

Modelling with PCRaster Python: fire spread

Judith Verstegen (j.a.verstegen@uni-muenster.de) and Derek Karssenberg (d.karssenberg@uu.nl)

1. How to work with PCRaster Python

To work with PCRaster Python, you need:

- To have installed Python, NumPy, and PCRaster, see for installation instructions: pcraster.geo.uu.nl/getting-started/
- A general or Python-specific editor to edit your script, e.g. IDLE, vi, Spyder, etc.
- A command prompt to run your script and to call Aguila to visualize your results

Download material: https://github.com/JudithVerstegen/PCRaster_Python_tutorial. Open a command prompt¹ and navigate to the directory `/script_and_data`. One of the three files is the Python script `fire.py`. It is the template of your model. If you want to run the script in Python, enter the command²:

```
python fire.py
```

To edit this script, open it in the editor of your choice, make your edits, and save them.

2. Fire spreading – conceptual model

The large-scale spatio-temporal pattern of an evolving fire is driven by local interactions at the fire front. Spatial neighbourhood functions are suitable for modelling these local interactions. In the first part of this tutorial, you will build a dynamic fire spread model with the following four transition rules that are applied in each time step:

1. A cell that was burning at the previous time step remains 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 and y direction (Von Neumann neighbourhood).
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 catches fire in the current time step becomes a burning cell.

¹ On Windows, you can do this by clicking on Start and typing 'cmd'.

² Alternatively, depending on which editor you are using, you can run the script from inside the editor.

3. Dynamic fire spread model - implementation

You are provided with a Python script, `fire.py`, that serves as the template of your model. Besides that, a number of PCRaster maps are provided to you. In this first part of the tutorial, you will use:

- `clone.map`: containing True cells inside the study area and False outside.
- `start.map`: containing True cells at the starting point of the fire.

In this section, you are given hints to add the 4 rules step by step. Make use of the list of all PCRaster functions required for the model, which is given in section 5. First, add statements to `fire.py` to represent rule 2.

- In the `initial()`, set the initial distribution of the fire (use the variable name `fire`) to `start.map`. Use the function `self.readmap()`.
- In the `dynamic()`, add a `window4total()` statement that calculates the number of neighbours that are burning, and put the result in a variable called `nrBurning`.
- In the `dynamic()`, use the `nrBurning` variable to make variable `neighbourBurns` that is True when at least one neighbouring cell is burning.
- Write the resulting maps to disk with the `self.report()` function, using the filename `'nb'`. Save the script and run it (see section 2. on how to run a script).

Load the resulting time series `nb` by calling Aguilu from the command prompt from time step 1 to 100 for the time series file name, as follows:

```
aguila --timesteps=[1,100] nb
```

Animate the time series by clicking on the yellow rectangle in the upper left corner. Is this model dynamic? Why (not)? And do the cells with True also include cells that were already burning in the previous step?

And finally for rule 2, add statements that calculate `potentialNewFire`, a map that is True *only* for cells that potentially catch fire. Combine `fire` and `neighbourBurns` using Boolean logic operations to calculate this. 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 in which each cell is True with a probability of 0.1.

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

Next, add a statement that calculates `newFire`, a map that is True for cells that actually catch fire in the current time step. To do this, combine `potentialNewFire` and `realization` using Boolean logic again. The resulting `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 to represent rules 1 and 4. Write the variable to disk, run it, and check the results. What is the pattern of the fire spreading over time? Do you think this is realistic given what you know about the real-world patterns of fire spread?

4. Extending the model using additional input data

In the first part of this tutorial, we assumed the probability of a cell catching fire to be the same everywhere, i.e. 0.1. In reality, however, a fire front extends faster uphill. A very simple representation of this process is used in this second part of the tutorial. The transition rules are kept the same, except you will change rule 3 into:

3. All cells with a burning direct neighbour in **downhill direction** have a probability to catch fire of 0.8. For all other cells, the probability remains 0.1.

You will use the elevation of each cell, from the other PCRaster map provided:

- `dem.map`: containing floating point values indicating the elevation in meters.

Make use of the list PCRaster functions in section 6. Save the script `fire.py` that was created in the previous exercise as `fireUphill.py` and open it in your editor. In the `initial()`, create the local drain direction (ldd) map using the digital elevation model `dem.map`. Remember that the ldd will be used in the `dynamic()`. Save the ldd to disk and display it together with the digital elevation model to evaluate your result; zoom in if you get an almost black map.

In the `dynamic()`, you need to add statements calculating for each time step the probability for cells to catch fire. As a start, add two statements (insert them directly above the calculation of the variable `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 the variable `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, run the model, and have a look at the results.

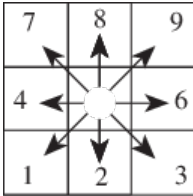
Finally, you need to change the calculation of `realization` as it needs to take into account the spatial pattern (for each time step) of the probability. Do this by changing the statement, using the computed variable `probability`. Save and run the model. Animate the resulting fire pattern together with the local drain direction map and the digital elevation model. What is the pattern of the fire spreading over time? Do you think this is more realistic compared to the previous model?

5. PCRaster functions needed for section 3

Name function /operator	Syntax for input(s) x (and y), output z	Input(s)	Output
<	$z = x < y$	x) First scalar map or value y) Second scalar map or value	Boolean map with True cells where first map/value is smaller than the second.
>	$z = x > y$	x) First scalar map or value y) Second scalar map or value	Boolean map with True cells where first map/value is larger than the second.
pcrand()	$z = \text{pcrand}(x, y)$	x) First Boolean map or value y) Second Boolean map or value	Boolean map with True cells where both the first and second map/value are True.
pcrnot()	$z = \text{pcrnot}(x)$	x) Boolean map or value	Boolean map with True cells where cells in the input map are False; False cells where cells in input map are True.
pcror()	$z = \text{pcror}(x, y)$	x) First Boolean map or value y) Second Boolean map or value	Boolean map with True cells where either the first or second map/value or both are True.
scalar()	$z = \text{scalar}(x)$	x) Map (any type) or value	Input map/value converted to scalar (floating point) map.
self.readmap()	$z = \text{self.readmap}(x)$	x) String of the map filename (without extension)	A PCRaster map object with the data from the map file.
self.report()	self.report(x, y) (z comes on disk)	x) Python map object to write to disk y) Filename (without extension)	A single map on disk, when used in the initial(), or a time series of maps, when used in the dynamic().
uniform()	$z = \text{uniform}(x)$	x) Boolean map or value	Scalar map with for each True cell in the input map a random floating point number from U(0,1).
window4total()	$z = \text{window4total}(x)$	x) Scalar map	Scalar map with in each cell the sum of its four (upper, lower, left and right) neighbouring cells.

For all other PCRaster functions, see <http://pcraster.geo.uu.nl/support/documentation/>, Functional list of applications and operations.

6. Additional PCRaster functions needed for section 4

Name function /operator	Syntax for input(s) x (and y), output z	Input(s)	Output
downstream()	$z = \text{downstream}(x, y)$	x) Local drain direction map (ldd) y) Map (any type) or value	Map (type depends on y) in which cells obtain the cell value from map y of their downstream direct neighbour.
lddcreate()	$z = \text{lddcreate}(x, y, y, y, y)$	x) Scalar elevation map y) Not important to understand at this point, just put the value $1 \cdot 10^{31}$ (1e31 in Python)	Map with drain direction as integer between 1 and 9:  <p>A value 5 defines a cell without a drain direction (pit).</p>
ifthenelse()	$z = \text{ifthenelse}(x, y1, y2)$	x) Condition: Boolean map y1) Map (any type) or value that should be assigned to cells where condition is True y2) Map (same type as y1) or value that should be assigned to cells where condition is False	Map (type depends on y) with cell values from y1 where the condition is True and cell values from y2 where the condition is False.

For all other PCRaster functions, see <http://pcraster.geo.uu.nl/support/documentation/>, Functional list of applications and operations.