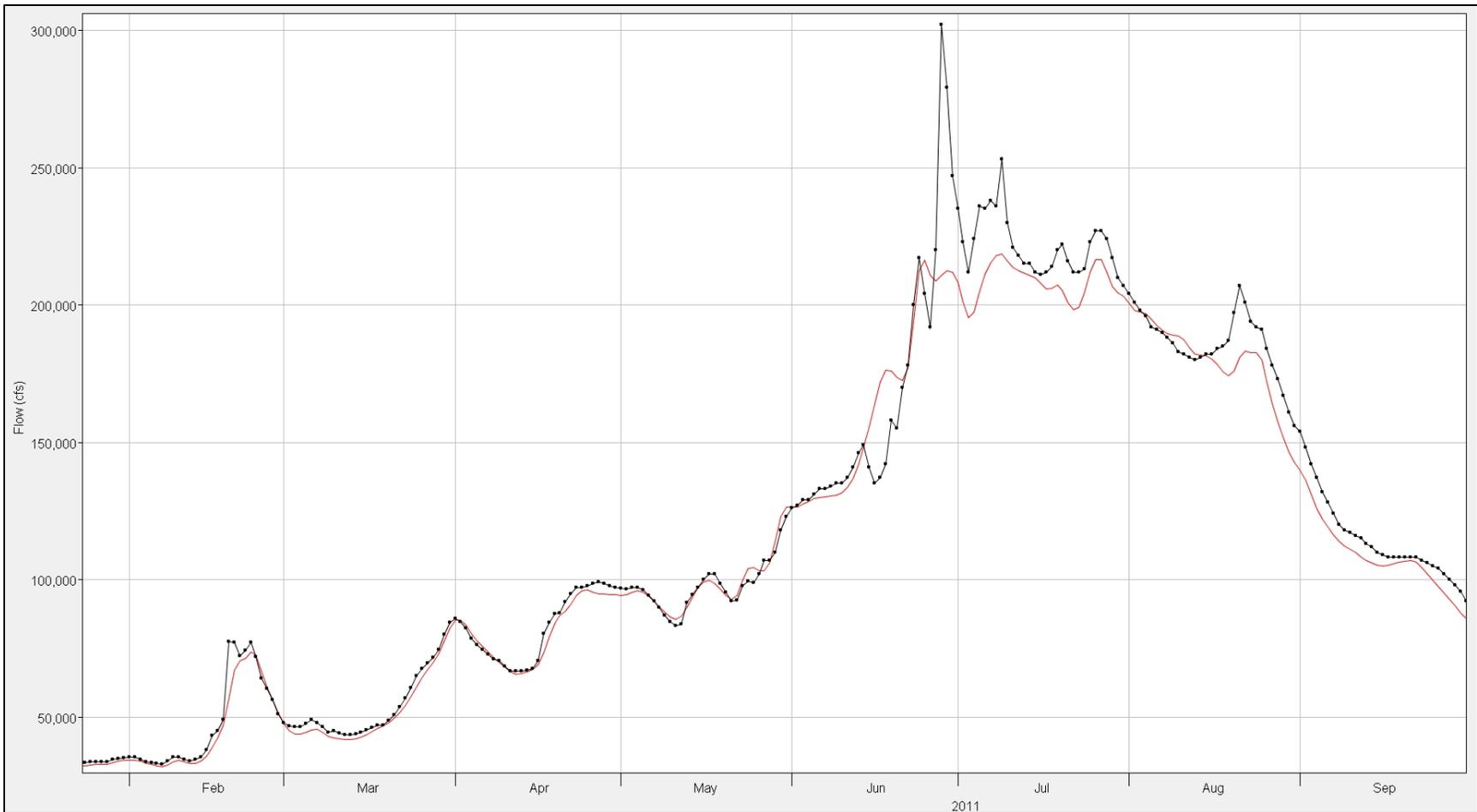
**Figure 8-22: Straddle-Stagger routing results for GAPT-SUX during 2011. Red data are the Straddle-Stagger data and black data are the observed data.**



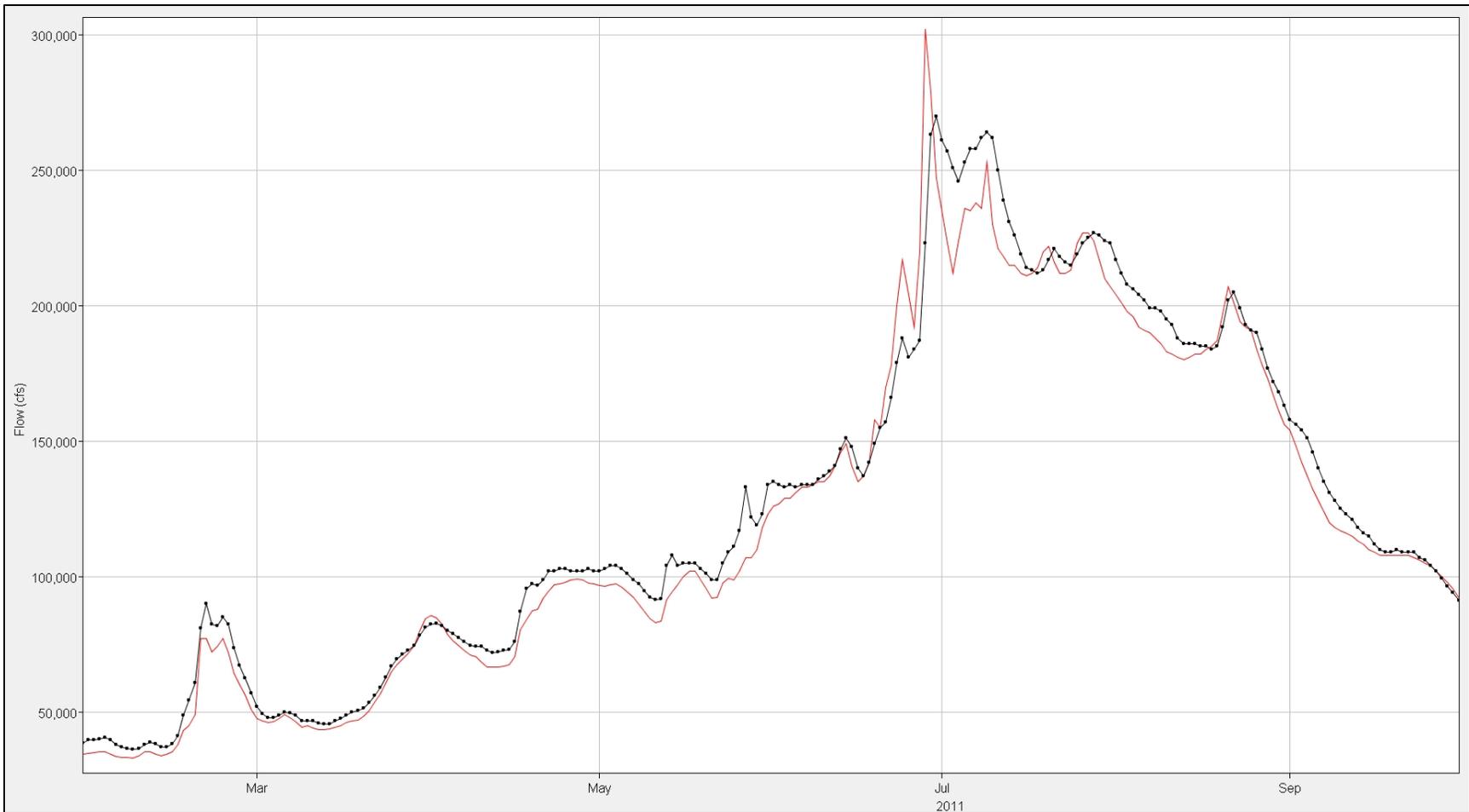
**Figure 8-23: Straddle-Stagger routing results for SUX-OMA during 2011. Red data are the Straddle-Stagger data and black data are the observed data.**



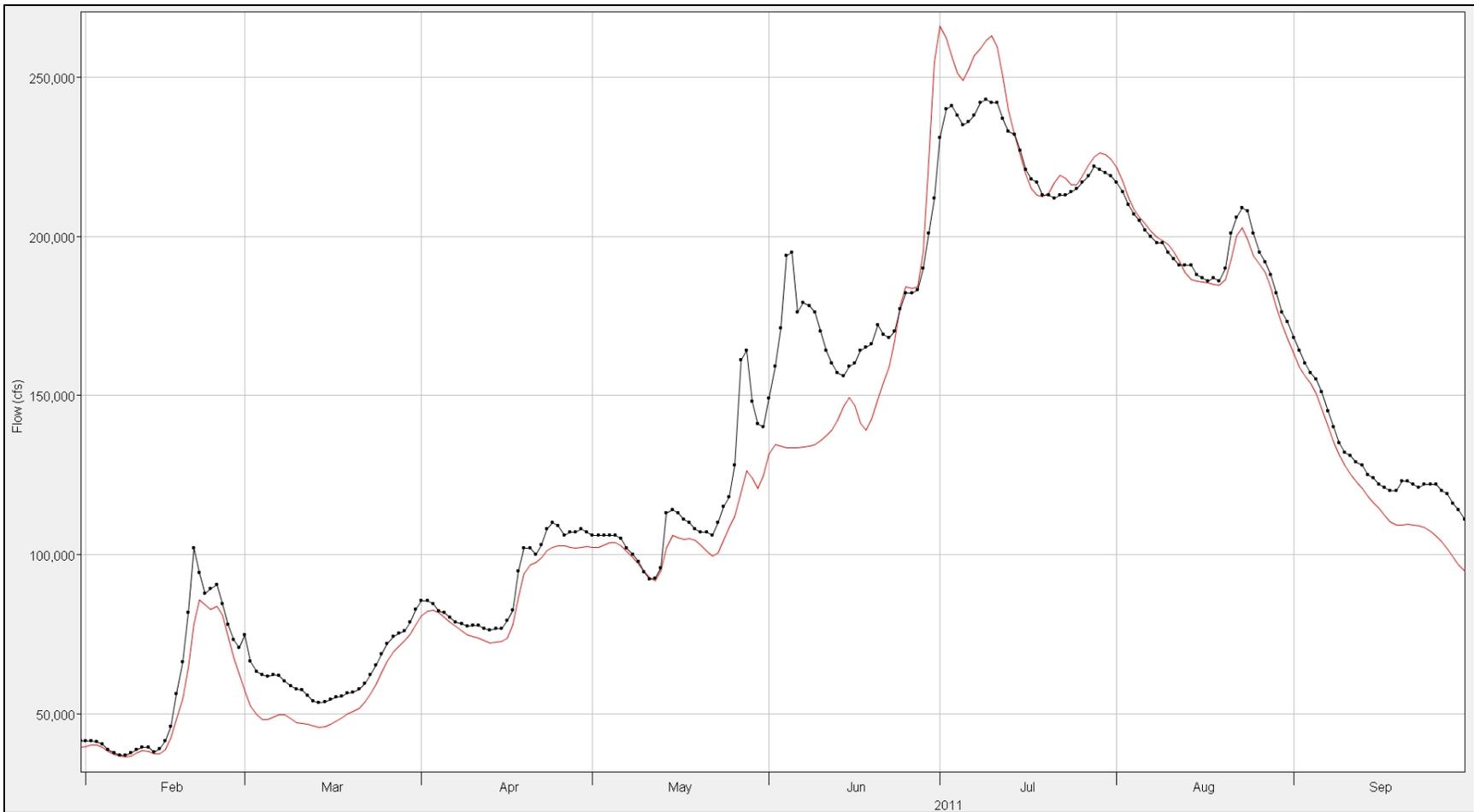
**Figure 8-24: Straddle-Stagger routing results for OMA-NCNE during 2011. Red data are the Straddle-Stagger data and black data are the observed data.**



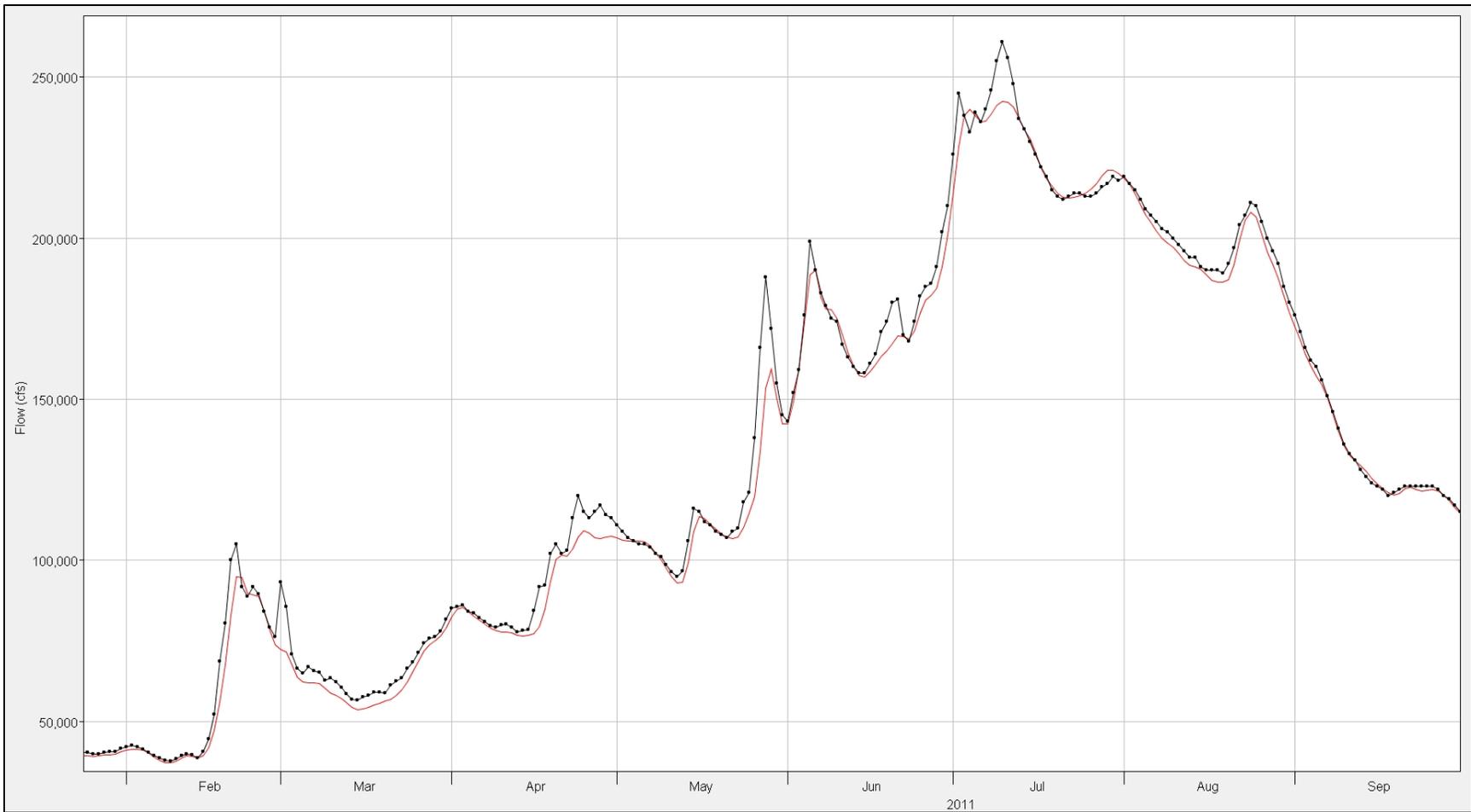
**Figure 8-25: Straddle-Stagger routing results for NCNE-RUNE during 2011. Red data are the Straddle-Stagger data and black data are the observed data.**



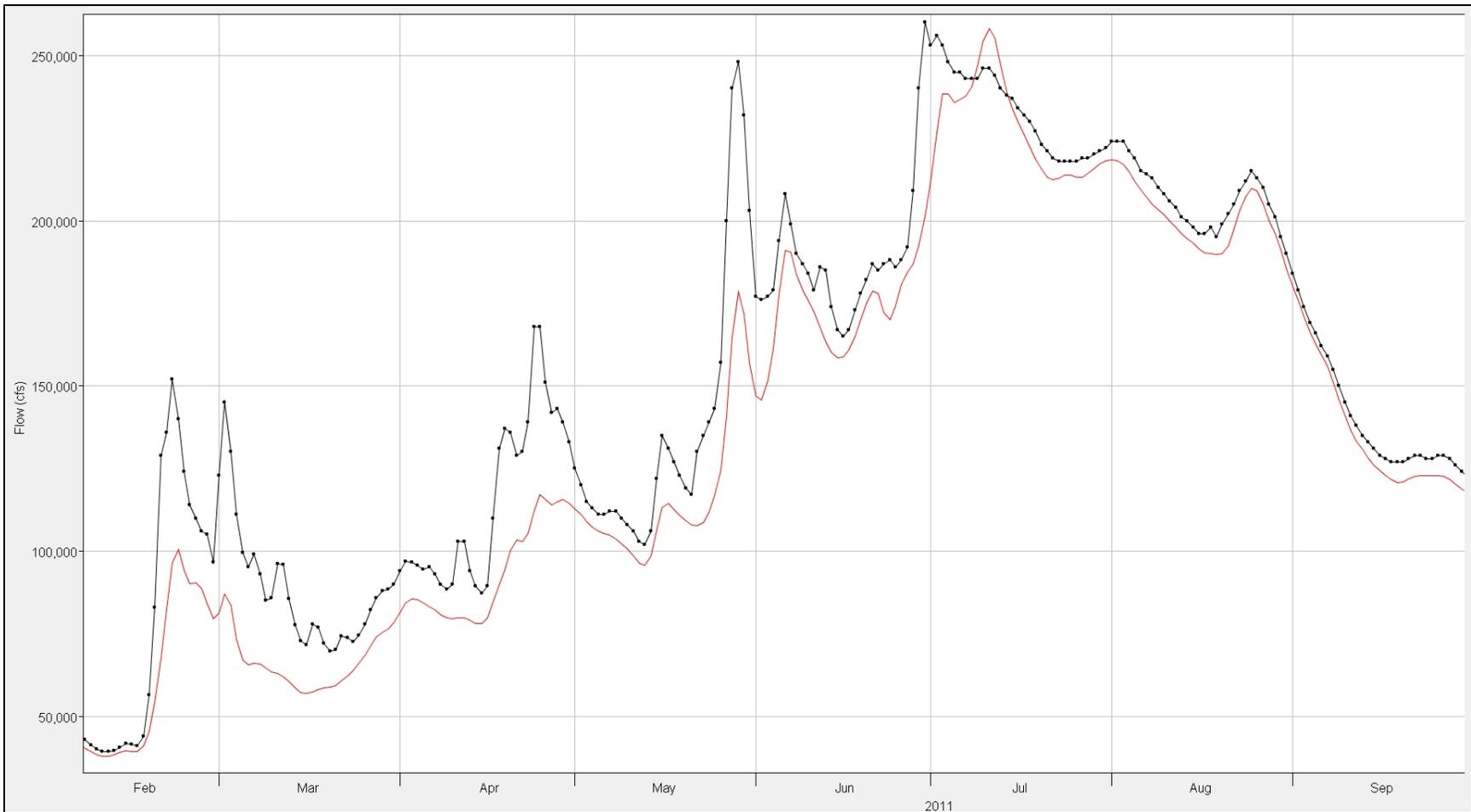
**Figure 8-26: Straddle-Stagger routing results for RUNE-STJ during 2011. Red data are the Straddle-Stagger data and black data are the observed data.**



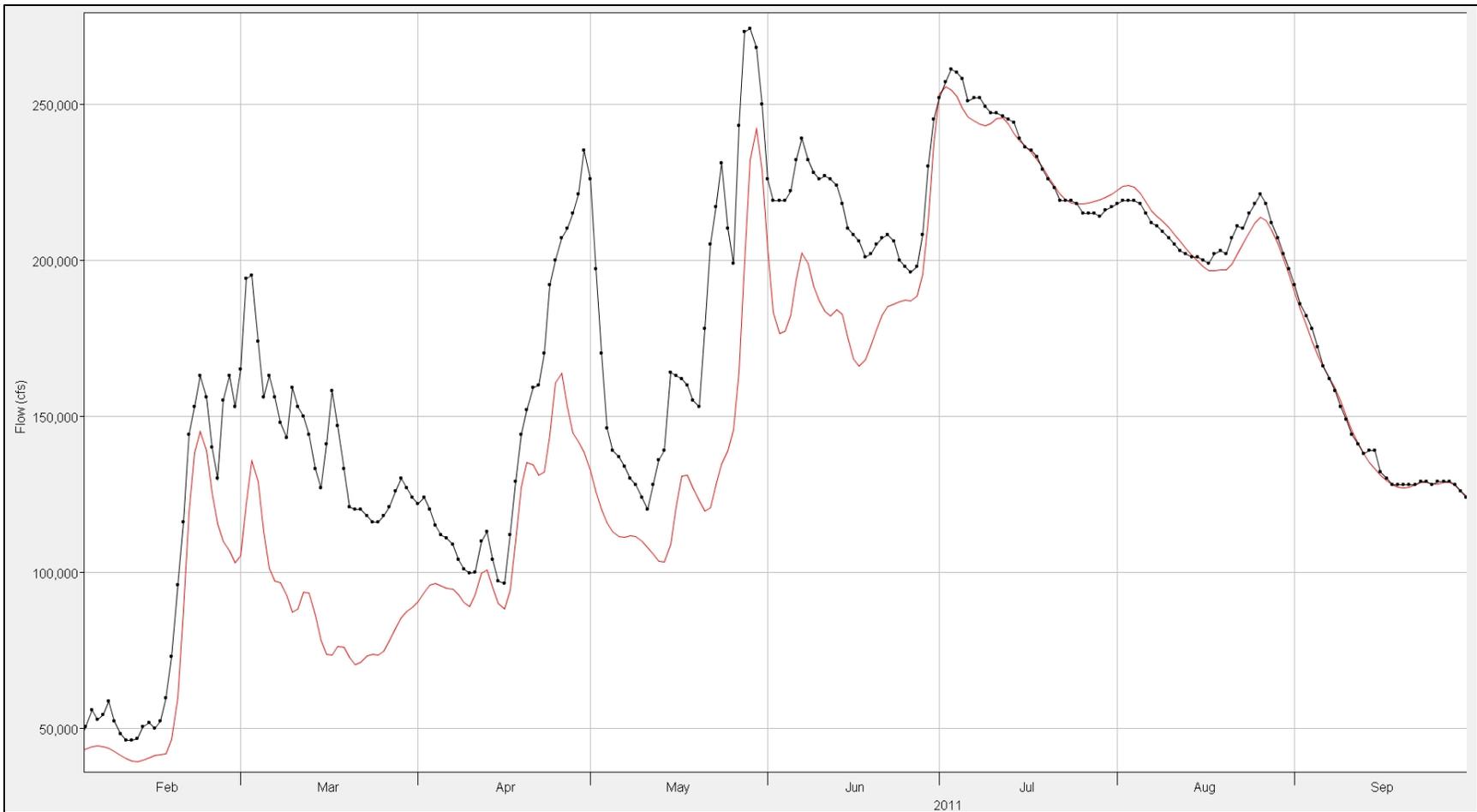
**Figure 8-27: Straddle-Stagger routing results for STJ-MKC during 2011. Red data are the Straddle-Stagger data and black data are the observed data.**



**Figure 8-28: Straddle-Stagger routing results for MKC-WVMO during 2011. Red data are the Straddle-Stagger data and black data are the observed data.**

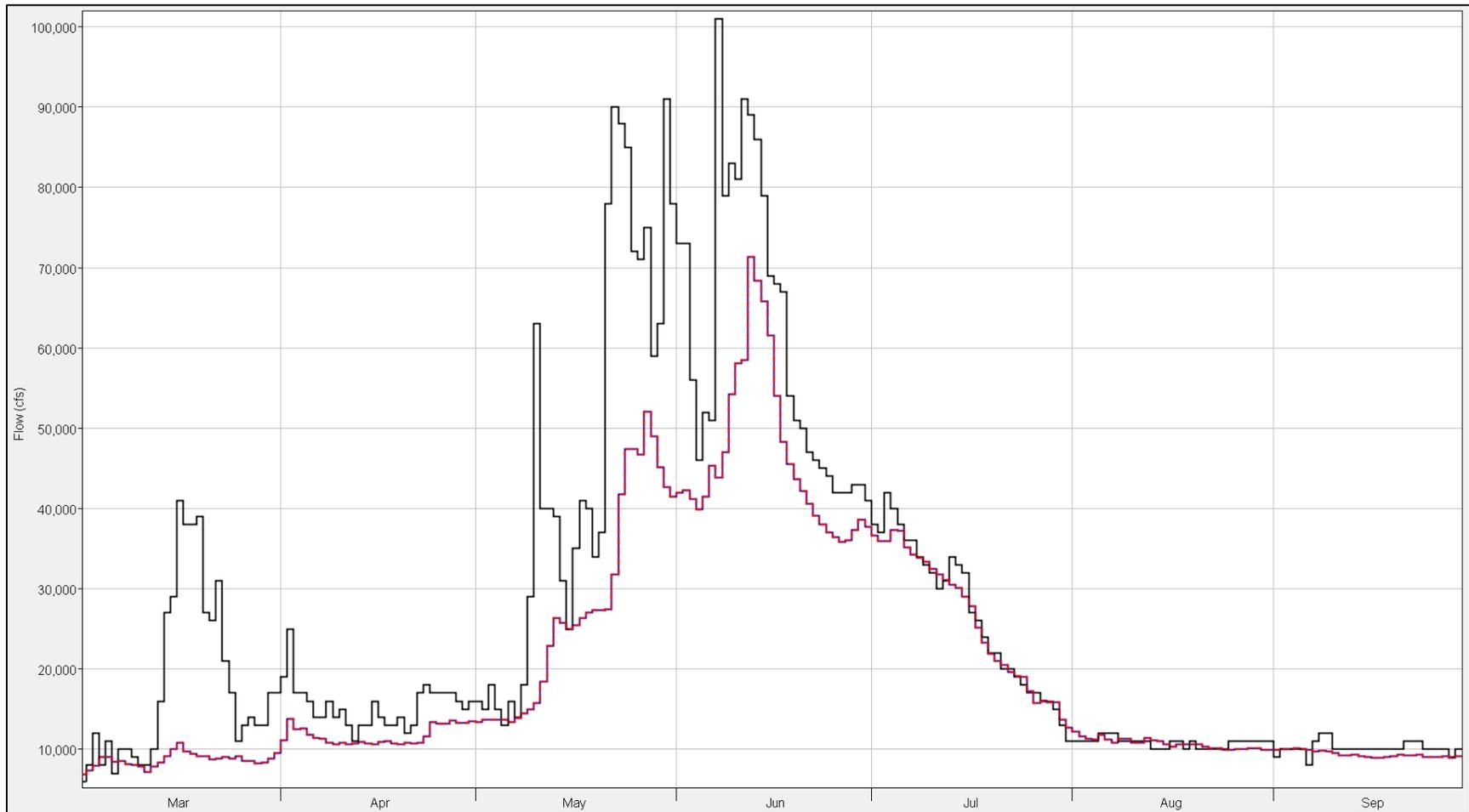


**Figure 8-29: Straddle-Stagger routing results for WVMO-BNMO during 2011. Red data are the Straddle-Stagger data and black data are the observed data.**

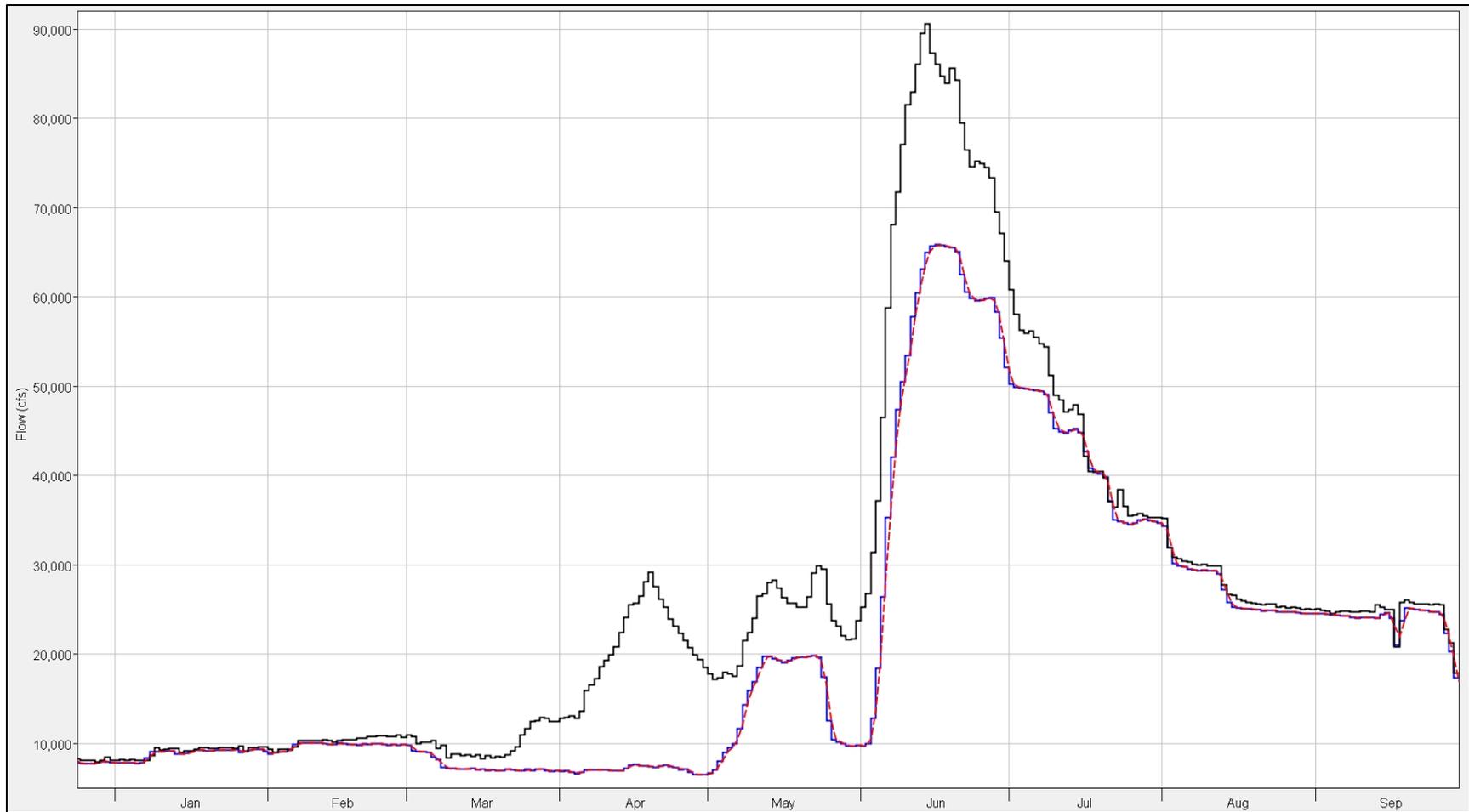


**Figure 8-30: Straddle-Stagger routing results for BNMO-HEMO during 2011. Red data are the Straddle-Stagger data and black data are the observed data.**

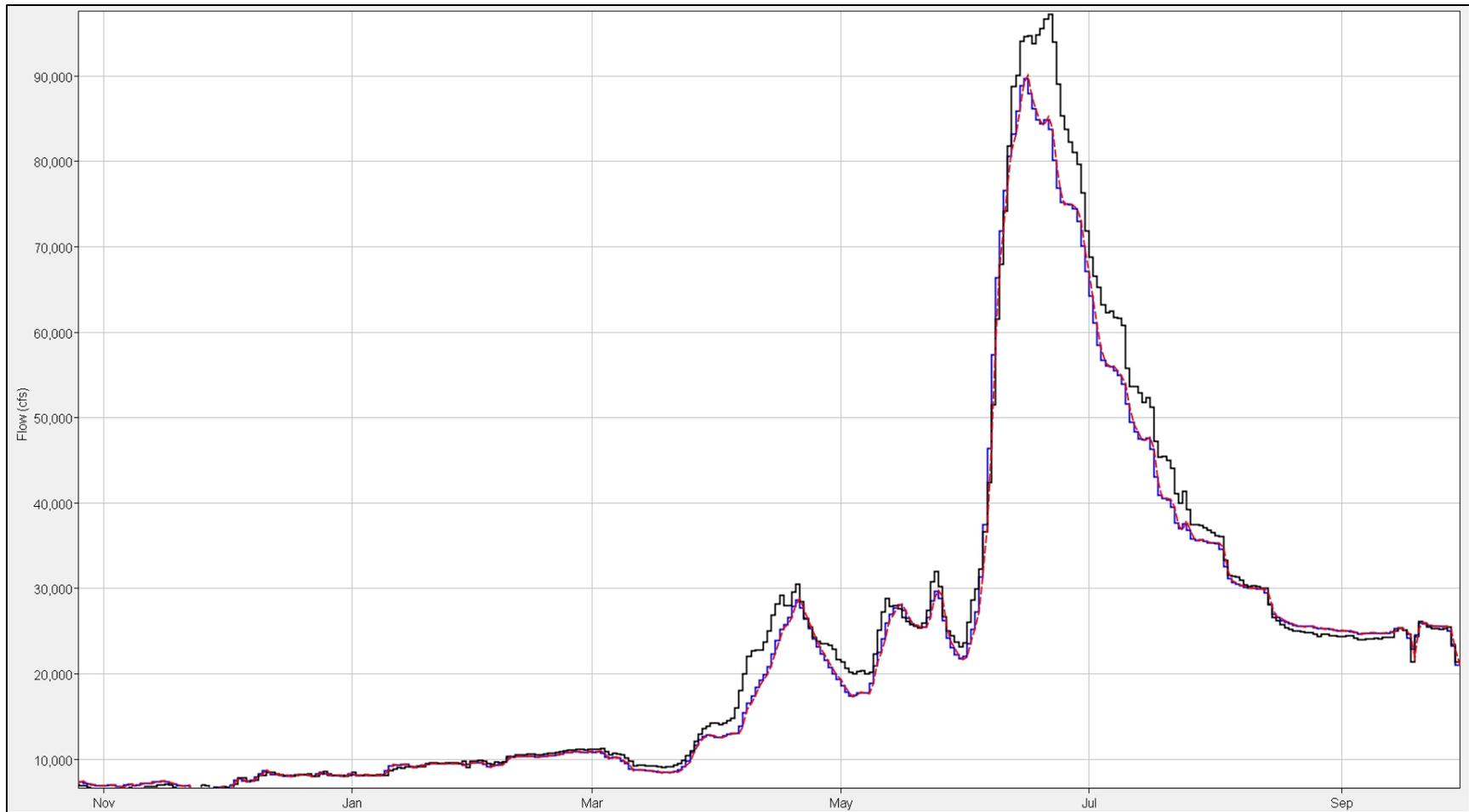
### 8.6.3 Straddle-Stagger vs. Coefficient Routing Plots



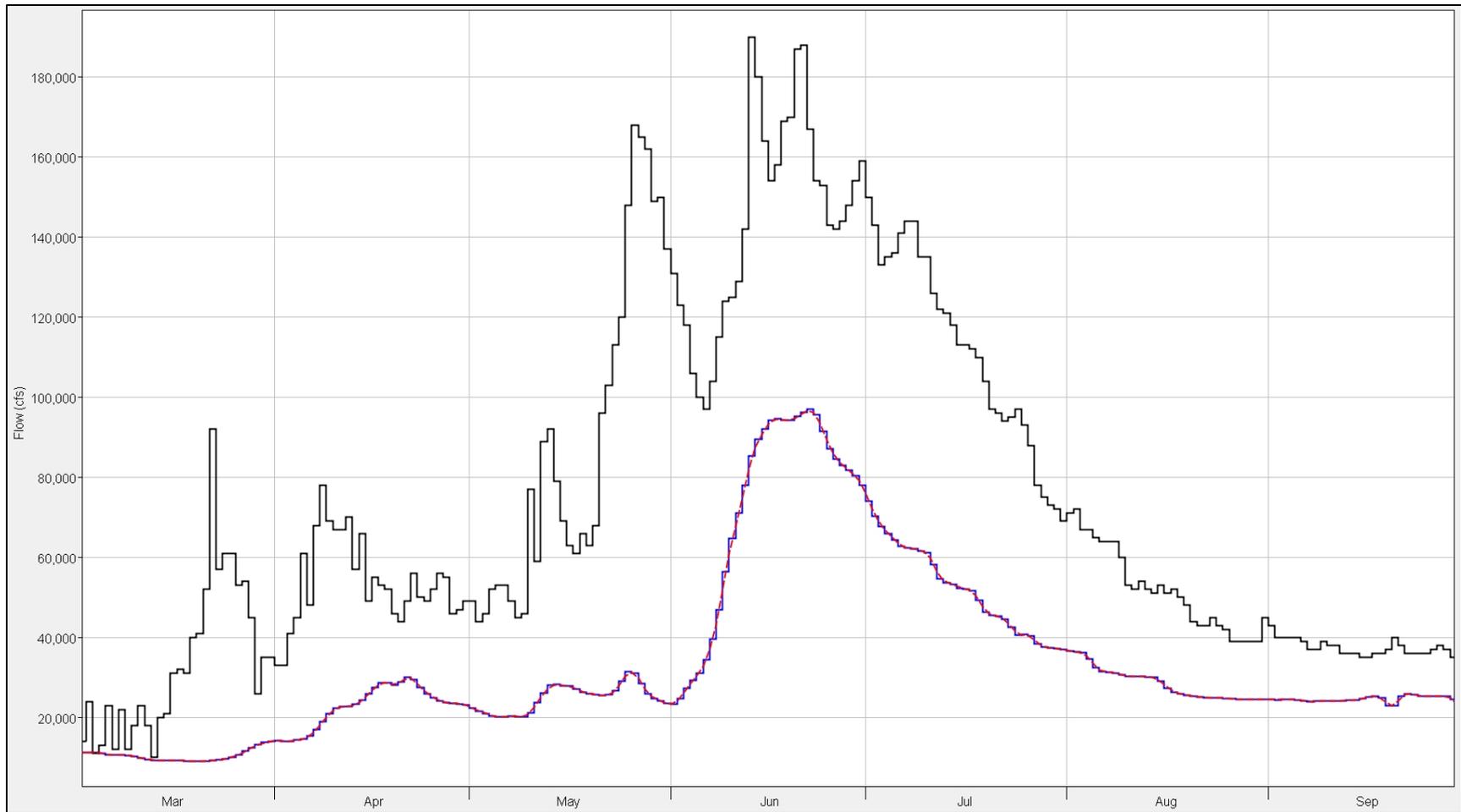
**Figure 8-31: Straddle-Stagger vs. Coefficient routing results for RBMT-FTPK during 2011. Red data are the Straddle-Stagger data, blue data are the Coefficient Routing data, and black data are the observed data.**



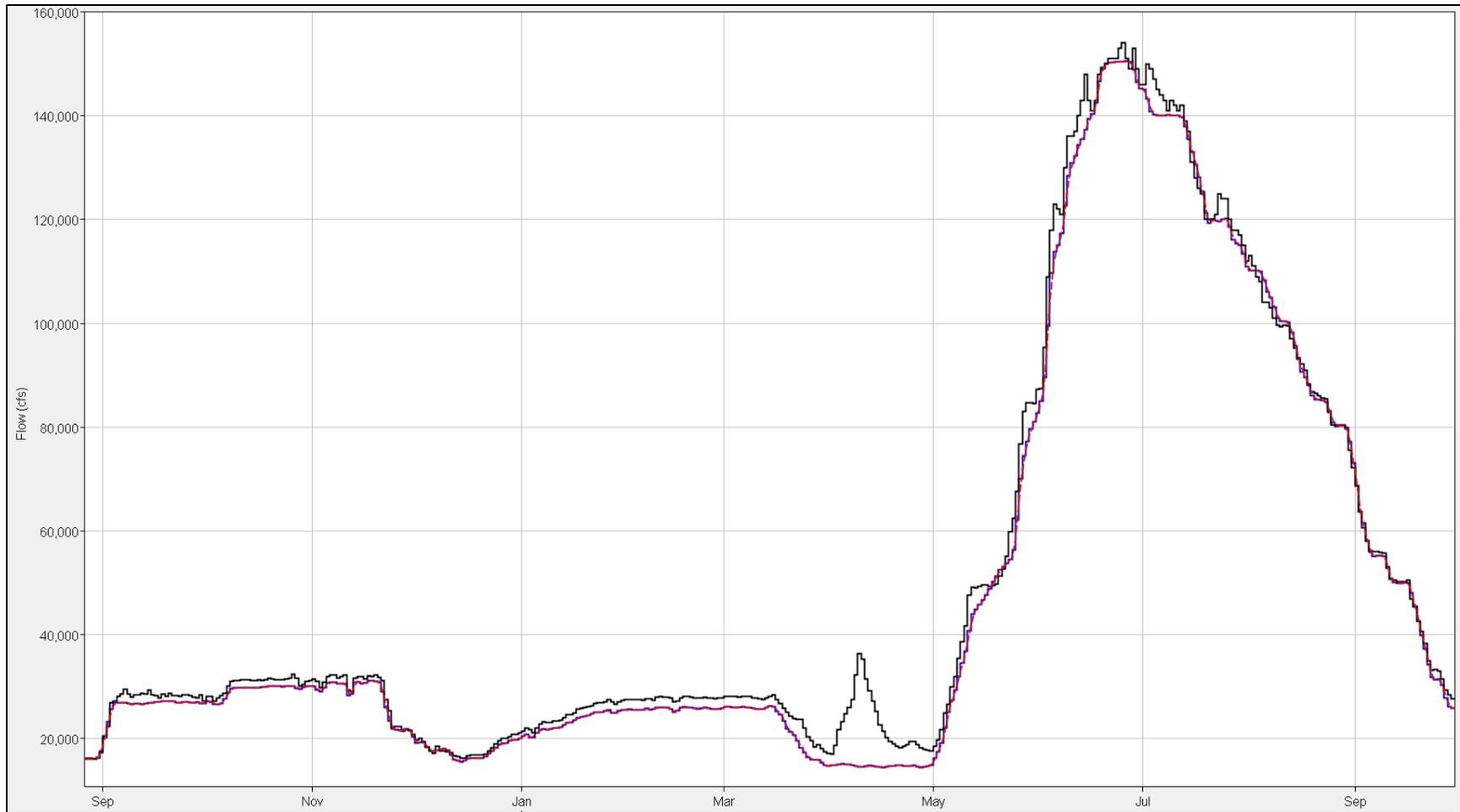
**Figure 8-32: Straddle-Stagger vs. Coefficient routing results for FTPK-WPMT during 2011. Red data are the Straddle-Stagger data, blue data are the Coefficient Routing data, and black data are the observed data.**



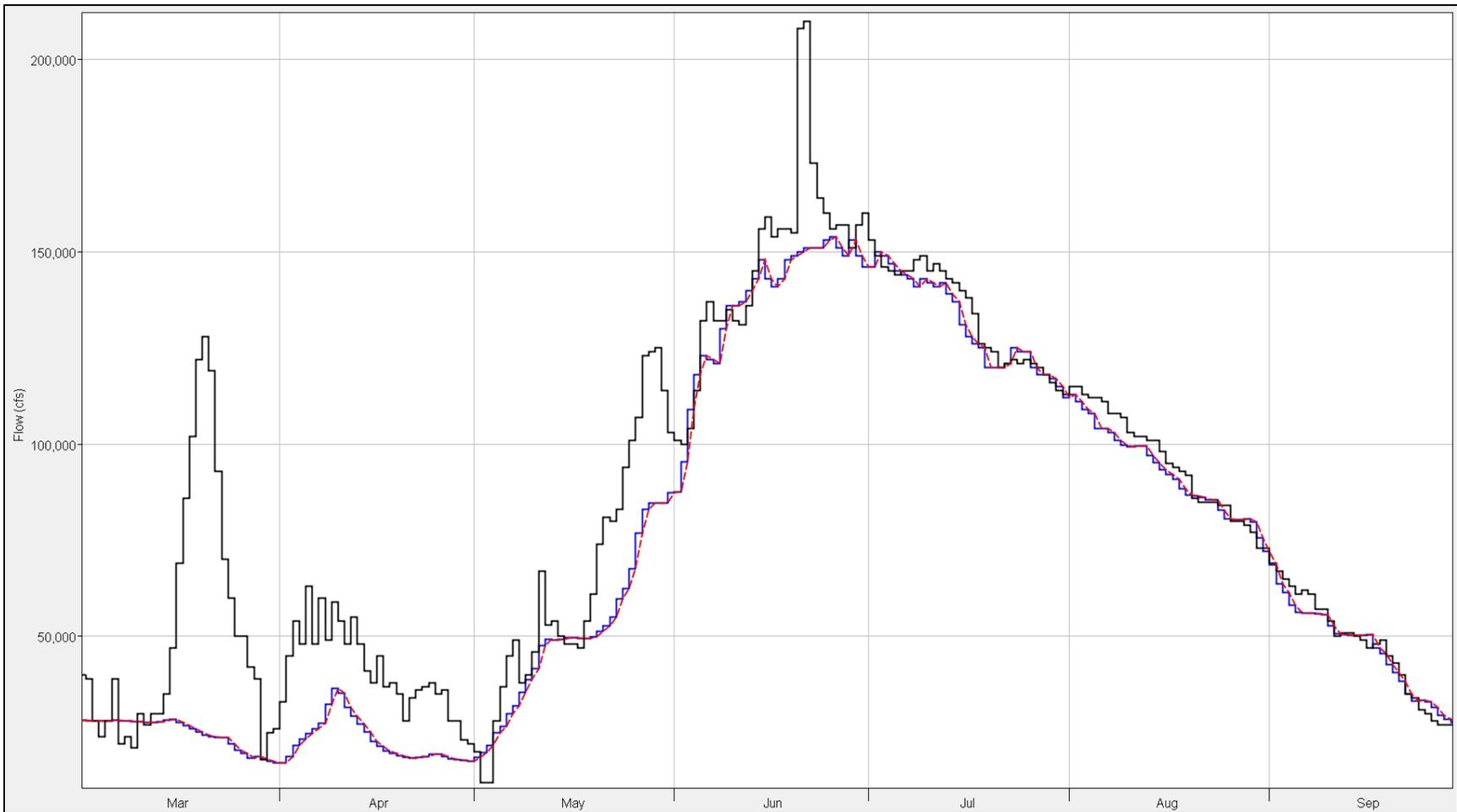
**Figure 8-33: Straddle-Stagger vs. Coefficient routing results for WPMT-CLMT during 2011. Red data are the Straddle-Stagger data, blue data are the Coefficient Routing data, and black data are the observed data.**



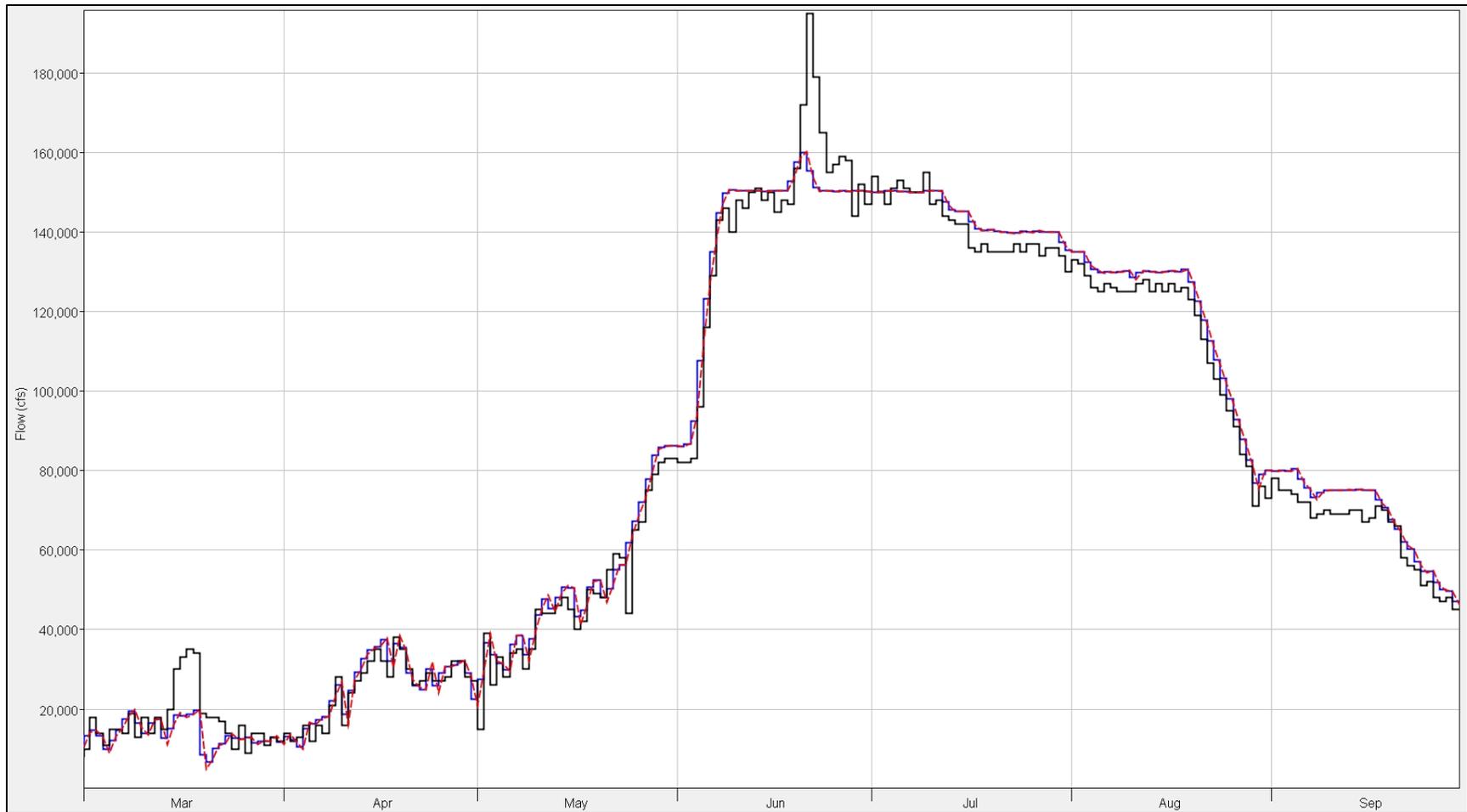
**Figure 8-34: Straddle-Stagger vs. Coefficient routing results for CLMT-GARR during 2011. Red data are the Straddle-Stagger data, blue data are the Coefficient Routing data, and black data are the observed data.**



**Figure 8-35: Straddle-Stagger vs. Coefficient routing results for GARR-BIS during 2011. Red data are the Straddle-Stagger data, blue data are the Coefficient Routing data, and black data are the observed data.**



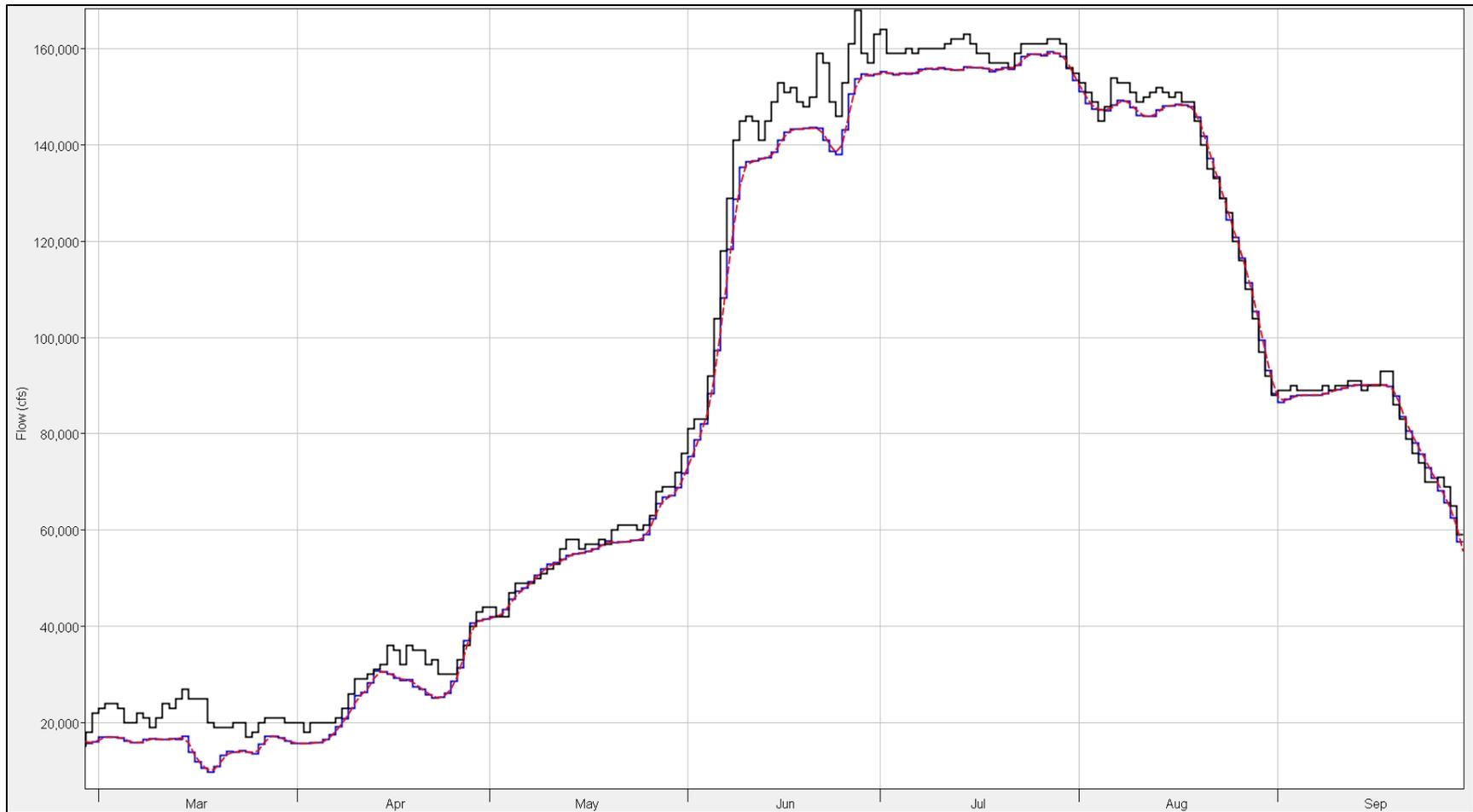
**Figure 8-36: Straddle-Stagger vs. Coefficient routing results for BIS-OAHE during 2011. Red data are the Straddle-Stagger data, blue data are the Coefficient Routing data, and black data are the observed data.**



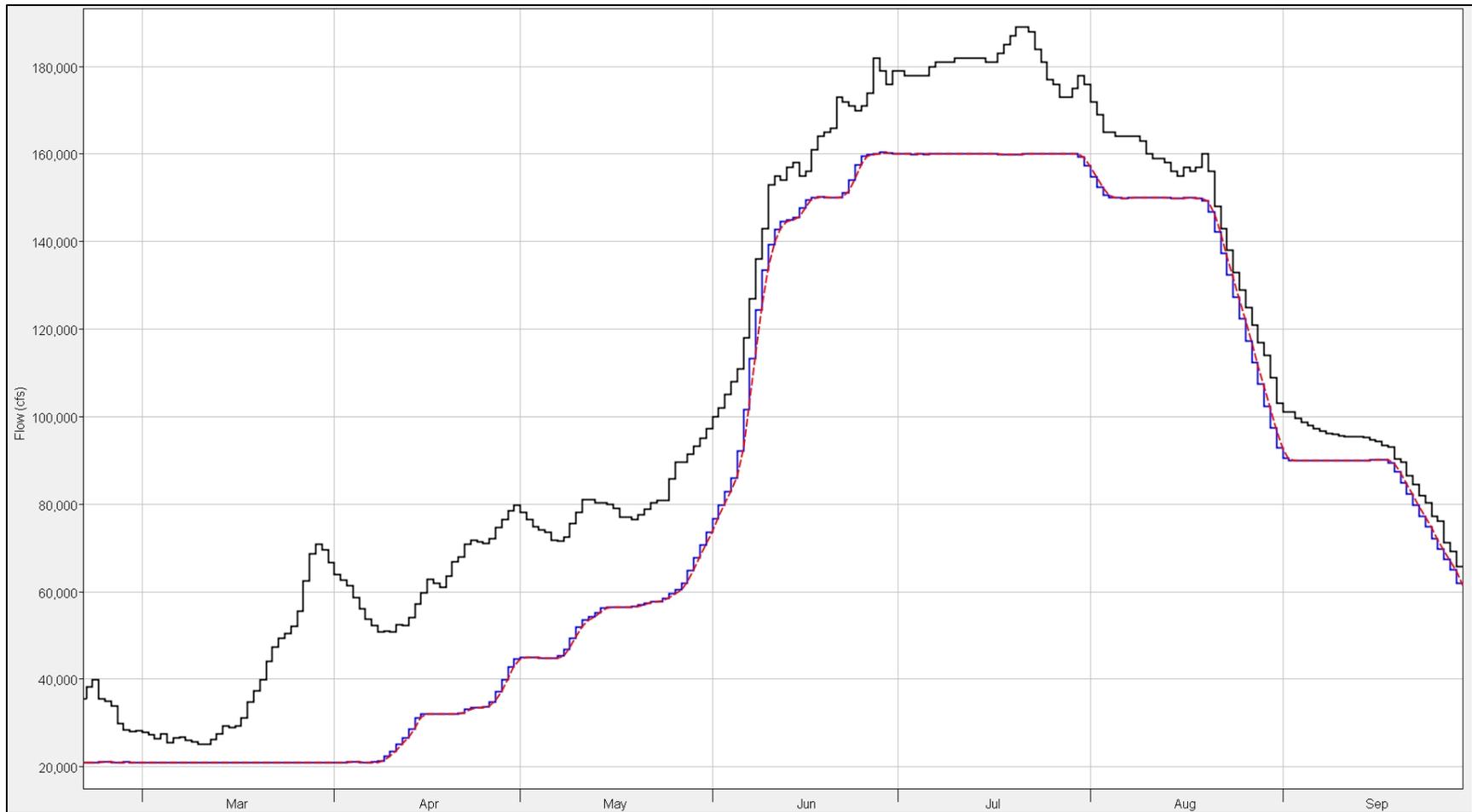
**Figure 8-37: Straddle-Stagger vs. Coefficient routing results for OAHE-BEND during 2011. Red data are the Straddle-Stagger data, blue data are the Coefficient Routing data, and black data are the observed data.**



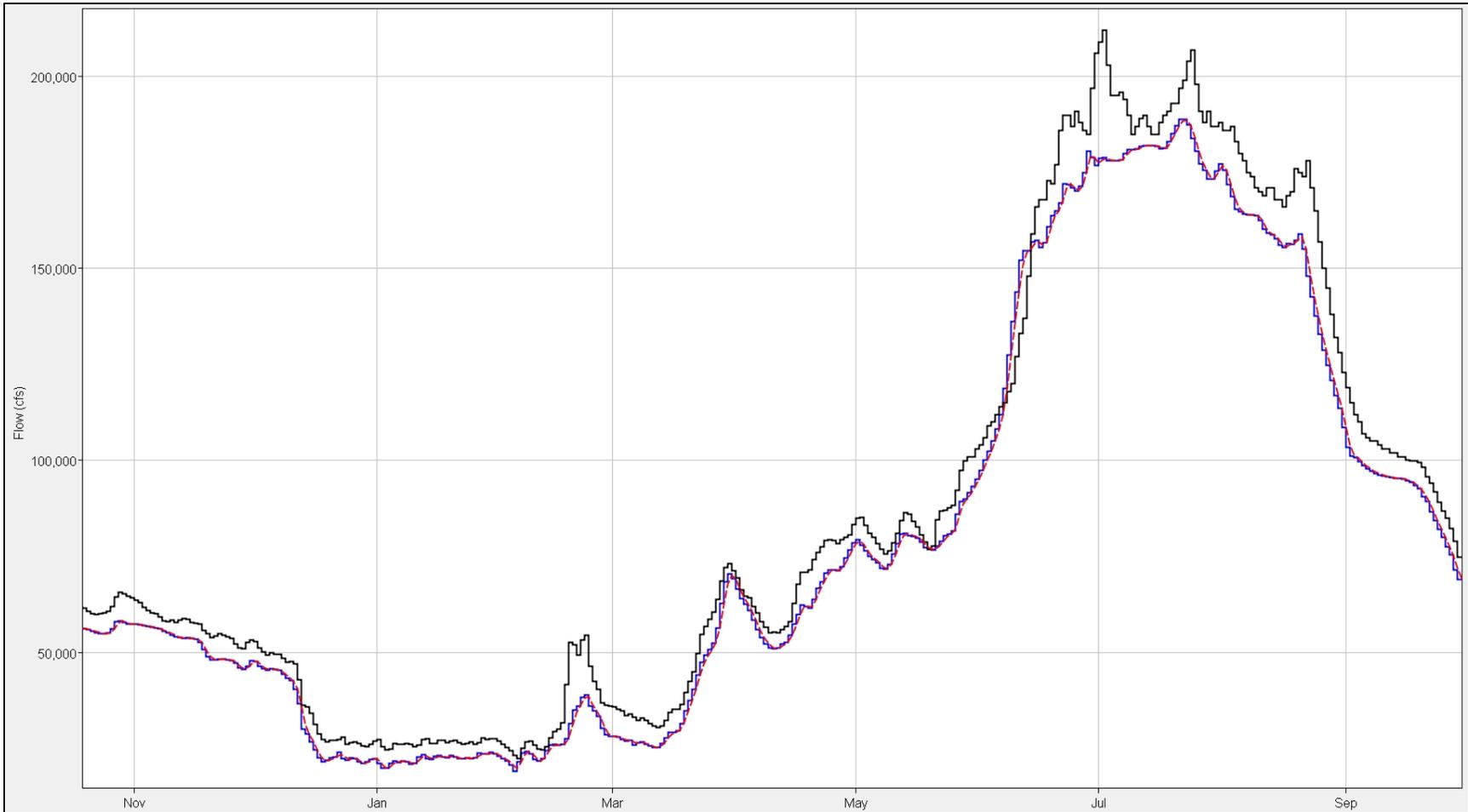
**Figure 8-38: Straddle-Stagger vs. Coefficient routing results for BEND-FTRA during 2011. Red data are the Straddle-Stagger data, blue data are the Coefficient Routing data, and black data are the observed data.**



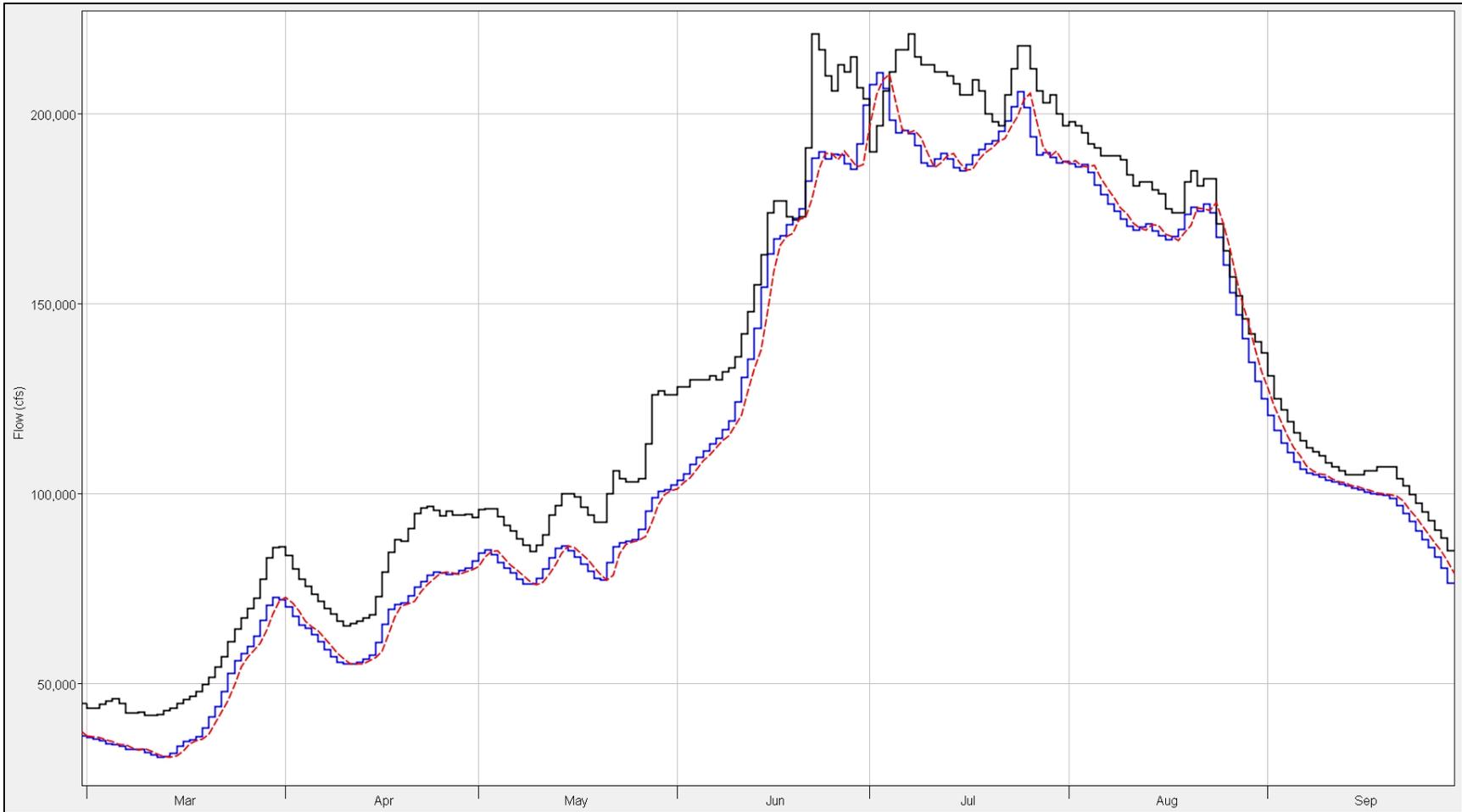
**Figure 8-39: Straddle-Stagger vs. Coefficient routing results for FTRA-GAPT during 2011. Red data are the Straddle-Stagger data, blue data are the Coefficient Routing data, and black data are the observed data.**



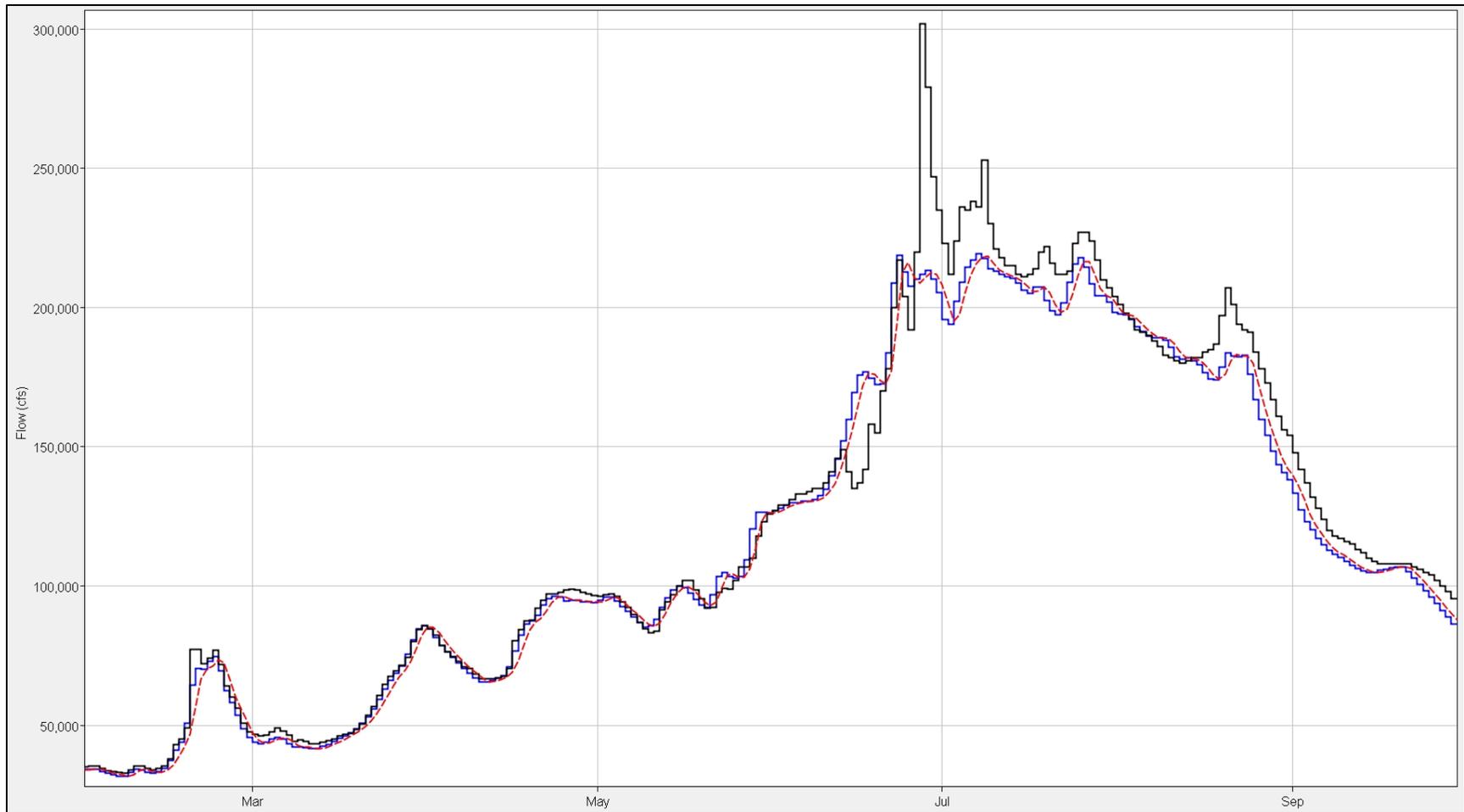
**Figure 8-40: Straddle-Stagger vs. Coefficient routing results for GAPT-SUX during 2011. Red data are the Straddle-Stagger data, blue data are the Coefficient Routing data, and black data are the observed data.**



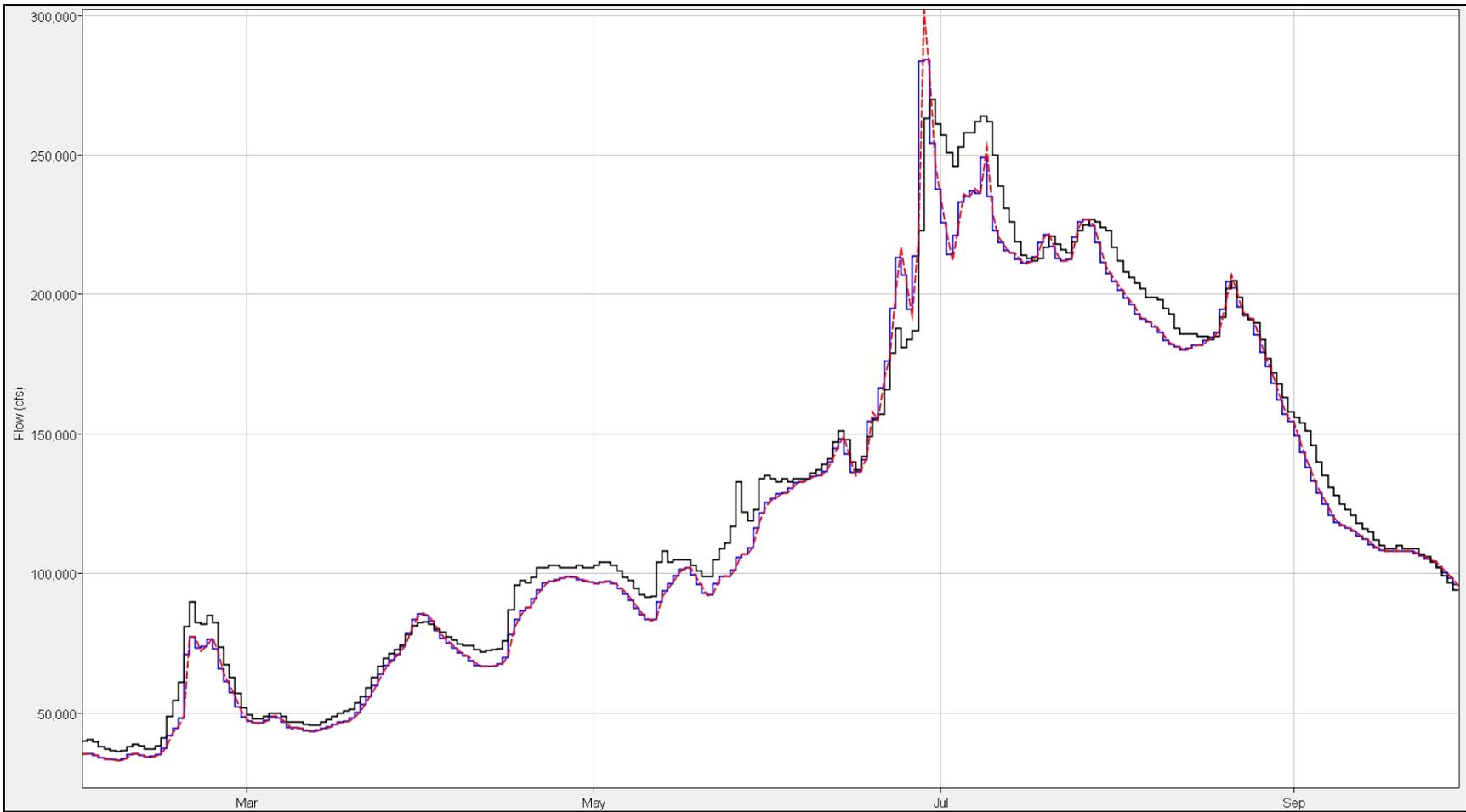
**Figure 8-41: Straddle-Stagger vs. Coefficient routing results for SUX-OMA during 2011. Red data are the Straddle-Stagger data, blue data are the Coefficient Routing data, and black data are the observed data.**



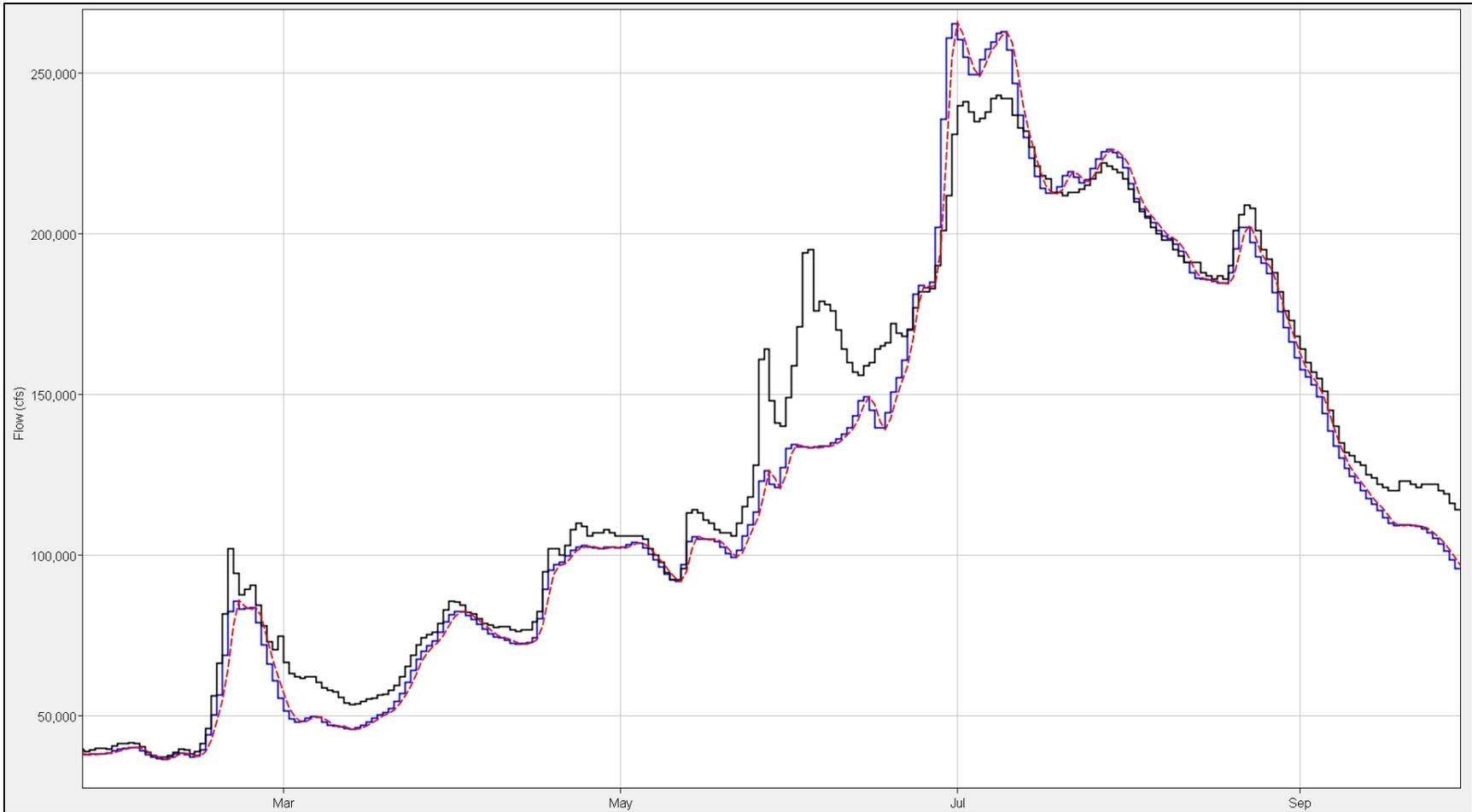
**Figure 8-42: Straddle-Stagger vs. Coefficient routing results for OMA-NCNE during 2011. Red data are the Straddle-Stagger data, blue data are the Coefficient Routing data, and black data are the observed data.**



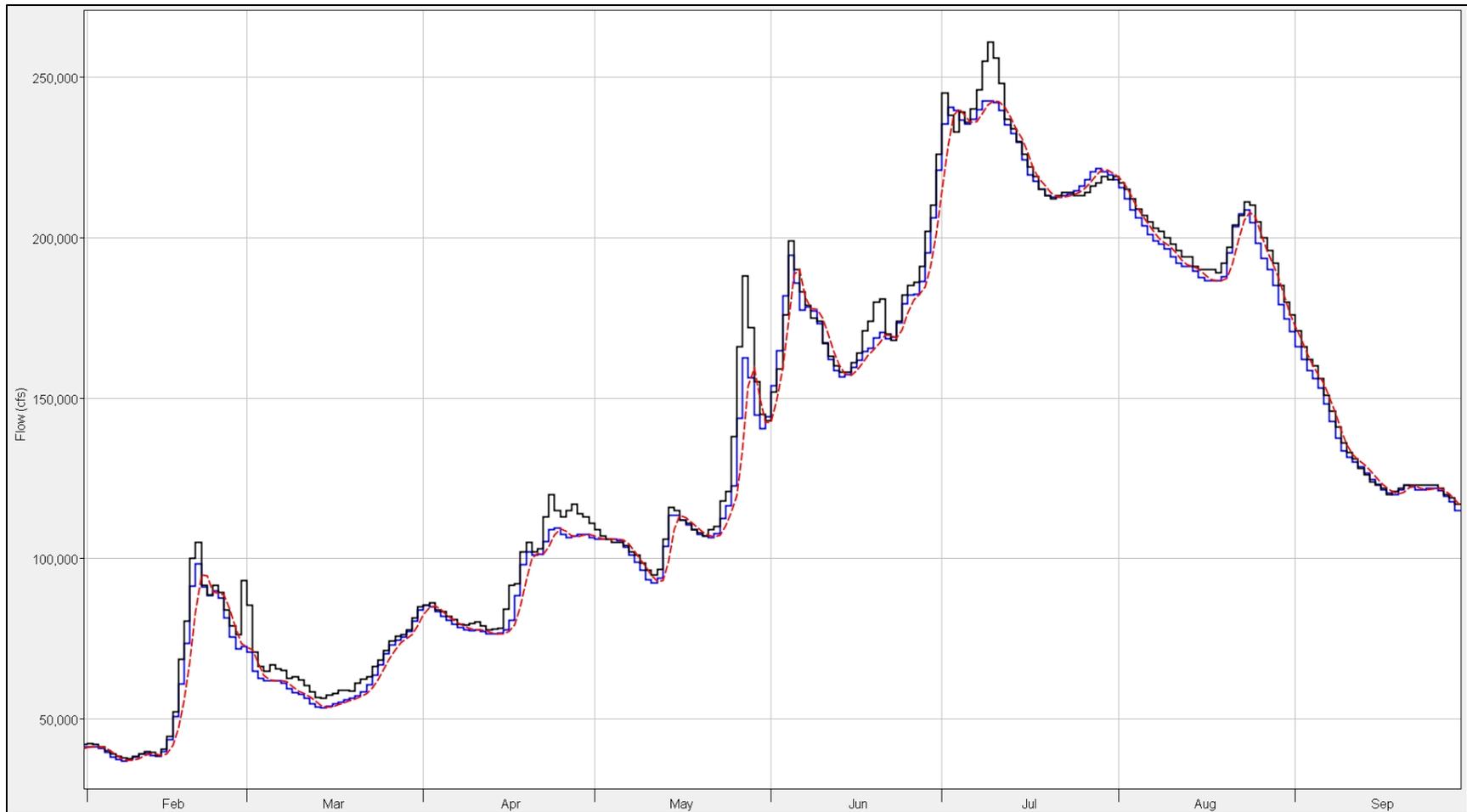
**Figure 8-43: Straddle-Stagger vs. Coefficient routing results for NCNE-RUNE during 2011. Red data are the Straddle-Stagger data, blue data are the Coefficient Routing data, and black data are the observed data.**



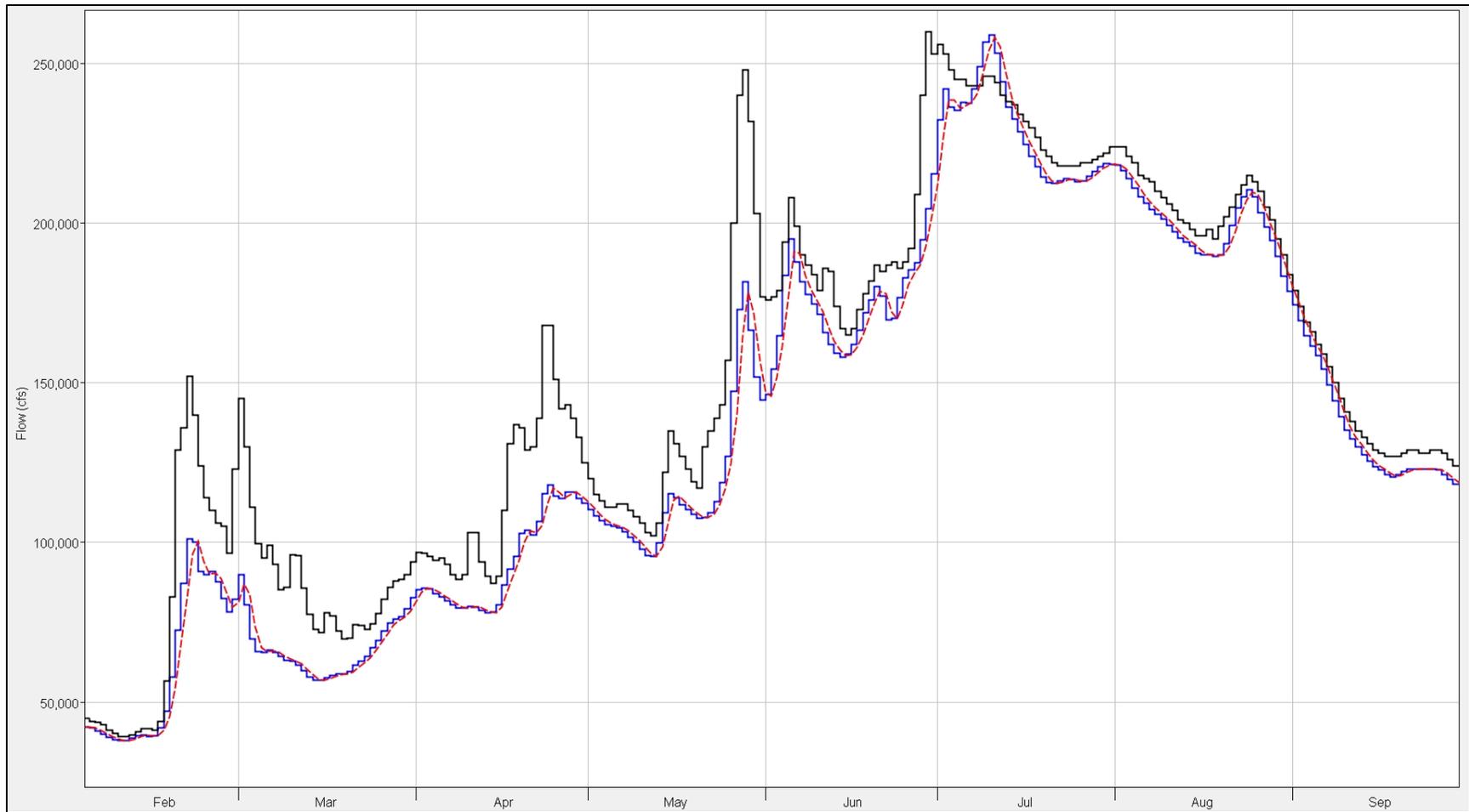
**Figure 8-44: Straddle-Stagger vs. Coefficient routing results for RUNE-STJ during 2011. Red data are the Straddle-Stagger data, blue data are the Coefficient Routing data, and black data are the observed data.**



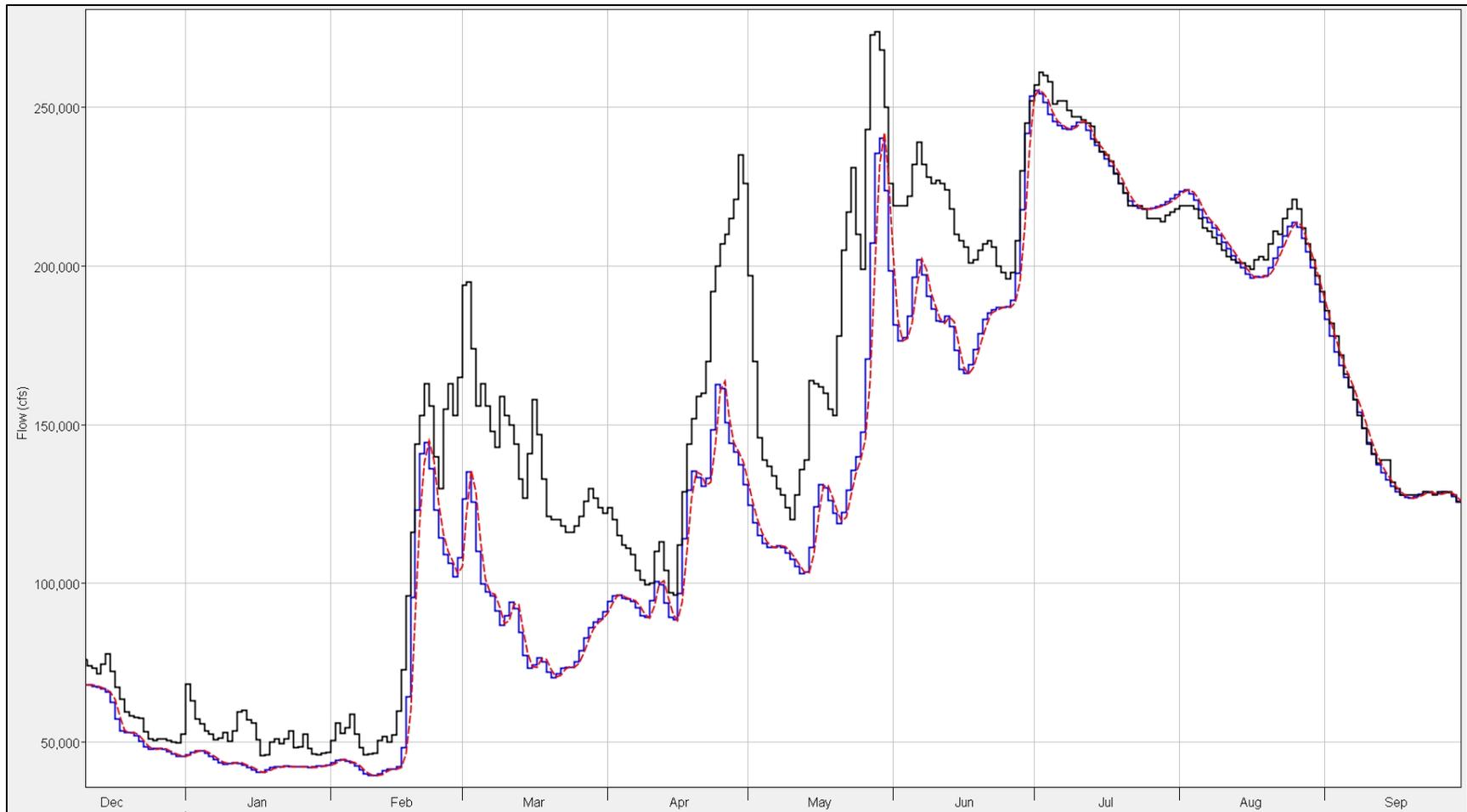
**Figure 8-45: Straddle-Stagger vs. Coefficient routing results for STJ-MKC during 2011. Red data are the Straddle-Stagger data, blue data are the Coefficient Routing data, and black data are the observed data.**



**Figure 8-46: Straddle-Stagger vs. Coefficient routing results for MKC-WVMO during 2011. Red data are the Straddle-Stagger data, blue data are the Coefficient Routing data, and black data are the observed data.**



**Figure 8-47: Straddle-Stagger vs. Coefficient routing results for WVMO-BNMO during 2011. Red data are the Straddle-Stagger data, blue data are the Coefficient Routing data, and black data are the observed data.**



**Figure 8-48: Straddle-Stagger vs. Coefficient routing results for BNMO-HEMO during 2011. Red data are the Straddle-Stagger data, blue data are the Coefficient Routing data, and black data are the observed data.**

9 APPENDIX C – ELEVATION-AREA-CAPACITY CURVES

Table 9-1: Fort Peck E-A-C curves.

Elevation (ft MSL)	1938		1961		1972		1986		2007	
	Area (acres)	Capacity (ac-ft)								
2030	0	0			0	0	0	0	0	0
2040	622	2,181			481	1,215	418	1,052	292	620
2050	2,655	16,992			1,523	11,910	1,480	11,057	1,479	9,482
2060	6,248	59,718			4,309	36,742	4,264	35,511	3,802	31,318
2070	11,523	146,693			10,211	105,667	10,043	103,630	9,100	88,046
2080	18,245	294,187			15,710	241,136	15,440	236,456	14,882	213,827
2090	25,561	513,490			21,562	421,887	21,086	413,557	20,221	385,404
2100	32,990	806,331			29,752	683,040	28,711	663,076	27,343	619,756
2110	41,900	1,177,239			37,914	1,018,844	37,047	990,681	35,709	933,443
2120	52,600	1,648,861			48,893	1,448,197	48,098	1,410,784	46,360	1,335,932
2130	63,342	2,229,806			61,013	1,999,578	60,371	1,955,609	58,579	1,861,991
2140	73,836	2,915,274			71,231	2,664,321	70,442	2,613,376	69,608	2,506,539
2150	84,458	3,707,094			80,779	3,422,616	79,683	3,362,492	79,022	3,252,813
2160	96,052	4,607,263			91,512	4,283,325	90,348	4,211,053	89,461	4,087,903
2170	109,574	5,633,247			105,630	5,261,144	104,794	5,178,658	103,394	5,045,002
2180	124,519	6,802,725			120,435	6,398,446	119,809	6,309,129	118,608	6,156,918
2190	140,022	8,125,401			135,050	7,670,171	134,099	7,573,749	132,175	7,415,889
2200	155,595	9,603,732			151,509	9,103,662	149,655	8,993,728	146,595	8,801,156
2210	171,820	11,239,176			167,734	10,700,542	164,592	10,565,907	163,400	10,349,820
2220	189,155	13,043,127			184,632	12,460,490	179,404	12,286,952	180,590	12,069,610
2230	207,287	15,024,749			203,422	14,398,579	200,565	14,169,679	201,130	13,964,500
2240	226,543	17,191,882			225,265	16,536,942	226,691	16,309,409	225,065	16,094,980
2250	246,486	19,557,492			248,844	18,908,686	245,898	18,687,731	245,405	18,462,840
2260					272,182	21,513,811	260,066	21,214,285	262,180	21,000,000

**Table 9-2: Garrison E-A-C curves.**

Elevation (ft MSL)	1960		1964		1969		1973		1979		1988		2011	
	Area (acres)	Capacity (ac-ft)												
1660					0	0			0	0	0	0	0	0
1670					50	45			16	12	31	43	7	5
1680					390	2,198			459	1,748	626	2,194	460	1,566
1690					2,943	13,823			2,837	13,947	3,464	14,592	2,632	10,805
1700					10,006	72,971			10,155	74,234	10,427	75,086	9,136	57,993
1710					20,424	223,716			20,454	225,156	20,738	226,141	19,492	196,970
1720					33,343	489,398			33,369	491,353	33,765	492,365	32,467	450,235
1730					50,527	901,503			50,696	903,851	50,705	904,837	47,931	848,553
1740					69,057	1,503,890			69,553	1,509,609	69,283	1,507,914	65,344	1,410,589
1750					86,123	2,279,887			86,735	2,291,547	86,512	2,289,440	83,684	2,156,262
1760					103,703	3,227,111			103,749	3,243,579	103,501	3,237,910	101,552	3,083,880
1770					120,663	4,353,155			120,533	4,366,654	120,369	4,359,411	118,070	4,186,230
1780				5,655,000	139,081	5,644,972			139,625	5,660,645	138,809	5,646,736	136,204	5,446,709
1790				7,164,000	162,084	7,147,374			162,477	7,169,843	161,295	7,139,184	157,953	6,913,512
1800				8,924,000	190,988	8,902,478			190,359	8,923,653	188,998	8,877,219	183,545	8,609,286
1810				11,015,000	223,593	10,977,253			221,396	10,985,496	219,955	10,921,980	215,125	10,589,550
1820				13,445,000	255,681	13,374,543			251,380	13,350,318	249,665	13,275,410	247,910	12,913,020
1830				16,227,000	287,896	16,092,973			280,843	16,014,439	280,520	15,916,490	280,485	15,547,850
1840				19,361,000	326,791	19,148,446			319,936	18,990,770	320,600	18,893,560	320,190	18,528,780
1850				22,855,000	368,139	22,635,302			365,281	22,429,151	364,265	22,331,620	364,935	21,956,050
1860				28,714,000	405,966	26,504,047			407,323	26,290,764	404,810	26,176,420		25,827,400

Table 9-3: Oahe E-A-C curves.

Elevation (ft MSL)	1958		1963		1976		1989		2010	
	Area (acres)	Capacity (ac-ft)								
1410	0	0	0	0	0	0	0	0	0	0
1420	388	479	141	95	94	104	107	73	44	8
1430	3,142	15,559	2,965	12,793	2,425	9,875	2,687	9,708	2,445	8,392
1440	8,144	69,670	7,597	64,642	6,754	54,231	6,995	55,360	6,485	50,377
1450	15,580	185,082	14,318	170,608	13,257	151,047	13,282	151,352	12,359	139,740
1460	24,499	385,321	22,982	356,213	21,539	324,427	20,735	322,090	19,962	299,110
1470	33,015	674,786	32,075	631,915	30,977	585,104	29,475	567,191	29,079	540,342
1480	42,040	1,047,136	41,438	998,783	40,571	944,926	39,166	912,471	39,042	881,474
1490	52,459	1,519,113	51,730	1,463,135	50,921	1,398,604	49,835	1,351,384	49,895	1,321,971
1500	63,459	2,098,234	62,784	2,035,614	62,188	1,965,744	61,420	1,909,988	61,082	1,879,701
1510	74,982	2,790,154	74,520	2,720,929	73,665	2,643,307	73,319	2,580,093	72,775	2,544,087
1520	87,973	3,602,003	87,132	3,528,919	85,492	3,440,773	85,462	3,376,665	85,356	3,335,994
1530	102,916	4,555,039	102,034	4,469,694	100,162	4,360,548	99,705	4,291,179	98,802	4,252,065
1540	119,558	5,665,381	118,947	5,575,386	117,493	5,451,212	116,560	5,373,030	115,352	5,314,664
1550	137,863	6,951,740	137,255	6,853,754	135,339	6,713,790	133,628	6,622,830	132,594	6,559,882
1560	159,859	8,433,616	159,673	8,332,298	157,881	8,170,491	155,510	8,049,792	152,181	7,968,796
1570	189,020	10,167,201	189,003	10,064,984	185,464	9,885,339	182,933	9,737,896	179,831	9,610,441
1580	221,114	12,222,428	221,076	12,120,275	217,121	11,890,306	213,150	11,711,030	212,675	11,569,960
1590	251,529	14,587,363	251,442	14,484,279	246,996	14,225,997	245,190	14,002,600	244,405	13,863,320
1600	283,829	17,259,233	283,800	17,155,559	279,626	16,839,532	281,010	16,618,390	279,520	16,461,230
1610	323,665	20,283,660	323,650	20,179,914	324,309	19,847,994	325,765	19,630,460	325,930	19,463,330
1620	402,345	23,750,945	402,327	23,646,924	372,842	23,337,619	384,075	23,136,960	385,585	22,982,900
1630		27,697,702	530,502	27,593,647	420,512	27,304,239	197,795	27,111,970		26,973,320
1640		27,722,841		27,618,786						

**Table 9-4: Big Bend E-A-C curves.**

Elevation (ft MSL)	1971		1975		1979		1983		1991		1997		2012	
	Area (acres)	Capacity (ac-ft)												
1340	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1350	1,093	3,568	1,066	3,411	970	3,058	983	3,090	1,105	3,688	836	2,445	816	2,256
1360	6,113	33,747	6,107	33,206	5,812	30,355	5,905	30,790	6,145	33,811	5,449	27,069	5,597	27,341
1370	12,913	127,749	12,863	127,413	12,483	121,194	12,594	123,029	12,720	128,285	11,747	113,160	12,035	115,925
1380	19,706	292,088	19,622	290,551	19,217	280,150	19,260	282,725	19,178	288,203	18,307	262,285	18,464	268,103
1390	26,616	522,019	26,556	520,053	25,960	505,600	25,910	508,251	25,623	511,864	24,659	479,172	24,532	484,949
1400	34,996	825,711	34,738	822,796	33,911	800,465	33,603	801,915	32,941	801,525	31,842	756,297	31,692	759,803
1410	45,576	1,223,842	45,317	1,216,897	44,899	1,186,424	44,679	1,183,202	43,898	1,173,817	43,146	1,119,548	43,478	1,122,745
1420	57,289	1,738,238	57,332	1,730,407	57,439	1,699,819	57,372	1,696,921	57,261	1,681,585	57,007	1,621,484	57,646	1,631,474
1430	68,992	2,369,650	69,295	2,363,543	69,598	2,334,939	69,284	2,330,055	70,189	2,318,770	70,615	2,259,568	71,120	2,275,184

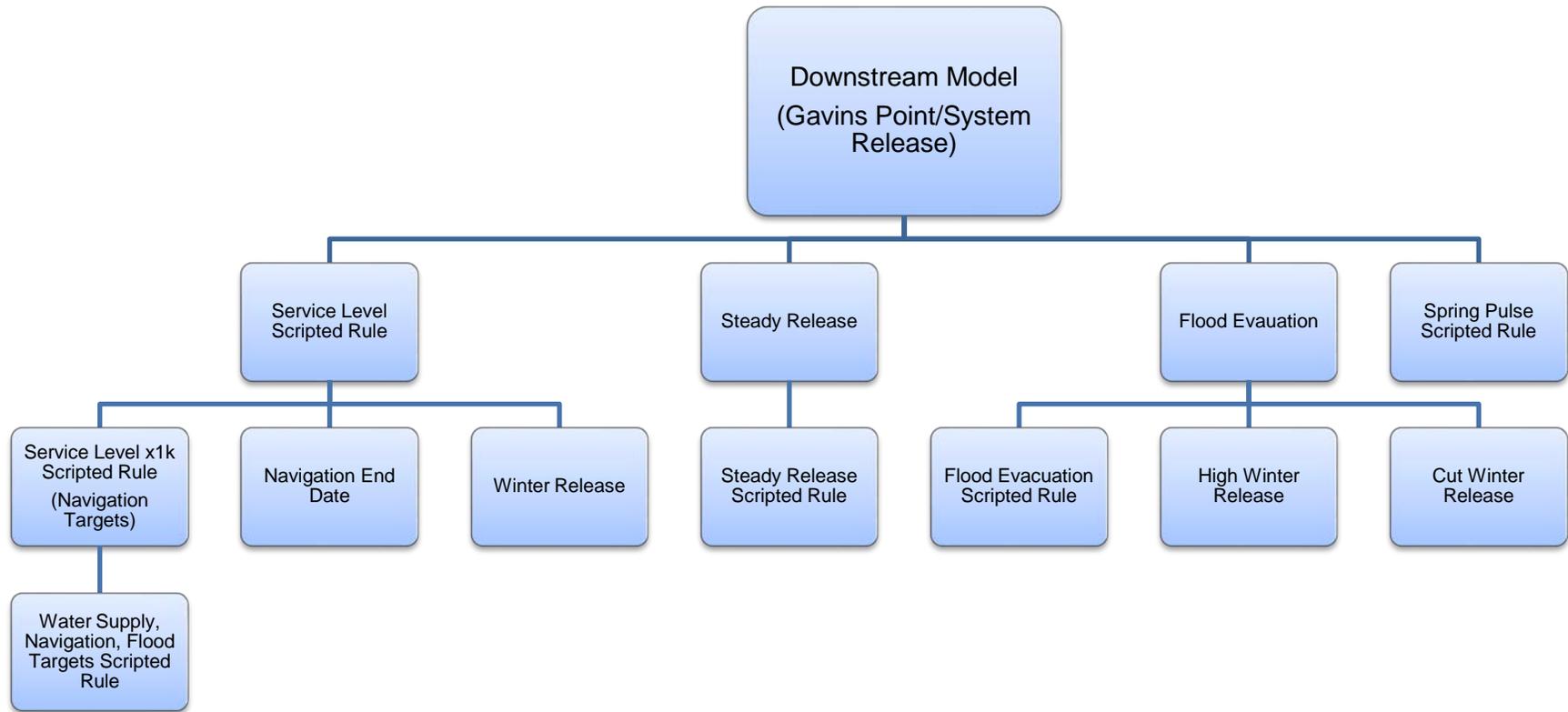
Table 9-5: Fort Randall E-A-C curves.

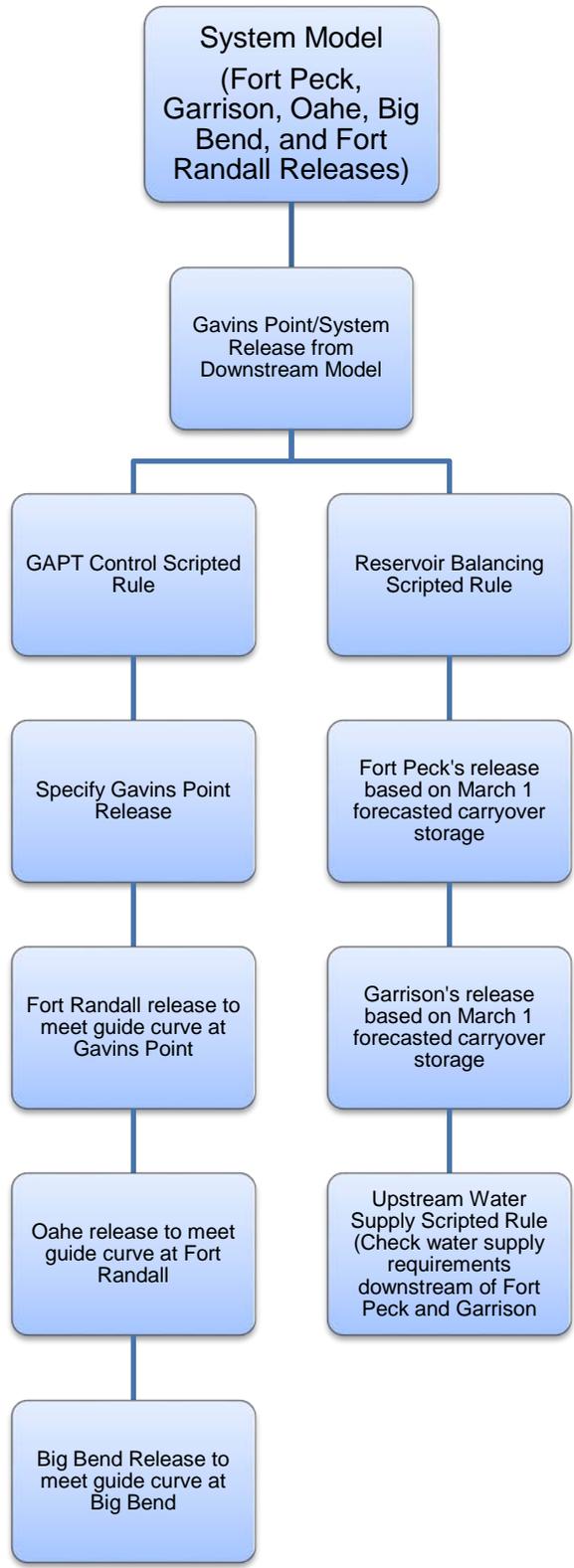
Elevation (ft MSL)	1953		1962		1967		1973		1977		1981		1986		1996		2011	
	Area (acres)	Capacity (ac-ft)																
1240	733	1,248	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1250	5,378	27,917	3,045	13,518	2,536	11,045	2,110	9,368	1,625	6,275	1,862	6,814	1,849	6,118	1,737	5,215	1,438	3,178
1260	11,207	110,127	9,424	78,250	8,697	66,648	7,717	55,824	7,425	46,563	7,623	48,313	7,523	49,913	7,637	48,875	7,486	46,443
1270	17,543	252,548	15,803	202,744	14,733	185,552	13,585	164,526	13,721	155,782	13,978	161,604	13,483	157,225	13,779	158,550	13,362	153,493
1280	24,279	461,328	22,931	394,927	21,299	361,757	20,220	328,193	20,377	321,323	20,390	328,222	19,907	319,981	20,061	324,578	19,276	313,738
1290	30,597	737,790	28,954	660,435	27,554	611,221	26,709	568,755	26,778	563,058	26,923	569,032	26,422	555,383	26,042	559,475	25,134	538,898
1300	36,908	1,073,281	33,869	973,089	31,578	911,014	30,707	860,337	30,696	854,868	30,700	860,594	30,678	846,562	30,297	843,949	28,936	814,716
1310	43,989	1,476,600	39,643	1,338,492	36,371	1,243,410	35,544	1,183,573	35,711	1,177,846	35,212	1,184,336	34,873	1,168,853	33,632	1,164,645	32,744	1,117,544
1320	51,503	1,953,477	45,069	1,765,786	43,107	1,640,182	42,281	1,572,921	41,966	1,570,252	40,523	1,567,781	39,787	1,544,734	37,911	1,517,486	36,100	1,469,353
1330	60,068	2,507,594	52,735	2,241,872	53,032	2,108,396	51,708	2,031,629	50,776	2,019,489	48,966	2,003,741	47,736	1,967,337	45,845	1,926,136	42,615	1,842,451
1340	70,172	3,156,164	64,790	2,824,241	66,946	2,704,215	65,053	2,610,434	64,039	2,589,583	62,908	2,560,989	61,155	2,504,173	59,783	2,439,591	57,772	2,329,032
1350	80,669	3,911,368	78,250	3,538,916	80,418	3,446,994	78,952	3,333,194	78,426	3,301,265	78,666	3,266,567	77,137	3,192,643	76,747	3,124,368	76,206	3,000,732
1360	89,985	4,768,539	89,564	4,387,428	90,871	4,309,984	90,090	4,187,121	90,186	4,155,848	90,887	4,125,597	90,214	4,044,439	89,808	3,971,266	89,779	3,849,085
1370	97,907	5,709,863	98,022	5,327,738	98,976	5,262,357	98,365	5,132,497	98,417	5,101,953	98,511	5,072,842	98,514	4,992,846	98,438	4,916,698	98,323	4,791,967
1380	105,559	6,726,414	105,837	6,347,239	106,605	6,289,050	105,995	6,153,801	106,001	6,123,540	106,029	6,094,354	105,987	6,013,896	106,176	5,939,141	106,236	5,814,844
1390	113,668	7,821,409	113,602	7,444,414	114,507	7,394,667	113,733	7,252,463	113,695	7,222,043	113,868	7,193,985	114,284	7,113,260	114,052	7,040,305	114,126	6,916,642

**Table 9-6: Gavins Point E-A-C curves.**

Elevation (ft MSL)	1966		1970		1975		1980		1985		1995		2007		2011	
	Area (acres)	Capacity (ac-ft)														
1160		0		0	0	0	0	0	0	0	0	0	0	0	0	0
1170		3,912		3,695	933	3,695	727	2,683	500	1,707	451	1,053	371	728	232	325
1180		28,393		26,426	4,252	26,380	3,976	22,650	3,489	17,687	3,486	15,631	3,393	14,543	2,855	11,211
1190		102,878		98,340	10,889	98,041	10,727	92,726	10,768	82,480	10,276	74,110	9,921	71,711	9,828	61,148
1200		265,196		255,885	20,666	252,880	20,313	245,285	20,234	239,709	19,713	223,547	18,819	215,126	18,259	209,203
1210		535,314		521,648	32,356	516,783	31,961	503,764	31,414	491,701	30,880	469,928	29,956	450,070	28,552	428,033
1220		920,674		902,211	44,331	901,209	44,257	886,525	42,323	867,354	42,677	841,701	43,373	815,335	41,878	782,807
1230					56,040	1,402,946	56,103	1,388,132	51,114	1,333,327	56,132	1,322,734			54,625	1,265,235

## 10 APPENDIX D – RESSIM MODELS FLOW-CHART





## 11 APPENDIX E – SERVICE LEVEL STATE VARIABLE

```

#####
##### STATE VARIABLE SCRIPT INITIALIZATION SECTION
#####

'''
Description: Version 2015-05-07 of the service level state variable script. It assesses the system using the procedure in the Master Manual and sets the
service level for navigation, the season length of the navigation season, and the winter releases for GAPT.
'''

from __future__ import with_statement
from bisect import bisect
from hec.client import ClientApp
from hec.heclib.dss import HecDss
from hec.heclib.util import HecTime
from hec.model import RunTimeStep
from hec.script import Constants
from hec.script import Plot
from hec.script import ResSim
from java.io import File
from java.util.prefs import Preferences
from javax.swing import JFileChooser
from javax.swing import JOptionPane
from zipfile import ZipFile
import os
import sys

# -----
# Set alternative depletions to read correct DSS datasets
# -----

Depletions = 'Present' # If Depletions = Present, then script will use local flow datasets that have been adjusted for present level depletions
# If Depletions = Historic, then script will use local flow datasets that are based on historic level depletions

#
# load the ResSimUtils module
#
wsDir = ResSim.getWatershed().getWorkspacePath()
sharedDir = os.path.normpath(os.path.join(wsDir, "shared"))
if sharedDir not in sys.path : sys.path.append(sharedDir)
import ResSimUtils

configText = ""
#
# Service Level Storage Dates
#
SERVICE_LEVEL_START_DATE = "MAR_15" # Don't store service level before this date
SERVICE_LEVEL_END_OFFSET = 0 # Don't store service level after this many days before navigation season end date
#
# MAR 15 Service Level Thresholds: Current Storage in MAF
#
MAR_15_MIN_SERVICE_LOWER_THRESHOLD = 31.0 # No service below this, min service above
MAR_15_MIN_SERVICE_UPPER_THRESHOLD = 49.0 # Interpolate between min & full service above this
MAR_15_FULL_SERVICE_LOWER_THRESHOLD = 54.5 # Full service at or above this
#

```

```

# JUL 01 Service Level Thresholds: Current Storage in MAF
#
JUL_01_MIN_SERVICE_LOWER_THRESHOLD = 0.0 # No service below this, min service above
JUL_01_MIN_SERVICE_UPPER_THRESHOLD = 50.5 # Interpolate between min & full service above this
JUL_01_FULL_SERVICE_LOWER_THRESHOLD = 57.0 # Full service at or above this
#
# Navigation Season End Dates
#
EARLY_NAV_END_DATE_1 = "OCT_01"
EARLY_NAV_END_DATE_2 = "NOV_01"
NORMAL_NAV_END_DATE = "DEC_01"
EXTENDED_NAV_END_DATE = "DEC_10"
BELOW_MEDIAN_FULL_SVC_STEADY_RELEASE = 34.3 # Steady state release in KCFS when service level is at or above full service and forecast runoff is median or above
BELOW_MEDIAN_MIN_SVC_STEADY_RELEASE = 28.3 # Steady state release in KCFS when service level is at or below minimum service and forecast runoff is median or above
MEDIAN_OR_ABOVE_FULL_SVC_STEADY_RELEASE = 31.6 # Steady state release in KCFS when service level is at or above full service and forecast runoff is below median
MEDIAN_OR_ABOVE_MIN_SVC_STEADY_RELEASE = 25.6 # Steady state release in KCFS when service level is at or below minimum service and forecast runoff below median

# Navigation Season Thresholds: Current Storage in MAF
#
EARLY_NAV_END_DATE_1_UPPER_THRESHOLD = 36.5 # Early nav end date 1 below this, interpolate above
EARLY_NAV_END_DATE_2_LOWER_THRESHOLD = 41.0 # Early nav end date 2 above this, interpolate below
EARLY_NAV_END_DATE_2_UPPER_THRESHOLD = 46.8 # Early nav end date 2 below this, interpolate above
NORMAL_NAV_END_DATE_LOWER_THRESHOLD = 51.5 # Normal nav end date above this, interpolate below
EXTENDED_NAV_END_DATE_LOWER_THRESHOLD = 60.0 # Extended nav end date at or above this, normal nav end date below
#
# Winter Releases in KCFS
#
LOW_WINTER_RELEASE = 12.0
MID_WINTER_RELEASE = 17.0
HIGH_WINTER_RELEASE = 20.0
#
# Winter Release Thresholds: Current Storage in MAF
#
LOW_WINTER_RELEASE_UPPER_THRESHOLD = 55.0 # low winter release at or below this, interpolate above
MID_WINTER_RELEASE_LOWER_THRESHOLD = 58.0 # Mid winter release at or above this, interpolate below
MID_WINTER_RELEASE_UPPER_THRESHOLD = 60.0 # Mid winter release below this, interpolate above
HIGH_WINTER_RELEASE_LOWER_THRESHOLD = 60.0 # High winter release at or above this
#
# Steady-State and Cycled Release Values
#
STEADY_STATE_COMPUTATION_THRESHOLD = 44.0 # Service level in KCFS above which to not compute steady state & cycled releases
CYCLED_RELEASE_CUTBACK = 6.0 # Amount to cut back from steady state in KCFS
CYCLED_RELEASE_STEADY_PERIOD = 1 # Number of days of steady-state flow per cycle
CYCLED_RELEASE_CUTBACK_PERIOD = 1 # Number of days of cutback flow per cycle
'''
exec(configText)
configFilename = None

FTT = "Flow To Target"
SRFTT = "Steady Release -> Flow To Target"
CRFTT = "Cycled Release -> Flow To Target"
scenarios = [FTT, SRFTT, CRFTT]
scenario = JOptionPane.showInputDialog(

```

```
None,          # dialog parent component
"Select Scenario", # message
"State Variable Script", # title
JOptionPane.PLAIN_MESSAGE, # message type (sets default icon)
None,          # icon to override default
scenarios,     # list of selections
FTT)           # initial selection
```

```
if scenario :
    preferences = Preferences.userRoot().node("script").node("ResSim").node("ServiceLevelScript")
    jfc = JFileChooser()
    lastConfigFile = preferences.get("lastConfigFile", "")
    if lastConfigFile : jfc.setSelectedFile(File(lastConfigFile))
    result = jfc.showDialog(None, "Choose Configuration File")
    if result == JFileChooser.APPROVE_OPTION :
        configFilename = jfc.getSelectedFile().getCanonicalPath()
        preferences.put("lastConfigFile", configFilename)
        with open(configFilename, "r") as f : exec(f.read())
```

```
#
# initialization function. optional.
# set up tables and other things that only need to be performed once at the start of the compute.
#
# variables that are passed to this script during the compute initialization:
#     currentVariable - the StateVariable that holds this script
#     network - the ResSim network
#
```

```
def initStateVariable(currentVariable, network):
    global MAR_15_MIN_SERVICE_LOWER_THRESHOLD
    global MAR_15_MIN_SERVICE_UPPER_THRESHOLD
    global MAR_15_FULL_SERVICE_LOWER_THRESHOLD
    global JUL_01_MIN_SERVICE_LOWER_THRESHOLD
    global JUL_01_MIN_SERVICE_UPPER_THRESHOLD
    global JUL_01_FULL_SERVICE_LOWER_THRESHOLD
    global EARLY_NAV_END_DATE_1
    global EARLY_NAV_END_DATE_2
    global NORMAL_NAV_END_DATE
    global EXTENDED_NAV_END_DATE
    global EARLY_NAV_END_DATE_1_UPPER_THRESHOLD
    global EARLY_NAV_END_DATE_2_LOWER_THRESHOLD
    global EARLY_NAV_END_DATE_2_UPPER_THRESHOLD
    global NORMAL_NAV_END_DATE_LOWER_THRESHOLD
    global EXTENDED_NAV_END_DATE_LOWER_THRESHOLD
    global LOW_WINTER_RELEASE
    global MID_WINTER_RELEASE
    global HIGH_WINTER_RELEASE
    global LOW_WINTER_RELEASE_UPPER_THRESHOLD
    global MID_WINTER_RELEASE_LOWER_THRESHOLD
    global MID_WINTER_RELEASE_UPPER_THRESHOLD
    global HIGH_WINTER_RELEASE_LOWER_THRESHOLD
    global STEADY_STATE_COMPUTATION_THRESHOLD
    global BELOW_MEDIAN_FULL_SVC_STEADY_RELEASE
    global BELOW_MEDIAN_MIN_SVC_STEADY_RELEASE
```

```
global MEDIAN_OR_ABOVE_FULL_SVC_STEADY_RELEASE
global MEDIAN_OR_ABOVE_MIN_SVC_STEADY_RELEASE
global CYCLED_RELEASE_CUTBACK
global CYCLED_RELEASE_STEADY_PERIOD
global CYCLED_RELEASE_CUTBACK_PERIOD
global LAST_NAV_END_DATE
```

```
if not scenario : return False
```

```
separator = "=====
```

```
verbose = False # set to False to reduce console output volume
```

```
useExtraDates = True # for extra assessment dates between MAY_01 and JUL_01
```

```
outputByDate = {}
```

```
outputDate = ""
```

```
userInfo = {
```

```
    "outputByDate" : outputByDate,
```

```
    "outputDate" : outputDate
```

```
}
```

```
def setOutputDate(yr, mon, day) :
```

```
    outputDate = "%4.4d%2.2d%2.2d" % (yr, mon, day)
```

```
    storedUserInfo = currentVariable.varGet("userInfo")
```

```
    if storedUserInfo : thisUserInfo = storedUserInfo
```

```
    else : thisUserInfo = userInfo
```

```
    thisUserInfo["outputDate"] = outputDate
```

```
    thisUserInfo["outputByDate"][outputDate] = []
```

```
def output(message) :
```

```
    storedUserInfo = currentVariable.varGet("userInfo")
```

```
    if storedUserInfo : thisUserInfo = storedUserInfo
```

```
    else : thisUserInfo = userInfo
```

```
    outputDate = thisUserInfo["outputDate"]
```

```
    outputByDate = thisUserInfo["outputByDate"][outputDate]
```

```
    outputByDate.append(message)
```

```
setOutputDate(0,0,0)
```

```
output(separator)
```

```
output("Scenario : %s" % scenario)
```

```
output("Configuration file: %s" % (configFilename, "<No Configuration File Used>")[configFilename is None])
```

```
output("Configurable parameters are as follows:")
```

```
output("\tSERVICE_LEVEL_START_DATE\t= %s" % SERVICE_LEVEL_START_DATE)
```

```
output("\tSERVICE_LEVEL_END_OFFSET\t= %s" % SERVICE_LEVEL_END_OFFSET)
```

```
output("")
```

```
output("\tMAR_15_MIN_SERVICE_LOWER_THRESHOLD\t= %s MAF" % MAR_15_MIN_SERVICE_LOWER_THRESHOLD)
```

```
output("\tMAR_15_MIN_SERVICE_UPPER_THRESHOLD\t= %s MAF" % MAR_15_MIN_SERVICE_UPPER_THRESHOLD)
```

```
output("\tMAR_15_FULL_SERVICE_LOWER_THRESHOLD\t= %s MAF" % MAR_15_FULL_SERVICE_LOWER_THRESHOLD)
```

```
output("")
```

```
output("\tJUL_01_MIN_SERVICE_LOWER_THRESHOLD\t= %s MAF" % JUL_01_MIN_SERVICE_LOWER_THRESHOLD)
```

```
output("\tJUL_01_MIN_SERVICE_UPPER_THRESHOLD\t= %s MAF" % JUL_01_MIN_SERVICE_UPPER_THRESHOLD)
```

```
output("\tJUL_01_FULL_SERVICE_LOWER_THRESHOLD\t= %s MAF" % JUL_01_FULL_SERVICE_LOWER_THRESHOLD)
```

```
output("")
```

```
output("\tEARLY_NAV_END_DATE_1\t= %s" % EARLY_NAV_END_DATE_1)
```

```
output("\tEARLY_NAV_END_DATE_2\t= %s" % EARLY_NAV_END_DATE_2)
```

```
output("\tNORMAL_NAV_END_DATE\t= %s" % NORMAL_NAV_END_DATE)
```

```

output("\tEXTENDED_NAV_END_DATE\t= %s" % EXTENDED_NAV_END_DATE)
output("")
output("\tEARLY_NAV_END_DATE_1_UPPER_THRESHOLD\t= %s MAF" % EARLY_NAV_END_DATE_1_UPPER_THRESHOLD)
output("\tEARLY_NAV_END_DATE_2_LOWER_THRESHOLD\t= %s MAF" % EARLY_NAV_END_DATE_2_LOWER_THRESHOLD)
output("\tEARLY_NAV_END_DATE_2_UPPER_THRESHOLD\t= %s MAF" % EARLY_NAV_END_DATE_2_UPPER_THRESHOLD)
output("\tNORMAL_NAV_END_DATE_LOWER_THRESHOLD\t= %s MAF" % NORMAL_NAV_END_DATE_LOWER_THRESHOLD)
output("\tEXTENDED_NAV_END_DATE_LOWER_THRESHOLD\t= %s MAF" % EXTENDED_NAV_END_DATE_LOWER_THRESHOLD)
output("")
output("\tLOW_WINTER_RELEASE\t= %s KCFS" % LOW_WINTER_RELEASE)
output("\tMID_WINTER_RELEASE\t= %s KCFS" % MID_WINTER_RELEASE)
output("\tHIGH_WINTER_RELEASE\t= %s KCFS" % HIGH_WINTER_RELEASE)
output("")
output("\tLOW_WINTER_RELEASE_UPPER_THRESHOLD\t= %s MAF" % LOW_WINTER_RELEASE_UPPER_THRESHOLD)
output("\tMID_WINTER_RELEASE_LOWER_THRESHOLD\t= %s MAF" % MID_WINTER_RELEASE_LOWER_THRESHOLD)
output("\tMID_WINTER_RELEASE_UPPER_THRESHOLD\t= %s MAF" % MID_WINTER_RELEASE_UPPER_THRESHOLD)
output("\tHIGH_WINTER_RELEASE_LOWER_THRESHOLD\t= %s MAF" % HIGH_WINTER_RELEASE_LOWER_THRESHOLD)
output("")
output("\tSTEADY_STATE_COMPUTATION_THRESHOLD\t= %s MAF" % STEADY_STATE_COMPUTATION_THRESHOLD)
output("\tBELOW_MEDIAN_FULL_SVC_STEADY_RELEASE\t= %s KCFS" % BELOW_MEDIAN_FULL_SVC_STEADY_RELEASE)
output("\tBELOW_MEDIAN_MIN_SVC_STEADY_RELEASE\t= %s KCFS" % BELOW_MEDIAN_MIN_SVC_STEADY_RELEASE)
output("\tMEDIAN_OR_ABOVE_FULL_SVC_STEADY_RELEASE\t= %s KCFS" % MEDIAN_OR_ABOVE_FULL_SVC_STEADY_RELEASE)
output("\tMEDIAN_OR_ABOVE_MIN_SVC_STEADY_RELEASE\t= %s KCFS" % MEDIAN_OR_ABOVE_MIN_SVC_STEADY_RELEASE)
output("\tCYCLED_RELEASE_CUTBACK\t= %s KCFS" % CYCLED_RELEASE_CUTBACK)
output("\tCYCLED_RELEASE_STEADY_PERIOD\t= %s day(s)" % CYCLED_RELEASE_STEADY_PERIOD)
output("\tCYCLED_RELEASE_CUTBACK_PERIOD\t= %s day(s)" % CYCLED_RELEASE_CUTBACK_PERIOD)

```

```

# -----
# DSS File/Data
# -----

```

```

fPart = ResSimUtils.getFPart(ResSimUtils.getSelectedAlternativeNames()[0])
dssFilename = ResSimUtils.getSimulationDSSFileName()
simulation = ResSimUtils.getSimulation()
dssFile = HecDss.open(dssFilename)
dssFile.setTimeWindow(simulation.getLookbackDateString(), simulation.getEndDateString())

```

```

if Depletions == 'Present':

```

```

    RBMT_Local = dssFile.read('/MISSOURI RIVER/RBMT/FLOW-LOCAL//1DAY/FWOP: CALCULATED/').getData().values
    FTPK_Local = dssFile.read('/MISSOURI RIVER/FTPK/FLOW-LOCAL//1DAY/FWOP: CALCULATED/').getData().values
    WPMT_Local = dssFile.read('/MISSOURI RIVER/WPMT/FLOW-LOCAL//1DAY/FWOP: CALCULATED/').getData().values
    CLMT_Local = dssFile.read('/MISSOURI RIVER/CLMT/FLOW-LOCAL//1DAY/FWOP: CALCULATED/').getData().values
    GARR_Local = dssFile.read('/MISSOURI RIVER/GARR/FLOW-LOCAL//1DAY/FWOP: CALCULATED/').getData().values
    BIS_Local = dssFile.read('/MISSOURI RIVER/BIS/FLOW-LOCAL//1DAY/FWOP: CALCULATED/').getData().values
    OAHE_Local = dssFile.read('/MISSOURI RIVER/OAHE/FLOW-LOCAL//1DAY/FWOP: CALCULATED/').getData().values
    BEND_Local = dssFile.read('/MISSOURI RIVER/BEND/FLOW-LOCAL//1DAY/FWOP: CALCULATED/').getData().values
    FTRA_Local = dssFile.read('/MISSOURI RIVER/FTRA/FLOW-LOCAL//1DAY/FWOP: CALCULATED/').getData().values
    GAPT_Local = dssFile.read('/MISSOURI RIVER/GAPT/FLOW-LOCAL//1DAY/FWOP: CALCULATED/').getData().values

```

```

elif Depletions == 'Historic':

```

```

    RBMT_Local = dssFile.read('/MISSOURI RIVER/RBMT/FLOW-OUT//1DAY/HISTORIC: OBSERVED/').getData().values
    FTPK_Local = dssFile.read('/MISSOURI RIVER/FTPK/FLOW-LOCAL//1DAY/HISTORIC: CALCULATED/').getData().values
    WPMT_Local = dssFile.read('/MISSOURI RIVER/WPMT/FLOW-LOCAL//1DAY/HISTORIC: CALCULATED/').getData().values
    CLMT_Local = dssFile.read('/MISSOURI RIVER/CLMT/FLOW-LOCAL//1DAY/HISTORIC: CALCULATED/').getData().values
    GARR_Local = dssFile.read('/MISSOURI RIVER/GARR/FLOW-LOCAL//1DAY/HISTORIC: CALCULATED/').getData().values

```

```

BIS_Local = dssFile.read('/MISSOURI RIVER/BIS/FLOW-LOCAL//1DAY/HISTORIC: CALCULATED/').getData().values
OAHE_Local = dssFile.read('/MISSOURI RIVER/OAHE/FLOW-LOCAL//1DAY/HISTORIC: CALCULATED/').getData().values
BEND_Local = dssFile.read('/MISSOURI RIVER/BEND/FLOW-LOCAL//1DAY/HISTORIC: CALCULATED/').getData().values
FTRA_Local = dssFile.read('/MISSOURI RIVER/FTRA/FLOW-LOCAL//1DAY/HISTORIC: CALCULATED/').getData().values
GAPT_Local = dssFile.read('/MISSOURI RIVER/GAPT/FLOW-LOCAL//1DAY/HISTORIC: CALCULATED/').getData().values
else:
    raise AssertionError, 'Depletions error. Specify as either Present or Historic.'

```

```

# Routing Coefficients
coeffDict = { 'RBMT': [0.000, 1.000], # RBMT to FTPK coefficients
              'WPMT' : [0.237, 0.444, 0.319], # WPMT to CLMT coefficients
              'CLMT': [0.237, 0.444, 0.319], # CLMT to GARR coefficients
              'BIS'  : [0.057, 0.503, 0.440] # BIS to OAHE coefficients
            }

```

```

#
# DRM forecast months
#
months = ["Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul"]
months.append(["%2.2d" % (i+1) for i in range(len(months[0])]))
#
# defined service levels
#
NO_SERVICE    = Constants.UNDEFINED
MINIMUM_SERVICE = 29.
FULL_SERVICE   = 35.
#
# defined julian days
#
FEB_01 = 32 # early assessment date, currently no action taken
MAR_01 = 60 # end of winter release, spring pulse assessment date, currently no action taken
MAR_15 = 74 # initial summer assessment date, sets service level
MAR_19 = 78
MAY_01 = 121 # interim summer assessment date
MAY_15 = 135 # start of steady release
JUN_01 = 152 # interim summer assessment date
JUL_01 = 182 # final summer assessment date, sets service level and end of navigation season, end of steady release
JUL_15 = 196 # steady/cycled release end date
JUL_21 = 189 # interim summer assessment date
AUG_01 = 213 # interim assessment date
SEP_01 = 244 # winter release planning assessment date, sets GAPT average winter release target
OCT_01 = 274 # interim assessment date
NOV_01 = 305 # interim assessment date
#
# assessment dates
#
assessmentDates = (FEB_01, MAR_15, MAY_01, JUN_01, JUL_01, AUG_01, SEP_01)
extraAssessmentDates = []
#
# defined quantiles to use in analysis
#
quantiles = {

```

```

        .10 : (1, 10), # lower decile
        .25 : (1, 4), # lower quartile
        .50 : (1, 2), # median
        .75 : (3, 4), # upper quartile
        .90 : (9, 10)} # upper decile
#
# project identifiers and names (BEND/Lake Sharpe is included in FTRA/Lake Francis Case)
#
projects = [
    ["FTPK", "Fort Peck Lake"],
    ["GARR", "Lake Sakakawea"],
    ["OAHE", "Lake Oahe"],
    ["FTRA", "Lake Francis Case"],
    ["GAPT", "Lewis and Clark Lake"]]
projectIds, projectNames = zip(*projects)
#
# historical runoff in KAF at projects for specified months and quantiles
# (data from "Annual" tab of "ForecastedRemainingCalendarYearRunoff_ExampleProcedure.xlsx")
#
historicRunoffText = \

```

```

'''

```

	FTPK	FTPK	FTPK	FTPK	FTPK	GARR	GARR	GARR	GARR	GARR	OAHE	OAHE	OAHE	OAHE	OAHE	FTRA	FTRA	FTRA	FTRA	FTRA	GAPT	GAPT	GAPT	GAPT	GAPT
	.90	.75	.50	.25	.10	.90	.75	.50	.25	.10	.90	.75	.50	.25	.10	.90	.75	.50	.25	.10	.90	.75	.50	.25	.10
Month	KAF																								
JAN	295	270	250	220	210	325	300	260	220	135	25	10	0	-10	-20	0	-5	-10	-15	-20	105	95	85	75	70
FEB	395	360	345	305	290	415	380	320	275	255	125	115	95	75	65	60	50	45	40	35	165	140	115	100	95
MAR	650	595	475	420	400	1090	1000	950	875	755	740	625	480	420	405	305	270	245	150	140	250	225	215	180	170
APR	790	715	600	485	470	1355	1240	870	705	575	545	520	405	180	170	425	285	155	90	85	225	185	145	125	115
MAY	1590	1445	1180	955	845	1840	1685	1325	1110	1055	360	305	195	130	115	220	180	140	65	60	345	300	160	140	130
JUN	2465	2245	1810	1480	1195	3425	3130	3095	2635	2205	1265	1010	780	275	255	150	140	135	125	115	290	240	175	150	140
JUL	1205	1100	840	665	610	2715	2480	1860	1585	1080	215	185	160	140	125	90	80	70	35	25	215	175	100	85	80
AUG	450	410	315	285	270	835	760	595	505	360	110	95	75	65	50	85	75	65	25	15	185	165	90	75	65
SEP	375	340	295	255	245	570	520	460	390	160	150	125	95	75	65	80	70	30	0	-10	135	120	95	80	70
OCT	525	480	430	340	320	645	590	495	420	390	95	65	35	15	5	30	15	0	-20	-30	155	140	120	110	100
NOV	415	375	360	330	315	515	470	390	330	295	215	110	60	25	15	40	30	20	-15	-25	140	130	120	105	95
DEC	345	315	300	260	230	270	245	180	150	135	-45	-65	-80	-90	-100	15	10	5	-30	-40	90	85	80	75	70

```

'''

```

```

'''Adds historicRunoffText to a dictionary '''
historicRunoff = {}
lines = historicRunoffText.strip().split("\n")
for i in range(3, len(lines)) :
    fields = lines[i].split()[1:]
    for j in range(len(projectIds)) :
        vals = map(float, fields[5*j:5*j+5][::-1])
        print i, projectIds[j], vals
        historicRunoff.setdefault(projectIds[j], []).append(vals)
#
# initialize the service level to NO_SERVICE for the entire simulation period
#
rtw = ResSim.getCurrentModule().getSimulation().getRunTimeWindow()
rts = RunTimeStep(rtw)
currentVariable.setValueRange(rts, rtw.getLookbackTime(), rtw.getEndTime(), NO_SERVICE)
#

```

```

# load the system level curves from DSS
#
shared = ResSim.getWatershed().getSharedDirectory()
dssFilename = os.path.join(shared, "Input_Data.dss")
output(separator)
output("Processing DSS file %s" % dssFilename)
serviceLevelPathname = "/MISSOURI RIVER/MAINSTEM/DATE-VOLUME///SERVICE LEVEL/"
thresholdsPathname = "/MISSOURI RIVER/MAINSTEM/DATE-VOLUME///THRESHOLDS/"
serviceLevelPdc = None
thresholdsPdc = None
errorMessage = None
LAST_NAV_END_DATE = {}
#
# verify the service level info is available
#
while True:
    if not os.path.exists(dssFilename) :
        errorMessage = "DSS file does not exist: %s" % dssFilename
        break
    dssFile = HecDss.open(dssFilename)
    pathnames = dssFile.getPathnameList()
    if serviceLevelPathname not in pathnames :
        errorMessage = "DSS file %s does not contain pathname %s" % (dssFilename, serviceLevelPathname)
        break
    if thresholdsPathname not in pathnames :
        errorMessage = "DSS file %s does not contain pathname %s" % (dssFilename, thresholdsPathname)
        break
    break
if errorMessage :
    output("ERROR : " + errorMessage)
    return False
#
# get the service level information
#
serviceLevelPdc = dssFile.get(serviceLevelPathname)
levelValues = [float(label) for label in serviceLevelPdc.labels]
thresholdsPdc = dssFile.get(thresholdsPathname)
thresholdCurves = {}
for i in range(len(thresholdsPdc.labels)) :
    thresholdCurves[thresholdsPdc.labels[i]] = i
#
# read the DRM forecast runoff information for Jan-Jul
#
fcstZipfileName = os.path.join(shared, "DRM_Forecast_text.zip")
output(separator)
output("Processing DRM forecast file %s" % fcstZipfileName)
fcstFlows = {}
zf = ZipFile(fcstZipfileName)
for itemname in zf.namelist() :
    if len(itemname) != 10 or not itemname.lower().startswith("fc") or not itemname.lower().endswith(".txt") :
        output("Ignoring zipped file %s" % itemname)
        continue
    try :

```

```

        year = int(itemname[2:6])
except :
    output("Ignoring zipped file %s" % itemname)
    continue
lines = zf.read(itemname).strip().split("\n")
for i in range(len(lines)) :
    fields = lines[i].split()
    if i == 0 :
        assert not len(fields) < 3 and fields[0] == "FORECAST" and fields[2] == "KA-F"
        if int(fields[1]) != year :
            output("WARNING : " + "Zipped file %s indicates year %s in header" % (itemname, fields[1]))
    elif i == 1 :
        assert not (len(fields) < 7 and fields[2:7] == list(projectIds))
    else :
        assert not len(fields) < 7 and fields[0] in months[0]
        fcstMonth = fields[0]
        mon, yr = fields[1].split("-")
        assert mon in months[1]
        if int(yr) != year :
            output("WARNING : " + "Zipped file %s indicates year %s at line %d" % (itemname, yr, i+1))
        for j in range(len(projectIds)) :
            try :
                val = float(fields[2+j])
            except :
                output("WARNING : " +
                    "Invalid discharge value (%s) for project %s at line %d in zipped file %s, using 0" % \
                    (fields[2+j], projectIds[j], i+1, itemname))
                val = 0
            fcstFlows.setdefault(year, {}).setdefault(projectIds[j], {}).setdefault(fcstMonth, []).append(val)

zf.close()
#
# process the forecast runoff into annual totals for forecast month through Jul for each project
#
years = fcstFlows.keys()
years.sort()
output("Forecast flows read for years %d - %d" % (years[0], years[-1]))
missingYears = []
for i in range(1, len(years)) :
    if years[i] - years[i-1] > 1 :
        missingYears.extend(range(years[i-1]+1, years[i]))
if missingYears :
    output("WARNING : " + "The following years are missing: %s" % ", ".join(missingYears))
output(separator)
#
# define functions
#
debug = False
def outputDebug(debug, arg1="", arg2="", arg3="", arg4="", arg5="", arg6="", arg7="", arg8="", arg9="", arg10="", arg11="", arg12="", arg13="", \
    arg14="", arg15=""):
    PassCounter = network.getComputePassCounter() # Pass number of simulation
    PassNumber = PassCounter + 1 # Pass number starts at 0 so add 1 to start it at 1
    DateTimeString = currentVariable.varGet('DateTimeString')
    if debug == True:

```

```

        print 'Pass Number =', PassNumber, ' ', DateTimeString, '\t', arg1, arg2, \
              arg3, arg4, arg5, arg6, arg7, arg8, arg9, arg10, arg11, arg12, arg13, arg14, arg15, '\n'

def getFraction(val, loVal, hiVal):
    return float(val - loVal) / (hiVal - loVal)

def historicQuantilesTotal(projectId, start, end) :
    """
    Returns the historic runoff (LD, LQ, M, UQ, UD) for the specified
    project and month range (1-12, inclusive)
    """
    userInfo = currentVariable.varGet("userInfo")
    historicRunoff = userInfo["historicRunoff"]
    return map(sum, zip(*historicRunoff[projectId][start-1:end]))

def getForecast(mon, year, day = 1) : # Day is an optional value
    global FTPK_Accum_Stor
    global GARR_Accum_Stor
    global OAHE_Accum_Stor
    global BEND_and_FTRA_Accum_Stor
    global GAPT_Accum_Stor

    """
    Determines the expected runoff for the remainder of the calendar year
    """
    def maxDay(mon, year) :
        """
        Return the max day for the month and year
        """
        maxday = None
        if mon in (1,3,5,7,8,10,12) :
            maxday = 31
        elif mon in (4,6,9,11) :
            maxday = 30
        elif year % 4 != 0 or (year % 100 == 0 and year % 400 != 0) :
            maxday = 28
        else :
            maxday = 29
        return maxday

    userInfo = currentVariable.varGet("userInfo")
    fcstFlows = userInfo["fcstFlows"] # DRM forecasts
    historicRunoff = userInfo["historicRunoff"] # Runoff statistics
    months = userInfo["months"]
    output = userInfo["output"]
    projectIds = userInfo["projectIds"]
    verbose = userInfo["verbose"] # Adds additional output to the console if set to True

    if verbose :
        output(separator)
        output("Getting forecast runoff for %4.4d/%2.2d/%2.2d" % (year, mon, day))

    totalFcstRunoff = 0

```

```

systemQuantiles = 5 * [0.]
# Track observed runoff and base Aug through Dec forecasts on quantiles that are based on Jan through Jul observed runoff
Mar_Jul_Runoff_List = [ currentVariable.varGet('FTP_K_Accum_Stor'), currentVariable.varGet('GARR_Accum_Stor'),
                        currentVariable.varGet('OAHE_Accum_Stor'), currentVariable.varGet('BEND_and_FTRA_Accum_Stor'),
                        currentVariable.varGet('GAPT_Accum_Stor')]
Mon_Runoff_List = [ currentVariable.varGet('FTP_K_Mon_Accum_Stor'), currentVariable.varGet('GARR_Mon_Accum_Stor'),
                    currentVariable.varGet('OAHE_Mon_Accum_Stor'), currentVariable.varGet('BEND_and_FTRA_Mon_Accum_Stor'),
                    currentVariable.varGet('GAPT_Mon_Accum_Stor')]

if mon > 7 :
    for t in range(len(projectIds)) :
        totalProjYearMon = 0
        quantileVals1 = historicQuantilesTotal(projectIds[t], 3, 7) # Returns tuple of statistics for the project Mar through Jul
        quantileVals2 = historicQuantilesTotal(projectIds[t], mon, 12) # Returns tuple of statistics for the project current month through December
        # Need to create a fraction of the runoff when computing on days > 1
        for i in range(len(quantileVals1))[:-1] : # Selects which quantile to use for the system based on the Jan through Jul observed runoff
                                                    # by rounding down to the nearest quantile
            if Mar_Jul_Runoff_List[t] > quantileVals1[i] : break
        totalProjYearMon += (quantileVals2[i] - Mon_Runoff_List[t])
        #
        # Add the "forecast" runoff for the project to the total
        #
        totalFcstRunoff += totalProjYearMon
else :
    monthname = months[0][mon-1]
    for projectId in projectIds :
        if verbose : output("-- Project %s" % projectId)
        #
        # sum the forecast monthly runoff from the forecast month to Jul
        #
        if day == 1 : # If day = 1, take the whole DRM monthly forecast
            if verbose : output("---- Using 100%% of %s forecast runoff" % monthname)
            totalProjYearMon = sum(fcstFlows[year][projectId][monthname])
        else :
            fraction = 1. - float(day) / maxDay(mon, year)
            if verbose : output("---- Using %.1f%% of %s forecast runoff" % (fraction * 100, monthname))
            fcstVals = fcstFlows[year][projectId][monthname]
            totalProjYearMon = fcstVals[0] * fraction + sum(fcstVals[1:])
        if verbose : output("---- %s to Jul forecast total = %s KAF" % (monthname, totalProjYearMon))
        #
        # determine which quantile the total forecast runoff belongs to
        #
        quantileVals1 = historicQuantilesTotal(projectId, mon, 7) # Returns tuple of statistics for the project through July
        quantileVals2 = historicQuantilesTotal(projectId, 8, 12) # Returns tuple of statistics for the project August through December
        if (mon, day) == (3, 15) and (not userInfo.has_key("systemQuantiles") or not userInfo["systemQuantiles"].has_key(year)) :
            systemQuantiles = map(sum, zip(systemQuantiles, quantileVals1, quantileVals2)) # Creates the system runoff statistics for each quantile
            userInfo.setdefault("systemQuantiles", {})[year] = systemQuantiles
        if verbose :
            output("---- Quantile historic %s to Jul runoff = %s KAF" % (monthname, quantileVals1))
            output("---- Quantile historic Aug to Dec runoff = %s KAF" % quantileVals2)
        for i in range(len(quantileVals1))[:-1] : # Selects which quantile to use for the system based on the current month through July runoff (DRM)
                                                    # by rounding down to the nearest quantile
            if totalProjYearMon > quantileVals1[i] : break

```

```

if verbose : output("---- Using quantile %s (%s)" % ((".10",".25",".50",".75",".90")[i], ("LD","LQ","M","UQ","UD")[i]))
#
# add the historical Aug to Dec runoff for the same quantile
# to determine the "forecast" runoff for the remainder of
# the calendar year for this project
#
totalProjYearMon += quantileVals2[i] # Based on quantileVals1[i], add the Aug through Dec runoff
if verbose :
    output("---- Aug to Dec forecast total = %.0f KAF" % quantileVals2[i])
    output("---- %s to Dec forecast total = %.0f KAF" % (monthname, totalProjYearMon))
#
# Add the "forecast" runoff for the project to the total
#
totalFcstRunoff += totalProjYearMon
if verbose : output("-- %s to Dec forecast total = %.0f KAF" % (monthname, totalFcstRunoff))

```

```

return totalFcstRunoff

```

```

def getInterp(seq, val) :
    """
    returns interpolation/extrapolation info for a tuple/list. Returns the location in the list and distance from next value
    """
    assert len(seq) > 1
    if val > seq[-1] :
        lo = len(seq) - 2
    else :
        lo = max(0, bisect(seq, val) - 1)
    fraction = getFraction(val, seq[lo], seq[lo+1])
    if verbose :
        if fraction > 1. : output("*** Value %f above high value of %f" % (val, seq[-1]))
        elif fraction < 0. : output("*** Value %f below low value of %f" % (val, seq[0]))
    return lo, fraction

```

```

def julianDay(*args) :
    """
    get the julian day for a month and day or a string, or a month and day for a julian day
    """
    # J F M A M J J A S O N D
    daysPrev = 0, 31, 59, 90,120,151,181,212,243,273,304,334
    argCount = len(args)
    if argCount == 1 :
        if type(args[0]) == type(0) :
            jday = args[0]
            if not 1 <= jday <= 365 :
                raise ValueError("Julian day out of range 1..365")
            for i in range(12)[::-1] :
                if jday > daysPrev[i] :
                    mon = i + 1
                    break
            day = jday - daysPrev[mon-1]
            return mon, day
        else :
            mon, day = args[0].split("_")

```

```

        mon = ["JAN", "FEB", "MAR", "APR", "MAY", "JUN", "JUL", "AUG", "SEP", "OCT", "NOV", "DEC"].index(mon.upper()) + 1
        day = int(day)
        return julianDay(mon, day)
elif argCount == 2 :
    mon, day = args
    if not 1 <= mon <= 12 :
        raise ValueError("Month is out of range 1..12")
    if not 1 <= day <= 31 :
        raise ValueError("Day is out of range 1..31")
    if mon == 2 and day == 29 : day = 28
    return daysPrev[mon-1] + day
else :
    raise ValueError("Expected 1 or 2 arguments, got %d" % argCount)

def assessSystemState(mon, day, year, curStor, fcstRunoff) :
    """
    perform main system assessment
    """
    def getYValue(pdc, lo, fraction, curve) :
        hi = lo + 1
        return pdc.yOrdinates[curve][lo] + fraction * (pdc.yOrdinates[curve][hi] - pdc.yOrdinates[curve][lo])

    #
    # verify assessment date
    #
    userInfo = currentVariable.varGet("userInfo")
    output = userInfo["output"]
    outputDebug = userInfo["outputDebug"]
    verbose = userInfo["verbose"]
    julian = userInfo["julianDay"](mon, day)
    useExtraDates = userInfo["useExtraDates"]
    extraAssessmentDates = userInfo["extraAssessmentDates"]
    PassCounter = network.getComputePassCounter() # Pass number of simulation
    PassNumber = PassCounter + 1 # Pass number starts at 0 so add 1 to start it at 1

    if julian not in assessmentDates + tuple(extraAssessmentDates) :
        raise ValueError("Invalid assessment date: %2.2d/%2.2d" % (mon, day))
    if verbose :
        output(separator)
        output("System assessment for %4.4d/%2.2d/%2.2d" % (year, mon, day))

    #
    # initialize variables
    #
    gaptAvg = None
    serviceLevel = None
    navSeasonEnd = None
    fcstRunoff /= 1000 # convert from KAF to MAF
    curStor /= 1e6 # convert to MAF
    pdc = userInfo["thresholds"]
    lo, fraction = getInterp(pdc.xOrdinates, julian)
    evacuationThreshold = getYValue(pdc, lo, fraction, userInfo["thresholdCurves"]["EVACUATION"])
    waterSupply = curStor + fcstRunoff
    if verbose :

```

```

output("-- current storage = %6.2f MAF" % curStor)
output("-- forecast runoff = %6.2f MAF" % fcstRunoff)
output("-- water supply = %6.2f MAF" % waterSupply)
output("-- evacuation level = %6.2f MAF" % evacuationThreshold)
#-----#
# initialize year-based dictionaries if they don't already exist #
#-----#
for varName in ("systemQuantiles", "Evacuation_Year", "MAR_15_Level", "JUL_01_Level", "LAST_NAV_END_DATE") :
    try : userInfo[varName]
    except : userInfo[varName] = {}
#
# perform the assessment appropriate for the date
#
if julian == FEB_01 :
    if verbose : output("-- No action currently taken on Feb 01")
    pass
elif julian == MAR_01 :
    if verbose : output("-- No action currently taken on Mar 01")
    pass
if MAR_15 <= julian <= NOV_01 :
    #
    # set service level
    #
    if waterSupply > evacuationThreshold :
        output("%4.4d/%2.2d/%2.2d: Water Supply (%s) > Evacuation Threshold (%s)" % (year, mon, day, waterSupply, evacuationThreshold))
        userInfo["Evacuation_Year"][year] = True
        pdc = userInfo["serviceLevels"]
        levelValues = userInfo["levelValues"]
        lo, fraction = getInterp(pdc.xOrdinates, julian)
        serviceLevels = [getYValue(pdc, lo, fraction, i) for i in range(len(pdc.yOrdinates))]
        lo, fraction = getInterp(serviceLevels, waterSupply)
        hi = lo + 1
        serviceLevel = levelValues[lo] + fraction * (levelValues[hi] - levelValues[lo])
        if julian == MAR_15 :
            userInfo["MAR_15_Level"][year] = serviceLevel
            output("%4.4d/%2.2d/%2.2d: Setting Evacuation_Year to TRUE (MAR_15_Level = %s)" % (year, mon, day, serviceLevel))
        elif julian == MAY_01 and useExtraDates and MAY_01+7 not in extraAssessmentDates :
            extraAssessmentDates.extend([j for j in range(MAY_01+7, JUN_01, 7)])
            output("%4.4d/%2.2d/%2.2d: Adding extra assessment dates MAY 8, MAY 15, MAY 22 and MAY 29" % (year, mon, day))
        elif julian == JUN_01 and useExtraDates and JUN_01+7 not in extraAssessmentDates :
            extraAssessmentDates.extend([j for j in range(JUN_01+7, JUL_01, 7)])
            output("%4.4d/%2.2d/%2.2d: Adding extra assessment dates JUN 8, JUN 15, JUN 22 and JUN 29" % (year, mon, day))
        elif julian == JUL_01 and useExtraDates and JUL_01+7 not in extraAssessmentDates :
            extraAssessmentDates.extend([j for j in range(JUL_01+7, AUG_01, 7)])
            output("%4.4d/%2.2d/%2.2d: Adding extra assessment dates JUL 8, JUL 15, JUL 22 and JUL 29" % (year, mon, day))
        elif julian == AUG_01 and useExtraDates and AUG_01+7 not in extraAssessmentDates :
            extraAssessmentDates.extend([j for j in range(AUG_01+7, SEP_01, 7)])
            output("%4.4d/%2.2d/%2.2d: Adding extra assessment dates AUG 8, AUG 15, AUG 22 and AUG 29" % (year, mon, day))
        elif julian == SEP_01 and useExtraDates and SEP_01+7 not in extraAssessmentDates :
            extraAssessmentDates.extend([j for j in range(SEP_01+7, OCT_01, 7)])
            output("%4.4d/%2.2d/%2.2d: Adding extra assessment dates SEP 8, SEP 15, SEP 22 and SEP 29" % (year, mon, day))
        elif julian == OCT_01 and useExtraDates and OCT_01+7 not in extraAssessmentDates :
            extraAssessmentDates.extend([j for j in range(OCT_01+7, NOV_01, 7)])

```

```

        output("%4.4d/%2.2d/%2.2d: Adding extra assessment dates OCT 8, OCT 15, OCT 22 and OCT 29" % (year, mon, day))
else :
    if julian == MAR_15 :
        extraAssessmentDates = []
        userInfo["Evacuation_Year"][year] = False
        output("%4.4d/%2.2d/%2.2d: Setting Evacuation_Year to FALSE" % (year, mon, day))
        if curStor >= MAR_15_FULL_SERVICE_LOWER_THRESHOLD :
            serviceLevel = FULL_SERVICE
            output("%4.4d/%2.2d/%2.2d: Current Storage (%s) >= MAR_15_FULL_SERVICE_LOWER_THRESHOLD (%s) : Setting Service Level to %s" % (year, mon, day, curStor,
MAR_15_FULL_SERVICE_LOWER_THRESHOLD, serviceLevel))
        elif curStor >= MAR_15_MIN_SERVICE_UPPER_THRESHOLD :
            serviceLevel = MINIMUM_SERVICE + getFraction(
                curStor,
                MAR_15_MIN_SERVICE_UPPER_THRESHOLD,
                MAR_15_FULL_SERVICE_LOWER_THRESHOLD) * (FULL_SERVICE - MINIMUM_SERVICE)
            output("%4.4d/%2.2d/%2.2d: Current Storage (%s) >= MAR_15_MIN_SERVICE_UPPER_THRESHOLD (%s) : Setting Service Level to %s" % (year, mon, day, curStor,
MAR_15_MIN_SERVICE_UPPER_THRESHOLD, serviceLevel))
        elif curStor >= MAR_15_MIN_SERVICE_LOWER_THRESHOLD :
            serviceLevel = MINIMUM_SERVICE
            output("%4.4d/%2.2d/%2.2d: Current Storage (%s) >= MAR_15_MIN_SERVICE_LOWER_THRESHOLD (%s) : Setting Service Level to %s" % (year, mon, day, curStor,
MAR_15_MIN_SERVICE_LOWER_THRESHOLD, serviceLevel))
        else :
            serviceLevel = NO_SERVICE
            output("%4.4d/%2.2d/%2.2d: Current Storage (%s) < MAR_15_MIN_SERVICE_LOWER_THRESHOLD (%s) : Setting Service Level to %s" % (year, mon, day, curStor,
MAR_15_MIN_SERVICE_LOWER_THRESHOLD, serviceLevel))
            userInfo["MAR_15_Level"][year] = serviceLevel
    elif julian in (MAY_01, JUN_01) + tuple(extraAssessmentDates) :
        # Due to the iterations, sometimes this dictionary does not get populated
        try: userInfo["JUL_01_Level"][year]
        except: userInfo["JUL_01_Level"][year] = FULL_SERVICE

    if julian < JUL_01:
        if userInfo["Evacuation_Year"][year] is True:
            serviceLevel = FULL_SERVICE
        else:
            serviceLevel = userInfo["MAR_15_Level"][year]
    elif julian > JUL_01:
        if userInfo["Evacuation_Year"][year] is True and userInfo["JUL_01_Level"][year] >= FULL_SERVICE:
            serviceLevel = FULL_SERVICE
        else:
            serviceLevel = userInfo["JUL_01_Level"][year]
    elif julian == JUL_01 :
        if userInfo["MAR_15_Level"][year] == NO_SERVICE:
            serviceLevel = NO_SERVICE
            output("%4.4d/%2.2d/%2.2d: MAR 15 Level = %s : Setting Service Level to %s" % (year, mon, day, userInfo["MAR_15_Level"][year], serviceLevel))
        elif curStor >= JUL_01_FULL_SERVICE_LOWER_THRESHOLD :
            serviceLevel = FULL_SERVICE
            output("%4.4d/%2.2d/%2.2d: Current Storage (%s) >= JUL_01_FULL_SERVICE_LOWER_THRESHOLD (%s) : Setting Service Level to %s" % (year, mon, day, curStor,
JUL_01_FULL_SERVICE_LOWER_THRESHOLD, serviceLevel))
        elif curStor >= JUL_01_MIN_SERVICE_UPPER_THRESHOLD :
            serviceLevel = MINIMUM_SERVICE + getFraction(
                curStor,
                JUL_01_MIN_SERVICE_UPPER_THRESHOLD,

```

```

        JUL_01_FULL_SERVICE_LOWER_THRESHOLD) * (FULL_SERVICE - MINIMUM_SERVICE)
        output("%4.4d/%2.2d/%2.2d: Current Storage (%s) >= JUL_01_MIN_SERVICE_UPPER_THRESHOLD (%s) : Setting Service Level to %s" % (year, mon, day, curStor,
JUL_01_MIN_SERVICE_UPPER_THRESHOLD, serviceLevel))
        elif curStor >= JUL_01_MIN_SERVICE_LOWER_THRESHOLD :
            serviceLevel = MINIMUM_SERVICE
            output("%4.4d/%2.2d/%2.2d: Current Storage (%s) >= JUL_01_MIN_SERVICE_LOWER_THRESHOLD (%s) : Setting Service Level to %s" % (year, mon, day, curStor,
JUL_01_MIN_SERVICE_LOWER_THRESHOLD, serviceLevel))
        else :
            serviceLevel = NO_SERVICE
            output("%4.4d/%2.2d/%2.2d: Current Storage (%s) < JUL_01_MIN_SERVICE_LOWER_THRESHOLD (%s) : Setting Service Level to %s" % (year, mon, day, curStor,
JUL_01_MIN_SERVICE_LOWER_THRESHOLD, serviceLevel))
            userInfo["JUL_01_Level"][year] = serviceLevel
    if verbose : output("-- service level    = %s KCFS" % serviceLevel)
if julian in (JUL_01, SEP_01) :
    extraAssessmentDates = []
    #
    # compute the navigation season end
    #
    if julian == JUL_01:
        if userInfo["MAR_15_Level"][year] == NO_SERVICE:
            navSeasonEnd = NO_SERVICE
        elif curStor < EARLY_NAV_END_DATE_1_UPPER_THRESHOLD:
            navSeasonEnd = EARLY_NAV_END_DATE_1
        elif curStor < EARLY_NAV_END_DATE_2_LOWER_THRESHOLD:
            navSeasonEnd = EARLY_NAV_END_DATE_1 + int(getFraction(
                curStor,
                EARLY_NAV_END_DATE_1_UPPER_THRESHOLD,
                EARLY_NAV_END_DATE_2_LOWER_THRESHOLD) * (EARLY_NAV_END_DATE_2 - EARLY_NAV_END_DATE_1))
        elif curStor < EARLY_NAV_END_DATE_2_UPPER_THRESHOLD:
            navSeasonEnd = EARLY_NAV_END_DATE_2
        elif curStor < NORMAL_NAV_END_DATE_LOWER_THRESHOLD:
            navSeasonEnd = EARLY_NAV_END_DATE_2 + int(getFraction(
                curStor,
                EARLY_NAV_END_DATE_2_UPPER_THRESHOLD,
                NORMAL_NAV_END_DATE_LOWER_THRESHOLD) * (NORMAL_NAV_END_DATE - EARLY_NAV_END_DATE_2))
        else:
            navSeasonEnd = NORMAL_NAV_END_DATE
    if julian == SEP_01:
        if curStor >= EXTENDED_NAV_END_DATE_LOWER_THRESHOLD:
            navSeasonEnd = EXTENDED_NAV_END_DATE
        else:
            navSeasonEnd = userInfo["LAST_NAV_END_DATE"][year]
            if navSeasonEnd is None: navSeasonEnd = NORMAL_NAV_END_DATE
    userInfo["LAST_NAV_END_DATE"][year] = navSeasonEnd

    if verbose :
        mon, day = userInfo["julianDay"](navSeasonEnd)
        output("-- Navigation season ends %2.2d/%2.2d" % (mon, day))
    if julian == SEP_01 :
        #
        # winter release planning
        #

```

```

    if curStor >= HIGH_WINTER_RELEASE_LOWER_THRESHOLD :
        gaptAvg = HIGH_WINTER_RELEASE
    elif curStor >= MID_WINTER_RELEASE_LOWER_THRESHOLD :
        gaptAvg = MID_WINTER_RELEASE
    elif curStor > LOW_WINTER_RELEASE_UPPER_THRESHOLD :
        gaptAvg = LOW_WINTER_RELEASE + (
            curStor - LOW_WINTER_RELEASE_UPPER_THRESHOLD) /\
            (MID_WINTER_RELEASE_LOWER_THRESHOLD - LOW_WINTER_RELEASE_UPPER_THRESHOLD) * \
            (MID_WINTER_RELEASE - LOW_WINTER_RELEASE)
    else :
        gaptAvg = LOW_WINTER_RELEASE
    if verbose :
        output(separator)
        output("Setting GAPT average winter release to %s" % gaptAvg)
return serviceLevel, navSeasonEnd, gaptAvg

```

```
def plotStateVariables() :
```

```
    scriptName = "StateVariablePlot"
```

```
    pathnames = {
```

```

        "ServiceLevel"      : "//SERVICELEVEL/FLOW-SERVICE//1DAY/%s/",
        "GAPTSteadyRelease" : "//GAPTSTEADYRELEASE/FLOW-STEADY//1DAY/%s/",
        "GAPTCycledRelease" : "//GAPTCYCLEDRELEASE/FLOW-CYCLED//1DAY/%s/",
        "GAPTWinterRelease" : "//GAPTWINTERRELEASE/FLOW-WINTER-GAPT//1DAY/%s/",
        "NavigationEndDate" : "//NAVIGATIONENDDATE/COUNT-NAV_END_JULIAN//1DAY/%s/",
        "MainstemStorage"   : "//MAINSTEMSTORAGE/STOR-MAINSTEM//1DAY/%s/",
        "FcstRunoff"        : "//FCSTRUNOFF/STOR-FCST//1DAY/%s/",
        "WaterSupply"       : "//WATERSUPPLY/STOR-WATSUP//1DAY/%s/"
    }

```

```
}
```

```

fPart = ResSimUtils.getFPart(ResSimUtils.getSelectedAlternativeNames()[0])
for stateVariable in pathnames.keys() :
    pathnames[stateVariable] = pathnames[stateVariable] % fPart
dssFilename = ResSimUtils.getSimulationDSSFileName()
simulation = ResSimUtils.getSimulation()
dssFile = HecDss.open(dssFilename)
dssFile.setTimeWindow(simulation.getStartDateString(), simulation.getEndDateString())

```

```

serviceLevelData = dssFile.read(pathnames["ServiceLevel"]).multiply(1000).getData()
steadyReleaseData = dssFile.get(pathnames["GAPTSteadyRelease"])
cycledReleaseData = dssFile.get(pathnames["GAPTCycledRelease"])
winterReleaseData = dssFile.get(pathnames["GAPTWinterRelease"])
navigationData = dssFile.get(pathnames["NavigationEndDate"])
storageData = dssFile.get(pathnames["MainstemStorage"])
fcstData = dssFile.get(pathnames["FcstRunoff"])
waterSupplyData = dssFile.get(pathnames["WaterSupply"])

```

```
dssFile.close()
```

```

for dataset in serviceLevelData, steadyReleaseData, cycledReleaseData, winterReleaseData, navigationData, storageData, fcstData, waterSupplyData :
    dataset.type = "PER-AVER"

```

```

plot = Plot.newPlot("State Variables")
plot.setSize(600, 800)
plotLayout = Plot.newPlotLayout()
vp1 = plotLayout.addViewPort(2)
vp1.addCurve("Y1", serviceLevelData)
vp1.addCurve("Y1", steadyReleaseData)
vp1.addCurve("Y1", cycledReleaseData)
vp1.addCurve("Y1", winterReleaseData)
vp2 = plotLayout.addViewPort(1)
vp2.addCurve("Y1", navigationData)
vp3 = plotLayout.addViewPort(2)
vp3.addCurve("Y1", waterSupplyData)
vp3.addCurve("Y1", storageData)
vp3.addCurve("Y1", fcstData)
plot.configurePlotLayout(plotLayout)
plot.showPlot()
plot.applyTemplate(os.path.join(sharedDir, "StateVariables.template"))

```

```

EARLY_NAV_END_DATE_1 = julianDay(EARLY_NAV_END_DATE_1)
EARLY_NAV_END_DATE_2 = julianDay(EARLY_NAV_END_DATE_2)
NORMAL_NAV_END_DATE = julianDay(NORMAL_NAV_END_DATE)
EXTENDED_NAV_END_DATE = julianDay(EXTENDED_NAV_END_DATE)
LOW_WINTER_RELEASE      *= 1000.
MID_WINTER_RELEASE      *= 1000.
HIGH_WINTER_RELEASE     *= 1000.
BELOW_MEDIAN_FULL_SVC_STEADY_RELEASE *= 1000.
BELOW_MEDIAN_MIN_SVC_STEADY_RELEASE *= 1000.
MEDIAN_OR_ABOVE_FULL_SVC_STEADY_RELEASE *= 1000.
MEDIAN_OR_ABOVE_MIN_SVC_STEADY_RELEASE *= 1000.
CYCLED_RELEASE_CUTBACK      *= 1000.
#
# store info into simulation for quick reference
#
userInfo = {
    "assessmentDates"          : assessmentDates,
    "assessSystemState"       : assessSystemState,
    "BELOW_MEDIAN_FULL_SVC_STEADY_RELEASE" : BELOW_MEDIAN_FULL_SVC_STEADY_RELEASE,
    "BELOW_MEDIAN_MIN_SVC_STEADY_RELEASE"  : BELOW_MEDIAN_MIN_SVC_STEADY_RELEASE,
    'BEND_Local'              : BEND_Local,
    'BIS_Local'               : BIS_Local,
    'coeffDict'               : coeffDict,
    'CLMT_Local'              : CLMT_Local,
    "constants"               : Constants,
    "CRFTT"                   : CRFTT,
    "CYCLED_RELEASE_CUTBACK"   : CYCLED_RELEASE_CUTBACK,
    "CYCLED_RELEASE_CUTBACK_PERIOD" : CYCLED_RELEASE_CUTBACK_PERIOD,
    "CYCLED_RELEASE_STEADY_PERIOD" : CYCLED_RELEASE_STEADY_PERIOD,
    'debug'                   : debug,
    "EARLY_NAV_END_DATE_1"     : EARLY_NAV_END_DATE_1,
    "EARLY_NAV_END_DATE_1_UPPER_THRESHOLD" : EARLY_NAV_END_DATE_1_UPPER_THRESHOLD,
    "EARLY_NAV_END_DATE_2"     : EARLY_NAV_END_DATE_2,
    "EARLY_NAV_END_DATE_2_LOWER_THRESHOLD" : EARLY_NAV_END_DATE_2_LOWER_THRESHOLD,
    "EARLY_NAV_END_DATE_2_UPPER_THRESHOLD" : EARLY_NAV_END_DATE_2_UPPER_THRESHOLD,

```

"EXTENDED\_NAV\_END\_DATE" : EXTENDED\_NAV\_END\_DATE,  
"EXTENDED\_NAV\_END\_DATE\_LOWER\_THRESHOLD" : EXTENDED\_NAV\_END\_DATE\_LOWER\_THRESHOLD,  
"extraAssessmentDates" : extraAssessmentDates,  
"fcstFlows" : fcstFlows,  
"FEB\_01" : FEB\_01,  
'FTPK\_Local' : FTPK\_Local,  
'FTRA\_Local' : FTRA\_Local,  
"FTT" : FTT,  
"FULL\_SERVICE" : FULL\_SERVICE,  
'GAPT\_Local' : GAPT\_Local,  
'GARR\_Local' : GARR\_Local,  
"getForecast" : getForecast,  
"HecTime" : HecTime,  
"HIGH\_WINTER\_RELEASE" : HIGH\_WINTER\_RELEASE,  
"HIGH\_WINTER\_RELEASE\_LOWER\_THRESHOLD" : HIGH\_WINTER\_RELEASE\_LOWER\_THRESHOLD,  
"historicRunoff" : historicRunoff,  
"JUL\_01" : JUL\_01,  
"JUL\_01\_FULL\_SERVICE\_LOWER\_THRESHOLD" : JUL\_01\_FULL\_SERVICE\_LOWER\_THRESHOLD,  
"JUL\_01\_MIN\_SERVICE\_LOWER\_THRESHOLD" : JUL\_01\_MIN\_SERVICE\_LOWER\_THRESHOLD,  
"JUL\_01\_MIN\_SERVICE\_UPPER\_THRESHOLD" : JUL\_01\_MIN\_SERVICE\_UPPER\_THRESHOLD,  
"JUL\_15" : JUL\_15,  
"julianDay" : julianDay,  
"JUN\_01" : JUN\_01,  
"LAST\_NAV\_END\_DATE" : {},  
"levelValues" : levelValues,  
"LOW\_WINTER\_RELEASE" : LOW\_WINTER\_RELEASE,  
"LOW\_WINTER\_RELEASE\_UPPER\_THRESHOLD" : LOW\_WINTER\_RELEASE\_UPPER\_THRESHOLD,  
"MAR\_01" : MAR\_01,  
"MAR\_15" : MAR\_15,  
"MAR\_19" : MAR\_19,  
"MAR\_15\_FULL\_SERVICE\_LOWER\_THRESHOLD" : MAR\_15\_FULL\_SERVICE\_LOWER\_THRESHOLD,  
"MAR\_15\_Level" : {},  
"MAR\_15\_MIN\_SERVICE\_LOWER\_THRESHOLD" : MAR\_15\_MIN\_SERVICE\_LOWER\_THRESHOLD,  
"MAR\_15\_MIN\_SERVICE\_UPPER\_THRESHOLD" : MAR\_15\_MIN\_SERVICE\_UPPER\_THRESHOLD,  
"MAY\_01" : MAY\_01,  
"MAY\_15" : MAY\_15,  
"MEDIAN\_OR\_ABOVE\_FULL\_SVC\_STEADY\_RELEASE" : MEDIAN\_OR\_ABOVE\_FULL\_SVC\_STEADY\_RELEASE,  
"MEDIAN\_OR\_ABOVE\_MIN\_SVC\_STEADY\_RELEASE" : MEDIAN\_OR\_ABOVE\_MIN\_SVC\_STEADY\_RELEASE,  
"MID\_WINTER\_RELEASE" : MID\_WINTER\_RELEASE,  
"MID\_WINTER\_RELEASE\_LOWER\_THRESHOLD" : MID\_WINTER\_RELEASE\_LOWER\_THRESHOLD,  
"MID\_WINTER\_RELEASE\_UPPER\_THRESHOLD" : MID\_WINTER\_RELEASE\_UPPER\_THRESHOLD,  
"MINIMUM\_SERVICE" : MINIMUM\_SERVICE,  
"months" : months,  
"NO\_SERVICE" : NO\_SERVICE,  
'NOV\_01' : NOV\_01,  
"NORMAL\_NAV\_END\_DATE" : NORMAL\_NAV\_END\_DATE,  
"NORMAL\_NAV\_END\_DATE\_LOWER\_THRESHOLD" : NORMAL\_NAV\_END\_DATE\_LOWER\_THRESHOLD,  
'OAHE\_Local' : OAHE\_Local,  
'OCT\_01' : OCT\_01,  
"output" : output,  
"outputByDate" : outputByDate,  
"outputDate" : outputDate,  
'outputDebug' : outputDebug,

```

"plotStateVariables"      : plotStateVariables,
"projectIds"              : projectIds,
"projectNames"           : projectNames,
'RBMT_Local'              : RBMT_Local,
"ResSim"                  : ResSim,
"RunTimeStep"             : rts,
"RunTimeWindow"          : rtw,
"scenario"                : scenario,
"SEP_01"                  : SEP_01,
"SERVICE_LEVEL_END_OFFSET" : SERVICE_LEVEL_END_OFFSET,
"SERVICE_LEVEL_START_DATE" : SERVICE_LEVEL_START_DATE,
"serviceLevels"          : serviceLevelPdc,
"setOutputDate"          : setOutputDate,
"SRFTT"                  : SRFTT,
"STEADY_STATE_COMPUTATION_THRESHOLD" : STEADY_STATE_COMPUTATION_THRESHOLD,
"thresholdCurves"       : thresholdCurves,
"thresholds"             : thresholdsPdc,
"useExtraDates"          : useExtraDates,
"verbose"                : verbose,
'WPMT_Local'             : WPMT_Local}
currentVariable.varPut("userInfo", userInfo)
return True

```

```

#####
##### STATE VARIABLE SCRIPT COMPUTATION SECTION
#####

```

```

'''
Description: Version 2015-05-07 of the service level state variable script. It assesses the system using the procedure in the Master Manual and sets the
service level for navigation, the season length of the navigation season, and the winter releases for GAPT.
'''

```

```

# no return values are used by the compute from this script.
#
# variables that are available to this script during the compute:
#   currentVariable - the StateVariable that holds this script
#   currentRuntimestep - the current RunTime step
#   network - the ResSim network

```

```

# The following represents an undefined value in a time series
#   Constants.UNDEFINED

```

```

#
# access the minimum info needed to continue
#
userInfo = currentVariable.varGet("userInfo")
assessmentDates = userInfo["assessmentDates"]
BELOW_MEDIAN_FULL_SVC_STEADY_RELEASE = userInfo["BELOW_MEDIAN_FULL_SVC_STEADY_RELEASE"]
BELOW_MEDIAN_MIN_SVC_STEADY_RELEASE = userInfo["BELOW_MEDIAN_MIN_SVC_STEADY_RELEASE"]
BEND_Local = userInfo['BEND_Local']
BIS_Local = userInfo['BIS_Local']
CLMT_Local = userInfo['CLMT_Local']
coeffDict = userInfo['coeffDict']
CRFTT = userInfo["CRFTT"]

```

```

CYCLED_RELEASE_CUTBACK          = userInfo["CYCLED_RELEASE_CUTBACK"]
CYCLED_RELEASE_CUTBACK_PERIOD    = userInfo["CYCLED_RELEASE_CUTBACK_PERIOD"]
CYCLED_RELEASE_STEADY_PERIOD     = userInfo["CYCLED_RELEASE_STEADY_PERIOD"]
debug                            = userInfo['debug']
extraAssessmentDates            = userInfo["extraAssessmentDates"]
FTPK_Local                      = userInfo['FTPK_Local']
FTRA_Local                      = userInfo['FTRA_Local']
FTT                              = userInfo["FTT"]
FULL_SERVICE                    = userInfo["FULL_SERVICE"]
GAPT_Local                      = userInfo['GAPT_Local']
GARR_Local                      = userInfo['GARR_Local']
HecTime                        = userInfo["HecTime"]
JUL_01                          = userInfo["JUL_01"]
JUL_15                          = userInfo["JUL_15"]
julianDay                       = userInfo["julianDay"]
JUN_01                          = userInfo["JUN_01"]
MAR_01                          = userInfo["MAR_01"]
MAR_15                          = userInfo["MAR_15"]
MAR_19                          = userInfo["MAR_19"]
MAR_15_Level                    = userInfo['MAR_15_Level']
MAY_15                          = userInfo["MAY_15"]
MEDIAN_OR_ABOVE_FULL_SVC_STEADY_RELEASE = userInfo["MEDIAN_OR_ABOVE_FULL_SVC_STEADY_RELEASE"]
MEDIAN_OR_ABOVE_MIN_SVC_STEADY_RELEASE = userInfo["MEDIAN_OR_ABOVE_MIN_SVC_STEADY_RELEASE"]
MINIMUM_SERVICE                 = userInfo["MINIMUM_SERVICE"]
NO_SERVICE                      = userInfo["NO_SERVICE"]
NOV_01                          = userInfo["NOV_01"]
NORMAL_NAV_END_DATE             = userInfo["NORMAL_NAV_END_DATE"]
OAHE_Local                      = userInfo['OAHE_Local']
output                          = userInfo['output']
outputDebug                     = userInfo['outputDebug']
projectNames                    = userInfo["projectNames"]
RBMT_Local                      = userInfo['RBMT_Local']
scenario                        = userInfo["scenario"]
SERVICE_LEVEL_END_OFFSET       = userInfo["SERVICE_LEVEL_END_OFFSET"]
SERVICE_LEVEL_START_DATE       = userInfo["SERVICE_LEVEL_START_DATE"]
SEP_01                          = userInfo["SEP_01"]
SRFTT                           = userInfo["SRFTT"]
STEADY_STATE_COMPUTATION_THRESHOLD = userInfo["STEADY_STATE_COMPUTATION_THRESHOLD"]
WPMT_Local                      = userInfo['WPMT_Local']

```

```

t = currentRuntimestep.getHecTime()
year = t.year()
mon = t.month()
day = t.day()
jDay = julianDay(mon, day)
rtw = currentRuntimestep.getRunTimeWindow() # Run time window
currentVariable.varPut('DateTimeString', currentRuntimestep.dateTimeString() + currentRuntimestep.getHecTime().hourMinutes())
curStep = currentRuntimestep.getStep() # Current step
numLookbackSteps = rtw.getNumLookbackSteps() # Number of lookback steps in simulation
simStartStep = numLookbackSteps + 1 # First step after the lookback period
PassCounter = network.getComputePassCounter() # Pass number of simulation
PassNumber = PassCounter + 1 # Pass number starts at 0 so add 1 to start it at 1
# Conversions

```

```
CFS_to_ACRE_FT = 86400.0/43560.0
ACRE_FT_to_CFS = 43560.0/86400.0
```

```
if PassNumber < 3:
```

```
    pass
```

```
else:
```

```
    #
```

```
    # compute the system storage (in AF)
```

```
    #
```

```
    stor = 0.
```

```
    storTs = network.getTimeSeries("Reservoir","System Reservoir", "Pool", "Stor")
```

```
    if storTs :
```

```
        stor = storTs.getPreviousValue(currentRuntimestep)
```

```
    else :
```

```
        projects = ["Fort Peck Lake", "Lake Sakakawea", "Lake Oahe", "Lake Sharpe", "Lake Francis Case", "Lewis and Clark Lake"]
```

```
        for projectName in projects :
```

```
            storTs = network.getTimeSeries("Reservoir", projectName, "Pool", "Stor")
```

```
            if storTs :
```

```
                stor += storTs.getPreviousValue(currentRuntimestep)
```

```
            else :
```

```
                output("WARNING : " + "Cannot retrieve storage time series for %s" % projectName)
```

```
    # -----
```

```
    # Jan through Jul and Monthly Accumulated Incremental Runoff
```

```
    # -----
```

```
if 3 <= mon <= 7:
```

```
    if curStep == simStartStep or (mon == 3 and day == 1):
```

```
        currentVariable.varPut('Mar_01_Step', curStep)
```

```
    Mar_01_Step = currentVariable.varGet('Mar_01_Step')
```

```
    beginRange = Mar_01_Step
```

```
    endRange = curStep
```

```
    FTPK_Accum_Stor = (sum(RBMT_Local[beginRange:endRange]) + sum(FTPK_Local[beginRange:endRange])) * CFS_to_ACRE_FT / 1000
```

```
    GARR_Accum_Stor = (sum(WPMT_Local[beginRange:endRange]) + sum(CLMT_Local[beginRange:endRange]) + sum(GARR_Local[beginRange:endRange])) * CFS_to_ACRE_FT / 1000
```

```
    OAHE_Accum_Stor = (sum(BIS_Local[beginRange:endRange]) + sum(OAHE_Local[beginRange:endRange])) * CFS_to_ACRE_FT / 1000
```

```
    BEND_and_FTFR_Accum_Stor = (sum(BEND_Local[beginRange:endRange]) + sum(FTRA_Local[beginRange:endRange])) * CFS_to_ACRE_FT / 1000
```

```
    GAPT_Accum_Stor = sum(GAPT_Local[beginRange:endRange]) * CFS_to_ACRE_FT / 1000
```

```
    currentVariable.varPut('FTPK_Accum_Stor', FTPK_Accum_Stor)
```

```
    currentVariable.varPut('GARR_Accum_Stor', GARR_Accum_Stor)
```

```
    currentVariable.varPut('OAHE_Accum_Stor', OAHE_Accum_Stor)
```

```
    currentVariable.varPut('BEND_and_FTFR_Accum_Stor', BEND_and_FTFR_Accum_Stor)
```

```
    currentVariable.varPut('GAPT_Accum_Stor', GAPT_Accum_Stor)
```

```
elif mon > 7:
```

```
    beginRange = curStep - (day - 1)
```

```
    endRange = curStep
```

```
    FTPK_Mon_Accum_Stor = (sum(RBMT_Local[beginRange:endRange]) + sum(FTPK_Local[beginRange:endRange])) * CFS_to_ACRE_FT / 1000
```

```
    GARR_Mon_Accum_Stor = (sum(WPMT_Local[beginRange:endRange]) + sum(CLMT_Local[beginRange:endRange]) + sum(GARR_Local[beginRange:endRange])) * CFS_to_ACRE_FT / 1000
```

```
    OAHE_Mon_Accum_Stor = (sum(BIS_Local[beginRange:endRange]) + sum(OAHE_Local[beginRange:endRange])) * CFS_to_ACRE_FT / 1000
```

```
    BEND_and_FTFR_Mon_Accum_Stor = (sum(BEND_Local[beginRange:endRange]) + sum(FTRA_Local[beginRange:endRange])) * CFS_to_ACRE_FT / 1000
```

```
    GAPT_Mon_Accum_Stor = sum(GAPT_Local[beginRange:endRange]) * CFS_to_ACRE_FT / 1000
```

```
    currentVariable.varPut('FTPK_Mon_Accum_Stor', FTPK_Mon_Accum_Stor)
```

```

currentVariable.varPut('GARR_Mon_Accum_Stor', GARR_Mon_Accum_Stor)
currentVariable.varPut('OAHE_Mon_Accum_Stor', OAHE_Mon_Accum_Stor)
currentVariable.varPut('BEND_and_FTRA_Mon_Accum_Stor', BEND_and_FTRA_Mon_Accum_Stor)
currentVariable.varPut('GAPT_Mon_Accum_Stor', GAPT_Mon_Accum_Stor)

network.getStateVariable("MainstemStorage").setValue(currentRuntimestep, stor)
#
# Reset extraAssessmentDates in the info dictionary
#
if day == 15 and mon == 3:
    userInfo['extraAssessmentDates'] = []
    extraAssessmentDates = userInfo['extraAssessmentDates']
#
# Add OCT_01 and NOV_01 as extra assessment dates if the navigation season length is exceeds them
#
if day == 15 and mon == 9:
    if userInfo['OCT_01'] <= userInfo["LAST_NAV_END_DATE"][year] and userInfo['OCT_01'] not in userInfo['extraAssessmentDates']:
        userInfo['extraAssessmentDates'].append(userInfo['OCT_01'])
    if userInfo['NOV_01'] <= userInfo["LAST_NAV_END_DATE"][year] and userInfo['NOV_01'] not in userInfo['extraAssessmentDates']:
        userInfo['extraAssessmentDates'].append(userInfo['NOV_01'])

allAssessmentDates = assessmentDates + tuple(extraAssessmentDates)
#
# Set WaterSupply and FcstRunoff values from previous passes to Constants.UNDEFINED if they are no longer in assessment dates
#
WaterSupplyForecast = network.getStateVariable('WaterSupply').getValue(currentRuntimestep)
if jDay not in allAssessmentDates and WaterSupplyForecast > 0:
    network.getStateVariable("WaterSupply").setValue(currentRuntimestep, Constants.UNDEFINED)
    network.getStateVariable("FcstRunoff").setValue(currentRuntimestep, Constants.UNDEFINED)

if jDay in (MAR_01,) + allAssessmentDates :
    userInfo["setOutputDate"](year, mon, day)
    #
    # access info needed by all branches
    #
    Constants = userInfo["constants"]
    output = userInfo["output"]
    rtw = userInfo["RunTimeWindow"]
    verbose = userInfo["verbose"]
    if jDay == (MAR_01) :
        #
        # un-set the Gavins Point average winter release
        #
        network.getStateVariable("GAPTWinterRelease").setValueRange(
            currentRuntimestep,
            t,
            rtw.getEndTime(),
            Constants.UNDEFINED)
    else : # jDay in allAssessmentDates getPreviousValue(hec.model.RunTimeStep)
        #
        # assess the system based on current date, current storage, and projected runoff for remainder of year
        #
        assessSystemState = userInfo["assessSystemState"]

```

```

getForecast = userInfo["getForecast"]
#
# get the forecast accumulated runoff for the remainder of the calendar year (in KAF)
#
fcst = getForecast(mon, year, day)
network.getStateVariable("FcstRunoff").setValue(currentRuntimestep, fcst * 1000)
network.getStateVariable("WaterSupply").setValue(currentRuntimestep, stor + fcst * 1000)
#
# call the multi-purpose function
#
serviceLevel, navSeasonEnd, gaptAvg = assessSystemState(mon, day, year, stor, fcst)
# Because of the extra assessment dates after SEP_01, need to set navSeasonEnd to the date stored when it ran on SEP_01
if SEP_01 < jDay <= userInfo["LAST_NAV_END_DATE"][year]:
    navSeasonEnd = userInfo["LAST_NAV_END_DATE"][year]

if serviceLevel is not None :
    #
    # set the service level for the remainder of the simulation period
    #
    jDayServiceLevelStart = julianDay(SERVICE_LEVEL_START_DATE)
    if jDay < jDayServiceLevelStart :
        serviceLevelStart = HecTime(t)
        startMon, startDay = julianDay(jDayServiceLevelStart)
        serviceLevelStart.setYearMonthDay(year, startMon, startDay)
    else :
        serviceLevelStart = t
    if verbose : output("Setting Service Level to %s beginning %s" % (serviceLevel, serviceLevelStart.dateAndTime(4)))
    currentVariable.setValueRange(
        currentRuntimestep,
        serviceLevelStart,
        rtw.getEndTime(),
        serviceLevel)
    # network.getStateVariable("SLFlood1MKC").setValue(currentRuntimestep, serviceLevel+ 36000)
    if jDay == MAR_15 and scenario != FTT and serviceLevel < STEADY_STATE_COMPUTATION_THRESHOLD and serviceLevel != NO_SERVICE:
        #
        # set steady release and cycled release state variables
        #
        steadyRelease = Constants.UNDEFINED
        quantiles = userInfo["systemQuantiles"][year]
        for i in range(5)[::-1] :
            if fcst >= quantiles[i] : break
        if i < 2 : # below median forecast runoff
            if serviceLevel >= FULL_SERVICE :
                steadyRelease = BELOW_MEDIAN_FULL_SVC_STEADY_RELEASE
            elif serviceLevel <= MINIMUM_SERVICE :
                steadyRelease = BELOW_MEDIAN_MIN_SVC_STEADY_RELEASE
            else :
                steadyRelease = BELOW_MEDIAN_MIN_SVC_STEADY_RELEASE + \
                    (serviceLevel - MINIMUM_SERVICE) / \
                    (FULL_SERVICE - MINIMUM_SERVICE) * \
                    (BELOW_MEDIAN_FULL_SVC_STEADY_RELEASE - BELOW_MEDIAN_MIN_SVC_STEADY_RELEASE)
        else: # median or above forecast runoff
            if serviceLevel >= FULL_SERVICE :

```

```

        steadyRelease = MEDIAN_OR_ABOVE_FULL_SVC_STEADY_RELEASE
    elif serviceLevel <= MINIMUM_SERVICE :
        steadyRelease = MEDIAN_OR_ABOVE_MIN_SVC_STEADY_RELEASE
    else :
        steadyRelease = MEDIAN_OR_ABOVE_MIN_SVC_STEADY_RELEASE + \
            (serviceLevel - MINIMUM_SERVICE) / \
            (FULL_SERVICE - MINIMUM_SERVICE) * \
            (MEDIAN_OR_ABOVE_FULL_SVC_STEADY_RELEASE - MEDIAN_OR_ABOVE_MIN_SVC_STEADY_RELEASE)
startOffset = MAY_15 - jDay
transOffset = JUN_01 - jDay
endOffset = JUL_15 - jDay - 1
curStep = currentRuntimestep.getStep()
for offset in range(startOffset, endOffset) :
    if scenario == CRFTT and offset < transOffset :
        cyclePeriod = (offset - MAY_15) % (CYCLED_RELEASE_STEADY_PERIOD + CYCLED_RELEASE_CUTBACK_PERIOD) + 1
        if cyclePeriod <= CYCLED_RELEASE_STEADY_PERIOD :
            release = steadyRelease
        else :
            release = steadyRelease - CYCLED_RELEASE_CUTBACK
    currentRuntimestep.setStep(curStep + offset)
    if scenario == SRFTT :
        network.getStateVariable("GAPTSteadyRelease").setValue(currentRuntimestep, steadyRelease)
    else : # scenario == CRFTT
        network.getStateVariable("GAPTCycledRelease").setValue(currentRuntimestep, release)
currentRuntimestep.setStep(curStep)

if navSeasonEnd is not None :
    #
    # set the navigation season end-date for the remainder of the simulation period
    #
    network.getStateVariable("NavigationEndDate").setValueRange(
        currentRuntimestep,
        t,
        rtw.getEndTime(),
        navSeasonEnd)
    if SEP_01 < jDay <= navSeasonEnd:
        curStep = currentRuntimestep.getStep()
        #
        # un-set the winter release from NORMAL_NAV_END_DATE to navigation season end-date
        #
        if navSeasonEnd > NORMAL_NAV_END_DATE :
            if verbose : output("Navigation season end = %d" % navSeasonEnd)
            currentRuntimestep.setStep(curStep + NORMAL_NAV_END_DATE - jDay + 1)
            t1 = currentRuntimestep.getHecTime()
            t2 = HecTime(t1)
            t2.addMinutes((navSeasonEnd - NORMAL_NAV_END_DATE) * 1440)
            if verbose : output("Trimming beginning of winter release from %s to %s" % (t1.dateAndTime(), t2.dateAndTime()))
            network.getStateVariable("GAPTWinterRelease").setValueRange(
                currentRuntimestep,
                t1,
                t2,
                Constants.UNDEFINED)
        #

```

```

# un-set the navigation season end-date after it ends
#
currentRuntimestep.setStep(curStep + navSeasonEnd - jDay + 1)
t1 = currentRuntimestep.getHecTime()
t2 = rtw.getEndTime()
network.getStateVariable("NavigationEndDate").setValueRange(
    currentRuntimestep,
    t1,
    t2,
    Constants.UNDEFINED)
#
# set the service level to NO_SERVICE after the navigation season end-date minus an offset
#
currentRuntimestep.setStep(curStep + navSeasonEnd - SERVICE_LEVEL_END_OFFSET - jDay + 1)
t1 = currentRuntimestep.getHecTime()
if verbose : output("Turning Service Level off beginning %s" % t1.dateAndTime(4))
currentVariable.setValueRange(
    currentRuntimestep,
    t1,
    t2,
    Constants.UNDEFINED)
currentRuntimestep.setStep(curStep)

if gaptAvg is not None :
    #
    # set the Gavins Point average winter release
    #

    assert(navSeasonEnd is not None)
    winterReleaseStart = max(NORMAL_NAV_END_DATE, navSeasonEnd)
    curStep = currentRuntimestep.getStep()
    currentRuntimestep.setStep(curStep + winterReleaseStart - jDay + 1)
    t = currentRuntimestep.getHecTime()
    network.getStateVariable("GAPTWinterRelease").setValueRange(
        currentRuntimestep,
        t,
        rtw.getEndTime(),
        gaptAvg)
    currentRuntimestep.setStep(curStep)

```

```

#####
##### STATE VARIABLE SCRIPT CLEANUP SECTION
#####

```

```

from hec.script import Constants
#
# script to be run only once, at the end of the compute. optional.

```

```

# variables that are available to this script during the compute:
#     currentVariable - the StateVariable that holds this script
#     network - the ResSim network

```

```

# The following represents an undefined value in a time series:

```

```
# Constants.UNDEFINED

userInfo = currentVariable.varGet("userInfo")
outputByDate = userInfo["outputByDate"]
simDates = outputByDate.keys()
simDates.sort()
for simDate in simDates :
    for line in outputByDate[simDate] :
        network.printLogMessage(line)
outputByDate = {}
# userInfo["plotStateVariables"]()
```

## 12 APPENDIX F – SERVICE LEVEL X1K STATE VARIABLE

```
#####
##### STATE VARIABLE SCRIPT INITIALIZATION SECTION
#####

'''
Description: Version 2015-05-18 of the service level x1k state variable script. It uses the service level from the service level state variable
to calculate the target discharges for each target location and writes them to slave state variables.
'''
```

```
from hec.script import Constants
#
# initialization function. optional.
# set up tables and other things that only need to be performed once at the start of the compute.
#
# variables that are passed to this script during the compute initialization:
#     currentVariable - the StateVariable that holds this script
#     network - the ResSim network
#
def initStateVariable(currentVariable, network):

    # -----
    # Functions
    # -----

    # Debugging Function: Set debug = True to print all debug statements and = False to turn them off
    # Initialization Debugging Function: Set debug = True to print all debug statements and = False to turn them off
    debug = True
    def initOutputDebug(debug, arg1=", arg2=", arg3=", arg4=", arg5=", arg6=", arg7=", arg8=", arg9=", arg10=", arg11=", arg12=", arg13=", arg14="):
        if debug == True:
            print 'Initialization Debug  \t', arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9, arg10, arg11, arg12, arg13, arg14

    # Debugging Function: Set debug = True to print all debug statements and = False to turn them off
    def outputDebug(currentRuntimestep, debug, arg1=", arg2=", arg3=", arg4=", arg5=", arg6=", arg7=", arg8=", arg9=", arg10=", arg11=", arg12=", arg13=", arg14=", arg15="):
        PassCounter = network.getComputePassCounter() # Pass number of simulation
        PassNumber = PassCounter + 1 # Pass number starts at 0 so add 1 to start it at 1
        if debug == True:
            print 'Pass Number =', PassNumber, ' ', currentRuntimestep.dateTimeString(), \
                '\t', arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9, arg10, arg11, arg12, arg13, arg14, arg15, '\n'

    # julianDay Function: Get the julian day for a month and day or a string, or a month and day for a julian day
    def julianDay(*args) :
        #     J F M A M J J A S O N D
        daysPrev = 0, 31, 59, 90,120,151,181,212,243,273,304,334
        argCount = len(args)
        if argCount == 1 :
            if type(args[0]) == type(0) :
                jday = args[0]
                if not 1 <= jday <= 365 :
                    raise ValueError("Julian day out of range 1..365")
            for i in range(12)[::-1] :
                if jday > daysPrev[i] :
                    mon = i + 1
                    break
```

```

        day = jday - daysPrev[mon-1]
        return mon, day
    else :
        mon, day = args[0].split(" ")
        mon = ["JAN", "FEB", "MAR", "APR", "MAY", "JUN", "JUL", "AUG", "SEP", "OCT", "NOV", "DEC"].index(mon.upper()) + 1
        day = int(day);
        return julianDay(mon, day)
elif argCount == 2 :
    mon, day = args
    if not 1 <= mon <= 12 :
        raise ValueError("Month is out of range 1..12")
    if not 1 <= day <= 31 :
        raise ValueError("Day is out of range 1..31")
    if mon == 2 and day == 29 : day = 28
    return daysPrev[mon-1] + day
else :
    raise ValueError("Expected 1 or 2 arguments, got %d" % argCount)

```

```

# -----
# init info that is passed to runRuleScript()
# -----

```

```

initInfo = {
    'debug'          : debug,
    'julianDay'      : julianDay,
    'outputDebug'    : outputDebug
}

```

```

currentVariable.varPut('initInfo', initInfo)

```

```

# return Constants.TRUE if the initialization is successful and Constants.FALSE if it failed.
# Returning Constants.FALSE will halt the compute.
return Constants.TRUE

```

```

#####
##### STATE VARIABLE SCRIPT COMPUTATION SECTION
#####

```

```

'''
Description: Version 2015-05-18 of the service level x1k state variable script. It uses the service level from the service level state variable
to calculate the target discharges for each target location and writes them to slave state variables.
'''

```

```

# variables that are available to this script during the compute:
#   currentVariable - the StateVariable that holds this script
#   currentRuntimestep - the current RunTime step
#   network - the ResSim network

```

```

# The following represents an undefined value in a time series
#   Constants.UNDEFINED

```

```

# required imports to create the OpValue return object.
from hec.rss.model import OpValue

```

```

from hec.rss.model import OpRule
from hec.script import Constants

# -----
# Retrieve init info from initRuleScript() and set equal to variables
# -----

initInfo = currentVariable.varGet('initInfo')

debug      = initInfo['debug']
julianDay  = initInfo['julianDay']
outputDebug = initInfo['outputDebug']

# -----
# Input Data
# -----

# Simulation Time & Step Info
currentDate = currentRuntimestep.getHecTime()
rtw = currentRuntimestep.getRunTimeWindow() # Run time window
curStep = currentRuntimestep.getStep() # Current step
currentMonth = currentRuntimestep.getHecTime().month()
currentDay = currentRuntimestep.getHecTime().day()
jDay = julianDay(currentMonth, currentDay)

# Get the service level @ currentTimeStep
serviceLevel = network.getStateVariable("ServiceLevel").getValue(currentRuntimestep)
NavigationEndDate = network.getStateVariable("NavigationEndDate").getValue(currentRuntimestep)
serviceLevel_x1K = Constants.UNDEFINED

# Get the GAPT Winter Release @ currentTimeStep
GAPTwinterRelease = network.getStateVariable("GAPTWinterRelease").getValue(currentRuntimestep)

# Dry Year Check
dryYear = network.getStateVariable("DryYear").getPreviousValue(currentRuntimestep)

if currentMonth == 3 and currentDay >= 15 :
    if serviceLevel == Constants.UNDEFINED :
        dryYear = Constants.TRUE
    else :
        dryYear = Constants.FALSE
elif dryYear == Constants.TRUE and 3 <= currentMonth <= 12 :
    dryYear = Constants.TRUE
else :
    dryYear = Constants.FALSE
network.getStateVariable("DryYear").setValue(currentRuntimestep, dryYear)

# Set Service Level
if serviceLevel >= 0 and (dryYear == Constants.FALSE or dryYear == Constants.UNDEFINED):
    if jDay <= julianDay('JUL_01'):
        NavigationEndDate = julianDay('DEC_01')

    #SL = Constants.TRUE

```

```
serviceLevel_x1K = 1000 * serviceLevel
SL_SUX = serviceLevel_x1K - 4000
SL_OMA = serviceLevel_x1K - 4000
SL_NCNE = serviceLevel_x1K + 2000
SL_MKC = serviceLevel_x1K + 6000
```

```
# Set step for navigation start dates for assessing target discharges to account for travel time. Dates are based on opening dates listed
```

```
# in Section 7-03.4.1 of the Master Manual.
```

```
SUX_Nav_Start_Date = julianDay('MAR_23')
OMA_Nav_Start_Date = julianDay('MAR_25')
NCNE_Nav_Start_Date = julianDay('MAR_26')
MKC_Nav_Start_Date = julianDay('MAR_28')
```

```
# Set julian day for navigation end dates. Dates are based on end dates listed in Section 7-03.4.1 of the Master Manual.
```

```
SUX_Nav_End_Date = NavigationEndDate - 9
OMA_Nav_End_Date = NavigationEndDate - 7
NCNE_Nav_End_Date = NavigationEndDate - 6
MKC_Nav_End_Date = NavigationEndDate - 4
if jDay < SUX_Nav_Start_Date or jDay > SUX_Nav_End_Date:
    SL_SUX = Constants.UNDEFINED
if jDay < OMA_Nav_Start_Date or jDay > OMA_Nav_End_Date:
    SL_OMA = Constants.UNDEFINED
if jDay < NCNE_Nav_Start_Date or jDay > NCNE_Nav_End_Date:
    SL_NCNE = Constants.UNDEFINED
if jDay < MKC_Nav_Start_Date or jDay > MKC_Nav_End_Date:
    SL_MKC = Constants.UNDEFINED
```

```
else :
```

```
serviceLevel_x1K = Constants.UNDEFINED
SL_SUX = Constants.UNDEFINED
SL_OMA = Constants.UNDEFINED
SL_NCNE = Constants.UNDEFINED
SL_MKC = Constants.UNDEFINED
```

```
# Save the Service Level in multiples of 1K
```

```
currentVariable.setValue(currentRuntimestep, serviceLevel_x1K)
network.getStateVariable("SL_SUX").setValue(currentRuntimestep, SL_SUX)
network.getStateVariable("SL_OMA").setValue(currentRuntimestep, SL_OMA)
network.getStateVariable("SL_NCNE").setValue(currentRuntimestep, SL_NCNE)
network.getStateVariable("SL_MKC").setValue(currentRuntimestep, SL_MKC)
```

```
#####
```

```
##### STATE VARIABLE SCRIPT CLEANUP SECTION
```

```
#####
```

```
from hec.script import Constants
```

```
#
```

```
# script to be run only once, at the end of the compute. optional.
```

```
# variables that are available to this script during the compute:
```

```
# currentVariable - the StateVariable that holds this script
```

```
# network - the ResSim network
```

```
# The following represents an undefined value in a time series:  
# Constants.UNDEFINED  
  
# add your code here...
```

## 13 APPENDIX G – SPRING PULSE SCRIPTED RULE

'''

Author: Ryan Larsen

Date: 06-22-2015

Description: Logic for the spring pulse as summarized from the Missouri River Master Manual. Accounts for James River flows at Scotland, SD and uses forecasted local inflows. This script uses the FWOP local flow datasets and should be used for all alternatives that are compared to the FWOP alternatives.

# -----  
# Spring Pulse Table Summary  
# -----

	Criteria for March Spring Pulse -----	Criteria for May Spring Pulse -----
Drought Preclude	System storage at 40.0 or below measured on March 1st	System storage at 40.0 or below measured on May 1st
Drought Proration of Pulse Magnitude Based on System storage	None, 5 kcfs added to navigation releases, but no greater than 35 kcfs and linear interpolation to 12 kcfs at 40.0 MAF	Prorated from 16 kcfs based on a May 1st System storage check; 16 kcfs at 54.5 MAF
Drought Proration of Pulse Magnitude Based on System storage	None releases, but no greater than 35 kcfs	After proration for System storage, magnitude is further adjusted based on forecasted runoff: Additional 0 kcfs for Median runoff Linear interpolation to additional 4 kcfs for runoff >= Upper Quartile Linear interpolation to additional -4 kcfs for runoff <= Lower Quartile
Intitiation of Pulse	Extend the stepped System release increases that precede the beginning of the navigation season	Between May 1 to May 19, depending on Missouri River water temperature immediately below Gavins Point Dam. If possible, pulse will be intitiated after the second daily occurrence of a 16 degree Celsius water temperature; however, this decision will be made with consideration for potential 'take' of threatened and endangered bird species. FOR RESSIM, THIS WILL BE IGNORED.
Rate of Rise before Peak	Approximately 5 kcfs per day	Approximately 6 kcfs per day
Duration of Peak	2 days	2 days
Rate of Fall after Peak	Drop over 5 days to navigation target release	Approximately 30% drop over 2 days followed by a proportional reduction in releases back to the existing Master Manual criteria over an 8 day period
Downstream Constraints	Flows at OMA, NCNE, and MKC are not exceed 41000, 47000, and 71000, respectively	Flows at OMA, NCNE, and MKC are not exceed 41000, 47000, and 71000, respectively'''

# -----  
# Required imports to create the OpValue return object.  
# -----  
from hec.rss.model import OpRule  
from hec.script import Constants  
from hec.heclib.util import \*  
from hec.rss.model import \*

```

from hec.heclib.dss import *
from hec.hecmath import *
from hec.model import *
from hec.script import *
from hec.io import TimeSeriesContainer
from hec.client import ClientApp
from java.lang import String
from zipfile import ZipFile
import traceback

# -----
# Global variables
# -----
global SUX_Local
global OMA_Local
global NCNE_Local
global RUNE_Local
global STJ_Local
global MKC_Local
global System_Release_Data
global curStep

# initialization function. optional.
# set up tables and other things that only need to be performed once during the compute.
# currentRule is the rule that holds this script.
# network is the ResSim network.
def initRuleScript(currentRule, network):
    # -----
    # Set alternative name and depletions to read correct DSS datasets
    # -----

    Depletions = 'Present' # If Depletions = Present, then script will use local flow datasets that have been adjusted for present level depletions
                           # If Depletions = Historic, then script will use local flow datasets that are based on historic level depletions

    # -----
    # Functions
    # -----

    # Debugging Function: Set debug = True to print all debug statements and = False to turn them off
    # Initialization Debugging Function: Set debug = True to print all debug statements and = False to turn them off
    debug = False
    def initOutputDebug(debug, arg1="", arg2="", arg3="", arg4="", arg5="", arg6="", arg7="", arg8="", arg9="", arg10="", arg11="", arg12="", arg13="", arg14=""):
        if debug == True:
            print 'Initialization Debug  \t', arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9, arg10, arg11, arg12, arg13, arg14

    # Debugging Function: Set debug = True to print all debug statements and = False to turn them off
    def outputDebug(currentRuntimestep, debug, arg1="", arg2="", arg3="", arg4="", arg5="", arg6="", arg7="", arg8="", arg9="", arg10="", arg11="", arg12="", arg13="", arg14="", arg15=""):
        PassCounter = network.getComputePassCounter() # Pass number of simulation
        PassNumber = PassCounter + 1 # Pass number starts at 0 so add 1 to start it at 1
        if debug == True:
            print 'Pass Number =', PassNumber, ' ', currentRuntimestep.dateTimeString(), \
                '\t', arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9, arg10, arg11, arg12, arg13, arg14, arg15, '\n'

```

```

# floodLimits Function: Check flood limits associated with spring pulse within the Master Manual
def floodLimits(      currentRuntimestep, # currentRuntimestep variable passed in through the runRuleScript()
                    curStep,           # Current step of the simulation
                    Route_Step,        # Step adjustment used for routing of flows
                    System_Release,    # Gavins Point release
                    Pulse               # Magnitude of the pulse that is added to Gavins Point release
                    ):
# Travel times for target locations
SUX_Travel = len(initInfo['coeffDict']['GAPT']) - 1 # Travel time from GAPT to SUX
OMA_Travel = SUX_Travel + (len(initInfo['coeffDict']['SUX']) - 1) # Travel time from GAPT to OMA
NCNE_Travel = OMA_Travel + (len(initInfo['coeffDict']['OMA']) - 1) # Travel time form GAPT to NCNE
RUNE_Travel = NCNE_Travel + (len(initInfo['coeffDict']['NCNE']) - 1) # Travel time from GAPT to RUNE
STJ_Travel = RUNE_Travel + (len(initInfo['coeffDict']['RUNE']) - 1) # Travel time from GAPT to STJ
MKC_Travel = STJ_Travel + (len(initInfo['coeffDict']['STJ']) - 1) # Travel time from GAPT to MKC

# Route GAPT_Release to OMA, NCNE, and MKC
Total_SUX, Total_OMA, Total_NCNE, Total_RUNE, Total_STJ, Total_MKC = routeFlow_GAPT_to_MKC(currentRuntimestep, curStep, Route_Step, System_Release,
SUX_Local, OMA_Local, NCNE_Local, RUNE_Local, STJ_Local, MKC_Local)

# Check flood constraints
OMA_MAX = max(Total_OMA[Route_Step + OMA_Travel:])
NCNE_MAX = max(Total_NCNE[Route_Step + NCNE_Travel:])
MKC_MAX = max(Total_MKC[Route_Step + MKC_Travel:])

if OMA_MAX > constDict['OMA'] or NCNE_MAX > constDict['NCNE'] or MKC_MAX > constDict['MKC']:
    exceeded = 'Yes'
else: exceeded = 'No'
return exceeded

# forecastLocal Function: Forecast local inflow
def forecastLocal(      currentRuntimestep, # currentRuntimestep variable passed in through the runRuleScript()
                    curStep,           # Current step of the simulation
                    Location,          # DCP ID of forecast location
                    Flow_Data         # Local flow dataset used for forecasting
                    ):
Forecast_Flow_Data = Flow_Data[:]
for i in range(len(Forecast_Coeff_Dict[Location])):
    Forecasted_Local = 0
    # Calculate forecasted inflow for i days in the future
    for t in range(len(Forecast_Coeff_Dict[Location][i])):
        Forecasted_Local += Flow_Data[curStep - t] * Forecast_Coeff_Dict[Location][i][t]
        outputDebug(currentRuntimestep, currentRuntimestep, False, '%s forecasted local inflow = %.0f' % (Location, Forecasted_Local),
            '\t%s observed local inflow = %.0f' % (Location, Flow_Data[curStep - t]),
            '\t%s forecast coefficient = ' % Location, Forecast_Coeff_Dict[Location][i][t])
    Forecast_Flow_Data[curStep + 1 + i] = Forecasted_Local
outputDebug(currentRuntimestep, False, '%s observed local flow = ' % Location, Flow_Data[curStep:(curStep + len(Forecast_Coeff_Dict[Location]) + 1)],
    '\n\t\t%s forecast local flow = ' % Location, Forecast_Flow_Data[curStep:(curStep + len(Forecast_Coeff_Dict[Location]) + 1)])
return Forecast_Flow_Data

# julianDay Function: Get the julian day for a month and day or a string, or a month and day for a julian day
def julianDay(*args) :
#     J F M A M J J A S O N D
    daysPrev = 0, 31, 59, 90,120,151,181,212,243,273,304,334

```

```

argCount = len(args)
if argCount == 1 :
    if type(args[0]) == type(0) :
        jday = args[0]
        if not 1 <= jday <= 365 :
            raise ValueError("Julian day out of range 1..365")
        for i in range(12)[::-1] :
            if jday > daysPrev[i] :
                mon = i + 1
                break
        day = jday - daysPrev[mon-1]
        return mon, day
    else :
        mon, day = args[0].split(" ")
        mon = ["JAN", "FEB", "MAR", "APR", "MAY", "JUN", "JUL", "AUG", "SEP", "OCT", "NOV", "DEC"].index(mon.upper()) + 1
        day = int(day);
        return julianDay(mon, day)
elif argCount == 2 :
    mon, day = args
    if not 1 <= mon <= 12 :
        raise ValueError("Month is out of range 1..12")
    if not 1 <= day <= 31 :
        raise ValueError("Day is out of range 1..31")
    if mon == 2 and day == 29 : day = 28
    return daysPrev[mon-1] + day
else :
    raise ValueError("Expected 1 or 2 arguments, got %d" % argCount)

# March_Pulse_Release Function: Add pulse magnitude to the navigaiton releases to create the March pulse release pattern
def March_Pulse_Release( curStep, # Current step of the simulation
                        Pulse_Release_Data, # Release data that incorporates the pulse
                        Navigation_Release, # Original navigation release
                        System_Release_Data, # Gavins Point release data
                        Pulse, # Magnitude of the pulse that is added to Gavins Point release
                        ):
    # 1st and 2nd day of peak
    Pulse_Release_Data[curStep] = Navigation_Release + Pulse
    Pulse_Release_Data[curStep + 1] = Navigation_Release + Pulse

    # Descending limb of pulse
    for step in range(2, 27):
        Pulse_Release_Data[curStep + step] = Pulse_Release_Data[curStep + step - 1] - 1000

# May_Pulse_Release Function: Add pulse magnitude to the navigaiton releases to create the May pulse release pattern
def May_Pulse_Release( curStep, # Current step of the simulation
                      Pulse_Release_Data, # Release data that incorporates the pulse
                      Navigation_Release, # Original navigation release
                      Pulse, # Magnitude of the pulse that is added to Gavins Point release
                      ):
    # May 01-02: 1st and 2nd day of release increase
    Pulse_Release_Data[curStep] = Navigation_Release + (Pulse / 3)
    Pulse_Release_Data[curStep + 1] = Pulse_Release_Data[curStep] + (Pulse / 3)

```

```

# May 03-04: 1st and 2nd day of peak
Pulse_Release_Data[curStep + 2] = Pulse_Release_Data[curStep + 1] + (Pulse / 3)
Pulse_Release_Data[curStep + 3] = Pulse_Release_Data[curStep + 2]

# May 05-06: 1st and 2nd day of release decrease
Pulse_Release_Data[curStep + 4] = Pulse_Release_Data[curStep + 3] - (Pulse * 0.15)
Pulse_Release_Data[curStep + 5] = Pulse_Release_Data[curStep + 4] - (Pulse * 0.15)

# May 07-14: Decrease release back to navigation releases
for step in range(6, 20):
    Pulse_Release_Data[curStep + step] = Pulse_Release_Data[curStep + step - 1] - ((Pulse * 0.7) / 8)

```

```

# retrieveModelVariables Function: Retrieve model variables needed for the script
def retrieveModelVariables( currentRuntimestep, # currentRuntimestep variable passed in through the runRuleScript()
                           totalSteps        # Total number of steps in the simulation
                           ):
    Model_Variable = { # System/GAPT Model Variables
                      'System_Previous_Release' : network.getTimeSeries("Reservoir","System Reservoir", "Pool", "Flow-OUT").getPreviousValue(currentRuntimestep),
                      'System_Laggedx2_Release' : network.getTimeSeries("Reservoir","System Reservoir", "Pool", "Flow-OUT").getLaggedValue(currentRuntimestep, 2),
                      'System_Previous_Storage' : network.getTimeSeries("Reservoir","System Reservoir", "Pool", "Stor").getPreviousValue(currentRuntimestep),
                      'System_Release_Data'    : network.getTimeSeries("Reservoir","System Reservoir", "Pool", "Flow-OUT").getDataArrayFor(0, totalSteps)
                    }
    return Model_Variable

```

```

# retrieveModelVariables Function: Retrieve model variables needed for the script
def retrieveStateVariables( currentRuntimestep # currentRuntimestep variable passed in through the runRuleScript()
                            ):
    State_Variable = { # System State Variables
                     'System_Current_Storage_SV' : network.getStateVariable("MainstemStorage").getValue(currentRuntimestep),
                     'May_Forecast_System_Runoff_SV' : network.getStateVariable("FcstRunoff").getValue(currentRuntimestep)
                    }
    return State_Variable

```

```

# routeFlow Function: Route flow from an upstream location to a downstream location
def routeFlow(currentRuntimestep, # currentRuntimestep variable passed in through the runRuleScript()
              curStep,           # Current step of the simulation
              Upstream_Location, # Upstream DCP ID
              Flow_Dataset       # Flow data that will be routed
              ):
    coeffList = coeffDict[Upstream_Location]
    lag = len(coeffList) - 1 # Length of time before any portion of flow reaches the downstream location
    routedFlow = []
    for i in range(len(Flow_Dataset)):
        if i < lag:
            routedFlow.append(0) # Add a place holder of 0 for days less than the lag time
        else:
            routedFlowTempList = []
            for c in range(len(coeffList)):
                flow = Flow_Dataset[i-c] * coeffList[c]
                routedFlowTempList.append(flow)
            release = sum(routedFlowTempList)
            routedFlow.append(release)
    return routedFlow

```

```

# routeFlow_GAPT_to_MKC Function: Route flow from GAPT to all locations downstream to MKC
def routeFlow_GAPT_to_MKC(      currentRuntimeStep, # currentRuntimeStep variable passed in through the runRuleScript()
                                curStep,          # Current step of the simulation
                                Route_Step,      # Step adjustment used for routing of flows
                                System_Release,   # GAPT releases
                                SUX_Local,       # Local flows at SUX
                                OMA_Local,       # Local flows at OMA
                                NCNE_Local,     # Local flows at NCNE
                                RUNE_Local,     # Local flows at RUNE
                                STJ_Local,      # Local flows at STJ
                                MKC_Local       # Local flows at MKC
                                ):
    Total_SUX, Total_OMA, Total_NCNE, Total_RUNE, Total_STJ, Total_MKC = [], [], [], [], [], []

    # Route GAPT release to SUX
    GAPT_to_SUX = routeFlow(currentRuntimeStep, curStep, 'GAPT', System_Release[curStep - Route_Step:curStep + Route_Step])
    # Forecast local inflows for GAPT to SUX reach
    SUX_Local_Forecast = forecastLocal(currentRuntimeStep, curStep, 'SUX', SUX_Local)
    # Combine routed GAPT release and SUX forecasted local inflows
    for step in range(len(GAPT_to_SUX)):
        Total_SUX.append(GAPT_to_SUX[step] + SUX_Local_Forecast[curStep - Route_Step + step])

    # Route forecasted SUX flow to OMA
    SUX_to_OMA = routeFlow(currentRuntimeStep, curStep, 'SUX', Total_SUX)
    # Forecast local inflows for SUX to OMA reach
    OMA_Local_Forecast = forecastLocal(currentRuntimeStep, curStep, 'OMA', OMA_Local)
    # Combine routed SUX flow and OMA forecasted local inflows
    for step in range(len(Total_SUX)):
        Total_OMA.append(SUX_to_OMA[step] + OMA_Local_Forecast[curStep - Route_Step + step])

    # Route forecasted OMA flow to NCNE
    OMA_to_NCNE = routeFlow(currentRuntimeStep, curStep, 'OMA', Total_OMA)
    # Forecast local inflows for OMA to NCNE reach
    NCNE_Local_Forecast = forecastLocal(currentRuntimeStep, curStep, 'NCNE', NCNE_Local)
    # Combine routed OMA flow and NCNE forecasted local inflows
    for step in range(len(Total_OMA)):
        Total_NCNE.append(OMA_to_NCNE[step] + NCNE_Local[curStep - Route_Step + step])

    # Route forecasted NCNE flow to RUNE
    NCNE_to_RUNE = routeFlow(currentRuntimeStep, curStep, 'NCNE', Total_NCNE)
    # Forecast local inflows for NCNE to RUNE reach
    RUNE_Local_Forecast = forecastLocal(currentRuntimeStep, curStep, 'RUNE', RUNE_Local)
    # Combine routed NCNE flow and RUNE forecasted local inflows
    for step in range(len(Total_NCNE)):
        Total_RUNE.append(NCNE_to_RUNE[step] + RUNE_Local_Forecast[curStep - Route_Step + step])

    # Route forecasted RUNE flow to STJ
    RUNE_to_STJ = routeFlow(currentRuntimeStep, curStep, 'RUNE', Total_RUNE)
    # Forecast local inflows for RUNE to STJ reach
    STJ_Local_Forecast = forecastLocal(currentRuntimeStep, curStep, 'STJ', STJ_Local)
    # Combine routed RUNE flow and STJ forecasted local inflows
    for step in range(len(Total_RUNE)):

```

```

        Total_STJ.append(RUNE_to_STJ[step] + STJ_Local_Forecast[curStep - Route_Step + step])

# Route forecasted STJ flow to MKC
STJ_to_MKC = routeFlow(currentRuntimestep, curStep, 'STJ', Total_STJ)
# Forecast local inflows for STJ to MKC reach
MKC_Local_Forecast = forecastLocal(currentRuntimestep, curStep, 'MKC', MKC_Local)
# Combine routed STJ flow and MKC forecasted local inflows
for step in range(len(Total_STJ)):
    Total_MKC.append(STJ_to_MKC[step] + MKC_Local_Forecast[curStep - Route_Step + step])

return Total_SUX, Total_OMA, Total_NCNE, Total_RUNE, Total_STJ, Total_MKC

# -----
# Input Data
# -----

# Simulation Time & Step Info
rtw = ResSim.getCurrentModule().getSimulation().getRunTimeWindow()
totalSteps = rtw.getNumSteps() / 24 # Total number of steps in the simulation. rtw gives steps in hours so divide by 24 to get days
startRtw = rtw.getStartTime() # Start time of simulation
lookbackRtw = rtw.getLookbackTime() # Start of lookback period
endRtw = rtw.getEndTime() # End time of simulation
numLookbackSteps = rtw.getNumLookbackSteps() / 24 # Number of lookback steps in simulation. rtw gives steps in hours so divide by 24 to get days
simStartStep = numLookbackSteps + 1 # First step after the lookback period

initOutputDebug(debug, 'Run Time Window Start Time =', rtw.getStartTimeString(), '\tRun Time Window End Time =', rtw.getEndTimeString(), \
    '\tNumber of Lookback Steps =', numLookbackSteps, '\tStart of Simulation Step =', simStartStep, '\tTotal Number of Steps =', totalSteps)

# DSS Data
shared = ResSim.getWatershed()
module = ClientAppWrapper.getCurrentModule()
if module.getName() != 'Simulation':
    raise AssertionError, 'ResSim in %s module. Simulation module is required.' % module
shared = str(shared)
filePath = shared + '/shared/Input_Data.dss'
filePath = filePath.replace('/', '\\')
dssFile = HecDss.open(filePath)
dssFile.setTimeWindow(str(lookbackRtw), str(endRtw)) # Open dss file with a time window equal to the run time window in ResSim

initOutputDebug(debug, 'DSS File =', filePath, '\tDSS Time Window: %s - %s' % (str(lookbackRtw), str(endRtw)))

if Depletions == 'Present':
    SUX_Local = dssFile.read('/MISSOURI RIVER/SUX/FLOW-LOCAL//1DAY/FWOP: CALCULATED/').getData().values
    OMA_Local = dssFile.read('/MISSOURI RIVER/OMA/FLOW-LOCAL//1DAY/FWOP: CALCULATED/').getData().values
    NCNE_Local = dssFile.read('/MISSOURI RIVER/NCNE/FLOW-LOCAL//1DAY/FWOP: CALCULATED/').getData().values
    RUNE_Local = dssFile.read('/MISSOURI RIVER/RUNE/FLOW-LOCAL//1DAY/FWOP: CALCULATED/').getData().values
    STJ_Local = dssFile.read('/MISSOURI RIVER/STJ/FLOW-LOCAL//1DAY/FWOP: CALCULATED/').getData().values
    MKC_Local = dssFile.read('/MISSOURI RIVER/MKC/FLOW-LOCAL//1DAY/FWOP: CALCULATED/').getData().values
    SCSD_Flow = dssFile.read('/JAMES RIVER/SCSD/FLOW-OUT//1DAY/HISTORIC: OBSERVED/').getData().values
elif Depletions == 'Historic':
    SUX_Local = dssFile.read('/MISSOURI RIVER/SUX/FLOW-LOCAL//1DAY/HISTORIC: CALCULATED SMOOTH/').getData().values
    OMA_Local = dssFile.read('/MISSOURI RIVER/OMA/FLOW-LOCAL//1DAY/HISTORIC: CALCULATED SMOOTH/').getData().values
    NCNE_Local = dssFile.read('/MISSOURI RIVER/NCNE/FLOW-LOCAL//1DAY/HISTORIC: CALCULATED SMOOTH/').getData().values

```

```

RUNE_Local = dssFile.read('/MISSOURI RIVER/RUNE/FLOW-LOCAL//1DAY/HISTORIC: CALCULATED SMOOTH/').getData().values
STJ_Local = dssFile.read('/MISSOURI RIVER/STJ/FLOW-LOCAL//1DAY/HISTORIC: CALCULATED SMOOTH/').getData().values
MKC_Local = dssFile.read('/MISSOURI RIVER/MKC/FLOW-LOCAL//1DAY/HISTORIC: CALCULATED REGULATED SMOOTH/').getData().values
SCSD_Flow = dssFile.read('/JAMES RIVER/SCSD/FLOW-OUT//1DAY/HISTORIC: OBSERVED/').getData().values

```

```

else:
    raise AssertionError, 'Depletions error. Specify as either Present or Historic.'

```

```

# Spring Pulse Data
Pulse_Preclude = 40.0 # Required system storage for either of the spring pulses
March_Pulse_ROC_Initiate = 1000 # Based on the "Stop Spike" rule. ROC for increasing releases during navigation season is 1440 cfs/day.
MAY_01_Upper_Quartile_Runoff = 14.7
MAY_01_Lower_Quartile_Runoff = 9.575
MAY_01_Max_System_Storage_1P = 54.5 # System storage for max pulse during first proration of magnitude
Max_May_Pulse_1P = 16000.0 # Max pulse during first proration of magnitude
Min_May_Pulse_1P = 12000.0 # Min pulse during first proration of magnitude
Max_May_Pulse_2P = 4000.0 # Max additional release added to first prorated pulse magnitude
Min_May_Pulse_2P = -3000.0 # Min additional release added to first prorated pulse magnitude
Pulse_Release_Data = [] # Array of discharges that the pulse discharges will be written to
for step in range(len(SUX_Local)):
    Pulse_Release_Data.append(0)

```

```

# Downstream Flood Limits: Limits are from the Master Manual and are based on the flood constraints when at full service
constDict = {
    'OMA' : 41000,
              'NCNE': 47000,
              'MKC' : 71000
}

```

```

# Routing Coefficients
coeffDict = {
    'GAPT' : [0.175, 0.538, 0.287], # GAPT to SUX coefficients
    'SUX' : [0.168, 0.722, 0.110], # SUX to OMA coefficients
    'OMA' : [0.588, 0.412], # OMA to NCNE coefficients
    'NCNE': [0.588, 0.412], # NCNE to RUNE coefficients
    'RUNE': [0.775, 0.225], # RUNE to STJ coefficients
    'STJ' : [0.426, 0.449, 0.125], # STJ to MKC coefficients
}

```

```

# Lag times based on routing coefficients
GAPT_to_SUX_Lag = len(coeffDict['GAPT']) - 1
SUX_to_OMA_Lag = len(coeffDict['SUX']) - 1
OMA_to_NCNE_Lag = len(coeffDict['OMA']) - 1
NCNE_to_RUNE_Lag = len(coeffDict['NCNE']) - 1
RUNE_to_STJ_Lag = len(coeffDict['RUNE']) - 1
STJ_to_MKC_Lag = len(coeffDict['STJ']) - 1

```

```

# Local flow forecasting regression coefficients. Taken from the DRM.
Forecast_Coeff_Dict = {
    'SUX' : [
        [1.00000, 0.00000], # Day 1 forecast coefficients
        [0.85055, 0.14945], # Day 2 forecast coefficients
        [0.84419, 0.15581], # Day 3 forecast coefficients
        [0.80906, 0.19094], # Day 4 forecast coefficients
        [0.85679, 0.14321], # Day 5 forecast coefficients
        [0.90832, 0.09168], # Day 6 forecast coefficients
        [0.90832, 0.09168], # Day 7 forecast coefficients
        [0.90832, 0.09168], # Day 8 forecast coefficients
    ]
}

```

```

[0.90832, 0.09168], # Day 9 forecast coefficients
[0.90832, 0.09168], # Day 10 forecast coefficients
],
'OMA' : [ [1.00000, 0.00000], # Day 1 forecast coefficients
[0.98034, 0.01966], # Day 2 forecast coefficients
[0.85435, 0.14565], # Day 3 forecast coefficients
[0.81078, 0.18922], # Day 4 forecast coefficients
[0.77796, 0.22204], # Day 5 forecast coefficients
[0.77233, 0.22767], # Day 6 forecast coefficients
[0.77233, 0.22767], # Day 7 forecast coefficients
[0.77233, 0.22767], # Day 8 forecast coefficients
[0.77233, 0.22767], # Day 9 forecast coefficients
[0.77233, 0.22767], # Day 10 forecast coefficients
],
'NCNE': [ [1.00000, 0.00000], # Day 1 forecast coefficients
[1.00000, 0.00000], # Day 2 forecast coefficients
[0.95657, 0.04343], # Day 3 forecast coefficients
[0.88933, 0.11067], # Day 4 forecast coefficients
[0.81157, 0.18843], # Day 5 forecast coefficients
[0.79474, 0.20526], # Day 6 forecast coefficients
[0.79474, 0.20526], # Day 7 forecast coefficients
[0.79474, 0.20526], # Day 8 forecast coefficients
[0.79474, 0.20526], # Day 9 forecast coefficients
[0.79474, 0.20526], # Day 10 forecast coefficients
],
'RUNE': [ [1.00000, 0.00000], # Day 1 forecast coefficients
[0.77435, 0.22565], # Day 2 forecast coefficients
[0.64459, 0.35541], # Day 3 forecast coefficients
[0.65309, 0.34691], # Day 4 forecast coefficients
[0.65455, 0.34545], # Day 5 forecast coefficients
[0.61762, 0.38238], # Day 6 forecast coefficients
[0.61762, 0.38238], # Day 7 forecast coefficients
[0.61762, 0.38238], # Day 8 forecast coefficients
[0.61762, 0.38238], # Day 9 forecast coefficients
[0.61762, 0.38238], # Day 10 forecast coefficients
],
'STJ' : [ [1.00000, 0.00000], # Day 1 forecast coefficients
[0.95747, 0.04253], # Day 2 forecast coefficients
[0.84365, 0.15635], # Day 3 forecast coefficients
[0.75376, 0.24624], # Day 4 forecast coefficients
[0.71377, 0.28623], # Day 5 forecast coefficients
[0.65505, 0.34495], # Day 6 forecast coefficients
[0.65505, 0.34495], # Day 7 forecast coefficients
[0.65505, 0.34495], # Day 8 forecast coefficients
[0.65505, 0.34495], # Day 9 forecast coefficients
[0.65505, 0.34495], # Day 10 forecast coefficients
],
'MKC' : [ [1.00000, 0.00000], # Day 1 forecast coefficients
[1.00000, 0.00000], # Day 2 forecast coefficients
[1.00000, 0.00000], # Day 3 forecast coefficients
[0.98830, 0.01170], # Day 4 forecast coefficients
[0.90689, 0.09311], # Day 5 forecast coefficients
[0.83491, 0.16509], # Day 6 forecast coefficients

```

```
[0.83491, 0.16509], # Day 7 forecast coefficients
[0.83491, 0.16509], # Day 8 forecast coefficients
[0.83491, 0.16509], # Day 9 forecast coefficients
[0.83491, 0.16509], # Day 10 forecast coefficients
],
```

```
}
```

```
# -----
# init info that is passed to runRuleScript()
# -----
```

```
initInfo = {
    'coeffDict'           : coeffDict,
    'constDict'          : constDict,
    'debug'              : debug,
    'floodLimits'        : floodLimits,
    'forecastLocal'      : forecastLocal,
    'Forecast_Coeff_Dict': Forecast_Coeff_Dict,
    'GAPT_to_SUX_Lag'    : GAPT_to_SUX_Lag,
    'julianDay'          : julianDay,
    'March_Pulse_Release': March_Pulse_Release,
    'March_Pulse_ROC_Initiate': March_Pulse_ROC_Initiate,
    'Max_May_Pulse_1P'   : Max_May_Pulse_1P,
    'Max_May_Pulse_2P'   : Max_May_Pulse_2P,
    'MAY_01_Lower_Quartile_Runoff': MAY_01_Lower_Quartile_Runoff,
    'MAY_01_Max_System_Storage_1P': MAY_01_Max_System_Storage_1P,
    'MAY_01_Upper_Quartile_Runoff': MAY_01_Upper_Quartile_Runoff,
    'May_Pulse_Release'  : May_Pulse_Release,
    'Min_May_Pulse_1P'   : Min_May_Pulse_1P,
    'Min_May_Pulse_2P'   : Min_May_Pulse_2P,
    'MKC_Local'          : MKC_Local,
    'NCNE_Local'         : NCNE_Local,
    'NCNE_to_RUNE_Lag'   : NCNE_to_RUNE_Lag,
    'OMA_Local'          : OMA_Local,
    'OMA_to_NCNE_Lag'    : OMA_to_NCNE_Lag,
    'outputDebug'        : outputDebug,
    'Pulse_Release_Data' : Pulse_Release_Data,
    'Pulse_Preclude'     : Pulse_Preclude,
    'retrieveModelVariables': retrieveModelVariables,
    'retrieveStateVariables': retrieveStateVariables,
    'routeFlow'          : routeFlow,
    'routeFlow_GAPT_to_MKC': routeFlow_GAPT_to_MKC,
    'RUNE_Local'         : RUNE_Local,
    'RUNE_to_STJ_Lag'    : RUNE_to_STJ_Lag,
    'SCSD_Flow'          : SCSD_Flow,
    'STJ_Local'          : STJ_Local,
    'STJ_to_MKC_Lag'     : STJ_to_MKC_Lag,
    'SUX_Local'          : SUX_Local,
    'SUX_to_OMA_Lag'     : SUX_to_OMA_Lag
}
```

```
currentRule.varPut('initInfo', initInfo)
```

```
# return Constants.TRUE if the initialization is successful
# and Constants.FALSE if it failed. Returning Constants.FALSE
# will halt the compute.
```

```
return Constants.TRUE
```

```
# runRuleScript() is the entry point that is called during the
# compute.
#
```

```
# currentRule is the rule that holds this script
# network is the ResSim network
# currentRuntimestep is the current Run Time Step
```

```
def runRuleScript(currentRule, network, currentRuntimestep):
```

```
    # create new Operation Value (OpValue) to return
    opValue = OpValue()
```

```
    # -----
```

```
    # Retrieve init info from initRuleScript() and set equal to variables
```

```
    # -----
```

```
    initInfo = currentRule.varGet('initInfo')
```

```
    coeffDict           = initInfo['coeffDict']
    constDict           = initInfo['constDict']
    debug               = initInfo['debug']
    floodLimits         = initInfo['floodLimits']
    forecastLocal       = initInfo['forecastLocal']
    Forecast_Coeff_Dict = initInfo['Forecast_Coeff_Dict']
    GAPT_to_SUX_Lag     = initInfo['GAPT_to_SUX_Lag']
    julianDay           = initInfo['julianDay']
    March_Pulse_Release = initInfo['March_Pulse_Release']
    March_Pulse_ROC_Initiate = initInfo['March_Pulse_ROC_Initiate']
    Max_May_Pulse_1P    = initInfo['Max_May_Pulse_1P']
    Max_May_Pulse_2P    = initInfo['Max_May_Pulse_2P']
    MAY_01_Lower_Quartile_Runoff = initInfo['MAY_01_Lower_Quartile_Runoff']
    MAY_01_Max_System_Storage_1P = initInfo['MAY_01_Max_System_Storage_1P']
    MAY_01_Upper_Quartile_Runoff = initInfo['MAY_01_Upper_Quartile_Runoff']
    May_Pulse_Release   = initInfo['May_Pulse_Release']
    Min_May_Pulse_1P    = initInfo['Min_May_Pulse_1P']
    Min_May_Pulse_2P    = initInfo['Min_May_Pulse_2P']
    MKC_Local           = initInfo['MKC_Local']
    NCNE_Local          = initInfo['NCNE_Local']
    NCNE_to_RUNE_Lag   = initInfo['NCNE_to_RUNE_Lag']
    OMA_Local           = initInfo['OMA_Local']
    OMA_to_NCNE_Lag     = initInfo['OMA_to_NCNE_Lag']
    outputDebug         = initInfo['outputDebug']
    Pulse_Release_Data  = initInfo['Pulse_Release_Data']
    Pulse_Preclude      = initInfo['Pulse_Preclude']
    retrieveModelVariables = initInfo['retrieveModelVariables']
    retrieveStateVariables = initInfo['retrieveStateVariables']
    routeFlow           = initInfo['routeFlow']
    routeFlow_GAPT_to_MKC = initInfo['routeFlow_GAPT_to_MKC']
```

```

RUNE_Local                = initInfo['RUNE_Local']
RUNE_to_STJ_Lag           = initInfo['RUNE_to_STJ_Lag']
SCSD_Flow                 = initInfo['SCSD_Flow']
STJ_Local                 = initInfo['STJ_Local']
STJ_to_MKC_Lag           = initInfo['STJ_to_MKC_Lag']
SUX_Local                 = initInfo['SUX_Local']
SUX_to_OMA_Lag           = initInfo['SUX_to_OMA_Lag']

# -----
# Input Data
# -----

# Simulation Time & Step Info
rtw = currentRuntimestep.getRuntimeWindow() # Run time window
startRtw = rtw.getStartTime() # Start time of simulation
lookbackRtw = rtw.getLookbackTime() # Start of lookback period
endRtw = rtw.getEndTime() # End time of simulation
totalSteps = rtw.getNumSteps() # Total number of steps in the simulation
curStep = currentRuntimestep.getStep() # Current step
mon = currentRuntimestep.getHecTime().month() # Current month
day = currentRuntimestep.getHecTime().day() # Current day
hour = currentRuntimestep.getHecTime().hour() # Hour of current time step
jDay = julianDay(mon, day) # Current julian day
numLookbackSteps = rtw.getNumLookbackSteps() # Number of lookback steps in simulation
simStartStep = numLookbackSteps + 1 # First step after the lookback period
PassCounter = network.getComputePassCounter() # Pass number of simulation
PassNumber = PassCounter + 1 # Pass number starts at 0 so add 1 to start it at 1
Route_Step = 10 # Step adjustment used in routing of upstream flows

if curStep == simStartStep:
    outputDebug(currentRuntimestep, debug, 'Lookback Start Time =', rtw.getLookbackTimeString(), '\tRun Time Window Start Time =', rtw.getStartTimeString(), \
        '\tRun Time Window End Time =', rtw.getEndTimeString(), '\tTotal Number of Steps =', totalSteps)

# -----
# Retrieve model variables needed for the script in a dictionary
# -----

Model_Variable = retrieveModelVariables(currentRuntimestep, totalSteps)
State_Variable = retrieveStateVariables(currentRuntimestep)

if curStep == simStartStep:
    currentRule.varPut('MAR_01_System_Storage', 56.0) #System_Current_Storage / 1e6
    currentRule.varPut('MAY_01_System_Storage', 56.0) #System_Current_Storage / 1e6
    currentRule.varPut('March_Pulse_Initiated', 'No') # Set March_Pulse_Initiated to "No" to allow for the spring pulse initiation to occur
    currentRule.varPut('Julian_Mar_Pulse_Initiation', julianDay('DEC_31')) # Initially set March pulse initiation date to Dec 31 so the pulse
                                                # logic won't be evaluated until the pulse has been initiated
    currentRule.varPut('Pulse_Cancelled', 'No') # Variable that records if March pulse has been cancelled
    currentRule.varPut('May_Pulse', 0) # Initially set May pulse magnitude to 0
    currentRule.varPut('May_Pulse_Release_Data', Model_Variable['System_Release_Data']) # Set the pulse release data = system release data. This will be overwritten in Pass 4

# -----
# Main Script
# -----

```

```

if PassNumber < 3:

    # set type and value for OpValue
    # type is one of:
    # OpRule.RULETYPE_MAX - maximum flow
    # OpRule.RULETYPE_MIN - minimum flow
    # OpRule.RULETYPE_SPEC - specified flow

    opValue.init(OpRule.RULETYPE_MIN, 0)
else:
    # -----
    # March Spring Pulse Logic
    #
    # Initialization begins once navigation release increase has peaked. GAPD releases are then increased by 5,000 cfs for the next 2 days.
    # Over the next 5 days, GAPD release is decreased by 1000 cfs per day.
    # -----

    # Replace values with actual releases for routing purposes
    Pulse_Release_Data[curStep - 1] = Model_Variable['System_Previous_Release']

    # -----
    # Check system storage on March 1
    # -----

    if jDay == julianDay('MAR_01'):
        currentRule.varPut('MAR_01_System_Storage', Model_Variable['System_Previous_Storage'] / 1e6)
        currentRule.varPut('March_Pulse_Initiated', 'No') # Set March_Pulse_Initiated to "No" to allow for the spring pulse initiation to occur
        currentRule.varPut('Julian_Mar_Pulse_Initiation', julianDay('MAY_01')) # Initially set March pulse initiation date to May 1 so the pulse
                                                                                                     # logic won't be evaluated until the pulse has been initiated

        currentRule.varPut('Pulse_Cancelled', 'No') # Variable that records if March pulse has been cancelled

        outputDebug(currentRuntimestep, debug, 'Mar 1 system storage = %.1f' % currentRule.varGet('MAR_01_System_Storage'))

    MAR_01_System_Storage = currentRule.varGet('MAR_01_System_Storage')

    if MAR_01_System_Storage < Pulse_Preclude and jDay == julianDay('MAR_01'):
        currentRule.varPut('Pulse_Cancelled', 'Yes')

        outputDebug(currentRuntimestep, debug, 'System storage is too low for March pulse.')

    # -----
    # Determine date of March spring pulse initiation
    # -----

    March_Pulse_Initiated = currentRule.varGet('March_Pulse_Initiated')
    Pulse_Cancelled = currentRule.varGet('Pulse_Cancelled')

    if julianDay('MAR_23') < jDay < julianDay('MAY_01') and March_Pulse_Initiated == 'No' and Pulse_Cancelled == 'No':
        Release_Difference = Model_Variable['System_Previous_Release'] - Model_Variable['System_Laggedx2_Release']

        if Release_Difference >= March_Pulse_ROC_Initiate:
            March_Pulse_Initiated = 'No'

```

```

else:
    March_Pulse_Initiated = 'Yes'

currentRule.varPut('March_Pulse_Initiated', March_Pulse_Initiated)

outputDebug(currentRuntimestep, debug, 'System Previous Release = %.0f' % Model_Variable['System_Previous_Release'],
            '\tSystem Lagged x2 Release = %.0f' % Model_Variable['System_Laggedx2_Release'])

# Save the pulse initiation day and previous system release to memory
if March_Pulse_Initiated == 'Yes':
    currentRule.varPut('Initial_Release', Model_Variable['System_Previous_Release'])
    currentRule.varPut('Julian_Mar_Pulse_Initiation', jDay)

    outputDebug(currentRuntimestep, debug, 'March spring pulse has been initiated.')

Julian_Mar_Pulse_Initiation = currentRule.varGet('Julian_Mar_Pulse_Initiation')

# -----
# Set releases for March pulse
# -----

if jDay == Julian_Mar_Pulse_Initiation and Pulse_Cancelled == 'No':
    Initial_Release = currentRule.varGet('Initial_Release')
    Max_March_Pulse = 5000
    March_Pulse = Max_March_Pulse - max(SCSD_Flow[(curStep-1):(curStep + 1)]) # Account for James River at Scotland, SD flow. Flow is lagged by 1 day

    outputDebug(currentRuntimestep, debug, 'James River Flow = %.0f cfs' % max(SCSD_Flow[(curStep-1):(curStep + 1)]), '\tMax Pulse = %.0f cfs' % 5000,
                '\tMarch Pulse = %.0f cfs' % March_Pulse)

# Check spring pulse limit
if (Initial_Release + March_Pulse) > 35000:
    March_Pulse = 35000 - Initial_Release

    if March_Pulse <= 0:
        currentRule.varPut('Pulse_Cancelled', 'Yes')
        currentRule.varPut('March_Pulse_Initiated', 'No')
        #currentRule.varPut('Pulse_Release_Data', Model_Variable['System_Release_Data'][:])

        outputDebug(currentRuntimestep, debug, 'March pulse has been cancelled. Navigation releases >= 35000')

if March_Pulse > 0:
    # Initially set March pulse to max pulse
    March_Pulse_Release(curStep, Pulse_Release_Data, Initial_Release, Model_Variable['System_Previous_Release'], March_Pulse)

    # Check flood constraints for the max pulse
    Constraints_Exceeded = floodLimits(currentRuntimestep, curStep, Route_Step, Pulse_Release_Data, March_Pulse)

    outputDebug(currentRuntimestep, debug, 'Does a pulse of %.0f cfs cause flood limits to be exceeded?' % March_Pulse, Constraints_Exceeded)

    # If the max pulse causes flooding, reduce the pulse by 500 cfs until flooding doesn't occur
    while Constraints_Exceeded == 'Yes' and March_Pulse > 0:
        # Reset Pulse_Release_Data to System_Release_Data
        for step in range(27):

```

```

        Pulse_Release_Data[curStep + step] = 0

# Reduce pulse by 500 cfs
March_Pulse = March_Pulse - 500

# Set March pulse releases equal to lowered pulse
March_Pulse_Release(curStep, Pulse_Release_Data, Initial_Release, Model_Variable['System_Previous_Release'], March_Pulse)

# Check flood constraints
Constraints_Exceeded = floodLimits(currentRuntimestep, curStep, Route_Step, Pulse_Release_Data, March_Pulse)

outputDebug(currentRuntimestep, debug, 'Does a pulse of %.0f cfs cause flood limits to be exceeded?' % March_Pulse, Constraints_Exceeded)

if 0 < March_Pulse <= 5000:

    outputDebug(currentRuntimestep, debug, 'March pulse = %.0f cfs.' % March_Pulse, ' System pulse release = %.0f cfs.' % Pulse_Release_Data[curStep])

    currentRule.varPut('Pulse_Release_Data', Pulse_Release_Data)
elif March_Pulse <= 0:

    outputDebug(currentRuntimestep, debug, 'March pulse has been cancelled because of downstream flood constraints.')

```

```

else:
    outputDebug(currentRuntimestep, debug, 'May 1 system storage = %.1f' % MAY_01_System_Storage, '\tMay 1 - Jul 31 Forecasted System Runoff = %.2f' % MAY_01_Forecast_System_Runoff)

# Retrieve variables for May pulse from memory
Pulse_Cancelled = currentRule.varGet('Pulse_Cancelled')
MAY_01_System_Storage = currentRule.varGet('MAY_01_System_Storage')
MAY_01_Forecast_System_Runoff = currentRule.varGet('MAY_01_Forecast_System_Runoff')

if jDay == julianDay('MAY_02') and Pulse_Cancelled == 'No':

    # -----
    # Calculate pulse magnitude based on 2 step proration
    # -----

    # First proration has a maximum pulse magnitude of 16000 when system storage is 54.5 MAF and a minimum pulse magnitude when system storage
    # is 40.0 MAF. The maximum pulse magnitude cannot exceed 16000
    firstProratedSlope = (Max_May_Pulse_1P - Min_May_Pulse_1P)/(MAY_01_Max_System_Storage_1P - Pulse_Preclude)
    First_Prорated_May_Pulse = (firstProratedSlope * (MAY_01_System_Storage - Pulse_Preclude)) + Min_May_Pulse_1P
    if First_Prорated_May_Pulse > Max_May_Pulse_1P:
        First_Prорated_May_Pulse = Max_May_Pulse_1P

    # Second proration has a maximum additional pulse magnitude of 4000 when the May 1 forecasted system runoff is an upper quartile runoff
    # and a minimum additional pulse magnitude of -3000 when the May 1 forecasted system runoff is a lower quartile runoff.
    secondProratedSlope = (Max_May_Pulse_2P - Min_May_Pulse_2P)/(MAY_01_Upper_Quartile_Runoff - MAY_01_Lower_Quartile_Runoff)
    Second_Prорated_Adjustment = secondProratedSlope * (MAY_01_Forecast_System_Runoff - MAY_01_Lower_Quartile_Runoff) + Min_May_Pulse_2P
    if Second_Prорated_Adjustment > Max_May_Pulse_2P:
        Second_Prорated_Adjustment = Max_May_Pulse_2P
    elif Second_Prорated_Adjustment < Min_May_Pulse_2P:
        Second_Prорated_Adjustment = Min_May_Pulse_2P
    Prorated_May_Pulse = First_Prорated_May_Pulse + Second_Prорated_Adjustment

    outputDebug(currentRuntimestep, debug, 'First_Prорated_May_Pulse = %.0f' % First_Prорated_May_Pulse, '\tSecond_Prорated_Adjustment = %.0f' % Second_Prорated_Adjustment,
        '\tProrated_May_Pulse = %.0f' % Prorated_May_Pulse, '\n\t\t\tMAY_01_Forecast_System_Runoff = %.4f' % MAY_01_Forecast_System_Runoff)

    May_Pulse = Prorated_May_Pulse - max(SCSD_Flow[(curStep + 1):(curStep + 3)]) # Account for James River at Scotland, SD flow. Flow is lagged by 1 day

    outputDebug(currentRuntimestep, debug, 'James River Flow = %.0f cfs' % max(SCSD_Flow[(curStep + 1):(curStep + 3)]), '\tMax May Pulse = %.0f cfs' % Prorated_May_Pulse,
        '\tMay Pulse = %.0f cfs' % May_Pulse)

    # -----
    # Increase release for May pulse
    # -----

    # Set May pulse release pattern to max pulse
    May_Pulse_Release(curStep, Pulse_Release_Data, Model_Variable['System_Previous_Release'], May_Pulse)

    # Check flood constraints for the max pulse
    Constraints_Exceeded = floodLimits(currentRuntimestep, curStep, Route_Step, Pulse_Release_Data, May_Pulse)

    outputDebug(currentRuntimestep, debug, 'Does a pulse of %.0f cfs cause flood limits to be exceeded?' % May_Pulse, Constraints_Exceeded)

    # If the max pulse causes flooding, reduce the pulse by 500 cfs until flooding doesn't occur
    while Constraints_Exceeded == 'Yes' and May_Pulse > 0:

```

```

# Reset Pulse_Release_Data to System_Release_Data
for step in range(27):
    Pulse_Release_Data[curStep + step] = 0

# Reduce pulse by 500 cfs
May_Pulse = May_Pulse - 500

# Set May pulse releases equal to lowered pulse
May_Pulse_Release(curStep, Pulse_Release_Data, Model_Variable['System_Previous_Release'], May_Pulse)

# Check flood constraints
Constraints_Exceeded = floodLimits(currentRuntimestep, curStep, Route_Step, Pulse_Release_Data, May_Pulse)

outputDebug(currentRuntimestep, debug, 'Does a pulse of %.0f cfs cause flood limits to be exceeded?' % May_Pulse, Constraints_Exceeded)

if 0 < May_Pulse:

    outputDebug(currentRuntimestep, debug, 'May pulse = %.0f cfs.' % May_Pulse, ' System pulse release = %.0f cfs.' % Pulse_Release_Data[curStep + 2])

    currentRule.varPut('Pulse_Release_Data', Pulse_Release_Data)
elif May_Pulse <= 0:

    outputDebug(currentRuntimestep, debug, 'May pulse has been cancelled because of downstream flood constraints.')

    currentRule.varPut('Pulse_Cancelled', 'Yes')
    # Reset Pulse_Release_Data to System_Release_Data
    for step in range(27):
        Pulse_Release_Data[curStep + step] = 0

# set type and value for OpValue
# type is one of:
# OpRule.RULETYPE_MAX - maximum flow
# OpRule.RULETYPE_MIN - minimum flow
# OpRule.RULETYPE_SPEC - specified flow

opValue.init(OpRule.RULETYPE_MIN, Pulse_Release_Data[curStep])

# return the Operation Value.
# return "None" to have no effect on the compute
return opValue

```

## 14 APPENDIX H – FLOOD EVACUATION SCRIPTED RULE

```

'''
Author: Alex Flanigan
Date: 06-10-2015
Description: Logic for setting GAPT releases when water needs to be evacuated from the reservoirs
'''
# -----
# Required imports to create the OpValue return object.
# -----
from hec.rss.model      import OpRule
from hec.script        import Constants
from hec.heclib.util   import *
from hec.rss.model     import *
from hec.heclib.dss    import *
from hec.hecmath       import *
from hec.model         import *
from hec.script        import *
from hec.io            import TimeSeriesContainer
from hec.client        import ClientApp
from java.lang         import String
from zipfile           import ZipFile
import os, sys
import traceback
import copy

# -----
# Global variables
# -----

global SUX_Local

#
# initialization function. optional.
#
# set up tables and other things that only need to be performed once during
# the compute.
#
# currentRule is the rule that holds this script
# network is the ResSim network
#
#
def initRuleScript(currentRule, network):
    # -----
    # Functions
    # -----

    # Debugging Function: Set debug = True to print all debug statements and = False to turn them off
    # Initialization Debugging Function: Set debug = True to print all debug statements and = False to turn them off
    debug = False
    def initOutputDebug(debug, arg1="", arg2="", arg3="", arg4="", arg5="", arg6="", arg7="", arg8="", arg9="", arg10="", arg11="", arg12="", arg13="", arg14=""):
        if debug == True:
            print 'Initialization Debug  \t', arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9, arg10, arg11, arg12, arg13, arg14

    # Debugging Function: Set debug = True to print all debug statements and = False to turn them off

```

```

def outputDebug(currentRuntimestep, debug, arg1=",", arg2=",", arg3=",", arg4=",", arg5=",", arg6=",", arg7=",", arg8=",", arg9=",", arg10=",", arg11=",", arg12=",", arg13=",", arg14=",", arg15=""):
    PassCounter = network.getComputePassCounter() # Pass number of simulation
    PassNumber = PassCounter + 1 # Pass number starts at 0 so add 1 to start it at 1
    if debug == True:
        print 'Pass Number =', PassNumber, ' ', currentRuntimestep.dateTimeString(), \
            '\t', arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9, arg10, arg11, arg12, arg13, arg14, arg15, '\n'

# floodEvac Function: Minimum GAPT releases used to shape the evacuation of flood storage
def floodEvac(currentRuntimestep, # currentRuntimestep variable passed in through the runRuleScript()
    curStep, # Current step of the simulation
    SYS_Stor, # Current system storage
    SYS_Remaining_Runoff, # Remaining forecasted runoff above GAPT
    SYS_Remaining_Release, # Remaining forecasted GAPT releases
    Winter_Release, # Winter release established on SEP 01
    SYS_Carryover_and_Multiple_Use_Stor, # Storage at the top of the Carryover and Multiple Use Zone
    NavigationEndDate, # Julian date of the end of the navigation season
    Service_Level, # System service level
    jDay, # Current julian day
    mon, # Current month
    year # Current year
):
    SYS_MAR_01_Stor = SYS_Stor + SYS_Remaining_Runoff - SYS_Remaining_Release

outputDebug(currentRuntimestep, debug, 'Remaining System Runoff = %.0f acre-ft' % SYS_Remaining_Runoff,
    '\tRemaining System Release = %.0f acre-ft' % SYS_Remaining_Release, '\tForecasted Mar 1 Storage = %.0f acre-ft' % SYS_MAR_01_Stor)

if mon < 4:
    gaptRelease_Min = 0
    # 2011 Special Circumstances
    if year == 2011 and Service_Level >= 35.0:
        gaptRelease_Min = 25000
elif mon == 4:
    gaptRelease_Min = 0
    # 1997 Special Circumstances
    if year == 1997 and Service_Level >= 50.0 and jDay < julianDay('APR_15'):
        gaptRelease_Min = 40000
    elif year == 1997 and Service_Level >= 50.0 and jDay >= julianDay('APR_15'):
        gaptRelease_Min = 55000
    # 2011 Special Circumstances
    if year == 2011 and Service_Level >= 35.0:
        gaptRelease_Min = 35000
elif mon == 5:
    if 68000000 <= SYS_MAR_01_Stor < 70000000:
        gaptRelease_Min = 45000
    elif 70000000 <= SYS_MAR_01_Stor:
        gaptRelease_Min = 60000
    else:
        gaptRelease_Min = 0
    # 1997 Special Circumstances
    if year == 1997 and Service_Level >= 50.0:
        gaptRelease_Min = 60000
    # 2011 Special Circumstances
    if SYS_MAR_01_Stor > 68000000 and year == 2011 and jDay >= julianDay('MAY_15'):

```

```

        gaptRelease_Min = 60000
elif mon == 6:
    if SYS_MAR_01_Stor < 64600000:
        gaptRelease_Min = 25000
    elif 64600000 <= SYS_MAR_01_Stor < 66600000:
        gaptRelease_Min = 30000
    elif 66600000 <= SYS_MAR_01_Stor < 67600000:
        gaptRelease_Min = 35000
    else:
        gaptRelease_Min = 0
    # 1997 Special Circumstances
    if year == 1997:
        gaptRelease_Min = 55000
    # 2011 Special Circumstances
    if SYS_MAR_01_Stor > 74000000 and year == 2011:
        gaptRelease_Min = 160000
elif mon == 7:
    if SYS_MAR_01_Stor < 64600000:
        gaptRelease_Min = 25000
    elif 64600000 <= SYS_MAR_01_Stor < 66000000:
        gaptRelease_Min = 40000
    elif 66000000 <= SYS_MAR_01_Stor < 68000000:
        gaptRelease_Min = 45000
    elif 68000000 <= SYS_MAR_01_Stor:
        gaptRelease_Min = 55000
    else:
        gaptRelease_Min = 0
    # 1997 Special Circumstances
    if year == 1997:
        gaptRelease_Min = 55000
    # 2011 Special Circumstances
    if SYS_MAR_01_Stor > 71000000 and year == 2011:
        gaptRelease_Min = 160000
elif mon == 8 and jDay < julianDay('AUG_15'):
    if 56600000 <= SYS_MAR_01_Stor < 58600000:
        gaptRelease_Min = 20000
    elif 58600000 <= SYS_MAR_01_Stor < 59600000:
        gaptRelease_Min = 42000
    elif 59600000 <= SYS_MAR_01_Stor < 60600000:
        gaptRelease_Min = 45000
    elif 60600000 <= SYS_MAR_01_Stor < 61600000:
        gaptRelease_Min = 47000
    elif 61600000 <= SYS_MAR_01_Stor < 62600000:
        gaptRelease_Min = 49000
    elif 62600000 <= SYS_MAR_01_Stor < 64600000:
        gaptRelease_Min = 52000
    elif 64600000 <= SYS_MAR_01_Stor:
        gaptRelease_Min = 62000
    else:
        gaptRelease_Min = 0
    # 2011 Special Circumstances
    if SYS_MAR_01_Stor > 63000000 and year == 2011:
        gaptRelease_Min = 150000

```

```

elif jDay >= julianDay('AUG_15'):
    # Change step to select Winter Release after Sep 01 assessment
    if julianDay('SEP_01') <= jDay:
        Winter_Release_Step = curStep + (julianDay('DEC_31') - jDay)
        currentRuntimestep.setStep(Winter_Release_Step)
        Winter_Release = network.getStateVariable("GAPTWinterRelease").getValue(currentRuntimestep)
        currentRuntimestep.setStep(curStep)

    if Winter_Release <= 0:
        Winter_Release = 17000
    Winter_Release_Volume = Winter_Release * ((julianDay('DEC_31') - NavigationEndDate) + (julianDay('MAR_01') - julianDay('JAN_01'))) * initInfo['CFS_to_ACRE_FT']
    Fall_Release_Volume = SYS_Stor + SYS_Remaining_Runoff - Winter_Release_Volume - SYS_Carryover_and_Multiple_Use_Stor
    outputDebug(currentRuntimestep, debug, 'Winter Release = %.0f' % Winter_Release, '\tWinter Release Volume = %.0f acre-ft' % Winter_Release_Volume,
        '\n\t\t\t\tCurrent System Storage = %.0f acre-ft' % SYS_Stor, '\tRemaining Runoff = %.0f acre-ft' % SYS_Remaining_Runoff,
        '\tCarryover Zone Storage = %.0f acre-ft' % SYS_Carryover_and_Multiple_Use_Stor)
    if jDay >= NavigationEndDate:
        gaptRelease_Min = 0
    else:
        gaptRelease_Min = (Fall_Release_Volume / (NavigationEndDate - jDay)) * initInfo['ACRE_FT_to_CFS']
    outputDebug(currentRuntimestep, debug, 'Minimum GAPT Flood Evac Release = %.0f cfs' % gaptRelease_Min)
    currentRule.varPut('Prev_gaptRelease_Min', gaptRelease_Min)
    return gaptRelease_Min

```

# janJulSysRunoff Function: Calculate observed Jan-Jul accumulated runoff

```

def janJulSysRunoff( currentRuntimestep, # currentRuntimestep variable passed in through the runRuleScript()
    RBMT_Inc_Inflow, # RBMT incremental inflow
    FTPK_Inc_Inflow, # FTPK incremental inflow
    WPMT_Inc_Inflow, # WPMT incremental inflow
    CLMT_Inc_Inflow, # CLMT incremental inflow
    GARR_Inc_Inflow, # GARR incremental inflow
    BIS_Inc_Inflow, # BIS incremental inflow
    OAHE_Inc_Inflow, # OAHE incremental inflow
    BEND_Inc_Inflow, # BEND incremental inflow
    FTRA_Inc_Inflow, # FTRA incremental inflow
    GAPT_Inc_Inflow # GAPT incremental inflow
):
    rtw = currentRuntimestep.getRunTimeWindow() # Run time window
    curStep = currentRuntimestep.getStep() # Current step
    year = currentRuntimestep.getHecTime().year() # Current year
    mon = currentRuntimestep.getHecTime().month() # Current month
    day = currentRuntimestep.getHecTime().day() # Current day
    numLookbackSteps = rtw.getNumLookbackSteps() # Number of lookback steps in simulation
    simStartStep = numLookbackSteps + 1 # First step after the lookback period

    # Calculate Jan-Jul System Runoff to Adjust Normal Forecasted Runoff
    if curStep == simStartStep:
        currentRule.varPut('Start_Year', year)
        currentRule.varPut('System_Accum_Runoff', [])
        currentRule.varPut('Runoff_Ratio', 2.0)
    elif mon == 1 and day == 1:
        currentRule.varPut('System_Accum_Runoff', [])
        currentRule.varPut('Runoff_Ratio', 2.0)

```

```

if mon <= 7 and curStep > simStartStep:
    System_Accum_Runoff = currentRule.varGet('System_Accum_Runoff')
    runoff = (RBMT_Inc_Inflow + FTPK_Inc_Inflow + WPMT_Inc_Inflow + CLMT_Inc_Inflow + GARR_Inc_Inflow + BIS_Inc_Inflow + OAHE_Inc_Inflow + \
        BEND_Inc_Inflow + FTRA_Inc_Inflow + GAPT_Inc_Inflow) * initInfo['CFS_to_ACRE_FT']
    System_Accum_Runoff.append(runoff)
    currentRule.varPut('System_Accum_Runoff', System_Accum_Runoff)
elif mon == 8 and day == 1:
    System_Accum_Runoff = currentRule.varGet('System_Accum_Runoff')
    runoff = (RBMT_Inc_Inflow + FTPK_Inc_Inflow + WPMT_Inc_Inflow + CLMT_Inc_Inflow + GARR_Inc_Inflow + BIS_Inc_Inflow + OAHE_Inc_Inflow + \
        BEND_Inc_Inflow + FTRA_Inc_Inflow + GAPT_Inc_Inflow) * initInfo['CFS_to_ACRE_FT']
    System_Accum_Runoff.append(runoff)
    System_Accum_Runoff = sum(System_Accum_Runoff)
    # Normal runoff for Jan-Jul based on 2014 statistics
    Normal_System_Runoff = 18017000 # acre-ft
    Runoff_Ratio = System_Accum_Runoff / Normal_System_Runoff
    currentRule.varPut('Runoff_Ratio', Runoff_Ratio)

```

# julianDay Function: Get the julian day for a month and day or a string, or a month and day for a julian day

```

def julianDay(*args) :
    #   J F M A M J J A S O N D
    daysPrev = 0, 31, 59, 90,120,151,181,212,243,273,304,334
    argCount = len(args)
    if argCount == 1 :
        if type(args[0]) == type(0) :
            jday = args[0]
            if not 1 <= jday <= 365 :
                raise ValueError("Julian day out of range 1..365")
            for i in range(12)[::-1] :
                if jday > daysPrev[i] :
                    mon = i + 1
                    break
            day = jday - daysPrev[mon-1]
            return mon, day
        else :
            mon, day = args[0].split("_")
            mon = ["JAN", "FEB", "MAR", "APR", "MAY", "JUN", "JUL", "AUG", "SEP", "OCT", "NOV", "DEC"].index(mon.upper()) + 1
            day = int(day);
            return julianDay(mon, day)
    elif argCount == 2 :
        mon, day = args
        if not 1 <= mon <= 12 :
            raise ValueError("Month is out of range 1..12")
        if not 1 <= day <= 31 :
            raise ValueError("Day is out of range 1..31")
        if mon == 2 and day == 29 : day = 28
        return daysPrev[mon-1] + day
    else :
        raise ValueError("Expected 1 or 2 arguments, got %d" % argCount)

```

# monthlyAccumRunoff Function: Track monthly runoff for each project

```

def monthlyAccumRunoff(    currentRuntimestep, # currentRuntimestep variable passed in through the runRuleScript()
                        RBMT_Inc_Inflow,    # RBMT incremental inflow
                        FTPK_Inc_Inflow,    # FTPK incremental inflow

```

```

        WPMT_Inc_Inflow, # WPMT incremental inflow
        CLMT_Inc_Inflow, # CLMT incremental inflow
        GARR_Inc_Inflow, # GARR incremental inflow
        BIS_Inc_Inflow,  # BIS incremental inflow
        OAHE_Inc_Inflow, # OAHE incremental inflow
        BEND_Inc_Inflow, # BEND incremental inflow
        FTRA_Inc_Inflow, # FTRA incremental inflow
        GAPT_Inc_Inflow, # GAPT incremental inflow
        System_Release_Data # System release data
    ):
    rtw = currentRuntimestep.getRunTimeWindow() # Run time window
    curStep = currentRuntimestep.getStep() # Current step
    year = currentRuntimestep.getHecTime().year() # Current year
    mon = currentRuntimestep.getHecTime().month() # Current month
    day = currentRuntimestep.getHecTime().day() # Current day
    numLookbackSteps = rtw.getNumLookbackSteps() # Number of lookback steps in simulation
    simStartStep = numLookbackSteps + 1 # First step after the lookback period

    if curStep == simStartStep or day == 1:
        currentRule.varPut('FTPK_Accum_Stor', [])
        currentRule.varPut('GARR_Accum_Stor', [])
        currentRule.varPut('OAHE_Accum_Stor', [])
        currentRule.varPut('BEND_and_FTRA_Accum_Stor', [])
        currentRule.varPut('GAPT_Accum_Stor', [])
        currentRule.varPut('SYS_Accum_Release', [])

        FTPK_Accum_Stor = currentRule.varGet('FTPK_Accum_Stor')
        GARR_Accum_Stor = currentRule.varGet('GARR_Accum_Stor')
        OAHE_Accum_Stor = currentRule.varGet('OAHE_Accum_Stor')
        BEND_and_FTRA_Accum_Stor = currentRule.varGet('BEND_and_FTRA_Accum_Stor')
        GAPT_Accum_Stor = currentRule.varGet('GAPT_Accum_Stor')
        SYS_Accum_Release = currentRule.varGet('SYS_Accum_Release')
    else:
        FTPK_Accum_Stor = currentRule.varGet('FTPK_Accum_Stor')
        GARR_Accum_Stor = currentRule.varGet('GARR_Accum_Stor')
        OAHE_Accum_Stor = currentRule.varGet('OAHE_Accum_Stor')
        BEND_and_FTRA_Accum_Stor = currentRule.varGet('BEND_and_FTRA_Accum_Stor')
        GAPT_Accum_Stor = currentRule.varGet('GAPT_Accum_Stor')
        SYS_Accum_Release = currentRule.varGet('SYS_Accum_Release')

        FTPK_Accum_Stor.append((FTPK_Inc_Inflow + RBMT_Inc_Inflow) * initInfo['CFS_to_ACRE_FT'])
        GARR_Accum_Stor.append((WPMT_Inc_Inflow + CLMT_Inc_Inflow + GARR_Inc_Inflow) * initInfo['CFS_to_ACRE_FT'])
        OAHE_Accum_Stor.append(OAHE_Inc_Inflow * initInfo['CFS_to_ACRE_FT'])
        BEND_and_FTRA_Accum_Stor.append((BEND_Inc_Inflow + FTRA_Inc_Inflow) * initInfo['CFS_to_ACRE_FT'])
        GAPT_Accum_Stor.append(GAPT_Inc_Inflow * initInfo['CFS_to_ACRE_FT'])
        SYS_Accum_Release.append(System_Release_Data * CFS_to_ACRE_FT)
    return FTPK_Accum_Stor, GARR_Accum_Stor, OAHE_Accum_Stor, BEND_and_FTRA_Accum_Stor, GAPT_Accum_Stor, SYS_Accum_Release

# remainingAnnualRunoff Function: Calculate remaining annual incremental runoff
def remainingAnnualRunoff( currentRuntimestep, # currentRuntimestep variable passed in through the runRuleScript()
    FTPK_Accum_Stor, # FTPK monthly accumulated runoff
    GARR_Accum_Stor, # GARR monthly accumulated runoff
    OAHE_Accum_Stor, # OAHE monthly accumulated runoff

```

```

        BEND_and_FTRA_Accum_Stor, # BEND and FTRA monthly accumulated runoff
        GAPT_Accum_Stor, # GAPT monthly accumulated runoff
        SYS_Accum_Release # System monthly accumulated release
    ):
    rtw = currentRuntimestep.getRunTimeWindow() # Run time window
    curStep = currentRuntimestep.getStep() # Current step
    year = currentRuntimestep.getHecTime().year() # Current year
    mon = currentRuntimestep.getHecTime().month() # Current month
    day = currentRuntimestep.getHecTime().day() # Current day
    numLookbackSteps = rtw.getNumLookbackSteps() # Number of lookback steps in simulation
    simStartStep = numLookbackSteps + 1 # First step after the lookback period

# Calculate Remaining Annual Incremental Runoff
Runoff_Ratio = currentRule.varGet('Runoff_Ratio')
Start_Year = currentRule.varGet('Start_Year')
monString = monDict[mon][0]
FTPK_Observed_Runoff = sum(FTPK_Accum_Stor)
GARR_Observed_Runoff = sum(GARR_Accum_Stor)
OAHE_Observed_Runoff = sum(OAHE_Accum_Stor)
BEND_and_FTRA_Observed_Runoff = sum(BEND_and_FTRA_Accum_Stor)
GAPT_Observed_Runoff = sum(GAPT_Accum_Stor)
SYS_Observed_Release = sum(SYS_Accum_Release)

# If Jan-Jul accumulated runoff is less than the normal runoff for the period, decrease the Aug-Dec normal runoff values to reflect a drier year
# Don't factor the first year because simulation starts in March
if 8 <= mon <= 12 and Runoff_Ratio < 1.0 and year > Start_Year:
    FTPK_Next_Year_Jan_Feb_Fcst = fcstFlows[year + 1]['FTPK']['Jan'][0] + fcstFlows[year + 1]['FTPK']['Jan'][1]
    GARR_Next_Year_Jan_Feb_Fcst = fcstFlows[year + 1]['GARR']['Jan'][0] + fcstFlows[year + 1]['GARR']['Jan'][1]
    OAHE_Next_Year_Jan_Feb_Fcst = fcstFlows[year + 1]['OAHE']['Jan'][0] + fcstFlows[year + 1]['OAHE']['Jan'][1]
    BEND_and_FTRA_Next_Year_Jan_Feb_Fcst = fcstFlows[year + 1]['FTRA']['Jan'][0] + fcstFlows[year + 1]['FTRA']['Jan'][1]
    GAPT_Next_Year_Jan_Feb_Fcst = fcstFlows[year + 1]['GAPT']['Jan'][0] + fcstFlows[year + 1]['GAPT']['Jan'][1]

    FTPK_Normal_Runoff_Fcst = EOY_Runoff['FTPK'][year][monString] - FTPK_Next_Year_Jan_Feb_Fcst
    GARR_Normal_Runoff_Fcst = EOY_Runoff['GARR'][year][monString] - GARR_Next_Year_Jan_Feb_Fcst
    OAHE_Normal_Runoff_Fcst = EOY_Runoff['OAHE'][year][monString] - OAHE_Next_Year_Jan_Feb_Fcst
    BEND_and_FTRA_Normal_Runoff_Fcst = EOY_Runoff['FTRA'][year][monString] - BEND_and_FTRA_Next_Year_Jan_Feb_Fcst
    GAPT_Normal_Runoff_Fcst = EOY_Runoff['GAPT'][year][monString] - GAPT_Next_Year_Jan_Feb_Fcst

    FTPK_Forecasted_Runoff = (FTPK_Next_Year_Jan_Feb_Fcst + (FTPK_Normal_Runoff_Fcst * Runoff_Ratio)) * 1000
    GARR_Forecasted_Runoff = (GARR_Next_Year_Jan_Feb_Fcst + (GARR_Normal_Runoff_Fcst * Runoff_Ratio)) * 1000
    OAHE_Forecasted_Runoff = (OAHE_Next_Year_Jan_Feb_Fcst + (OAHE_Normal_Runoff_Fcst * Runoff_Ratio)) * 1000
    BEND_and_FTRA_Forecasted_Runoff = (BEND_and_FTRA_Next_Year_Jan_Feb_Fcst + (BEND_and_FTRA_Normal_Runoff_Fcst * Runoff_Ratio)) * 1000
    GAPT_Forecasted_Runoff = (GAPT_Next_Year_Jan_Feb_Fcst + (GAPT_Normal_Runoff_Fcst * Runoff_Ratio)) * 1000
    SYS_Forecasted_Release = EOYnormalReleaseAF['GAPT'][monString] * 1000 # Don't factor system release
else:
    FTPK_Forecasted_Runoff = EOY_Runoff['FTPK'][year][monString] * 1000
    GARR_Forecasted_Runoff = EOY_Runoff['GARR'][year][monString] * 1000
    OAHE_Forecasted_Runoff = EOY_Runoff['OAHE'][year][monString] * 1000
    BEND_and_FTRA_Forecasted_Runoff = EOY_Runoff['FTRA'][year][monString] * 1000
    GAPT_Forecasted_Runoff = EOY_Runoff['GAPT'][year][monString] * 1000
    SYS_Forecasted_Release = EOYnormalReleaseAF['GAPT'][monString] * 1000

FTPK_Remaining_Runoff = FTPK_Forecasted_Runoff - FTPK_Observed_Runoff

```

```

GARR_Remaining_Runoff = GARR_Forecasted_Runoff - GARR_Observed_Runoff
OAHE_Remaining_Runoff = OAHE_Forecasted_Runoff - OAHE_Observed_Runoff
BEND_and_FTRA_Remaining_Runoff = BEND_and_FTRA_Forecasted_Runoff - BEND_and_FTRA_Observed_Runoff
GAPT_Remaining_Runoff = GAPT_Forecasted_Runoff - GAPT_Observed_Runoff
SYS_Remaining_Runoff = FTPK_Remaining_Runoff + GARR_Remaining_Runoff + OAHE_Remaining_Runoff + BEND_and_FTRA_Remaining_Runoff + GAPT_Remaining_Runoff
SYS_Remaining_Release = SYS_Forecasted_Release - SYS_Observed_Release
return SYS_Remaining_Runoff, SYS_Remaining_Release

```

```

# retrieveModelVariables Function: Retrieve model variables needed for the script
def retrieveModelVariables(currentRuntimestep):

```

```

    Model_Variable = {
        # RBMT Model Variables
        'RBMT_Previous_Local_Inflow' : network.findJunction("System Inflow").getLocalFlowTimeSeries("RBMT_Flow").getPreviousValue(currentRuntimestep),
        # FTPK Model Variables
        'FTPK_Previous_Local_Inflow' : network.findJunction("System Inflow").getLocalFlowTimeSeries("FTPK_Local").getPreviousValue(currentRuntimestep),
        # WPMT Model Variables
        'WPMT_Previous_Local_Inflow' : network.findJunction("System Inflow").getLocalFlowTimeSeries("WPMT_Local").getPreviousValue(currentRuntimestep),
        # CLMT Model Variables
        'CLMT_Previous_Local_Inflow' : network.findJunction("System Inflow").getLocalFlowTimeSeries("CLMT_Local").getPreviousValue(currentRuntimestep),
        # GARR Model Variables
        'GARR_Previous_Local_Inflow' : network.findJunction("System Inflow").getLocalFlowTimeSeries("GARR_Local").getPreviousValue(currentRuntimestep),
        # BIS Model Variables
        'BIS_Previous_Local_Inflow' : network.findJunction("System Inflow").getLocalFlowTimeSeries("BIS_Local").getPreviousValue(currentRuntimestep),
        # OAHE Model Variables
        'OAHE_Previous_Local_Inflow' : network.findJunction("System Inflow").getLocalFlowTimeSeries("OAHE_Local").getPreviousValue(currentRuntimestep),
        # BEND Model Variables
        'BEND_Previous_Local_Inflow' : network.findJunction("System Inflow").getLocalFlowTimeSeries("BEND_Local").getPreviousValue(currentRuntimestep),
        # FTRA Model Variables
        'FTRA_Previous_Local_Inflow' : network.findJunction("System Inflow").getLocalFlowTimeSeries("FTRA_Local").getPreviousValue(currentRuntimestep),
        # GAPT Model Variables
        'GAPT_Previous_Local_Inflow' : network.findJunction("System Inflow").getLocalFlowTimeSeries("GAPT_Local").getPreviousValue(currentRuntimestep),
        # System Model Variables
        'System_Release_Data' : network.getTimeSeries("Reservoir","System Reservoir", "Pool", "Flow-OUT").getPreviousValue(currentRuntimestep),
        'SYS_Stor' : network.getTimeSeries("Reservoir","System Reservoir", "Pool", "Stor").getPreviousValue(currentRuntimestep),
        'System_Carryover_and_Multiple_Use_Stor': network.getTimeSeries("Reservoir","System Reservoir", "Carryover and Multiple Use", "Stor-ZONE").getPreviousValue(currentRuntimestep)
    }

    return Model_Variable

```

```

# retrieveModelVariables Function: Retrieve model variables needed for the script
def retrieveStateVariables(currentRuntimestep):

```

```

    State_Variable = {
        # Winter Release State Variables
        'Winter_Release' : network.getStateVariable("GAPTWinterRelease").getValue(currentRuntimestep),
        'Service_Level' : network.getStateVariable("ServiceLevel").getValue(currentRuntimestep),
        'NavigationEndDate' : network.getStateVariable("NavigationEndDate").getValue(currentRuntimestep)
    }

    return State_Variable

```

```

# -----
# Input Data
# -----

```

```

# Simulation Time & Step Info
rtw = ResSim.getCurrentModule().getSimulation().getRunTimeWindow()
startRtw = rtw.getStartTime() # Start time of simulation

```

```
lookbackRtw = rtw.getLookbackTime() # Start of lookback period
endRtw = rtw.getEndTime() # End time of simulation
```

```
# -----
# Runoff Forecasts
# -----
```

```
# Month conversions
```

```
monDict = {
    1 : ['Jan', 'January'],
    2 : ['Feb', 'February'],
    3 : ['Mar', 'March'],
    4 : ['Apr', 'April'],
    5 : ['May', 'May'],
    6 : ['Jun', 'June'],
    7 : ['Jul', 'July'],
    8 : ['Aug', 'August'],
    9 : ['Sep', 'September'],
    10 : ['Oct', 'October'],
    11 : ['Nov', 'November'],
    12 : ['Dec', 'December']
}
```

```
# Project identifiers and names (BEND/Lake Sharpe is included in FTRA/Lake Francis Case)
```

```
projects = [ ["FTPK", "Fort Peck Lake"],
             ["GARR", "Lake Sakakawea"],
             ["OAHE", "Lake Oahe"],
             ["FTRA", "Lake Francis Case"],
             ["GAPT", "Lewis and Clark Lake"]
           ]
```

```
projectIds, projectNames = zip(*projects)
```

```
# Normal runoff values for months not included in DRM forecast files. Normal values taken from CY 2014 Runoff Forecast
```

```
forecastNormalRunoffMonths = ['Aug', 'Sep', 'Oct', 'Nov', 'Dec']
#
# normalRunoff = {
#   Location      Aug  Sep      Oct      Nov      Dec      Jan      Feb
#   'FTPK' :      [356, 330, 380, 381, 327, 312, 361],
#   'GARR' :      [609, 448, 527, 393, 249, 261, 355],
#   'OAHE' :      [71, 110, 72, 67, 1, 12, 95],
#   'FTRA' :      [41, 37, 5, 4, 12, 28, 54],
#   'GAPT' :      [116, 111, 120, 118, 100, 100, 132]
# }
```

```
# Sum of the normal runoff through the following February
```

```
# Dictionary Format = Project : Starting Month : Total Runoff through February
```

```
EOYnormalRunoff = {}
```

```
for projectId in projectIds:
```

```
    for m in range(len(forecastNormalRunoffMonths)):
```

```
        runoff = sum(normalRunoff[projectId][m:])
```

```
        EOYnormalRunoff.setdefault(projectId, {}).setdefault(forecastNormalRunoffMonths[m], runoff)
```

```
forecastReleaseMonths = ['Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec', 'Jan', 'Feb']
```

```
normalReleaseAF = { 'GAPT' : [1642, 1642, 1721, 1660, 1949, 2041, 1940, 1967, 1851, 1045, 1045, 944]}
```

```
projectsGAPT = [{"GAPT", "Lewis and Clark Lake"}]
```

```

projectIdGAPT, projectGAPT = zip(*projectsGAPT)
# Sum of the normal GAPT release through the following February
# Dictionary Format = Project : Starting Month : Total Runoff through February
EOYnormalReleaseAF = {}
for projectgp in projectIdGAPT:
    for m in range(len(forecastReleaseMonths)):
        release = sum(normalReleaseAF[projectgp][m:])
        EOYnormalReleaseAF.setdefault(projectgp, {}).setdefault(forecastReleaseMonths[m], release)

# DRM forecasts
sharedDirectory = ResSim.getWatershed().getSharedDirectory()

months = [['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul']]
months.append(["%2.2d" % (i+1) for i in range(len(months[0]))])
fcstZipfileName = os.path.join(sharedDirectory, 'DRM_Forecast_text.zip')

fcstFlows = {}
zf = ZipFile(fcstZipfileName)
for itemname in zf.namelist() :
    if len(itemname) != 10 or not itemname.lower().startswith("fc") or not itemname.lower().endswith(".txt") :
        #outputDebug(debug, 'Ignoring zipped file %s' % itemname)
        continue
    try :
        year = int(itemname[2:6])
    except :
        #outputdebug(debug, 'Ignoring zipped file %s' % itemname)
        continue

    lines = zf.read(itemname).strip().split("\n")
    for i in range(len(lines)) :
        fields = lines[i].split()
        if i == 0 :
            assert not len(fields) < 3 and fields[0] == "FORECAST" and fields[2] == "KA-F"
            if int(fields[1]) != year :
                #outputDebug(debug, 'WARNING : ' + 'Zipped file %s indicates year %s in header' % (itemname, fields[1]))
                p=0
        elif i == 1 :
            assert not (len(fields) < 7 and fields[2:7] == list(projectIds))
        else :
            assert not len(fields) < 7 and fields[0] in months[0]
            fcstMonth = fields[0]
            mon, yr = fields[1].split("-")
            assert mon in months[1]
            if int(yr) != year :
                #outputDebug(debug, 'WARNING : ' + 'Zipped file %s indicates year %s at line %d' % (itemname, yr, i+1))
                p =0
            for j in range(len(projectIds)) :
                try :
                    val = float(fields[2+j])
                except :
                    outputDebug(debug, 'WARNING : ' +
                        'Invalid discharge value (%s) for project %s at line %d in zipped file %s, using 0' % \
                        (fields[2+j],projectIds[j], i+1, itemname))

```

```

                val = 0
                fcstFlows.setdefault(year, {}).setdefault(projectIds[j], {}).setdefault(fcstMonth, []).append(val)
zf.close()

# Calculate EOY runoff forecasts
# Process the DRM forecast runoff into annual totals for forecast month through Feb of the following year for each project. Aug-Dec runoff
# values are normal runoff values
EOY_Runoff = {}
EOYforecastRunoffMonths = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']
for projectId in projectIds:
    for year in range(1898, max(fcstFlows.keys()) + 1, 1): # year will be equal to the year of the last DRM forecasting file
        for month in EOYforecastRunoffMonths:
            if month == 'Mar' or month == 'Apr' or month == 'May' or month == 'Jun' or month == 'Jul':
                runoff = sum(fcstFlows[year][projectId][month]) + EOYnormalRunoff[projectId]['Aug']
                EOY_Runoff.setdefault(projectId, {}).setdefault(year, {}).setdefault(month, runoff)
            elif month == 'Aug' or month == 'Sep' or month == 'Oct' or month == 'Nov' or month == 'Dec':
                runoff = EOYnormalRunoff[projectId][month]
                EOY_Runoff.setdefault(projectId, {}).setdefault(year, {}).setdefault(month, runoff)
            elif month == 'Jan':
                runoff = sum(fcstFlows[year][projectId][month]) - sum(fcstFlows[year][projectId][month][2:])
                EOY_Runoff.setdefault(projectId, {}).setdefault(year, {}).setdefault(month, runoff)
            elif month == 'Feb':
                runoff = sum(fcstFlows[year][projectId][month]) - sum(fcstFlows[year][projectId][month][1:])
                EOY_Runoff.setdefault(projectId, {}).setdefault(year, {}).setdefault(month, runoff)

# -----
# Conversions
# -----

CFS_to_ACRE_FT = 86400.0/43560.0
ACRE_FT_to_CFS = 43560.0/86400.0

# -----
# init info that is passed to runRuleScript()
# -----

initInfo = {
    'ACRE_FT_to_CFS'      : ACRE_FT_to_CFS,
    'CFS_to_ACRE_FT'     : CFS_to_ACRE_FT,
    'debug'              : debug,
    'EOYnormalReleaseAF' : EOYnormalReleaseAF,
    'EOY_Runoff'         : EOY_Runoff,
    'floodEvac'          : floodEvac,
    'janJulSysRunoff'    : janJulSysRunoff,
    'julianDay'          : julianDay,
    'monDict'            : monDict,
    'monthlyAccumRunoff' : monthlyAccumRunoff,
    'outputDebug'        : outputDebug,
    'projectIds'         : projectIds,
    'projectNames'       : projectNames,
    'remainingAnnualRunoff' : remainingAnnualRunoff,
    'retrieveModelVariables' : retrieveModelVariables,
    'retrieveStateVariables' : retrieveStateVariables
}

```

```

    }

currentRule.varPut('initInfo', initInfo)

# return Constants.TRUE if the initialization is successful
# and Constants.FALSE if it failed. Returning Constants.FALSE
# will halt the compute.
return Constants.TRUE

# runRuleScript() is the entry point that is called during the
# compute.
#
# currentRule is the rule that holds this script
# network is the ResSim network
# currentRuntimestep is the current Run Time Step
def runRuleScript(currentRule, network, currentRuntimestep):

# -----
# Retrieve init info from initRuleScript() and set equal to variables
# -----

initInfo = currentRule.varGet('initInfo')

ACRE_FT_to_CFS      = initInfo['ACRE_FT_to_CFS']
CFS_to_ACRE_FT     = initInfo['CFS_to_ACRE_FT']
debug              = initInfo['debug']
EOYnormalReleaseAF = initInfo['EOYnormalReleaseAF']
EOY_Runoff         = initInfo['EOY_Runoff']
floodEvac          = initInfo['floodEvac']
janJulSysRunoff   = initInfo['janJulSysRunoff']
julianDay          = initInfo['julianDay']
monDict            = initInfo['monDict']
monthlyAccumRunoff = initInfo['monthlyAccumRunoff']
outputDebug        = initInfo['outputDebug']
projectIds         = initInfo['projectIds']
projectNames       = initInfo['projectNames']
remainingAnnualRunoff = initInfo['remainingAnnualRunoff']
retrieveModelVariables = initInfo['retrieveModelVariables']
retrieveStateVariables = initInfo['retrieveStateVariables']

# create new Operation Value (OpValue) to return
opValue = OpValue()

# -----
# Input Data
# -----

# Simulation Time & Step Info
rtw = currentRuntimestep.getRunTimeWindow() # Run time window
startRtw = rtw.getStartTime() # Start time of simulation
lookbackRtw = rtw.getLookbackTime() # Start of lookback period
endRtw = rtw.getEndTime() # End time of simulation

```

```

totalSteps = rtw.getNumSteps() # Total number of steps in the simulation
curStep = currentRuntimestep.getStep() # Current step
year = currentRuntimestep.getHecTime().year() # Current year
mon = currentRuntimestep.getHecTime().month() # Current month
day = currentRuntimestep.getHecTime().day() # Current day
jDay = julianDay(mon, day) # Current julian day
numLookbackSteps = rtw.getNumLookbackSteps() # Number of lookback steps in simulation
simStartStep = numLookbackSteps + 1 # First step after the lookback period
PassCounter = network.getComputePassCounter() # Pass number of simulation
PassNumber = PassCounter + 1 # Pass number starts at 0 so add 1 to start it at 1

# -----
# Main Script
# -----

if PassNumber < 3:
    opValue.init(OpRule.RULETYPE_MIN, 0)
else:
    # -----
    # Retrieve model variables needed for the script in a dictionary
    # -----

    Model_Variable = retrieveModelVariables(currentRuntimestep)
    State_Variable = retrieveStateVariables(currentRuntimestep)

    # -----
    # Monthly Accumulated Incremental Runoff
    # -----

    FTPK_Accum_Stor, GARR_Accum_Stor, OAHE_Accum_Stor, BEND_and_FTFR_Accum_Stor, GAPT_Accum_Stor, SYS_Accum_Release = monthlyAccumRunoff(currentRuntimestep,
        Model_Variable['RBMT_Previous_Local_Inflow'], Model_Variable['FTPK_Previous_Local_Inflow'], Model_Variable['WPMT_Previous_Local_Inflow'],
        Model_Variable['CLMT_Previous_Local_Inflow'], Model_Variable['GARR_Previous_Local_Inflow'], Model_Variable['BIS_Previous_Local_Inflow'],
        Model_Variable['OAHE_Previous_Local_Inflow'], Model_Variable['BEND_Previous_Local_Inflow'], Model_Variable['FTFR_Previous_Local_Inflow'],
        Model_Variable['GAPT_Previous_Local_Inflow'], Model_Variable['System_Release_Data'])

    # -----
    # Calculate Jan-Jul System Runoff to Adjust Normal Forecasted Runoff
    # -----

    janJulSysRunoff(currentRuntimestep, Model_Variable['RBMT_Previous_Local_Inflow'], Model_Variable['FTPK_Previous_Local_Inflow'],
        Model_Variable['WPMT_Previous_Local_Inflow'], Model_Variable['CLMT_Previous_Local_Inflow'], Model_Variable['GARR_Previous_Local_Inflow'],
        Model_Variable['BIS_Previous_Local_Inflow'], Model_Variable['OAHE_Previous_Local_Inflow'], Model_Variable['BEND_Previous_Local_Inflow'],
        Model_Variable['FTFR_Previous_Local_Inflow'], Model_Variable['GAPT_Previous_Local_Inflow'])

    if curStep == simStartStep or day == 1 or day == 15:
        # -----
        # Calculate Remaining Annual Incremental Runoff
        # -----

        SYS_Remaining_Runoff, SYS_Remaining_Release = remainingAnnualRunoff(currentRuntimestep, FTPK_Accum_Stor, GARR_Accum_Stor, OAHE_Accum_Stor,
            BEND_and_FTFR_Accum_Stor, GAPT_Accum_Stor, SYS_Accum_Release)

        gaptRelease_Min = floodEvac(currentRuntimestep, curStep, Model_Variable['SYS_Stor'], SYS_Remaining_Runoff, SYS_Remaining_Release,

```

```
        State_Variable['Winter_Release'], Model_Variable['System_Carryover_and_Multiple_Use_Stor'], State_Variable['NavigationEndDate'],
        State_Variable['Service_Level'], jDay, mon, year)
else:
    Prev_gaptRelease_Min = currentRule.varGet('Prev_gaptRelease_Min')
    gaptRelease_Min = Prev_gaptRelease_Min

# set type and value for OpValue
# type is one of:
# OpRule.RULETYPE_MAX - maximum flow
# OpRule.RULETYPE_MIN - minimum flow
# OpRule.RULETYPE_SPEC - specified flow
opValue.init(OpRule.RULETYPE_MIN, gaptRelease_Min)

# return the Operation Value.
# return "None" to have no effect on the compute
return opValue
```

## 15 APPENDIX I – STEADY RELEASE SCRIPTED RULE

```

'''
Author: Alex Flanigan
Date: 06-16-2015
Description: Logic for setting GAPT steady releases. Steady release magnitude calculations are completed in the ServiceLevel state variable script.
'''
# -----
# Required imports to create the OpValue return object.
# -----
from hec.rss.model import OpValue
from hec.rss.model import OpRule
from hec.script import Constants

def initRuleScript(currentRule, network):
    # -----
    # Functions
    # -----

    # Debugging Function: Set debug = True to print all debug statements and = False to turn them off
    # Initialization Debugging Function: Set debug = True to print all debug statements and = False to turn them off
    debug = False
    def initOutputDebug(debug, arg1="", arg2="", arg3="", arg4="", arg5="", arg6="", arg7="", arg8="", arg9="", arg10="", arg11="", arg12="", arg13="", arg14=""):
        if debug == True:
            print 'Initialization Debug  \t', arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9, arg10, arg11, arg12, arg13, arg14

    # Debugging Function: Set debug = True to print all debug statements and = False to turn them off
    def outputDebug(currentRuntimestep, debug, arg1="", arg2="", arg3="", arg4="", arg5="", arg6="", arg7="", arg8="", arg9="", arg10="", arg11="", arg12="",
        arg13="", arg14="", arg15="", arg16="", arg17=""):
        PassCounter = network.getComputePassCounter() # Pass number of simulation
        PassNumber = PassCounter + 1 # Pass number starts at 0 so add 1 to start it at 1
        if debug == True:
            print 'Pass Number =', PassNumber, ' ', currentRuntimestep.dateTimeString(), \
                '\t', arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9, arg10, arg11, arg12, arg13, arg14, arg15, arg16, arg17, '\n'

    # julianDay Function: Get the julian day for a month and day or a string, or a month and day for a julian day
    def julianDay(*args) :
        # J F M A M J J A S O N D
        daysPrev = 0, 31, 59, 90,120,151,181,212,243,273,304,334
        argCount = len(args)
        if argCount == 1 :
            if type(args[0]) == type(0) :
                jday = args[0]
                if not 1 <= jday <= 365 :
                    raise ValueError("Julian day out of range 1..365")
                for i in range(12)[::-1] :
                    if jday > daysPrev[i] :
                        mon = i + 1
                        break
                day = jday - daysPrev[mon-1]
                return mon, day
            else :
                mon, day = args[0].split("_")
                mon = ["JAN","FEB","MAR","APR","MAY","JUN","JUL","AUG","SEP","OCT","NOV","DEC"].index(mon.upper()) + 1
                day = int(day);

```

```

        return julianDay(mon, day)
elif argCount == 2 :
    mon, day = args
    if not 1 <= mon <= 12 :
        raise ValueError("Month is out of range 1..12")
    if not 1 <= day <= 31 :
        raise ValueError("Day is out of range 1..31")
    if mon == 2 and day == 29 : day = 28
    return daysPrev[mon-1] + day
else :
    raise ValueError("Expected 1 or 2 arguments, got %d" % argCount)

# retrieveModelVariables Function: Retrieve model variables needed for the script
def retrieveModelVariables( currentRuntimestep, # currentRuntimestep variable passed in through the runRuleScript()
                           totalSteps        # Total number of steps in the simulation
                           ):
    Model_Variable = { # GAPT Model Variables
        'gaptReleaseYesterday' : network.getTimeSeries("Reservoir", "System Reservoir", "Pool", "Flow-OUT").getPreviousValue(currentRuntimestep),
        'gaptRelease'           : network.getTimeSeries("Reservoir", "System Reservoir", "Pool", "Flow-OUT").getDataArrayFor(0, totalSteps),
        'James_Flow'            : network.findJunction("SCSD").getLocalFlowTimeSeries("SCSD_Flow").getPreviousValue(currentRuntimestep)
    }

    return Model_Variable

# retrieveModelVariables Function: Retrieve model variables needed for the script
def retrieveStateVariables(currentRuntimestep):
    State_Variable = { # Service Level State Variables
        'Service_Level'       : network.getStateVariable("ServiceLevel").getValue(currentRuntimestep),
        'Steady_Release'      : network.getStateVariable("GAPTSteadyRelease").getValue(currentRuntimestep),
        'High_Steady_Release' : network.getStateVariable("HighSteadyRelease").getValue(currentRuntimestep),
        'SL_MKC'              : network.getStateVariable("SL_MKC").getValue(currentRuntimestep)
    }

    return State_Variable

# Peak steady release variable
PeakSR = 0

# -----
# init info that is passed to runRuleScript()
# -----

initInfo = {
    'debug'           : debug,
    'julianDay'       : julianDay,
    'outputDebug'     : outputDebug,
    'PeakSR'          : PeakSR,
    'retrieveModelVariables' : retrieveModelVariables,
    'retrieveStateVariables' : retrieveStateVariables
}

currentRule.varPut('initInfo', initInfo)

```

```

#currentRule.varPut('julianDay', julianDay)
# return Constants.TRUE if the initialization is successful
# and Constants.FALSE if it failed. Returning Constants.FALSE
# will halt the compute.
return Constants.TRUE

```

```

# runRuleScript() is the entry point that is called during the
# compute.
#
# currentRule is the rule that holds this script
# network is the ResSim network
# currentRuntimestep is the current Run Time Step
def runRuleScript(currentRule, network, currentRuntimestep):

```

```

# create new Operation Value (OpValue) to return
opValue = OpValue()

```

```

# -----
# Retrieve init info from initRuleScript() and set equal to variables
# -----

```

```

initInfo = currentRule.varGet('initInfo')

```

```

debug                = initInfo['debug']
julianDay             = initInfo['julianDay']
outputDebug          = initInfo['outputDebug']
PeakSR               = initInfo['PeakSR']
retrieveModelVariables = initInfo['retrieveModelVariables']
retrieveStateVariables = initInfo['retrieveStateVariables']

```

```

# -----
# Input Data
# -----

```

```

# Simulation Time & Step Info
mon = currentRuntimestep.getHecTime().month()
day = currentRuntimestep.getHecTime().day()
jDay = julianDay(mon, day)
currentHecTime = currentRuntimestep.getHecTime()
rtw = currentRuntimestep.getRunTimeWindow() # Run time window
totalSteps = rtw.getNumSteps() # Total number of steps in the simulation. rtw gives steps in hours so divide by 24 to get days
curStep = currentRuntimestep.getStep() # Current step
PassCounter = network.getComputePassCounter() # Pass number of simulation
PassNumber = PassCounter + 1

```

```

# -----
# Retrieve model variables needed for the script in a dictionary
# -----

```

```

Model_Variable = retrieveModelVariables(currentRuntimestep, totalSteps)
State_Variable = retrieveStateVariables(currentRuntimestep)

```

```

if PassNumber < 3 or State_Variable['Service_Level'] < 29 or Model_Variable['gaptReleaseYesterday'] > 45000:
    opValue.init(OpRule.RULETYPE_MIN, 0)
else:
    # Don't let steady release exceed the target discharge at Kansas City
    if State_Variable['High_Steady_Release'] > State_Variable['SL_MKC']:
        State_Variable['High_Steady_Release'] = State_Variable['SL_MKC']
        outputDebug(currentRuntimestep, debug, 'High Steady Release Set to Kansas City Target: %.0f cfs' % State_Variable['High_Steady_Release'])
    outputDebug(currentRuntimestep, debug, 'Steady Release = %.0f cfs' % State_Variable['Steady_Release'])
    # Don't use steady release if system releases are high (flood evacuation) or outside of the steady release period
    if State_Variable['Steady_Release'] <= 0:
        # Set High Steady Release to undefined when outside of steady release period
        network.getStateVariable("HighSteadyRelease").setValueRange(currentRuntimestep, currentHecTime, rtw.getEndTime(), Constants.UNDEFINED)
        State_Variable['High_Steady_Release'] = network.getStateVariable("HighSteadyRelease").getValue(currentRuntimestep)
        opValue.init(OpRule.RULETYPE_MIN, 0)
    else:
        # If the 'book' value for steady release was exceeded (High Steady Release) because of downstream needs, set the new steady release equal to
        # the High Steady Release
        if State_Variable['High_Steady_Release'] > State_Variable['Steady_Release']:
            State_Variable['Steady_Release'] = State_Variable['High_Steady_Release']

        if (Model_Variable['gaptReleaseYesterday'] - State_Variable['Steady_Release']) >= 0:
            # Save new steady release value to High Steady Release if previous release exceeded the current steady release value
            gaptRelease = Model_Variable['gaptReleaseYesterday']
            network.getStateVariable("HighSteadyRelease").setValueRange(currentRuntimestep, currentHecTime, rtw.getEndTime(), gaptRelease)
        else:
            # Increase current system release if previous system release was less than the steady release value
            gaptRelease = Model_Variable['gaptReleaseYesterday'] + abs(Model_Variable['gaptReleaseYesterday'] - State_Variable['Steady_Release'])
            rocLimit = gaptRelease - Model_Variable['gaptReleaseYesterday']
            if rocLimit > 2400:
                gaptRelease = Model_Variable['gaptReleaseYesterday'] + 2400

            # Don't let the system release exceed the target discharge at Kansas City
            if State_Variable['SL_MKC'] > 0 and gaptRelease > State_Variable['SL_MKC']:
                gaptRelease = State_Variable['SL_MKC']

            # Save system release as the High Steady Release
            network.getStateVariable("HighSteadyRelease").setValueRange(currentRuntimestep, currentHecTime, rtw.getEndTime(), gaptRelease)
        opValue.init(OpRule.RULETYPE_MIN, gaptRelease)
    # return the Operation Value.
    # return "None" to have no effect on the compute
    return opValue

```

## 16 APPENDIX J – GAPD CONTROL SCRIPTED RULE

```

'''
Author: Ryan Larsen
Date: 06-17-2015
Alternative: FWOP
Description: Estimates OAHE, BEND, and FTRA releases using forecasted GAPT release values calculated from Missouri River Downstream model with all
system storage included in GAPT. This script first calculates specified releases for FTRA based on GAPT releases calculated from the Downstream
model that allows GAPT to remain at guide curve. Then it calculates OAHE releases based on FTRA releases that allows FTRA to remain at guide
curve. Finally, it calculates BEND releases based on its guide curve and the inflow into the reservoir.
'''
# -----
# Required imports to create the OpValue return object.
# -----

from hec.rss.model import OpRule
from hec.script import Constants
from hec.heclib.util import *
from hec.rss.model import *
from hec.heclib.dss import *
from hec.hecmath import *
from hec.model import *
from hec.script import *
from hec.io import TimeSeriesContainer
from hec.client import ClientApp
from java.lang import String
import traceback

# -----
# Global variables
# -----

global totalSteps
global curStep
global numLookbackSteps
global GAPT_futStep
global FTRA_futStep
global lag
global CFS_to_ACRE_FT
global ACRE_FT_to_CFS
global Initial_Upstream_Release
global FTRA_Previous_Release

# initialization function. optional.
# set up tables and other things that only need to be performed once during the compute.
# currentRule is the rule that holds this script.
# network is the ResSim network.
def initRuleScript(currentRule, network):
    # -----
    # Set alternative name and depletions to read correct DSS datasets
    # -----

    Alt_Name      = 'FWOP'
    Depletions    = 'Present'    # If Depletions = Present, then script will use local flow datasets that have been adjusted for present level depletions
                                # If Depletions = Historic, then script will use local flow datasets that are based on historic level depletions

```

```

# -----
# Functions
# -----
# Initialization Debugging Function: Set debug = True to print all debug statements and = False to turn them off
debug = False
def initOutputDebug(debug, arg1=", arg2=", arg3=", arg4=", arg5=", arg6=", arg7=", arg8=", arg9=", arg10=", arg11=", arg12=", arg13=", arg14="):
    if debug == True:
        print 'Initialization Debug  \t', arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9, arg10, arg11, arg12, arg13, arg14

# Debugging Function: Set debug = True to print all debug statements and = False to turn them off
def outputDebug(currentRuntimestep, debug, arg1=", arg2=", arg3=", arg4=", arg5=", arg6=", arg7=", arg8=", arg9=", arg10=", arg11=", arg12=", arg13=", arg14=", arg15="):
    PassCounter = network.getComputePassCounter() # Pass number of simulation
    PassNumber = PassCounter + 1 # Pass number starts at 0 so add 1 to start it at 1
    if debug == True:
        print 'Pass Number =', PassNumber, ' ', currentRuntimestep.dateTimeString(), \
            '\t', arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9, arg10, arg11, arg12, arg13, arg14, arg15, '\n'

# adjustFallRelease Function: Adjust fall releases if evacuating flood water
def adjustFallRelease(    currentRuntimestep, # currentRuntimestep variable passed in through the runRuleScript()
                        jDay,              # Julian day of the current simulation step
                        curStep,           # Current step of the simulation
                        GAPT_Release_Change # Array of GAPT release differences
                        ):
    Step_Adjustment = julianDay('NOV_30') - julianDay('SEP_01')
    NOV_30_Step = curStep + Step_Adjustment
    currentRuntimestep.setStep(NOV_30_Step)
    NOV_30_GAPT_Release = GAPT_Specified_Release[curStep]
    currentRuntimestep.setStep(curStep)
    if NOV_30_GAPT_Release >= 40000 and sum(GAPT_Release_Change) < 0:
        Step_Adjustment = julianDay('DEC_10') - julianDay('SEP_01')
        DEC_10_Step = curStep + Step_Adjustment
        for step in range(DEC_10_Step, curStep, -1):
            currentRuntimestep.setStep(step)
            if GAPT_Specified_Release[step] > 40000:
                Adjustment = GAPT_Specified_Release[step] - 40000
                GAPT_Specified_Release[step] = 40000
                GAPT_Release_Change.append(Adjustment)
                outputDebug(currentRuntimestep, True, 'Reduce releases by %.0f cfs' % Adjustment, '\tRelease change = %.0f' % sum(GAPT_Release_Change))
                currentRule.varPut('GAPT_Release_Change', GAPT_Release_Change)
            if sum(GAPT_Release_Change) >= 0:
                outputDebug(currentRuntimestep, True, 'GAPT release change = 0')
                break
    currentRuntimestep.setStep(curStep)

# adjustWinterRelease Function: Adjust winter releases if the specified GAPT releases were not released during the year
def adjustWinterRelease(    currentRuntimestep, # currentRuntimestep variable passed in through the runRuleScript()
                           jDay,              # Current julian day
                           curStep,           # Current step of the simulation
                           year,              # Current year of simulation
                           End_Year,          # Ending year of simulation
                           End_rtw_JulianDay, # Julian day of the last step of the simulation
                           GAPT_Release_Change # Array of GAPT release differences

```

```

):
Minimum_Winter_Release = 12000
Total_Adjustment_Days = (julianDay('DEC_31') - julianDay('DEC_15')) + julianDay('MAR_01')
Winter_Release_Adjust = sum(GAPT_Release_Change) / Total_Adjustment_Days
outputDebug(currentRuntimestep, True, 'Daily winter release adjustment = %.0f' % Winter_Release_Adjust)
if ((year + 1) == End_Year or year == End_Year) and End_rtw_JulianDay > julianDay('DEC_15'):
    for d in range(End_rtw_JulianDay - julianDay('DEC_15')):
        if (GAPT_Specified_Release[curStep + d] + Winter_Release_Adjust) > Minimum_Winter_Release:
            GAPT_Specified_Release[curStep + d] = GAPT_Specified_Release[curStep + d] + Winter_Release_Adjust
            GAPT_Release_Change.append(-Winter_Release_Adjust)
        else:
            GAPT_Release_Change.append(Minimum_Winter_Release - (GAPT_Specified_Release[curStep + d] + Winter_Release_Adjust))
            GAPT_Specified_Release[curStep + d] = Minimum_Winter_Release
elif (year + 1) == End_Year and End_rtw_JulianDay < julianDay('MAR_01'):
    for d in range((julianDay('DEC_31') - julianDay('DEC_15')) + End_rtw_JulianDay):
        if (GAPT_Specified_Release[curStep + d] + Winter_Release_Adjust) > Minimum_Winter_Release:
            GAPT_Specified_Release[curStep + d] = GAPT_Specified_Release[curStep + d] + Winter_Release_Adjust
            GAPT_Release_Change.append(-Winter_Release_Adjust)
        else:
            GAPT_Release_Change.append(Minimum_Winter_Release - (GAPT_Specified_Release[curStep + d] + Winter_Release_Adjust))
            GAPT_Specified_Release[curStep + d] = Minimum_Winter_Release
else:
    for d in range(Total_Adjustment_Days):
        if (GAPT_Specified_Release[curStep + d] + Winter_Release_Adjust) >= Minimum_Winter_Release:
            GAPT_Specified_Release[curStep + d] = GAPT_Specified_Release[curStep + d] + Winter_Release_Adjust
            GAPT_Release_Change.append(-Winter_Release_Adjust)
        elif GAPT_Specified_Release[curStep + d] > Minimum_Winter_Release and (GAPT_Specified_Release[curStep + d] + Winter_Release_Adjust) < Minimum_Winter_Release:
            GAPT_Release_Change.append(GAPT_Specified_Release[curStep + d] - Minimum_Winter_Release)
            GAPT_Specified_Release[curStep + d] = Minimum_Winter_Release
    outputDebug(currentRuntimestep, True, 'Daily winter release adjustment = %.0f' % Winter_Release_Adjust)
    currentRule.varPut('GAPT_Release_Change', GAPT_Release_Change)

```

```

# forecastBENDReleases Function: Set BEND releases based on guide curve operation
def forecastBENDReleases( curStep,
                        # Current step of the simulation
                        Upstream_Location, # DCP ID of upstream dam
                        Upstream_Release, # Specified release from upstream dam
                        Local_Flow,      # Local flow at current dam
                        Previous_Elev,    # Previous pool elevation at current dam
                        Previous_Stor,    # Previous storage at current dam
                        GC_Previous_Stor, # Previous guide curve storage at current dam
                        GC_Previous_Elev # Previous guide curve elevation at current dam
):
    if Previous_Elev > (GC_Previous_Elev + 0.3):
        Routed_Release = routeFlow(Upstream_Location, Upstream_Release)
        Inflow = Routed_Release[curStep] + Local_Flow[curStep]
        Current_Specified_Release = Inflow + ((Previous_Stor - GC_Previous_Stor) * ACRE_FT_to_CFS)
    elif Previous_Elev < (GC_Previous_Elev - 0.3):
        Routed_Release = routeFlow(Upstream_Location, Upstream_Release)
        Inflow = Routed_Release[curStep] + Local_Flow[curStep]
        Current_Specified_Release = Inflow - ((GC_Previous_Stor - Previous_Stor) * ACRE_FT_to_CFS)
    else:
        Routed_Release = routeFlow(Upstream_Location, Upstream_Release)
        Inflow = Routed_Release[curStep] + Local_Flow[curStep]

```

```

Current_Specified_Release = Inflow
return Current_Specified_Release

```

```

# forecastFTRARelases Function: Forecast FTRA releases

```

```

def forecastFTRARelases( currentRuntimestep, # currentRuntimestep variable passed in through the runRuleScript()
                        lag, # Lag time between FTRA and GAPT
                        Upstream_Release, # FTRA release
                        Downstream_Release, # GAPT release
                        Previous_Stor # Previous FTRA storage
                        ):

```

```

outputDebug(currentRuntimestep, debug, 'FORECAST FTRA RELEASES')
curStep = currentRuntimestep.getStep() # Current step
forecastPeriod = lag * 2 # Number of days to run the release forecast
threshold = 1000 # Allowable difference between the guide curve storage and forecasted storage
FTRA_Increase_ROC = 12000 # Daily increase ROC of change for FTRA releases
FTRA_Decrease_ROC = -17000 # Daily decrease ROC of change for FTRA releases
GAPT_Forecast_Previous_Stor = [None] * (lag * 4)
GAPT_Forecast_Previous_Stor[0] = Previous_Stor

```

```

# Reset previous forecast data to original releases

```

```

for step in range(forecastPeriod):
    if (curStep + step) > totalSteps or (curStep + step) > (curStep + forecastPeriod):
        break
    Upstream_Release[curStep + step] = Upstream_Release[curStep - 1 + step]

```

```

# Calculate FTRA releases for a forecasted period so GAPT remains near guide curve

```

```

for step in range(0, forecastPeriod, lag):
    Iteration_Counter = 0
    currentRuntimestep.setStep(curStep + step + lag - 1) # Set currentRuntimestep equal to a future step
    GAPT_GC_Future_Stor = network.getTimeSeries("Reservoir", "Lewis and Clark Lake", "Guide Curve", "Stor-ZONE").getCurrentValue(currentRuntimestep) # Guide curve storage based on a future timestep
    currentRuntimestep.setStep(curStep) # Reset currentRuntimestep to the current step

```

```

# Calculate initial storage error

```

```

forecastStep = curStep + step
if (forecastStep + lag) > totalSteps or (forecastStep + lag) > (curStep + forecastPeriod):
    break
routedRelease = routeFlow('FTRA', Upstream_Release[forecastStep - lag:forecastStep + lag])
Upstream_Release_Inflow = sum(routedRelease[lag:])
Local_Inflow = sum(GAPT_Local[forecastStep:forecastStep + lag])
Release = sum(Downstream_Release[forecastStep:forecastStep + lag])
Storage_Change = (Local_Inflow + Upstream_Release_Inflow - Release) * CFS_to_ACRE_FT
Downstream_Future_Storage = GAPT_Forecast_Previous_Stor[step] + Storage_Change
error = Downstream_Future_Storage - GAPT_GC_Future_Stor
outputDebug(currentRuntimestep, False, 'Sum of Routed FTRA Release = %.0f cfs' % Upstream_Release_Inflow,
           '\n\t\t\tSum of Local Inflow = %.0f cfs' % Local_Inflow,
           '\n\t\t\tSum of GAPT Release = %.0f cfs' % Release,
           '\n\t\t\tGAPT Previous Stor = %.0f acre-ft' % GAPT_Forecast_Previous_Stor[step],
           '\n\t\t\tStorage Change = %.0f acre-ft' % Storage_Change,
           '\n\t\t\tStorage Error = %.0f acre-ft' % error)
delta = 10000 # Upstream release increment

```

```

# Optimize upstream releases to keep downstream reservoir at guide curve

```

```

while abs(error) > threshold:

```

```

Iteration_Counter += 1
if Iteration_Counter > 100:
    break
if error < 0:
    # Increase the flow for the current and all future timesteps within the forecasted period
    while error < 0 and abs(error) > threshold:
        Iteration_Counter += 1
        if Iteration_Counter > 100:
            break
        Total_ROC_Increase, Total_ROC_Decrease, Total_Max_Min = 0, 0, 0
        for t in range(lag):
            Upstream_Release[forecastStep + t] = Upstream_Release[forecastStep + t] + delta
            Release_Change = Upstream_Release[forecastStep + t] - Upstream_Release[forecastStep + t - 1]
            # Check releases against max release and rate of change rules
            if Upstream_Release[forecastStep + t] >= Max_Min_Release_Dict['FTRA']['Max_Release']:
                Upstream_Release[forecastStep + t] = Max_Min_Release_Dict['FTRA']['Max_Release']
                Total_Max_Min += Max_Min_Release_Dict['FTRA']['Max_Release']
            if Release_Change >= FTRA_Increase_ROC:
                Upstream_Release[forecastStep + t] = Upstream_Release[forecastStep + t - 1] + FTRA_Increase_ROC
                Total_ROC_Increase += FTRA_Increase_ROC
            if Release_Change <= FTRA_Decrease_ROC:
                Upstream_Release[forecastStep + t] = Upstream_Release[forecastStep + t - 1] + FTRA_Decrease_ROC
                Total_ROC_Decrease += FTRA_Decrease_ROC
        # Route the flow from upstream to downstream and calculate new storage
        routedRelease = routeFlow('FTRA', Upstream_Release[forecastStep - lag:forecastStep + lag])
        Upstream_Release_Inflow = sum(routedRelease[lag:])
        Storage_Change = (Local_Inflow + Upstream_Release_Inflow - Release) * CFS_to_ACRE_FT
        Downstream_Future_Storage = GAPT_Forecast_Previous_Stor[step] + Storage_Change
        error = Downstream_Future_Storage - GAPT_GC_Future_Stor
        outputDebug(currentRuntimestep, False, 'Sum of Routed FTRA Release = %.0f cfs' % Upstream_Release_Inflow,
            '\n\t\t\tSum of Local Inflow = %.0f cfs' % Local_Inflow,
            '\n\t\t\tSum of GAPT Release = %.0f cfs' % Release,
            '\n\t\t\tGAPT Previous Stor = %.0f acre-ft' % GAPT_Forecast_Previous_Stor[step],
            '\n\t\t\tStorage Change = %.0f acre-ft' % Storage_Change,
            '\n\t\t\tStorage Error = %.0f acre-ft' % error)

        # If release is set to maximum release or exceeds ROC, break the loop
        if Total_Max_Min == (Max_Min_Release_Dict['FTRA']['Max_Release'] * lag) or Total_ROC_Increase == (FTRA_Increase_ROC * lag) \
            or Total_ROC_Decrease == (FTRA_Decrease_ROC * lag):
            break
        delta = delta / 3
elif error > 0:
    # Decrease the flow for the current and all future timesteps within the forecasted period
    while error > 0 and abs(error) > threshold:
        Iteration_Counter += 1
        if Iteration_Counter > 100:
            break
        Total_ROC_Increase, Total_ROC_Decrease, Total_Max_Min = 0, 0, 0
        for t in range(lag):
            Upstream_Release[forecastStep + t] = Upstream_Release[forecastStep + t] - delta
            Release_Change = Upstream_Release[forecastStep + t] - Upstream_Release[forecastStep + t - 1]
            # Check releases against min release and rate of change rules
            if Upstream_Release[forecastStep + t] <= Max_Min_Release_Dict['FTRA']['Min_Release']:

```

```

        Upstream_Release[forecastStep + t] = Max_Min_Release_Dict['FTRA']['Min_Release']
        Total_Max_Min += Max_Min_Release_Dict['FTRA']['Min_Release']
    if Release_Change >= FTRA_Increase_ROC:
        Upstream_Release[forecastStep + t] = Upstream_Release[forecastStep + t - 1] + FTRA_Increase_ROC
        Total_ROC_Increase += FTRA_Increase_ROC
    if Release_Change <= FTRA_Decrease_ROC:
        Upstream_Release[forecastStep + t] = Upstream_Release[forecastStep + t - 1] + FTRA_Decrease_ROC
        Total_ROC_Decrease += FTRA_Decrease_ROC

# Route the flow from upstream to downstream and calculate new storage
routedRelease = routeFlow('FTRA', Upstream_Release[forecastStep - lag:forecastStep + lag])
Upstream_Release_Inflow = sum(routedRelease[lag:])
Storage_Change = (Local_Inflow + Upstream_Release_Inflow - Release) * CFS_to_ACRE_FT
Downstream_Future_Storage = GAPT_Forecast_Previous_Stor[step] + Storage_Change
error = Downstream_Future_Storage - GAPT_GC_Future_Stor
outputDebug(currentRuntimestep, False, 'Sum of Routed FTRA Release = %.0f cfs' % Upstream_Release_Inflow,
            '\n\t\t\tSum of Local Inflow = %.0f cfs' % Local_Inflow,
            '\n\t\t\tSum of GAPT Release = %.0f cfs' % Release,
            '\n\t\t\tGAPT Previous Stor = %.0f acre-ft' % GAPT_Forecast_Previous_Stor[step],
            '\n\t\t\tStorage Change = %.0f acre-ft' % Storage_Change,
            '\n\t\t\tStorage Error = %.0f acre-ft' % error)

# If release is set to minimum release or exceeds ROC, break the loop
if Total_Max_Min == (Max_Min_Release_Dict['FTRA']['Min_Release'] * lag) or Total_ROC_Increase == (FTRA_Increase_ROC * lag) \
or Total_ROC_Decrease == (FTRA_Decrease_ROC * lag):
    break
    delta = delta / 3
# If release is set to max/min release or exceeds ROC rules, break the loop
if Total_Max_Min == (Max_Min_Release_Dict['FTRA']['Max_Release'] * lag) \
or Total_Max_Min == (Max_Min_Release_Dict['FTRA']['Min_Release'] * lag) \
or Total_ROC_Increase == (FTRA_Increase_ROC * lag) or Total_ROC_Decrease == (FTRA_Decrease_ROC * lag):
    break
outputDebug(currentRuntimestep, debug, 'Forecasted Elevation = %.2f ft' % GAPT_ElevStorTable.interpolate(Downstream_Future_Storage),
            '\tForecasted GC Elevation = %.2f ft' % GAPT_ElevStorTable.interpolate(GAPT_GC_Future_Stor))
GAPT_Forecast_Previous_Stor[step + lag] = Downstream_Future_Storage

# Check elevation constraints and increase/decrease releases if elevation constraints are exceeded
outputDebug(currentRuntimestep, debug, 'FTRA Release Schedule =', Upstream_Release[curStep:curStep + forecastPeriod])
Storage = Previous_Stor
routedRelease = routeFlow('FTRA', Upstream_Release)
for step in range(lag * 2):
    if (curStep + step) > totalSteps:
        break
    currentRuntimestep.setStep(curStep + step) # Set currentRuntimestep equal to a future step
    GAPT_GC_Future_Elev = network.getTimeSeries("Reservoir", "Lewis and Clark Lake", "Guide Curve", "Elev-ZONE").getCurrentValue(currentRuntimestep) # Guide curve storage based on a future timestep
    currentRuntimestep.setStep(curStep) # Reset currentRuntimestep to the current step
    Low_Elev_Constraint = GAPT_GC_Future_Elev - 1.0
    High_Elev_Constraint = GAPT_GC_Future_Elev + 1.0
    Storage_Change = (routedRelease[curStep + step] + GAPT_Local[curStep + step] - Downstream_Release[curStep + step]) * CFS_to_ACRE_FT
    Storage = Storage + Storage_Change
    Elevation = GAPT_ElevStorTable.interpolate(Storage)
    outputDebug(currentRuntimestep, debug, 'FTRA Release Inflow = %.0f cfs' % routedRelease[curStep + step],
                '\tGAPT Local Inflow = %.0f cfs' % GAPT_Local[curStep + step], '\tGAPT Release = %.0f' % Downstream_Release[curStep + step],

```

```

\tStorage Change = %.0f acre-ft' % Storage_Change,
\n\t\t\t\tForecasted Elevation = %.2f ft' % Elevation, '\tGuide Curve = %.2f ft' % GAPT_GC_Future_Elev,
\n\t\t\t\tcurStep + step = %.0f' % (curStep + step), '\tcurStep + lag = %.0f' % (curStep + lag))
if Elevation > High_Elev_Constraint and (curStep + lag) <= (curStep + step) <= (curStep + lag + 1):
    outputDebug(currentRuntimestep, debug, 'Reduce FTRA releases by 5000 cfs')
    for step in range(lag):
        Upstream_Release[curStep + step] = Upstream_Release[curStep + step] - 5000
    currentRule.varPut('FTRAstepTracker', curStep + lag)
    break
elif Elevation < Low_Elev_Constraint and (curStep + lag) <= (curStep + step) <= (curStep + lag + 1):
    outputDebug(currentRuntimestep, debug, 'Increase FTRA releases by 5000 cfs')
    for step in range(lag):
        Upstream_Release[curStep + step] = Upstream_Release[curStep + step] + 5000
    currentRule.varPut('FTRAstepTracker', curStep + lag)
    break
else:
    currentRule.varPut('FTRAstepTracker', curStep + lag)
outputDebug(currentRuntimestep, debug, 'FTRA Release Schedule =', Upstream_Release[curStep:curStep + forecastPeriod])

# forecastOAHEFloodReleases Function: OAHE releases during flooding
def forecastOAHEFloodReleases( currentRuntimestep, # currentRuntimestep variable passed in through the runRuleScript()
    Upstream_Location, # DCP ID string of upstream dam (i.e. 'GARR')
    Location, # DCP ID string of dam where flood releases will be made (i.e. 'OAHE')
    Downstream_Location, # DCP ID string of downstream dam (i.e. 'FTRA')
    Lower_Flood_Elev_Constraint, # Lower flood elevation constraint at the current dam
    Upper_Flood_Elev_Constraint, # Upper flood elevation constraint at the current dam
    Down_Lower_Flood_Elev_Constraint, # Lower flood elevation constraint at the downstream dam
    Local_Flow, # Local flow data for current dam
    Elev_Stor_Table # Elev-Stor table for current dam
):
    outputDebug(currentRuntimestep, debug, 'FORECAST %s FLOOD RELEASES' % Location)
    curStep = currentRuntimestep.getStep() # Current step
    forecastPeriod = 7
    lag = len(coeffDict[Upstream_Location])
    Upstream_Reservoir = initInfo['projectDict'][Upstream_Location]
    Reservoir = initInfo['projectDict'][Location]
    Downstream_Reservoir = initInfo['projectDict'][Downstream_Location]
    Upstream_Releases = network.getTimeSeries("Reservoir", Upstream_Reservoir, "Pool", "Flow-OUT").getDataArrayFor(curStep - 1, -lag)
    Previous_Elev = network.getTimeSeries("Reservoir", Reservoir, "Pool", "Elev").getPreviousValue(currentRuntimestep)
    Previous_Stor = network.getTimeSeries("Reservoir", Reservoir, "Pool", "Stor").getPreviousValue(currentRuntimestep)
    Lagged_Elev = network.getTimeSeries("Reservoir", Reservoir, "Pool", "Elev").getLaggedValue(currentRuntimestep, 2)
    Downstream_Elev = network.getTimeSeries("Reservoir", Downstream_Reservoir, "Pool", "Elev").getPreviousValue(currentRuntimestep)
    Downstream_Release = network.getTimeSeries("Reservoir", Downstream_Reservoir, "Pool", "Flow-OUT").getPreviousValue(currentRuntimestep)
    Downstream_Release_Array = network.getTimeSeries("Reservoir", Downstream_Reservoir, "Pool", "Flow-OUT").getDataArrayFor(curStep - 1, -7)
    delta = 2000
    OAHE_Max_Power_Release = 55400
    OAHE_Max_Flood_Release = Max_Min_Release_Dict['OAHE']['Max_Release']

    # Set starting release at either the previous day's release or the previous release less 8000 cfs
    Avg_Downstream_Release = sum(Downstream_Release_Array) / len(Downstream_Release_Array)
    if Downstream_Elev < Down_Lower_Flood_Elev_Constraint:
        Release = network.getTimeSeries("Reservoir", Reservoir, "Pool", "Flow-OUT").getPreviousValue(currentRuntimestep)
        if Avg_Downstream_Release > Release:

```

```

        Release = Avg_Downstream_Release
        Flood_Elev_Constraint = Lower_Flood_Elev_Constraint
else:
    Release = Avg_Downstream_Release - 15000
    Flood_Elev_Constraint = Upper_Flood_Elev_Constraint

# Calculate releases to keep pool elevations below constraints
while Flood_Elev_Constraint <= Upper_Flood_Elev_Constraint:
    Routed_Upstream_Release = routeFlow(Upstream_Location, Upstream_Releases)
    Upstream_Release_Inflow = Routed_Upstream_Release[-1]
    Previous_Storage = Previous_Stor
    for step in range(forecastPeriod):
        Iteration_Counter = 0
        if (curStep + step + lag) > totalSteps:
            break
        # Forecast elevations during the forecast period
        Storage_Change = (Upstream_Release_Inflow + Local_Flow[curStep + step] - Release) * CFS_to_ACRE_FT
        Storage = Previous_Storage + Storage_Change
        Elevation = Elev_Stor_Table.interpolate(Storage)
        outputDebug(currentRuntimestep, debug, 'forecastStep =', curStep + step,
            '\n\t\t\t\tUpstream Release Inflow = %.0f cfs' % Upstream_Release_Inflow, '\tLocal Inflow = %.0f cfs' % Local_Flow[curStep + step],
            '\tRelease = %.0f' % Release, '\tStorage Change = %.0f acre-ft' % Storage_Change,
            '\n\t\t\t\tForecasted Elevation = %.2f ft' % Elevation)
        # Increase releases until constraints are met
        while Elevation >= Flood_Elev_Constraint:
            if Iteration_Counter > 1000:
                break
            Release += delta
            Storage_Change = (Upstream_Release_Inflow + Local_Flow[curStep + step] - Release) * CFS_to_ACRE_FT
            Storage = Previous_Storage + Storage_Change
            Elevation = Elev_Stor_Table.interpolate(Storage)
            outputDebug(currentRuntimestep, debug, 'Upstream Release Inflow = %.0f cfs' % Upstream_Release_Inflow, '\tLocal Inflow = %.0f cfs' % Local_Flow[curStep + step],
                '\tRelease = %.0f' % Release, '\tStorage Change = %.0f acre-ft' % Storage_Change,
                '\n\t\t\t\tForecasted Elevation = %.2f ft' % Elevation)
            Iteration_Counter += 1
            if Flood_Elev_Constraint < Upper_Flood_Elev_Constraint and Release > OAHE_Max_Power_Release:
                break
        Previous_Storage = Storage
        outputDebug(currentRuntimestep, debug, 'Forecast Elevation = %.2f' % Elevation, '\tFlood Constraint Elevation = %.2f' % Flood_Elev_Constraint)
        # If the constraint is met and releases are below maximum power house releases, break the loop
        if Release > OAHE_Max_Power_Release and Elevation > Flood_Elev_Constraint:
            break
        # If the constraint is met and releases are below maximum power house releases, break the loop. Otherwise, increase teh elevation constraint
        # until the maximum constraint is reached
        if Release <= OAHE_Max_Power_Release:
            break
    else:
        if Flood_Elev_Constraint < Upper_Flood_Elev_Constraint:
            Release = OAHE_Max_Power_Release
            Flood_Elev_Constraint += 1.0
# If release is greater than maximum, set to maximum
if Release > OAHE_Max_Flood_Release:
    Release = OAHE_Max_Flood_Release

```

```

currentRule.varPut('OAHEstepTracker', curStep + 1)
return Release

```

```

# forecastOAHEReleases Function: Forecast OAHE releases

```

```

def forecastOAHEReleases( currentRuntimestep,      # currentRuntimestep variable passed in through the runRuleScript()
                        lag,                      # Lag time between OAHE and FTRA
                        Upstream_Release,        # OAHE release
                        Downstream_Release,      # FTRA release
                        Previous_Stor           # FTRA previous storage
                        ):
    outputDebug(currentRuntimestep, debug, 'FORECAST OAHE RELEASES')
    curStep = currentRuntimestep.getStep() # Current step
    forecastPeriod = lag * 2 # Number of days to run the release forecast
    threshold = 6000 # Allowable difference between the guide curve storage and forecasted storage
    OAHE_Increase_ROC = 20000 # Daily increase ROC of change for OAHE releases
    OAHE_Decrease_ROC = -20000 # Daily decrease ROC of change for OAHE releases
    FTRA_Forecast_Previous_Stor = [None] * (lag * 3)
    FTRA_Forecast_Previous_Stor[0] = Previous_Stor
    FTRA_Previous_Elev = FTRA_ElevStorTable.interpolate(Previous_Stor)

    # Reset previous forecast data to original releases
    for step in range(forecastPeriod):
        if (curStep + step) > totalSteps or (curStep + step) > (curStep + forecastPeriod):
            break
        Upstream_Release[curStep + step] = Upstream_Release[curStep - 1 + step]

    # Calculate OAHE releases for a forecasted period so FTRA remains near guide curve
    for step in range(0, forecastPeriod, lag):
        Increase_Loop_Break, Decrease_Loop_Break, Max_Min_Loop_Break = 'No', 'No', 'No'
        Iteration_Counter = 0
        currentRuntimestep.setStep(curStep + step + lag - 1) # Set currentRuntimestep equal to a future step
        FTRA_GC_Future_Stor = network.getTimeSeries("Reservoir", "Lake Francis Case", "Guide Curve", "Stor-ZONE").getCurrentValue(currentRuntimestep) # Guide curve storage based on a future timestep
        currentRuntimestep.setStep(curStep) # Reset currentRuntimestep to the current step
        if FTRA_Previous_Elev > 1357.0:
            FTRA_GC_Future_Elev = FTRA_Previous_Elev - (0.3 * (step + lag - 1))
            FTRA_GC_Future_Stor = FTRA_StorElevTable.interpolate(FTRA_GC_Future_Elev)

    # Calculate initial storage error
    forecastStep = curStep + step
    if (forecastStep + lag) > totalSteps or (forecastStep + lag) > (curStep + forecastPeriod):
        break
    BEND_Local_Cut = BEND_Local[forecastStep - lag:forecastStep + lag]
    routedRelease = routeFlow('OAHE', Upstream_Release[forecastStep - lag:forecastStep + lag])
    BEND_Release = []
    for t in range(len(routedRelease)):
        BEND_Release.append(BEND_Local_Cut[t] + routedRelease[t])
    routedRelease = routeFlow('BEND', BEND_Release)
    Upstream_Release_Inflow = sum(routedRelease[lag:])
    Local_Inflow = sum(FTRA_Local[forecastStep:forecastStep + lag])
    Release = sum(FTRA_Specified_Release[forecastStep:forecastStep + lag])
    Storage_Change = (Local_Inflow + Upstream_Release_Inflow - Release) * CFS_to_ACRE_FT
    Downstream_Future_Storage = FTRA_Forecast_Previous_Stor[step] + Storage_Change
    error = Downstream_Future_Storage - FTRA_GC_Future_Stor

```

```

outputDebug(currentRuntimestep, False, 'Sum of Routed Upstream Release = %.0f cfs' % Upstream_Release_Inflow,
          '\n\t\t\tSum of Local Inflow = %.0f cfs' % Local_Inflow,
          '\n\t\t\tSum of FTRA Release = %.0f cfs' % Release,
          '\n\t\t\tFTRA Previous Stor = %.0f acre-ft' % FTRA_Forecast_Previous_Stor[step],
          '\n\t\t\tFTRA Guide Curve =', FTRA_GC_Future_Stor,
          '\n\t\t\tStorage Change = %.0f acre-ft' % Storage_Change,
          '\n\t\t\tStorage Error = %.0f acre-ft' % error)
delta = 5000 # Upstream release increment

# Optimize upstream releases to keep downstream reservoir at guide curve
while abs(error) > threshold:
    Iteration_Counter += 1
    if Iteration_Counter > 100:
        break
    if error < 0:
        # Increase the flow for the current and all future timesteps within the forecasted period
        while error < 0 and abs(error) > threshold:
            Iteration_Counter += 1
            if Iteration_Counter > 100:
                break
            Total_ROC_Increase, Total_ROC_Decrease, Total_Max_Min = 0, 0, 0
            for t in range(lag):
                Upstream_Release[forecastStep + t] = Upstream_Release[forecastStep + t] + delta
                Release_Change = Upstream_Release[forecastStep + t] - Upstream_Release[forecastStep + t - 1]
                # Check releases against max release and rate of change rules
                if Upstream_Release[forecastStep + t] >= Max_Min_Release_Dict['OAHE']['Max_Release']:
                    Upstream_Release[forecastStep + t] = Max_Min_Release_Dict['OAHE']['Max_Release']
                    Total_Max_Min += Max_Min_Release_Dict['OAHE']['Max_Release']
                if Release_Change >= OAHE_Increase_ROC:
                    Upstream_Release[forecastStep + t] = Upstream_Release[forecastStep + t - 1] + OAHE_Increase_ROC
                    Total_ROC_Increase += OAHE_Increase_ROC
                if Release_Change <= OAHE_Decrease_ROC:
                    Upstream_Release[forecastStep + t] = Upstream_Release[forecastStep + t - 1] + OAHE_Decrease_ROC
                    Total_ROC_Decrease += OAHE_Decrease_ROC
            # Route the flow from upstream to downstream and calculate new storage
            BEND_Local_Cut = BEND_Local[forecastStep - lag:forecastStep + lag]
            routedRelease = routeFlow('OAHE', Upstream_Release[forecastStep - lag:forecastStep + lag])
            BEND_Release = []
            for t in range(len(routedRelease)):
                BEND_Release.append(BEND_Local_Cut[t] + routedRelease[t])
            routedRelease = routeFlow('BEND', BEND_Release)
            Upstream_Release_Inflow = sum(routedRelease[lag:])
            Local_Inflow = sum(FTRA_Local[forecastStep:forecastStep + lag])
            Release = sum(FTRA_Specified_Release[forecastStep:forecastStep + lag])
            Storage_Change = (Local_Inflow + Upstream_Release_Inflow - Release) * CFS_to_ACRE_FT
            Downstream_Future_Storage = FTRA_Forecast_Previous_Stor[step] + Storage_Change
            error = Downstream_Future_Storage - FTRA_GC_Future_Stor
            outputDebug(currentRuntimestep, False, 'Sum of Routed Upstream Release = %.0f cfs' % Upstream_Release_Inflow,
                    '\n\t\t\tSum of Local Inflow = %.0f cfs' % Local_Inflow,
                    '\n\t\t\tSum of FTRA Release = %.0f cfs' % Release,
                    '\n\t\t\tFTRA Previous Stor = %.0f acre-ft' % FTRA_Forecast_Previous_Stor[step],
                    '\n\t\t\tFTRA Guide Curve =', FTRA_GC_Future_Stor,
                    '\n\t\t\tStorage Change = %.0f acre-ft' % Storage_Change,

```

```

        '\n\t\t\t\t\tStorage Error = %.0f acre-ft' % error)

    # If release is set to maximum release or exceeds ROC, break the loop
    if Total_Max_Min == (Max_Min_Release_Dict['OAHE']['Max_Release'] * lag) or Total_ROC_Increase == (OAHE_Increase_ROC * lag) \
        or Total_ROC_Decrease == (OAHE_Decrease_ROC * lag):
        break
    delta = delta / 3
elif error > 0:
    # Decrease the flow for the current and all future timesteps within the forecasted period
    while error > 0 and abs(error) > threshold:
        Iteration_Counter += 1
        if Iteration_Counter > 100:
            break
        Total_ROC_Increase, Total_ROC_Decrease, Total_Max_Min = 0, 0, 0
        for t in range(lag):
            Upstream_Release[forecastStep + t] = Upstream_Release[forecastStep + t] - delta
            Release_Change = Upstream_Release[forecastStep + t] - Upstream_Release[forecastStep + t - 1]
            # Check releases against min release and rate of change rules
            if Upstream_Release[forecastStep + t] <= Max_Min_Release_Dict['OAHE']['Min_Release']:
                Upstream_Release[forecastStep + t] = Max_Min_Release_Dict['OAHE']['Min_Release']
                Total_Max_Min += Max_Min_Release_Dict['OAHE']['Min_Release']
            if Release_Change >= OAHE_Increase_ROC:
                Upstream_Release[forecastStep + t] = Upstream_Release[forecastStep + t - 1] + OAHE_Increase_ROC
                Total_ROC_Increase += OAHE_Increase_ROC
            if Release_Change <= OAHE_Decrease_ROC:
                Upstream_Release[forecastStep + t] = Upstream_Release[forecastStep + t - 1] + OAHE_Decrease_ROC
                Total_ROC_Decrease += OAHE_Decrease_ROC
        # Route the flow from upstream to downstream and calculate new storage
        BEND_Local_Cut = BEND_Local[forecastStep - lag:forecastStep + lag]
        routedRelease = routeFlow('OAHE', Upstream_Release[forecastStep - lag:forecastStep + lag])
        BEND_Release = []
        for t in range(len(routedRelease)):
            BEND_Release.append(BEND_Local_Cut[t] + routedRelease[t])
        routedRelease = routeFlow('BEND', BEND_Release)
        Upstream_Release_Inflow = sum(routedRelease[lag:])
        Local_Inflow = sum(FTRA_Local[forecastStep:forecastStep + lag])
        Release = sum(FTRA_Specified_Release[forecastStep:forecastStep + lag])
        Storage_Change = (Local_Inflow + Upstream_Release_Inflow - Release) * CFS_to_ACRE_FT
        Downstream_Future_Storage = FTRA_Forecast_Previous_Stor[step] + Storage_Change
        error = Downstream_Future_Storage - FTRA_GC_Future_Stor
        outputDebug(currentRuntimestep, False, 'Sum of Routed Upstream Release = %.0f cfs' % Upstream_Release_Inflow,
            '\n\t\t\t\t\tSum of Local Inflow = %.0f cfs' % Local_Inflow,
            '\n\t\t\t\t\tSum of FTRA Release = %.0f cfs' % Release,
            '\n\t\t\t\t\tFTRA Previous Stor = %.0f acre-ft' % FTRA_Forecast_Previous_Stor[step],
            '\n\t\t\t\t\tFTRA Guide Curve =', FTRA_GC_Future_Stor,
            '\n\t\t\t\t\tStorage Change = %.0f acre-ft' % Storage_Change,
            '\n\t\t\t\t\tStorage Error = %.0f acre-ft' % error)

    # If release is set to minimum release or exceeds ROC, break the loop
    if Total_Max_Min == (Max_Min_Release_Dict['OAHE']['Min_Release'] * lag) or Total_ROC_Increase == (OAHE_Increase_ROC * lag) \
        or Total_ROC_Decrease == (OAHE_Decrease_ROC * lag):
        break
    delta = delta / 3

```

```

        # If release is set to max/min release or exceeds ROC rules, break the loop
        if Total_Max_Min == (Max_Min_Release_Dict['OAHE']['Max_Release'] * lag) \
            or Total_Max_Min == (Max_Min_Release_Dict['OAHE']['Min_Release'] * lag) \
            or Total_ROC_Increase == (OAHE_Increase_ROC * lag) or Total_ROC_Decrease == (OAHE_Decrease_ROC * lag):
            break
        outputDebug(currentRuntimestep, debug, 'Forecasted Elevation = %.2f ft' % FTRA_ElevStorTable.interpolate(Downstream_Future_Storage),
            '\tForecasted GC Elevation = %.2f ft' % FTRA_ElevStorTable.interpolate(FTRA_GC_Future_Stor))
        FTRA_Forecast_Previous_Stor[step + lag] = Downstream_Future_Storage
        outputDebug(currentRuntimestep, debug, 'OAHE Release Schedule =', Upstream_Release[curStep:curStep + lag])
        currentRule.varPut('OAHEstepTracker', curStep + len(coeffDict['OAHE']) + len(coeffDict['BEND']))

```

# julianDay Function: Get the julian day for a month and day or a string, or a month and day for a julian day

```

def julianDay(*args) :
    #   J F M A M J J A S O N D
    daysPrev = 0, 31, 59, 90,120,151,181,212,243,273,304,334
    argCount = len(args)
    if argCount == 1 :
        if type(args[0]) == type(0) :
            jday = args[0]
            if not 1 <= jday <= 365 :
                raise ValueError("Julian day out of range 1..365")
            for i in range(12)[::-1] :
                if jday > daysPrev[i] :
                    mon = i + 1
                    break
            day = jday - daysPrev[mon-1]
            return mon, day
        else :
            mon, day = args[0].split("_")
            mon = ["JAN", "FEB", "MAR", "APR", "MAY", "JUN", "JUL", "AUG", "SEP", "OCT", "NOV", "DEC"].index(mon.upper()) + 1
            day = int(day)
            return julianDay(mon, day)
    elif argCount == 2 :
        mon, day = args
        if not 1 <= mon <= 12 :
            raise ValueError("Month is out of range 1..12")
        if not 1 <= day <= 31 :
            raise ValueError("Day is out of range 1..31")
        if mon == 2 and day == 29 : day = 28
        return daysPrev[mon-1] + day
    else :
        raise ValueError("Expected 1 or 2 arguments, got %d" % argCount)

```

# Adjust FTRA guide curve so drawdown coincides with the end of navigation season

```

def navEndFTRAGC( currentRuntimestep, # currentRuntimestep variable passed in through the runRuleScript()
                  FTRA_StorElevTable, # Storage-elevation table for FTRA
                  Nav_End_Date,      # Navigation end date in julian days
                  ):
    outputDebug(currentRuntimestep, debug, 'Shifting FTRA guide curve to coincide with end of navigation season.')

    rtw = currentRuntimestep.getRunTimeWindow() # Run time window
    totalSteps = rtw.getNumSteps() # Total number of steps in the simulation
    curStep = currentRuntimestep.getStep() # Current step

```

```

# Get the guide curve for the active operations set
Reservoir_Element = network.findReservoir('Lake Francis Case')
Reservoir_Operations = Reservoir_Element.getReservoirOp()
Operation_Set = Reservoir_Operations.getActiveOpSet()
Guide_Curve_Zone_Name = Operation_Set.getGuideCurveZone()._name

# Get the doubleArrayContainers that hold the time series for the guide curve zone
gcElevTS = network.getTimeSeries("Reservoir", 'Lake Francis Case', Guide_Curve_Zone_Name, "Elev-ZONE").getTSContainer()
gcStorTS = network.getTimeSeries("Reservoir", 'Lake Francis Case', Guide_Curve_Zone_Name, "Stor-ZONE").getTSContainer()

Total_Drawdown_Days = Nav_End_Date - julianDay('SEP_01')
Total_Increase_Days = julianDay('MAR_01') + (julianDay('DEC_31') - Nav_End_Date)
Total_Days = Total_Drawdown_Days + Total_Increase_Days

Daily_Decrease = (1355.0 - 1337.5) / Total_Drawdown_Days
Daily_Increase = (1350.0 - 1337.5) / Total_Increase_Days

for d in range(Total_Days):
    currentRuntimestep.setStep(curStep + d) # Set currentRuntimestep equal to a future step

    if currentRuntimestep.getStep() > totalSteps:
        outputDebug(currentRuntimestep, debug, 'Break the for loop. Total steps = %.0f' % totalSteps, '\tCurrent step = %.0f' % currentRuntimestep.getStep())
        currentRuntimestep.setStep(curStep) # Reset currentRuntimestep to the current step
        # Break loop if currentRuntimestep is greater than end of simulation
        break

    if d < Total_Drawdown_Days:
        # Reduce guide curve so drawdown coincides with end of navigation
        FTRA_GC_Elev = 1355.0 - (d * Daily_Decrease)
        FTRA_GC_Stor = FTRA_StorElevTable.interpolate(FTRA_GC_Elev)

        # Set the Guide Curve Zone Elevation and Storage Values
        gcElevTS.array[currentRuntimestep.step] = FTRA_GC_Elev
        gcStorTS.array[currentRuntimestep.step] = FTRA_GC_Stor

        outputDebug(currentRuntimestep, debug, 'Reduce FTRA guide curve to %.2f ft' % FTRA_GC_Elev, '\tCurrent step =', (currentRuntimestep.step))
    else:
        # Increase guide curve so initial guide curve is still reached by Mar 01
        FTRA_GC_Elev = 1337.5 + ((d - Total_Drawdown_Days) * Daily_Increase)
        FTRA_GC_Stor = FTRA_StorElevTable.interpolate(FTRA_GC_Elev)

        # Set the Guide Curve Zone Elevation and Storage Values
        gcElevTS.array[currentRuntimestep.step] = FTRA_GC_Elev
        gcStorTS.array[currentRuntimestep.step] = FTRA_GC_Stor

        outputDebug(currentRuntimestep, debug, 'Increase FTRA guide curve to %.2f ft' % FTRA_GC_Elev)

    currentRuntimestep.setStep(curStep) # Reset currentRuntimestep to the current step

# retrieveModelVariables Function: Retrieve model variables needed for the script
def retrieveModelVariables(currentRuntimestep):
    Model_Variable = { # GAPT Model Variables

```

```

'GAPT_Previous_Evap' : network.getTimeSeries("Reservoir", "Lewis and Clark Lake", "Pool", "Flow-EVAP").getPreviousValue(currentRuntimestep),
'GAPT_Previous_Release' : network.getTimeSeries("Reservoir", "Lewis and Clark Lake", "Pool", "Flow-OUT").getPreviousValue(currentRuntimestep),
'GAPT_Previous_Stor' : network.getTimeSeries("Reservoir", "Lewis and Clark Lake", "Pool", "Stor").getPreviousValue(currentRuntimestep),
'GAPT_Previous_Elev' : network.getTimeSeries("Reservoir", "Lewis and Clark Lake", "Pool", "Elev").getPreviousValue(currentRuntimestep),
# FTRA Model Variables
'FTRA_Previous_Evap' : network.getTimeSeries("Reservoir", "Lake Francis Case", "Pool", "Flow-EVAP").getPreviousValue(currentRuntimestep),
'FTRA_Previous_Elev' : network.getTimeSeries("Reservoir", "Lake Francis Case", "Pool", "Elev").getPreviousValue(currentRuntimestep),
'FTRA_Previous_Stor' : network.getTimeSeries("Reservoir", "Lake Francis Case", "Pool", "Stor").getPreviousValue(currentRuntimestep),
'FTRA_Previous_Release' : network.getTimeSeries("Reservoir", "Lake Francis Case", "Pool", "Flow-OUT").getPreviousValue(currentRuntimestep),
'FTRA_Flood_and_Multiple_Use_Elev' : network.getTimeSeries("Reservoir", "Lake Francis Case", "Flood Control and Multiple Use", "Elev-ZONE").getPreviousValue(currentRuntimestep),
# BEND Model Variables
'BEND_Previous_Evap' : network.getTimeSeries("Reservoir", "Lake Sharpe", "Pool", "Flow-EVAP").getPreviousValue(currentRuntimestep),
'BEND_Previous_Elev' : network.getTimeSeries("Reservoir", "Lake Sharpe", "Pool", "Elev").getPreviousValue(currentRuntimestep),
'BEND_Previous_Stor' : network.getTimeSeries("Reservoir", "Lake Sharpe", "Pool", "Stor").getPreviousValue(currentRuntimestep),
'BEND_Previous_Release' : network.getTimeSeries("Reservoir", "Lake Sharpe", "Pool", "Flow-OUT").getPreviousValue(currentRuntimestep),
'BEND_GC_Previous_Elev' : network.getTimeSeries("Reservoir", "Lake Sharpe", "Guide Curve", "Elev-ZONE").getPreviousValue(currentRuntimestep),
'BEND_GC_Previous_Stor' : network.getTimeSeries("Reservoir", "Lake Sharpe", "Guide Curve", "Stor-ZONE").getPreviousValue(currentRuntimestep),
# OAHE Model Variables
'OAHE_Previous_Elev' : network.getTimeSeries("Reservoir", "Lake Oahe", "Pool", "Elev").getPreviousValue(currentRuntimestep),
'OAHE_Previous_Release' : network.getTimeSeries("Reservoir", "Lake Oahe", "Pool", "Flow-OUT").getPreviousValue(currentRuntimestep),
# State Variables
'Nav_End_Date' : network.getStateVariable("NavigationEndDate").getPreviousValue(currentRuntimestep)
}

return Model_Variable

# routeFlow Function: Flow routing function
def routeFlow(Upstream_Location, Flow_Dataset):
    #Downstream_Location = locationList[locationList.index(Upstream_Location) + 1]
    coeffList = coeffDict[Upstream_Location]
    lag = len(coeffList) - 1 # Length of time before any portion of flow reaches the downstream location
    routedFlow = []
    for i in range(len(Flow_Dataset)):
        if i < lag:
            routedFlow.append(0) # Add a place holder of 0 for days less than the lag time
        else:
            routedFlowTempList = []
            for c in range(len(coeffList)):
                flow = Flow_Dataset[i-c] * coeffList[c]
                routedFlowTempList.append(flow)
            release = sum(routedFlowTempList)
            routedFlow.append(release)
    return routedFlow

# -----
# Input Data
# -----

# Simulation Time & Step Info
rtw = ResSim.getCurrentModule().getSimulation().getRunTimeWindow()
totalSteps = rtw.getNumSteps() / 24 # Total number of steps in the simulation. rtw gives steps in hours so divide by 24 to get days
startRtw = rtw.getStartTime() # Start time of simulation
lookbackRtw = rtw.getLookbackTime() # Start of lookback period
endRtw = rtw.getEndTime() # End time of simulation

```

```

numLookbackSteps = rtw.getNumLookbackSteps() / 24 # Number of lookback steps in simulation. rtw gives steps in hours so divide by 24 to get days
simStartStep = numLookbackSteps + 1 # First step after the lookback period

initOutputDebug(debug, 'Run Time Window Start Time =', rtw.getStartTimeString(), '\tRun Time Window End Time =', rtw.getEndTimeString(), \
    '\tNumber of Lookback Steps =', numLookbackSteps, '\tStart of Simulation Step =', simStartStep, '\tTotal Number of Steps =', totalSteps)

# DSS Data
shared = ResSim.getWatershed()
module = ClientAppWrapper.getCurrentModule()
if module.getName() != 'Simulation':
    raise AssertionError, 'ResSim in %s module. Simulation module is required.' % module
shared = str(shared)
filePath = shared + '/shared/Input_Data.dss'
filePath = filePath.replace('/', '\\')
dssFile = HecDss.open(filePath)
dssFile.setTimeWindow(str(lookbackRtw), str(endRtw)) # Open dss file with a time window equal to the run time window in ResSim

initOutputDebug(debug, 'DSS File =', filePath, '\tDSS Time Window: %s - %s' % (str(lookbackRtw), str(endRtw)))

GAPT_Specified_Release = dssFile.read('/MISSOURI RIVER/GAPT/FLOW-OUT//1DAY/%s: FORECAST RELEASE/' % Alt_Name).getData().values
if Depletions == 'Present':
    GAPT_Local = dssFile.read('/MISSOURI RIVER/GAPT/FLOW-LOCAL//1DAY/FWOP: CALCULATED/').getData().values
    FTRA_Local = dssFile.read('/MISSOURI RIVER/FTRA/FLOW-LOCAL//1DAY/FWOP: CALCULATED/').getData().values
    BEND_Local = dssFile.read('/MISSOURI RIVER/BEND/FLOW-LOCAL//1DAY/FWOP: CALCULATED/').getData().values
    OAHE_Local = dssFile.read('/MISSOURI RIVER/OAHE/FLOW-LOCAL//1DAY/FWOP: CALCULATED/').getData().values
elif Depletions == 'Historic':
    GAPT_Local = dssFile.read('/MISSOURI RIVER/GAPT/FLOW-LOCAL//1DAY/HISTORIC: CALCULATED/').getData().values
    FTRA_Local = dssFile.read('/MISSOURI RIVER/FTRA/FLOW-LOCAL//1DAY/HISTORIC: CALCULATED/').getData().values
    BEND_Local = dssFile.read('/MISSOURI RIVER/BEND/FLOW-LOCAL//1DAY/HISTORIC: CALCULATED/').getData().values
    OAHE_Local = dssFile.read('/MISSOURI RIVER/OAHE/FLOW-LOCAL//1DAY/HISTORIC: CALCULATED/').getData().values
else:
    raise AssertionError, 'Depletions error. Specify as either Present or Historic.'

# Routing locations
locationList = ['OAHE', 'BEND', 'FTRA', 'GAPT']

# Project Dictionary
projectDict = {
    'FTPK' : 'Fort Peck Lake',
    'GARR' : 'Lake Sakakawea',
    'OAHE' : 'Lake Oahe',
    'BEND' : 'Lake Sharpe',
    'FTRA' : 'Lake Francis Case',
    'GAPT' : 'Lewis and Clark Lake'
}

# Routing Coefficients
coeffDict = {
    'GARR': [0.057, 0.503, 0.440], # GARR to OAHE coefficients
    'OAHE': [0.766, 0.234], # OAHE to BEND coefficients
    'BEND': [0.647, 0.353], # BEND to FTRA coefficients
    'FTRA': [0.005, 0.637, 0.358] # FTRA to GAPT coefficients
}

# Max and Min Releases

```

```

Max_Min_Release_Dict = {
    'FTRA': {
        'Min_Release': 100,
        'Max_Release': 160000
    },
    'BEND': {
        'Min_Release': BEND_Local,
        'Max_Release': 166000
    },
    'OAHE': {
        'Min_Release': 1000,
        'Max_Release': 150000
    }
}

```

```

# Conversions
CFS_to_ACRE_FT = 86400.0/43560.0
ACRE_FT_to_CFS = 43560.0/86400.0

```

```

# -----
# Initial forecast releases
# -----

```

```

Initial_Upstream_Release = []

```

```

for step in range(len(GAPT_Specified_Release)):
    Initial_Upstream_Release.append(25000)

```

```

# Initial Upstream Releases
FTRA_Specified_Release = Initial_Upstream_Release[:]
BEND_Specified_Release = Initial_Upstream_Release[:]
OAHE_Specified_Release = Initial_Upstream_Release[:]

```

```

# -----
# Elevation-Storage Relationships
# -----

```

```

# OAHE Elev-Stor
reservoirName = 'Lake Oahe'
reservoirElement = network.findReservoir(reservoirName)
storageElement = reservoirElement.getStorageFunction()
storageTable = storageElement.getElevationStorageValues()
elevValues = storageTable.getXArray()
storValues = storageTable.getYArray()
OAHE_StorElevTable, OAHE_ElevStorTable = PairedValuesExt(), PairedValuesExt()
OAHE_StorElevTable.setArrays(elevValues, storValues)
OAHE_ElevStorTable.setArrays(storValues, elevValues)

```

```

# FTRA Elev-Stor
reservoirName = 'Lake Francis Case'
reservoirElement = network.findReservoir(reservoirName)
storageElement = reservoirElement.getStorageFunction()
storageTable = storageElement.getElevationStorageValues()
elevValues = storageTable.getXArray()
storValues = storageTable.getYArray()
FTRA_StorElevTable, FTRA_ElevStorTable = PairedValuesExt(), PairedValuesExt()
FTRA_StorElevTable.setArrays(elevValues, storValues)

```

```

FTRA_ElevStorTable.setArrays(storValues, elevValues)

# GAPT Elev-Stor
reservoirName = 'Lewis and Clark Lake'
reservoirElement = network.findReservoir(reservoirName)
storageElement = reservoirElement.getStorageFunction()
storageTable = storageElement.getElevationStorageValues()
elevValues = storageTable.getXArray()
storValues = storageTable.getYArray()
GAPT_StorElevTable, GAPT_ElevStorTable = PairedValuesExt(), PairedValuesExt()
GAPT_StorElevTable.setArrays(elevValues, storValues)
GAPT_ElevStorTable.setArrays(storValues, elevValues)

```

```

# -----
# init info that is passed to runRuleScript()
# -----

```

```

initInfo = {
    'ACRE_FT_to_CFS'           : ACRE_FT_to_CFS,
    'adjustFallRelease'       : adjustFallRelease,
    'adjustWinterRelease'     : adjustWinterRelease,
    'Alt_Name'                : Alt_Name,
    'BEND_Local'              : BEND_Local,
    'BEND_Specified_Release'  : BEND_Specified_Release,
    'CFS_to_ACRE_FT'         : CFS_to_ACRE_FT,
    'coeffDict'               : coeffDict,
    'debug'                   : debug,
    'forecastBENDReleases'    : forecastBENDReleases,
    'forecastOAHEFloodReleases' : forecastOAHEFloodReleases,
    'forecastFTRARReleases'   : forecastFTRARReleases,
    'forecastOAHEReleases'    : forecastOAHEReleases,
    'FTRA_ElevStorTable'      : FTRA_ElevStorTable,
    'FTRA_Local'              : FTRA_Local,
    'FTRA_Specified_Release'  : FTRA_Specified_Release,
    'FTRA_StorElevTable'      : FTRA_StorElevTable,
    'GAPT_ElevStorTable'      : GAPT_ElevStorTable,
    'GAPT_Local'              : GAPT_Local,
    'GAPT_Specified_Release'  : GAPT_Specified_Release,
    'GAPT_StorElevTable'      : GAPT_StorElevTable,
    'julianDay'               : julianDay,
    'locationList'            : locationList,
    'Max_Min_Release_Dict'    : Max_Min_Release_Dict,
    'navEndFTRAGC'            : navEndFTRAGC,
    'OAHE_ElevStorTable'      : OAHE_ElevStorTable,
    'OAHE_Local'              : OAHE_Local,
    'OAHE_Specified_Release'  : OAHE_Specified_Release,
    'OAHE_StorElevTable'      : OAHE_StorElevTable,
    'outputDebug'             : outputDebug,
    'projectDict'              : projectDict,
    'retrieveModelVariables'  : retrieveModelVariables,
    'routeFlow'               : routeFlow
}

```

```

currentRule.varPut('initInfo', initInfo)

# return Constants.TRUE if the initialization is successful
# and Constants.FALSE if it failed. Returning Constants.FALSE
# will halt the compute.
return Constants.TRUE

```

```

# runRuleScript() is the entry point that is called during the
# compute.
#
# currentRule is the rule that holds this script
# network is the ResSim network
# currentRuntimestep is the current Run Time Step

```

```

def runRuleScript(currentRule, network, currentRuntimestep):
    # create new Operation Value (OpValue) to return
    opValue = OpValue()

```

```

# -----
# Retrieve init info from initRuleScript() and set equal to variables
# -----

```

```

initInfo = currentRule.varGet('initInfo')

```

```

ACRE_FT_to_CFS           = initInfo['ACRE_FT_to_CFS']
adjustFallRelease        = initInfo['adjustFallRelease']
adjustWinterRelease      = initInfo['adjustWinterRelease']
BEND_Local               = initInfo['BEND_Local']
BEND_Specified_Release   = initInfo['BEND_Specified_Release']
CFS_to_ACRE_FT           = initInfo['CFS_to_ACRE_FT']
coeffDict                = initInfo['coeffDict']
debug                    = initInfo['debug']
forecastBENDReleases     = initInfo['forecastBENDReleases']
forecastOAHEFloodReleases = initInfo['forecastOAHEFloodReleases']
forecastFTRARReleases    = initInfo['forecastFTRARReleases']
forecastOAHERReleases    = initInfo['forecastOAHERReleases']
FTRA_ElevStorTable       = initInfo['FTRA_ElevStorTable']
FTRA_Local               = initInfo['FTRA_Local']
FTRA_Specified_Release   = initInfo['FTRA_Specified_Release']
FTRA_StorElevTable       = initInfo['FTRA_StorElevTable']
GAPT_ElevStorTable       = initInfo['GAPT_ElevStorTable']
GAPT_Local               = initInfo['GAPT_Local']
GAPT_Specified_Release   = initInfo['GAPT_Specified_Release']
GAPT_StorElevTable       = initInfo['GAPT_StorElevTable']
julianDay                 = initInfo['julianDay']
locationList              = initInfo['locationList']
Max_Min_Release_Dict     = initInfo['Max_Min_Release_Dict']
navEndFTRAGC             = initInfo['navEndFTRAGC']
OAHE_ElevStorTable       = initInfo['OAHE_ElevStorTable']
OAHE_Local               = initInfo['OAHE_Local']
OAHE_StorElevTable       = initInfo['OAHE_StorElevTable']
OAHE_Specified_Release   = initInfo['OAHE_Specified_Release']
outputDebug              = initInfo['outputDebug']

```

```

retrieveModelVariables      = initInfo['retrieveModelVariables']
routeFlow                   = initInfo['routeFlow']

# -----
# Input Data
# -----

# Simulation Time & Step Info
rtw = currentRuntimestep.getRuntimeWindow() # Run time window
startRtw = rtw.getStartTime() # Start time of simulation
lookbackRtw = rtw.getLookbackTime() # Start of lookback period
endRtw = rtw.getEndTime() # End time of simulation
endRtwString = rtw.getEndTimeString() # String of end time of simulation
totalSteps = rtw.getNumSteps() # Total number of steps in the simulation
curStep = currentRuntimestep.getStep() # Current step
year = currentRuntimestep.getHecTime().year() # Current year
mon = currentRuntimestep.getHecTime().month() # Current month
day = currentRuntimestep.getHecTime().day() # Current day
jDay = julianDay(mon, day) # Current julian day
numLookbackSteps = rtw.getNumLookbackSteps() # Number of lookback steps in simulation
simStartStep = numLookbackSteps + 1 # First step after the lookback period
PassCounter = network.getComputePassCounter() # Pass number of simulation
PassNumber = PassCounter + 1 # Pass number starts at 0 so add 1 to start it at 1

if curStep == simStartStep:
    outputDebug(currentRuntimestep, debug, 'Lookback Start Time =', rtw.getLookbackTimeString(), '\tRun Time Window Start Time =', rtw.getStartTimeString(), \
        '\tRun Time Window End Time =', rtw.getEndTimeString(), '\tTotal Number of Steps =', totalSteps)

# -----
# Main Script
# -----

if PassNumber < 3:
    # Get specified release state variables and set equal to initial specified release values for Pass 1
    network.getStateVariable("FTRA_Specified_Release").setValue(currentRuntimestep, FTRA_Specified_Release[curStep])
    network.getStateVariable("BEND_Specified_Release").setValue(currentRuntimestep, BEND_Specified_Release[curStep])
    network.getStateVariable("OAHE_Specified_Release").setValue(currentRuntimestep, OAHE_Specified_Release[curStep])
    # opValue equal to OAHE release
    opValue.init(OpRule.RULETYPE_SPEC, OAHE_Specified_Release[curStep])
else:
    # -----
    # Retrieve model variables needed for the script in a dictionary
    # -----

    Model_Variable = retrieveModelVariables(currentRuntimestep)

    # -----
    # Initialize and retrieve step tracker variables
    # -----

    if curStep == simStartStep: # Start the forecast release adjustments after 100% of upstream releases have reached the downstream location
        currentRule.varPut('FTRAstpTracker', curStep) # stepTracker will keep track of when the guide curve release adjustments will occur
        currentRule.varPut('BENDstpTracker', curStep) # stepTracker will keep track of when the guide curve release adjustments will occur

```

```

    currentRule.varPut('OAHEstepTracker', curStep) # stepTracker will keep track of when the guide curve release adjustments will occur
FTRAstepTracker = currentRule.varGet('FTRAstepTracker')
BENDstepTracker = currentRule.varGet('BENDstepTracker')
OAHEstepTracker = currentRule.varGet('OAHEstepTracker')

# -----
# If calculated releases were not released, insert actual releases into data
# -----

# Since there are rules in ResSim that can cause the specified release to not be released, first save the previous specified release to memory
# and then replace the specified release with ResSim previous release
if curStep > simStartStep:
    FTRA_Specified_Release[curStep - 1] = Model_Variable['FTRA_Previous_Release']
    BEND_Specified_Release[curStep - 1] = Model_Variable['BEND_Previous_Release']
    OAHE_Specified_Release[curStep - 1] = Model_Variable['OAHE_Previous_Release']

# -----
# Adjust FTRA guide curve so drawdown coincides with the end of navigation season
# -----

if jDay == julianDay('SEP_01') and Model_Variable['Nav_End_Date'] != julianDay('DEC_01'):
    navEndFTRAGC(currentRuntimestep, FTRA_StorElevTable, Model_Variable['Nav_End_Date'])

# -----
# Forecast FTRA, BEND, and OAHE Releases
# -----

# FTRA Releases
if curStep == FTRAstepTracker and curStep < (totalSteps - len(coeffDict['FTRA'])):
    # At times when FTRA is near the top of exclusive flood control zone, increase GAPT releases
    if Model_Variable['FTRA_Previous_Elev'] > Model_Variable['FTRA_Flood_and_Multiple_Use_Elev']:
        for step in range(len(coeffDict['FTRA'])):
            if GAPT_Specified_Release[curStep + step] < 35000:
                GAPT_Specified_Release[curStep + step] = GAPT_Specified_Release[curStep + step] + 15000
                GAPT_Release_Change = currentRule.varGet('GAPT_Release_Change')
                GAPT_Release_Change.append(-15000)
                currentRule.varPut('GAPT_Release_Change', GAPT_Release_Change)
            outputDebug(currentRuntimestep, debug, 'Increase GAPT releases because FTRA is full.')

    forecastFTRAReleases(currentRuntimestep, len(coeffDict['FTRA']), FTRA_Specified_Release, GAPT_Specified_Release, Model_Variable['GAPT_Previous_Stor'])

# OAHE releases
if curStep == OAHEstepTracker and curStep < (totalSteps - len(coeffDict['OAHE']) - len(coeffDict['BEND'])):
    forecastOAHEReleases(currentRuntimestep, (len(coeffDict['OAHE']) + len(coeffDict['BEND'])), OAHE_Specified_Release, FTRA_Specified_Release,
        Model_Variable['FTRA_Previous_Stor'])

# When pool elevation is above flood release elevation, ignore guide curve operations and release enough to
# keep pool below specified threshold
OAHE_Flood_Release_Elevation = 1616.0
if Model_Variable['OAHE_Previous_Elev'] > OAHE_Flood_Release_Elevation:
    OAHE_Flood_Release = forecastOAHEFloodReleases(currentRuntimestep, 'GARR', 'OAHE', 'FTRA', 1618.5, 1619.5, 1369.0, OAHE_Local, OAHE_ElevStorTable)
    # Set OAHE specified release equal to OAHE flood release if the flood release is greater
    outputDebug(currentRuntimestep, debug, 'OAHE Flood Release = %.0f cfs' % OAHE_Flood_Release, '\tOAHE Specified Release = %.0f cfs' % OAHE_Specified_Release[curStep])

```

```

    if OAHE_Flood_Release > OAHE_Specified_Release[curStep]:
        OAHE_Specified_Release[curStep] = OAHE_Flood_Release

# BEND releases
BEND_Current_Specified_Release = forecastBENDReleases(curStep, 'OAHE', OAHE_Specified_Release, BEND_Local, Model_Variable['BEND_Previous_Elev'], Model_Variable['BEND_Previous_Stor'],
    Model_Variable['BEND_GC_Previous_Stor'], Model_Variable['BEND_GC_Previous_Elev'])

# If FTRA pool elevation exceeds 1374.7, start increasing FTRA's releases to keep FTRA from having uncontrolled releases
if Model_Variable['FTRA_Previous_Elev'] > 1374.7:
    FTRA_Specified_Release[curStep] = BEND_Current_Specified_Release + FTRA_Local[curStep] + 5000
# If GAPT pool elevation exceeds 1210.0, increase GAPT's releases for the next 5 days to keep GAPT from having uncontrolled releases
if Model_Variable['GAPT_Previous_Elev'] > 1210.0:
    for day in range(6):
        GAPT_Specified_Release[curStep + day] = GAPT_Specified_Release[curStep] + 1500

# -----
# Track difference between System model GAPT release and Downstream
# model GAPT release and apply differences to the fall and winter releases
# -----

# Set the Winter_Release_Variable to 0 and set GAPT_Release_Change to [] on the first timestep
if curStep == simStartStep:
    currentRule.varPut('Winter_Release_Adjust', 0)
    currentRule.varPut('GAPT_Release_Change', [])
    # Set GAPT current release
    GAPT_Current_Specified_Release = GAPT_Specified_Release[curStep]
else:
    # Only calculate difference in specified release and ResSim release after Mar 15 so the winter release adjustments do not get counted
    GAPT_Release_Change = currentRule.varGet('GAPT_Release_Change')
    GAPT_Release_Difference = GAPT_Specified_Release[curStep - 1] - Model_Variable['GAPT_Previous_Release']
    GAPT_Release_Change.append(GAPT_Release_Difference)
    if GAPT_Release_Difference != 0:
        outputDebug(currentRuntimestep, debug, 'GAPT release difference = %.2f. Total GAPT release change = %.2f' % (GAPT_Release_Difference, sum(GAPT_Release_Change)))
        currentRule.varPut('GAPT_Release_Change', GAPT_Release_Change)

# Determine the julian day of the last step in the simulation
currentRuntimestep.setStep(totalSteps)
End_Day = currentRuntimestep.getHecTime().day()
End_Mon = currentRuntimestep.getHecTime().month()
End_Year = currentRuntimestep.getHecTime().year()
currentRuntimestep.setStep(curStep)
End_rtw_JulianDay = julianDay(End_Mon, End_Day)

# Calculate and apply the change in releases to fall releases if evacuating flood water
if jDay == julianDay('SEP_01'):
    adjustFallRelease(currentRuntimestep, jDay, curStep, currentRule.varGet('GAPT_Release_Change'))
    outputDebug(currentRuntimestep, debug, 'Release change after fall adjustment = %.0f' % sum(GAPT_Release_Change))
# Calculate and apply the change in releases to the winter releases
if jDay == julianDay('DEC_15'):
    adjustWinterRelease(currentRuntimestep, jDay, curStep, year, End_Year, End_rtw_JulianDay, currentRule.varGet('GAPT_Release_Change'))

# Set GAPT current release
GAPT_Current_Specified_Release = GAPT_Specified_Release[curStep]

```

```

# -----
# Adjust releases for evap
# -----

if curStep == simStartStep:
    FTRA_Current_Specified_Release = FTRA_Specified_Release[curStep]
    OAHE_Current_Specified_Release = OAHE_Specified_Release[curStep]
elif curStep > simStartStep:
    FTRA_Current_Specified_Release = FTRA_Specified_Release[curStep] + Model_Variable['GAPT_Previous_Evap']
    OAHE_Current_Specified_Release = OAHE_Specified_Release[curStep] + Model_Variable['BEND_Previous_Evap'] + Model_Variable['FTRA_Previous_Evap']
    # Save adjusted release to specified release array
    FTRA_Specified_Release[curStep] = FTRA_Current_Specified_Release
    OAHE_Specified_Release[curStep] = OAHE_Current_Specified_Release

# -----
# Write releases to state variables and opValue
# -----

network.getStateVariable("GAPT_Specified_Release").setValue(currentRuntimestep, GAPT_Current_Specified_Release)
network.getStateVariable("FTRA_Specified_Release").setValue(currentRuntimestep, FTRA_Current_Specified_Release)
network.getStateVariable("BEND_Specified_Release").setValue(currentRuntimestep, BEND_Current_Specified_Release)
network.getStateVariable("OAHE_Specified_Release").setValue(currentRuntimestep, OAHE_Current_Specified_Release)

outputDebug(currentRuntimestep, debug, 'OAHE Current Release = %.0f' % OAHE_Current_Specified_Release, '\tBEND Current Release = %.0f' % BEND_Current_Specified_Release, \
\tFTRA Current Release = %.0f' % FTRA_Current_Specified_Release, '\tGAPT Current Release = %.0f' % GAPT_Current_Specified_Release)

GAPT_Previous_Inflow = network.getTimeSeries("Reservoir", "Lewis and Clark Lake", "Pool", "Flow-IN").getPreviousValue(currentRuntimestep)
outputDebug(currentRuntimestep, debug, 'GAPT Previous Release = %.0f cfs' % Model_Variable['GAPT_Previous_Release'], '\tGAPT Previous Storage = %.0f acre-ft' % Model_Variable['GAPT_Previous_Stor'],
\t\t\t\t\tFTRA Previous Release = %.0f cfs' % Model_Variable['FTRA_Previous_Release'], '\tFTRA Previous Storage = %.0f acre-ft' % Model_Variable['FTRA_Previous_Stor'],
\t\t\t\t\tOAHE Previous Release = %.0f cfs' % Model_Variable['OAHE_Previous_Release'])
# set type and value for OpValue
# type is one of:
# OpRule.RULETYPE_MAX - maximum flow
# OpRule.RULETYPE_MIN - minimum flow
# OpRule.RULETYPE_SPEC - specified flow
opValue.init(OpRule.RULETYPE_MIN, 0)

# return the Operation Value.
# return "None" to have no effect on the compute
return opValue

```

## 17 APPENDIX K – RESERVIOR BALANCING SCRIPTED RULE

""  
 Author: Ryan Larsen  
 Date: 03-16-2015  
 Alternative: FWOP  
 Description: Logic for reservoir unbalancing/balancing system storage. Calculates the required releases out of FTPK and GARR that either balance or unbalance the upstream reservoirs. This calculation is performed on the 1st and 15th of each month and possibly on the 7th and 21st and accounts for forecasted inflow into FTPK, GARR, and OAHE. Since the forecast text files do not have data for Aug-Dec, the normal runoff for each project was used. The lower 3 Mar 1 target storage is based on the Mar 1 guide curve values. The guide curve values are currently hard coded into the script.  
 ""

```

# -----
# Required imports to create the OpValue return object.
# -----

from hec.rss.model      import OpRule
from hec.script        import Constants
from hec.heclib.util   import *
from hec.rss.model     import *
from hec.heclib.dss    import *
from hec.hecmath       import *
from hec.model         import *
from hec.script        import *
from hec.io            import TimeSeriesContainer
from hec.client        import ClientApp
from java.lang         import String
from zipfile           import ZipFile
import os, sys
import traceback
import copy
# Load the ResSimUtils module
wsDir = ResSim.getWatershed().getWorkspacePath()
scriptDir = os.path.normpath(os.path.join(wsDir, "shared"))
if scriptDir not in sys.path : sys.path.append(scriptDir)
import ResSimUtils

# initialization function. optional.
# set up tables and other things that only need to be performed once during the compute.
# currentRule is the rule that holds this script.
# network is the ResSim network.
def initRuleScript(currentRule, network):
    # -----
    # Set alternative names to read correct DSS datasets
    # -----

    Alt_Name = 'FWOP'

    # -----
    # Functions
    # -----

    # Debugging Function: Set debug = True to print all debug statements and = False to turn them off
    # Initialization Debugging Function: Set debug = True to print all debug statements and = False to turn them off
    debug = False
    def initOutputDebug(debug, arg1=",", arg2=",", arg3=",", arg4=",", arg5=",", arg6=",", arg7=",", arg8=",", arg9=",", arg10=",", arg11=",", arg12=",", arg13=",", arg14=""):
        if debug == True:
            print 'Initialization Debug  \t', arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9, arg10, arg11, arg12, arg13, arg14

    # Debugging Function: Set debug = True to print all debug statements and = False to turn them off
    def outputDebug(currentRuntimeStep, debug, arg1=",", arg2=",", arg3=",", arg4=",", arg5=",", arg6=",", arg7=",", arg8=",", arg9=",", arg10=",", arg11=",", arg12=",", arg13=",", arg14=",", arg15=""):
        PassCounter = network.getComputePassCounter() # Pass number of simulation
        PassNumber = PassCounter + 1 # Pass number starts at 0 so add 1 to start it at 1
        if debug == True:
            print 'Pass Number =', PassNumber, ' ', currentRuntimeStep.dateTimeString(), \
                '\t', arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9, arg10, arg11, arg12, arg13, arg14, arg15, '\n'

    # adjustMultiUseStor Function: Adjust the multiple use storages for FTPK, GARR, and OAHE based on unbalancing factor
    def adjustMultiUseStor(      currentRuntimeStep,          # currentRuntimeStep variable passed in through the
runRuleScript()

                                Unbalancing_Schedule,      # Storage unbalancing schedule
                                FTPK_Previous_Elev,          # FTPK previous simulated elevation
                                GARR_Previous_Elev,          # GARR previous simulated elevation
                                OAHE_Previous_Elev,          # OAHE previous simulated elevation
                                Balance_Factor,              # Number of feet to unbalance system
                                FTPK_Multiple_Use_Elev,      # Top of the multiple use/carryover elevation at
FTPK
                                GARR_Multiple_Use_Elev,      # Top of the multiple use/carryover elevation at
GARR
                                OAHE_Multiple_Use_Elev,      # Top of the multiple use/carryover elevation at
OAHE
                                State_Variable                # Dictionary of state variables retrieved
                                during the simulation
                                ):
        FTPK_Multiple_Use_Stor = FTPK_StorElevTable.interpolate(FTPK_Multiple_Use_Elev)
  
```

```

GARR_Multiple_Use_Stor = GARR_StorElevTable.interpolate(GARR_Multiple_Use_Elev)
OAHE_Multiple_Use_Stor = OAHE_StorElevTable.interpolate(OAHE_Multiple_Use_Elev)

# Check pool elevations at each project against unbalancing limits
if FTPK_Previous_Elev < 2227.0 or GARR_Previous_Elev < 1827.0 or OAHE_Previous_Elev < 1600.0:
    Balance_Factor = 0

if Unbalancing_Schedule['FTPK'] == 'High':
    # Write unbalancing schedule to state variables
    State_Variable['FTPK_Unbalancing_SV'].setValue(currentRuntimestep, Balance_Factor)
    State_Variable['GARR_Unbalancing_SV'].setValue(currentRuntimestep, -Balance_Factor)
    State_Variable['OAHE_Unbalancing_SV'].setValue(currentRuntimestep, 0)

    # Increase FTPK pool elevation by the Balance_Factor
    FTPK_Adj_Multiple_Use_Elev = FTPK_Multiple_Use_Elev + Balance_Factor
    FTPK_Adj_Multiple_Use_Stor = FTPK_StorElevTable.interpolate(FTPK_Adj_Multiple_Use_Elev)
    FTPK_Volume_Adjustment = FTPK_Adj_Multiple_Use_Stor - FTPK_Multiple_Use_Stor
    currentRule.varPut('FTPK_Adj_Multiple_Use_Stor', FTPK_Adj_Multiple_Use_Stor)

    # Decrease GARR by the storage amount FTPK was increased
    GARR_Adj_Multiple_Use_Stor = GARR_Multiple_Use_Stor - FTPK_Volume_Adjustment
    currentRule.varPut('GARR_Adj_Multiple_Use_Stor', GARR_Adj_Multiple_Use_Stor)

    # OAHE remains unchanged
    OAHE_Adj_Multiple_Use_Stor = OAHE_Multiple_Use_Stor
    currentRule.varPut('OAHE_Adj_Multiple_Use_Stor', OAHE_Adj_Multiple_Use_Stor)
elif Unbalancing_Schedule['FTPK'] == 'Float':
    # Write unbalancing schedule to state variables
    State_Variable['FTPK_Unbalancing_SV'].setValue(currentRuntimestep, 0)
    State_Variable['GARR_Unbalancing_SV'].setValue(currentRuntimestep, Balance_Factor)
    State_Variable['OAHE_Unbalancing_SV'].setValue(currentRuntimestep, -Balance_Factor)

    # Increase GARR pool elevation by the Balance_Factor
    GARR_Adj_Multiple_Use_Elev = GARR_Multiple_Use_Elev + Balance_Factor
    GARR_Adj_Multiple_Use_Stor = GARR_StorElevTable.interpolate(GARR_Adj_Multiple_Use_Elev)
    GARR_Volume_Adjustment = GARR_Adj_Multiple_Use_Stor - GARR_Multiple_Use_Stor
    currentRule.varPut('GARR_Adj_Multiple_Use_Stor', GARR_Adj_Multiple_Use_Stor)

    # Decrease OAHE by the storage amount GARR was increased
    OAHE_Adj_Multiple_Use_Stor = OAHE_Multiple_Use_Stor - GARR_Volume_Adjustment
    currentRule.varPut('OAHE_Adj_Multiple_Use_Stor', OAHE_Adj_Multiple_Use_Stor)

    # FTPK remains unchanged
    FTPK_Adj_Multiple_Use_Stor = FTPK_Multiple_Use_Stor
    currentRule.varPut('FTPK_Adj_Multiple_Use_Stor', FTPK_Adj_Multiple_Use_Stor)
elif Unbalancing_Schedule['FTPK'] == 'Low':
    # Write unbalancing schedule to state variables
    State_Variable['FTPK_Unbalancing_SV'].setValue(currentRuntimestep, -Balance_Factor)
    State_Variable['GARR_Unbalancing_SV'].setValue(currentRuntimestep, 0)
    State_Variable['OAHE_Unbalancing_SV'].setValue(currentRuntimestep, Balance_Factor)

    # Increase OAHE pool elevation by the Balance_Factor
    OAHE_Adj_Multiple_Use_Elev = OAHE_Multiple_Use_Elev + Balance_Factor
    OAHE_Adj_Multiple_Use_Stor = OAHE_StorElevTable.interpolate(OAHE_Adj_Multiple_Use_Elev)
    OAHE_Volume_Adjustment = OAHE_Adj_Multiple_Use_Stor - OAHE_Multiple_Use_Stor
    currentRule.varPut('OAHE_Adj_Multiple_Use_Stor', OAHE_Adj_Multiple_Use_Stor)

    # Decrease FTPK by the storage amount OAHE was increased
    FTPK_Adj_Multiple_Use_Stor = FTPK_Multiple_Use_Stor - OAHE_Volume_Adjustment
    currentRule.varPut('FTPK_Adj_Multiple_Use_Stor', FTPK_Adj_Multiple_Use_Stor)

    # GARR remains unchanged
    GARR_Adj_Multiple_Use_Stor = GARR_Multiple_Use_Stor
    currentRule.varPut('GARR_Adj_Multiple_Use_Stor', GARR_Adj_Multiple_Use_Stor)

# balanceStorage Function: Set FTPK and GARR releases to balance storages in FTPK, GARR, and OAHE based on Mar 1 target storages
def balanceStorage( currentRuntimestep, # currentRuntimestep variable passed in through the runRuleScript()
                    jDay, # Julian day based on current day and month
                    First_Res, # DCP ID of the reservoir whose target storage will
                    be met first
                    Second_Res, # DCP ID of the reservoir whose target storage will
                    be met second
                    First_Res_Previous_Stor, # Previous storage of the reservoir whose target storage will be
                    met first
                    Second_Res_Previous_Stor, # Previous storage of the reservoir whose target storage will be
                    met second
                    First_Res_Target_Stor, # Target storage of the reservoir whose target storage will
                    be met first
                    Second_Res_Target_Stor, # Target storage of the reservoir whose target storage will
                    be met second
                    First_Res_Forecasted_Stor, # Forecasted storage on Mar 1 of the reservoir whose target
                    storage will be met first
                    Second_Res_Forecasted_Stor, # Forecasted storage on Mar 1 of the reservoir whose
                    target storage will be met second
                    First_Res_Remaining_Runoff, # Remaining incremental runoff of the reservoir whose
                    target storage will be met first
                    Second_Res_Remaining_Runoff, # Remaining incremental runoff of the reservoir whose target
                    storage will be met second

```

```

Schedule_Index, # Index used to select current scheduled release
Monthly_Releases, # Release pattern dictionary for FTPK and GARR
GAPT_Release_Volume, # Release volume from current step until Mar 1 of following

year at GAPT

FTPK_Drought_Release_Elev, # Elevation limit at FTPK where drought releases begin
GARR_Drought_Release_Elev, # Elevation limit at GARR where drought releases begin
FTPK_Multiple_Use_Elev, # FTPK multipurpose pool elevation
GARR_Multiple_Use_Elev, # GARR multipurpose pool elevation
FTPK_Previous_Elev, # Previous simulated pool elevation at FTPK
GARR_Previous_Elev # Previous simulated pool elevation at GARR
):

# Scale minimum and maximum releases at FTPK and GARR
scaleMinMaxRelease(currentRuntimestep, 'FTPK', Monthly_Releases, Schedule_Index, FTPK_Drought_Release_Elev,
FTPK_Multiple_Use_Elev, FTPK_Previous_Elev)
scaleMinMaxRelease(currentRuntimestep, 'GARR', Monthly_Releases, Schedule_Index, GARR_Drought_Release_Elev,
GARR_Multiple_Use_Elev, GARR_Previous_Elev)

if First_Res == 'FTPK':
    First_Release_Location = 'FTPK'
    Second_Release_Location = 'GARR'
elif First_Res == 'OAHE':
    First_Release_Location = 'GARR'
    Second_Release_Location = 'FTPK'

# -----
# Set FTPK and GARR releases to reach target storages
# -----

# Increase a reservoir's releases to reach First_Res_Target_Stor
if First_Res_Forecasted_Stor > First_Res_Target_Stor:
    # Increase releases until target storage is reached
    while First_Res_Forecasted_Stor > First_Res_Target_Stor:
        Max_Release_Diff = 0
        for m in range(Schedule_Index, len(Monthly_Releases['Month'])):
            if First_Res == 'OAHE':
                Monthly_Releases[First_Release_Location]['Standard'][m] =
Monthly_Releases[First_Release_Location]['Standard'][m] - 100
            else:
                Monthly_Releases[First_Release_Location]['Standard'][m] =
Monthly_Releases[First_Release_Location]['Standard'][m] + 100
            # Check to make sure calculated releases do not exceed max releases
            if Monthly_Releases[First_Release_Location]['Standard'][m] >
Monthly_Releases[First_Release_Location]['Max'][m]:
                Monthly_Releases[First_Release_Location]['Standard'][m] =
Monthly_Releases[First_Release_Location]['Max'][m]
            Max_Release_Diff += Monthly_Releases[First_Release_Location]['Standard'][m] -
Monthly_Releases[First_Release_Location]['Max'][m]
            # Check forecasted Mar 01 storage vs target storage
            First_Release_Volume = releaseVolume(jDay, Monthly_Releases, First_Release_Location)
            assert First_Res != 'GARR', 'GARR cannot be assessed first'
            if First_Res == 'FTPK':
                First_Res_Forecasted_Stor = First_Res_Previous_Stor + First_Res_Remaining_Runoff -
First_Release_Volume
            elif First_Res == 'OAHE':
                First_Res_Forecasted_Stor = First_Res_Previous_Stor + First_Res_Remaining_Runoff +
First_Release_Volume - GAPT_Release_Volume
            # If all releases are equal to maximum releases, break the loop because the target storage cannot be reached
            if Max_Release_Diff == 0:
                break

# Decrease a reservoir's releases to reach First_Res_Target_Stor
elif First_Res_Forecasted_Stor < First_Res_Target_Stor:
    # Decrease releases until target storage is reached
    while First_Res_Forecasted_Stor < First_Res_Target_Stor:
        Min_Release_Diff = 0
        for m in range(Schedule_Index, len(Monthly_Releases['Month'])):
            if First_Res == 'OAHE':
                Monthly_Releases[First_Release_Location]['Standard'][m] =
Monthly_Releases[First_Release_Location]['Standard'][m] + 100
            else:
                Monthly_Releases[First_Release_Location]['Standard'][m] =
Monthly_Releases[First_Release_Location]['Standard'][m] - 100
            # Check to make sure calculated releases do not exceed min releases
            if Monthly_Releases[First_Release_Location]['Standard'][m] <
Monthly_Releases[First_Release_Location]['Min'][m]:
                Monthly_Releases[First_Release_Location]['Standard'][m] =
Monthly_Releases[First_Release_Location]['Min'][m]
            Min_Release_Diff += Monthly_Releases[First_Release_Location]['Standard'][m] -
Monthly_Releases[First_Release_Location]['Min'][m]
            # Check forecasted Mar 01 storage vs target storage
            First_Release_Volume = releaseVolume(jDay, Monthly_Releases, First_Release_Location)
            assert First_Res != 'GARR', 'GARR cannot be assessed first'
            if First_Res == 'FTPK':
                First_Res_Forecasted_Stor = First_Res_Previous_Stor + First_Res_Remaining_Runoff -
First_Release_Volume
            elif First_Res == 'OAHE':
                First_Res_Forecasted_Stor = First_Res_Previous_Stor + First_Res_Remaining_Runoff +
First_Release_Volume - GAPT_Release_Volume
            # If all releases are equal to minimum releases, break the loop because the target storage cannot be reached

```

```

        if Min_Release_Diff == 0:
            break

    # Increase a reservoir's releases to reach Second_Res_Target_Stor
    if Second_Res_Forecasted_Stor > Second_Res_Target_Stor:
        # Increase releases until target storage is reached
        while Second_Res_Forecasted_Stor > Second_Res_Target_Stor:
            Max_Release_Diff = 0
            for m in range(Schedule_Index, len(Monthly_Releases['Month'])):
                if Second_Res == 'OAHE':
                    Monthly_Releases[Second_Release_Location]['Standard'][m] =
Monthly_Releases[Second_Release_Location]['Standard'][m] - 100
                else:
                    Monthly_Releases[Second_Release_Location]['Standard'][m] =
Monthly_Releases[Second_Release_Location]['Standard'][m] + 100
                # Check to make sure calculated releases do not exceed max releases
                if Monthly_Releases[Second_Release_Location]['Standard'][m] >
Monthly_Releases[Second_Release_Location]['Max'][m]:
                    Monthly_Releases[Second_Release_Location]['Standard'][m] =
Monthly_Releases[Second_Release_Location]['Max'][m]
                Max_Release_Diff += Monthly_Releases[Second_Release_Location]['Standard'][m] -
Monthly_Releases[Second_Release_Location]['Max'][m]
                # Check forecasted Mar 01 storage vs target storage
                Second_Release_Volume = releaseVolume(jDay, Monthly_Releases, Second_Release_Location)
                assert Second_Res != 'OAHE', 'OAHE cannot be assessed first'
                if Second_Res == 'FTPK':
                    Second_Res_Forecasted_Stor = Second_Res_Previous_Stor + Second_Res_Remaining_Runoff -
Second_Release_Volume
            elif Second_Res == 'GARR':
                Second_Res_Forecasted_Stor = Second_Res_Previous_Stor + Second_Res_Remaining_Runoff +
First_Release_Volume - Second_Release_Volume
            # If all releases are equal to maximum releases, break the loop because the target storage cannot be reached
            if Max_Release_Diff == 0:
                break

    # Decrease a reservoir's releases to reach Second_Res_Target_Stor
    elif Second_Res_Forecasted_Stor < Second_Res_Target_Stor:
        # Decrease releases until target storage is reached
        while Second_Res_Forecasted_Stor < Second_Res_Target_Stor:
            Min_Release_Diff = 0
            for m in range(Schedule_Index, len(Monthly_Releases['Month'])):
                if Second_Res == 'OAHE':
                    Monthly_Releases[Second_Release_Location]['Standard'][m] =
Monthly_Releases[Second_Release_Location]['Standard'][m] + 100
                else:
                    Monthly_Releases[Second_Release_Location]['Standard'][m] =
Monthly_Releases[Second_Release_Location]['Standard'][m] - 100
                # Check to make sure calculated releases do not exceed min releases
                if Monthly_Releases[Second_Release_Location]['Standard'][m] <
Monthly_Releases[Second_Release_Location]['Min'][m]:
                    Monthly_Releases[Second_Release_Location]['Standard'][m] =
Monthly_Releases[Second_Release_Location]['Min'][m]
                Min_Release_Diff += Monthly_Releases[Second_Release_Location]['Standard'][m] -
Monthly_Releases[Second_Release_Location]['Min'][m]
                # Check forecasted Mar 01 storage vs target storage
                Second_Release_Volume = releaseVolume(jDay, Monthly_Releases, Second_Release_Location)
                assert Second_Res != 'OAHE', 'OAHE cannot be assessed first'
                if Second_Res == 'FTPK':
                    Second_Res_Forecasted_Stor = Second_Res_Previous_Stor + Second_Res_Remaining_Runoff -
Second_Release_Volume
            elif Second_Res == 'GARR':
                Second_Res_Forecasted_Stor = Second_Res_Previous_Stor + Second_Res_Remaining_Runoff +
First_Release_Volume - Second_Release_Volume
            # If all releases are equal to minimum releases, break the loop because the target storage cannot be reached
            if Min_Release_Diff == 0:
                break
    return First_Res_Forecasted_Stor, Second_Res_Forecasted_Stor

# currentReleaseMonth Function: Calculates the current month for the release schedules
def currentReleaseMonth(jDay, Monthly_Releases):
    if jDay > julianDay('FEB_28'):
        for d in range(len(Monthly_Releases['Month'])):
            if jDay <= julianDay(Monthly_Releases['Month'][d]):
                Schedule_Index = d
                break # Stop the for loop
    else:
        for d in range(Monthly_Releases['Month'].index('JAN_31'), len(Monthly_Releases['Month'])):
            if jDay <= julianDay(Monthly_Releases['Month'][d]):
                Schedule_Index = d
                break # Stop the for loop
    return Schedule_Index

# floodDroughtReleases Function: Set releases to flood or drought conservation releases if limits are exceeded
def floodDroughtReleases( currentRun timestep, # currentRun timestep variable passed in through the
runRuleScript()
                        Location, # DCP ID string of dam
                        Upstream_Location, # DCP ID string of upstream dam
                        (i.e. 'GARR')
USACE—Omaha District
DRAFT

```

```

Downstream_Location, # DCP ID string of downstream dam
(i.e. 'FTRA')
Schedule_Index, # Current scheduled month
Monthly_Releases, # Release pattern dictionary for
FTPK and GARR
Drought_Elevation, # Elevation limit at location where
drought releases begin
Downstream_Drought_Elevation, # Elevation limit at downstream location
where drought releases begin
Flood_Elevation, # Elevation limit at location where
flood releases begin
Lower_Flood_Elev_Constraint, # Elevation constraint at the current dam if
the downstream dam is below its lower elevation constraint
Upper_Flood_Elev_Constraint, # Elevation constraint at the current dam if
the downstream dam is above its lower elevation constraint
Downstream_Flood_Elev_Constraint, # Lower flood elevation constraint at the
downstream dam
Local_Flow, # Local flow data for current
dam
Elev_Stor_Table, # Elev-Stor table for current dam
Pool_Falling, # Is the pool elevation rising or
falling? Yes or No
Exclusive_Flood_Elev # Elevation at the top of the exclusive flood
control zone
):
outputDebug(currentRuntimestep, debug, 'FORECAST %s FLOOD/DROUGHT RELEASES' % Location)
rtw = currentRuntimestep.getRuntimeWindow() # Run time window
totalSteps = rtw.getNumSteps() # Total number of steps in the simulation
curStep = currentRuntimestep.getStep() # Current step
jDay = julianDay(currentRuntimestep.getHecTime().month(),
currentRuntimestep.getHecTime().day())
forecastPeriod = 7
Reservoir = initInfo['projectDict'][Location]
Downstream_Reservoir = initInfo['projectDict'][Downstream_Location]
if Upstream_Location is not None:
    Upstream_Reservoir = initInfo['projectDict'][Upstream_Location]
    lag = len(coeffDict[Upstream_Location])
    Upstream_Releases = network.getTimeSeries("Reservoir", Upstream_Reservoir, "Pool", "Flow-
OUT").getDataArrayFor(curStep - 1, -lag)
else:
    Upstream_Reservoir = None
    Upstream_Releases = None
    lag = 0
    Previous_Elev = network.getTimeSeries("Reservoir", Reservoir, "Pool",
"Elev").getPreviousValue(currentRuntimestep)
    Previous_Stor = network.getTimeSeries("Reservoir", Reservoir, "Pool",
"Stor").getPreviousValue(currentRuntimestep)
    Previous_Release = network.getTimeSeries("Reservoir", Reservoir, "Pool", "Flow-
OUT").getPreviousValue(currentRuntimestep)
    Lagged_Elev = network.getTimeSeries("Reservoir", Reservoir, "Pool",
"Elev").getLaggedValue(currentRuntimestep, 2)
    Downstream_Elev = network.getTimeSeries("Reservoir", Downstream_Reservoir, "Pool",
"Elev").getPreviousValue(currentRuntimestep)
    Downstream_Release = network.getTimeSeries("Reservoir", Downstream_Reservoir, "Pool", "Flow-
OUT").getPreviousValue(currentRuntimestep)
    Downstream_Release_Array = network.getTimeSeries("Reservoir", Downstream_Reservoir, "Pool", "Flow-
OUT").getDataArrayFor(curStep - 1, -7)
    Forecast_Elevations = [0] * forecastPeriod
    delta = 2000

# Drought Conservation
if Previous_Elev < Drought_Elevation and Downstream_Elev > Downstream_Drought_Elevation:
    Monthly_Releases[Location]['Standard'][Schedule_Index] = Monthly_Releases[Location]['Min'][Schedule_Index]

# Flood releases when current reservoir has high pool
if Previous_Elev >= Flood_Elevation and Pool_Falling == 'No':
    if Downstream_Elev < Downstream_Flood_Elev_Constraint:
        Flood_Elev_Constraint = Lower_Flood_Elev_Constraint
    else:
        Flood_Elev_Constraint = Upper_Flood_Elev_Constraint
    Flood_Release = Monthly_Releases[Location]['Standard'][Schedule_Index]
    while Flood_Elev_Constraint <= Upper_Flood_Elev_Constraint:
        if Upstream_Releases is not None:
            Routed_Upstream_Release = routeFlow(Upstream_Location, Upstream_Releases)
        else:
            Routed_Upstream_Release = [0]
        Upstream_Release_Inflow = Routed_Upstream_Release[-1]
        Previous_Storage = Previous_Stor
        # Forecast elevation within forecast period
        for step in range(forecastPeriod):
            Iteration_Counter = 0
            if (curStep + step + lag) > totalSteps:
                break
            Storage_Change = (Upstream_Release_Inflow + Local_Flow[curStep + step] - Flood_Release) *
CFS_to_ACRE_FT

Storage = Previous_Storage + Storage_Change
Elevation = Elev_Stor_Table.interpolate(Storage)
if Elevation >= Flood_Elev_Constraint:

```

```

# Increase releases while forecasted elevations exceed elevation constraint
while Elevation >= Flood_Elev_Constraint:
    if Iteration_Counter > 1000:
        break
    Flood_Release += delta
    Storage_Change = (Upstream_Release_Inflow + Local_Flow[curStep + step] - Flood_Release)
* CFS_to_ACRE_FT
    Storage = Previous_Storage + Storage_Change
    Elevation = Elev_Stor_Table.interpolate(Storage)
    if Flood_Elev_Constraint < Upper_Flood_Elev_Constraint and Flood_Release >
Monthly_Releases[Location]['Flood_Max'][Schedule_Index]:
    break
    Iteration_Counter += 1
    Forecast_Elevations[step] = Elevation
    Previous_Storage = Storage
    if Flood_Release > Monthly_Releases[Location]['Flood_Max'][Schedule_Index] and Elevation >
Flood_Elev_Constraint:
    break
    if Flood_Release <= Monthly_Releases[Location]['Flood_Max'][Schedule_Index]:
        break
    else:
        if Flood_Elev_Constraint < Upper_Flood_Elev_Constraint:
            Flood_Release = Previous_Release
            Flood_Elev_Constraint += 1.0
        # If flood release is greater than maximum flood release, set to maximum flood release
        if Flood_Release > Monthly_Releases[Location]['Extreme_Flood_Max'][Schedule_Index]:
            Flood_Release = Monthly_Releases[Location]['Extreme_Flood_Max'][Schedule_Index]
        # If flood release is greater than the standard release, set standard release to the flood release
        if Flood_Release > Monthly_Releases[Location]['Standard'][Schedule_Index]:
            Monthly_Releases[Location]['Standard'][Schedule_Index] = Flood_Release
        # Flood releases when downstream reservoir has high pool but current reservoir does not
        elif Downstream_Elev >= Downstream_Flood_Elev_Constraint and Previous_Elev < (Exclusive_Flood_Elev - 1.5):
            Monthly_Releases[Location]['Standard'][Schedule_Index] = Monthly_Releases[Location]['Min'][Schedule_Index]

# forecastPoolElevations Function: Forecast pool elevations during forecast period to see if the pool elevations is rising or falling
def forecastPoolElevations( currentRuntimestep, # currentRuntimestep variable passed in through the runRuleScript()
    curStep, # Current step in the simulation
    totalSteps, # Total number of steps in the simulation
    Location, # DCP ID for current reservoir
    Upstream_Location, # DCP ID for upstream reservoir
    Monthly_Releases, # Release pattern
    Schedule_Index, # Current scheduled month
    Previous_Stor, # Previous storage for current reservoir
    Local_Flow, # Local flow for current reservoir
    Elev_Stor_Table # Elevation storage table for current reservoir
):
    outputDebug(currentRuntimestep, debug, 'FORECAST %s POOL ELEVATIONS' % Location)
    forecastPeriod = 15
    Forecast_Elevations = [0] * forecastPeriod
    if Upstream_Location is not None:
        Upstream_Reservoir = initInfo['projectDict'][Upstream_Location]
        lag = len(coeffDict[Upstream_Location])
        Upstream_Releases = network.getTimeSeries("Reservoir", Upstream_Reservoir, "Pool", "Flow-
OUT").getDataArrayFor(curStep - 1, -lag)
    else:
        Upstream_Reservoir = None
        Upstream_Releases = None
        lag = 0

    if Upstream_Location is not None:
        Routed_Upstream_Release = routeFlow(Upstream_Location, Upstream_Releases)
    else:
        Routed_Upstream_Release = [0]
    Upstream_Release_Inflow = Routed_Upstream_Release[-1]
    Previous_Storage = Previous_Stor
    # Forecast elevation within forecast period
    for step in range(forecastPeriod):
        Iteration_Counter = 0
        if curStep + step > totalSteps:
            break
        Storage_Change = (Upstream_Release_Inflow + Local_Flow[curStep + step] -
Monthly_Releases[Location]['Standard'][Schedule_Index]) * CFS_to_ACRE_FT
        Storage = Previous_Storage + Storage_Change
        Elevation = Elev_Stor_Table.interpolate(Storage)
        Forecast_Elevations[step] = Elevation
        Previous_Storage = Storage

    # Check to see if pool elevation is falling
    Elev_Diff_Array = []
    for step in range(len(Forecast_Elevations) - 1):
        Elev_Difference = Forecast_Elevations[step + 1] - Forecast_Elevations[step]
        if Elev_Difference <= 0:
            Elev_Diff_Array.append(Elev_Difference)
    if len(Elev_Diff_Array) == (len(Forecast_Elevations) - 1):
        return 'Yes'
    else:
        return 'No'

```

```

# holdSummerRelease Function: If criteria is met, keep releases constant during nesting season
def holdSummerRelease( currentRuntimestep, # currentRuntimestep variable passed in through the
runRuleScript()
                        Location, # DCP ID for current reservoir
                        jDay, # Julian day based on current day and
month
                        Schedule_Index, # Current scheduled month
                        Monthly_Releases # Release pattern dictionary for FTPK and GARR
                        ):
# Retrieve GARR summer release from memory
if Location == 'FTPK':
    Summer_Release = currentRule.varGet('FTPK_Summer_Release')
elif Location == 'GARR':
    Summer_Release = currentRule.varGet('GARR_Summer_Release')

Monthly_Releases[Location]['Standard'][Schedule_Index] = Summer_Release

# janJulSysRunoff Function: Calculate observed Jan-Jul accumulated runoff
def janJulSysRunoff( currentRuntimestep, # currentRuntimestep variable passed in through the runRuleScript()
                    FTPK_Inc_Inflow, # FTPK incremental inflow
                    GARR_Inc_Inflow, # GARR incremental inflow
                    OAHE_Inc_Inflow, # OAHE incremental inflow
                    BEND_Inc_Inflow, # BEND incremental inflow
                    FTRA_Inc_Inflow, # FTRA incremental inflow
                    GAPT_Inc_Inflow # GAPT incremental inflow
                    ):
    rtw = currentRuntimestep.getRunTimeWindow() # Run time window
    curStep = currentRuntimestep.getStep() # Current step
    year = currentRuntimestep.getHecTime().year() # Current year
    mon = currentRuntimestep.getHecTime().month() # Current month
    day = currentRuntimestep.getHecTime().day() # Current day
    numLookbackSteps = rtw.getNumLookbackSteps() # Number of lookback steps in simulation
    simStartStep = numLookbackSteps + 1 # First step after the lookback period

# Calculate Jan-Jul System Runoff to Adjust Normal Forecasted Runoff
if curStep == simStartStep:
    currentRule.varPut('Start_Year', year)
    currentRule.varPut('System_Accum_Runoff', [])
    currentRule.varPut('Runoff_Ratio', 2.0)
elif mon == 1 and day == 1:
    currentRule.varPut('System_Accum_Runoff', [])
    currentRule.varPut('Runoff_Ratio', 2.0)

if mon <= 7 and curStep > simStartStep:
    System_Accum_Runoff = currentRule.varGet('System_Accum_Runoff')
    runoff = (FTPK_Inc_Inflow + GARR_Inc_Inflow + OAHE_Inc_Inflow + BEND_Inc_Inflow + FTRA_Inc_Inflow +
GAPT_Inc_Inflow) * initInfo['CFS_to_ACRE_FT']
    System_Accum_Runoff.append(runoff)
    currentRule.varPut('System_Accum_Runoff', System_Accum_Runoff)
elif mon == 8 and day == 1:
    System_Accum_Runoff = currentRule.varGet('System_Accum_Runoff')
    runoff = (FTPK_Inc_Inflow + GARR_Inc_Inflow + OAHE_Inc_Inflow + BEND_Inc_Inflow + FTRA_Inc_Inflow +
GAPT_Inc_Inflow) * initInfo['CFS_to_ACRE_FT']
    System_Accum_Runoff.append(runoff)
    System_Accum_Runoff = sum(System_Accum_Runoff)
    # Normal runoff for Jan-Jul based on 2014 statistics
    Normal_System_Runoff = 18017000 # acre-ft
    Runoff_Ratio = System_Accum_Runoff / Normal_System_Runoff
    currentRule.varPut('Runoff_Ratio', Runoff_Ratio)

# julianDay Function: Get the julian day for a month and day or a string, or a month and day for a julian day
def julianDay(*args) :
# J F M A M J J A S O N D
daysPrev = 0, 31, 59, 90,120,151,181,212,243,273,304,334
argCount = len(args)
if argCount == 1 :
    if type(args[0]) == type(0) :
        jday = args[0]
        if not 1 <= jday <= 365 :
            raise ValueError("Julian day out of range 1..365")
        for i in range(12)[::-1] :
            if jday > daysPrev[i] :
                mon = i + 1
                break
        day = jday - daysPrev[mon-1]
        return mon, day
    else :
        mon, day = args[0].split("_")
        mon = ["JAN", "FEB", "MAR", "APR", "MAY", "JUN", "JUL", "AUG", "SEP", "OCT", "NOV", "DEC"].index(mon.upper()) + 1
        day = int(day);
        return julianDay(mon, day)
elif argCount == 2 :
    mon, day = args
    if not 1 <= mon <= 12 :
        raise ValueError("Month is out of range 1..12")
    if not 1 <= day <= 31 :
        raise ValueError("Day is out of range 1..31")
    if mon == 2 and day == 29 : day = 28

```

```

        return daysPrev[mon-1] + day
    else :
        raise ValueError("Expected 1 or 2 arguments, got %d" % argCount)

# mar1PercentFull Function: Calculate upper system (FTPK, GARR, and OAHE) Mar 1 percent full of carryover storage
def mar1PercentFull( currentRuntimestep, # currentRuntimestep variable passed in through the runRuleScript()
                    GAPT_Release, # GAPT specified release
                    FTPK_Remaining_Runoff, # FTPK remaining runoff
                    GARR_Remaining_Runoff, # GARR remaining runoff
                    OAHE_Remaining_Runoff, # OAHE remaining runoff
                    BEND_and_FTRA_Remaining_Runoff, # BEND and FTRA remaining runoff
                    GAPT_Remaining_Runoff, # GAPT remaining runoff
                    FTPK_Multiple_Use_Stor, # FTPK storage at top of multiple use zone
                    FTPK_Permanent_Stor, # FTPK storage at top of permanent zone
                    GARR_Multiple_Use_Stor, # GARR storage at top of multiple use zone
                    GARR_Permanent_Stor, # GARR storage at top of permanent zone
                    OAHE_Multiple_Use_Stor, # OAHE storage at top of multiple use zone
                    OAHE_Permanent_Stor, # OAHE storage at top of permanent zone
                    FTPK_Previous_Stor, # FTPK previous storage
                    GARR_Previous_Stor, # GARR previous storage
                    OAHE_Previous_Stor, # OAHE previous storage
                    BEND_Previous_Stor, # BEND previous storage
                    FTRA_Previous_Stor, # FTRA previous storage
                    GAPT_Previous_Stor # GAPT previous storage
                    ):
    BEND_Mar_01_GC_Elev, FTRA_Mar_01_GC_Elev, GAPT_Mar_01_GC_Elev = 1420.5, 1350.0, 1206.0
    BEND_Mar_01_GC_Stor = initInfo['BEND_StorElevTable'].interpolate(BEND_Mar_01_GC_Elev)
    FTRA_Mar_01_GC_Stor = initInfo['FTRA_StorElevTable'].interpolate(FTRA_Mar_01_GC_Elev)
    GAPT_Mar_01_GC_Stor = initInfo['GAPT_StorElevTable'].interpolate(FTRA_Mar_01_GC_Elev)

    # Calculate Upstream Storage (FTPK, GARR, and OAHE) Required to Meet GAPT Releases
    # Total System Release from current day to Mar 1 of the next year. Accounts for downstream incremental runoff of and storage.
    Total_GAPT_Release_Volume = sum(GAPT_Release) * initInfo['CFS_to_ACRE_FT'] # Calculated from 'Downstream ResSim Model'

    #
    # FTRA Storage Change
    # BEND Storage Change
    # GAPT Storage Change
    Lower_System_Storage = (BEND_Previous_Stor - BEND_Mar_01_GC_Stor) + (FTRA_Previous_Stor - FTRA_Mar_01_GC_Stor) +
(GAPT_Previous_Stor - GAPT_Mar_01_GC_Stor)

    # Remaining runoff from lower 3 reservoirs (BEND, FTRA, and GAPT) and upper 3 reservoirs (FTPK, GARR, and OAHE)
    Lower_System_Remaining_Runoff = BEND_and_FTRA_Remaining_Runoff + GAPT_Remaining_Runoff
    Upper_System_Remaining_Runoff = FTPK_Remaining_Runoff + GARR_Remaining_Runoff + OAHE_Remaining_Runoff

    # Part of GAPT release volume will be met with storage and runoff from the lower reservoirs (BEND, FTRA, and GAPT). The
remaining volume will be
    # supplied from the upper 3 reservoirs' (FTPK, GARR, and OAHE) runoff and storage
    Upper_System_Volume = Total_GAPT_Release_Volume - (Lower_System_Storage + Lower_System_Remaining_Runoff)
    Required_Upper_System_Storage = Upper_System_Volume - Upper_System_Remaining_Runoff

    # Total carryover storage in upper 3 reservoirs
    Upper_System_Carryover_Stor = (FTPK_Multiple_Use_Stor - FTPK_Permanent_Stor) + (GARR_Multiple_Use_Stor -
GARR_Permanent_Stor) + \
(OAHE_Multiple_Use_Stor - OAHE_Permanent_Stor)

    # Current carryover storage available in upper 3 reservoirs
    Upper_System_Current_Stor = (FTPK_Previous_Stor - FTPK_Permanent_Stor) + (GARR_Previous_Stor - GARR_Permanent_Stor)
+ (OAHE_Previous_Stor - OAHE_Permanent_Stor)

    # Mar 1 carryover storage in upper 3 reservoirs
    Upper_System_Mar1_Storage = Upper_System_Current_Stor - Required_Upper_System_Storage

    # Carryover storage percent full
    Upper_System_Percent_Full = Upper_System_Mar1_Storage / Upper_System_Carryover_Stor

    return Upper_System_Percent_Full, Total_GAPT_Release_Volume

# monthlyAccumRunoff Function: Track monthly runoff for each project
def monthlyAccumRunoff( currentRuntimestep, # currentRuntimestep variable passed in through the runRuleScript()
                      FTPK_Inc_Inflow, # FTPK incremental inflow
                      GARR_Inc_Inflow, # GARR incremental inflow
                      OAHE_Inc_Inflow, # OAHE incremental inflow
                      BEND_Inc_Inflow, # BEND incremental inflow
                      FTRA_Inc_Inflow, # FTRA incremental inflow
                      GAPT_Inc_Inflow # GAPT incremental inflow
                      ):
    rtw = currentRuntimestep.getRunTimeWindow() # Run time window
    curStep = currentRuntimestep.getStep() # Current step
    year = currentRuntimestep.getHecTime().year() # Current year
    mon = currentRuntimestep.getHecTime().month() # Current month
    day = currentRuntimestep.getHecTime().day() # Current day
    numLookbackSteps = rtw.getNumLookbackSteps() # Number of lookback steps in simulation
    simStartStep = numLookbackSteps + 1 # First step after the lookback period

    if curStep == simStartStep or day == 1:
        currentRule.varPut('FTPK_Accum_Stor', [])
        currentRule.varPut('GARR_Accum_Stor', [])
        currentRule.varPut('OAHE_Accum_Stor', [])
        currentRule.varPut('BEND_and_FTRA_Accum_Stor', [])

```

```

currentRule.varPut('GAPT_Accum_Stor', [])

FTPK_Accum_Stor = currentRule.varGet('FTPK_Accum_Stor')
GARR_Accum_Stor = currentRule.varGet('GARR_Accum_Stor')
OAHE_Accum_Stor = currentRule.varGet('OAHE_Accum_Stor')
BEND_and_FTRA_Accum_Stor = currentRule.varGet('BEND_and_FTRA_Accum_Stor')
GAPT_Accum_Stor = currentRule.varGet('GAPT_Accum_Stor')
else:
    FTPK_Accum_Stor = currentRule.varGet('FTPK_Accum_Stor')
    GARR_Accum_Stor = currentRule.varGet('GARR_Accum_Stor')
    OAHE_Accum_Stor = currentRule.varGet('OAHE_Accum_Stor')
    BEND_and_FTRA_Accum_Stor = currentRule.varGet('BEND_and_FTRA_Accum_Stor')
    GAPT_Accum_Stor = currentRule.varGet('GAPT_Accum_Stor')

    FTPK_Accum_Stor.append(FTPK_Inc_Inflow * initInfo['CFS_to_ACRE_FT'])
    GARR_Accum_Stor.append(GARR_Inc_Inflow * initInfo['CFS_to_ACRE_FT'])
    OAHE_Accum_Stor.append(OAHE_Inc_Inflow * initInfo['CFS_to_ACRE_FT'])
    BEND_and_FTRA_Accum_Stor.append((BEND_Inc_Inflow + FTRA_Inc_Inflow) * initInfo['CFS_to_ACRE_FT'])
    GAPT_Accum_Stor.append(GAPT_Inc_Inflow * initInfo['CFS_to_ACRE_FT'])
return FTPK_Accum_Stor, GARR_Accum_Stor, OAHE_Accum_Stor, BEND_and_FTRA_Accum_Stor, GAPT_Accum_Stor

# releaseVolume Function: Based on a release pattern, calculate the total volume released from a reservoir
def releaseVolume( jDay, # Julian day of the current step
                  Release_Schedule, # Monthly releases for FTRA and GARR
                  Location # DCP ID for current location
                  ):
    Current_Schedule_Index = currentReleaseMonth(jDay, Release_Schedule)
    Partial_Month = julianDay(Release_Schedule['Month'][Current_Schedule_Index]) - jDay

    Release_Schedule_Months_Remaining = Release_Schedule['Month'][Current_Schedule_Index:]
    Release_Schedule_Discharge_Remaining = Release_Schedule[Location]['Standard'][Current_Schedule_Index:]
    for d in range(len(Release_Schedule_Months_Remaining)):
        if d == 0:
            EOY_Release_Volume = Release_Schedule_Discharge_Remaining[d] * Partial_Month * CFS_to_ACRE_FT
        elif Release_Schedule_Months_Remaining[d] == 'JAN_31' and d != 0:
            EOY_Release_Volume = EOY_Release_Volume + (julianDay(Release_Schedule_Months_Remaining[d]) *
Release_Schedule_Discharge_Remaining[d] * CFS_to_ACRE_FT)
        else:
            EOY_Release_Volume = EOY_Release_Volume + ((julianDay(Release_Schedule_Months_Remaining[d]) -
julianDay(Release_Schedule_Months_Remaining[d-1])) * Release_Schedule_Discharge_Remaining[d] * CFS_to_ACRE_FT)

    return EOY_Release_Volume

# remainingAnnualRunoff Function: Calculate remaining annual incremental runoff
def remainingAnnualRunoff( currentRuntimestep, # currentRuntimestep variable passed in through the runRuleScript()
                          FTPK_Accum_Stor, # FTPK monthly accumulated runoff
                          GARR_Accum_Stor, # GARR monthly accumulated runoff
                          OAHE_Accum_Stor, # OAHE monthly accumulated runoff
                          BEND_and_FTRA_Accum_Stor, # BEND and FTRA monthly accumulated runoff
                          GAPT_Accum_Stor # GAPT monthly accumulated runoff
                          ):
    rtw = currentRuntimestep.getRunTimeWindow() # Run time window
    curStep = currentRuntimestep.getStep() # Current step
    year = currentRuntimestep.getHecTime().year() # Current year
    mon = currentRuntimestep.getHecTime().month() # Current month
    day = currentRuntimestep.getHecTime().day() # Current day
    numLookbackSteps = rtw.getNumLookbackSteps() # Number of lookback steps in simulation
    simStartStep = numLookbackSteps + 1 # First step after the lookback period

    # Calculate Remaining Annual Incremental Runoff
    Runoff_Ratio = currentRule.varGet('Runoff_Ratio')
    Start_Year = currentRule.varGet('Start_Year')
    monString = monDict[mon][0]
    FTPK_Observed_Runoff = sum(FTPK_Accum_Stor)
    GARR_Observed_Runoff = sum(GARR_Accum_Stor)
    OAHE_Observed_Runoff = sum(OAHE_Accum_Stor)
    BEND_and_FTRA_Observed_Runoff = sum(BEND_and_FTRA_Accum_Stor)
    GAPT_Observed_Runoff = sum(GAPT_Accum_Stor)

    # If Jan-Jul accumulated runoff is less than the normal runoff for the period, decrease the Aug-Dec normal runoff values to reflect a
drier year
    # Don't factor the first year because simulation starts in March
    if 8 <= mon <= 12 and Runoff_Ratio < 1.0 and year > Start_Year:
        FTPK_Next_Year_Jan_Feb_Fcst = fcstFlows[year + 1]['FTPK']['Jan'][0] + fcstFlows[year + 1]['FTPK']['Jan'][1]
        GARR_Next_Year_Jan_Feb_Fcst = fcstFlows[year + 1]['GARR']['Jan'][0] + fcstFlows[year + 1]['GARR']['Jan'][1]
        OAHE_Next_Year_Jan_Feb_Fcst = fcstFlows[year + 1]['OAHE']['Jan'][0] + fcstFlows[year + 1]['OAHE']['Jan'][1]
        BEND_and_FTRA_Next_Year_Jan_Feb_Fcst = fcstFlows[year + 1]['FTRA']['Jan'][0] + fcstFlows[year + 1]['FTRA']['Jan'][1]
        GAPT_Next_Year_Jan_Feb_Fcst = fcstFlows[year + 1]['GAPT']['Jan'][0] + fcstFlows[year + 1]['GAPT']['Jan'][1]

        FTPK_Normal_Runoff_Fcst = EOY_Runoff['FTPK'][year][monString] - FTPK_Next_Year_Jan_Feb_Fcst
        GARR_Normal_Runoff_Fcst = EOY_Runoff['GARR'][year][monString] - GARR_Next_Year_Jan_Feb_Fcst
        OAHE_Normal_Runoff_Fcst = EOY_Runoff['OAHE'][year][monString] - OAHE_Next_Year_Jan_Feb_Fcst
        BEND_and_FTRA_Normal_Runoff_Fcst = EOY_Runoff['FTRA'][year][monString] -
BEND_and_FTRA_Next_Year_Jan_Feb_Fcst
        GAPT_Normal_Runoff_Fcst = EOY_Runoff['GAPT'][year][monString] - GAPT_Next_Year_Jan_Feb_Fcst

        FTPK_Forecasted_Runoff = (FTPK_Next_Year_Jan_Feb_Fcst + (FTPK_Normal_Runoff_Fcst * Runoff_Ratio)) * 1000
        GARR_Forecasted_Runoff = (GARR_Next_Year_Jan_Feb_Fcst + (GARR_Normal_Runoff_Fcst * Runoff_Ratio)) * 1000

```

```

    OAHE_Forecasted_Runoff = (OAHE_Next_Year_Jan_Feb_Fcst + (OAHE_Normal_Runoff_Fcst * Runoff_Ratio)) * 1000
    BEND_and_FTRA_Forecasted_Runoff = (BEND_and_FTRA_Next_Year_Jan_Feb_Fcst +
(BEND_and_FTRA_Normal_Runoff_Fcst * Runoff_Ratio)) * 1000
    GAPT_Forecasted_Runoff = (GAPT_Next_Year_Jan_Feb_Fcst + (GAPT_Normal_Runoff_Fcst * Runoff_Ratio)) * 1000

```

```
else:
```

```

    FTPK_Forecasted_Runoff = EOY_Runoff['FTPK'][year][monString] * 1000
    GARR_Forecasted_Runoff = EOY_Runoff['GARR'][year][monString] * 1000
    OAHE_Forecasted_Runoff = EOY_Runoff['OAHE'][year][monString] * 1000
    BEND_and_FTRA_Forecasted_Runoff = EOY_Runoff['FTRA'][year][monString] * 1000
    GAPT_Forecasted_Runoff = EOY_Runoff['GAPT'][year][monString] * 1000

```

```

    FTPK_Remaining_Runoff = FTPK_Forecasted_Runoff - FTPK_Observed_Runoff
    GARR_Remaining_Runoff = GARR_Forecasted_Runoff - GARR_Observed_Runoff
    OAHE_Remaining_Runoff = OAHE_Forecasted_Runoff - OAHE_Observed_Runoff
    BEND_and_FTRA_Remaining_Runoff = BEND_and_FTRA_Forecasted_Runoff - BEND_and_FTRA_Observed_Runoff
    GAPT_Remaining_Runoff = GAPT_Forecasted_Runoff - GAPT_Observed_Runoff
    return FTPK_Remaining_Runoff, GARR_Remaining_Runoff, OAHE_Remaining_Runoff, BEND_and_FTRA_Remaining_Runoff,
GAPT_Remaining_Runoff

```

```

# retrieveModelVariables Function: Retrieve model variables needed for the script
def retrieveModelVariables(currentRuntimestep):

```

```

    Model_Variable = { # FTPK Model Variables
        'FTPK_Previous_Stor' : network.getTimeSeries("Reservoir", "Fort Peck Lake",
"Pool", "Stor").getPreviousValue(currentRuntimestep),
        'FTPK_Previous_Elev' : network.getTimeSeries("Reservoir", "Fort Peck
Lake", "Pool", "Elev").getPreviousValue(currentRuntimestep),
        'FTPK_Previous_Inflow' : network.getTimeSeries("Reservoir", "Fort Peck
Lake", "Pool", "Flow-IN").getPreviousValue(currentRuntimestep),
        'FTPK_Multiple_Use_Stor' : network.getTimeSeries("Reservoir", "Fort Peck Lake",
"Carryover Multiple Use", "Stor-ZONE").getPreviousValue(currentRuntimestep),
        'FTPK_Permanent_Stor' : network.getTimeSeries("Reservoir", "Fort Peck
Lake", "Permanent", "Stor-ZONE").getPreviousValue(currentRuntimestep),
        'FTPK_Exclusive_Flood_Elev' : network.getTimeSeries("Reservoir", "Fort Peck
Lake", "Exclusive Flood Control", "Elev-ZONE").getPreviousValue(currentRuntimestep),
        'FTPK_Flood_and_Multiple_Use_Elev' : network.getTimeSeries("Reservoir", "Fort Peck Lake",
"Flood Control and Multiple Use", "Elev-ZONE").getPreviousValue(currentRuntimestep),
        'FTPK_Multiple_Use_Elev' : network.getTimeSeries("Reservoir", "Fort Peck Lake",
"Carryover Multiple Use", "Elev-ZONE").getPreviousValue(currentRuntimestep),
        # GARR Model Variables
        'GARR_Previous_Stor' : network.getTimeSeries("Reservoir", "Lake
Sakakawea", "Pool", "Stor").getPreviousValue(currentRuntimestep),
        'GARR_Previous_Elev' : network.getTimeSeries("Reservoir", "Lake
Sakakawea", "Pool", "Elev").getPreviousValue(currentRuntimestep),
        'GARR_Multiple_Use_Stor' : network.getTimeSeries("Reservoir", "Lake Sakakawea",
"Carryover Multiple Use", "Stor-ZONE").getPreviousValue(currentRuntimestep),
        'GARR_Permanent_Stor' : network.getTimeSeries("Reservoir", "Lake
Sakakawea", "Permanent", "Stor-ZONE").getPreviousValue(currentRuntimestep),
        'GARR_Exclusive_Flood_Elev' : network.getTimeSeries("Reservoir", "Lake
Sakakawea", "Exclusive Flood Control", "Elev-ZONE").getPreviousValue(currentRuntimestep),
        'GARR_Flood_and_Multiple_Use_Elev' : network.getTimeSeries("Reservoir", "Lake Sakakawea",
"Flood Control and Multiple Use", "Elev-ZONE").getPreviousValue(currentRuntimestep),
        'GARR_Multiple_Use_Elev' : network.getTimeSeries("Reservoir", "Lake Sakakawea",
"Carryover Multiple Use", "Elev-ZONE").getPreviousValue(currentRuntimestep),
        # OAHE Model Variables
        'OAHE_Previous_Stor' : network.getTimeSeries("Reservoir", "Lake Oahe",
"Pool", "Stor").getPreviousValue(currentRuntimestep),
        'OAHE_Previous_Elev' : network.getTimeSeries("Reservoir", "Lake Oahe",
"Pool", "Elev").getPreviousValue(currentRuntimestep),
        'OAHE_Multiple_Use_Stor' : network.getTimeSeries("Reservoir", "Lake Oahe",
"Carryover Multiple Use", "Stor-ZONE").getPreviousValue(currentRuntimestep),
        'OAHE_Permanent_Stor' : network.getTimeSeries("Reservoir", "Lake Oahe",
"Permanent", "Stor-ZONE").getPreviousValue(currentRuntimestep),
        'OAHE_Multiple_Use_Elev' : network.getTimeSeries("Reservoir", "Lake Oahe",
"Carryover Multiple Use", "Elev-ZONE").getPreviousValue(currentRuntimestep),
        # BEND Model Variables
        'BEND_Previous_Stor' : network.getTimeSeries("Reservoir", "Lake
Sharpe", "Pool", "Stor").getPreviousValue(currentRuntimestep),
        'BEND_Previous_Local_Inflow' : network.findJunction("BEND-
Inflow").getLocalFlowTimeSeries("BEND_Local").getPreviousValue(currentRuntimestep),
        # FTRA Model Variables
        'FTRA_Previous_Stor' : network.getTimeSeries("Reservoir", "Lake Francis
Case", "Pool", "Stor").getPreviousValue(currentRuntimestep),
        'FTRA_Previous_Local_Inflow' : network.findJunction("FTRA-
Inflow").getLocalFlowTimeSeries("FTRA_Local").getPreviousValue(currentRuntimestep),
        # GAPT Model Variables
        'GAPT_Previous_Stor' : network.getTimeSeries("Reservoir", "Lewis and
Clark Lake", "Pool", "Stor").getPreviousValue(currentRuntimestep),
        'GAPT_Previous_Local_Inflow' : network.findJunction("GAPT-
Inflow").getLocalFlowTimeSeries("GAPT_Local").getPreviousValue(currentRuntimestep)
    }

    return Model_Variable

```

```

# retrieveModelVariables Function: Retrieve model variables needed for the script
def retrieveStateVariables(currentRuntimestep):

```

```

    State_Variable = { # FTPK State Variables
        'FTPK_Release_SV' : network.getStateVariable("FTPK_Release"),
        'FTPK_Target_Storage_SV' : network.getStateVariable("FTPK_Target_Storage"),

```

```

        'FTPK_Percent_Carryover_SV'           :
network.getStateVariable("FTPK_Percent_Carryover"),
        'FTPK_Unbalancing_SV'               : network.getStateVariable("FTPK_Unbalancing"),
        # GARR State Variables
        'GARR_Release_SV'                   : network.getStateVariable("GARR_Release"),
        'GARR_Target_Storage_SV'           : network.getStateVariable("GARR_Target_Storage"),
        'GARR_Percent_Carryover_SV'        :
network.getStateVariable("GARR_Percent_Carryover"),
        'GARR_Unbalancing_SV'               : network.getStateVariable("GARR_Unbalancing"),
        # OAHE State Variables
        'OAHE_Target_Storage_SV'           : network.getStateVariable("OAHE_Target_Storage"),
        'OAHE_Percent_Carryover_SV'        :
network.getStateVariable("OAHE_Percent_Carryover"),
        'OAHE_Unbalancing_SV'               : network.getStateVariable("OAHE_Unbalancing"),
        # GAPT State Variables
        'GAPT_Specified_Release_SV'        :
network.getStateVariable("GAPT_Specified_Release"),
        # System State Variables
        'System_Percent_Carryover_SV'      : network.getStateVariable("System_Percent_Carryover")
    }

    return State_Variable

# Flow routing function
def routeFlow(Upstream_Location, Flow_Dataset):
    coeffList = coeffDict[Upstream_Location]
    lag = len(coeffList) - 1 # Length of time before any portion of flow reaches the downstream location
    routedFlow = []
    for i in range(len(Flow_Dataset)):
        if i < lag:
            routedFlow.append(0) # Add a place holder of 0 for days less than the lag time
        else:
            routedFlowTempList = []
            for c in range(len(coeffList)):
                flow = Flow_Dataset[i-c] * coeffList[c]
                routedFlowTempList.append(flow)
            release = sum(routedFlowTempList)
            routedFlow.append(release)
    return routedFlow

# scaleMinMaxRelease Function: Scale the minimum and maximum releases for FTPK and GARR
def scaleMinMaxRelease(    currentRuntimestep,    # currentRuntimestep variable passed in through the runRuleScript()
                          Location,              # DCP ID of current reservoir
                          Monthly_Releases,      # Release pattern dictionary for FTPK and GARR
                          Schedule_Index,        # Current scheduled month
                          Drought_Release_Elev,  # Elevation limit at which drought releases begin
                          Multiple_Use_Elev,     # Elevation at top of multipurpose zone
                          Previous_Elev          # Previous simulated elevation
                          ):

    # Scale minimum releases based on ratio of current pool elevation to drought conservation pool
    if Previous_Elev <= Drought_Release_Elev:
        Monthly_Releases[Location]['Min'][Schedule_Index] = Monthly_Releases[Location]['Drought_Min'][Schedule_Index]
    elif Drought_Release_Elev < Previous_Elev <= Multiple_Use_Elev:
        GARR_Min_Discharge_Scale = (Monthly_Releases[Location]['Min'][Schedule_Index] -
Monthly_Releases[Location]['Drought_Min'][Schedule_Index]) / (Multiple_Use_Elev - Drought_Release_Elev)
        for m in range(Schedule_Index, len(Monthly_Releases['Month'])):
            Monthly_Releases[Location]['Min'][m] = GARR_Min_Discharge_Scale * (Previous_Elev - Drought_Release_Elev) +
Monthly_Releases[Location]['Drought_Min'][m]
        # Scale maximum release during summer and fall months. Winter and spring maximum releases should not be increased.
    elif Multiple_Use_Elev < Previous_Elev:
        for m in range(Schedule_Index, len(Monthly_Releases['Month'])):
            if 2 < m < 11:
                Monthly_Releases[Location]['Max'][m] = Monthly_Releases[Location]['Extreme_Flood_Max'][m]

# targetStorages Function: Calculate target storages for FTPK, GARR, and OAHE
def targetStorages(    currentRuntimestep,    # currentRuntimestep variable passed in through the runRuleScript()
                     Percent_Full,          # Percent of occupied carryover storage for FTPK, GARR, and
OAHE
                     FTPK_Adj_Multiple_Use_Stor, # FTPK adjusted top of multiple purpose zone based on
unbalancing
                     GARR_Adj_Multiple_Use_Stor, # GARR adjusted top of multiple purpose zone based on
unbalancing
                     OAHE_Adj_Multiple_Use_Stor, # OAHE adjusted top of multiple purpose zone based on
unbalancing
                     FTPK_Permanent_Stor,      # Top of FTPK permanent storage zone
                     GARR_Permanent_Stor,     # Top of GARR permanent storage zone
                     OAHE_Permanent_Stor,     # Top of OAHE permanent storage zone
                     FTPK_Multiple_Use_Stor,  # Top of the multiple use/carryover storage at FTPK
                     GARR_Multiple_Use_Stor,  # Top of the multiple use/carryover storage at GARR
                     OAHE_Multiple_Use_Stor   # Top of the multiple use/carryover storage at OAHE
                     ):
    FTPK_Permanent_Elev = FTPK_ElevStorTable.interpolate(FTPK_Permanent_Stor)
    GARR_Permanent_Elev = GARR_ElevStorTable.interpolate(GARR_Permanent_Stor)
    OAHE_Permanent_Elev = OAHE_ElevStorTable.interpolate(OAHE_Permanent_Stor)
    FTPK_Min_Target_Stor = FTPK_StorElevTable.interpolate(FTPK_Permanent_Elev + 35)
    GARR_Min_Target_Stor = GARR_StorElevTable.interpolate(GARR_Permanent_Elev + 30)
    OAHE_Min_Target_Stor = OAHE_StorElevTable.interpolate(OAHE_Permanent_Elev + 25)

```

```

if (Percent_Full * 100) < 100:
    FTPK_Target_Stor = ((FTPK_Adj_Multiple_Use_Stor - FTPK_Permanent_Stor) * Percent_Full) + FTPK_Permanent_Stor
    GARR_Target_Stor = ((GARR_Adj_Multiple_Use_Stor - GARR_Permanent_Stor) * Percent_Full) + GARR_Permanent_Stor
    OAHE_Target_Stor = ((OAHE_Adj_Multiple_Use_Stor - OAHE_Permanent_Stor) * Percent_Full) + OAHE_Permanent_Stor
    # Make sure target storages do not fall below minimum target storages
    if FTPK_Target_Stor < FTPK_Min_Target_Stor:
        FTPK_Target_Stor = FTPK_Min_Target_Stor
    if GARR_Target_Stor < GARR_Min_Target_Stor:
        GARR_Target_Stor = GARR_Min_Target_Stor
    if OAHE_Target_Stor < OAHE_Min_Target_Stor:
        OAHE_Target_Stor = OAHE_Min_Target_Stor
    currentRule.varPut('FTPK_Target_Stor', FTPK_Target_Stor)
    currentRule.varPut('GARR_Target_Stor', GARR_Target_Stor)
    currentRule.varPut('OAHE_Target_Stor', OAHE_Target_Stor)
else:
    FTPK_Target_Stor = FTPK_Adj_Multiple_Use_Stor
    GARR_Target_Stor = GARR_Adj_Multiple_Use_Stor
    OAHE_Target_Stor = OAHE_Adj_Multiple_Use_Stor
    currentRule.varPut('FTPK_Target_Stor', FTPK_Target_Stor)
    currentRule.varPut('GARR_Target_Stor', GARR_Target_Stor)
    currentRule.varPut('OAHE_Target_Stor', OAHE_Target_Stor)
return FTPK_Target_Stor, GARR_Target_Stor, OAHE_Target_Stor

# unbalanceReleaseAdj Function: Adjust releases if reservoirs are too far out of balance
def unbalanceReleaseAdj(    currentRuntimestep,    # currentRuntimestep variable passed in through the runRuleScript()
                          Location,                # DCP ID for current reservoir
                          Downstream_Location,    # DCP ID for downstream reservoir
                          Percent_Full,           # Percent of occupied carryover storage at current
reservoir
                          Downstream_Percent_Full, # Percent of occupied carryover storage at downstream
reservoir
                          Monthly_Releases,       # Release pattern
                          Schedule_Index         # Current scheduled month
                          ):
    Release_Adj = 200 * (Percent_Full - Downstream_Percent_Full)
    Release = Monthly_Releases[Location]['Standard'][Schedule_Index] + Release_Adj
    # Check the adjusted release against the minimum and maximum release patterns. If the adjust release exceeds the min or max, set
the release
    # to the minimum or maximum allowable release
    if Monthly_Releases[Location]['Min'][Schedule_Index] < Release < Monthly_Releases[Location]['Max'][Schedule_Index]:
        Monthly_Releases[Location]['Standard'][Schedule_Index] = Release
    elif Monthly_Releases[Location]['Max'][Schedule_Index] < Release:
        Monthly_Releases[Location]['Standard'][Schedule_Index] = Monthly_Releases[Location]['Max'][Schedule_Index]
    elif Release < Monthly_Releases[Location]['Min'][Schedule_Index]:
        Monthly_Releases[Location]['Standard'][Schedule_Index] = Monthly_Releases[Location]['Min'][Schedule_Index]

# unbalancingSchedule Function: Set the unbalancing schedule for the next year
def unbalancingSchedule(Schedule):
    if Unbalancing_Schedule['FTPK'] == 'High':
        initInfo['Unbalancing_Schedule']['FTPK'] = 'Float'
        initInfo['Unbalancing_Schedule']['GARR'] = 'High'
        initInfo['Unbalancing_Schedule']['OAHE'] = 'Low'
    elif Unbalancing_Schedule['FTPK'] == 'Float':
        initInfo['Unbalancing_Schedule']['FTPK'] = 'Low'
        initInfo['Unbalancing_Schedule']['GARR'] = 'Float'
        initInfo['Unbalancing_Schedule']['OAHE'] = 'High'
    elif Unbalancing_Schedule['FTPK'] == 'Low':
        initInfo['Unbalancing_Schedule']['FTPK'] = 'High'
        initInfo['Unbalancing_Schedule']['GARR'] = 'Low'
        initInfo['Unbalancing_Schedule']['OAHE'] = 'Float'

# -----
# DSS File
# -----

fPart = ResSimUtils.getFPart(ResSimUtils.getSelectedAlternativeNames())[0])
dssFilename = ResSimUtils.getSimulationDSSFileName()
simulation = ResSimUtils.getSimulation()
dssFile = HecDss.open(dssFilename)
dssFile.setTimeWindow(simulation.getLookbackDateString(), simulation.getEndDateString())

FTPK_CumLocal = dssFile.read('//FTPK-INFLOW/FLOW-CUMLOC//1DAY/%s/' % fPart).getData().values
GARR_CumLocal = dssFile.read('//GARR-INFLOW/FLOW-CUMLOC//1DAY/%s/' % fPart).getData().values
OAHE_CumLocal = dssFile.read('//OAHE-INFLOW/FLOW-CUMLOC//1DAY/%s/' % fPart).getData().values

for i in range(len(GARR_CumLocal)):
    if FTPK_CumLocal[i] < -100000:
        FTPK_CumLocal[i] = FTPK_CumLocal[i - 1]
    if GARR_CumLocal[i] < -100000:
        GARR_CumLocal[i] = GARR_CumLocal[i - 1]
    if OAHE_CumLocal[i] < -100000:
        OAHE_CumLocal[i] = OAHE_CumLocal[i - 1]

shared = ResSim.getWatershed()
module = ClientAppWrapper.getCurrentModule()
if module.getName() != 'Simulation':
    raise AssertionError, 'ResSim in %s module. Simulation module is required.' % module
shared = str(shared)

```

```
filePath = shared + '/shared/Input_Data.dss'
filePath = filePath.replace('/', '\\')
```

```
# Project Dictionary
```

```
projectDict = {
    'FTPK' : 'Fort Peck Lake',
    'GARR' : 'Lake Sakakawea',
    'OAHE' : 'Lake Oahe',
    'BEND' : 'Lake Sharpe',
    'FTRA' : 'Lake Francis Case',
    'GAPT' : 'Lewis and Clark Lake'
}
```

```
# Routing Coefficients
```

```
coeffDict = {
    'FTPK' : [0.237, 0.444, 0.319], # FTPK to GARR coefficients
    'GARR' : [0.057, 0.503, 0.440], # GARR to OAHE coefficients
    'OAHE' : [0.766, 0.234], # OAHE to BEND coefficients
    'BEND' : [0.647, 0.353], # BEND to FTRA coefficients
    'FTRA' : [0.005, 0.637, 0.358] # FTRA to GAPT coefficients
}
```

```
# -----
# Set pool elevation when drought and flood releases start
# -----
```

```
# Drought elevations trigger drought minimum releases specified in Release Patterns dictionary
FTPK_Drought_Elevation, GARR_Drought_Elevation, OAHE_Drought_Elevation = 2195.0, 1785.0, 1545.0
# Flood elevations trigger floodDroughtReleases Function
FTPK_Flood_Elevation, GARR_Flood_Elevation, OAHE_Flood_Elevation = 2244.0, 1845.0, 1616.0
# Lower and Upper flood elevation constraints are used in the floodDroughtReleases Function
FTPK_Lower_Flood_Elev_Constraint, FTPK_Upper_Flood_Elev_Constraint = 2246.0, 2250.0
GARR_Lower_Flood_Elev_Constraint, GARR_Upper_Flood_Elev_Constraint = 1850.0, 1854.0
```

```
# -----
# Balance/Unbalance
# -----
```

```
# Set to 0 to balance the upstream reservoirs. If unbalancing is needed, set equal to the amount of feet the reservoirs will be unbalanced
Balance_Factor = 0
```

```
# Unbalancing starting schedule:
```

```
# To start, set FTPK = Low, GARR = Float, and OAHE = High. This will set FTPK = High, GARR = Low, and OAHE = Float in the first year of the simulation
```

```
Unbalancing_Schedule = {
    'FTPK' : 'High',
    'GARR' : 'Low',
    'OAHE' : 'Float'
}
```

```
# FTPK and GARR monthly release schedule
```

```
Original_Monthly_Release = {
    'Month': ['MAR_31', 'APR_30', 'MAY_14', 'MAY_31', 'JUN_30', 'JUL_31', 'AUG_31', 'SEP_14',
    'SEP_30', 'OCT_31', 'NOV_30', 'DEC_31', 'JAN_31', 'FEB_28'],
    'FTPK' : {
        'Standard' : [7500, 7500, 7500, 10400, 10400, 10400, 10400, 8500, 8500,
        8500, 8500, 10800, 10800, 10800], # Monthly releases under normal operating conditions
        'Min' : [6800, 6800, 6800, 6400, 6400, 6400, 6400, 4500, 4500,
        4500, 4500, 6800, 6800, 6800], # Minimum monthly release under normal operating conditions
        'Max' : [13000, 13000, 13000, 14000, 14000, 14000, 14000, 14000, 14000,
        14000, 14000, 13000, 13000, 13000], # Maximum monthly release under normal operating conditions
        'Drought_Min' : [3000, 3000, 3000, 5000, 5000, 5000, 5000, 3000, 3000,
        3000, 3000, 5000, 5000, 5000], # Minimum monthly release under drought conditions
        'Flood_Max' : [13000, 13000, 13000, 14000, 14000, 14000, 14000, 14000, 14000,
        14000, 14000, 13000, 13000, 13000], # Maximum monthly release under minor flooding conditions
        'Extreme_Flood_Max' : [66000, 66000, 66000, 66000, 66000, 66000, 66000, 66000, 66000,
        66000, 66000, 66000, 66000, 66000] # Maximum monthly release under extreme flooding
        conditions
    },
```

```
    'GARR': {
        'Standard' : [23100, 23100, 23100, 25300, 25300, 25300, 25300, 20000, 20000,
        20000, 20000, 20000, 23100, 23100], # Monthly releases under normal operating conditions
        'Min' : [12000, 12000, 12000, 15000, 15000, 15000, 15000, 15000, 9000,
        9000, 9000, 12000, 12000, 12000], # Minimum monthly release under normal operating conditions
        'Max' : [25000, 25000, 25000, 36000, 36000, 36000, 36000, 36000, 36000,
        36000, 36000, 25000, 25000, 25000], # Maximum monthly release under normal operating conditions
        'Drought_Min' : [10000, 10000, 10000, 10000, 10000, 10000, 10000, 9000, 9000,
        9000, 9000, 12000, 12000, 12000], # Minimum monthly release under drought conditions
        'Flood_Max' : [36000, 36000, 36000, 41000, 41000, 41000, 41000, 41000, 41000,
        41000, 41000, 25000, 25000, 25000], # Maximum monthly release under minor flooding conditions
        'Extreme_Flood_Max' : [151000, 151000, 151000, 151000, 151000, 151000, 151000, 151000, 151000,
        151000, 151000, 151000, 151000, 151000] # Maximum monthly release under extreme
        flooding conditions
    }
}
```

```
# -----
# Runoff Forecasts
# -----
```

```

# Month conversions
monDict = {
    1 : ['Jan', 'January'],
    2 : ['Feb', 'February'],
    3 : ['Mar', 'March'],
    4 : ['Apr', 'April'],
    5 : ['May', 'May'],
    6 : ['Jun', 'June'],
    7 : ['Jul', 'July'],
    8 : ['Aug', 'August'],
    9 : ['Sep', 'September'],
    10 : ['Oct', 'October'],
    11 : ['Nov', 'November'],
    12 : ['Dec', 'December']
}

# Project identifiers and names (BEND/Lake Sharpe is included in FTRA/Lake Francis Case)
projects = [
    ["FTPK", "Fort Peck Lake"],
    ["GARR", "Lake Sakakawea"],
    ["OAHE", "Lake Oahe"],
    ["FTRA", "Lake Francis Case"],
    ["GAPT", "Lewis and Clark Lake"]
]

projectIds, projectNames = zip(*projects)

# Normal runoff values for months not included in DRM forecast files. Normal values taken from CY 2014 Runoff Forecast
forecastNormalRunoffMonths = ['Aug', 'Sep', 'Oct', 'Nov', 'Dec']
#
# Location      Aug  Sep  Oct  Nov  Dec  Jan  Feb
normalRunoff = {
    'FTPK' : [356, 330, 380, 381, 327, 312, 361],
    'GARR' : [609, 448, 527, 393, 249, 261, 355],
    'OAHE' : [71, 110, 72, 67, 1, 12, 95],
    'FTRA' : [41, 37, 5, 4, 12, 28, 54],
    'GAPT' : [116, 111, 120, 118, 100, 100, 132]
}

# Sum of the normal runoff through the following February
# Dictionary Format = Project : Starting Month : Total Runoff through February
EOYnormalRunoff = {}
for projectId in projectIds:
    for m in range(len(forecastNormalRunoffMonths)):
        runoff = sum(normalRunoff[projectId][m:])
        EOYnormalRunoff.setdefault(projectId, {}).setdefault(forecastNormalRunoffMonths[m], runoff)

# DRM forecasts
sharedDirectory = ResSim.getWatershed().getSharedDirectory()

months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul']
months.append(["%2.2d" % (i+1) for i in range(len(months[0]))])
fcstZipfileName = os.path.join(sharedDirectory, 'DRM_Forecast_text.zip')

fcstFlows = {}
zf = ZipFile(fcstZipfileName)
for itemname in zf.namelist():
    if len(itemname) != 10 or not itemname.lower().startswith("fc") or not itemname.lower().endswith(".txt"):
        outputDebug(currentRuntimestep, debug, 'Ignoring zipped file %s' % itemname)
        continue
    try :
        year = int(itemname[2:6])
    except :
        outputDebug(debug, 'Ignoring zipped file %s' % itemname)
        continue

    lines = zf.read(itemname).strip().split("\n")
    for i in range(len(lines)):
        fields = lines[i].split()
        if i == 0:
            assert not len(fields) < 3 and fields[0] == "FORECAST" and fields[2] == "KA-F"
            if int(fields[1]) != year:
                outputDebug(currentRuntimestep, debug, 'WARNING : ' + 'Zipped file %s indicates year %s in header' %
                    (itemname, fields[1]))
        elif i == 1:
            assert not (len(fields) < 7 and fields[2:7] == list(projectIds))
        else:
            assert not len(fields) < 7 and fields[0] in months[0]
            fcstMonth = fields[0]
            mon, yr = fields[1].split("-")
            assert mon in months[1]
            if int(yr) != year:
                outputDebug(currentRuntimestep, debug, 'WARNING : ' + 'Zipped file %s indicates year %s at line %d' %
                    (itemname, yr, i+1))

            for j in range(len(projectIds)):
                try:
                    val = float(fields[2+j])
                except:
                    outputDebug(currentRuntimestep, debug, 'WARNING : ' +
                        'Invalid discharge value (%s) for project %s at line %d in zipped file %s, using 0' % \
                        (fields[2+j], projectIds[j], i+1, itemname))
                    val = 0

```

```

                fcstFlows.setdefault(year, {}).setdefault(projectIds[j], {}).setdefault(fcstMonth, []).append(val)
zf.close()

# Calculate EOY runoff forecasts
# Process the DRM forecast runoff into annual totals for forecast month through Feb of the following year for each project. Aug-Dec runoff
# values are normal runoff values
EOY_Runoff = {}
EOYforecastRunoffMonths = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']
for projectId in projectIds:
    for year in range(1898, max(fcstFlows.keys()) + 1, 1): # year will be equal to the year of the last DRM forecasting file
        for month in EOYforecastRunoffMonths:
            if month == 'Mar' or month == 'Apr' or month == 'May' or month == 'Jun' or month == 'Jul':
                runoff = sum(fcstFlows[year][projectId][month]) + EOYnormalRunoff[projectId]['Aug']
                EOY_Runoff.setdefault(projectId, {}).setdefault(year, {}).setdefault(month, runoff)
            elif month == 'Aug' or month == 'Sep' or month == 'Oct' or month == 'Nov' or month == 'Dec':
                runoff = EOYnormalRunoff[projectId][month]
                EOY_Runoff.setdefault(projectId, {}).setdefault(year, {}).setdefault(month, runoff)
            elif month == 'Jan':
                runoff = sum(fcstFlows[year][projectId][month]) - sum(fcstFlows[year][projectId][month][2:])
                EOY_Runoff.setdefault(projectId, {}).setdefault(year, {}).setdefault(month, runoff)
            elif month == 'Feb':
                runoff = sum(fcstFlows[year][projectId][month]) - sum(fcstFlows[year][projectId][month][1:])
                EOY_Runoff.setdefault(projectId, {}).setdefault(year, {}).setdefault(month, runoff)

# -----
# Elevation-Storage Relationships
# -----

# FTPK Elev-Stor
reservoirName = 'Fort Peck Lake'
reservoirElement = network.findReservoir(reservoirName)
storageElement = reservoirElement.getStorageFunction()
storageTable = storageElement.getElevationStorageValues()
elevValues = storageTable.getXArray()
storValues = storageTable.getYArray()
FTPK_StorElevTable, FTPK_ElevStorTable = PairedValuesExt(), PairedValuesExt()
FTPK_StorElevTable.setArrays(elevValues, storValues)
FTPK_ElevStorTable.setArrays(storValues, elevValues)

# GARR Elev-Stor
reservoirName = 'Lake Sakakawea'
reservoirElement = network.findReservoir(reservoirName)
storageElement = reservoirElement.getStorageFunction()
storageTable = storageElement.getElevationStorageValues()
elevValues = storageTable.getXArray()
storValues = storageTable.getYArray()
GARR_StorElevTable, GARR_ElevStorTable = PairedValuesExt(), PairedValuesExt()
GARR_StorElevTable.setArrays(elevValues, storValues)
GARR_ElevStorTable.setArrays(storValues, elevValues)

# OAHE Elev-Stor
reservoirName = 'Lake Oahe'
reservoirElement = network.findReservoir(reservoirName)
storageElement = reservoirElement.getStorageFunction()
storageTable = storageElement.getElevationStorageValues()
elevValues = storageTable.getXArray()
storValues = storageTable.getYArray()
OAHE_StorElevTable, OAHE_ElevStorTable = PairedValuesExt(), PairedValuesExt()
OAHE_StorElevTable.setArrays(elevValues, storValues)
OAHE_ElevStorTable.setArrays(storValues, elevValues)

# BEND Elev-Stor
reservoirName = 'Lake Sharpe'
reservoirElement = network.findReservoir(reservoirName)
storageElement = reservoirElement.getStorageFunction()
storageTable = storageElement.getElevationStorageValues()
elevValues = storageTable.getXArray()
storValues = storageTable.getYArray()
BEND_StorElevTable, BEND_ElevStorTable = PairedValuesExt(), PairedValuesExt()
BEND_StorElevTable.setArrays(elevValues, storValues)
BEND_ElevStorTable.setArrays(storValues, elevValues)

# FTRA Elev-Stor
reservoirName = 'Lake Francis Case'
reservoirElement = network.findReservoir(reservoirName)
storageElement = reservoirElement.getStorageFunction()
storageTable = storageElement.getElevationStorageValues()
elevValues = storageTable.getXArray()
storValues = storageTable.getYArray()
FTRA_StorElevTable, FTRA_ElevStorTable = PairedValuesExt(), PairedValuesExt()
FTRA_StorElevTable.setArrays(elevValues, storValues)
FTRA_ElevStorTable.setArrays(storValues, elevValues)

# GAPT Elev-Stor
reservoirName = 'Lewis and Clark Lake'
reservoirElement = network.findReservoir(reservoirName)
storageElement = reservoirElement.getStorageFunction()
storageTable = storageElement.getElevationStorageValues()

```

```

elevValues = storageTable.getXArray()
storValues = storageTable.getYArray()
GAPT_StorElevTable, GAPT_ElevStorTable = PairedValuesExt(), PairedValuesExt()
GAPT_StorElevTable.setArrays(elevValues, storValues)
GAPT_ElevStorTable.setArrays(storValues, elevValues)

```

```

# -----
# Conversions
# -----

```

```

CFS_to_ACRE_FT = 86400.0/43560.0
ACRE_FT_to_CFS = 43560.0/86400.0

```

```

# -----
# init info that is passed to runRuleScript()
# -----

```

```

initInfo = {
    'ACRE_FT_to_CFS'           : ACRE_FT_to_CFS,
    'adjustMultiUseStor'      : adjustMultiUseStor,
    'Alt_Name'                : Alt_Name,
    'balanceStorage'         : balanceStorage,
    'Balance_Factor'         : Balance_Factor,
    'BEND_ElevStorTable'     : BEND_ElevStorTable,
    'BEND_StorElevTable'     : BEND_StorElevTable,
    'coeffDict'              : coeffDict,
    'CFS_to_ACRE_FT'         : CFS_to_ACRE_FT,
    'currentReleaseMonth'    : currentReleaseMonth,
    'debug'                   : debug,
    'EOY_Runoff'             : EOY_Runoff,
    'fcstFlows'              : fcstFlows,
    'filePath'               : filePath,
    'floodDroughtReleases'   : floodDroughtReleases,
    'forecastPoolElevations' : forecastPoolElevations,
    'FTRA_ElevStorTable'     : FTRA_ElevStorTable,
    'FTRA_StorElevTable'     : FTRA_StorElevTable,
    'FTPK_CumLocal'          : FTPK_CumLocal,
    'FTPK_Drought_Elevation' : FTPK_Drought_Elevation,
    'FTPK_ElevStorTable'     : FTPK_ElevStorTable,
    'FTPK_Flood_Elevation'   : FTPK_Flood_Elevation,
    'FTPK_Lower_Flood_Elev_Constraint' : FTPK_Lower_Flood_Elev_Constraint,
    'FTPK_StorElevTable'     : FTPK_StorElevTable,
    'FTPK_Upper_Flood_Elev_Constraint' : FTPK_Upper_Flood_Elev_Constraint,
    'GAPT_ElevStorTable'     : GAPT_ElevStorTable,
    'GAPT_StorElevTable'     : GAPT_StorElevTable,
    'GARR_CumLocal'          : GARR_CumLocal,
    'GARR_Drought_Elevation' : GARR_Drought_Elevation,
    'GARR_ElevStorTable'     : GARR_ElevStorTable,
    'GARR_Flood_Elevation'   : GARR_Flood_Elevation,
    'GARR_Lower_Flood_Elev_Constraint' : GARR_Lower_Flood_Elev_Constraint,
    'GARR_StorElevTable'     : GARR_StorElevTable,
    'GARR_Upper_Flood_Elev_Constraint' : GARR_Upper_Flood_Elev_Constraint,
    'holdSummerRelease'     : holdSummerRelease,
    'janJulSysRunoff'        : janJulSysRunoff,
    'julianDay'              : julianDay,
    'mar1PercentFull'        : mar1PercentFull,
    'monDict'                : monDict,
    'monthlyAccumRunoff'     : monthlyAccumRunoff,
    'OAHE_CumLocal'          : OAHE_CumLocal,
    'OAHE_Drought_Elevation' : OAHE_Drought_Elevation,
    'OAHE_ElevStorTable'     : OAHE_ElevStorTable,
    'OAHE_Flood_Elevation'   : OAHE_Flood_Elevation,
    'OAHE_StorElevTable'     : OAHE_StorElevTable,
    'outputDebug'           : outputDebug,
    'projectDict'            : projectDict,
    'Original_Monthly_Release' : Original_Monthly_Release,
    'remainingAnnualRunoff'  : remainingAnnualRunoff,
    'releaseVolume'          : releaseVolume,
    'retrieveModelVariables' : retrieveModelVariables,
    'retrieveStateVariables' : retrieveStateVariables,
    'routeFlow'              : routeFlow,
    'scaleMinMaxRelease'     : scaleMinMaxRelease,
    'shared'                 : shared,
    'targetStorages'         : targetStorages,
    'unbalanceReleaseAdj'    : unbalanceReleaseAdj,
    'Unbalancing_Schedule'   : Unbalancing_Schedule,
    'unbalancingSchedule'    : unbalancingSchedule
}

```

```

currentRule.varPut('initInfo', initInfo)

# return Constants.TRUE if the initialization is successful
# and Constants.FALSE if it failed. Returning Constants.FALSE
# will halt the compute.

return Constants.TRUE

```

```

# runRuleScript() is the entry point that is called during the
# compute.

```

```

#
# currentRule is the rule that holds this script
# network is the ResSim network
# currentRuntimestep is the current Run Time Step
def runRuleScript(currentRule, network, currentRuntimestep):

    # -----
    # Retrieve init info from initRuleScript() and set equal to variables
    # -----

    initInfo = currentRule.varGet('initInfo')

    adjustMultiUseStor          = initInfo['adjustMultiUseStor']
    Balance_Factor              = initInfo['Balance_Factor']
    balanceStorage              = initInfo['balanceStorage']
    coeffDict                   = initInfo['coeffDict']
    currentReleaseMonth         = initInfo['currentReleaseMonth']
    debug                       = initInfo['debug']
    EOY_Runoff                  = initInfo['EOY_Runoff']
    fcstFlows                   = initInfo['fcstFlows']
    filePath                    = initInfo['filePath']
    floodDroughtReleases        = initInfo['floodDroughtReleases']
    forecastPoolElevations      = initInfo['forecastPoolElevations']
    FTPK_CumLocal               = initInfo['FTPK_CumLocal']
    FTPK_Drought_Elevation      = initInfo['FTPK_Drought_Elevation']
    FTPK_ElevStorTable          = initInfo['FTPK_ElevStorTable']
    FTPK_Flood_Elevation        = initInfo['FTPK_Flood_Elevation']
    FTPK_Lower_Flood_Elev_Constraint = initInfo['FTPK_Lower_Flood_Elev_Constraint']
    FTPK_StorElevTable          = initInfo['FTPK_StorElevTable']
    FTPK_Upper_Flood_Elev_Constraint = initInfo['FTPK_Upper_Flood_Elev_Constraint']
    GARR_CumLocal               = initInfo['GARR_CumLocal']
    GARR_Drought_Elevation      = initInfo['GARR_Drought_Elevation']
    GARR_ElevStorTable          = initInfo['GARR_ElevStorTable']
    GARR_Flood_Elevation        = initInfo['GARR_Flood_Elevation']
    GARR_Lower_Flood_Elev_Constraint = initInfo['GARR_Lower_Flood_Elev_Constraint']
    GARR_StorElevTable          = initInfo['GARR_StorElevTable']
    GARR_Upper_Flood_Elev_Constraint = initInfo['GARR_Upper_Flood_Elev_Constraint']
    holdSummerRelease           = initInfo['holdSummerRelease']
    janJulSysRunoff             = initInfo['janJulSysRunoff']
    julianDay                   = initInfo['julianDay']
    mar1PercentFull             = initInfo['mar1PercentFull']
    monDict                     = initInfo['monDict']
    monthlyAccumRunoff          = initInfo['monthlyAccumRunoff']
    OAHE_CumLocal               = initInfo['OAHE_CumLocal']
    OAHE_Drought_Elevation      = initInfo['OAHE_Drought_Elevation']
    OAHE_Flood_Elevation        = initInfo['OAHE_Flood_Elevation']
    OAHE_StorElevTable          = initInfo['OAHE_StorElevTable']
    outputDebug                 = initInfo['outputDebug']
    releaseVolume               = initInfo['releaseVolume']
    remainingAnnualRunoff       = initInfo['remainingAnnualRunoff']
    retrieveModelVariables      = initInfo['retrieveModelVariables']
    retrieveStateVariables       = initInfo['retrieveStateVariables']
    routeFlow                   = initInfo['routeFlow']
    scaleMinMaxRelease          = initInfo['scaleMinMaxRelease']
    shared                       = initInfo['shared']
    targetStorages              = initInfo['targetStorages']
    unbalanceReleaseAdj         = initInfo['unbalanceReleaseAdj']
    Unbalancing_Schedule        = initInfo['Unbalancing_Schedule']
    unbalancingSchedule         = initInfo['unbalancingSchedule']

    # create new Operation Value (OpValue) to return
    opValue = OpValue()

    # -----
    # Input Data
    # -----

    # Simulation Time & Step Info
    rtw = currentRuntimestep.getRunTimeWindow() # Run time window
    startRtw = rtw.getStartTime() # Start time of simulation
    lookbackRtw = rtw.getLookbackTime() # Start of lookback period
    endRtw = rtw.getEndTime() # End time of simulation
    endRtwString = rtw.getEndTimeString() # String of end time of simulation
    totalSteps = rtw.getNumSteps() # Total number of steps in the simulation
    curStep = currentRuntimestep.getStep() # Current step
    year = currentRuntimestep.getHecTime().year() # Current year
    mon = currentRuntimestep.getHecTime().month() # Current month
    day = currentRuntimestep.getHecTime().day() # Current day
    jDay = julianDay(mon, day) # Current julian day
    numLookbackSteps = rtw.getNumLookbackSteps() # Number of lookback steps in simulation
    simStartStep = numLookbackSteps + 1 # First step after the lookback period
    PassCounter = network.getComputePassCounter() # Pass number of simulation
    PassNumber = PassCounter + 1 # Pass number starts at 0 so add 1 to start it at 1

    # DSS Data
    # Since the forecasts always use Mar 1 of the following year as the end date when the upper 3 reservoirs need to reach their target storages
and
    # the end of the simulation may not be Mar 1, set the end year equal to the following year for all forecast years except for the last year of the
USACE—Omaha District
DRAFT

```

```

# simulation.
dssFile = HecDss.open(filePath)
startWindow = '%02d %s %.0f, 00:00' % (day, monDict[mon][0], year)
if curStep == simStartStep or (mon == 3 and day == 1):
    End_Year = endRtwString[(len(endRtwString) - 9):(len(endRtwString) - 5)]
    endWindow = '01 Mar %.0f, 00:00' % (year + 1)
    currentRule.varPut('endWindow', endWindow)
endWindow = currentRule.varGet('endWindow')
dssFile.setTimeWindow(startWindow, endWindow) # Open dss file with a specified time window
# GAPT Specified Releases calculated from Downstream Model
GAPT_Specified_Release = dssFile.read('/MISSOURI RIVER/GAPT/FLOW-OUT//1DAY/%s: FORECAST RELEASE/' %
initInfo['Alt_Name']).getData().values

# -----
# Main Script
# -----

if PassNumber < 3:
    opValue.init(OpRule.RULETYPE_MIN, 0)
else:
    # -----
    # Retrieve model and state variables needed for the script in a dictionary
    # -----

    Model_Variable = retrieveModelVariables(currentRuntimestep)
    State_Variable = retrieveStateVariables(currentRuntimestep)

    # -----
    # Release Patterns
    # -----

    if curStep == simStartStep or day == 1 or day == 15:
        # Reset Monthly_Releases to the original release pattern
        Monthly_Releases = copy.deepcopy(initInfo['Original_Monthly_Release'])
        currentRule.varPut('Monthly_Releases', Monthly_Releases)

        # Determine what the current scheduled month should be used for releases
        Monthly_Releases = currentRule.varGet('Monthly_Releases')
        Schedule_Index = currentReleaseMonth(jDay, Monthly_Releases)

    # -----
    # Unbalancing schedule for upcoming runoff season
    # -----

    if day == 28 and mon == 2:
        unbalancingSchedule(initInfo['Unbalancing_Schedule'])

    # -----
    # Monthly Accumulated Incremental Runoff
    # -----

    FTPK_Accum_Stor, GARR_Accum_Stor, OAHE_Accum_Stor, BEND_and_FTFR_Accum_Stor, GAPT_Accum_Stor =
monthlyAccumRunoff(currentRuntimestep,
    Model_Variable['FTPK_Previous_Inflow'], GARR_CumLocal[curStep - 1], OAHE_CumLocal[curStep - 1],
Model_Variable['BEND_Previous_Local_Inflow'], Model_Variable['FTFR_Previous_Local_Inflow'],
    Model_Variable['GAPT_Previous_Local_Inflow'])

    # -----
    # Calculate Jan-Jul System Runoff to Adjust Normal Forecasted Runoff
    # -----

    janJulSysRunoff(currentRuntimestep, Model_Variable['FTPK_Previous_Inflow'], GARR_CumLocal[curStep - 1],
OAHE_CumLocal[curStep - 1], \
    Model_Variable['BEND_Previous_Local_Inflow'], Model_Variable['FTFR_Previous_Local_Inflow'],
Model_Variable['GAPT_Previous_Local_Inflow'])

    # -----
    # Balance/unbalance upper 3 reservoirs
    # -----

    # Determine what the current scheduled month should be used for releases
    Monthly_Releases = currentRule.varGet('Monthly_Releases')
    Schedule_Index = currentReleaseMonth(jDay, Monthly_Releases)

    if curStep == simStartStep or day == 1 or day == 15 or ((day == 7 or day == 21) and (Model_Variable['FTPK_Previous_Elev'] >=
FTPK_Flood_Elevation or \
    Model_Variable['GARR_Previous_Elev'] >= GARR_Flood_Elevation or Model_Variable['OAHE_Previous_Elev'] >=
OAHE_Flood_Elevation)):
        # -----
        # Calculate Remaining Annual Incremental Runoff
        # -----

        FTPK_Remaining_Runoff, GARR_Remaining_Runoff, \
        OAHE_Remaining_Runoff, BEND_and_FTFR_Remaining_Runoff, \
        GAPT_Remaining_Runoff = remainingAnnualRunoff(currentRuntimestep, FTPK_Accum_Stor, GARR_Accum_Stor,
OAHE_Accum_Stor, BEND_and_FTFR_Accum_Stor, GAPT_Accum_Stor)

```

```

# -----
# Calculate Upper System Mar 1 Carryover Storage (FTPK, GARR, and OAHE)
# -----

Upper_System_Percent_Full, Total_GAPT_Release_Volume = mar1PercentFull(currentRuntimestep,
GAPT_Specified_Release, FTPK_Remaining_Runoff,
    GARR_Remaining_Runoff, OAHE_Remaining_Runoff, BEND_and_FTRA_Remaining_Runoff,
GAPT_Remaining_Runoff, Model_Variable['FTPK_Multiple_Use_Stor'],
    Model_Variable['FTPK_Permanent_Stor'], Model_Variable['GARR_Multiple_Use_Stor'],
Model_Variable['GARR_Permanent_Stor'], Model_Variable['OAHE_Multiple_Use_Stor'], Model_Variable['OAHE_Permanent_Stor'],
Model_Variable['FTPK_Previous_Stor'],
    Model_Variable['GARR_Previous_Stor'], Model_Variable['OAHE_Previous_Stor'],
Model_Variable['BEND_Previous_Stor'], Model_Variable['FTRA_Previous_Stor'], Model_Variable['GAPT_Previous_Stor'])
currentRule.varPut('Upper_System_Percent_Full', Upper_System_Percent_Full)

# -----
# Establish adjusted multiple use storages for upper 3 reservoirs
# -----

if curStep == simStartStep or (mon == 3 and day == 1):
    adjustMultiUseStor(currentRuntimestep, Unbalancing_Schedule, Model_Variable['FTPK_Previous_Elev'],
Model_Variable['GARR_Previous_Elev'], Model_Variable['OAHE_Previous_Elev'], Balance_Factor,
    Model_Variable['FTPK_Multiple_Use_Elev'], Model_Variable['GARR_Multiple_Use_Elev'],
Model_Variable['OAHE_Multiple_Use_Elev'], State_Variable)

    FTPK_Adj_Multiple_Use_Stor = currentRule.varGet('FTPK_Adj_Multiple_Use_Stor')
    GARR_Adj_Multiple_Use_Stor = currentRule.varGet('GARR_Adj_Multiple_Use_Stor')
    OAHE_Adj_Multiple_Use_Stor = currentRule.varGet('OAHE_Adj_Multiple_Use_Stor')

# -----
# Calculate target storages for upper 3 reservoirs
# -----

FTPK_Target_Stor, GARR_Target_Stor, OAHE_Target_Stor = targetStorages(currentRuntimestep,
Upper_System_Percent_Full, FTPK_Adj_Multiple_Use_Stor,
    GARR_Adj_Multiple_Use_Stor, OAHE_Adj_Multiple_Use_Stor, Model_Variable['FTPK_Permanent_Stor'],
Model_Variable['GARR_Permanent_Stor'], Model_Variable['OAHE_Permanent_Stor'],
    Model_Variable['FTPK_Multiple_Use_Stor'], Model_Variable['GARR_Multiple_Use_Stor'],
Model_Variable['OAHE_Multiple_Use_Stor'])

# -----
# Input for balancing/unbalancing
# -----

Monthly_Releases = currentRule.varGet('Monthly_Releases')

# Forecasted Releases through March 1 of next Year
FTPK_EOY_Release_Volume = releaseVolume(jDay, Monthly_Releases, 'FTPK')
FTPK_Mar_01_Forecasted_Stor = Model_Variable['FTPK_Previous_Stor'] + FTPK_Remaining_Runoff -
FTPK_EOY_Release_Volume
GARR_EOY_Release_Volume = releaseVolume(jDay, Monthly_Releases, 'GARR')
GARR_Mar_01_Forecasted_Stor = Model_Variable['GARR_Previous_Stor'] + GARR_Remaining_Runoff +
FTPK_EOY_Release_Volume - GARR_EOY_Release_Volume
OAHE_Mar_01_Forecasted_Stor = Model_Variable['OAHE_Previous_Stor'] + OAHE_Remaining_Runoff +
GARR_EOY_Release_Volume - Total_GAPT_Release_Volume

# -----
# Set FTPK and GARR releases to balance system
# -----

for iteration in range(2):
    if initInfo['Unbalancing_Schedule']['FTPK'] == 'High' or Balance_Factor == 0:
        FTPK_Mar_01_Forecasted_Stor, GARR_Mar_01_Forecasted_Stor = balanceStorage(currentRuntimestep,
jDay, 'FTPK', 'GARR', Model_Variable['FTPK_Previous_Stor'], Model_Variable['GARR_Previous_Stor'],
    FTPK_Target_Stor, GARR_Target_Stor, FTPK_Mar_01_Forecasted_Stor,
GARR_Mar_01_Forecasted_Stor, FTPK_Remaining_Runoff, GARR_Remaining_Runoff,
    Schedule_Index, Monthly_Releases, Total_GAPT_Release_Volume, FTPK_Drought_Elevation,
GARR_Drought_Elevation,
    Model_Variable['FTPK_Multiple_Use_Elev'], Model_Variable['GARR_Multiple_Use_Elev'],
Model_Variable['FTPK_Previous_Elev'], Model_Variable['GARR_Previous_Elev'])

        GARR_EOY_Release_Volume = releaseVolume(jDay, Monthly_Releases, 'GARR')
        OAHE_Mar_01_Forecasted_Stor = Model_Variable['OAHE_Previous_Stor'] + OAHE_Remaining_Runoff +
GARR_EOY_Release_Volume - Total_GAPT_Release_Volume

    elif initInfo['Unbalancing_Schedule']['FTPK'] == 'Float' and Balance_Factor > 0:
        OAHE_Mar_01_Forecasted_Stor, GARR_Mar_01_Forecasted_Stor = balanceStorage(currentRuntimestep,
jDay, 'OAHE', 'GARR', Model_Variable['OAHE_Previous_Stor'], Model_Variable['GARR_Previous_Stor'],
    OAHE_Target_Stor, GARR_Target_Stor, OAHE_Mar_01_Forecasted_Stor,
GARR_Mar_01_Forecasted_Stor, OAHE_Remaining_Runoff, GARR_Remaining_Runoff,
    Schedule_Index, Monthly_Releases, Total_GAPT_Release_Volume, FTPK_Drought_Elevation,
GARR_Drought_Elevation,
    Model_Variable['FTPK_Multiple_Use_Elev'], Model_Variable['GARR_Multiple_Use_Elev'],
Model_Variable['FTPK_Previous_Elev'], Model_Variable['GARR_Previous_Elev'])

        FTPK_EOY_Release_Volume = releaseVolume(jDay, Monthly_Releases, 'FTPK')
        FTPK_Mar_01_Forecasted_Stor = Model_Variable['FTPK_Previous_Stor'] + FTPK_Remaining_Runoff -
FTPK_EOY_Release_Volume

```

```

        elif initInfo['Unbalancing_Schedule']['FTPK'] == 'Low' and Balance_Factor > 0:
            OAHE_Mar_01_Forecasted_Stor, FTPK_Mar_01_Forecasted_Stor = balanceStorage(currentRuntimestep,
jDay, 'OAHE', 'FTPK', Model_Variable['OAHE_Previous_Stor'], Model_Variable['FTPK_Previous_Stor'],
            OAHE_Target_Stor, FTPK_Target_Stor, OAHE_Mar_01_Forecasted_Stor,
FTPK_Mar_01_Forecasted_Stor, OAHE_Remaining_Runoff, FTPK_Remaining_Runoff,
            Schedule_Index, Monthly_Releases, Total_GAPT_Release_Volume, FTPK_Drought_Elevation,
GARR_Drought_Elevation,
            Model_Variable['FTPK_Multiple_Use_Elev'], Model_Variable['GARR_Multiple_Use_Elev'],
Model_Variable['FTPK_Previous_Elev'], Model_Variable['GARR_Previous_Elev'])

            FTPK_EOY_Release_Volume = releaseVolume(jDay, Monthly_Releases, 'FTPK')
            GARR_EOY_Release_Volume = releaseVolume(jDay, Monthly_Releases, 'GARR')
            GARR_Mar_01_Forecasted_Stor = Model_Variable['GARR_Previous_Stor'] + GARR_Remaining_Runoff +
FTPK_EOY_Release_Volume - GARR_EOY_Release_Volume

            outputDebug(currentRuntimestep, debug, 'FTPK Target Mar 1 Storage = %.0f acre-ft' % FTPK_Target_Stor,
'\tForecasted FTPK Mar 1 Storage = %.0f acre-ft' % FTPK_Mar_01_Forecasted_Stor,
            '\n\t\t\t\t\tGARR Target Mar 1 Storage = %.0f acre-ft' % GARR_Target_Stor, '\tForecasted GARR Mar 1 Storage =
%.0f acre-ft' % GARR_Mar_01_Forecasted_Stor,
            '\n\t\t\t\t\tOAHE Target Mar 1 Storage = %.0f acre-ft' % OAHE_Target_Stor, '\tForecasted OAHE Mar 1 Storage =
%.0f acre-ft' % OAHE_Mar_01_Forecasted_Stor)

            # Check OAHE forecasted storage. Adjust FTPK and GARR target storages based on OAHE forecasted storage to
help keep OAHE closer to
            # top of carry over zone
            if OAHE_Mar_01_Forecasted_Stor > OAHE_Target_Stor:
                Storage_Difference = OAHE_Mar_01_Forecasted_Stor - OAHE_Target_Stor
                FTPK_Target_Stor = FTPK_Target_Stor + (Storage_Difference / 2)
                GARR_Target_Stor = GARR_Target_Stor + (Storage_Difference / 2)
                outputDebug(currentRuntimestep, debug, 'Adjust FTPK and GARR target storages by %.0f acre-ft to keep
OAHE closer to its target storage.' % (Storage_Difference / 2))
            else:
                break

            FTPK_Percent_Full = (Model_Variable['FTPK_Previous_Stor'] - Model_Variable['FTPK_Permanent_Stor']) /
(Model_Variable['FTPK_Multiple_Use_Stor'] - Model_Variable['FTPK_Permanent_Stor']) * 100
            GARR_Percent_Full = (Model_Variable['GARR_Previous_Stor'] - Model_Variable['GARR_Permanent_Stor']) /
(Model_Variable['GARR_Multiple_Use_Stor'] - Model_Variable['GARR_Permanent_Stor']) * 100
            OAHE_Percent_Full = (Model_Variable['OAHE_Previous_Stor'] - Model_Variable['OAHE_Permanent_Stor']) /
(Model_Variable['OAHE_Multiple_Use_Stor'] - Model_Variable['OAHE_Permanent_Stor']) * 100

            # -----
            # Additional adjustments to GARR release
            # -----

            # Hold summer releases if possible during nesting season
            if julianDay('MAY_15') < jDay < julianDay('SEP_15') and GARR_Drought_Elevation <
Model_Variable['GARR_Previous_Elev'] < Model_Variable['GARR_Flood_and_Multiple_Use_Elev'] and \
            OAHE_Drought_Elevation < Model_Variable['OAHE_Previous_Elev']:

                holdSummerRelease(currentRuntimestep, 'GARR', jDay, Schedule_Index, Monthly_Releases)
                # Adjust GARR releases based on GARR and OAHE difference in storage
                elif abs(GARR_Percent_Full - OAHE_Percent_Full) > 5 and Balance_Factor == 0:
                    unbalanceReleaseAdj(currentRuntimestep, 'GARR', 'OAHE', GARR_Percent_Full, OAHE_Percent_Full,
Monthly_Releases, Schedule_Index)
                elif abs(GARR_Percent_Full - OAHE_Percent_Full) > 10 and Balance_Factor > 0:
                    unbalanceReleaseAdj(currentRuntimestep, 'GARR', 'OAHE', GARR_Percent_Full, OAHE_Percent_Full,
Monthly_Releases, Schedule_Index)

            # Forecast pool elevations
            Pool_Falling = forecastPoolElevations(currentRuntimestep, curStep, totalSteps, 'GARR', 'FTPK', Monthly_Releases,
Schedule_Index, Model_Variable['GARR_Previous_Stor'],
            GARR_CumLocal, GARR_ElevStorTable)
            # Reduce releases to minimum if operating during a drought or set flood releases
            if (Pool_Falling == 'No' and Model_Variable['GARR_Previous_Elev'] >= GARR_Flood_Elevation) or
Model_Variable['OAHE_Previous_Elev'] >= OAHE_Flood_Elevation or \
            (Model_Variable['GARR_Previous_Elev'] < GARR_Drought_Elevation and Model_Variable['OAHE_Previous_Elev'] >
OAHE_Drought_Elevation):

                floodDroughtReleases(currentRuntimestep, 'GARR', 'FTPK', 'OAHE', Schedule_Index, Monthly_Releases,
GARR_Drought_Elevation, OAHE_Drought_Elevation,
            GARR_Flood_Elevation, GARR_Lower_Flood_Elev_Constraint, GARR_Upper_Flood_Elev_Constraint,
1618.5, GARR_CumLocal, GARR_ElevStorTable,
            Pool_Falling, Model_Variable['GARR_Exclusive_Flood_Elev'])

            # Write summer release to memory
            if jDay == julianDay('MAY_15'):
                currentRule.varPut('GARR_Summer_Release', Monthly_Releases['GARR']['Standard']['Schedule_Index'])

            # -----
            # Additional adjustments to FTPK release
            # -----

            # Hold summer releases if possible during nesting season
            if julianDay('MAY_15') < jDay < julianDay('SEP_15') and FTPK_Drought_Elevation < Model_Variable['FTPK_Previous_Elev']
< Model_Variable['FTPK_Flood_and_Multiple_Use_Elev'] and \
            GARR_Drought_Elevation < Model_Variable['GARR_Previous_Elev']:

```

```

        holdSummerRelease(currentRuntimestep, 'FTPK', jDay, Schedule_Index, Monthly_Releases)
    # Adjust FTPK releases based on FTPK and GARR difference in storage
    elif abs(FTPK_Percent_Full - GARR_Percent_Full) > 5 and Balance_Factor == 0:
        unbalanceReleaseAdj(currentRuntimestep, 'FTPK', 'GARR', FTPK_Percent_Full, GARR_Percent_Full,
Monthly_Releases, Schedule_Index)
    elif abs(FTPK_Percent_Full - GARR_Percent_Full) > 10 and Balance_Factor > 0:
        unbalanceReleaseAdj(currentRuntimestep, 'FTPK', 'GARR', FTPK_Percent_Full, GARR_Percent_Full,
Monthly_Releases, Schedule_Index)

    # Forecast pool elevations
    Pool_Falling = forecastPoolElevations(currentRuntimestep, curStep, totalSteps, 'FTPK', None, Monthly_Releases,
Schedule_Index, Model_Variable['FTPK_Previous_Stor'],
        FTPK_CumLocal, FTPK_ElevStorTable)
    # Reduce releases to minimum if operating during a drought or set flood releases
    if (Pool_Falling == 'No' and Model_Variable['FTPK_Previous_Elev'] >= FTPK_Flood_Elevation) or
Model_Variable['GARR_Previous_Elev'] >= GARR_Flood_Elevation or \
        (Model_Variable['FTPK_Previous_Elev'] < FTPK_Drought_Elevation and Model_Variable['GARR_Previous_Elev'] >
GARR_Drought_Elevation):

        floodDroughtReleases(currentRuntimestep, 'FTPK', None, 'GARR', Schedule_Index, Monthly_Releases,
FTPK_Drought_Elevation, GARR_Drought_Elevation,
            FTPK_Flood_Elevation, FTPK_Lower_Flood_Elev_Constraint, FTPK_Upper_Flood_Elev_Constraint, 1852.0,
FTPK_CumLocal, FTPK_ElevStorTable,
                Pool_Falling, Model_Variable['FTPK_Exclusive_Flood_Elev'])

    # Write summer release to memory
    if jDay == julianDay('MAY_15'):
        currentRule.varPut('FTPK_Summer_Release', Monthly_Releases['FTPK']['Standard'][Schedule_Index])

    currentRule.varPut('Monthly_Releases', Monthly_Releases)

# -----
# Set FTPK and GARR releases
# -----

State_Variable['FTPK_Release_SV'].setValue(currentRuntimestep, Monthly_Releases['FTPK']['Standard'][Schedule_Index])
State_Variable['GARR_Release_SV'].setValue(currentRuntimestep, Monthly_Releases['GARR']['Standard'][Schedule_Index])

# -----
# Write target storages to a state variable
# -----

State_Variable['FTPK_Target_Storage_SV'].setValue(currentRuntimestep, currentRule.varGet('FTPK_Target_Stor'))
State_Variable['GARR_Target_Storage_SV'].setValue(currentRuntimestep, currentRule.varGet('GARR_Target_Stor'))
State_Variable['OAHE_Target_Storage_SV'].setValue(currentRuntimestep, currentRule.varGet('OAHE_Target_Stor'))

# -----
# Write percent of carryover storage to a state variables
# -----

FTPK_Percent_Full = (Model_Variable['FTPK_Previous_Stor'] - Model_Variable['FTPK_Permanent_Stor']) /
(Model_Variable['FTPK_Multiple_Use_Stor'] - Model_Variable['FTPK_Permanent_Stor']) * 100
GARR_Percent_Full = (Model_Variable['GARR_Previous_Stor'] - Model_Variable['GARR_Permanent_Stor']) /
(Model_Variable['GARR_Multiple_Use_Stor'] - Model_Variable['GARR_Permanent_Stor']) * 100
OAHE_Percent_Full = (Model_Variable['OAHE_Previous_Stor'] - Model_Variable['OAHE_Permanent_Stor']) /
(Model_Variable['OAHE_Multiple_Use_Stor'] - Model_Variable['OAHE_Permanent_Stor']) * 100
Upper_System_Percent_Full = currentRule.varGet('Upper_System_Percent_Full') * 100
if Upper_System_Percent_Full > 100:
    Upper_System_Percent_Full = 100

State_Variable['FTPK_Percent_Carryover_SV'].setValue(currentRuntimestep, FTPK_Percent_Full)
State_Variable['GARR_Percent_Carryover_SV'].setValue(currentRuntimestep, GARR_Percent_Full)
State_Variable['OAHE_Percent_Carryover_SV'].setValue(currentRuntimestep, OAHE_Percent_Full)
State_Variable['System_Percent_Carryover_SV'].setValue(currentRuntimestep, Upper_System_Percent_Full)

#####
# Still need to subtract evaporation for the forecasted storage calculations
#####

# set type and value for OpValue
# t0type is one of:
# OpRule.RULETYPE_MAX - maximum flow
# OpRule.RULETYPE_MIN - minimum flow
# OpRule.RULETYPE_SPEC - specified flow
opValue.init(OpRule.RULETYPE_MIN, Monthly_Releases['FTPK']['Standard'][Schedule_Index])

# return the Operation Value.
# return "None" to have no effect on the compute
return opValue

```