Environmental Modelling & Software 25 (2010) 489–502

Contents lists available at ScienceDirect

Environmental Modelling & Software

journal homepage: www.elsevier.com/locate/envsoft

A software framework for construction of process-based stochastic spatio-temporal models and data assimilation

Derek Karssenberg [a,*], Oliver Schmitz [a], Peter Salamon [b,1], Kor de Jong [a], Marc F.P. Bierkens [a,c]

[a] Department of Physical Geography, Faculty of Geosciences, Utrecht University, Heidelberglaan 2, PO Box 80115, 3508 TC, Utrecht, The Netherlands
[b] Land Management and Natural Hazards Unit, Institute for Environment and Sustainability, DG Joint Research Centre, European Commission, Via Enrico Fermi 2749, TP 261, 21027 Ispra (Va), Italy
[c] Deltares, Unit Soil and Groundwater, PO Box 3508 TA, Utrecht, The Netherlands

ARTICLE INFO

Article history:
Received 23 March 2009
Received in revised form
7 October 2009
Accepted 12 October 2009
Available online 14 November 2009

Keywords:
Data assimilation
Particle filter
Ensemble kalman filter
Hydrology
PCRaster
Python
Snow
Environmental model
Calibration
Spatio-temporal model

ABSTRACT

Process-based spatio-temporal models simulate changes over time using equations that represent real world processes. They are widely applied in geography and earth science. Software implementation of the model itself and integrating model results with observations through data assimilation are two important steps in the model development cycle. Unlike most software frameworks that provide tools for either implementation of the model or data assimilation, this paper describes a software framework that integrates both steps. The software framework includes generic operations on 2D map and 3D block data that can be combined in a Python script using a framework for time iterations and Monte Carlo simulation. In addition, the framework contains components for data assimilation with the Ensemble Kalman Filter and the Particle filter. Two case studies of distributed hydrological models show how the framework integrates model construction and data assimilation.

Software availability

Name: PCRaster
Developer: Department of Physical Geography, PO Box 80115, 3508 TC Utrecht, the Netherlands
Contact: d.karssenberg@geo.uu.nl
Required software: PCRaster, Windows (free), Linux and UNIX on request; http://pcraster.geo.uu.nl
Python: all major platforms; http://www.python.org
NumPy: http://numpy.scipy.org
Online courses: http://pcraster.geo.uu.nl

1. Introduction

Spatio-temporal numerical models simulating geographic change are one of the cornerstones of research in geography and earth science and are frequently used in management, planning, and risk assessment in application domains such as land use change (Ligtenberg et al., 2004; Moulin et al., 2004), hazards and evacuation (Helbing et al., 2000), ecosystem studies (Sydelko et al., 2001; Gimblett et al., 2003), spread of diseases (Breukers et al., 2006), criminology (Groff, 2007), land degradation and geomorphology (Karssenberg and Bridge, 2008; Wilkinson et al., 2009), or hydrology (Beven, 2002; Ajami et al., 2007; Blöschl et al., 2008; Brown et al., 2008). Although the application field may vary, spatio-temporal numerical models have in common that they simulate change over time using equations that represent real world processes (Wesseling et al., 1996; Burrough, 1998), whereby the state of the modelled system at each moment in time is a function of its state in the past. Another common characteristic is that processes are modelled in a spatially-explicit way, which means that spatial patterns and spatial interaction in the system are taken into account (Karssenberg and De Jong, 2005a). Spatio-temporal numerical models are either individual-based or field-based. Individual-based models, also referred to as agent-based or object-based models, consider the geographic space as a set of objects

* Corresponding author. Tel.: +31 30 2532768; fax: +31 30 2531145.
E-mail address: d.karssenberg@geo.uu.nl (D. Karssenberg).
[1] Tel.: +39 332 786013; fax: +39 332 786653. E-mail: peter.salamon@jrc.it.

1364-8152/$ – see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.envsoft.2009.10.004

(Benenson and Torrens, 2004; Grimm and Railsback, 2005). Field-based models represent the geographic space using continuous fields of attributes that have a value at all locations (Burrough and McDonnell, 1998). The focus in this paper is on field-based models, although many concepts presented here apply to individual-based models, too.

As it is required to use models tailored to the research goals of a project, the available data, and the properties of the system being modelled (Karssenberg et al., 2006), model development is central in almost any research project that involves modelling. Three important steps in the model development cycle (Karssenberg et al., 2006) are the conversion of the conceptual model structure to computer code, i.e. the implementation or construction of the model, model calibration, and state estimation by assimilation of spatio-temporal observational data collected by remote sensing, automatic data loggers, or questionnaires, or retrieved from large data bases. The term calibration is used for the process that aims at finding model parameters that result in an optimal fit between modelled and observed state variables (e.g., Hill and Tiedeman, 2007). The term data assimilation refers here to sequential Bayesian estimation. This procedure sequentially updates the model state at time steps when observations of state variables or parameters are available (e.g., Gelb, 1974; Simon, 2006). Data assimilation is increasingly being used to integrate data with spatio-temporal models in a wide range of different fields in the earth sciences, such as oceanography (van Leeuwen, 2003), hydrology (Clark et al., 2006; Moradkhani, 2008), ecology (Chen et al., 2008), or crop science (Naud et al., 2007). Below, we use the term optimization to refer to both calibration and data assimilation. Although both model development and optimization can be done by programming software from scratch using system programming languages, it is preferable to use software frameworks at a higher level of abstraction that can be used by scientists and modellers without specialist knowledge in programming (van Deursen et al., 2000; Karssenberg, 2002).

A number of software frameworks exist for construction of temporal numerical models in geography and earth science. Widely used are graphical modelling languages (ModelMaker, 2009; STELLA, 2009), languages incorporated in Geographical Information Systems (GRASS, 2009; ESRI, 2009), technical computer languages (MATLAB, 2008), and modelling languages designed for spatio-temporal modelling in geography (SIMUMAP, Pullar, 2004; PCRaster, 2009). Karssenberg (2002) and Karssenberg and De Jong (2005a) evaluate and discuss these frameworks. Apart from technical computer languages, none of these frameworks come with integrated tools for calibration of models or data assimilation. This is mostly done by interfacing the model with an external framework that incorporates solution schemes for calibration (e.g., PEST, 2008) or data assimilation (e.g., BUGS, 2008; COSTA, 2008; ReBEL, 2009).

The use of two different software frameworks for model construction and optimization has the disadvantage that the user requires knowledge of two different frameworks. This can be a problem as the frameworks will have totally different programming and visualisation environments. Also, the implementation of the interface between the model construction and optimization frameworks can be cumbersome and hinders modification of the model. The latter is because changing the model often comes with changes in the variables and parameters. As the optimization framework interfaces with the model through these variables and parameters, the interface that handles this needs to be adjusted. In many cases modifying the interface is not feasible within a project. As a result, exploratory model development whereby a number of candidate models are developed and optimized in order to find the optimal model is often not possible. A possible solution to these problems is the use of a single framework that supports model construction *and* optimization. This approach is followed here. Such integrated frameworks have not yet been widely developed as the focus of software development teams has been on either frameworks for model construction or model optimization. The proprietary MATLAB framework allows doing both when using the external ReBEL toolkit for optimization that runs inside MATLAB. In this paper we extend the PCRaster model construction framework (van Deursen, 1995; Wesseling et al., 1996; PCRaster, 2009). New modules for data assimilation with the widely used Ensemble Kalman Filter (e.g., Evensen, 2003) and the particle filter (van Leeuwen, 2003; Weerts and El Serafy, 2006) are added resulting in an integrated framework for model construction and optimization. The modeller has access to these components and combines them with the generic Python scripting language (Python, 2009). Stochastic spatio-temporal model inputs and outputs can be analysed with an integrated, interactive visualisation program. In addition to optimization of models built within the framework, the framework provides an interface to external models. The framework also integrates a calibration toolbox using Genetic Algorithms. For a description of this component the reader is referred to (AMORI, 2009).

The purpose of this paper is to explain how the integrated framework is used for model construction and data assimilation, and to evaluate the framework with two case studies of distributed models. The first case study is a simplified snowmelt model that is constructed inside the framework. We will assimilate distributed snow cover data into the model to improve estimation of snow cover and discharge. The assimilation of snow cover data is expected to improve the prediction of snow cover and discharge, as has been shown by others using remotely sensed snow cover data (e.g., Clark et al., 2006; Nagler et al., 2008). The second case study shows how the external LISFLOOD model (Van der Knijf et al., in press) can be optimized with the framework. LISFLOOD is a hydrological model that runs at the river basin scale. The purpose of the paper is mainly to show how the different filter techniques can be used and does not pretend to provide an extensive comparison of the performance of the filters. However, we provide a preliminary comparison of the Ensemble Kalman Filter and the Particle Filter.

2. Stochastic spatio-temporal modelling

2.1. Monte Carlo simulation and concepts of the framework

We first outline modelling concepts and define notations for the case without data assimilation or calibration. Let the vector $\mathbf{z}_t$ be the state variables of the model at time index $t = 1, 2, \ldots, T$. Given an initial state $\mathbf{z}_0$, $\mathbf{z}_t$ evolves over time according to the governing equation:

$$\mathbf{z}_t = \mathbf{f}_t(\mathbf{z}_{t-1}, \mathbf{i}_t, \mathbf{p}_t), \quad \text{for each } t. \tag{1}$$

In Eq. (1), $\mathbf{f}_t$ is a system transition function that mimics real world processes and $\mathbf{p}_t$ is a vector containing the parameters used in $\mathbf{f}_t$. The vector $\mathbf{i}_t$ contains the inputs or boundary conditions of the system. Each of the vectors in Eq. (1) may represent spatial attributes in two- or three-dimensional geographic space. In a stochastic model at least one vector contains stochastic variables. Also, $\mathbf{f}_t$ may be a sample from a probability distribution of different possible system transition functions.

Our software framework solves Eq. (1) by Monte Carlo simulation using the scheme:

for each n: (2a)

for each t: (2b)

$$\mathbf{z}_t^{(n)} = \mathbf{f}_t^{(n)}(\mathbf{z}_{t-1}^{(n)}, \mathbf{i}_t^{(n)}, \mathbf{p}_t^{(n)})$$

for each t: (2c)

$$\mathbf{Z}_t = g(\mathbf{z}_t^{(1,\ldots,N)}), \mathbf{I}_t = \mathbf{g}(\mathbf{i}_t^{(1,\ldots,N)}), \mathbf{P}_t = \mathbf{g}(\mathbf{p}_t^{(1,\ldots,N)})$$

In Eq. (2), the probability density functions (PDF) of the components of Eq. (1) are represented by a collection of N independent realizations, respectively, $\mathbf{z}_t^{(n)}$, $\mathbf{i}_t^{(n)}$, $\mathbf{p}_t^{(n)}$, and $\mathbf{f}_t^{(n)}$, with $n = 1, 2, \ldots, N$. The iterations over time (Eq. (2b)) to evaluate the governing Eq. (1) are performed inside the loop (Eq. (2a)) that iterates over the realizations. After finishing the loops (Eqs. (2a) and (2b)), the PDFs of the required components in Eq. (1) are available. The function $\mathbf{g}$ calculates sample statistics (e.g., moments, quantiles) from these PDFs and stores these in the vectors $\mathbf{Z}_t$, $\mathbf{I}_t$, and $\mathbf{P}_t$, respectively. The calculation order in Eq. (2) was chosen because of its generic application and ease of implementation. Alternative schemes are discussed in (Karssenberg and De Jong, 2006).

The scheme in Eq. (2) contains components that are generic and components that are specific for a particular model. The generation of Monte Carlo samples and iteration over time steps, the use of 2D or 3D spatio-temporal attributes and the calculations of sample statistics are generic components (Karssenberg and De Jong, 2005b; Karssenberg and De Jong, 2005a). The framework standardizes these as pre-programmed methods, functions and data types. However, the system transition function ($\mathbf{f}_t$) is specific for a particular model and it needs to be defined by the model builder. The modeller can do this by combining standard spatio-temporal functions on the 2D and 3D attributes. The framework includes a wide range of spatio-temporal functions taken from the PCRaster library (van Deursen, 1995; Wesseling et al., 1996; Karssenberg and De Jong, 2005a). In addition, pre-programmed functions are provided to create the realizations $\mathbf{z}_t^{(n)}$, $\mathbf{i}_t^{(n)}$, and $\mathbf{p}_t^{(n)}$.

2.2. Case study model

The use of the framework is illustrated with the implementation of a distributed snowmelt model. As it is only used for illustrative purposes, a number of processes are ignored and parameter values are based on assumptions or estimates from literature. We first outline the equations used by the model. The model uses a time step Δt (days) of one day. The precipitation (p_tm/day) is defined as a non-spatial stochastic time series:

$$p_t = p_{m,t} \times e_t \tag{3}$$

In Eq. (3), $p_{m,t}$ is the observed precipitation for time step t at the meteorological station in the study area, and $e_t \sim N(1,\ 0.04)$, a random variable for each time step t, independent of other time steps. The near-surface temperature $\mathbf{t}(\mathbf{s})_t$ (°C) is defined as a spatial stochastic variable:

$$\mathbf{t}(\mathbf{s})_t = t_{\mathrm{m},t} + \mathbf{h}(\mathbf{s}) \cdot l \tag{4}$$

In Eq. (4), $t_{\mathrm{m},t}$ (°C) is the observed near-surface temperature at the meteorological station in the study area and $\mathbf{h}(\mathbf{s})$ is a spatial field with the elevation (m) of each grid cell above the elevation at the meteorological station. The spatial fields are indicated by the spatial index $\mathbf{s}$, which is defined on a regular 2D grid (i.e. the study area). The lapse rate of the temperature (l, °C m^{-1}) is modelled as a static non-spatial stochastic variable, with $l \sim N(-0.005{,}1 \times 10^{-6})$. The value and the uncertainty of the season-averaged lapse rate were estimated from Marshall et al. (2007), Blandford et al. (2008) and Huang et al. (2008).

The snow pack $\mathbf{a}(\mathbf{s})_t$ (m water equivalent) is:

$$\mathbf{a}(\mathbf{s})_t = \mathbf{a}(\mathbf{s})_{t-1} + (\mathbf{p}_s(\mathbf{s})_t - \mathbf{b}(\mathbf{s})_t)\Delta t \tag{5}$$

In Eq. (5), $\mathbf{p}_s(\mathbf{s})_t$ is snowfall (m/day) and $\mathbf{b}(\mathbf{s})_t$ is snowmelt (m/day). These, and rainfall $\mathbf{p}_r(\mathbf{s})_t$ (m/day), are calculated at each cell as:

$$\left.\begin{array}{lll} \mathbf{p}_s(\mathbf{s})_t = p_t, & \mathbf{p}_r(\mathbf{s})_t = 0, & \mathbf{b}(\mathbf{s})_t = 0, \quad \text{for } \mathbf{t}(\mathbf{s})_t < 0 \\ \mathbf{p}_s(\mathbf{s})_t = 0, & \mathbf{p}_r(\mathbf{s})_t = p_t, & \mathbf{b}(\mathbf{s})_t = m \cdot \mathbf{t}(\mathbf{s})_t, \quad \text{for } \mathbf{t}(\mathbf{s})_t \geq 0 \end{array}\right\} \tag{6}$$

In Eq. (6), m (m day^{-1} °C^{-1}) is the degree–day factor, with a value of $m = 0.01$ m day^{-1} °C^{-1}. For each cell, the discharge ($\mathbf{q}(\mathbf{s})_t$) is calculated as the sum of $\mathbf{b}(\mathbf{s})_t + \mathbf{p}_r(\mathbf{s})_t$ values in its upstream cells. Upstream cells are derived from a local drain direction network calculated from the digital elevation model using the 8-point pour algorithm (Burrough and McDonnell, 1998).

The model is applied to a region in the Swiss Alps, south of the Vierwaldstätter See (centred around 46°45'N, 8°26'W). Observed rainfall ($p_{m,t}$) and temperature ($t_{m,t}$) time series for one winter season were synthesised from the ERA40 data set (Uppala et al., 2005). Elevation was taken from the GTOPO30 data set (EIONET, 2009).

We created an artificial data set to serve as observations in the data assimilation techniques described in the second part of the paper. The data set consists of one model realization having a lapse rate l of -0.004 °C m^{-1}.

2.3. Implementation of the snowmelt model

We will now explain the use of the frameworks with the implementation of a distributed snowmelt model. Below, references are made to lines in the entire script provided in Table 1. The modeller creates a model by defining a standard Python class, here `SnowModel` (line 4). Depending on the type of model, the modeller needs to implement a set of methods that are invoked by the associated frameworks. Here, the `DynamicFramework` requiring `initial` and `dynamic` methods (line 17 and 23), and the `MonteCarloFramework` requiring `premcloop` and `postmcloop` methods (line 10 and 38), are used. By deriving from preset classes the modeller is able to use methods allowing to query model specific attributes. In line 42, `self.timeSteps()` returning a list of time steps is an example of a method derived from the `DynamicModel` class. The modeller can use the preset classes and belonging methods as off-the-shelf components.

The approach followed in the design of the framework refactors optimization logic out of individual models into reusable framework classes. As a result the user models are easier to maintain, and the modeller is less burdened with framework logic. Furthermore, models can be used in combination with different optimization frameworks. An optimization framework is used by instantiating it while passing a user model object, and calling the `run` member function. Each framework thereby places requirements on the user model it is utilising. For example, the `DynamicFramework` requires its user model to have member functions called `initial` and `dynamic`. The `DynamicFramework` will call these functions once it is run. The frameworks not only can be instantiated with user models but also with other frameworks. For example, either `StaticFramework` or `DynamicFramework` objects can be passed to the `MonteCarloFramework` upon instantiation.

Model construction comes down to inserting the required functions inside the methods associated with a framework. Although any Python function could be used, here the model is

Table 1
Model script for stochastic snowmelt model.

```
1   from PCRaster import *
2   from PCRaster.Framework import *
3
4   class SnowModel(DynamicModel, MonteCarloModel):
5     def __init__(self):
6       DynamicModel. __init__(self)
7       MonteCarloModel.__init__(self)
8       setclone("clone.map")
9
10    def premcloop(self):
11      dem = self.readmap("dem")
12      self.ldd = lddcreate(dem, 1e31, 1e31, 1e31, 1e31)
13      elevationMeteoStation = scalar(2058.1)
14      self.elevationAboveMeteoStation = dem −
        elevationMeteoStation
15      self.degreeDayFactor = 0.01
16
17    def initial(self):
18      self.snow = scalar(0)
19      self.temperatureLapseRate = 0.005 + (mapnormal() * 0.001)
20      self.report(self.temperatureLapseRate, "lapse")
21      self.temperatureCorrection = self.elevationAboveMeteoStation *
        self.temperatureLapseRate
22
23    def dynamic(self):
24      temperatureObserved = self.readDeterministic("tavgo")
25      precipitationObserved = self.readDeterministic("pr")
26      precipitation = max(0, precipitationObserved *
        (mapnormal() * 0.2 + 1.0))
27      temperature = temperatureObserved − self.temperatureCorrection
28      snowFall = ifthenelse(temperature < 0, precipitation, 0)
29      self.snow = self.snow + snowFall
30      potentialMelt = ifthenelse(temperature > 0, temperature
        * self.degreeDayFactor, 0)
31      actualMelt = min(self.snow, potentialMelt)
32      self.snow = max(0, self.snow − actualMelt)
33      rain = ifthenelse(temperature >= 0, precipitation, 0)
34      discharge = accuflux(self.ldd, actualMelt + rain)
35      self.report(self.snow, "s")
36      self.report(discharge, "q")
37
38    def postmcloop(self):
39      names = ["s", "q"]
40      mcaveragevariance(names, self.sampleNumbers(),
        self.timeSteps())
41      percentiles = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
42      mcpercentiles(names, percentiles, self.sampleNumbers(),
        self.timeSteps())
43
44  myModel = SnowModel()
45  dynamicModel = DynamicFramework(myModel, 180)
46  mcModel = MonteCarloFramework(dynamicModel, 1000)
47  mcModel.run()
```

built using the set of functions from the PCRaster library. The framework provides these functions, as these are particularly useful for spatially explicit modelling in geoscientific domains. In principle, inputs and outputs of these functions are raster maps, although most functions take non-spatials (i.e., single values) as input, too.

The `premcloop` method is used to calculate parameters, inputs or variables that are constant and deterministic. The calculations defined in the `premcloop` are evaluated once, at the start of the model execution. At line 11, the digital elevation map (Fig. 1A) is read from disk and assigned to the map variable `dem`. In the next line, the function `lddcreate` derives the local drain direction map (Fig. 1B) from dem. Unlike the variable `dem`, the local drain direction map is defined as a member variable of the class `SnowModel`, using the `self` prefix. A variable is required by Python to be defined as member variable when it is used in other methods, too. Here, for instance, `ldd` is defined in the `premcloop` while it is used in the `dynamic` method.

The functions entered by the modeller in the `initial` and `dynamic` methods are evaluated for each Monte Carlo sample, representing the loop in Eq. (2a). All script variables calculated in these methods refer to realizations. The `initial` method is used to create or derive realizations of parameters, constant inputs, or the initial value of state variables. The `mapnormal()` function, drawing a realization from $N(0,1)$, is used in line 19 to create a realization of the temperature lapse rate (l in Eq. (4)). This non-spatial variable is used in line 21 to derive the realization `temperatureCorrection`, which gives for each cell a temperature corrected relative to an observed temperature at the meteo station. It represents $\mathbf{h}(\mathbf{s})\cdot l$ in Eq. (4). Also, the initial value of the state variable snow pack ($\mathbf{a}(\mathbf{s})_0$, in Eq. (5)) is set to zero assuming no snow pack at the start of the simulation.

The `dynamic` method contains calculations that represent $\mathbf{f}_t$ in Eq. (1). These are executed for each time step, for each Monte Carlo loop, with the order of calculations defined in Eq. (2). Observed temperature and precipitation ($p_{m,t}$ and $t_{m,t}$ in Eqs. (3) and (4)) are imported with the `self.readDeterministic` function (lines 24 and 25). This function reads for each time step a map from disk containing the required input for that time step. Line 26 creates a realization of the precipitation (p_t, Eq. (3)) by adding for each time step an independent realization to the observed precipitation. The realization of the temperature $\mathbf{t}(\mathbf{s})_t$ (Eq. (4)) is calculated in the next line, by adding `temperatureCorrection` to the observed temperature. Note that `temperatureCorrection` was created in the initial, so it is the same for all time steps according to the model description (Eq. (4)). Lines 27–33 represent Eq. (6) by a set of point operations on maps. The map `discharge` (Fig. 1B) is calculated for each time step in line 34 with the `accuflux` function that routes rainfall plus melt water downhill over the local drain direction map.

The calculation of sampling statistics (Eq. (2c)) and visualisation of model results requires the map data for all time steps and Monte Carlo samples. As this is typically a number of gigabytes of data, these data need to be stored to hard disk in Eqs. (2a) and (2b) and read from hard disk again in Eq. (2c). This is done in the script with the `self.report` function (used in lines 20, 35 and 36). Depending on the method in which it is used, it either stores a single map (when used in the `initial` method, e.g. in line 20) or a time series of maps (in the `dynamic`, e.g. lines 35 and 36). This is done for each Monte Carlo sample when these methods are used in a `MonteCarloFramework`. Variables are stored using rules for file names defined by the framework: numbered filename suffixes and directory names represent time steps and Monte Carlo samples, respectively. The same rules are used in visualisation routines and functions that read files from disk, such as the functions calculating sampling statistics explained below.

The `postmcloop` method contains functions to calculate sampling statistics from the ensemble map data written to disk. It represents Eq. (2c). The functions `mcaveragevariance` and `mcpercentiles` calculate mean, variance and percentiles of the file names defined in line 39, snow pack (`s`) and discharge (`q`). These sampling statistics are calculated for the time steps provided by the last argument in these functions. Here, the last argument is `self.timesteps()` returning a list containing all time step numbers. As a result, the sampling statistics are calculated here for all time steps. The `mcpercentiles` function takes the list `percentiles` defining the percentiles that need to be calculated. Results are stored using file names defined by the framework.

2.4. Visualisation routines and results of the model

The Aguila visualisation tool (Pebesma et al., 2007) is integrated with the framework. It allows prompt visualisation of model inputs and outputs without conversion because it reads map data from

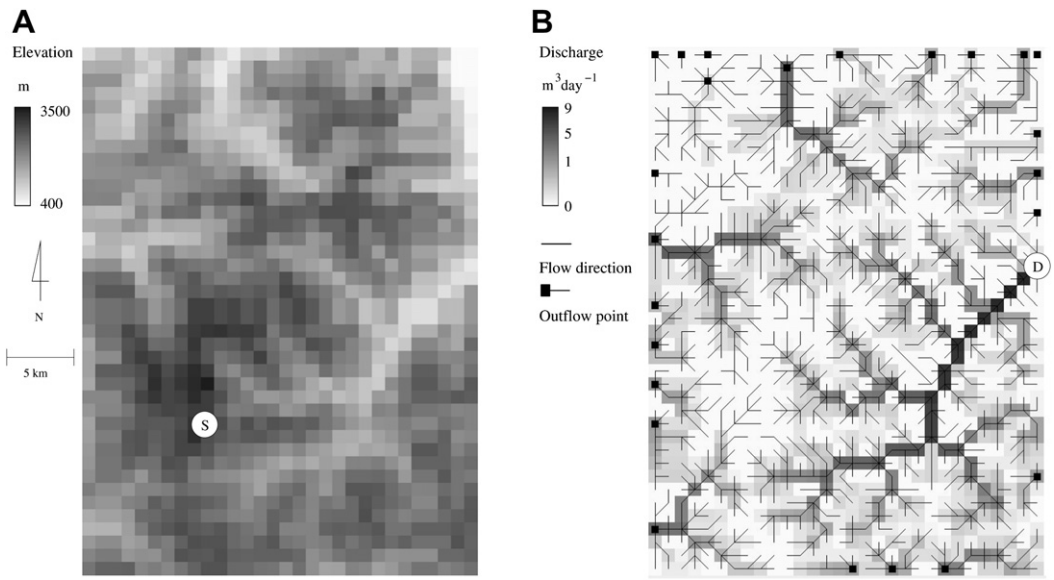

Fig. 1. (A) Digital elevation model (`dem`, m), (B) discharge (`discharge`, m^3/day) at time step 157 (days) with superimposed local drain direction network (ldd). The local drain direction network contains flow directions to the steepest down slope neighbour. The marked cells represent the snow pack and the discharge measurement locations, indicated by S and D on the left and right panel, respectively.

disk using rules for file names defined by the modelling framework. Spatial data are shown in map views. When stochastic data are visualised, Aguila shows a map view of a variable at a user defined percentile value. To visualise the full distribution of a variable at a location on the map, Aguila can create a cumulative probability distribution plot. In the case of temporal data, map views or cumulative probability distribution views can be animated through time. In addition, time series plots can be created containing the (percentile) value of a variable at a particular cell. Fig. 2 shows how Aguila visualises the snow pack. The top panels show the run with the stochastic model, the second row of panels represent a run with data assimilation, which will be described in the next sections. Aguila was started with the percentile data as input, written to disk by the function `mcpercentiles` as explained above. Aguila visualises the results of the model run representing a period between autumn and spring. The top right panel is a time series of the snow pack (m), showing the median value for a location selected on the map (top left panel). The panel in the top centre shows the cumulative probability distribution of the snow pack for the same location and the selected time step (day 145). As the Aguila software is interactive, the user can browse the map to show time series or cumulative probability distributions of other locations. In addition, the player window (bottom) can be used to steer the animation while locations, time steps or percentile values can be selected in the data window (bottom right).

The modelled median of the snow pack is greater than the observed snow pack, as illustrated by the time series in Fig. 3A. This is because the model uses a mean lapse rate ($l = -0.005\ ^\circ\mathrm{C\,m^{-1}}$) that is greater than the lapse rate used to generate the observational data ($l = -0.004\ ^\circ\mathrm{C\,m^{-1}}$), resulting in too much precipitation that falls as snow. This also results in an underestimation of discharge for most time steps in the period of snow accumulation (time steps 1–150), as shown in Fig. 4A and D. In the melting season (time steps 150–180), the model overestimates discharge most of the time, because too much snow is available for melting.

The width of the confidence interval of the snow pack increases with time (Fig. 3A and E). This is partly due to the accumulation of error introduced by the uncertainty in precipitation, which has an error for each time step independent of the other time steps. Also, the uncertainty in snow pack during the melting season at the end of the run is large, because snowmelt is dependent on the lapse rate having a large uncertainty.

3. General theory and framework for data assimilation

In sequential data assimilation, the model Eq. (1) is updated at time indices when observational data are available, referred to here as update moments. We give a short outline of the basic data assimilation formulations here. For a more extensive explanation the reader is referred to Doucet et al. (2001) and Simon (2006). Data assimilation is mostly done with observations of the state variables $\mathbf{z}_t$. In some cases, observations of model inputs $\mathbf{i}_t$ and parameters $\mathbf{p}_t$ are also assimilated. Let $\mathbf{x}_t$ ($t = 1, 2, \ldots, T$) be a vector of model components for which observations are available. It is a subset of $\mathbf{z}_t$, $\mathbf{i}_t$ and $\mathbf{p}_t$. Let $\mathbf{y}_t$ be a vector containing the corresponding instantaneous observation. It is defined as:

$$\mathbf{y}_t = \mathbf{H}_t(\mathbf{x}_t) + \mathbf{v}_t \tag{7}$$

for each update moment t when observations are available. In Eq. (7), $\mathbf{H}_t$ is the measurement operator that transforms the model state to the observation, and $\mathbf{v}_t$ is a zero-mean vector representing measurement error. Let $\mathbf{Y}_t$ be all past and current observations at time index t. A data assimilation filter estimates the conditional probability density function $p(\mathbf{x}_t|\mathbf{Y}_t)$. Each update moment it evaluates Bayes's formula:

$$p(\mathbf{x}_t|\mathbf{Y}_t) = p(\mathbf{y}_t|\mathbf{x}_t)p(\mathbf{x}_t|\mathbf{Y}_{t-1})/p(\mathbf{y}_t) \tag{8}$$

In (8), $p(\mathbf{x}_t|\mathbf{Y}_t)$ is the posterior probability density function of $\mathbf{x}_t$ at t, $p(\mathbf{x}_t|\mathbf{Y}_{t-1})$ is the prior probability density function at t calculated with Eq. (1). For time indices for which no observations are available, Eq. (1) is used to calculate $p(\mathbf{x}_t|\mathbf{Y}_t)$. Most approaches solve Eq. (8) using a Monte Carlo computational method. We have implemented the Particle Filter (e.g., Simon, 2006; Weerts and El Serafy, 2006) and the Ensemble Kalman Filter (e.g., Evensen, 2003). Eq. (8) retrieves the posterior probability density function of the model components for which observations are available only. However in most cases it is required to retrieve the posterior of all model components ((p($\mathbf{z}_t$|$\mathbf{Y}_t$),

$p(\mathbf{i}_t|\mathbf{Y}_t)$, and $p(\mathbf{p}_t|\mathbf{Y}_t)$) as these are required to evaluate Eq. (1) for time steps without observations. These posteriors are calculated by both filters implemented, although the Ensemble Kalman Filter requires additional solution procedures, such as state augmentation (Hendricks Franssen and Kinzelbach, 2008), as will be used here. The filters are implemented in the software framework by adding an extra loop to the scheme in Eq. (2):

for each period : (9a)

for each n: (9b)

for each t in period : (9c)

$$\mathbf{z}_t^{(n)} = \mathbf{f}_t^{(n)}(\mathbf{z}_{t-1}^{(n)}, \mathbf{i}_t^{(n)}, \mathbf{p}_t^{(n)})$$

evaluate Eq. (8)

for each t: (9d)

$$\mathbf{Z}_t = \mathbf{g}(\mathbf{z}_t^{(1,\ldots,N)}), \quad \mathbf{I}_t = \mathbf{g}(\mathbf{i}_t^{(1,\ldots,N)}), \quad \mathbf{P}_t = \mathbf{g}(\mathbf{p}_t^{(1,\ldots,N)})$$

Each update moment is associated with a period consisting of all time indices after the previous filter moment up to and including the time index of the respective update moment itself. For each period, the model is run in Monte Carlo mode. At the end of the period, after executing all time steps, the filter is applied.

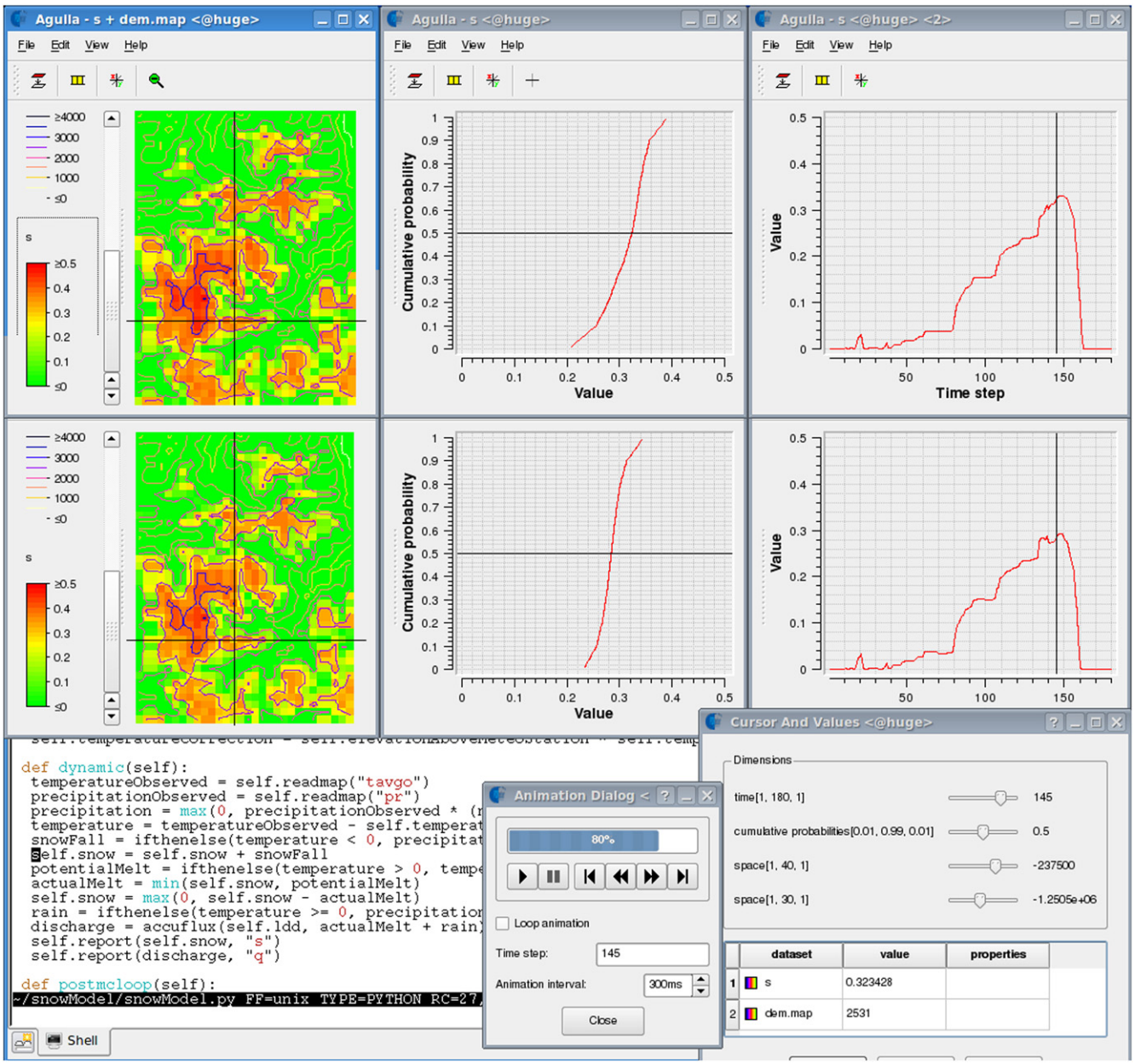

Fig. 2. Screenshot showing Aguila visualisations of the Monte Carlo (upper three windows) and the Particle Filter (lower three windows) modelled snow pack (snow, m). Left, map view; centre, cumulative probability density function; right, time series. The location, time step and percentile value for which results are shown are interactively selected in the plots or in the cursor window (bottom right). All windows show results for that cursor location in the spatial, temporal and stochastic dimension. Here, we selected the location shown by the cross in the left panels, the time step 145 (days) shown by the vertical line in the right panels and the percentile value of 0.5 (median) shown in the centre panels. All panels can be animated over time with the Animation Dialog (bottom centre panel).

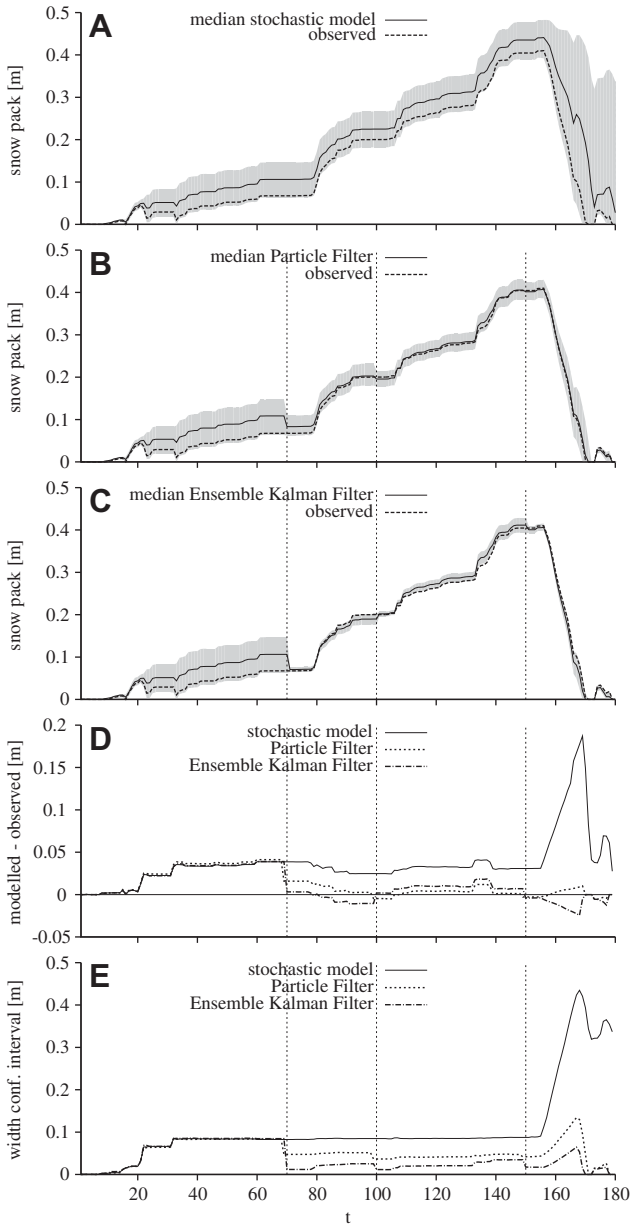

Fig. 3. Snow pack ($\mathbf{a}(\mathbf{s})_t$, m) at the location indicated in Fig. 1, left panel. (A) Stochastic model; (B) Particle Filter; (C) Ensemble Kalman Filter; solid line, median; grey area, values between 10th and 90th percentile; dotted line, observed (from artificial data set). (D) Modelled median value minus observed. (E) Width of confidence interval (90th percentile value minus 10th percentile value). Vertical dotted lines indicate update moments.

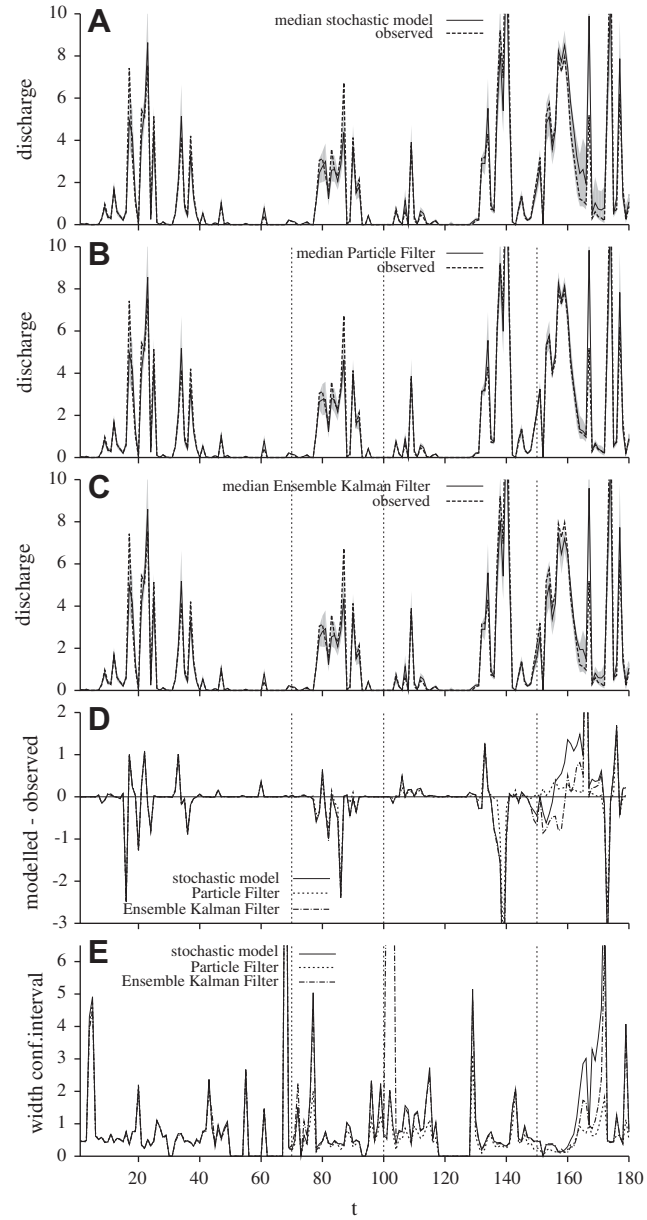

Fig. 4. Discharge ($\mathbf{q}(\mathbf{s})_t$, m^3/day) at the location indicated in Fig. 1, right panel. (A) Stochastic model; (B) Particle Filter; (C) Ensemble Kalman Filter; solid line, median; grey area, values between 10th and 90th percentile; dotted line, observed (from artificial data set). (D) Modelled median value minus observed value (m^3/day). (E) Width of confidence interval (90th percentile value minus 10th percentile value, m^3/day). Vertical dotted lines indicate update moments.

4. Particle filter

4.1. Theory

The Particle Filter approximates the posterior probability density function in Eq. (8) by the collection of Monte Carlo samples (i.e., particles), assigning a weight to each sample:

$$p(\mathbf{x}_t|\mathbf{Y}_t) \approx \sum_{n=1}^{N} p_t^{(n)} \delta(\mathbf{x}_t - \mathbf{x}_t^{(n)}) \tag{10}$$

The weights $p_t^{(n)}$, also referred to as probability masses, sum to one. In Eq. (10), $\delta(\,)$ denotes the Dirac delta function. For Gaussian measurement error $\mathbf{v}_t$, the weights are proportional to (Simon, 2006; Chin et al., 2007):

$$\left.\begin{aligned} a_t^n &= \exp(-[\mathbf{y}_t - \mathbf{H}_t(\mathbf{x}_t^{(n)})]^{\mathrm{T}}\mathbf{R}_t^{-1}[\mathbf{y}_t - \mathbf{H}_t(\mathbf{x}_t^{(n)})]/2) \\ p_t^n &\propto a_t^n \end{aligned}\right\} \tag{11}$$

where $\mathbf{R}_t$ is the covariance matrix of the measurement error $\mathbf{v}_t$. The weights are calculated by normalization of $a_t^{(n)}$:

$$p_t^{(n)} = a_t^{(n)} / \sum_{j=1}^{N} a_t^{(j)} \tag{12}$$

When the observation errors are uncorrelated the off-diagonal elements of $\mathbf{R}_t$ are zero and Eq. (11) is equivalent to the equations

provided by van Leeuwen (2003) and Weerts and El Serafy (2006). An example of the weights is provided in Fig. 5A. Next, a new set of N Monte Carlo samples is created that consists of exact copies of a subset of the Monte Carlo samples in the prior probability density $p(\mathbf{x}_t|\mathbf{Y}_{t-1})$. This step is referred to as resampling. As the weights in Eq. (12) can be used to calculate the posteriors of all model components ($p(\mathbf{z}_t|\mathbf{Y}_t)$, $p(\mathbf{i}_t|\mathbf{Y}_t)$, and $p(\mathbf{p}_t|\mathbf{Y}_t)$), exact copies of a subset of Monte Carlo samples in the prior probability density of all model components can be made to retrieve the posteriors of all model components. A number of different approaches exist to do the resampling step. Each approach represents the intuitive idea that the number of copies of a sample in the prior probability density should be approximately proportional to $p_t^{(n)}$. In Sequential Importance Resampling or SIR for short (e.g., Gelman et al., 2004), a cumulative distribution function is constructed from the weights $p_t^{(n)}$ (Fig. 5B). From this distribution, N samples are randomly drawn with replacement using a uniform distribution between zero and one. This draw of samples represents the posterior probability distribution of model states $p(\mathbf{x}_t|\mathbf{Y}_t)$ and all other model components (Fig. 5C). Residual Resampling (RR, Liu and Chen, 1998; Weerts and El Serafy, 2006) copies samples in two steps. In the first step, a sample is copied a number of times equal to $k_t^{(n)} = \text{round}(p_t^{(n)} \cdot N)$, where round($x$) is a function that rounds to the nearest integer towards zero. In the second step, residual weights $r_t^{(n)}$ are calculated:

$$r_t^{(n)} = \frac{p_t^{(n)} \cdot N - k_t^{(n)}}{N - \sum_{n=1}^{N} k_t^{(n)}} \tag{13}$$

A number of $N - \sum_{n=1}^{N} k_t^{(n)}$ additional copies of samples is made using the residual weights $r_t^{(n)}$. This is done in a similar way as in SIR by uniform sampling from a cumulative distribution function constructed now from the residual weights $r_t^{(n)}$. The samples copied in step one and two result again in N samples representing the posterior probability density of all model components.

4.2. Software framework

In order to illustrate how particle filtering can be done with the framework, snow pack data will be assimilated into the snowmelt model using sequential importance resampling (SIR). We mimic the availability of snow pack data from remote sensing or field observations by using the artificial observational data set generated with the model (see Section 2.2). This data set contains a map of snow pack $\mathbf{a}(\mathbf{s})_t$ for each time step. In principle this map could directly serve as observational data $\mathbf{y}_t$ (Eq. (7)). However, the covariance matrix $\mathbf{R}_t$ (Eq. (11)), in particular the non-diagonal elements, cannot be estimated here because the spatial correlation structure of the measurement error at the support of the grid cells of the model is unknown. We circumvent this problem by assimilating snow pack data at a larger spatial support under the assumption that the errors at a larger support can be considered independent. Five elevation zones with an equal area are created. For each time step, the snow pack observations are averaged over each area, resulting in five snow pack observations that are used as observational data $\mathbf{y}_t$ with associated errors $\mathbf{v}_t$. The standard deviation of the errors $\mathbf{v}_t$ is assumed to be 40% of the observed average snow pack in the area and provide the variances in $\mathbf{R}_t$. The covariances in $\mathbf{R}_t$ are set to zero representing independent errors.

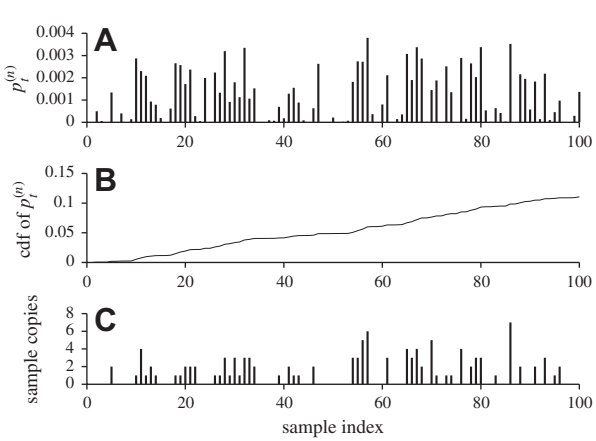

Fig. 5. Calculation of posterior state at update moment with Sequential Importance Sampling, first update moment. (A) Weights, $p_t^{(n)}$; (B) Cumulative distribution function of the weights; (C) Number of copies per sample. X-axis: sample (particle) number. Plots show first 100 particles out of a total number of 1000 particles.

We designed the software framework such that Particle Filtering can be done by a small number of additions to a script for stochastic dynamic modelling. These additions consist of the methods `suspend`, `updateWeight`, and `resume` that are added at the bottom of the script (Table 2). Aside from initialising the `ParticleFilterModel` class, the content of the methods defined in the original snowmelt model (Table 1) does not need to be adjusted for Particle Filtering. Thus, the user can easily switch between running the model with or without Particle Filtering, simply by using another framework.

The `suspend`, `updateWeight` and `resume` methods are invoked, in this order, at an update moment. The line numbers below refer to these in Table 2. The `suspend` method (line 44) needs to contain `self.report` functions to store the realizations representing the prior probability density functions of the state variables $p(\mathbf{z}_t|\mathbf{Y}_{t-1})$ and these of the stochastic parameters $p(\mathbf{p}_t|\mathbf{Y}_{t-1})$. Here, this is the lapse rate and the snow pack (line 45 and 46). The `suspend` method stores these for each sample in directories referred to as filter directories. These are different from the directories containing the data written to disk in the `dynamic` method. This is because the filter directories will be copied in the resampling step. The subdirectories containing the data written to

Table 2
Bottom part of script for Particle Filtering.

```
44  def suspend(self):
45      self.report(self.temperatureLapseRate, "lapse")
46      self.report(self.snow, "s")
47
48  def updateWeight(self):
49      modelledData = self.readmap("s")
50      modelledAverageMap = areaaverage(modelledData,
        "zones.map")
51      observedAverageMap = self.readmap("avgObs", "observations")
52      observedStdDevMap = ifthenelse(observedAverageMap > 0,
        observedAverageMap * 0.4, 0.01)
53      sum = maptotal(((observedAverageMap − modelledAverageMap)
         ** 2)/(2.0 * (observedStdDevMap ** 2)))
54      weight = exp(sum)
55      weightFloatingPoint, valid = cellvalue(weight, 1)
56      return weightFloatingPoint
57
58  def resume(self):
59      self.temperatureLapseRate = self.read("lapse")
60      self.temperatureCorrection = self.elevationAboveMeteoStation
         * self.temperatureLapseRate
61      self.snow = self.read("s")
62
63  myModel = SnowModel()
64  dynamicModel = DynamicFramework(myModel, 180)
65  mcModel = MonteCarloFramework(dynamicModel, 1000)
66  pfModel = SequentialImportanceResamplingFramework(mcModel,
        [70,100,150])
67  pfModel.run()
```

disk in the `dynamic` need to be kept untouched as these are required to calculate the sampling statistics in the `postmcloop`.

Next, the framework executes the resampling step copying the filter directories a number of times, either using the RR or the SIR algorithm. Both algorithms need for each sample the value of $a_t^{(n)}$, a value proportional to the weight of each sample (Eq. 11). This value is calculated by the user-defined functions in the `updateWeight` method (line 48). Thus, the calculation of $a_t^{(n)}$ itself is not done by the framework as in some cases the user may want to use a weight function different from Eq. (11), for instance when the errors $\mathbf{v}_t$ are non-Gaussian (e.g., van Leeuwen, 2003). Here, the `updateWeight` method contains PCRaster functions on maps to read the modelled snow pack from disk and to average this snow pack over the five elevation zones (lines 49 and 50). Line 51 reads the observed snow pack data, which were already averaged over the zones. Line 52 calculates the standard deviation of the snow pack data. Eq. (11) is represented by lines 53 and 54. Finally, the method needs to return to the framework the value that is proportional to the weight (line 56). The framework does the normalization (Eq. (12)).

After the resampling, the filter directories contain the realizations representing the posterior probability density functions of the state variables $p(\mathbf{z}_t|\mathbf{Y}_t)$ and of the stochastic parameters $p(\mathbf{p}_t|\mathbf{Y}_t)$. To evaluate the transfer function Eq. (1) for the next time step, these realizations need to be read from disk. This is done in the `resume` method (line 58). Here, it reads the temperature lapse rate maps from disk and calculates the variable `temperatureCorrection` derived from this parameter, as was done in the initial. Also, the state variable `snow` is read from disk. Now, the framework invokes the dynamic method for the next time step, using `temperatureCorrection` and `snow`, calculated in `resume`, as input.

Line 66 defines the algorithm used for resampling, which in this example is SIR. The time indices corresponding to the update moment are given as constructor arguments.

5. Ensemble Kalman Filter

5.1. Theory

The Ensemble Kalman Filter is a Monte Carlo approximation of the Kalman filter (e.g., Evensen, 2003; Simon, 2006). The evaluation scheme is identical to the one given in Eq. (9), and evaluation of Eq. (8) is done by:

$$\mathbf{u}_t^{(n),+} = \mathbf{u}_t^{(n),0} + \mathbf{P}_t^0\mathbf{H}_t^{\mathrm{T}}(\mathbf{H}_t\mathbf{P}_t^0\mathbf{H}_t^{\mathrm{T}} + \mathbf{R}_t)^{-1} (\mathbf{y}_t^{(n)} - \mathbf{H}_t\mathbf{u}_t^{(n),0}), \quad \text{for each } n \tag{14}$$

In a standard Ensemble Kalman Filter, $\mathbf{u}_t^{(n)}$ is equal to $\mathbf{z}_t^{(n)}$, the realizations of all state variables in the model. It contains the state variables for which observations are available, referred to above as $\mathbf{x}_t^{(n)}$, and all other state variables. Unlike the Particle Filter, the standard Ensemble Kalman Filter does not provide the posterior of the parameters $p(\mathbf{p}_t|\mathbf{Y}_t)$ and the inputs $p(\mathbf{i}_t|\mathbf{Y}_t)$. A number of different approaches exist to find the posterior of these model components (Hendricks Franssen and Kinzelbach, 2008). In our case study we use state augmentation, which is a procedure that is also supported by the framework. In this procedure, the state vector $\mathbf{u}_t^{(n)}$ (Eq. (14)) is extended with the model components ($\mathbf{p}_t$ and/or $\mathbf{i}_t$) for which posterior probability density functions are required to be calculated. In Eq. (14), the superscript 0 in $\mathbf{u}_t^{(n)}$ indicates the prior state vector and superscript + indicates the posterior state vector calculated by the update. $\mathbf{P}_t^0$ is the covariance matrix of $\mathbf{u}_t^{(n),0}$ and $\mathbf{y}_t^{(n)}$ is a realization of $\mathbf{y}_t$ (Eq. (7)). The size of the vectors and matrices in Eq. (14) is $\mathbf{u}_t^{(n)}$, $k_t + l$; $\mathbf{P}_t^0$, $(k_t + l) \times (k_t + l)$; $\mathbf{H}_t$, $(k_t \times k_t + l)$; $\mathbf{R}_t$, $(k_t \times k_t)$; $\mathbf{y}_t^{(n)}$, k_t; with k_t being the number of observations at t. In the standard Ensemble Kalman Filter, l is the number of values in the state variables without observations. In the augmented filter, l is the number of values in the state variables, the parameters and inputs minus the number of observations. We have implemented the Ensemble Kalman Filter following Evensen (2003).

5.2. Software framework

In our case study, snow data are assimilated with the Ensemble Kalman Filter using state augmentation in order to include the temperature lapse rate in the update. For each update moment, the state vector $\mathbf{u}_t^{(n)}$ contains the values of the snow pack and the temperature lapse rate. The measurement operator $\mathbf{H}_t$ is used to convert the individual cell values of the snow pack to five average values, i.e. an average value for each of the five elevation zones. The stochastic dynamic modelling script can again be used with small modifications (Table 3). The user has to provide the content of three methods passing information on observations and state variables between the model and the Ensemble Kalman Filter framework. This information is in the form of matrices as used by NumPy (Oliphant, 2006; NumPy, 2009). The software framework includes functions to convert PCRaster maps to NumPy matrices and vice-versa. These can be used, for instance, to convert between state variables stored as PCRaster maps and state variables stored in a Numeric Python matrix.

The `setObservations` method (Table 3, line 44) is run once per update moment passing the observational data as matrices to the framework. The content of the matrices needs to be defined by the user (computations omitted in Table 3). The matrices that need to be passed include the matrix `realizationObs`, containing the realizations of the observations ($\mathbf{y}_t^{(n)}$ for all n), the matrix `covErrorObs`, the covariance matrix of the measurement error on the observations ($\mathbf{R}_t$), and the matrix `measurementOperator` ($\mathbf{H}_t$). Setting the `measurementOperator` (line 50) is optional. By default, the `measurementOperator` is a matrix with ones on the main diagonal and zeros elsewhere, applicable when the first values in $\mathbf{u}_t^{(n)}$ can directly (one-to-one) be mapped to the values in $\mathbf{y}_t^{(n)}$.

Table 3
Bottom part of script for Ensemble Kalman Filter.

```
44  def setObservations(self):
45      # left out: create realizationObs and covErrorObs
46      # pass the matrices to the filter framework
47      self.setObservedMatrices(realizationObs, covErrorObs)
48      # left out: create measurementOperator matrix
49      # pass the measurement operator matrix to the framework (optional)
50      self.setMeasurementOperator(measurementOperator)
51
52  def setState(self):
53      # left out: collect snow pack values and temp. lapse rate in stateVector
54      # pass the state vector to the framework
55      return stateVector
56
57  def resume(self):
58      # retrieve updated state vector for current sample
59      updatedStateVector = self.getStateVector(self.currentSampleNumber())
60      # left out: convert state vector to model variables containing the state
61
62  myModel = SnowModel()
63  dynamicModel = DynamicFramework(myModel, 180)
64  mcModel = MonteCarloFramework(dynamicModel, samples)
65  ekfModel = EnsKalmanFilterFramework(mcModel, [70, 100, 150])
66  ekfModel.run()
```

Left out parts indicated.

The `setState` method (Table 3, line 52) is run for each sample n for each update moment. It is used to pass the `stateVector` variable, representing $\mathbf{u}_t^{(n)}$, to the framework. The user defines this vector by converting state variables, parameters and/or inputs to a single vector. At each filter moment, the framework evaluates Eq. (14) using the information provided by the `setObservations` and `setState` methods. It returns a state vector with the posterior state of the model. This is done in the resume method (line 57). The user has to provide code to convert this state vector to model variables that are used in the main script of the model.

Where the Particle Filter deletes or copies entire samples, the Ensemble Kalman Filter *adjusts* the values of state variables, parameters, and/or inputs of each sample. This is done for each sample in place, in the folder containing the results for that sample. Thus, the Ensemble Kalman Filter does not create a separate folder structure with cloned (resampled) samples, as is the case in the Particle Filter.

6. Results and discussion of filters applied to the snow model

When a small number of particles is copied a large number of times in the Particle Filter, it may happen that the posterior probability density function of the model is represented by a too small number of different, unique, particles. This is known as particle collapse or impoverishment. In our model, this problem does not occur as can be seen in the plots created from files stored by the framework (Figs. 5 and 6). Each update moment, a relatively large number of samples is copied. As a result, a diverse population of samples remains to exist up to and including the posterior distribution at the last update moment (Fig. 6). Hence, employing 1000 samples appears to be sufficient for the model and data used. This is confirmed by a model run using 10,000 samples that gave probability density functions that were comparable to the results reported here. Our runs indicate that with a time varying noise in the input (here, precipitation), particle collapse is not a problem in the application of the particle filter. However, further evaluation of the particle filter is required under various conditions of stochastic inputs and model structures.

To evaluate the performance of the stochastic model without data assimilation, the particle filter, and the Ensemble Kalman Filter, Figs. 3 and 4 provide time series of snow pack and discharge. From these time series values, statistics were calculated. The mean squared error (MSE) is calculated as $\mathrm{MSE} = \sum_{t=1}^{T}(\alpha_t - \alpha_{\mathrm{obs},t})^2/T$, with α_t, the median of the modelled variable (either snow pack or discharge) at time step t and $\alpha_{\mathrm{obs},t}$ the observed value at t (artificial data set). The mean width of the 80% confidence interval (MW) is $\mathrm{MW} = \sum_{t=1}^{T}(P_{90,t} - P_{10,t})/T$, with $P_{90,t}$, the 90th percentile value of the modelled variable at time step t and $P_{10,t}$, the 10th percentile value at t. The results are provided in Table 4. The table shows that the data assimilation techniques reduce the MSE and MW compared to the run without data assimilation. The reduction is largest for the snow pack, resulting in an MSE value that is 17% of the MSE value of the stochastic model. The Particle Filter and the Ensemble Kalman Filter use different methods to represent the Bayesian update. However, the results in terms of MSE and MW are comparable (Table 4). It should be noted that the results in Table 4 are not representative for performance of the two schemes in general terms, as performance depends on a number of factors not studied here, including the number of samples (particles), the linearity of the model and the statistical properties of the errors.

In the data assimilation runs, the width in the 80% confidence interval for snow pack reduces at update moments (Fig. 3), particularly at $t = 70$. Also, the filters are capable of reducing the large error in snow pack estimates during spring observed in the run without data assimilation, as can be seen by comparing the last part (the spring period) of the time series in Fig. 3A and B and this part of the time series shown in Fig. 3E. As a result, the 80% confidence intervals of discharge in this period also become narrower (Fig. 4A, B and E) as most discharge in the spring is generated from snowmelt.

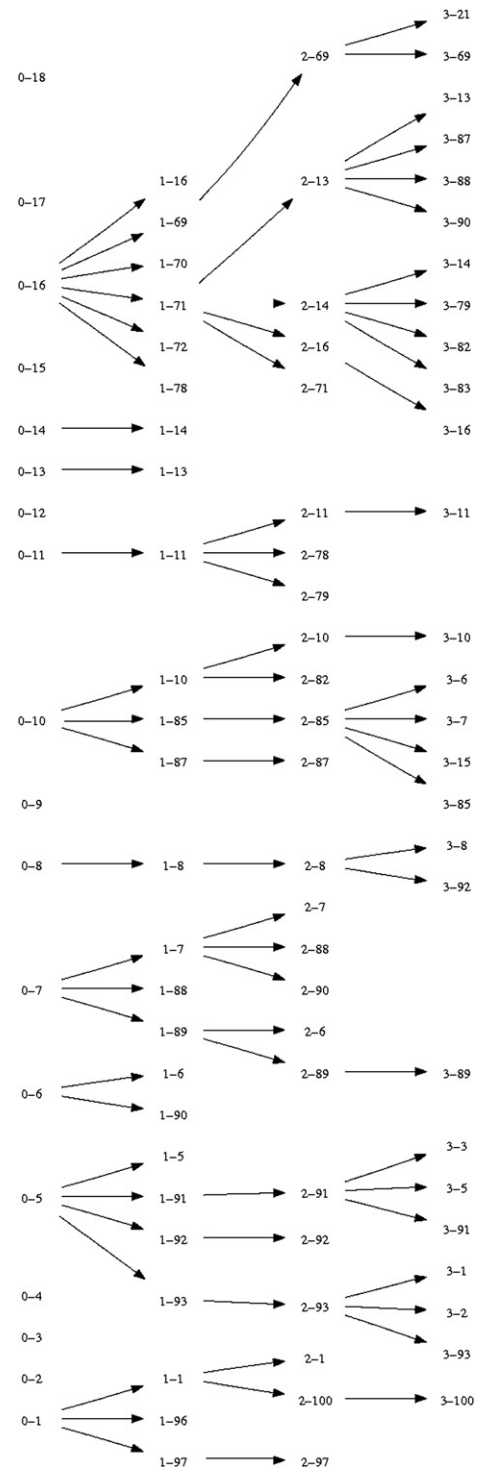

Fig. 6. Resampling of samples at the three update moments, Sequential Importance Sampling, snowmelt model. Arrows indicate copies of samples at update moments. Samples are represented by two numbers (e.g. 2 and 85), the first number is the update moment and the second number is the unique sample number. Only subset of samples shown.

Table 4
Mean squared error (MSE) and mean width of confidence interval (MW) for snow pack $\mathbf{a}(\mathbf{s})_t$ (m) and discharge $\mathbf{q}(\mathbf{s})_t$ (m^3/day).

	Snow pack		Discharge	
	MSE	MW (m)	MSE ($x10^{12}$)	MW ($x10^6$, m^2/day)
Stochastic model	0.00218 (100%)	0.105 (100%)	0.56 (100%)	0.74 (100%)
Particle Filter	0.00037 (17%)	0.051 (49%)	0.37 (66%)	0.56 (76%)
Ensemble Kalman Filter	0.00037 (17%)	0.036 (34 %)	0.42 (75%)	0.56 (76%)

The values are calculated for the locations indicated in Fig. 1. Values between brackets: value as a percentage of the corresponding value for the stochastic model.

As discussed in a previous section, Fig. 2 shows how the user can interactively evaluate different scenarios using the Aguila software. The panels in the first row show the results of the stochastic model while these in the second row show the results of the run with the Particle Filter. The panels in the centre can be used to compare the cumulative probability density function of the two runs. The panels show that the width of the probability density function for the Particle Filter is small compared to the width of the stochastic model confirming the results discussed above.

7. Data assimilation with external models

7.1. Framework concepts

The software framework provides a close integration between the definition of a model itself, i.e. the code describing the model equations, and the code to integrate observations using data assimilation. However, in some cases it is required to perform Monte Carlo simulation or data assimilation using an existing model. This is also supported by the software framework through functions that pass information from the software framework to the external model. The information that needs to be passed to the model includes the time steps of update moments, and directory names for storing model data according to the definitions of the software framework, using subdirectories for Monte Carlo samples. Below, we illustrate the possibility of calling external models with the LISFLOOD model Van der Knijf et al. (in press). The emphasis is here on the model and the results of Particle Filtering. For details on the use of the software framework when calling external models, the reader is referred to the manual (Schmitz et al., 2009).

7.2. Case study: discharge assimilation using the distributed rainfall-runoff model LISFLOOD

LISFLOOD is a spatially distributed, grid based rainfall-runoff model that has been developed for the simulation of hydrological processes in large European river basins. It is implemented using a combination of the PCRaster and the Python scripting language. The model was designed to facilitate the handling of large spatial data sets on soils, land cover, topography, and meteorology. LISFLOOD is driven by meteorological input time series and the simulated hydrological processes include snowmelt, infiltration, interception of rainfall, leaf drainage, evaporation and water uptake by vegetation, surface runoff, preferential flow, exchange of soil moisture between soil layers and drainage to the groundwater, sub-surface and groundwater flow, and flow through river channels. A more detailed description of LISFLOOD can be found in van der Knijf (forthcoming). In this case study we evaluate whether the assimilation of observed discharges using the Particle Filter can improve model output. This is especially important when model output derived from near-real time meteorological forcing is used as initial conditions in operational flood forecasting, as is the case for LISFLOOD, which is currently employed in the European Flood Alert System (Thielen et al., 2008).

The results of this case study are obtained using the Meuse catchment upstream of Borgharen, covering an area of approximately 21,000 km^2 (see Fig. 7). The Meuse catchment is situated in Belgium, France, and the Netherlands and is mainly fed by rain all year round with the highest flows occurring generally in winter and the lowest flows occurring during summer. The topography of the area has an elevation ranging from 50 to 700 m. To obtain all necessary soil and land use related parameters we employ the Soil Geographical Database of Europe (King et al., 1994), the HYPRES database on hydraulic soil properties (Wösten et al., 1999), and the CORINE Land Cover database (EIONET, 2009). The interpolated meteorological input grids were obtained using ordinary kriging with daily observations of approximately 23 stations from the Meteorological Archiving and Retrieving System (Rijks et al., 1998). The remaining parameters have been estimated by a previous calibration of the system on discharge at the Borgharen gauging station. The model setup employed uses a 5-km grid resolution and

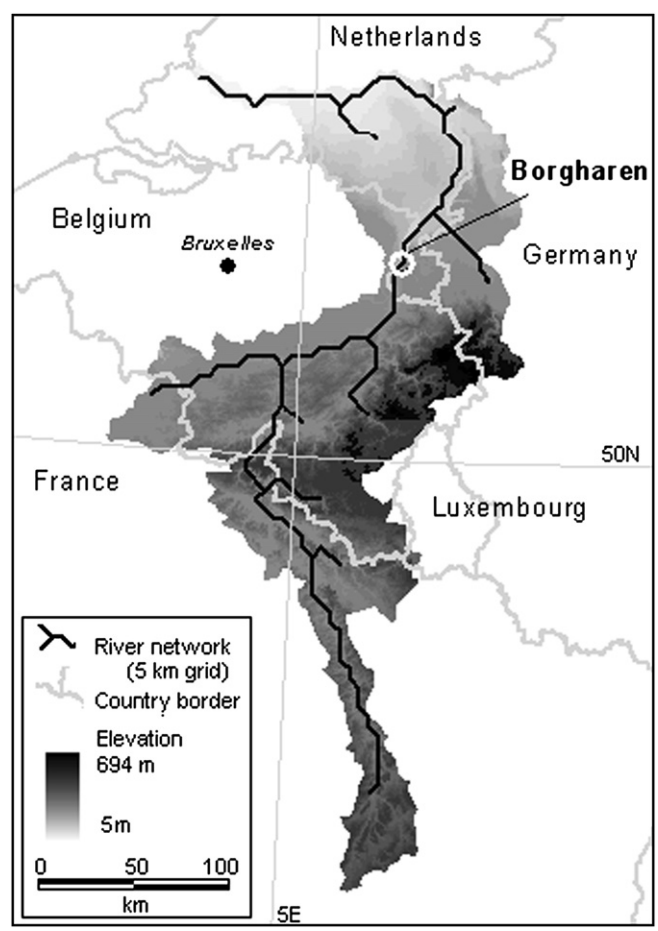

Fig. 7. Overview of the Meuse catchment with gauging station Borgharen (from Feyen et al., 2007).

a daily time step. Particle Filtering was performed using daily-observed discharges for the Borgharen gauging station for the period from 1/12/1993 to 30/04/1994.

Data assimilation using the Particle Filter requires the explicit specification of the measurement error model in order to calculate and update particle weights as shown in Eqs. (11)–(13). Most of the streamflow measurement error models assume a Gaussian distribution having zero mean and a heteroscedastic variance due to the nonlinear nature of the rating curves used to transform water levels into discharges (Thiemann et al., 2001). Here, we employ a non-parametric approach to estimate the error deviation as suggested by (Vrugt et al., 2005). An exponential function was fit through nine years of observed discharge data, which was then employed to derive the variance of the streamflow measurement error at each time step as a function of observed streamflow. In this case study we assume that precipitation is the largest source of uncertainty in modelling hydrological processes. Hence we perturb the observed precipitation grids according to Eq. (3) using a uniform distribution with an error range of $\pm 30\%$. As nowadays usually observed discharge values are available with a high frequency in real time we assimilate discharge for each day using a total of 100 particles. Similar to the snow model, the realizations of the state variables need to be stored in the filter directories, which for the case of LISFLOOD results in a total of twelve state variables maps for each filter moment.

Fig. 8 presents the results of the data assimilation using the SIR algorithm. A general improvement in modelled discharges in comparison to the simulated discharge without data assimilation can be observed. Especially, in phases with strong rainfall the Particle Filter clearly improves model output and the 95 percentile confidence interval embraces well the measured discharges. However, there are still periods where the model is not capable to reproduce the observed discharge properly. This is especially significant for periods not influenced by strong precipitation events, i.e., during the decrease in discharge after high flow periods or during low flow periods. During these stages the mismatch between modelled and observed discharges is mainly caused by model structural errors, which originate from an inadequate representation of the hydrological processes in the model. Including these uncertainties into the data assimilation would further improve model output. However, the proper treatment of model structural errors in hydrological modelling and its implementation in data assimilation techniques is not a trivial task and is beyond the scope of this paper. Nevertheless, the presented results illustrate that Particle Filtering is a valuable tool to merge various sources of data and their corresponding uncertainties into distributed hydrological models and therewith improve model output.

8. Discussion and conclusions

We have built a software framework that integrates a framework for model construction and routines for visualisation of model data. The software framework for model construction includes a large set of spatial operations on raster maps. These operations can be used in various framework classes that provide control flow for a number of different modelling approaches and activities: static modelling, spatio-temporal modelling, deterministic modelling, stochastic modelling, and data assimilation. The framework makes it fairly easy for the modeller to switch between the different modelling approaches as each of the framework classes use highly similar methods. As a result, model code can be reused or even copied without any changes when switching between modelling frameworks. It is for instance possible to switch from Particle Filtering to Monte Carlo mode (no data assimilation) by just selecting the Monte Carlo framework class at the bottom of the script. The software framework comes with routines for visualisation of stochastic spatio-temporal data read and written by models running in the framework. These routines allow prompt visualisation of data without conversion of data formats or configuration of displays. The software framework makes it fairly easy to explore and evaluate alternative process representations because changing the code of a model can be done by recombining high level functions on raster maps while evaluation of model inputs and outputs is possible with the integrated visualisation routines. As data assimilation techniques are integrated in the framework, the integration of data using advanced data assimilation techniques can be done in relatively little time. To our knowledge this is the first software framework that closely integrates construction of spatio-temporal stochastic models, data assimilation, and visualisation of spatio-temporal stochastic data.

It is preferable to perform Monte Carlo simulation or data assimilation with models that are constructed with the framework itself, as this guarantees seamless integration of the model and the framework. However, it is also possible to call external models, as shown in the case study with LISFLOOD. This can be done with any external model, given that the external model can be called from a command line. Also, the model needs to be capable of directing output data to directories containing results for the individual Monte Carlo samples, as specified by the framework. Alternatively small programs can be written that redirect the data. In the case of

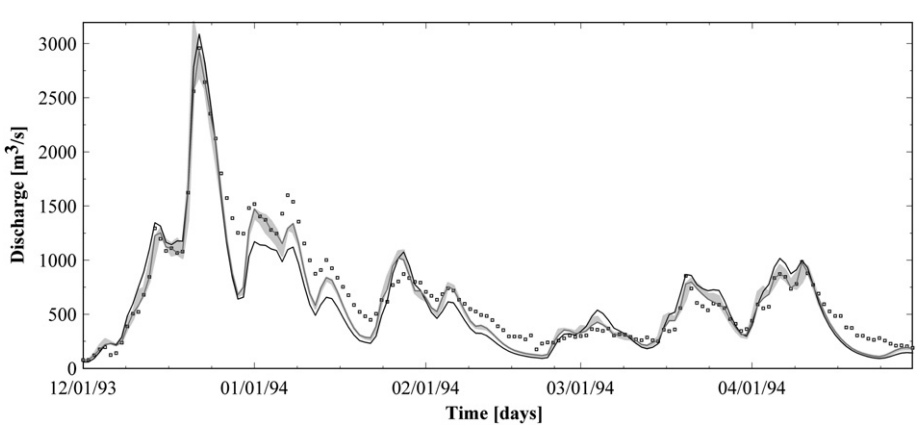

Fig. 8. Hydrograph at the Borgharen gauging station for the data assimilation period (12/01/1993–04/30/1994). Daily observed discharges are represented by squares. The black solid line denotes simulated discharge without data assimilation. The light grey shaded area denotes the prediction uncertainty (95% confidence interval) resulting from particle filtering with precipitation uncertainty. The dark grey shaded solid line depicts the simulated discharge of the particle with the highest weight during sequential data assimilation.

data assimilation, the external model also needs to be capable of being interrupted at update moments. At an update moment, the model needs to stop its execution, store its prior state to disk, and restart from disk reading the posterior state from disk again. In the LISFLOOD model, a Python wrapper redirects model input and output and modifies start and end time steps according to the filter periods.

The Particle Filter and the Ensemble Kalman Filter give comparable results when applied to the example snowmelt model. In a certain way this confirms the reliability of the filter techniques, because similar results are found by two different approaches to solve the Bayesian update equation. While the Ensemble Kalman Filter modifies the state vector of each Monte Carlo sample at the update moment, the Particle Filter leaves the state vector of each Monte Carlo sample unchanged, as it represents the Bayesian update by copying entire Monte Carlo samples. As a result, the problem of violation of the conservation of mass, momentum, or energy encountered sometimes with the Ensemble Kalman Filter does not occur with the Particle Filter. This is an advantage of the Particle Filter. The difference between the methods in how the Bayesian update equation is solved has also its consequences for the run time of a data assimilation procedure. In our implementation of the Ensemble Kalman Filter, the update equation involves an evaluation of matrices that can become very large in the case of a large number of observations. The Particle Filter does not have this problem because the update equation is solved in a computationally less intensive way. However, the Particle Filter is assumed to require a large number of Monte Carlo samples. A review and comparison of the filtering techniques implemented in our framework is provided by e.g. Evensen (2003), van Leeuwen (2003). In our opinion the Particle Filter needs further attention as its application in environmental modelling is relatively recent, and because it is promising, also because of its relatively simple approach to solve the Bayesian update.

The software framework presented in this paper can be extended by other components for data assimilation and calibration. An example of such a component is the AMORI software for calibration of spatio-temporal models using genetic algorithms (AMORI, 2009).

Acknowledgements

Cees Wesseling and Willem van Deursen (PCRaster Environmental Software) are thanked for their inputs in the development of the PCRaster Python software. This research was funded by 'Ruimte voor Geo-Informatie', project 'On-line coupling of spatial optimization tools and spatially distributed simulation models', RGI 313.

References

Ajami, N.K., Duan, Q., Sorooshian, S., 2007. An integrated hydrologic Bayesian multimodel combination framework: Confronting input, parameter, and model structural uncertainty in hydrologic prediction. Water Resources Research 43 (1).

AMORI, Februari 2009. Automatic Model Optimization Reference Implementation Available from: http://sourceforge.net/projects/amori.

Benenson, I., Torrens, P.M., 2004. Geosimulation: Automata-based Modeling of Urban Phenomena. Wiley, Chichester.

Beven, K.J., 2002. Rainfall-runoff Modelling The Primer. Wiley, Chichester.

Blandford, T.R., Humes, K.S., Harshburger, B.J., Moore, B.C., Walden, V.P., Ye, H., 2008. Seasonal and synoptic variations in near-surface air temperature lapse rates in a mountainous basin. Journal of Applied Meteorology and Climatology 47 (1), 249–261.

Blöschl, G., Reszler, C., Komma, J., 2008. A spatially distributed flash flood forecasting model. Environmental Modelling and Software 23 (4), 464–478.

Breukers, A., Kettenis, D.L., Mourits, M., van der Werf, W., Oude Lansink, A., 2006. Individual-based models in the analysis of disease transmission in plant production chains: An application to potato brown rot. Agricultural Systems 90 (1–3), 112–131.

Brown, L., Thorne, R., Woo, M.K., 2008. Using satellite imagery to validate snow distribution simulated by a hydrological model in large northern basins. Hydrological Processes 22 (15), 2777–2787.

Bugs, December 2008. The BUGS Project, Available from: http://www.mrc-bsu.cam.ac.uk/bugs/.

Burrough, P.A., 1998. Dynamic Modelling and Geocomputation. In: Longley, P.A., Brooks, S.M., McDonnel, R., MacMillan, B. (Eds.), Geocomputation: A Primer. Wiley, Chichester, pp. 165–191.

Burrough, P.A., McDonnell, R.A., 1998. Principles of Geographical Information Systems. Oxford University Press, Oxford, UK.

Chen, M., Liu, S., Tieszen, L.L., Hollinger, D.Y., 2008. An improved state-parameter analysis of ecosystem models using data assimilation. Ecological Modelling 219 (3–4), 317–326.

Chin, T.M., Turmon, M.J., Jewell, J.B., Ghil, M., 2007. An ensemble-based smoother with retrospectively updated weights for highly nonlinear systems. Monthly Weather Review 135 (1), 186–202.

Clark, M.P., Slater, A.G., Barrett, A.P., Hay, L.E., McCabe, G.J., Rajagopalan, B., Leavesley, G.H., 2006. Assimilation of snow covered area information into hydrologic and land-surface models. Advances in Water Resources 29 (8), 1209–1221.

COSTA, Januari 2008. Common Set of Tools for Assimilation of Data. Available from: http://www.costapse.org.

Doucet, A., de Freitas, N., Gordon, N., 2001. Sequential Monte Carlo Methods in Practice. Springer, New York.

EIONET, January 2009. The European Topic Centre Land Use and Spatial Information ETC-LUSI: Corine Land Cover Database 2000–100m. Available from: http://etc-lusi.eionet.europa.eu/.

ESRI, January 2009. Environmental Systems Research Institute, Available from: http://www.esri.com.

Evensen, G., 2003. The Ensemble Kalman Filter: theoretical formulation and practical implementation. Ocean Dynamics 53 (4), 343–367.

Feyen, L., Vrugt, J.A., Nuallai•in, B.O., van der Knijff, J., De Roo, A., 2007. Parameter optimisation and uncertainty assessment for large-scale streamflow simulation with the LISFLOOD model. Journal of Hydrology 332 (3–4), 276–289.

Gelb, E. (Ed.), 1974. Applied Optimal Estimation. The MIT Press, Cambridge.

Gelman, A., Carlin, J.B., Stern, H.S., Rubin, D.B., 2004. Bayesian Data Analysis. In: Texts in Statistical Science, second ed. Chapman & Hall/CRC, Boca Raton.

Gimblett, R., Lynch, J., Daniel, T., Ribes, L., Oye, G., 2003. Deriving artificial models of visitors from dispersed patterns of use in the Sierra Nevada Wilderness, California. Journal for Nature Conservation 11 (4), 287–296.

GRASS, January 2009. GRASS GIS, Available from: http://grass.osgeo.org.

Grimm, V., Railsback, S.F., 2005. Individual-Based Modelling and Ecology. Princeton University Press, Princeton.

Groff, E.R., 2007. 'Situating' simulation to model human spatio-temporal interactions: an example using crime events. Transactions in GIS 11 (4), 507–530.

Helbing, D., Farkas, I., Vicsek, T., 2000. Simulating dynamical features of escape panic. Nature 407 (6803), 487–490.

Hendricks Franssen, H.J., Kinzelbach, W., 2008. Real-time groundwater flow modeling with the Ensemble Kalman Filter: joint estimation of states and parameters and the filter inbreeding problem. Water Resources Research 44 (9), W01419. doi:10.1029/2007WR006097.

Hill, M.C., Tiedeman, C.R., 2007. Effective Groundwater Model Calibration: With Analysis of Data, Sensitivities, Predictions, and Uncertainty. John Wiley & Sons, Hoboken, New Jersey.

Huang, S.L., Rich, P.M., Crabtree, R.L., Potter, C.S., Fu, P.D., 2008. Modeling monthly near-surface air temperature from solar radiation and lapse rate: application over complex terrain in Yellowstone National Park. Physical Geography 29 (2), 158–178.

Karssenberg, D., 2002. The value of environmental modelling languages for building distributed hydrological models. Hydrological Processes 16 (14), 2751–2766.

Karssenberg, D., Bridge, J.S., 2008. A three-dimensional numerical model of sediment transport, erosion and deposition within a network of channel belts, floodplain and hill slope: extrinsic and intrinsic controls on floodplain dynamics and alluvial architecture. Sedimentology 55, 1717–1745.

Karssenberg, D., De Jong, K., 2005a. Dynamic environmental modelling in GIS: 1. Modelling in three spatial dimensions. International Journal of Geographical Information Science 19 (5), 559–579.

Karssenberg, D., De Jong, K., 2005b. Dynamic environmental modelling in GIS: 2. Modelling error propagation. International Journal of Geographical Information Science 19 (6), 623–637.

Karssenberg, D., De Jong, K., 2006. Towards improved solution schemes for Monte Carlo simulation in environmental modeling languages. In: Oosterom, P.J.M., Kreveld, M.J. (Eds.), Geo-information and Computational Geometry. NCG Nederlandse Commissie voor Geodesie, Netherlands Geodetic Commission, Delft.

Karssenberg, D., Pfeffer, K., Visssers, M., 2006. Software tools for hydrological modelling. In: Giupponi, C., Jakeman, A.J., Karssenberg, D., Hare, M.P. (Eds.), Sustainable Management of Water Resources: An Integrated Approach. Elgar, Cheltenham, pp. 235–262.

King, D., Daroussin, J., Tavernier, R., 1994. Development of a soil geographical database from the soil map of the European Communities. Catena 21 (1), 37–56.

Ligtenberg, A., Wachowicz, M., Bregt, A.K., Beulens, A., Kettenis, D.L., 2004. A design and application of a multi-agent system for simulation of multi-actor spatial planning. Journal of Environmental Management 72 (1–2), 43–55.

Liu, J.S., Chen, R., 1998. Sequential Monte Carlo methods for dynamic systems. Journal of the American Statistical Association 93 (443), 1032–1044.
Marshall, S.J., Sharp, M.J., Burgess, D.O., Anslow, F.S., 2007. Near-surface-temperature lapse rates on the Prince of Wales Icefield, Ellesmere Island, Canada: Implications for regional downscaling of temperature. International Journal of Climatology 27 (3), 385–398.
MATLAB, December 2008. Available from: http://www.mathworks.com.
ModelMaker, July 2009. ModelMaker Tools, Available from: http://www.modelmakertools.com.
Moradkhani, H., 2008. Hydrologic remote sensing and land surface data assimilation. Sensors 8 (5), 2986–3004.
Moulin, B., Chaker, W., Gancet, J., 2004. PADI-Simul: An agent-based geosimulation software supporting the design of geographic spaces. Computers, Environment and Urban Systems 28 (4), 387–420.
Nagler, T., Rott, H., Malcher, P., Müller, F., 2008. Assimilation of meteorological and remote sensing data for snowmelt runoff forecasting. Remote Sensing of Environment 112 (4), 1408–1420.
Naud, C., Makowski, D., Jeuffroy, M.H., 2007. Application of an interacting particle filter to improve nitrogen nutrition index predictions for winter wheat. Ecological Modelling 207 (2–4), 251–263.
NumPy, Februari 2009. NumPy at sourceforge, Available from: http://sourceforge.net/projects/numpy/.
Oliphant, T.E., 2006. Guide to NumPy. Brigham Young University. Available from: http://www.tramy.us.
PCRaster, January 2009. PCRaster Internet Site, Available from: http://pcraster.geo.uu.nl.
Pebesma, E.J., de Jong, K., Briggs, D., 2007. Interactive visualization of uncertain spatial and spatio-temporal data under different scenarios: an air quality example. International Journal of Geographical Information Science 21 (5), 515–527.
PEST, December 2008. Available from: http://www.sspa.com/pest.
Pullar, D., 2004. SimuMap: a computational system for spatial modelling. Environmental Modelling and Software 19 (3), 235–243.
Python, January 2009. Python Programming Language, Available from: http://www.python.org.
ReBEL, March 2009. Recursive Bayesian Estimation Library, Available from: http://choosh.csee.ogi.edu/rebel/.
Rijks, D., Terres, J.M., Vossen, P. (Eds) 1998. Agrometeorological Applications for Regional Crop Monitoring and Production Assessment, Tech. Rep. EUR 17735 EN. Eur. Comm. Joint Res. Cent., Ispra, Italy.
Schmitz, O., Karssenberg, D., de Vries, L.M., March 2009. Framework for spatio-temporal stochastic modelling, data assimilation and model calibration. 40 pp. Available from: http://pcraster.geo.uu.nl.
Simon, D., 2006. Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches. Wiley-Interscience, Hoboken, New Jersey.
STELLA, January 2009. Available from: http://www.iseesystems.com.
Sydelko, P.J., Hlohowskyj, I., Majerus, K., Christiansen, J., Dolph, J., 2001. An object-oriented framework for dynamic ecosystem modeling: application for integrated risk assessment. Science of the Total Environment 274 (1–3), 271–281.
Thielen, J., Bartholmes, J., Ramos, M.H., De Roo, A., 2008. The European flood alert system – part 1: concept and development. Hydrology and Earth System Sciences Discussions 5 (1), 257–287.
Thiemann, M., Trosset, M., Gupta, H., Sorooshian, S., 2001. Bayesian recursive parameter estimation for hydrologic models. Water Resources Research 37 (10), 2521–2535.
Uppala, S.M., Kållberg, P.W., Simmons, A.J., Andrae, U., da Costa Bechtold, V., Fiorino, M., Gibson, J.K., Haseler, J., Hernandez, A., Kelly, G.A., Li, X., Onogi, K., Saarinen, S., Sokka, N., Allan, R.P., Andersson, E., Arpe, K., Balmaseda, M.A., Beljaars, A.C.M., van de Berg, L., Bidlot, J., Bormann, N., Caires, S., Chevallier, F., Dethof, A., Dragosavac, M., Fisher, M., Fuentes, M., Hagemann, S., Hólm, E., Hoskins, B.J., Isaksen, L., Janssen, P.A.E.M., Jenne, R., McNally, A.P., Mahfouf, J.F., Morcrette, J.J., Rayner, N.A., Saunders, R.W., Simon, P., Sterl, A., Trenberth, K.E., Untch, A., Vasiljevic, D., Viterbo, P., Woollen, J., 2005. The ERA-40 re-analysis. Quarterly Journal of the Royal Meteorological Society 131 (612), 2961–3012.
Van der Knijf, J.M., Younis, J., de Roo, A.D.P., A GIS-based distributed model for river-basin scale water balance and flood simulation. International Journal of Geographical Information Science, in press. doi: 10.1080/13658810802549154.
van Deursen, W.P.A., 1995. Geographical Information Systems and Dynamic Models. Koninklijk Nederlands Aardrijkskundig Genootschap/Faculteit Ruimtelijke Wetenschappen. Universiteit Utrecht, Utrecht.
van Deursen, W.P.A., Wesseling, C., Karssenberg, D., 2000. How do we gain control over GIS technology? In: Parks, B.O., Clarke, K.M., Crane, M.P. (Eds.), International Conference on Integrating Geographic Information Systems and Environmental Modeling: Problems, Prospectus, and Needs for Research. Boulder: University of Colorado – Cooperative Institute for Research in Environmental Sciences, Denver: US Geologic Survey – Center for Biological Informatics, and Boulder: NOAA National Geophysical Data Center - Ecosystem Informatics, Banff, Canada.
van Leeuwen, P.J., 2003. A variance-minimizing filter for large-scale applications. Monthly Weather Review 131 (9), 2071–2084.
Vrugt, J.A., Diks, C.G.H., Gupta, H.V., Bouten, W., Verstraten, J.M., 2005. Improved treatment of uncertainty in hydrologic modeling: Combining the strengths of global optimization and data assimilation. Water Resources Research 41 (1), 1–17.
Weerts, A.H., El Serafy, G.Y.H., 2006. Particle filtering and ensemble Kalman filtering for state updating with hydrological conceptual rainfall-runoff models. Water Resources Research 42 (9), W09403. doi:10.1029/2005WR004093.
Wesseling, C.G., Karssenberg, D., van Deursen, W.P.A., Burrough, P.A., 1996. Integrating dynamic environmental models in GIS: the development of a Dynamic Modelling language. Transactions in GIS 1, 40–48.
Wilkinson, S.N., Prosser, I.P., Rustomji, P., Read, A.M., 2009. Modelling and testing spatially distributed sediment budgets to relate erosion processes to sediment yields. Environmental Modelling and Software 24 (4), 489–501.
Wösten, J.H.M., Lilly, A., Nemes, A., Le Bas, C., 1999. Development and use of a database of hydraulic properties of European soils. Geoderma 90 (3–4), 169–185.