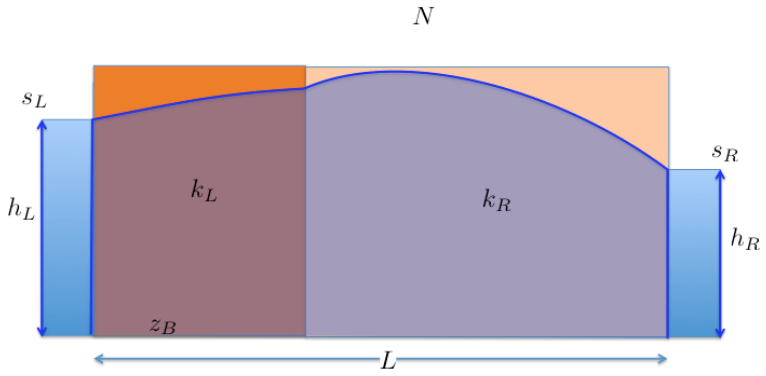
Calibration of groundwater models
Draft June 2013

Prof. dr.ir. T.N.Olsthoorn
Geohydrology 2

April 22, 2014



Contents

1 Introduction

No matter how carefully models are constructed, they generally need to be calibrated. That is, a set of user-chosen model parameters has to be optimized such that the deviation between measurements comply as well as possible with the model outcomes. There are many reasons why outcomes of a model initially deviate significantly from the measurements. For example, lack of accurate data. Differences between the scale of the model and the reach or scale of the measurements, both in time and space. Measurement errors should not be one of them, as the expectation of such errors tend to be or should be zero. Having said that, models also deviate from the measurements, because their concept differs from reality. This may be due to lack of insight in the geology as well as the processes involved. If the latter is the case, calibration may lead to the right outcomes for the wrong reasons. That is, the calibration process blindly optimizes model parameters in order to reduces the difference between model and measurements to a minimum, based on the wrong conceptual model, causing parameters to assume non-plausible values as trying to reduce the errors at the cost of other than the right parameters. Also, if the recharge has been wrongly estimated, the calibration will try to compensate that by adapting the conductivities in the model, which it is often perfectly abel to do so, as the head has tendency to depend on N/k instead of N or k separately. Without an independent measurement of the recharge, N, it may be impossible to obtain unique values for the conductivity. So, generally, calibration assumes that the model concept is correct, and that the chosen parameters are representative of the model. Different model concepts may have to be included or should be included in the calibration process, but this is seldom done, due to cost. Another problem with model calibration is too little data for too many parameters. This will cause the problem to be undefined or badly defined, so that the estimated parameter values will be highly uncertain, despite the model looking quite correct. All these pitfalls, that may yield a calibrated model that replicates the measurements quite well, will shoot you in the foot when the model is used for calibration, as the probability of producing a reliable prediction will be less, the greater the uncertainties in the parameters that have been included in the calibration. Prediction, in fact the aim of the model, has may also suffers from the fact that often it depends on other parameters than those included in the calibration. Next to all this, predictions may be required for circumstances that the model has not met in the past and, therefore, have not played a role in the calibration. This renders predictions unreliable. Often, parameters that were not very important in the calibration, that is, their uncertainty tends to remain high, may very well op paramount importance for the outcomes of prediction to be made by the model by the customer. Hence calibration and use of the model for predictions later on need careful evaluation.

2 Theory

Let us assume we have a set of measurements y and a a model that computes the values and the quantities represented by the measurements, whatever the measurements are (heads, flows, concentrations, ...). We may assume that the computed values in the measured locations depend linearly on the user-chosen parameter vector p at least when $p = p_{opt}$ and if we do not deviate too much from our initial estimate:

$$y_i = y_{p_0 i} + J_{i1}\left(p_{01} + \Delta p_1\right) + J_{i2}\left(p_{02} + \Delta p_2\right) + ...J_{in}\left(p_{0n} + \Delta p_n\right) + \epsilon_i$$

$$\Delta y_i = J_{i,1}\Delta p_1 + J_{i,2}\Delta p_2 + ...\epsilon$$

Where p_0 is our initial parameter set and y_0 the computed output for this parameter set. $\Delta p_i = p_i - p_{0i}$. Any such linear system is writable as

$$\Delta y = J\Delta p + \epsilon \tag{1}$$

Where y are the $n_m \times 1$ vector with the measurements, such as heads, etc, and p the n_p vector with the model parameters that the user choose to include in the calibration. J is an $n_m \times n_p$ coefficient array. J is the sensitivity matrix, also called the Jacobian.

$$J = \frac{\partial \hat{y}}{\partial p} \tag{2}$$

Each element in J, i.e. J_{ij} is the sensitivity of the quantity computed at measurement location i for a change of parameter j. Notice that the Jacobian is independent of the actual measurements, it is only dependent on measurement locations and times.

Each of the n_p columns of J is a sensitivity vector of the computed quantities $\hat{y}$ with respect to one parameter. That is, column i of J has n_m values as follows:

$$J_i = \frac{\partial \hat{y}}{\partial p_i} \tag{3}$$

It is clear that we assume here (eq: 1) that the sought quantity depends linearly on the parameters. This is generally not the case in real models, not even in linear models. But it is always approximately true if we allow only a small variations of the parameters. But as long as the parameters values are incorrect, and the relation between model-computed quantities and the model parameters is not linear, the derivative or sensitivities J depend on the parameter values, causing equation 1 to be incorrect. Therefore, calibration will be done stepwise, by continuously improving J such that the remaining errors ϵ between the model results and the measurements will be minimized.

Notice that we can always compute the sensitivity matrix or Jacobian with our model. Doing this is fundamental. It is generally done by computing y for the parameters at hand once and then n_p times, in which each parameters is changed by some some value in turn. Then

$$J_i = \frac{\hat{y}_{p_i+\Delta p_i, p_{j\neq i}} - \hat{y_p}}{\Delta p_i}$$

which requires $n_p + 1$ model runs for each of the J_i columns of the Jacobian. Sometimes, a somewhat more accurate form is used

$$J_i = \frac{\hat{y}_{p_i+\Delta p_i, p_{j\neq i}} - \hat{y}_{p_i-\Delta p_i, o_{j\neq i}}}{\Delta p_i}$$

which requires $2n_p - 1$ model runs to compute the full Jacobian.
It is also possible to use the adjoint state method, but this is generally too complex for use in general practices, because it requires to be built into the model code, whereas the previous methods can be done with the existing model code without any change. These methods are used by the well known programs that can optimize parameters of any model, i.e. UCODE and PEST.

Hence the objective of the calibration is to minimize the difference between model results and measurements. Taking the classical least squares approach as the simplest example, we want to optimize the values of the model parameters such that the minimize the cost function F, which is here the sum of the squares of the deviation of the measurements and the model:

$$miimize \;\; F = \sum_{i=1}^{N} \epsilon_i^2$$

where F is the cost function, also called objective function. In vector form:

$$F = (\Delta y - J\Delta p)^T (\Delta y - J\Delta p)$$

For convenience we now drop the Δ in front of y and p, but we have to remain aware of the meaning of these parameters.

Writing this out with Δ dropped yields

$$\begin{aligned} F &= y^T y - y^T Jp - (Jp)^T y + (Jp)^T Jp \\ F &= y^T y - y^T Jp - p^T J^T y + p^T J^T Jp \end{aligned}$$

2.1 Side step: differentiation of matrix equations

The matrix A and be expressed as a_{ji} where j is the column index and i the row index. A^T is the equivalent to a_{ij} with row and column interchanged.

The derivative of $A^T x$ with respect to x can be seen as follows. It is equivalent to $\sum_{j=1} a_{ij} x_j$. We'll do this by example

$$A = \begin{bmatrix} a & b & c \\ r & s & t \\ u & v & w \end{bmatrix},\ A^T = \begin{bmatrix} a & r & u \\ b & s & v \\ c & t & w \end{bmatrix},\ p = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix},\ y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

$$\begin{aligned} Ap &= \begin{bmatrix} ap_1 + bp_2 + cp_3 \\ rp_1 + sp_2 + tp_3 \\ up_1 + vp_2 + wp_3 \end{bmatrix} \\ y^T Ap &= y_1 \{ap_1 + bp_2 + cp_3\} + y_2 \{rp_1 + sp_2 + tp_3\} + y_3 \{up_1 + vp_2 + wp_3\} \end{aligned}$$

$$\begin{aligned} \frac{\partial y^T Ap}{\partial p} &= \begin{bmatrix} \{y_1 a + y_2 r + y_3 u\} & \{y_1 b + y_2 s + y_3 v\} & \{y_1 c + y_2 t + y_3 w\} \end{bmatrix} \\ &= A^T y \end{aligned}$$

and

$$\begin{aligned} A^T y &= \begin{bmatrix} ay_1 + ry_2 + uy_3 \\ by_1 + sy_2 + vy_3 \\ cy_1 + ty_2 + wy_3 \end{bmatrix} \\ p^T A^T y &= p_1 \{ay_1 + ry_2 + uy_3\} + p_2 \{by_1 + sy_2 + vy_3\} + p_3 \{cy_1 + ty_2 + wy_3\} \\ \frac{\partial p^T Ay}{\partial p} &= \begin{bmatrix} \{ay_1 + ry_2 + uy_3\} & \{by_1 + sy_2 + vy_3\} & \{cy_1 + ty_2 + wy_3\} \end{bmatrix} \\ &= A^T y \end{aligned}$$

But if we just focus on the multiplication of vector $\mathbf{a}_i^T x$ we get $\mathbf{a}_i^T = \sum_{j=1} a_{ij} x_j$ differentiating this with respect to x_k just yields a_{ik}. If we do this for all k with $1 \leq k \leq n_p$ we get the vector $\mathbf{a}$ (not transposed). Hence $\partial a_i^T \partial x = a_i$ and because this works the same for all vectors that constitute the matrix A we have

$$\frac{\partial A^T x}{\partial x} = A^T$$

The derivative of $x^T A$ with respect to x and be seen in a similar way. The coefficients in each column a_j are $\sum x_i a_{ij}$. Taking the derivative with respect to x_k of this sum yields a_{kj} which is exactly the column j. Hence for all columns

$$\frac{\partial x^T A}{\partial x} = A^T$$

The derivative of the quadratic form $x^T Ax$ is also important. Written out this form is

$$x^T Ax = \sum \sum x_j a_{ji} x_i$$

Notice that matrix A is symmetric for this multiplication to be possible. Hence summation $i = 1..n$, $j = 1..n$. If we want to differentiate with respect to x_k we need to assemble all coefficients that contain x_k. These are

$$x_k^2 a_{kk},\ \ x_k a_{kj} x_j,\ \ x_i a_{ik} x_k$$

where $k \neq j$ in the second form and $k \neq i$ in the third. These components in all their combinations constitute exactly 2 times the array A

so

$$\frac{\partial x^T Ax}{\partial x} = 2Ax$$

With this we can compute the derivative of the equation above to obtain a system with n linear equations in n unknowns.

2.2 Finishing the derivation

Differentiation with respect to p, using the results from the box in which it was shown that $\frac{\partial p^T J^T y}{\partial p} = \frac{\partial y^T J p}{\partial p} = J^T y$ then yields

$$\begin{aligned} \frac{\partial I}{\partial p} &= 0 - 2J^T \Delta y + 2J^T J \Delta p \\ J^T J \quad \Delta p &= J^T \Delta y \end{aligned}$$

so that

$$\begin{aligned} \Delta p &= \left(J^T J\right)^{-1} J^T \Delta y \\ p &= p_0 + \left(J^T J\right)^{-1} J^T (y - y_0) \end{aligned}$$

$J^T J$ is a square matrix and J^T has the number of rows equal to the number or parameters and the number of columns equal to the number of measurements.

In Matlab, you can solve a set of equations as follows

$$\begin{aligned} Ap &= y \\ y &= A \backslash p \end{aligned}$$

In Matlab you solve an over-determined system, like he have done above here, namely with more equations than unknowns in the same was

$$\begin{aligned} y &= \left(A^T A\right)^{-1} A^P p \\ y &= A \backslash p \end{aligned}$$

This will always yield the least squares solution.

We can, therefore, immediately compute the parameters if we have the Jacobian J. But the answer can only be true if the relation between computed quantities and parameters were linear, which is not generally the case. Therefore, the differences between our model results and the measurements will be larger than the mere measuring errors. The aim is to sequentially improve the parameter vector so that in the end the remaining errors are minimal.

3 Example test model

Let us assume the model is a groundwater system between two ditches with a given precipitation on it.

Continuity:

$$\frac{dq}{dx} = N \rightarrow q = q_0 + Nx$$

Considering the cross section to the left and the right of the boundary between the two conductivity regions with different length of k_L and k_R:

$$\begin{aligned} q \quad &= q_0 + Nx \quad &= -\frac{k_L}{2}\frac{dh^2}{dx}, \quad x \leq \alpha L \\ q \quad &= q_0 + Nx \quad &= -\frac{k_R}{2}\frac{dh^2}{dx} \quad x \geq \alpha L \end{aligned}$$

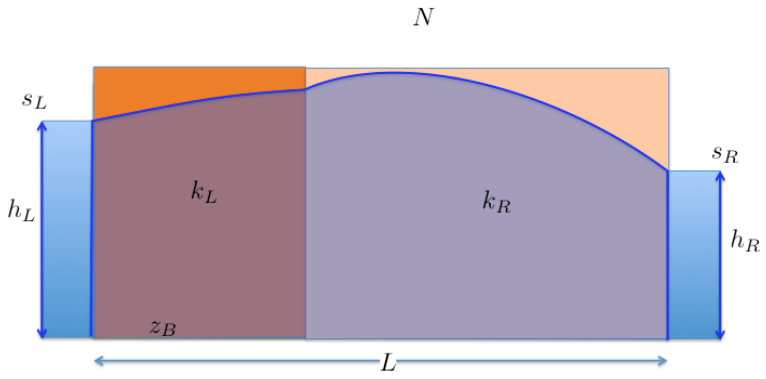


Figure 1: Simple model to use for test calibration, notice h is relative to the bottom of the aquifer, s is relative to an arbitrary fixed datum.

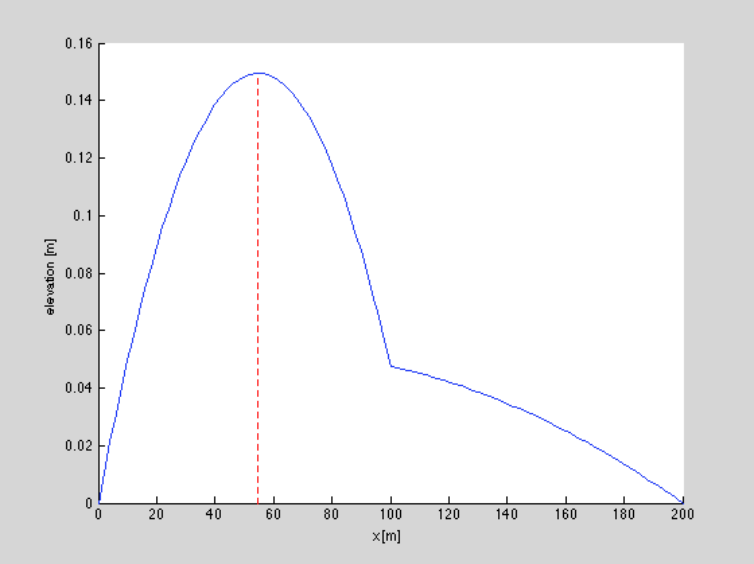


Figure 2: Head in the simple model for default values $z_B = -20$; $s_L = s_R = 0$; $\alpha = 0.5$; $k_L = 1$; $k_R = 20$; $L = 200$; $x = 0 : 2 : L$; $N = 0.002$; All dimensions in m and d units.

$$\begin{aligned} q_0 x + \frac{1}{2} N x^2 &= C_L - \frac{1}{2} k_L h^2, \;\; x \leq \alpha L \\ q_0 x + \frac{1}{2} N x^2 &= C_R - \frac{1}{2} k_R h^2, \;\; x \geq \alpha L \end{aligned}$$

Boundary conditions, $x = 0 \rightarrow h = h_L$ and $x = L \rightarrow h = h_R$
for $x < \alpha L$

$$\begin{aligned} C_L &= \frac{k_L}{2} h_L^2 \\ C_R &= \frac{k_R}{2} h_R^2 + q_0 L + \frac{1}{2} N L^2 \end{aligned}$$

yielding

$$\begin{aligned} \frac{k_L}{2} \left(h^2 - h_L^2 \right) &= -q_0 x - \frac{1}{2} N x^2, \;\; x \leq \alpha L \\ \frac{k_R}{2} \left(h^2 - h_R^2 \right) &= q_0 \left(L - x \right) + \frac{1}{2} N \left(L^2 - x^2 \right), \;\; \rightarrow x \geq \alpha L \end{aligned}$$

or

$$\begin{aligned} h^2 &= h_L^2 - \frac{2 q_0}{k_L} x - \frac{N}{k_L} x^2, \;\; x \leq \alpha L \\ h^2 &= h_R^2 + \frac{2 q_0}{k_R} \left(L - x \right) + \frac{N}{k_R} \left(L^2 - x^2 \right), \;\; \rightarrow x \geq \alpha L \end{aligned}$$

The maximum head can be found form

$$\begin{aligned} 2 \frac{dh}{dx} &= 0 = -\frac{2 q_0}{k_L} - \frac{2 N x_{peak}}{k_L} \\ x_{peak} &= -\frac{q_0}{N}, \;\; x_{peak} \leq \alpha L \end{aligned}$$

and

$$\begin{aligned} 0 &= -\frac{2 q_0}{k_R} - \frac{2 N}{k_R} x_{peak} \\ x_{peak} &= -\frac{q_0}{N}, \;\; x_{peak} \geq \alpha L \end{aligned}$$

Hence, the peak head is always at $-q_0/N$ irrespective of its location being at the left or right of $x = \alpha L$.
The maximum head in the cross section follows from

$$h_M = \sqrt{h_L^2 + \frac{q_0^2}{k_L N}}, \;\; x \leq \alpha L$$

$$h_M = \sqrt{h_R^2 + \frac{q_0^2}{k_R N} + \frac{2 q_0 L}{k_R} + \frac{N L^2}{k_R}}, \;\; \rightarrow x \geq \alpha L$$

we may now eliminate q_0 which follows from the heads being the same at $x = \alpha L, \;\; 0 \leq \alpha \leq 1$.

$$h_a^2 = h_L^2 - \frac{2q_0}{k_L}\alpha L - \frac{N}{k_L}\alpha^2 L^2 \tag{4}$$

$$h_a^2 = h_R^2 + \frac{2q_0}{k_R}(1-\alpha)L + \frac{N}{k_R}\left(1-\alpha^2\right)L^2 \tag{5}$$

$$h_L^2 - \frac{2q_0}{k_L}\alpha L - \frac{N}{k_L}\alpha^2 L^2 = h_R^2 + \frac{2q_0}{k_R}(1-\alpha)L + \frac{N}{k_R}\left(1-\alpha^2\right)L^2$$

$$\begin{aligned} 2q_0 L\left(\frac{(1-\alpha)}{k_R} + \frac{\alpha}{k_L}\right) &= \left(h_L^2 - h_R^2\right) - NL^2\left(\frac{\alpha^2}{k_L} + \frac{1-\alpha^2}{k_R}\right) \\ q_0 &= \frac{\left(h_L^2 - h_R^2\right) - NL^2\left(\frac{\alpha^2}{k_L} + \frac{1-\alpha^2}{k_R}\right)}{2L\left(\frac{\alpha}{k_L} + \frac{(1-\alpha)}{k_R}\right)} \end{aligned} \tag{6}$$

assume $k_R = k_L$ and $\alpha = 0.5$, $h_L = h_R$

$$q_0 = -\frac{NL}{2}$$

which is correct.

With equations 4, 5 and 6 we now have a suitable analytic test model with sufficient degrees of freedom to experiment with calibration. This model has been implemented in the file model.m. It has a selftest that produces the figure above.

Considering that in practice we have the heads and the recharge given, we have now at least 4 free parameters that would be uncertain in reality: k_L, k_R, α and z_B, the elevation of the bottom of the aquifer to adjust in order to make our model fit the data..

We will see to what extent we may calibrate some or all four of them given the heads in the ditches, the recharge and the head measurements.

If we compare this to a real case, our model may be in error because it misses essential ingredients, such as the entry resistance of the ditches. Maybe the aquifer is underlain by another aquifer with which it exchanges water through an aquitard. We might also misinterpret the distribution of the recharge, which may not be equal in practice due to different land use for instance. Of course, the conductivity may not be properly zoneable into two distinct zones, perhaps it has more zones or the conductivity varies in a more continuous fashion. Or the jump from one zone to the other is at a location different from what we assumed. We may as well have too many parameters in our model. This is always the case if there are more parameters than there are measurements. But it may also be that the transmissivity is so homogeneous that trying to estimate two conductivities does not make sense. In all cases where the model is a misconception of the real system, we may expect the optimized parameters compensate to the extent possible for the misconceptions in our model, and therefore, are wrong. Although in some circumstances, the model may show a reasonable fit with the date even with erroneous parameters, one may expect it to yield completely useless (uncertain) results when used for predictions under changed circumstances.

4 Use of the model

The model allows passing of a number of parameters. It will use default values for any unspecified parameter. How the input works can most easily be seen in the selftest function inside the model mfile.

The calibration is done with the script Calibration. It shows the four parameters and a usage through which parameters can be switched on and off. The initial parameter values are then specified. This is followed by computation of synthetic measurements, name y_M. These are generated by setting the parameters to the initial ones with some offset, after which random errors are added.

As soon as we have we can compute the sensitivities of the heads at all measurement locations with respect to all available parameters. This is done by computing the heads for the initial parameter set and then in

turn for all parameters while their value is changed a bit in turn. Then the initial heads area subtracted and the difference is divided by the parameter change that was used.

Having computed the sensitivities, we can compute the optimal parameters as was outlined above. This is done assuming the computed values at measurement locations varies linearly with the parameter values. We then compute the heads at the measurement locations using the updated parameters. We then have the difference between the optimized model and the measurements. Form this we can compute statistics, such as

- The covariance matrix of the parameters.
- The correlation matrix of the parameters.
- The standard error of the heads.
- The uncertainty of the parameters, assuming the linearity between computed values and parameters.

In this script we compute only one update of the parameters. In the real world of model calibration, we would start anew with the now updated parameter values. Updates are only necessary because of the non-linearity between model outcomes and parameter values. The parameter values have to be updates into the direction of the global optimum, assuming that such an optimum exists, which is to be made plausible by starting the calibration several times, each time starting with a substantially different initial parameter set. Each time the calibration should end with the same parameter end values. If not the calibration outcome is non-unique, i.e. the model errors are not sensitive to the parameters or there are more parameters than measurements, independent measurements.

It is interesting to see how many measurements are needed for a reasonable parameter optimization. It is also interesting to see how the uncertainty changes with the number of parameters that are included. In general, more parameters give a better fit but at the same time a higher parameter uncertainty.

```
%% Calibration simple analytic model
% TO 130619

clear variables

%% Parameters used in the calibration
usage =     [ 1     1       1     1 ];
parname = {'kL','kR','alpha','zB'};

use = usage~=0;

%% initial parameter values and perturbations
kL    = 1;        dkL = 0.05*kL;
kR    = 20;       dkR = 0.05*kR;
alpha = 0.25;     da  = 0.05*alpha;
zB    = -10;      dzB = 0.05*zB;
L     = 1000;

% Change of default of model parameters, add any (see model for possible
    parameters
defaults = {'L',L};

%% Number of observations and measurement locatons
Np    = 50;
xM    = unique(L * rand(Np,1));  % random locations between 0 and L

%% initial parameter vectors, usage above determines which ones are used.
p0  =[kL kR alpha zB]';  % initial paramter vector
dp0 =[dkL dkR da dzB];   % change applied to compute sensitivity (Jacobian)

```

```
30
31 %% Generate measurements
32 %
33 % The measurements are generated using the model with parameters a bit offset
34 % from the true parameters and with random errors added.
35 % Offset from true parameters
36 off_kL    =-0.2*kL;
37 off_kR    = 0.2*kR;
38 off_alpha = 0.2*alpha;
39 off_zB    = 0.0*zB;
40
41 % true parameters that will make the model equal to the measurements
42 pTrue = [kL+off_kL,kR+off_kR,alpha+off_alpha,zB+off_zB]';
43
44 % true model without random errors
45 yM = model('xM',xM,'kL',kL+off_kL,'kR',kR+off_kR,'alpha',alpha+off_alpha,'zB',
       zB+off_zB,defaults{:});
46
47 try   % try to load random errors (to keep them the same all the time)
48     load randErrors
49     if numel(yM)~=numel(randErrors)
50         error('Generating random errors');
51     end
52     fprintf('Random errors loaded.\n');
53 catch ME % renew random errors
54     fprintf('%s\nGenerating and saving random errors.',ME.message);
55     randErrors = 0.05*randn(size(yM));
56     save randErrors randErrors;
57 end
58
59 % simulated measurements
60 yM = yM + randErrors;
61
62 %% Initial parameters and model outcome
63 % The initial parameters for the calibration were given above.
64 % We have at most 4 parameters in this model kL kR alpha and zB.
65 % The active ones are selected with the usage near the top of this file.
66
67 %% Sensitivities computation (Jacobian)
68
69 % Model outcome for initial parameters
70 y0=  model('xM',xM,'kL',kL,'kR',kR,'alpha',alpha,'zB',zB,defaults{:});
71
72 %% perturbation of model parameter values
73 % Run model for all parameters in turn with a small value change
74 sp= [
75     model('xM',xM,'kL',kL+dkL,'kR',kR,'alpha',alpha,'zB',zB,defaults{:}),... %
           par1
76     model('xM',xM,'kL',kL,'kR',kR+dkR,'alpha',alpha,'zB',zB,defaults{:}),... %
           par2
77     model('xM',xM,'kL',kL,'kR',kR,'alpha',alpha+da,'zB',zB,defaults{:}),...  %
           par3
78     model('xM',xM,'kL',kL,'kR',kR,'alpha',alpha,'zB',zB+dzB,defaults{:})      %
           par4
```

```
79      ];
80
81  %% Compute Jacobian matrix (sensitivities)
82  J = bsxfun(@rdivide,bsxfun(@minus,sp(:,use),y0),dp0(use));
83
84  %% Optimal update of initial parameters
85  Inv     = (J'*J)^(-1);
86  B       = Inv*J';
87
88  dp      = B *(yM-y0); % dp  = (J'*J)^(-1)*J' * (yM-y0)
89
90  y       = y0+J*dp;      % end results, initial + update through parameter change
91  p       = p0(use)+dp;   % end results for parameters
92
93  %% Show results for comparison
94  fsz = 14; % fontsize plot
95
96  figure; axes('nextplot','add','fontsize',fsz);
97  xlabel('x [m]','fontsize',fsz);
98  ylabel('head [m]','fontsize',fsz);
99  title('Calibration: Head in measurement points','fontsize',fsz);
100
101 plot(xM,yM,'bx');  % model measured data
102 plot(xM,y0,'ro');  % model initial parameters
103 plot(xM,y ,'gs');  % model optimized parameters
104 legend('measured','initial','optmized');
105
106 %% Covariance matrix and other statistics
107 e       = (yM-y);                % heads errors, measured - computed
108 sigma   = std(e);                % errors in heads after calibration
109 Cov     = sigma^2*Inv;           % covariance matrix of the parameters
110 sigmaP  = sqrt(diag(Cov));       % std of the parameters
111 uncert  = 100*sigmaP./abs(p);    % uncertainty
112 Cor     = Cov./(sigmaP*sigmaP'); % correlation matrix of the parameters
113
114 %% Display results
115 display(Cov);
116 display(Cor);
117
118 %% Issue results for the parameters in readable format
119 fprintf('results: error = %.4g m\nUncertainty = 100*sigmaP/abs(p)\n',sigma);
120 fprintf('%10s%10s%10s%10s%10s%10s\n','parameter','pTrue','pInit','pEnd','
        sigmaP','uncert%');
121 k=0;
122 for i=find(use)
123     k=k+1;
124     fprintf('%10.4s',parname{i});
125     fprintf('%10.4g',pTrue(i));
126     fprintf('%10.4g',p0(i));
127     fprintf('%10.4g',p(k));
128     fprintf('%10.4g',sigmaP(k));
129     fprintf('%10.4g',uncert(k));
130     fprintf('\n');
131 end
```

```
132
133 %% Then next step is to change the initial parameters into the correct
134 % direction%% Simple calibration of a 1D analytic model
135 % TO May 2013
136
137 %% Intro
138 % The model is a 1D steady state phreatic head between two ditches at
139 % distance L. We have a fixed head in the ditches. The conductivity is kL
140 % between xL<alpha*L and kR voor x>alpha*L. The bottom of the aquifer is at
141 % zB = -13. Free parameters are kL kR alpha and zB.
142
143 %%
144 % We set up a calibration from scratch. Linearizing the relatin between
145 % model parameters and the outcomes at measurement locations and then
146 % updating the parameter vector such that we get a good fit between model
147 % and measurements. We use one step only. In this case we get generally a
148 % good fit. Clearly, because the relation between model parameters and
149 % heads is non-linear, we should repeat this procedure several times in
150 % real-world situation. Whether we reach a minimum or not, depends on the
151 % shape of the cost function and the method that is applied to reach such a
152 % minimum. The Marqardt-Levenber methed is the most applied non-linear
153 % search method, which is a weighted mix of steepest decend and the method
154 % shown here using the linearization and solving for the parameter vector
155 % update. The Marquardt-Levenberg methods is implemented in the Matlab
156 % function lsqnonlin (least squares non linear). This method is used in the
157 % other script called CalibNonLin in this same directory. It uses parObj to
158 % define parameters and is quite generic, so that with little effort much
159 % more complicated models may be calibrated.
160
161 %%
162 % In the current file we use a simple approach, with no fancy objects, so
163 % that every step is transparent.
164
165 % TO 130619
166
167 clear variables
168
169 %% Parameters used in the calibration
170 usage =     [ 1     0       1     1 ];  % set to 0 to switch off and 1 to switch
        on
171 parname = {'kL','kR','alpha','zB'};  % the switches pertain to these
        parameters
172
173 use = usage~=0;  % Use is now a logical array telling which of the parmeters
174                  % will be calibrated and which not.
175
176 %% Initial parameter values and perturbations
177 kL    = 1;        dkL = 0.05*kL;
178 kR    = 20;       dkR = 0.05*kR;
179 alpha = 0.25;     da  = 0.05*alpha;
180 zB    = -10;      dzB = 0.05*zB;
181
182 %% Change of default of model parameters, add any (see model for possible
        parameters
```

```
183 L      = 1000;
184 defaults = {'L',L};
185
186 %% Number of observations and measurement locatons
187 Np     = 50;
188 xM     = unique(L * rand(Np,1));  % random locations between 0 and L
189
190 %% Initial parameter vectors, usage above determines which ones are used.
191 p0  =[kL kR alpha zB]';  % initial paramter vector
192 dp0 =[dkL dkR da dzB];   % change applied to compute sensitivity (Jacobian)
193
194
195 %% Generate synthetic measurements
196 % The measurements are generated using the model in "model.m" with its
197 % parameters a bit offset from the "true" parameters and with random errors
        added.
198
199 %% Offset from true parameters (implemented in terms of multiplyers
200 off_kL     =-0.2*kL;
201 off_kR     = 0.2*kR;
202 off_alpha  = 0.2*alpha;
203 off_zB     = 0.0*zB;
204
205 %% True parameters that will make the model equal to the measurements
206 pTrue = [kL+off_kL,kR+off_kR,alpha+off_alpha,zB+off_zB]';
207
208 %% True model without random errors
209 yM = model('xM',xM,'kL',kL+off_kL,'kR',kR+off_kR,'alpha',alpha+off_alpha,'zB',
        zB+off_zB,defaults{:});
210
211 %% Add synthetic random errors to yM if necessary
212 try   % try to load random errors (to keep them the same all the time)
213     load randErrors
214     if numel(yM)~=numel(randErrors)
215         error('Generating random errors');
216     end
217     fprintf('Random errors loaded.\n');
218 catch ME  % renew random errors
219     fprintf('%s\nGenerating and saving random errors.',ME.message);
220     randErrors = 0.05*randn(size(yM));
221     save randErrors randErrors;
222 end
223
224 % simulated measurements
225 yM = yM + randErrors;
226
227 %% Initial parameters and model outcome
228 % The initial parameters for the calibration were given above.
229 % We have at most 4 parameters in this model kL kR alpha and zB.
230 % The active ones are selected with the usage near the top of this file.
231
232 %% Sensitivities computation (Jacobian)
233
234 % Model outcome for initial parameters
```

```
y0=  model('xM',xM,'kL',kL,'kR',kR,'alpha',alpha,'zB',zB,defaults{:});

%% perturbation of model parameter values
% Run model for all parameters in turn with a small value change
sp= [
     model('xM',xM,'kL',kL+dkL,'kR',kR,'alpha',alpha,'zB',zB,defaults{:}),... % par1
     model('xM',xM,'kL',kL,'kR',kR+dkR,'alpha',alpha,'zB',zB,defaults{:}),... % par2
     model('xM',xM,'kL',kL,'kR',kR,'alpha',alpha+da,'zB',zB,defaults{:}),...  % par3
     model('xM',xM,'kL',kL,'kR',kR,'alpha',alpha,'zB',zB+dzB,defaults{:})      % par4
     ];

%% Compute Jacobian matrix (sensitivities)
J = bsxfun(@rdivide,bsxfun(@minus,sp(:,use),y0),dp0(use));

%% Optimal update of initial parameters
Inv     = (J'*J)^(-1);
B       = Inv*J';

dp      = B *(yM-y0); % dp  = (J'*J)^(-1)*J' * (yM-y0)

y       = y0+J*dp;     % end results, initial + update through parameter change
p       = p0(use)+dp;  % end results for parameters

%% Show results for comparison
fsz = 14; % fontsize used in plot

figure; axes('nextplot','add','fontsize',fsz);
xlabel('x [m]','fontsize',fsz);
ylabel('head [m]','fontsize',fsz);
title('Calibration: Head in measurement points','fontsize',fsz);

plot(xM,yM,'bx');  % model measured data
plot(xM,y0,'ro');  % model initial parameters
plot(xM,y ,'gs');  % model optimized parameters
legend('measured','initial','optmized');

%% Covariance matrix and other statistics
e       = (yM-y);                % heads errors, measured - computed
sigma   = std(e);                % errors in heads after calibration
Cov     = sigma^2*Inv;           % covariance matrix of the parameters
sigmaP  = sqrt(diag(Cov));       % std of the parameters
uncert  = 100*sigmaP./abs(p);    % uncertainty
Cor     = Cov./(sigmaP*sigmaP'); % correlation matrix of the parameters

%% Display results
display(Cov);
display(Cor);

%% Issue results for the parameters in readable format
fprintf('results: error = %.4g m\nUncertainty = 100*sigmaP/abs(p)\n',sigma);
```

```
fprintf('%10s%10s%10s%10s%10s%10s\n','parameter','pTrue','pInit','pEnd','
    sigmaP','uncert%');
k=0;
for i=find(use)
    k=k+1;
    fprintf('%10.4s',parname{i});
    fprintf('%10.4g',pTrue(i));
    fprintf('%10.4g',p0(i));
    fprintf('%10.4g',p(k));
    fprintf('%10.4g',sigmaP(k));
    fprintf('%10.4g',uncert(k));
    fprintf('\n');
end

%% The next step
% the next step is to change the initial parameters into the correct direction
% This is done in mfCalib, using Matlab's lsqnonlin solver.
```

5 Non-linear weighted least squares optimization

The example on the mfLab site, mflab/examples/CIE5440-Geohydrology2/calibration/CalibNonLin carries out a least squares non-linear parameter optimization. It uses the matlab function lsqnonlin, which takes the parameter vector and yields the final parameters with optionally additional statistics. It also takes a pointer to the function that computes the error vector based on the parameters passed. This function is embedded in the function modelWrapper as it takes the parameter vector as input converts this into the actual promoters, launches the groundwater model, and when finished, extracts the error vector at the measurement locations, as required by the lsqnonlin function. Notice that this modelWrapper can be made arbitrarily complex using any combination of external models of necessary, as long as it accepts the parameter vector and yields the error vector.

It is nice to experiment with this model, by fixing parameters or just calibrating them. If all parameters are calibrated, then the model is completely over-defined. It will generate a perfect fit but with useless parameters. One can adapt the "measurements" by setting the parameter values in the "parTruer struct" and deleting meas.mat on disk. A new set of "measurements" will then be generated and stored to disk. The calibration will start from the initial parameters stored in "Par". Choose which parameters are to be calibrated by setting the userFlag (right most value for each parameter) to 1. The log flag indicates that the the log of the actual parameter will be optimized instead of its value. This is useful for parameters that cannot be smaller than zero and have an unlimited upper bound, at least in principle. Using log parameters is the same as multiplying by a factor or dividing by a factor instead of adding or subtracting a value. In this example, the parameter alpha (location of the boundary between the K_L and K_R zone, and z_B the elevation of the bottom of the aquifer are not log-transformed. They could be, by for instance changing alpha to

$$\alpha = \frac{atan\,(p)}{\pi}$$

so that when p varies between $-\infty$ and $+\infty$, then α varies between ± 0.5. And instead of the elevation of the bottom of the aquifer, one could choose aquifer depth as a variable that lends itself to be translated to its logarithm during the calibration. It is a good exercise to try that. An alternative is

$$\alpha = \frac{asin\,(p)}{\pi}, \quad -1 \leq p \leq 1$$

It is also instructive to force parameters to be 100% correlated. For instance, if $\alpha \approx 1$ then there is only one k zone, i.e. k_L. If then all parameters except N are fixed, we have a cross sectional model with one k and precipitation as its only degrees of freedom. Because the head then depends on N/k the parameters N and k will be 100% correlated. This is an extreme case. In this situation, there is no unique solution for either N

or k but only for N/k. The correlation will be directly observable form the outcome. Also the singular values resulting from the singular value decomposition of the Jacobian matrix (Jacobian is the sensitivity matrix), immediately reveals the rank of the Jacobian:

$$[U, S, V] = svd\,(J, 0)$$

such that

$$J = USV^T$$

and so

$$\begin{aligned} e &= USV^T p \\ &= U\left(S\left(V^T p\right)\right) \end{aligned}$$

S is the diagonal matrix with the singular values, and $V$the matrix of singular vectors, which rotate the parameter vector in the parameter space to an equivalent space where all components are orthogonal (i.e. mutulally independent, that is, with zero mutual correlation) parallel to the main axis of the Jacobian. This roate parameter vector is then $p_R = V^T p$. Because S is a diagonal matrix it stretches p_R to Sp_R. This is because premultiplying with a diaginal matrix is equivalent with multiplying the rows of the array or vector thereafter. This immediately clarifies that in rotated parameter space, the first principal component if p_R is multiplied by the first and biggest singular value and so on. Hence, the singular values can be considered of utmost importance in defining how many and which parameters can actually be optimized in a given setting.

Hance these singular vectors or rather the components of the rotated (aligned) parameter vector p_R can be seen as principal components, corresponding to the singular values, where the singular value indicates their importance. The number of singular values that are essentially different from zero are the rank of the Jacobian, which matches the number of parameters (i.e. principal components) that can be uniquely optimized. The singular vectors V then show how each of them is composed of the original parameters, and so, which parameter dominates each of them.

In the case of the explained example, in which there is only a single component (i.e. N/k) and the original parameters N and k are 100% correlated, we see that there is only one principal component essentially different form zero and that the contribution of the two composing parameters (N and k respectively) are both equal, and of opposite sign. Their final value are different form the true parameters by which the measurements have been computed, but their N/k ratio is exactly correct. Also, the fit between the model and the measurements is completely correct. This issue of non-uniqueness due to correlation between parameters is a very important topic in model calibration and optimization in general. The more parameters (more than required) the better the fit, but greater the uncertainty of the determined parameters. In this case, the uncertainty is infinite.

The statistics are shown in the box below. S = singular values; only the first one is essentially different from zero. V is singular vectors, where the top row is the contribution of the first parameter (N) and the second row that of the second parameter, (k_L). The initial, true and final parameter values are also shown. The demonstrate that the final parameters are substantially off compared to the true ones. However, the ration N/k matches exactly that of the true parameters. Further, the correlation between the parameters is 100% and the covariances are extremely high, in fact, the uncertainties of the determined parameters are, in fact, infinite (see column stdP). The message is, that if you know the behavior of your system, for example from approximate analytical solution, one may choose parameters possibly in a way that reduce correlation as much as possible. In this case it would be calibrating N/k and not N and k separably. Calibrating N/k can be done with very high accuracy. The other message is, that if parameters are highly correlated in a model setting, then it pays off to seek independent information for at least one of them, so here either N or k_L, so that this a priori information can be included in the calibration. The calibration then becomes Bayesian, solving the unknown parameters, given we know more about each of them, which can be expressed explicitly in the cost function that is minimized:

$$F = \mathbf{e}^T \mathbf{w}\, \mathbf{e} + \mathbf{p}^T \mathbf{q}\, \mathbf{p}$$

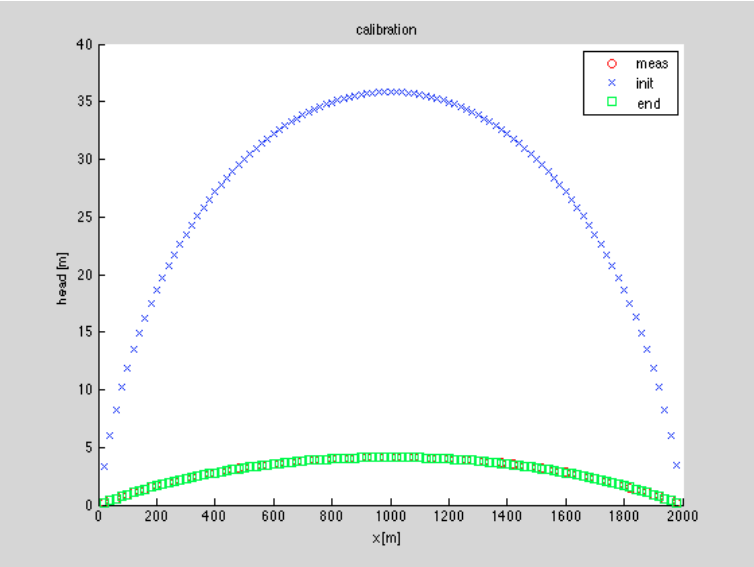


Figure 3: Large difference between initially model-computed heads and the perfect fit after the calibration.

where e are the errors such as head measured minus head computed with w a weighting matrix. And p is the difference between the a priori set parameter and the optimized parameter, also weighted, in this case with a weight matrix q. To scale the two terms properly , weighting should be done by the uncertainty of the errors. If the measurement errors are mutually independent, then w will be a diagonal matrix with values $1/\sigma_e^2$. If the paramour errors are mutually independent, then q a diagonal matrix with values $1/\sigma_p^2$, or rather p is the full sized inverse of the covariance matrix which as $1/\sigma_p^2$ values at its diagonal and is the linear approximation of the true covariance matrix. Using the $1/\sigma^2$ for error weights normalizes them irrespective of their dimension. This is also the result of a maximum likelihood analysis.

The cost function as defined suffers from the large difference between the number of parameters and the number of measurements, tending to favor the measurement error over the parameter errors. For instance if we have n measurements, the term $\mathbf{e}^T\mathbf{w}\,\mathbf{e} \approx n$. This imbalance can be compensate by a weighting factor λ

$$F = \mathbf{e}^T\mathbf{w}\,\mathbf{e} + \lambda\mathbf{p}^T\mathbf{q}\,\mathbf{p}$$

For instance, if the two terms are to be given equal importance, λ can be chosen to $\lambda = n_m/n_p$, with with n_m the number of measurements and n_p the number of parameters.

```
[U,S,V] = svd(J,0);

S =

   38.0743         0
         0    0.0001


V =

   -0.7071    0.7071
    0.7071    0.7071


Cov =

   1.0e+05 *
```

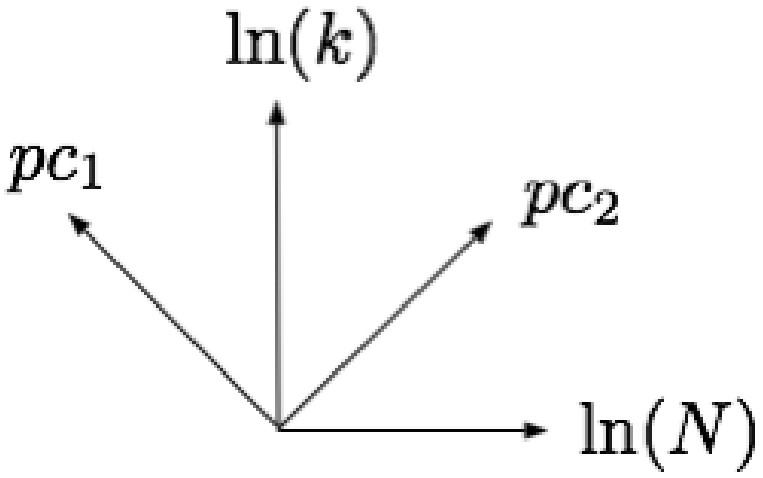
```
    2.3072    2.3072
    2.3072    2.3072


Cor =

    1.0000    1.0000
    1.0000    1.0000

  Parameter  useFlag  logFlag       oldPar      truePar       newPar         stdP
          N        1        1          0.2       0.001    0.0003359   2.01e+208
         kL        1        1          100          10        3.359   2.02e+208
         kR        0        1           10          10           10
      alpha        0        0            1           1            1
         zB        0        0          -10         -10          -10
```

Figure 4 shows the two principal components in the parameter space spanned by the two variables, i.e. the $\ln(N)$ and $\ln(k)$. The vectors at 45o on log scale implies that the heads depend on k/N or N/k respectively. The huge difference between the first and the second singular value, that latter being essentially zero relative to the first, implies that the problem depends on N/k only. In conclusion to estimate factors that determine the outcome of a model instead of individual parameters, it makes sense to first make sure all parameters are log-transformed during the calibration and then look at their correlation.

In a seond example we calibrate N, k_L and k_R while fixing $\alpha = 0.5$ both in true parameters. The results are in the box below.

```
Par = parObj({  % initial parameters
    % NAME VALUE LB   UB   logFlag useFlag
     'N'      0.2   0.0001  0.5  1  1
     'kL'     100     0.1   500    1   1
     'kR'      10     0.1   500    1   1
     'alpha'  0.5     0      1    0  0
     'zB'     -10    -5   -40    0  0
     });

parTrue = parObj({
         'N'        0.001
         'kL'         10
         'kR'         10
         'alpha'     0.5
         'zB'        -10
         });

S =

   32.9806         0         0
         0    4.9668         0
         0         0    0.0000


V =

   -0.8165    0.0004    0.5774
    0.4079   -0.7073    0.5774
    0.4086    0.7069    0.5774
```

Figure 4: Principal components of the 2 parameter problem where head depends on N/k precisely. The singular value of the first principal compoment was about 38 and that of the second 10^{-4}, hence, essentially zero.




```
      pEnd         pEnd

ans =

   1.0e+04 *

   -0.0003   -3.2228
   -0.0000   -3.2230
    0.0002   -3.2230


Cov =

   1.0e+10 *

    2.1077    2.1077    2.1077
    2.1077    2.1077    2.1077
    2.1077    2.1077    2.1077


Cor =

    1.0000    1.0000    1.0000
    1.0000    1.0000    1.0000
    1.0000    1.0000    1.0000

   Parameter  useFlag  logFlag       oldPar      truePar       newPar         stdP
           N        1        1          0.2        0.001     0.008581          Inf
          kL        1        1          100           10        85.65          Inf
          kR        1        1           10           10        85.91          Inf
       alpha        0        0          0.5          0.5          0.5
          zB        0        0          -10          -10          -10
```

Because of the correlation between N and k_L as well as k_R, the final parameters are way off their true

values, and their uncertainty is infinite, yet the fit of the model and the final data is perfect. The matrix S indicates that their are two parameters that could be optimized, i.e. two principal components. Yet the correlation between the optimized paramters is 100% as shown in the Cor matrix. This is also reflected in the composition of the principal components as shown in the vector V. Assuming that the the errors depend on both N/k_L and N/k_R, we suspect that the error depend in fact on

$$e = a\frac{N^2}{k_L k_R}$$

Working with log parmeters we have

$$e = \ln(a) + 2\ln(N) - \ln(k_L) - \ln(k_R)$$

So that $\partial e/\partial \ln(N) = 2,\ \partial e/\partial \ln k_L = -1,\ \partial e/\partial \ln k_R = -1$.

Hence, the principal compoment of this factor, assumed to be an essental parameter of the problem on hand has the following components, where $\sqrt{6}$ is used to scale its length to 1.

$$pc_1 = \left\{ \begin{array}{c} 2 \\ -1 \\ -1 \end{array} \right\} / \sqrt{6} = \left\{ \begin{array}{c} 0.8165 \\ -0.4082 \\ -0.4082 \end{array} \right\} \tag{7}$$

Clearly the sign of the entire vector is immaterial as the only point that matters it the relative values of its components. As can be seen these values correspond to the numerical values in matrix V. This relation, i.e. $N^2(k_L k_R)$ was not directly clear from the analytical solution of our model because we compute it in two steps, first q_0 and then use q_0 in the equation for either the left size or the right size of the separateion line of the k_L and k_R zone as given by parameter α. Nevertheless, because the head depends on both N/k_L and N/k_R is perfectly obvious to assume it depends on $N^2(k_L k_R)$ as long as $\alpha = 0.5$, in which case k_L and k_R are equally important. The more α deviates from the value 0.5, the more one of the conductivities will dominate over the other and the more the two values of 0.4082 in equation 7 will differ. For instance if $\alpha = 0.75$, k_L becomes more important than k_R

Parameter	useFlag	logFlag	oldPar	truePar	newPar	stdP
N	1	1	0.2	0.001	0.0109	Inf
kL	1	1	100	10	109.1	Inf
kR	1	1	10	10	109.1	Inf
alpha	0	0	0.75	0.75	0.75	
zB	0	0	−10	−10	−10	

S =

33.0703	0	0
0	6.7076	0
0	0	0.0000

V =

−0.8136	−0.0685	0.5774
0.4662	−0.6703	0.5774
0.3474	0.7389	0.5774

If we take $\alpha = 0.9$ the difference becomes greater, while the values for both N (top value) and k_L (second value) come closer to the system with only one parameter that was illustrated first.

Parameter	useFlag	logFlag	oldPar	truePar	newPar	stdP
N	1	1	0.2	0.001	0.009076	Inf
kL	1	1	100	10	90.82	Inf
kR	1	1	10	10	90.66	Inf

alpha	0	0	0.9	0.9	0.9
zB	0	0	−10	−10	−10

S =

34.5945	0	0
0	5.2645	0
0	0	0.0000

V =

−0.7771	−0.2505	−0.5774
0.6055	−0.5477	−0.5774
0.1716	0.7983	−0.5774

With $\alpha = 0.99$ the values are almost the same as in our first example where only k_L and N mattered. It can be seen that paramter 3, i.e. k_R plays no role of any importance in the composition of the first principal component. Hence it the principal factor in the model is $\approx N/k_L$. Yet the second principal component is no longer perfectly negligible. It has almost the same numerical values as the first principal component in our second example. So, the second principal component indicates an indepent parameter equal to $\approx k_R^2/\left(k_R N\right)$ but it is hard to tell without deriving the complete analytical solution for this case.

Parameter	useFlag	logFlag	oldPar	truePar	newPar	stdP
N	1	1	0.2	0.001	0.01496	Inf
kL	1	1	100	10	149.7	Inf
kR	1	1	10	10	150	Inf
alpha	0	0	0.99	0.99	0.99	
zB	0	0	−10	−10	−10	

S =

37.5870	0	0
0	0.7357	0
0	0	0.0000

V =

−0.7160	−0.3924	−0.5774
0.6978	−0.4239	−0.5774
0.0182	0.8163	−0.5773

In conclusion, singular value decomposition and principal component analysis helps a lot in decuding essential relationships between system parameters. Even though the the original parameters may correlate to as much as 100%, so that thet cannot and are not determined corretly, the components, coprehensive parameters factors may still be deduced and used. With added prior information about a sufficient number of paramters (n-1) in a factor consiting of n parameters, the individual paramters may be determined. Adding prior information, in fact turns the Jacobian to the space spanned by the original parameters, the stronger, the less the uncertainty about the orginal parameters is.

6 Relation between the factors in the singular value decomposition

The relation between errors (or deviations from the model as a function of the deviation of the parameters from their optimal value is given by

$$e - e_0 = J\left(p - p_0\right)$$

with all p being the logarithm of the original parameters, the difference $\Delta p = p - p_0$ implies a given relative difference. Wirting conveniently

$$\Delta e = J\Delta p$$

We may now use the svd as follows

$$\Delta e = USV^T\Delta p$$

And only considere the singular values that are essentially different form zero relative to the first one. This may result in a large reduction of parameters, as the width of U and the size of S will now equal the rank of the Jacobian, i.e. the highest number of parameters that can be detemined given the Jacobian.

Because the singular vectors V are ortogonal and of unit length, $\Delta p_r = V^T\Delta p$ is a rotation of Δp so that the original parameter vector is now expressed in the coordinates of the system with its axes parallel to that of the principal components. We may write

$$\Delta e = \left(US\right)\Delta p_r$$

and considereUS as the Jacobian of the principal components

$$\Delta e = J_r\Delta p_r$$

This equation is fundamental, as it is the optimal sensitivity equation, such that all parameters, now being principal components, are ortogonal, that is mutually independent. Moreover the svd procedure orders the columns of the Jacobian such that the the first one of J_r is most important. This can be seen as follows. The square matrix S is diagonal and of the size of the number of non-zero singular values, while the singular values are ordered form must highest to lowest along the diagonal. Multiplying U by S implies multiplying each column of U with its corresponding singular value, the first one being the largest and the last one the smallest. Hence the first column of J_r is dominant. It can also be read as follows. The product SV^Tp implies first rotation to Sp_r and then multiplying the vector p_r by S. Post multiplying a diagonal matrix with another matrix, implies multiplygin its rows with the corresponding diagonal value, hence the first parameter in p_r is most important. Also, SV^T multiplies each row, i.e. each singular vector with its corresponding singular value, hence, the first singular vector is most important.

The length of J_r equals the length of the measurement locations, i.e. the length of e, while its width equal the number of the essentially non-zero principal components.

7 Relation between singular values and eigen values

The relation between eigen values and singular values is direct, which can be seen as follows.

Any square positive definite matrix X can be factored as follows

$$X = VEV^T$$

where V is a matrix of eigen vectors each of length 1 and together spanning the paramter space. E is the diagonal matrix of eigen values, ordered such that $E(1,1)$ is the largerst and $E(end, end)$ the lowest.

The matrix J^TJ is such a square positive definite matrix, at least as it has full rank, that is, no non-zero eigen values. However, with very small eigen values, high loss of accuarcy may occur in the equation 8 below. A rank less than the number of parameters yields some zero eigen values.

For instance the over-determined system

$$e = Jp$$

that is, the vector of measurements e is much longer than the vector of parameters, is solved in a least squares sense by

$$\begin{aligned} J^T e &= J^T J p \\ p &= \left(J^T J\right)^{-1} J^T e \end{aligned} \tag{8}$$

On the other hand, any matrix, regardless of its size, may be factored by the singular value decomposition as follows

$$X = USW^T$$

in which W are the singular vectors, similar to those in the eigen vector/values decomposition, i.e. of unit length and mutually ortogonal, and S is the diagonal matrix with singular values with, again, the largest being the first and the smallest the last. U is a matrix having length equal to that of X, i.e. in general equal to the number of measurments, m.

If the dimensions (size) of X are m, n then that of S must be n, k adn that of W must be k, n where k could have any value. However, some or nany of the singular values may be zero. The number of non-zero singular value equal the rank of X, that, is the number of column vectors of X that are mutually independent. Because we are only interested in non-zero singular values, k equals the number of non-zero singular values and the retained portion of U has full length but the a width equal to the number of non-zero singular values. This may reduce its size tremendously. Also S may thus become must smaller than the original number of columns in X. Vinally the singular vectors W have the original length, but there are only k of them. Hence they span only a space of k independent components, namely those that correspond to non-zero singular values. Together they form an ortoginal space of principal components, leaving out a null-space that cannot be mapped by the given data and parameters.

Hence if we factor the Jacobina itself using the singular value decomposition we obtain

$$J = USV^T$$

while with eigen value/vector decomposition we obtrained

$$J^T J = VDV^{'T}$$

and so

$$JJ^T = \left(VDV^T\right)^T = VDV^T = J^T J$$

so

$$JJ^T = \left(VDV^T\right)^T = VDV^T = USW^T WSU^T = US^2U^T$$

Hence, the singular values are the square root of the eigen value.

8 Weighted least squares and maximum likelihood estimation

The text above describes least squares optimization, which is generally most applies in regression analysis. Least squares estimation yields unbiased results. Nevertheless, it may be necessary to apply weighted least squares estimation, when the value of data varies much or to deal with the contribution of different data types to the cost function that is to be minimized. Different data types like heads, flows and concentrations are generally not compatible in value magnitude, dimension and importance for the model. A flow or flux

may add much more to the stability of a calibration than heads. A flux is always necessary to obtain a unique solution. In the test model we chose the recharge to fulfill that role. It generally makes no sense to try to calibrate the recharge. This is because the heads depend on the ratio of the recharge over the aquifer transmissivity and without recharge, transmissivity cannot uniquely be determined.

One issue not discussed above is weighted least squares or maximum likelihood optimization. The latter contains information allowing to tradeoff between more parameters with a better fit and fewer parameters but with lower uncertainty.

9 Exercise

Is is a good exercise to embed this model in a global optimization that will update the parameters until the parameter values have converged to stable end values. This has been done in model2. This can be readily done in Matlab using Marquardt Levenberg optimization function. It may also be done with external programs like UCODE or PEST. HOwever, for instruction purposes, it may be most convenient to do the entire calibration within Matlab. It helps in understanding what these dedicated calibrating methods actually do. But more importantly one should grasp a feeling for the relation between parameters, model fit and uncertainty.

References

[1] Cooley, R.L. and R.L. Naff (1990) Regression Modeling of Ground-Water Flow. Techniques of Water Resources Investigations of the United States Geological Survey. Book 3, Applications of Hydraulics, 228pp.

[2] Doherty, J (2000) The Pest Manual. Model-Independent Parameter Estimation and Uncertainty Analysis. See for a list of literature references http://www.pesthomepage.org/Some_References.php.

[3] Doherty, J (2013) See http://www.pesthomepage.org/Downloads.php There are tutorials and downloads on that site. The software is free and internationally intensively used for calibration of models of any type.

[4] Hill, M. and C.R. Tiedeman (2005) Effective Groundwater Model Calibration with Analysis of Data, Sensitivities, Predictions and Uncertainty. Wiley 2007, 13- 978-0-471-77636-9. 455p

[5] Methods and Guidelines for Effective Model Calibration. USGS Water Resources Report 98-4005. With applications to UCODE, a computer code for universal inverse modeling, and MODFLOWP, a computer code for inverse modeling with MODFLOW.

[6] Olsthoorn, T.N (1998) Groundwater modeling: calibration and the use of spreadsheets. PhD thesis. TUDelft, 300pp

[7] Stark, H. andJ.W. Woods () Probability, Random Processes and Estimation Theory for Engineers. Second Edition, Prentica Hall, 1994. ISBN 0-13-728791-7. 617 pp.