Land surface process modelling with PCRaster Python

Release 2.5

Derek Karssenberg

February 13, 2014

Download this website as pdf.

Download this website as epub (for e-readers).

To request the data set for the exercises email d.karssenberg@uu.nl

CHAPTER

ONE

MAP ALGEBRA

To request the data set for the exercises email d.karssenberg@uu.nl

1.1 PCRaster maps, data types and display

1.1.1 Introduction

The aim of these exercises is to learn how Geographical Information Systems can be used for the analysis and synthesis of spatial data in physical geography, soil science, hydrology and environmental science. The emphasis is on the understanding of the methods of analysis rather than on data structures, methods of data storage or producing high quality graphic display.

The exercises use the PCRaster Python software.

1.1.2 Reading maps from disk and visualisation

To run the software, open a Python command prompt and type at the prompt:

```
from pcraster import * <Enter>
```

This loads the PCRaster module.

To use the PCRaster maps in the exercise data set, you need to be sure that the path in your Python command prompt is directing to the folder containing the exercise data set. In many Python environments you can set this in advance, using an option in the environment. But you can also set it on the fly.

To check your path, type (still at the Python command prompt):

```
import os <Enter>
print(os.getcwd())
```

If it prints the path to the folder with your data set, proceed below at 'To use a map..'! If not, you can choose to set the path at the prompt, using *os.chdir(stringWithPath)*, where *stringWithPath* defines the path, for instance:

```
os.chdir("C:\Exercises\MapAlgebra") <Enter>
```

Check your path again by typing:

```
import os <Enter>
print(os.getcwd())
```

If all is fine, continue below.

To use a map, you first need to read it from disk:

```
waterMap=readmap("water.map") <Enter>
```

which reads the file `water.map` and assigns it to the variable `waterMap`, which is stored in memory.

Now you can visualise the map by typing:

```
aguila(waterMap) <Enter>
```

The maps provided are: `buildg.map`, `firestat.map`, `iswater.map`, `phreatic.map`, `rainstor.map`, `roads.map`, `topo.map`, `water.map`, `dump.map`, `isroad.map`, `logging.map`, `points.map`, `rainyear.map`, `soils.map`, `trees.map`, `wells.map`.

Now read the `isroad.map` from disk using the same command and use a variable name `isroadMap` and display it. You can display as many maps as you like. If you want to have the same cursor position, however, on all maps, it is better to open them at once, by using multiple maps as input of the aguila command, for instance:

```
aguila(isroadMap,waterMap)
```

Click on the maps to find some cells where you would expect a bridge, the cross indicates the cursor position. You can select 'Show cursor and values' from the Aguila menu to retrieve cell values.

Read from disk and display `topo.map`. It is the digital elevation model of the area. Right-click on the legend and select 'Edit draw properties..' to change settings. For instance, change the display to contours.

1.1.3 Data types

PCRaster uses data typing of the data in the database: each map has one of the six data types used attached to it. These data types help you and PCRaster to structure the data. See the table below.

data type	description of attributes	domain	example
boolean	boolean	0 (FALSE), 1 (TRUE)	suitable/unsuitable, visible/non visible
nominal	classified, no order	whole values	soil classes
ordinal	classified, order	whole values	succession stages, income groups
scalar	continuous, linear	real values	temperature, concentration
directional	continuous, directional	0 to 360 degrees	aspect
ldd	direction to neighbour cell	codes of directions	drainage networks

---

Question: What is the data type of `buildg.map`?

1. boolean
2. nominal
3. ordinal
4. scalar
5. directional
6. ldd

1.2 Point operations

1.2.1 Introduction

Before we continue, type the following:

```
execfile("openmaps.py")
```

It reads all maps and assigns them to corresponding variable names. For instance, the file `phreaticMap` is assigned to the variable `phreaticMap`, or `topoMap` becomes `topoMap`. If you close Python and restart it again, you need to retype this command to get all your maps loaded!!

New maps can be derived from existing maps with PCRaster functions and operators. These provde a very powerfull calculator for maps. It offers you a wide range of operators (for instance plus, minus, slope, windowaverage) that act upon input maps. If needed, the operators can be combined, in the same way as it is done in algebraic calculations. For instance the statement:

```
resultMap = slope(topoMap) * (1 + phreaticMap)  <Enter>
```

Is perfectly valid. Try it! And display the `resultMap` with Aguila.

The software contains point operations, area operations, neighbourhood operations and map operations. This section covers point operations. These operate only on the values of the map layers relating to each cell; in other words, for each cell the operation is independent of the cell value(s) of neighbouring cells.

1.2.2 Quantitative computation with scalar maps

Quantitative calculation includes several mathematical operations, such as `*` (multiply), `sin`, `ln`, `+`. The input maps of a quantitative computation must always be maps of scalar data type.

To create an overlay which contains the depth of the unsaturated zone, you can subtract for every raster cell the phreatic level (i.e., the groundwater level, `phreaticMap`) from the surface level (`topoMap`). Both maps give the level in metres above sea level. Type:

```
unsatMap = topoMap - phreaticMap <Enter>
```

Display the result.

Calculate a map that gives the depth of the unsaturated zone in mm. Use the operator `*` and call the result map `unsatmmMap`. Display the map.

Somebody wants to determine a map with the infiltration rate on basis of the soil type (`soilsMap`) and the unsaturated zone (`unsatMap`), by typing the following operation:

```
infilMap = soilsMap * unsatMap <Enter>
```

Try the operation and note the error message (particularly the last line).

Question: Why is it not possible to multiply `soilsMap` with `unsatMap`?

1. The `unsatMap` contains negative cell values, which does not make sense.
2. The data type of `soilsMap` is nominal, and nominal data types do not allow multiplication.
3. The data type of `unsatMap` is nominal, and nominal data types do not allow multiplication.
4. The `unsatMap` contains very high cell values, which does not make sense.

1.2.3 Boolean algebra operations

Boolean algebra is a combination technique that assumes that cells contain only two different values. Operations for boolean algebra can only be applied on maps with the related Boolean data type and the result will be a map of Boolean data type. On such a map, cells are said to be TRUE (a cell value of 1) if a certain attribute is located in that cell, and they are said to be FALSE (a cell value of 0) if the attribute is not located in that cell. Maps containing these conditions can be combined with boolean algebra operations.

You have maps containing the location of the roads (`isroadMap`) and the location of the surface water (`iswaterMap`). Display `isroadMap` and `iswaterMap`.

If we are interested in the location of bridges, we can assume that bridges occur where water coincides with roads. This is calculated with the `&` (and) operator as shown in the table below.

Table: Boolean values in bold provide result of the **&** operator with two Boolean inputs, `isroadMap` and `iswaterMap`, for all combinations of True and False inputs on these maps.

The operator `and` can be used to evaluate this boolean relation. Type:

```
isbridgeMap = isroadMap & iswaterMap <Enter>
```

Display the inputs and results in one statement and check the location of the bridges:

```
aguila(isroadMap, iswaterMap, isbridgeMap) <Enter>
```

Other Boolean operators are ~ (not), | (or) and ^ (xor). Type:

```
x1Map = isroadMap & iswaterMap <Enter>
x2Map = isroadMap | iswaterMap <Enter>
x3Map = isroadMap ^ iswaterMap <Enter>
x4Map = isroadMap & ~ iswaterMap <Enter>
```

And display the resulting maps.

Listed below are four cross tables.

Table A		isroadMap	
		False	True
iswaterMap	False	**False**	**False**
	True	**False**	**True**

Table B		isroadMap	
		False	True
iswaterMap	False	**False**	**True**
	True	**False**	**False**

Table C		isroadMap	
		False	True
iswaterMap	False	**False**	**True**
	True	**True**	**True**

Table D		isroadMap	
		False	True
iswaterMap	False	**False**	**True**
	True	**True**	**False**

Question: Which of the cross tables belongs to `x1Map = isroadMap and iswaterMap`?

1. Table A
2. Table B
3. Table C
4. Table D

Question: Which of the cross tables belongs to `x2Map = isroadMap | iswaterMap`?

1. Table A
2. Table B
3. Table C
4. Table D

Question: Which of the cross tables belongs to `x3Map = isroadMap ^ iswaterMap`?

1. Table A
2. Table B
3. Table C
4. Table D

Question: Which of the cross tables belongs to `x4Map = isroadMap & ~ iswaterMap`?

1. Table A
2. Table B
3. Table C
4. Table D

1.2.4 Comparison operators resulting in boolean maps

For each cell, the comparison operators define a relation between the cell value on a first input map and a second input map. If this relation holds, a boolean TRUE is assigned to the cell. If it does not hold, a Boolean FALSE is assigned to the cell. The resulting map is a Boolean map. The two input maps may be real PCRaster maps or a constant value representing a map that is totally filled with cells of that value.

Try:

highMap = topoMap > 40 <Enter>

Display topoMap‘‘ and highMap‘‘.

The syntax of a PCRaster operator defines how the operator is used (for instance the number of input maps, the order in which the input maps must be typed in, the use of brackets). For instance, the syntax of the > (greater than) operator is: `Result = expression1 > expression2`. Result is the output map that is generated, expression1 and expression2 are the two input “maps”. The name expression is used here instead of a map because it does not always need to be a map name. The expressions may be:

- names of variables (e.g. `topoMap`)
- constant values (like 40 in the example given above), or;
- operations resulting in maps.

An example of using an operation resulting in a map is:

```
testMap = topoMap > (0.234 * 19)
```

In this case `Result` is `testMap`, `expression1` is `topoMap` and `expression2` is an operation resulting in a map: (0.234 * 19), 0.234 multiplied with 19.

In addition to the > (greater than) operator, you can use == (equal), >= (greater than or equal), <= (less than or equal), < (less than) and != (not equal). These operators have a syntax that corresponds with the > operator.

Display `buildgMap` and click on the map to find the cell value used to represent the mine (if you don’t see the legend, it is code 5 in the legend). Make a boolean map, call it `isMineMap`, that is TRUE at cells with the mine and FALSE at cells without the mine. Use the `buildgMap` and the == operator. Display `isMineMap` together with the `buildgMap`.

Give the command you typed to get `isMineMap`.

Question: What command did you use to get `isMineMap`.

1. `isMineMap = buildgMap = 5`
2. `isMineMap = buildgMap == mine`
3. `isMineMap = buildgMap == 5`
4. `isMineMap = buildgMap = mine`

1.2.5 Conditional operators with a boolean map

The conditional operators use an input map of Boolean data type to determine whether a first expression, a missing value or a second expression must be assigned to a particular cell.

There are two conditional operators. The first conditional operator is `ifthen`, with the following syntax: `Result = ifthen(condition,expression)` where `Result` is the output map, `condition` is a Boolean map or

expression and `expression` is the expression that is assigned to Result in those cells where condition has a TRUE value (cell value 1). For those cells where condition is FALSE (cell value 0) the value of Result is not defined and a missing value is assigned.

The elevation at the mine can be determined with `isMineMap` and `topoMap`. The `isMineMap` was created during the previous exercise. Type:

```
topatminMap = ifthen(isMineMap, topoMap) <Enter>
```

Display `topoMap`, `isMineMap` and `topatminMap`. The map `isMineMap` is TRUE (cell value 1) at the mine and the resulting map `topatminMap` has the value of `topoMap` for that area. The map `isMineMap` is FALSE (cell value 0) for all the remaining cells and `topatminMap` is assigned a missing value for these cells.

The second conditional operator is the if then else operator, with the following syntax: `Result = ifthenelse(condition,expression1,expression2)` where `Result` is the output map, `condition` is a boolean expression, `expression1` is the expression that is assigned to `Result` in those cells where condition has a TRUE value (cell value 1) and `expression2` is the expression that is assigned to `Result` in those cells where `condition` has a FALSE value (cell value 0)

In the future, the mine on `isMineMap` will be used for open-cast mining (Dutch: 'dagbouw'): 20 metres of ground will be removed at the mine. As a result the surface level at the mine will decrease with 20 metres compared to the current surface level (given on `topoMap`).

Calculate the elevation map of the whole study area (no missing values) that will result after digging down 20 metres of ground at the mine. Call this new elevation map `toponewMap`. Use the `ifthenelse operator`.

To answer the following question, calculate the lowest elevation value on `toponewMap`:

```
lowestPointOnTopoNew = mapminimum(toponewMap)
```

Question: What is the lowest elevation value on `toponewMap`?

1. 1.0
2. -1.0
3. 3.0
4. 12.0

1.3 Area operations for descriptive statistics

1.3.1 Introduction

Area operations compute a new value for each cell as a function of existing cell values of cells associated with an area containing that cell. The area operations are like point operations to the extent that they compute new cell values on basis of one or more map layers. Unlike point operations, however, each cell value is determined on the basis of the several cell values in the zone containing the cell under consideration. In most cases the operation represents a descriptive statistics calculation, for instance the mean value over each area.

1.3.2 Calculation of statistics of an area

For calculation of a map that contains for each soil class the average elevation in the soil class, the `areaaverage` operator is used. Try:

```
soiltopoMap = areaaverage(topoMap, soilsMap) <Enter>
```

Display `soiltopoMap`, `topoMap` and `soilsMap`.

Question: What is calcualated by the `areaaverage` operation in this case?

1. The operation assigns to each cell the average elevation value of cells in a map that belong to the same area (class on `soilsMap`) as the cell itself.
2. The operation assigns the area of the area (class on `soilsMap`) to which the cell belongs to the cell itself.
3. It gives an average idea of the distribution of the elevation in the soil classes.

For assigning the area of the soil class to which each cell belongs, the `areaarea` operator can be applied. Try:

```
soilareaMap = areaarea(soilsMap) <Enter>
```

Display `soilareaMap` and the `soilsMap`.

Question: What is name or cell code of the soil class that covers the largest area in the study area?

1. Gravel, with cell code 8.
2. Sand, with cell code 9.
3. Boulder clay, with cell code 10.

1.3.3 Finding contiguous areas; a study problem

The pine forests in this area are partly used for commercial logging. One of the constraints to evaluate whether a certain patch of pine trees will produce enough yield to create some profit when logged, is the size of the patch. Logging of pine trees is economically feasible only for patches (contiguous areas) with pines that are larger than 4 hectares. The goal of this question is to create a boolean map that has TRUE cell values for patches with pine that are feasible for logging and a boolean FALSE cell values for the remaining area.

Display `treesMap`.

If you cannot see the codes, it has the following cell codes:

- open, code 0
- pine, code 1
- deciduous, code 2
- mixed wood, code 3

Create a boolean map (call it `pineMap`) that contains a 1 (TRUE) for cells with pines and a 0 (FALSE) for cells without pines. Use the `treesMap` in an operation with ==. Display the map you just created, `pineMap`.

To be able to determine the size of each of the individual patches of pine-trees a separation in contiguous areas on `pineMap` has to be made. This is done with the clump operator. It groups in a boolean, nominal or ordinal map all those cells that have the same class-value and neighbour each other. Have a look at the PCRaster manual page for `clump` to get more information.

Every group ('clump') of cells satisfying these conditions is assigned a new class-value in the resulting map. Type:

```
pineclumMap = clump(pineMap) <Enter>
```

And display `pineMap` and `pineclumMap`.

Now, calculate a map (call it `pineareaMap`) that contains for each clump on `pineclumMap` the area of the clump under consideration. Use the `areaarea` operator. Display the `pineareaMap` together with `pineMap` by typing:

```
aguila(pineareaMap,pineMap) <Enter>
```

Note that the maximum value in the legend of `pineareaMap` is the large area without pine trees (which is also calculated by `areaarea`). As a result, some patches are hidden by this somewhat unpractical scale of the legend of `pineareaMap`. To get the areas of the patches of pine trees, click with the mouse on `pineMap` and read the values on `pineareaMap` from the data view.

You can change the cell values in the area without pine trees to 0 with the `ifthenelse operator`. Type:

```
pineare2Map = ifthenelse(pineMap, pineareaMap, 0) <Enter>
```

Display `pineMap` and `pineare2Map` and use the mouse.

Question: What is the data type of `pineare2Map`?

1. Nominal
2. Ordinal
3. Boolean
4. Scalar

Now you can find the answer to the study problem. Use the `pineare2Map` to find contiguous areas with pines that are larger than 4 hectares. Use the > operator in an operation with `pinare2Map`. Note that each cell is 50 x 50 m2.

Question: How many contiguous patches (clumps) exist greater than 4 hectares?

1. 1
2. 2
3. 8
4. 7

1.4 Neighbourhood operations: windows, frictions paths, visibility analysis

1.4.1 Introduction

Neighbourhood operations relate a cell to its neighbours. The value of each cell is changed on basis of a relation with neighbouring cells or flow of material (for instance water) from neighbouring cells. A rich suite of neighbourhood operators is available in PCRaster. This section introduces you to the window operators, operators for calculating distances over a map (spreading) and operators for visibility analysis. The next section covers neighbourhood operators for catchment analysis.

1.4.2 Direct neighbourhood operations: window operations

The window operators calculate statistical values (for instance average value, maximum value) of cells within a window that moves over the map. For each cell a new value is calculated on basis of the cell values within a square window around the cell. A wide range of window operators is available, but in the exercises only the windowaverage and windowdiversity will be used.

The map of the unsaturated zone (`unsatMap`) is rather 'noisy' and there are relatively many spikes and holes. This may be partly due to the interpolation routines used to create the surface elevation map and the phreatic level map. To create a smoother map from the `unsatMap` the `windowaverage` operator can be used.

The syntax of the windowaverage operator is: `Result = windowaverage(expression, windowlength)` where `expression` is the input map of scalar data type. The `windowlength` defines the size of the window used, it is the length of the square window in the distance units used on the maps.

The next `windowaverage` operation requires the `unsatMap`, created in a previous section. If you have not yet created it in your current session, recreate it by typing:

```
unsatMap = topoMap - phreaticMap <Enter>
```

Then, type:

```
unsat150Map  = windowaverage(unsatMap,150) <Enter>
unsat250Map  = windowaverage(unsatMap,250) <Enter>
```

Question: What is the data type of `pineare2Map`? What is the size of the window, counted in number of cells, used for the calculation of `unsat150Map`? Keep in mind that the size of one cell is equal to 50 by 50 metres!

1. 1 cell by 1 cell
2. 2 cells by 2 cells
3. 3 cells by 3 cells
4. 4 cells by 4 cells

Question: How does the windowaverage operator affect the frequency distribution of the values in the maps?

1. The windowaverage operator decreases the variation in the map and thus the frequency distribution. The larger the window, the more the variation is diminished.

2. The windowaverage operator increases the diversity in the map and thus enlarges the frequency distribution. The larger the window, the more the variation is enlarged.
3. The windowaverage operator does not affect the frequency distribution at all, since it originated from the data retreived from the input map.

The spatial diversity of an area can be studied with the windowdiversity operator. It has a syntax that corresponds with the windowaverage operator, but the input expression must have a data type boolean, nominal or ordinal. Try:

```
soidi150Map = windowdiversity(soilsMap,150) <Enter>
```

Also try it for different window lengths, e.g. 500.

Question: Explain the operation performed by the windowdiversity operator.

1. The windowdiversity operator finds the number of different cell values within a window and assigns this number to the cell for the result.
2. The windowdiversity operator determines the diversity within a window by calculating the differences between the cell values in that window.
3. The windowdiversity operator calculates the standard deviation per window and assigns that value to the cell.

1.4.3 Entire neighbourhood operations: absolute distance calculation

The spread operator is used to calculate for each cell the shortest distance to non zero cell values on a boolean, nominal or ordinal map. This distance may be absolute (the real distance) or relative (taking into account frictions). Both kind of distances can be calculated with the same spread operator.

First, calculation of the absolute distance will be explained. The next section covers relative distances.

The syntax of the spread operator is: `Result = spread(pointsexpression, initialdist, friction)` where `pointsexpression` is a map of data type boolean, nominal or ordinal. The `initialdist` and `friction` are meant for calculation of relative distances (see next section), if set to 0 and 1 respectively the absolute distance will be calculated, like it will be done in this section.

The `wellsMap` is a boolean map with wells used for drink water supply in the area. Display it. Now, try:

```
welldistMap = spread(wellsMap,0,1)  <Enter>
```

And display inputs and outputs.

Protection zones are to be designated in the area surrounding wells used for drinking water supply. Create a boolean map (call it `wellprotMap`) that is TRUE in the area within 200 m distance (excluding a distance of 200 m itself) from wells and FALSE in the area further away from wells.

Question: For how many wells do their protection zones overlap (or are connected)?

1. 2
2. 11
3. 4
4. 7

1.4.4 Entire neighbourhood operations: relative distance calculation

In the previous section, the friction map in the spread operator was set to 1 to obtain the absolute distance. By defining a different friction value or a map with friction values, friction can be included in the calculation and relative distances (for instance time) can be calculated. For instance, assume that the water is transported from the wells by a waterworks network (Dutch 'waterleiding netwerk'). On average, it takes 3 seconds for the water to be transported one metre. A map with the total travel time for the water from the nearest well to each cell is calculated as follows:

```
welltimeMap = spread(wellsMap,0,3) <Enter>
```

Display `welltimeMap`. It gives for each cell the time needed for the water to reach the cell under consideration (from the nearest well). The spread operator has multiplied the distance (in metres) with the time needed to move one metre.

1.5 Neighbourhood operations: DEM and catchment analysis

1.5.1 Introduction

A neighbourhood operation relates the cell to its neighbours. The value of each cell is changed on basis of some kind of relation with neighbouring cells or flow of material (for instance water) from neighbouring cells. This chapter will cover neighbourhood operators for catchment analysis.

1.5.2 Creating slope and aspect maps

The slope operator generates a slope map on basis of a digital elevation model (a fraction: increase in height per distance in horizontal direction, in a 3 x 3 cells window). Try:

```
slopeMap = slope(topoMap) <Enter>
```

Display `slopeMap` and `topoMap`.

With the atan operator, the slope given as an angle is calculated. Type:

```
slopedegMap = atan(slopeMap) <Enter>
```

Apply the operator `slope` to `slopeMap`. Type:

```
slope2Map = slope(slopeMap) <Enter>
```

The `slope2Map` gives the change in slope (second derivative).

The aspect operator results in a map of slope aspect in 360 degrees (clockwise, North is to the top of the map and is 0 degrees). Type:

```
aspectMap = aspect(topoMap) <Enter>
```

1.5.3 Creating the local drain direction map

In different kinds of environmental studies, such as erosion, surface runoff and hydrological research, it is important to be able to delineate a river course and drainage area. Digital Elevation Models (such as `topoMap`) offer the possibility for automated extraction of catchment characteristics by creating a local drain direction map which gives the flow pattern over a DEM.

The lddcreate operator is used to generate a local drain direction map. Its syntax is: `Result = lddcreate(dem, outflowdepth, corevolume, corearea, precipitation)` where `dem` is the digital elevation model. The other input maps (or constant values) are so-called pit removing thresholds, these are explained later.

First, set the pit removing thresholds to zero:

```
ldd0Map = lddcreate(topoMap,0,0,0,0) <Enter>
```

Display `topoMap` and `ldd0Map`.

The lddcreate operator has determined for each cell the direction of the steepest (downhill) slope, which is the direction of local drainage. For each cell, the drain direction is assigned to the local drain direction map `ldd0Map`.

The cells with a black square at the edge of the map represent outflow points from the map (you may need to zoom in a bit to see them). There are two similar cells `ldd0Map` that are not at the edge. These are called pits. A pit cell is surrounded by cells at a greater elevation than the pit cell itself. As result, a pit cell cannot drain to a neighbouring cell. In PCRaster, the cells can be assigned unique nominal values with the pit operator. Try:

```
pit0Map = pit(ldd0Map) <Enter>
```

Display the maps, including `topoMap` (in one `aguila` command). Enlarge the window of `pit0Map` and check the elevations at and around pit cells by clicking with the mouse on these cells and reading the topoMap values from the data window.

You can find the downstream paths over a local drain direction map with the path operator. It uses an boolean input map. All cells that are downstream of a TRUE cell on the boolean map are assigned a boolean TRUE on the resulting map.

The map `pointsMap` contains some arbitrary points with a boolean value TRUE. Display `pointsMap`. After that, try:

```
pathzeroMap = path(ldd0Map,pointsMap) <Enter>
aguila(ldd0Map, pointsMap, pit0Map, pathzeroMap) <Enter>
```

Compare `ldd0Map` and `path0Map`. You will see that the paths stop at the pit cells on the local drain direction map.

Two kinds of pits may occur in a DEM. The first kind is caused by natural depressions and sinkholes in a landscape, see the Figure below.

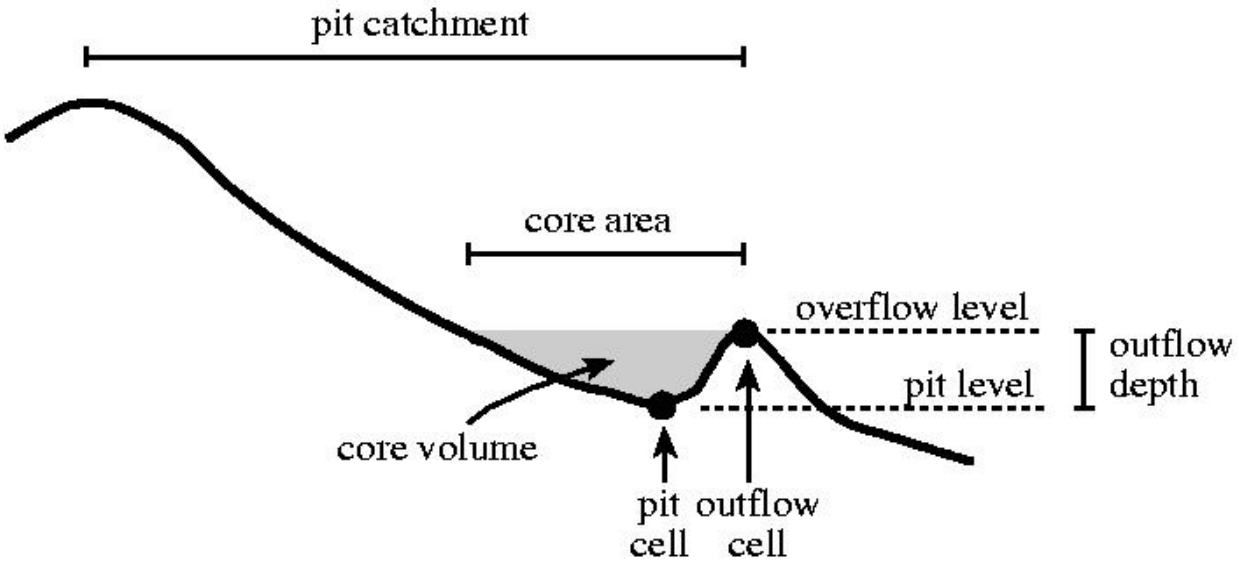

Figure: A sprinkling of the definitions used in theory of pit removing.

The pit will be at the cell with the smallest elevation in such a depression. The core of the pit is the area that will contain water if it would be filled with water until the overflow level is reached. The second kind of pits is introduced by the DEM itself. Due to the discretization of the elevation surface, the DEM is not an exact description of the landscape. As a result pits may occur on the DEM at locations without any real depressions in the landscape. These will have small and shallow cores on the DEM.

You can fill up the cores of all pits in a digital elevation model with the `lddcreatedem` operator. It generates a modified digital elevation model: the cores are filled up until the overflow level of the cores is reached. This can be compared with fluviatile sedimentation in the core depression until a maximum sedimentation level is reached (which is the overflow level of the pit). Type:

```
topomodiMap = lddcreatedem(topoMap,1e31,1e31,1e31,1e31)   <Enter>
```

The `1e31` values effect that all cores will be filled up. To find the difference between topoMap and topomodiMap (which is the depth of the pit cores), type:

```
coredeptMap = topomodiMap - topoMap <Enter>
```

Question: What is the maximum depth of the largest core in the map (in metres)?

1. 1.2
2. 4.0
3. 0.1
4. 3.0

For further analysis it is necessary to remove the pits from the local drain direction map. This can be done by changing the pit remover thresholds given in the `lddcreate` operator. These are: the depth of the pit, `outflowdepth`; the volume or area of the pit core, `corevolume` and `corearea` respectively and the amount of rain needed to fill up the pit core, `precipitation`. With these thresholds you can define which pits must be removed from the map. Pits that have dimensions smaller than the threshold dimensions will be removed, pits that are larger than one (or more) of the thresholds remain in the local drain direction map. In this exercise, the pit depth (outflowdepth) will be used. Try:

```
ldd2Map = lddcreate(topoMap,2,1e31,1e31,1e31) <Enter>
```

This operation removes all pits except these with a depth that is larger than or equal to 2 metre. To find the pits in `ldd2Map`, type:

```
pit2Map = pit(ldd2Map) <Enter>
```

To remove all the pits, also the ones on the sides of the map, using very high threshold values (here 1e31, i.e. 10 to the power of 31), type:

```
lddMap = lddcreate(topoMap,10,1e31,1e31,1e31) <Enter>
```

Calculate the downstream paths and display:

```
pathMap = path(lddMap, pointsMap) <Enter>
```

You will see that all paths end at an outflow cell. For further analysis, `lddMap` will be used.

1.5.4 Catchment analysis and routing with the ldd map.

The catchments of all outflow points are determined with the catchment operator.

Use the pit operator to create a nominal map with the outflow points (call it `outpoMap`) from the study area, uniquely numbered. Use `lddMap` which you created in the previous exercise. Now, use `catchment`, type:

```
catchmsMap = catchment(lddMap,outpoMap) <Enter>
```

For each non zero cell value on `outpoMap` its catchment has been determined and all cells in its catchment have been assigned this non zero value on catchmsMap.

The main use of the local drain direction map is for routing of surface water in downstream direction. The accuflux operator transports an input of material (for instance rain) downstream over the local drain direction network to the outflow cells. It calculates for each cell the material flux over the ldd during transport. All material is transported. The syntax of accuflux is: `Result = accuflux(ldd,material)` where `ldd` is the local drain direction map and `material` is a map of scalar data type that contains the input of material.

The map `rainstorMap` gives the amount and distribution of rain (m^3 per cell) that falls during a short rainstorm. Display the map. The discharge in the study area as a result of the rainstorm is calculated with the `accuflux` operator. It is assumed that no baseflow, no interception and no infiltration occurs. Type:

```
dischMap = accuflux(lddMap,rainstorMap) <Enter>
```

Display `dischMap`, `rainstorMap` and `lddMap`.

Question: What is the maximum value of the total discharge (m^3) as a result of the thunderstorm?

1. 1179.69
2. 12.69
3. 117969
4. 1.17969

By taking the base10 logarithm of dischMap you can get a better picture of the spatial pattern of the discharge. Type:

```
dischlogMap = log10(dischMap) <Enter>
```

Alternatively, you could change the view of `dischMap` in Aguila, right click on its legend, select 'Edit draw properties' and change colour assignment to True logarithmic. You may need to set the minimum cutoff to 0.1 as log(0) is not possible.

The local drain direction map can also be used for the calculation of the upstream area map. The upstream area map contains for each cell the area of the cell its catchment (including the cell itself) that drains to the cell. The map is calculated with the `accuflux` operator. Instead of defining a material input for the material map in the operation, a map that contains for each cell the area of a cell (2500 m^2) is used for material. As a result, this area 'drains downstream' and the catchment area of each cell is calculated. Try:

```
upareMap = accuflux(lddMap,2500)
```

Display `upareMap` and `lddMap`.

CHAPTER

TWO

DYNAMIC MODELLING

2.1 Visualisation of spatio-temporal data

2.1.1 Static data

PCRaster includes the Aguila software for visualisation of spatio-temporal stochastic data. With Aguila, you can easily explore large multi-dimensional data sets. For the exercises below, you will need to look up some options in the Aguila manual, available at http://pcraster.geo.uu.nl/pcraster/4.0.0/doc/aguila/Manual.html.

You will use the data from an Alpine area that will also be used for the construction of the distributed model in the dynamic modelling exercises.

Aguila is started from a command shell. On Linux or Mac this would be a standard terminal window. On Microsoft Windows, you need the PCRaster Command Prompt, which can be opened from Start, Programs, PCRaster, Command Prompt. Open a command shell and go to the directory containing your data for the exercises. Type cd to change directories. In the directory, you will find the folder `snowmelt` containing the data for the visualisation practical. In this directory, display the digitial elevation model and another map with Aguila:

```
aguila dem.map anArea.map <Enter>
```

Use the menu items (or right click on the legend to get more options) to zoom in/out, to change the color palette, and to change the drawing mode to contours. Also try changing the number of classes shown (or number of contours).

Question: What is shown on anArea.map?

1. True cells represent the higher part of the study area.
2. Idem, lower part.

You can also combine multiple maps in one view:

```
aguila anArea.map + dem.map <Enter>
```

The second map is draped over the first map. Make `dem.map` transparent by changing its display mode to contours. Now you are able to see both maps in one view.

2.1.2 Temporal spatial data

Temporal spatial data can be animated in PCRaster. The dataset contains a timeseries of maps with the amount of precipitation (m/day) in the area. It consists of 181 maps, each map contains precipitation for one day. The first map represents July 1st. Type `dir <Enter>` (or `ls` if you are on Unix) to get a list with all files in your dataset. You

will see it contains the files `precip00.001` up to `precip00.181`. These are the precipitation maps. To display them, type:

```
aguila --timesteps=[1,181,1] precip <Enter>
```

Animate the maps by clicking on the yellow menu item at the top of the view. Create a time series plot by right clicking on the legend, selecting 'Show time series'. The time series is plotted for the current cursor position on the map (so you can get a time series for any location, try it!).

Now, type:

```
aguila --timesteps=[60,181,10] precip <Enter>
```

Animate. It has opened a subset of maps now, i.e. map 60, 70, 80,.. up to 180. So the first value (60) that you provide between the square brackets is the first time step to be shown, the second value (181) is the last step, and the third value (10) is the number of time step between the maps that are shown.

Question: Compare the precipitation pattern with the elevation map (open them in one view, using a single Aguila command). What is the relation between precipitation and elevation?

1. There is no relation.
2. The higher the elevation, the higher the precipitation.
3. The lower the elevation, the higher the precipitation.

2.1.3 Temporal non-spatial data

You can also use observed timeseries of data in Aguila, e.g. observed precipitation at a meteo station. Such data are non-spatial (single location, or sometimes multiple locations). Have a look at the contents of a rainfaill timeseries included in the data set by typing:

```
type precip.tss
```

(on a Unix system you will use `less`). The time series file is an ASCII file (just like for instance python programs being ASCII files) and thus can be displayed with the standard command `type`. You could also open it in an ascii editor. The first column contains the time steps numbers (here each time step represents a day), the second column the observed value (here: precipitation in m/day). Display the timeseries by typing:

```
aguila precip.tss
```

2.2 The dynamic modelling framework

2.2.1 The dynamic modelling class

For dynamic (temporal) modelling, PCRaster comes with a predefined class. It will make it relatively simple to do time iterations, and to define inputs and outputs of a model. Open the file `dynMod.py` in your editor, on Microsoft Windows we recommend for instance the standard Python editor (IDLE) that comes with Python (available at Start, Programs, Python, IDLE). The script is the most basic script for dynamic modelling, it does only contain the statements that are needed for the control flow of the program. When you start a project, you can take this template and add statements to construct a model. But first run this 'empty' model.

Question: What does it print?

1. It prints dots; each dot represents a second run time.
2. Idem, each dot represents a time step.

When constructing the model, you will only add statements to the initial and dynamic methods. For more advanced models you may want to add also calculations outside these methods, but for now, you will see that the initial and dynamic methods provide most of what you need.

Now let's see how it executes things that you add to the initial and dynamic. Replace the `pass` statement in the initial with

```
print 'running the initial'
```

Replace the `pass` statement in the dynamic with

```
timeStep = self.currentTimeStep()
print 'running the dynamic for time step: ', timeStep
```

Save and run the modified model.

Question: What is the order in which the initial and dynamic are executed?

1. The initial and dynamic are executed at the same time.
2. First the initial is executed once. Next, the dynamic is executed 10 times.
3. The initial is not executed, it only runs the dynamic.

2.2.2 Modelling with feedback

Now, let's do forward modelling with feedback. We have a reservoir that is defined by the differential equation:

dx/dt = -cx

In hydrology, this is known as a linear reservoir. The variable x is the amount of water in the reservoir, and c is a rate constant. The script `feedback.py` solves the linear reservoir equation using an explicit solution. Open the script and run it.

The initial sets the initial content of the reservoir. Here it merely uses an initial value that is converted to another unit, using `conversionValue`. For the variable x in the differential equation, the script uses the variable name `reservoir`. The dynamic solves the differential equation by subtracting each timestep the `outflow` from the `reservoir`. Note that the variable `reservoir` is preceded by `self.`. This is needed because all other variables, e.g. `conversionValue` are local variables, i.e. they exist only in the method (initial or dynamic) were they have been defined. Preceding a variable with `self.` makes it a member variable of the class. This means it exists (can be used) in all other methods.

Add to the dynamic the line

```
print conversionValue
```

and run the model.

Question: What is printed? How can you solve this error if you really want to print it in the dynamic?

1. An error message is printed, indicating there is a typo in the script.
2. An error message is printed, which indicates that `conversionValue` is defined as a local variable. The solution is to use `self.conversionValue` throughout the script.
3. An error message is printed, which indicates that `conversionValue` is not defined as a global variable. The solution is to use `self.conversionValue` throughout the script.

An extension to the reservoir equation would be to add constant inflow of 0.5:

dx/dt = -cx + 0.5

Modify the script to simulate this inflow.

Question: What did you add to the script? Where did you type it?

1. I added `+ 0.5` to `self.reservoir` in the `initial`.
2. I added `inflow = 0.5` to the `initial` and changed the line in the dynamic to: `self.reservoir = self.reservoir - outflow + inflow`
3. I added `self.inflow = 0.5` to the `initial` and changed the line in the dynamic to: `self.reservoir = self.reservoir - outflow + self.inflow`

Question: If you run the model for a long time, what happens to the reservoir?

1. The reservoir decreases until it reaches 5.
2. The reservoir increases until it reaches 5.
3. The reservoir decreases until it reaches 3.
4. The reservoir increases until it reaches 3.

2.3 Reading and writing spatio-temporal data

Python data types (e.g. floating points) can be printed, written to disk, or read from disk using functions that come with standard Python. Think of `print` to print a variable on the screen, or the file objects to read and write data to disk. As these functions do not work with PCRaster maps, PCRaster Python provides its own functions for reading and writing map data.

2.3.1 Reading and writing static spatial data

Open the script `dynMod.py` and save it under the name `staticMap.py`. In the initial, add the following lines of code:

```
dem = self.readmap('dem')
slopeOfDem = slope(dem)
self.report(slopeOfDem,'gradient')
```

Run the model script. The `readmap` is a function to open a map stored on disk, you do not need to give the suffix `.map` of the filename that is opened (`dem.map`). The `report` function writes a map (here, `slopeOfDem`) to disk under a name given as the second argument, which is of data type string. When used in the `initial` (as you did here), it stores a single map, here the filename will be `gradient.map`.

Check the content of `gradient.map` by displaying it.

2.3.2 Reading and writing temporal spatial data

A very similar approach can be followed for temporal map data. However now we need to add statements to the `dynamic`. Let's import the time series of precipitation maps that you displayed in the visualisation part of the exercises to the model, convert it to mm/hours and write the result as a timeseries of maps.

Open `staticMap.py` created in the previous section and save it as `dynamicMap.py`. In the dynamic section:

- Remove `pass`.
- Add a statement that reads the precipitation from disk using `self.readmap('precip')`.
- Add a statement converting from m/day to mm/day.
- Write the result to disk using `self.report` (syntax is the same as in the `initial`). As second argument of the `report` function, use `pmm`.

Also, change the number of timesteps that the model is run to 181.

Save and run the script. In the PCRaster command shell, type `dir` (or `ls` on unix) to see what is stored. If everything is OK, you should see the filenames `pmm00000.001`, `pmm00000.002`, etc.

Question: Use aguila to animate the original precipitation timseries of maps that you read from disk, together with the same precipitation converted to mm/day (one Aguila command). Check the results. What command did you use?

1. `aguila --timesteps=[1,181,1] precip pmm`
2. `aguila --timesteps=1,181,1 precip pmm`
3. `aguila -timesteps=[1,181,1] precip pmm`
4. `aguila --timesteps=[1,181,1] precip.tss pmm.tss`

Now, try to make an animation of maps that shows where precipitation is above 0.01 m/day. Add a comparison operation to the dynamic that calculates a Boolean map with True where cells are above the threshold and False elsewhere. If needed use the PCRaster man pages. Store the output on harddisk using `report`. Run the model and animate the results with Aguila.

Question: At what elevation do we find particularly high values of precipitation?

1. There is no relation with elevation.
2. Large precipitation is mainly found in the valleys.
3. Precipitation is zero throughout the area.
4. Precipitation is high in the higher areas.

2.3.3 Reading timeseries

Timeseries files can be imported to a script using the `timeinput..` function. As most PCRaster functions operate on maps, the `timeinput..` function reads the value(s) from the time series for the current time step, and directly converts it to a PCRaster map that can be used in calculations. Let's read from disk the precipitation timeseries file that you had a look at in the visualisation section of the exercises.

Open `dynMod.py` and save it as `dynamicTimeSeries.py`. Change the number of time steps to 181 and replace the `pass` in the dynamic with the line:

```
precipitation=timeinputscalar('precip.tss',1)
```

As the `precip.tss` does not contain information on the data type of the data in the file, you need to provide this with the timeinput command. Precipitation are scalar data, so we use `timeinputscalar`. In case of nominal data, for instance, you would use `timeinputnominal`. The first input argument is the timeseries that is read. The second argument is the area to which the values in the timeseries is assigned. Here we want to assign the values in the timeseries value to the whole map, so we provide a *1* as second argument. This should be read as a map completely filled with one values. In case the timeseries contains multiple columns, i.e. multiple locations, you need to enter a map with regions as second argument. See the PCRaster man page on `timeinput..` for the details.

Add a `report` statement that writes `precipitation` to disk. Animate the results with Aguila.

Question: What is the content of the resulting timeseries?

1. The precipitation value from the `.tss` file is uniformly assigned to the whole map for each time step.
2. It contains a value one for each time step.
3. The precipitation is distributed according to the elevation.

2.4 Point operations: a snow melt model

2.4.1 Precipitation and temperature

In this exercise you will use the dataset introduced in the visualisation section to build a simple spatial hydrological model, including rainfall, snowfall, snowmelt, and runoff.

Open `dynamicTimeSeries.py` created in the previous exercises, save it under a new name, `temp.py`. It reads the precipitation timeseries from disk. Add a statement that reads the temperature timeseries `temp.tss` from disk and assigns the temperature values (degrees celcius) to the map variable `temperatureObserved`. Write the results to disk, run the model, and have a look at the output.

Question: Which function did you use to read the temperature from disk?

1. `open`
2. `timeinputscalar`
3. `timeinputnominal`
4. `timeinput`
5. `report`

The temperature timeseries contains the temperature observed at a meteostation that is close by. However, temperature will be spatially variable, as it depends on the surface elevation. The temperature t_i at a grid cell i can be calculated as:

$t_i = t_{obs} - l\ (e_i - e_{obs})$

In the equation, t_{obs} is the temperature observed at the meteostation, e_i is the elevation at a grid cell, e_{obs} is the elevation at the meteo station (here: 2058.1 m) and l is the temperature lapse rate (here: 0.005).

Add statements to your script `temp.py` to calculate a map (call it `temperatureCorrection`) containing the term $l\ (e_i - e_{obs})$. Write the map to disk and check the results.

Question: Where did you add the statements to calculate `temperatureCorrection`?

1. In the `__init__`.
2. In the `initial`.
3. In the `dynamic`.
4. In the `initial` and in the `dynamic`

Question: How much colder is it at the highest point in the area compared to the temperature at the meteostation (i.e., at 2058.1 m)?

1. About 2 degrees.
2. About 12 degrees.
3. About 7 degrees.
4. There is no difference.

As a next step, calculate and write to disk a time series of maps containing t_i. Save the script (`temp.py`), run and display the new output.

We assume that precipitation falls as snow when $t_i$is below zero. Add statements to calculate and store a time series of maps (use the variable name `freezing`) that is True at a cell if the temperature is below zero degrees and False elsewhere. Use a comparison operator. Run the script and check the results.

Create two time series of maps containing respectively snowfall (m/day, variable name `snowFall`) and rainfall (m/day, variable name `rainFall`). You will need the operations `ifthenelse` and a Boolean operator. Write them to disk and display the results.

Question: Which comparison operator did you use, and where?

1. ==, in the `dynamic`
2. `or`, in the `dynamic`
3. `while', in the ''initial`

d) > or < or >= or <=, in the `dynamic` a) ==, in the `initial`

2.4.2 The snow store

To simulate snow melt in each cell, a snow store is used containing a snow pack a_i(t) (m water equivalent) that changes over time:

$a_i(t) = a_i(t\text{-}1) + s_i(t) - b_i(t)$

In the equation, (t) refers to the current time step and (t-1) to the previous time step (day). The variable s_i(t) is snowfall and b_i(t) is snowmelt. Actual snowmelt b_i(t) is modelled as:

$b_i(t) = \min(a_i(t), b_{p,i}(t))$

with, $b_{p,i}$(t), potential snowmelt. Potential snowmelt is:

$b_{p,i}(t) = mt_i$, for $t_i > 0$

$b_{p,i}(t) = 0$, for $t_i <= 0$

with m, a degree day factor (here: 0.01).

Add this snow store in a stepwise manner to the model, in the following way. Let's first assume there is no snow melt. Open `temp.py` and save it as `snow.py`. Add the variable `snow` representing the snow store a_i, assuming there is no snow at the start of the model run (as it is end of summer), and increasing the snowstore with snowfall in each time step.

Question: Without snow melt (as was assumed here), what is the maximum snowpack thickness (m water equivalent) found in the area at the end of the model run? Idem, the minimum thickness?

1. Maximum thickness: 0.11 m, minimum: 0.01 m
2. Maximum thickness: 1.12 m, minimum: 0.15 m
3. Maximum thickness: 0.85 m, minimum: 0.00 m
4. Maximum thickness: 1.96 m, minimum: 0.62 m
5. Maximum thickness: 0.57 m, minimum: 0.01 m

Add snowmelt to the model. First, calculate a variable `potentialMelt`, containing $b_{p,i}$(t). Write to disk and check the results. Second, calculate actual snowmelt b_i(t). Check the results.

Question: How did you calculate actual snowmelt?

1. It is equal to the potential snow melt in this case.
2. It is a fraction of the potential melt.
3. It depends on the potential melt and the actual snow thickness.
4. It is constant, so it needs to be calculated in the `initial`.

And as the third step, update the snow store by subtracting actual snowmelt from the store. Write the snow store to disk (directly after subtracting actual snowmelt). Save the model under the same name `snow.py`.

Question: From what time step on is the main valley in the top-right corner snow-free, in spring? Idem, for the valley in the top-left corner?

1. Top-right corner: day 128. Top-left corner: day 130.
2. Top-right corner: day 110. Top-left corner: day 110.
3. Top-right corner: day 112. Top-left corner: day 140.
4. Top-right corner: day 92. Top-left corner: day 118.

2.4.3 Runoff generation

Runoff generated in each cell (m/day) is the sum of the rainfall in that cell and the snowmelt. Add a statement to `snow.py` calculating a time series of maps containing the total amount of runoff generated. Write results to disk and evaluate the results.

Question: In what time of the year is the largest amount of runoff generated? Explain your answer.

1. Melting snow generates a large amount of runoff around time step 70.
2. Melting snow generates a large amount of runoff around time step 90.
3. Melting snow generates a large amount of runoff around time step 110.
4. Melting snow generates a large amount of runoff around time step 130.

2.5 Neighbourhood operations with defined topology: the snow melt model

2.5.1 The local drain direction map

In the previous exercise you created a snowmelt model. Your model calculated for each time step (day) the total amount of runoff that was generated in a cell. Here, you will transport this runoff downstream. One approach is to assume that runoff occurs towards a neighbouring cell with the steepest downhill path. These directions are stored in a so-called local drain direction map.

Open `snow.py` and save it as `runoff.py`. Add a `lddcreate` statement calculating the local drain direction map. Run the model and have a look at the local drain direction map.

Question: Where in the script did you enter the `lddcreate` function? Explain.

1. At the top.
2. In the `__init__`.
3. In the `initial`.
4. In the `dynamic`

2.5.2 Drain all runoff within one time step: the accuflux function

When the size of a study area is sufficiently small and the time step duration sufficiently long that it can be assumed that all runoff generated in a time step reaches the outflow point in the same time step, runoff can be transported downstream using an *accu* function.

Add an `accuflux` statement to `runoff.py` calculating discharge for each grid cell. You need to calculate discharge as a result of the sum of rainfall (m/day) and snowmelt (m/day). Discharge should be calculated as m3/day. You can get the area of a cell (here in m3 as the length of the cells on the map is entered in metres when creating the map) with the function `cellarea`. Save the `runoff.py` script and run.

Animate the discharge using Aguila. Try a logarithmic scale for better visualisation (right-click on the legend, select 'Edit draw properties', set 'Colour assignment' to logarithmic *and* set the 'minimum cutoff' to a positive (non zero) value).

Question: What is the maximum discharge in the study area?

1. 0.3·10^6 m3/day
2. 1.3·10^6 m3/day
3. 2.3·10^6 m3/day
4. 1.3·10^7 m3/day

2.6 Calculating descriptive statistics: the snow melt model

2.6.1 Over the spatial domain

For analysis or application of model outputs, it is often needed to calculate statistical values (e.g., mean, maximum value) over regions or time periods. We focus first on regions.

Display the map `skiregio.map`. It contains two skiing regions in the study area.

Open the script `runoff.py` and save as `ski.py`. For evaluation of the regions for possibilities for skiing, calculate the average snowpack (m water equivalent) in each of these regions, for each time step. Use the function `areaaverage`. Write the results to disk and display. Plot a time series (right-click on the legend) and click on one of the regions to get the time series for that particular region.

Question: Which of the two regions has in general the largest amount of snow?

1. Region 1.
2. Region 2.

Although average snow pack is important, it is more important that the snowpack thickness is above a certain threshold required for skiing. Calculate for each time step:

- A map of the study area that is True at cells with a snowpack that is thicker than 0.2 m (water equivalent thickness), and
- A map with the proportion (a value between zero and one) of each area that has a snowpack that is thicker than 0.2 m.

For the second step, convert the map created in the first step to scalar data type using `scalar`. Now you can use it in an `areatotal` function. To get the fraction, you need the number of cells for each region, which can be calculated by:

```
areaarea(skiregio.map)/cellarea()
```

Store under the same filename `ski.py` and run.

Question: Compare the two regions by plotting timeseries. For approximately how many days is each region fully covered with a snow pack that is sufficiently thick for skiing ?

1. Based on the results we can conclude that there is never sufficient snow in both areas.
2. 5 days.
3. 10 days.
4. 40 days.

2.6.2 Over time

It is also possible to calculate statistics over time.

Calculate a map with the maximum snowpack thickness (water equivalent) over time. Use the function `max`. Add the required statements to `ski.py`.

Question: What is the maximum snow thickness in the bottom right corner cell?

1. 0.30 m
2. 0.91 m
3. 0.27 m
4. 1.90 m

2.7 Direct neighbourhood operations

2.7.1 Discrete attributes: cellular automata

Initializing Game of Life

Game of Life is the best known cellular automata model. It models a population using Boolean valued grid cells, where True represents an alive cell and False a dead cell. Although it uses very simple deterministic transition rules it results in pretty unexpected results, as you will see.

The following transition rules are applied for each time step (generation). For each cell, the number of eight direct neighbourcells that is True (i.e., alive) is counted. This count is used to determine what happens with the cell in the time step:

1. Birth: if the current cell is False and the count is 3, the current cell becomes True.

2. Surivival: if (a) the count is 2, or (b) the count is 3 and the current cell is True, the current cell is left unchanged.
3. Death: if the count is less than 2 or greater than 3, the current cell becomes False.

Go to the folder `neighbourhood/life` and open `life.py`. It is an almost empty model that runs only one time step. In a later phase you can increase the number of time steps. First you need to create an initial map of the distribution of alive cells. We assume that initially 10% percent of the cells is alive, at randomly chosen locations. In the initial section, add:

```
aUniformMap = uniform(1)
```

Store the result to disk and run.

Question: What is done by the `uniform` function?

1. It draws a realization between 0 and 1, for each cell independently.
2. It draws a realization between 0 and 1, for all cells at the same time.
3. It sorts values on a map, and numbers the cells according to the order.

Now, use `aUniformMap` in a calculation to create the initial distribution of alive cells. You need the PCRaster function less than (<). Name the result of this calculation `self.alive` and write it to disk. Run and look at the results.

Question: What other (in addition to <) PCRaster operations did you use to create `self.alive` ?

1. `ifthen`
2. `ifthenelse`
3. `boolean`
4. No other operation was required.

Implementing the update rules

To represent the Birth, Survival, and Death update rules, we need to add statements to the dynamic. To count the number of neighbouring cells that is True, the boolean map `self.alive` needs to be converted to a map with a scalar data type. In the dynamic, type:

```
aliveScalar = scalar(self.alive)
```

The function `scalar` assigns a scalar 1.0 to cells that are True on its input and 0.0 otherwise.

Now, add a statement to the dynamic calculating for each cell the number of neighbouring cells that is True. It should only count in a neighbourhood consisting of 8 direct neighbours (excluding the cell itself). You need to use at least the `windowtotal` function. Assign the result to the variable `numberOfAliveNeighbours` and write the result to disk. Run and compare the result with the map containing the alive cells (`self.alive`).

As a next step, let's add the birth update rule. First create a Boolean map that is True for cells with 3 alive neighbours. Write to disk and check the results. Add another statement resulting in a map (`birth`) that returns True for cells with birth. You will need the functions `pcrand` and `pcrnot`.

Question: What statement did you use ?

1. `birth=pcrnot(threeAliveNeighbours,pcrand(self.alive))`
2. `birth=pcrand(self.alive,pcrnot(self.alive))`
3. `birth=pcrand(self.alive,pcrnot(threeAliveNeighbours))`
4. `birth=pcrand(threeAliveNeighbours,pcrnot(self.alive))`

Now, add Survival. Add a statement that calculates a map (`survivalA`) that is True according to Survival rule (a) (i.e., count is 2). Also, calculate `survivalB` containing True according to Survival rule (b). Combine `survivalA` and `survivalB` to create a map (`survival`) with cells that survive.

Death does not need to be represented by a separate calculation, because it is implicit when Birth and Survival is combined.

As a last step, combine the maps `birth` and `survival` that updates the `self.alive` map. Write the `self.alive` map to disk at the bottom of the dynamic, and run the model again. By comparing the `self.alive` map in the initial and after the first timestep, check whether your script really follows the three update rules of Game of Life.

Now, run the model for 100 timesteps by changing the number of timesteps at the bottom of the script. Animate the `self.alive` maps. Do you see similar patterns given at at http://mathworld.wolfram.com/GameofLife.html ?

Question: How did you combine the `survivalA` AND `survivalB` maps?

1. `survival=pcrand(survivalA, survivalB)`
2. `survival=pcror(survivalA, survivalB)`
3. `survival=pcrxor(survivalA, survivalB)`
4. `survival=survivalA + survivalB`

2.7.2 Continuous attributes: spatial diffusion in vegetation modelling

A logistic growth model

Logistic growth of vegetation can be modelled as:

$$\frac{dB}{dt} = rB(1 - \frac{B}{k}) - (c_0 + c_{inc}t)\frac{B^2}{B^2 + 1} + dD_{x,y} + e$$

With:

B, biomass
r, growth rate (0.08)
k, carrying capacity (10)
c_0, initial grazing pressure (0.1)
c_{inc}, increase in grazing pressure (0.00006)
d, dipsersion rate (0.01)
$D_{x,y}$, dispersion
e, random noise

In your exercises directory, go to the directory `neighbourhood/growth` and open `growth.py`. Program the logistic growth model assuming d, $D_{x,y}$, and e are zero. The initial value of B is 8.5. Use an explicit solution, which means that for each timestep, you calculate dB/dt according to the equation above and simply add it to B. Use `self.x` as variable name for the biomass. Initialize it in the initial by typing:

```
self.x = spatial(scalar(8.5))
```

This is needed to tell PCRaster that `self.x` is a map (`spatial`) of data type scalar. Another hint: you can update the grazing pressure, i.e. the term $(c_0 + c_{inc}t)$ by typing in the dynamic:

```
self.c = self.c + self.cI
```

where `c` is the grazing pressure and `cI` is the increase in grazing pressure. In the initial you need to assign the initial value to grazing pressure.

Run the model for 2500 time steps and look at the result.

Question: What is the change in the biomass over time?

1. It remains almost constant.
2. It gradually increases.
3. It increases exponentially.
4. It gradually decreases and then drops quickly.

The observed change is known as a critical shift or critical transition in the literature. This means that the state of the ecosystem abruptly changes while the change in the driver is gradual. It is notably hard to predict such critical shifts, because the change in biomass is very small until the shift (here: abrupt decrease) occurs. Early warning signals have been described that change well ahead of the shift. One early warning signal is the spatial variance of the biomass. This can only be observed when some random noise is added to the biomass.

Add a statement to the dynamic that adds `randomnoise` e to the biomass. You can create a map with random noise by typing:

```
randomNoise = normal(1)/10.0
```

Run the model again, and animate the map with the random noise, and the biomass.

Question: What is the effect of the random noise on the time series of maps of biomass?

1. The time step for the critical shift differs spatially.
2. The program hangs after some time steps due to random inputs.
3. The resulting timing of the shift changes but is the same for all cells.
4. The time series gets shorter.

Extend your script with statements that calculate the variance of the biomass map values. Remember that variance is:

$$var(B) = \frac{\sum_{i=1}^{N} (B_i - B_{mean})^2}{N}$$

With:

B_i, a biomass value at a grid cell,
B_{mean}, the mean biomass value of all grid cells,
N, the number of grid cells.

First, calculate B_{mean}, using `maptotal`, and note that the map is 40 by 40 cells. Use this map containing the mean value to calculate a map with the variance of biomass. The calculation needs to be done each time step, so add the statements to the dynamic. Store the output, run the model, display, and display the time series of variance.

Question: What is the change in the variance over time? Do you think it can be use to detect the upcoming downward shift in biomass?

1. It decreases immediately, so it cannot be used to forecast the shift, which occurs much later.
2. It increases immediately, so it cannot be used to forecast the shift, which occurs much later.
3. It decreases well ahead of the shift, so it can be used to forecast the shift.
4. It increases well ahead of the shift, so it can be used to forecast the shift.

Diffusion of biomass

Thus far, we neglected spatial interactions in the model. To represent clonal growth and other dispersal mechanisms of biomass, the model presented in the previous section includes dispersion D. It is calculated as:

$$D_{x,y} = B_{x,y-1} + B_{x+1,y} + B_{x+1,y} + B_{x-1,y} - n_{x,y}B_{x,y}$$

With,

$D_{x,y}$, diffusion term at a cell located at (x,y),
$B_{x,y-1}$, biomass at a direct neighbour in the negative y direction,
$n_{x,y}$, the number of direct neighbouring cells.

For all cells except cells at the edge of the map, n is equal to four neighbours, however at the edge cells one neighbour is missing, so we need to adjust n here. In PCRaster Python, you can represent the equation by typing in the dynamic:

```
diffusion = self.d*(window4total(self.x)-self.numberOfNeighbours*self.x)
```

The `window4total` function sums the cell values of the four direct (top, right, bottom, left) neighbouring cells, `self.d` is the diffusion or dispersion rate (0.01). In the initial you need to calculate:

```
self.numberOfNeighbours = window4total(spatial(scalar(1)))
```

You also need to assign 0.01 to `self.d` in the initial. Add these statements to the model, such that diffusion $D_{x,y}$ is added to the biomass for each time step. Run the model and animate the biomass maps, using timeseries plots to evaluate the results.

Question: As you can see, the diffusion process results in patches of lower and higher biomass. What happens with the patch size over time (e.g. compare the pattern at the start of the model run, with the pattern just before the shift towards a lower biomass)?

1. The patch size increases over time.
2. The patches change, but the size stays the same.
3. The patch size decreases over time.

2.8 Probabilistic spatial models: forest fire and seed dispersal

2.8.1 Random variables in point models

Continuous random variables

PCRaster comes with a number of functions to draw realizations from continuous probability distributions.

Go to the folder /probab/fire in your exercises data set and open `randomVar.py` To the dynamic, add a line that draws a realization using the function `normal` Store the result. In a similar way, use `uniform` and store the result. Animate the outputs.

> **Question** What is done by these functions?

Now, test the function `mapuniform`

> **Question** What is done by `mapuniform` In what situations would you use `uniform` and in what situations would you use `mapuniform`

Now, add a statement that draws realizations from a Gaussian distributed variable with mean a mean of 10 and standard deviation of 5.

> **Question** What did you add to the script?

Discrete random variables

Functions to draw realizations of discrete (classified) random variables are not provided. These can be derived from the functions for continuous random variables that you used in the previous section.

Open `randomVar.py` Add (a) statement(s) to the script that draws realizations (for each cell independently and for each timestep) of a discrete random variable with two possible outcomes, True and False:

$P(\text{True}) = 0.8$
$P(\text{False}) = 0.2$

Write results to disk, run the script, and display to check the result.

> **Question** What did you add to the script?

Now, add statements that draws realizations of a discrete random variable with three possible outcomes, coded as 1, 2, and 3:

$P(1) = 0.01$
$P(2) = 0.9$
$P(3) = 0.09$

Suggestion: use lookupnominal. Write to disk to check the result.

Question What did you add to the script?

In quite some situations, it is required to randomly select a number of cells and mark these as True. This can easily be done. Let's select for each time step 10 random cell locations. First, use `order` on the map that was created with the `uniform` operation. Write the result to disk and animate. Now, add a statement (with the result of `order` as input) to the script that selects exactly 10 cells, at random locations. Write to disk, save the script and run.

Question Give an example of a spatial process that could be modelled in this way.

2.8.2 Direct neighbourhood interaction: forest fire

Fire spreading

Direct neighbourhood functions (as used in cellular automata) are suitable for modelling forest fire. This is because the large scale spatio-temporal pattern of an evolving fire is driven by local interactions at the fire front. These local interactions will be represented here using a stochastic direct neighbourhood model. The model uses the following transition rules that are applied for each time step:

1. A cell that was burning at the previous time step stays burning.
2. Cells that are not burning at the previous time step potentially catch fire only when one or more neighbouring cells are burning at the previous time step. Neighbouring cells are the four direct neighbours in the x, y direction.
3. The cells under item 2 actually catch fire with a probability of 0.1.
4. A cell that was not burning at the previous time step and actually catched fire in the current timestep becomes a burning cell.

Display `dem.map` and `start.map`. The `start.map` contains True cells representing the starting area of a fire. Open `fire.py`. First add statements to represent rule 2.

- Set the initial distribution of the fire (use the variable name `fire`) to `start.map`.
- In the dynamic, add a `window4total` statement that calculates the number of neighbours that are burning.
- In the dynamic, use the result of `window4total` to calculate a map `neighbourBurns` that is True when one or more neighbouring cells are burning.
- Write the resulting maps to disk, save the script, and run.

Question Animate `neighbourBurns`. Do the cells with True also include cells that were already burning in the previous time step?

And finally for rule 2, add (a) statement(s) that calculate(s) `potentialNewFire`, a map that is True *only* for cells that potentially catch fire. To calculate this map, you need to combine `fire` and `neighbourBurns` using Boolean logic operations. Write to disk and check the result.

Now, let's add rule 3.

- Add the following statement to the dynamic calculating for each time step a map that is True with a probability of 0.1 and false elsewhere. Write the result to disk to check it.

```
realization = uniform(1) < 0.1
```

- Add a statement that calculates `newFire`, a map that is True for cells that actually catch fire in the current time step. `newFire` represents rule 3. Write to disk, run, and check the result.

And finally, update `fire` by combining `newFire` and `fire` from the previous time step. Write to disk, store your model script, run it, and check the results.

Question How does the fire evolve over time?

Fire spreading determined by the topography

In the previous exercise, we assumed a probability of a cell catching fire to be everywhere the same, i.e. 0.1. In reality, however, a fire front mainly extends uphill. A very simple representation of this process is used here. The transition rules are kept the same, except for the probability used in rule 2. Now we will calculate this probability as:

- All cells with a direct neighbour in downhill direction have a probability to catch fire of 0.8.
- For all other cells, this probability is 0.1.

Open the script `fire.py` that was created in the previous exercise and save it as `fireUphill.py`. In the initial, create the map `ldd` using the digital elevation model `dem.map`. You need `self.ldd` as variable name as it will be used in the dynamic. Display the local drain direction map on top of the digital elevation model, zoom in if you get an almost black map! Have a look in the Aguila manual if needed, to see how to display maps on top of each other, see http://pcraster.sourceforge.net/Aguila/1.1.0/Manual.html.

In the dynamic, you need to add statements calculating for each map the probability for cells to catch fire. As a start, add two statements (insert them directly above the calculation of `realization`:

- A statement calculating `downhillBurning`, a map that is True for cells with a cell in downhill direction that is burning. Use the function `downstream`.
- A statement calculating `probability`, a map with the probabilities that a cell catches fire. Use `downhillBurning` as input.

Let the model write the results of these two calculations to disk. Save and run the model to see the results.

> **Question** What did you add to the script?

Now, you need to change the calculation of `realization` as it needs to take into account the spatial pattern (for each time step) of the `probability`. Do this by changing the statement. Save and run the model. Animate the resulting fire pattern together with the local drain direction map and the digital elevation model.

> **Question** What is the pattern of the fire spreading over time? Do you think this is more realistic compared to the previous model? Give two suggestions to further improve the model.

2.8.3 Neighbourhood interaction over a distance: plant seed dispersal

Plant dispersal occurs either as clonal growth or through seed dispersal. Clonal growth can be modelled using direct neighbourhood operations as new plants only occur directly neighbouring existing cells. Thus, it is actually a quite similar process, from the modelling point of view, as forest fire. Seed dispersal, however, allows plants to disperse over longer distances: new plants may establish also at cells that are not neighbours of cells containing plants in the previous time step. Here we will construct a simple seed dispersal model.

We use the following set of rules:

1. A single plant species disperses over an area.
2. For each time step, the probability P(E) that new plants establish at cells that are not yet occupied by the species is modelled as:

$P(E) = 0.1^{(d/r)}$

with, d, the distance away (m) from the nearest cell containing the plant, and r, a parameter (m, r = 40 m).

3. For each time step, the new distribution of the plants consists of the distribution of the plants in the previous time step combined with the distribution of the plants that established in the current time step.

Go to the directory `probab/plants` and display `veg.map`. We assume that the plant species to be modelled is artificially introduced (before dispersal starts) at all pixels containing dense herbs (class with code 5). Open `plants.py`.

In the initial, create a Boolean map (use the variable name `self.plants`) that is True at dense herbs and false elsewhere. This is the map containing the plants that will disperse over time.

To represent the transition rules, add statements to the dynamic creating a map `probability` with *P*(E) for each time step. For now, you can assume that the plant does not yet spread (i.e., `self.plants` does not change), so `probability` will not change over time. Store the script, and run. Display the results.

Question What is the probability in the bottom left corner of the study area?

Using `probability`, create a new map `newPlants` that is True at cells where the plant establishes and false elsewhere. You need to use `probability` as input, and the `uniform` function. Write the result to disk and check the outcome.

Now, update `self.plants` by 'adding' the newly established plants to `self.plants`. Write `self.plants` to disk in the dynamic. Save the script, run, and animate the plants dispersal, together with the probability maps.

Question Run the model twice and compare the results.

Question How long does it approximately take to reach the Open Shrub patch on the bottom left side of the area?

We can make all kinds of interesting extensions to the model. Let's see what happens if we assume the plant can only grow at vegetation units Open Shrub (code 3) and Dense herbs (code 5). In the initial, create a map `biotope` that is True at cells containing Open shrub or Dense herbs. In the dynamic, modify the script such that the plant will only exist or establish at these open shrub or dens herbs cells. Save the script, run.

Question What is the difference in the dispersal compared to the earlier runs? How long does it approximately take to reach the Open shrub patch on the left side of the area? Explain.

2.9 Beyond the Model class

Thus far we only used predefined Model class for control flow in the script. Of course you can use all Python constructs inside and outside this Model class. You may need this either to better structure your model, or to calculate things that cannot be done with using the time iteration provided by the Model class.

For instance, it is often wise to rewrite blocks of code as functions. Go to the subdirectory morePython and open `functions.py`. It is the fire model you programmed earlier. At the top of the dynamic are a number of lines finally calculating `potentialNewFire`. To make this script better readable, rewrite the program such that this upper block is encapsulated in a function `calculatePotentialNewFire`. Define this function at the top of the program (just below `from pcraster.framework import *` and call it in the dynamic. Be sure to test the program!

Question Give another example of how Python constructs can be used in modelling.

- Functions
- Conditional execution

CHAPTER

THREE

STOCHASTIC MODELLING

Download this website as pdf.

Download this website as epub (for e-readers).

To request the data set for the exercises email d.karssenberg@geo.uu.nl

3.1 Visualisation of stochastic data

3.1.1 Realizations

PCRaster includes the Aguila software for visualisation of spatio-temporal stochastic data. With Aguila, you can easily explore large multi-dimensional data sets. For the exercises below, you will need to look up some options in the Aguila manual, available at http://pcraster.sourceforge.net/Aguila/1.1.0/Manual.html.

To learn how to visualize stochastic data you will use hydrological model outputs from a model of a catchment in Spain.

Aguila is started from a command shell. Open a command shell and go to the directory containing your data for the exercises. Type cd to change directories. In the directory, you will find the folder `visualisation` containing the data for the visualisation practical. In this directory, display the digital elevation model, the vegetation map, and the local drain direction map:

```
aguila dem.map veg.map ldd.map <Enter>
```

Use the menu items (or right click on the legend to get more options) to zoom in/out, to change the color palette, and to change the drawing mode to contours. Also try changing the number of classes shown (or number of contours).

To represent saturated conductivity as a stochastic variable, realizations were drawn from probability distributions. These were used in the stochastic hydrological model. In PCRaster Python, realizations are stored in subdirectories numbered 1, 2,..,N, with N the number of realizations. Use the command `dir` (or `ls`) on linux to check the number of realizations available in the data set.

Also, go to the folder `1` and type `dir` to have a look at its contents. You will see that it contants a `ksat.map` which is a realization of saturated conductivity (m/h).

Go back to the main folder `visualisation`.

You can display realizations of static data (without time component) by typing:

```
aguila --scenarios='{1,2,3,4,5,6,7,8,9,10,11,12}' ksat
```

This opens the realizations in separate windows. It is more convenient to display in one window, by typing:

```
aguila --scenarios='{1,2,3,4,5,6,7,8,9,10,11,12}' --multi=3x4 ksat
```

You can combine the visualisation with the vegetation map:

```
aguila veg.map --scenarios='{1,2,3,4,5,6,7,8,9,10,11,12}' --multi=3x4 ksat
```

Question: What is the value of saturated conductivity at the 25th, 50th and 75th percentile, respectively, in the unit dense herbs? Make a rough estimate. Use the realizations at a single cell location.

1. 25th percentile: 0.064; 50th percentile: 0.061; 75th percentile: 0.059.
2. 25th percentile: 0.059; 50th percentile: 0.061; 75th percentile: 0.064.
3. 25th percentile: 0.059; 50th percentile: 0.059; 75th percentile: 0.059.
4. 25th percentile: 0.061; 50th percentile: 0.059; 75th percentile: 0.064.

Now, lets have a look at temporal data. Display the timeseries of precipitation:

```
aguila precip.tss
```

It contains precipitation (mm/h) for 10 hours. It was used as input to a stochastic model calculating actual infiltration for each time step, using the realizations of infiltration capacity.

The folders 1 to 12 contain realizations of actual infiltration (mm/h), numbered from 1 to 10 (`i0000000.001`, `i0000000.002`, etc), ie for each time step of an hour one map. Check this.

To animate these realizations, type in the `visualisation` folder:

```
aguila veg.map --scenarios='{1,2,3,4,5,6,7,8,9,10,11,12}' --multi=3x4  --timesteps=[1,10,1] i
```

Plot a timeseries by right clicking on the legend of the infiltration maps `i`.

Question: By clicking on cells in the 'Dense herbs' area, have a look at the timeseries for this region. When is the uncertainty largest, with large amounts of rain or with low amounts of rain, or does rain not affect the uncertainty? Can you explain this?

1. The uncertainty is largest for low amounts of rainfall. This is because for these cases the variation in rainfall is largest.
2. The uncertainty is largest for high amounts of rainfall. This is because for these cases the variation in rainfall is largest.
3. The uncertainty does not vary with rainfall amount.
4. The uncertainty is largest during peak rainfall. With large amounts of rain, the differences among realizations of infiltration become clear.

```
aguila --scenarios='{1,2,3,4,5,6,7,8,9,10,11,12}' --multi=3x4  --timesteps=[1,10,1] q + ldd.map
```

The maps `q` contain discharge (m3/h). Zoom in to see the content of the map. Click on the main channel and plot a timeseries. Animate.

3.1.2 Probability distributions and exceedance probabilities

Probability distributions are stored as a set of percentile maps. Unlike realizations, these are stored in the main folder `visualisation`. Type:

```
dir ksat_*map
```

Or in linux you would use `ls`. Note that `*` is a wildcard, representing all possible characters. These are maps containing the percentile values. The percentiles are given as fractional values between zero and 1 in the file name. You can display these maps just like other maps, e.g.:

```
aguila ksat_0.5.map
```

However, it is also possible to display them at once, resulting in a plot of the probability density distribution:

```
aguila --quantiles=[0.025,0.975,0.005] ksat veg.map
```

This opens all ksat percentile maps (and the vegetation map), and displays the probability distribution between the 2.5% and 97.5% percentiles, with a step of 0.5%. Note that, as the information is only available at a limited number of percentile values, aguila interpolates. You can open the probability density view by right clicking on the legend. Read through the Aguila manual at http://pcraster.sourceforge.net/Aguila/1.1.0/Views.html#probability-graph-view for an explanation.

Question: Give the boundaries of the 95% confidence interval (i.e., between 2.5 and 97.5% percentiles) of saturated conductivity in the dense herbs area.

1. 2.5%: 0.0345 m/h; 97.5%: 0.0694 m/h
2. 2.5%: 0.0485 m/h; 97.5%: 0.0485 m/h
3. 2.5%: 0.0485 m/h; 97.5%: 0.0694 m/h
4. 2.5%: 0.0694 m/h; 97.5%: 0.0485 m/h

You can retreive cumulative probabilities by clicking on the + in the menu at the top of the probability density distribution view. After this, you can slide the vertical line in the probability density view, to get cumulative probabilities for different threshold values. If exceedance intervals are needed, right-click on the legend, select Edit draw properties, and select 'Exceedance probabilities'.

Question: Give the probability that saturated conductivity exceeds 0.056 in the dense herbs area.

1. The probability is 0.71
2. The probability is 0.29
3. The probability is 0.0071
4. The probability is 0.0029

Probability density distributions can also be displayed for dynamic data.

```
dir i_*
```

These are the percentile maps of modelled actual infiltration, for each time step of an hour. Filenames refer to time steps and the percentile. To display and animate these, type:

```
aguila --quantiles=[0.025,0.975,0.005] --timesteps=[1,10,1] i veg.map
```

By right-clicking on the legend of `i`, open timeseries and the probality distribution plot. Animate. Click on different locations of the map to evaluate the uncertainty in infiltration.

In a similar way, create an animation of the probability distribution of the discharge, `q`.

Question: Change the map view such that it shows the cumulative probability for a threshold value of 20000 m3/h. Note that the probability that `q` exceeds the threshold is one minus the cumulative probability. Animate the cumulative probability maps. Describe the pattern in the exceedance probabilities (in space and time).

1. The higher the discharge, the lower the exceedance probability.
2. The higher the discharge, the higher the exceedance probability.

3.1.3 Confidence intervals

To plot confidence intervals, type:

```
aguila --quantiles=[0.025,0.975,0.005] ksat veg.map
```

Open the probability density plot. Click on the '+' in the probability density plot window to switch to cumulative probabilities. Now, right click on the legend, select Edit draw properties, and in the colour assignment panel, select 'Confidence interval'. If you want you can change the alpha level of the confidence interval in the field below. Click 'OK'. Now, move the threshold value in the probability density plot (the vertical line), and Aguila interactively shows the confidence interval for that threshold value.

Question: Which vegetation units are certainly (i.e. with an alpha value of 0.05) above a threshold value of 0.01 ?

1. All units.
2. Pine forest, dense herbs, and open shrubs.
3. Open shrubs.
4. All units except open shrubs.

3.2 The stochastic modelling framework

3.2.1 The static stochastic modelling class

Go to the folder `framework` in your folder containing the exercise data. For static stochastic modelling, you can use the template script `stochStaticMod.py`. Open the script, and run it. Type `dir` to see the contents of the folder.

Question: The model class contains a number of methods. In which section (method) do we need to put calculations representing processes (or importing data) that are stochastic, without time? Idem, purely deterministic calculations without uncertainty?

1. Stochastic without time: premcloop. Deterministic: initial.

2. Stochastic without time: postmcloop. Deterministic: initial.
3. Stochastic without time: initial. Deterministic: initial.
4. Stochastic without time: initial. Deterministic: premcloop.

3.2.2 The dynamic stochastic modelling class

In the same `framework` folder, open and run `stochDynamicMod.py`. It is a template script for dynamic stochastic modelling.

Question: What is the dynamic method doing in the `stochStaticMod.py` script?

1. It is run for each Monte Carlo realization, and each time step.
2. It is run for the first Monte Carlo realization, for each time step.
3. It is run for each Monte Carlo realization, without time steps.
4. It passes information from the initial to the postmcloop.

3.3 Writing to disk, drawing realizations

3.3.1 Writing realizations to disk

Just like in deterministic modelling, `report` can be used to write data to disk.

In the `framework` folder, Open `stochDynamic.py` and save it as `real.py`. Add the following lines to the premcloop:

```
a = normal(1)
self.report(a,'a')
```

Add the same two statements to the initial and dynamic, using variable names `b` and `c`, respectively (instead of `a`). Save the script and run. Use `dir` to check the contents of the main folder `framework` and the folders containing the realizations, e.g. `1`. Use aguila to display `a`, `b`, and `c`.

Question: What is stored by reporting in the dynamic?

1. It stores files in subdirectories referring to the Monte Carlo realizations, where each file has a number referring to the time step. Each file is a time series.
2. It stores files in subdirectories referring to the time steps, where each file has a number referring to the Monte Carlo realization. Each file is a time series.
3. It stores files in subdirectories referring to the Monte Carlo realizations, where each file has a number referring to the time step. Each file is a map.
4. It stores files in subdirectories referring to the time steps, where each file has a number referring to the Monte Carlo realization. Each file is a map.

Opening files from disk can be done with `self.readmap`, just like in dynamic modelling.

3.3.2 Drawing realizations

The function `normal` draws realizations from a stochastic variable. PCRaster includes a number of other functions that draw realizations from stochastic variables, e.g. `mapnormal`, `uniform`. These were explained in the dynamic modelling exercises. Here, you will learn to use `areanormal`.

Open the map `regions.map`. Boolean True cells are ski regions. For the ski regions, vegetation height is a stochastic variable with a mean 0.2 and a standard deviation of 0.05 (m). For the remaining area, the mean is 0.3 with a standard deviation 0.1.

Open `real.py` and add statements (in the initial) to create realizations of vegetation height using `areanormal`. Use the variable name `vegetationHeight` and write to disk as `v`. Save the script and run. Use aguila to display 10 realizations of `v`. Hints:

- Read `regions.map` from disk in the premcloop. It is a deterministic map thus needs to be opened there.
- Use `ifthenelse` to create factors for mean and standard deviation.

Question: What is done by the `areanormal` function?

1. It draws a random value for each cell independently.
2. It draws a random value for each area independently, these are different between Monte Carlo realizations.
3. It normalizes input values within the boundaries of a normal distribution.
4. It draws a random value for each area independently, these are the same over the Monte Carlo realizations.

3.4 Functions calculating statistics over realizations

3.4.1 Mean and variance, static data

In the folder `framework`, open `real.py`. It is the script created in the previous exercise drawing realizations from stochastic variables. PCRaster includes functions to calculate statistics over these realizations. This is done in the postmcloop. These functions read the data written to disk in the initial or the dynamic, and calculate the statistics over these data, writing the results to disk.

The function `mcaveragevariance` calculates the mean and variance of realizations, on a cell-by-cell basis.

Save `real.py` under the new name `statistics.py`. Add the following lines to the postmcloop:

```
names=['b']
sampleNumbers=self.sampleNumbers()
print sampleNumbers
timeSteps=[0]
mcaveragevariance(names,sampleNumbers,timeSteps)
```

The `mcaveragevariance` function requires three inputs:

- `names` is a list of filenames (given as strings) of the map variables for which the statistics are needed. As these are read from disk, you need to have stored these in the initial or the dynamic using report. Here we use a list of length 1, containing one variable.
- `sampleNumbers` is a list of Monte Carlo samples over which statistics are calculated. Here we create it with the python function `range`. It is also printed to check it.

- `timesteps` is a list of time steps for which the statistics are created. As `b` represents a static variable, we provide a list containing one zero element. This tells `mcaveragevariance` that it gets static data.

After adding the lines, change the number of Monte Carlo samples to use to 100, for more precision. Save the script and run. Type `dir` in the main folder `framework`. The `mcaveragevariance` function has created the maps `b-ave.map` and `b-var.map`, containing the mean and variance. Display these maps.

Now, change `names` to calculate the statistics of vegetation height also. Save and run the script. Type `dir` in the main folder `framework` to see what it has stored. Display the mean and variance maps of vegetation height.

Question: How well does your Monte Carlo script calculate the original statistics of the stochastic variables ?

1. For the variable `b`, the variance is approximately between 50% and 150% of the provided value.
2. Idem, between 10% and 190%.
3. Idem, between 1% and 1000%.
4. Idem, between 99% and 101%.

3.4.2 Percentiles, static data

The calculation of percentile data is almost just as easy.

Open `statistics.py` and add the following two lines to the postmcloop, below the other statements:

```
percentiles=[0.025,0.05,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,0.95,0.975]
mcpercentiles(names,percentiles,sampleNumbers,timeSteps)
```

Save the script and run. The functions `percentiles` works similar to `mcaveragevariance`, with the exception that it needs a list providing the percentiles that need to be calculated. In the `framework` folder, type `dir` to see what the script has stored.

Create a probability density distribution plot of the vegetation height and the variable `b`. Click on the map to retreive the plot for different cell locations.

Question: What is the 25% percentile value of `b`? What is the probability that `b` exceeds 1.0?

1. The 25% percentile value is about -0.05, the exceedance probability for 1.0 is about 0.2 (values depend on cell location).
2. The 25% percentile value is about -0.5, the exceedance probability for 1.0 is about 0.2 (values depend on cell location).
3. The 25% percentile value is about -0.01, the exceedance probability for 1.0 is about 0.2 (values depend on cell location).
4. The 25% percentile value is about -0.005, the exceedance probability for 1.0 is about 2.0 (values depend on cell location).

3.4.3 Dynamic data

A very similar approach can be used when calculating statistics over dynamic data.

Open `statistics.py`. Let's first create a more interesting dynamic variable. At the top of the dynamic, type:

```
c = normal(1)
self.report(c,'c')
```

At the bottom of the dynamic, type:

```
self.d=self.d+c+1
self.report(self.d,'d')
```

Initialize `self.d` in the initial by adding:

```
self.d=scalar(0)
```

Also, increase the number of timesteps to 10. Save `statistics.py` and run the script. Type `dir` in the folder `1` to see what is stored.

In the main folder `framework` animate a visualisation of 12 realizations of `c` and `d`. Use the Aguila options `--scenarios`, `--timesteps` and `--multi`.

Question: What happens with the uncertainty in `c` and `d` over time?

1. The uncertainty in `c` and `d` increases with time.
2. The uncertainty in `c` and `d` decreases with time.
3. The uncertainty in `c` is time-independent, while the uncertainty in `d` increases with time.
4. The uncertainty in `c` increases with time, while the uncertainty in `d` is time-independent.

To calculate the sample statistics, add the following block of code at the bottom of the postmcloop:

```
names=['c','d']
sampleNumbers=range(1,self.nrSamples()+1)
timeSteps=self.timeSteps()
print timeSteps
mcaveragevariance(names,sampleNumbers,timeSteps)
mcpercentiles(names,percentiles,sampleNumbers,timeSteps)
```

Note that this is very similar to the code needed for static data. We only need to change the `names` and the `timeSteps`. Note that `self.timeSteps()` generates a list of the timesteps in the model. It is printed here.

Save the script and run. It will take some time to run as it needs to calculate statistics for each time step.

Type `dir c*` to see what is stored by the script. Also try `dir d*`.

Create probability distribution plots of `c` and `d`. Also, plot time series and animate.

Question: Set the map view of `d` to retrieve the exceedance probability of a threshold value of 5. What happens with this probability over time?

1. It increases over time up to a value of about 0.01 at the last time step.
2. It increases over time up to a value of about 0.1 at the last time step.
3. It increases over time up to a value of about 1 at the last time step.

4. It increases over time up to a value of about 0.5 at the last time step.

3.5 Static modelling with point operations: forest fire

3.5.1 Introduction and gradient calculation

In a static model, we can calculate the time that the front of a forest fire reaches a certain cell location using weighted (relative) distance calculation, assuming the fire spreading is restricted to neighbouring cells in uphill direction only. This approach, further explained in the next section, requires a map with cell values representing the velocity of the forest fire spreading at that cell, given in hours per metre distance. The value of this velocity (h/m) is calculated as:

$h = be^{s/a}$

with, b, a parameter, s, the gradient of the topographical surface (m/m), a a parameter, and h velocity of forest fire spreading at the cell. The value of b is known, $b = 0.002$, and it is assumed here that the gradient can be calculated from the digitial elevation model (assumed to have zero uncertainty). The value of a is known with a certain uncertainty, and thus a is modelled as a stochastic parameter (constant over the whole area) with a probability distribution, defined by a mean of 0.5 and a variance of 0.01.

Go to the folder `fire` and display all maps in the folder, including `dem.map`, the digital elevation model that can be used to calculate the gradient. Open the script `stochStaticMod.py`. Add statements to calculate a variable `gradient` containing the slope of the topographical surface, write to disk under the file name `gradient`. Save and run the script, and display `gradient`.

Question: Where did you type the statements?

1. In the premcloop.
2. In the initial.
3. In the postmcloop.
4. At the top of the script, below the from.. statements.

3.5.2 Point operation to calculate fire front velocity

In `stochStaticMod.py`, add a statement that calculates realizations of a. Use the variable name `a` written to disk with a file name `a`. You will need to use the function `mapnormal`. Save the script and run. Check the output.

Calculate the mean, variance, and percentiles of `a` in the postmcloop. Save the script and run. Display the output. Be sure to check whether the mean and variance of `a` are correct.

Calculate h, save the variable as `hpm` (i.e., hours per minute) and calculate mean, variance, and percentiles. Display the output.

Question: What is the probability density distribution of the velocity of the forest fire, i.e. what is the shape? Explain the shape using the equation used to calculate it.

1. The probability density distribution has a zero standard deviation, due to the exponent, reducing the variation.

2. The shape of the probability density distribution is normal, because the equation uses a normally distributed variable in the exponent of a natural logarithm.
3. The shape of the probability density distribution is log-normal, because the equation uses a normally distributed variable in the exponent of a natural logarithm.
4. The shape of the probability density distribution is uniform, which is due to the exponent.

Question: What is the variance and the standard deviation of the velocity of forest fire (h/m) at the gridcell location indicated on `house.map`? Write down and store, you will need it later on.

1. average = 0.042, variance = 0.43·10-9, standard deviation = 2·10-10
2. average = 0.42, variance = 4.3·10-9, standard deviation = 2·10-9
3. average = 0.0042, variance = 9.3·10-9, standard deviation = 1·10-9
4. average = 0.0042, variance = 4.3·10-9, standard deviation = 2·10-9

Question: Display `hpm` together with the slope map `gradient`. Plot the 95% confidence interval for a (threshold) value of 0.005. A velocity below 0.005 is considered a high forest fire front spreading velocity. For what areas in terms of slope values is it not distinguishable whether the value is above or below the threshold?

1. For areas with a slope of about 0.1.
2. For areas with a slope of about 0.2.
3. For areas with a slope of about 0.5.
4. For areas with a slope of about 0.7.

3.6 Static stochastic modelling: neighbourhood operations

3.6.1 Drawing realizations of a stochastic digital elevation model

In the previous exercise, you used the following equation to calculate the velocity of forest fire spreading at a given cell, given in hours per metre. It was calculated as:

$h = be^{s/a}$

with, b, a parameter, s, the gradient of the topographical surface (m/m), a a parameter, and h velocity of forest fire spreading at the cell.

We assumed only a was unknown. However, as the digital elevation model will include error, we need to propagate the error in the digital elevation model to the output of the model h. Let's assume the error in the elevation (m) is for each pixel modelled by a stochastic variable D:

$D = e + Z$

with, Z a stochastic variable with zero mean and a variance of 4.0.

In the folder `fire`, open `stochStaticMod.py` and save as `slope.py`. First, calculate realizations of D in the initial. In the premcloop, you still need to read the deterministic dem from disk. However, you need to add `self.` as the map is needed in the initial. So, the premcloop should look like this:

```
self.dem=self.readmap('dem')
self.gradient=slope(self.dem)
self.report(self.gradient,'gradient')
```

In the initial, you need to calculate the realizations by adding realizations of Z to the deterministic dem. As we need realizations on a cell-by-cell basis, you need to use `normal`. For D, use a variable name `demStoch` and write to disk as `dS`. Save the script and run.

Use Aguila to display 12 realizations of `dS`. Are the results approximately correct?

Now, calculate mean, variance, and percentiles of `dS`. Save the script and run. Display the variance map and the probability distribution plots.

Question: What is calculated by the script for the variance of `dS`? Is it similar to the value of 4.0 provided.

1. On average it is about 4, but it ranges between 3 and 5 depending on the cell location.
2. On average it is about 4, but it ranges between 1 and 7 depending on the cell location.
3. On average it is about 4, but it ranges between 3.9 and 4.1 depending on the cell location.
4. On average it is about 4, but it ranges between 3.99 and 4.01 depending on the cell location.

3.6.2 Propagate DEM uncertainty through slope calculation

Using the stochastic digital elevation model defined in the previous section, you can now calculate the effect of this uncertainty on the uncertainty in the slope calculation. Thus far, slope was calculated in the premcloop. You need to move this calculation to the initial, using the realizations of the digital elevation model as input.

Open `slope.py` and make the required modifications. Be sure to calculate mean, variance, and percentiles of the slope. Save the script and run. Display the slope results to check your calculations.

Question: The uncertainty in fire spreading velocity of includes now the uncertainty of the parameter *a and* the uncertainy of the digitial elevation model. Display `house.map` and the variance map of the forest fire spreading velocity (h/m). What is the variance and standard deviation of the forest fire spreading velocity (m/h) at the location of the house? Compare the result with the run where we only included error in the *a* parameter (previous exercise). What is the contribution to uncertainty in forest fire spreading of the uncertainty of the digitial elevation model, approximately?

1. The uncertainty becomes zero.
2. The uncertainty has not changed (changes are due to numerical error).
3. The uncertainty is much lower now.
4. The uncertainty is much higher now.

3.6.3 Neighbourhood defined by topology: relative distance calculation

With the fire spreading velocity (h/m) you can calculate a map with the time it takes until the fire reaches a certain location, given a specified starting location of the fire. We oversimplify here somewhat, by assuming that fire burns only in uphill direction. The `lddist` function can be used to calculate a relative distance, i.e. the absolute distance (m) that is travelled through a cell multiplied by a relative 'weighting' for that cell, here the fire spreading velocity (h/m). The result is the time (h) for the fire to reach a location. It restricts the distances calculated to uphill local drain directions.

Have a look at `lddist` in the PCRaster manual. Have a look at `start.map`, it is the starting point of a fire.

Open `slope.py` and save as `fire.py`. Make the following additions or changes:

- Read `start.map` from disk in the premcloop.
- Calculate the local drain direction map in premcloop and write to disk.
- In initial, use `lddist` to calculate a map with the time the fire takes to reach a cell. As inputs, you need the local drain direction map, and the starting point of the fire, and the fire velocity map in hours per metre. Name the resulting map `time` and write to disk with report as `time`.
- In the postmcloop, calculate sampling statistics (mean, variance, percentiles) of fire.

Save the script and run. Display the mean, variance, and the probability density function of `time`, together with the local drain direction map and `house.map`.

Question: What is the mean and standard deviation of the time it takes until the fire reaches the house?

1. Mean: 2.5 h. Standard deviation: 0.037 h.
2. Mean: 2.5 h. Standard deviation: 0.37 h.
3. Mean: 2.5 h. Standard deviation: 3.7 h.
4. Mean: 4.5 h. Standard deviation: 0.37 h.

Question: Assume that it takes two hours (after the initiation of the fire) until the owners of the house can be informed they should leave the house. Create the confidence interval plot for a threshold of two hours, using an alpha value of 0.1. Can we be certain with this model that the owners can be informed in time?

1. Yes. The alpha value is below 2 hours, so we do even not need to run the model.
2. Well, it is unclear. The fire might be there within two hours, but it can also take longer.
3. No, according to the model we can be certain that the fire takes less than two hours to reach the house. The owners cannot be informed in time.
4. Yes, according to the model we can be certain that the fire takes more than two hours to reach the house. The owners can be informed in time.

3.7 Dynamic stochastic modelling: snow melt model

3.7.1 Snow melt as a stochastic model

In the dynamic modelling exercises you created a deterministic snow melt model. Here we will built upon this work by propagating input errors to its outputs. As a start, the data set contains a script that is exactly the same as the script created in the deterministic modelling exercises, apart from that the script has been modified to be able to do stochastic modelling. Also, we will use smaller input maps to speed up calculations.

Go to the folder `snow` in the folder containing your exercise data, and open the script `snow.py`. Be sure to remind what calculations were included (if needed, have a look at the deterministic modelling exercises). It uses the stochastic dynamic modelling framework, and thus it will run for 50 Monte Carlo loops (as you can see at the bottom). However in the `snow.py`, no stochastic inputs are used (yet).

Run the script. Display the input timeseries, two realizations of snow cover and discharge, and check whether they are the same. Also display `dem.map` and you will see that we are using a smaller area now.

3.7.2 Stochastic dynamic input: precipitation

To represent uncertainty in precipitation, we define the following error model:

$p(t) = p_m(t)$ x $e(t)$

With $p_m(t)$ the observed precipitation, $p(t)$ the stochastic variable precipitation (including uncertainty), and $e(t)$ a random non-spatial (i.e., the same for all cells) variable for each time step t with mean one and variance 0.04.

Open `snow.py` and save as `precip.py`. Modify `precip.py` to represent rainfall as a stochastic variable. You need to use `mapnormal`. And note that you will need to convert variance to standard deviation to get the realizations of $p(t)$.

Save the script and run the model. Display 12 realizations of precipitation and snow cover. You need the `scenarios`, `multi`, and `timesteps` Aguila options. Animate them, and create timeseries for different locations.

Question: Do you think our error model results in a large error in the total amount of rain over the whole modelling period? Explain.

1. No, the errors are assigned each time step thus resulting in a large error (additive error).
2. Yes, the errors are assigned each time step thus resulting in a large error (additive error).
3. No, the errors cancel each other out as they are assigned each time step independently.
4. Yes, the errors cancel each other out as they are assigned each time step independently.

To be able to create plots of probability density distributions, calculate mean, variance, and percentiles in the postmcloop. Do this for precipitation, snow depth (`snow`), and discharge (`q`). To reduce the calculation time, use a limited set of percentiles:

```
percentiles=[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9]
```

Save the `precip.py` script and run. Note that it will take some time at the end to calculate the percentiles. If the script is really too slow, you can consider using less Monte Carlo samples (e.g. 25 instead of 50), but note that this will result in a less precise calculation of the errors.

Use aguila to display and animate probability density distributions of precipitation, snow depth, and discharge, together with the map `locations.map`. Also, plot the variance of these attributes using Aguila, together with `locations.map`.

Question: What is the variance in snow depth at the two locations on `locations.map`, for time step 120 and 170 days? Write it down too.

1. Day 120: Location 1, 6.16, Location 2, 1.55; Day 170: Location 1, 7.90; Location 2, 1.12
2. Day 120: Location 1, 5.12, Location 2, 9.28; Day 170: Location 1, 1.75; Location 2, 2.28
3. Day 120: Location 1, 4.99, Location 2, 1.41; Day 170: Location 1, 6.44; Location 2, 1.35
4. Day 120: Location 1, 5.16, Location 2, 4.41; Day 170: Location 1, 7.75; Location 2, 6.28

Question: What is the variance in discharge at the two locations on `locations.map`, for time step 120 and 170 days? Write it down too.

1. Day 120: Location 1, 0.01, Location 2, 0.00; Day 170: Location 1, 1.75 * 10^10; Location 2, 3.87 * 10^13
2. Day 120: Location 1, 0.04, Location 2, 0.00; Day 170: Location 1, 2.75 * 10^10; Location 2, 1.61 * 10^13
3. Day 120: Location 1, 0.09, Location 2, 0.00; Day 170: Location 1, 3.75 * 10^10; Location 2, 9.61 * 10^13
4. Day 120: Location 1, 0.00, Location 2, 0.00; Day 170: Location 1, 3.75 * 10^10; Location 2, 3.61 * 10^13

3.7.3 Stochastic parameters: temperature lapse rate

To represent uncertainty in the temperature lapse rate parameter, we replace the deterministic value of l by a non-spatial (i.e. the same for all grid cells) stochastic variable with mean 0.005 and variance 0.000001.

Open `precip.py` and save as `lapse.py`. Modify the script using the stochastic l instead of the deterministic fixed value. As the uncertainty in lapse rate has effect on the uncertainty in temperature, also calculate the mean, variance, and percentiles for `temp`. Run the script.

Display the probability density function of temperature, to see the effect of uncertain lapse rate on temperature.

Question: Write down the variance in snow depth at the two locations on `locations.map`, for time step 120 and 170 days. Compare the result with the version of the model that ignored uncertainty in lapse rate.

1. Day 120: Location 1, 4.53, Location 2, 4.71; Day 170: Location 1, 7.39; Location 2, 8.72
2. Day 120: Location 1, 5.12, Location 2, 9.28; Day 170: Location 1, 1.75; Location 2, 2.29
3. Day 120: Location 1, 4.99, Location 2, 1.41; Day 170: Location 1, 6.44; Location 2, 0.35
4. Day 120: Location 1, 5.16, Location 2, 4.41; Day 170: Location 1, 1.75; Location 2, 6.28
