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.

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. To reduce the impact of a single event that spans a large number of water levels and happens to have a large error in recharge depth on the rise curve, Spowtd will provide a scheme to ensure that events are effectively weighted equally, and which explicitly considers the effects of a bad estimate of recharge depth on observations of the rise event at multiple water levels.

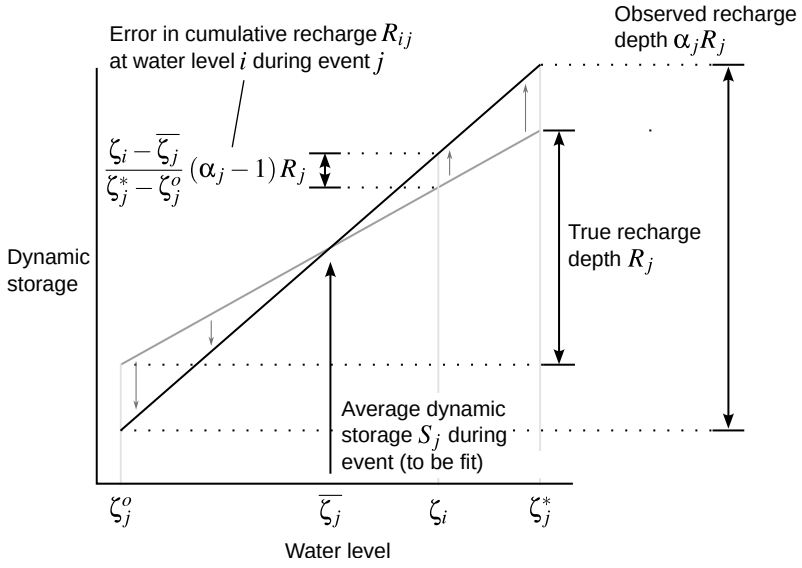

Figure 1: Model for error in recharge depth observations.

As discussed in Cobb et al. [1] and Cobb and Harvey [2], the assembly of the rise curve involves finding the mean dynamic storage S_j during each rise event j (Fig. 1), and results in the following equation:

$$R_{ij} + S_j = \frac{1}{|J_i|}\left(\sum_{j' \in J_i} R_{ij'} + S_{j'}\right) \tag{1}$$

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. Note that j indexes rise events, while i indexes water levels ζ_i (Table 1).

Rearranging, for each of the $|J_i|$ rise events j that cross a discrete water level i, we have an equation

$$\left(\frac{1}{|J_i|}\sum_{j' \in J_i} S'_j\right) - S_j = R_{ij} - \frac{1}{|J_i|}\sum_{j' \in J_i} R_{ij'} \tag{2}$$

which can be written in matrix form with typical row

$$\left[\begin{array}{ccccccc} \frac{1}{J_i} & \frac{1}{J_i} & \cdots & \frac{1}{J_i} - 1 & \cdots & \frac{1}{J_i} \end{array}\right] \left[\begin{array}{c} S_1 \\ S_2 \\ \vdots \\ S_j \\ \vdots \\ S_{J-1} \end{array}\right] = \left[R_{ij} - \frac{1}{|J_i|}\sum_{j' \in J_i} R_{ij'}\right]. \tag{3}$$

This represents $|J_i|$ equations. The quantity on the right is a scalar; there is one such equation—with the same row and column vectors on the left, but a different row vector on the right—for each rise event j of the $|J_i|$ rise events crossing water level i. 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 set of equations (3) for the average dynamic storage S_j during each event can be written as a matrix equation

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{4}$$

where $\mathbf{A}$ is a dense matrix with $K = \sum_i |J_i|$ rows and $J - 1$ columns, $\mathbf{x}$ is the vector of unknown dynamic storage values S and $\mathbf{b}$ contains the right-hand-side values from equation 3.

Ideally, the true recharge depth R_j for a storm j would be known, and this system of equations $\mathbf{A}\mathbf{x} = \mathbf{b}$ would have a solution after elimination of redundant equations. In practice, errors in recharge depth affect all equations and introduce covariance among the components of the error vector $\boldsymbol{\varepsilon}$ associated with the system of equations $\mathbf{A}\mathbf{x} = \mathbf{b}$.

Suppose recharge depth observations $\widehat{\mathbf{R}}$ are affected by multiplicative independent errors $\boldsymbol{\alpha}$, each identically distributed with mean 1 and variance var(α). For a rise event j, our observations at every water level i crossed by the rise event are contaminated by that error α_j:

$$\widehat{R}_{ij} = f_{ij}\alpha_j R_{ij} \tag{5}$$

with f_{ij} defined as

$$f_{ij} = \frac{\zeta_i - \overline{\zeta_j}}{\zeta_j^* - \zeta_j^o} \tag{6}$$

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).

The error e_{ij} in each observed recharge depth is

$$e_{ij} = \widehat{R}_{ij} - R_{ij} = (\alpha_j - 1) R_j f_{ij} \tag{7}$$

with zero mean

$$\langle e_{ij} \rangle = [\langle \alpha_j \rangle - 1] R_j f_{ij} = 0. \tag{8}$$

The error ε_k in the k^{th} equation is a linear combination of the errors e_{ij} in the recharge depth

$$\varepsilon_k = e_{ij} - \beta_i \tag{9}$$

where i and j are the water level i and rise j associated with the kth equation, and β_i is the average error across storm events crossing water level i

$$\beta_i = \frac{1}{|J_i|} \sum_{j' \in J_i} e_{ij'} \tag{10}$$

Symbol	*Definition*
$\langle \cdot \rangle$	Expected value
e_{ij}	Error in cumulative recharge at water level i during event j
f_{ij}	Weighting coefficient for error in cumulative recharge at water level i for 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
R_{ij}	True cumulative recharge at water level i during event j
$\widehat{R}_{ij}$	Observed cumulative recharge at water level i during event j
R_j	True total recharge depth during rise event j
S	Dynamic storage
S_j	Average dynamic storage during event j
S_y	Specific yield
α_j	Error in recharge depth during rise event j
ε_k	Error in equation k
ζ	Water level

Table 1: Notation for assembly of rise curves

We hypothesize that we can obtain a truer fit by deriving an estimate of the variance-covariance matrix $\mathbf{\Omega}$ of the error vector, and then solving the generalized least-squares problem

$$\mathbf{A}^T\mathbf{\Omega}^{-1}\mathbf{A}\mathbf{x} = \mathbf{A}^T\mathbf{\Omega}^{-1}\mathbf{b}. \tag{11}$$

We will use our model for the recharge depth error α to derive an estimate for the covariance between elements of the error vector $\boldsymbol{\varepsilon}$ (variance of error vector $\boldsymbol{\varepsilon}$).

The covariance $\mathrm{cov}(\varepsilon_{k_1}, \varepsilon_{k_2})$ between two components of the error vector is the expected value of the product of their deviations. Because the errors $\mathbf{e}$ in recharge have zero mean (7), inspection of (9) shows that the elements of $\boldsymbol{\varepsilon}$ also have zero mean, and therefore an element of the variance-covariance matrix of the error vector $\boldsymbol{\varepsilon}$ is given by

$$\mathrm{cov}\,(\varepsilon_{k_1}, \varepsilon_{k_2}) = \langle \varepsilon_{k_1} \varepsilon_{k_2} \rangle \tag{12}$$

which, after substitution from (9), expands to

$$\mathrm{cov}\,(\varepsilon_{k_1}, \varepsilon_{k_2}) = \left\langle e_{i_1 j_1} e_{i_2 j_2} \right\rangle - \langle e_{i_1 j_1} \beta_{i_2} \rangle - \langle e_{i_2 j_2} \beta_{i_1} \rangle + \langle \beta_{i_1} \beta_{i_2} \rangle \tag{13}$$

We first derive expressions for the terms in this equation (13). It is convenient to divide these terms by the variance $\mathrm{var}(\alpha)$ in α. The expected product of pairs of errors e_{ij} in observed recharge depth is

$$\left\langle e_{i_1,j_1}\, e_{i_2,j_2} \right\rangle = [\langle \alpha_{j_1} \alpha_{j_2} \rangle - 1] R_{j_1} R_{j_2} f_{i_1 j_1} f_{i_2 j_2}. \tag{14}$$

Because the components of the error $\boldsymbol{\alpha}$ are independent, the expected value of their product $\langle \alpha_{j_1} \alpha_{j_2} \rangle$ is $1 + \mathrm{var}(\alpha)$ if $j_1 = j_2$ and 1 otherwise. Therefore, the expected product of recharge depth observation errors, normalized by $\mathrm{var}(\alpha)$, is

$$\frac{\left\langle e_{i_1,j_1}\, e_{i_2,j_2} \right\rangle}{\mathrm{var}(\alpha)} = R_{j_1}^2 f_{i_1 j_1} f_{i_2 j_1} [j_1 = j_2]. \tag{15}$$

where 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.

The terms $\langle e_{i_1 j_1} \beta_{i_2} \rangle$, again normalized by the variance $\mathrm{var}(\alpha)$ of α, after substitution and rearrangement, are given by

$$\frac{\langle e_{i_1 j_1} \beta_{i_2} \rangle}{\mathrm{var}(\alpha)} = \frac{1}{|J_{i_2}|} \sum_{j' \in J_{i_2}} f_{i_2 j'} \frac{\langle e_{i_1 j_1} e_{i_2 j'} \rangle}{\mathrm{var}(\alpha)} \tag{16}$$

which, because any j appears at most once in J_{i_2}, can be expressed, again using the Iverson bracket, as

$$\frac{\langle e_{i_1 j_1} \beta_{i_2} \rangle}{\mathrm{var}(\alpha)} = \frac{R_{j_1}^2 f_{i_1 j_1} f_{i_2 j_1}}{|J_{i_2}|} [j_1 \in J_{i_2}] \tag{17}$$

Finally, the terms $\langle \beta_{i_1} \beta_{i_2} \rangle$ are given by substitution from (10) as

$$\langle \beta_{i_1} \beta_{i_2} \rangle = \left\langle \left(\frac{1}{|J_{i_1}|} \sum_{j' \in J_{i_1}} e_{i_1 j'} \right) \left(\frac{1}{|J_{i_2}|} \sum_{j' \in J_{i_2}} e_{i_2 j'} \right) \right\rangle \tag{18}$$

which, after normalizing by the variance of α, expands to

$$\frac{\langle \beta_{i_1} \beta_{i_2} \rangle}{\mathrm{var}(\alpha)} = \frac{1}{|J_{i_1}||J_{i_2}|} \sum_{j' \in J_{i_1}, j'' \in J_{i_2}} \frac{\langle e_{i_1 j'} e_{i_2 j''} \rangle}{\mathrm{var}(\alpha)} \tag{19}$$

and then, using (15), simplifies to

$$\frac{\langle \beta_{i_1} \beta_{i_2} \rangle}{\mathrm{var}(\alpha)} = \frac{1}{|J_{i_1}||J_{i_2}|} \sum_{j' \in J_{i_1} \cap J_{i_2}} R_{j'}^2 f_{i_1 j'} f_{i_2 j'} \tag{20}$$

Finally, substituting from the expressions (15), (17) and (20) for terms in (13), we have the elements $\mathrm{cov}(\varepsilon_{k_1}, \varepsilon_{k_2})$ of the variance-covariance matrix $\mathbf{\Omega}$ of the error vector ε, normalized by the recharge depth error variance $\mathrm{var}(\alpha)$, as

$$\begin{aligned} \frac{\mathrm{cov}(\varepsilon_{k_1}, \varepsilon_{k_2})}{\mathrm{var}(\alpha)} =& R_{j_1}^2 f_{i_1 j_1} f_{i_2 j_1} [j_1 = j_2] \\ &- \frac{R_{j_1}^2 f_{i_1 j_1} f_{i_2 j_1}}{|J_{i_2}|} [j_1 \in J_{i_2}] \\ &- \frac{R_{j_2}^2 f_{i_2 j_2} f_{i_1 j_2}}{|J_{i_1}|} [j_2 \in J_{i_1}] \\ &+ \frac{1}{|J_{i_1}||J_{i_2}|} \sum_{j' \in J_{i_1} \cap J_{i_2}} R_{j'}^2 f_{i_1 j'} f_{i_2 j'} \end{aligned} \tag{21}$$

We do not have the true recharge depths R_j, but only recharge depth observations $\widehat{R}_j$, so we approximate the covariance terms using these observations instead. We note in passing that the expected value of $\widehat{R}_j^2$ is $[1 + \mathrm{var}(\alpha)]R_j^2$, but because all terms are scaled by the same scalar $1 + \mathrm{var}(\alpha)$, neglecting this does not change the equations analytically.

For numerical reasons, to keep the norm of $\mathbf{\Omega}$ close to 1, we nondimensionalize the recharge observations $\widehat{R}_j$ by dividing them by their mean prior to further calculations (dividing $\mathbf{\Omega}$ by a scalar has no analytical effect because its inverse appears once on the left and the right of (11)).

Using (21), we populate the variance-covariance matrix $\mathbf{\Omega} = \mathrm{var}(\varepsilon)$, compute its inverse, and then solve the generalized least-squares problem (11) to obtain the average storage $S_j, j \in \{1, 2, \ldots, J-1\}$ to a constant from the entries in $\mathbf{x}$.