Stochastic spatio-temporal modelling

Model
\[ z_t = f(x_{t-1}, k, p_t) \]

for all time steps \( t = 1, 2, ..., T \)

state variables \( x \) \ninputs \( k \) \nparameters \( p \)

transition function

Solution scheme

for each \( n \) in Monte Carlo samples:

for each \( t \) in time steps:

\[ z_t = f(x_t, k, p_t) \]

Building blocks

discharge = kinetic(flowDir, precipitation, ...)

time map spatial function input maps

Building blocks to construct the transition function are functions on spatial data types (raster maps). Functions were developed in C++ and are available as Python functions (Python extension).

Solution framework (Python)

from PCRaster import *
from PCRaster.Framework import *

class SnowModel(DynamicModel, MonteCarloModel):
    def __init__(self, ...):
        self.snow = scalar(0)

    def dynamic(self):
        self.snow = scalar(0)

    def initial(self):
        self.snow = scalar(0)

    def suspend(self):
        self.snow = scalar(0)

    def postmcloop(self):

        self.report(self.snow, 's')

        self.suspend()

        self.read('s')

        return weight

        self.updateWeight()

        self.report(self.snow, 's')

        self.suspend()

        self.read('s')

        return weight

        self.updateWeight()

PCRaster

- Is targeted at the development of spatio-temporal models
- Fast model development and execution
- Scripting environments: PCCalc and Python
- Rich set of model building blocks for manipulating raster maps
- Framework for stochastic spatio-temporal model building
- Framework for data assimilation
- Tool for visualisation of spatio-temporal stochastic data
- Runs on Linux, Microsoft Windows and Apple OS X
- Can be downloaded for free and is soon open source

Data assimilation

PCRaster on supercomputers (parallel execution)

The high computational requirements for stochastic spatio-temporal modelling, and an increasing demand to run models over large areas at high resolution, e.g. in global hydrological modelling, require an optimal use of available, heterogeneous computing resources by the modelling framework. Current work in the context of the eWaterCycle project is on a parallel implementation of the modelling engine, capable of running on a high-performance computing infrastructure such as clusters and supercomputers.

Global model runs are distributed over multiple compute nodes (using eScience Technology Platform eStEP), where each node models one watershed. Each watershed is modelled by using all processors in the node (GPUs and CPUs), which is enabled by an OpenCL implementation of PCRaster functions. This will allow us to scale up to hundreds of machines, with thousands of compute cores.

PCRaster at EGU (selection)

Poster R187, EGU2013-11126 (Tuesday), Alberti et al. A web-application for visualizing uncertainty in numerical ensemble models.

Poster R215, EGU2013-13337 (Wednesday), Sutanudjaja et al., eWaterCycle: Developing a hyper resolution global hydrological model.

Poster R373, EGU 2013-10215 (Thursday), Wanders et al., The benefits of using remotely sensed soil moisture in parameter identification of large-scale hydrological models.

Poster R293, EGU 2013-10355 (Friday), Straatsma et al., Water2Invest: Global facility for calculating investments needed to bridge the climate-induced water gap.

Current work: integrated modelling

Global watershed models

models

Current work: integrated modelling

Local site phenomena

Global site phenomena

Particle Filter

Monte Carlo

Browsing in all dimensions

Visualisation of stochastic spatio-temporal data

Animated maps

Cumulative Probability Distributions

Time series

References

Information and download at: http://www.pcraster.eu

Large scale stochastic spatio-temporal modelling with PCRaster Python

Derek Karsenberg1, Niels Drost2, Oliver Schirr2, Kor de Jong1, Marc Bierkens1, 1Faculty of Geosciences, Utrecht University, 2Netherlands eScience Center, the Netherlands, info@pcraster.eu

Large scale stochastic spatio-temporal modelling

Solution framework

from PCRaster import *
from PCRaster.Framework import *

class SnowModel(DynamicModel, MonteCarloModel):
    def __init__(self, ...):
        self.snow = scalar(0)

    def dynamic(self):
        self.snow = scalar(0)

    def initial(self):
        self.snow = scalar(0)

    def suspend(self):
        self.snow = scalar(0)

    def postmcloop(self):

        self.report(self.snow, 's')

        self.suspend()

        self.read('s')

        return weight

        self.updateWeight()

        self.report(self.snow, 's')

        self.suspend()

        self.read('s')

        return weight

        self.updateWeight()