User Guide for Spowtd v0.6.0

Alex Cobb

This is the user guide for Spowtd, which implements the scalar parameterization of water table dynamics described in Cobb et al. [1] and Cobb and Harvey [2].

1 The steps of scalar parameterization

Scalar parameterization involves these essential steps:

1. Load water level, precipitation and evapotranspiration data;
2. Identify dry intervals and storm intervals;
3. Match intervals of rising water levels to rainstorms;
4. Construct a master rising curve;
5. Construct a master recession curve;
6. Fit a preliminary specific yield function to the master rising curve;
7. Jointly fit a specific yield and a conductivity (equivalently, transmissivity) function to the master rising and recession curves.

2 The spowtd script

The `spowtd` script provides a command-line interface to perform calculations with Spowtd.

2.1 Dependencies

Running the script requires Python 3 and the Python packages Matplotlib, Numpy, and Pytz.

2.2 Using the script

The `spowtd` script has these subcommands (typically run in this order):

- `spowtd load`: Load water level, precipitation and evapotranspiration data.
- `spowtd classify`: Classify data into storm and interstorm intervals.
- `spowtd set-zeta-grid`: Set up water level grid for master curves.
- `spowtd recession`: Assemble recession curve.
- `spowtd rise`: Assemble rise curve.
- `spowtd plot`: Plot data.
- `spowtd set-curvature`: Set site curvature.
- `spowtd simulate`: Simulate data rise curve, recession curve, or rising and receding intervals.
- `spowtd pestfiles`: Generate input files for calibration with PEST.

The first step is to load the precipitation, evapotranspiration and water level data. The input text files must be in an UTF-8-compatible encoding (ASCII is fine). The time zone is stored with the dataset and will be used in plots (all times are stored internally as UNIX timestamps). For example, to load data into a new dataset file called `ekolongouma.sqlite3`:

```
spowtd load ekolongouma.sqlite3 \
  -vvv \
  --precipitation src/precipitation_Ekolongouma.txt \
  --evapotranspiration src/evapotranspiration_Ekolongouma.txt \
  --water-level src/waterlevel_Ekolongouma.txt \
  --timezone Africa/Lagos
```

The verbosity flags (`-vvv`) are not required; they cause the script to report more on what is being done.

Next, classify the water level and precipitation time series into storm and interstorm intervals based on thresholds for rainfall intensity and rates of increase in water level. For example, this command classifies intervals with precipitation of at least 4 mm / h as storms, and intervals in which the water level is increasing at a rate of least 8 mm / h as storm response.

```
spowtd classify ekolongouma.sqlite3 \
  -vvv \
  --storm-rain-threshold-mm-h 4.0 \
  --rising-jump-threshold-mm-h 8.0
```

(For details on how Spowtd matches storms with rises in water level, see Appendix A.)

At this stage the classification can be plotted. A basic interactive plot showing the classified water level and precipitation time series can be produced with:

```
spowtd plot time-series ekolongouma.sqlite3
```

An additional panel showing evapotranspiration is plotted if the `-e` or `--plot-evapotranspiration` flag is passed. The parts of the water level time series marked as interstorms are on a light red background, and the parts of the water level time series marked as storm response are on a light green background. The parts of the precipitation time series marked as storms are on a light blue background. You can pan in the plot with the right mouse button and zoom with a left mouse button, or use the magnifying glass to zoom in. You can revert to earlier zoom and pan values with the arrow buttons.

Adding `-f` or `--flags` highlights the parts of the water level time series that have been classified as storm response and interstorms, and the parts of the precipitation time series

```
spowtd plot time-series ekolongouma.sqlite3  -f
```

The rising intervals are highlighted in blue, intervals with rising intervals that could not be matched to rain storms are highlighted in magenta, and rain storms are highlighted in red.

The next step is to establish a uniform grid for water levels. This grid is used when storm and interstorm intervals are assembled into rising and recession curves.

```
spowtd set-zeta-grid -vvv ekolongouma.sqlite3
```

The next two steps assemble the recession and rise curves:

```
spowtd recession -vvv ekolongouma.sqlite3
```

```
spowtd rise -vvv ekolongouma.sqlite3
```

If desired, the correlated errors produced by imprecision in recharge depth measurement can be taken into account when assembling the rise curve (see Appendix B, “Event weighting in rise analysis”). To do so, pass in a relative weight to assign to these errors, vs. direct errors in water level measurement, via `--recharge-error-weight`:

```
spowtd rise -vvv ekolongouma.sqlite3 --recharge-error-weight=1e3
```

To examine the error covariance matrix used in the rise curve assembly, append the `--dump-covariance` flag and the error covariance matrix will be written to a file (default is standard output) as JSON.

The recession and rise curves are now assembled, and can be plotted.

```
spowtd plot recession ekolongouma.sqlite3
```

```
spowtd plot rise ekolongouma.sqlite3
```

These plots can be interacted with in the same way: left mouse button to pan, right mouse button to zoom, disk icon to save.

3 Parameterization

Parameters are provided to `spowtd` in YAML format.

Currently two types of parameter sets are supported: (1) Cubic spline for specific yield, piecewise linear for the logarithm of conductivity; and (2) The PEATCLSM parameterization.

The spline parameterizations look like this:

```
specific_yield:
  type: spline
  zeta_knots_mm:
    - -291.7
    - -183.1
    - -15.74
    - 10.65
    - 38.78
    - 168.3
  sy_knots:  # Specific yield, dimensionless
```

```
    - 0.1358
    - 0.1671
    - 0.2541
    - 0.2907
    - 0.2892
    - 0.6857
transmissivity:
  type: spline
  zeta_knots_mm:
    - -291.7
    - -5.167
    - 168.3
    - 1000
  K_knots_km_d:  # Conductivity, km/d
    - 5.356e-3
    - 1.002
    - 6577.0
    - 8.430e+3
  minimum_transmissivity_m2_d: 7.442  # Minimum transmissivity, m2/d
```

and the PEATCLSM parameterizations look like this:

```
specific_yield:
  type: peatclsm
  sd: 0.162  # standard deviation of microtopographic distribution, m
  theta_s: 0.88  # saturated moisture content, m^3/m^3
  b: 7.4  # shape parameter, dimensionless
  psi_s: -0.024  # air entry pressure, m
transmissivity:
  type: peatclsm
  Ksmacz0: 7.3  # m/s
  alpha: 3  # dimensionless
  zeta_max_cm: 5.0
```

(the text following each parameter, after the #, is a comment and invisible to spowtd).

The specific yield and transmissivity curves can be plotted with

```
  spowtd plot WHAT parameters.yml WATER_LEVEL_MIN_CM WATER_LEVEL_MAX_CM
```

where `WHAT` is one of `specific-yield`, `conductivity` or `transmissivity`, `parameters.yml` is a YAML file containing hydraulic parameters, and the last two arguments specify the range of water levels over which to plot the curve.

The plotting commands `plot rise`, `plot recession` and `plot time-series` support a parameter `-p`, `--parameters`; if a YAML file containing hydraulic parameters is passed to one of these commands, the corresponding plot (rising curve, recession curve, rising and receding intervals) is simulated using those parameters.

The simulated curves and corresponding data can be obtained as text using `spowtd simulate`

`WHAT data.sqlite3 parameters.yml` where `WHAT` is `rise`, `recession`, or `intervals`. These commands write simulated data, water level data, and / or residuals to an output file (standard output by default) as delimited text. For example,

```
spowtd simulate rise ekolongouma.sqlite3 parameters.yml
```

reads data from `ekolongouma.sqlite3` and parameters from the file `parameters.yml` and writes the assembled and simulated rise curves to standard output. The curves are simulated over the range of water levels in the data, so plots from different SQLite files may look different even if the parameters are identical.

To simulate (or plot) recession requires setting the large-scale curvature of the site. The command

```
  spowtd set-curvature ekolongouma.sqlite3 1.0
```

sets the site curvature to 1 m/km/km, whereafter

```
  spowtd simulate recession ekolongouma.sqlite3 parameters.yml
```

simulates the water table recession.

4 Calibration with PEST

The simulation scripts make it possible to calibrate the specific yield and transmissivity functions against rise and recession of the water level using the PEST software package and tools for model-independent parameter estimation and uncertainty analysis. It should also be possible to calibrate using PEST++, which is designed to have the same text-based interface, by following a similar procedure.

PEST is a highly configurable set of tools. One of its strengths is that it is possible to start with a fairly simple approach and incorporate more sophisticated functionality as it is needed. As an introduction, we illustrate calibration of specific yield parameters against the rise curve.

For a calibration with PEST, you need to create five text files:

1. A PEST control file (`.pst`), which configures how PEST will perform the calibration (including identifying the other files used during calibration);
2. A parameter template file (`.tpl`), into which PEST will substitute parameter values in a format that can be read by Spowtd;
3. An output template file, or PEST "instruction file" (`.ins`), which teaches PEST how to extract "observations" from `spowtd simulate` output;
4. A vector of initial parameters (`.par`) to start the calibration; and
5. A script to execute the rise simulation.

The first step is to create the PEST control file (`.pst`) following the PEST documentation [3]. For a PEATCLSM parameterization, the control file will describe the four PEATCLSM parameters for specific yield (`sd`, `theta_s`, `b`, and `psi_s`), each with their own parameter group. The control file must also include an "observation data" section with a single line giving mean dynamic storage for each water level in the rise curve. The "model command line" section must provide the command line needed to run the script to generate the rise curve. The script itself can be, for example, a bash script that calls `spowtd simulate rise`. Finally, the "model input/output" section specifies the path to the parameter template file, the path to the the parameter file that PEST will create by substituting parameter values into the template, the path to the instruction file (`.ins`) that PEST uses to interpret the simulation output, and the path to the output file created by a single run of the simulation script.

The parameter template file (`.tpl`) and the parameter vector file (`.par`) are created by replacing values in a Spowtd YAML parameter file by placeholders, as described in the PEST documentation. In the case of calibration of PEATCLSM specific yield parameters against a rise curve, the template file might look like this:

```
ptf @
specific_yield:
  type: peatclsm
  sd: @sd                        @
  theta_s: @theta_s                   @
  b: @b                          @
  psi_s: @psi_s                    @
transmissivity:
  type: peatclsm
  Ksmacz0: 7.3  # m/s
  alpha: 3  # dimensionless
  zeta_max_cm: 5.0
```

and an initial parameter vector file might look like this:

```
double point
          sd     0.162                       1.000000         0.000000
     theta_s     0.88                        1.000000         0.000000
           b     7.4                         1.000000         0.000000
       psi_s     -0.024                      1.000000         0.000000
```

In this example, the parameters `Ksmacz0` and `alpha` are not included in the parameter vector file because only the rise curve is being fitted. The parameters of the transmissivity are not free (they do not affect the rising curve fit), and therefore these values are fixed in the template file and omitted from the parameter vector file.

To verify the format of a template `rise_pars.yml.tpl` and initial parameters `rise_init.par`, use the PEST `tempchek` command:

```
tempchek rise_pars.yml.tpl rise_pars.yml rise_init.par
```

This command should exit without errors and produce a valid parameter file at `rise_pars.yml`.

The parameter file can then be verified by running your script. Your script might, for example, contain the command

```
spowtd simulate rise ekolongouma.sqlite3 rise_pars.yml -o rise_observations.yml
     --observations
```

which generates simulated dynamic storage values (without water levels or measured dynamic storage values) in `rise_observations.yml`; in PEST, simulated output values are referred to as "observations."

The resulting output file can then be checked against a PEST instruction file (`.ins`) that you create for extracting observation data, which might be called `rise_observations.ins`, using the PEST command `inschek`:

```
inschek rise_observations.ins rise_observations.yml
```

To then ensure that the correct initial parameters are used in the calibration, substitute these into the control file using `parrep`

```
parrep rise_init.par rise_calibration.in.pst rise_calibration.pst
```

To then calibrate specific yield parameters against the rise curve (alone) using the PEST control file `rise_calibration`, call:

```
pestchek rise_calibration &&
(pest rise_calibration.pst ;
 tempchek rise_pars.yml.tpl rise_opt.yml rise_calibration.par)
```

These commands check the PEST control file, perform the calibration, and then substitute the calibrated parameter values from `rise_calibration.par` into `rise_opt.yml`.

You can then examine the fit by plotting the rise curve with the calibrated parameters:

```
spowtd plot rise ekolongouma.sqlite3 --parameters rise_opt.yml
```

4.1 Generating PEST input files with Spowtd

As a convenience, Spowtd can generate input files for calibration with PEST, either against the rise curve (`spowtd pestfiles rise`) or against both rise and recession curves (`spowtd pestfiles curves`). The arguments to both subcommands are the same. Taking calibration against the rise curve as an example, a template file can be created with

```
spowtd pestfiles rise ekolongouma.sqlite3 parameters.yml tpl \\
  -o rise_parameters.yml.tpl
```

An instruction file can similarly be created with

```
spowtd pestfiles rise ekolongouma.sqlite3 parameters.yml ins \\
  -o ekolongouma_rise_observations.ins
```

and a control file can be created with

```
spowtd pestfiles rise ekolongouma.sqlite3 parameters.yml pst \\
  -o ekolongouma_rise_calibration.in.pst
```

The template and instruction files can be used as-is. The generated PEST control file will require substitution of valid starting parameters and bounds, substitution of paths to input files and the invocation for simualtion, adjustment of PEST control parameters, etc.

References

[1] Alexander R. Cobb, Alison May Hoyt, Laure Gandois, Jangarun Eri, René Dommain, Kamariah Abu Salim, Fuu Ming Kai, Nur Salihah Haji Su'ut, and Charles F. Harvey. How temporal patterns in rainfall determine the geomorphology and carbon fluxes of tropical peatlands. *Proceedings of the National Academy of Sciences of the United States of America*, 114: E5187–E5196, 2017. doi: 10.1073/pnas.1701090114.

[2] Alexander R. Cobb and Charles F. Harvey. Scalar simulation and parameterization of water table dynamics in tropical peatlands. *Water Resources Research*, 55(11):9351–9377, 2019. doi: 10.1029/2019wr025411.

[3] John Doherty. *PEST: Model-Independent Parameter Estimation User Manual*. Watermark Numerical Computing, 5th edition, 2010.

[4] C. C. Paige. Computer solution and perturbation analysis of generalized linear least squares problems. *Mathematics of Computation*, 33(145):171–183, 1979. doi: 10.1090/s0025-5718-1979-0514817-3.

A Matching of storms and water table rise

For construction of rise curves, Spowtd matches intervals of rapidly increasing water level ("rises") to intervals of heavy rain ("storms") in such a way that each storm is matched to no more than one rise and each rise is matched to no more than one storm. This matching is performed in two steps. First, all storms and rises that overlap in time are matched. This first step may result in matching from a single storm to multiple rises and vice versa. This step is followed by an arbitration step based on a variant of the Gale-Shapley deferred acceptance algorithm for the stable matching problem: it finds a set of matches between storms and rises that is stable in the sense that, by switching a pair of matches between storms and rises, one cannot improve the agreement in duration and start time for both matches.

The arbitration step favors agreement in duration over agreement in start time in matches by using a property of the Gale-Shapley algorithm: it is guaranteed to yield the stable matching that is most favorable for the proposing parties, and least favorable for the parties accepting or rejecting proposals. In matching between storms and rises, each storm tries to match with the rise with the closest duration; the rise is then able to reject that first match if another storm with a closer start time proposes a match. However, a storm that is already matched with a rise with a more similar duration will never propose to the rise with a closer start time. Thus, arbitration results in the stable matching that results in the best agreement in duration between storms and rises.

Note that matching may still result in bad mismatches in storm and rise duration depending on the thresholds set for identifying intense rain (for storms) and rapid increase in water level (for rises). If differences between storm and rise duration are larger than you think they should be, try adjusting one of these thresholds to get better agreement.

B Event weighting in rise analysis

Assembly of recession curves benefits from the fact that both water level and time usually can be measured fairly accurately. In contrast, rise events may be affected by large errors in measurement of the recharge depth, either because of the difficulties in measuring the precipitation that passes through the canopy and litter to reach the water table, or because of spatial separation or scale differences between where precipitation is measured and where the water table response is evaluated.

In some cases the error in recharge depth measurement may be large enough to cause problems in assembling a reasonable rise curve. Because each rise event is represented as a straight line segment between the water level and dynamic storage at the beginning and end of the event, an error in the recharge depth is manifested as an error in slope of this segment. In the case of a range of water levels where specific yield is approximately uniform so that the true slope of the rise curve is approximately constant, this error in slope causes an error in observed cumulative recharge that increases in proportion to distance from the center of the rise event line segment (Fig. 1). A single event spans multiple water levels, and thus affects multiple equations. In addition, previous work on precipitation measurement indicates larger error for higher precipitation, suggesting proportional errors.

Here we extend the unweighted approach to rise curve assembly in Cobb et al. [1] and Cobb and Harvey [2] to increase its applicability to cases with larger errors in recharge measurement. We first re-express the unweighted equations in the general form for a linear estimation problem given by Paige [4]. This form includes explicit terms for the additive uncorrelated errors that were minimized with that approach. We then show how we modified the approach to accommodate large, proportional errors in recharge observations.

Any linear estimation problem can be written in the following general form [4]:

$$\begin{aligned} \min_{\mathbf{s},\mathbf{v}} \quad & \mathbf{v}^{\mathsf{T}}\mathbf{v} \\ \text{subject to} \quad & \mathbf{y} = \mathbf{A}\mathbf{s} + \mathbf{B}\mathbf{v} \end{aligned} \tag{1}$$

where $\mathbf{y} = \mathbf{A}\mathbf{s}$ is the overdetermined system of linear equations to be solved for the unknowns $\mathbf{s}$, and $\mathbf{v}$ is a vector of errors with zero mean and unit variance, which is then appropriately scaled and distributed across equations by the matrix $\mathbf{B}$. When $\mathbf{B}$ is a square matrix with full rank, a typical approach to solving the linear estimation problem (1) is to assemble the error covariance matrix $\mathbf{\Omega} = \mathbf{B}\mathbf{B}^\mathsf{T}$, which will then be symmetric and positive definite and therefore invertible, and solve the generalized least-squares system

$$\mathbf{A}^\mathsf{T}\mathbf{\Omega}^{-1}\mathbf{A}\,\mathbf{s} = \mathbf{A}^\mathsf{T}\mathbf{\Omega}^{-1}\mathbf{y} \tag{2}$$

for the unknowns $\mathbf{s}$.

We now show how the the unweighted rise curve assembly problem [1, 2] corresponds to the general form of the linear estimation problem above (1). The unweighted rise curve assembly problem involves finding a storage offset s_j for each rise event j (Fig. 1). Each rise event j and water level i (Table 1) yield a single equation that can be written as

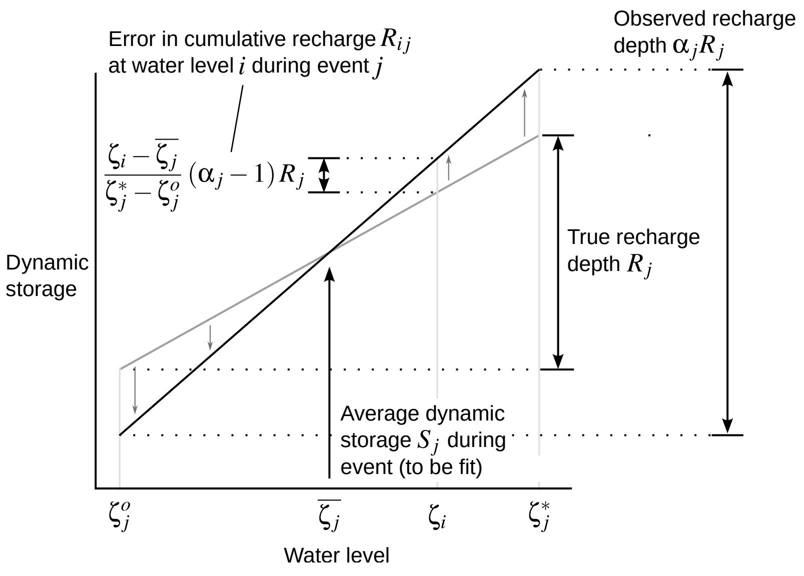
$$f_{ij} r_j + s_j - \frac{1}{|J_i|} \sum_{j' \in J_i} \left(r_{ij'} + s_{j'} \right) = e_k \tag{3}$$

where J_i is the set of rises crossing ζ_i, $|J_i|$ is the size (cardinality) of J_i and j' is a dummy index of summation. The weighting coefficient f_{ij} defined as

$$f_{ij} = \frac{\zeta_i - \overline{\zeta_j}}{\zeta_j^* - \zeta_j^o} \tag{4}$$

where ζ_j^o is the water level at the beginning and ζ_j^* the water level at the end of event j (Fig. 1).

In (3), the first two terms on the left represent the dynamic storage for event j when it crosses water level i, while the summation term in the round brackets computes the unweighted mean dynamic storage at water level i over all events. The error e_k in the kth equation represents the deviation of the dynamic storage for the jth event, when it crosses water level i, from the mean dynamic storage across all events crossing that water level. These deviations can arise due to complexities that are



Error in cumulative recharge R_{ij} at water level i during event j
Observed recharge depth $\alpha_j R_j$
$\frac{\zeta_i - \overline{\zeta_j}}{\zeta_j^* - \zeta_j^o}(\alpha_j - 1)R_j$
True recharge depth R_j
Dynamic storage
Average dynamic storage S_j during event (to be fit)
ζ_j^o
$\overline{\zeta_j}$
ζ_i
ζ_j^*
Water level

Figure 1: Model for error in recharge depth observations.

ignored in the simple scalar model underlying rise curve assembly, and from the approximation of the rise curve with straight line segments. For simplicity, in the unweighted implementation these factors were assumed to lead to uncorrelated errors with a similar typical magnitude across all equations.

Note that the mean dynamic storage at i (the term in the round brackets in (3)) is not known a priori because it itself depends on the offsets $\mathbf{s}$ for which we solve. However, the mean is completely specified by any particular choice of offsets $\mathbf{s}$; in fact it is a linear function of them. Therefore, unweighted rise curve assembly [1, 2] was able to incorporate the calculation of the mean into the overdetermined system of linear equations that is then solved by least squares.

The mean dynamic storage s_J for rise event J is set to 0 and excluded from the unknowns to make the problem nonsingular; after solving for the other dynamic storage values, a constant is subtracted from all of these to define the dynamic storage as zero when the water level is at the peat surface ($\zeta = 0$), as described in Cobb and Harvey [2].

The entire system of K unweighted equations can be written

$$\mathbf{FDr} + \mathbf{DTs} - \mathbf{M}_u(\mathbf{FDr} + \mathbf{DTs}) = \mathbf{e} \tag{5}$$

where, in this unweighted case, the averaging operator $\mathbf{M}_u$ is a simple arithmetic mean

$$\mathbf{M}_u = \begin{bmatrix} \frac{1}{|J_{i_1}|}[j_1 \in J_{i_1}] & \cdots & \frac{1}{|J_{i_1}|}[j_K \in J_{i_1}] \\ \vdots & \ddots & \vdots \\ \frac{1}{|J_{i_K}|}[j_1 \in J_{i_K}] & \cdots & \frac{1}{|J_{i_K}|}[j_K \in J_{i_K}] \end{bmatrix}, \tag{6}$$

which computes a mean across all events that cross each water level, and then distributes each mean to all of the K equations for that water level. Here we use the Iverson bracket $[P]$ to denote the indicator function that takes the value 1 if the statement P within the brackets is true and 0 otherwise, so that only the relevant events are selected in each row. The $K \times J$ equation-event incidence matrix $\mathbf{D}$ records the event j associated with each equation j and has exactly one nonzero value (equal to 1) in each row

$$\mathbf{D} = \begin{bmatrix} [j_1 = 1] & \cdots & [j_1 = J] \\ \vdots & \ddots & \vdots \\ [j_K = 1] & \cdots & [j_K = J] \end{bmatrix}. \tag{7}$$

Operating on a J-vector of event-specific values, $\mathbf{D}$ distributes these to the corresponding equations in a K-vector. The truncation matrix $\mathbf{T}$

$$\mathbf{T} = \begin{bmatrix} \mathbf{I}_{J-1} \\ \mathbf{0}_{1,J-1} \end{bmatrix} \tag{8}$$

enforces the boundary condition $s_J = 0$ by transforming the $J-1$-vector of unknowns s into a J-vector with its last element set to 0 ($\mathbf{T}\,\mathbf{s}$ is $\mathbf{s}$ with a zero concatenated to its end).

Rearranging (5) to put the unknown vector with errors $\mathbf{e}$ on the right-hand-side, we have

$$(\mathbf{I} - \mathbf{M}_u)\,\mathbf{FD}\,\mathbf{r} = -\,(\mathbf{I} - \mathbf{M}_u)\,\mathbf{DT}\,\mathbf{s} + \mathbf{e} \tag{9}$$

If we then define $\mathbf{y}_u = (\mathbf{I} - \mathbf{M}_u)\,\mathbf{F}\mathbf{D}\,\mathbf{r}$, $\mathbf{A}_u = -\,(\mathbf{I} - \mathbf{M}_u)\,\mathbf{D}\mathbf{T}$, and $\mathbf{B}_u = \sigma_e \mathbf{I}$ where σ_e is the positive square root of the (uniform) variance of the errors $\mathbf{e}$, so that $\mathbf{e} = \mathbf{B}_u \mathbf{v}$, we have expressed the unweighted problem in the standard form (1). Because the covariance $\boldsymbol{\Omega}_u = \mathbf{B}_u \mathbf{B}_u^{\top} = \sigma_e^2 \mathbf{I}$ in this unweighted case is just a positive multiple of the identity matrix, so too is its inverse $\boldsymbol{\Omega}_u^{-1}$, and it cancels from (2) yielding the ordinary least-squares problem

$$\mathbf{A}_u^{\top} \mathbf{A}_u \mathbf{s} = \mathbf{A}_u^{\top} \mathbf{y}_u. \tag{10}$$

Solving this ordinary least-squares problem (10) for $\mathbf{s}$ is equivalent to what was done in Cobb et al. [1] and Cobb and Harvey [2]; what we have outlined, then, is the same procedure but with more general notation.

We now extend this analysis to the weighted case with large, proportional errors in recharge observations. Suppose recharge depth observations $\tilde{\mathbf{r}}$ are affected by multiplicative independent errors

Symbol	*Dimensions*	*Definition*
$\mathbf{e}$	$K \times 1$	Deviation of storage at water level i for event j from mean across all events at that water level
$\mathbf{f}$	$K \times 1$	Weighting coefficient for error in cumulative recharge for each water level i and event j
i		Index over discrete water levels ζ_i
j		Index over rise events
J_i		Set of rise events crossing water level i
k		Index over equations; enumerates (i, j) pairs identifying water levels crossed by recharge events
$\tilde{\mathbf{r}}$	$J \times 1$	Observed recharge for each event
$\mathbf{r}$	$J \times 1$	True recharge depth for each event
$\mathbf{s}$	$(J-1) \times 1$	Dynamic storage offset for each event
$\mathbf{D}$	$K \times J$	Equation-event incidence matrix
$\mathbf{F}$	$K \times K$	diag($\mathbf{f}$)
I		Total number of water levels
$\mathbf{I}$	$K \times K$	Identity matrix
J		Total number of events
K		Total number of equations $\sum_i \lvert J_i \rvert$
$\mathbf{M}$	$K \times K$	Weighted averaging operator
S_y		Specific yield
$\mathbf{T}$	$J \times (J-1)$	Truncation matrix to set boundary condition $s_J = 0$
$\boldsymbol{\alpha}$	$J \times 1$	Multiplicative error in each recharge depth observation
σ_α^2		Variance of components of $\boldsymbol{\alpha}$
σ_e^2		Variance of components of $\mathbf{e}$
ζ		Water level
$\boldsymbol{\Omega}$	$K \times K$	Error covariance matrix

Table 1: Notation for assembly of rise curves

$\boldsymbol{\alpha}$, each identically distributed with mean 1 and variance σ_α^2. In this case, for an event j, instead of the true recharge r_j we measure an observed recharge $\tilde{r}_j = \alpha_j r_j$, so that the true recharge $\mathbf{r}$ is related to observed recharge $\tilde{\mathbf{r}}$ by

$$\mathbf{r} = \tilde{\mathbf{r}} - \operatorname{diag}(\mathbf{r})\,(\boldsymbol{\alpha} - \mathbf{1})\,. \tag{11}$$

In addition, because the variance of errors in recharge is no longer the same for each event j crossing a water level i, we will replace the unweighted averaging operator $\mathbf{M}_u$ by the inverse-variance-weighted-mean operator $\mathbf{M}$. The inverse-variance-weighted mean dynamic storage at water level i is computed as

$$\frac{\sum_{j\in J_i}(f_{ij}\tilde{r}_j + s_j)\sigma_{ij}^{-2}}{\sum_{j\in J_i}\sigma_{ij}^{-2}} \tag{12}$$

where $\sigma_{ij}^2 = \operatorname{var}(f_{ij}\tilde{r}_j + s_j)$. Because errors in observed recharge $\tilde{r}_j$ and storage s_j are uncorrelated,

$$\sigma_{ij}^2 = \operatorname{var}(\alpha_j f_{ij}\tilde{r}_j) + \operatorname{var}(s_j) = f_{ij}^2 r_j^2 \sigma_\alpha^2 + \sigma_e^2. \tag{13}$$

By (12), the weighted mean operator is given by

$$\mathbf{M} = \begin{bmatrix} \frac{\sigma_{i_1 j_1}^{-2}}{\sum_{j\in J_{i_1}}\sigma_{i_1 j}^{-2}}[j_1 \in J_{i_1}] & \cdots & \frac{\sigma_{i_1 j_K}^{-2}}{\sum_{j\in J_{i_1}}\sigma_{i_1 j}^{-2}}[j_K \in J_{i_1}] \\ \vdots & \ddots & \vdots \\ \frac{\sigma_{i_K j_1}^{-2}}{\sum_{j\in J_{i_K}}\sigma_{i_K j}^{-2}}[j_1 \in J_{i_K}] & \cdots & \frac{\sigma_{i_K j_K}^{-2}}{\sum_{j\in J_{i_K}}\sigma_{i_K j}^{-2}}[j_K \in J_{i_K}] \end{bmatrix} \tag{14}$$

Substituting the weighted mean operator $\mathbf{M}$ (14) and the expression for the recharge with observation error $\tilde{\mathbf{r}}$ (11) into the unweighted system of equations (9) yields

$$(\mathbf{I} - \mathbf{M})\,\mathbf{F}\mathbf{D}\,(\tilde{\mathbf{r}} - \operatorname{diag}(\mathbf{r})\,(\boldsymbol{\alpha} - \mathbf{1})) = -\,(\mathbf{I} - \mathbf{M})\,\mathbf{D}\mathbf{T}\,\mathbf{s} + \mathbf{e} \tag{15}$$

Because of the multiplicative errors $\boldsymbol{\alpha}$ in recharge, we now have an additional error term on the left-hand-side. To express our problem in the general form for a linear estimation problem (1), we will absorb the recharge errors into the error term $\mathbf{B}\mathbf{v}$.

Accordingly, we define a submatrix

$$\mathbf{B}_2 = \sigma_\alpha\,(\mathbf{I} - \mathbf{M})\,\mathbf{F}\mathbf{D}\operatorname{diag}(\mathbf{r}) \tag{16}$$

and a subvector $\mathbf{v}_2$ for the recharge errors, normalized to zero mean and unit variance

$$\mathbf{v}_2 = \frac{\boldsymbol{\alpha} - \mathbf{1}}{\sigma_\alpha} \tag{17}$$

so that we can write

$$(\mathbf{I} - \mathbf{M})\,\mathbf{F}\mathbf{D}\tilde{\mathbf{r}} = -\,(\mathbf{I} - \mathbf{M})\,\mathbf{D}\mathbf{T}\,\mathbf{s} + \mathbf{e} + \mathbf{B}_2\mathbf{v}_2 \tag{18}$$

Then, defining a second submatrix $\mathbf{B}_1 = \sigma_e\mathbf{I}$ and subvector $\mathbf{v}_1 = \mathbf{e}/\sigma_e$ to deal with the uncorrelated errors, we combine these in a block matrix $\mathbf{B} = [\mathbf{B}_1\mathbf{B}_2]$ and a block vector

$$\mathbf{v} = \begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \end{bmatrix}. \tag{19}$$

Finally, after defining $\mathbf{y} = (\mathbf{I} - \mathbf{M})\,\mathbf{F}\mathbf{D}\tilde{\mathbf{r}}$ and $\mathbf{A} = -\,(\mathbf{I} - \mathbf{M})\,\mathbf{D}\mathbf{T}$ and substituting into (18), we arrive back at the general form for a linear estimation problem (1).

The resulting weighted linear estimation problem can again be solved by the typical approach to generalized least squares (2) because its error covariance matrix $\boldsymbol{\Omega} = \mathbf{B}\mathbf{B}^{\mathsf{T}}$ is invertible. Expanding, $\mathbf{B}\mathbf{B}^{\mathsf{T}} = \mathbf{B}_1\mathbf{B}_1^{\mathsf{T}} + \mathbf{B}_2\mathbf{B}_2^{\mathsf{T}} = \sigma_e^2\mathbf{I} + \mathbf{B}_2\mathbf{B}_2^{\mathsf{T}}$. The addition of a positive multiple of the identity matrix $\sigma_e^2\mathbf{I}$ to the symmetric positive semidefinite matrix $\mathbf{B}_2\mathbf{B}_2^{\mathsf{T}}$ results in a symmetric positive definite, and therefore invertible, covariance matrix $\boldsymbol{\Omega}$.

So far we have ignored the fact that assembling the covariance matrix factor $\mathbf{B}_2$ (16) requires the true recharge $\mathbf{r}$, which we do not know. Nonetheless, we initially have as an estimate the imperfect observations $\tilde{\mathbf{r}}$, and by solving the linear estimation problem (1) we arrive at an estimate of the recharge errors $\boldsymbol{\alpha}$ (17), which we can use to improve our recharge estimates. In practice, we found that repeatedly solving the weighted estimation problem and updating the recharge $\tilde{\mathbf{r}}$ resulted in convergence within a few iterations to a self-consistent solution for storage offsets $\mathbf{s}$ and errors $\mathbf{v}$.

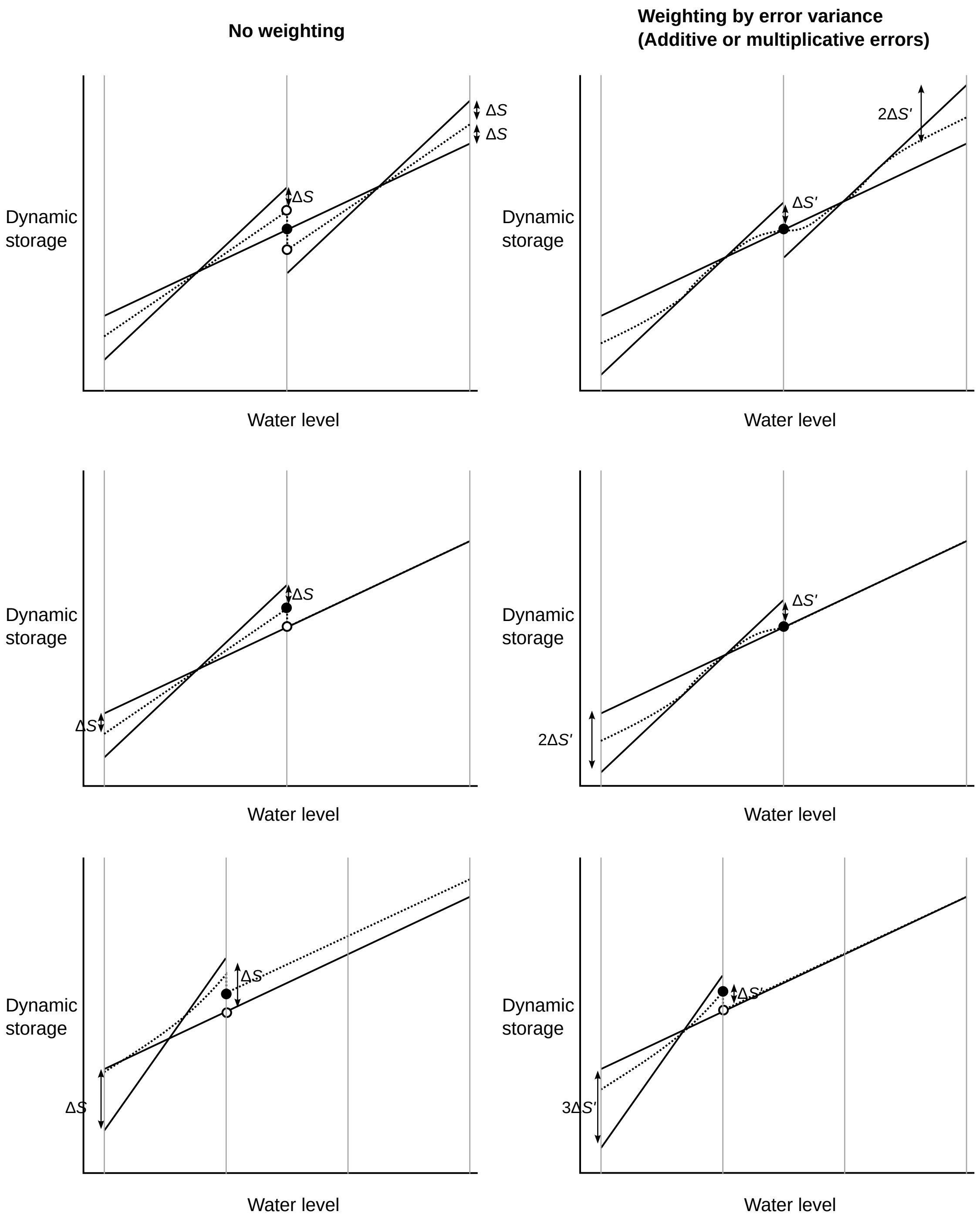


Figure 2: Effect of recharge error weighting in some simple test cases.

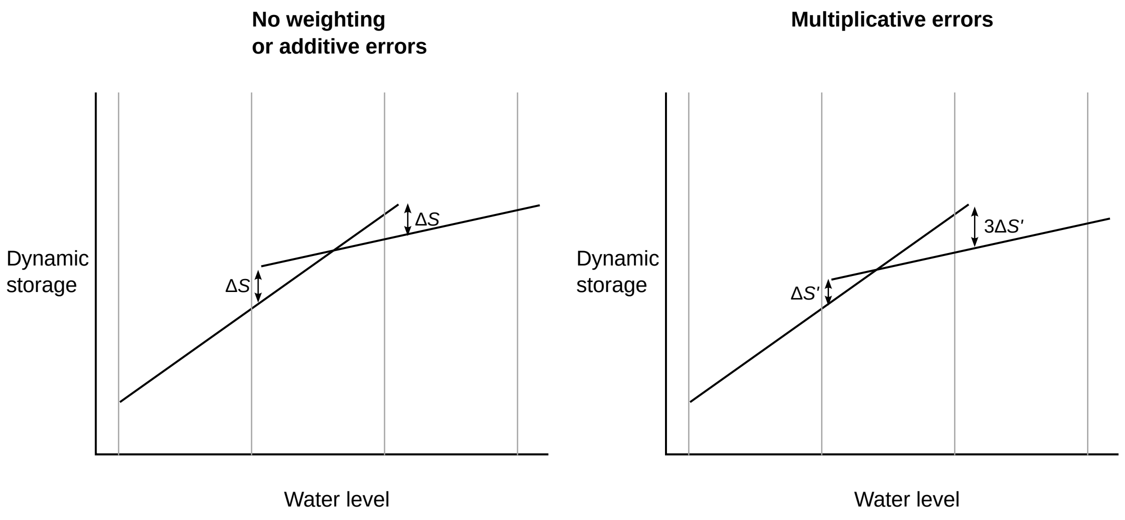


Figure 3: Effect of multiplicative vs. additive error in some simple test cases.