PCRaster Python

PCRaster Python user manual

PCRaster Python: PCRaster Python user manual

Published This document was generated 2011-03-22.

Table of Contents

List of Tables

List of Examples

Chapter 1. Introduction

PCRaster is a powerful package of software for environmental dynamic modelling. It is a dynamic modelling system for distributed simulation models. The main applications of PCRaster are found in environmental modelling: geography, hydrology, ecology to name a few. Examples include rainfall runoff models, vegetation competition models and slope stability models.

PCRaster offers a large number of operations for spatio-temporal analysis. These high-level operations allow the development of models that are substantial shorter than programs offering the same functionality, but developed with system programming languages like C or FORTRAN. This makes PCRaster model scripts easy to construct, to modify and to understand. More information about PCRaster, including software, manuals and course material can be found at the PCRaster website: pcraster.geo.uu.nl [http://pcraster.geo.uu.nl].

Python is an interpreted, object-oriented programming language. Python scripts are platform independent, they can be executed on different operating systems like UNIX, Linux, Windows, Macintosh and others. Python has modules, classes, exceptions, very high level dynamic data types, dynamic typing and a clean, understandable syntax. More information about Python, including software and manuals can be found at the Python website: www.python.org [http://www.python.org].

The PCRaster Python extension enables you to use the PCRaster modelling engine from Python. This means that you can combine and call more than 150 PCRaster operations from Python. Combined with the flexible nature of the Python programming language this gives a powerful environment for creating environmental models.

1.1. Requirements

We assume that the user is familiar with creating models in PCRaster and has basic Python programming knowledge.

The minimum requirement to use PCRaster Python is a functioning install of Python [http://www.python.org/] version 2.5. If you want to use the conversion functions to/from NumPy, in addition Numerical Python [http://numpy.scipy.org/] must be installed.

The PCRaster Python extension is part of the standard PCRaster installation. See the separate installation note for further information.

Chapter 2. PCRaster Python

2.1. Quickstart

The Python example below is included in the workspace/Demo directory of the PCRaster distribution and shows the general usage of PCRaster functions within a Python script. To run the script change to the demo directory and execute **python2.5 example.py**. The syntax of the PCRaster Python functions equates the syntax of the PCRaster functions. The results of the functions are written to disk with the `report` function.

The following sections give a detailed introduction about how to build environmental models in the Python scripting language.

Example 2.1. Python script `example.py` demonstrating the use of PCRaster functions.

```
# Python
from PCRaster import *

setclone("dem.map")

# calculate a map with the distances to the nearest rainstation
raindist = spread("rainstat.map", scalar(0), scalar(1))
# writing map 'raindist' with filename 'raindist.map' to disk
report(raindist, "raindist.map")

# Calculate the infiltration capacity map by crossing the soil map
# and the infilcap.tbl
infilcap = lookupscalar("infilcap.tbl", "soil.map")
report(infilcap, "infilcap.map")

# Generate a local drain direction map on basis of the digital
# elevation map.
ldd = lddcreate("dem.map", 1e31, 1e31, 1e31, 1e31)
report(ldd, "ldd.map")


# Generating a map with a random value taken from a normal distribution
randomField = max(scalar(0), scalar(0.005) + mapnormal() / scalar(1000))
report(randomField, "randomField.map")

# Execute the accuthreshold operator with simulated rainfall
runoff = accuthresholdflux("ldd.map", randomField, "infilcap.map")
report(runoff, "runoff.map")

# Generating a map holding elevation values above 95m
uplandArea = ifthen("dem.map" > scalar(95), "dem.map")
report(uplandArea, "upland.map")
```

2.2. Introduction

You can start using PCRaster Python by importing the main module in your Python script:

```
# Python
import PCRaster
```

This will put all PCRaster functions into the pcraster namespace. Here is an example of how you can use the `slope` function to calculate the slope of a digital elevation model:

```
# Python
import PCRaster

gradient = PCRaster.slope("dem.map")
PCRaster.report(gradient, "gradient.map")
```

This is equivalent to the PCRcalc script

```
# PCRcalc
report gradient.map = slope(dem.map);
```

After importing the pcraster module the `slope` function is called with a filename as its argument. The function will open the raster file, read its values and calculate the slope. The resulting raster is returned and assigned to the variable `gradient`.

Both, PCRcalc and PCRaster Python operations use exactly the same algorithm. If you compare the Python and the PCRcalc code you will see a minimal difference. For information about `slope` function or any of the other functions of PCRaster you can look them up in the PCRaster reference manual.

The resulting gradient calculated above can be used as input to another function like this:

```
# Python script
import PCRaster

gradient = PCRaster.slope("dem.map")
smoothGradient = PCRaster.windowaverage(gradient, 3)
PCRaster.report(smoothGradient, "smoothGradient.map")
```

By combining functions like this environmental models can be created.

In the examples given so far we had to explicitly state the module which the PCRaster functions are part of (PCRaster). We can do better by importing all symbols from the pcraster module into the current scope. This way our script will become shorter and easier to read:

```
# Python script
from PCRaster import *

gradient = slope("dem.map")
smoothGradient = windowaverage(gradient, 3)
report(smoothGradient, "smoothGradient.map")
```

Note

From now on the `import` statement will be discarded from the examples.

Warning

By importing all symbols from a module there is an increasing chance that symbols with the same name end op in the same namespace. Keep this in mind when, for example, the abs function seems to work but gives an unexpected result: which function got called, math.abs() or PCRaster.abs()?

2.3. Operators

Besides functions operators can be used in expressions. Most boolean, comparison and arithmetic operators of PCRcalc have corresponding operators defined in PCRaster Python. Here's an example of the use of an operator:

```
# Python script
gradient = slope("dem.map")
steepSlopes = gradient > 20
```

Below is a table of all operators defined in PCRaster Python and their their equivalents in PCRcalc. For more information about operators see the PCRaster manual pages and the Python operator module.

Table 2.1. List of PCRaster Python Operators

PCRaster Python operators	PCRaster Python functions	PCRcalc equivalents
a & b		a and b
a | b		a or b
a ^ b		a xor b
~a		not a
a != b	ne(a, b)	a ne b, a != b
a == b	eq(a, b)	a eq b, a == b
a > b	gt(a, b)	a gt b, a > b
a >= b	ge(a, b)	a ge b, a >= b
a < b	lt(a, b)	a lt b, a < b
a <= b	le(a, b)	a le b, a <= b
a * b	mul(a, b)	a * b
a / b	div(a, b)	a / b, a div b
a // b	floordiv(a, b)	a idiv b
a ** b	pow(a, b)	a ** b
a % b	mod(a, b)	a mod b
a + b	add(a, b)	a + b
a - b	sub(a, b)	a - b
-a		- a
+a		+ a

Chapter 3. Differences between PCRcalc and PCRaster Python

The majority of PCRcalc operations can be easily adjusted to PCRaster Python operations. Nevertheless, both modelling languages differ in a few areas. The following sections explain these differences and give sample applications. The differences in the operators (e.g +,-,...) are already listed in Table 2.1.

3.1. Setting geographical attributes of a model

To use the PCRaster operations you need to define the model area and resolution. While this is done in PCRcalc with the `areamap` section, PCRasterPython does not make use of sections. Apply the `setclone` operation instead:

```
setclone(map)
   Sets the clone.

   map      Filename of clone map.
```

Thus, the PCRcalc areamap section

```
areamap clone.map;
```

would become in a PCRaster Python script

```
setclone("clone.map")
```

Note

The first operation in a script must be the `setclone` operation.

Like in PCRcalc scripts it is not possible to mix maps with different extents, i.e. varying cellsize, number of rows or columns.

3.2. Reading input maps

An input map is identified by a Python string containing the pathname to the disk dataset. The function readmap(pathname) reads a map.

- PCRaster operations with a function syntax can accept the pathname.

```
# Python code
Dem = readmap("dem.map")
Slope1 = slope(Dem)
# or direct by pathname
Slope2 = slope("dem.map")
```

- PCRaster operations with an operator syntax do not accept a pathname.

```
# Python code
# will yield "dem.mapdem.mapdem.map"
stringTimes3 = "dem.map" * 3
# will yield a Spatial object
DemTime3 = readmap("dem.map") * 3
```

- When using the Framework one usually uses the class method readmap(self, filename) instead of the function readmap(pathname).

```
# Python code  ... denotes removed fragments
...
class RunoffModel(object):
...
  def initial(self):
    # read static data
    self.dem = self.readmap("dem.map")
  def dynamic(self):
    # reads rain0000.001, rain0000.002, etc.
    self.rainFall = self.readmap("rain")

dynModel = DynamicFramework(RunoffModel, 50)
dynModel.run()
```

3.3. Writing output maps

To store maps use:

```
report(map, filename)
  Writes a map to a file.

  map      Map you want to write.
  filename Filename to use.
```

- Note the differences between PCRcalc and Python

```
# PCRcalc script code
binding
  gradient = output.map;
  dem = dem.map;
initial
   report gradient = slope(dem);
# Python code
gradient = slope("dem.map")
report(gradient, "gradient.map")
```

- The `report` operation only writes spatial data to disk. For writing non spatial data as timeseries in the DynamicFramework use the TimeoutTimeseries construct.

3.4. Accessing cell values of a map

With PCRaster Python you can query cell values from a map:

```
cellvalue(map, row, col)
  Returns a cell value from a map.

  map      Map you want to query.
  row      Row index of a cell in the map. Indices range from
  [1, numer-of-rows].
  col      Col index of a cell in the map. Indices range from
  [1, numer-of-cols].
  Returns a tuple with two elements: the first is the cell value, the second
  is a boolean value which shows whether the first element, is valid or not.
  If the second element is False, the cell contains a missing value.


cellvalue(map, index)
```

```
  Returns a cell value from a map.

  map      Map you want to query.
  index    Linear index of a cell in the map. Indices range from
  [1, numer-of-cells].
  Returns a tuple with two elements: the first is the cell value, the second
  is a boolean value which shows whether the first element, is valid or not.
  If the second element is False, the cell contains a missing value.
```

3.5. Global options

To set the global options of a script use

```
setglobaloption(name)
  Sets the global option name. The option name must not contain the leading
  dashes as used on the command line of pcrcalc.

  Python example:
    setglobaloption("unitcell")

  The pcrcalc equivalent:
    pcrcalc --unitcell -f model.mod
```

3.6. Writing time series

PCRaster Python does not provide a `timeoutput` operation like PCRcalc does. Instead, a separate class in the modelling framework is handling the PCRcalc timeoutput style timeseries.

Therefore, the PCRcalc

```
binding
  outlet=samples.map;
dynamic
  ...
  Q = ...
  report runoff.tss = timeoutput(outlet , Q);
```

will become in your Python model script

```
from PCRaster.Framework import *

class RunoffModel(object):
  def initial(self):
    ...
    self.runoffTss=TimeoutputTimeseries("runoff",self,"samples.map",noHeader=False)

  def dynamic(self):
    ...
    runoff = ...
    self.runoffTss.sample(runoff)

myModel = RunoffModel("clone.map")
dynModelFw = DynamicFramework(myModel, endTimeStep=28, firstTimestep=1)
dynModelFw.run()
```

In the `initial` section of the model class you create a member variable `self.runoffTss` holding the `TimeoutputTimeseries` object. The output is written to the file `runoff.tss` (for the DynamicFramework in the current working directory, for the MonteCarloFramework it will store the file into the corresponding sample subdirectories).

`samples.map` is either a boolean, nominal or ordinal map holding the sample locations. By default the header section is written to the timeseries file.

In the `dynamic` section `self.runoffTss.sample(runOff)` samples the values of the expression (here `runoff`) at the given locations for the current timestep. Note that for sequenced calls of `sample()` the values of the last call are sampled.

See also the example script `runoff.py` in the deterministic subdirectory of the workspace/framework directory.

3.7. Converting to and from NumPy arrays

Note:

The conversion functions are no longer provided by default by importing the PCRaster module, instead use:

```
from PCRaster.NumPy import *
```

You can convert PCRaster maps to NumPy arrays and vice versa with the following conversion functions:

```
numpy2pcr(dataType, array, mv)
  Converts entities from NumPy to PCRaster.

  dataType Either Boolean, Nominal, Ordinal, Scalar, Directional or Ldd.
  array    Array you want to convert.
  mv       Value that identifies a missing value in array.
  Returns a map

pcr2numpy(map, mv)
  Converts entities from PCRaster to NumPy.

  map      Map you want to convert.
  mv       Value to use in result array cells as a missing value.
  Returns an array
```

The following example script demonstrates the usage of the conversion functions:

```
# Python
import numpy
from PCRaster import *
from PCRaster.NumPy import *

setclone("clone.map")

a = numpy.array([[12,5,21],[9,7,3],[20,8,2],[5,6,-3]])

# conver a to a PCRaster Python map
# with the value 20 indicating a 'missing value' cell
n2p = numpy2pcr(Nominal, a, 20)
print "cellvalue:", cellvalue(n2p, 2, 3)[0]

# write it to a PCRaster map
report(n2p, "n2p.map")
# read the PCRaster map back
p2n = readmap("n2p.map")
# print it as a numpy array
# missing value is replaced by 9999
print pcr2numpy(p2n, 9999)
```

Execution of the script will result in the following output:

```
cellvalue: 3
[[  12    5   21]
 [   9    7    3]
 [9999    8    2]
 [   5    6   -3]]
```

3.8. PCRcalc operations returning multiple results

In PCRcalc some operations can be combined and return two result values like:

```
# PCRcalc
state,flux = accufractionstate,accufractionflux(ldd.map,material.map,0.5);
```

This syntax is not supported in PCRaster Python scripts. Use two separate operations instead, although this will double the execution time:

```
# Python
state = accufractionstate("ldd.map", "material.map", 0.5)
flux = accufractionflux("ldd.map", "material.map", 0.5)
```

3.9. Boolean operations

Do not use PCRaster objects in context of Boolean operations.

"In the context of Boolean operations, and also when expressions are used by control flow statements, the following values are interpreted as false: None, numeric zero of all types, empty sequences (strings, tuples and lists), and empty mappings (dictionaries). All other values are interpreted as true." —Python Reference Manual [http://docs.python.org/ref].

This means that PCRaster objects will always be interpreted as true when used in the above mentioned cases. In this case, the PCRaster function `ifthen(else)` should be used.

Chapter 4. Known bugs

The following PCRaster Python operations are known to be deficient:

nominal, ordinal	Some nominal and ordinal numbers larger than 198000000 are not correct represented. To avoid the problem use lower class values. In some cases large nominal or ordinal numbers are used in models where scalar numbers are more natural.
argorder and related functions	A variable number of arguments is currently not supported

Chapter 5. License

**** End-User License Agreement ****

This End-User License Agreement is a legal contract between you, as either an individual or a single business entity, and PCRaster Environmental Software (PES Consultancy BV).

1. Use of the software implies that you have read this license agreement and agree fully with what is stated in this agreement.
2. If you obtained the free Microsoft Windows version of PCRaster, for each personal registration you are licensed to use the software on one computer. For any other use of PCRaster you have purchased the correct license from PCRaster Environmental Software.
3. Any unauthorized duplication of PCRaster or its accompanying documentation is prohibited.
4. The software package PCRaster is developed by and copyright (c) since 1987 by PCRaster Environmental Software and the Faculty of Geographical Sciences of Utrecht University.
5. Use of the software is permitted under the following conditions:
   a. Not to sell or give away the software or any products which incorporate parts of or the total PCRaster package.
   b. Not to use the software or the practical exercises to give courses to third parties without prior approval of the owners of PCRaster.
   c. Not to develop any product which incorporate PCRaster or can be considered a shell executing PCRaster commands.
   d. Not to change the software, alter the software, decompile or use 'reverse engineering' or disassembly techniques, develop products based on the software or to create 'look and feel' products derived from the software.
   e. To accept the program as is, with no obligation on the part of the developers to provide corrections or assistance, beyond the support contract offered to you.
6. NO LIABILITY FOR CONSEQUENTIAL DAMAGES. In no event will the author be liable to you for any additional damages, including any lost profits, lost savings, or other incidental or consequential damages arising from the use of, or inability to use, this software and its accompanying documentation, even if the author has been advised of the possibility of such damages.
7. The authors of PCRaster make no warranty of any kind, expressed or implied, including without limitation any warranties of merchantability and/or fitness for a particular purpose. The authors do not assume any liability for the use of PCRaster beyond the original purchase price of this software.

**** End Of End-User License Agreement ****