Supplement of Geosci. Model Dev., 13, 6093–6110, 2020
https://doi.org/10.5194/gmd-13-6093-2020-supplement

Supplement of

A distributed simple dynamical systems approach (dS2 v1.0) for computationally efficient hydrological modelling at high spatio-temporal resolution

Joost Buitink et al.

Correspondence to: Joost Buitink (joost.buitink@wur.nl)

dS2 manual

January 10, 2020

1 dS2 guide and example application

This document will give a short guide on how to setup the model, and how to run the model with some example data. This example will give 1 month of forcing data from the ERA5 reanalysis product for the Rhine basin. The model is described by Buitink et al. (2019).

1.1 Required input data

1.1.1 Forcing data

- `forc_precipitation.dat`
- `forc_evaporation.dat`
- `forc_temperature.dat`

The .dat files are Numpy memmap files, meaning that no metadata is stored within these files. See this link for more information and examples on how to use this dataformat. Essentially, these files contain 2D arrays with the following shape (*number_of_timesteps*, *number_of_pixels*). As these files do not contain metadata, it is important to enter the correct start and end timestamp of the forcing data in the model settings file (data_start and data_end).

Example of the transformation of forcing data, values indicate indices as x.y where `x` indicates the time index, and `y` the pixel index. Pixels with `nan` are pixels situated outside the catchment.

```
[[[nan, 1.1, nan],
  [1.2, 1.3, 1.4],
  [nan, 1.5, 1.6]],
 [[nan, 2.1, nan],
  [2.2, 2.3, 2.4],
  [nan, 2.5, 2.6]],
  ... ]
```

which is tranformed to

```
[[1.1, 1.2, 1.3, 1.4, 1.5, 1.6],
 [2.1, 2.2, 2.3, 2.4, 2.5, 2.6],
 ... ]
```

Note: radiation data is currently not included, in order to keep the example as clean as possible. Radiation data can easily be added when transformed into the same format as the other input data.

1.1.2 Static maps

- `map_catchment.asc`
- `map_distance.asc`
- `routing_info.pkl`

Catchment map is a 2D array indicating the location of the (sub-)catchments, with 'nan' indicating pixels outside the catchment. The number of pixels with a value should be equal to the *number_of_pixels* mentioned above.

The distance map indicates the distance from each pixel to the main catchment outlet. This can be created using `pcraster` for example.

The pickle object `routing_info.pkl` contains three objects in the following order:

1. `outletIndices`: a Python dictionary with the outlet IDs as keys, and the row/column index of the location of the outlet as value, according to the `catchment.asc`.
2. `allIndices`: a Python dictionary with the outlet IDs as keys, and all row/column indices where that (sub-)catchment is located as value.
3. `outletINFO`: a Python dictionary with the outlet IDs as keys, and a list with two items as value: the first item shows the ID of the downstream basin, and the distance from the outlet to the downstream outlet (["nan", 0] for the main outlet).

An example script is given in the `input` directory on how one can calculate these values.

1.1.3 Parameter maps

(Optional, can be defined in the model settings file)

- `par_alpha.asc`
- `par_beta.asc`
- `par_gamma.asc`

Parameter maps can be defined with the same shape and extend as the `map_catchment.asc` map. The value in each pixel represents the parameter value of that pixel. Parameter maps can currently be entered for the α, β, γ, ϵ parameters. Uniform parameter values can be entered in the settings file.

1.1.4 Initial conditions

(Optional, can be defined in the model settings file)

- `init_Qsim.asc`
- `init_Sstore.asc`

Initial maps can be entered for discharge and snow storage. Uniform initial conditions can be entered in the settings file.

1.2 Running the model

1.2.1 Preparing the Python environment

First we need to load the required libraries. The model code is located in the parent directory of this one, and we import the model settings from the Python file in this example directory.

[1]:
```
# Add the folder above to the Python path
import sys
sys.path.append("..")

# Import the model code and model settings
from dS2_model import dS2
from dS2_settings import ModelSettings
```

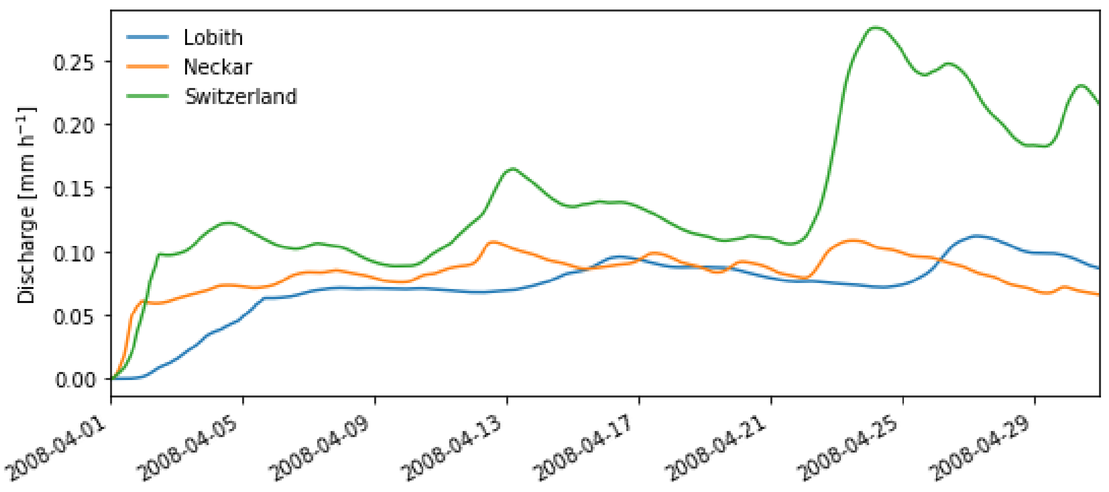
Now that we have loaded all the required libraries, we can create a model and run it. This can be done with the following commands.

[2]:
```
# Load the model settings
Settings = ModelSettings()

# Create the model
mod = dS2()

# Load the settings into the model
mod.model_setup_class(Settings)
# Generate the runoff for every pixel
mod.generate_runoff() # add this argument for a progressbar: progress=True
# Transport the generated runoff through the river network
mod.routing()
```

```
>>> Preparing input…
>>> Generating discharge…
>>> Routing…
```

When the model is done running, the results can easily be plotted with Python plotting libraries, such as matplotlib.

[3]:
```
%matplotlib inline
import matplotlib.pyplot as plt
from pandas.plotting import register_matplotlib_converters
register_matplotlib_converters()

xdate = mod.sim_period.to_pydatetime()

fig=plt.figure(figsize=(9, 4))
ax = fig.subplots(1)
ax.plot(xdate, mod.Qrout["1.0"], label="Lobith")
ax.plot(xdate, mod.Qrout["11.0"], label="Neckar")
ax.plot(xdate, mod.Qrout["13.0"], label="Switzerland")
ax.set_xlim([xdate[0], xdate[-1]])
ax.set_ylabel("Discharge [mm h$^{-1}$]")
ax.legend(frameon=False)
fig.autofmt_xdate()
```

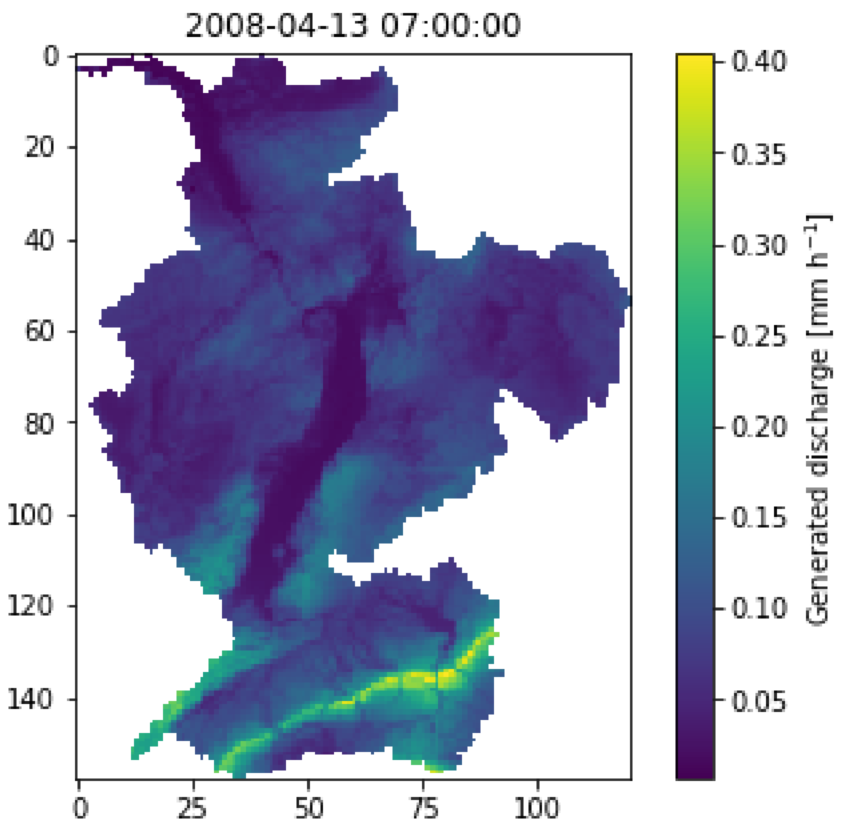


To make a map of the discharge generated at a specific timestep, the following code could be used. This code translates the 2D data (similar to the forcing) to a 3D array, respecting the shape of the catchment. Using the `output_Qsim` flag in the settings file, and the `dS2.export()` function, one can export the states in the model to a 3D NetCDF file.

[4]:
```
### Convert model states to NetCDF ###
# Check of the ``output_XXXX`` flags are set to true for the variables of␣
↪interest
# Before running the routing() function, call the export() function
# The NetCDF files should be created in the output directory
### Code example ###
# Settings = ModelSettings()
# Settings.output_Qsim = True
# mod = dS2()
# mod.model_setup_class(Settings)
# mod.generate_runoff()
# mod.export()
# mod.routing()

import numpy as np

# Copy the catchment map in order to get the shape of the catchment
map_qsim = mod.catchment.copy()

idx = 295

# Extract the values from the model (extract one row from the 2D array)
values = mod.Qsim[idx]
```

```
# Replace the non-NaN values with the values (translate the 1D array to a␣
↪spatial 2D array)
map_qsim[~np.isnan(map_qsim)] = values

# Plot the figure
fig = plt.figure(figsize=(6, 5), clear=True)
ax = fig.subplots(1)
ax.set_title(mod.sim_period[idx])
im = ax.imshow(map_qsim)
plt.colorbar(im, label="Generated discharge [mm h$^{-1}$]")
```

```
[4]: <matplotlib.colorbar.Colorbar at 0x26f43a377c8>
```


2008-04-13 07:00:00
0
20
40
60
80
100
120
140
0
25
50
75
100
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
Generated discharge [mm h$^{-1}$]