jCbc: An Open Source MILP Solver for use in WRIMS

Babak Moazzez, [1], Mahdi Ghamkhari[2], Kevin Kao[3], Junaid As-Salek[4], Zhaojun Bai[5], Matthias Koeppe[6], Tariq Kadir[7], Prabhjot Sandhu[8] and Sanjaya Seneviratne[9]

[1]Kennesaw State University, bmoazzez@kennesaw.edu
[2]University of California, Davis, ghamkhari@ucdavis.edu
[3]California Department of Water Resources, kuo-cheng.kao@water.ca.gov
[4]United States Bureau of Reclamation, jassalek@usbr.gov
[5]University of California, Davis, bai@cs.ucdavis.edu
[6]University of California, Davis, mkoeppe@math.ucdavis.edu
[7]California Department of Water Resources, Tariq.Kadir@water.ca.gov
[8]California Department of Water Resources, Prabhjot.Sandhu@water.ca.gov
[9]California Department of Water Resources, Sanjaya.Seneviratne@water.ca.gov

Abstract

We describe an open-source linear programming (LP) and mixed integer linear programming (MILP) solver, called jCbc, developed for use by CalSim/CalLite models through the WRIMS interface. jCbc is a Java Native Interface (JNI) for the COIN-OR MILP solver Cbc and LP solver Clp with modifications and new capabilities. The development of jCbc is based on a comprehensive review and performance test of commercial and open-source LP and MILP solvers for CalSim/CalLite models. jCbc exploits the domain-specific features for reducing the solution time of MILPs. Two examples are presented to demonstrate the functionality and accuracy of jCbc.

Keywords: Cbc, MILP, solver, Branch-and-Bound, WRIMS, CalSim, CalLite

1 Introduction

jCbc is a Java Native Interface for COIN OR Mixed Integer Linear Programming Solver Cbc [3] and Linear Programming Solver Clp [2], with some modifications and new capabilities added.

A Mixed Integer Programming Problem (MILP) has the following general form

$$\begin{aligned} \text{min} \quad & c^T x \\ \text{s.t.} \quad & Ax \le b \\ & Bx = d \\ & l \le x \le u \\ & x_i \in \mathbb{Z} \quad \text{for } i \in I, \end{aligned} \tag{1}$$

where x is the column vector of variables, and c, b, d, l and u are column vectors. Also, A and B are the matrices of coefficients and I is a subset of variable indices. The equality and inequality constraints in (1) are element-wise and l and u can be $\pm\infty$. jCbc solves problems that have the general form of (1). An MILP model is either constructed using the functions and objects that are available in jCbc (see Section 4.1), or it is fed to the solver using an LP or MPS format file (see Section 4.2). In either case, the solver applies a Branch-and-Bound algorithm to find an optimal solution or declares that the problem has no solution or it is unbounded. See [6, Chapter 12] for an introduction on the MILP and algorithms to solve the MILP.

The MILPs in the CalSim and CalLite models are in the form of (1) and therefore can be solved by jCbc. WRIMS is a water resources modeling system for evaluating operational alternatives of large and complex river basins [1], and is the GUI through which CalSim/CalLite models are solved. The development of jCbc is supported in part by a grant from the Bureau of Reclamation, US Department of the Interior. The latest stable version of jCbc can be downloaded from `https://github.com/JNICbc/jCbc`. jCbc comes with a user manual that can be downloaded from the above link and includes detailed discussions about the existing tools in jCbc for solving MILPs.

The rest of this paper is organized as follows. Section 2 discusses the the components that are used in building jCbc, the supported platforms and licensing in using jCbc. The available objects and models in jCbc are discussed in Section 3. Two example Java codes are discussed in Section 4 illustrating how jCbc can be used for solving MILPs. Conclusions and discussion of futures works are presented in Section 5.

2 jCbc

jCbc uses an open-source software Simplified Wrapper and Interface Generator (SWIG) [4]. SWIG is a software development tool that connects libraries written in C and C++ with a variety of high-level programming languages. Most of Cbc and Clp objects and functions are carried over to the Java environment using jCbc along with several new functions that are useful for different purposes such as warm starting, reducing solve time, customizing pre-solve and pre-process steps, coefficient scanning, parallel attempts to solve a single model with different tunings, and customizing cutting planes and branching methods. Currently, there are two main solvers implemented in jCbc:

1. Function `solve()` calls internal jCbc solver with the following tunings: Cutting planes are applied only at root node, all heuristics are turned off, and there is no preprocessing and no presolve steps used. In addition, the Primal Simplex algorithm is used for the LP part of the Branch-and-Bound algorithm.
2. Function `solve_whs()` first uses ClpPresolve to simplify the model. Then, the information from a previous model is passed to `solve_whs()` in terms of names and values of integer variables. Using that information, a feasible solution is calculated for the new simplified model which in turn is used for hot starting the Cbc Branch-and-Bound algorithm. As a result, the solving time of `solve_whs()` function is significantly lower that the solving time of `solve()` function.

The jCbc github repository contains the following files and sub-folders: `manual.pdf`, `jCbc.dll`, `jCbc.i`, `jCbc.cpp`, `src`, `examples`, `make.bat`. `jCbc.dll` is the shared library that is loaded into Java. `jCbc.i` is the SWIG input file and `jCbc.cpp` is the C++ source file. `src` is the source folder and `examples` contains example Java codes. `make.bat` is the *makefile* to compile jCbc. Each one of these files and folders along with detailed process of building and using jCbc are discussed in `manual.pdf`.

jCbc takes advantages of third party packages ASL [7], BLAS [8], LAPACK [11], METIS [10] and MUMPS [9] to reduce the time in solving an MILP. jCbc also uses the `lpthread` library to support multi-threading in solving an MILP. jCbc is currently supported on Windows 64-bit and Linux 64-bit platforms jCbc is free software, i.e., one can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, version 3.

3 Objects and Functions in jCbc

In this section, we introduce some of jCbc objects and functions that are needed to build a model. See jCbc manual [5] for a complete list of objects and functions in jCbc along with their descriptions.

3.1 Objects

jCbc has different types of objects for defining models, solver interfaces and arrays of doubles and integers:

1. `jCbcModel`: this is the core object to build a model.
2. `jOsiClpSolverInterface`: this is the LP solver used inside a `jCbcModel` object. It needs to be created before `jCbcModel` since most of the operations such as reading and writing a file or building a model require this object.
3. `jCoinModel`: it is faster to define variables, bounds and constraints in a `jCoinModel` object and then add them at once from this object to `jCbcModel`.
4. `SWIGTYPE_p_double`: this is a SWIG pointer for doubles.
5. `jarray_double`: this is a JNI wrapped object that must be used instead of an array of doubles in Java to pass information to objects. Use command
   ```
   SWIGTYPE_p_double A = jCbc.new_jarray_double(n)
   ```
   to create a `jarray_double` of size n. To add an element to `jarray_double`, use the command
   ```
   jCbc.jarray_double_setitem(jarray_double A,i,k)
   ```
   to set $A[i] = k$. Use the command
   ```
   jCbc.jarray_double_getitem(jarray_double A,i)
   ```
   to access the element $A[i]$.
6. `SWIGTYPE_p_int`: this is a SWIG pointer for integers.
7. `jarray_int`: this is a JNI wrapped object that must be used instead of an array of integers in Java to pass information to objects.

3.2 Functions

The functions in jCbc let a user to set a name for a defined model object, add rows to the model, and assign a solver interface to the model:

1. `void addRow(jCoinModel A, int numberInRow, int [] index, double [] values, double rowlb, double rowup, String name)`: adds a row of the form $l \leq ax \leq u$ to a `jCoinModel` object. The upper and lower bounds can be `Double.MAX_VALUE` (∞) and `-Double.MAX_VALUE` ($-\infty$) respectively. `numberInRow` is the number of variables with non-zero coefficients in the row, and the coefficient of `index[k]` th variable in the row is `values[k]`. The elements of `index` are *not* required to be sorted.
2. `void addRows(jOsiClpSolverInterface A, jCoinModel B)`: add all rows at once from a `jCoinModel B` object to a `jCbcModel A` object. Adding rows one by one to a `jCoinModel` object and then copying all the resulted rows from the `jCoinModel` object to a `jCbcModel` object is faster than adding rows one by one to a `jCbcModel`. The source in copying is `jCoinModel B` and the destination is `jCbcModel A`.
3. `void assignSolver(jCbcModel A, jOsiClpSolverInterface B)`: assigns `jOsiClpSolverInterface B` as the LP solver interface for `jCbcModel A`.
4. `void setModelName(jOsiClpSolverInterface A, String name)`: sets the model name.

4 Examples

In this section we provide two examples to show how jCbc can be used for solving an MILP in the general form of (1). The interested readers may refer to the jCbc manual for more illustrative examples. The Java codes in these examples are executed on a dual core personal computer with CPU frequency @2.60GHz and 8Gb RAM.

4.1 Example 1

Consider the following MILP:

$$
\begin{array}{ll}
\text{min} & x_1 + 2x_2 + 4x_3 \\
\text{s.t.} & x_1 + 3x_3 \geq 1 \\
& -x_1 + 2x_2 - 0.5x_3 \leq 3 \\
& x_1 - x_2 + 2x_3 = \frac{7}{2} \\
& 0 \leq x_1 \leq 1 \\
& -\infty < x_2 \leq 10 \\
& 2 \leq x_3 \leq +\infty \\
& x_1, x_3 \in \mathbb{Z}.
\end{array}
\tag{2}
$$

The following example code shows how to build a model in `jCbc` for the MILP in (1), solve the model and finally retrieve the solution.

```java
import src.jCbc;
import src.SWIGTYPE_p_CbcModel;
import src.SWIGTYPE_p_double;
import src.SWIGTYPE_p_int;
import src.SWIGTYPE_p_CoinModel;
import src.SWIGTYPE_p_OsiClpSolverInterface;

public class example1 {
public static void main(String argv[]) {
System.loadLibrary("jCbc");

SWIGTYPE_p_CoinModel modelObject = jCbc.new_jCoinModel();

SWIGTYPE_p_OsiClpSolverInterface solver =
                                  jCbc.new_jOsiClpSolverInterface();

double objValue[] = {1.0, 2.0, 4.0};
double upper[] = {1.0, 10.0, Double.MAX_VALUE};
double lower[] = {0.0,-Double.MAX_VALUE, 2.0};

jCbc.addCol(modelObject,lower[0],upper[0],objValue[0],"x1",true);
jCbc.addCol(modelObject,lower[1],upper[1],objValue[1],"x2",false);
jCbc.addCol(modelObject,lower[2],upper[2],objValue[2],"x3",true);

SWIGTYPE_p_int row1Index = jCbc.new_jarray_int(2);
jCbc.jarray_int_setitem(row1Index,0,0);
jCbc.jarray_int_setitem(row1Index,1,2);

SWIGTYPE_p_double row1Value = jCbc.new_jarray_double(2);
jCbc.jarray_double_setitem(row1Value,0,1.0);
jCbc.jarray_double_setitem(row1Value,1,3.0);

jCbc.addRow(modelObject,2, row1Index, row1Value, 1.0,
                                                Double.MAX_VALUE,"r1");

SWIGTYPE_p_int row2Index = jCbc.new_jarray_int(3);
jCbc.jarray_int_setitem(row2Index,0,0);
jCbc.jarray_int_setitem(row2Index,1,1);
jCbc.jarray_int_setitem(row2Index,2,2);

SWIGTYPE_p_double row2Value = jCbc.new_jarray_double(3);
42: jCbc.jarray_double_setitem(row2Value,0,-1.0);
43: jCbc.jarray_double_setitem(row2Value,1,2.0);
44: jCbc.jarray_double_setitem(row2Value,2,-0.5);
45:
46: jCbc.addRow(modelObject,3, row2Index, row2Value,
47:                                        Double.MAX_VALUE, 3.0,"r2");
48:
49: SWIGTYPE_p_int row3Index = jCbc.new_jarray_int(3);
50: jCbc.jarray_int_setitem(row3Index,0,0);
51: jCbc.jarray_int_setitem(row3Index,1,1);
52: jCbc.jarray_int_setitem(row3Index,2,2);
53:
54: SWIGTYPE_p_double row3Value = jCbc.new_jarray_double(3);
55: jCbc.jarray_double_setitem(row3Value,0,1.0);
56: jCbc.jarray_double_setitem(row3Value,1,-1.0);
57: jCbc.jarray_double_setitem(row3Value,2,2.0);
58:
59: jCbc.addRow(modelObject,3,row3Index,row3Value,7/2.,7/2.,"r3");
60:
61: jCbc.addRows(solver, modelObject);
62:
63: SWIGTYPE_p_CbcModel Model = jCbc.new_jCbcModel();
64:
65: jCbc.assignSolver(Model,solver);
66:
67: jCbc.setModelName(solver, "test");
68:
69: jCbc.solve(Model,solver,0);
70: SWIGTYPE_p_double sol = jCbc.new_jarray_double(3);
71: sol = jCbc.getColSolution(Model);
72:
73: System.out.println("Solution:");
74:
75: System.out.println("Objective_Value=" + jCbc.getObjValue(Model));
76: for (int j = 0; j < 3; j++){
77: System.out.println(jCbc.getColName(solver,j)+"="
78:             +jCbc.jarray_double_getitem(sol,j));}}}
```

The Java code has the following structure:

- Initializing and defining objects (Lines 1-16), in particular,
  - Line 12: defining a `CoinModel` for adding rows and columns faster
  - Line 14: defining the LP solver interface
- Building the model (Lines 17-31), in particular,
  - Line 17: coefficients of the variables in objective function are set
  - Line 18: upper bounds for variables are set
  - Line 19: lower bounds for variables are set
  - Line 21-23: optimization variables are defined
  - Line 25-27: defining and initializing a SWIG type array of integers
  - Line 29-31: defining and initializing a SWIG type array of doubles
- Solving the model, and getting and printing out the solution (Lines 33-78), in particular,
  - Line 33: adding a row to `CoinModel`
  - Line 61: adding all rows to `OsiClpSolverInterface`

- Line 63: defining a new empty `CbcModel`
- Line 65: assigning the `solver` to `CbcModel`
- Line 69: solving the model
- Line 71: getting the solution
- Line 73-78: printing the solution

Assuming that the above Java code is saved in a file `example1.java`, the code can be executed by typing the following commands in a Microsoft Windows `cmd` or Linux terminal:

- `javac example1.java`
- `java example1`

The output of the Java code in this example is as follows:

```
Solution:Objective_Value=9.0
x1=0.0
x2=0.5
x3=2.0
```

We note that both jCbc and the MILP solver Gurobi [12] give the same optimal values for the objective function and variables of the MILP in (1), which shows that jCbc's solution is accurate.

4.2 Example 2

This example shows a Java code that reads a CalSim/CalLite model from the `model.lp` file, corresponding to an MILP in the general form of (2), solves the model and reports the optimal objective value as well as optimal values for integer variables. The `model.lp` file is generated by WRIMS [1]. This MILP has 6914 variables (19 of which are integer variables) and 5770 constraints.

```
01: import src.jCbc;
02: import src.SWIGTYPE_p_CbcModel;
03: import src.SWIGTYPE_p_double;
04: import src.SWIGTYPE_p_int;
05: import src.SWIGTYPE_p_CoinModel;
06: import src.SWIGTYPE_p_OsiClpSolverInterface;
07:
08: public class example2 {
09: public static void main(String argv[]) {
10: System.loadLibrary("jCbc");
11:
12: SWIGTYPE_p_OsiClpSolverInterface solver =
13:                                   jCbc.new_jOsiClpSolverInterface();
14: SWIGTYPE_p_CbcModel Model = jCbc.new_jCbcModel();
15:
16: jCbc.assignSolver(Model,solver);
17: jCbc.readLp(solver,"model.lp");
18: jCbc.solve(Model,solver);
19:
20: System.out.println("Solution:");
21: System.out.println("Objective_Value=" + jCbc.getObjValue(Model));
22:
23: for (int j = 0; j < nCols; j++){
24: if ( jCbc.isInteger(Model,j)==1)
25: System.out.println("x["+j+"]="+jCbc.jarray_double_getitem(sol,j));}}}
```

The Java code in this example has the following structure:

- Initializing and defining the model object (Lines 1-12), in particular,
  - Line 12: defining LP solver interface
  - Line 14: defining `CbcModel` object
- Solving the model (Lines 16-18), in particular,
  - Line 16: assigning `solver` to `Model`
  - Line 17: reading the model from `model.lp` file
  - Line 18: solving the model
- Getting the solution and printing it out (Lines 20-21), in particular,
  - Line 20-21: getting and printing the optimal objective value
  - Line 23-25: getting and printing the optimal values for the integer variables.

Assuming that the above Java code is in saved in a file `example2.java`, the code can be executed by typing the following commands in a Microsoft Windows `cmd` or Linux terminal:

- `javac example2.java`
- `java example2`

The output of executing the Java code in this example is as follows:

```
Solution:
Objective_Value=-1.74658261388485E10
x[2760]=1.0
x[2782]=0.0
x[2785]=0.0
x[2787]=0.0
x[2789]=0.0
x[2790]=0.0
x[2793]=0.0
x[2794]=0.0
x[2796]=0.0
x[2797]=0.0
x[2799]=0.0
x[2801]=0.0
x[2888]=1.0
x[4465]=1.0
x[4518]=1.0
x[5443]=0.0
x[6746]=0.0
x[6747]=0.0
x[6907]=1.0
```

We note that Gurobi [12] gives the same objective value for the model in this example.

5 Concluding Remarks

In this paper, we introduced jCbc, an open source solver for MILP and discussed the objects and tools that come with this solver. In jCbc, the state-of-art MILP algorithm is tuned for CalSim/CalLite models through the WRIMS interface [13]. There is an on-going effort on full integration of jCBC solvers with WRIMS, and performance tuning and optimization on various computing platforms. In addition, we are exploring a number of strategies to improve jCbc's solution quality and solving time. Finally, we plan to develop a sensitivity analyzer module in jCbc for so that it will reveal the sensitivity of the optimal values to the perturbations of key parameters in CalSim and CalLite models.

References

[1] WRIMS: Water Resource Integrated Modeling System, *California Department of Water Resources*, `http://baydeltaoffice.water.ca.gov/modeling/hydrology/CalSim`

[2] Clp: COIN-OR Linear Programming, `https://projects.coin-or.org/Clp`

[3] Cbc: COIN-OR Branch-and-Cut `https://projects.coin-or.org/Cbc`

[4] SWIG: Simplified Wrapper and Interface Generator `http://www.swig.org`

[5] jCbc User's Manual, `https://github.com/JNICbc/jCbc/blob/master/Manual.pdf`

[6] D. S. Chen, R. G. Batson and Y. Dang, *"Applied Integer Programming: Modeling and Solution"*, John Wiley & Sons Inc., Hoboken, New Jersey, 2010

[7] ASL: `https://github.com/ampl/mp`

[8] BLAS: `http://www.netlib.org/blas`

[9] MUMPS: `http://mumps.enseeiht.fr`

[10] METIS: `http://glaros.dtc.umn.edu/gkhome/views/metis`

[11] LAPACK: `http://www.netlib.org/lapack`

[12] Gurobi, `http://www.gurobi.com/resources/getting-started/mip-basics`

[13] Kevin Kao, *Implementation of Open-Source jCbc Solver for WRIMS2 and CalSim*, Presentation at California Water and Environmental Modeling Forum Annual Meeting, Folsom, CA, March 20-22, 2017

A Fairy Tale Hindsight on the Emergence of the Concept of Leakance to Estimate Seepage and a Suggested Remedy

Hubert J. Morel-Seytoux[1]
Calvin Miler[2]
Steffen Mehl[3]

1 Hydroprose International Consulting, 684 Benicia Drive Unit 71, Santa Rosa, CA 95409, hydroprose@sonic.net
2 Miller Groundwater Engineering, Fort Collins, CO 80521, calvin@millergroundwater.com
3 Dept. of Civil Engineering, CSU Chico, Chico, CA 95929-0930, smehl@csuchico.edu

Abstract

Most groundwater models currently in use like to claim that they are physically based and, by extension, scientifically defensible. This article reports a partial investigation of several such groundwater models. The study indicates that they may not have a solid physical basis when used in large-scale regional studies to estimate the stream-aquifer flow exchange. A possible remedy exists that does not require a change in the 1-dimensional equation used to calculate seepage. The empirical leakance coefficient is calibrated on the exact analytical solution to the two-dimensional flow configuration. Several simulation examples illustrate that the seepage is significantly affected by the degree of river penetration of the water-table aquifer. Most importantly the simulations demonstrate clearly that the leakance coefficient varies in time and is affected by the conditions prevailing in the river itself and its environment.

Keywords. Stream-aquifer interaction, Seepage, Stream depletion, Integrated Hydrologic Models

1.0 Introduction

Let us be clear. Given the complexity of nature it is not possible to represent correctly every inch of soil or of aquifer material. Thus models by necessity are and always will be largely conceptual. With some hindsight let us look at how, in a legendary and fictional way, the leakance concept might have originated. More seriously we shall look at how the empirical concept can be salvaged by relating it to another concept, that one derived from exact analytical considerations. Only the case of a **permanent saturated hydraulic connection** is discussed in this article. The case of incipient unsaturated connection is discussed elsewhere (Morel-Seytoux et al., 2015).

2.0 MODFLOW's conceptualization of the stream-aquifer flow exchange

Once upon a time in a distant land a team of alchemists wondered how to model the stream-aquifer interaction. "If only one could see how water flows under the ground!!!!!" So they ask Superman who has Xray vision to help. *Alas* **even he** could not see through! Ah! said a bright one, that proves there is a clogging layer in the streambed. But said a Thomas doubter what about the banks of the river? Some **illustrator** had created a, no doubt, highly imaginative picture of the flow for a *children's book*, and it looked like this:

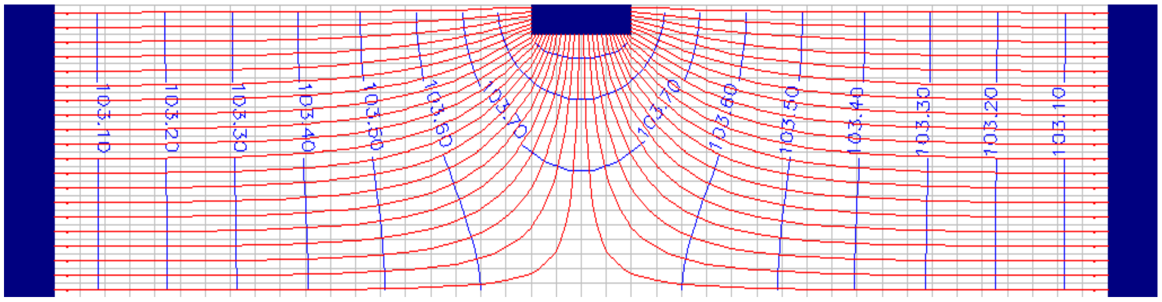

Figure 1. Very fine numerical grid needed to describe the turning of the flow lines.

Of course one cannot trust a children's book illustrator to provide a "scientifically defensible" picture. Yet some members of the team were a little perplexed.
One alchemist had a brilliant idea. **Why don't we fold the sides flat?** Then all the water will flow vertically through the clogging layer…..

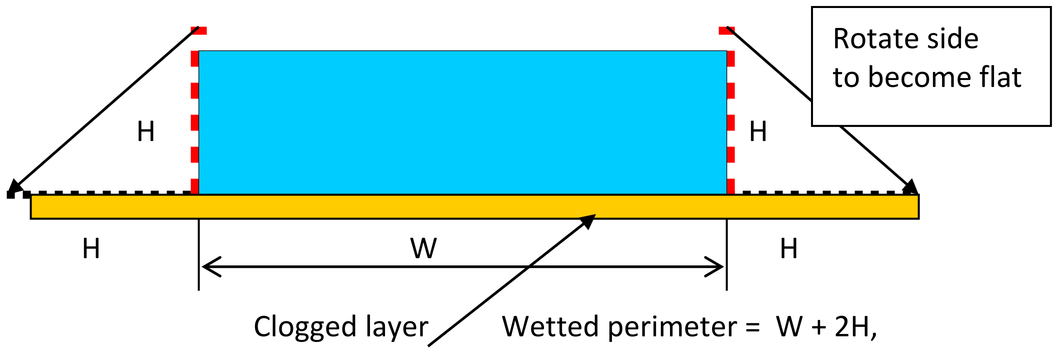

Figure 2. Flattening the sides. Wetted perimeter is preserved. However no river penetration

That way flow through the clogging layer is entirely vertical…in other words…. seepage is vertical flow,…. seepage is vertical flow, ….seepage is vertical flow…."Why should the human brain be structured so that mere repetition, without added evidence, causes us to believe a claim more strongly." (Tsipursky, 2017). As is well known in political circles if an "alternate fact" is repeated time and again, soon enough it becomes the "truth"… and, for proof, this recent figure. It is as if a Fairy had built an invisible sheet pile where the sides used to be.

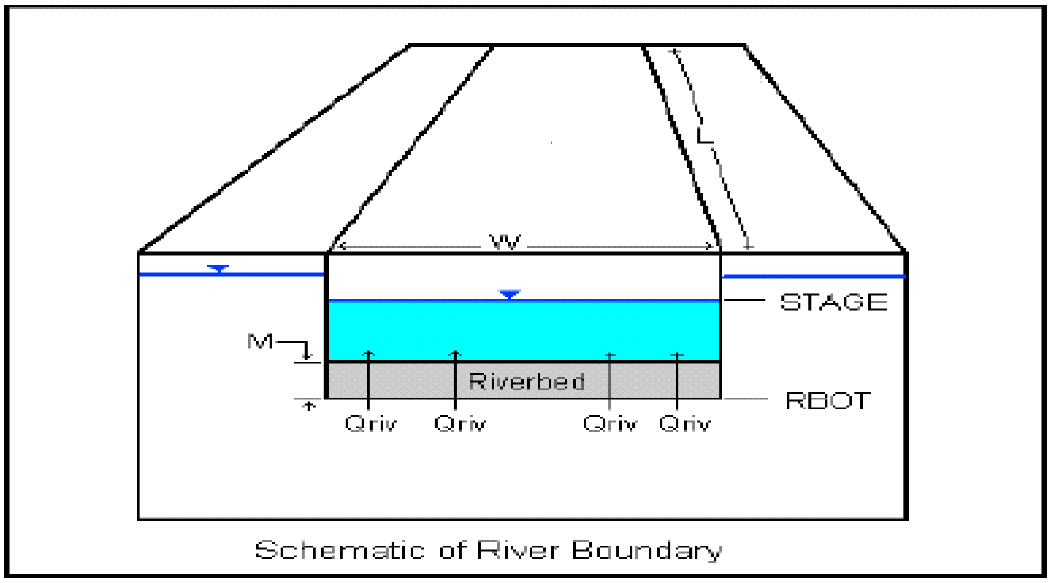

Figure 3. Conceptualization of seepage calculation. After Waterloo Hydrogeologic, Inc. (2006).

Ultimately the equation to calculate the seepage discharge from the river is:

$$Q = \left(\frac{K_u}{e_u}\right) L (W + 2H)(h_S - h_f) = \Lambda_u L W_p (h_S - h_f) \quad (1)$$

where Q is the seepage discharge, K_u is an (undefined) hydraulic conductivity, e_u is an (undefined) length, L is the length of the river reach, W is the width of the river, H is the river water stage, h_S is the river head and h_f is the head at the center of the aquifer cell that contains the river reach (the river cell). $(W+2H)=W_p$ is the wetted perimeter of the cross-section. K_u is often interpreted as the conductivity of a streambed clogging layer and e_u as its thickness but other interpretations are possible (McDonald and Harbaugh. 1988). Eq.(1) expresses Darcy's law for flow in a **vertical** direction. $(\frac{K_u}{e_u})$ is referred to as the “leakance coefficient”, L_u. Clearly as shown in Figure 1 the direction of flow right at the sides is certainly not vertical and in the case of a rectangular cross-section it is actually horizontal.

<u>3.0 MIKE-SHE ’s conceptualization of the stream-aquifer flow exchange</u>

Now in another much further East land it is said that after consulting Thor, one of the supreme Viking gods, another team of alchemists decided that rivers have no bottom and flow is only through the sides.

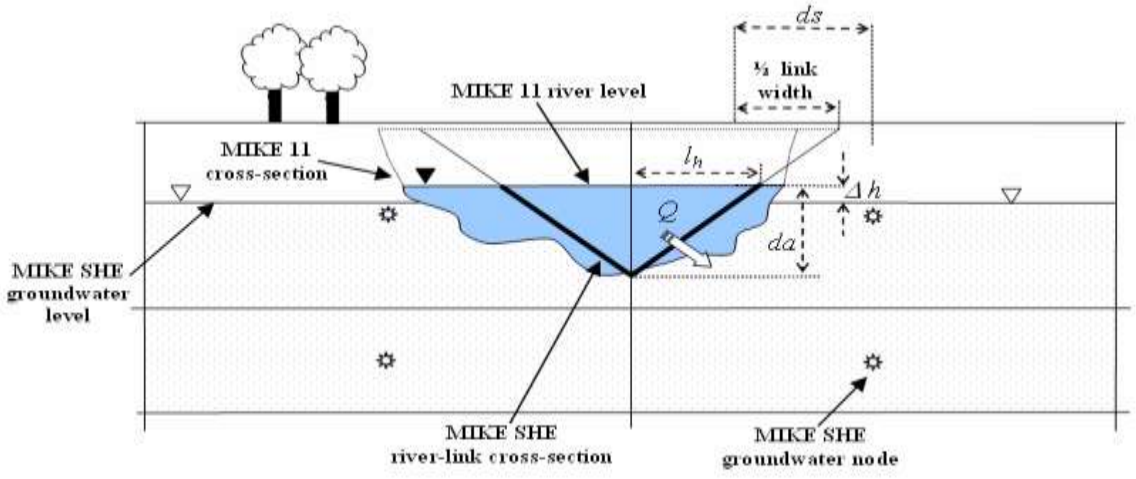
ds
½ link width
MIKE 11 river level
MIKE 11 cross-section
l_h
Δh
Q
da
MIKE SHE groundwater level
MIKE SHE river-link cross-section
MIKE SHE groundwater node

Figure 4. MIKE-SHE’s conceptualization of seepage calculation. After Figure 7.11 Section 7.6.2, page 203. A typical simplified MIKE SHE river link cross-section compared to the equivalent MIKE 11 cross-section.

*In a real **conflict of civilizations** today the modeling world remains divided: Is seepage through the bottom or through the sides?*

On page 203 of a MIKE-SHE manual it is stated: “…MIKE SHE assumes that the **primary exchange between the river and the aquifer takes place through the river banks**. For more detail on the river aquifer exchange see Groundwater Exchange with MIKE 11 (V.1 p. 207)”. In the case of MIKE-SHE to calculate the seepage the cross-section of the river is assumed to be triangular. Of course for a triangular cross-section there is no bottom; so flow has to be from the sides. It is true that to calculate the surface flow in the river reach a correct cross-section is used in MIKE-SHE (see Figure 4). However to calculate seepage the triangular cross-section is still assumed.

The contrast could not be sharper between MODFLOW and this code. Whereas in one model flow is strictly through the bottom, in the other it is strictly through the sides. So here we have the example of two different models that both claim to be physically based, “scientifically defensible” and yet describe the process of seepage in fundamentally different ways. This raises a few questions. Given that these “leakance coefficients” are **calibrated** *does it matter if incorrect equations are used to estimate seepage?* What errors will result from these two different conceptualizations?

From their respective conceptualizations one could infer that, if the river is very wide and the river stage low, then MODFLOW would provide a more realistic description of the phenomenon than MIKE-SHE. On the other hand if there is a great deal of anisotropy in the aquifer most of the flow will take place through the sides since resistance in the horizontal direction will be much less than in the vertical direction. Thus in that case MIKE-SHE would have an edge over MODFLOW.

Actually this discussion is really futile because regardless of the assumptions made to justify the leakance coefficient used in Eq.(1) it is the same **identical** equation which is used in both models after calibration of the leakance coefficient.

Thus the problem is not with the assumptions and the conceptualization of the processes that led to the equation, Eq.(1), but with the equation itself.

4.0 Can river penetration be neglected? Is a constant value of the leakance adequate?

Given these preliminaries our intention is to show that it is possible to give back to the empirical leakance coefficient in the empirical Eq.(1) a physical basis. Actually this has already being done and recently published (Morel-Seytoux et al., 2016). In this cited article it was shown that one could derive an **equivalent** leakance coefficient based on the "SAFE (Stream Aquifer Flow Exchange) dimensionless conductance" which, when introduced in Eq.(1,) would give the exact dimensional solution for a configuration such as shown for example in Figure 1. Thus the internal structure of MODFLOW does not have to be changed at all.

The two major problems with the empirical leakance coefficient are that: (1) there are no guidelines on how to estimate it, so that it is usually obtained by calibration, and (2) it is assumed that it is constant in time. In other words it does not vary with the surrounding conditions, whether the river is in flood or recession, whether the weather is changing or not, whether additional wells will be added and pump in the future, etc.

There are two purposes to the following demonstration. The first one is to demonstrate that neglecting the river penetration into the aquifer can lead to significant errors in the estimation of seepage. The second is to show that the leakance coefficient varies in time.

Figure 5 shows the cross-section of a river reach in saturated hydraulic connection with a water table aquifer. The system is subject to two external excitations: the river head and the head in the adjacent aquifer cells. Both vary with time. These excitations are displayed in Tables 1 and 2.

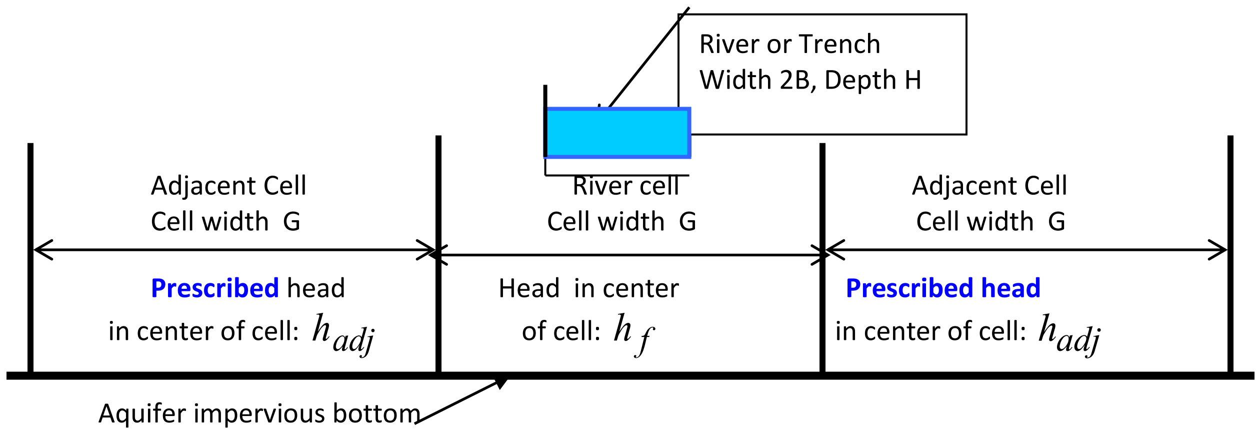

Figure 5. System cross-section

Table 1 Evolution of river stages with time

Day n	$n \le 2$	$3 \le n \le 12$	$13 \le n \le 32$	$33 \le n \le 70$
$H(n)$	3.0	$H(2)+0.1(n-2)$	$H(12)-0.1(n-12)$	$H(32)e^{-\frac{(n-32)}{100}}$

Table 2. Evolution of head in the adjacent aquifer cell with time (days)

For	$n £ 2$	$3 £ n £ 15$	$16 £ n £ 30$	$31 £ n £ 50$	$51 £ n £ 60$	$61 £ n £ 70$
D$h_{adj}(n)$	3.0	2.0	3.0	4.0	2.0	1.0

The responses are the head in the river cell (the aquifer cell that contains the river reach), the seepage rate from the river (positive if a real seepage and negative otherwise for return flow) and the leakance coefficient. The parameters of the system are for specific yield, j_e (effective porosity) 0.2, for horizontal conductivity, K_H, 10 m/day, aquifer thickness below the river bottom, D_{brb}, 15 m, degree of anisotropy, $\frac{K_V}{K_H}$, 0.1, grid size, G, 120 m, river reach length, L, 40 km, and river width, $2B$, 10 m.

Figure 6 displays the calculated seepage values under the external excitations provided in Tables 1 and 2, for 3 different scenarios. In scenario 1 the wetted perimeter is fully preserved but the river does not penetrate the aquifer. Degree of penetration is zero. In scenario 2 the wetted perimeter is also fully preserved but the river penetrates the aquifer. The degree of penetration is $\frac{H}{\bar{D}_{aq}}$ where $\bar{D}_{aq}$ is the average thickness of the aquifer in the vicinity of the river. Finally scenario 3 corresponds to no penetration but in addition only the river bottom transmits flow, not the sides, (more or less as shown in Figure 4). This is not a very coherent scenario and hopefully rarely used in practice. It preserves the head but not the wetted perimeter.

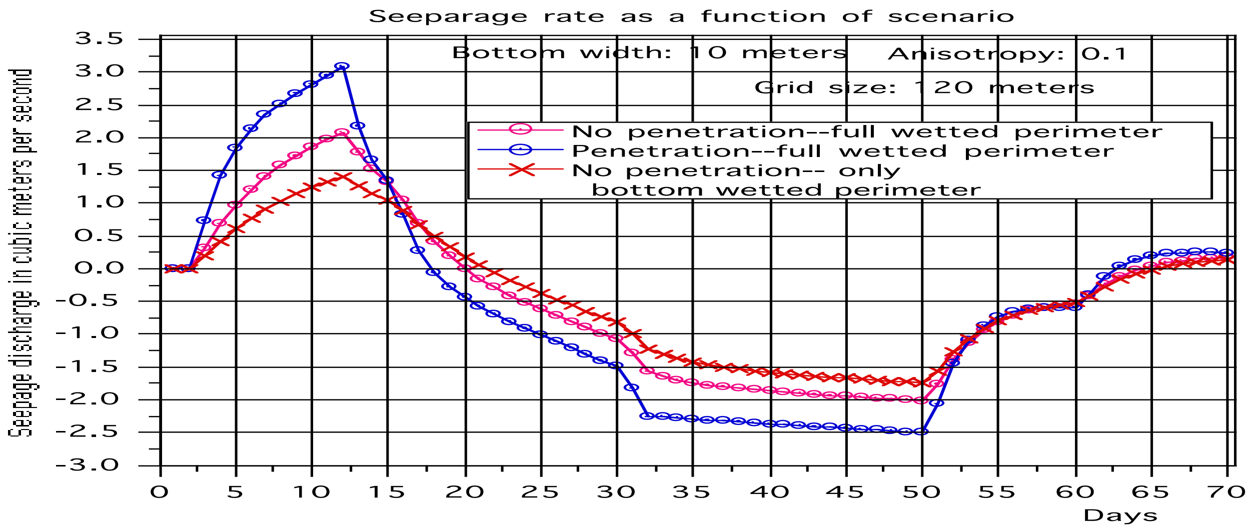

Figure 6. Seepage rates as a function of scenarios.

Whether in the case of seepage or return flow with the added effect of penetration the exchange rate is higher. This is amplified in this case as a result of anisotropy. The side fraction of the wetted perimeter varies as stage in the river varies but it is of the order of 28% (2/7). One might have expected a ratio of discharges of the order of 0.28x10+0.72x1 = 3.5 compared to the case of no penetration. However it is much smaller than that, providing evidence of numerous compensation factors. Figure 7 shows the variation of the leakances with the scenarios. One can see that the leakance ratio is high and can be of the order of as much as 3.0 but as low as 2.0. In Eq.(1) the factor leakance plays a role as well as the wetted perimeter and the head difference. In this particular comparison (scenarios 1 and 2) the wetted perimeter is the same, thus the difference comes entirely from the two other factors. Given that that the ratios of the seepages are not that high there must be a strong compensation effect that reduces the head difference more in the case of scenario 2 than in the case of scenario 1. This makes sense because when the seepage is high the head of the receiving river cell rises quickly especially when the

head of the adjacent cell is high preventing the lateral evacuation toward the adjacent cells. Figure 8 shows the variation of head in the system and it is very quickly complex.
Nevertheless the point is made that (1) the degree of penetration of the river has a significant effect on the amount of seepage and (2) very clearly the leakance coefficient is not a constant in time, independent of the surrounding conditions.

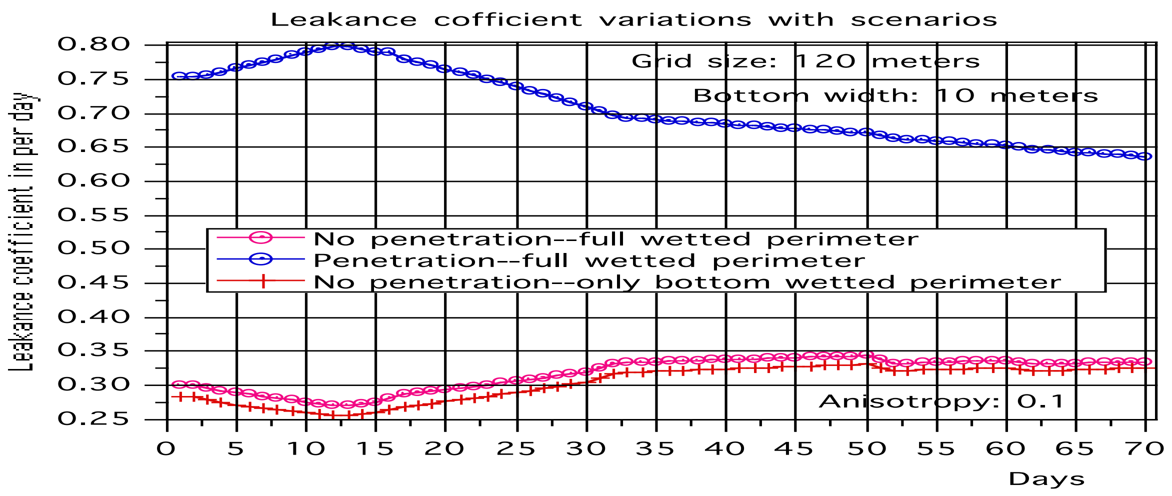

Figure 7. Variation of the leakance coefficient with the scenarios.

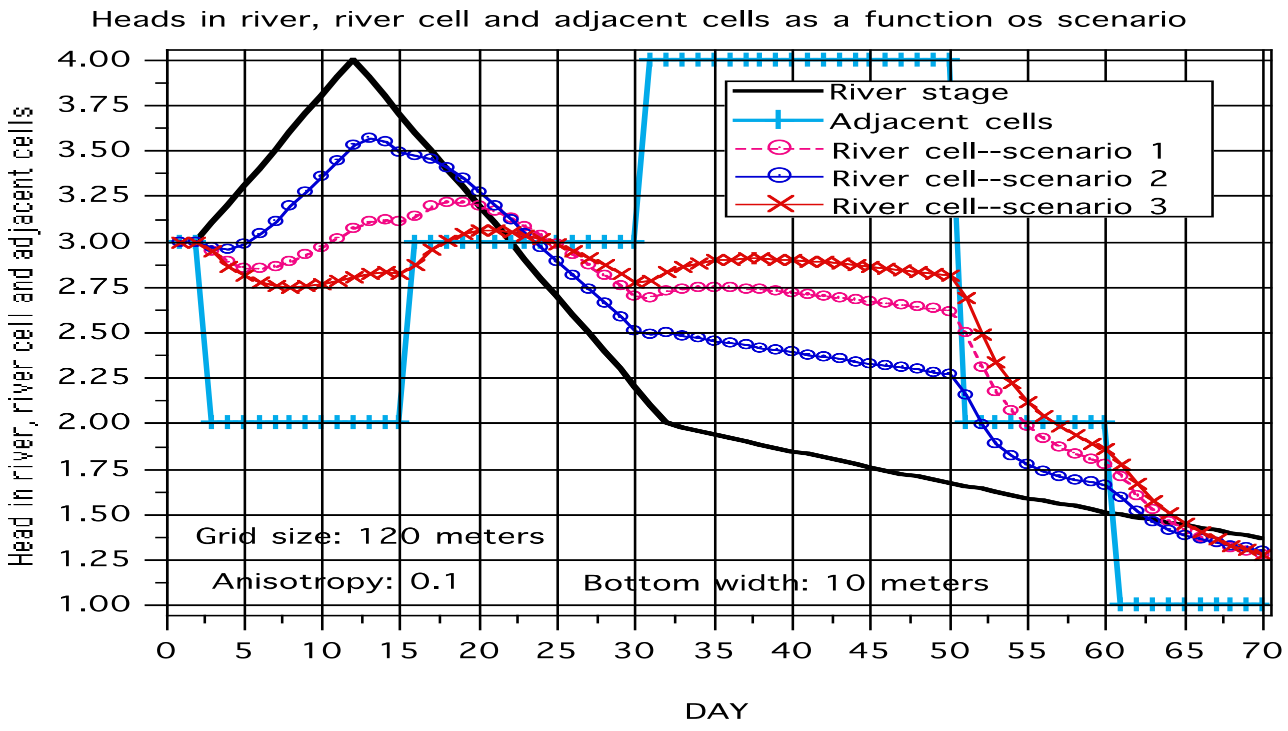

Figure 8. Variation of head in the system depending on scenario and on the external conditions.

<u>5.0 Additonal simulation to comfirm the previous conclusion</u>

To see a little better through the complex inter-relation between heads in the system under transient conditions a simpler, less variable, set of external conditions was imposed. The head in the river is maintained constant at 2 m and the heads in the adjacent cells vary as a succession of pulses. Figure 9 shows the seepage rates with or without penetration. Steady state occurs within about 15 days for the case of penetration and 20 for the case of no penetration. The seepage with penetration as steady state is reached is 33% higher than when penetration is neglected.

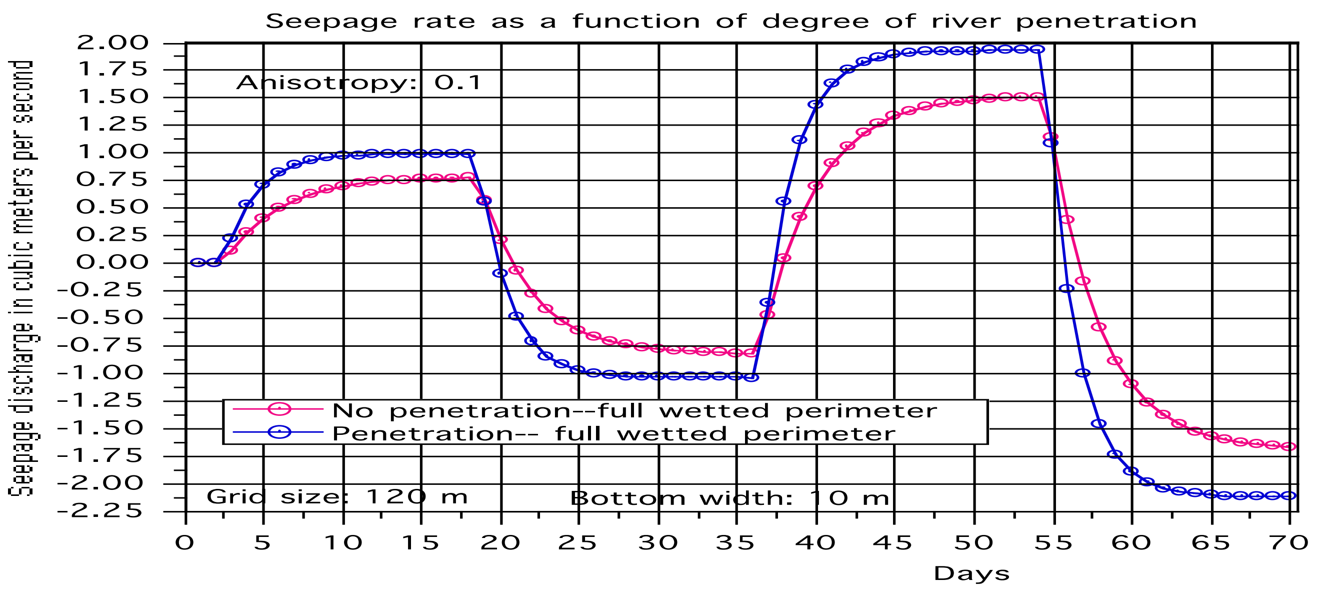

Figure 9. Seepage rates with or without penetration.

Figure 10 shows the leakance coefficients. In this case the leakance coefficient does not change much with time because the river stage remains constant in time. The SAFE dimensionless conductance does depend on degree of river penetration but in this case only because the aquifer thickness itself varies.

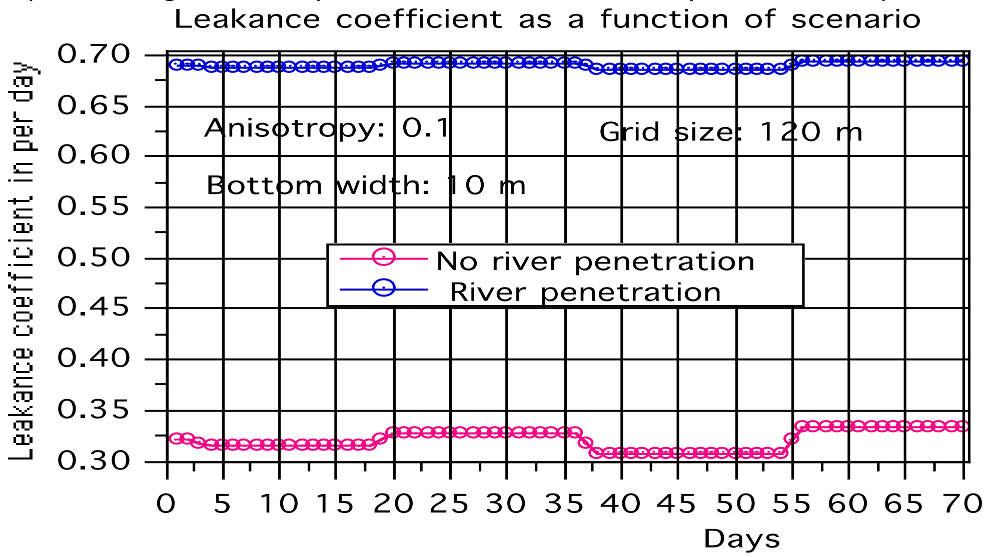

Figure 10. Leakance coefficients with or without penetration

Figure 11 shows the variation in the heads for this case. The head difference is smaller in the case of river penetration but it is more than compensated by the fact that the leakance coefficient is higher by a factor of roughly 2.

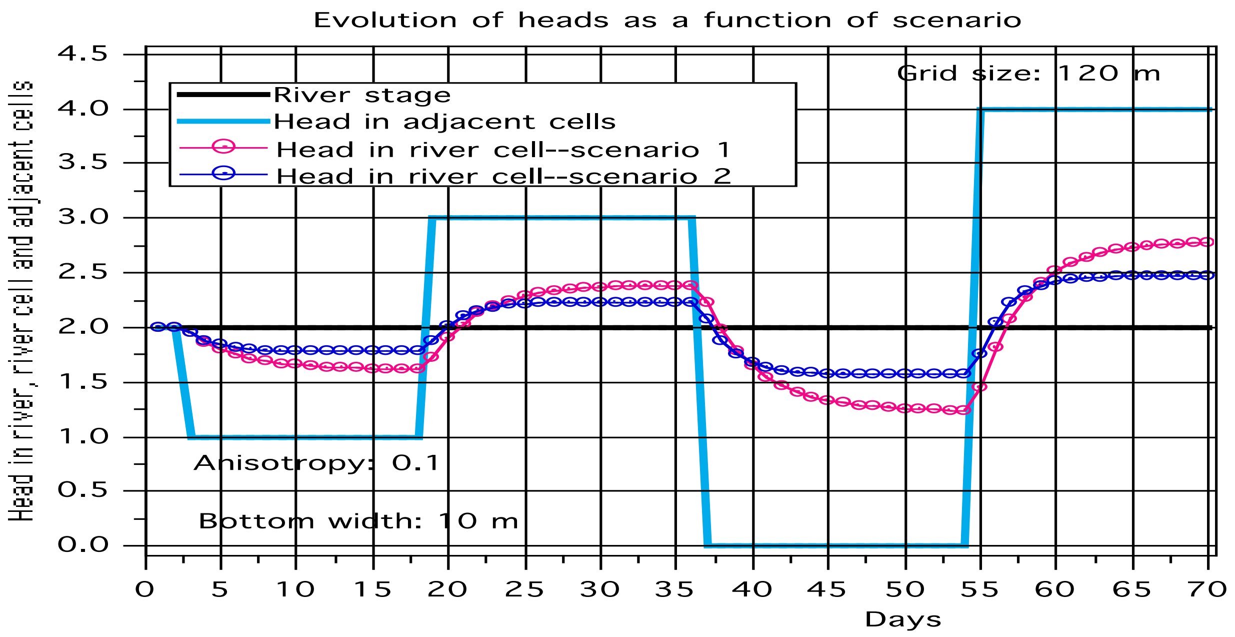

Figure 11. Dependence of head distribution on the degree of river penetration.

6. Conclusion

"Models fail for a variety of reasons. Often a theoretical model is too simple to capture all the important factors that influence a system. …Even excellent models can prove to be inadequate if they fail to incorporate an essential feature. Chaos and complexity theory tell us that that very small influences, which may have been left out of the model, can have a significant effect. " (Wallach, 2017).

Models will never be perfect but they are useful nevertheless. With some effort they can be marginally improved. Such improvement is suggested.

4. References.

MIKE_SHE_Printed_V1.pdf. USER MANUAL. User Guide. October 2013. Particularly sections 7.6.2 to 7.6.6, Pages 202-212.

Morel-Seytoux, H. J., Steffen Mehl, and Kyle Morgado (2014), Factors influencing the stream-aquifer flow exchange coefficient, GroundwaterJ., doi:10.1111/gwat.12112.

Morel-Seytoux, H.J. , Cinzia Miracapillo and Steffen Mehl (2015). Impact of aquifer desaturation on steady-state river seepage.
h_t_t_p_:_/_/_d_x_._d_o_i_._o_r_g_/_1_0_._1_0_1_6_/_j_._a_d_v_w_a_t_r_e_s_._2_0_1_5_._0_9_._0_1_2_ _0_3_0_9_-_1_7_0_8_/_© _2_0_1_5_ _E_l_s_e_v_i_e_r_ _L_t_d_._

Morel-Seytoux, H.J. , Calvin D. Miller, Cinzia Miracapillo and Steffen Mehl (2016). River Seepage Conductance in Large-Scale Regional Studies.
November 2016. ©2016,National GroundWater Association, doi: 10.1111/gwat.12491

Tsipursky, G. 2017. "The brain science of political deception and the 2016 election.". Article in Free Inquiry, April-May 2017, pp. 56-58.

Wallach, W. 2017. "A dangerous master". Article In Free Inquiry, April-May, 2017, pages 22-27.

Waterloo Hydrogeologic, Inc. (2006). Visual MODFLOW v.4.1 User's Manual. Waterloo, Ontario, Canada.