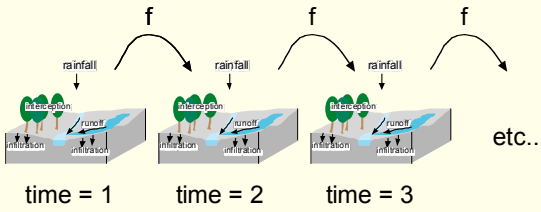
Dynamic Modelling with PCRaster Python

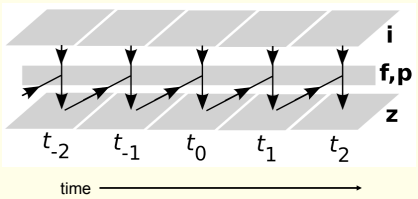
Derek Karssenberg, Faculty of Geosciences, Utrecht University

Process-based model



Examples:
- land degradation model
- vegetation competition model
- land use change model

Model components



Spatial dynamic model

$$z_{1..m}(t) = f(z_{1..m}(t\text{-}1), i_{1..n}(t), p_{1..l})$$

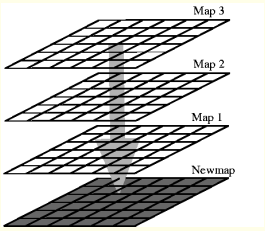
state variables

inputs (i.e. drivers or boundary conditions)

parameters

transition function

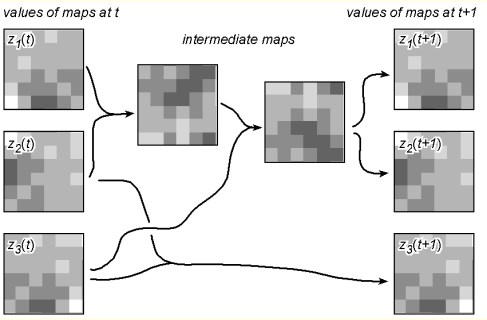
Functions and operators: building blocks of the transition function



inputs

output(s)

Representing the transition function



Input and output of building blocks: maps

Data types:

data type	description attributes	domain	example
boolean	boolean	0 (false), 1 (true)	suitable/unsuitable, visible/non visible
nominal	classified, no order	0...255, whole values	soil classes, administrative regions
ordinal	classified, order	0...255, whole values	succession stages, income groups
scalar	continuous, lineair	- 10exp(37)...10exp(37), real values	elevation, temperature
directional	continuous, directional	0 to 2 pi (radians), or to 360 (degrees), and -1 (no direction), real values	aspect
ldd	local drain direction to neighbour cell	1...9 (codes of drain directions)	drainage networks, wind directions

Syntax of operators

Result = *expression1* operator *expression2*

operator:

- the name of the operator

expression1, expression2 are the arguments (i.e. inputs):

- maps
- expressions resulting in a map (i.e., nesting of expressions is possible)

Result is the return value (i.e. what is created):

- one map

Example, multiply two maps (for each cell), arguments are maps:

```
MapA = MapB * MapC
```

Example, multiply maps (for each cell), second arguments is another expressions:

```
MapA = MapB * (MapC+MapD)
```

Syntax of functions

Result = function(*expression1*, *expression2*,..,*expressionn*)

function:

- the name of the function

expression1, *expression2*,..,*expressionn* are the arguments (i.e. inputs):

- maps
- expressions resulting in a map

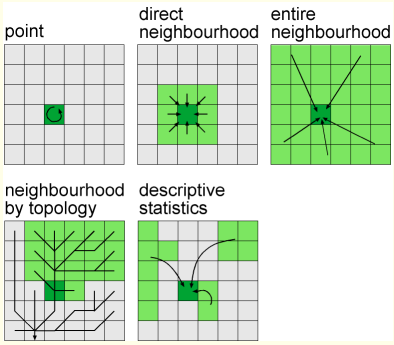
Result is the return value (i.e. what is created):

- one map (sometimes two)

Example, water flow over a local drain direction network (**accuflux** function):

```
RunoffMap = accuflux(LddMap,1000*RainMm)
```

PCRaster operations



Dynamic modelling framework

```
from pcraster import *
from pcraster.framework import *

class MyFirstModel(DynamicModel):
  def __init__(self):
    DynamicModel.__init__(self)
    setclone('dem.map')

  def initial(self):
    print 'running the initial'

  def dynamic(self):
    print 'running the dynamic'

nrOfTimeSteps=10
myModel = MyFirstModel()
dynamicModel = DynamicFramework(myModel,nrOfTimeSteps)
dynamicModel.run()
```

Import PCRaster module

Initialize class instance

Initial definitions

Transition function

Run the model

Do not change anything, except..

```
from pcraster import *
from pcraster.framework import *

class MyFirstModel(DynamicModel):
  def __init__(self):
    DynamicModel.__init__(self)
    setclone('dem.map')

  def initial(self):
    print 'running the initial'

  def dynamic(self):
    print 'running the dynamic'

nrOfTimeSteps=10
myModel = MyFirstModel()
dynamicModel = DynamicFramework(myModel,nrOfTimeSteps)
dynamicModel.run()
```

Provide a clone map

Insert map functions

Insert map functions

Provide nr. of timesteps

Example using Python types and operators

```
from pcraster import *
from pcraster.framework import *

class MyFirstModel(DynamicModel):
  def __init__(self):
    DynamicModel.__init__(self)
    setclone('dem.map')

  def initial(self):
    conversionValue = 3.0
    self.reservoir = 30.0 / conversionValue
    print 'initial reservoir is: ', self.reservoir

  def dynamic(self):
    outflow = 0.1 * self.reservoir
    self.reservoir = self.reservoir - outflow + 0.5
    print self.reservoir

nrOfTimeSteps=100
myModel = MyFirstModel()
dynamicModel = DynamicFramework(myModel,nrOfTimeSteps)
dynamicModel.run()
```

Making variables 'global': use self. !!

```
from pcraster import *
from pcraster.framework import *

class MyFirstModel(DynamicModel):
  def __init__(self):
    DynamicModel.__init__(self)
    setclone('dem.map')

  def initial(self):
    conversionValue = 3.0
    self.reservoir = 30.0 / conversionValue
    print 'initial reservoir is: ', self.reservoir

  def dynamic(self):
    outflow = 0.1 * self.reservoir
    self.reservoir = self.reservoir - outflow + 0.5
    print self.reservoir

nrOfTimeSteps=100
myModel = MyFirstModel()
dynamicModel = DynamicFramework(myModel,nrOfTimeSteps)
dynamicModel.run()
```

Defined in initial method

Used in dynamic

Example using PCRaster functions and operators

```
from pcraster import *
from pcraster.framework import *

class MyFirstModel(DynamicModel):
  def __init__(self):
    DynamicModel.__init__(self)
    setclone('clone.map')

  def initial(self):
    aUniformMap = uniform(1)
    self.report(aUniformMap,'uni')
    self.alive = aUniformMap < 0.1
    self.report(self.alive,'ini')

  def dynamic(self):
    aliveScalar=scalar(self.alive)
    numberOfAliveNeighbours=windowtotal(aliveScalar,3)-aliveScalar;
    self.report(numberOfAliveNeighbours,'na')

    threeAliveNeighbours = numberOfAliveNeighbours == 3
    self.report(threeAliveNeighbours,'tan')
```

Storing static maps: self.report(..,..)

```
from pcraster import *
from pcraster.framework import *

class MyFirstModel(DynamicModel):
  def __init__(self):
    DynamicModel.__init__(self)
    setclone('clone.map')

  def initial(self):
    aUniformMap = uniform(1)
    self.report(aUniformMap,'uni')
    self.alive = aUniformMap < 0.1
    self.report(self.alive,'ini')

  def dynamic(self):
    aliveScalar=scalar(self.alive)
    numberOfAliveNeighbours=windowtotal(aliveScalar,3)-aliveScalar;
    self.report(numberOfAliveNeighbours,'na')

    threeAliveNeighbours = numberOfAliveNeighbours == 3
    self.report(threeAliveNeighbours,'tan')
```

Stores the map variable `aUniformMap` using the file name `uni.map`.

Storing dynamic maps: self.report(..,..)

```
from pcraster import *
from pcraster.framework import *

class MyFirstModel(DynamicModel):
  def __init__(self):
    DynamicModel.__init__(self)
    setclone('clone.map')

  def initial(self):
    aUniformMap = uniform(1)
    self.report(aUniformMap,'uni')
    self.alive = aUniformMap < 0.1
    self.report(self.alive,'ini')

  def dynamic(self):
    aliveScalar=scalar(self.alive)
    numberOfAliveNeighbours=windowtotal(aliveScalar,3)-aliveScalar;
    self.report(numberOfAliveNeighbours,'na')

    threeAliveNeighbours = numberOfAliveNeighbours == 3
    self.report(threeAliveNeighbours,'tan')
```

Stores the map variable `numberOfAliveNeighbours` using the file names `na000000.001`, `na000000.002`, `na000000.003`, etc.

Reading static maps from disk: `self.readmap(..)`

```
from pcraster import *
from pcraster.framework import *

class MyFirstModel(DynamicModel):
  def __init__(self):
    DynamicModel.__init__(self)
    setclone('dem.map')

  def initial(self):
    self.dem = self.readmap('dem')
    slopeOfDem = slope(self.dem)
    self.report(slopeOfDem,"gradient")

  def dynamic(self):
    precipitation=self.readmap('precip')
    precipitationMMPerHour=precipitation*1000.0
    self.report(precipitationMMPerHour,"pmm")
    highPrecipitation=precipitation > 0.01
```

Reads the file `dem.map` from disk and assigns it to the variable `self.dem`

Reading dynamic maps from disk: `self.readmap(..)`

```
from pcraster import *
from pcraster.framework import *

class MyFirstModel(DynamicModel):
  def __init__(self):
    DynamicModel.__init__(self)
    setclone('dem.map')

  def initial(self):
    self.dem = self.readmap('dem')
    slopeOfDem = slope(self.dem)
    self.report(slopeOfDem,"gradient")

  def dynamic(self):
    precipitation=self.readmap('precip')
    precipitationMMPerHour=precipitation*1000.0
    self.report(precipitationMMPerHour,"pmm")
    highPrecipitation=precipitation > 0.01
    self.report(highPrecipitation,"high")
```

Reads the files `precip00.001`, `precip00.002`, `etc` from disk and assigns it to the variable `precipitation` for each time step