Ruhr-Universität Bochum
Institute of Hydrology, Water Resources Management
and Environmental Engineering
Prof. Dr. rer. nat. habil. A. Schumann

Using the HydPy framework to develop, improve, test, document, and share hydrological models, and to combine them in Delft-FEWS applications.

Christoph Tyralla (RUB → BCE)
Gordon Horn (RUB)
Gernot Belger (BCE)
Bastian Klein (BfG)
Peter Krahe (BfG)
Dennis Meißner (BfG)

The German Federal Institute of Hydrology
Department M2: Water Balance, Forecasting and Predictions

Software configuration and development requires testing

Setting up an automated test environment in the context of Delft-FEWS (using 'workflowTestRun')

Using the HydPy framework to develop, improve, test, document, and share hydrological models, and to combine them in Delft-FEWS applications.

Complex systems require testing on different levels!

Large systems require test automation!

Open systems require understandable tests!

We require good tools for testing!

From research to practice

University‘s new HBV idea:

non-linear base flow

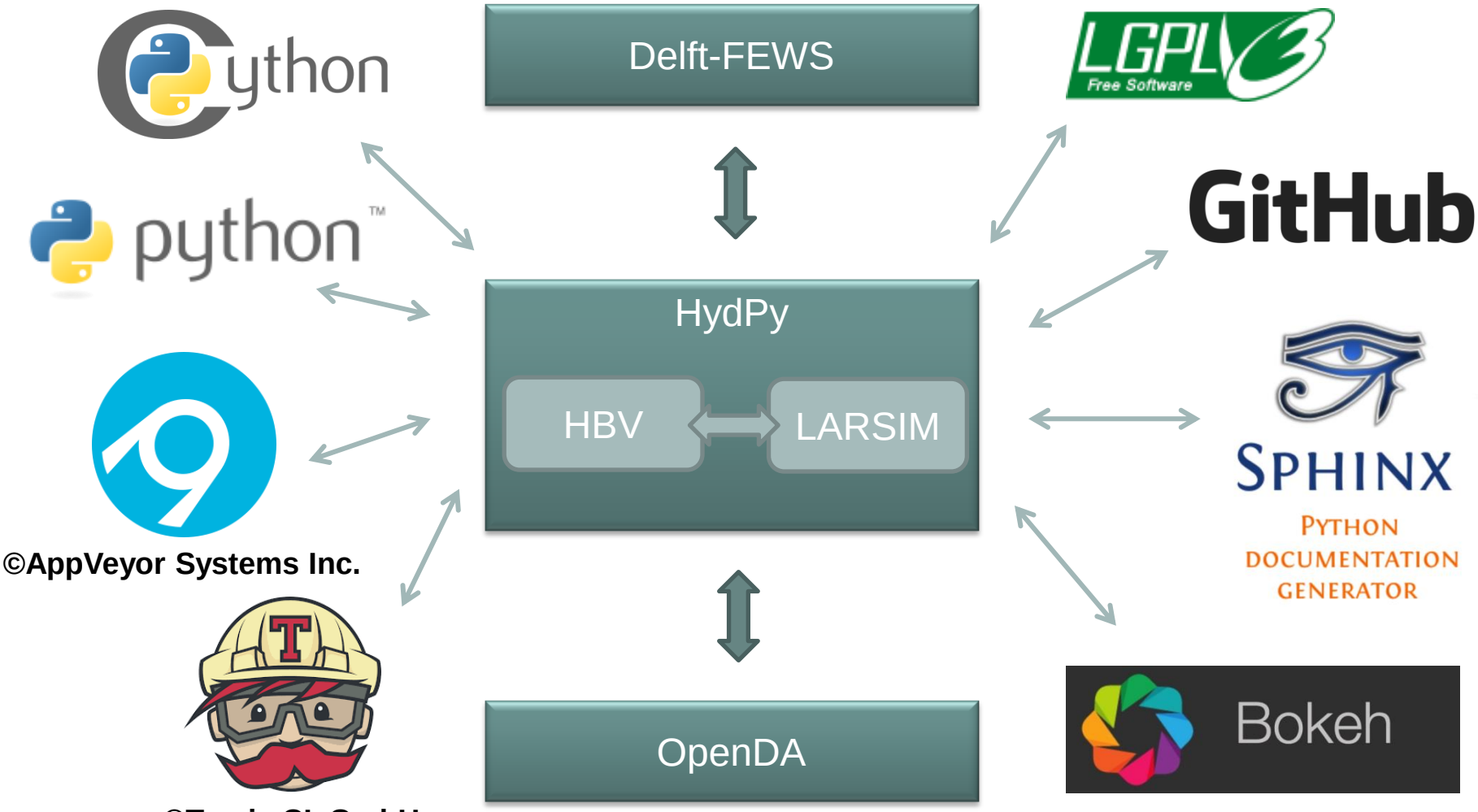
$$Q1 = K4 \cdot LZ^{Gamma}$$

1. just change the equation,
2. but code all other equations first

3. program IO for yourself
4. couple with a calibration algorithm
5. perform the analysis
6. write a paper

7. reprogram IO for others
8. get the model running in different contexts
9. add (at least a few) tests
10. write the documentation
11. choose a licence
12. deploy the model
13. write a FEWS adapter
14. …

Separation of concerns



Coding = documenting = testing

```
14  def calc_nkor_v1(self):
15      """Adjust the given precipitation values.
16
17      Required control parameters:
18        |NHRU|
19        |KG|
20
21      Required input sequence:
22        |Nied|
23
24      Calculated flux sequence:
25        |NKor|
26
27      Basic equation:
28        :math:`NKor = KG \\cdot Nied`
29
30      Example:
31
32          >>> from hydpy.models.lland import *
33          >>> parameterstep('1d')
34          >>> nhru(3)
35          >>> kg(0.8, 1.0, 1.2)
36          >>> inputs.nied = 10.0
37          >>> model.calc_nkor_v1()
38          >>> fluxes.nkor
39          nkor(8.0, 10.0, 12.0)
40      """
41      con = self.parameters.control.fastaccess
42      inp = self.sequences.inputs.fastaccess
43      flu = self.sequences.fluxes.fastaccess
44      for k in range(con.nhru):
45          flu.nkor[k] = con.kg[k] * inp.nied
```

hydpy.models.lland.lland_model.**calc_nkor_v1**(*self*)

Adjust the given precipitation values.

Required control parameters:

NHRU KG

Required input sequence:

Nied

Calculated flux sequence:

NKor

Basic equation:

$$NKor = KG \cdot Nied$$

Example:

```
>>> from hydpy.models.lland import *
>>> parameterstep('1d')
>>> nhru(3)
>>> kg(0.8, 1.0, 1.2)
>>> inputs.nied = 10.
>>> model.calc_nkor_v1()
>>> fluxes.nkor
nkor(8.0, 10.0, 12.0)
```

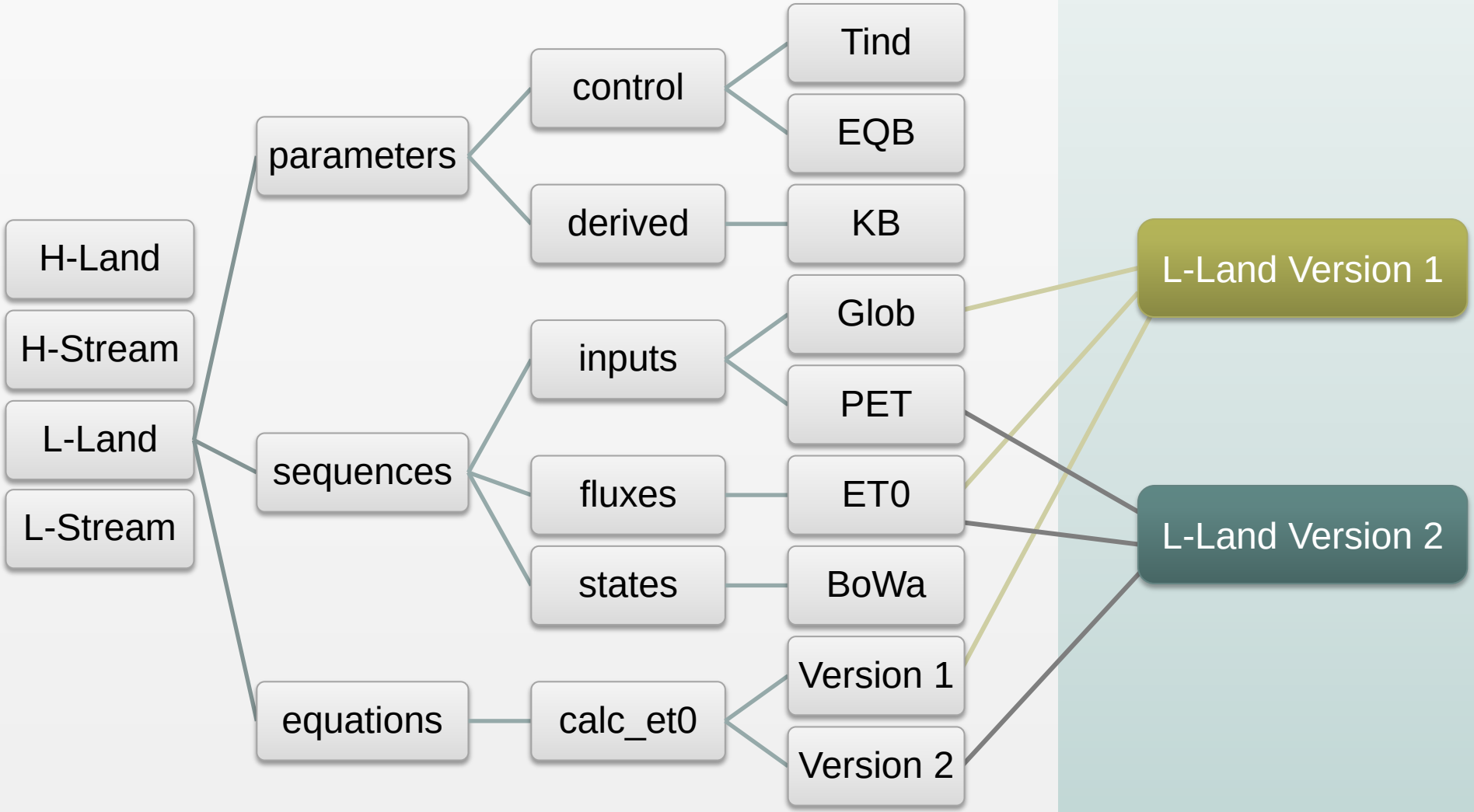
Coding = documenting = testing

```
14  def calc_nkor_v1(self):
15      """Adjust the given precipitation values.
16
17      Required control parameters:
18        |NHRU|
19        |KG|
20
21      Required input sequence:
22        |Nied|
23
24      Calculated flux sequence:
25        |NKor|
26
27      Basic equation:
28        :math:`NKor = KG \\cdot Nied`
29
30      Example:
31
32          >>> from hydpy.models.lland import *
33          >>> parameterstep('1d')
34          >>> nhru(3)
35          >>> kg(0.8, 1.0, 1.2)
36          >>> inputs.nied = 10.0
37          >>> model.calc_nkor_v1()
38          >>> fluxes.nkor
39          nkor(8.0, 10.0, 12.0)
40      """
41      con = self.parameters.control.fastaccess
42      inp = self.sequences.inputs.fastaccess
43      flu = self.sequences.fluxes.fastaccess
44      for k in range(con.nhru):
45          flu.nkor[k] = con.kg[k] * inp.nied
```

```
14      def calc_nkor_v1(self):
15          """Adjust the given precipitation values.
16
17          Required control parameters:
18            |NHRU|
19            |KG|
20
21          Required input sequence:
22            |Nied|
23
24          Calculated flux sequence:
25            |NKor|
26
27          Basic equation:
28            :math:`NKor = KG \\cdot Nied`
29
30          Example:
31
32              >>> from hydpy.models.lland import *
33              >>> parameterstep('1d')
34              >>> nhru(3)
35              >>> kg(0.8, 1.0, 1.2)
36              >>> inputs.nied = 10.
37              >>> model.calc_nkor_v1()
38              >>> fluxes.nkor
39              nkor(8.0, 10.0, 12.0)
40          """
41          con = self.parameters.control.fastaccess
42          inp = self.sequences.inputs.fastaccess
43          flu = self.sequences.fluxes.fastaccess
44          for k in range(con.nhru):
45              flu.nkor[k] = con.kg[k] * inp.nied
```

Model development

base model vs. application model

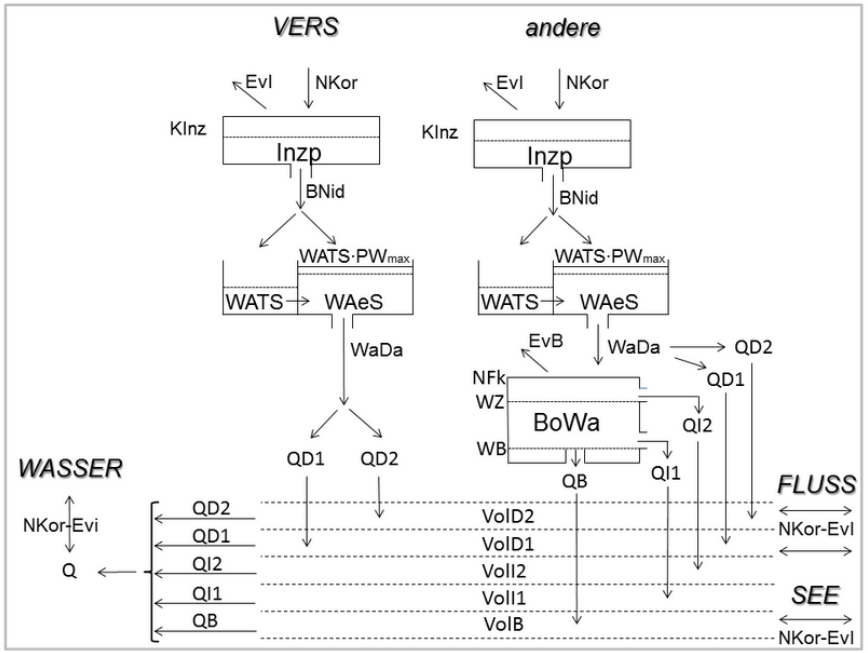


Integration testing

lland_v1 (LARSIM-Xinanjiang-Turc-Wendling version of HydPy-L-Land) ¶

Version 1 of the L-Land model is designed to agree with the LARSIM-ME configuration of the LARSIM model used by the German Federal Institute of Hydrology (BfG), but offers more flexibility in some regards (e.g. in parameterization). It can briefly be summarized as follows:

- Simple routines for adjusting the meteorological input data.
- Reference evapotranspiration after Turc-Wendling.
- An enhanced degree-day-method for calculating snow melt.
- A simple snow retention routine.
- Landuse and month specific potential evapotranspiration.
- Acual soil evapotranspiration after ATV-DVWK- 504 (2002).
- A Soil routine based on the Xinanjiang model.
- One base flow, two interflow and two direct flow components
- Seperate linear storages for modelling runoff concentration.
- Additional evaporation from water areas.



Integration tests:

All integration tests are performed over a period of five days. Despite of the mentioned limitation of the Turc-Wendling equation, an hourly simulation step size is selected (this results in evaporation values that are unrealistically high, but allows for inspecting the effect of evaporative soil moisture depletion within this short simulation period):

```
>>> from hydpy import pub
>>> pub.timegrids = '01.01.2000', '05.01.2000', '1h'
```

Prepare the model instance and build the connections to element *land* and node *outlet*:

```
>>> from hydpy.models.lland_v1 import *
>>> parameterstep('1h')
>>> from hydpy import Node, Element
>>> outlet = Node('outlet')
>>> land = Element('land', outlets=outlet)
>>> land.connect(model)
```

All tests shall be performed using a single hydrological response unit with a size of one square kilometre at an altitude of 100 meter:

```
>>> nhru(1)
>>> ft(1.0)
>>> fhru(1.0)
>>> hnn(100.0)
```

Initialize a test function object, which prepares and runs the tests and prints their results for the given sequences:

```
>>> from hydpy import IntegrationTest
>>> IntegrationTest.plotting_options.height = 800
>>> IntegrationTest.plotting_options.activated=(
...     inputs.nied, inputs.teml, fluxes.q)
>>> test = IntegrationTest(land)
>>> test.dateformat = '%d.%m.'
```

Example 1

In the first example, arable land is selected as the only land use class (for all other land types, except the ones mentioned below, the results would be the same):

```
>>> lnk(ACKER)
```

The following set of control parameter values tries to configure application model `lland_v1` in a manner that allows to retrace the influence of the different implemented methods on the shown results:

```
>>> kg(1.2)
>>> kt(0.8)
>>> ke(0.4)
>>> kf(0.6)
>>> fln(0.5)
>>> hinz(0.2)
>>> lai(4.0)
>>> treft(0.0)
>>> trefn(0.0)
>>> tgr(1.0)
>>> tsp(2.0)
>>> gtf(0.5)
```

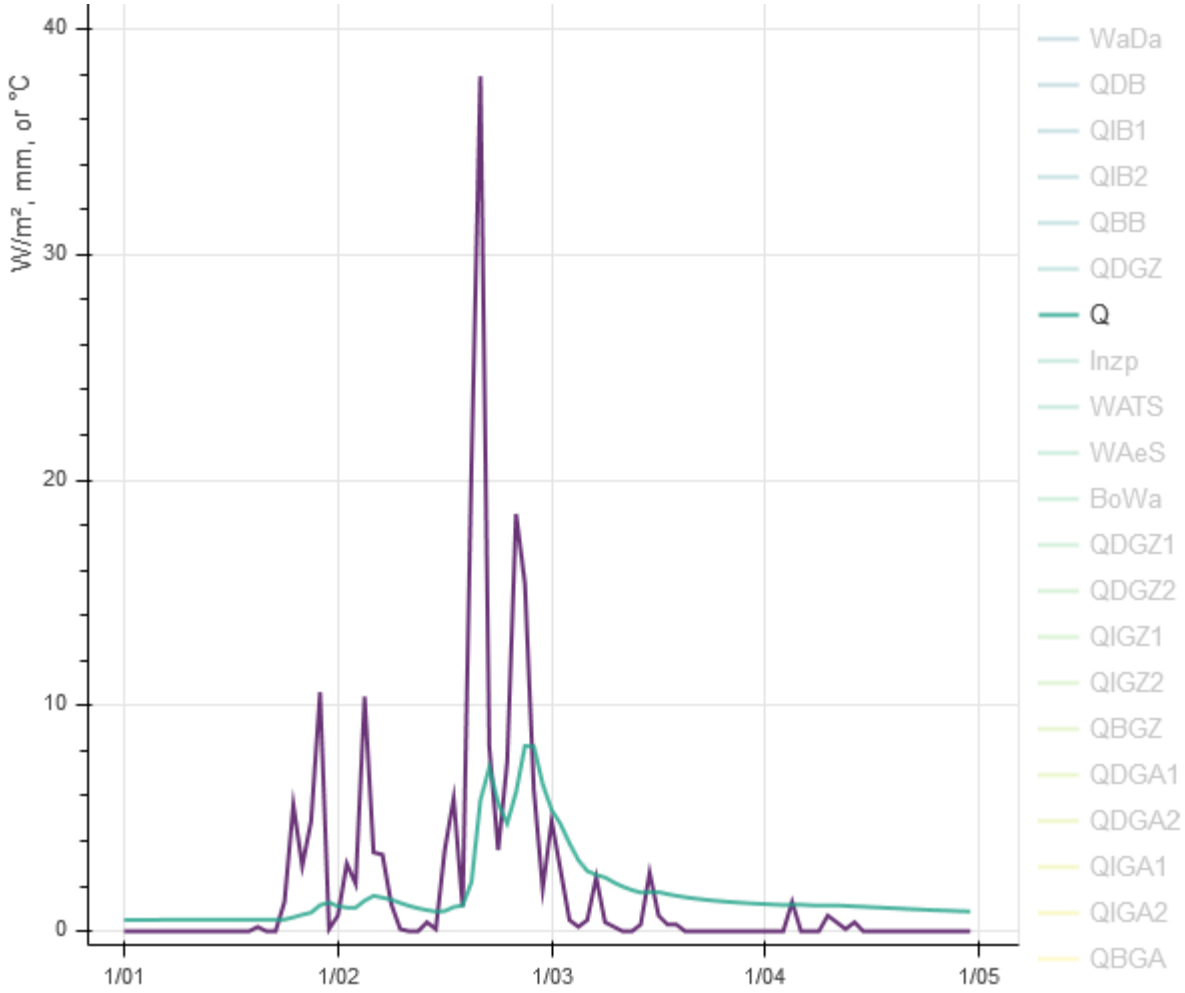
The first input data set mimics a extreme precipitation event in summer:

```
>>> inputs.nied.series = (
...     0.0, 0.0,  0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
...     0.0, 0.0, 0.2, 0.0, 0.0, 1.3, 5.6, 2.9, 4.9, 10.6, 0.1, 0.7, 3.0,
...     2.1, 10.4, 3.5, 3.4, 1.2, 0.1, 0.0, 0.0, 0.4, 0.1, 3.6, 5.9, 1.1,
...     20.7, 37.9, 8.2, 3.6, 7.5, 18.5, 15.4, 6.3, 1.9, 4.9, 2.7, 0.5,
...     0.2, 0.5, 2.4, 0.4, 0.2, 0.0, 0.0, 0.3, 2.6, 0.7, 0.3, 0.3, 0.0,
...     0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.3, 0.0,
...     0.0, 0.0, 0.7, 0.4, 0.1, 0.4, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
...     0.0, 0.0, 0.0, 0.0, 0.0, 0.0)
```

The following results show that all relevant model components, except the snow routines, are activated at least once within the simulation period. Take your time to click and scroll through the figure, to see e.g. how the soil moisture content `BoWa` is varying over time. One might realize the "linear storage" type of relationship between inflow `Nied` and outflow `Q`. This is due to the dominance of the direct runoff generation (`QDGZ`) based on the Xinanjiang model and runoff concentration being modelled by linear storages only (easy inspectable through clicking e.g. on `QDGZ1` and `QDGA1`):

```
>>> test('lland_v1_ex1')
|   date | nied | teml |  glob |  nkor | tkor |      et0 |     evpo |      nbes | sbes |      evi
-------------------------------------------------------------------------------------------------
| 01.01. |  0.0 | 21.2 |   0.0 |   0.0 | 22.0 | 0.040283 | 0.020141 |       0.0 |  0.0 |       0.0
| 01.01. |  0.0 | 19.4 |   0.0 |   0.0 | 20.2 | 0.039121 |  0.01956 |       0.0 |  0.0 |       0.0
| 01.01. |  0.0 | 18.9 |   0.0 |   0.0 | 19.7 | 0.038793 | 0.019396 |       0.0 |  0.0 |       0.0
| 01.01. |  0.0 | 18.3 |   0.0 |   0.0 | 19.1 | 0.038396 | 0.019198 |       0.0 |  0.0 |       0.0
| 01.01. |  0.0 | 18.9 |   0.0 |   0.0 | 19.7 | 0.038793 | 0.019396 |       0.0 |  0.0 |       0.0
| 01.01. |  0.0 | 22.5 |   0.0 |   0.0 | 23.3 | 0.041105 | 0.020552 |       0.0 |  0.0 |       0.0
| 01.01. |  0.0 | 25.1 |  11.2 |   0.0 | 25.9 | 0.116763 | 0.058382 |       0.0 |  0.0 |       0.0
| 01.01. |  0.0 | 28.3 | 105.5 |   0.0 | 29.1 |  0.77315 | 0.386575 |       0.0 |  0.0 |       0.0
| 01.01. |  0.0 | 27.8 | 248.3 |   0.0 | 28.6 | 1.747814 | 0.873907 |       0.0 |  0.0 |       0.0
| 01.01. |  0.0 | 31.4 | 401.3 |   0.0 | 32.2 | 2.927022 | 1.463511 |       0.0 |  0.0 |       0.0
| 01.01. |  0.0 | 32.2 | 449.7 |   0.0 | 33.0 | 3.305745 | 1.652872 |       0.0 |  0.0 |       0.0
| 01.01. |  0.0 | 35.2 | 493.4 |   0.0 | 36.0 | 3.747947 | 1.873973 |       0.0 |  0.0 |       0.0
| 01.01. |  0.0 | 37.1 | 261.5 |   0.0 | 37.9 | 2.050471 | 1.025236 |       0.0 |  0.0 |       0.0
| 01.01. |  0.0 | 31.2 | 363.6 |   0.0 | 32.0 | 2.650012 | 1.325006 |       0.0 |  0.0 |       0.0
| 01.01. |  0.0 | 24.3 | 446.2 |   0.0 | 25.1 | 2.959048 | 1.479524 |       0.0 |  0.0 |       0.0
| 01.01. |  0.2 | 25.4 | 137.6 |  0.24 | 26.2 | 0.956604 | 0.478302 |       0.0 |  0.0 |      0.24
| 01.01. |  0.0 | 25.9 | 103.0 |   0.0 | 26.7 | 0.731933 | 0.365967 |       0.0 |  0.0 |       0.0
| 01.01. |  0.0 | 23.7 |  63.7 |   0.0 | 24.5 | 0.454628 | 0.227314 |       0.0 |  0.0 |       0.0
| 01.01. |  1.3 | 21.6 |  41.4 |  1.56 | 22.4 | 0.300394 | 0.150197 |      0.76 |  0.0 |  0.150197
| 01.01. |  5.6 | 21.2 |   7.9 |  6.72 | 22.0 | 0.089558 | 0.044779 |  6.569803 |  0.0 |  0.044779
| 01.01. |  2.9 | 20.4 |   0.0 |  3.48 | 21.2 |  0.03977 | 0.019885 |  3.435221 |  0.0 |  0.019885
| 01.01. |  4.9 | 19.8 |   0.0 |  5.88 | 20.6 | 0.039381 | 0.019691 |  5.860115 |  0.0 |  0.019691
| 01.01. | 10.6 | 19.6 |   0.0 | 12.72 | 20.4 | 0.039251 | 0.019626 | 12.700309 |  0.0 |  0.019626
| 01.01. |  0.1 | 19.2 |   0.0 |  0.12 | 20.0 |  0.03899 | 0.019495 |  0.100374 |  0.0 |  0.019495
| 02.01. |  0.7 | 19.2 |   0.0 |  0.84 | 20.0 |  0.03899 | 0.019495 |  0.820505 |  0.0 |  0.019495
| 02.01. |  3.0 | 19.2 |   0.0 |   3.6 | 20.0 |  0.03899 | 0.019495 |  3.580505 |  0.0 |  0.019495
| 02.01. |  2.1 | 18.9 |   0.0 |  2.52 | 19.7 | 0.038793 | 0.019396 |  2.500505 |  0.0 |  0.019396
| 02.01. | 10.4 | 18.7 |   0.0 | 12.48 | 19.5 | 0.038661 |  0.01933 | 12.460604 |  0.0 |  0.01933
```

HydPy-L-Land (LARSIM): ACKER



Model configuration

```
# -*- coding: utf-8 -*-

from hydpy.models.lland_v2 import *

simulationstep("1h")
parameterstep("1d")

ft(9.0)
nhru(7)
lnk(VERS, ACKER, GRUE_I, NADELW, LAUBW, MISCHW, SEE)
fhru(acker=0.33, grue_i=0.21, laubw=0.02, mischw=0.04,
     nadelw=0.32, see=0.01, vers=0.06)    # ToDo
kg(1.0)
kt(0.0)
ke(0.8)
nfk(acker=191.6, grue_i=165.6, laubw=220.2, mischw=146.4,
    nadelw=187.1, see=nan, vers=nan)
fln(pyfile='landuse_parameters.py')
negq(True)


if __name__ == '__main__':
    assert sum(fhru) == 1.0
```

Workflow configuration (for FEWS)

```
<?xml version="1.0" encoding="UTF-8"?>
<config xmlns="https://github.com/hydpy-dev/hydpy/tree/master/hydpy/conf/config.xsd">

    <options...>

    <timegrid>
        <firstdate><!--|firstdate|--></firstdate>
        <lastdate><!--|lastdate|--></lastdate>
        <stepsize>1h</stepsize>
    </timegrid>

    <selections>
        complete
    </selections>

    <devices...>

    <conditions_io>
        <inputdir>indir</inputdir>
        <outputdir>outdir</outputdir>
        <zip>true</zip>
    </conditions_io>

    <series_io>

        <filetype>
            <general>nc</general>
        </filetype>
```

To-do:

- increase the test coverage of currently 93 % !
- perform a static code analysis ?
- add beginner tutorials to the online documentation !
- release a FEWS-HydPy demo project !
- finish the OpenDA-HydPy wrapper !
- support High-Performance Computing ?

- invite colleagues to apply HydPy and to implement their own models

Thank you for your attention!

Christoph Tyralla
c.tyralla@bjoernsen.de
https://github.com/hydpy-dev/hydpy
https://hydpy-dev.github.io/hydpy/index.html