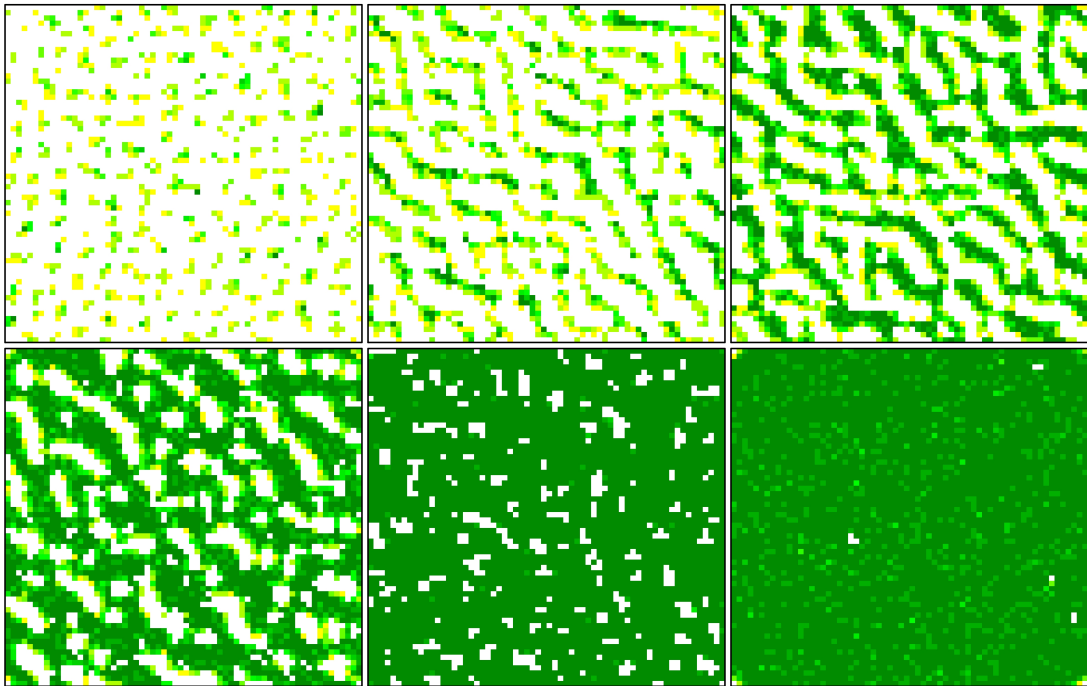


# Ecohydraulical Feedback Simulation Documentation and Manual



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 8<sup>th</sup> February, 2013, 17:58



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Theoretical principles</b>	<b>2</b>
2.1	Simulation model . . . . .	2
2.2	Runoff generation . . . . .	2
2.3	Surface water flow . . . . .	2
2.4	Short range facilitation . . . . .	2
2.5	Evaporation and transpiration . . . . .	2
2.6	Vegetation change . . . . .	2
2.7	Microtopography . . . . .	2
<b>3</b>	<b>Application</b>	<b>3</b>
3.1	Build from source . . . . .	3
3.1.1	Unix Systems . . . . .	3
3.2	Installation . . . . .	3
3.2.1	System requirements . . . . .	3
3.3	Application prerequisites . . . . .	3
3.4	Data preperation . . . . .	3
3.5	Parameterization . . . . .	3
3.6	Simulation . . . . .	3
3.7	Output . . . . .	3
3.8	Appraise and visualize results . . . . .	3
3.8.1	in R . . . . .	3
3.8.2	in R and $\LaTeX$ (knitr-method) . . . . .	3
3.8.3	in Excel . . . . .	3
<b>4</b>	<b>Implementation</b>	<b>4</b>
4.1	Overview . . . . .	4
4.2	Implementation strategy . . . . .	4
4.3	Subroutines . . . . .	4
4.3.1	Simulation . . . . .	4
4.3.2	Green-ampt infiltration . . . . .	6
4.3.3	<i>Nanu:</i> <u>TwoDRandPos</u> [cannot guess what this means] . . . . .	6
4.3.4	Kinematic wave primer . . . . .	7
4.3.5	Kinematic wave . . . . .	7
4.3.6	Kinematic wave order . . . . .	8
4.3.7	OneDRandList (a,mn) . . . . .	8
4.3.8	Lookup for direction . . . . .	8
4.3.9	Routing with kernel . . . . .	9
4.3.10	Infiltration Probability . . . . .	9
4.3.11	List Convolve . . . . .	10
4.3.12	Erosion . . . . .	10
4.3.13	Splash . . . . .	10
4.3.14	Find holes . . . . .	10
4.3.15	Evaporation . . . . .	11
4.3.16	Neighbours . . . . .	12
4.3.17	LSDs . . . . .	12
4.3.18	Array rotation . . . . .	13
4.3.19	Pos1D . . . . .	13
4.3.20	GD8 fow directions . . . . .	13

4.3.21	New GD8 flow directions . . . . .	13
4.3.22	fdirLookup(dirnxy, idirn) . . . . .	14
4.3.23	Array sort . . . . .	14
4.3.24	endShift . . . . .	14
4.3.25	makeOrds . . . . .	15
4.3.26	Vegetation Change . . . . .	15
4.4	Customizability . . . . .	16

<b>List of Figures, Tables and Listings</b>	<b>17</b>
---	-----------

<b>Index</b>	<b>20</b>
--------------	-----------

<b>References</b>	<b>22</b>
-------------------	-----------

<b>A Appendix</b>	<b>23</b>
-------------------	-----------

## 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
<code>climParams</code>	[8-byte real]	[4]			climate parameters
<code>infiltrParams</code>	[8-byte real]	[6]			vegetation and soil kernel parameters
<code>evapParams</code>	[8-byte real]	[8]			evaporation parameters
<code>discharge</code>	[integer]	[m,n]	$Q$	[m <sup>3</sup> /s]	discharge
<code>alpha</code>	[8-byte real]	[m,n]	$\alpha$		

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}
```

```
\annotate[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:

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

### 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 Build from source

#### 3.1.1 Unix Systems

To use the gfortran compiler execute the following in terminal:

```
gfortran coupledModel.f90
```

On Unix systems an `.out` file is generated. To execute this, type the following into the terminal:

```
./a.out
```

### 3.2 Installation

#### 3.2.1 System requirements

### 3.3 Application prerequisites

### 3.4 Data preperation

### 3.5 Parameterization

### 3.6 Simulation

### 3.7 Output

### 3.8 Appraise and visualize results

#### 3.8.1 in R

#### 3.8.2 in R and $\text{\LaTeX}$ (knitr-method)

<http://yihui.name/knitr/>

#### 3.8.3 in Excel



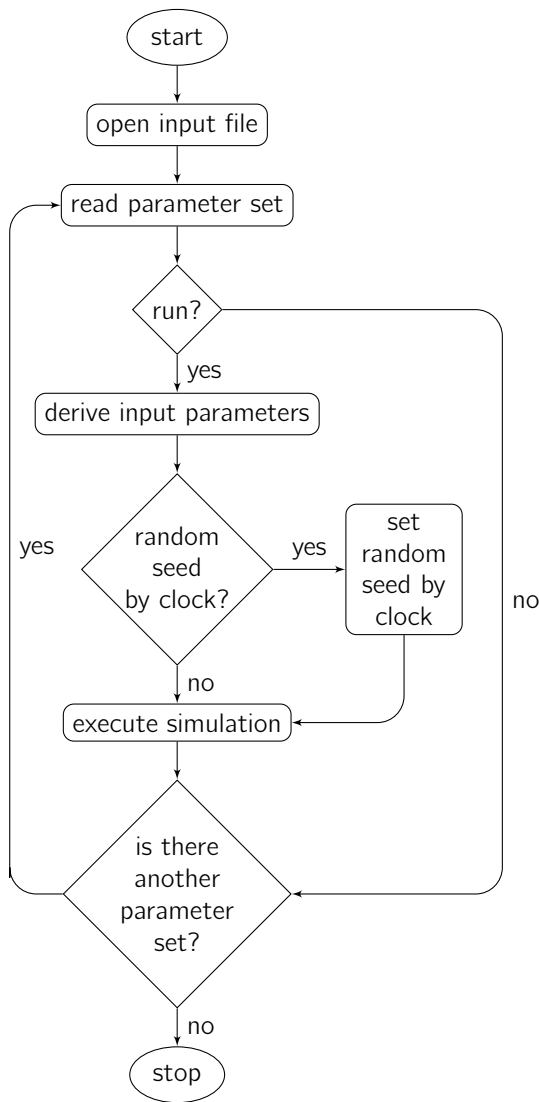


Figure 1: flow chart of the main program

## 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)

```

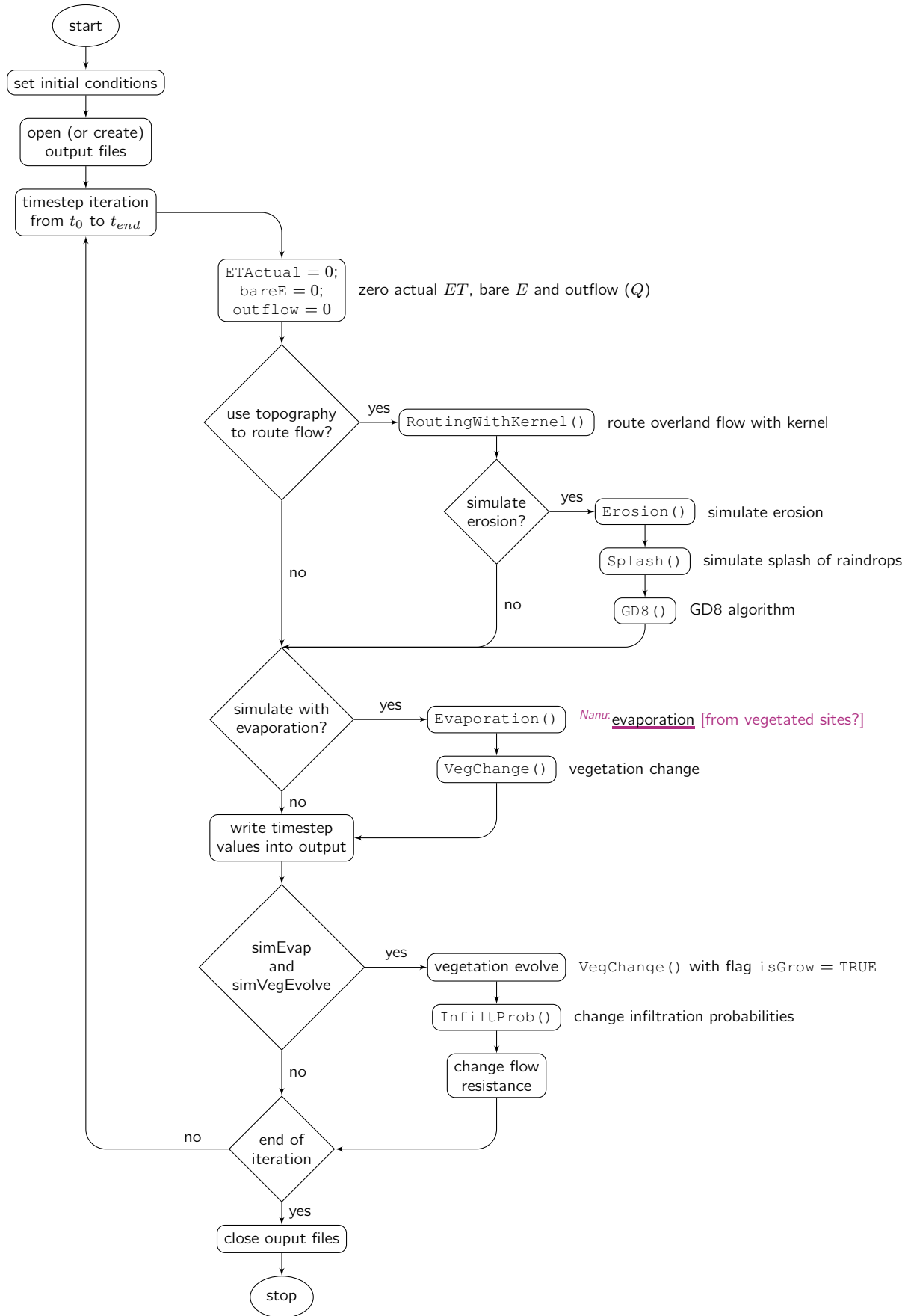


Figure 2: flow chart of subroutine SimCode (simulation execution)

**input**

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

**output**

no output

Uses subroutines:

GD8(), cf. page 13  
 InfiltrProb(), cf. page 9  
 RoutingWithKernel(), cf. page 9  
 Erosion(), cf. page 10  
 Splash(), cf. page 10  
 Evaporation(), cf. page 11  
 VegChange(), cf. page 15

**4.3.2 Green-ampt infiltration**

**GAInfiltr**(m, n, dt, inflow, Ksat, wfs, cumInfiltr, infex)

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

**input**

m, n	[integer]	
dt	[8-byte real]	
inflow	[8-byte real] [m,n]	surface runoff + precipitation ( $R + P$ )
wfs	[8-byte real] [m,n]	<i>Nanu:</i> <u>wetting front suction * moisture deficit</u> [formula?]
cumInfiltr	[8-byte real] [m,n]	<i>Nanu:</i> <u>cumulative infiltration</u> [nomenclature?]

**output**

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

#### 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 13

KMWOrder(), cf. page 8

Neighbours(), cf. page 12

Lookupfdir(), cf. page 8

#### 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 8

**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)

This subroutine routes water particles through the landscape.

#### input

m, n	[integer]	
mn	[integer]	
infiltKern	[8-byte real]	[m,n] infiltration probability
storeKern	[8-byte real]	[m,n] storage capacity
topog	[8-byte real]	[m,n] topography
precip	[integer]	[m,n] rainfall
store	[integer]	[m,n]
newflowdirns	[integer]	[m,n] flow directions

#### output

outflow	[integer]		cumulative outflow (leaving model domain)
store	[integer]	[m,n]	water infiltrated
newflowdirns	[integer]	[m,n]	new flow directions
discharge	[integer]	[m,n]	cumulative amount of overland flow

**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 8  
 TwoDRandPos(), cf. page 6  
 Lookupfdir(), cf. page 8  
 Neighbours(), cf. page 12  
 fdirLookup(), cf. page 14

### 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 mm/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 8

#### 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]	m <sup>2</sup> /kyr

**output**

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

Uses subroutines:

Neighbours(), cf. page 12

#### 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

```

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 6

**4.3.16 Neighbours**

**Neighbours**(order, posij, dom, neighbours)

**input**

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

**output**

neighbours	[integer]	[ <i>Nanu:</i> <u><math>(order*2+1)**2,2</math></u> [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 12

RotateArray(), cf. page 13

**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 15  
 Pos1d(), cf. page 13  
 endShift(), cf. page 14  
 LSDs(), cf. page 12  
 fdirLookup(), cf. page 14  
 RotateArray(), cf. page 13

**4.3.21 New GD8 flow directions****NewGD8**(topog, lakes, flowdirns, m, n)

Calculates flow directions following the GD8-algorithm of Paik (2008)

**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 15  
 Pos1d(), cf. page 13  
 endShift(), cf. page 14  
 LSDs(), cf. page 12  
 fdirLookup(), cf. page 14  
 Neighbours(), cf. page 12

#### 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 $\geq n$ ; sequence of indices $1, \dots, n$ permuted in the same fashion as x would be: $y(i) = x(ind(i))$
--------	-----------	--

ind is initialized to the ordered sequence of indices  $1, \dots, n$ , and all interchanges are applied to ind. x is divided 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  $\geq \log(n)$  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 14

**4.3.26 Vegetation Change**

**VegChange**(veg,m,n,vegmax, storEmerge, etPersist, useStorEmerge, store, actualET,isGrow)

This subroutine implements the simple algorithm to change vegetation/bare soil status and update biomass

**input**

m, n	[integer]	
etPersists	[integer]	$T_c$ [L T <sup>-1</sup> ] annual transpiration rate over which plants start to grow (transpiration threshold for carbon assimilation)
vegmax	[integer]	$b_{max}$ maximum biomass
isGrow	[integer]	
isEmerge	[integer]	
vegParams	[8-byte real] [5]	
veg	[integer] [m,n]	
store	[integer] [m,n]	
actualET	[integer] [m,n]	$T_a$ [L T <sup>-1</sup> ] the spatial distribution of actual evapotranspiration

**output**

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

Note: there is no variable for  $\Delta\tilde{b}$ , the normalized vegetation growth delta. It is simply set to *nanur*.1. [1 is not consistend with the following text! And maybe we should make this an input variable. ]

$$\Delta\tilde{b} = \frac{\Delta b}{b_{max}}$$

with  $\Delta b$  ... annualized avarage growth rate of plants. Mcgrath et al. (2012) used  $b_{max} = 2 \text{ kg m}^{-2}$  and  $\Delta\tilde{b} = 0.1$  or rather  $\Delta b = 0.2 \text{ kg m}^{-2} \text{ yr}^{-1}$

## **4.4 Customizability**

## List of Figures

1	flow chart of the main program . . . . .	4
2	flow chart of subroutine <code>SimCode</code> (simulation execution) . . . . .	5

List of Tables

1 variables . . . . . V

## List of Listings

- 1 a program listing could look like this; notice the language sensitive formatting . . . . . 11



## Index

a, 8  
actualET, 15  
alpha, 7, 8  
arr, 15  
  
bareE, 12  
base, 10  
  
climParams, 6  
convol, 10  
cumInfilt, 6  
  
deltax, 7, 8  
dirn, 8  
dirnxy, 14  
discharge, 9, 10  
disOld, disNew, 8  
disOld, disNew, 8  
dom, 12  
dt, 6, 8  
Dv, Db, 10  
dx, dy, 9, 12  
dx, dy, 8  
  
endShift(), 13, **14**, 14  
Erosion(), 6, **10**  
erosion, 10  
erParams, 6  
eTActual, 12  
etPersists, 15  
Evaporation(), 6, **11**  
evaporation, 11  
evapParams, 6  
  
fdirLookup(), 9, 13, **14**, 14  
FindHoles(), **10**  
flow directions, 13  
flowdirns, 7, 8, 13  
flowResistance, 10  
  
GAInfilt(), **6**  
GD8(), 6, 7, **13**  
GD8-algorithm, 13  
  
holes, 11  
  
idirn, 14  
ie, 9  
iex, 8  
incdec, 14  
ind(n), 14  
infex, 6  
infiltKern, 9  
infiltParams, 6  
InfiltProb(), 6, **9**  
infiltration, 9  
inflow, 6  
iProb, 9  
isEmerge, 15  
isGrow, 15  
  
K0, 9  
kernel, 10  
kf, 9  
kinematic wave, 7  
KinematicWave(), **7**  
Kmax, 9  
KMWOrder(), 7, **8**  
KWPrimer(), **7**  
  
lakes, 13  
leftorRight, 13  
list, 13  
ListConvolve(), **10**  
Lookupfdir(), 7, **8**, 8–10  
lsdList, 12  
LSDs(), **12**, 13, 14  
  
m, **6**  
m, n, 6–10, 12, 13, 15  
m1, n1, 10  
makeOrds(), 13, 14, **15**  
manningsN, 7  
mask, 7, 8  
match, 13  
mn, 6, 8, 9, 15  
  
n, 6, 14, 15  
ndx, 8  
Neighbours(), 7, 9, 10, **12**, 12, 14  
neighbs, 12  
newflowdirns, 9, 10  
NewGD8(), **13**  
newtopog, 11  
  
OneDRandList(), 7, **8**, 9  
order, 12  
ords, 15  
outflow, 9  
  
params, 12  
Posld(), **13**, 13, 14  
posij, 12  
posxy, 12  
precip, 9  
  
qsortd(), **14**, 15  
  
randOrder, 6  
rcx, rcy, kc, 12  
resultsFID, 6  
rfx, rfy, 9  
RotateArray(), 12, **13**, 13  
RoutingWithKernel(), 6, **9**  
rown, 15  
rownum, 13  
  
SimCODE(), **4**  
simflags, 6  
simulation, 4  
solMax, 7, 8

solOrder, 7, 8  
solutionOrder, 8  
Splash(), 6, **10**  
store, 9, 12, 15  
storeKern, 9  
  
topog, 7, 9, 10, 12, 13, 15  
tsteps, 12  
TwoDRandPos(), **6**, 9, 12  
  
veg, 9, 10, 12, 15  
VegChange(), 6, **15**  
vegmax, 15  
vegParams, 6, 15  
  
water balance, 2  
wfs, 6  
  
x(n), 14

## References

- Mcgrath, G. S., Paik, K., and Hinz, C. (2012). Microtopography alters self – organized vegetation patterns in water limited ecosystems. *Journal of Geophysical Research*.
- Paik, K. (2008). Global search algorithm for nondispersive flow path extraction. *Journal of Geophysical Research*, 113(F4):F04001.

## **A Appendix**