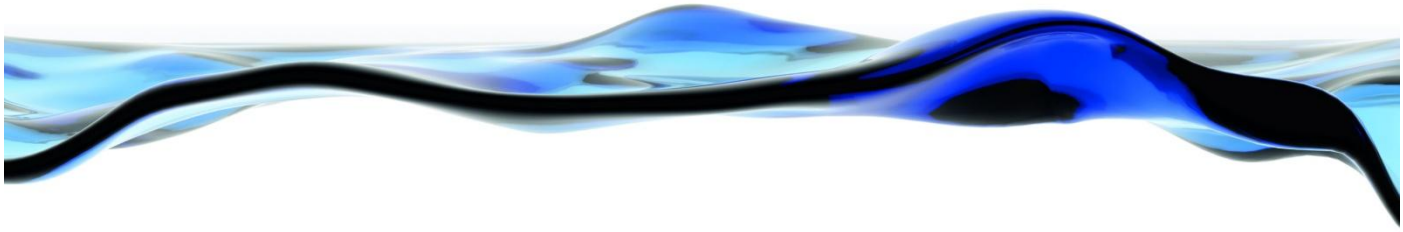
Modelling salt dynamics on the River Murray floodplain in South Australia: Modelling approaches – Appendices

EDITOR

Juliette Woods

CONTRIBUTORS

Flinders University: Tariq Laattoe
CSIRO: Peter Cook
DEWNR: Carl Purczel, Virginia Riches
AWE: Le Dang



Goyder Institute for Water Research
Technical Report Series No. 15/11

GOYDER INSTITUTE FOR WATER RESEARCH

www.goyderinstitute.org

Goyder Institute for Water Research Technical Report Series ISSN: 1839-2725

The Goyder Institute for Water Research is a partnership between the South Australian Government through the Department of Environment, Water and Natural Resources, CSIRO, Flinders University, the University of Adelaide and the University of South Australia. The Institute will enhance the South Australian Government's capacity to develop and deliver science-based policy solutions in water management. It brings together the best scientists and researchers across Australia to provide expert and independent scientific advice to inform good government water policy and identify future threats and opportunities to water security.

The following Associate organisations contributed to this report:

Enquires should be addressed to: Goyder Institute for Water Research
Level 1, Torrens Building
220 Victoria Square, Adelaide, SA, 5000
tel: 08-8303 8952
e-mail: enquiries@goyderinstitute.org

Citation
Woods, J (ed.) 2015, *Modelling salt dynamics on the River Murray floodplain in South Australia: Modelling approaches – Appendices,* Goyder Institute for Water Research Technical Report Series No. 2015/11, Adelaide, South Australia

Copyright
© 2015 Flinders University. To the extent permitted by law, all rights are reserved and no part of this publication covered by copyright may be reproduced or copied in any form or by any means except with the written permission of Flinders University.

Disclaimer
The Participants advise that the information contained in this publication comprises general statements based on scientific research and does not warrant or represent the completeness of any information or material in this publication.

Acknowledgements

This project was funded by the Goyder Institute for Water Research and supported by the SA Department of Environment, Water and Natural Resources (DEWNR) and the National Centre for Groundwater Research and Training (NCGRT).

The project scope was developed by the project team with input from Judith Kirk (DEWNR), Neil Power (DEWNR/Goyder), Wei Yan (DEWNR), Kate Holland (CSIRO), Glen Walker (CSIRO), and Paul Dalby (In Fusion Consulting).

The Project Management Team for the project consisted of: Juliette Woods and Tariq Laattoe of Flinders University, Kate Holland and Peter Cook of CSIRO, and Wei Yan, Matt Gibbs, Linda Vears, and Graham Green of DEWNR.

The Policy Advisory Committee for this project was chaired by Judith Kirk with Linda Vears as Executive Officer, both from DEWNR. Other members were Danni Oliver of the Goyder Institute, Okke Batelaan of Flinders University, and Tony Herbert, Chris Wright, Dragana Zulfic, Tumi Bjornsson and Whendee Young of DEWNR.

The principal external reviewers were Glen Walker (NCGRT) and Anthony Knapton (CloudGMS). Danni Oliver provided comments on the May 2015 draft.

Amanda Nixon of Flinders University and Jason Tan of eResearch SA arranged collaborative data storage.

Nikki Harrington of Flinders University provided sound advice.

Contents

Appendices

Introduction

Juliette Woods

The River Murray is the principal river of the western Murray Basin in southeastern Australia. It is highly unusual as it is a major river system that flows through an extensive landscape with highly saline groundwater. The river is naturally prone to salinity, and this propensity has increased over the past century due to the construction of river locks and the introduction of large-scale land clearance and irrigation. River flow volumes and river level variability have been reduced, while the watertable has risen. The overall impact has been to degrade riverine ecosystems and increase river and floodplain salinity.

Much research has been undertaken to monitor, conceptualise and simulate the dynamics of floodplain salinity. However, there is currently no consistent and comprehensive approach to modelling the interaction of surface water and groundwater in the lower River Murray floodplains. To address this need, in 2014 the Goyder Institute for Water Research commissioned the study *Modelling salt dynamics on the River Murray floodplain in South Australia*, a collaborative research project with contributions from Flinders University, CSIRO Land and Water, and the Department of Environment, Water and Natural Resources (DEWNR). The study area consists of the floodplains of the River Murray in South Australia, from the Border to Morgan, within the South Australian part of the Murray Basin. Due to the breadth of the review required, feedback was sought from a wide range of experts and stakeholders.

The study resulted in three report volumes, all of which are available at the Goyder Institute website: http://goyderinstitute.org/. The first report, *Modelling salt dynamics on the River Murray floodplain in South Australia: Conceptual model, data review and salinity risk approaches* (Woods, 2015a), documents the conceptual model, data review, and salinity risk methodology discussion. The second report, *Modelling salt dynamics on the River Murray floodplain in South Australia: Modelling approaches* (Woods, 2015b) documents the literature review and testing of modelling approaches. This is the third report, which contains the Appendices.

Appendix A provides the Python scripts used In Section 4.3.1 of Woods (2015b). They relate to the interpolation of river stage data and the selection of MODFLOW stress periods based on changes in river stage.

Appendix B provides the FORTRAN programs used in Chapter 5 of Woods (2015b). Note that the amended Source program is not included in this Appendix, as Source is the intellectual property of eWater.

Appendix C provides groundwater model results for Chapter 4 of Woods (2015b).

References

Woods, J (ed.) (2015a) Modelling salt dynamics on the River Murray floodplain in South Australia: Conceptual model, data review and salinity risk approaches. Goyder Institute for Water Research Technical Report Series No. 2015/9, Adelaide, South Australia.

Woods J (ed.) (2015b) Modelling salt dynamics on the River Murray floodplain in South Australia: Modelling approaches, Goyder Institute for Water Research Technical Report Series No. 2015/10, Adelaide, South Australia.

A. Python scripts

Tariq Laattoe

Appendix A1: Stage data gap interpolator

This Python script interpolates missing values in water level data obtained from Water Connect. Note this script may be used to interpolate any form of daily data provided it is stored in a .csv file with dates in column 1 formatted as dd/mm/yyyy and corresponding data in column 2.

The input for the script is discussed above and within the script itself. The output of the script is a completed dataset text file and a figure displaying where the interpolated values occur. The version of Python used when the script was created is 2.7.6.0. A copy and paste of the text below should provide a working script (tested and developed in Spyder).

```
#**********************************************************************#
#Importing libraries
import pylab
from pylab import *
import datetime
from datetime import timedelta
import csv
import os
#**********************************************************************#
#Changing the working directory
os.chdir('C:\\Workspace\\laat0003\\river')
#**********************************************************************#
#Opening and reading csv data file comprising 2 columns
#Column 1 has dates in dd/mm/yyyy format
#Column 2 has stage elevations in meters with 2 decimals
#NOTE this file has no headers or column titles
infile = open('DSL5_1992_2012_original.csv','r')
#read csv into table0
table0 = [row for row in csv.reader(infile)]
#creates empty list of dates and heights
dates , heights = [], []
#populate date list from table0
[dates.append(table0[r][0]) for r in range(0,len(table0))]
#populate heights from table0
[heights.append(table0[r][1]) for r in range(0,len(table0))]
#change heights from stings to floats
heights = [float(heights[x]) for x in range(0,len(heights))]
#make the heights list into an array
h=np.asarray(heights)
#convert the date strings into date objects
dates= [datetime.datetime.strptime(dates[r],'%d/%m/%Y').date() for r in range(0,len(dates))]
#**********************************************************************#
#Identifying and Locating the missing dates
#Initialise variables for algorithm
# convert dates list to a set for faster membership tests than lists
date_set=set(dates)
#set a time step to check one day at a time
one_day = timedelta(days=1)
# starting point is the first date in our dates set in which some dates are absent
test_date = dates[0]
#create a list that will contain the missing date objects when test loop is completed
missing_dates=[]
#set a counter for the cell numbers as we loop through the testing
cellnum = 0
```

```
#create a list to contain the cell indices of the missing dates
cells =[]
#loop through sequential dates starting from the first date in our original list
while test_date < dates[-1]:
#conditional statement (if the current date is not in our date set)
    if test_date not in date_set:
#record the current date being tested
        missing_dates.append(test_date)
#record current cell number
        cells.append(cellnum)
#go to the next cell number
    cellnum += 1
#go to the next date defined by our time step one_day
    test_date += one_day
#make a working copy of the heights array in which to enter unknown values
hfill = copy(h)
#insert a nan where there is missing height data using the cell indices from new date entries
for r in range(0,len(cells)):
    hfill = insert(hfill, cells[r], nan)
#create a mask array from the nan entries in the heights array
mask = isnan(hfill)
#copy the working array with the nan values we need it later for plotting
hfill_nan = copy(hfill)
#use linear interpolation to and from nearest known neighbour to replace nan values
hfill[mask] = interp(flatnonzero(mask), flatnonzero(~mask), hfill[~mask])
#create an array with the dates formatted correctly for printing to file
newdates = arange(datetime64(dates[0]), datetime64(dates[-1])+1, dtype='datetime64')
#open a file to write the data to
f = open('River_data_interp.txt','w')
#loop through all values and write them as pairs of date, heights in the named file above
for r in range(0,len(newdates)):
   f.write('{}, {}\n'.format(newdates[r], hfill[r]))
#close the file
f.close()
#plot the data highlighting the interpolated values with red x's
#change labels accordingly
fig, ax1 = matplotlib.pyplot.subplots()
ax1.plot_date(newdates,hfill_nan,fmt='g-', label = 'original')
ax1.plot_date(missing_dates, hfill[mask], fmt='rx', label = 'interpolated')
ax1.legend()
title = 'Downstream Lock 5 1992-2012'
plt.title(title)
plt.xlabel('year')
plt.ylabel('stage (m)')
#Save the file
F=pylab.gcf()
F.set_size_inches(11,5)
F.savefig("%s.png"%title, dpi=150)
```

Appendix A2: River package adaptive stress period creator

This Python script provides the step function stress period setup for river cells in MODFLOW. Note this script only provides the transient data for a single river cell either side of the lock. It uses the downstream and upstream completed datasets for a given lock and determines the time split for stress periods based on yearly means, monthly means, and user-defined parameters for an adaptive method.

The input for the script is discussed above and within the script itself. The output of the script is a yearly mean stress period setup, a monthly mean stress period setup, and an adaptive stress period setup. The user-defined values that govern the adaptive time split for each stress period are modified within the script itself. The version of Python used when the script was created is 2.7.6.0. A copy and paste of the text below should provide a working script (tested and developed in Spyder).

```
#**************************************************************************#
#Importing libraries
import pylab
from pylab import *
import datetime
from math import trunc
from itertools import groupby
import itertools
from operator import itemgetter
import os
import csv
#**************************************************************************#
#declare a grouping function used to group data for stress periods
def mygrouper(n, iterable):
    args = [iter(iterable)] * n
    return ([e for e in t if e != None] for t in itertools.izip_longest(*args))
#**************************************************************************#
#Changing the working directory
os.chdir('C:\\workspace\\laat0003\\river')
#**************************************************************************#
#These input files are the output of the data interpolation
#Opening and reading csv data file comprising 2 columns
#Column 1 has dates in dd/mm/yyyy format
#Column 2 has stage elevations in meters with 2 decimals
#We are dealing with 2 sets of data 1 for upstream lock
#and 1 for downstream lock
#NOTE this script assumes column titles or a 1 line header
infile1 = open('River.csv','r')
infile2 = open('River2.csv','r')
#read csv into table0
table0 = [row for row in csv.reader(infile1)]
table1 = [row for row in csv.reader(infile2)]
#creates empty list of dates and heights
dates , heights, dts = [], [], []
dates1 , heights1, dts1 = [], [], []
#populate date list from table0
[dates.append(table0[r][0]) for r in range(1,len(table0))]
[dates1.append(table0[r][0]) for r in range(1,len(table1))]
[dts.append(table0[r][0]) for r in range(1,len(table0))]
[dts1.append(table0[r][0]) for r in range(1,len(table1))]
#populate heights from table0
[heights.append(table0[r][1]) for r in range(1,len(table0))]
[heights1.append(table1[r][1]) for r in range(1,len(table1))]
#change heights from stings to floats
heights = [float(heights[x]) for x in range(0,len(heights))]
heights1 = [float(heights1[x]) for x in range(0,len(heights1))]
#make the heights list into an array
```

```
h=asarray(heights)
h1=asarray(heights1)
#convert the date strings into date objects
dates= [datetime.datetime.strptime(dates[r],'%d/%m/%Y').date() for r in range(0,len(dates))]
dates1= [datetime.datetime.strptime(dates1[r],'%d/%m/%Y').date() for r in range(0,len(dates1))]
#create lists to populate with dates conducive to sorting month and year
ls, ls1, yls, yls1 = [], [], [], []
#populate the lists
for r in range(0,len(dates)):
    tmp = []
    tmp1 = []
    tmp2 = []
    tmp3 = []
    tmp.append(str(dates[r])[:-3])
    tmp1.append(str(dates1[r])[:-3])
    tmp2.append(str(dates[r])[:-6])
    tmp3.append(str(dates1[r])[:-6])
    tmp.append(dates[r])
    tmp1.append(dates1[r])
    tmp2.append(dates[r])
    tmp3.append(dates1[r])
    tmp.append(heights[r])
    tmp1.append(heights1[r])
    tmp2.append(heights[r])
    tmp3.append(heights1[r])
    yls.append(tmp2)
    yls1.append(tmp3)
    ls.append(tmp)
    ls1.append(tmp1)
#create groups to sort months and years
groups = []
uniquekeys =[]
for k,g in groupby(ls,key=itemgetter(0)):
  groups.append(list(g))
  uniquekeys.append(k)
groups1 = []
uniquekeys1 =[]
for k,g in groupby(ls1,key=itemgetter(0)):
  groups1.append(list(g))
  uniquekeys1.append(k)
ygroups = []
yuniquekeys =[]
for k,g in groupby(yls,key=itemgetter(0)):
  ygroups.append(list(g))
  yuniquekeys.append(k)
ygroups1 = []
yuniquekeys1 =[]
for k,g in groupby(yls1,key=itemgetter(0)):
  ygroups1.append(list(g))
  yuniquekeys1.append(k)
#create lists to work out means, max and min for a month
tmp = []
mtmp = []
mm = []
for r in range(0,len(groups)):
  tmp =[]
  mtmp =[]
  for i in range(0,len(groups[r])):
    tmp.append(groups[r][i][2])
```

```
    m = round(sum(tmp)/len(tmp),2)
    mtmp.append(groups[r][0][0])
    mtmp.append(m)
    mtmp.append(max(tmp))
    mtmp.append(min(tmp))
    mtmp.append(round(max(tmp)-min(tmp),2))
    mm.append(mtmp)
#same as above but for second dataset
tmp1 = []
mtmp1 = []
mm1 = []
for r in range(0,len(groups1)):
    tmp1 =[]
    mtmp1 =[]
    for i in range(0,len(groups1[r])):
        tmp1.append(groups1[r][i][2])
    m1 = round(sum(tmp1)/len(tmp1),2)
    mtmp1.append(groups1[r][0][0])
    mtmp1.append(m1)
    mtmp1.append(max(tmp1))
    mtmp1.append(min(tmp1))
    mtmp1.append(round(max(tmp1)-min(tmp1),2))
    mm1.append(mtmp1)
#same as above but yearly
tmp2 = []
mtmp2 = []
ym = []
for r in range(0,len(ygroups)):
    tmp2 =[]
    mtmp2 =[]
    for i in range(0,len(ygroups[r])):
        tmp2.append(ygroups[r][i][2])
    m2 = round(sum(tmp2)/len(tmp2),2)
    mtmp2.append(ygroups[r][0][0])
    mtmp2.append(m2)
    mtmp2.append(max(tmp2))
    mtmp2.append(min(tmp2))
    mtmp2.append(round(max(tmp2)-min(tmp2),2))
    ym.append(mtmp2)
#same as above but for the second dataset
tmp3 = []
mtmp3 = []
ym1 = []
for r in range(0,len(ygroups1)):
    tmp3 =[]
    mtmp3 =[]
    for i in range(0,len(ygroups1[r])):
        tmp3.append(ygroups1[r][i][2])
    m3 = round(sum(tmp3)/len(tmp3),2)
    mtmp3.append(ygroups1[r][0][0])
    mtmp3.append(m3)
    mtmp3.append(max(tmp3))
    mtmp3.append(min(tmp3))
    mtmp3.append(round(max(tmp3)-min(tmp3),2))
    ym1.append(mtmp3)
#Open files for writing
#Write the monthly values for downstream data set
f = open('dsmonth_str_per.txt','w')
f.write('Month, mean, period, days\n')
```

```
for r in range(0, len(mm)):
        f.write('{}, {}, {}, {}\n'.format\
        (mm[r][0], mm[r][1], r, len(groups[r])))
f.close()
#write the monthly values for the upstream data set
f = open('usmonth_str_per.txt','w')
f.write('Month, mean, period, days\n')
for r in range(0, len(mm1)):
        f.write('{}, {}, {}, {}\n'.format\
        (mm1[r][0], mm1[r][1], r, len(groups[r])))
f.close()
#write the yearly values for the downstream data set
f = open('dsyear_str_per.txt','w')
f.write('Year, mean, period, days\n')
for r in range(0, len(ym)):
        f.write('{}, {}, {}, {}\n'.format\
        (ym[r][0], ym[r][1], r, len(ygroups[r])))
f.close()
#write yearly values for the upstream dataset
f = open('usyear_str_per.txt','w')
f.write('Year, mean, period, days\n')
for r in range(0, len(ym1)):
        f.write('{}, {}, {}, {}\n'.format\
        (ym1[r][0], ym1[r][1], r, len(ygroups[r])))
f.close()
"""#############################################################################"""
# Modify the following parameters to change the threshold stage heights and monthly splits
# split month by 2
th1, th1sp = 0.1, 2
# split month by 4
th2, th2sp = 0.25, 4
# split month by 5
th3, th3sp = 0.5, 5
# split month by 14 (effectively 2 day periods)
th4, th4sp = 0.75, 14
# NOTE you should not split a month by more than 14 to avoid any possible issues with February
"""#############################################################################"""
#The loop below performs the splits for the months
splitr = []
for r in range(0,len(mm)):
    if mm[r][4]<=th1:
        splitr.append(len(groups[r]))
    elif mm[r][4] > th1 and mm[r][4] <= th2:
        splitr.append(trunc(len(groups[r])/th1sp))
    elif mm[r][4] > th2 and mm[r][4] <= th3:
        splitr.append(trunc(len(groups[r])/th2sp))
    elif mm[r][4] > th3 and mm[r][4] <= th4:
        splitr.append(trunc(len(groups[r])/th3sp))
    else:
        splitr.append(trunc(len(groups[r])/th4sp))
#create lists for output
spnumd = [] #number stress period days
spst =[]    #stress period start
spfn = []   #stress period finish
splev =[]   #downstream of lock stress period stage
splev1 = [] #upstream of lock stress period stage
allsp = []  #combined stress period list
#Populate the lists
for r in range(0,len(groups)):
```

```
    msplt = list(mygrouper(splitr[r],groups[r]))
    msplt1 =  list(mygrouper(splitr[r],groups1[r]))
    if len(msplt[-1]) <= len(msplt[0])/2:
        msplt[-2].extend(msplt[-1])
        msplt1[-2].extend(msplt1[-1])
        msplt.remove(msplt[-1])
        msplt1.remove(msplt1[-1])
    for k in range(0,len(msplt)):
        spnumd.append(len(msplt[k]))
        spst.append(msplt[k][0][1])
        spfn.append(msplt[k][-1][1])
        ct = 0
        ct1 = 0
        tot = 0
        tot1 = 0
        for i in range(0,len(msplt[k])):
            ct = msplt[k][i][2]
            ct1 = msplt1[k][i][2]
            tot += ct
            tot1 += ct1
        splev.append(round(tot/len(msplt[k]),2))
        splev1.append(round(tot1/len(msplt1[k]),2))
#create lists for plotting
for r in range(0,len(splev)):
    sp = []
    sp.append(int(r))
    sp.append(spnumd[r])
    sp.append(splev[r])
    sp.append(spst[r])
    sp.append(spfn[r])
    sp.append(splev1[r])
    allsp.append(sp)
#Write file for downstream adaptive stress periods
f = open('dslockriv_str_per.txt','w')
f.write('Period_number, length (days), river_level (m), start_date, end_date\n')
for r in range(0, len(allsp)):
        f.write('{}, {}, {}, {}, {}\n'.format\
        (allsp[r][0], allsp[r][1], allsp[r][2],\
        allsp[r][3].strftime('%d/%m/%Y'),\
        allsp[r][4].strftime('%d/%m/%Y')))
f.close()
#Write file for upstream adaptive stress periods
f = open('uslockriv_str_per.txt','w')
f.write('Period_number, length (days), river_level (m), start_date, end_date\n')
for r in range(0, len(allsp)):
        f.write('{}, {}, {}, {}, {}\n'.format\
        (allsp[r][0], allsp[r][1], allsp[r][5],\
        allsp[r][3].strftime('%d/%m/%Y'),\
        allsp[r][4].strftime('%d/%m/%Y')))
f.close()
#Plot and save figures to show where stress periods occur for adaptive method
fig, ax1 = matplotlib.pyplot.subplots()
ax1.plot_date(dates,heights, fmt='g-',xdate=True, ydate=False, label = 'river level')
for r in range(0,len(allsp)):
        ax1.plot_date(spst[r], splev[r], fmt='rx',xdate=True, ydate=False)
for r in range(0,len(allsp)):
      ax1.plot_date(spfn[r], splev[r], fmt='kx',xdate=True, ydate=False)
plt.title('%s Stress Periods (red x = start, black x = end)'%len(allsp))
plt.legend()
```

```
F=pylab.gcf()
F.set_size_inches(11.692,8.267)
F.savefig("DS_Stress_periods.png", dpi = (100))
fig1, ax2 = matplotlib.pyplot.subplots()
ax2.plot_date(dates1,heights1, fmt='g-',xdate=True, ydate=False, label = 'river level')
for r in range(0,len(allsp)):
       ax2.plot_date(spst[r], splev1[r], fmt='rx',xdate=True, ydate=False)
for r in range(0,len(allsp)):
      ax2.plot_date(spfn[r], splev1[r], fmt='kx',xdate=True, ydate=False)
plt.title('%s Stress Periods (red x = start, black x = end)'%len(allsp))
plt.legend()
F=pylab.gcf()
F.set_size_inches(11.692,8.267)
F.savefig("US_Stress_periods.png", dpi = (100))
plt.show()
```

Appendix A3: Reservoir package stress period creator (single pass)

This Python script uses a Gaussian convolution on the datasets obtained from the previous interpolating script. The purpose of the convolution is to remove much of the noise in the daily stage elevations. We only apply the smoothing to the downstream dataset as it has the largest range of stage variation. The stress periods are then determined by identifying the peaks and troughs in the smoothed data. It was suggested that the first of each month be included in the stress period setup, for the purposes of reporting results consistently. If this needs to be removed, the comments included within the script will identify where to do so.

It should be noted that this script will affect the maximum heights of the flood peaks. If the detail of the flood peaks need to be preserved then it is best to use the script from Appendix A4.

The input for the script is the completed upstream and downstream dataset output from the interpolation to nearest neighbour script. The outputs of the script are a stress period data set for Lake or Reservoir package in MODFLOW and a plot of the linear function overlaying the original data. The version of Python used when the script was created is 2.7.6.0. A copy and paste of the text below should provide a working script (tested and developed in Spyder).

```
#**********************************************************************#
#Importing libraries
import pylab
from scipy import ndimage
from pylab import *
import datetime
import os
import csv
#**********************************************************************#
#declare a rounding function used to round smoothed data
def myround(a, decimals=2):
    return np.around(a-10**(-(decimals+5)), decimals=decimals)
#**********************************************************************#
#Changing the working directory
os.chdir('C:\\workspace\\laat0003\\river')
#**********************************************************************#
#These input files are the output of the data interpolation
#Opening and reading csv data file comprising 2 columns
#Column 1 has dates in dd/mm/yyyy format
#Column 2 has stage elevations in meters with 2 decimals
#We are dealing with 2 sets of data 1 for upstream lock
#and 1 for downstteam lock
#NOTE this script assumes column titles or a 1 line header
infile1 = open('River.csv','r')
infile2 = open('River2.csv','r')
#read csv into table0
table0 = [row for row in csv.reader(infile1)]
table1 = [row for row in csv.reader(infile2)]
#creates empty list of dates and heights
dates , heights = [], []
dates1 , heights1 = [], []
#populate date list from table0
[dates.append(table0[r][0]) for r in range(1,len(table0))]
[dates1.append(table0[r][0]) for r in range(1,len(table1))]
#populate heights from table0
[heights.append(table0[r][1]) for r in range(1,len(table0))]
[heights1.append(table1[r][1]) for r in range(1,len(table1))]
#change heights from stings to floats
heights = [float(heights[x]) for x in range(0,len(heights))]
heights1 = [float(heights1[x]) for x in range(0,len(heights1))]
#make the heights list into an array
h=asarray(heights)
```

```
h1=asarray(heights1)
#convert the date strings into date objects
dates= [datetime.datetime.strptime(dates[r],'%d/%m/%Y').date() for r in range(0,len(dates))]
dates1= [datetime.datetime.strptime(dates1[r],'%d/%m/%Y').date() for r in range(0,len(dates1))]
#set the size of the Gaussian window in number of days
days = 25
#make a copy of the heights array as we need an unmodified version later
h_init = np.copy(h)
#smooth the data set with the user defined window size above
gf4 = ndimage.gaussian_filter1d(h_init,days)
#Create a list of the first of the month for each month in the dataset
#Remove the next 2 lines if you don't require the first of each month in the stress periods
temp = []
[temp.append(k) for k in dates if k.day == 1]
#Create a list of the cell index in dates for the first of each month
#Remove the next 2 lines if you don't require the first of each month in the stress periods
first = []
[first.append(dates.index(k)) for k in temp]
#Locate the peaks in the dataset as array indices
pks1 = find(diff(sign(diff(gf4)))<0)+1          #locates the peaks as array indices
#Locate the troughs in the dataset as array indices
trs1 = find(diff(sign(diff(gf4)))>0)+1          #locates the troughs as array indices
#Convert both peaks and trough index arrays to lists
lspks1 = pks1.tolist()
lstrs1 = trs1.tolist()
#Combine all the index values
#Remove the "+first" below if you don't require the first of each month in the stress periods
stps = lspks1+lstrs1+first
#Sort the index values this ensures the dates are sequential
stps = sort(list(set(stps)))
#Create lists for each item to be printed in the output file
#The number of days for the stress period
numdays=[]
#The start and end dates for the stress period
stress_dates=[]
#The start and end values for the stress period from downstream dataset
stress_values =[]
#The start and end values for the stress period from upstream dataset
stress_values1 = []
#The stress period dates in datetime format for plotting
sdt = []
#Start the number of days list with 0
numdays.append(0)

#Loop through all the index numbers and grab
for k in stps:
#The dates for the output file
    stress_dates.append(dates[k].strftime('%d/%m/%Y'))
#The smoothed downstream lock values for the output file
    stress_values.append(gf4[k])
#The unsmoothed upstream lock values for the output file
    stress_values1.append(heights1[k])
#The dates for plotting
    sdt.append(dates[k])
#Loop through the plot dates to populate the number of days list
for r in range(1,len(sdt)):
    numdays.append((sdt[r]-sdt[r-1]).days)
#Open a file to write the downstream stress period data to
f = open('DSres_1smooth_stress.txt','w')
```

```
#Write the data to the file as dates, number of days, heights
for r in range(0, len(stress_values)):
    f.write('{}, {}, {}\n'.format(stress_dates[r], numdays[r], myround(stress_values[r]),2))
#Close the file
f.close()
#Open a file to write the upstream stress period data to
f = open('USres_1smooth_stress.txt','w')
#Write the data to the file as dates, number of days, heights
for r in range(0, len(stress_values)):
    f.write('{}, {}, {}\n'.format(stress_dates[r], numdays[r], myround(stress_values1[r]),2))
#Close the file
f.close()

#Plot the original downstream data overlaid by the smoothed data to observe match
fig, ax1 = matplotlib.pyplot.subplots()
ax1.plot_date(dates,h, fmt='g-',xdate=True, ydate=False, label = 'original')
ax1.plot_date(sdt,stress_values, fmt='b-',xdate=True, ydate=False, label = 'reservoir')
plt.title('No data split %i day Gaussian Kernel %s stress periods'%(days,len(stress_values)))
ax1.legend()
plt.xlabel('year')
plt.ylabel('stage (m)')
xmin,xmax = xlim()
xlim(xmax = xmax+1)
ymin,ymax = ylim()
ylim(ymax=ymax+0.01)
#Save the plotted figure
F=pylab.gcf()
F.set_size_inches(11,5)
F.savefig("Res1smooth_Stress_periods.png", dpi = (150))

matplotlib.pyplot.show()
```

Appendix A4: Reservoir package stress period creator (multi pass)

This Python script also implements a Gaussian convolution on the filled datasets. Here multiple convolutions are performed in an attempt to minimize the number of stress periods while still capturing the flood peak behaviour in its entirety. The script requires the user to enter the pool level of the lock reach as well as a threshold stage elevation. The threshold setting will affect the smoothing effect of the convolution. A "high pass" convolution is performed on the data below the threshold while a "low pass" is performed on the data above the threshold. The result is significant smoothing of the stage elevation data closer to pool level and minimal change to the flood peaks above the threshold. In contrast to the previous script this one also utilizes local second derivatives to locate inflection points in the smoothed dataset. This provides increased accuracy in matching the original data set with a piecewise linear function.

The input for the script is the completed upstream and downstream dataset output from the interpolation to nearest neighbour script. The outputs are a plot of the data used and created as well as a text file with the stage elevations and dates for the piecewise linear function. The version of Python used when the script was created is 2.7.6.0. A copy and paste of the text below should provide a working script (tested and developed in Spyder).

```
#**********************************************************************#
#Import Libraries
from scipy import ndimage
from pylab import *
import datetime
import os
import csv
#**********************************************************************#
#declare a rounding function used to round smoothed data
def myround(a, decimals=2):
    return np.around(a-10**(-(decimals+5)), decimals=decimals)
#**********************************************************************#
#Changing the working directory
os.chdir('C:\\Workspace\\laat0003\\river')
#**********************************************************************#
#Opening and reading csv data file comprising 2 columns
#Column 1 has dates in dd/mm/yyyy format
#Column 2 has stage elevations in meters with 2 decimals
#NOTE this file has a 1 line header or column title
infile = open('River_levels.csv','r')
#read csv into table0
table0 = [row for row in csv.reader(infile)]
#creates empty list of dates and heights
dates , heights = [], []
#populate date list from table0
[dates.append(table0[r][0]) for r in range(1,len(table0))]
#populate heights from table0
[heights.append(table0[r][1]) for r in range(1,len(table0))]
#change heights from stings to floats
heights = [float(heights[x]) for x in range(0,len(heights))]
#make the heights list into an array
h=asarray(heights)
#convert the date strings into date objects
dates= [datetime.datetime.strptime(dates[r],'%d/%m/%Y').date() for r in range(0,len(dates))]
#**********************************************************************#
#Enter the pool level for the data
pool_lev = 13.2
#Enter the head threshold to be used to split the data
h_tsh = 1.1
#Create a copy of the original heights
h_init = np.copy(h)
#Split the data set by finding all days with stage above threshold
split = find(h_init>(pool_lev+h_tsh))
```

```
#make all stages above threshold equal to threshold
for r in range(0,len(split)):
    h_init[split[r]]=h_tsh+pool_lev
#Smooth the modified data set with a window size of 30 days
#NOTE adjust the 30 in the statement below to change the window size
#The window size must be greater than 2
gf4 = ndimage.gaussian_filter1d(h_init,30)
gf4_split = gf4
#replace the stage elevations above threshold
for r in range(0,len(split)):
    gf4_split[split[r]] = h[split[r]]
gf4a = gf4_split
#Need to smooth these to avoid sharp spikes
gf4 = ndimage.gaussian_filter1d(gf4a,5)
#get local first derivative of the smoothed dataset
dvs = empty([len(dates)-1,])
for r in range(1,len(dates)-1):
        dvs[r] = (gf4[r+1]-gf4[r-1])/(dates[r+1]-dates[r-1]).days
#create a date list for the local first derivatives
dvs_dates = dates[0:-1]
#smooth the local first derivatives
dvs = ndimage.gaussian_filter1d(dvs,7)
#Get the local second derivatives of the smoothed dataset
dvs2 = empty([len(dvs_dates)-1,])
for r in range(1,len(dvs_dates)-1):
        dvs2[r] = (dvs[r+1]-dvs[r-1])/(dvs_dates[r+1]-dvs_dates[r-1]).days
#create a date list for the local second derivatives
dvs2_dates = dvs_dates[0:-1]
#Smooth the local second derivatives
dvs2 = ndimage.gaussian_filter1d(dvs2,7)
#Locate the peaks and troughs in the smoothed data
pks = find(diff(sign(diff(gf4)))<0)+1
trs = find(diff(sign(diff(gf4)))>0)+1
#locate the peaks and troughs in the second derivatives
#These coincide with inflection points in the smoothed dataset
pks1 = find(diff(sign(diff(dvs2)))<0)+1
trs1 = find(diff(sign(diff(dvs2)))>0)+1
#Create lists for each item to be printed in the output file
#The number of days for the stress period
numdays=[]
#The start and end dates for the stress period
stress_dates=[]
#The start and end values for the stress period from downstream dataset
stress_values =[]
#The stress period dates in datetime format for plotting
sdt = []
#Start the lists with the first value of the dataset at time = 0
numdays.append(0)
sdt.append(dates[0])
stress_dates.append(dates[0].strftime('%d/%m/%Y'))
stress_values.append(h[0])
#The following if statement builds the lists in the correct order for file output
#and plotting
#This if statement only considers the peaaks and troughs in the smoothed dataset
if pks[0]>trs[0]:
    d0 = dates[0]
    d1 = dates[trs[0]]
    delta = d1-d0
    numdays.append(str(delta.days))
```

```
    sdt.append(dates[trs[0]])
    stress_dates.append(dates[trs[0]].strftime('%d/%m/%Y'))
    stress_values.append(gf4[trs[0]])
    for r in range(1,len(trs)):
        numdays.append(str((dates[pks[r-1]]-dates[trs[r-1]]).days))
        sdt.append(dates[pks[r-1]])
        stress_dates.append(dates[pks[r-1]].strftime('%d/%m/%Y'))
        stress_values.append(gf4[pks[r-1]])
        numdays.append(str((dates[trs[r]]-dates[pks[r-1]]).days))
        sdt.append(dates[trs[r]])
        stress_dates.append(dates[trs[r]].strftime('%d/%m/%Y'))
        stress_values.append(gf4[trs[r]])
elif pks[0]<trs[0]:
    d0 = dates[0]
    d1 = dates[pks[0]]
    delta = d1-d0
    numdays.append(str(delta.days))
    sdt.append(dates[pks[0]])
    stress_dates.append(dates[pks[0]].strftime('%d/%m/%Y'))
    stress_values.append(gf4[trs[0]])
    for r in range(1,len(pks)):
        numdays.append(str((dates[trs[r-1]]-dates[pks[r-1]]).days))
        sdt.append(dates[trs[r-1]])
        stress_dates.append(dates[trs[r-1]].strftime('%d/%m/%Y'))
        stress_values.append(gf4[trs[r-1]])
        numdays.append(str((dates[pks[r]]-dates[trs[r-1]]).days))
        sdt.append(dates[pks[r]])
        stress_dates.append(dates[pks[r]].strftime('%d/%m/%Y'))
        stress_values.append(gf4[pks[r]])
#We repeat the list building process but this time we consider the inflection points
sdt2 =[]
numdays2=[]
stress_dates2=[]
stress_values2=[]
diff2= []
numdays2.append(0)
sdt2.append(dates[0])
stress_dates2.append(dates[0].strftime('%d/%m/%Y'))
stress_values2.append(h[0])
#This only considers the inflection points obtained via the local second
#derivatives
if pks1[0]>trs1[0]:
    d0 = dates[0]
    d1 = dates[trs1[0]]
    delta = d1-d0
    numdays2.append(str(delta.days))
    sdt2.append(dates[trs1[0]])
    stress_dates2.append(dates[trs1[0]].strftime('%d/%m/%Y'))
    stress_values2.append(gf4[trs1[0]])
    for r in range(1,len(trs1)):
        numdays2.append(str((dates[pks1[r-1]]-dates[trs1[r-1]]).days))
        sdt2.append(dates[pks1[r-1]])
        stress_dates2.append(dates[pks1[r-1]].strftime('%d/%m/%Y'))
        stress_values2.append(gf4[pks1[r-1]])
        numdays2.append(str((dates[trs1[r]]-dates[pks1[r-1]]).days))
        sdt2.append(dates[trs1[r]])
        stress_dates2.append(dates[trs1[r]].strftime('%d/%m/%Y'))
        stress_values2.append(gf4[trs1[r]])
elif pks1[0]<trs1[0]:
```

```
    d0 = dates[0]
    d1 = dates[pks1[0]]
    delta = d1-d0
    numdays2.append(str(delta.days))
    sdt2.append(dates[pks1[0]])
    stress_dates2.append(dates[pks1[0]].strftime('%d/%m/%Y'))
    stress_values2.append(gf4[trs1[0]])
    for r in range(1,len(pks1)):
        numdays2.append(str((dates[trs1[r-1]]-dates[pks1[r-1]]).days))
        sdt2.append(dates[trs1[r-1]])
        stress_dates2.append(dates[trs1[r-1]].strftime('%d/%m/%Y'))
        stress_values2.append(gf4[trs1[r-1]])
        numdays2.append(str((dates[pks1[r]]-dates[trs1[r-1]]).days))
        sdt2.append(dates[pks1[r]])
        stress_dates2.append(dates[pks1[r]].strftime('%d/%m/%Y'))
        stress_values2.append(gf4[pks1[r]])
#convert the lists into arrays
sdt = np.asarray(sdt)
sdt2 = np.asarray(sdt2)
stress_values = np.asarray(stress_values)
stress_values2 = np.asarray(stress_values2)
#write the output file
f = open('model_river_stress.txt','w')
for r in range(0, len(stress_values)):
    f.write('{}, {}, {}\n'.format(stress_dates[r], numdays[r], myround(stress_values[r]),2))
f.close()
#plot the results
fig, ax1 = matplotlib.pyplot.subplots()
ax1.plot_date(dates,h, fmt='g-',xdate=True, ydate=False)
ax1.plot_date(sdt,stress_values, fmt='b-',xdate=True, ydate=False, label = 'peak and trough only')
ax1.plot_date(sdt2,stress_values2, fmt='r-',xdate=True, ydate=False, label = 'with inflection point')
for r in range(0,len(pks)):
    ax1.plot_date(dates[pks[r]], gf4[pks[r]], fmt='ro',xdate=True, ydate=False)
for r in range(0,len(trs)):
    ax1.plot_date(dates[trs[r]], gf4[trs[r]], fmt='ko',xdate=True, ydate=False)
plt.xlabel('year')
plt.ylabel('stage (m)')
ax2 = ax1.twinx()
ax2.plot(dvs2_dates,dvs2,'k-', label = '2nd derivative')
for r in range(0,len(pks1)):
    ax1.plot_date(dates[pks1[r]], gf4[pks1[r]], fmt='cs',xdate=True, ydate=False)
for r in range(0,len(trs1)):
    ax1.plot_date(dates[trs1[r]], gf4[trs1[r]], fmt='gs',xdate=True, ydate=False)
ax1.legend()
plt.ylabel('2nd derivative')
ax2.legend(loc=9)
matplotlib.pyplot.show()
```

B. Fortran programs to compare effects of groundwater velocity on salt loads

Peter Cook

This appendix contains the Fortran programs used to examine the effect of groundwater velocity on salt flux to the river. Three different approaches were compared:

1. Constant groundwater velocity
2. Groundwater velocity that varies in time, and is driven by wetland leakage (referred to as the Pipe Model)
3. Same as (2), but with evapotranspiration occurring along the groundwater flow path.

All models read in the same input data, which is daily values of time, wetland level, wetland read data, flow to the wetland (not used), wetland concentration, wetland leakage, and leakage concentration. This table of data is calculated by a separate surface water model, which calculates wetland flows based on river flow and concentration data.

Appendix B1: Surface water model

```
!
! This program calculates the wetland surface water balance
!
            program wetlandsw2
            implicit none
            character*16 infile1, infile2, outfile1, outfile2, outfile3
            real evap
            real rivbott, chabott, chawidt, chaleng, charoug
            real wetbott, wetarea
            real chawatsl, chawatdep
            real maxleakconc
            real wetevap(30000000), hc, Kc, loss
            real inputtime(30000), inputrivlev(30000), inputrivconc(30000)
            real wetlev(30000000), wetvol(30000000)
            real time(30000000), rivlev(30000000)
            real wetflow(30000000), wetleak(30000000), cumleak(30000000), leakconc(30000000)
            real rivconc(30000000), wetmas(30000000), wetconc(30000000)
            real dt, simtime
            real dayrivlev(30000), daywetlev(30000),daywetflow(30000), daywetconc(30000),
                                                                              dayleakconc(30000)
            real daywetleak(30000)
            integer novalues
            integer n, i, j
!
!           units are metres and days
!
            infile1 = 'riverdata5.dat'
            open (unit=1, file=infile1, status='old')
              read (1, *) novalues
              do i=1, novalues
                 read(1, *) inputtime(i), inputrivlev(i), inputrivconc(i)
              enddo
              close (unit=1)
!
            print *, ' Finished reading river data ...'
!
              dt = 0.001
!
!           NOTE: PROGRAM IS NOT WRITTEN FOR VALUES OF DT OTHER THAN 0.001
!
              simtime = 25949.0
!
!           First interpolate river levels and concentrations
!
              do i=1, int(simtime/dt)
                time(i) = i*dt
                j= 1 + (i-mod(i,1000))/1000
                rivlev(i)=inputrivlev(j)+(inputrivlev(j+1)-inputrivlev(j)) * &
                                (time(i)-inputtime(j))/(inputtime(j+1)-inputtime(j))
                rivconc(i)=inputrivconc(j)+(inputrivconc(j+1)-inputrivconc(j)) * &
                                (time(i)-inputtime(j))/(inputtime(j+1)-inputtime(j))
              enddo
!
            print *, ' Finished interpolating river levels and concentrations ... '
!
```

```
      rivbott = 12.0
      chabott = 16.0
      chawidt = 10.0
      chaleng = 250.0
      charoug = 0.25
      wetbott = 15.0
      wetarea = 100000.0
      wetlev(1) = 15.0
      wetvol(1) = (wetlev(1)-wetbott)*wetarea
      wetflow(1) = 0.0
      wetleak(1) = 0.0
      cumleak(1) = 0.0
      rivconc(1) = 500.0
      wetmas(1) = 0.0
      wetconc(1) = 0.0
      evap = 0.005
      hc = 1.0
      Kc = 0.02
      novalues = 5
      maxleakconc = 200000.0
!
!     First set up do loop on time
!
      print *, ' Commencing flow calculations ... '
!
      do i=2, int(simtime/dt)
!
      cumleak(i) = 0.0
!
!     First calculate the flow between the wetland and the river
!
        if((rivlev(i).lt.chabott).and.(wetlev(i-1).lt.chabott)) then
          wetflow(i)=0.0
        else
          chawatsl = (max(rivlev(i),chabott)-max(wetlev(i-1),chabott))/chaleng
          chawatdep = (max(rivlev(i),chabott)+max(wetlev(i-1),chabott))/2.0-chabott
          if (rivlev(i).gt.wetlev(i-1)) then
                wetflow(i) = chawatdep**0.67 * chawatsl**0.5 * chawatdep*chawidt / charoug
                                                                        *86400.0 *dt
              else
                wetflow(i) = -(chawatdep**0.67) * (-chawatsl)**0.5 * chawatdep*chawidt /
                                                                        charoug *86400.0*dt
                wetflow(i) = max(wetflow(i), -wetvol(i-1))
              endif
        endif
!
!     Next adjust the wetland water level for flow as required
!
        wetlev(i) = wetlev(i-1)+wetflow(i)/wetarea
        wetvol(i) = (wetlev(i)-wetbott)*wetarea
!
!     Now do leakage and evaporation
!
        if(wetvol(i).gt.0.0) then
!
          wetleak(i) = ((wetlev(i)-wetbott)/hc+1)*Kc*dt
          wetevap(i) = evap*dt
          loss = wetleak(i) + wetevap(i)
!
```

```
!           The next lines scale back evap and leakage if there is not enough water in the wetland
!
              if(wetevap(i)+wetleak(i).gt.wetlev(i)-wetbott) then
                wetevap(i) = wetevap(i) * (wetlev(i)-wetbott)/loss
                wetleak(i) = wetleak(i)*(wetlev(i)-wetbott)/loss
              endif
            else
              wetevap(i) = 0.0
              wetleak(i) = 0.0
            endif
           wetlev(i) = wetlev(i) - wetevap(i) - wetleak(i)
           wetlev(i) = max(wetlev(i),wetbott)
            wetvol(i) = (wetlev(i)-wetbott)*wetarea
            cumleak(i) = cumleak(i-1) + wetleak(i)
!
           enddo
!
           print *, ' Completed flow calculations ...'
!
!          Now do concentrations
!
           do i=2, int(simtime/dt)
!
            if (wetflow(i).gt.0.0) then
              leakconc(i) = (wetmas(i-1)+wetflow(i)*rivconc(i))/(wetflow(i)+wetvol(i-1))
              leakconc(i) = min(maxleakconc, leakconc(i))
              wetmas(i) = wetmas(i-1)+(wetflow(i)*rivconc(i) - wetleak(i) *leakconc(i))
            else
              leakconc(i) = wetconc(i-1)
              leakconc(i) = min(maxleakconc, leakconc(i))
              wetmas(i) = wetmas(i-1)+(wetflow(i)*wetconc(i-1)-wetleak(i)*leakconc(i))
            endif
!
            if (wetvol(i).gt.0.0) then
              wetconc(i) = wetmas(i)/wetvol(i)
            endif

           enddo
!
!          Now sum back to daily data
!
           print *, ' Now summing back to daily data ...'
!
            dayrivlev(1) = 0.0
            daywetlev(1) = 0.0
            daywetflow(1) = 0.0
            daywetconc(1) = 0.0
            daywetleak(1) = 0.0
            dayleakconc(1) = 0.0
!
           do i=2, 1000
            dayrivlev(1) = dayrivlev(1) + rivlev(i)/999
            daywetlev(1) = daywetlev(1) + wetlev(i)/999
            daywetflow(1) = daywetflow(1) + wetflow(i)*1000/999
            daywetconc(1) = daywetconc(1) + wetconc(i)/999
            daywetleak(1) = daywetleak(1) + wetleak(i)*1000/999
            dayleakconc(1) = dayleakconc(1) + leakconc(i)/999
           enddo
!
```

```
            do j=1, int(simtime)
!
              dayrivlev(j) = 0.0
              daywetlev(j) = 0.0
              daywetflow(j) = 0.0
              daywetconc(j) = 0.0
              daywetleak(j) = 0.0
              dayleakconc(j) = 0.0
!
              do i=(j-1)*1000+1, j*1000
                dayrivlev(j) = dayrivlev(j) + rivlev(i)/1000.0
                daywetlev(j) = daywetlev(j) + wetlev(i)/1000.0
                daywetflow(j) = daywetflow(j) + wetflow(i)
                daywetconc(j) = daywetconc(j) + wetconc(i)/1000.0
                daywetleak(j) = daywetleak(j) + wetleak(i)
                dayleakconc(j) = dayleakconc(j) + leakconc(i)/1000.0
              enddo
            enddo
!
!           outfile1 = 'outputflow.dat'
!           open (unit=2, file=outfile1, status='replace')
!           do i=1, int(simtime/dt), 1
!             write (2, 120) time(i), rivlev(i), wetlev(i), wetflow(i), wetevap(i)*wetarea, &
!                wetleak(i)*wetarea, wetvol(i), wetconc(i), wetmas(i)
!           enddo
!           close(unit=2)
!120        format(1x, f9.4, 3x, f7.4, 3x, f7.4, 3x, f7.2, 3x, f7.2, 3x, f7.2, 3x, f12.2, 3x, f9.1, 3x, f15.0)
!
!           outfile2 = 'outputconc.dat'
!           open (unit=3, file=outfile2, status='replace')
!           do i=1, int(simtime/dt), 1
!             write (3, 130) time(i), rivlev(i), wetlev(i), wetleak(i), cumleak(i), leakconc(i)
!           enddo
!           close(unit=3)
!130        format(1x, f9.4, 3x, f7.4, 3x, f7.4, 3x, f9.6, 3x, f9.3, 3x, f12.1)
!
            outfile3 = 'dailyoutput.dat'
            open (unit=4, file=outfile3, status='replace')
            write (4, 135) int(simtime)
            do i=1, int(simtime)
              write (4, 140) i,dayrivlev(i),daywetlev(i),daywetflow(i),daywetconc(i),daywetleak(i), dayleakconc(i)
            enddo
            close(unit=4)
135         format(1x, i7)
140         format(1x, i7, 2x, f6.3, 2x, f6.3, 2x, f12.3, 2x, f12.1, 2x, f7.3, 2x, f12.1)
!
!
            end
```

Appendix B2: Constant groundwater velocity

```
!
!       In this model, all wetland leakage moves to the river at a constant, specified velocity.
!
        program wetlandgw13
        implicit none
        character*20 infile1, outfile1, outfile2
        real (kind=kind(1.0d0)) counter
        real (kind=kind(1.0d0)) wetarea, wetdist, wetwidth, aqthick, poros, veloc
        real (kind=kind(1.0d0)) inputtime(60000)
        real (kind=kind(1.0d0)) time(6000000), rivlev(6000000)
        real (kind=kind(1.0d0)) wetflow(6000000), wetleak(6000000), cumleak(6000000)
        real (kind=kind(1.0d0)) leakconc(6000000)
        real (kind=kind(1.0d0)) traveltime(6000000)
        real (kind=kind(1.0d0)) saltdisch(6000000)
        real (kind=kind(1.0d0)) dt, simtime
       real (kind=kind(1.0d0)) dayrivlev(60000), daywetlev(60000),daywetflow(60000),
                                    daywetconc(60000)
        real (kind=kind(1.0d0)) daywetleak(60000), daysalt(60000), dayleakconc(60000)
        integer dischstep(6000000)
        integer novalues, noloops
        integer n, i, j
!
!       units are metres and days
!
        infile1 = 'dailyoutput.dat'
        open (unit=1, file=infile1, status='old')
          read (1, *) novalues
          do i=1, novalues
            read(1, *) inputtime(i), dayrivlev(i), daywetlev(i), daywetflow(i), daywetconc(i), &
                                    daywetleak(i), dayleakconc(i)
          enddo
          close (unit=1)
!
        print *, ' Finished reading from surface water file ...'
!
          dt = 0.01
!
!       First establish loops on input data and interpolate wetland leakage data
!
        noloops = 2
        simtime = real(novalues * noloops)
!
        do i=novalues+1, int(simtime)
          inputtime(i) = real(i)
          dayrivlev(i) = dayrivlev(mod(i,novalues))
          daywetlev(i) = daywetlev(mod(i,novalues))
          daywetflow(i) = daywetflow(mod(i,novalues))
          daywetconc(i) = daywetconc(mod(i,novalues))
          daywetleak(i) = daywetleak(mod(i,novalues))
          dayleakconc(i) = dayleakconc(mod(i,novalues))
        enddo
!
          do i=1, int(simtime/dt)
            time(i) = real(i)*dt
            j= 1 + int(real((i-mod(i,int(1/dt))))*dt)
            wetleak(i)=daywetleak(j)+(daywetleak(j+1)-daywetleak(j)) * &
```

```
                              (time(i)-inputtime(j))/(inputtime(j+1)-inputtime(j))
          leakconc(i)=dayleakconc(j)+(dayleakconc(j+1)-dayleakconc(j)) * &
                              (time(i)-inputtime(j))/(inputtime(j+1)-inputtime(j))
          enddo
!
          print *, ' Finished interpolating river levels and concentrations ... '
!
          veloc = 0.05
!
          wetdist = 250.0
          wetarea = 100000.0
          poros = 0.4
          wetwidth = 100.0
          aqthick = 100.0
!
          cumleak(1) = 0.0
          dischstep(1)=0
!
!         First set up do loop on time
!
          do i=2, int(simtime/dt)
           cumleak(i) = cumleak(i-1) + wetleak(i)*dt
          enddo
!
!         First calculate the arrival time (and timestep) for each bit of leakage
!
          do i=2, int(simtime/dt)
!
           saltdisch(i) = 0.0
!
           if(mod(i,int(1/dt)).eq.0) then
            print *, 'Day ... ', i*dt
           endif
!
           if(wetleak(i).gt.0.0) then
            traveltime(i) = wetdist / veloc
            dischstep(i) = i + int(traveltime(i)/dt)
           endif
!
          enddo
!
!         Now find all the leakage bits that arrive at each time
!
          do i=2, int(simtime/dt)
           if(wetleak(i).gt.0.0) then
            if(dischstep(i).le.int(simtime/dt)) then
             saltdisch(dischstep(i)) = saltdisch(dischstep(i)) + wetleak(i)*wetarea*leakconc(i)*dt
            endif
           endif
          enddo
!
!
!         Now sum back to daily data
!
          print *, 'Now summing back to daily data ...'
!
          do j=1, int(simtime)-1
!
           daysalt(j) = 0.0
```

```
!
              do i=int((j-1)/dt)+1, int(j/dt)
                daysalt(j) = daysalt(j) + saltdisch(i)
              enddo
            enddo
!
!
            outfile1 = 'outputall-gw13.dat'
            open (unit=2, file=outfile1, status='replace')
            do i=1, int(simtime)-1
              write (2, 140) i,dayrivlev(i),daywetlev(i),daywetflow(i),daywetconc(i), &
                        daywetleak(i), dayleakconc(i), daysalt(i)
            enddo
            close(unit=2)
140         format(1x, i7, 2x, f7.4, 2x, f7.4, 2x, f9.3, 2x, f12.1, 2x, f9.3, 2x, f12.1, 2x, f15.0)
!
!           outfile2 = 'check-gw13.dat'
!           open (unit=3, file=outfile2, status='replace')
!           do i=1, int(simtime/dt)-1
!             write (3, 135) i, wetleak(i), cumleak(i), leakconc(i), dischstep(i), saltdisch(i)
!           enddo
!           close(unit=3)
!135        format(1x, i7, 2x, f9.3, 2x, f9.3, 2x, f15.1, 2x, i7, 2x, f15.0)
!
!
            end
```

Appendix B3: Pipe model

```
!
!Groundwater velocity varies with time, and is driven by wetland leakage.
!
            program wetlandgw10
             implicit none
             character*20 infile1, outfile1, outfile2
             real (kind=kind(1.0d0)) counter
             real (kind=kind(1.0d0)) wetarea, wetdist, wetwidth, aqthick, poros
             real (kind=kind(1.0d0)) inputtime(60000)
             real (kind=kind(1.0d0)) time(6000000), rivlev(6000000)
             real (kind=kind(1.0d0)) wetflow(6000000), wetleak(6000000), cumleak(6000000)
             real (kind=kind(1.0d0)) leakconc(6000000)
             real (kind=kind(1.0d0)) saltdisch(6000000)
             real (kind=kind(1.0d0)) dt, simtime
             real (kind=kind(1.0d0)) dayrivlev(60000), daywetlev(60000),daywetflow(60000), daywetconc(60000)
             real daywetleak(60000), daysalt(60000), dayleakconc(60000)
             integer dischstep(6000000)
             integer novalues, noloops
             integer i, j, k
!
!            units are metres and days
!
             infile1 = 'dailyoutput.dat'
             open (unit=1, file=infile1, status='old')
               read (1, *) novalues
               do i=1, novalues
                 read(1, *) inputtime(i), dayrivlev(i), daywetlev(i), daywetflow(i), daywetconc(i), &
                                          daywetleak(i), dayleakconc(i)
               enddo
               close (unit=1)
!
             print *, ' Finished reading from surface water file ...'
!
               dt = 0.01
!
!
!            First establish loops on input data and interpolate wetland leakage data
!
             noloops = 2
             simtime = real(novalues * noloops)
!
             do i=novalues+1, int(simtime)
               inputtime(i) = real(i)
               dayrivlev(i) = dayrivlev(mod(i,novalues))
               daywetlev(i) = daywetlev(mod(i,novalues))
               daywetflow(i) = daywetflow(mod(i,novalues))
               daywetconc(i) = daywetconc(mod(i,novalues))
               daywetleak(i) = daywetleak(mod(i,novalues))
               dayleakconc(i) = dayleakconc(mod(i,novalues))
             enddo
!
               do i=1, int(simtime/dt)
                 time(i) = real(i)*dt
                 j= 1 + int(real((i-mod(i,int(1/dt))))*dt)
                 wetleak(i)=daywetleak(j)+(daywetleak(j+1)-daywetleak(j)) * &
                              (time(i)-inputtime(j))/(inputtime(j+1)-inputtime(j))
```

```
            leakconc(i)=dayleakconc(j)+(dayleakconc(j+1)-dayleakconc(j)) * &
                              (time(i)-inputtime(j))/(inputtime(j+1)-inputtime(j))
          enddo
!
          print *, ' Finished interpolating river levels and concentrations ... '
!
          wetdist = 250.0
          wetarea = 100000.0
          poros = 0.4
          wetwidth = 100.0
          aqthick = 100.0
!
          cumleak(1) = 0.0
          dischstep(1)=0
!
!         First set up do loop on time
!
          do i=2, int(simtime/dt)
           cumleak(i) = cumleak(i-1) + wetleak(i)*dt
          enddo
!
!
!         First calculate the arrival time (and timestep) for each bit of leakage
!
          do i=2, int(simtime/dt)
!
           saltdisch(i) = 0.0
!
           if(mod(i,int(1/dt)).eq.0) then
            print *, 'Day ... ', i*dt
           endif
!
!          The next few lines stop a search for dischstep when it is beyong the simulation time
!
           if(cumleak(i)*wetwidth+poros*wetdist*aqthick.gt.cumleak(int(simtime/dt))*wetwidth)
                                                                                    then
            do k=i, int(simtime/dt)
             dischstep(i)=0.0
            enddo
            exit
           endif
!
           do j=dischstep(i-1), int(simtime/dt)
            if(cumleak(j)*wetwidth.ge.cumleak(i)*wetwidth+poros*wetdist*aqthick) then
             dischstep(i)=j
             exit
            endif
           enddo
          enddo
!
!         Now find all the leakage bits that arrive at each time
!
          do i=2, int(simtime/dt)
           if(dischstep(i).le.int(simtime/dt)) then
            saltdisch(dischstep(i)) = saltdisch(dischstep(i)) + wetleak(i)*wetarea*leakconc(i)*dt
           endif
          enddo
!
!
```

```
!           Now sum back to daily data
!
            print *, 'Now summing back to daily data ...'
!
            do j=1, int(simtime)-1
!
              daysalt(j) = 0.0
!
              do i=int((j-1)/dt)+1, int(j/dt)
                daysalt(j) = daysalt(j) + saltdisch(i)
              enddo
            enddo
!
!
!
            outfile1 = 'outputall-gw10.dat'
            open (unit=2, file=outfile1, status='replace')
            do i=1, int(simtime)-1
              write (2, 140) i,dayrivlev(i),daywetlev(i),daywetflow(i),daywetconc(i), &
                        daywetleak(i), dayleakconc(i), daysalt(i)
            enddo
            close(unit=2)
140         format(1x, i7, 2x, f7.4, 2x, f7.4, 2x, f9.3, 2x, f12.1, 2x, f9.3, 2x, f12.1, 2x, f15.0)

            outfile2 = 'check-gw10.dat'
            open (unit=3, file=outfile2, status='replace')
            do i=1, int(simtime/dt)-1
              write (3, 135) i, wetleak(i), cumleak(i), leakconc(i), dischstep(i), saltdisch(i)
            enddo
            close(unit=3)
135         format(1x, i7, 2x, f9.3, 2x, f9.3, 2x, f15.1, 2x, i7, 2x, f15.0)
!
!
            end
```

Appendix B4: Pipe model with vegetation

```
!
!       This version includes groundwater loss by ET along the flowpath      to the river
!
        program wetlandgw12
        implicit none
        character*20 infile1, outfile1, outfile2
        real (kind=kind(1.0d0)) counter
        real (kind=kind(1.0d0)) wetarea, wetdist, wetwidth, aqthick, poros
        real (kind=kind(1.0d0)) veget, traveltime, wetbott
        real (kind=kind(1.0d0)) inputtime(60000)
        real (kind=kind(1.0d0)) time(6000000), rivlev(6000000), wetlev(6000000)
        real (kind=kind(1.0d0)) wetflow(6000000), wetleak(6000000), cumleak(6000000)
        real (kind=kind(1.0d0)) leakconc(6000000)
        real (kind=kind(1.0d0)) saltdisch(6000000), enrich(6000000)
        real (kind=kind(1.0d0)) dt, simtime
        real (kind=kind(1.0d0)) dayrivlev(60000), daywetlev(60000),daywetflow(60000),
                                                                    daywetconc(60000)
        real (kind=kind(1.0d0)) daywetleak(60000), daysalt(60000), dayleakconc(60000),
                                                                    dayenrich(60000)
        integer dischstep(6000000)
        integer novalues, noloops
        integer i, j, k
!
!       units are metres and days
!
        infile1 = 'dailyoutput.dat'
        open (unit=1, file=infile1, status='old')
          read (1, *) novalues
          do i=1, novalues
            read(1, *) inputtime(i), dayrivlev(i), daywetlev(i), daywetflow(i), daywetconc(i), &
                                    daywetleak(i), dayleakconc(i)
          enddo
          close (unit=1)
!
        print *, ' Finished reading from surface water file ...'
!
          dt = 0.01
!
!
!       First establish loops on input data and interpolate wetland leakage data
!
        noloops = 2
        simtime = real(novalues * noloops)
!
        do i=novalues+1, int(simtime)
          inputtime(i) = real(i)
          dayrivlev(i) = dayrivlev(mod(i,novalues))
          daywetlev(i) = daywetlev(mod(i,novalues))
          daywetflow(i) = daywetflow(mod(i,novalues))
          daywetconc(i) = daywetconc(mod(i,novalues))
          daywetleak(i) = daywetleak(mod(i,novalues))
          dayleakconc(i) = dayleakconc(mod(i,novalues))
        enddo
!
!       Note: The j counter below just works out which day we are on
```

```
!
              do i=1, int(simtime/dt)
                time(i) = real(i)*dt
                j= 1 + int(real((i-mod(i,int(1.0/dt))))*dt)
                wetleak(i)=daywetleak(j)+(daywetleak(j+1)-daywetleak(j)) * &
                                (time(i)-inputtime(j))/(inputtime(j+1)-inputtime(j))
                leakconc(i)=dayleakconc(j)+(dayleakconc(j+1)-dayleakconc(j)) * &
                                (time(i)-inputtime(j))/(inputtime(j+1)-inputtime(j))
                wetlev(i)=daywetlev(j)+(daywetlev(j+1)-daywetlev(j)) * &
                                (time(i)-inputtime(j))/(inputtime(j+1)-inputtime(j))
              enddo
!
            print *, ' Finished interpolating river levels and concentrations ... '
!
            wetbott = 13.0
            wetdist = 250.0
            wetarea = 100000.0
            poros = 0.4
            wetwidth = 100.0
            aqthick = 100.0
            veget = 0.001
!
            cumleak(1) = 0.0
            enrich(1) = 0.0
            dischstep(1) = 0
!
!           First set up do loop on time
!
            do i=2, int(simtime/dt)
              cumleak(i) = cumleak(i-1) + wetleak(i)*wetwidth*dt - veget*(wetdist-wetwidth/2.0)*dt
!             cumleak(i) = cumleak(i-1) + wetleak(i)*wetwidth*dt - 0.002
                if(wetlev(i).eq.wetbott) then
                  cumleak(i) = cumleak(i) - veget*wetwidth*dt
                endif
            enddo
!
!
!           First calculate the arrival time (and timestep) for each bit of leakage
!
            do i=2, int(simtime/dt)
!
              saltdisch(i) = 0.0
              dischstep(i) = 0
!
              if(mod(i,int(1/dt)).eq.0) then
                print *, 'Day ... ', i*dt
              endif
!
!             The next few lines stop a search for dischstep when it is beyong the simulation time
!
              if(cumleak(i)+poros*wetdist*aqthick.gt.cumleak(int(simtime/dt))) then
                do k=i, int(simtime/dt)
                  dischstep(i)=0
                enddo
                exit
              endif
!
              do j=dischstep(i-1), int(simtime/dt)
                if(cumleak(j).ge.cumleak(i)+poros*wetdist*aqthick) then
```

```
              dischstep(i)=j
              exit
            endif
          enddo
        enddo
!
!       Now find all the leakage bits that arrive at each time
!
        do i=2, int(simtime/dt)
          enrich(i) = 0.0
          if((dischstep(i).le.int(simtime/dt)).and.(wetleak(i).gt.0.0) &
               .and.(dischstep(i).gt.0)) then
            traveltime = (dischstep(i)-i)*dt
            enrich(i) = aqthick*poros / (aqthick*poros-veget*traveltime)
            saltdisch(dischstep(i)) = saltdisch(dischstep(i)) + wetleak(i) * wetarea * &
                    leakconc(i) * dt
          endif
        enddo
!
!
!       Now sum back to daily data
!
        print *, 'Now summing back to daily data ...'
!
        do j=1, int(simtime)-1
!
          daysalt(j) = 0.0
          dayenrich(j) = 0.0
!
          do i=int((j-1)/dt)+1, int(j/dt)
            daysalt(j) = daysalt(j) + saltdisch(i)
            dayenrich(j) = dayenrich(j) + enrich(i)*dt
          enddo
        enddo
!
!
!
        outfile1 = 'outputall-gw12.dat'
        open (unit=2, file=outfile1, status='replace')
        do i=1, int(simtime)-1
          write (2, 140) i,dayrivlev(i),daywetlev(i),daywetflow(i),daywetconc(i), &
                  daywetleak(i), dayleakconc(i), dayenrich(i), daysalt(i)
        enddo
        close(unit=2)
140     format(1x, i7, 2x, f7.4, 2x, f7.4, 2x, f9.3, 2x, f12.1, 2x, f9.3, 2x, f12.1, 2x, f7.3, 2x, f15.0)
!
        outfile2 = 'check-gw12.dat'
        open (unit=3, file=outfile2, status='replace')
        do i=1, int(simtime/dt)-1
          write (3, 135) i, wetleak(i), cumleak(i), leakconc(i), enrich(i), dischstep(i), saltdisch(i)
        enddo
        close(unit=3)
135     format(1x, i7, 2x, f9.3, 2x, f12.3, 2x, f15.1, 2x, f7.3, 2x, i7, 2x, f15.0)
!
!
        end
```

C. MODFLOW output

Virginia Riches, Tariq Laattoe, Le Dang & Carl Purczel

This Appendix presents the groundwater model results for Chapter 4 of Woods (2015b). The model has a river running horizontally through it, effectively splitting it in two. The north refers to the area with higher highland/floodplain boundary elevation while the south is the region with the wetland depression.

Appendix C1 compares modelled hydrographs for simulations conducted using a variety of MODFLOW code variants: MODFLOW2000, MODFLOW2005, MODFLOW SURFACT, and MODFLOW-USG (see Section 4.2.82 of Woods, 2015b).

The remaining Appendices provide groundwater model outputs for simulations which are combinations of the following: three scenarios, three hydrodynamic cases, and two weirpool conditions. The scenarios differ in which floodplain process varies over time: in Scenario 1, the river stage varies over time; in Scenario 2, the evapotranspiration varies over time; and in Scenario 3, the vertical recharge changes to simulate the impact of inundation during floods. The three hydrodynamic cases refer to losing stream conditions (Case A), throughflow conditions (Case B), and gaining stream conditions (Case C) for the River Murray at low flow. Furthermore, the simulation may include a change in weirpool level at a lock in the centre of the domain (i.e. "Locked" conditions) or may have no lock ("Not locked). Details of the simulations and their assumptions are given in Woods (2015b)

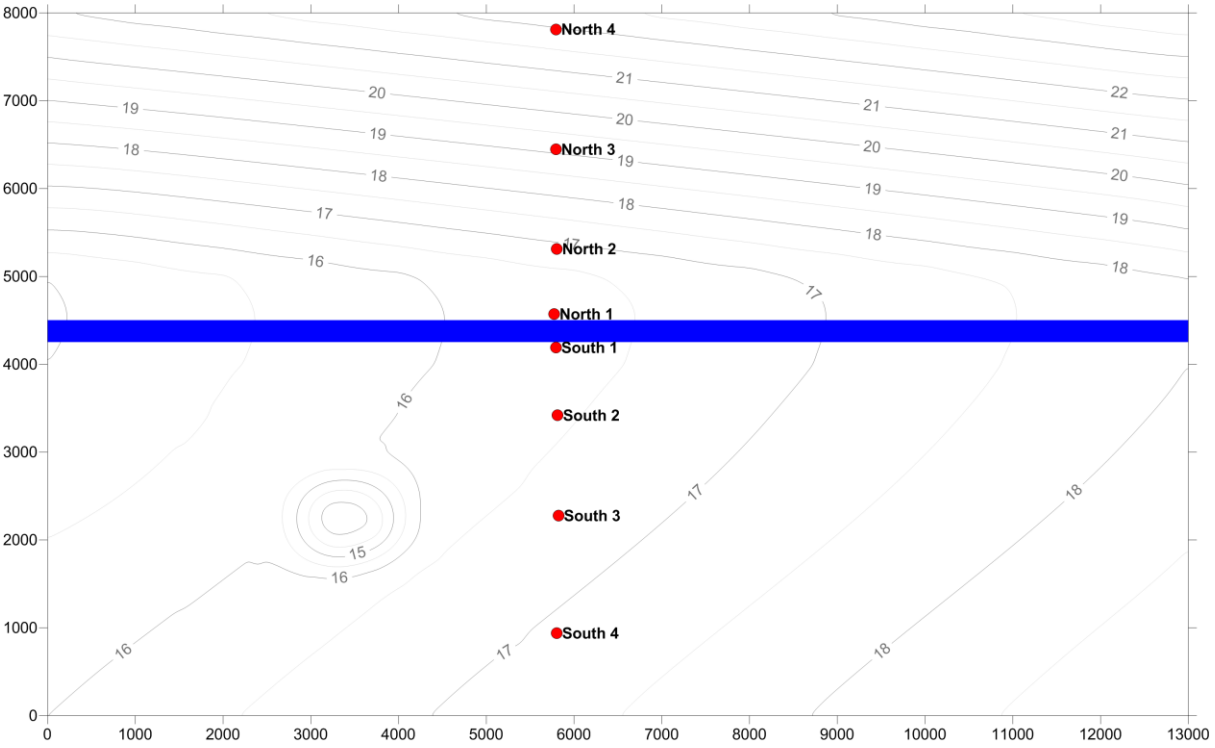
Results are presented as hydrographs, mass balance figures, cumulative salt to river plots, and net river condition figures.

The figure on the next page shows the location of simulated observation wells in simulations (a) without a lock or (b) with a lock and hence change of weirpool level. The contours show the elevation of the ground surface.

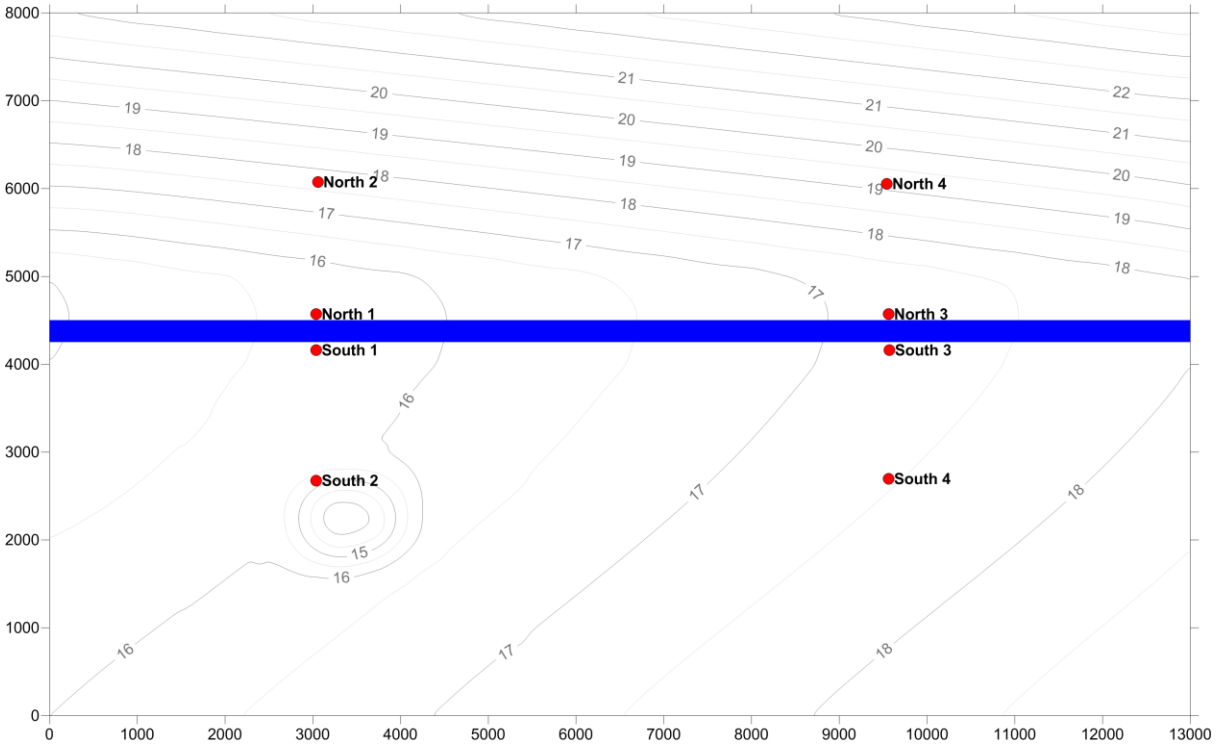
Mass balance figures provide a snapshot for a specific time during a model simulation, depicting the source and sink fluxes of the model. Care needs to be taken when examing the graph, as a flux in refers to a source sending flux into groundwater. For example, "river in" should be interpreted as "from the river into the groundwater" and not the opposite. The different coloured bars represent the stress period discretisation schemes.

The plots of cumualtive salt load to river show modelled salt entering the river, derived from solute transport simulations.

The net river condition graphs present the river flux condition for the entire river throughout the model run, i.e. whether the river was under gaining, throughflow, or losing conditions.


North 4
North 3
North 2
North 1
South 1
South 2
South 3
South 4

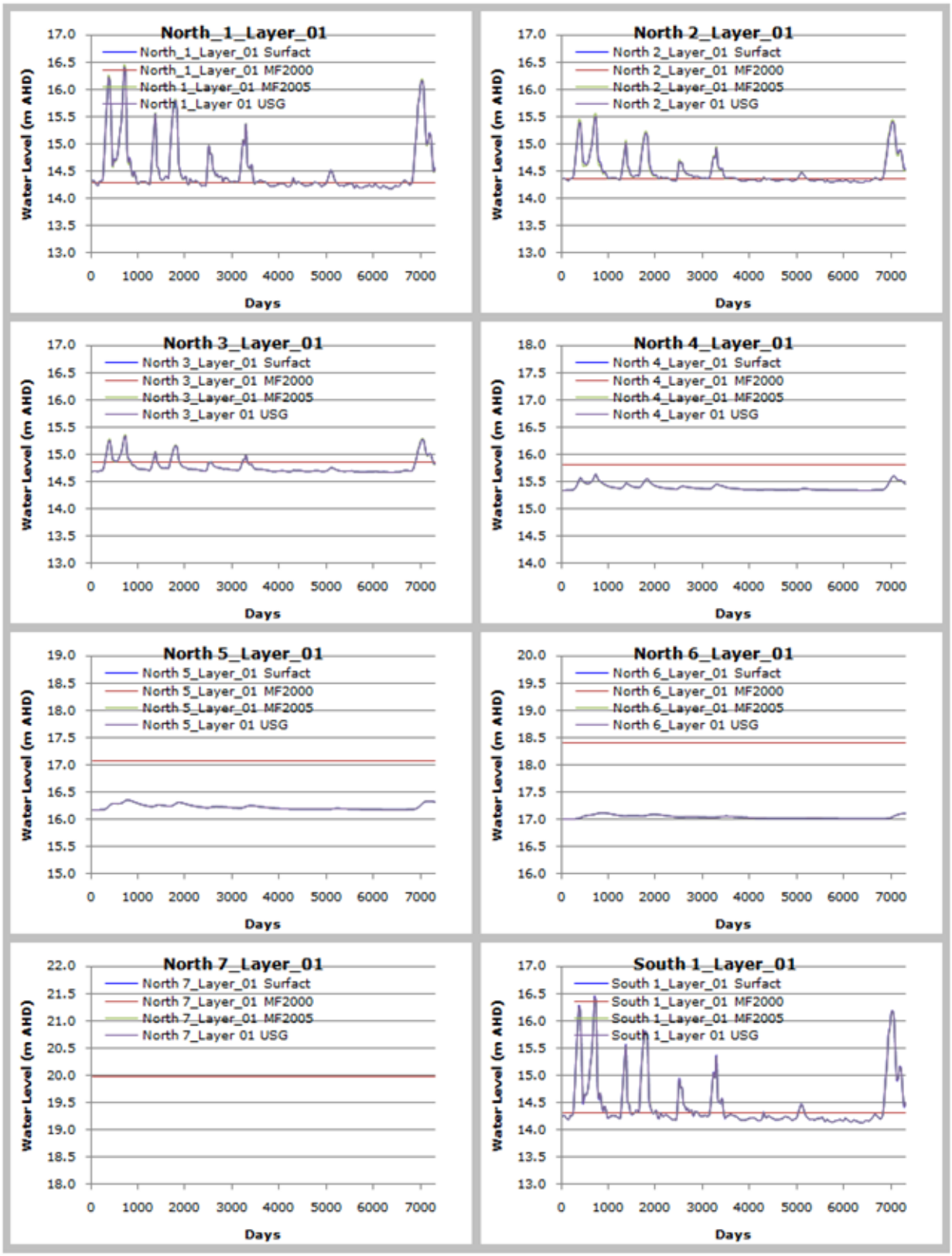
(a)


North 2
North 4
North 1
North 3
South 1
South 3
South 2
South 4

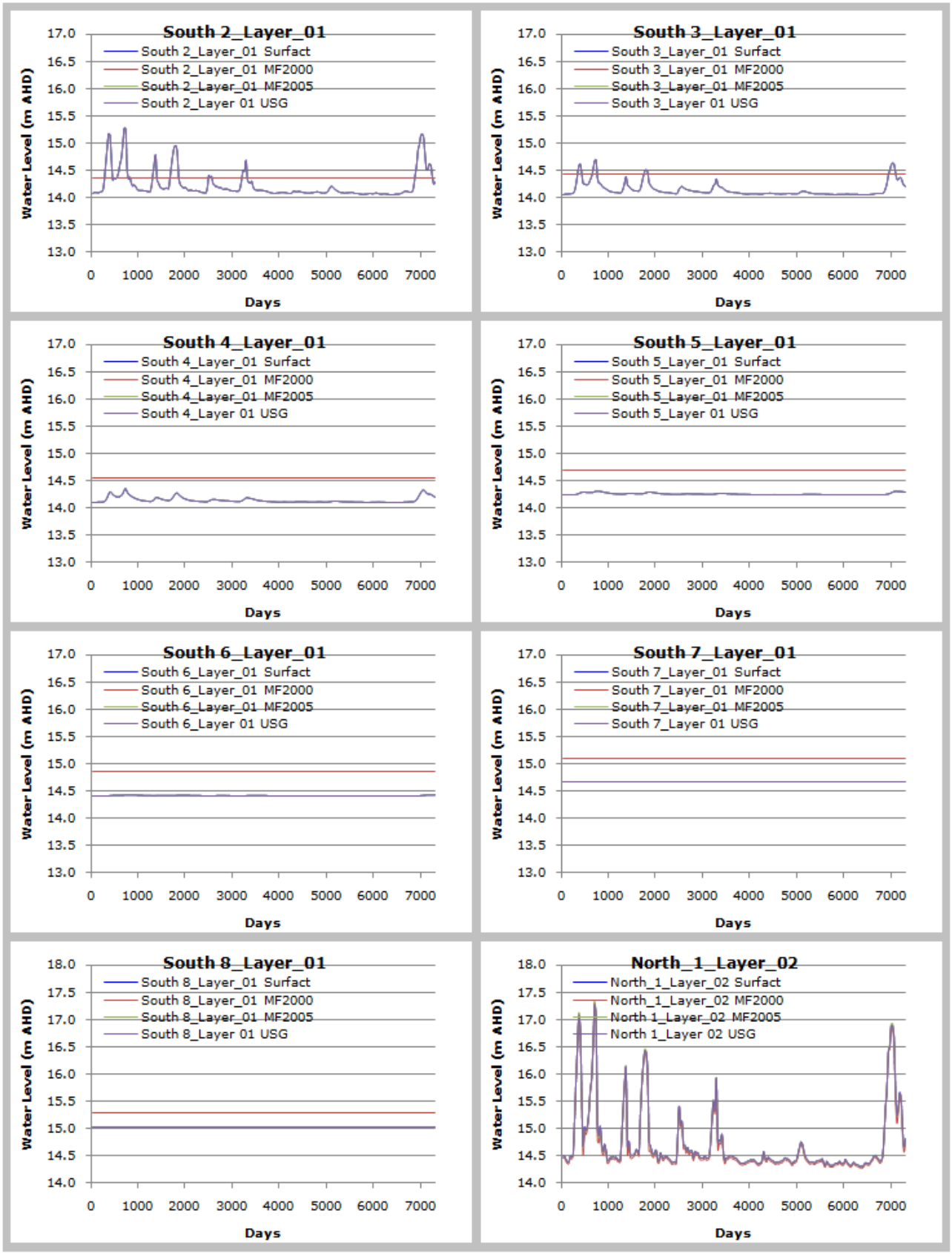
(b)

Appendix C1: Hydrographs compared between different groundwater modelling codes

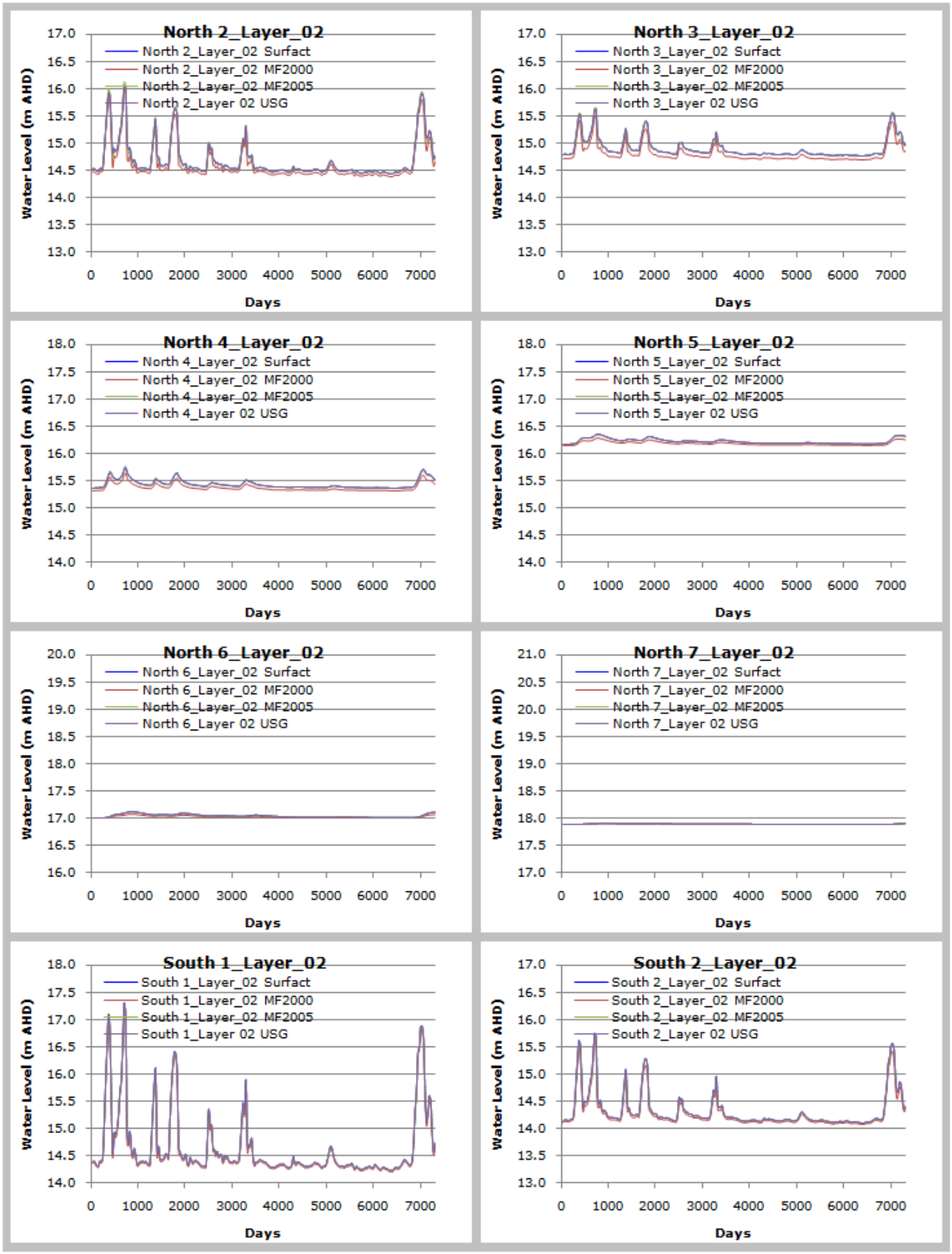
The following figures present the differences in rewetting ability of 4 MODFLOW variants, namely MODFLOW SURFACT, MODFLOW 2000, MODFLOW 2005 (with NWT) and MODFLOW USG.



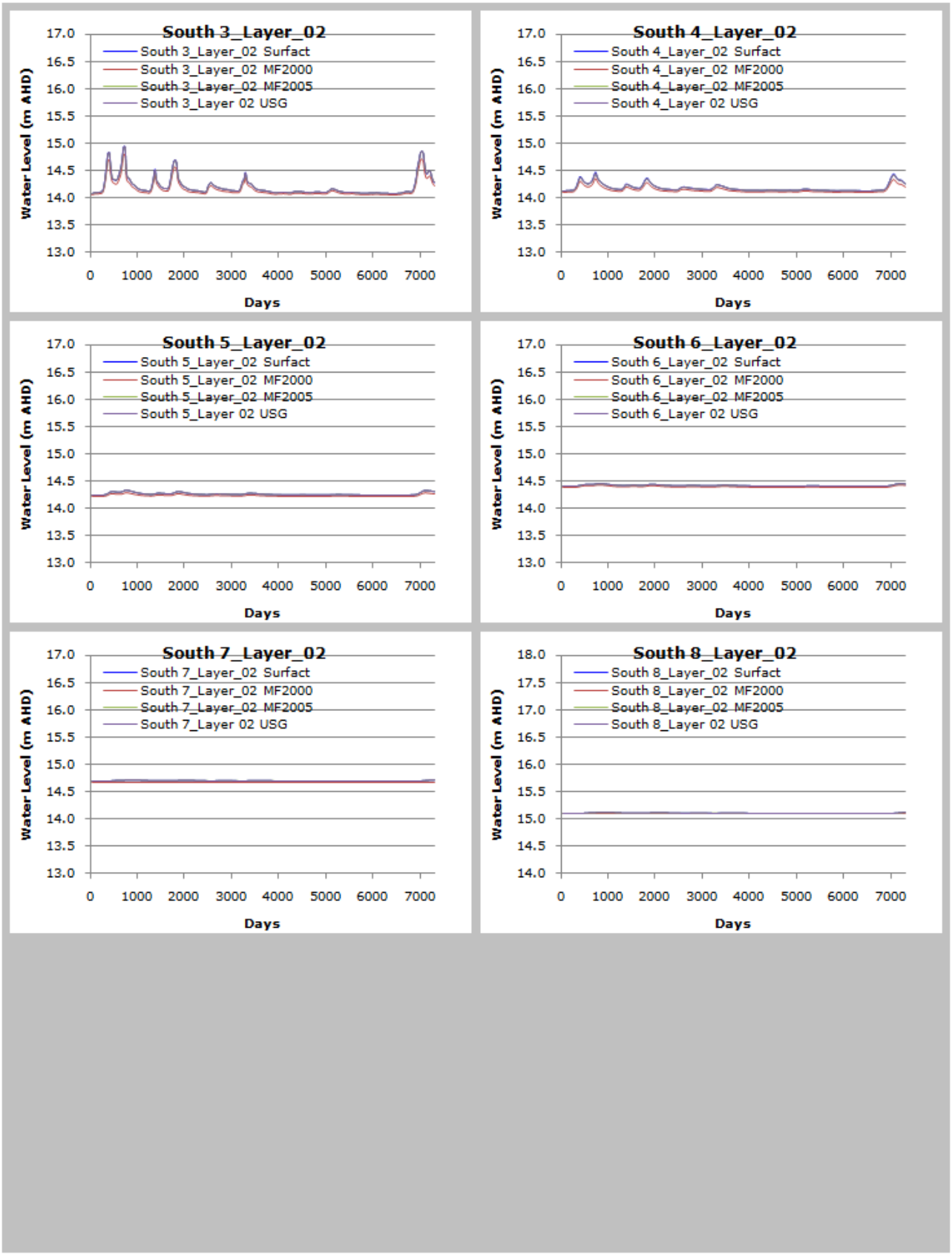
GOYDER MODEL BASECASE A



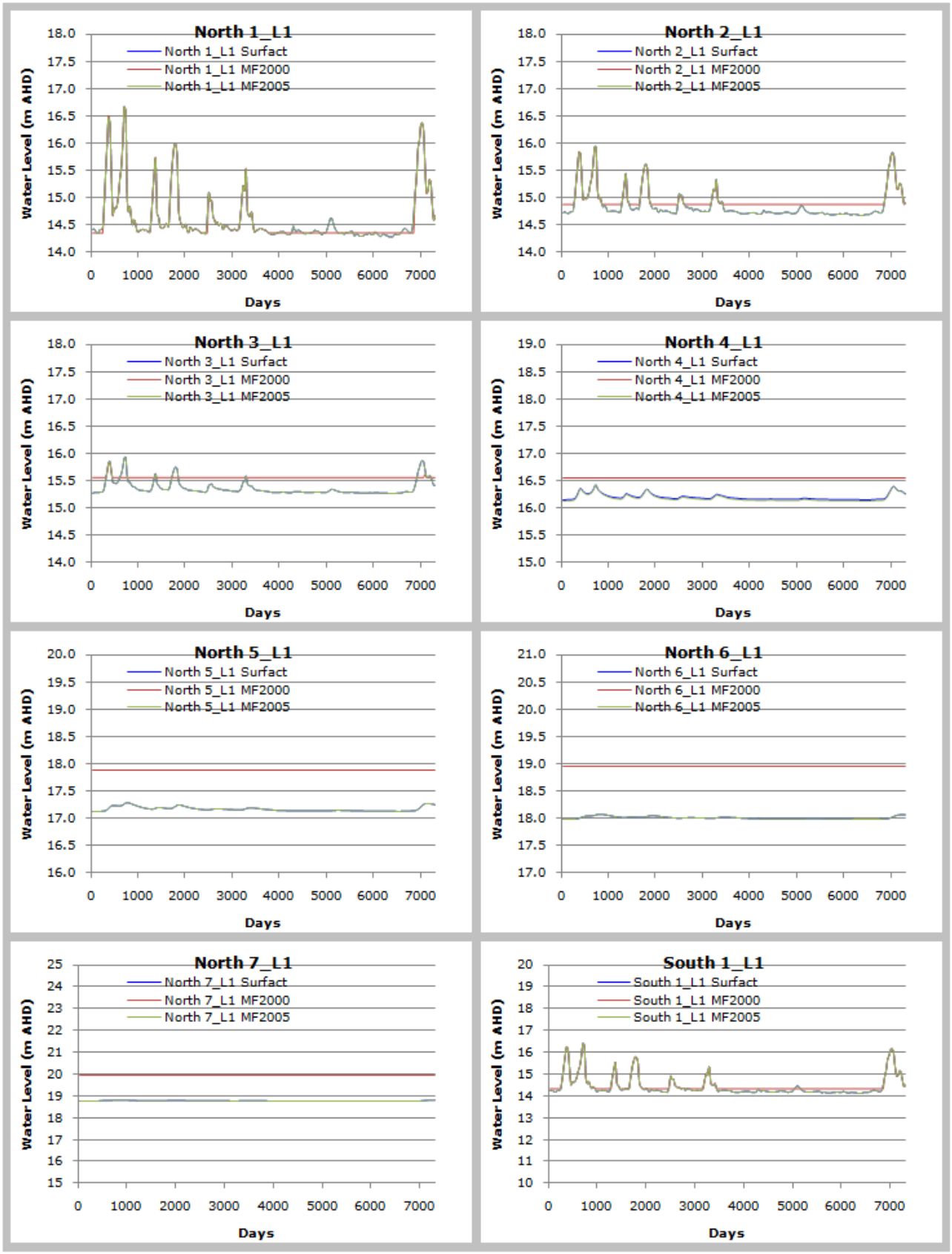
GOYDER MODEL BASECASE A



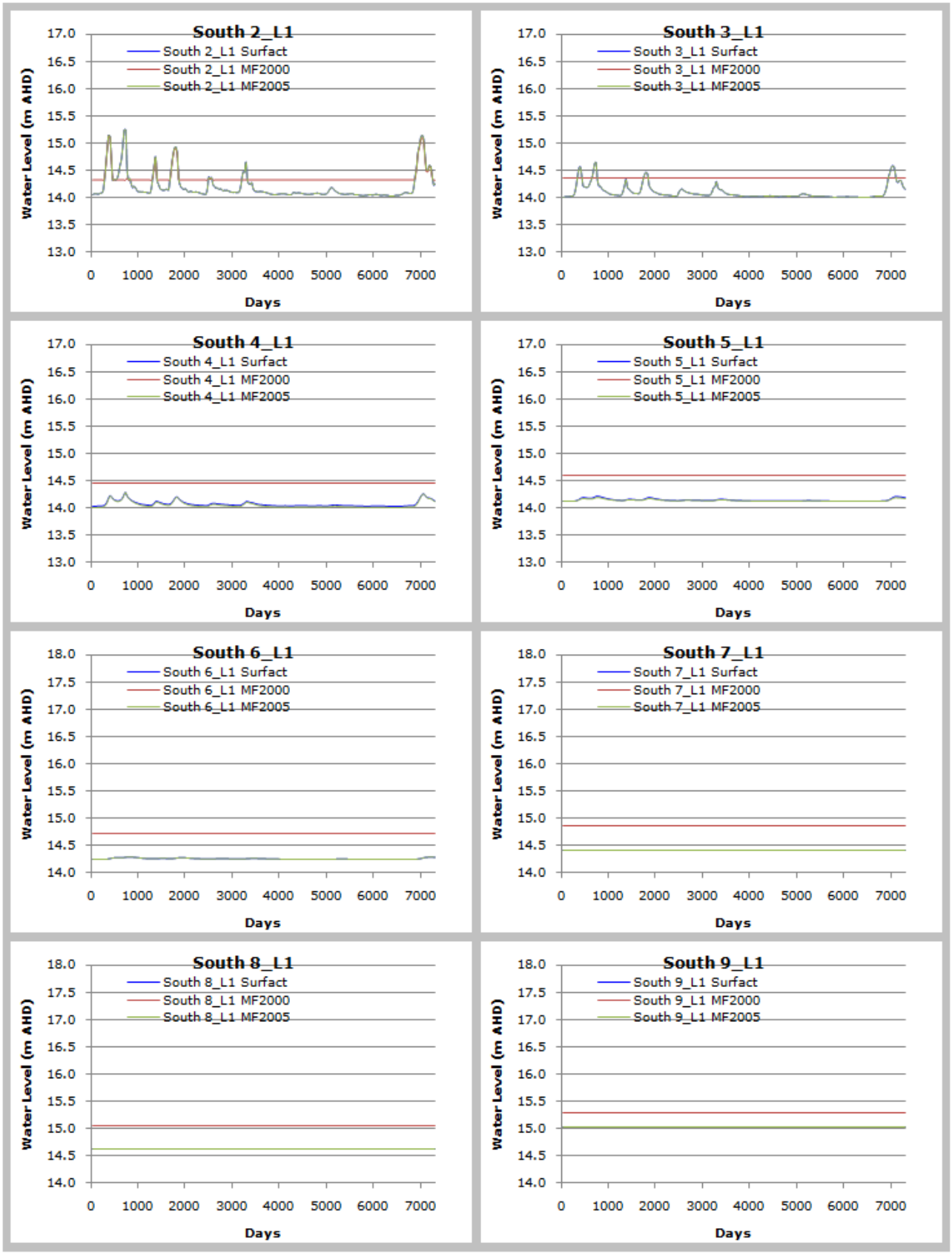
GOYDER MODEL BASECASE A



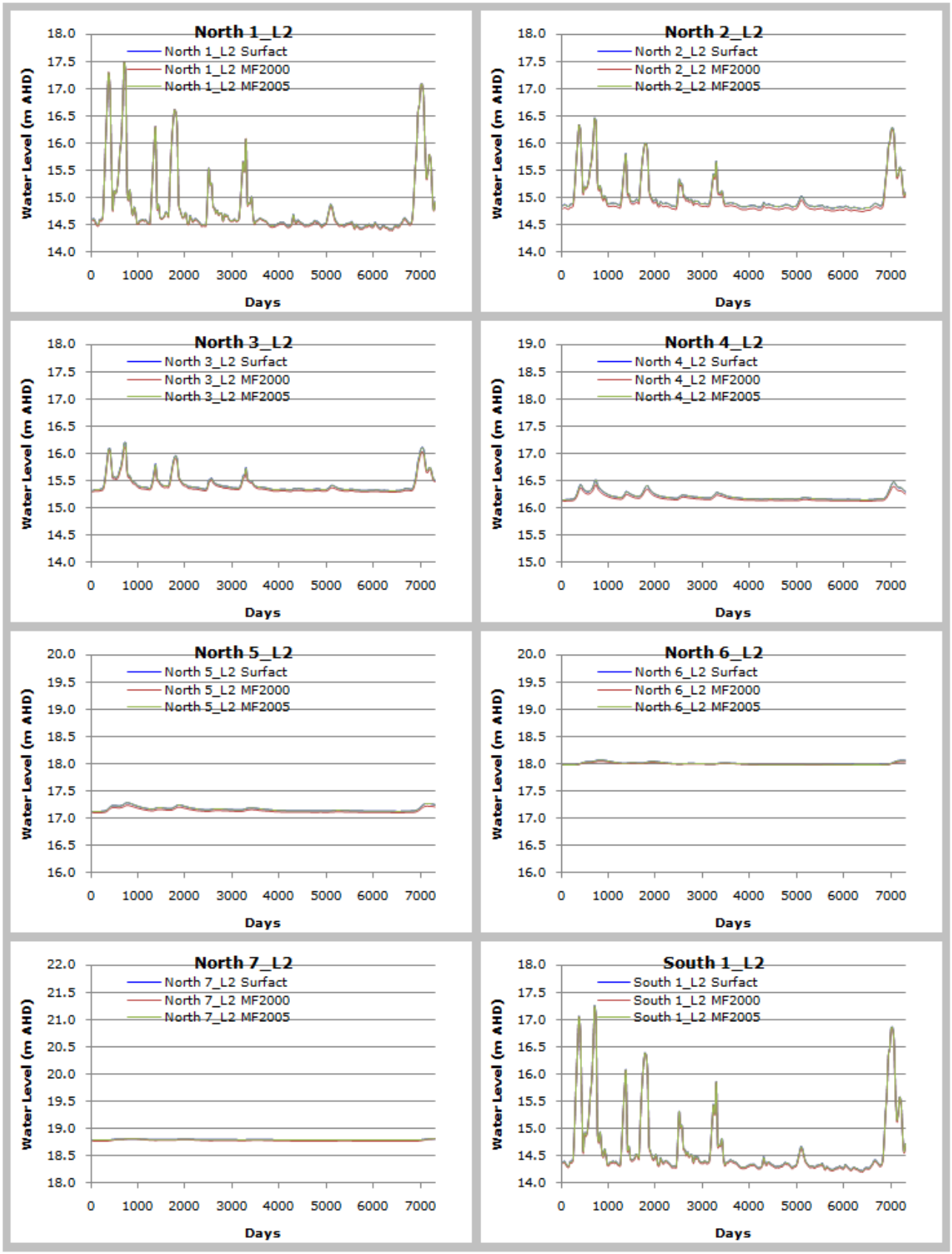
GOYDER MODEL BASECASE A



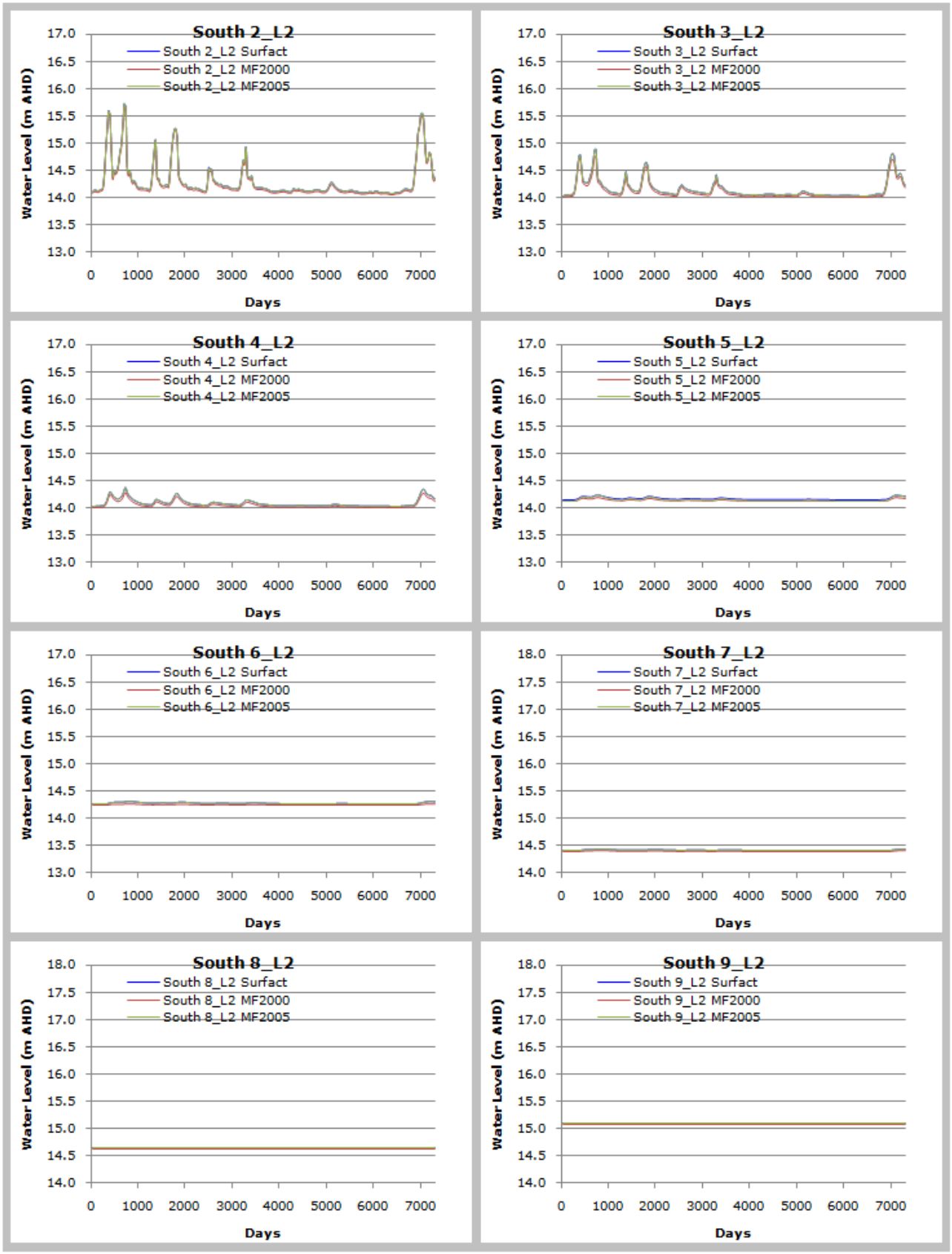
GOYDER MODEL BASECASE B



GOYDER MODEL BASECASE B



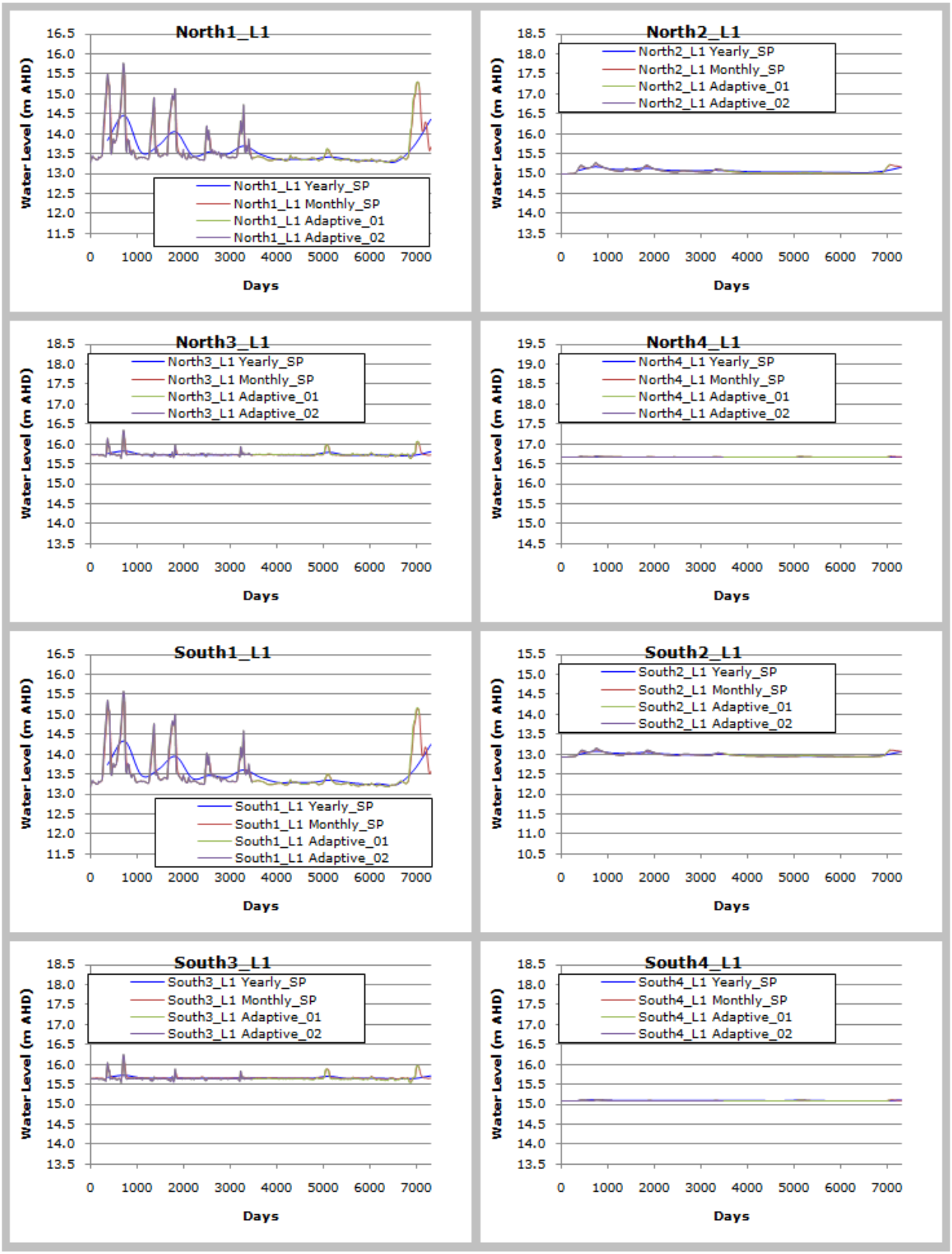
GOYDER MODEL BASECASE B



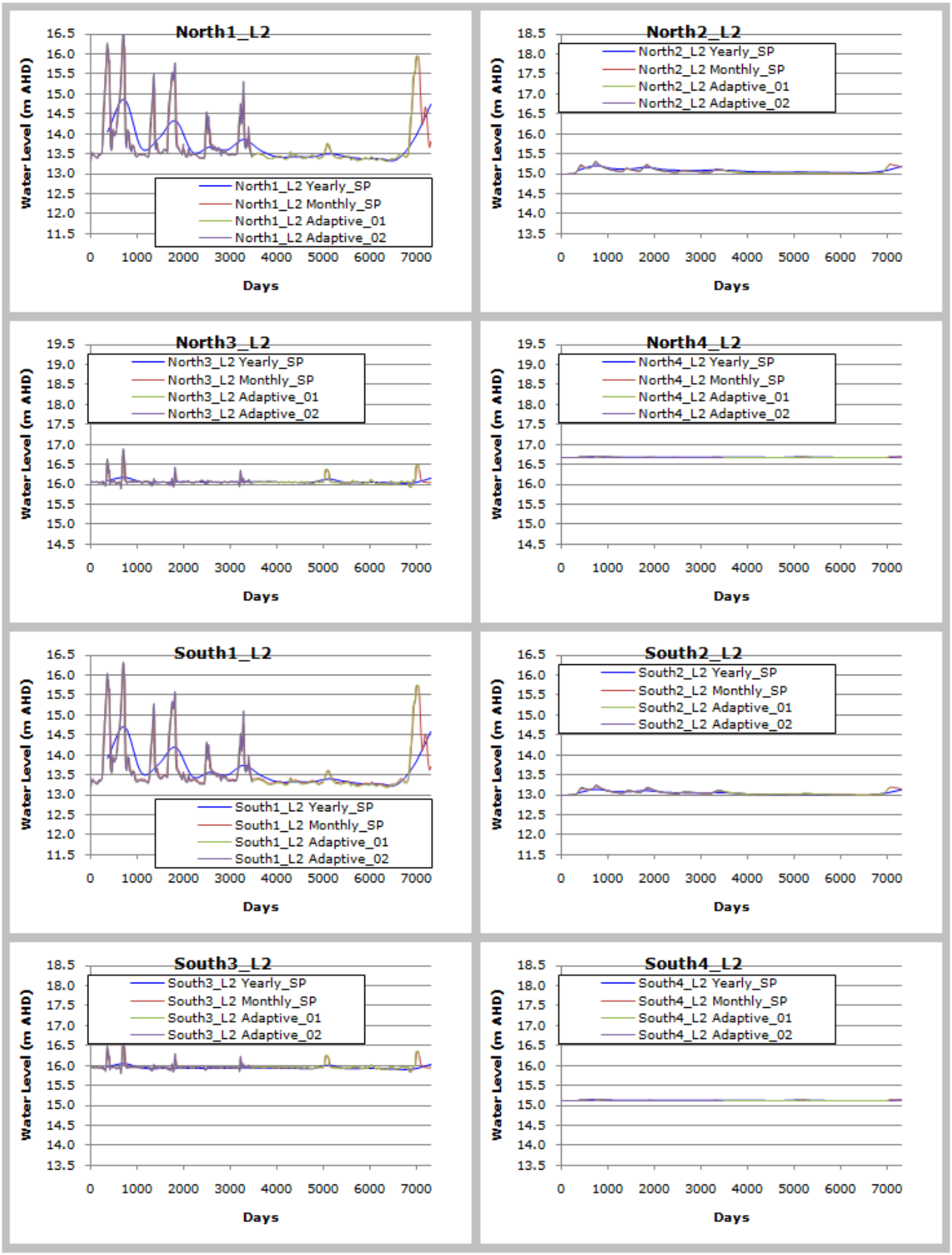
GOYDER MODEL BASECASE B

Appendix C2: Model results for Scenario 1 – Model A – Locked

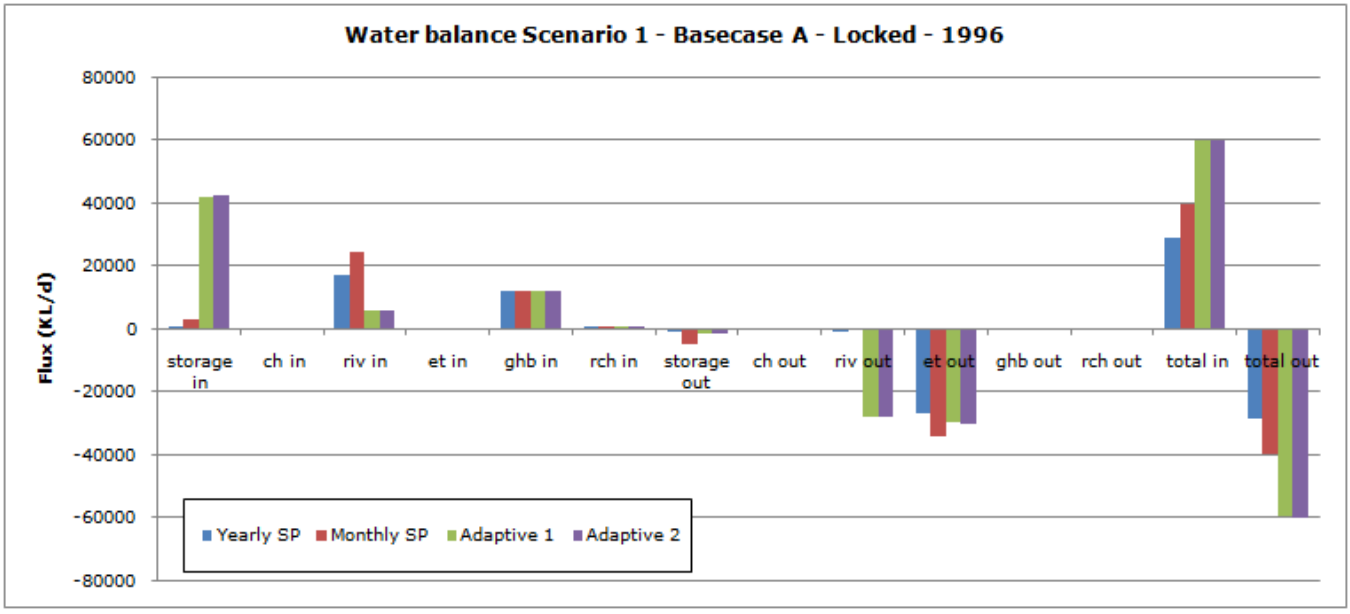
The following figures present the transient model behaviour for the scenario in the title. The lines represent each temporal discretisation approach with yearly, monthly, and adaptive variants. The adaptive variants utilize a stress period determined via the Python scripts included in Appendix A.

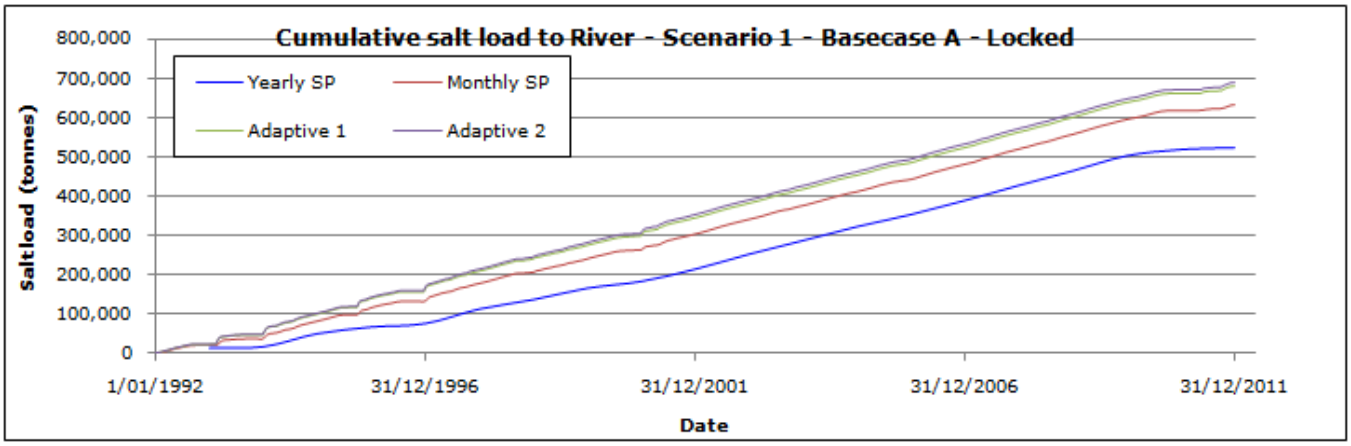


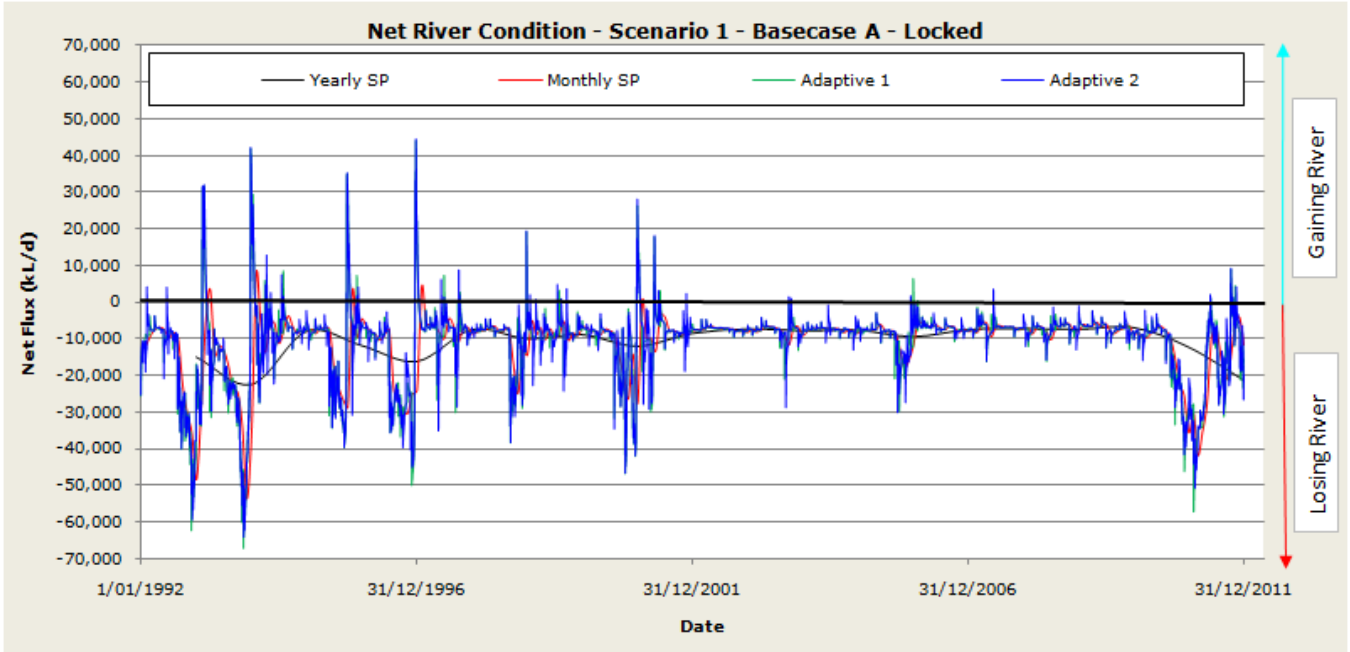
SCENARIO 1 GOYDER MODEL A - LOCKED



SCENARIO 1 GOYDER MODEL A - LOCKED

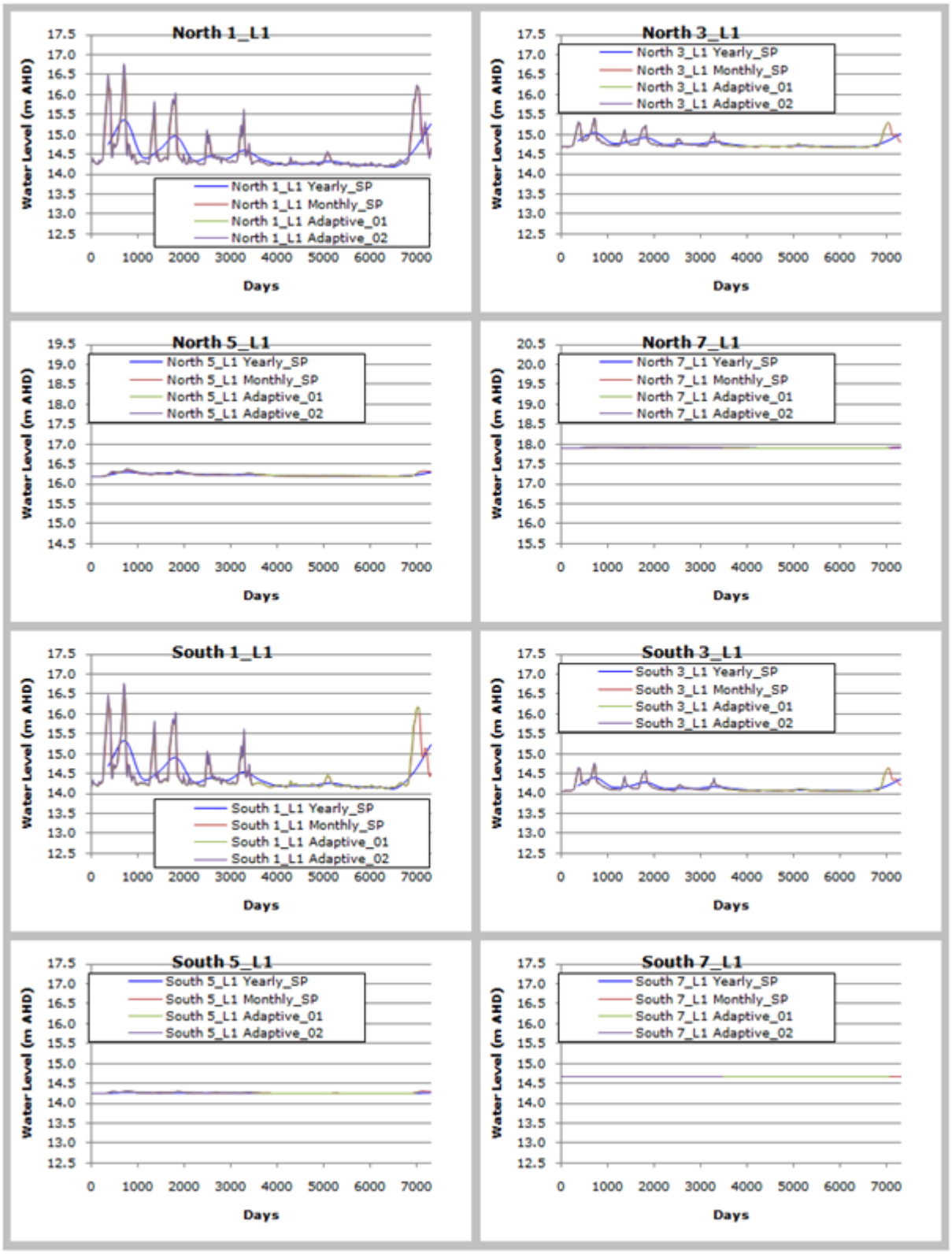

Water balance Scenario 1 - Basecase A - Locked - 1996
Flux (KL/d)
80000
60000
40000
20000
0
-20000
-40000
-60000
-80000
storage in
ch in
riv in
et in
ghb in
rch in
storage out
ch out
riv out
et out
ghb out
rch out
total in
total out
Yearly SP
Monthly SP
Adaptive 1
Adaptive 2


Cumulative salt load to River - Scenario 1 - Basecase A - Locked
Salt load (tonnes)
800,000
700,000
600,000
500,000
400,000
300,000
200,000
100,000
0
Yearly SP
Monthly SP
Adaptive 1
Adaptive 2
1/01/1992
31/12/1996
31/12/2001
31/12/2006
31/12/2011
Date

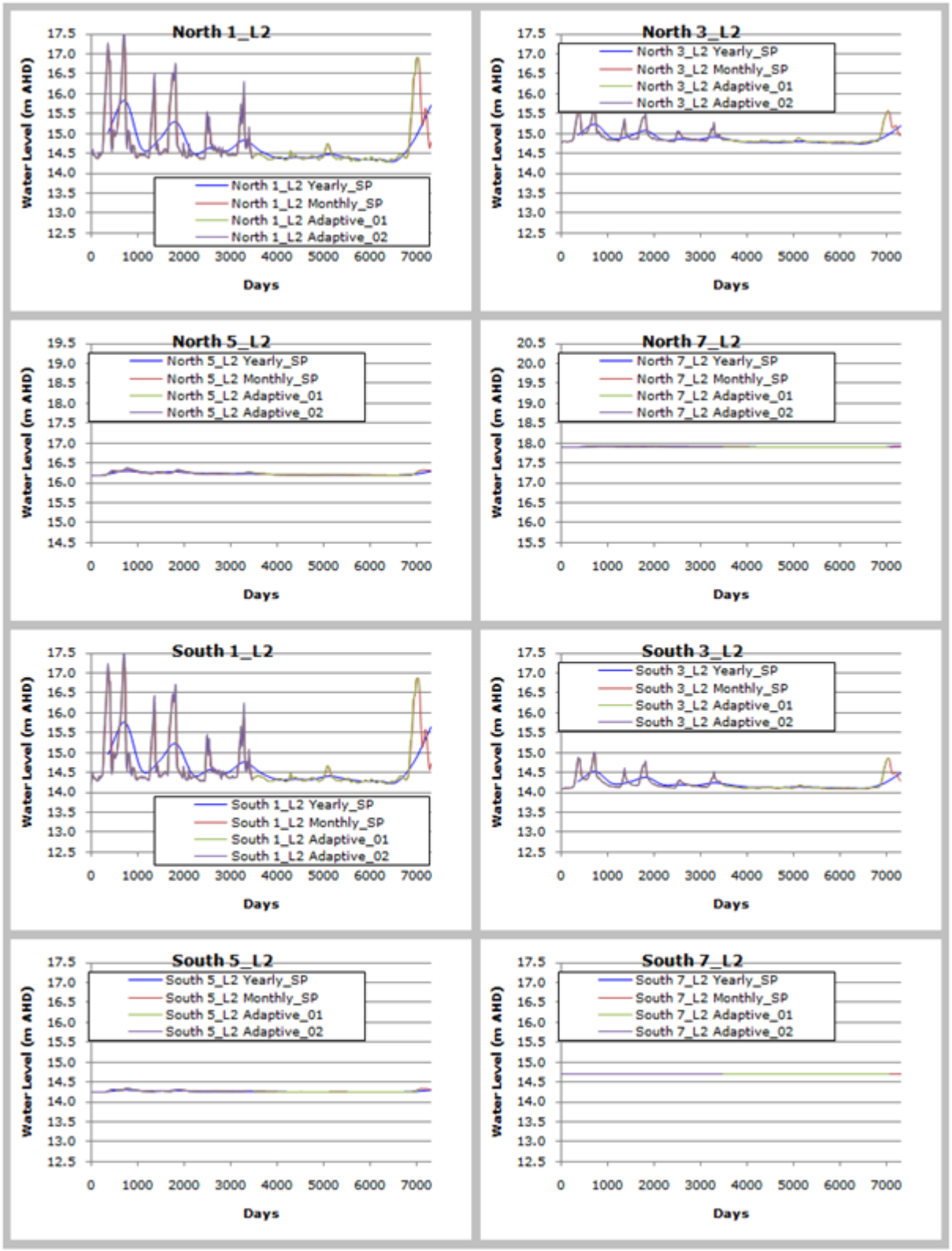

Net River Condition - Scenario 1 - Basecase A - Locked
Net Flux (kL/d)
70,000
60,000
50,000
40,000
30,000
20,000
10,000
0
-10,000
-20,000
-30,000
-40,000
-50,000
-60,000
-70,000
Yearly SP
Monthly SP
Adaptive 1
Adaptive 2
Gaining River
Losing River
1/01/1992
31/12/1996
31/12/2001
31/12/2006
31/12/2011
Date

Appendix C3: Model results for Scenario 1 – Model A – Not locked

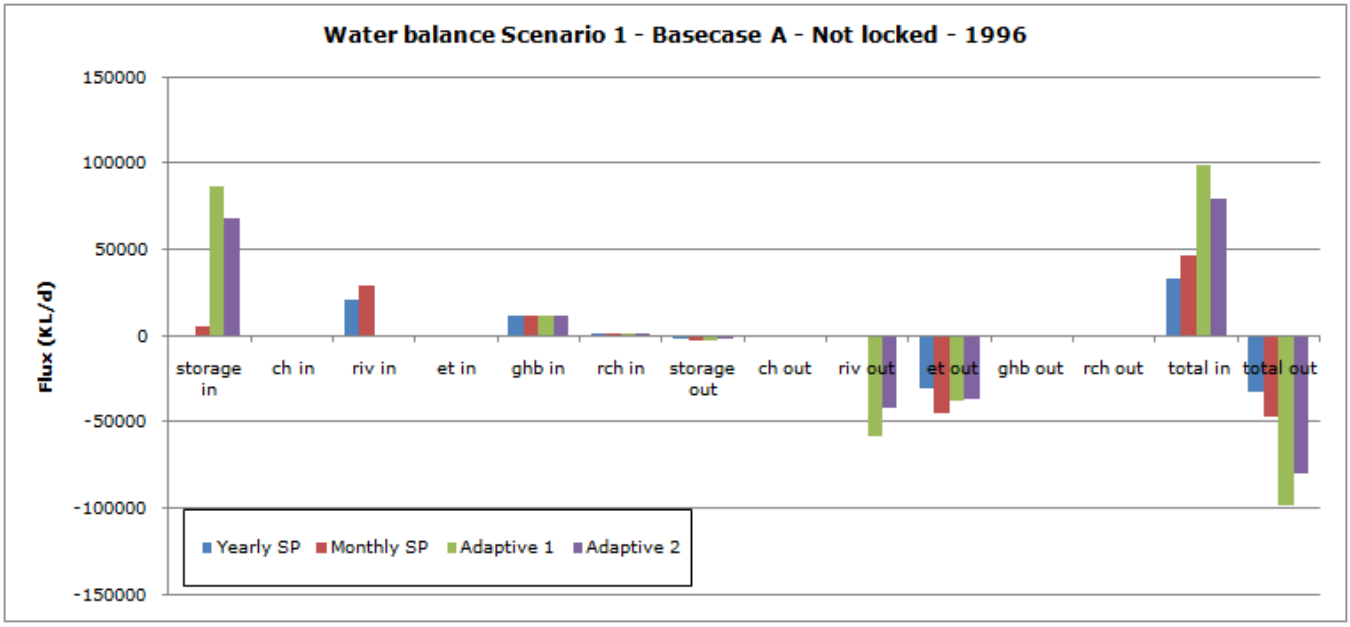
The following figures present the transient model behaviour for the scenario in the title. The lines represent each temporal discretisation approach with yearly, monthly, and adaptive variants. The adaptive variants utilize a stress period determined via the Python scripts included in Appendix A.

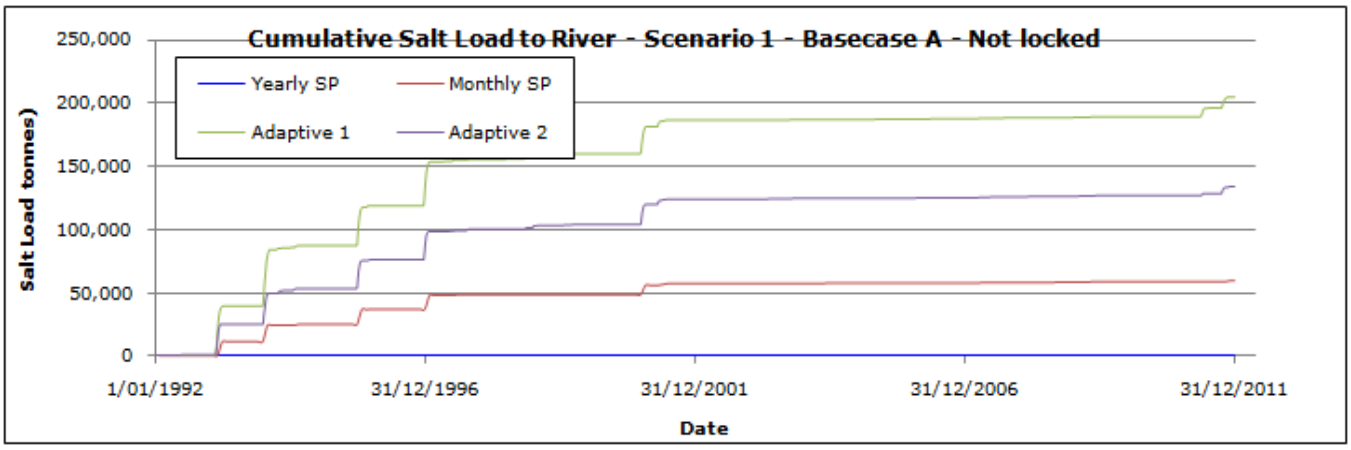


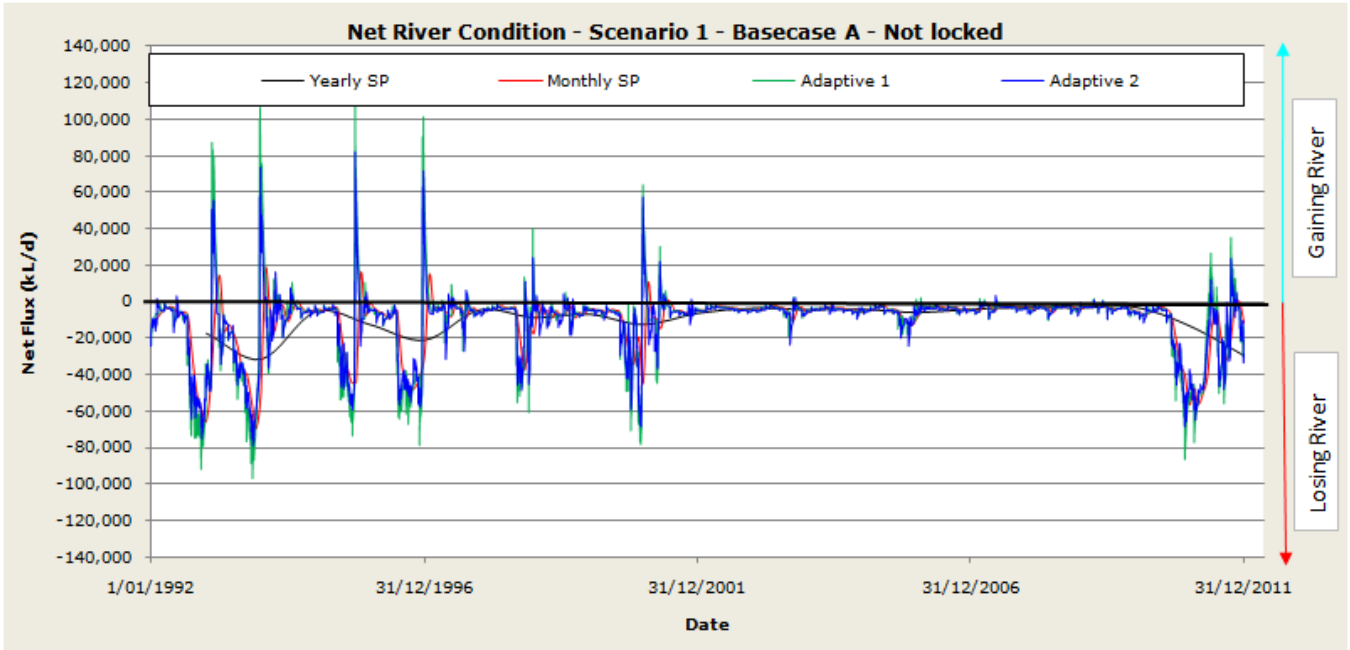
SCENARIO 1 - GOYDER MODEL A - NOT LOCKED



SCENARIO 1 - GOYDER MODEL A - NOT LOCKED

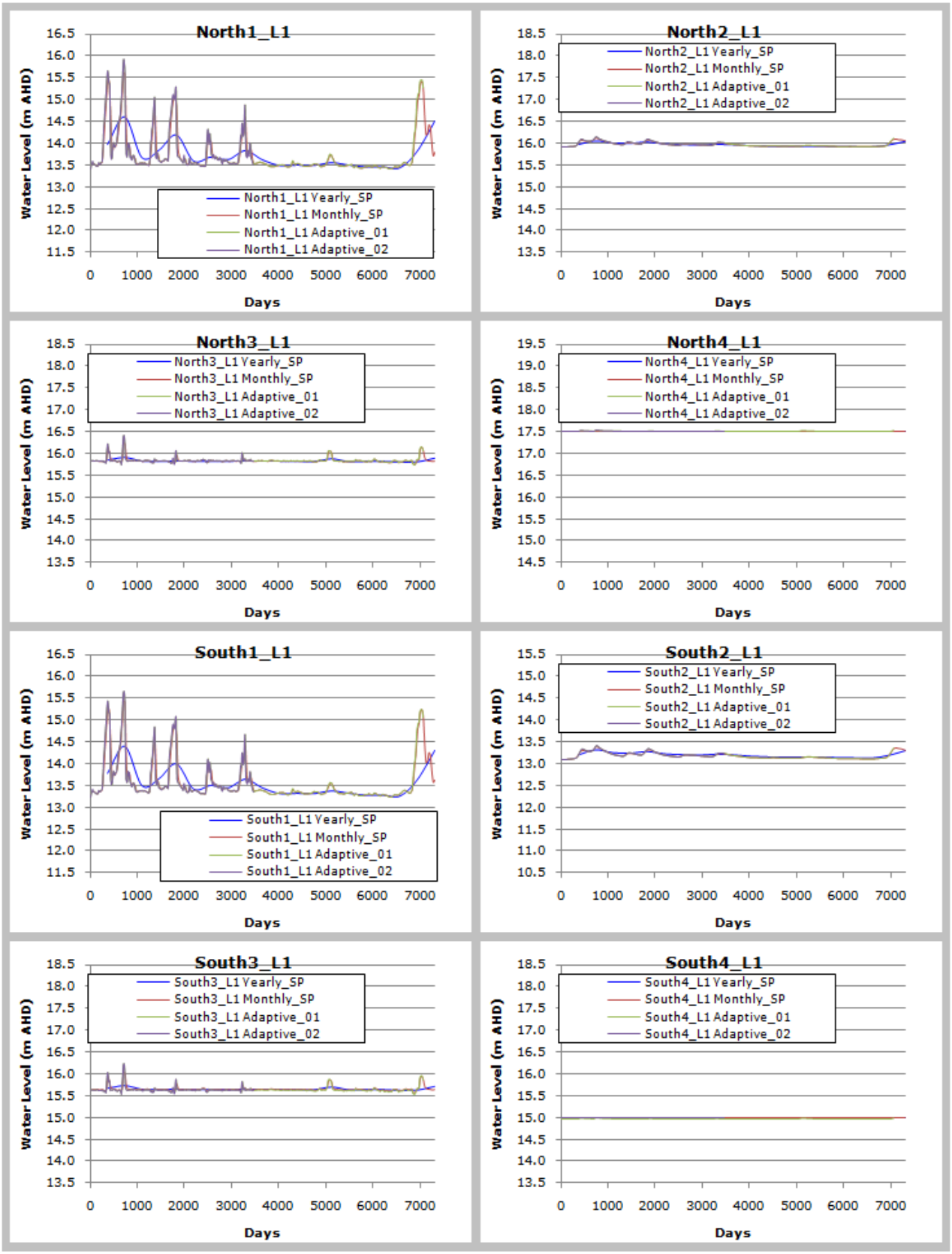

Water balance Scenario 1 - Basecase A - Not locked - 1996
Flux (KL/d)
150000
100000
50000
0
-50000
-100000
-150000
storage in
ch in
riv in
et in
ghb in
rch in
storage out
ch out
riv out
et out
ghb out
rch out
total in
total out
Yearly SP
Monthly SP
Adaptive 1
Adaptive 2


Cumulative Salt Load to River - Scenario 1 - Basecase A - Not locked
Salt Load (tonnes)
250,000
200,000
150,000
100,000
50,000
0
Yearly SP
Monthly SP
Adaptive 1
Adaptive 2
1/01/1992
31/12/1996
31/12/2001
31/12/2006
31/12/2011
Date

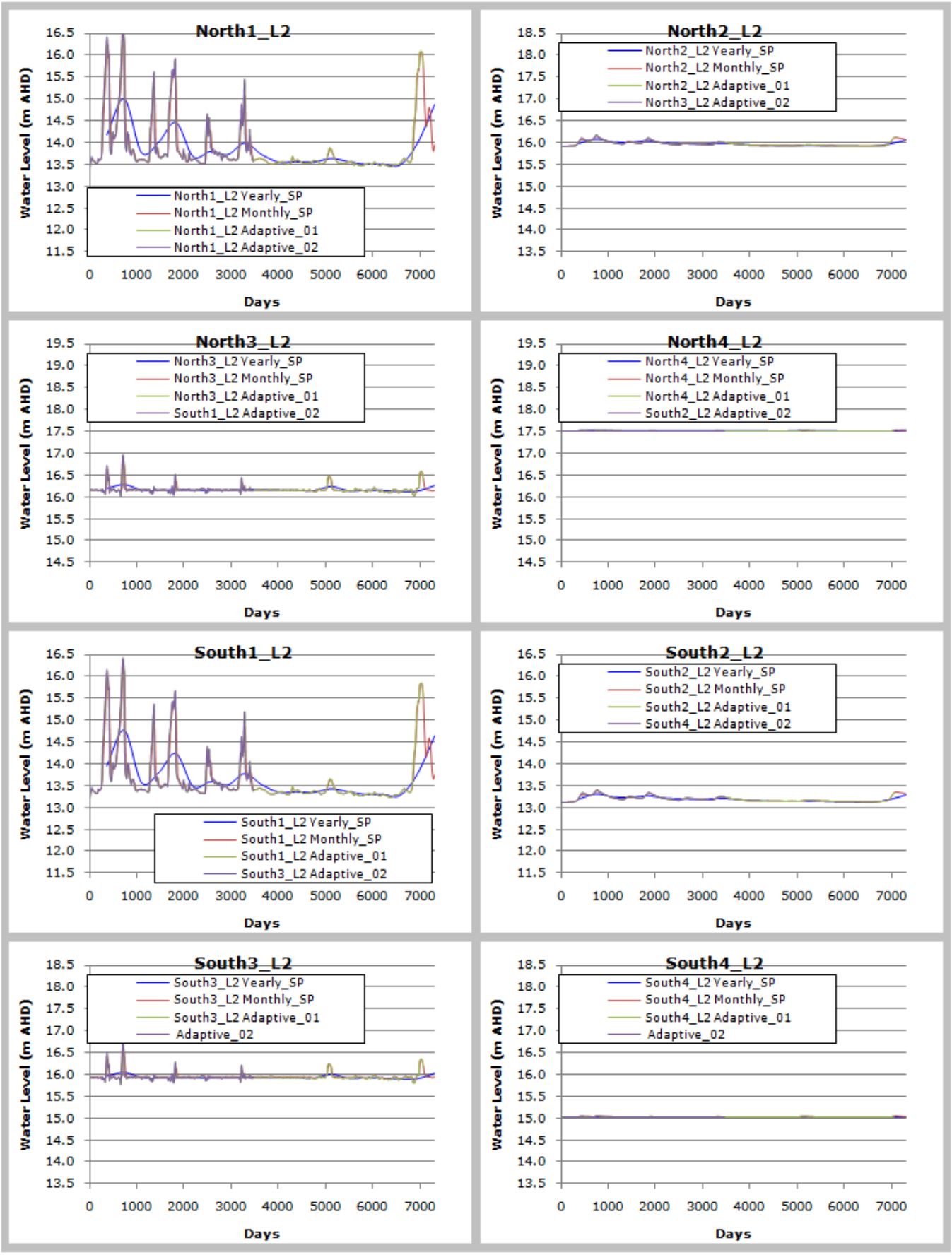

Net River Condition - Scenario 1 - Basecase A - Not locked
Net Flux (kL/d)
140,000
120,000
100,000
80,000
60,000
40,000
20,000
0
-20,000
-40,000
-60,000
-80,000
-100,000
-120,000
-140,000
Yearly SP
Monthly SP
Adaptive 1
Adaptive 2
Gaining River
Losing River
1/01/1992
31/12/1996
31/12/2001
31/12/2006
31/12/2011
Date

Appendix C4: Model result Scenario 1 – Model B – Locked

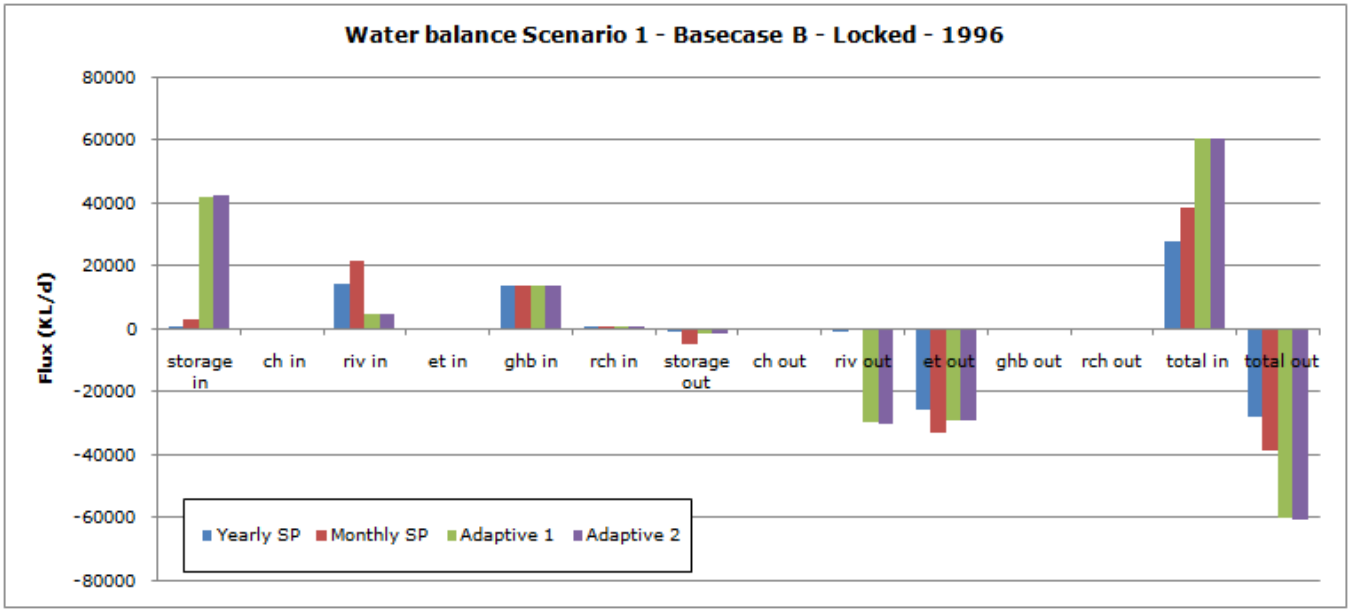
The following figures present the transient model behaviour for the scenario in the title. The lines represent each temporal discretisation approach with yearly, monthly, and adaptive variants. The adaptive variants utilize a stress period determined via the Python scripts included in Appendix A.

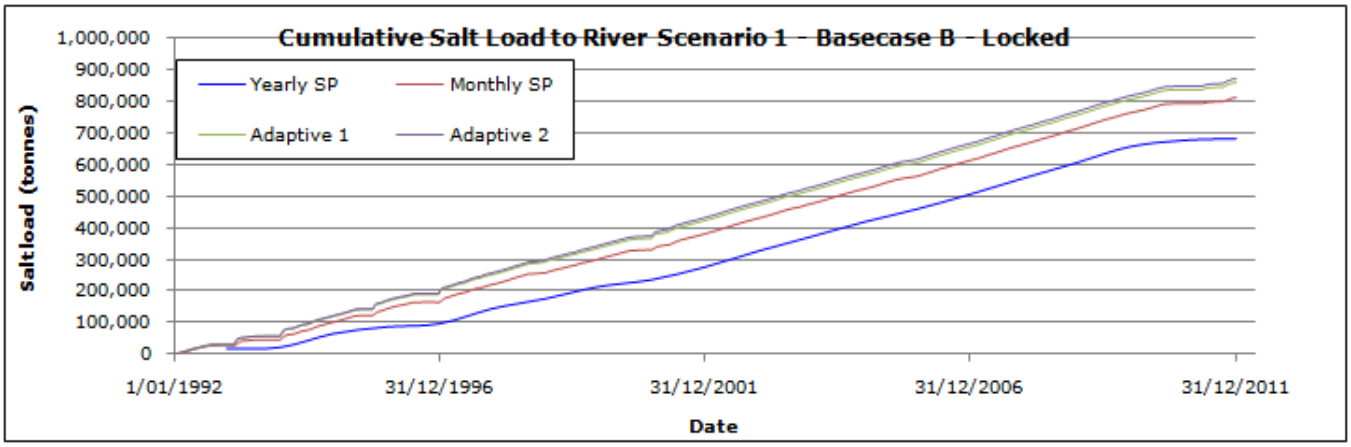


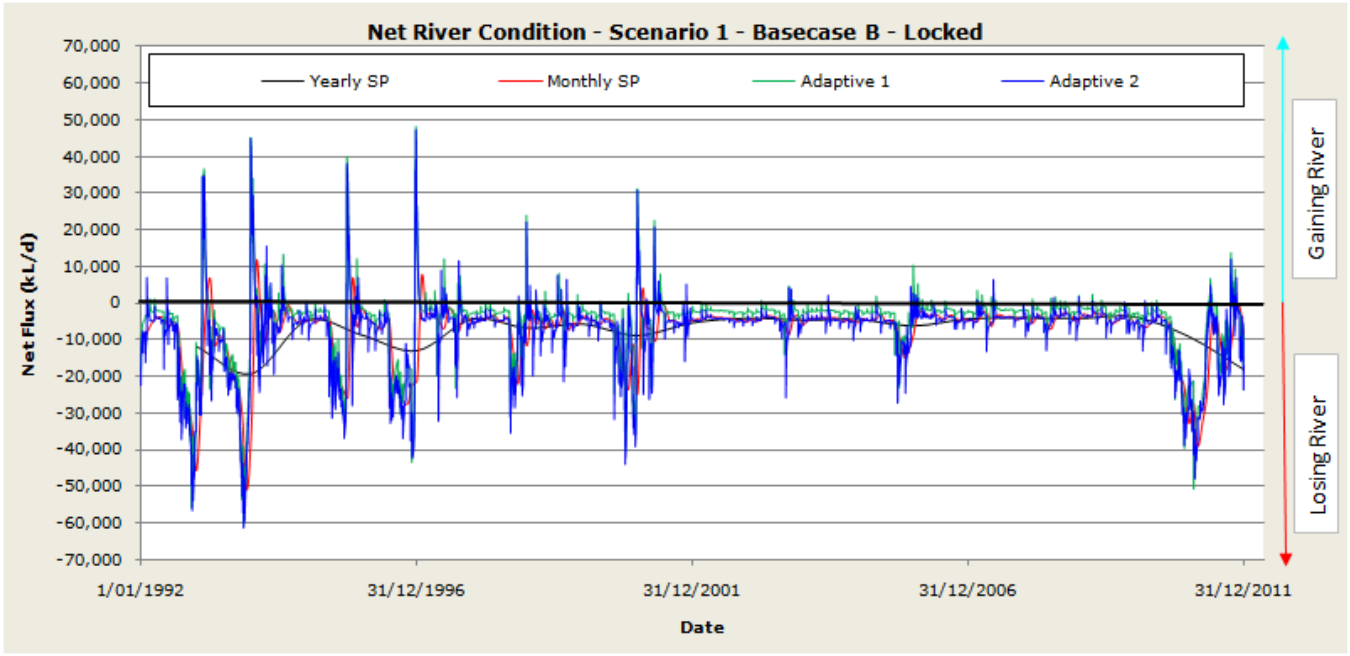
SCENARIO 1 GOYDER MODEL B - LOCKED



SCENARIO 1 GOYDER MODEL B - LOCKED

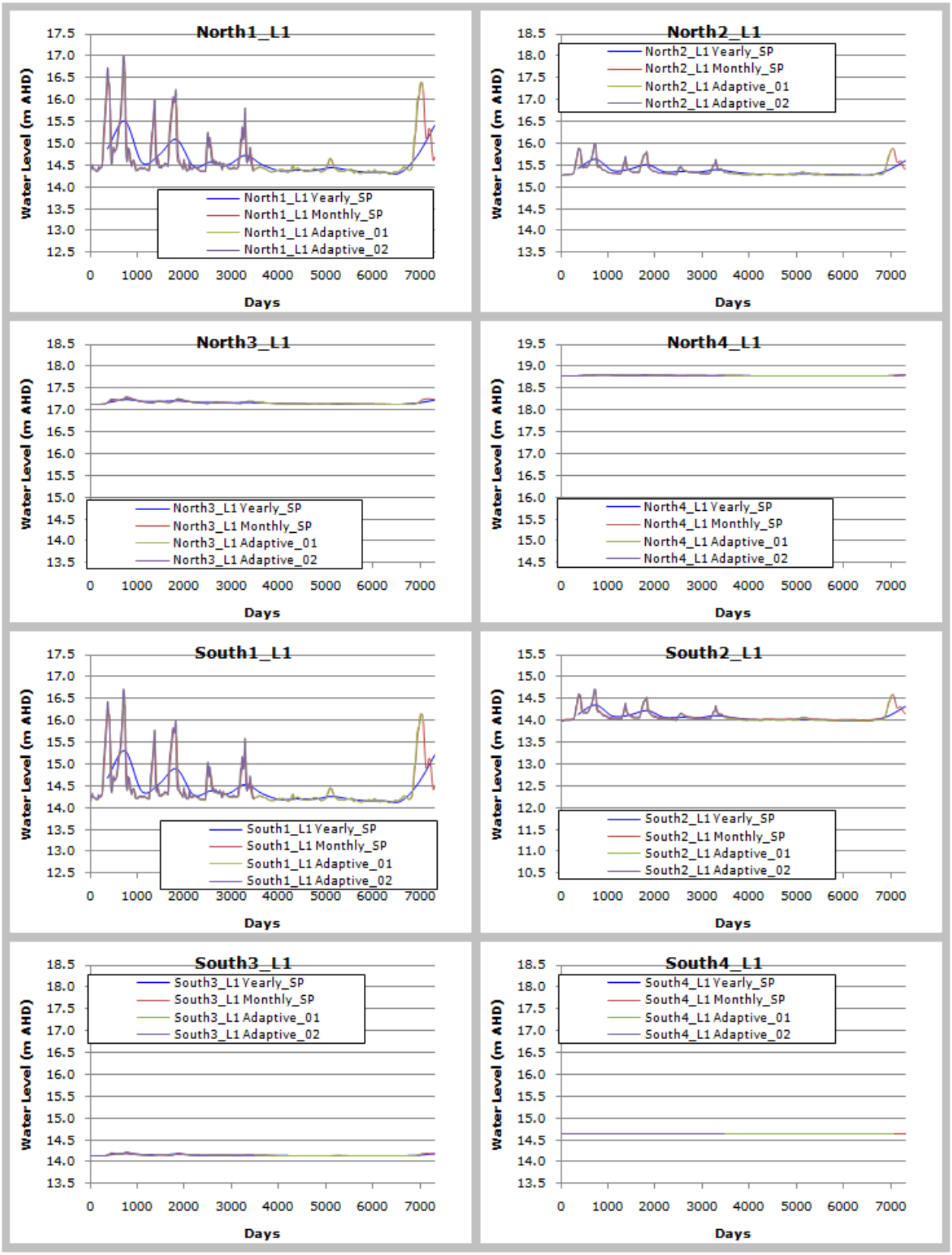

Water balance Scenario 1 - Basecase B - Locked - 1996
Flux (KL/d)
80000
60000
40000
20000
0
-20000
-40000
-60000
-80000
storage in
ch in
riv in
et in
ghb in
rch in
storage out
ch out
riv out
et out
ghb out
rch out
total in
total out
Yearly SP
Monthly SP
Adaptive 1
Adaptive 2


Cumulative Salt Load to River Scenario 1 - Basecase B - Locked
Salt load (tonnes)
1,000,000
900,000
800,000
700,000
600,000
500,000
400,000
300,000
200,000
100,000
0
Yearly SP
Monthly SP
Adaptive 1
Adaptive 2
1/01/1992
31/12/1996
31/12/2001
31/12/2006
31/12/2011
Date

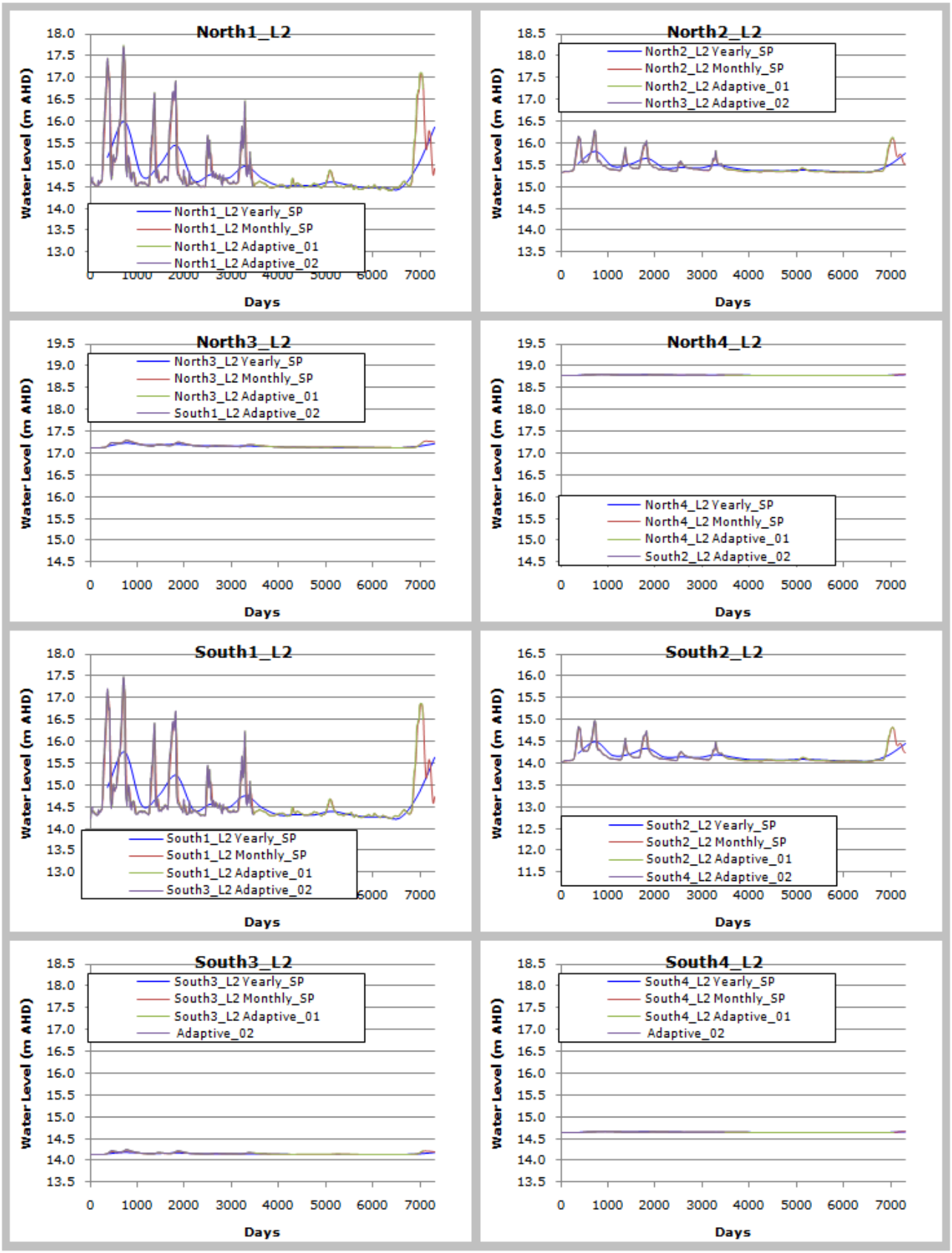

Net River Condition - Scenario 1 - Basecase B - Locked
Net Flux (kL/d)
70,000
60,000
50,000
40,000
30,000
20,000
10,000
0
-10,000
-20,000
-30,000
-40,000
-50,000
-60,000
-70,000
Yearly SP
Monthly SP
Adaptive 1
Adaptive 2
Gaining River
Losing River
1/01/1992
31/12/1996
31/12/2001
31/12/2006
31/12/2011
Date

Appendix C5: Model results for Scenario 1 – Model B – Not locked

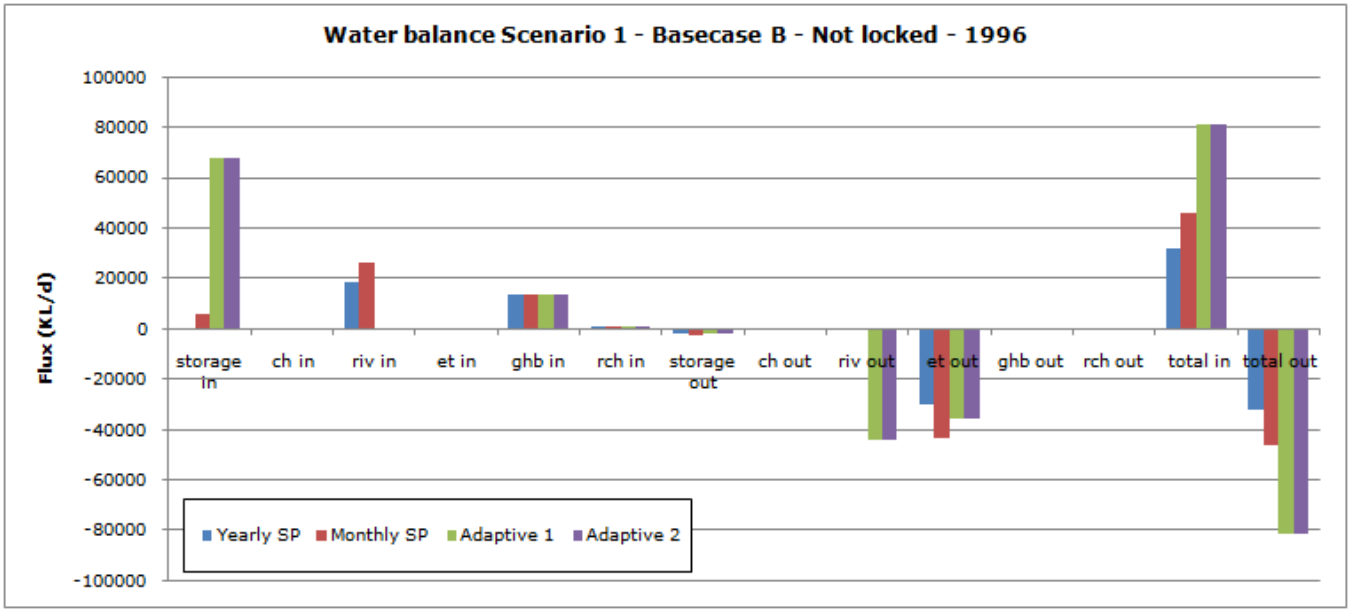
The following figures present the transient model behaviour for the scenario in the title. The lines represent each temporal discretisation approach with yearly, monthly, and adaptive variants. The adaptive variants utilize a stress period determined via the Python scripts included in Appendix A.

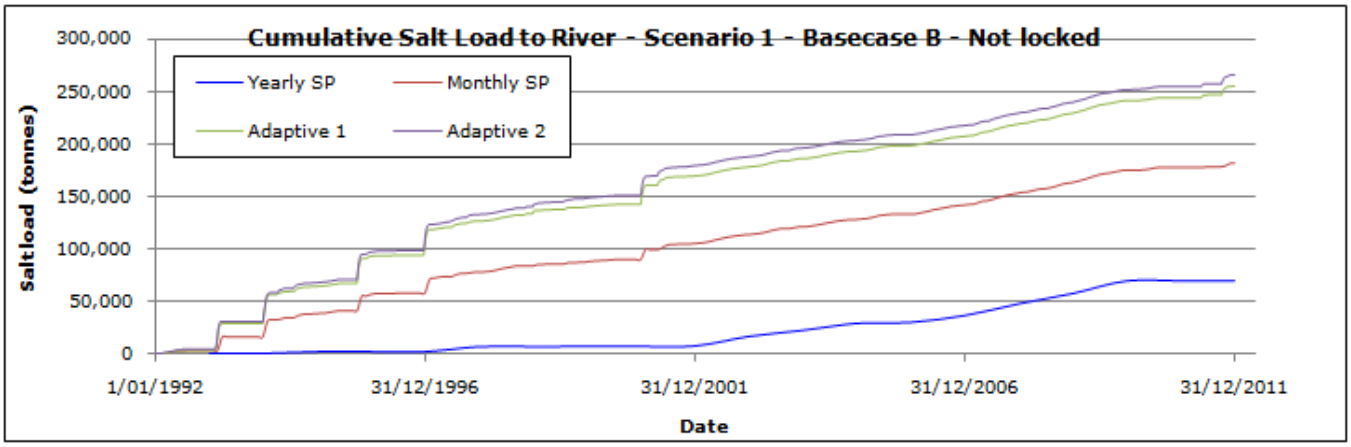


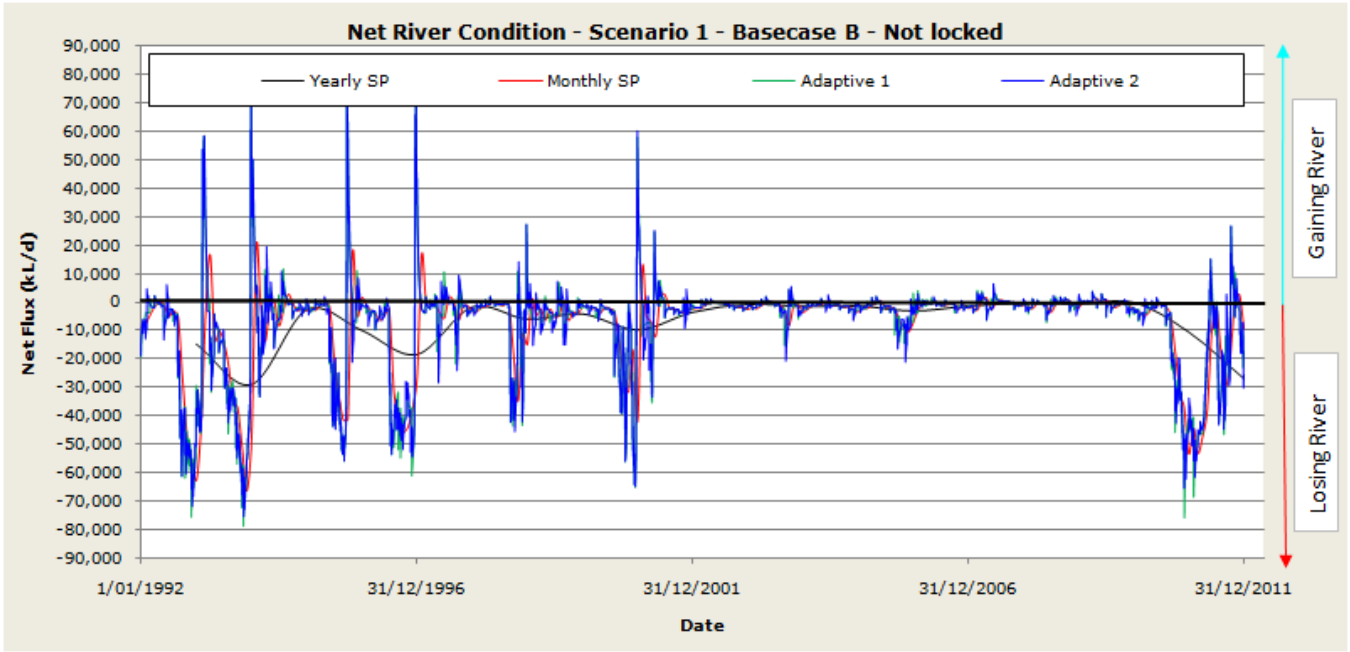
SCENARIO 1 - GOYDER MODEL B - NOT LOCKED



SCENARIO 1 - GOYDER MODEL B - NOT LOCKED

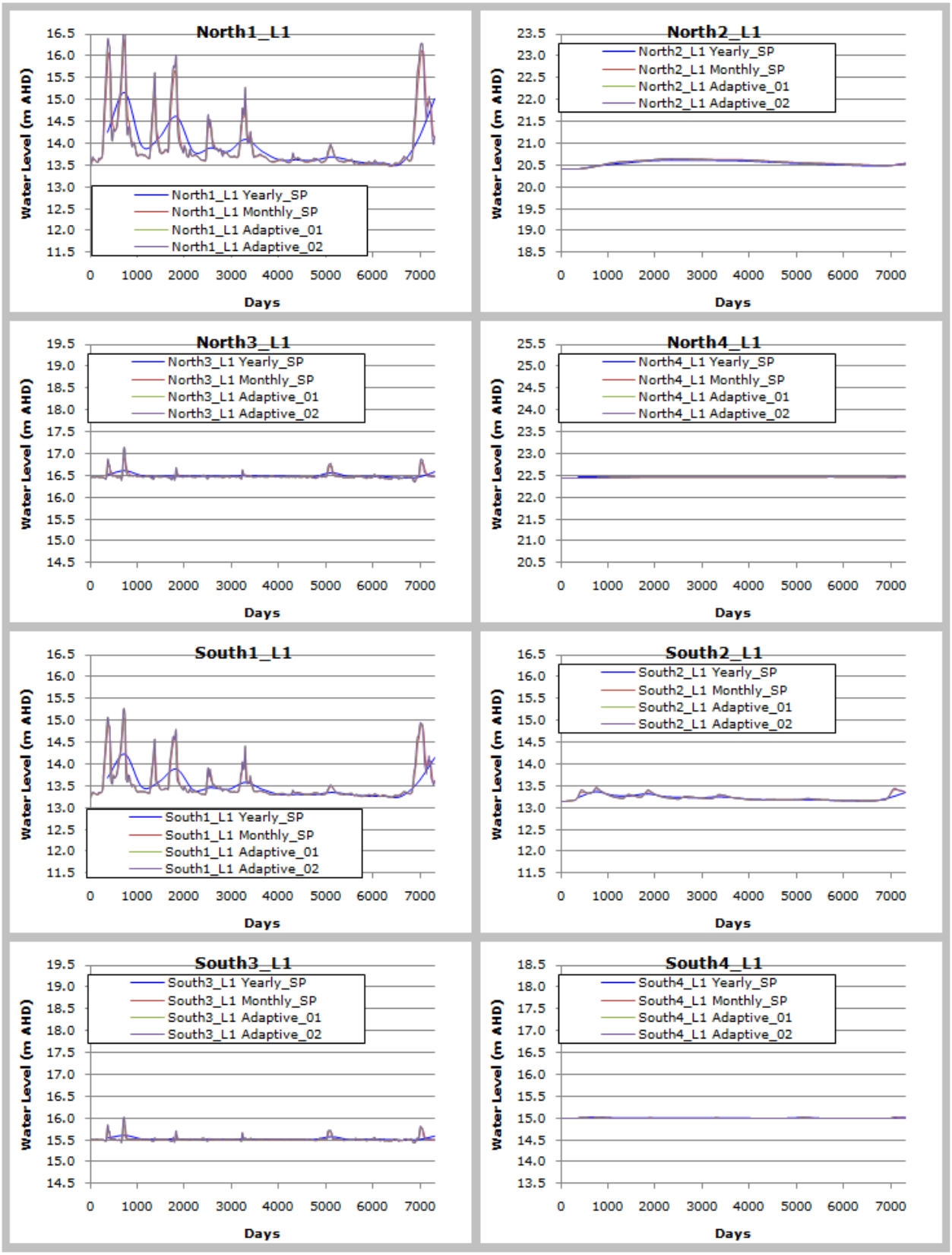

Water balance Scenario 1 - Basecase B - Not locked - 1996
Flux (KL/d)
100000
80000
60000
40000
20000
0
-20000
-40000
-60000
-80000
-100000
storage in
ch in
riv in
et in
ghb in
rch in
storage out
ch out
riv out
et out
ghb out
rch out
total in
total out
Yearly SP
Monthly SP
Adaptive 1
Adaptive 2


Cumulative Salt Load to River - Scenario 1 - Basecase B - Not locked
Salt load (tonnes)
300,000
250,000
200,000
150,000
100,000
50,000
0
Yearly SP
Monthly SP
Adaptive 1
Adaptive 2
1/01/1992
31/12/1996
31/12/2001
31/12/2006
31/12/2011
Date

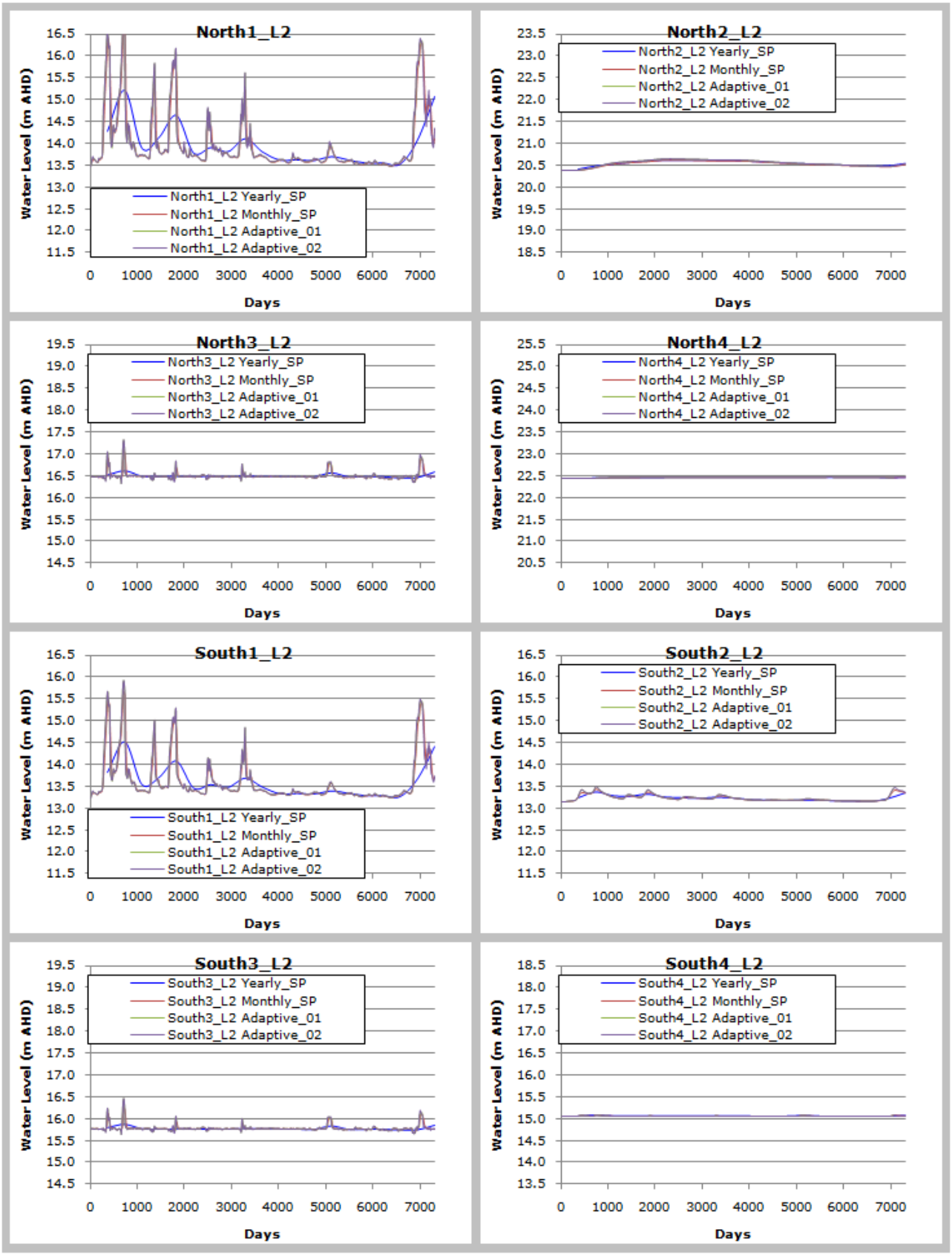

Net River Condition - Scenario 1 - Basecase B - Not locked
Net Flux (kL/d)
90,000
80,000
70,000
60,000
50,000
40,000
30,000
20,000
10,000
0
-10,000
-20,000
-30,000
-40,000
-50,000
-60,000
-70,000
-80,000
-90,000
Yearly SP
Monthly SP
Adaptive 1
Adaptive 2
Gaining River
Losing River
1/01/1992
31/12/1996
31/12/2001
31/12/2006
31/12/2011
Date

Appendix C6: Model results for Scenario 1 – Model C – Locked

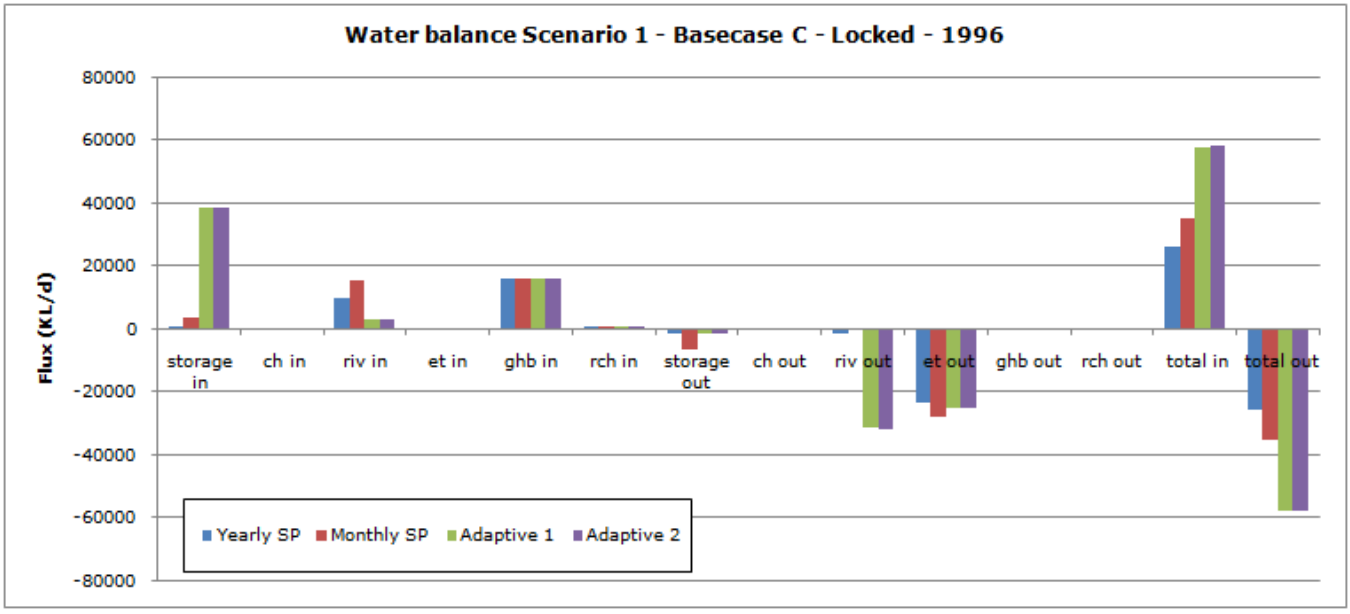
The following figures present the transient model behaviour for the scenario in the title. The lines represent each temporal discretisation approach with yearly, monthly, and adaptive variants. The adaptive variants utilize a stress period determined via the Python scripts included in Appendix A.

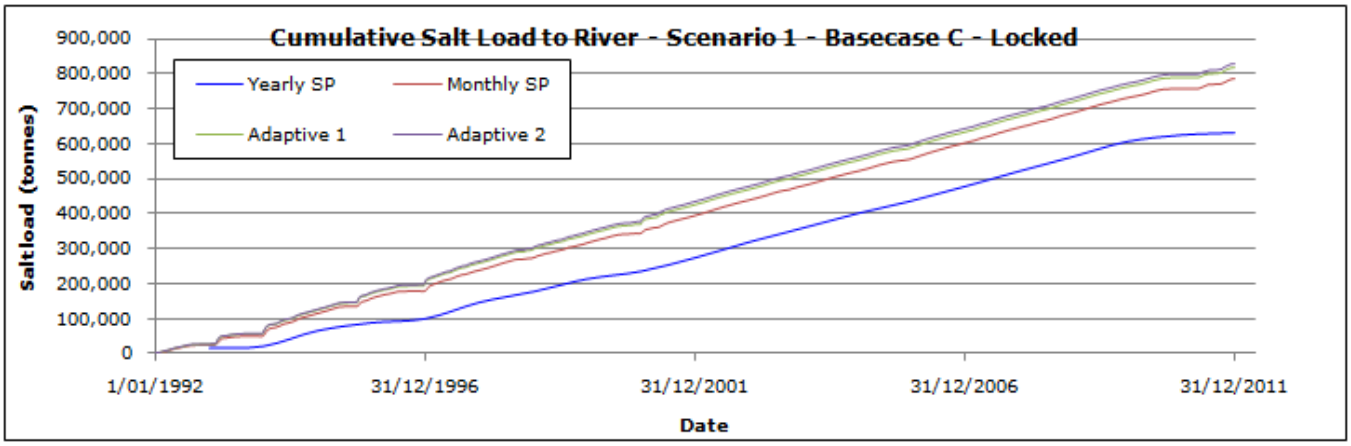


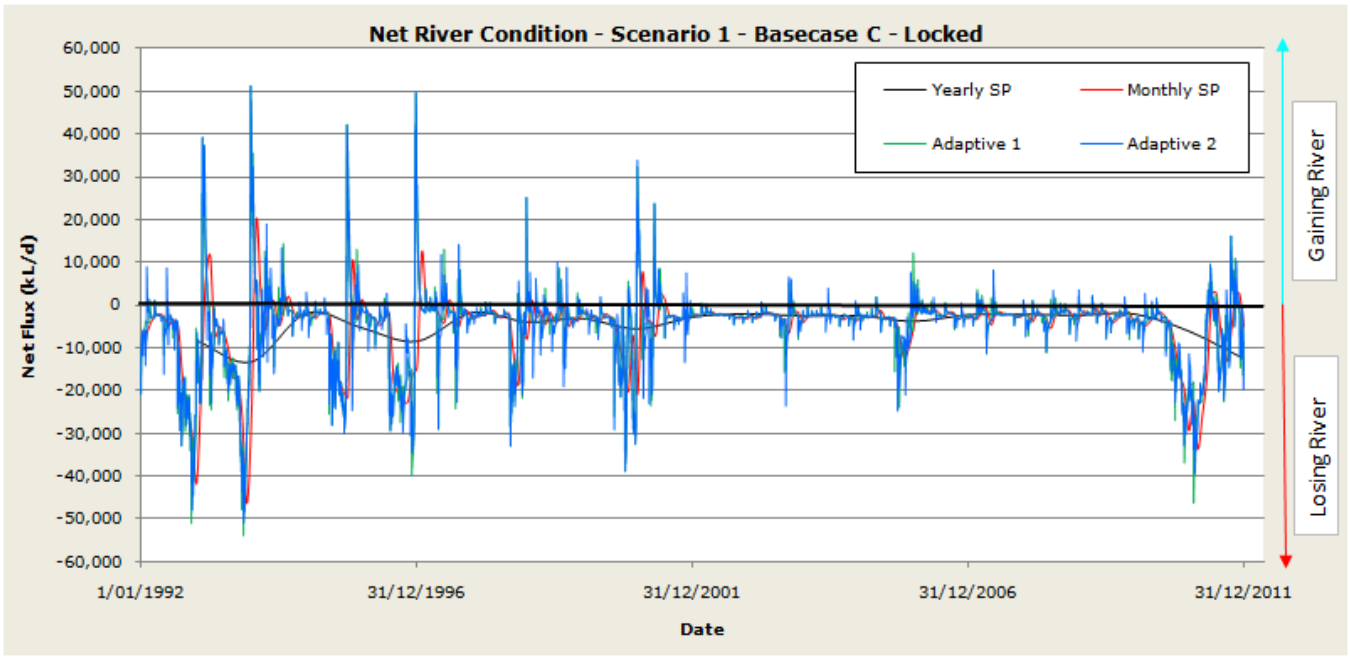
SCENARIO 1 GOYDER MODEL C - LOCKED



SCENARIO 1 GOYDER MODEL C - LOCKED

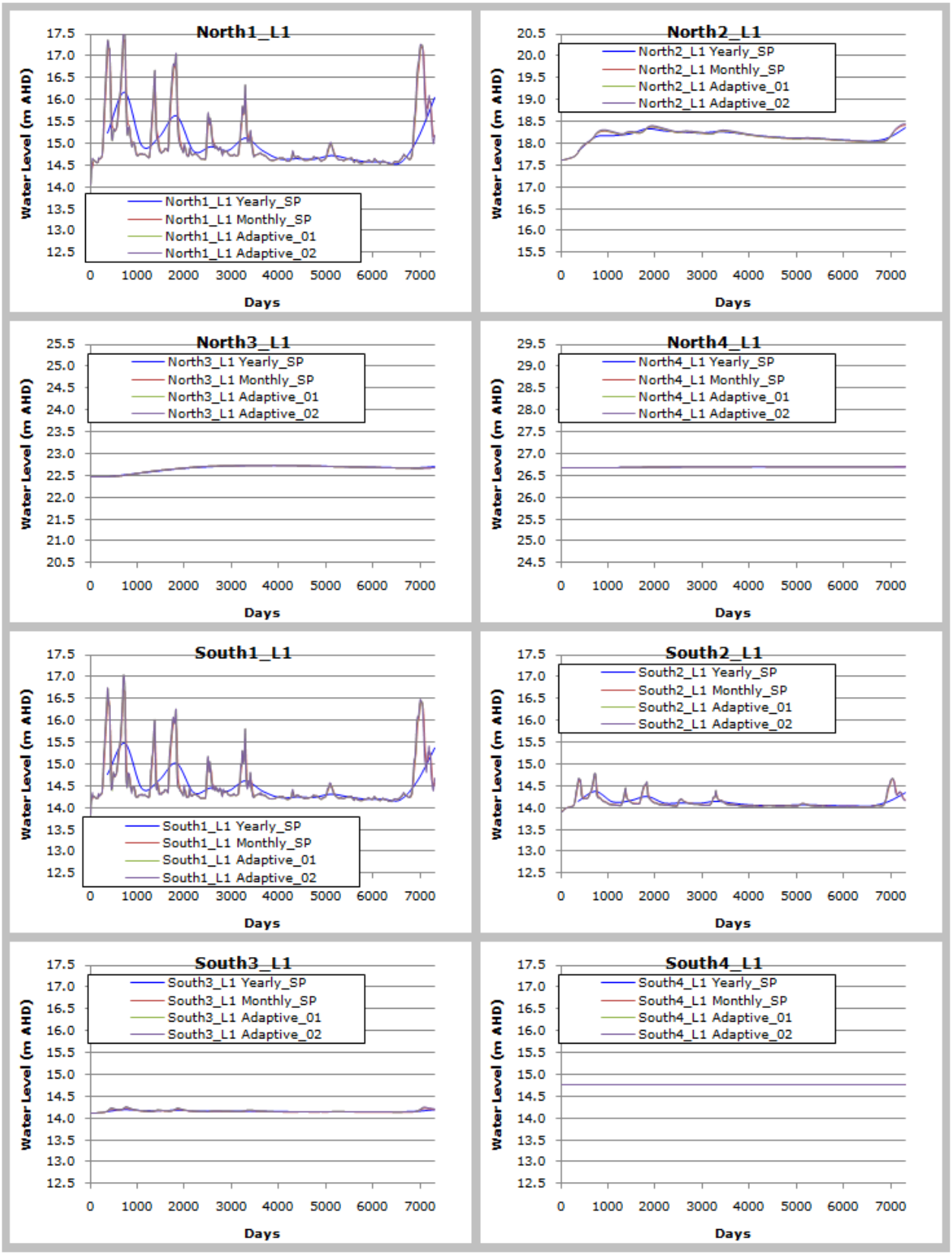

Water balance Scenario 1 - Basecase C - Locked - 1996
Flux (KL/d)
80000
60000
40000
20000
0
-20000
-40000
-60000
-80000
storage in
ch in
riv in
et in
ghb in
rch in
storage out
ch out
riv out
et out
ghb out
rch out
total in
total out
Yearly SP
Monthly SP
Adaptive 1
Adaptive 2


Cumulative Salt Load to River - Scenario 1 - Basecase C - Locked
Salt load (tonnes)
900,000
800,000
700,000
600,000
500,000
400,000
300,000
200,000
100,000
0
Yearly SP
Monthly SP
Adaptive 1
Adaptive 2
1/01/1992
31/12/1996
31/12/2001
31/12/2006
31/12/2011
Date

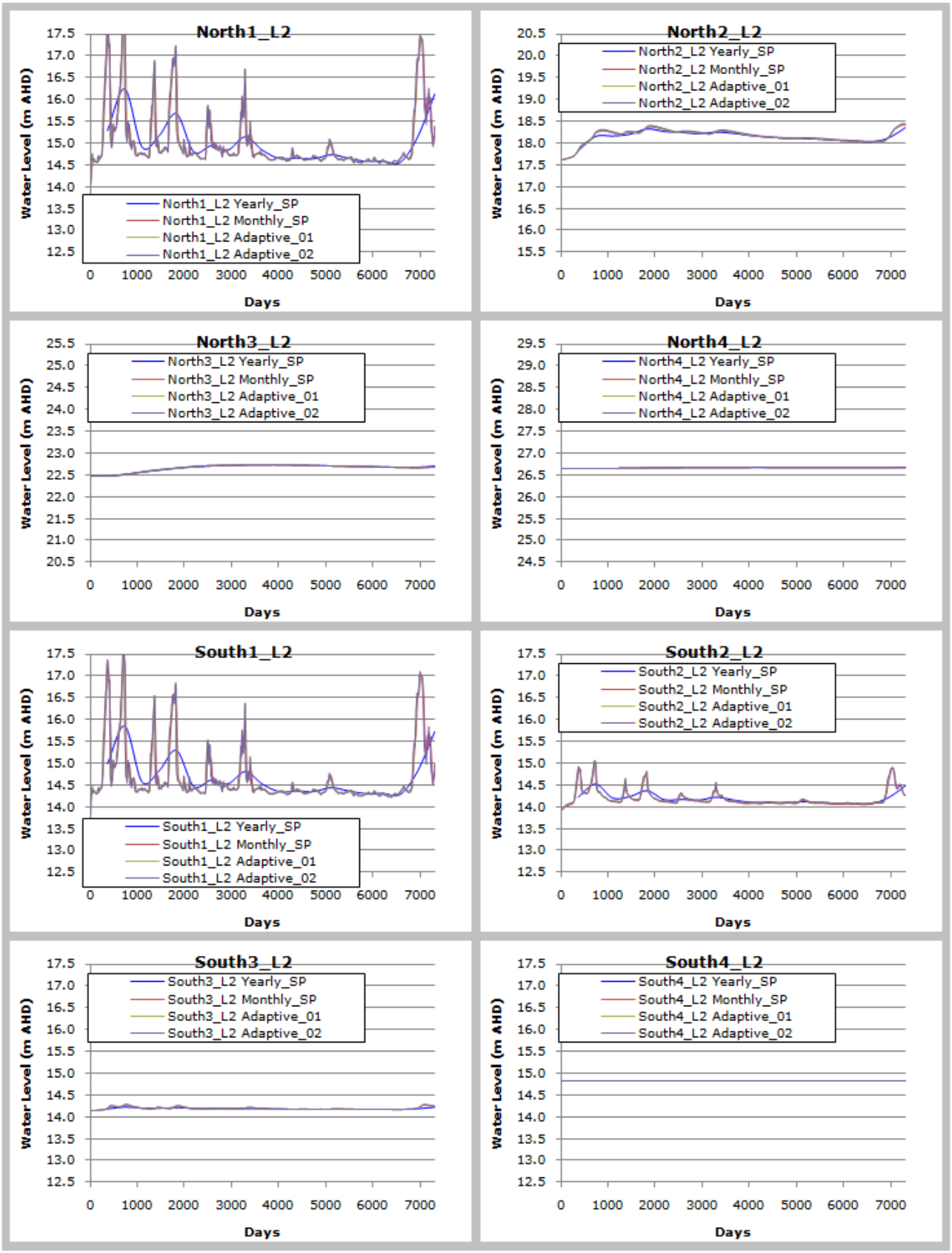

Net River Condition - Scenario 1 - Basecase C - Locked
Net Flux (kL/d)
60,000
50,000
40,000
30,000
20,000
10,000
0
-10,000
-20,000
-30,000
-40,000
-50,000
-60,000
Yearly SP
Monthly SP
Adaptive 1
Adaptive 2
Gaining River
Losing River
1/01/1992
31/12/1996
31/12/2001
31/12/2006
31/12/2011
Date

Appendix C7: Model results for Scenario 1 – Model C – Not locked

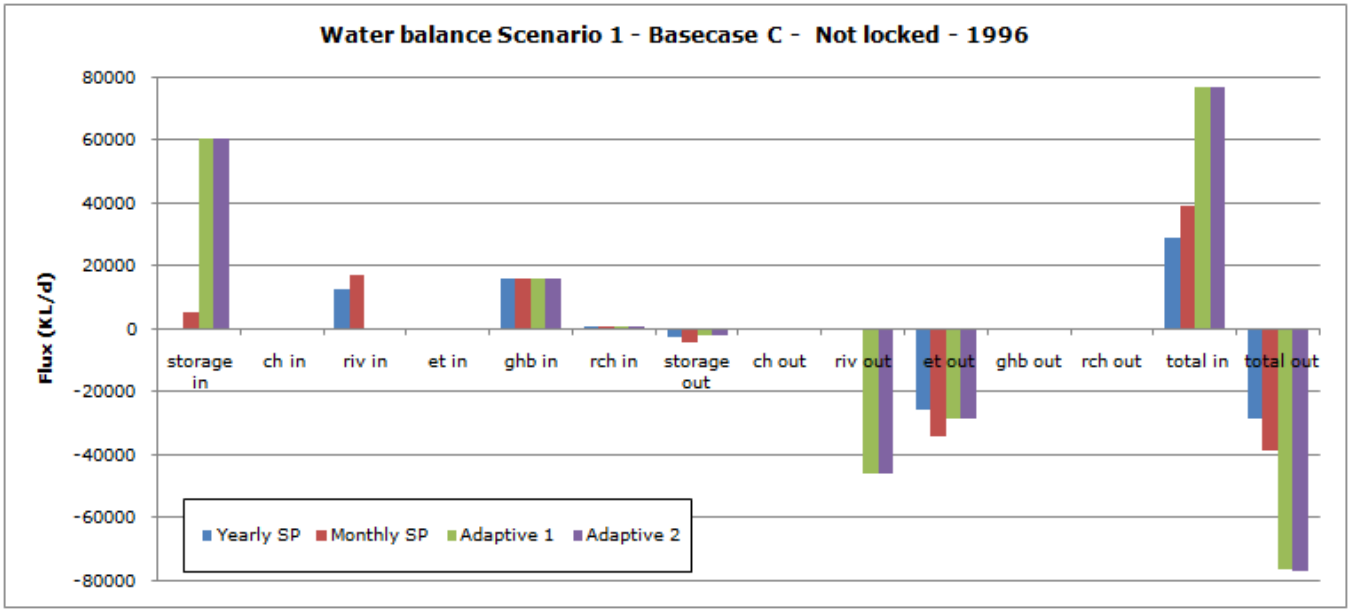
The following figures present the transient model behaviour for the scenario in the title. The lines represent each temporal discretisation approach with yearly, monthly, and adaptive variants. The adaptive variants utilize a stress period determined via the Python scripts included in Appendix A.

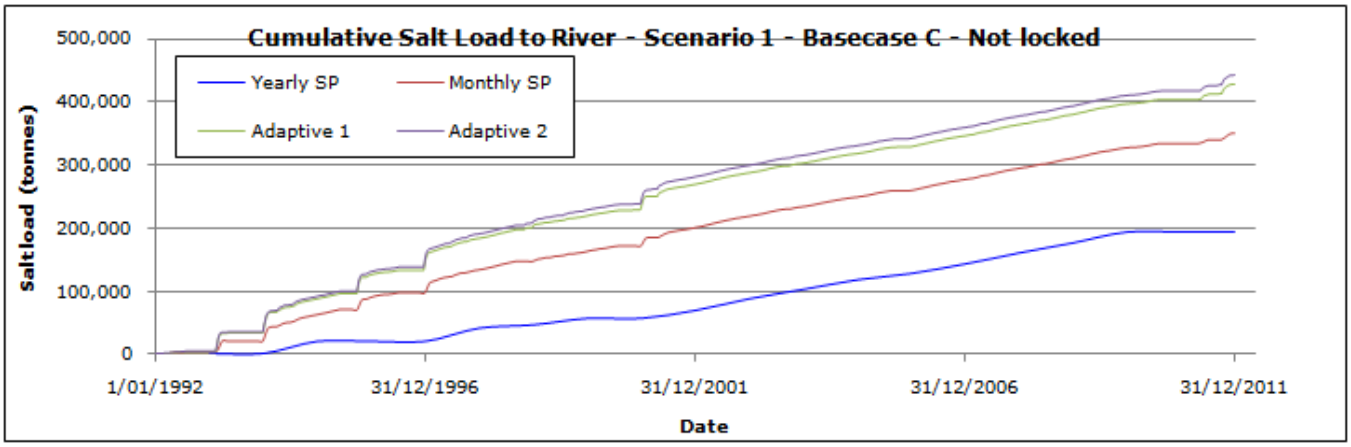


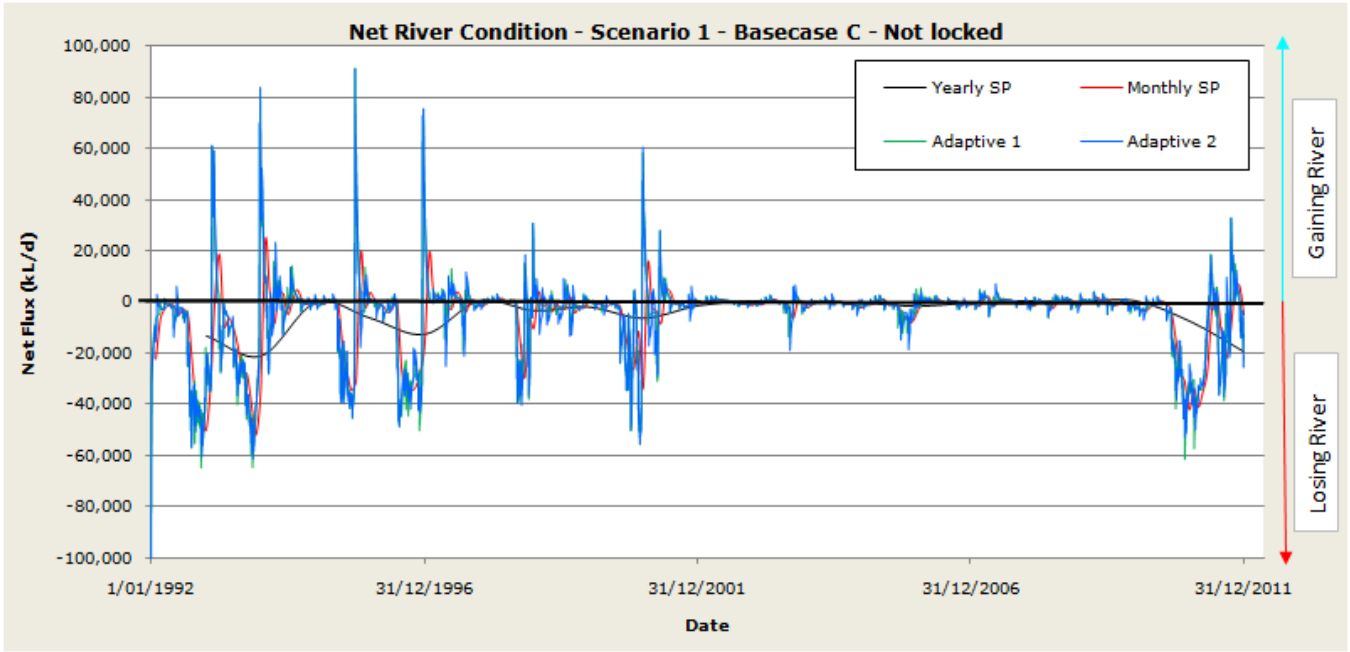
SCENARIO 1 - GOYDER MODEL C - NOT LOCKED



SCENARIO 1 - GOYDER MODEL C - NOT LOCKED


Water balance Scenario 1 - Basecase C - Not locked - 1996
Flux (KL/d)
80000
60000
40000
20000
0
-20000
-40000
-60000
-80000
storage in
ch in
riv in
et in
ghb in
rch in
storage out
ch out
riv out
et out
ghb out
rch out
total in
total out
Yearly SP
Monthly SP
Adaptive 1
Adaptive 2


Cumulative Salt Load to River - Scenario 1 - Basecase C - Not locked
Salt load (tonnes)
500,000
400,000
300,000
200,000
100,000
0
Yearly SP
Monthly SP
Adaptive 1
Adaptive 2
1/01/1992
31/12/1996
31/12/2001
31/12/2006
31/12/2011
Date

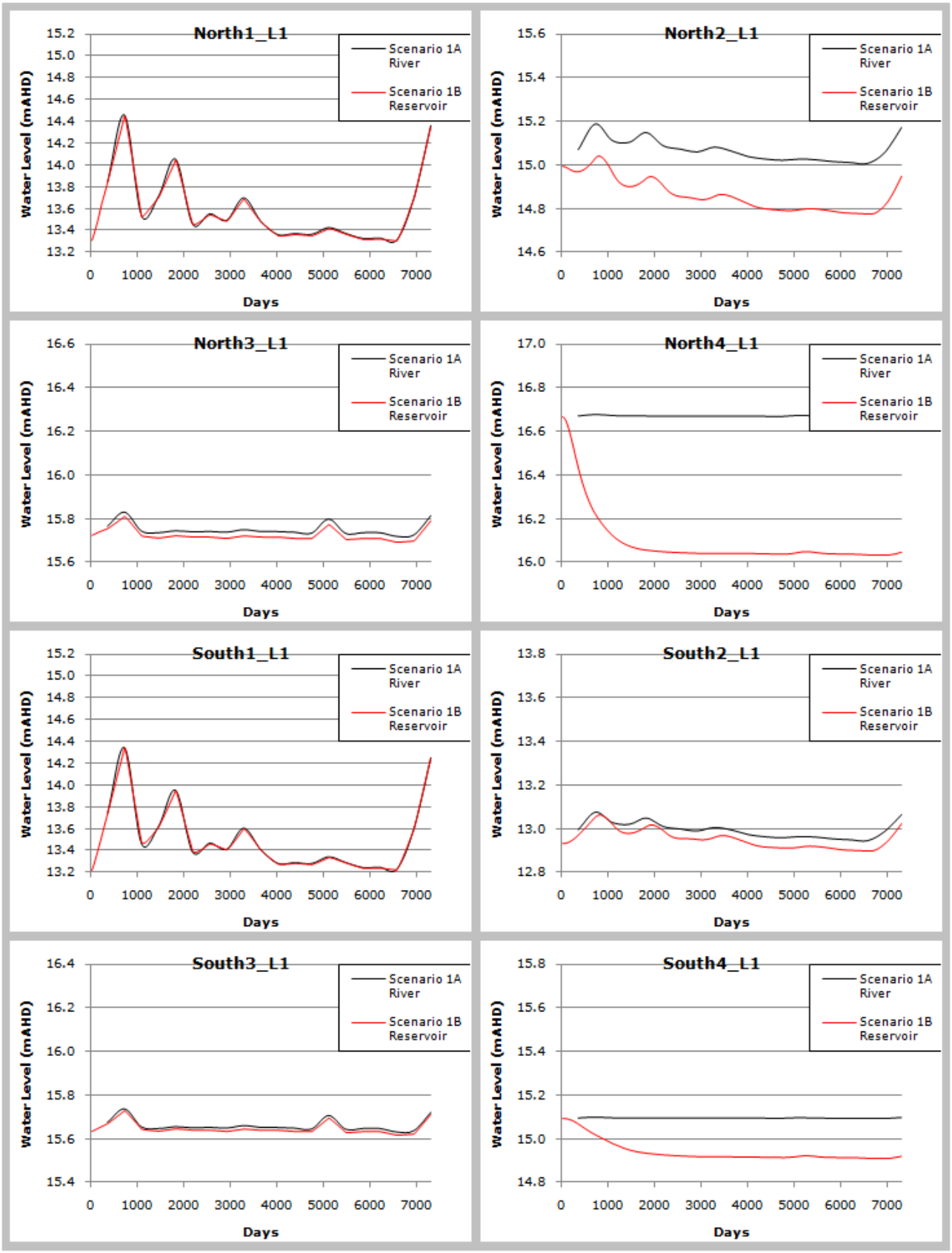

Net River Condition - Scenario 1 - Basecase C - Not locked
Net Flux (kL/d)
100,000
80,000
60,000
40,000
20,000
0
-20,000
-40,000
-60,000
-80,000
-100,000
Yearly SP
Monthly SP
Adaptive 1
Adaptive 2
Gaining River
Losing River
1/01/1992
31/12/1996
31/12/2001
31/12/2006
31/12/2011
Date

Appendix C8: Model results for Scenario 1 – Model A – Reservoir

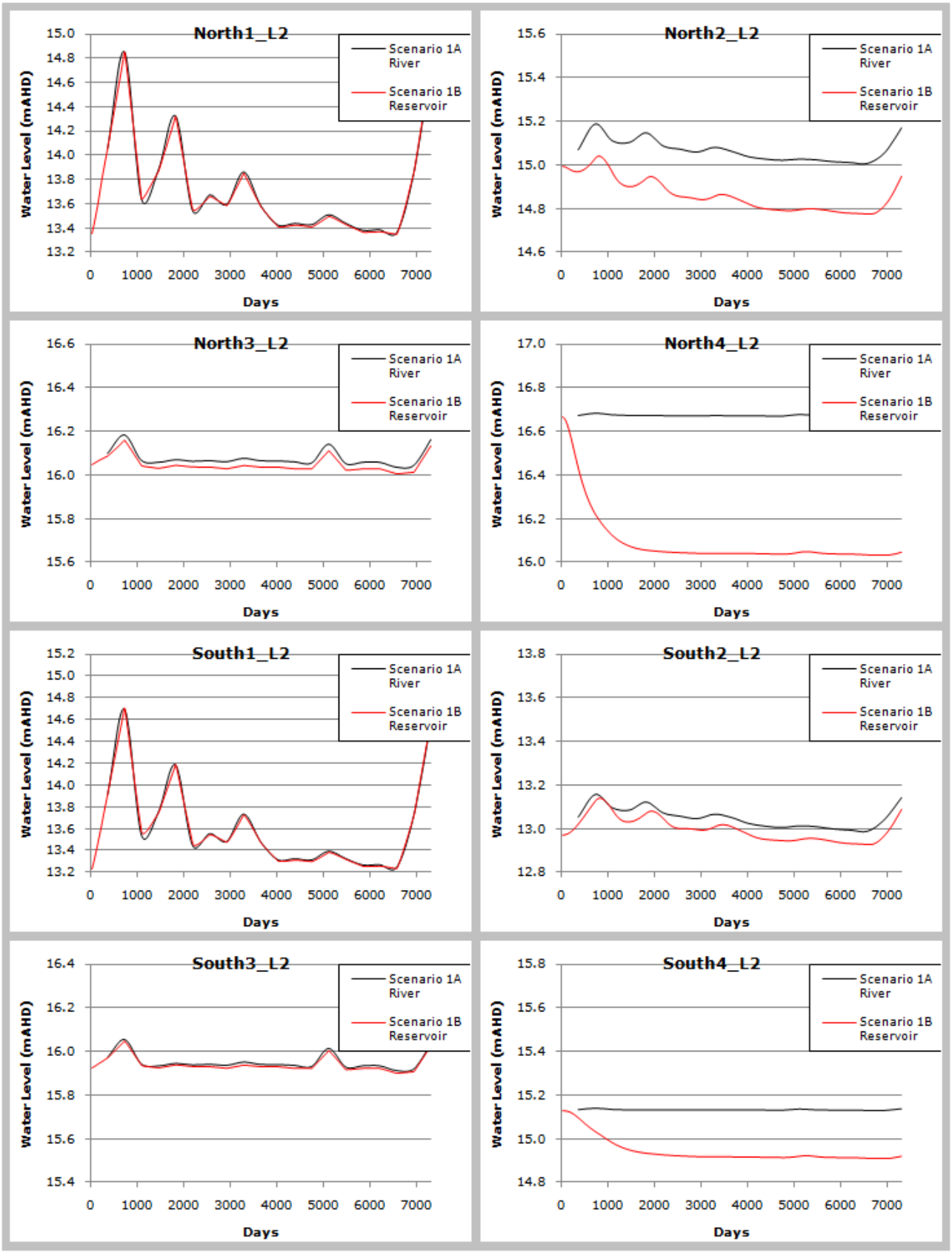
The following figures present the transient model behaviour for the scenario in the title. The lines represent each temporal discretisation approach with yearly, monthly, and adaptive variants. The adaptive variants utilize a stress period determined via the Python scripts included in Appendix A.

The river can be simulated in MODFLOW with either the River package or the Reservoir package. The River package has a set stage for every stress period while the reservoir package will interpolate valuses for stage elevation btewwn stress periods updating at every time step. The graphs depict the effect in the floodplain when using the different packages with different stress period setups.

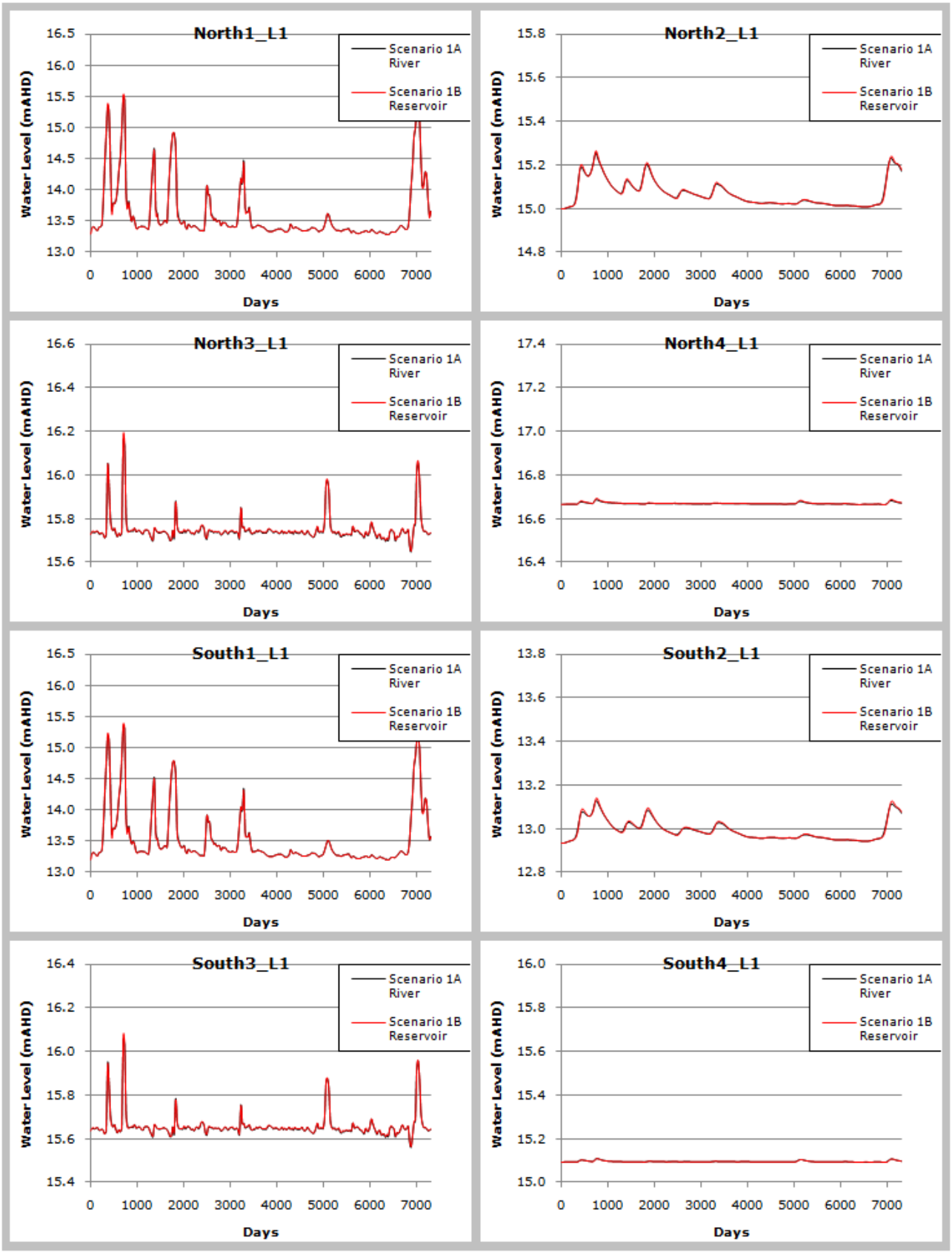
Scenario S1A refers to the river package simualtions
Scenario S1B refers to the reservoir package simulations.



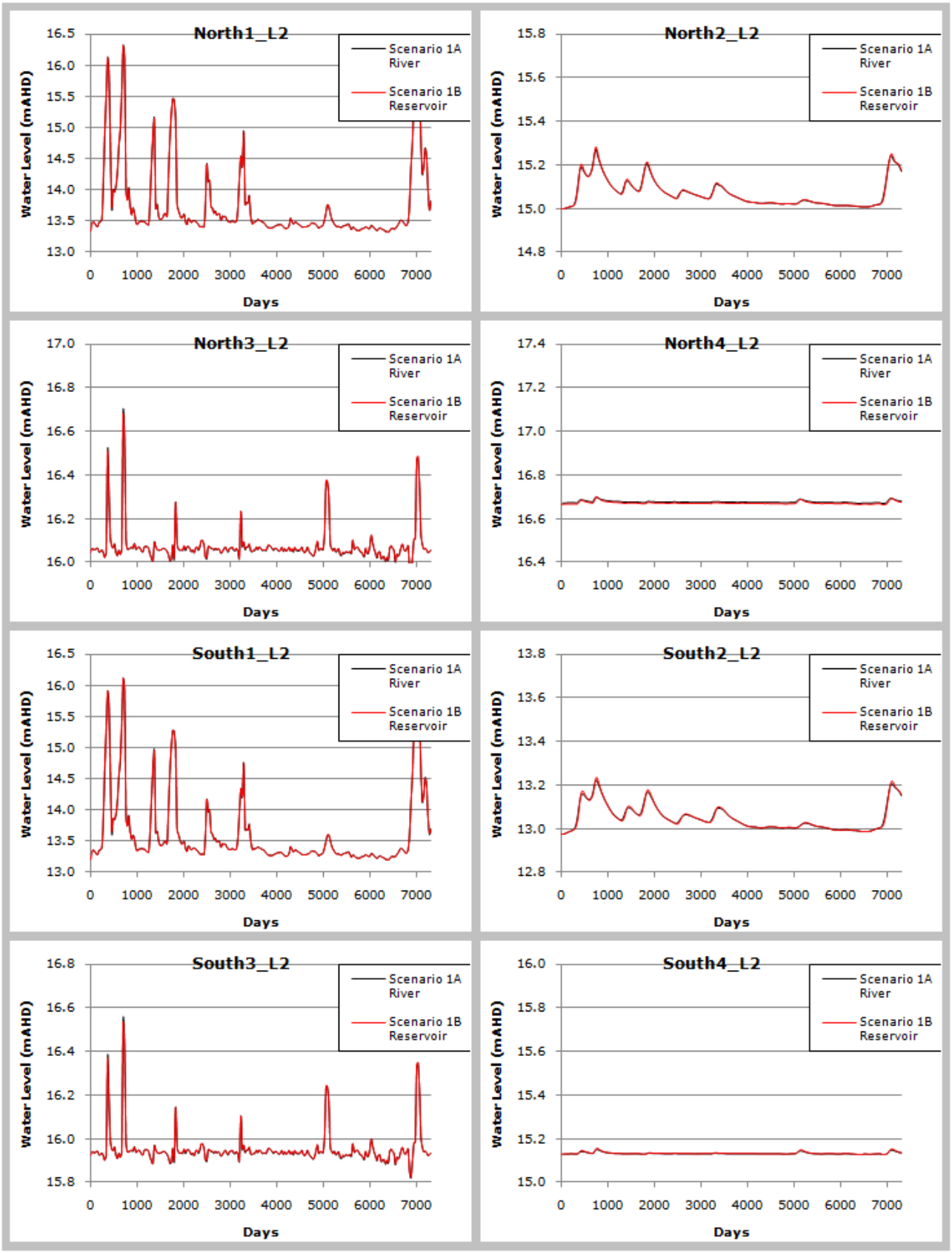
SCENARIO 1 - GOYDER MODEL A - YEARLY SP RIVER VS RESERVOIR



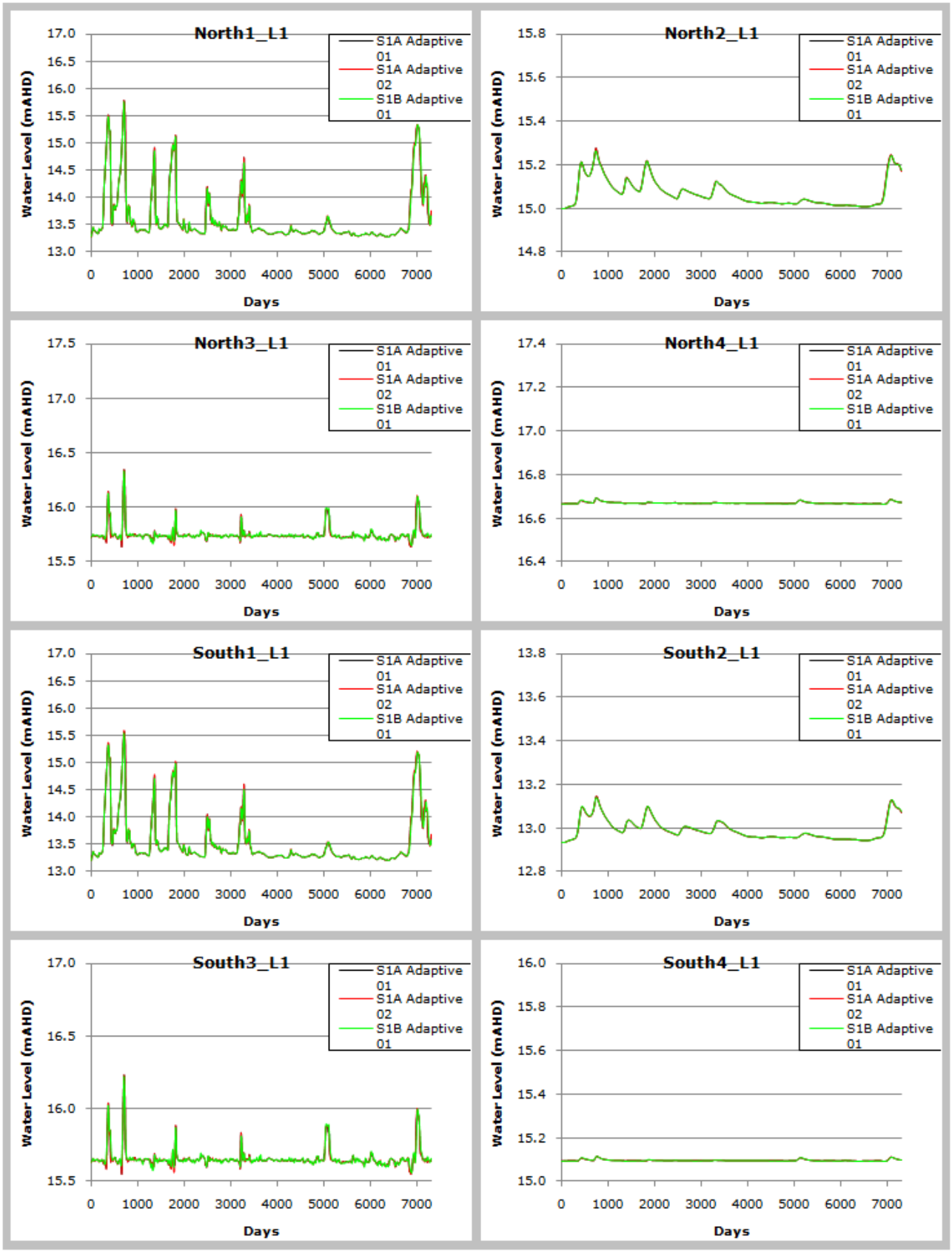
SCENARIO 1 - GOYDER MODEL A - YEARLY SP RIVER VS RESERVOIR



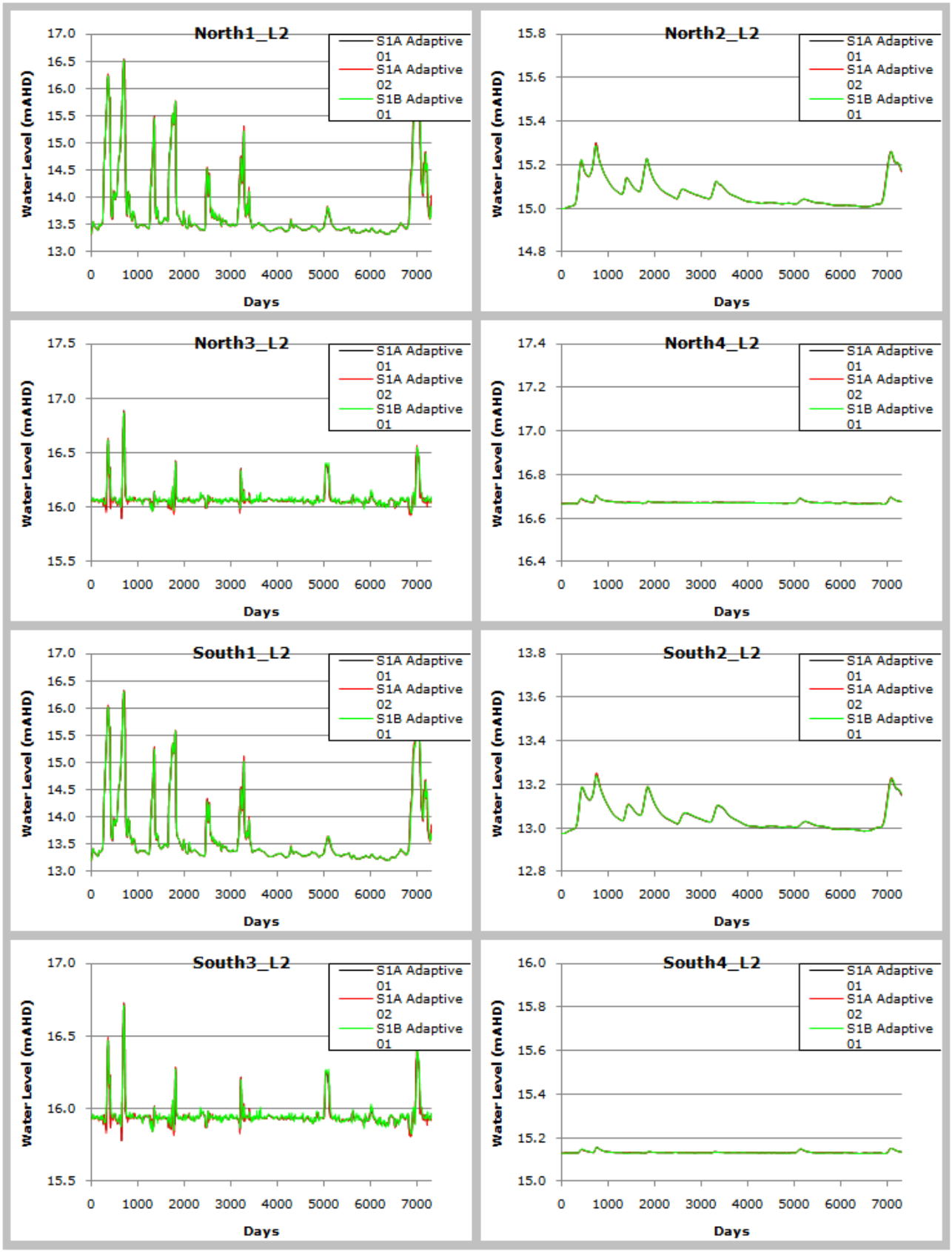
SCENARIO 1 - GOYDER MODEL A MONTHLY SP RIVER VS RESERVOIR



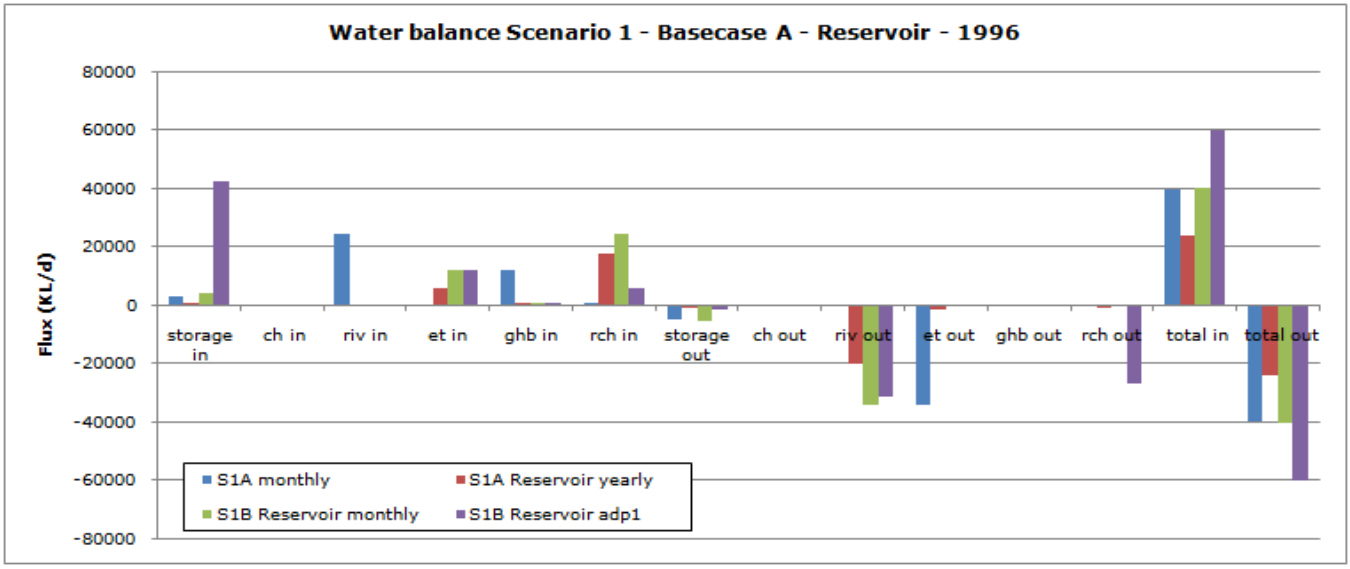
SCENARIO 1 - GOYDER MODEL A MONTHLY SP RIVER VS RESERVOIR

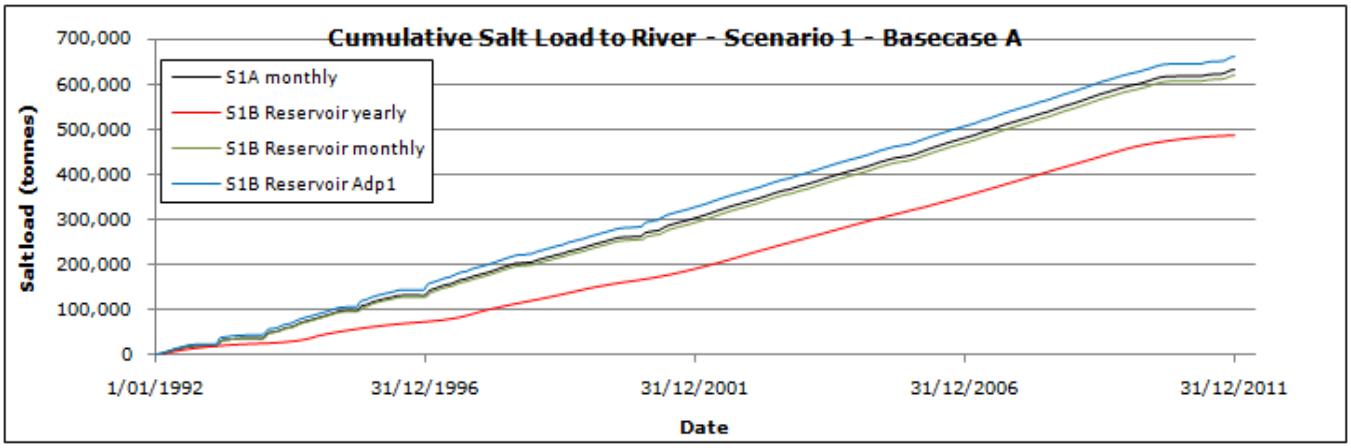


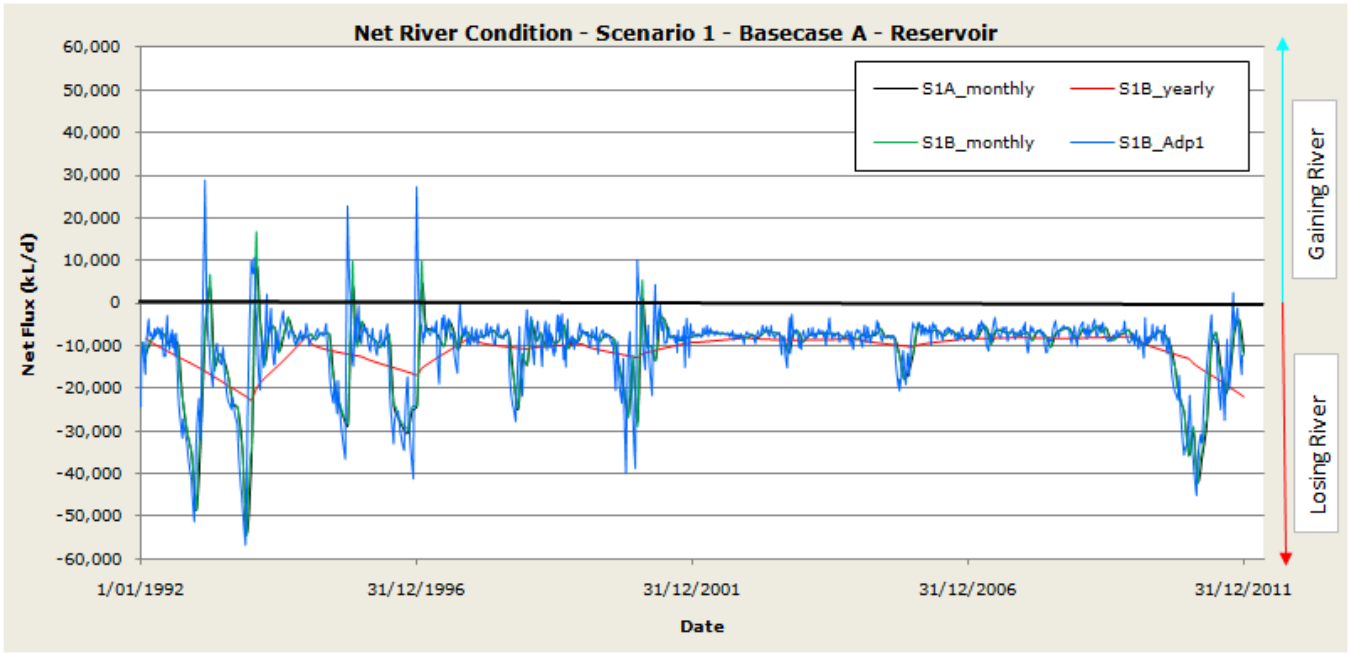
SCENARIO 1 - GOYDER MODEL A - ADAPTIVE SP RIVER VS RESERVOIR



SCENARIO 1 - GOYDER MODEL A - ADAPTIVE SP RIVER VS RESERVOIR


Water balance Scenario 1 - Basecase A - Reservoir - 1996
Flux (KL/d)
80000
60000
40000
20000
0
-20000
-40000
-60000
-80000
storage in
ch in
riv in
et in
ghb in
rch in
storage out
ch out
riv out
et out
ghb out
rch out
total in
total out
S1A monthly
S1A Reservoir yearly
S1B Reservoir monthly
S1B Reservoir adp1


Cumulative Salt Load to River - Scenario 1 - Basecase A
Saltload (tonnes)
700,000
600,000
500,000
400,000
300,000
200,000
100,000
0
S1A monthly
S1B Reservoir yearly
S1B Reservoir monthly
S1B Reservoir Adp1
1/01/1992
31/12/1996
31/12/2001
31/12/2006
31/12/2011
Date

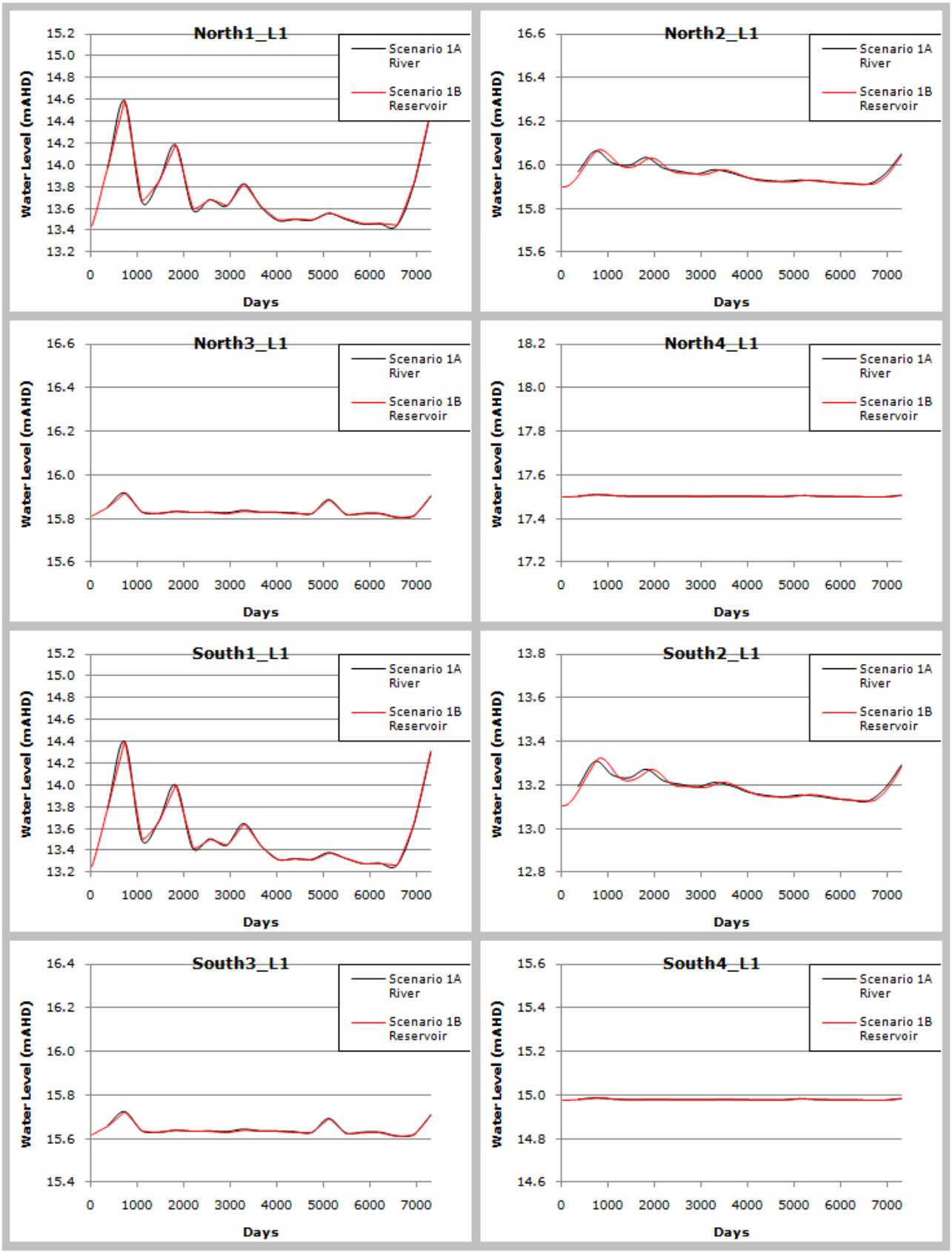

Net River Condition - Scenario 1 - Basecase A - Reservoir
Net Flux (kL/d)
60,000
50,000
40,000
30,000
20,000
10,000
0
-10,000
-20,000
-30,000
-40,000
-50,000
-60,000
S1A_monthly
S1B_yearly
S1B_monthly
S1B_Adp1
Gaining River
Losing River
1/01/1992
31/12/1996
31/12/2001
31/12/2006
31/12/2011
Date

Appendix C9: Model results for Scenario 1 – Model B – Reservoir

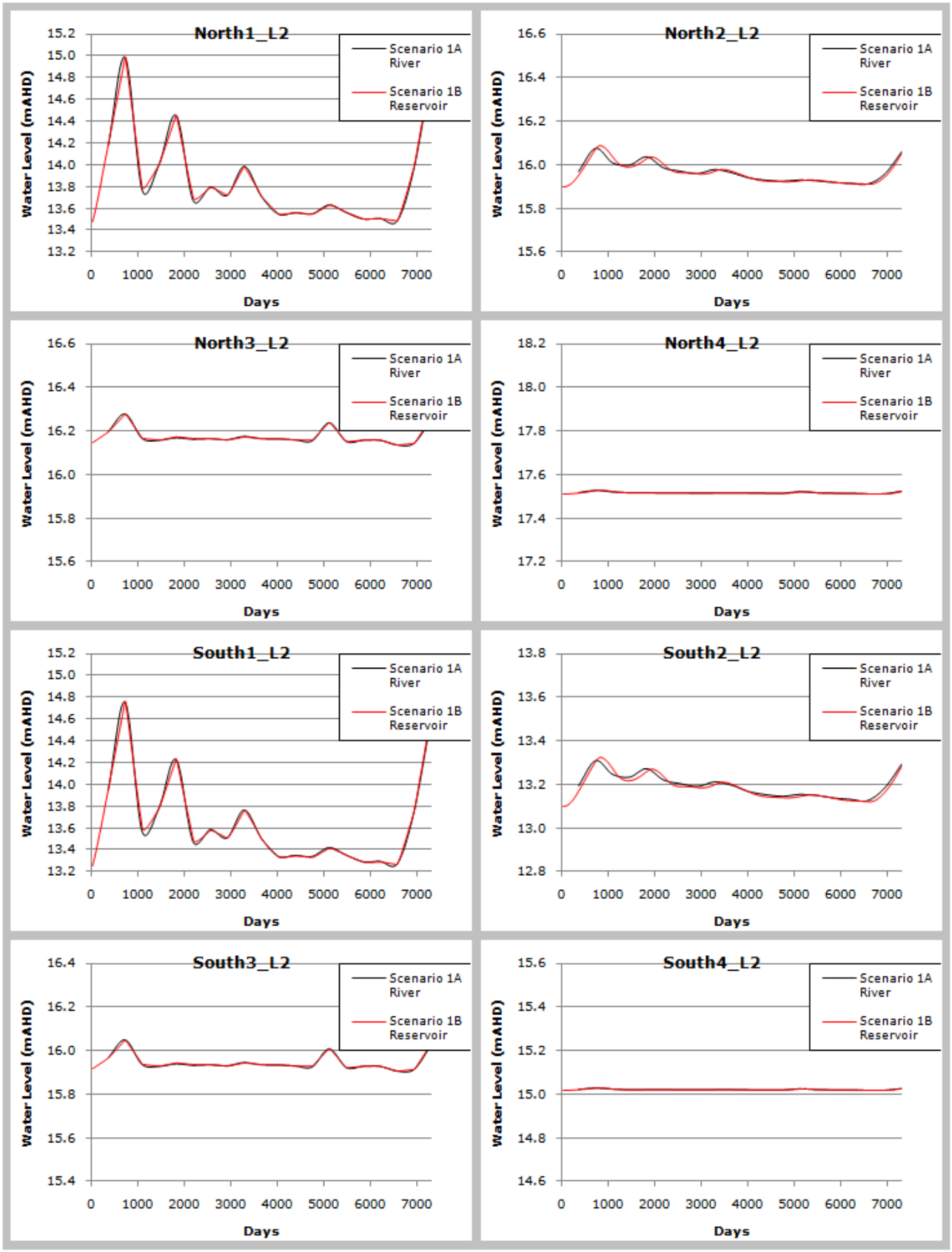
The following figures present the transient model behaviour for the scenario in the title. The lines represent each temporal discretisation approach with yearly, monthly, and adaptive variants. The adaptive variants utilize a stress period determined via the Python scripts included in Appendix A.

The river can be simulated in MODFLOW with either the River package or the Reservoir package. The River package has a set stage for every stress period while the reservoir package will interpolate valuses for stage elevation btewwn stress periods updating at every time step. The graphs depict the effect in the floodplain when using the different packages with different stress period setups.

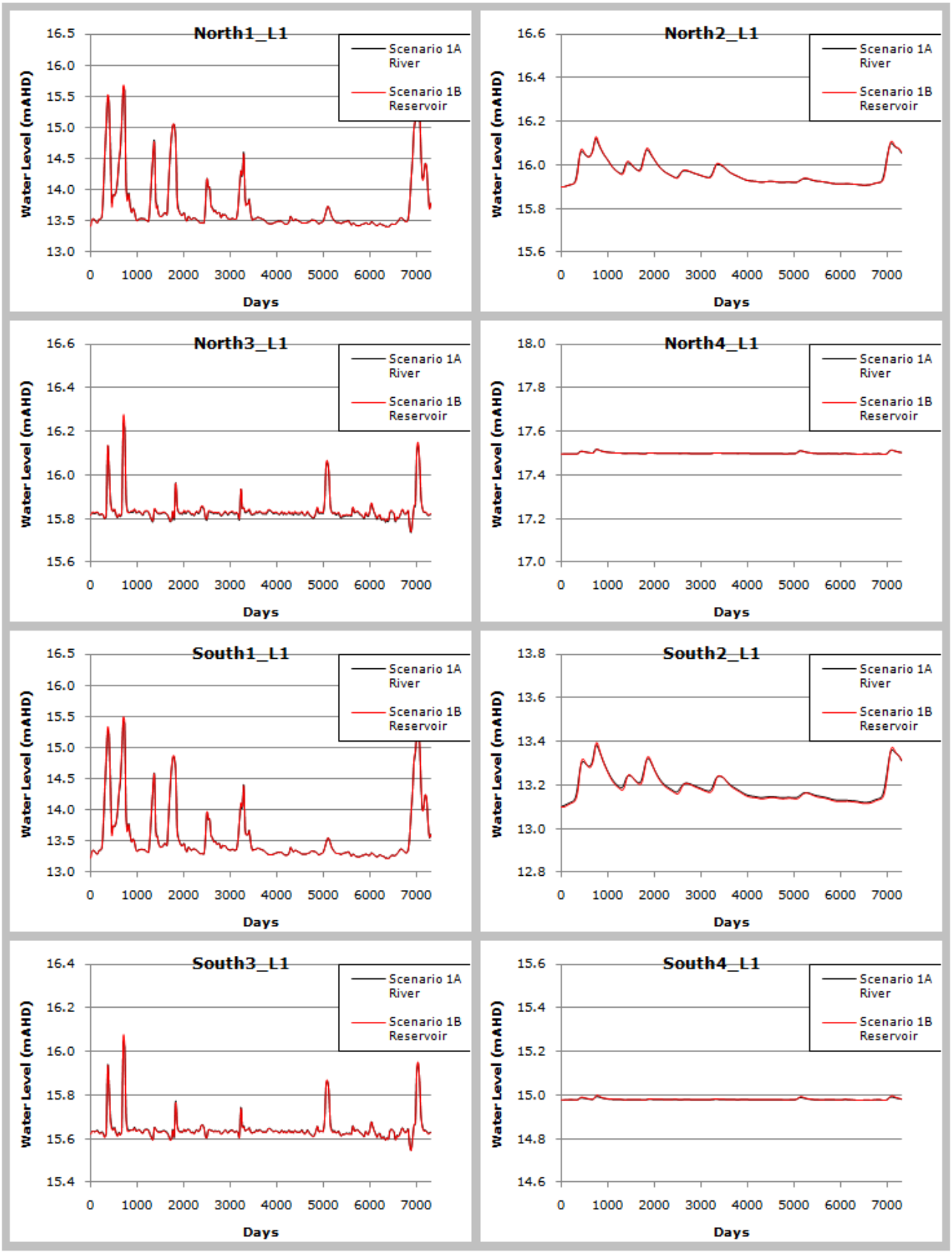
Scenario S1A refers to the river package simualtions
Scenario S1B refers to the reservoir package simulations.



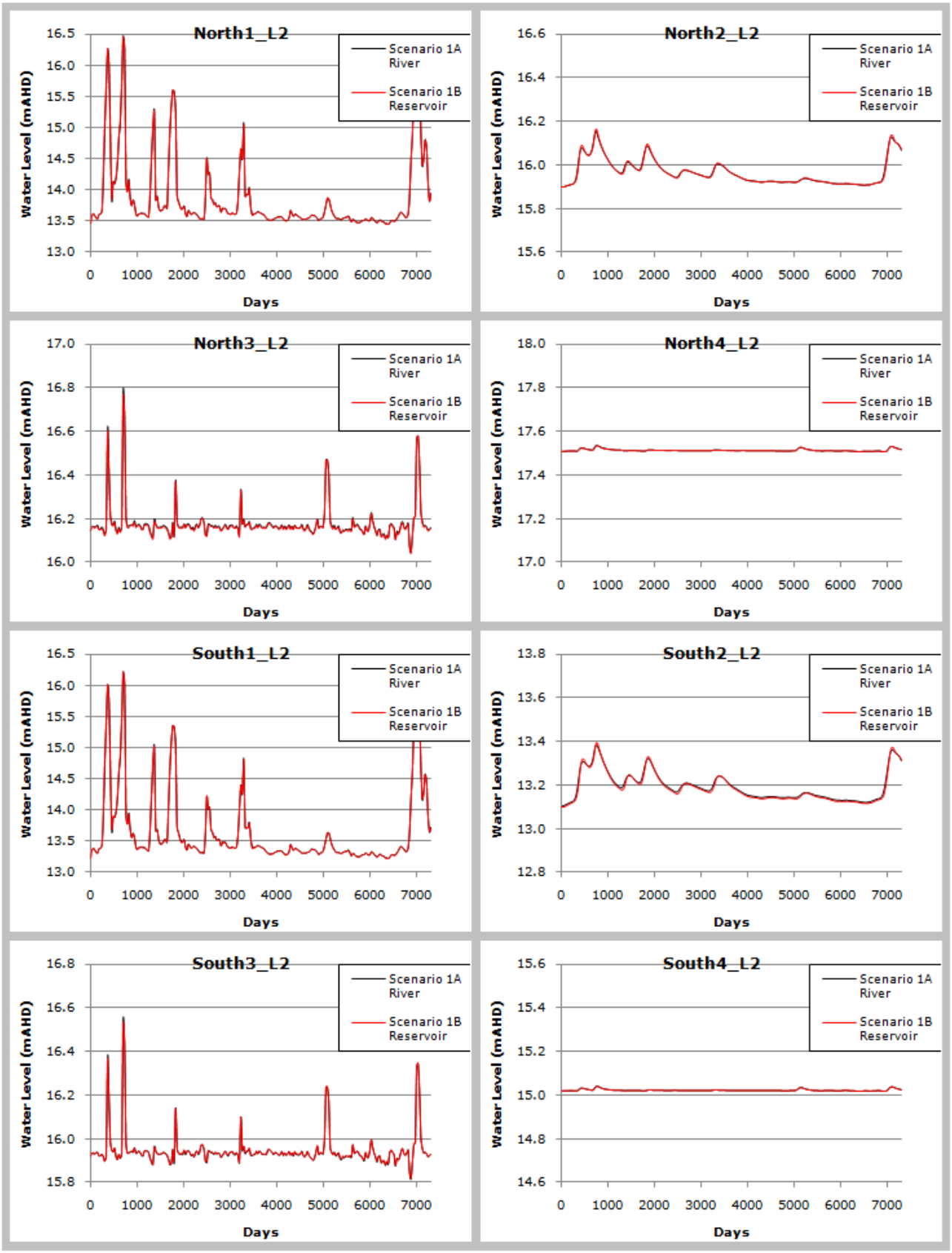
SCENARIO 1 - GOYDER MODEL B - YEARLY SP RIVER VS RESERVOIR



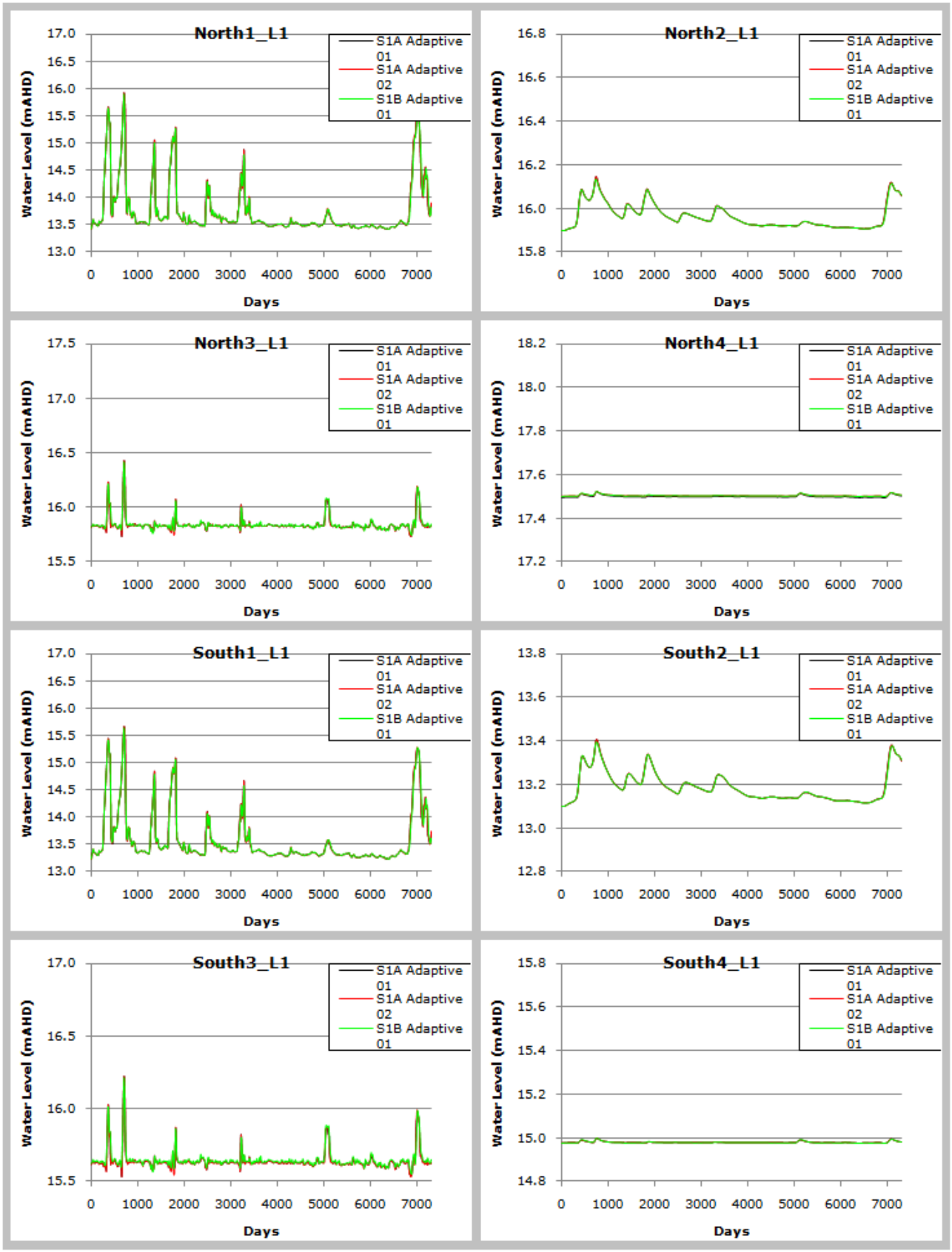
SCENARIO 1 - GOYDER MODEL B - YEARLY SP RIVER VS RESERVOIR



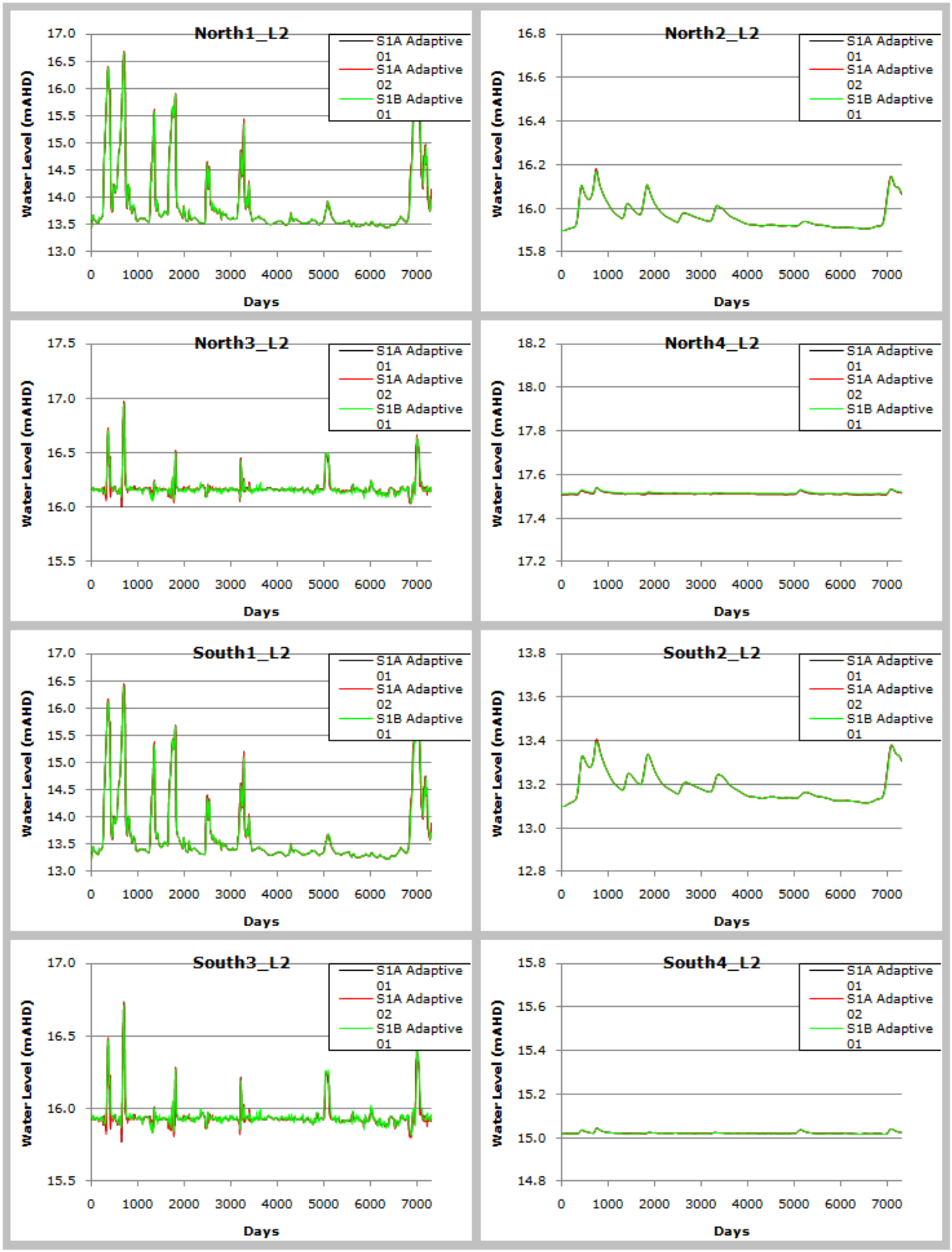
SCENARIO 1 - GOYDER MODEL A MONTHLY SP RIVER VS RESERVOIR



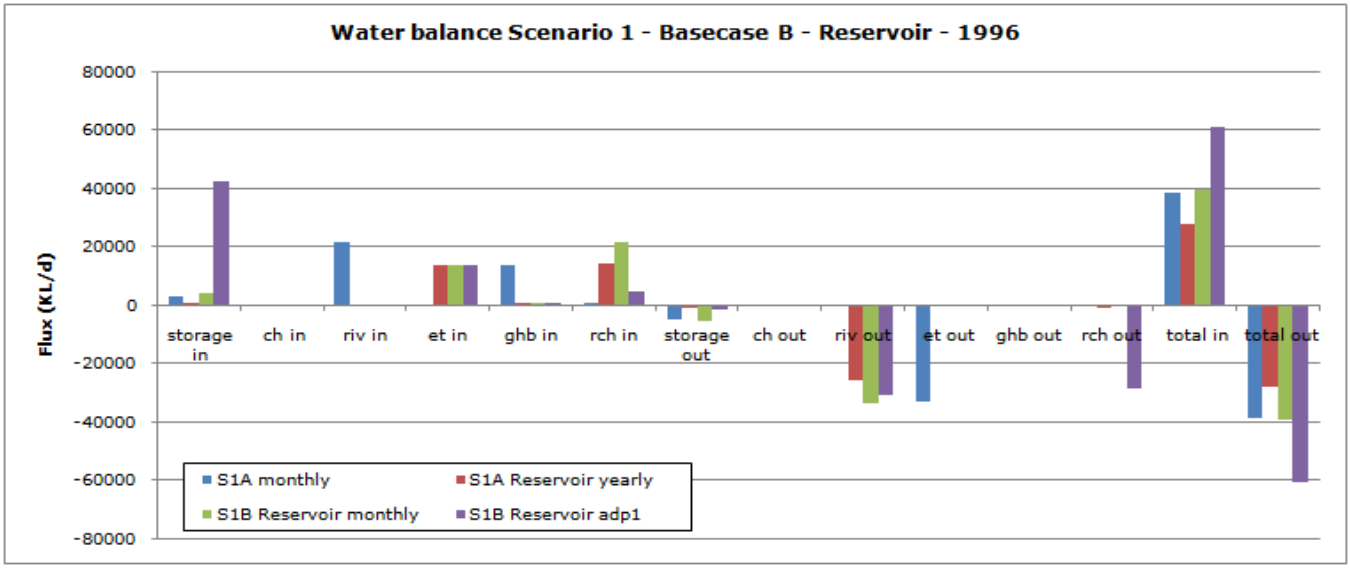
SCENARIO 1 - GOYDER MODEL A MONTHLY SP RIVER VS RESERVOIR

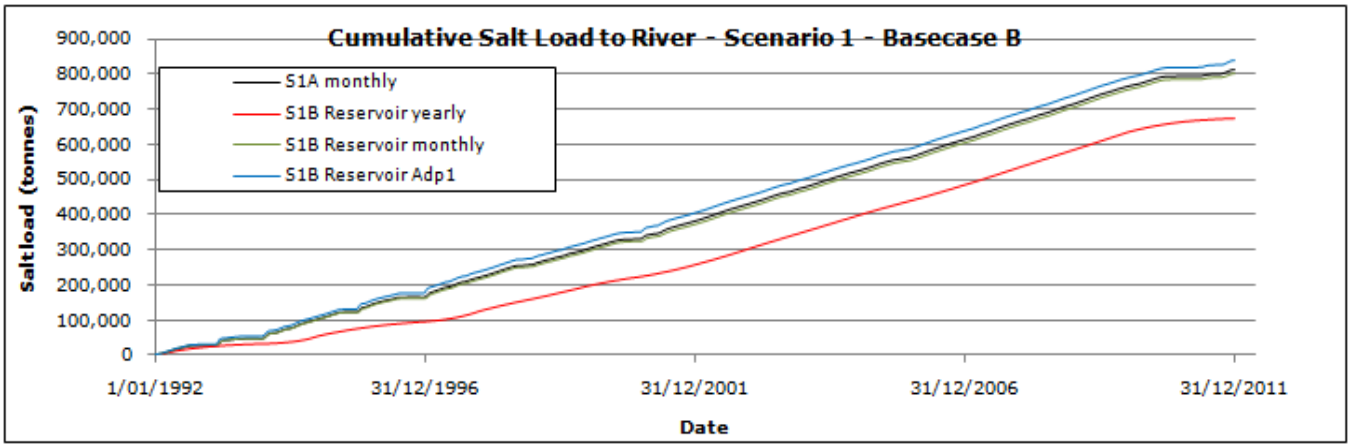


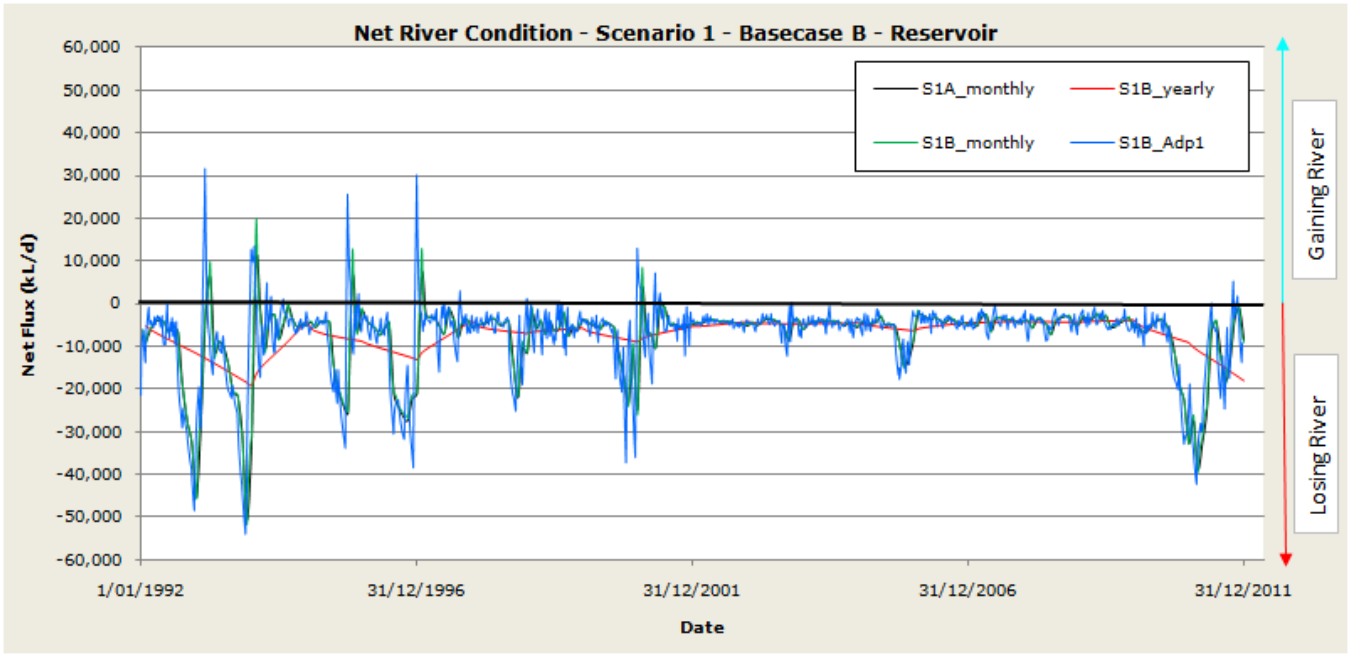
SCENARIO 1 - GOYDER MODEL B - ADAPTIVE SP RIVER VS RESERVOIR



SCENARIO 1 - GOYDER MODEL B - ADAPTIVE SP RIVER VS RESERVOIR


Water balance Scenario 1 - Basecase B - Reservoir - 1996
Flux (KL/d)
80000
60000
40000
20000
0
-20000
-40000
-60000
-80000
storage in
ch in
riv in
et in
ghb in
rch in
storage out
ch out
riv out
et out
ghb out
rch out
total in
total out
S1A monthly
S1A Reservoir yearly
S1B Reservoir monthly
S1B Reservoir adp1


Cumulative Salt Load to River - Scenario 1 - Basecase B
Salt load (tonnes)
900,000
800,000
700,000
600,000
500,000
400,000
300,000
200,000
100,000
0
S1A monthly
S1B Reservoir yearly
S1B Reservoir monthly
S1B Reservoir Adp1
1/01/1992
31/12/1996
31/12/2001
31/12/2006
31/12/2011
Date

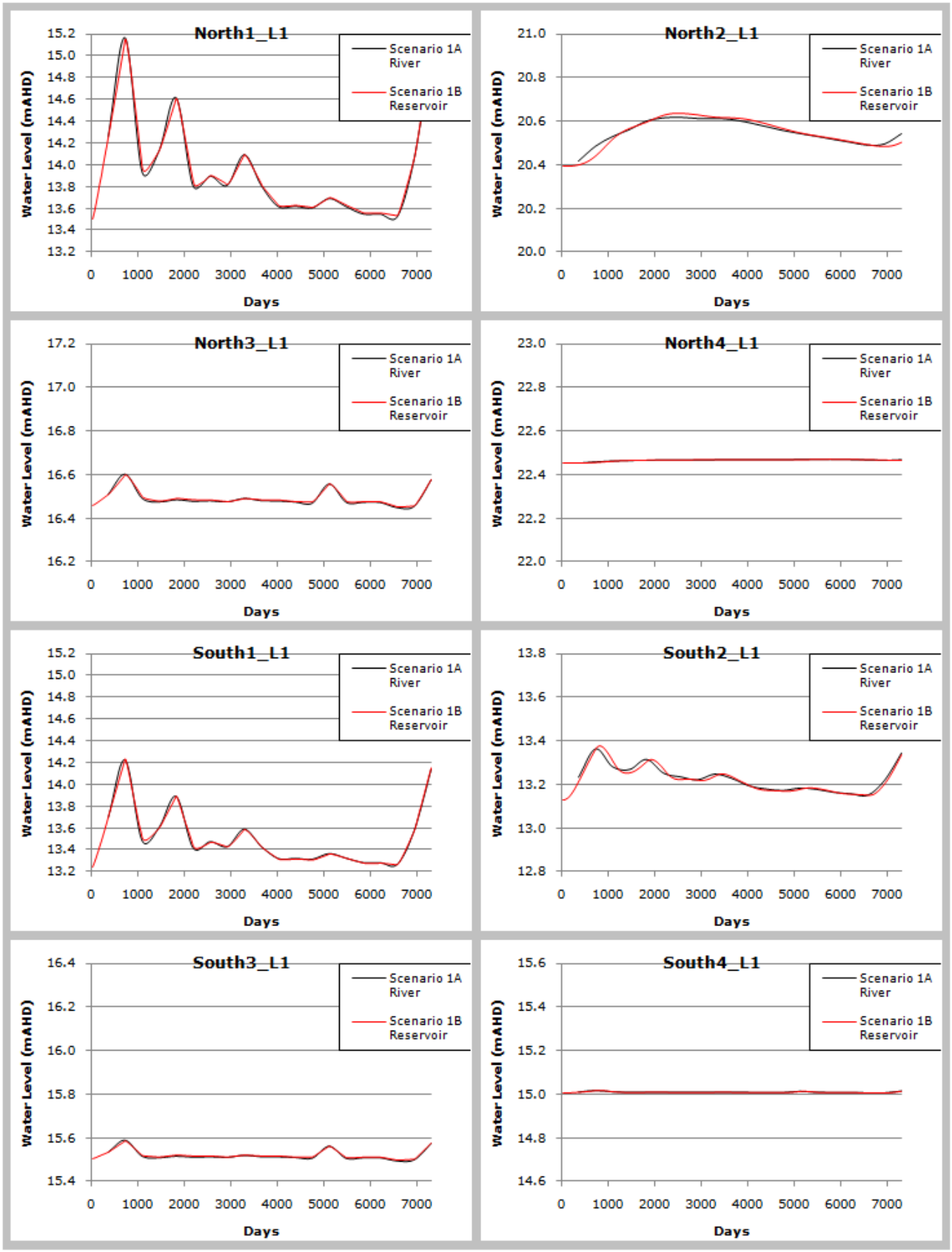

Net River Condition - Scenario 1 - Basecase B - Reservoir
Net Flux (kL/d)
60,000
50,000
40,000
30,000
20,000
10,000
0
-10,000
-20,000
-30,000
-40,000
-50,000
-60,000
S1A_monthly
S1B_yearly
S1B_monthly
S1B_Adp1
Gaining River
Losing River
1/01/1992
31/12/1996
31/12/2001
31/12/2006
31/12/2011
Date

Appendix C10: Model results for Scenario 1 – Model C – Reservoir

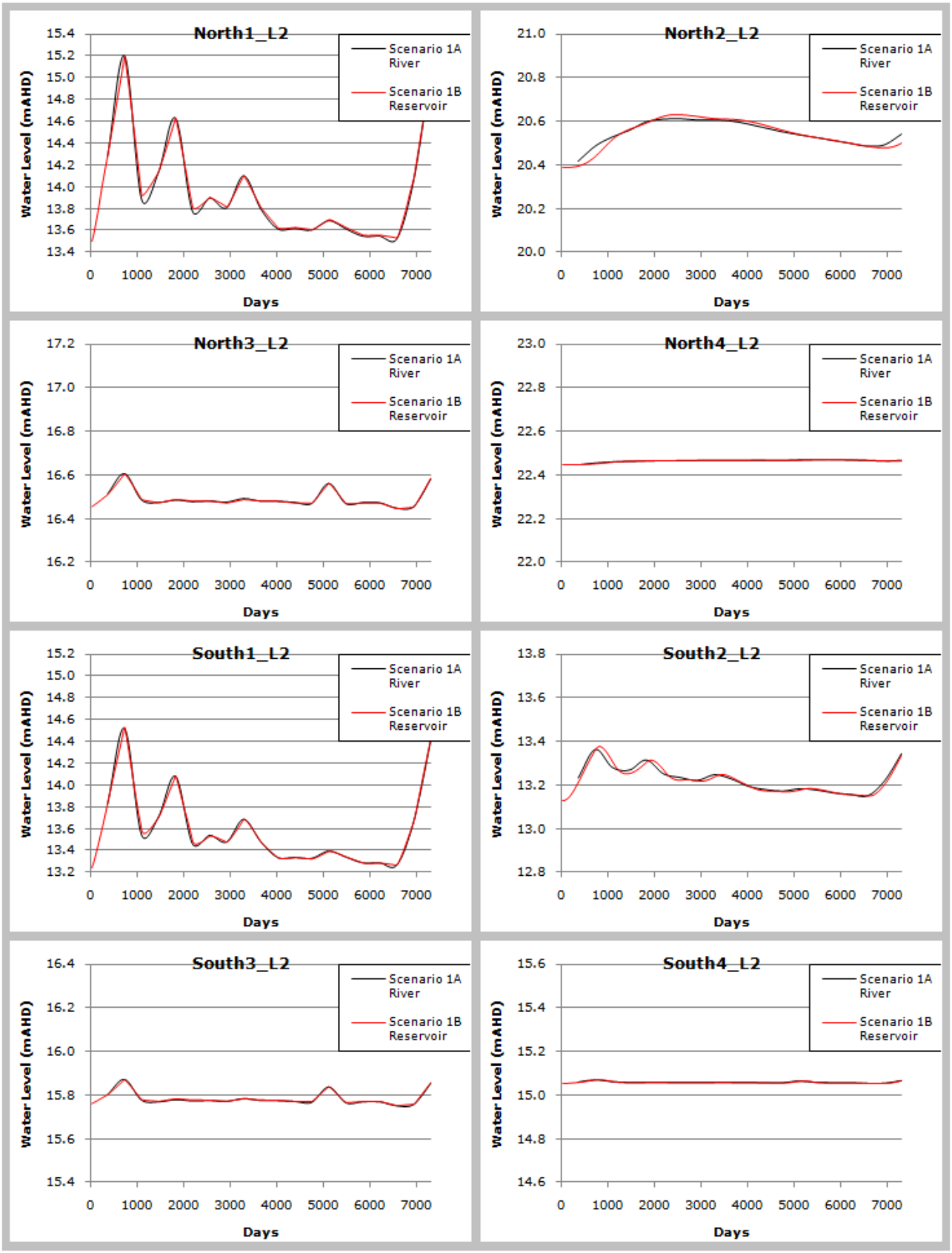
The following figures present the transient model behaviour for the scenario in the title. The lines represent each temporal discretisation approach with yearly, monthly, and adaptive variants. The adaptive variants utilize a stress period determined via the Python scripts included in Appendix A.

The river can be simulated in MODFLOW with either the River package or the Reservoir package. The River package has a set stage for every stress period while the reservoir package will interpolate valuses for stage elevation btewwn stress periods updating at every time step. The graphs depict the effect in the floodplain when using the different packages with different stress period setups.

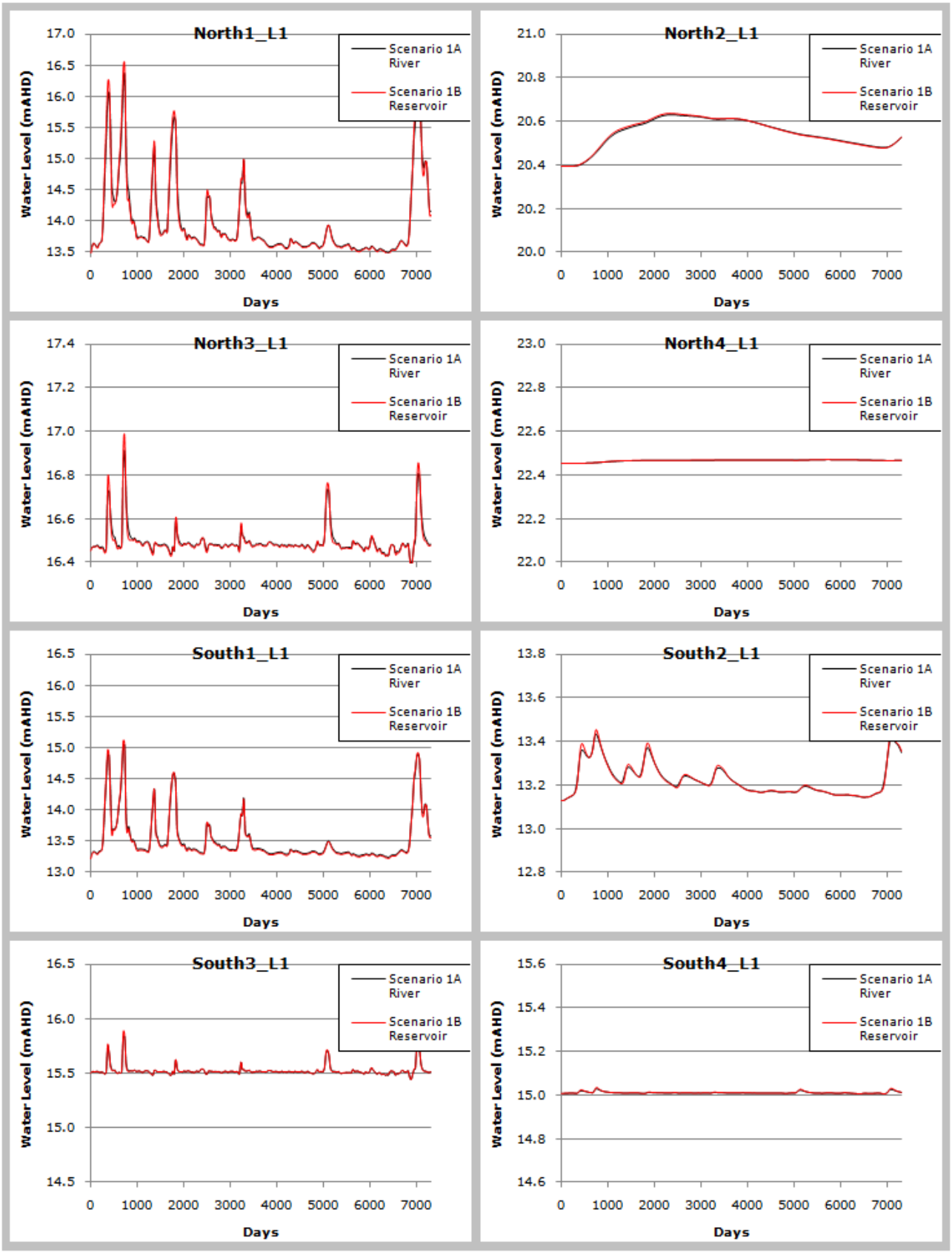
Scenario S1A refers to the river package simualtions
Scenario S1B refers to the reservoir package simulations.



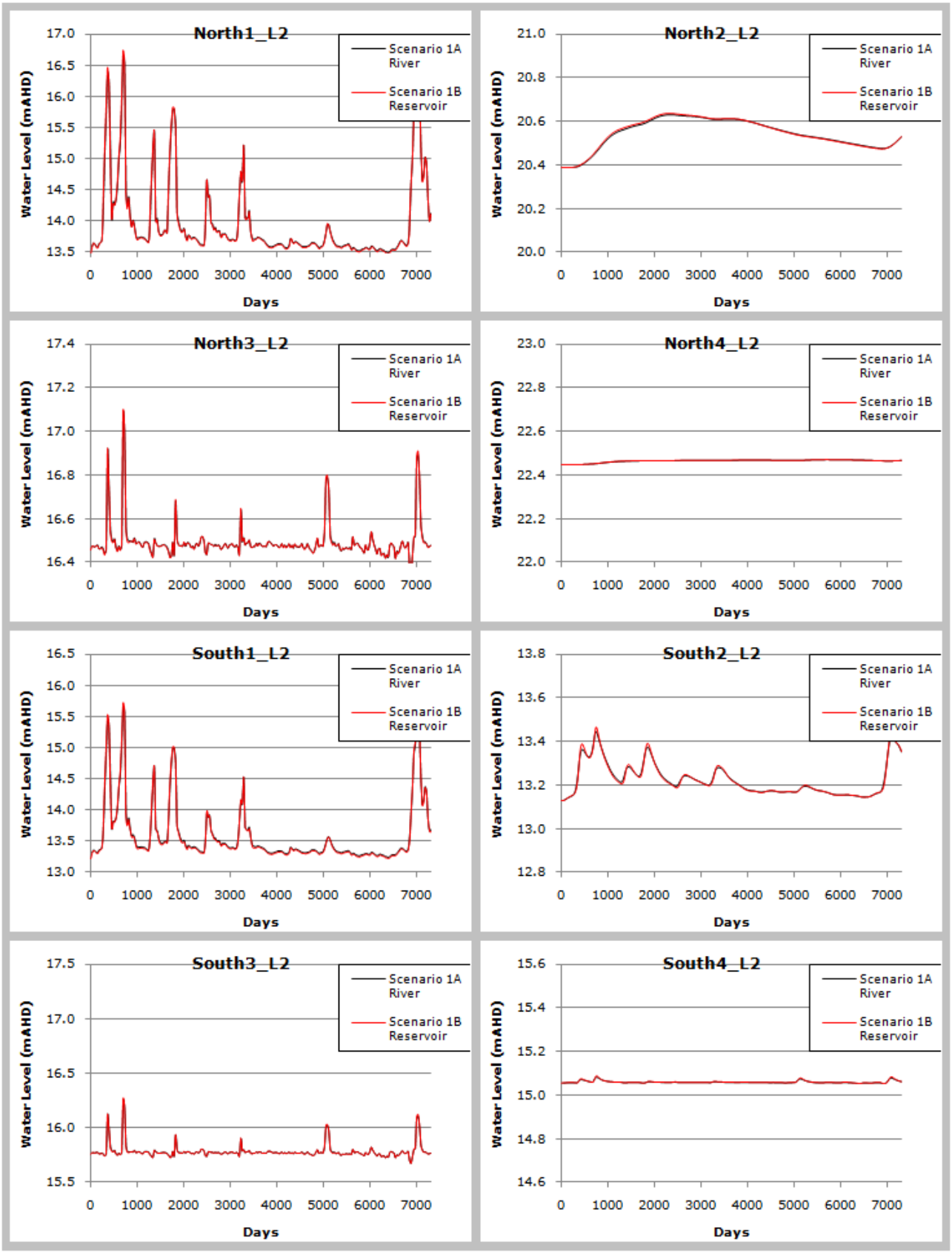
SCENARIO 1 - GOYDER MODEL C - YEARLY SP RIVER VS RESERVOIR



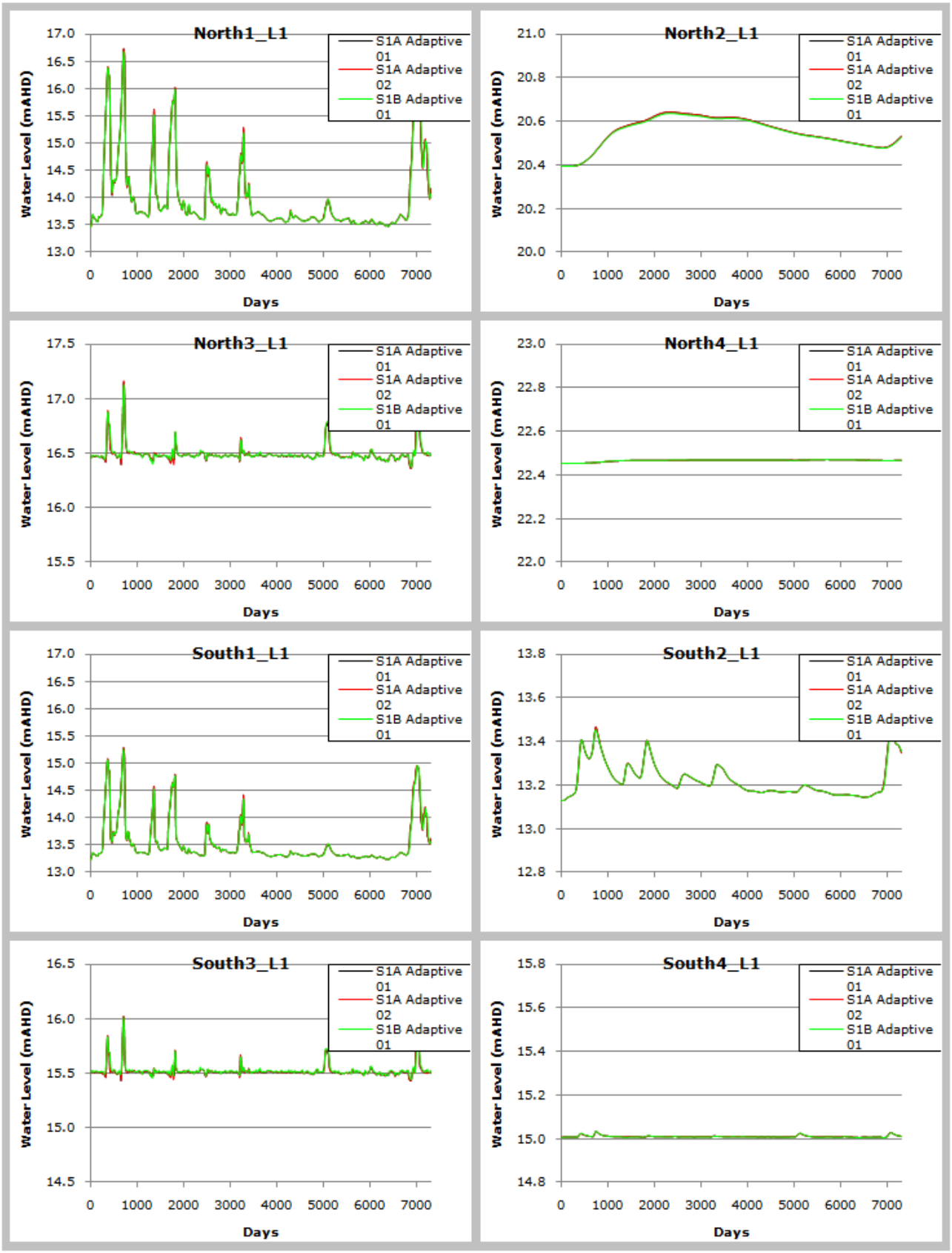
SCENARIO 1 - GOYDER MODEL C - YEARLY SP RIVER VS RESERVOIR



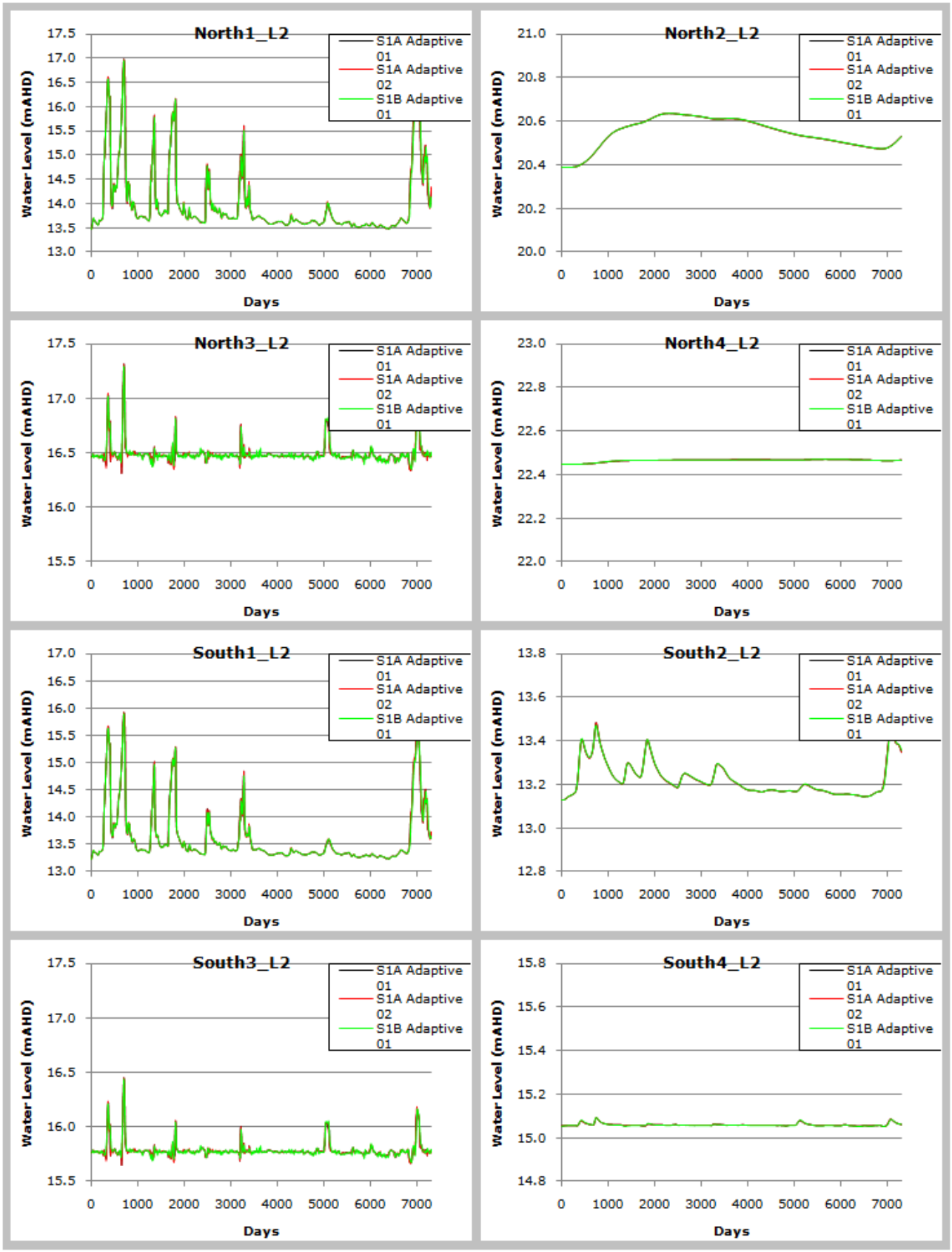
SCENARIO 1 - GOYDER MODEL C MONTHLY SP RIVER VS RESERVOIR



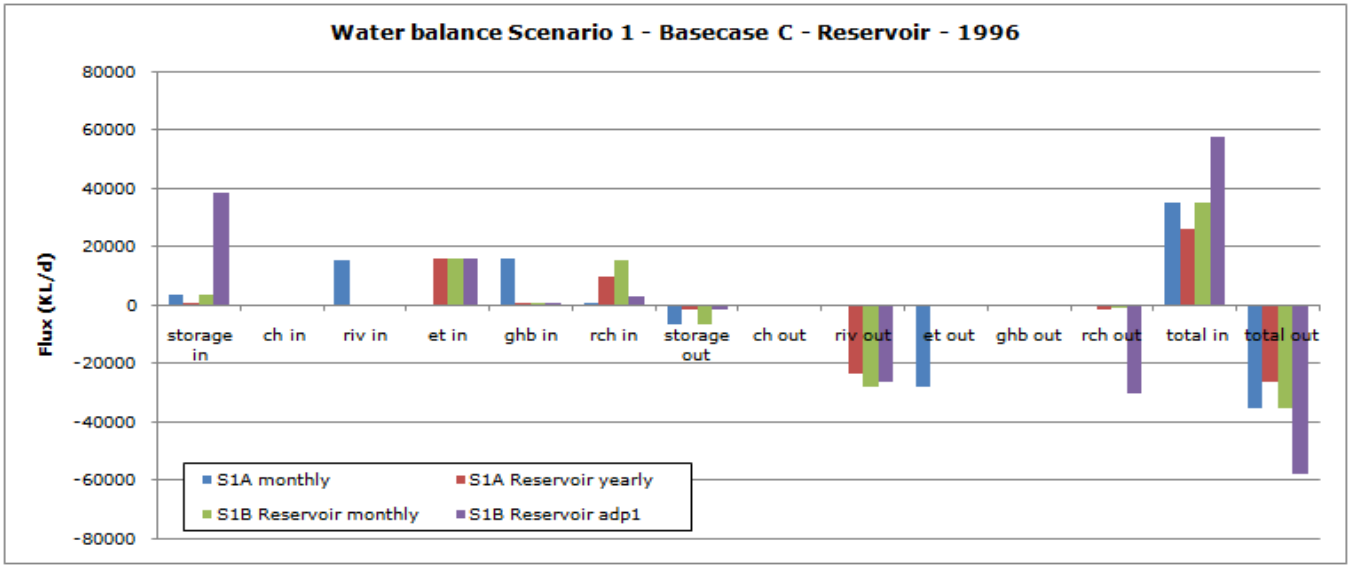
SCENARIO 1 - GOYDER MODEL C MONTHLY SP RIVER VS RESERVOIR

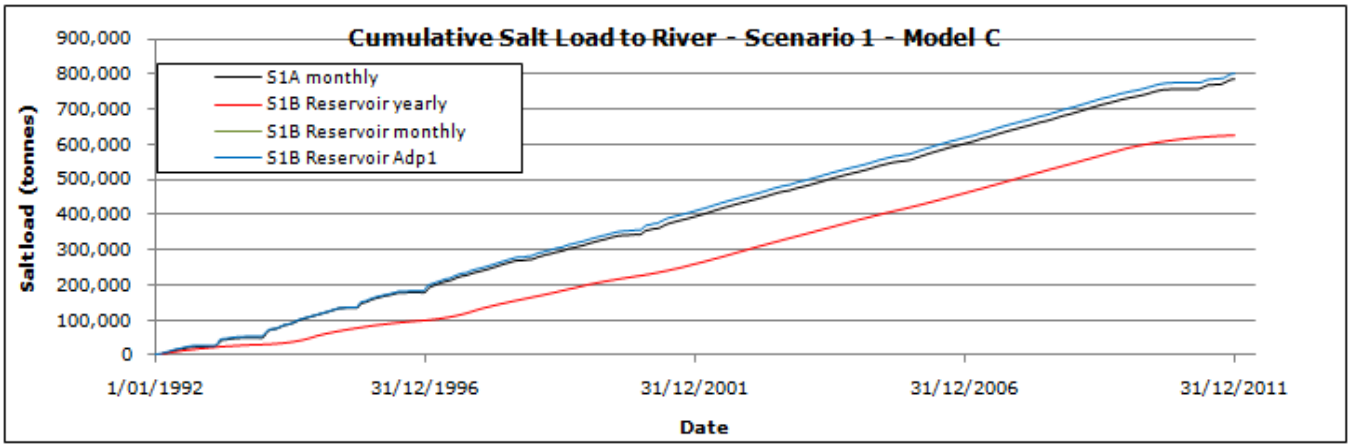


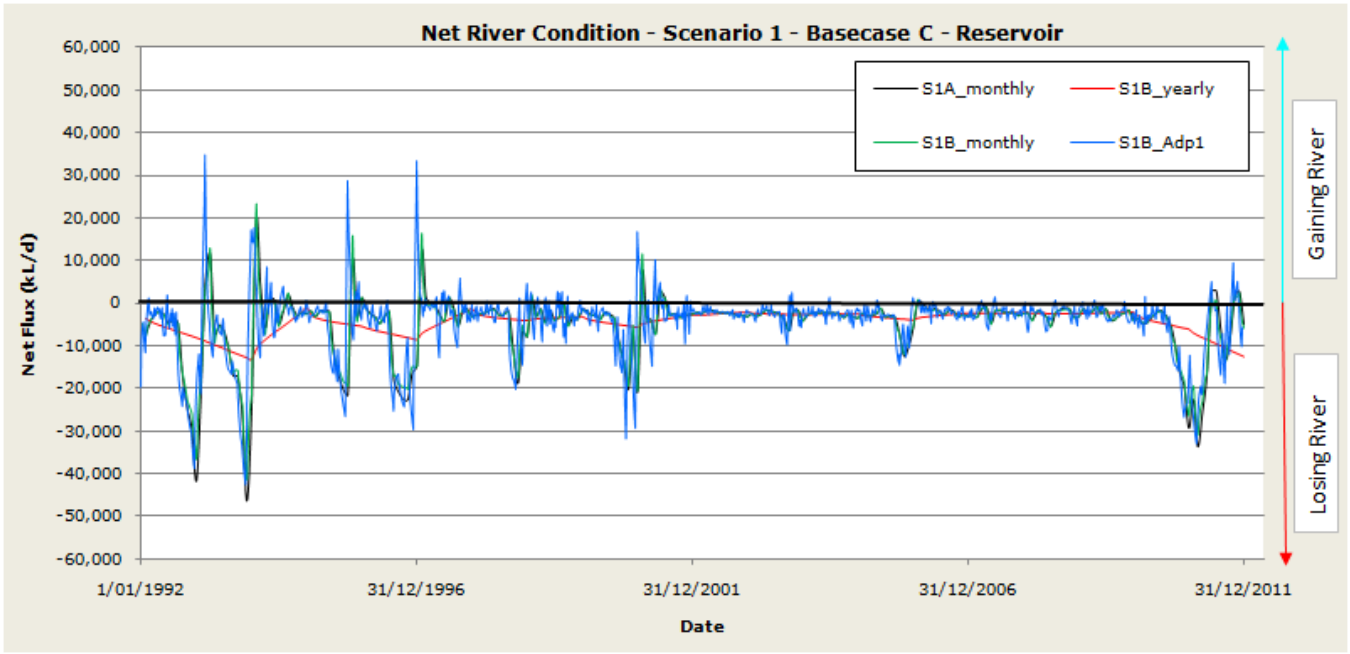
SCENARIO 1 - GOYDER MODEL C - ADAPTIVE SP RIVER VS RESERVOIR



SCENARIO 1 - GOYDER MODEL C - ADAPTIVE SP RIVER VS RESERVOIR


Water balance Scenario 1 - Basecase C - Reservoir - 1996
Flux (KL/d)
80000
60000
40000
20000
0
-20000
-40000
-60000
-80000
storage in
ch in
riv in
et in
ghb in
rch in
storage out
ch out
riv out
et out
ghb out
rch out
total in
total out
S1A monthly
S1A Reservoir yearly
S1B Reservoir monthly
S1B Reservoir adp1

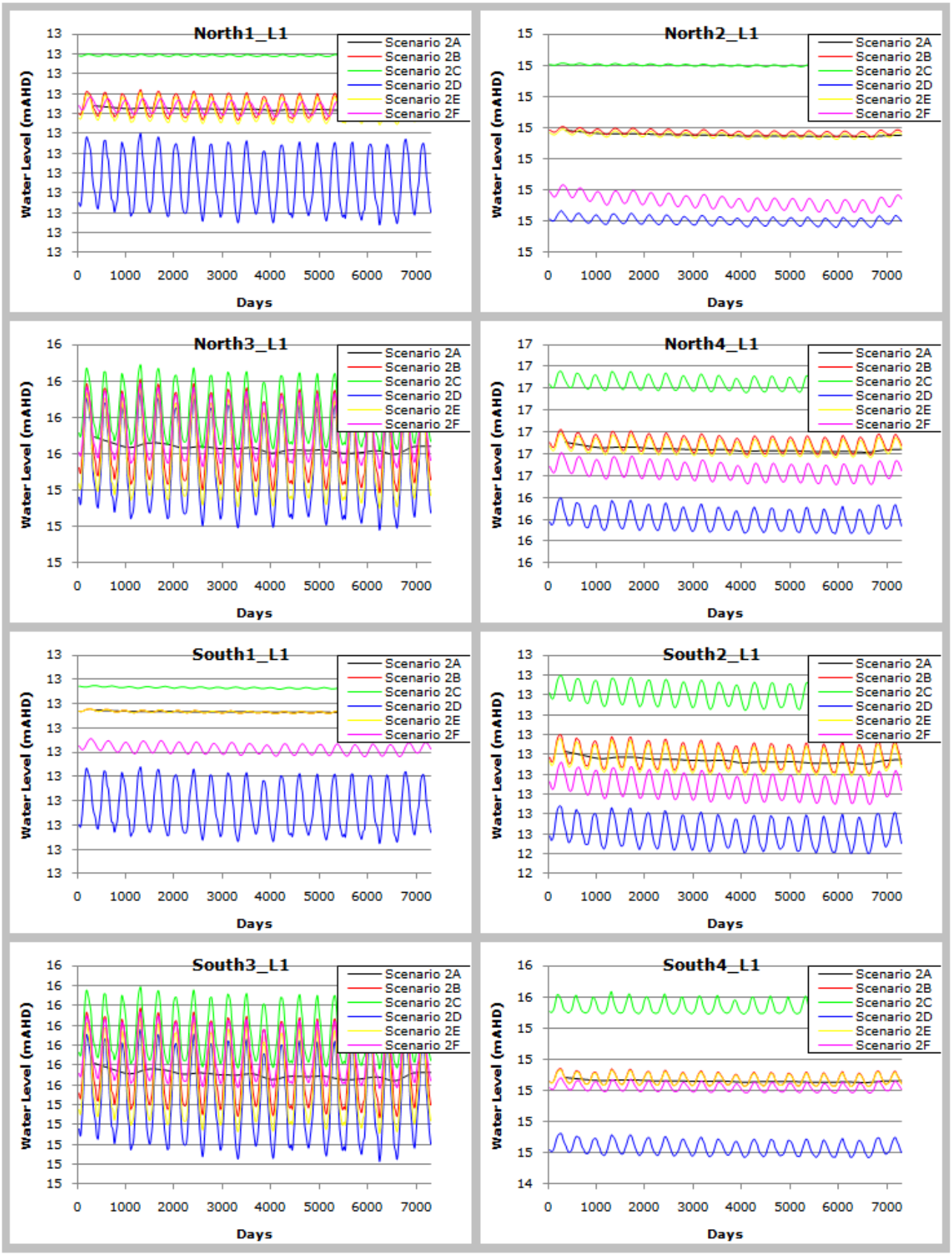

Cumulative Salt Load to River - Scenario 1 - Model C
Salt load (tonnes)
900,000
800,000
700,000
600,000
500,000
400,000
300,000
200,000
100,000
0
S1A monthly
S1B Reservoir yearly
S1B Reservoir monthly
S1B Reservoir Adp1
1/01/1992
31/12/1996
31/12/2001
31/12/2006
31/12/2011
Date


Net River Condition - Scenario 1 - Basecase C - Reservoir
Net Flux (kL/d)
60,000
50,000
40,000
30,000
20,000
10,000
0
-10,000
-20,000
-30,000
-40,000
-50,000
-60,000
S1A_monthly
S1B_yearly
S1B_monthly
S1B_Adp1
Gaining River
Losing River
1/01/1992
31/12/1996
31/12/2001
31/12/2006
31/12/2011
Date

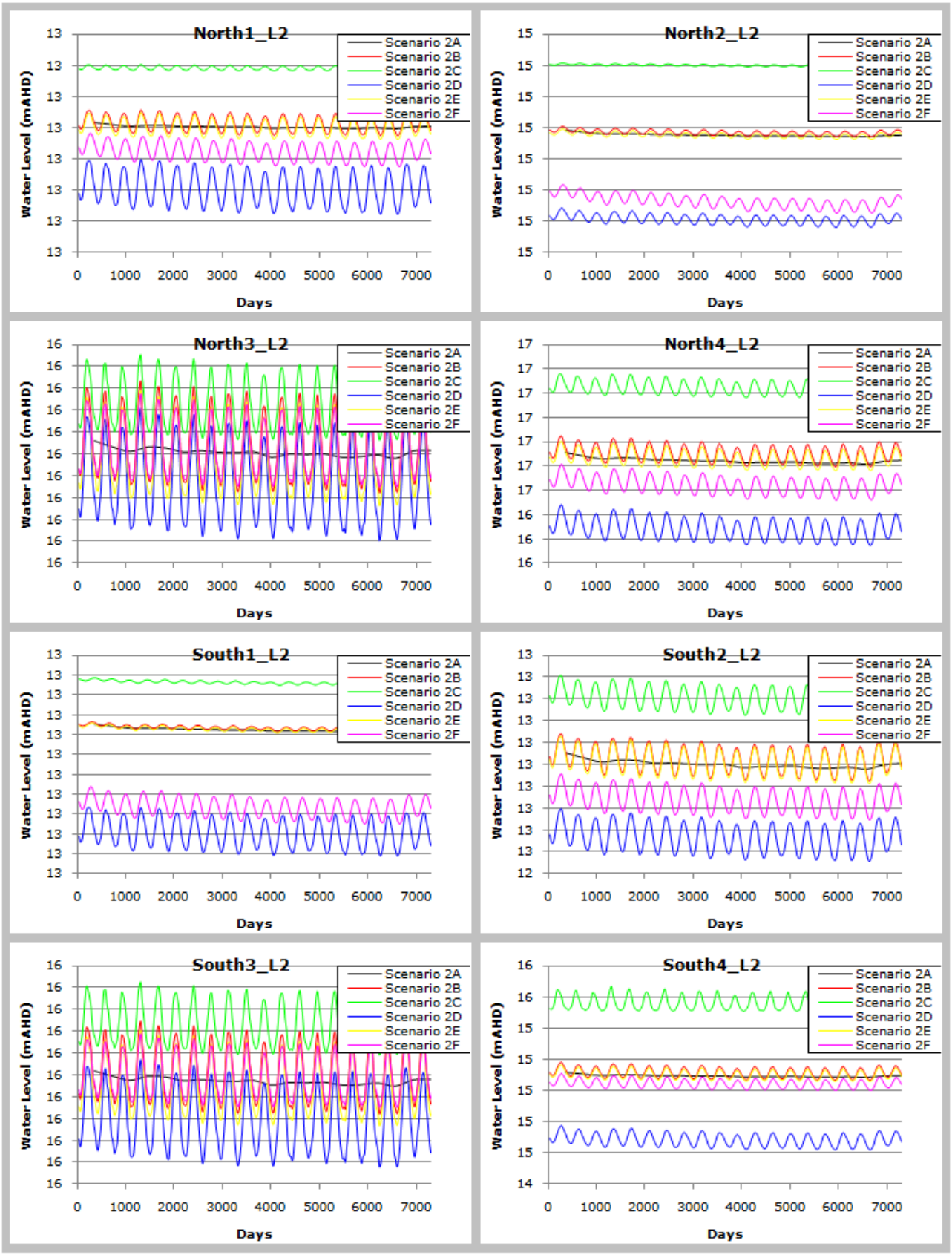
Appendix C11: Model results for Scenario 2 – Model A – Locked

The following figures present the transient information of model behaviour for the scenario in the title. These simulations feature different representations of the evapotranspiration function and the effects in the oservation bores. Details of the specific functions are given below

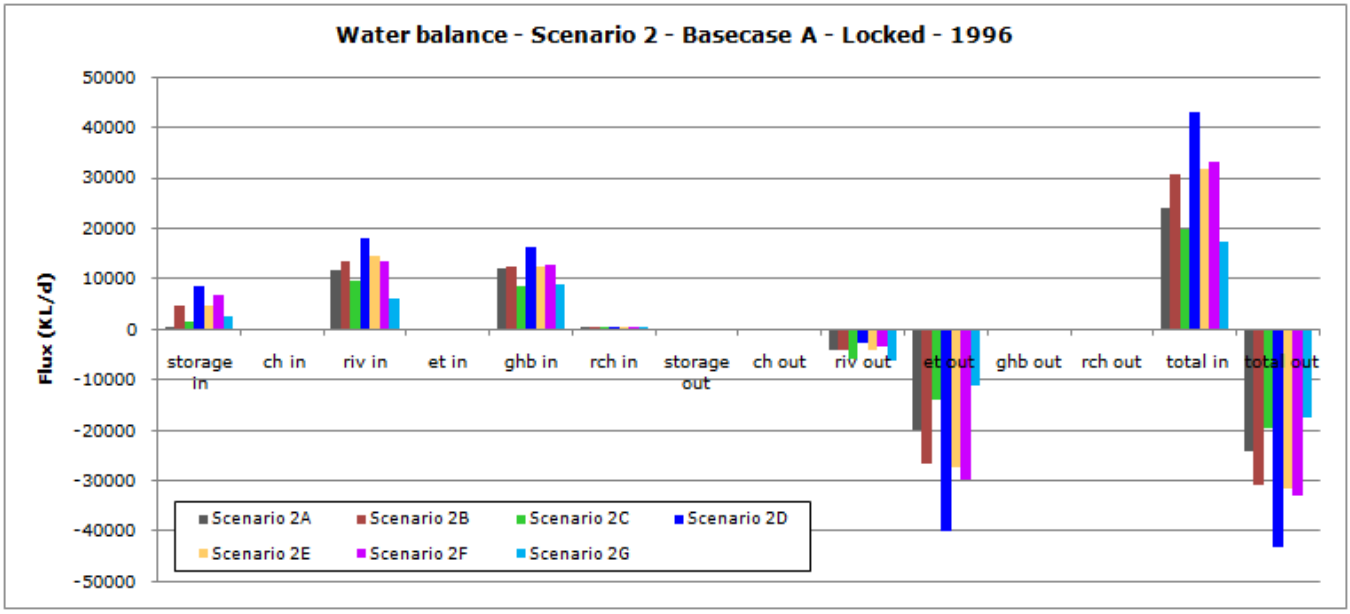
- Evapotranspiration varying yearly (Scenario 2A)
- Evapotranspiration varying monthly (Scenario 2B)
- Evapotranspiration varying monthly with a 0.5 m reduction in extinction depth (Scenario 2C)
- Evapotranspiration varying monthly with a 0.5 m increase in extinction depth (Scenario 2D)
- Spatial varying evapotranspiration rate including monthly variation (Scenario 2E)
- Spatial varying evapotranspiration rate and extinction depth, including monthly variation (Scenario 2F)
- Evapotranspiration varying monthly, using a non-linear evapotranspiration versus depth function, ETS curve (Scenario 2G)

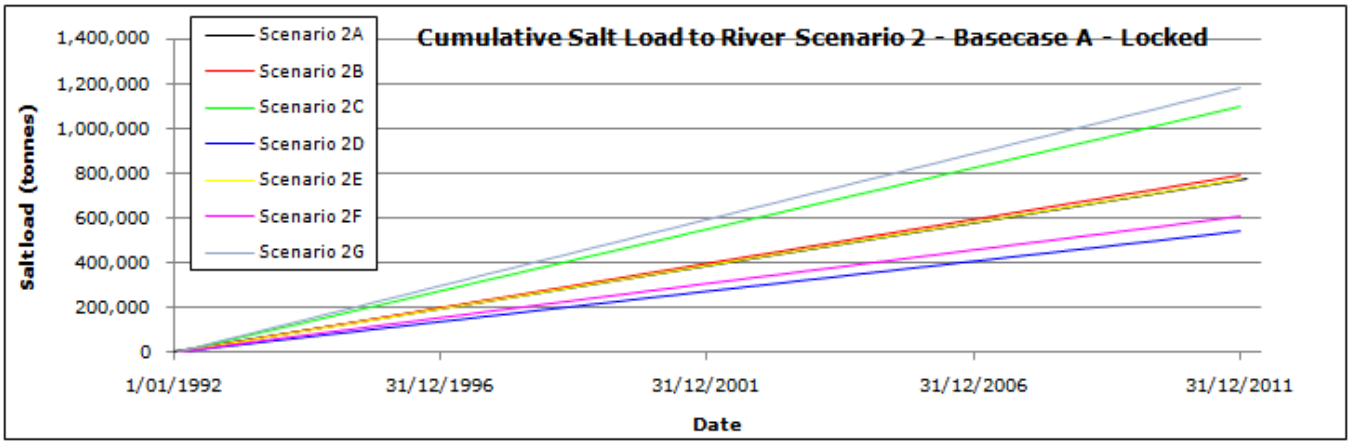


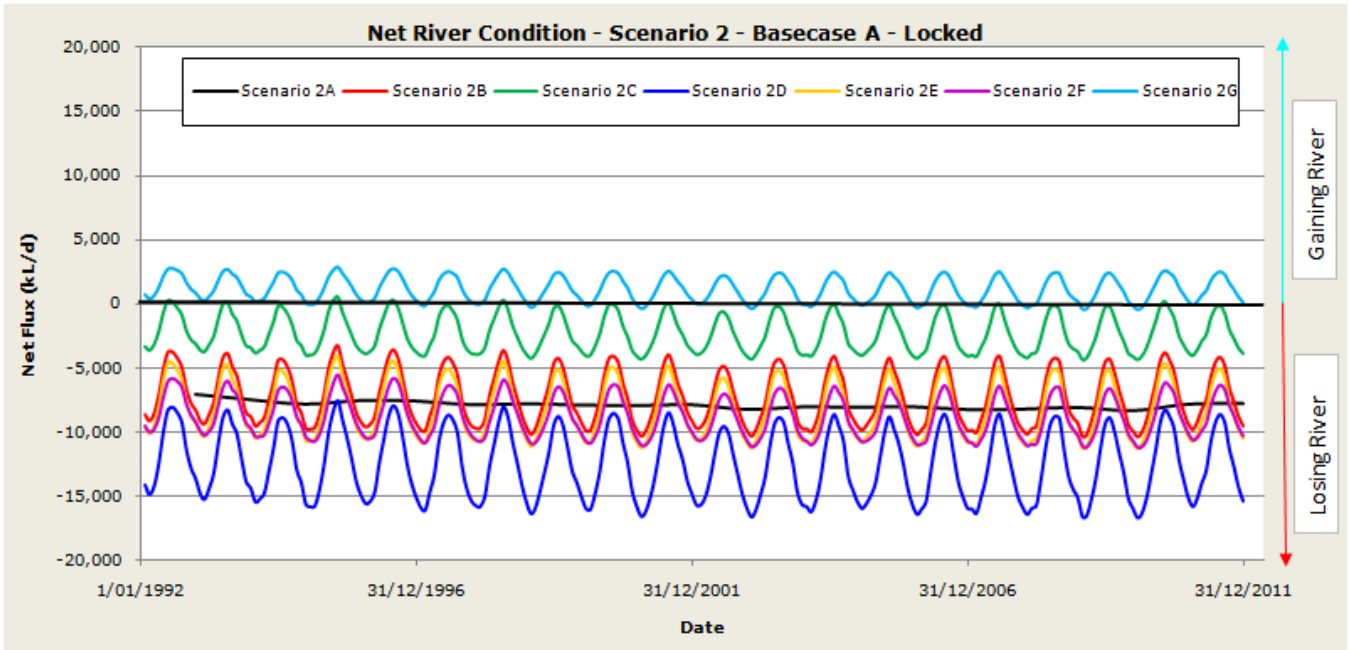
SCENARIO 2 GOYDER MODEL A - LOCKED



SCENARIO 2 GOYDER MODEL A - LOCKED


Water balance - Scenario 2 - Basecase A - Locked - 1996
Flux (KL/d)
50000
40000
30000
20000
10000
0
-10000
-20000
-30000
-40000
-50000
storage in
ch in
riv in
et in
ghb in
rch in
storage out
ch out
riv out
et out
ghb out
rch out
total in
total out
Scenario 2A
Scenario 2B
Scenario 2C
Scenario 2D
Scenario 2E
Scenario 2F
Scenario 2G

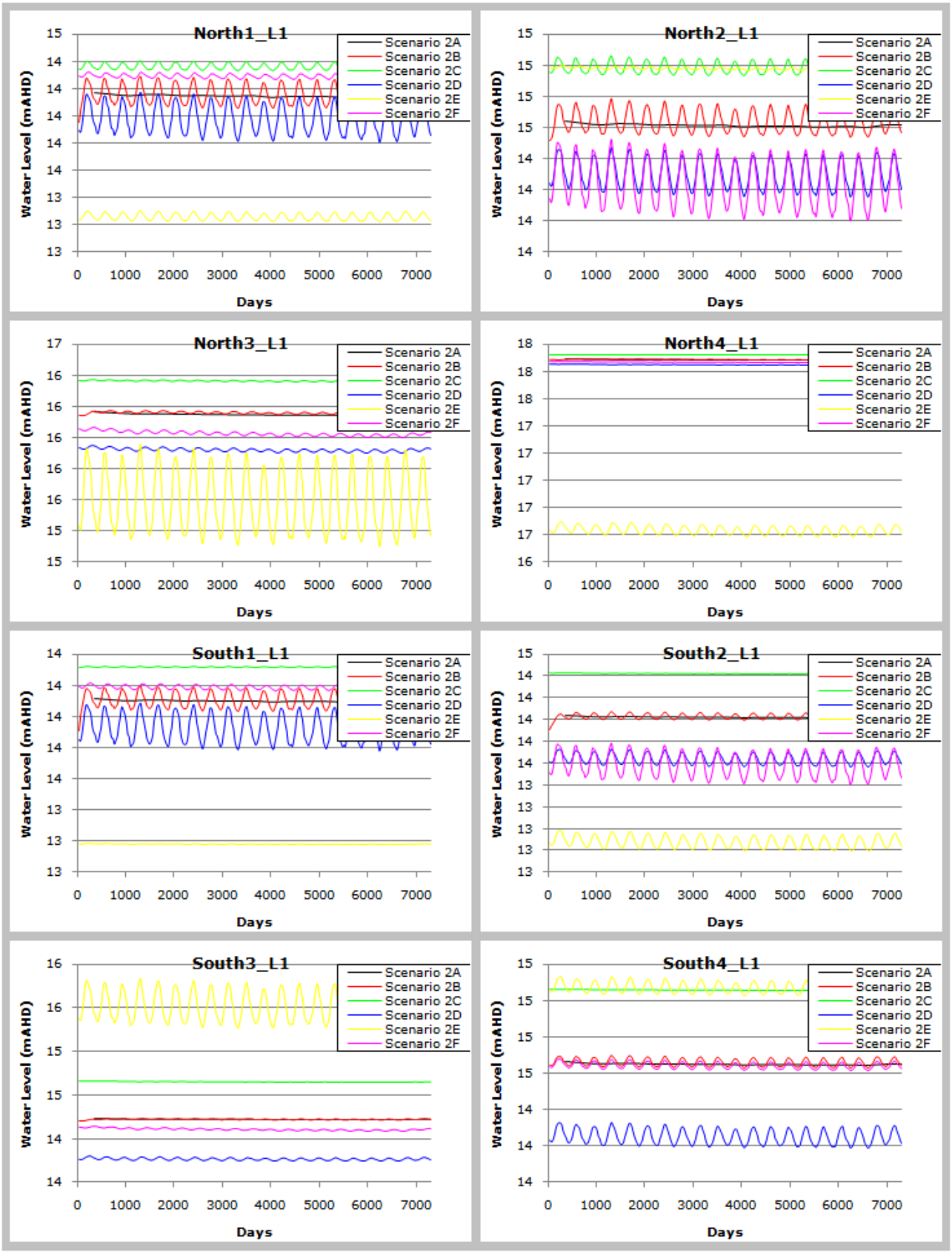

Cumulative Salt Load to River Scenario 2 - Basecase A - Locked
Salt load (tonnes)
1,400,000
1,200,000
1,000,000
800,000
600,000
400,000
200,000
0
1/01/1992
31/12/1996
31/12/2001
31/12/2006
31/12/2011
Date
Scenario 2A
Scenario 2B
Scenario 2C
Scenario 2D
Scenario 2E
Scenario 2F
Scenario 2G


Net River Condition - Scenario 2 - Basecase A - Locked
Net Flux (kL/d)
20,000
15,000
10,000
5,000
0
-5,000
-10,000
-15,000
-20,000
1/01/1992
31/12/1996
31/12/2001
31/12/2006
31/12/2011
Date
Scenario 2A
Scenario 2B
Scenario 2C
Scenario 2D
Scenario 2E
Scenario 2F
Scenario 2G
Gaining River
Losing River

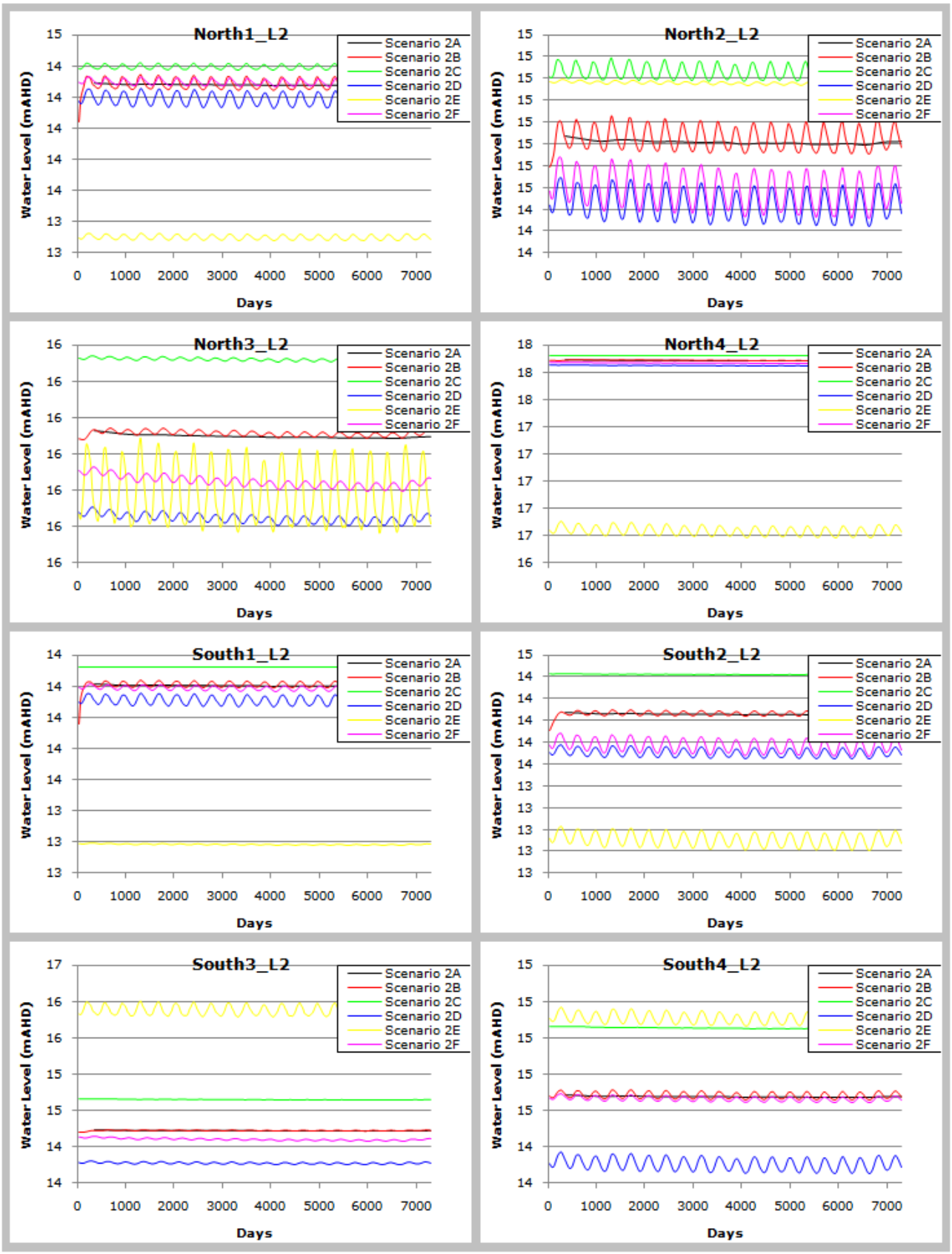
Appendix C12: Model results for Scenario 2 – Model A – Not locked

The following figures present the transient information of model behaviour for the scenario in the title. These simulations feature different representations of the evapotranspiration function and the effects in the oservation bores. Details of the specific functions are given below

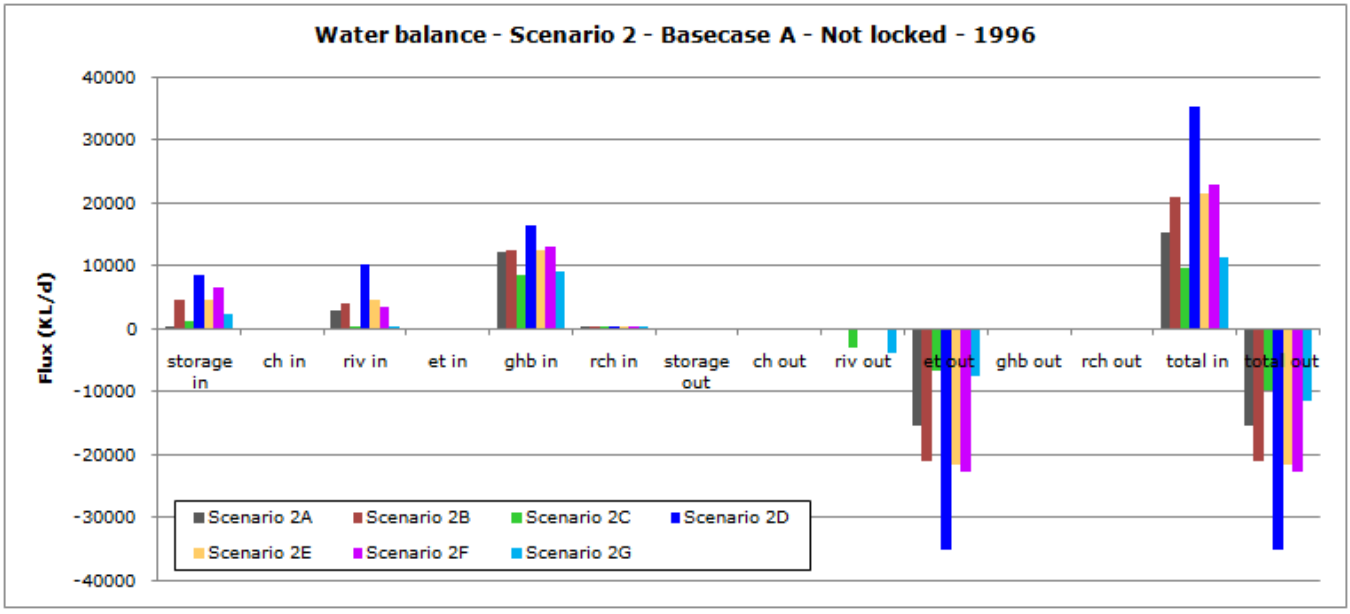
- Evapotranspiration varying yearly (Scenario 2A)
- Evapotranspiration varying monthly (Scenario 2B)
- Evapotranspiration varying monthly with a 0.5 m reduction in extinction depth (Scenario 2C)
- Evapotranspiration varying monthly with a 0.5 m increase in extinction depth (Scenario 2D)
- Spatial varying evapotranspiration rate including monthly variation (Scenario 2E)
- Spatial varying evapotranspiration rate and extinction depth, including monthly variation (Scenario 2F)
- Evapotranspiration varying monthly, using a non-linear evapotranspiration versus depth function, ETS curve (Scenario 2G)

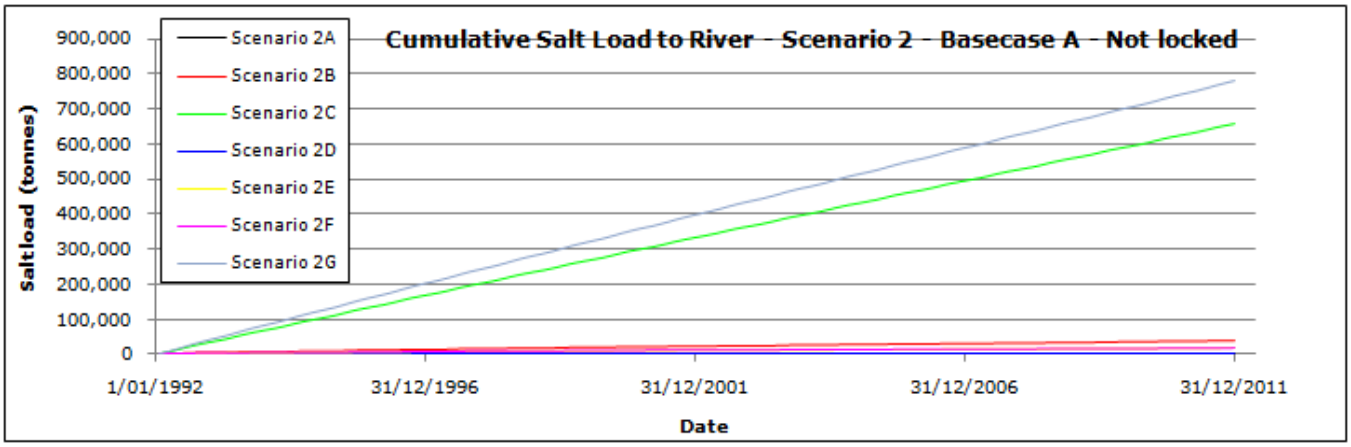


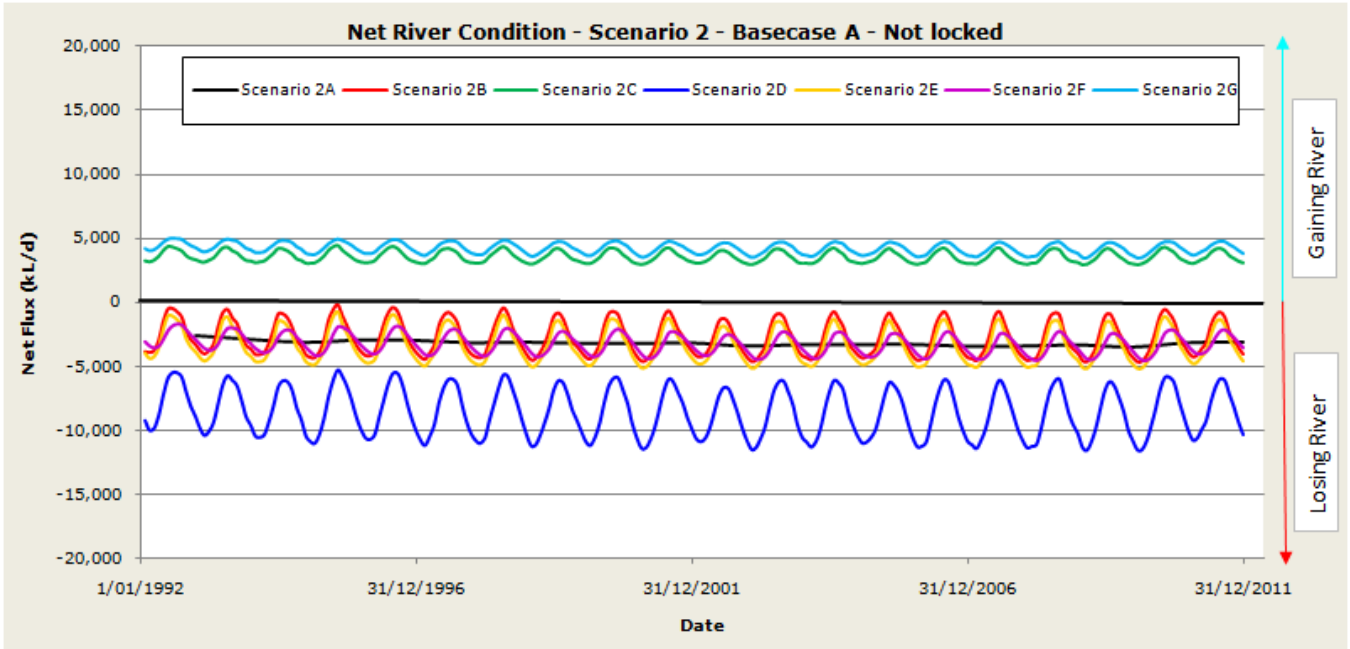
SCENARIO 2 - GOYDER MODEL A - NOT LOCKED



SCENARIO 2 - GOYDER MODEL A - NOT LOCKED


Water balance - Scenario 2 - Basecase A - Not locked - 1996
Flux (KL/d)
40000
30000
20000
10000
0
-10000
-20000
-30000
-40000
storage in
ch in
riv in
et in
ghb in
rch in
storage out
ch out
riv out
et out
ghb out
rch out
total in
total out
Scenario 2A
Scenario 2B
Scenario 2C
Scenario 2D
Scenario 2E
Scenario 2F
Scenario 2G

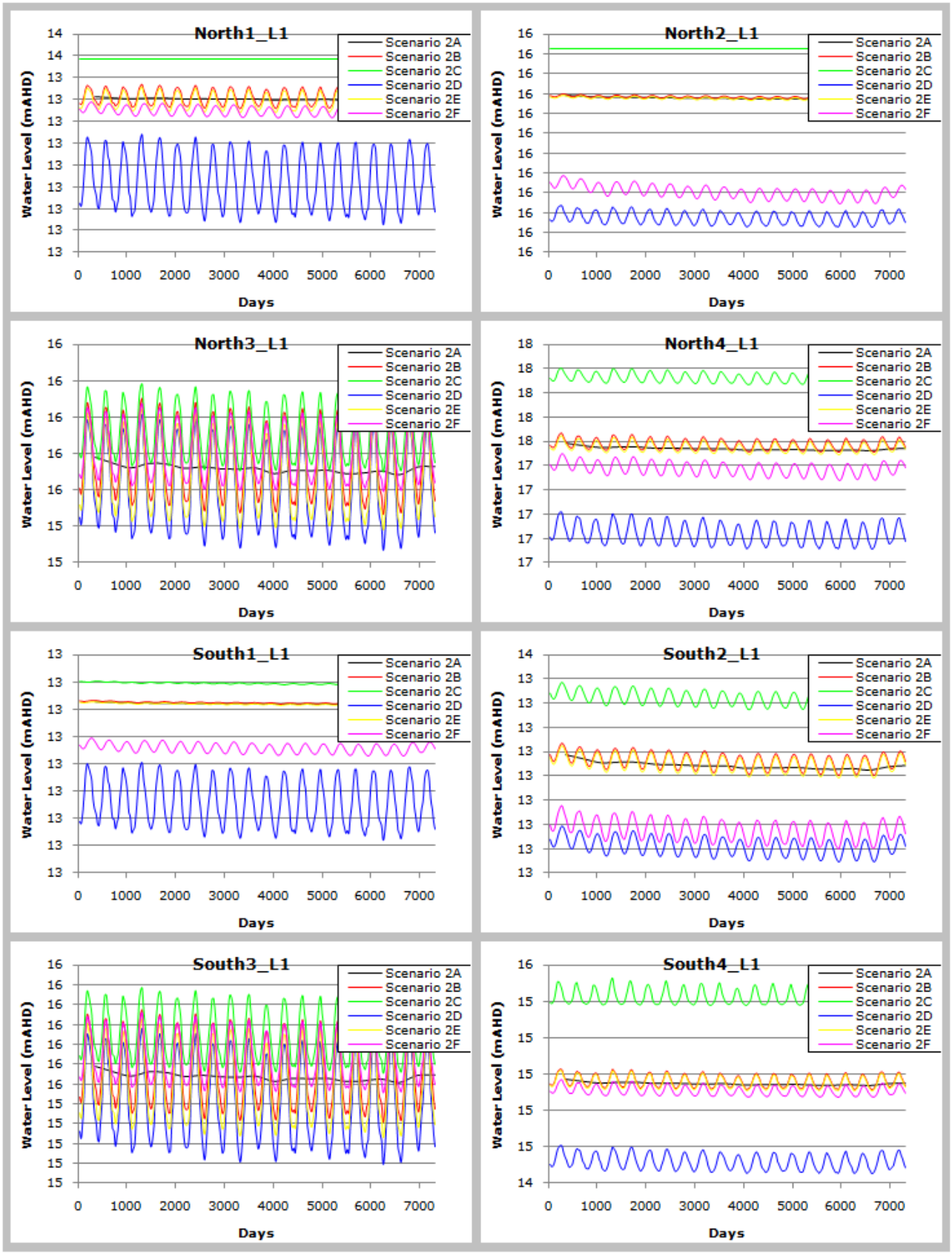

Cumulative Salt Load to River - Scenario 2 - Basecase A - Not locked
Salt load (tonnes)
900,000
800,000
700,000
600,000
500,000
400,000
300,000
200,000
100,000
0
Scenario 2A
Scenario 2B
Scenario 2C
Scenario 2D
Scenario 2E
Scenario 2F
Scenario 2G
1/01/1992
31/12/1996
31/12/2001
31/12/2006
31/12/2011
Date


Net River Condition - Scenario 2 - Basecase A - Not locked
Net Flux (kL/d)
20,000
15,000
10,000
5,000
0
-5,000
-10,000
-15,000
-20,000
Scenario 2A
Scenario 2B
Scenario 2C
Scenario 2D
Scenario 2E
Scenario 2F
Scenario 2G
Gaining River
Losing River
1/01/1992
31/12/1996
31/12/2001
31/12/2006
31/12/2011
Date

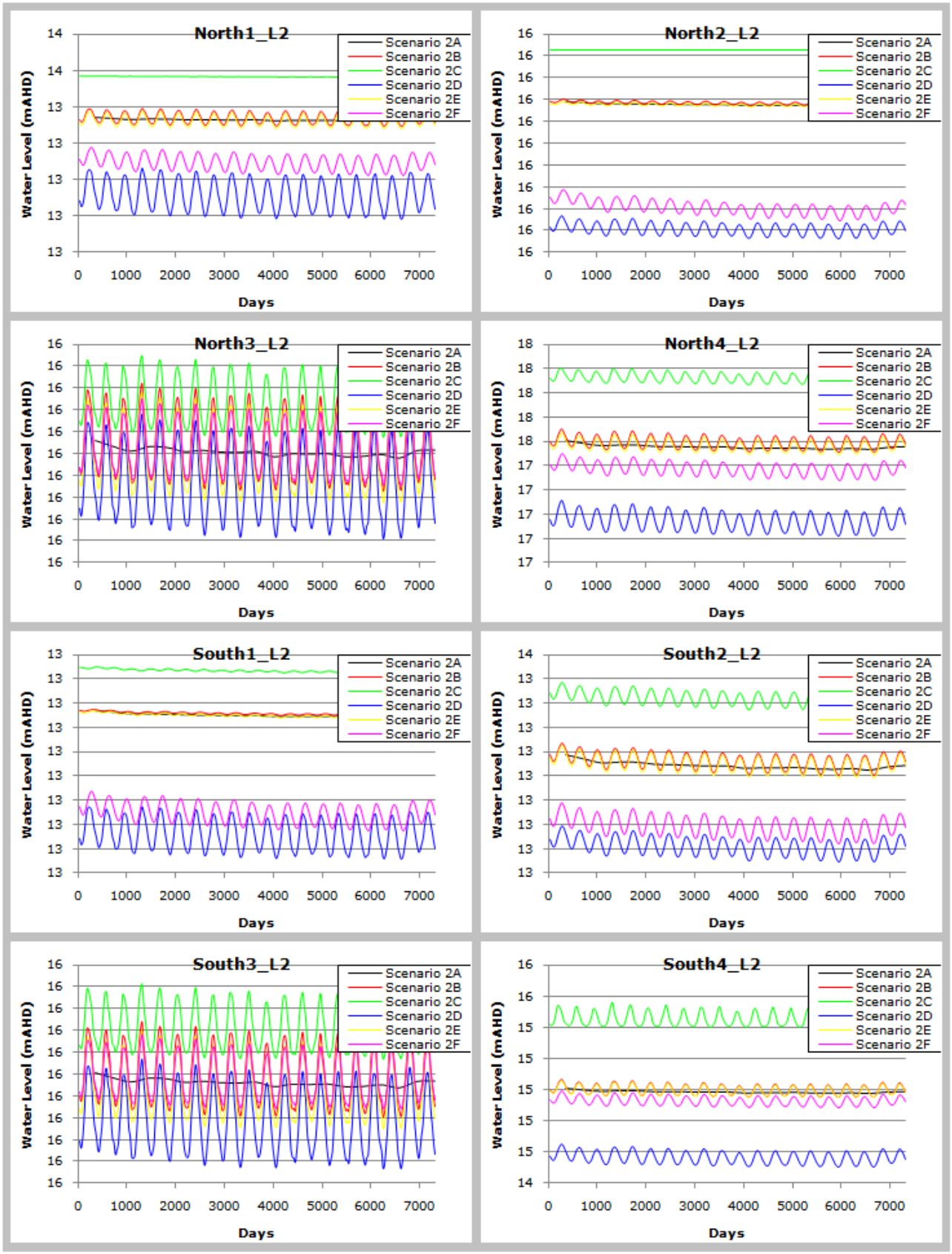
Appendix C13: Model results for Scenario 2 – Model B – Locked

The following figures present the transient information of model behaviour for the scenario in the title. These simulations feature different representations of the evapotranspiration function and the effects in the oservation bores. Details of the specific functions are given below

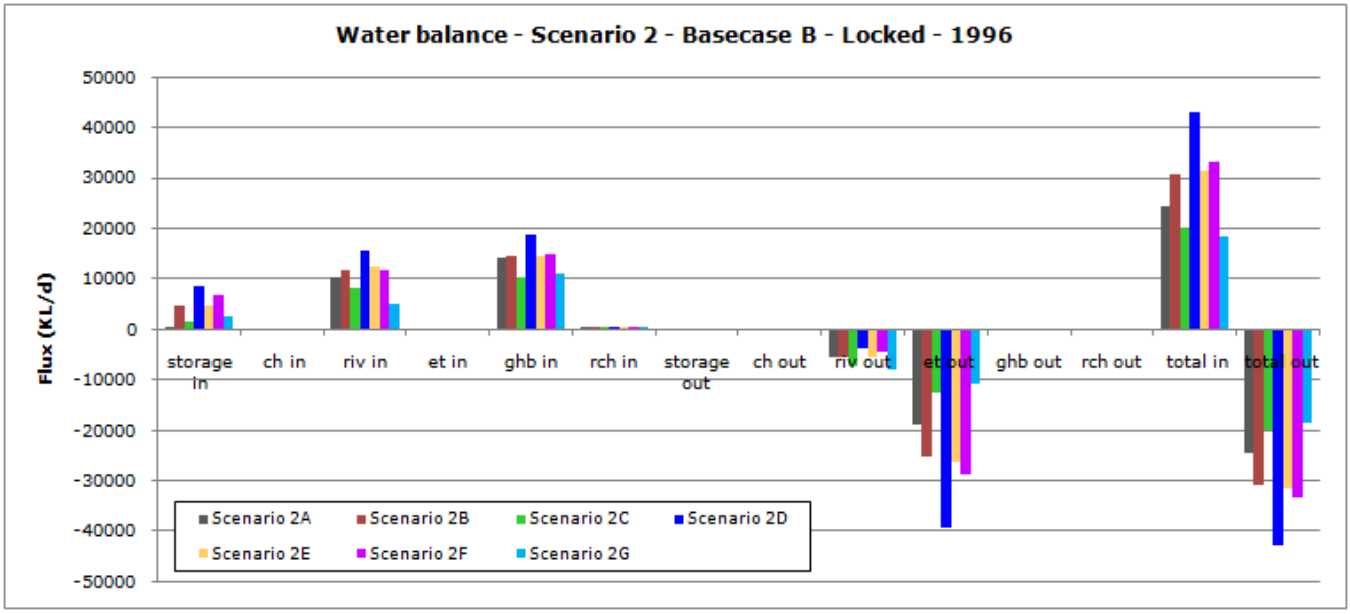
- Evapotranspiration varying yearly (Scenario 2A)
- Evapotranspiration varying monthly (Scenario 2B)
- Evapotranspiration varying monthly with a 0.5 m reduction in extinction depth (Scenario 2C)
- Evapotranspiration varying monthly with a 0.5 m increase in extinction depth (Scenario 2D)
- Spatial varying evapotranspiration rate including monthly variation (Scenario 2E)
- Spatial varying evapotranspiration rate and extinction depth, including monthly variation (Scenario 2F)
- Evapotranspiration varying monthly, using a non-linear evapotranspiration versus depth function, ETS curve (Scenario 2G)

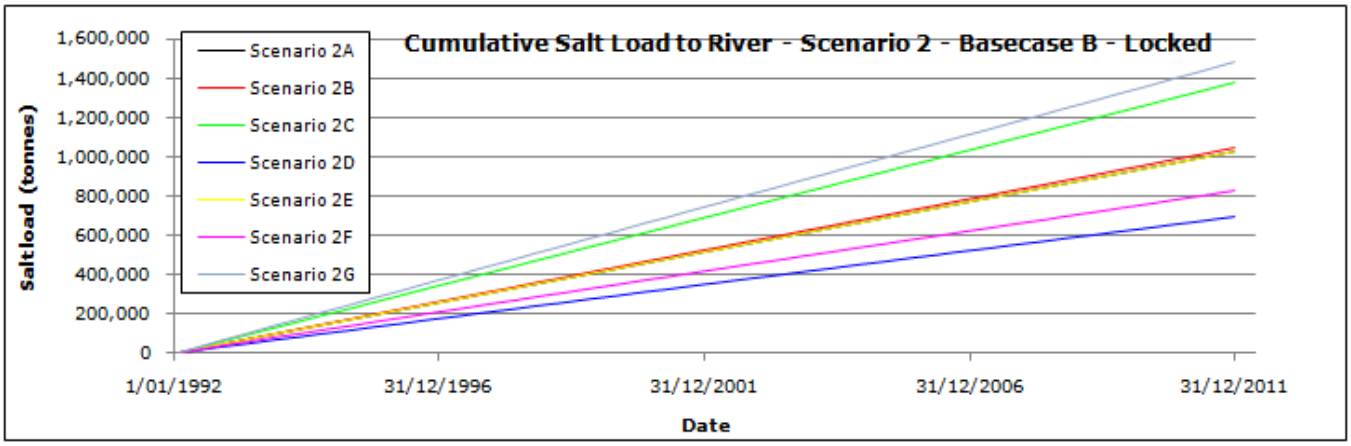


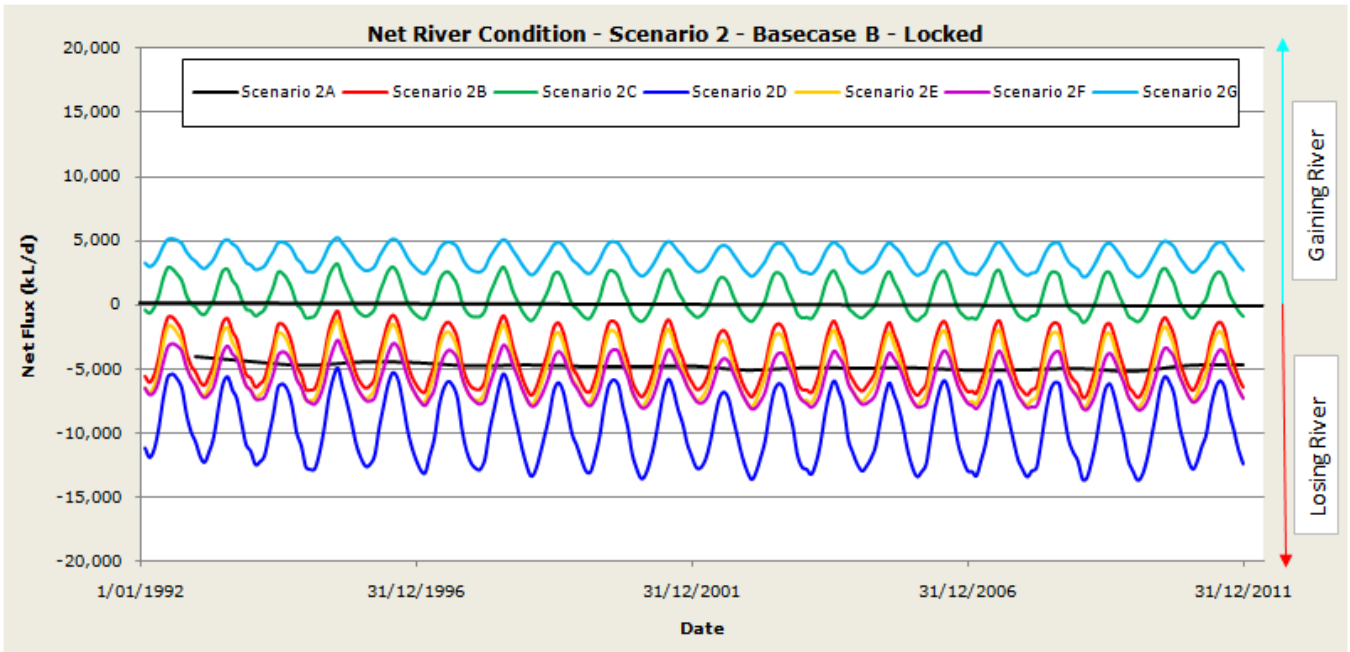
SCENARIO 2 GOYDER MODEL B - LOCKED



SCENARIO 2 GOYDER MODEL B - LOCKED


Water balance - Scenario 2 - Basecase B - Locked - 1996
Flux (KL/d)
50000
40000
30000
20000
10000
0
-10000
-20000
-30000
-40000
-50000
storage in
ch in
riv in
et in
ghb in
rch in
storage out
ch out
riv out
et out
ghb out
rch out
total in
total out
Scenario 2A
Scenario 2B
Scenario 2C
Scenario 2D
Scenario 2E
Scenario 2F
Scenario 2G

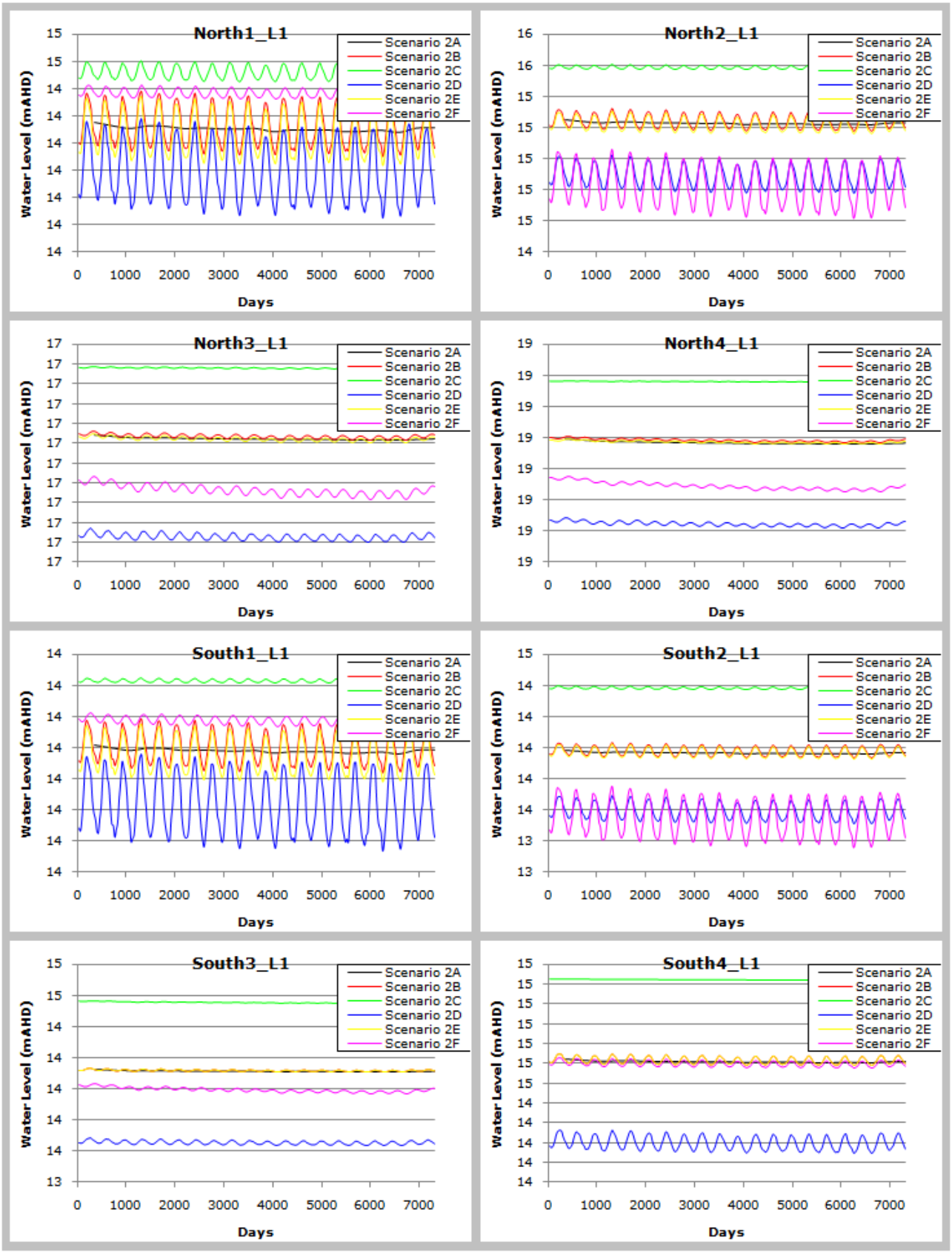

Cumulative Salt Load to River - Scenario 2 - Basecase B - Locked
Saltload (tonnes)
1,600,000
1,400,000
1,200,000
1,000,000
800,000
600,000
400,000
200,000
0
Scenario 2A
Scenario 2B
Scenario 2C
Scenario 2D
Scenario 2E
Scenario 2F
Scenario 2G
1/01/1992
31/12/1996
31/12/2001
31/12/2006
31/12/2011
Date


Net River Condition - Scenario 2 - Basecase B - Locked
Net Flux (kL/d)
20,000
15,000
10,000
5,000
0
-5,000
-10,000
-15,000
-20,000
Scenario 2A
Scenario 2B
Scenario 2C
Scenario 2D
Scenario 2E
Scenario 2F
Scenario 2G
Gaining River
Losing River
1/01/1992
31/12/1996
31/12/2001
31/12/2006
31/12/2011
Date

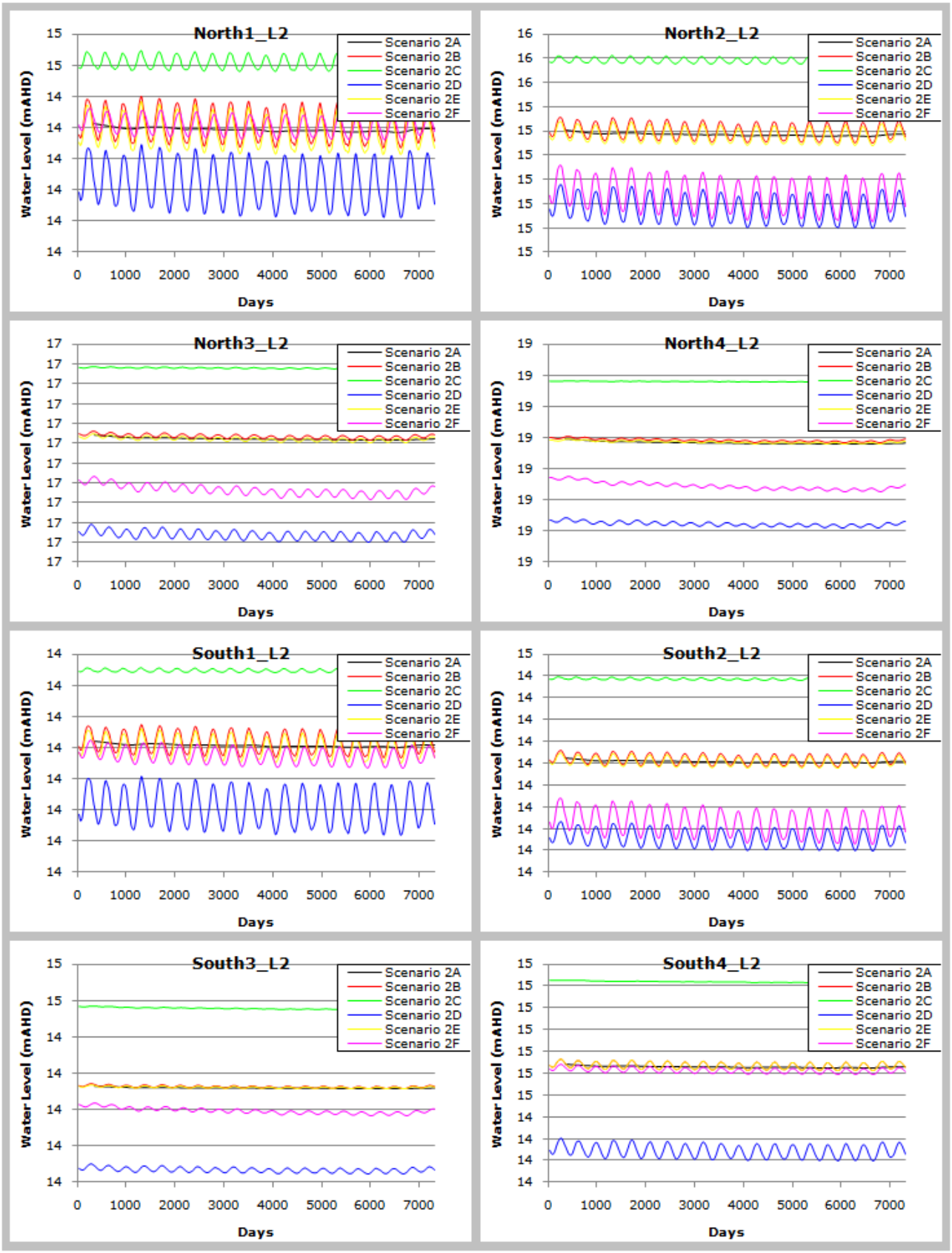
Appendix C14: Model results for Scenario 2 – Model B – Not locked

The following figures present the transient information of model behaviour for the scenario in the title. These simulations feature different representations of the evapotranspiration function and the effects in the oservation bores. Details of the specific functions are given below

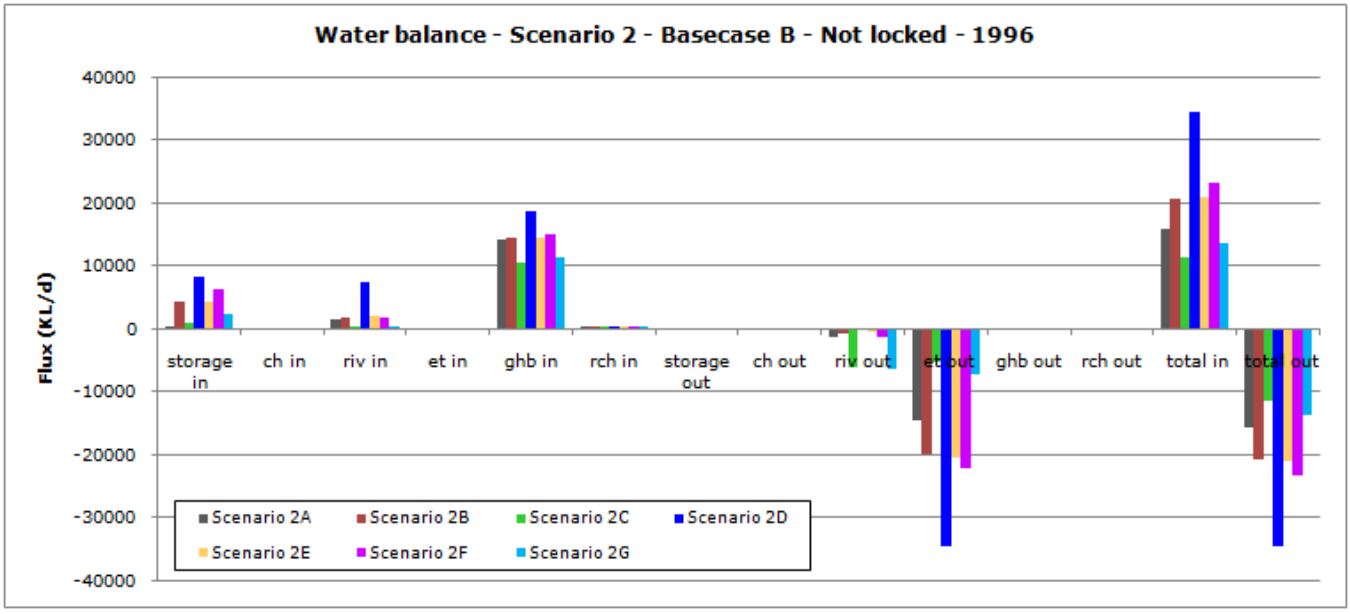
- Evapotranspiration varying yearly (Scenario 2A)
- Evapotranspiration varying monthly (Scenario 2B)
- Evapotranspiration varying monthly with a 0.5 m reduction in extinction depth (Scenario 2C)
- Evapotranspiration varying monthly with a 0.5 m increase in extinction depth (Scenario 2D)
- Spatial varying evapotranspiration rate including monthly variation (Scenario 2E)
- Spatial varying evapotranspiration rate and extinction depth, including monthly variation (Scenario 2F)
- Evapotranspiration varying monthly, using a non-linear evapotranspiration versus depth function, ETS curve (Scenario 2G)

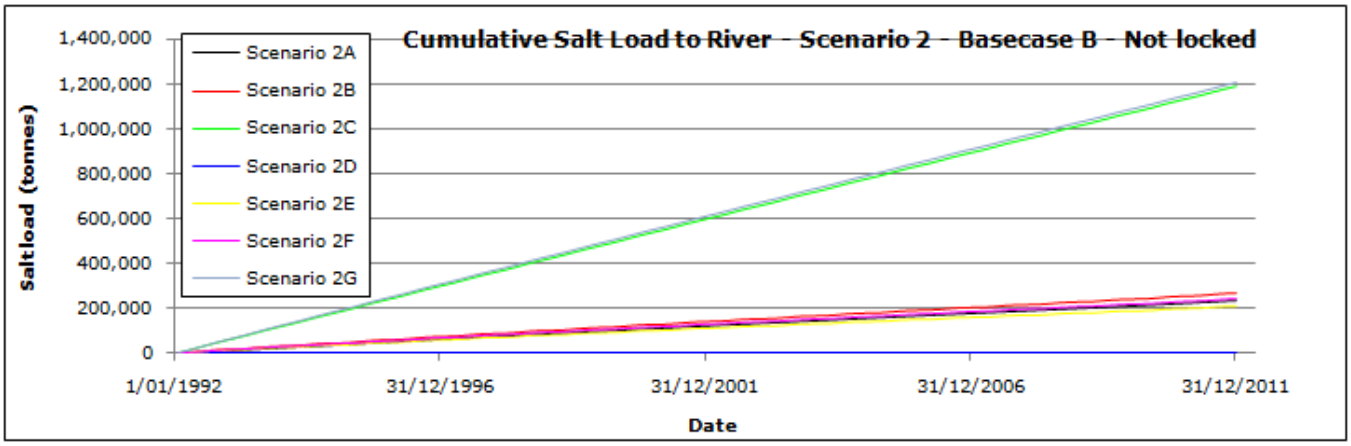


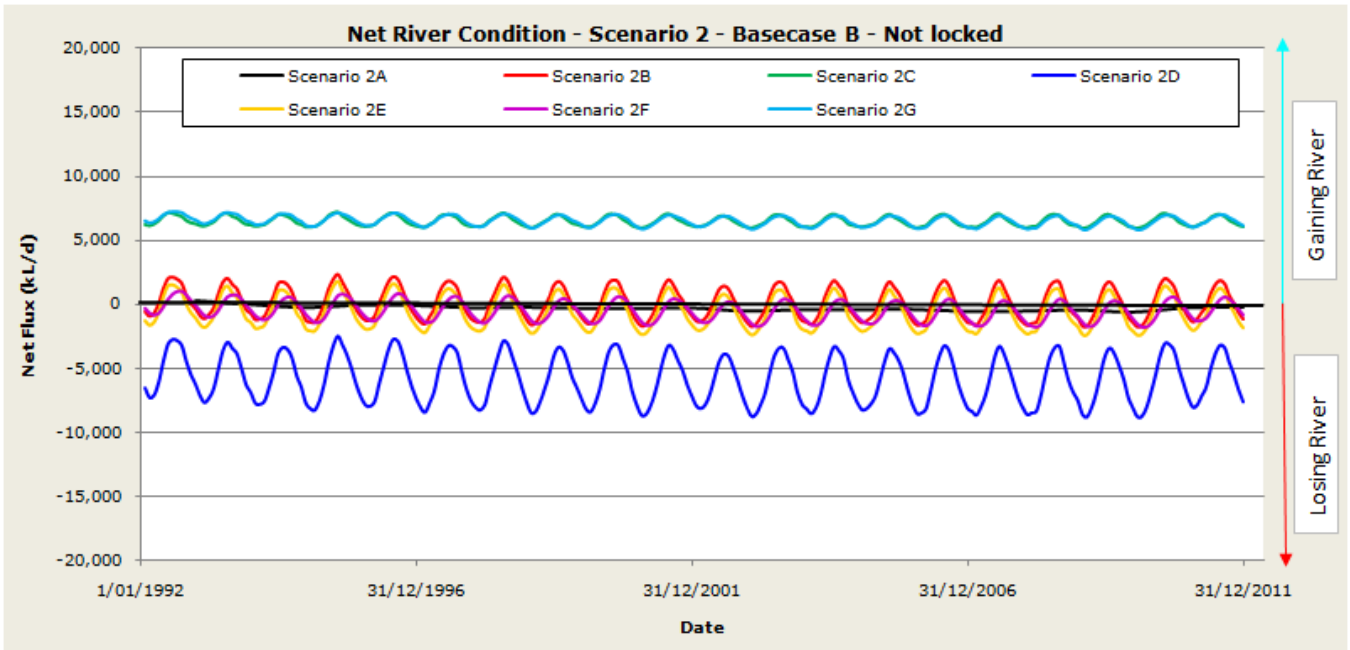
SCENARIO 2 - GOYDER MODEL B - NOT LOCKED



SCENARIO 2 - GOYDER MODEL B - NOT LOCKED


Water balance - Scenario 2 - Basecase B - Not locked - 1996
Flux (KL/d)
40000
30000
20000
10000
0
-10000
-20000
-30000
-40000
storage in
ch in
riv in
et in
ghb in
rch in
storage out
ch out
riv out
et out
ghb out
rch out
total in
total out
Scenario 2A
Scenario 2B
Scenario 2C
Scenario 2D
Scenario 2E
Scenario 2F
Scenario 2G

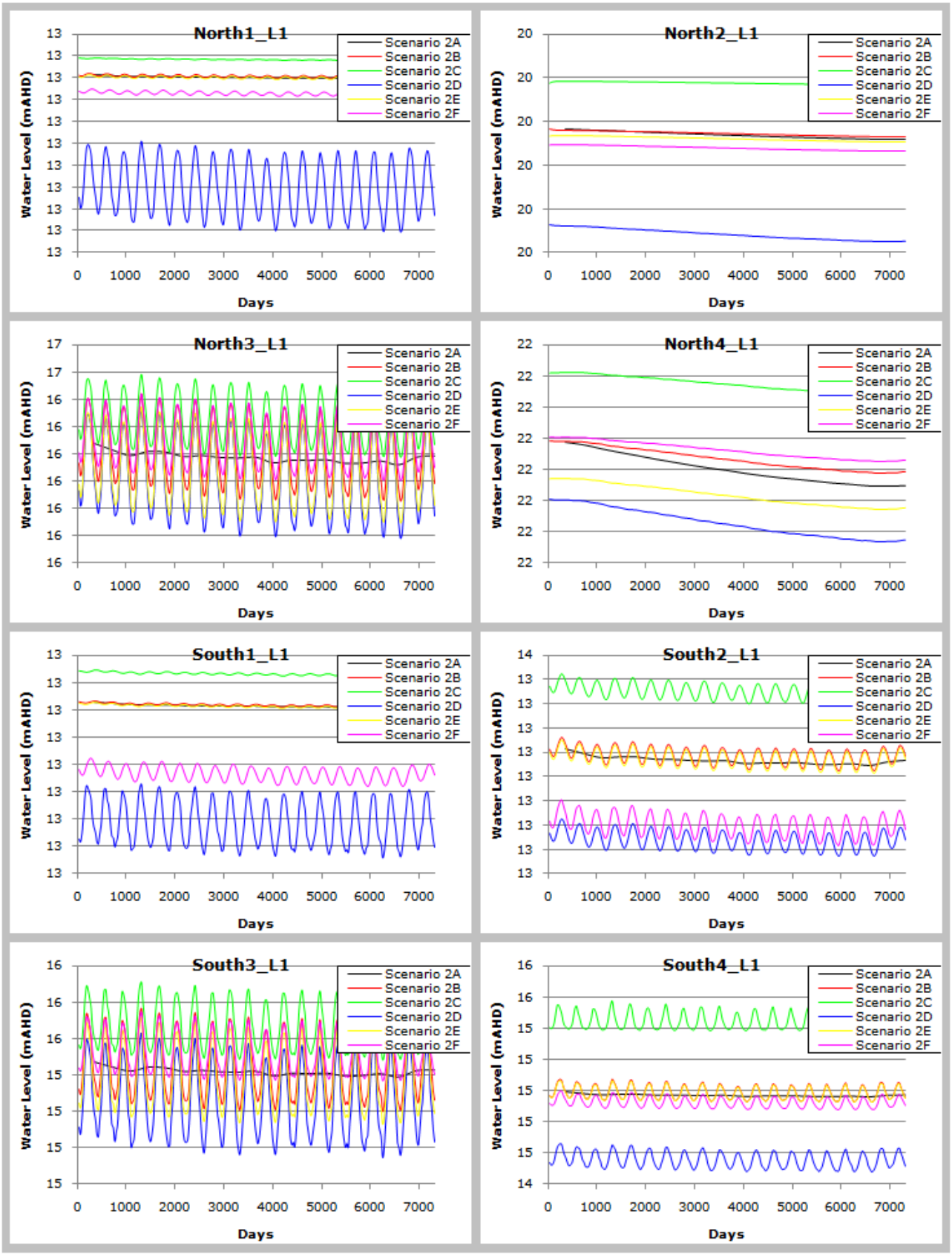

Cumulative Salt Load to River - Scenario 2 - Basecase B - Not locked
Salt load (tonnes)
1,400,000
1,200,000
1,000,000
800,000
600,000
400,000
200,000
0
1/01/1992
31/12/1996
31/12/2001
31/12/2006
31/12/2011
Date
Scenario 2A
Scenario 2B
Scenario 2C
Scenario 2D
Scenario 2E
Scenario 2F
Scenario 2G


Net River Condition - Scenario 2 - Basecase B - Not locked
Net Flux (kL/d)
20,000
15,000
10,000
5,000
0
-5,000
-10,000
-15,000
-20,000
1/01/1992
31/12/1996
31/12/2001
31/12/2006
31/12/2011
Date
Scenario 2A
Scenario 2B
Scenario 2C
Scenario 2D
Scenario 2E
Scenario 2F
Scenario 2G
Gaining River
Losing River

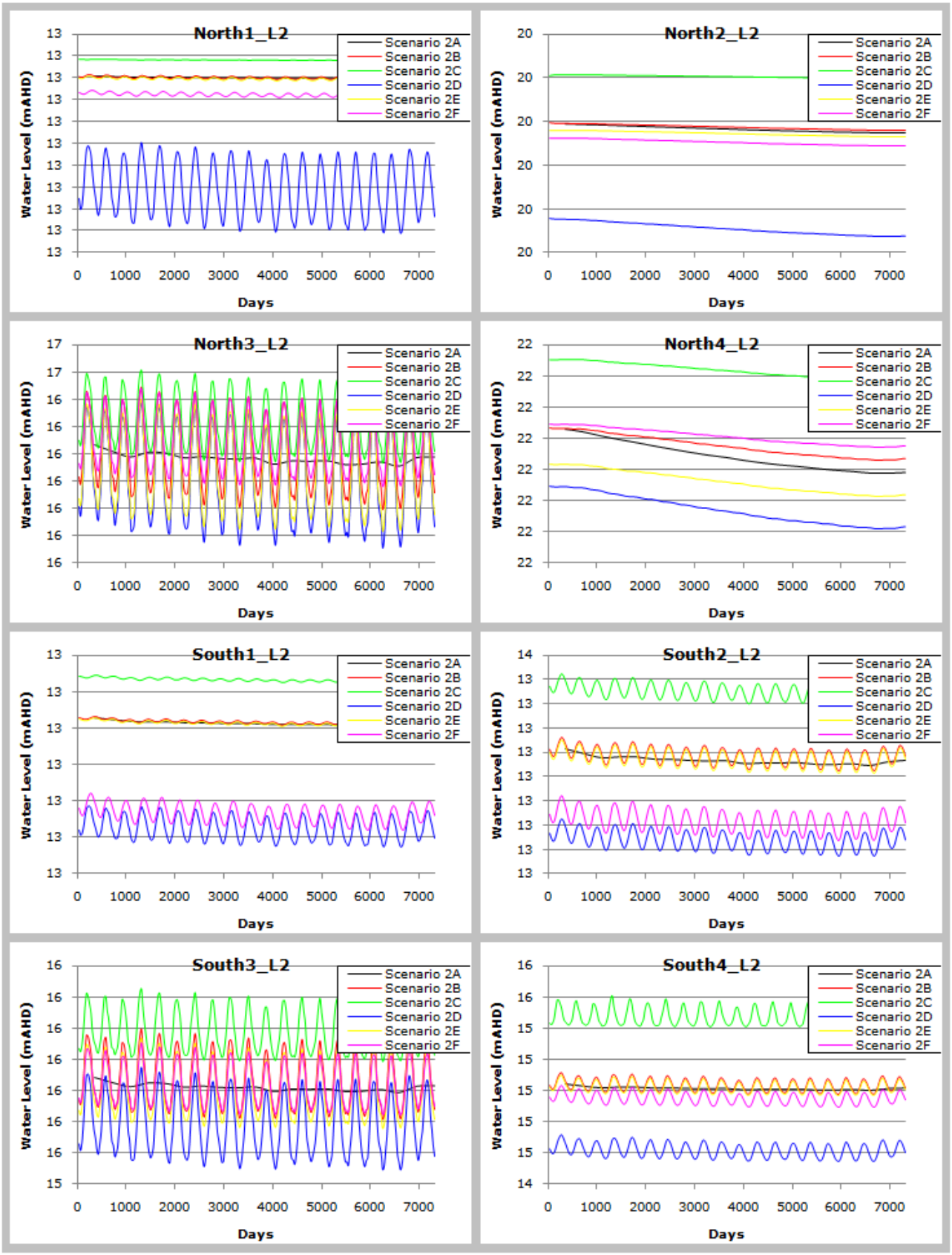
Appendix C15: Model results for Scenario 2 – Model C – Locked

The following figures present the transient information of model behaviour for the scenario in the title. These simulations feature different representations of the evapotranspiration function and the effects in the oservation bores. Details of the specific functions are given below

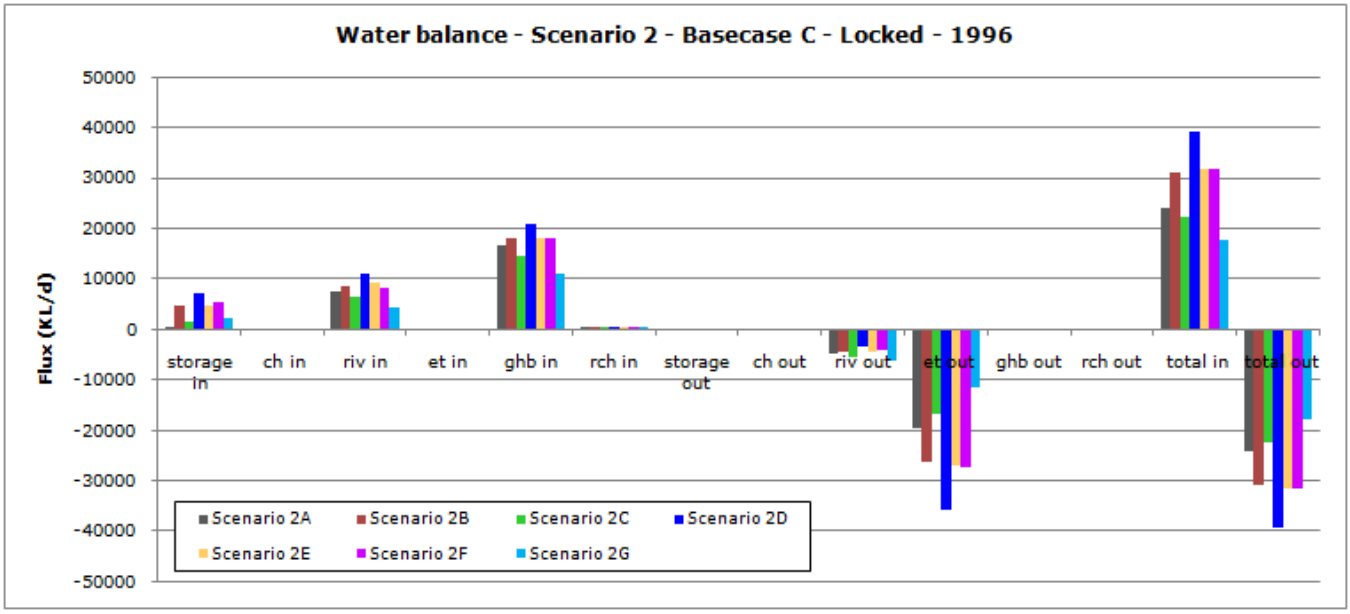
- Evapotranspiration varying yearly (Scenario 2A)
- Evapotranspiration varying monthly (Scenario 2B)
- Evapotranspiration varying monthly with a 0.5 m reduction in extinction depth (Scenario 2C)
- Evapotranspiration varying monthly with a 0.5 m increase in extinction depth (Scenario 2D)
- Spatial varying evapotranspiration rate including monthly variation (Scenario 2E)
- Spatial varying evapotranspiration rate and extinction depth, including monthly variation (Scenario 2F)
- Evapotranspiration varying monthly, using a non-linear evapotranspiration versus depth function, ETS curve (Scenario 2G)

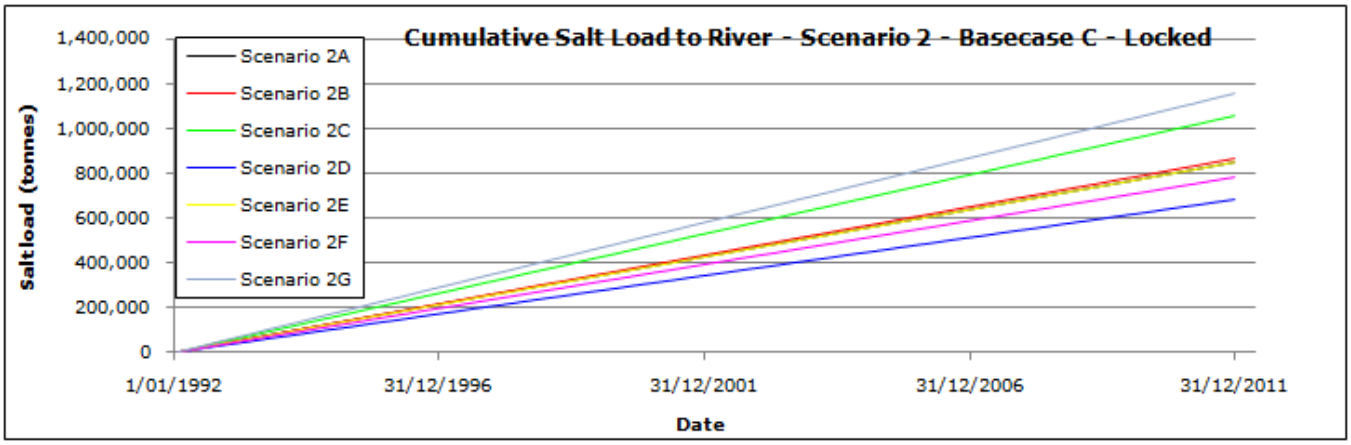


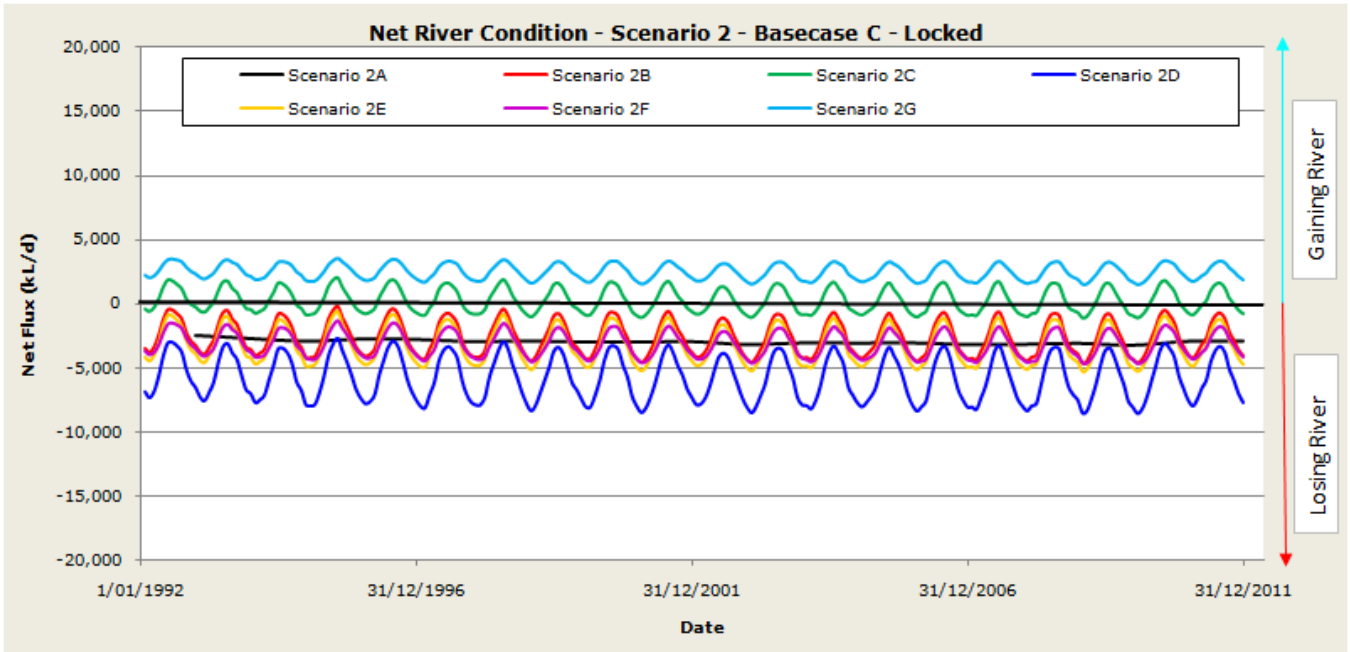
SCENARIO 2 GOYDER MODEL C - LOCKED



SCENARIO 2 GOYDER MODEL C - LOCKED


Water balance - Scenario 2 - Basecase C - Locked - 1996
Flux (KL/d)
50000
40000
30000
20000
10000
0
-10000
-20000
-30000
-40000
-50000
storage in
ch in
riv in
et in
ghb in
rch in
storage out
ch out
riv out
et out
ghb out
rch out
total in
total out
Scenario 2A
Scenario 2B
Scenario 2C
Scenario 2D
Scenario 2E
Scenario 2F
Scenario 2G

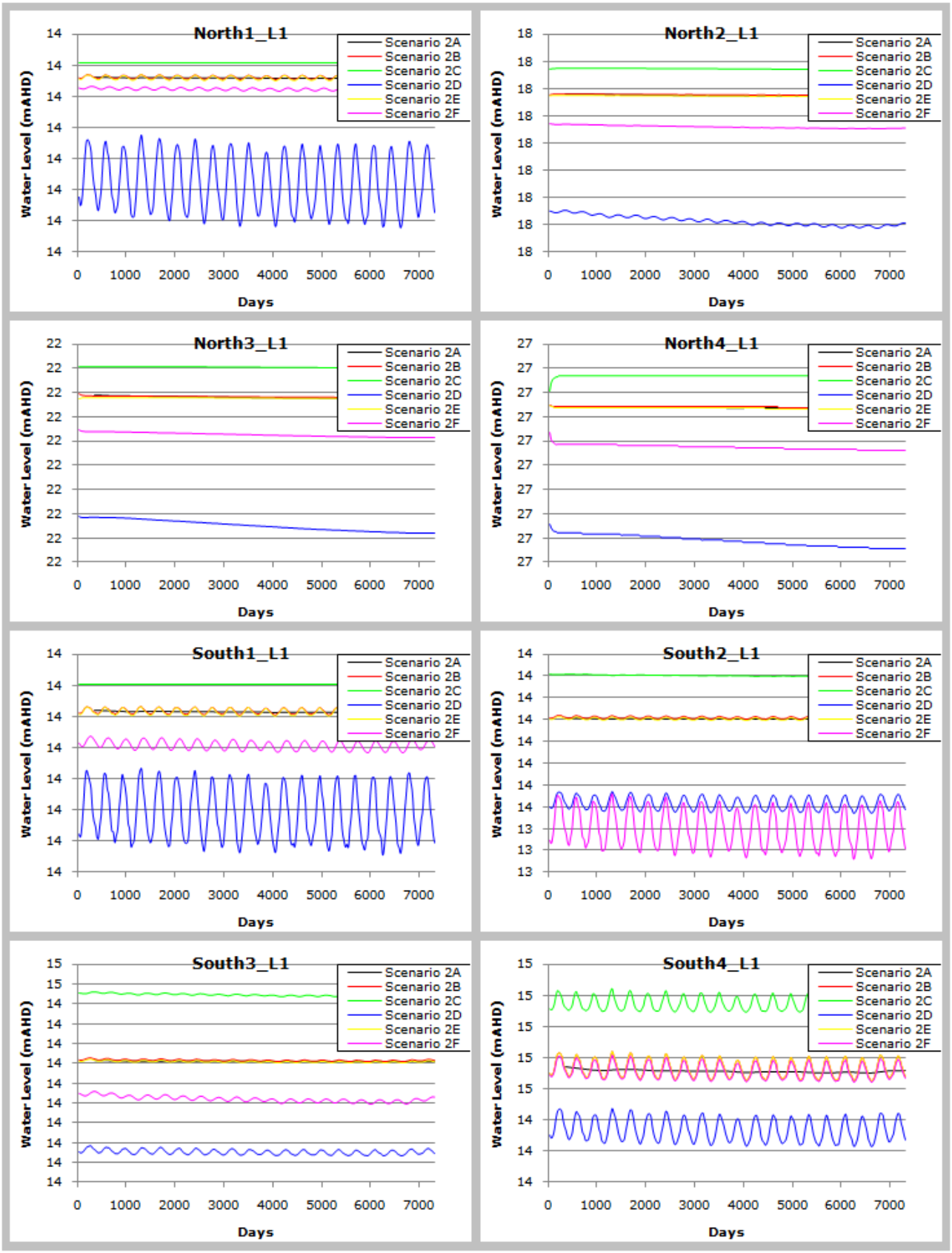

Cumulative Salt Load to River - Scenario 2 - Basecase C - Locked
Salt load (tonnes)
1,400,000
1,200,000
1,000,000
800,000
600,000
400,000
200,000
0
1/01/1992
31/12/1996
31/12/2001
31/12/2006
31/12/2011
Date
Scenario 2A
Scenario 2B
Scenario 2C
Scenario 2D
Scenario 2E
Scenario 2F
Scenario 2G


Net River Condition - Scenario 2 - Basecase C - Locked
Net Flux (kL/d)
20,000
15,000
10,000
5,000
0
-5,000
-10,000
-15,000
-20,000
1/01/1992
31/12/1996
31/12/2001
31/12/2006
31/12/2011
Date
Scenario 2A
Scenario 2B
Scenario 2C
Scenario 2D
Scenario 2E
Scenario 2F
Scenario 2G
Gaining River
Losing River

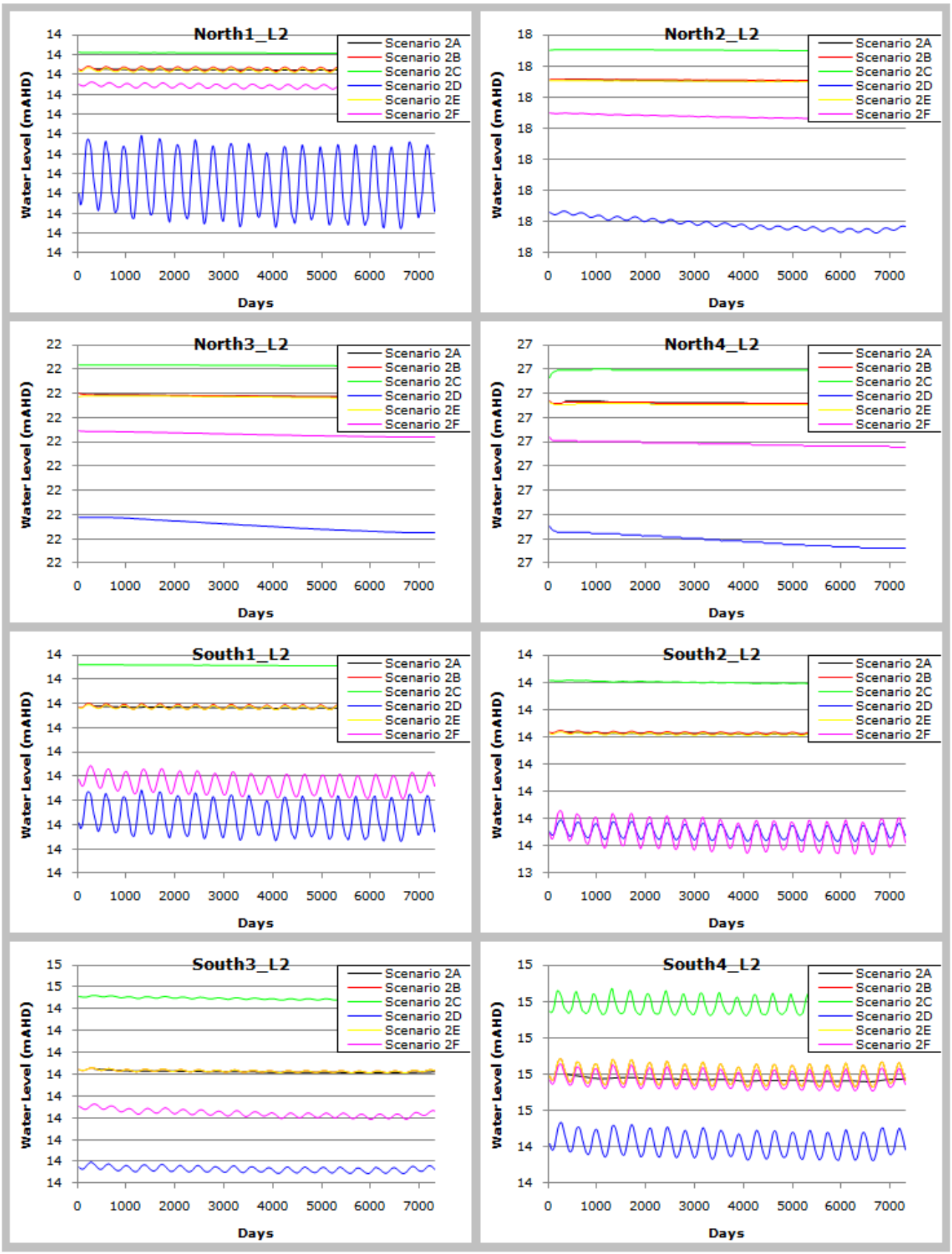
Appendix C16: Model results for Scenario 2 – Model C – Not locked

The following figures present the transient information of model behaviour for the scenario in the title. These simulations feature different representations of the evapotranspiration function and the effects in the oservation bores. Details of the specific functions are given below

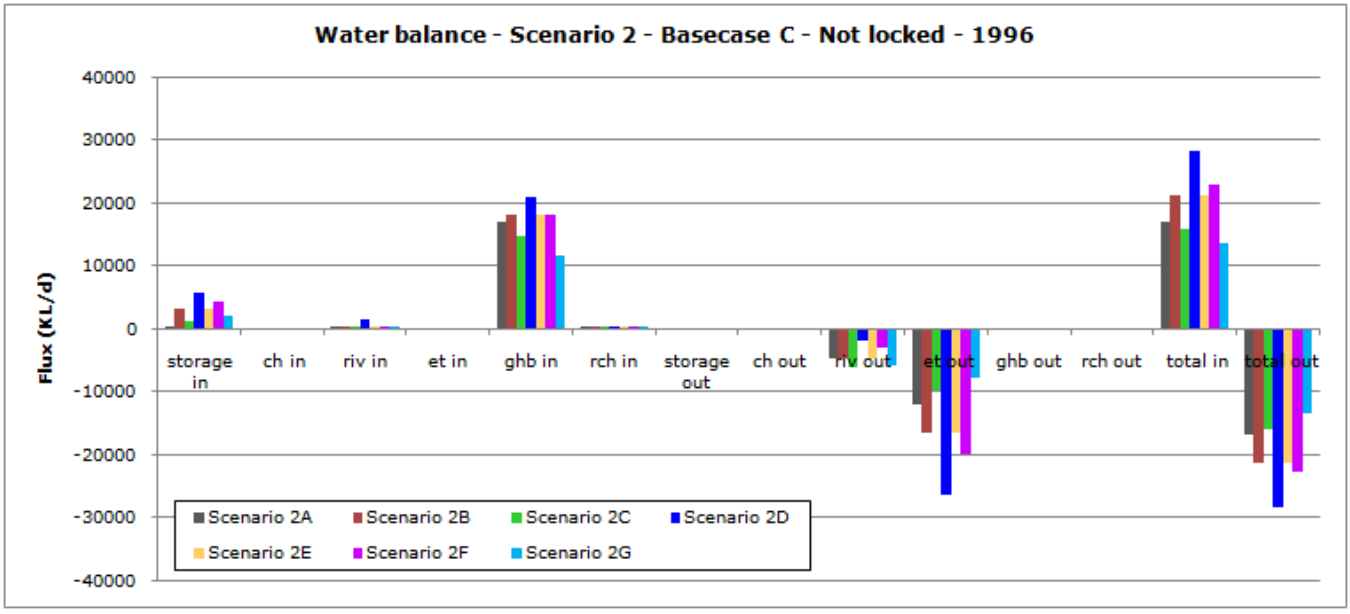
- Evapotranspiration varying yearly (Scenario 2A)
- Evapotranspiration varying monthly (Scenario 2B)
- Evapotranspiration varying monthly with a 0.5 m reduction in extinction depth (Scenario 2C)
- Evapotranspiration varying monthly with a 0.5 m increase in extinction depth (Scenario 2D)
- Spatial varying evapotranspiration rate including monthly variation (Scenario 2E)
- Spatial varying evapotranspiration rate and extinction depth, including monthly variation (Scenario 2F)
- Evapotranspiration varying monthly, using a non-linear evapotranspiration versus depth function, ETS curve (Scenario 2G)

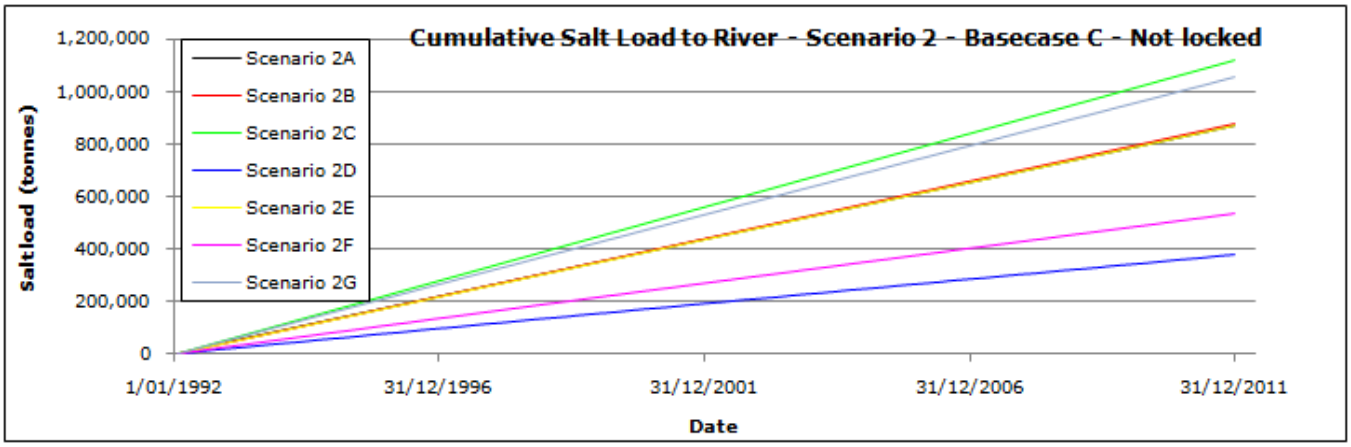


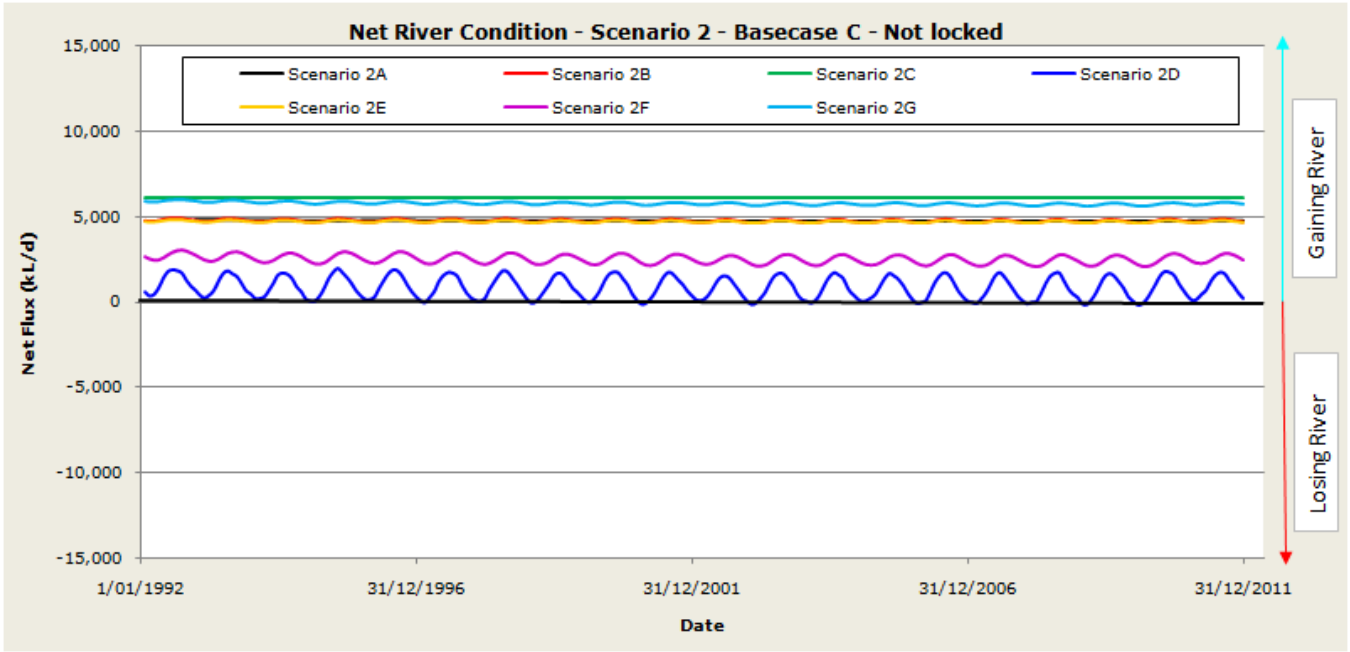
SCENARIO 2 - GOYDER MODEL C - NOT LOCKED



SCENARIO 2 - GOYDER MODEL C - NOT LOCKED


Water balance - Scenario 2 - Basecase C - Not locked - 1996
Flux (KL/d)
40000
30000
20000
10000
0
-10000
-20000
-30000
-40000
storage in
ch in
riv in
et in
ghb in
rch in
storage out
ch out
riv out
et out
ghb out
rch out
total in
total out
Scenario 2A
Scenario 2B
Scenario 2C
Scenario 2D
Scenario 2E
Scenario 2F
Scenario 2G

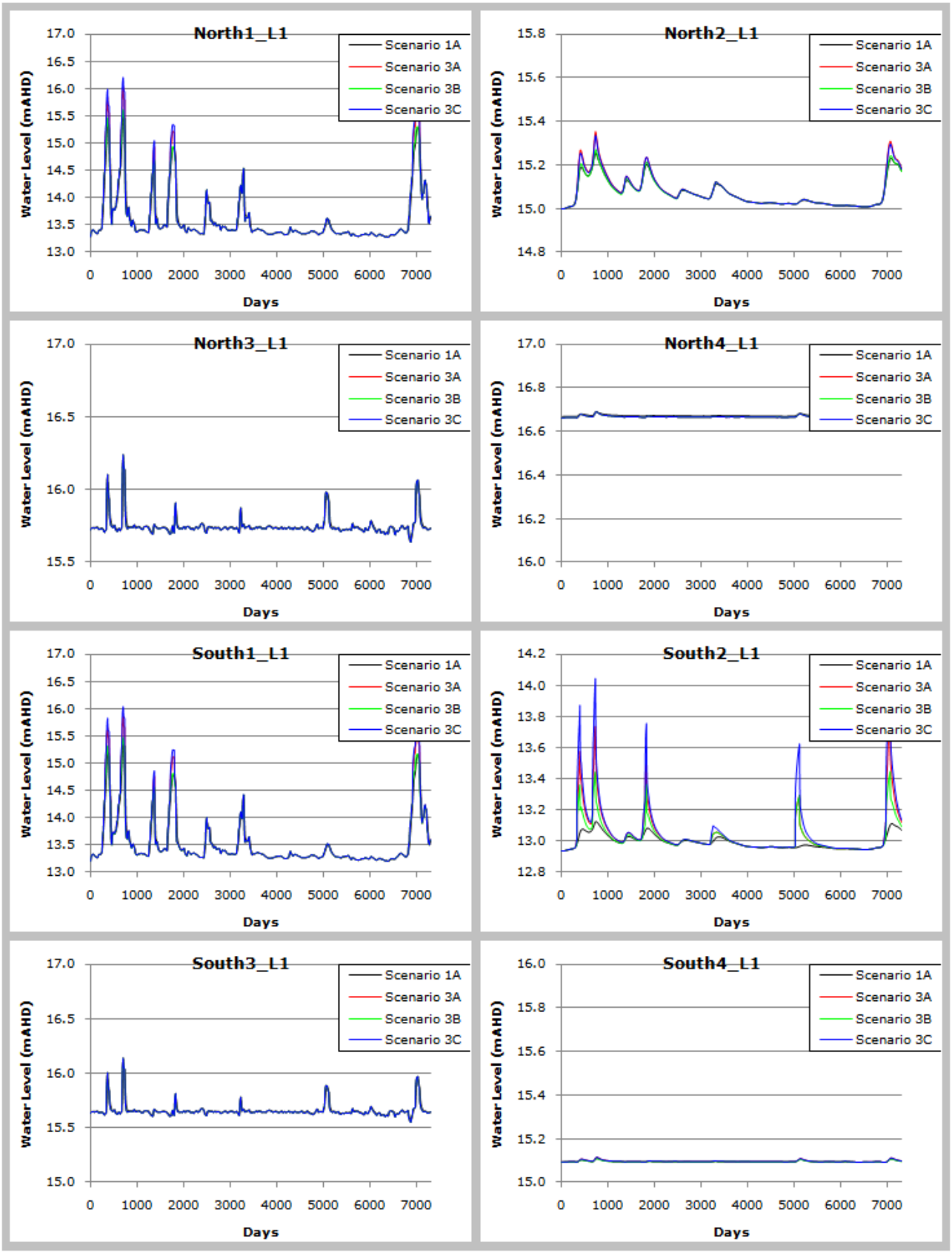

Cumulative Salt Load to River - Scenario 2 - Basecase C - Not locked
Salt load (tonnes)
1,200,000
1,000,000
800,000
600,000
400,000
200,000
0
Scenario 2A
Scenario 2B
Scenario 2C
Scenario 2D
Scenario 2E
Scenario 2F
Scenario 2G
1/01/1992
31/12/1996
31/12/2001
31/12/2006
31/12/2011
Date


Net River Condition - Scenario 2 - Basecase C - Not locked
Net Flux (kL/d)
15,000
10,000
5,000
0
-5,000
-10,000
-15,000
Scenario 2A
Scenario 2B
Scenario 2C
Scenario 2D
Scenario 2E
Scenario 2F
Scenario 2G
Gaining River
Losing River
1/01/1992
31/12/1996
31/12/2001
31/12/2006
31/12/2011
Date

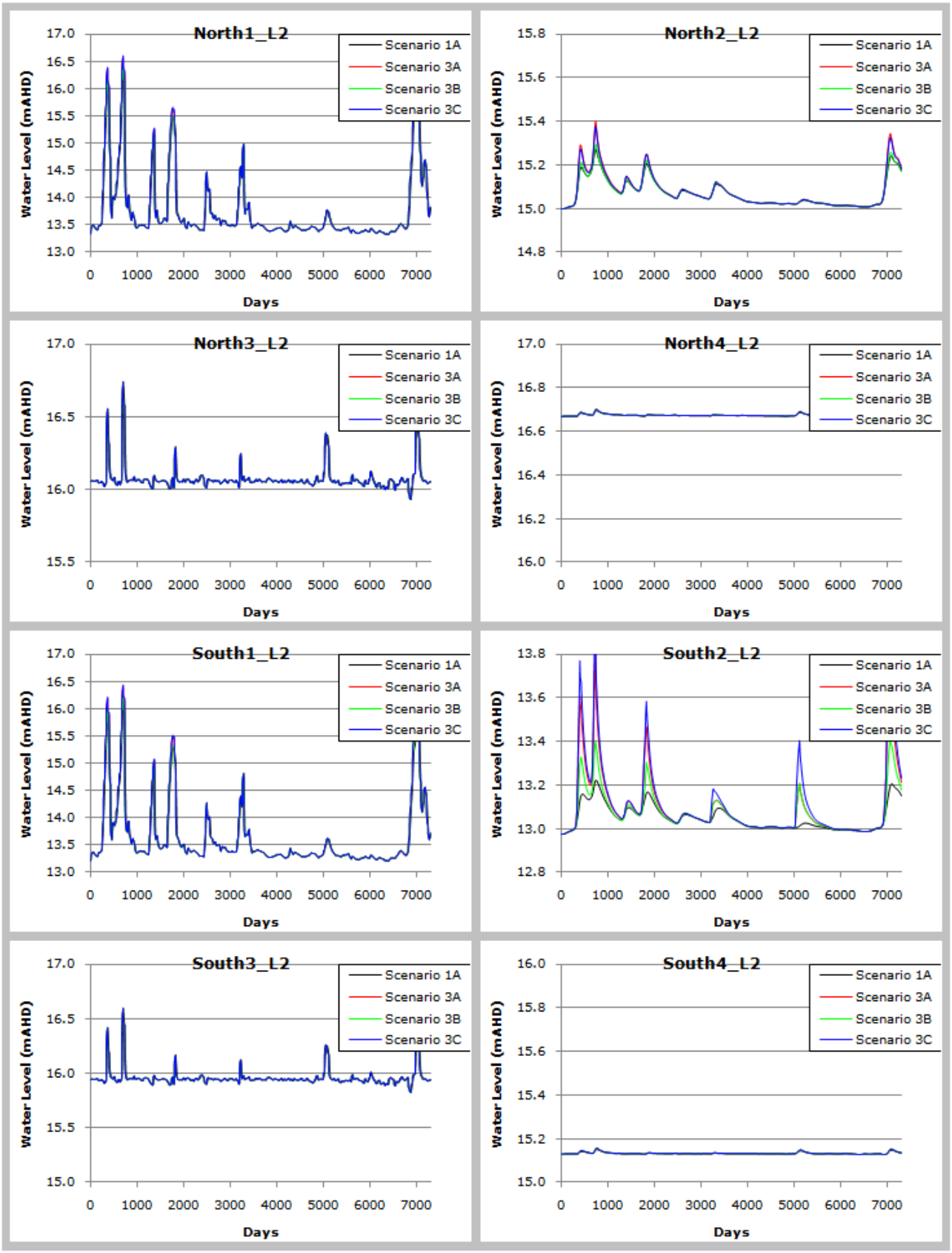
Appendix C17: Model results from Scenario 3 – Model A – Locked

The following figures present the transient information of model behaviour for the scenario in the title. These simulations feature flood inundation that matches the monthly stress period setup. The flood innundation is simulated via increased recharge at times when the river stage is above elevation. The various scenarios are described below.

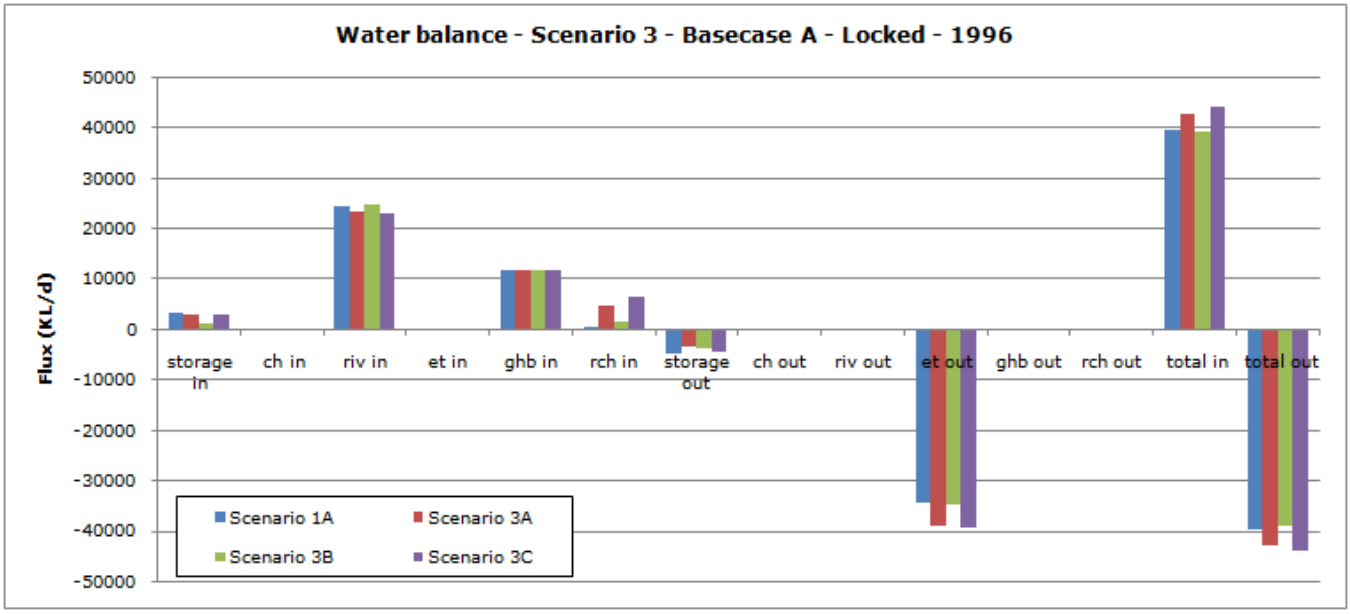
Scenario name	Stress Periods	Inundation Recharge Rate (mm/day)	Area inundated	Spatial variation
Scenario 3A	Monthly	1	Floodplain + Wetland	No
Scenario 3B	Monthly	1	Wetland only	No
Scenario 3C	Monthly	0.5 - 2	Floodplain + Wetland	Yes

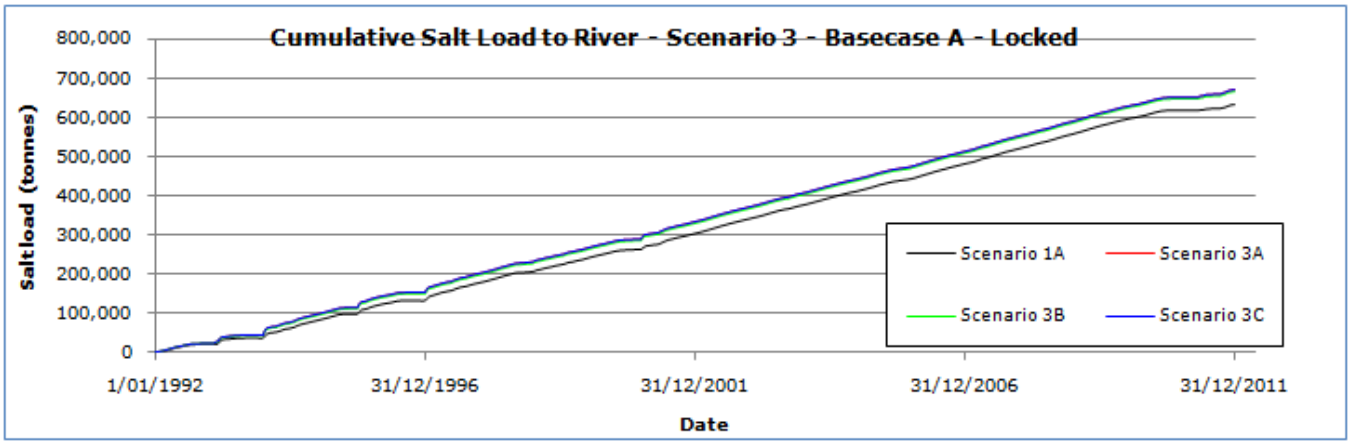


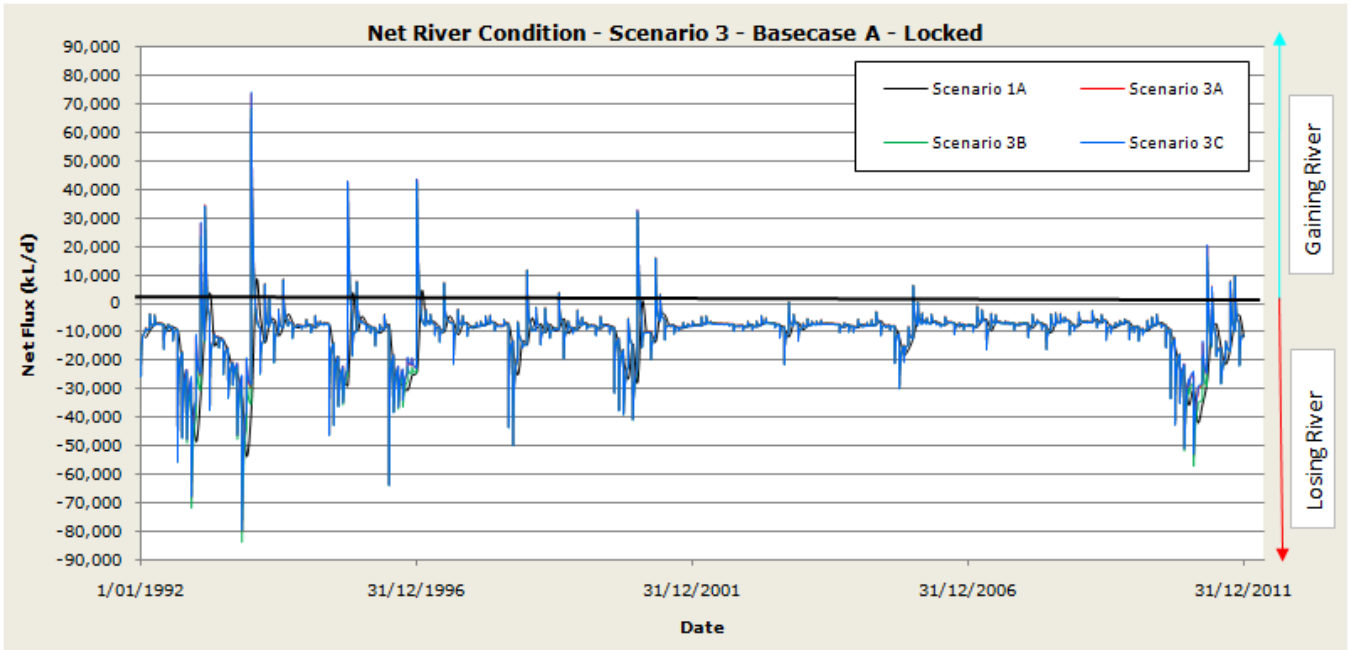
SCENARIO 3 - GOYDER MODEL A - LOCKED



SCENARIO 3 - GOYDER MODEL A - LOCKED


Water balance - Scenario 3 - Basecase A - Locked - 1996
Flux (KL/d)
50000
40000
30000
20000
10000
0
-10000
-20000
-30000
-40000
-50000
storage in
ch in
riv in
et in
ghb in
rch in
storage out
ch out
riv out
et out
ghb out
rch out
total in
total out
Scenario 1A
Scenario 3A
Scenario 3B
Scenario 3C

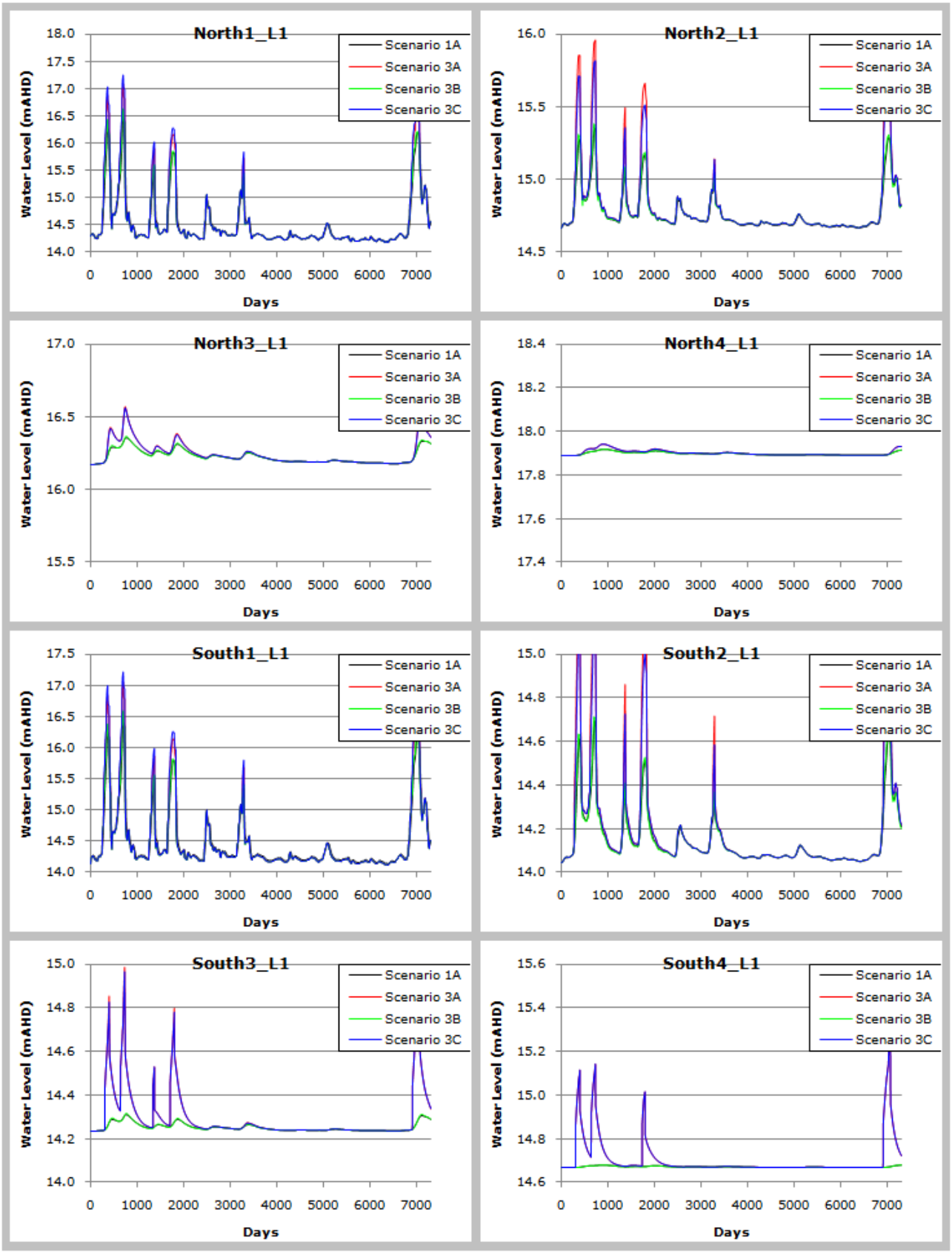

Cumulative Salt Load to River - Scenario 3 - Basecase A - Locked
Salt load (tonnes)
800,000
700,000
600,000
500,000
400,000
300,000
200,000
100,000
0
1/01/1992
31/12/1996
31/12/2001
31/12/2006
31/12/2011
Date
Scenario 1A
Scenario 3A
Scenario 3B
Scenario 3C


Net River Condition - Scenario 3 - Basecase A - Locked
Net Flux (kL/d)
90,000
80,000
70,000
60,000
50,000
40,000
30,000
20,000
10,000
0
-10,000
-20,000
-30,000
-40,000
-50,000
-60,000
-70,000
-80,000
-90,000
1/01/1992
31/12/1996
31/12/2001
31/12/2006
31/12/2011
Date
Scenario 1A
Scenario 3A
Scenario 3B
Scenario 3C
Gaining River
Losing River

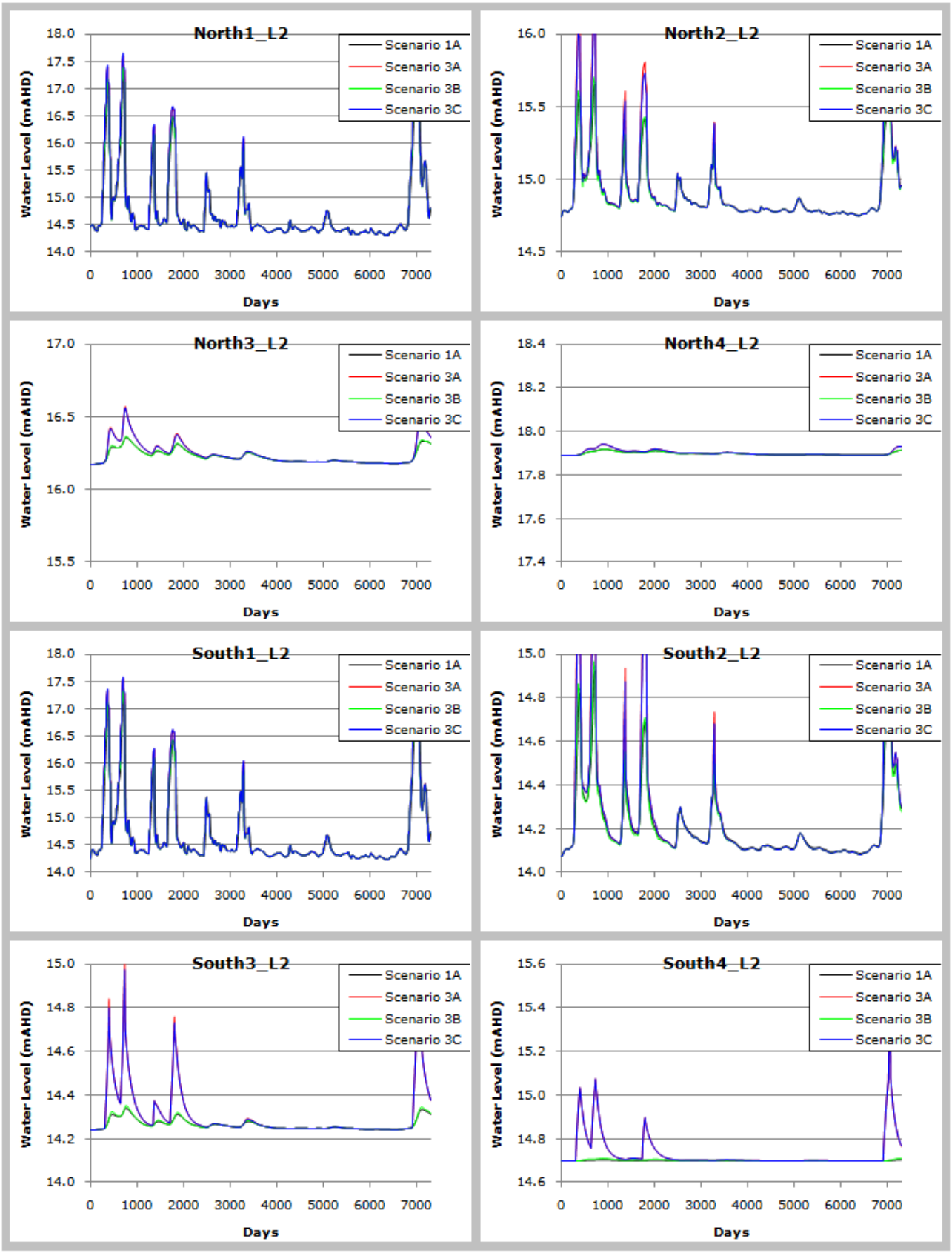
Appendix C18: Model results for Scenario 3 – Model A – Not locked

The following figures present the transient information of model behaviour for the scenario in the title. These simulations feature flood inundation that matches the monthly stress period setup. The flood innundation is simulated via increased recharge at times when the river stage is above elevation. The various scenarios are described below.

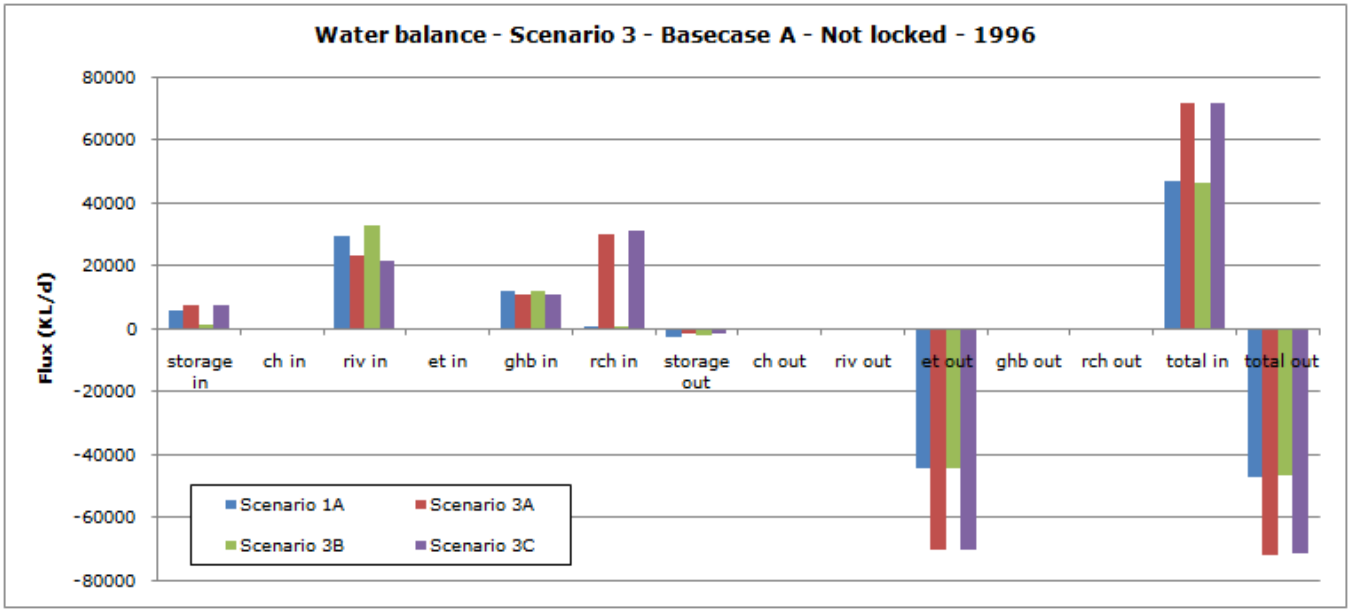
Scenario name	Stress Periods	Inundation Recharge Rate (mm/day)	Area inundated	Spatial variation
Scenario 3A	Monthly	1	Floodplain + Wetland	No
Scenario 3B	Monthly	1	Wetland only	No
Scenario 3C	Monthly	0.5 - 2	Floodplain + Wetland	Yes

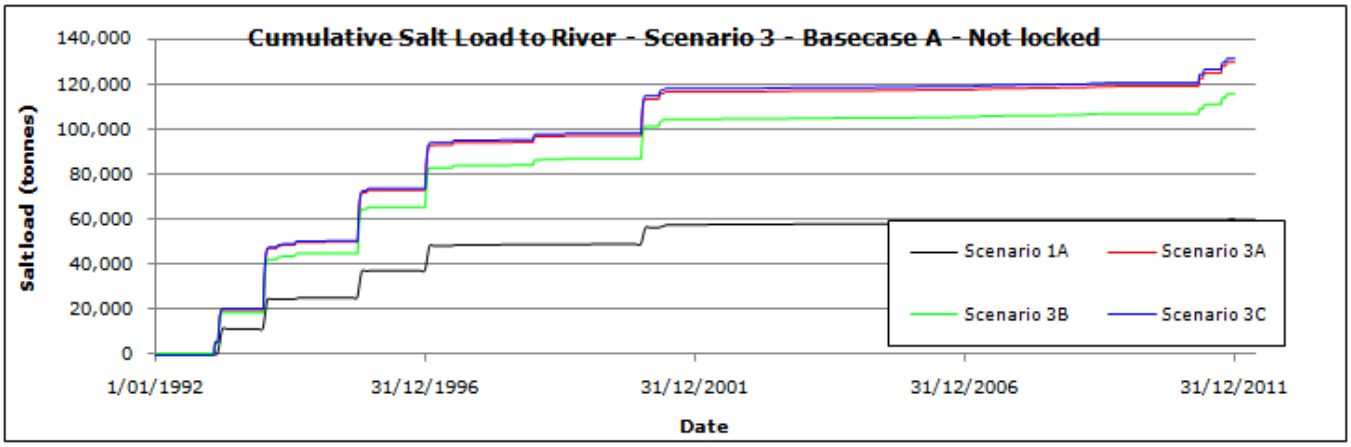


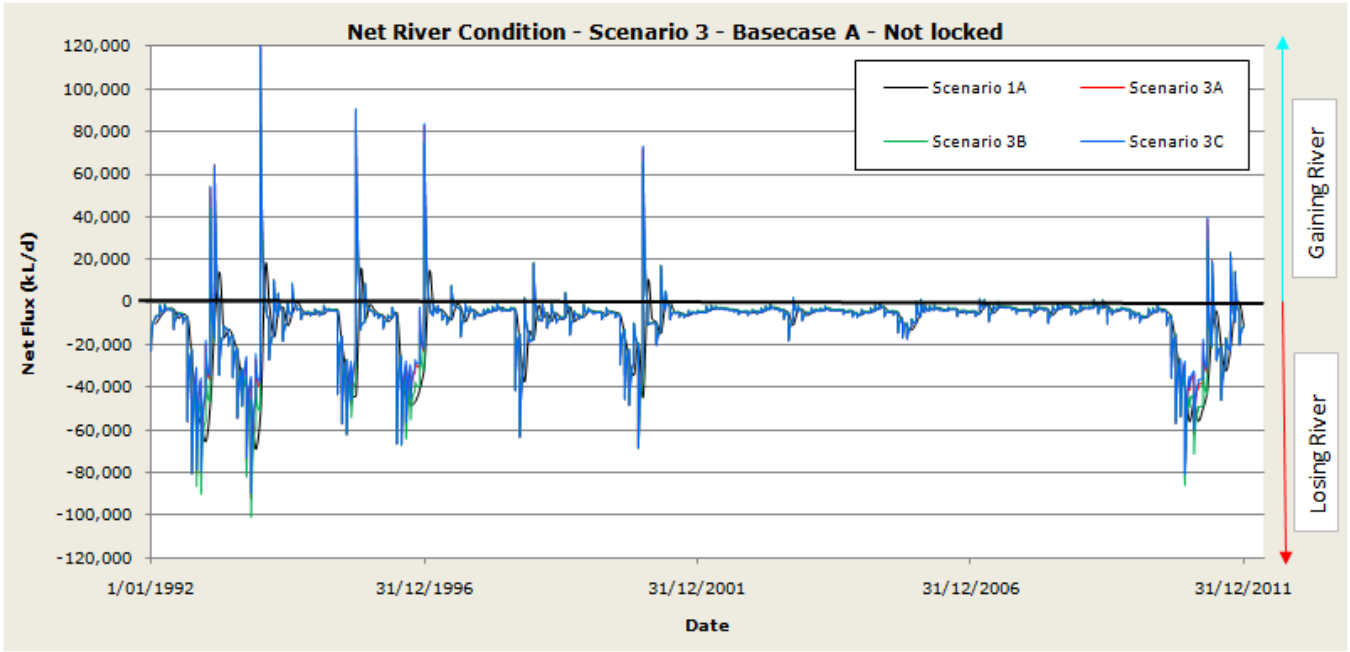
SCENARIO 3 - GOYDER MODEL A - NOT LOCKED



SCENARIO 3 - GOYDER MODEL A - NOT LOCKED


Water balance - Scenario 3 - Basecase A - Not locked - 1996
Flux (KL/d)
80000
60000
40000
20000
0
-20000
-40000
-60000
-80000
storage in
ch in
riv in
et in
ghb in
rch in
storage out
ch out
riv out
et out
ghb out
rch out
total in
total out
Scenario 1A
Scenario 3A
Scenario 3B
Scenario 3C

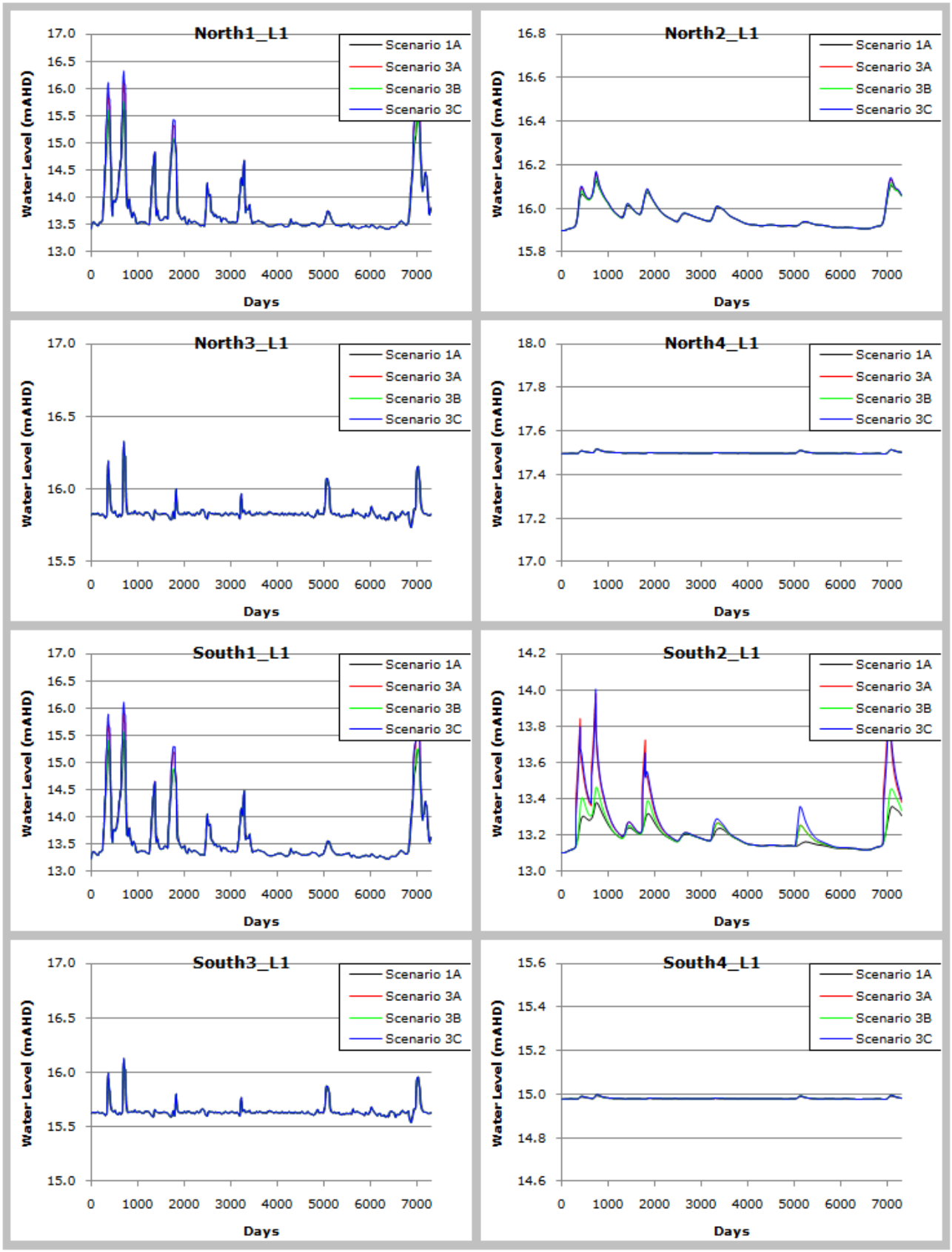

Cumulative Salt Load to River - Scenario 3 - Basecase A - Not locked
Salt load (tonnes)
140,000
120,000
100,000
80,000
60,000
40,000
20,000
0
1/01/1992
31/12/1996
31/12/2001
31/12/2006
31/12/2011
Date
Scenario 1A
Scenario 3A
Scenario 3B
Scenario 3C


Net River Condition - Scenario 3 - Basecase A - Not locked
Net Flux (kL/d)
120,000
100,000
80,000
60,000
40,000
20,000
0
-20,000
-40,000
-60,000
-80,000
-100,000
-120,000
1/01/1992
31/12/1996
31/12/2001
31/12/2006
31/12/2011
Date
Scenario 1A
Scenario 3A
Scenario 3B
Scenario 3C
Gaining River
Losing River

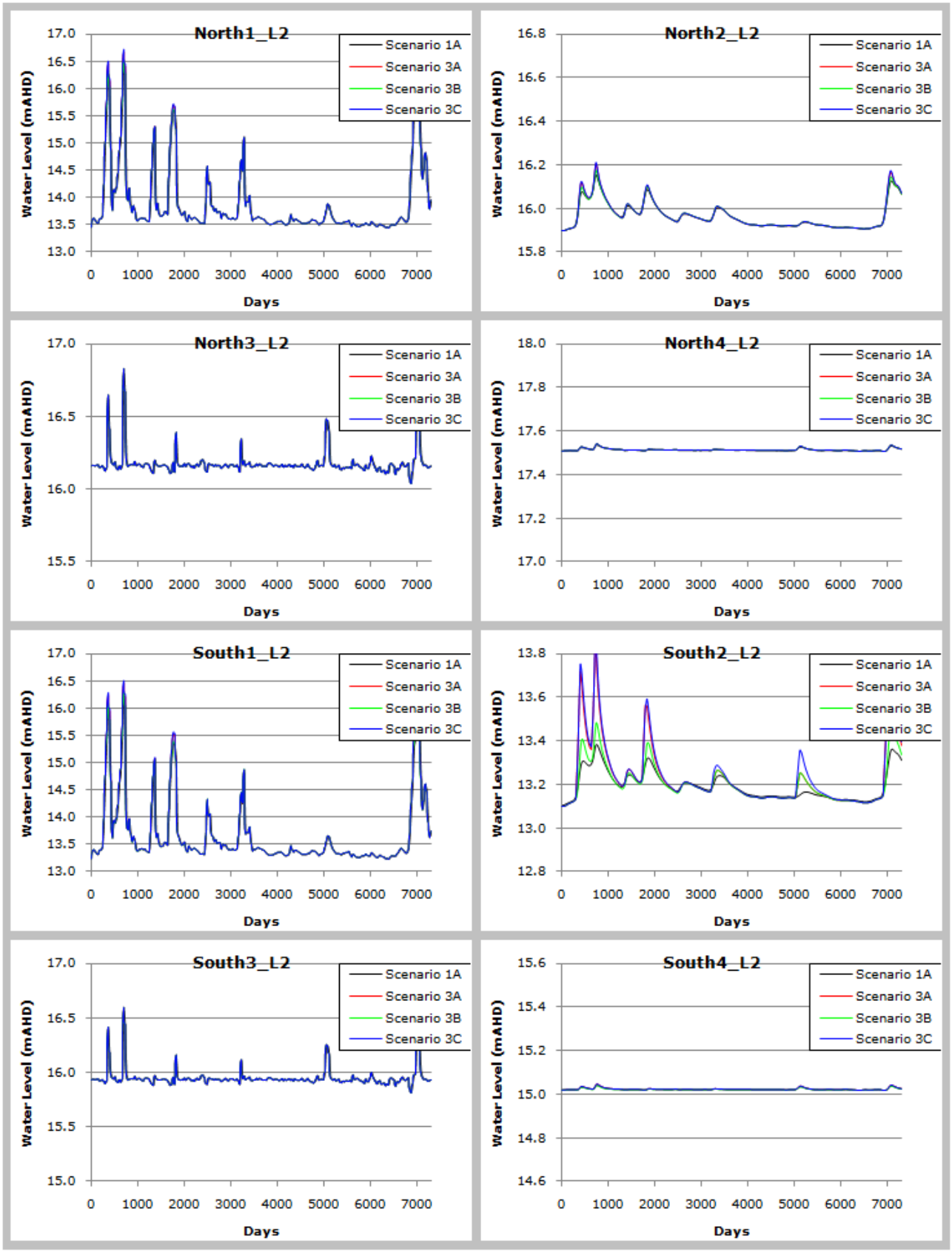
Appendix C19: Model results for Scenario 3 – Model B – Locked

The following figures present the transient information of model behaviour for the scenario in the title. These simulations feature flood inundation that matches the monthly stress period setup. The flood innundation is simulated via increased recharge at times when the river stage is above elevation. The various scenarios are described below.

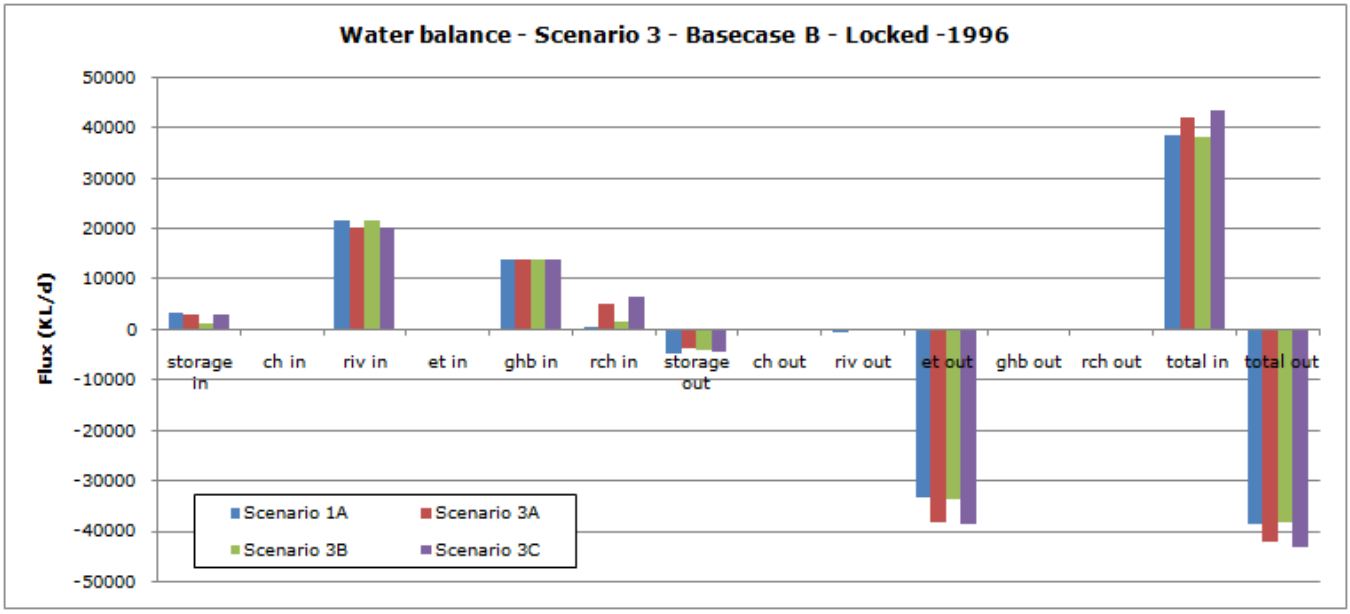
Scenario name	Stress Periods	Inundation Recharge Rate (mm/day)	Area inundated	Spatial variation
Scenario 3A	Monthly	1	Floodplain + Wetland	No
Scenario 3B	Monthly	1	Wetland only	No
Scenario 3C	Monthly	0.5 - 2	Floodplain + Wetland	Yes

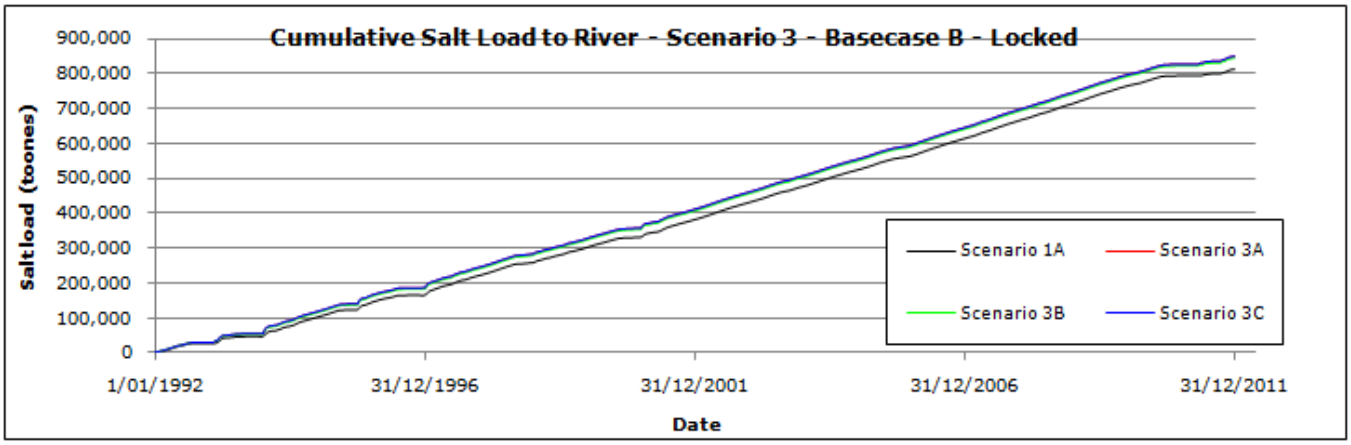


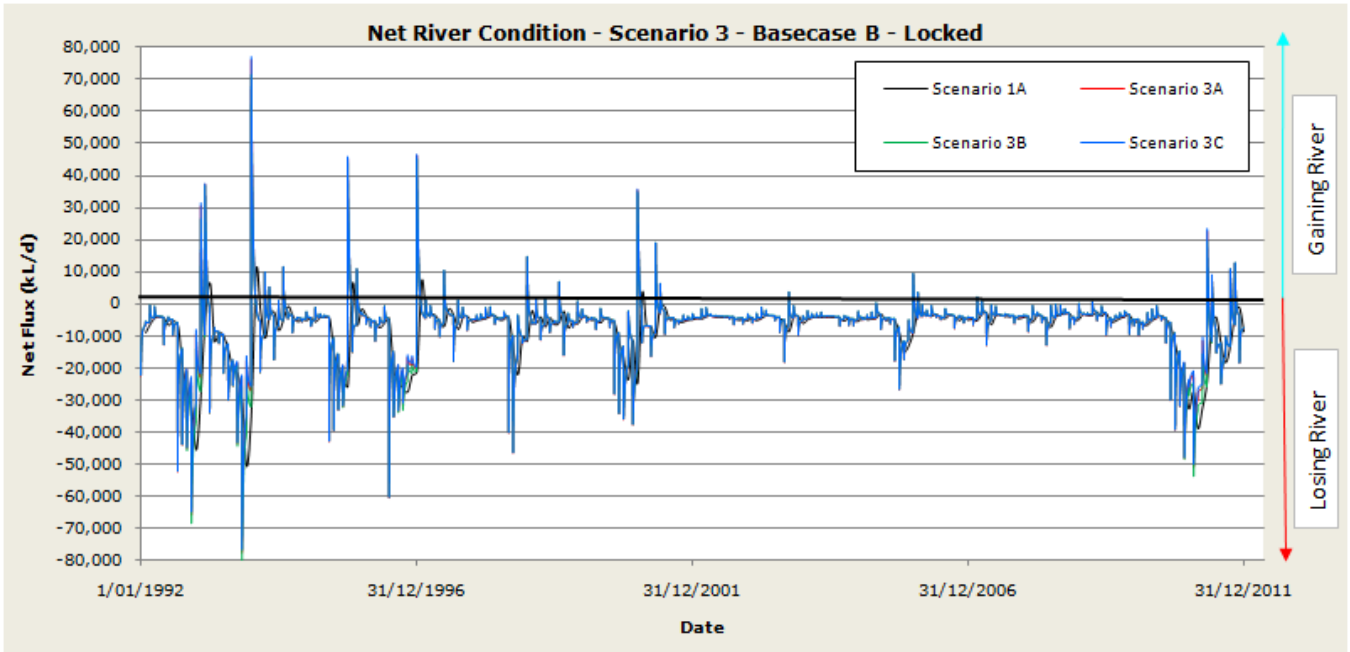
SCENARIO 3 - GOYDER MODEL B - LOCKED



SCENARIO 3 - GOYDER MODEL B - LOCKED


Water balance - Scenario 3 - Basecase B - Locked -1996
Flux (KL/d)
50000
40000
30000
20000
10000
0
-10000
-20000
-30000
-40000
-50000
storage in
ch in
riv in
et in
ghb in
rch in
storage out
ch out
riv out
et out
ghb out
rch out
total in
total out
Scenario 1A
Scenario 3A
Scenario 3B
Scenario 3C

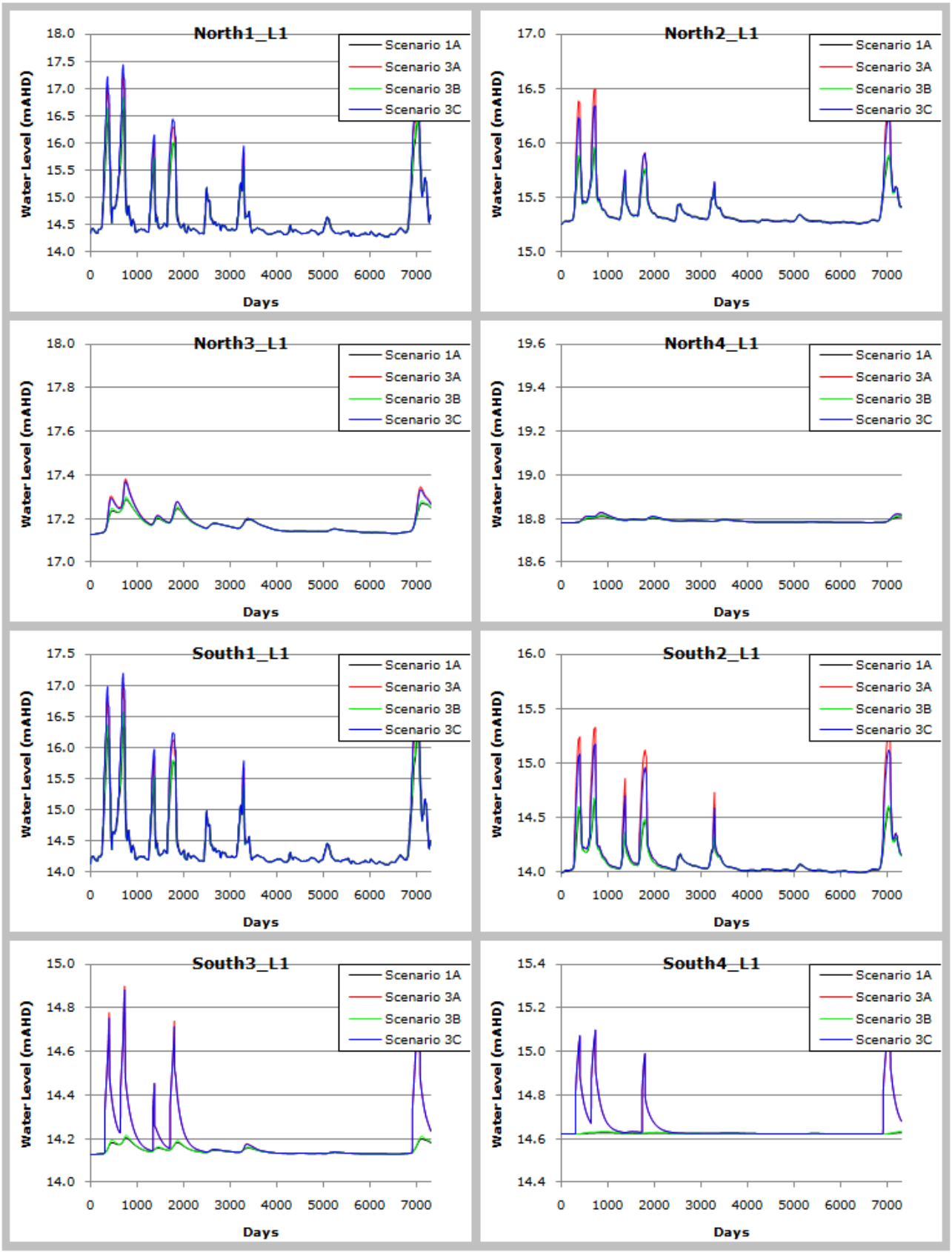

Cumulative Salt Load to River - Scenario 3 - Basecase B - Locked
Salt load (toones)
900,000
800,000
700,000
600,000
500,000
400,000
300,000
200,000
100,000
0
1/01/1992
31/12/1996
31/12/2001
31/12/2006
31/12/2011
Date
Scenario 1A
Scenario 3A
Scenario 3B
Scenario 3C


Net River Condition - Scenario 3 - Basecase B - Locked
Net Flux (kL/d)
80,000
70,000
60,000
50,000
40,000
30,000
20,000
10,000
0
-10,000
-20,000
-30,000
-40,000
-50,000
-60,000
-70,000
-80,000
1/01/1992
31/12/1996
31/12/2001
31/12/2006
31/12/2011
Date
Scenario 1A
Scenario 3A
Scenario 3B
Scenario 3C
Gaining River
Losing River

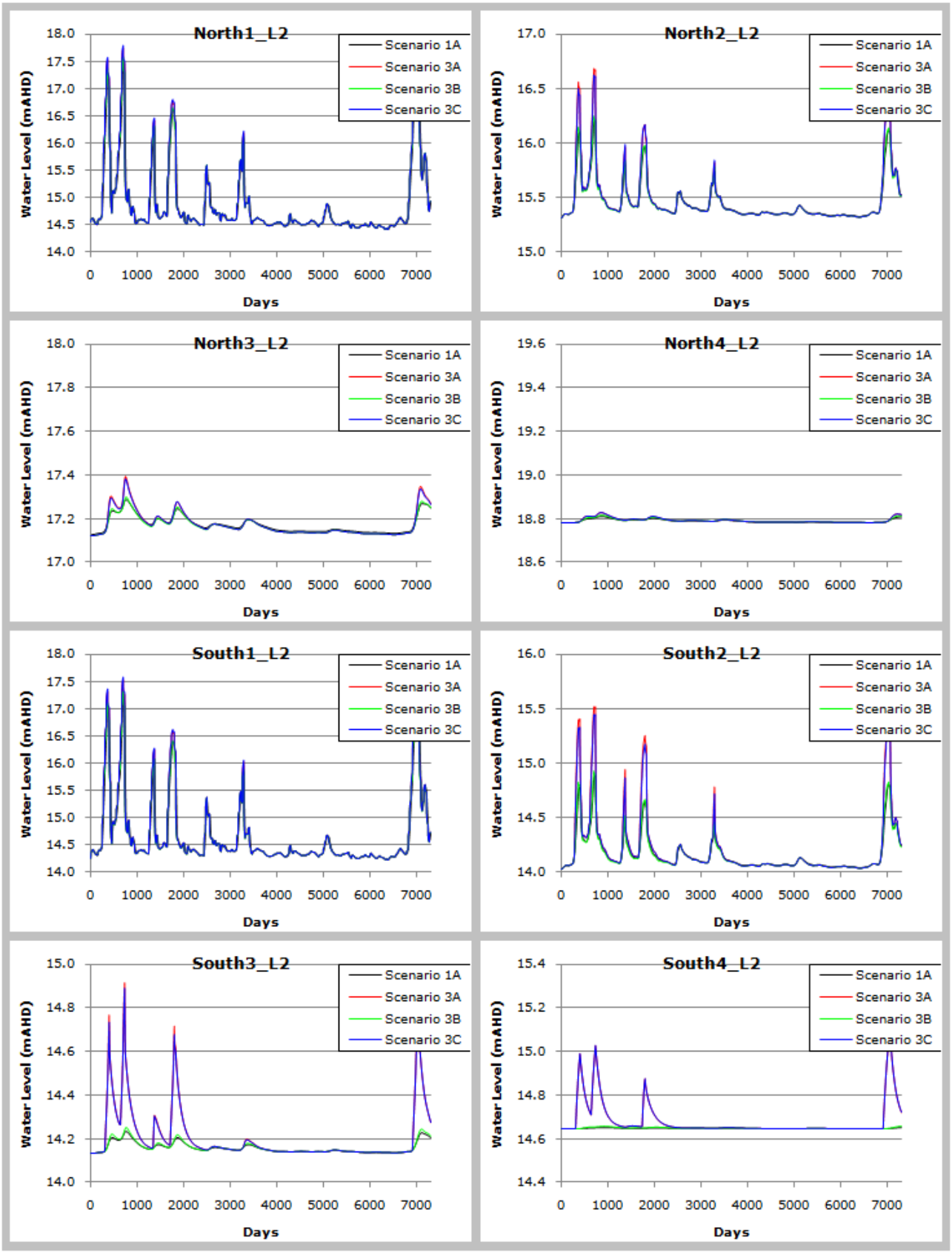
Appendix C20: Model results for Scenario 3 – Model B – Not locked

The following figures present the transient information of model behaviour for the scenario in the title. These simulations feature flood inundation that matches the monthly stress period setup. The flood innundation is simulated via increased recharge at times when the river stage is above elevation. The various scenarios are described below.

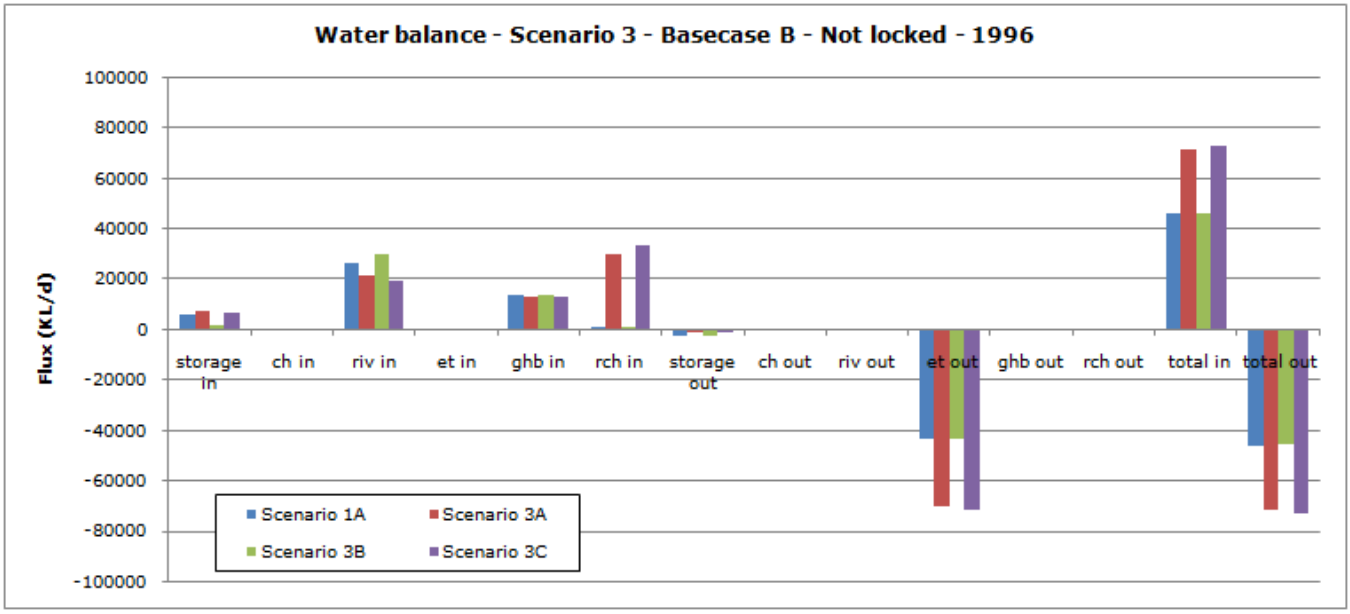
Scenario name	Stress Periods	Inundation Recharge Rate (mm/day)	Area inundated	Spatial variation
Scenario 3A	Monthly	1	Floodplain + Wetland	No
Scenario 3B	Monthly	1	Wetland only	No
Scenario 3C	Monthly	0.5 - 2	Floodplain + Wetland	Yes

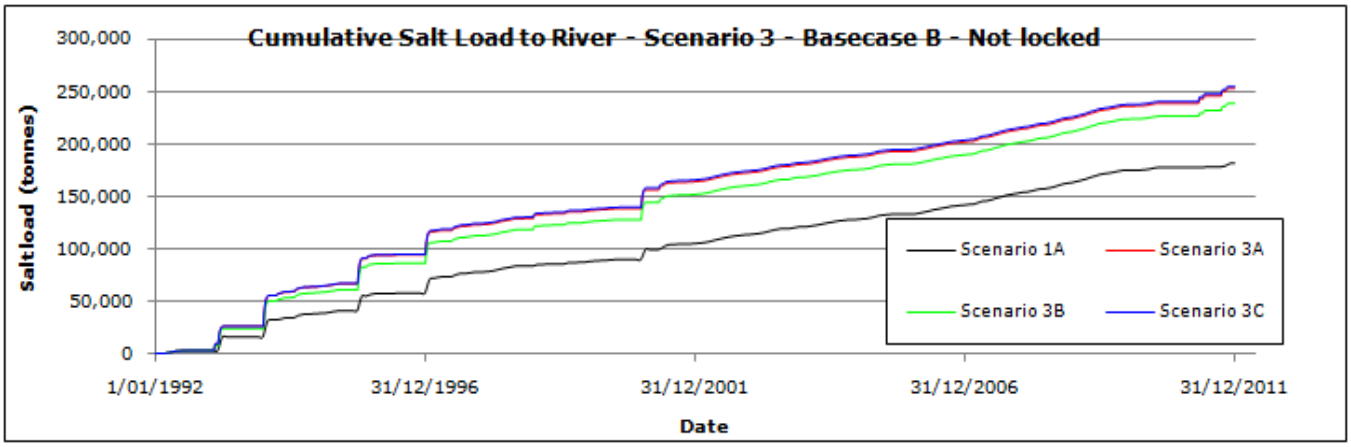


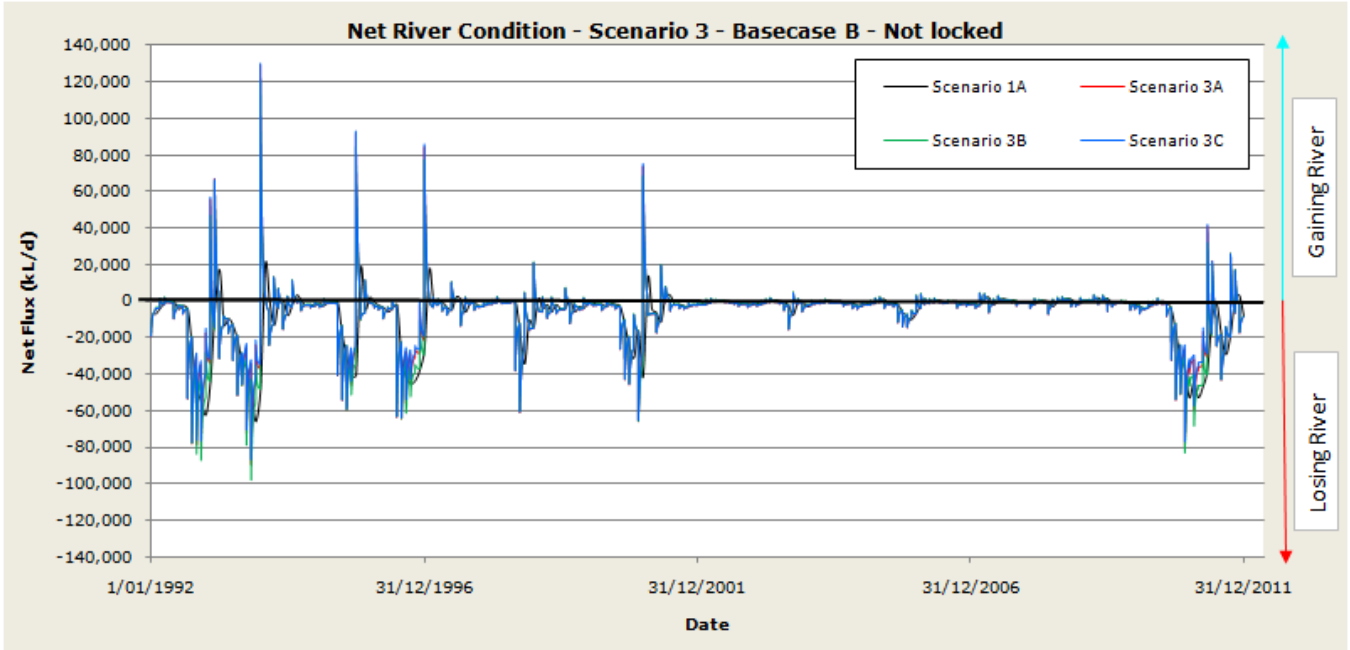
SCENARIO 3 - GOYDER MODEL B - NOT LOCKED



SCENARIO 3 - GOYDER MODEL B - NOT LOCKED


Water balance - Scenario 3 - Basecase B - Not locked - 1996
Flux (KL/d)
100000
80000
60000
40000
20000
0
-20000
-40000
-60000
-80000
-100000
storage in
ch in
riv in
et in
ghb in
rch in
storage out
ch out
riv out
et out
ghb out
rch out
total in
total out
Scenario 1A
Scenario 3A
Scenario 3B
Scenario 3C

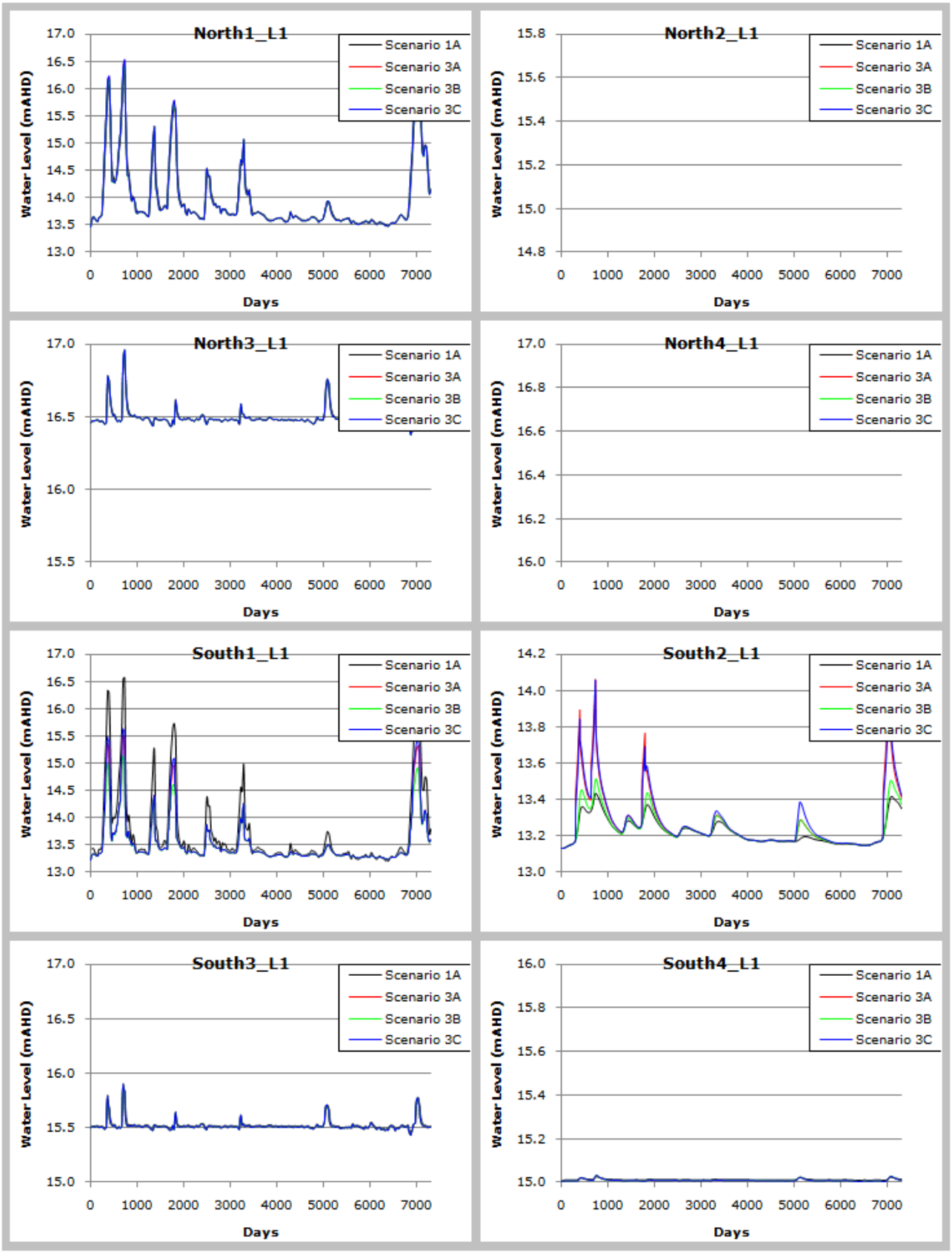

Cumulative Salt Load to River - Scenario 3 - Basecase B - Not locked
Salt load (tonnes)
300,000
250,000
200,000
150,000
100,000
50,000
0
1/01/1992
31/12/1996
31/12/2001
31/12/2006
31/12/2011
Date
Scenario 1A
Scenario 3A
Scenario 3B
Scenario 3C


Net River Condition - Scenario 3 - Basecase B - Not locked
Net Flux (kL/d)
140,000
120,000
100,000
80,000
60,000
40,000
20,000
0
-20,000
-40,000
-60,000
-80,000
-100,000
-120,000
-140,000
1/01/1992
31/12/1996
31/12/2001
31/12/2006
31/12/2011
Date
Scenario 1A
Scenario 3A
Scenario 3B
Scenario 3C
Gaining River
Losing River

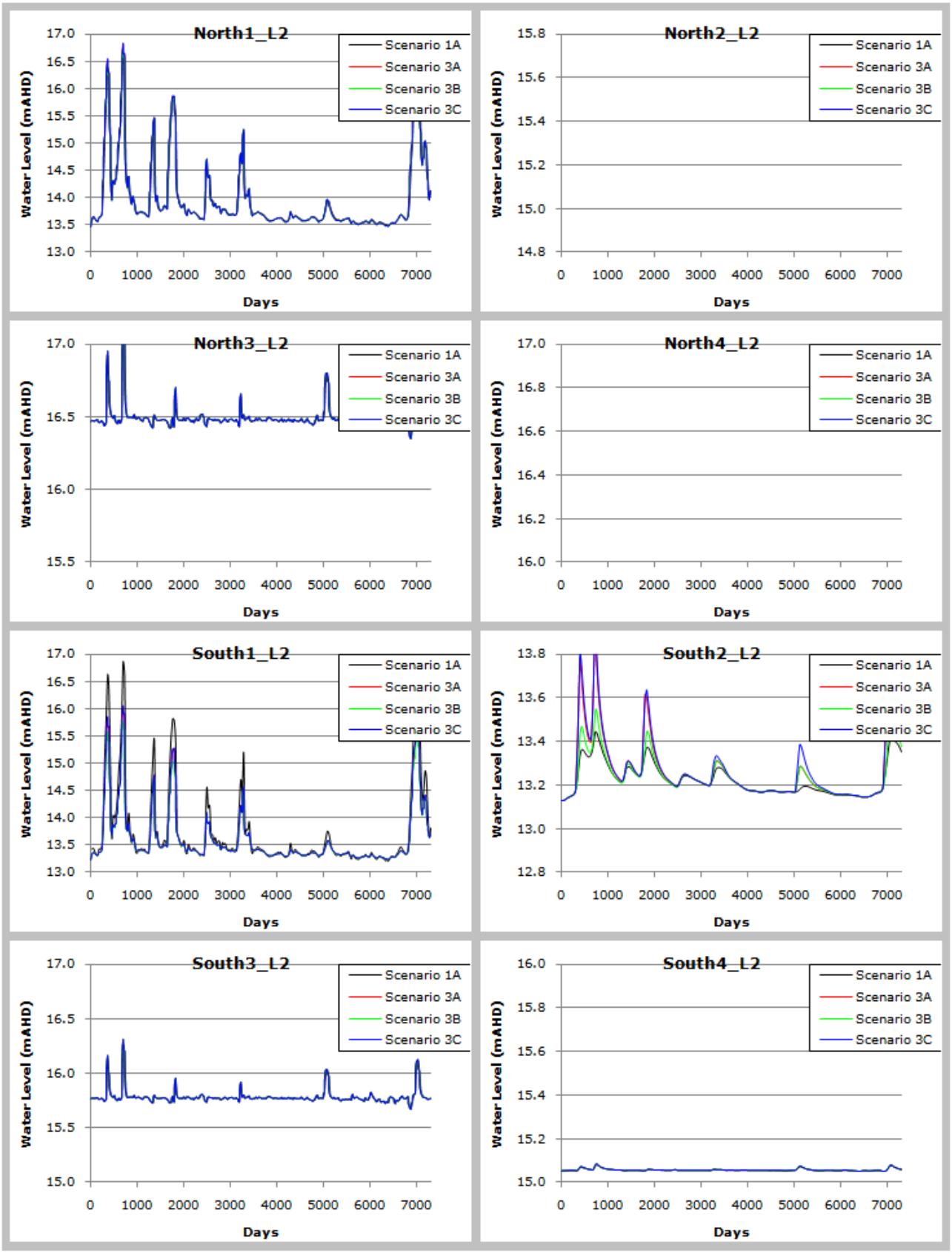
Appendix C21: Model results for Scenario 3 – Model C – Locked

The following figures present the transient information of model behaviour for the scenario in the title. These simulations feature flood inundation that matches the monthly stress period setup. The flood innundation is simulated via increased recharge at times when the river stage is above elevation. The various scenarios are described below.

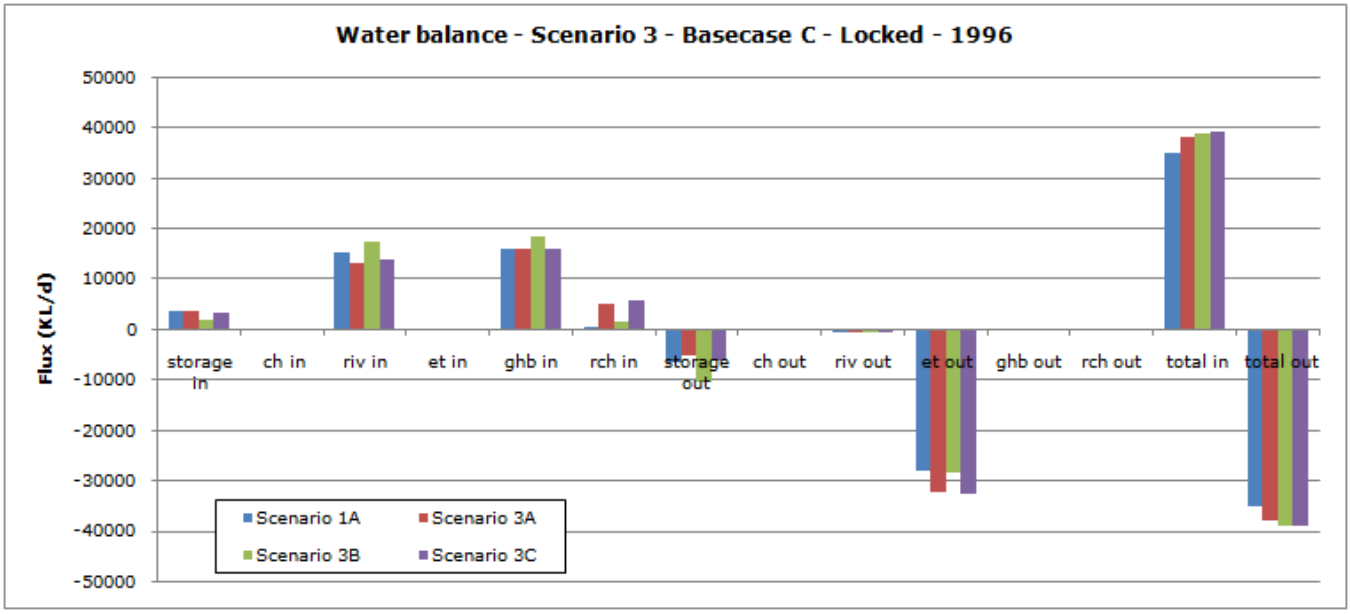
Scenario name	Stress Periods	Inundation Recharge Rate (mm/day)	Area inundated	Spatial variation
Scenario 3A	Monthly	1	Floodplain + Wetland	No
Scenario 3B	Monthly	1	Wetland only	No
Scenario 3C	Monthly	0.5 - 2	Floodplain + Wetland	Yes

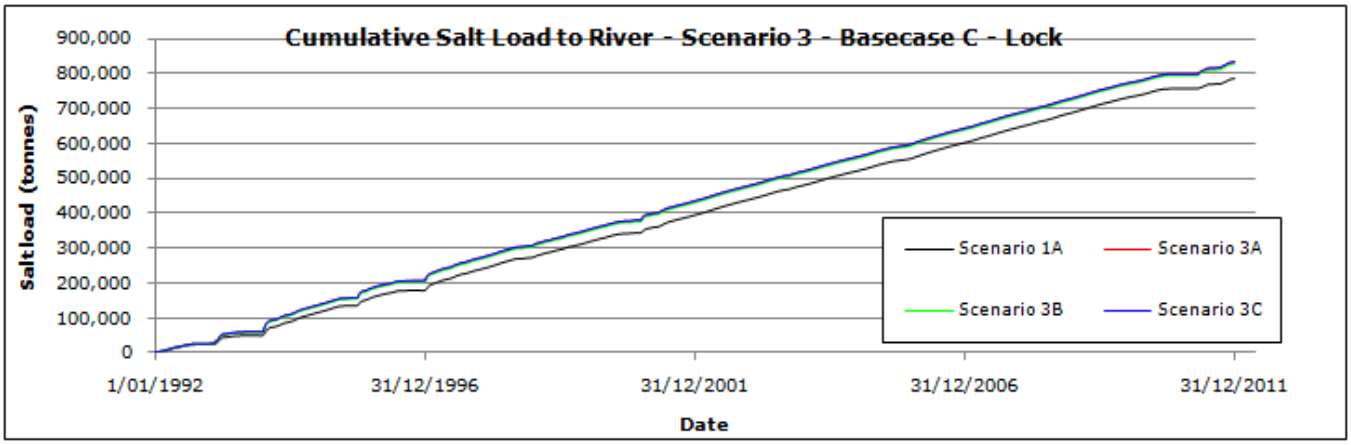


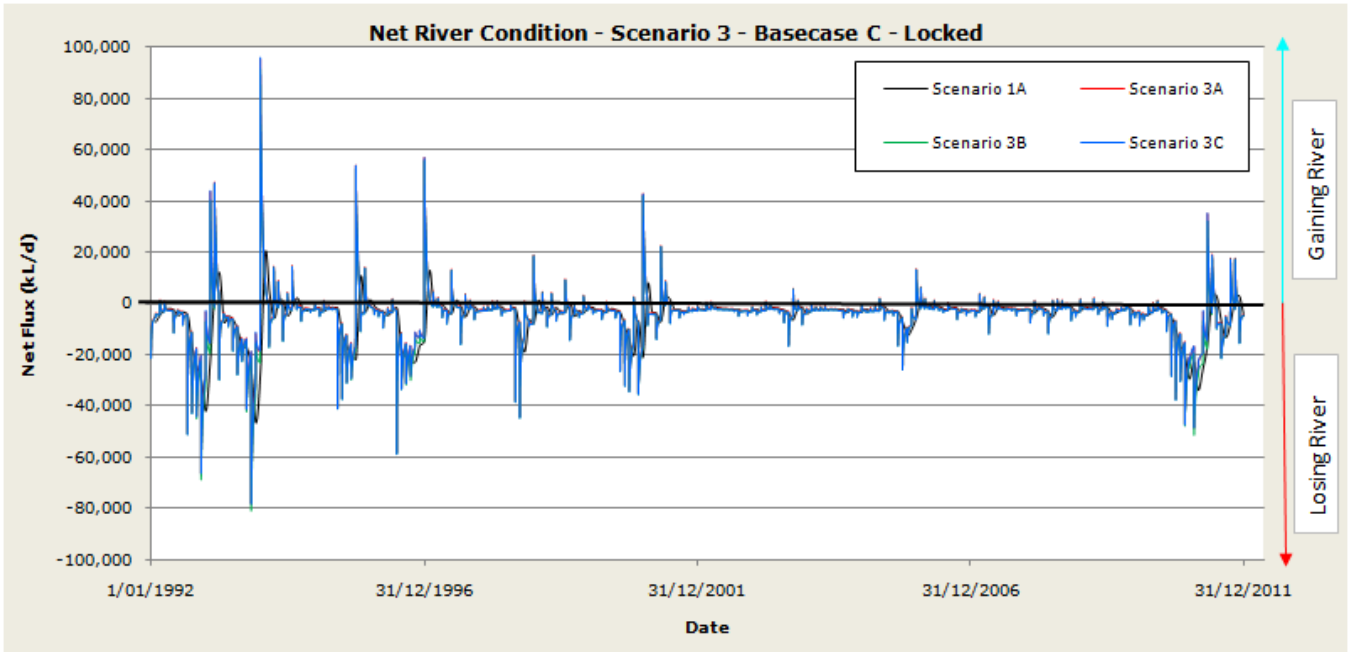
SCENARIO 3 - GOYDER MODEL C - LOCKED



SCENARIO 3 - GOYDER MODEL C - LOCKED


Water balance - Scenario 3 - Basecase C - Locked - 1996
Flux (KL/d)
50000
40000
30000
20000
10000
0
-10000
-20000
-30000
-40000
-50000
storage in
ch in
riv in
et in
ghb in
rch in
storage out
ch out
riv out
et out
ghb out
rch out
total in
total out
Scenario 1A
Scenario 3A
Scenario 3B
Scenario 3C

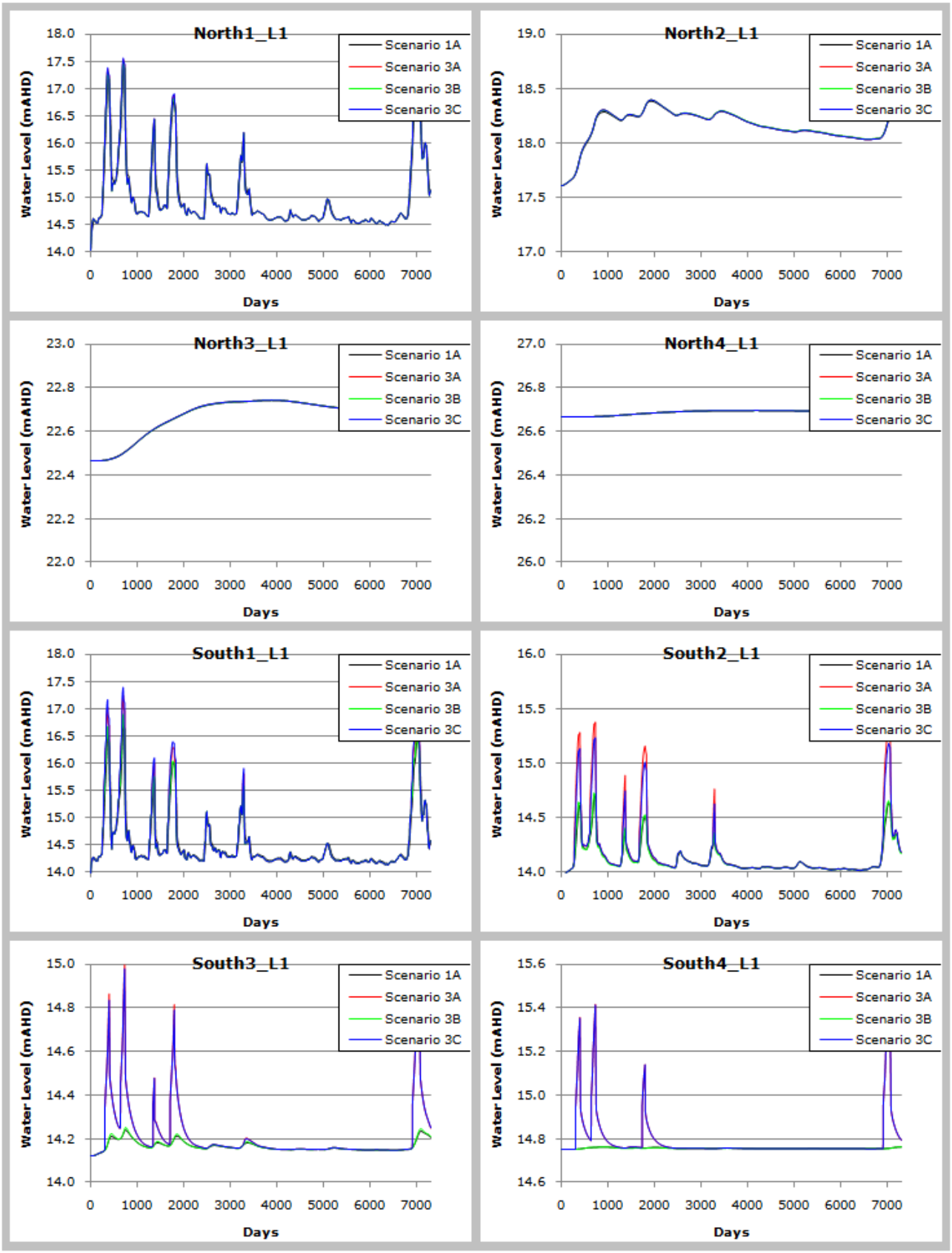

Cumulative Salt Load to River - Scenario 3 - Basecase C - Lock
Salt load (tonnes)
900,000
800,000
700,000
600,000
500,000
400,000
300,000
200,000
100,000
0
1/01/1992
31/12/1996
31/12/2001
31/12/2006
31/12/2011
Date
Scenario 1A
Scenario 3A
Scenario 3B
Scenario 3C


Net River Condition - Scenario 3 - Basecase C - Locked
Net Flux (kL/d)
100,000
80,000
60,000
40,000
20,000
0
-20,000
-40,000
-60,000
-80,000
-100,000
1/01/1992
31/12/1996
31/12/2001
31/12/2006
31/12/2011
Date
Scenario 1A
Scenario 3A
Scenario 3B
Scenario 3C
Gaining River
Losing River

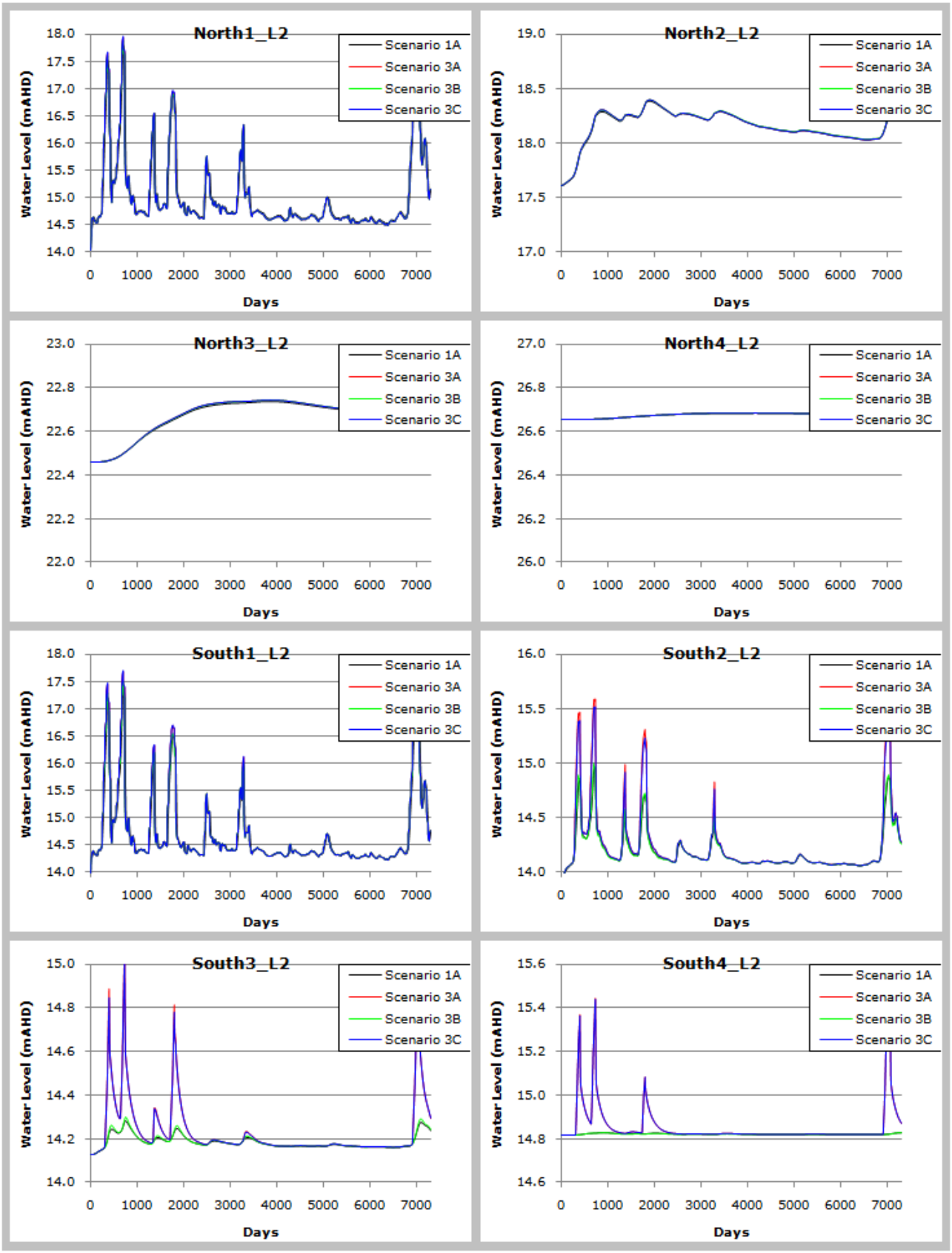
Appendix C22: Model results for Scenario 3 – Model C – Not locked

The following figures present the transient information of model behaviour for the scenario in the title. These simulations feature flood inundation that matches the monthly stress period setup. The flood innundation is simulated via increased recharge at times when the river stage is above elevation. The various scenarios are described below.

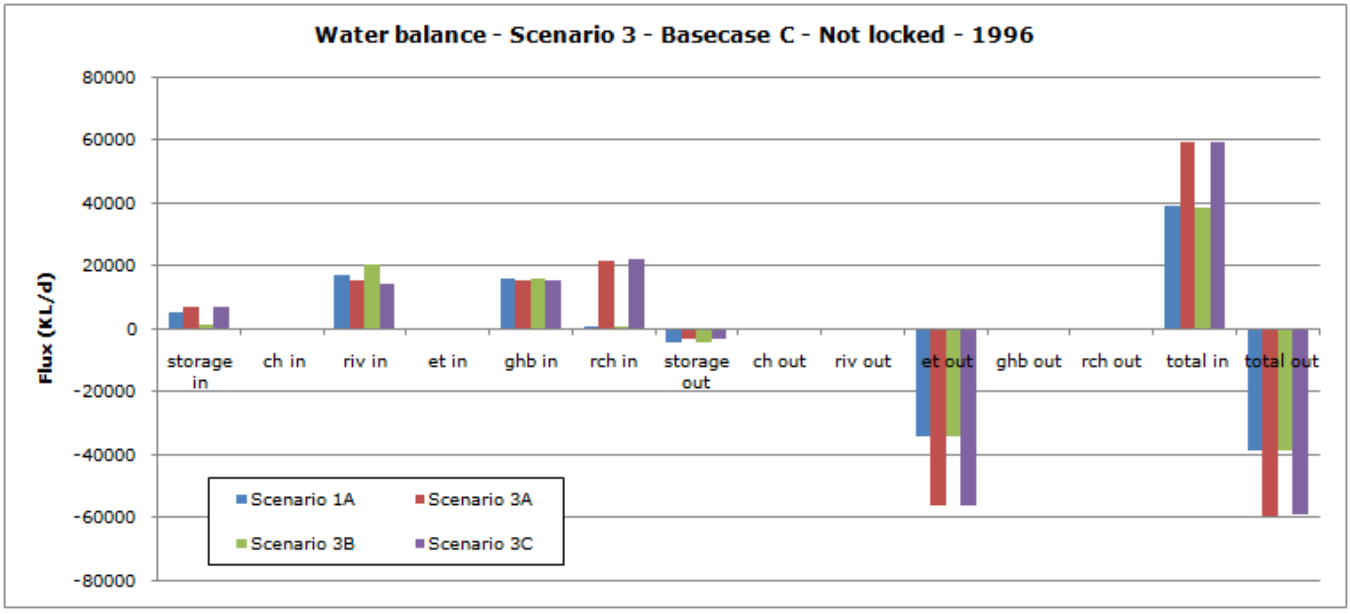
Scenario name	Stress Periods	Inundation Recharge Rate (mm/day)	Area inundated	Spatial variation
Scenario 3A	Monthly	1	Floodplain + Wetland	No
Scenario 3B	Monthly	1	Wetland only	No
Scenario 3C	Monthly	0.5 - 2	Floodplain + Wetland	Yes

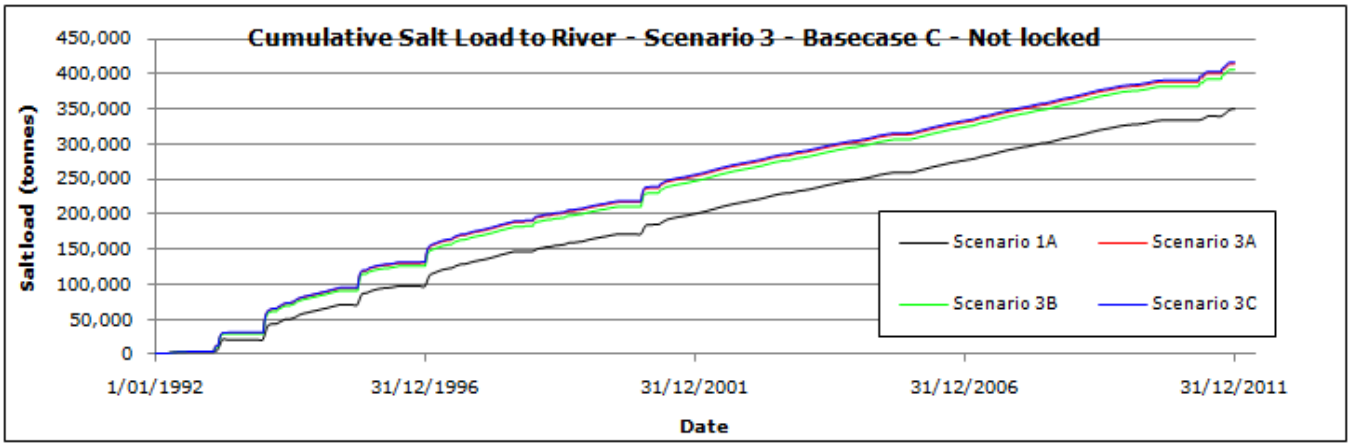


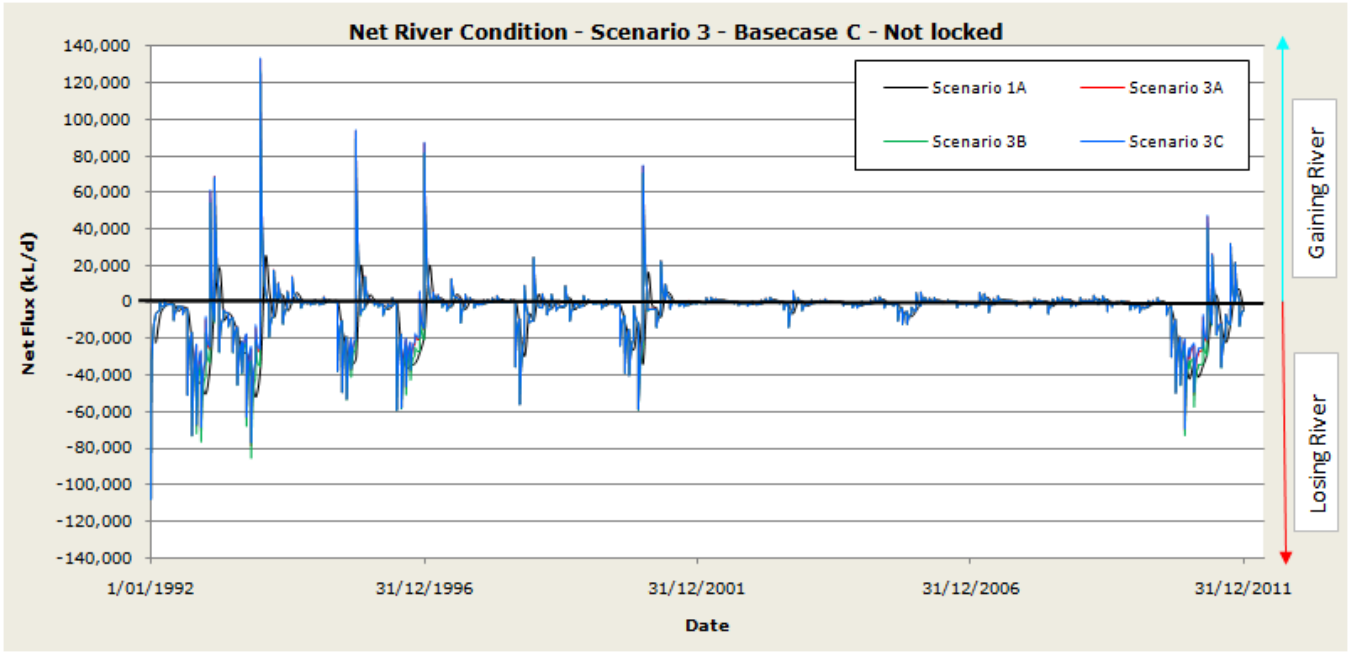
SCENARIO 3 - GOYDER MODEL C - NOT LOCKED



SCENARIO 3 - GOYDER MODEL C - NOT LOCKED


Water balance - Scenario 3 - Basecase C - Not locked - 1996
Flux (KL/d)
80000
60000
40000
20000
0
-20000
-40000
-60000
-80000
storage in
ch in
riv in
et in
ghb in
rch in
storage out
ch out
riv out
et out
ghb out
rch out
total in
total out
Scenario 1A
Scenario 3A
Scenario 3B
Scenario 3C


Cumulative Salt Load to River - Scenario 3 - Basecase C - Not locked
Salt load (tonnes)
450,000
400,000
350,000
300,000
250,000
200,000
150,000
100,000
50,000
0
1/01/1992
31/12/1996
31/12/2001
31/12/2006
31/12/2011
Date
Scenario 1A
Scenario 3A
Scenario 3B
Scenario 3C


Net River Condition - Scenario 3 - Basecase C - Not locked
Net Flux (kL/d)
140,000
120,000
100,000
80,000
60,000
40,000
20,000
0
-20,000
-40,000
-60,000
-80,000
-100,000
-120,000
-140,000
1/01/1992
31/12/1996
31/12/2001
31/12/2006
31/12/2011
Date
Scenario 1A
Scenario 3A
Scenario 3B
Scenario 3C
Gaining River
Losing River

GOYDER INSTITUTE FOR WATER RESEARCH

Government of South Australia
Department of Environment, Water and Natural Resources

CSIRO

THE UNIVERSITY of ADELAIDE

University of South Australia

Flinders UNIVERSITY ADELAIDE • AUSTRALIA

The Goyder Institute for Water Research is a partnership between the South Australian Government through the Department of Environment, Water and Natural Resources, CSIRO, Flinders University, the University of Adelaide and the University of South Australia.