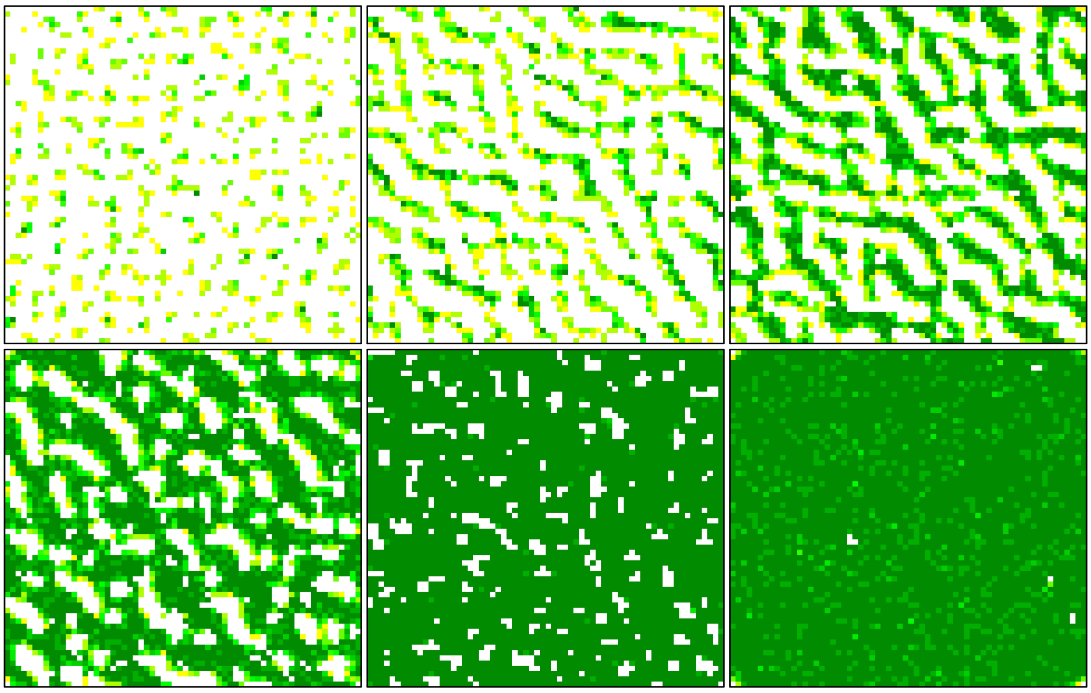
Ecohydraulical Feedback Simulation Documentation and Manual



Gavan McGrath
School of Earth and Environment, The University of Western Australia,
M087, 35 Stirling Highway, Crawley, Western Australia, 6009

Christoph Hinz
Tobias Nanu Frechen
Brandenburg University of Technology, Hydrology and Water Resources Management,
Konrad-Wachsmann-Allee 6, 03046 Cottbus, Germany

DRAFT of 30th April, 2013, 13:32

Contents

Draft status

- Suggestions for overall structuring welcomed!
- what means Dimensions: [:,:] ?
- subroutines and comments on the variables are from file `coupledModel.f90`
- `\subroutine{<name>}{<arguments>}` for displaying a subroutine with its arguments. <name> automatically gets an (emphasized) index entry and a label.
- `\inout{<input entries>}{<output entries>}` is for a list of input and output arguments. Missing values get replaced by "no input" or "no output".
- `\inoutentry[<option>]{<variableName>}{type}{dimensions}{description}` for an entry in the input output list. <option> can be empty or "emph" for an emphasized index entry or "none" for no index entry.
- `\begin{usessubs}` is an environment of used subroutines.
- `\usessubentry{<subroutineName>}` is for an entry in the `usessubs` environment. Entries automatically get an (not emphasized) index entry and a reference to the associated subroutine.
- I am using the keyword index to index all parameter and variable names. Good idea? Or shall we make a list with all parameter and variable names and their description and unit? Like:

Table 1: variables

variable	type	dimensions	nom	unit	description
`climParams`	[8-byte real]	[4]			climate parameters
`infiltParams`	[8-byte real]	[6]			vegetation and soil kernel parameters
`evapParams`	[8-byte real]	[8]			evaporation parameters
`discharge`	[integer]	[m,n]	Q	$[\mathrm{m}^3/\mathrm{s}]$	discharge
`alpha`	[8-byte real]	[m,n]	α		

My suggestion for collaboration: let's use the package "trackchanges". Nanu: Editing [could look like this] with the following commands:

```
\add[editor]{added text}

\remove[editor]{removed text}

\change[editor]{removed text}{added text}

\note[editor]{note text}

\annote[editor]{text to annotate}{note text}
```

1 Introduction

2 Theoretical principles

2.1 Simulation model

Runoff, soil moisture storage, transpiration and plant-bare soil transitions are numerically modeled on a lattice of square cells. The model simulates the following spatially distributed water balance:

$$\frac{\partial w_i}{\partial t} = P_i + R_i - Q_i - E_i - \sum_n T_n \tag{1}$$

2.2 Runoff generation

2.3 Surface water flow

2.4 Short range facilitation

2.5 Evaporation and transpiration

2.6 Vegetation change

2.7 Microtopography

3 Application

3.1 Installation

3.1.1 System requirements

The program is platform independent. For compilation you need the gfortran (GNU Fortran) compiler. Read how to install on www.gcc.gnu.org/fortran/. Other Fortran compilers might work, but are not tested.

Output is in `.csv` (comma seperated values) format. So you will be able to read it into any statistical Software like Microsoft Excel or LibreOffice. For this reason you might not need anything of the following.

To use the miscellaneous scripts provided in the folder `./evaluation/`, you need the statistical software environment "R" that can be obtained from www.r-project.org. See section 3.7.1 on how to use these scripts.

We also recommend the graphical user interface to R called "RStudio" that can be obtained from www.rstudio.com/ide/

There are some shell scripts to automate compilation, simulation run and execution of R-scripts. These can be used only on Unix-Systems. See section 3.5.1.

There are some scripts, that generate automated reports as `.pdf` or `.tex` on the data the simulation produces. These require a running LaTeX system. Have a look into section 3.7.2 on how to use these. You might not need these scripts, if you don't want to use LaTeX to write your papers. Graphics can also be exported from the R system.

3.1.2 Program files

In the main folder we have following folders, containing:

`./model/`	the model code;
`./evaluation/`	evaluation scripts for the output;
`./documentation/`	the documentation files;
`./example simulation run/`	example input Parameters and example output, calculated from this input;

Note, that in the following we will always describe file and folder names relative from the main folder (`/ecohydraulical-feedback/`). So we assume that in the console you first go into this folder with this command:

```
cd /path/to/ecohydraulical-feedback/
```

replacing the path with the correct path to the `/ecohydraulical-feedback/` folder.

3.1.3 Build from source

To use the gfortran compiler execute the following in terminal (also called Shell, Bash, command line, console):

```
gfortran ./model/ecohydModel.f90
```

On Unix systems the file `a.out` will be generated, on Windows the file is called `a.exe`.

If you want to give the executable a more reasonable name and put it in a specific folder, change the command to something similar to this:

```
gfortran -o /path/myexecutable.out ecohydModel.f90
```

3.2 Application prerequisites

3.3 Data preperation

3.4 Parameterization

All input parameters are set in a text file, that you choose as input at runtime. Have a look at the file `./example simulation run/exampleParameters.txt` to get an idea of which parameters can be set. Best way to feed in your parameters is to copy and modify this file. There also are some files with parameters that refer to the figures used by **?** in the same folder.

The input file must contain parameters in the form `name = value`. You can make comments after an `!` in the same line as the parameter or in an own line. Empty lines are also ignored.

The `title` parameter you set in the input file will be the prefix of your output files (cf. section 3.6). Note that existing output gets overwritten without warning if you run a parameter set with the same title.

If you wish to make a cascade of simulation runs you can do this with one input file. Just put another line defining another `title` parameter in the input file. Following you can define the parameters you want to be different to the first run. You don't have to repeat parameters that shall be the same as in the previous parameter set.

Maybe you want to check whether all data gets read in correctly. In this case you can set the parameter `run = F`. In this case you can execute the input parameter set and the program will try to read the input, but doesn't start simulation run. If something isn't correct in the input file, you will get an error message describing which parameter couldn't be read. Maybe you want to do this before you start a cascade of simulation runs.

3.5 Simulation run

After you have compiled an executable following section 3.1.3, you can execute the simulation.

Run simulation from command line with the following syntax:

```
./a.out <inputFile.txt> <outputFolder>
```

or on Windows:

```
./a.exe <inputFile.txt> <outputFolder>
```

Replace `<inputFile.txt>` with the absolute path to the file containing your input parameters. There is an example file that you could modify in `./example simulation run/exampleParameters.txt`. Replace `<outputFolder>` with the path to the folder where you want your output files. As with the input file only use absolute paths.

If you chose a different name for the compiled executable like described in section 3.1.3, you have to replace `./a.out` with your chosen file path and name. For example `/path/myexecutable.out`.

3.5.1 Batch run

The script `./evaluation/BatchRun.sh` can be used to automate Fortran compilation, execution of simulation, conversion to `.RData` and additional evaluation scripts in R. Feel free to modify the script to your needs. The batch script has to be run from within the `./evaluation/` folder, so first:

```
cd ./evaluation/
```

into the evaluation folder and then execute:

```
./BatchRun.sh <inputFile.txt> <outputFolder>
```

The executable, generated by this script is called `ecohydModel.out` respectable `ecohydModel.exe` on Windows and can be found in the `./model/` folder.

3.6 Output

The output is in `.csv` format. That means "comma separated values" and is basically a text file with numbers, separated by ";". You can read it into Microsoft Excel or similar programs.

There is one output file for every grid, that gets generated. You find them in the output folder you defined in the command line (cf. section 3.5). Files are formatted as followed:

`<title>_<grid-name>.csv`

Where `<title>` is the parameter set title you defined in the input file and `<grid-name>` the name of the grid (for example `vegetation`, `discharge` etc.). Additional there is a file

`<title>_SummaryResults.csv`

which contains some calculated summary values. Lastly there is

`<title>_inputParameter.txt`

which contains all the input Parameters that were used to generate the output files of the `<title>` simulation run.

3.6.1 Binary output

There is also a possibility to generate binary output of the grid data. For this you have to change the parameter `outputFormat = csv` to `outputFormat = binary` in the Parameter input File.

The binary output is formatted as followed:

1. first 4 bytes are an index integer describing how many bytes follow
2. the following binarys are the data of the first time step and are formated either in 4 or 8 byte blocks, depending on whether the data is 4-byte integer or 8-byte float (`real*8` in Fortran syntax). For integer values the preceding index number is for example $10000 \cdot 4 = 40000$ for a 100×100 grid. For float numbers this would be $10000 \cdot 8 = 80000$.
3. the index number gets repeated after every time step

The next timestep starts again with an index number, so its: `index; data; index; index; data; index; ...` There are no line breaks in the file.

For an example on how to read this data into R see the file `./evaluation/read-binary-files.r`.

3.7 Appraise and visualize results

3.7.1 in R

We recommend to evaluate the data in R. There are several scripts in the folder `./evaluation/`:

`readCSV.r`	reads the `.csv` output into R
`CSVtoRData.r`	reads `.csv` output and saves as `.RData` file; uses `readCSV.r`
`readAllCSV.r`	reads all `.csv` files in the provided folder and converts them to `.RData`; uses `CSVtoRData.r`
`readAllCSV.sh`	Shell script that executes `readAllCSV.r`
`BatchRun.sh`	Shell script that runs compilation, executes the simulation and converts to `.RData`; makes use of `readAllCSV.sh`
`read-binary-files.r`	reads binary output; not fully implemented yet, because there was no noticable speed enhancement
`coverRatio.r`	function to calculate a cover ratio diagram (developing cover ratio over time)

cover ratio The R-script `./evaluation/coverRatio.r` prints the vegetation cover ratio as:

$$\frac{cells_{filled}}{cells_{total}}$$

This is calculated for every time step and printed as a line plot, so that the development over time can be observed.

To avoid that edge effects at the system boundary produce inconsistent oscillations in the graph, the line 1 and column 1 get truncated, before the ratio is derived (cf. figure 1). So $cells_{filled}$ is defined as the filled cells in:

```
vegetation[[i]][2:m,2:n]
```

Where `[[i]]` refers to time step i and `[2:m,2:n]` to rows $2...m$ and colums $2...n$.

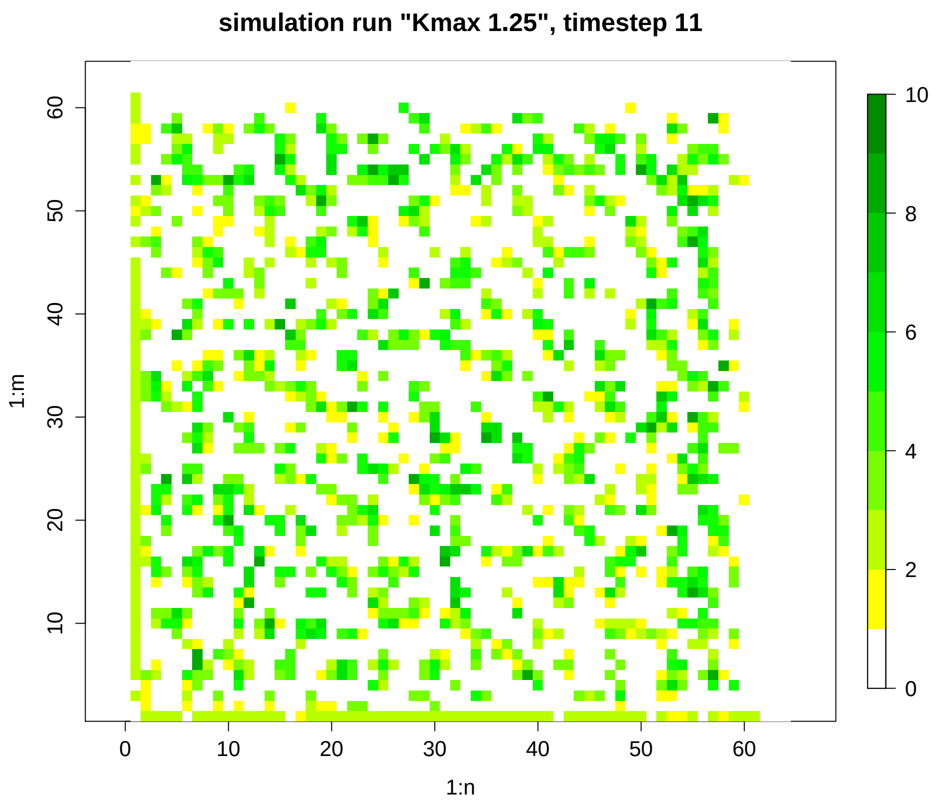
The total cells are:

$$cells_{total} = (n-1) \cdot (m-1)$$

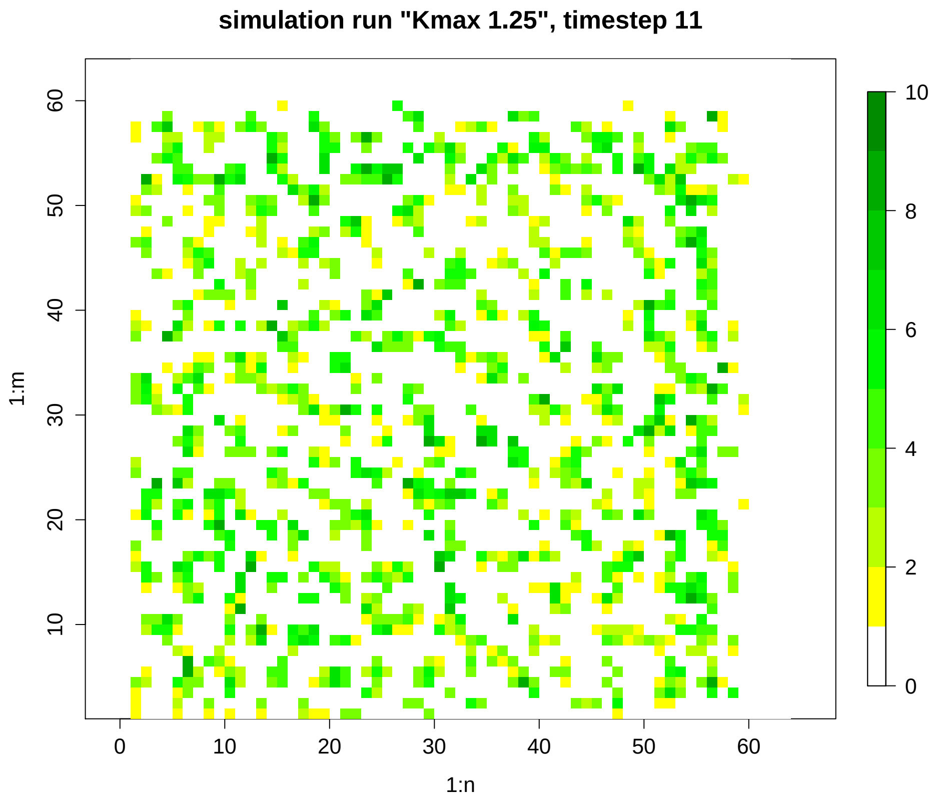
3.7.2 in R and LaTeX (knitr-method)

`http://yihui.name/knitr/`

3.7.3 in Excel



(a) Oscillating cells in row 1 and column 1



(b) Truncated first row and colmn

Figure 1: Edge effects on the vegetation grid

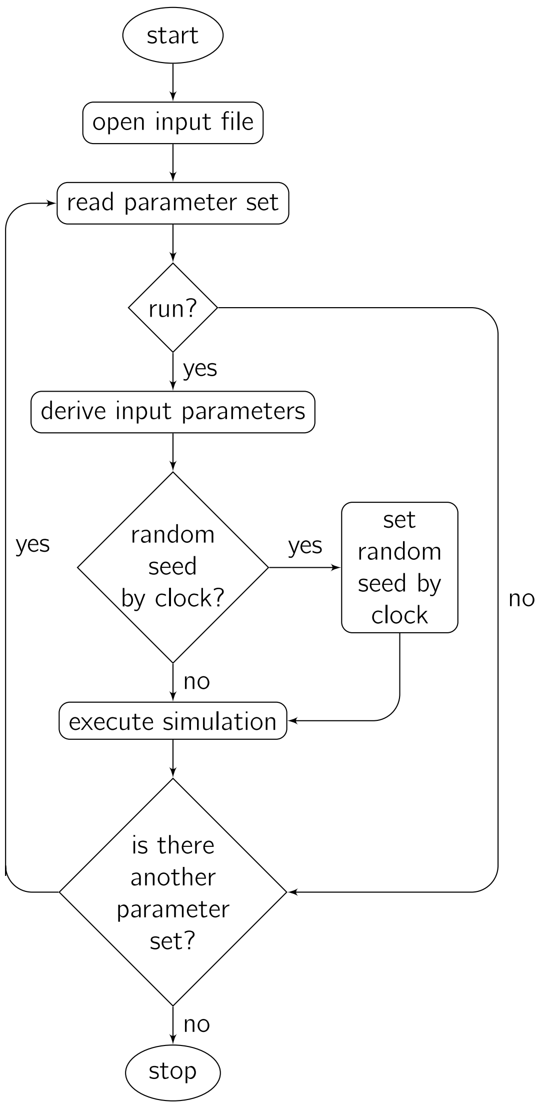


Figure 2: flow chart of the main program

4 Implementation

4.1 Overview

4.2 Implementation strategy

4.3 Subroutines

4.3.1 Simulation

```
SimCODE(m, n, mn, simflags, climParams, infiltParams, evapParams,
vegParams, erParams, resultsFID)
```

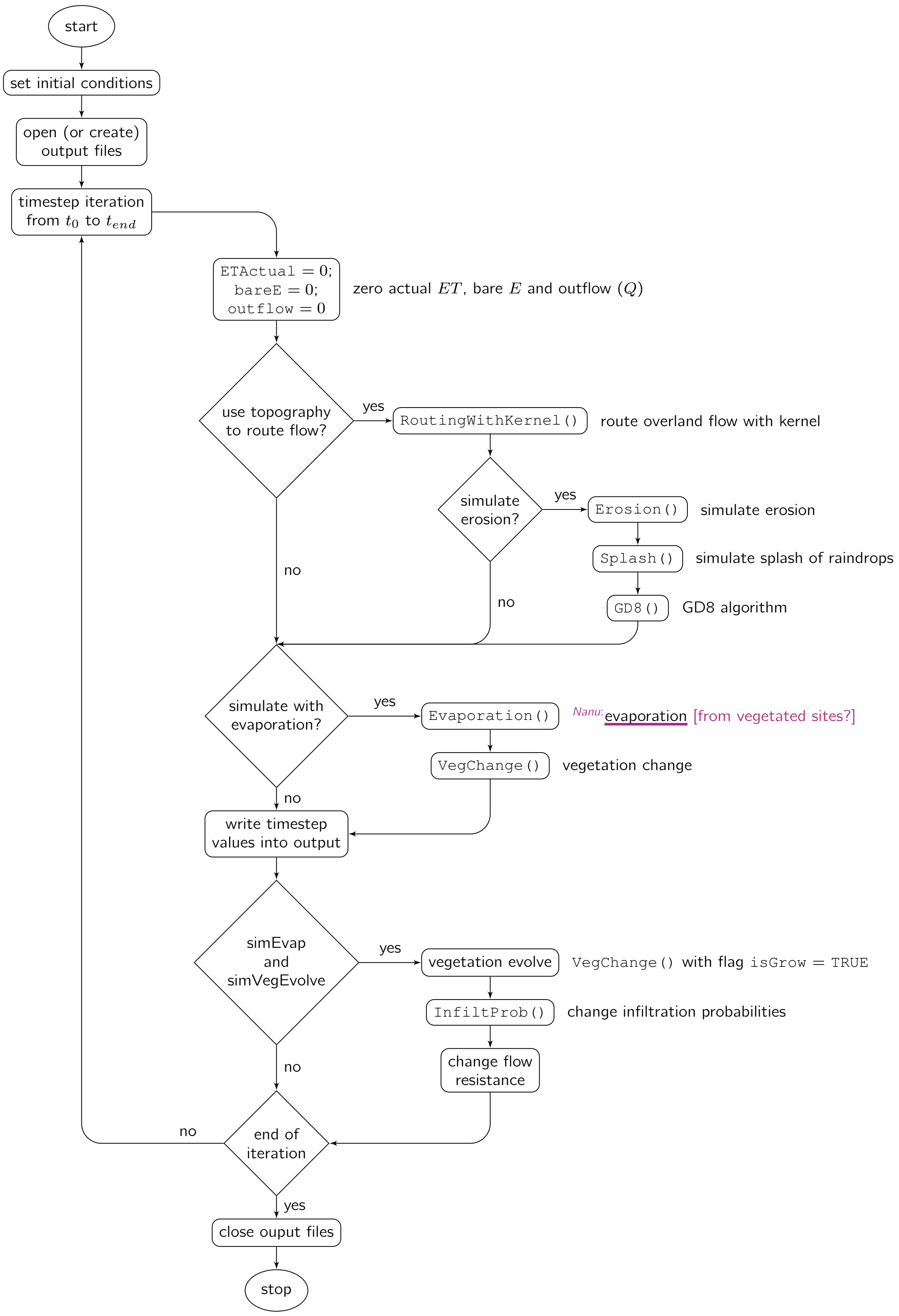

start
set initial conditions
open (or create) output files
timestep iteration from t_0 to t_{end}
ETActual = 0; bareE = 0; outflow = 0
zero actual ET, bare E and outflow (Q)
use topography to route flow?
yes
RoutingWithKernel()
route overland flow with kernel
simulate erosion?
yes
Erosion()
simulate erosion
Splash()
simulate splash of raindrops
GD8()
GD8 algorithm
no
no
simulate with evaporation?
yes
Evaporation()
Nanu: evaporation [from vegetated sites?]
VegChange()
vegetation change
no
write timestep values into output
simEvap and simVegEvolve
yes
vegetation evolve
VegChange() with flag isGrow = TRUE
InfiltProb()
change infiltration probabilities
change flow resistance
no
no
end of iteration
yes
close ouput files
stop

Figure 3: flow chart of subroutine SimCode (simulation execution)

input

m	[integer]		number of rows
n	[integer]		number of colums
mn	[integer]		= m·n
simflags	[integer]	[6]	[1] ... is for this, [2] ... is for that, Nanu: [should be explained!]
climParams	[8-byte real]	[4]	Nanu: climate parameters [how to input these?]
infiltParams	[8-byte real]	[6]	Nanu: ~~vegetation and~~ soil kernel parameters
evapParams	[8-byte real]	[8]	evaporation parameters
vegParams	[8-byte real]	[5]	vegetation Nanu: ~~and soil kernel~~ parameters
erParams	[8-byte real]	[4]	Nanu: vegetation and soil kernel parameters [what specific?]
resultsFID	[character]	[len=8]	results file id code

output

no output

Uses subroutines:

`GD8()`, cf. page 17
`InfiltProb()`, cf. page 13
`RoutingWithKernel()`, cf. page 13
`Erosion()`, cf. page 14
`Splash()`, cf. page 14
`Evaporation()`, cf. page 15
`VegChange()`, cf. page 19

4.3.2 Green-ampt infiltration

`GAInfilt``(m, n, dt, inflow, Ksat, wfs, cumInfilt, infex)`

Nanu: Green-ampt infiltration calculation. [Literature?]

input

m,n	[integer]		
dt	[8-byte real]		
inflow	[8-byte real]	[m,n]	surface runon + precipitation $(R + P)$
wfs	[8-byte real]	[m,n]	Nanu: wetting front suction * moisture deficit [formula?]
cumInfilt	[8-byte real]	[m,n]	Nanu: cumulative infiltration [nomenclature?]

output

infex	[8-byte real]	[m,n]	infiltration excess
cumInfilt	[8-byte real]	[m,n]	cumulative infiltration

4.3.3 Nanu: TwoDRandPos [cannot guess what this means]

`TwoDRandPos``(randOrder, m, n, mn)`

input

m,n	[integer]	
randOrder	[integer]	[mn,2]

output

randOrder	[integer]	[mn,2]

Uses subroutines:

`OneDRandList()`, cf. page 12

4.3.4 Kinematic wave primer

`KWPrimer``(m, n, topog, manningsN, mask, solOrder, flowdirns, alpha, deltax, solMax)`

Iterate over space to generate calculation a mask. The mask is a 9 x 1 array where 1s indicate that a neighbour (as defined by the position in mask(x,y)) discharges to the cell at x,y

input

`m,n`	[integer]		
`topog`	[integer]	[`mn,n`]	
`manningsN`	[integer]	[`mn,n`]	

output

`deltax`	[8-byte real]	[m,n]	
`alpha`	[8-byte real]	[m,n]	
`solMax`	[integer]		
`solOrder`	[integer]	[m,n]	
`flowdirns`	[integer]	[m,n]	flow directions
`mask`	[integer]	[m,n,9]	

Uses subroutines:

`GD8()`, cf. page 17
`KMWOrder()`, cf. page 12
`Neighbours()`, cf. page 16
`Lookupfdir()`, cf. page 12

4.3.5 Kinematic wave

`KinematicWave``(m, n, ndx, dt, iex, flowdirns, solOrder, solMax, mask, alpha, deltax, disOld, disNew)`

This subroutine calculates the kinematic wave equation for surface runoff in a network for a single time step. The flow network is given by the GD8 algorithm calculated on the topography. We assume no interaction between adjacent flow pathways i.e. water does not overflow in a direction not specified by the GD8 network. `iex` is the excess water from Nanu: precip - infiltration + GW discharge [display as formula?].

$$iex(m, n, 1) = qit$$

Nanu: [should this be displayed as formula or as code?]

$$iex(m, n, 2) = qitPlus$$

Nanu: `manningsN` the roughness coefficient [is this in the right place?]

Nanu: `init` a value passed to initialise the variables [same for this]

input

`m,n`	[integer]		
`ndx`	[integer]		
`solMax`	[integer]		
`flowdirns`	[integer]	[m,n]	flow directions
`solOrder`	[integer]	[m,n]	
`mask`	[integer]	[m,n]	
`alpha`	[8-byte real]	[m,n]	
`deltax`	[8-byte real]	[m,n]	
`dt`	[8-byte real]		time step
`disOld,disNew`	[8-byte real]	[m,n,ndx]	
`iex`	[8-byte real]	[m,n,2]	

output

`disOld, disNew`	[8-byte real]	[m,n,ndx]

4.3.6 Kinematic wave order

`KMWOrder``(flowdirns, m, n, solutionOrder)`

Subroutine assigns values to the matrix `solutionOrder` to tell the Kinematic Wave subroutine in which order to solve the KM equation on the drainage network. Values of 1 assigned to top of catchment, 2 to 1st downstream node etc.. Value at a point is the maximum travel distance to that point from all points above it.

input

`m,n`	[integer]		
`flowdirns`	[integer]	[m,n]	flow directions

output

`solutionOrder`	[integer]	[n,n]

Uses subroutines:
`Lookupfdir()`, cf. page 12

4.3.7 `OneDRandList(a,mn)`

`OneDRandList``(a, mn)`

input

`mn`	[integer]	
`a`	[integer]	[mn,2]

output

`a`	[integer]	[mn,2]

4.3.8 Lookup for direction

`Lookupfdir``(dirn, dx, dy)`

input

`dirn`	[integer]

output

`dx,dy`	[integer]

4.3.9 Routing with kernel

RoutingWithKernel(m, n, mn, precip, infiltKern, storeKern, newflowdirns, topog, store, discharge, outflow)

input

m,n	[integer]		
mn	[integer]		
infiltKern	[8-byte real]	[m,n]	
storeKern	[8-byte real]	[m,n]	
topog	[8-byte real]	[m,n]	
precip	[integer]	[m,n]	
store	[integer]	[m,n]	
newflowdirns	[integer]	[m,n]	flow directions

output

outflow	[integer]		
store	[integer]	[m,n]	
newflowdirns	[integer]	[m,n]	new flow directions
discharge	[integer]	[m,n]	

Notice: Periodic boundary conditions can only really be defined simply for an inclined plane with two adjacent edges defined as the boundary from which particles are routed to the opposite boundary. For simplicity it is assumed that the landscape slopes downwards in the direction of lower x and y.

Uses subroutines:
OneDRandList(), cf. page 12
TwoDRandPos(), cf. page 10
Lookupfdir(), cf. page 12
Neighbours(), cf. page 16
fdirLookup(), cf. page 18

4.3.10 Infiltration Probability

InfiltProb(veg, m, n, K0, ie, rfx, rfy, kf, Kmax, dx, dy, iProb)

This subroutine calculates the spatial distributed infiltration probability.

input

m,n	[integer]		dimensions of the spatial arrays
rfx, rfy	[integer]		maximum radius for plant effects on soil properties
kf	[integer]		
veg	[integer]	[m,n]	vegetation matrix
K0	[8-byte real]		
ie	[8-byte real]		
Kmax	[8-byte real]		
dx, dy	[8-byte real]		length scales of lattice

output

iProb	[8-byte real]	[m,n]	infiltration probability matrix

4.3.11 List Convolve

`ListConvolve``(base, kernel, convol, m, n, m1, n1)`

input

`m,n`	[integer]	
`m1,n1`	[integer]	
`base`	[integer]	[m,n]
`kernel`	[integer]	[m,n]

output

`convol`	[]	[m,n]

4.3.12 Erosion

`Erosion``(discharge, topog, newflowdirns, flowResistance, m, n)`

input

`m,n`	[integer]		dimensions of the spatial arrays
`discharge`	[integer]	[m,n]	discharge has units of $^{\mathrm{mm}}/_{\mathrm{year}}$
`newflowdirns`	[integer]	[m,n]	new flow directions
`flowResistance`	[8-byte real]	[m,n]	flow resistance; effective d_{40} grainsize in mm
`topog`	[8-byte real]	[m,n]	topography

output

`topog`	[8-byte real]	[m,n]	topography

Uses subroutines:
`Lookupfdir()`, cf. page 12

4.3.13 Splash

`Splash``(topog, veg, Dv, Db, m, n)`

input

`m,n`	[integer]		dimensions of the spatial arrays
`veg`	[integer]	[m,n]	
`topog`	[8-byte real]	[m,n]	topography
`Dv,Db`	[8-byte real]		$^{\mathrm{m}^2}/_{\mathrm{kyr}}$

output

`topog`	[8-byte real]	[m,n]	topography

Uses subroutines:
`Neighbours()`, cf. page 16

4.3.14 Find holes

`FindHoles``(newtopog,holes)`

input

`newtopog`	[8-byte real]	[:,:] topography

output

`holes`	[8-byte real]	[:,:]

Listing 1: a program listing could look like this; notice the language sensitive formatting

```fortran
 1  SUBROUTINE FindHoles(newtopog,holes)
 2  IMPLICIT NONE
 3
 4  REAL*8, DIMENSION(:,:), INTENT(IN) :: newtopog
 5  INTEGER, INTENT(OUT) :: holes
 6
 7  INTEGER :: m,n,i,j,k,l
 8  m=SIZE(newtopog,1)
 9  n=SIZE(newtopog,2)
10
11  DO i=2,m-1
12  DO j=2,n-1
13      holes=0
14      DO k=-1,1
15      DO l=-1,1
16          IF (newtopog(i,j) < newtopog(i + k, j + l)) THEN
17              holes = holes + 1
18          END IF
19      END DO
20      END DO
21      IF (holes.ge.8) THEN
22          holes = 1
23          RETURN
24      END IF
25  END DO
26  END DO
27
28  END SUBROUTINE FindHoles
```

Listing 1 shows, how code can be inserted into the document. With line numbers and Fortran specific code highlighting.

It is possible to refer to individual lines inside the listing with LaTeX commands `\label{}` and `\ref{}`, so references get updated if the code changes. For example line 16 in listing 1, where the first `IF`-condition begins.

The code in the listing could be loaded from an external file (for example the actual Fortran file).

4.3.15 Evaporation

`Evaporation``(veg, eTActual, bareE, store, tsteps, rcx, rcy, kc, dx, dy, params)`

This version cycles through sites and evaporates water from site and neighbouring sites if vegetated.

input

`veg`	[integer]	[:,:]
`eTActual`	[integer]	[:,:]
`bareE`	[integer]	[:,:]
`store`	[integer]	[:,:]
`tsteps`	[integer]	
`rcx, rcy, kc`	[8-byte real]	
`dx, dy`	[8-byte real]	
`params`	[8-byte real]	[7]

output

`store`	[integer]	[:,:]
`eTActual`	[integer]	[:,:]
`bareE`	[integer]	[:,:]

Uses subroutines:
`TwoDRandPos()`, cf. page 10

4.3.16 Neighbours

`Neighbours``(order, posij, dom, neighbs)`

input

`order`	[integer]	
`posij`	[integer]	[2]
`dom`	[integer]	[2]

output

`neighbs`	[integer]	[Nanu: (order*2+1)**2,2 [as formula?]]

4.3.17 LSDs

`LSDs``(order, posxy, topog, m, n, lsdList)`

This function returns the matrix positions:

LSD1: the position of the neighbouring cell with the steepest slope downhill

LSD2: the position of the neighbouring cell with second steepest slope adjacent LSD1

otherpos: the position of the other neighbouring cell the mirror reflection about LSD1 of LSD2

input

`m,n`	[integer]	
`posxy`	[integer]	[2]
`topog`	[8-byte real]	[`m,n`]

output

`lsdList`	[integer]	[3,2]

Uses subroutines:
`Neighbours()`, cf. page 16
`RotateArray()`, cf. page 17

4.3.18 Array rotation

`RotateArray``(list, m, n, leftorRight)`

input

`m,n`	[integer]	
`list`	[integer]	[m,n]
`leftorRight`	[integer]	

output

`list`	[integer]	[m,n]

4.3.19 Pos1D

`Pos1d``(list, m, n, match, rownum)`

input

`m,n`	[integer]	
`list`	[integer]	[m,n]
`match`	[integer]	[n]

output

`rownum`	[integer]	[m]

4.3.20 GD8 fow directions

`GD8``(topog, flowdirns, m, n)`

input

`m,n`	[integer]	
`topog`	[8-byte real]	[m,n]

output

`flowdirns`	[integer]	[m,n]	flow directions

Uses subroutines:
`makeOrds()`, cf. page 19
`Pos1d()`, cf. page 17
`endShift()`, cf. page 18
`LSDs()`, cf. page 16
`fdirLookup()`, cf. page 18
`RotateArray()`, cf. page 17

4.3.21 New GD8 flow directions

`NewGD8``(topog, lakes, flowdirns, m, n)`

Calculates flow directions following the GD8-algorithm of **?**

input

`m,n`	[integer]	
`lakes`	[integer]	[m,n]
`topog`	[8-byte real]	[m,n]

output

`flowdirns`	[integer]	[m,n]	flow directions

Uses subroutines:
`makeOrds()`, cf. page 19
`Pos1d()`, cf. page 17
`endShift()`, cf. page 18
`LSDs()`, cf. page 16
`fdirLookup()`, cf. page 18
`Neighbours()`, cf. page 16

4.3.22 fdirLookup(dirnxy, idirn)

fdirLookup(dirnxy, idirn)

input

dirnxy	[integer]	[2]

output

idirn	[integer]

4.3.23 Array sort

qsortd(x, ind, n, incdec)

This subroutine uses an order $n \cdot \log(n)$ quick sort to sort a real (double precision/8-byte) array $x(n)$ into increasing order.

input

x(n)	[8-byte real]	vector of length n to be sorted
n	[integer]	length of the array x(n)
incdec	[integer]	if positive the ind is returned so values decreasing order
incdec	[integer]	

output

ind(n)	[integer]	vector of length $\geqq$ n; sequence of indices $1, \ldots, n$ permuted in the same fashion as x would be: $y(i) = x\,(ind(i))$

`ind` is initialized to the ordered sequence of indices $1, \ldots, n$, and all interchanges are applied to `ind`. `x` is devided into two portions by picking a central element t. The first and last elements are compared with t, and interchanges are applied as necessary so that the three values are in ascending order. Interchanges are then applied so that all elements greater than t are in the upper portion of the array and all elements less than t are in the lower portion. The upper and lower indices of one of the portions are saved in local arrays, and the process is repeated iteratively on the other portion. When a portion is completely sorted, the process begins again by retrieving the indices bounding another unsorted portion.

Note: IU and IL must be dimensioned $\geqq \log(n)$ where $\log$ has base 2.

Credit goes to Robert Renka Oak Ridge Natl. Lab.

4.3.24 endShift

endShift(arr, rownum, mn, n)

input

`rown`	[integer]	
`n`	[integer]	
`mn`	[integer]	
`arr`	[integer]	[mn,n]

output

`arr`	[integer]	[mn,n]

4.3.25 makeOrds

`makeOrds``(topog, ords, m, n)`

input

`m,n`	[integer]	
`topog`	[8-byte real]	[m,n]

output

`ords`	[integer]	[size(topog),2]

Uses subroutines:
`qsortd()`, cf. page 18

4.3.26 Vegetation Change

`VegChange``(veg, m, n, vegParams, store, actualET, isEmerge, isGrow)`

input

`m,n`	[integer]	
`isGrow`	[integer]	
`isEmerge`	[integer]	
`vegParams`	[8-byte real]	[5]
`veg`	[integer]	[m,n]
`store`	[integer]	[m,n]
`actualET`	[integer]	[m,n]

output

`veg`	[integer]	[m,n]
`store`	[integer]	[m,n]
`actualET`	[integer]	[m,n]

4.4 Customizability

List of Figures

List of Tables

List of Listings

A Appendix