Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

Swiss Confederation

Federal Department of Home Affairs FDHA
Federal Office of Meteorology and Climatology MeteoSwiss

SLF

NIX standalone

Nander Wever[1,2],Sascha Bellaire[2], Varun Sharma[2], Michael Lehning[1,3], Jean-Marie Bettems[2] et al.

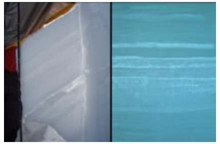
[1]WSL Institute for Snow and Avalanche Research SLF, Davos, Switzerland
[2]MeteoSwiss, Zurich, Switzerland
[3]CRYOS, School of Architecture, Civil and Environmental Engineering, EPFL, Lausanne, Switzerland

Content

1. Introduction SNOWPACK vs NIX
2. Recent developments in NIX
3. How to set up NIX and how to use NIX output
4. Validation (comparing NIX with SNOWPACK)
5. Integrating snow analysis

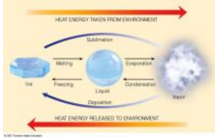
Snow modelling – SNOWPACK

- SNOWPACK developed originally for the Swiss avalanche warning
- Later extended with modules and modifications for permafrost, soil, canopy, sea ice, firn on ice sheets
- Multi-layer, detailed, snow cover scheme
- Stronger focus on accurate representation of snow processes rather than computational performance
- Written in C++
- Core processes: heat equation, compaction (settling), advanced water transport, vapour transport, snow microstructure
- Snow stability (avalanche forecasting)



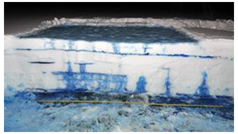
Layering (create/aggregate/split

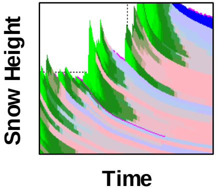
Heat Equation(implicit) $\frac{¶T}{¶t} = a\frac{¶^2 T}{¶x^2}; \quad 0 £ x £ L; \quad t ^3 0$



Phase Changes

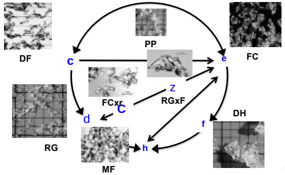
Water transport





Settling

Snow microstructure

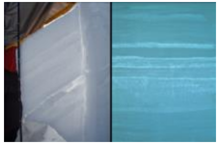


Photo's: SLF

Snow modelling (What is NIX?)

a.k.a SNOWPOLINO

- Adaptation of sophisticated snow cover model SNOWPACK
- Stronger focus on computational performance over accurate representation of snow processes
- Parameterizations of physical ‘core’ processes
- Two version (~unified):
  - stand-alone (offline)
  - fully-coupled (online; ICON) --> Sascha's part
- Modular structure, coded in Fortran



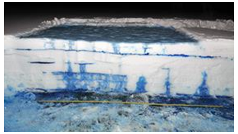
Layering (create/aggregate/split)

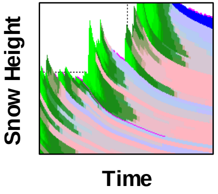
Heat Equation(implicit) $\frac{¶T}{?¶t} = a\frac{¶^2T}{¶x^2}; \quad 0 £ x £ L; \quad t ³ 0$

HEAT ENERGY TAKEN FROM ENVIRONMENT
Sublimation
Melting
Evaporation
Ice
Freezing
Condensation
Liquid
Vapor
Deposition
HEAT ENERGY RELEASED TO ENVIRONMENT

Phase Changes

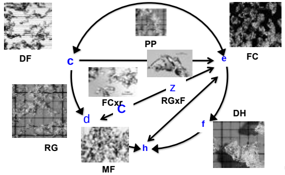
Water transport





Settling

No microstructure yet!



Photo's: SLF

NIX – standalone

Off-line, 1-dimensional point model

- Strives to be identical to NIX in ICON for core modules
- Allows for easy testing, further development and bug hunting
- Can run from initial conditions provided by NIX in ICON
- Can run from ICON meteorological forcing
- https://github.com/COSMO-ORG/nix-alone

NIX – developments

Developments using nix-alone

- Several bug fixes and conceptual design changes --> ported to ICON version
- Verifying layer spacing and temporal timestep independency of results
  - Some improvements in heat equation solver --> also ported to ICON version
- Better layer spacing management
  - Separate minimum snowfall criterion from minimum layer spacing --> also ported to ICON version
- Improved workflow to set up a nix-alone simulation
- NIX-alone only: Adding output for easy visualization using:
  - niViz: https://niViz.org
    - NiViz is developed by SLF
    - Interactive visualization tailored to snow cover visualizations
    - developed for SNOWPACK
  - Snowpat library: https://gitlabext.wsl.ch/patrick.leibersperger/snowpat
    - Developed by SLF
    - Scripted plotting using python tailored to snow cover plotting

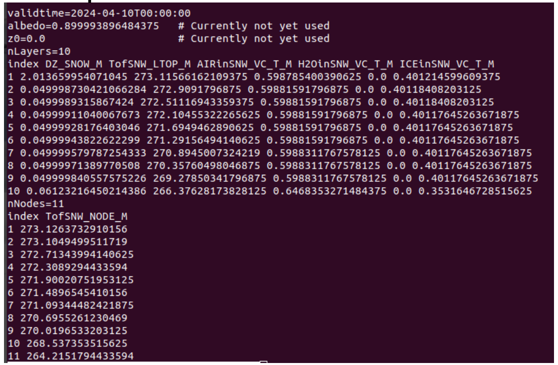
Preparing and running NIX-alone

https://github.com/COSMO-ORG/nix-alone/blob/dev/scripts/python/convert_to_nixalone.ipynb

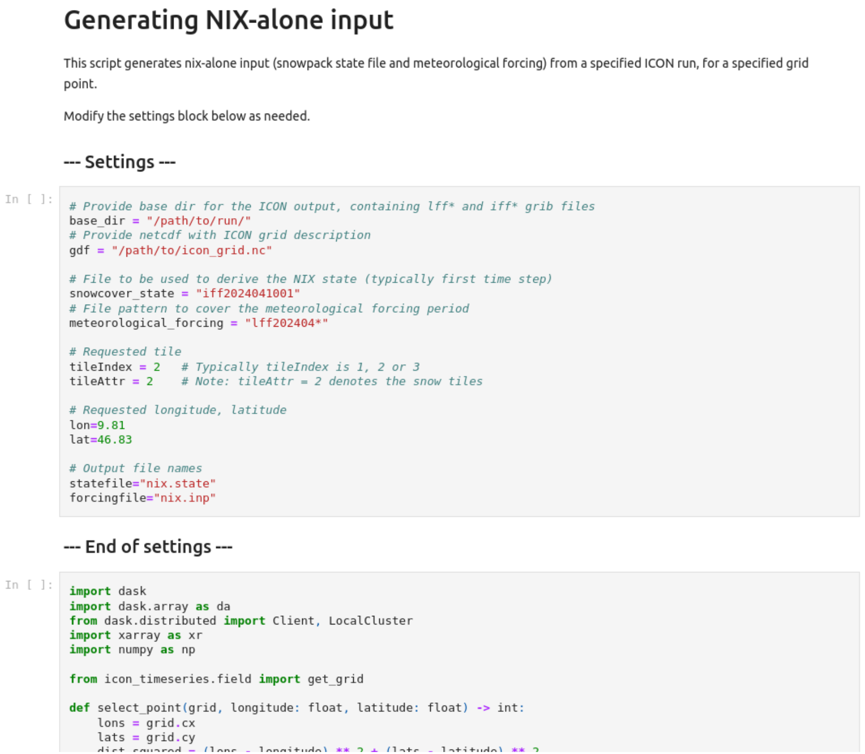
`mo_nix_config.f90`: itype_nix_start

- CASE(1): ‘hard’ cold start – all snow is wiped out
- CASE(2): ‘soft’ cold start – not implemented yet in nix-alone
- CASE(3): warm start – read NIX prognostic fields from initial conditions (`nix.state` file)

Example `nix.state` file:



```
validtime=2024-04-10T00:00:00
albedo=0.899993896484375   # Currently not yet used
z0=0.0                     # Currently not yet used
nLayers=10
index DZ_SNOW_M TofSNW_LTOP_M AIRinSNW_VC_T_M H2OinSNW_VC_T_M ICEinSNW_VC_T_M
1 2.013659954071045 273.11566162109375 0.598785400390625 0.0 0.401214599609375
2 0.049998730421066284 272.9091796875 0.59881591796875 0.0 0.40118408203125
3 0.0499989315867424 272.51116943359375 0.59881591796875 0.0 0.40118408203125
4 0.04999911040067673 272.10455322265625 0.59881591796875 0.0 0.40117645263671875
5 0.04999928176403046 271.6949462890625 0.59881591796875 0.0 0.40117645263671875
6 0.04999943822622299 271.29156494140625 0.59881591796875 0.0 0.40117645263671875
7 0.04999957978725433 270.8945007324219 0.5988311767578125 0.0 0.40117645263671875
8 0.04999971389770508 270.35760498046875 0.5988311767578125 0.0 0.40117645263671875
9 0.04999840557575226 269.27850341796875 0.5988311767578125 0.0 0.40117645263671875
10 0.06123216450214386 266.37628173828125 0.6468353271484375 0.0 0.3531646728515625
nNodes=11
index TofSNW_NODE_M
1 273.1263732910156
2 273.1049499511719
3 272.71343994140625
4 272.3089294433594
5 271.90020751953125
6 271.4896545410156
7 271.09344482421875
8 270.6955261230469
9 270.0196533203125
10 268.537353515625
11 264.2151794433594
```



Generating NIX-alone input

This script generates nix-alone input (snowpack state file and meteorological forcing) from a specified ICON run, for a specified grid point.

Modify the settings block below as needed.

--- Settings ---

In [ ]:

```
# Provide base dir for the ICON output, containing lff* and iff* grib files
base_dir = "/path/to/run/"
# Provide netcdf with ICON grid description
gdf = "/path/to/icon_grid.nc"

# File to be used to derive the NIX state (typically first time step)
snowcover_state = "iff2024041001"
# File pattern to cover the meteorological forcing period
meteorological_forcing = "lff202404*"

# Requested tile
tileIndex = 2   # Typically tileIndex is 1, 2 or 3
tileAttr = 2    # Note: tileAttr = 2 denotes the snow tiles

# Requested longitude, latitude
lon=9.81
lat=46.83

# Output file names
statefile="nix.state"
forcingfile="nix.inp"
```

--- End of settings ---

In [ ]:

```
import dask
import dask.array as da
from dask.distributed import Client, LocalCluster
import xarray as xr
import numpy as np

from icon_timeseries.field import get_grid

def select_point(grid, longitude: float, latitude: float) -> int:
    lons = grid.cx
    lats = grid.cy
    dist_squared = (lons - longitude) ** 2 + (lats - latitude) ** 2
```

Preparing and running NIX-alone

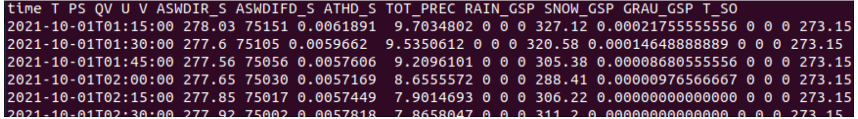
https://github.com/COSMO-ORG/nix-alone/blob/dev/scripts/python/convert_to_nixalone.ipynb

Meteorological forcing

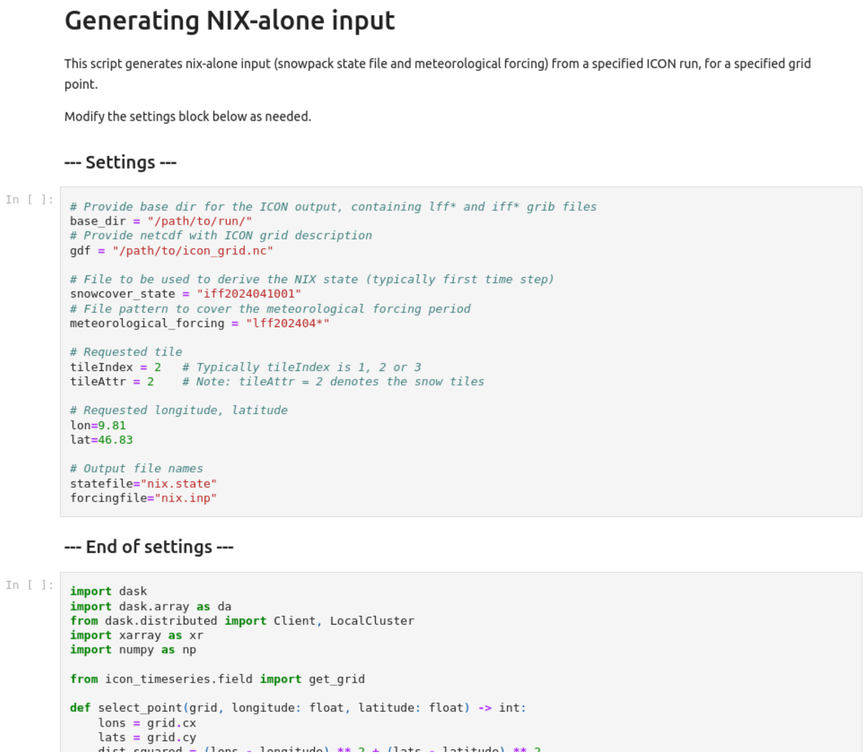
- Air temperature
- Pressure
- Specific humidity
- Wind speed
- Incoming shortwave and longwave radiation
- Precipitation amount
- Precipitation Phase

Note: SNOWPACK requires same input

Example `nix.inp` file:



```
time T PS QV U V ASWDIR_S ASWDIFD_S ATHD_S TOT_PREC RAIN_GSP SNOW_GSP GRAU_GSP T_SO
2021-10-01T01:15:00 278.03 75151 0.0061891  9.7034802 0 0 0 327.12 0.00021755555556 0 0 0 273.15
2021-10-01T01:30:00 277.6 75105 0.0059662  9.5350612 0 0 0 320.58 0.00014648888889 0 0 0 273.15
2021-10-01T01:45:00 277.56 75056 0.0057606  9.2096101 0 0 0 305.38 0.00008680555556 0 0 0 273.15
2021-10-01T02:00:00 277.65 75030 0.0057169  8.6555572 0 0 0 288.41 0.00000976566667 0 0 0 273.15
2021-10-01T02:15:00 277.85 75017 0.0057449  7.9014693 0 0 0 306.22 0.00000000000000 0 0 0 273.15
2021-10-01T02:30:00 277.92 75002 0.0057818  7.8658047 0 0 0 311.2 0.00000000000000 0 0 0 273.15
```



Generating NIX-alone input

This script generates nix-alone input (snowpack state file and meteorological forcing) from a specified ICON run, for a specified grid point.

Modify the settings block below as needed.

--- Settings ---

In [ ]:

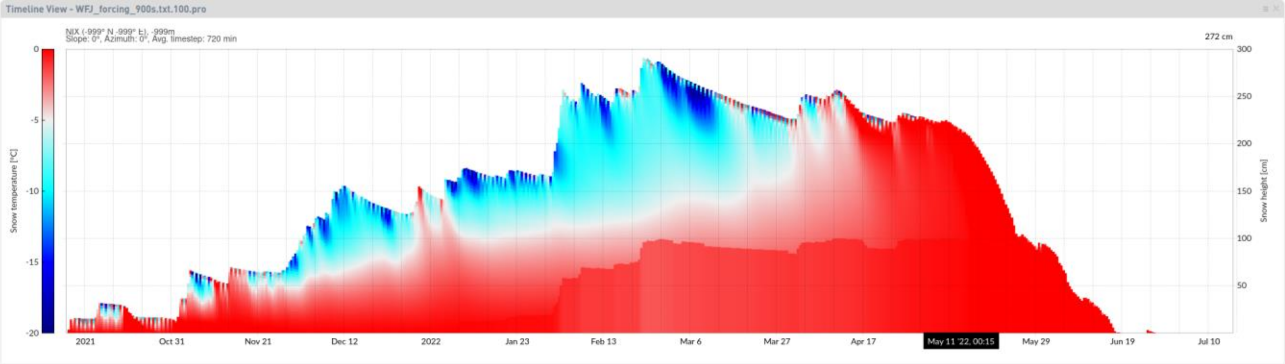
```python
# Provide base dir for the ICON output, containing lff* and iff* grib files
base_dir = "/path/to/run/"
# Provide netcdf with ICON grid description
gdf = "/path/to/icon_grid.nc"

# File to be used to derive the NIX state (typically first time step)
snowcover_state = "iff2024041001"
# File pattern to cover the meteorological forcing period
meteorological_forcing = "lff202404*"

# Requested tile
tileIndex = 2   # Typically tileIndex is 1, 2 or 3
tileAttr = 2    # Note: tileAttr = 2 denotes the snow tiles

# Requested longitude, latitude
lon=9.81
lat=46.83

# Output file names
statefile="nix.state"
forcingfile="nix.inp"
```

--- End of settings ---

In [ ]:

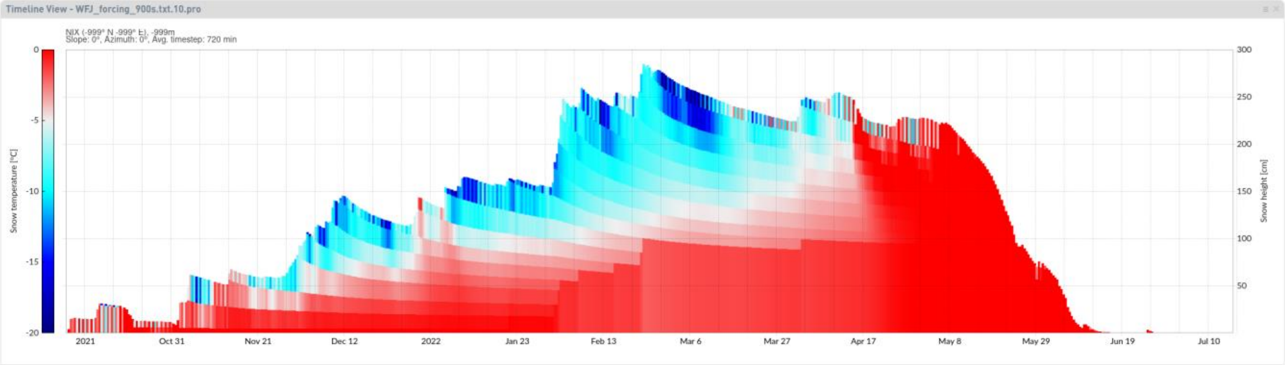
```python
import dask
import dask.array as da
from dask.distributed import Client, LocalCluster
import xarray as xr
import numpy as np

from icon_timeseries.field import get_grid

def select_point(grid, longitude: float, latitude: float) -> int:
    lons = grid.cx
    lats = grid.cy
    dist_squared = (lons - longitude) ** 2 + (lats - latitude) ** 2
```

niViz example – visualizing snowpacks



Timeline View - WFJ_forcing_900s.txt.100.pro

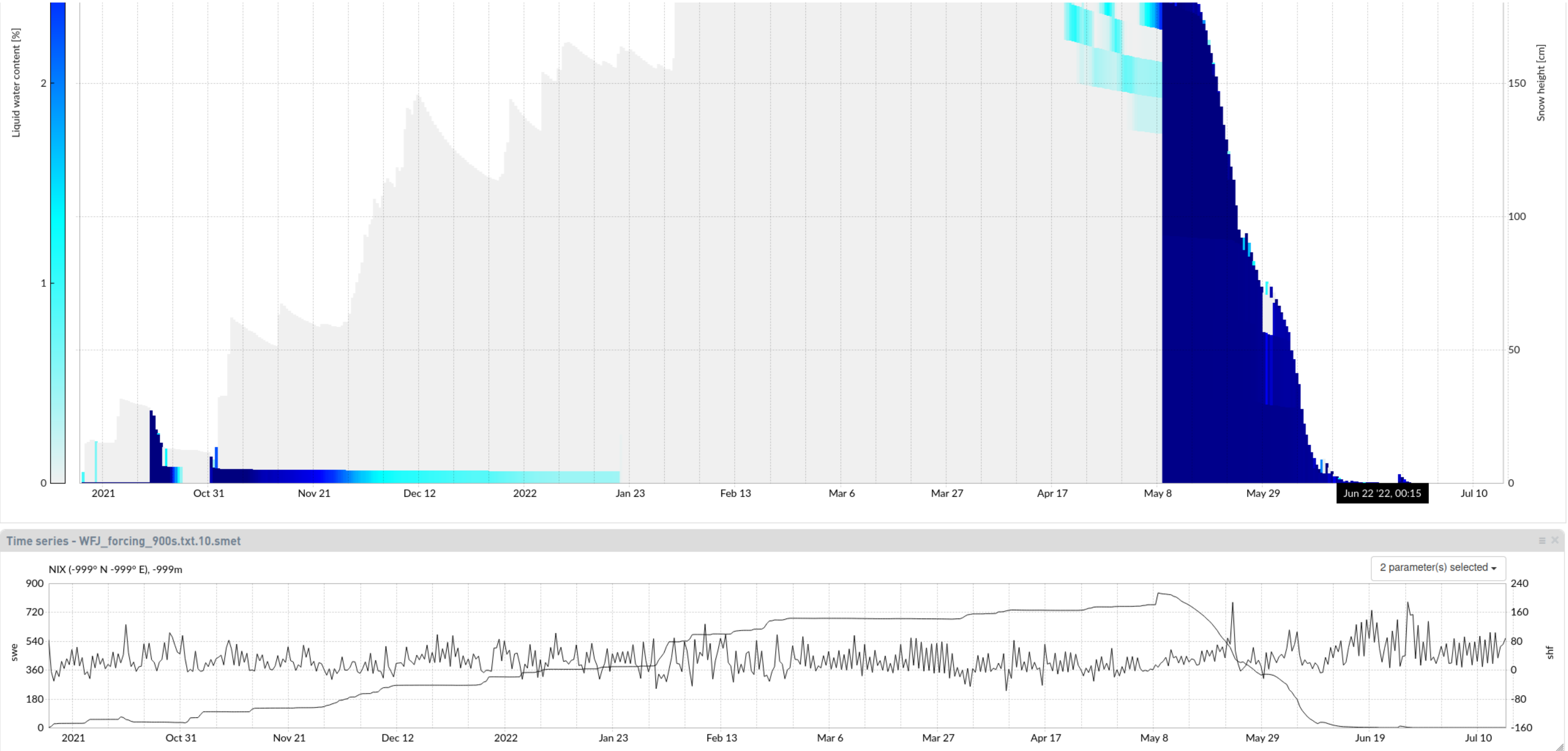
NIX (-999° N -999° E), -999m
Slope: 0°, Azimuth: 0°, Avg. timestep: 720 min



Timeline View - WFJ_forcing_900s.txt.10.pro

NIX (-999° N -999° E), -999m
Slope: 0°, Azimuth: 0°, Avg. timestep: 720 min

niViz example – visualizing snowpacks

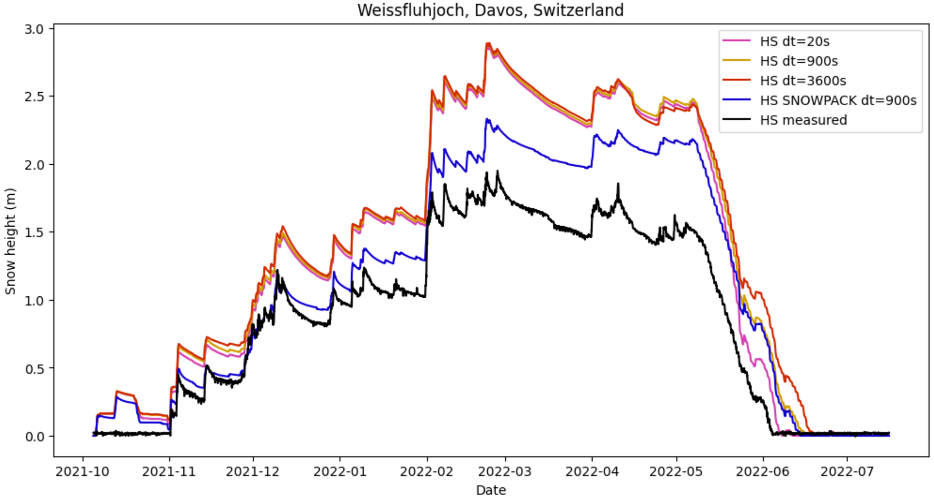


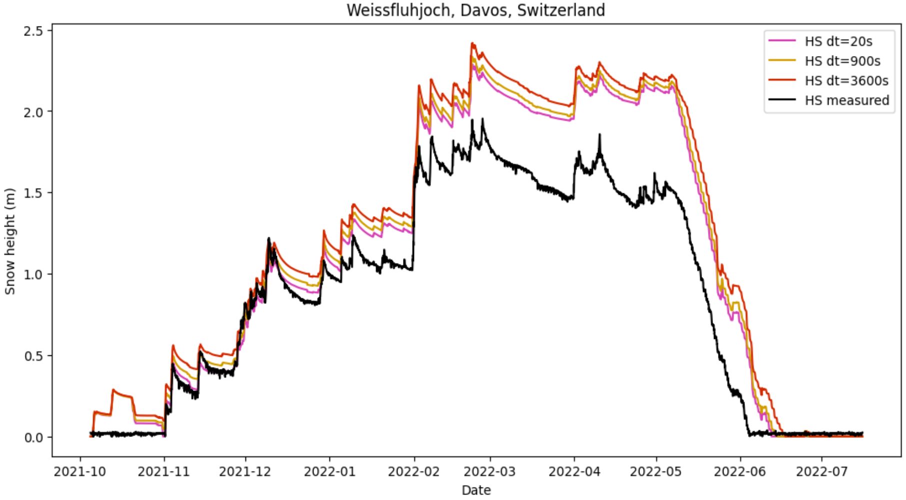
Time series - WFJ_forcing_900s.txt.10.smet

NIX (-999° N -999° E), -999m

2 parameter(s) selected ▾

Time step dependency





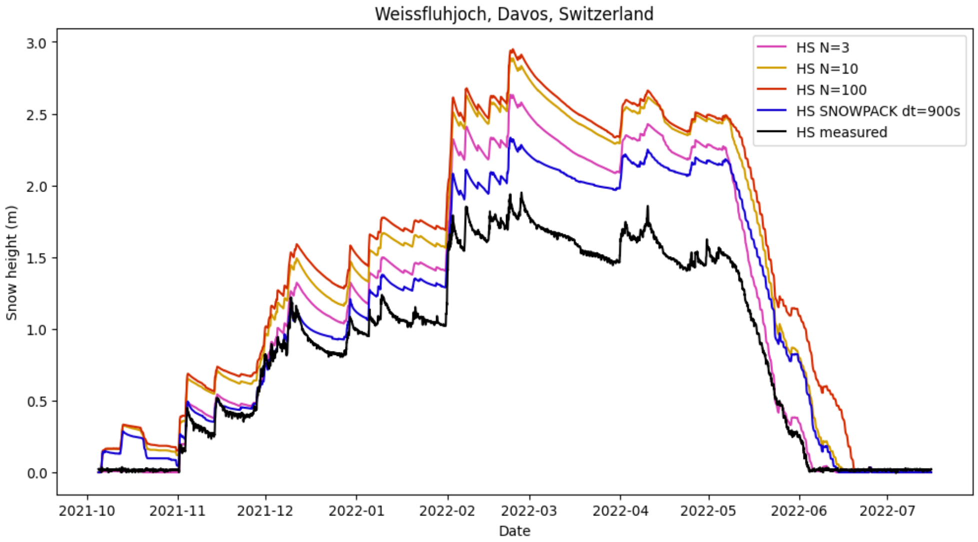
NIX 10L 3600s : 1.09 s
NIX 10L 900s : 1.35 s
NIX 10L 20s : 13.06 s

SNOWPACK 3600s : 28.20 s
SNOWPACK 900s : 45.77 s
SNOWPACK 20s : 2117.24 s

(~500 layers)

Flexibility in choice of time step is important for future developments

Layer spacing dependency



Layer spacing impacts how well temperature gradients can be represented

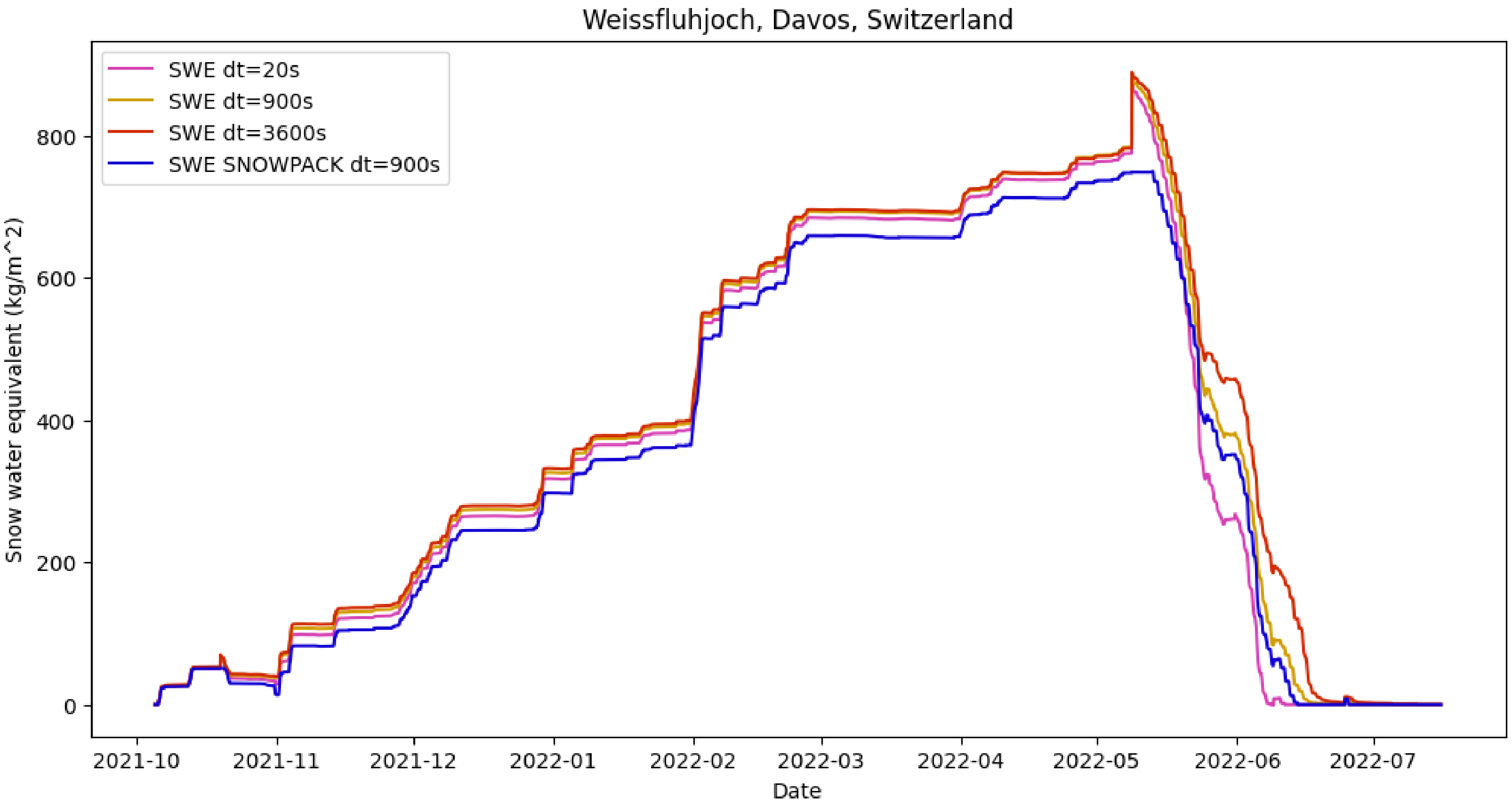
We find stable model behaviour independent of layer spacing.

A minimum of 10 layers seems recommended

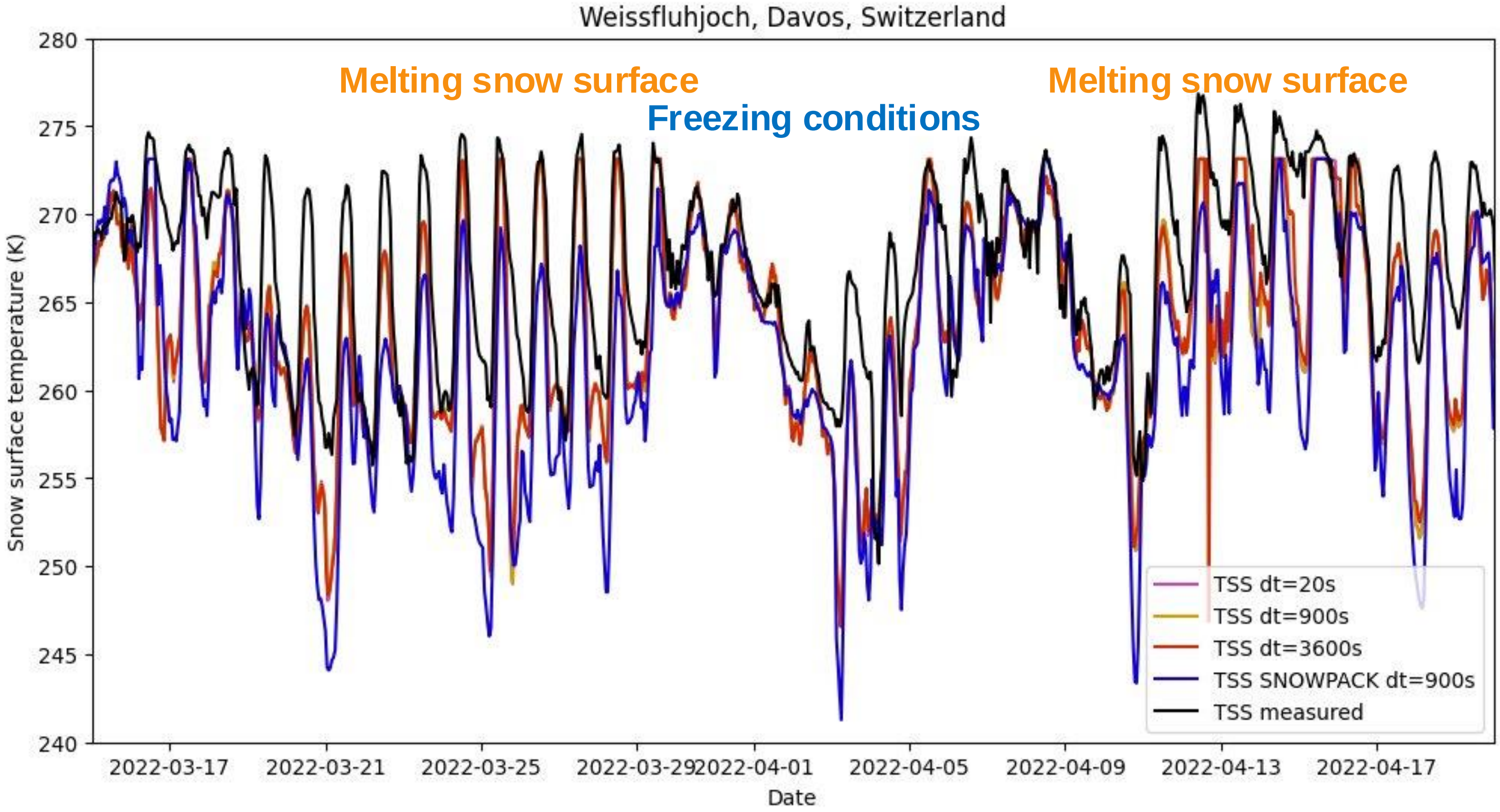
NIX 10L 3600s : 1.09
NIX 10L 900s : 1.35
NIX 10L 20s : 13.06

NIX 100L 3600s : 1.59
NIX 100L 900s : 2.10
NIX 100L 20s : 32.94

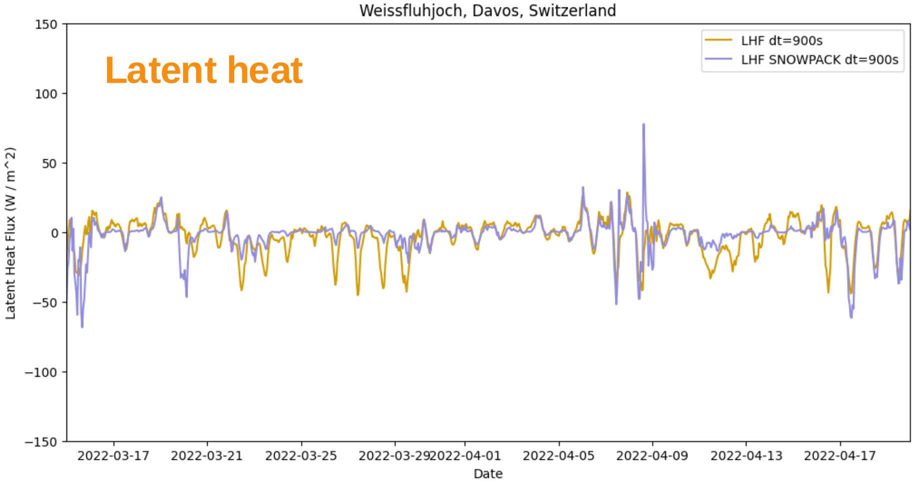
Validation – Snow Water Equivalent (SWE)

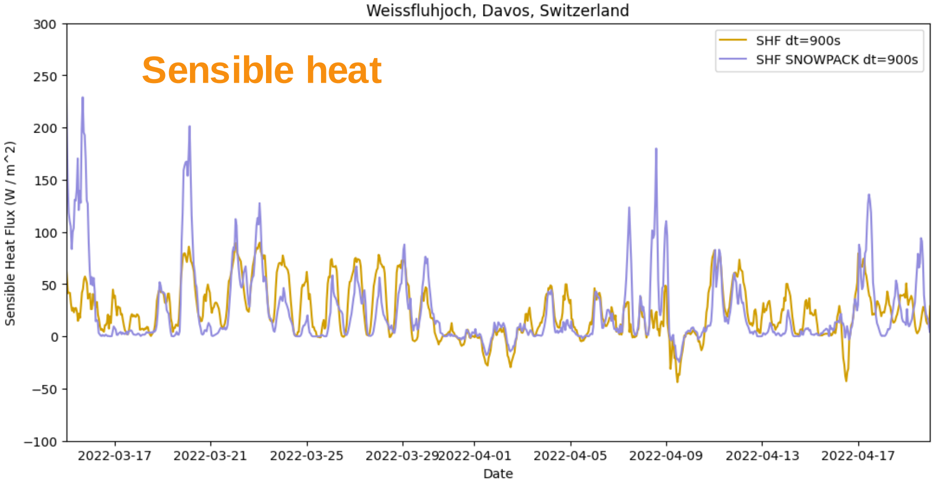


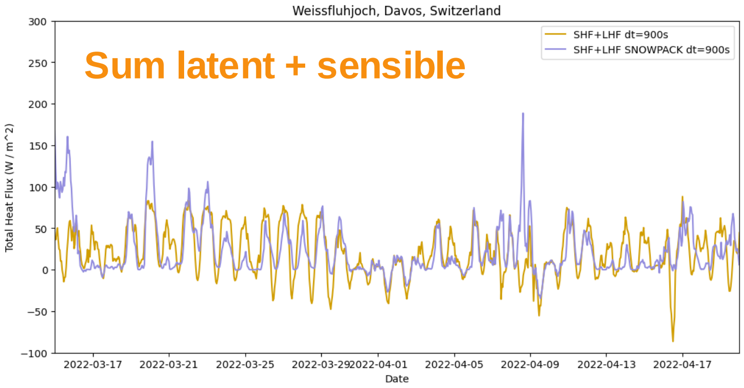
Validation – snow surface temperature



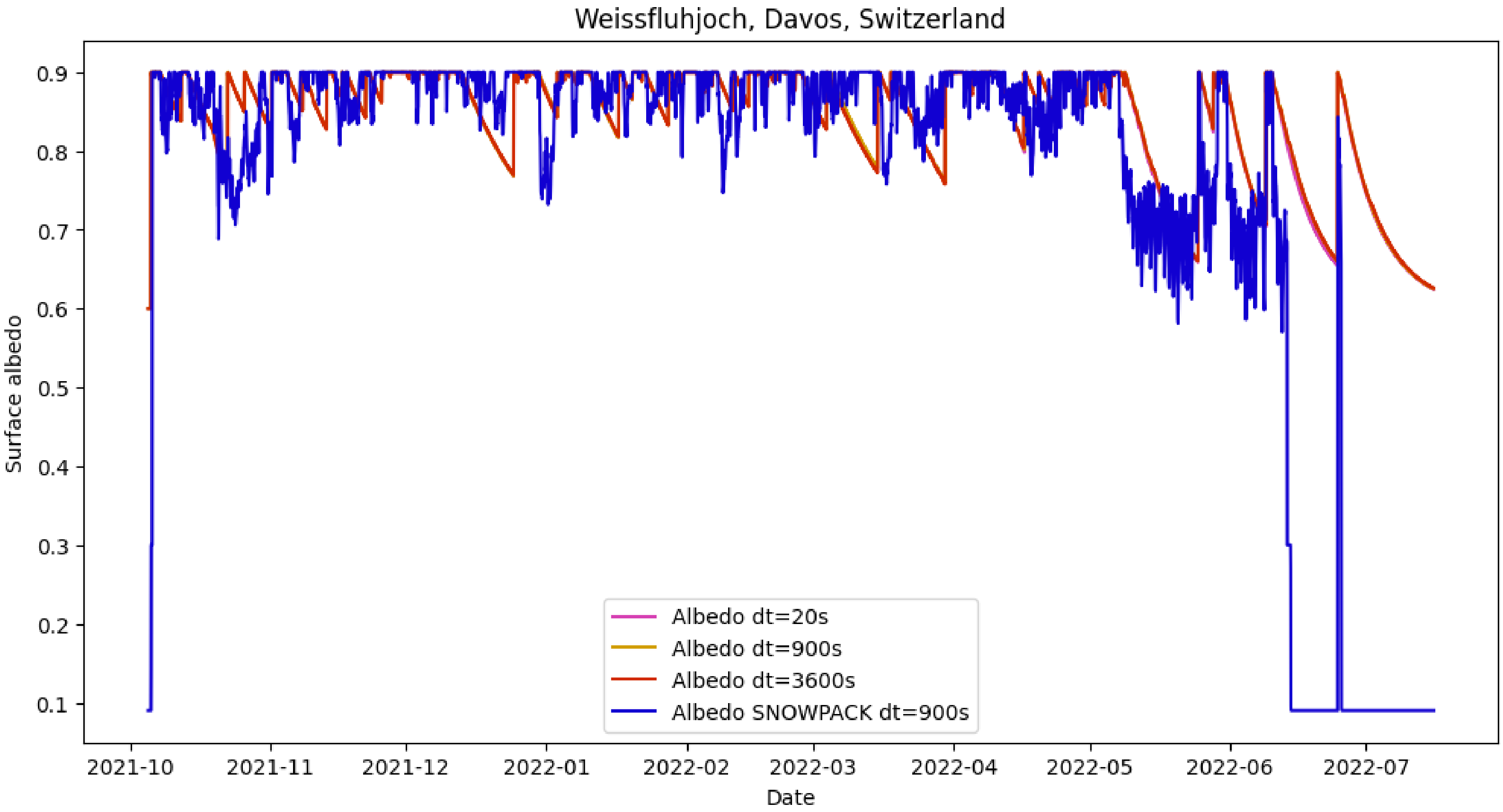
Validation – turbulent fluxes


Latent heat


Sensible heat


Sum latent + sensible

Validation – albedo



Snow - analysis

If initial h_snow != sum of snow layer depths in NIX:

1. If both > 20 cm: do nothing
2. If h_snow == 0: remove all snow in NIX
3. If h_snow > 0 and NIX == 0: built snowpack
   - If air temperature above 0 C: typical wet snow conditions (high density, low albedo)
   - If air temperature below 0 C: typical dry snow conditions (low density, high albedo)
4. If h_snow > NIX snow depth: duplicate and scale layers to increase snow depth
5. If h_snow < NIX snow layers: scale layers

Conclusions

- Time step and layer spacing dependency is acceptable
- Comparison between nix-alone and SNOWPACK on some key snowpack variables that are important for the interactions with other parts:
  - Mass balance highly comparable with SNOWPACK
  - Turbulent fluxes compare well with SNOWPACK
- NIX results generally acceptable for use in ICON
- Implementation to assimilate snow analysis into NIX

Outlook

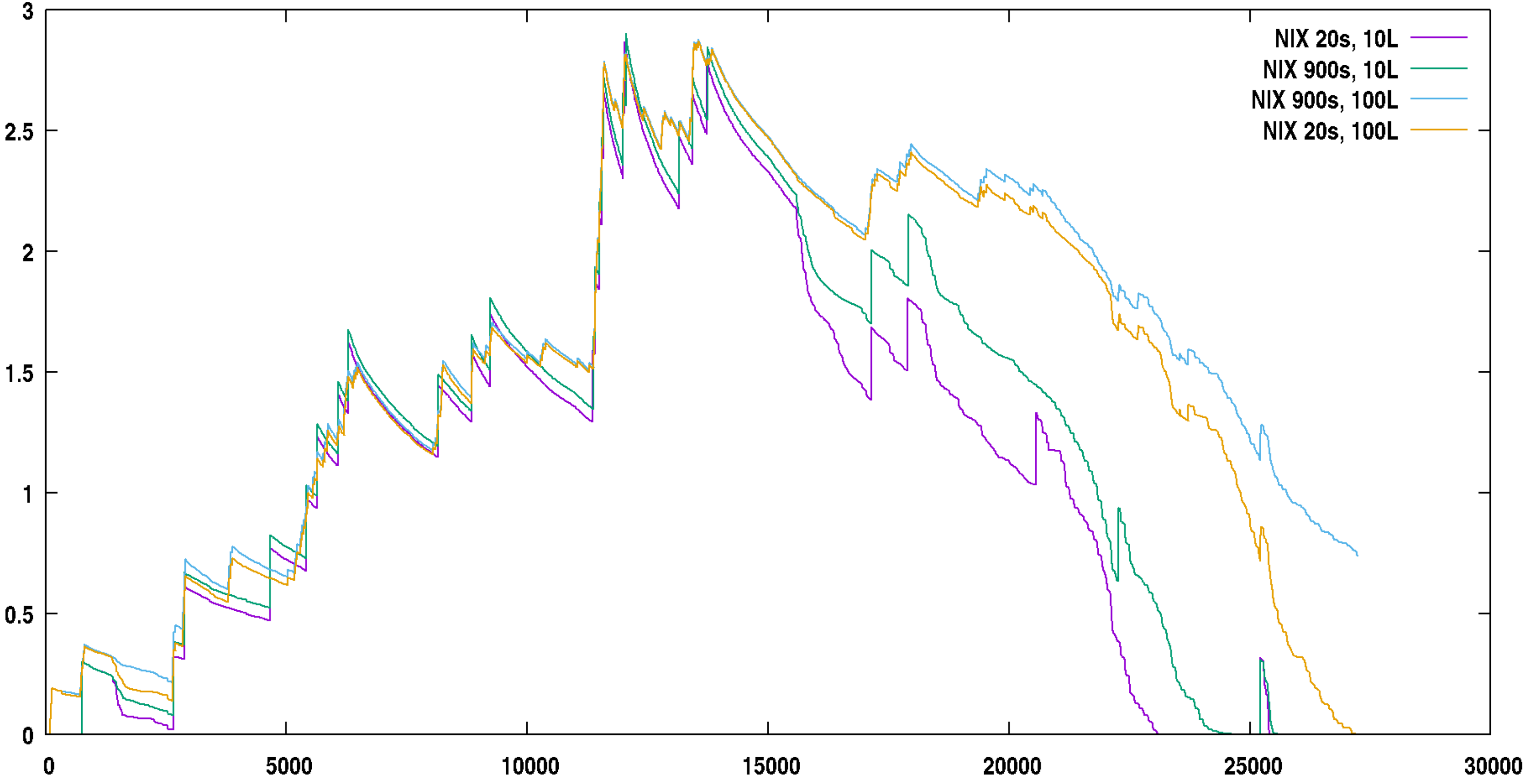
- Basic microstructure to improve albedo
- Improved cold starts by including cold content

Thank you!
Questions or Comments?

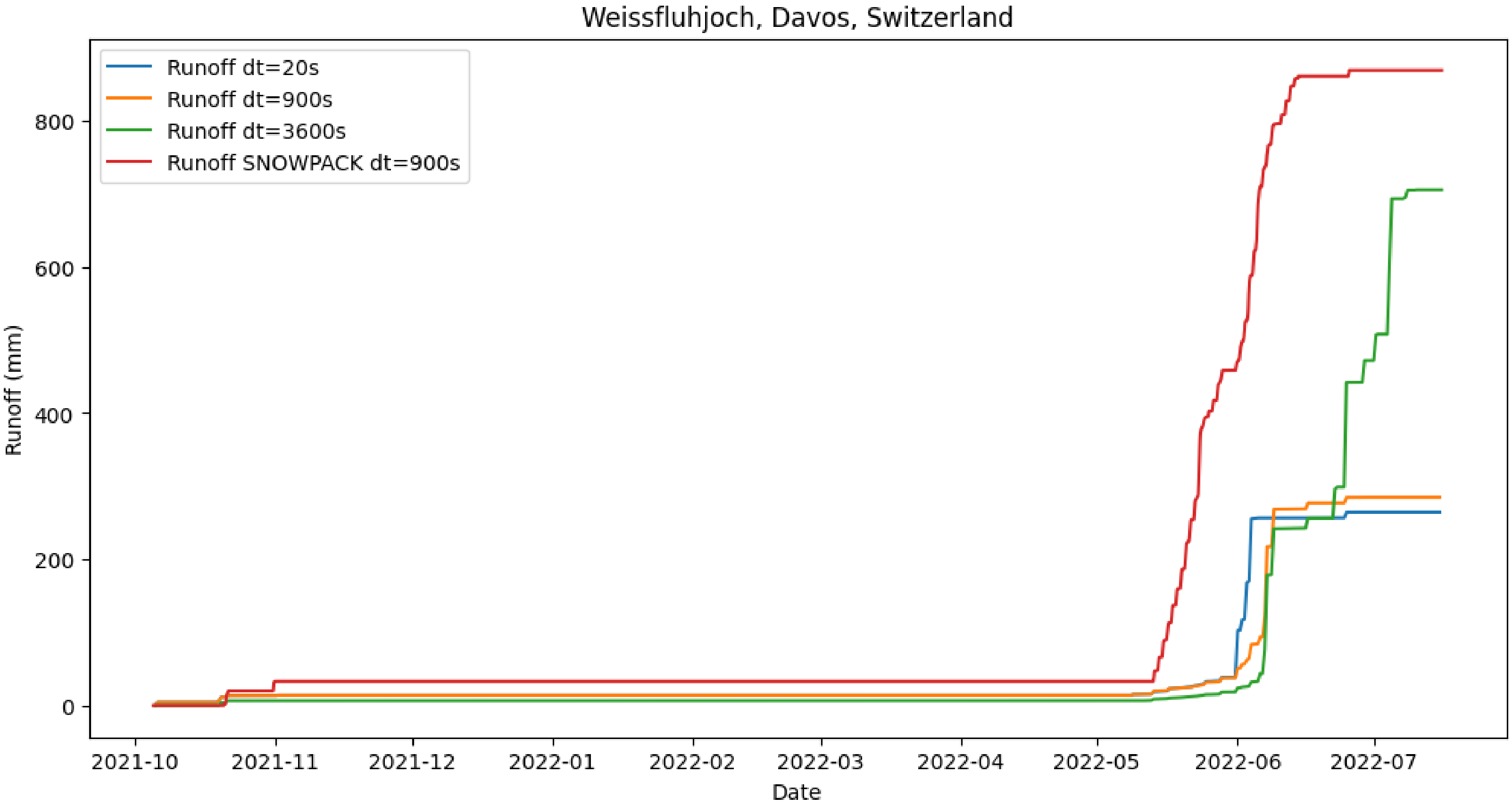
Nander Wever, Sascha Bellaire, Varun Sharma, Michael Lehning, Jean-Marie Bettems et al.

Contact: nander.wever@slf.ch

Time step dependency – old version



Validation – runoff


Weissfluhjoch, Davos, Switzerland
Runoff dt=20s
Runoff dt=900s
Runoff dt=3600s
Runoff SNOWPACK dt=900s
Runoff (mm)
Date