
SuPy Documentation

Release 2020.2.29

Dr Ting Sun, Dr Hamidreza Omidvar and Prof Sue Grimmond

Feb 28, 2020

1	Tutorials	3
1.1	Quickstart of SuPy	3
1.2	Impact Studies Using SuPy	13
1.3	Interaction between SuPy and external models	22
1.4	Modelling Surface Energy Balance at an AmeriFlux Site Using SuPy	31
1.5	Python 101 before SuPy	52
2	Key IO Data Structures in SuPy	53
2.1	Introduction	53
2.2	Input	53
2.3	Output	56
3	API reference	59
3.1	Top-level Functions	59
3.2	Utility Functions	63
3.3	Command-Line Tools	69
3.4	Key Data Structures	70
4	FAQ	129
4.1	I cannot install SuPy following the docs, what is wrong there?	129
4.2	How do I know which version of SuPy I am using?	130
4.3	A <code>kernel may have died exception</code> happened, where did I go wrong?	130
4.4	How can I upgrade SuPy to an up-to-date version?	130
5	Version History	131
5.1	Version Dev	131
5.2	Version 20200202	131
5.3	Version 20190829	132
5.4	Version 2019.7.17	132
5.5	Version 2019.6.8	133
5.6	Version 2019.5.28	133
5.7	Version 2019.4.29	134
5.8	Version 2019.4.17	134
5.9	Version 2019.4.15	134
5.10	Version 2019.3.21	135
5.11	Version 2019.3.14	135
5.12	Version 2019.2.25	136

5.13	Version 2019.2.24	136
5.14	Version 2019.2.19	136
5.15	Version 2019.2.8	137
5.16	Version 2019.1.1 (preview release, 01 Jan 2019)	137
5.17	Version 2018.12.15 (internal test release in December 2018)	138
Index		139

- **What is SuPy?**

SuPy is a Python-enhanced urban climate model with [SUEWS](#) as its computation core.

The scientific rigour in SuPy results is thus gurranteed by SUEWS (see [SUEWS publications](#) and [Parameterisations and sub-models within SUEWS](#)).

Meanwhile, the data analysis ability of SuPy is greatly enhanced by the [Python-based SciPy Stack](#), notably [numpy](#) and [pandas](#). More details are described in [our SuPy paper](#).

- **How to get SuPy?**

SuPy is available on all major platforms (macOS, Windows, Linux) for Python 3.6+ (64-bit only) via [PyPI](#):

```
python3 -m pip install supy --upgrade
```

- **How to use SuPy?**

- Please follow [Quickstart of SuPy](#) and *other tutorials*.
- Please see [API reference](#) for details.
- Please see [FAQ](#) if any issue.

- **How to contribute to SuPy?**

- Add your development via [Pull Request](#)
- Report issues via the [GitHub](#) page.
- Cite [our SuPy paper](#).
- Provide suggestions and feedback.

To familiarise users with SuPy urban climate modelling and to demonstrate the functionality of SuPy, we provide the following tutorials in [Jupyter notebooks](#):

The following section was generated from `docs/source/tutorial/quick-start.ipynb`

1.1 Quickstart of SuPy

This quickstart demonstrates the essential and simplest workflow of `supy` in SUEWS simulation:

1. *load input files*
2. *run simulation*
3. *examine results*

More advanced use of `supy` are available in the [tutorials](#)

Before start, we need to load the following necessary packages.

```
[1]: import matplotlib.pyplot as plt
import supy as sp
import pandas as pd
import numpy as np
from pathlib import Path
get_ipython().run_line_magic('matplotlib', 'inline')

# produce high-quality figures, which can also be set as one of ['svg', 'pdf', 'retina', 'png']
# 'svg' produces high quality vector figures
%config InlineBackend.figure_format = 'svg'

[2]: sp.show_version()
```

```
supy: 2019.8.30dev
supy_driver: 2019a4
```

1.1.1 Load input files

For existing SUEWS users:

First, a path to SUEWS `RunControl.nml` should be specified, which will direct supy to locate input files.

```
[3]: path_runcontrol = Path('../sample_run') / 'RunControl.nml'
```

```
[4]: df_state_init = sp.init_supy(path_runcontrol)
```

```
INFO:root:All cache cleared.
```

A sample `df_state_init` looks below (note that `.T` is used here to a nicer tableform view):

```
[5]: df_state_init.filter(like='method').T
```

```
[5]: grid          98
    var          ind_dim
aerodynamicresistancemethod 0      2
evapmethod                  0      2
emissionsmethod             0      2
netradiationmethod          0      3
roughlenheatmethod          0      2
roughlenmommomethod         0      2
smdmethod                   0      0
stabilitymethod             0      3
storageheatmethod           0      1
waterusemethod              0      0
```

Following the convention of SUEWS, supy loads meteorological forcing (met-forcing) files at the grid level.

```
[6]: grid = df_state_init.index[0]
    df_forcing = sp.load_forcing_grid(path_runcontrol, grid)
```

```
INFO:root:All cache cleared.
```

For new users to SUEWS/SuPy:

To ease the input file preparation, a helper function `load_SampleData` is provided to get the sample input for SuPy simulations

```
[7]: df_state_init, df_forcing = sp.load_SampleData()
```

```
INFO:root:All cache cleared.
```

Overview of SuPy input

`df_state_init`

`df_state_init` includes model Initial state consisting of:

- surface characteristics (e.g., albedo, emissivity, land cover fractions, etc.; full details refer to [SUEWS documentation](#))
- model configurations (e.g., stability; full details refer to [SUEWS documentation](#))

Detailed description of variables in `df_state_init` refers to [SuPy input](#)

Surface land cover fraction information in the sample input dataset:

```
[8]: df_state_init.loc[:, ['bldgh', 'evetreeh', 'dectreeh']]
```

```
[8]: var      bldgh dectreeh evetreeh
ind_dim      0         0         0
grid
98          22.0      13.1      13.1
```

```
[9]: df_state_init.filter(like='sfr')
```

```
[9]: var      sfr
ind_dim  (0,)  (1,)  (2,)  (3,)  (4,)  (5,)  (6,)
grid
98          0.43  0.38  0.001  0.019  0.029  0.001  0.14
```

df_forcing

`df_forcing` includes meteorological and other external forcing information.

Detailed description of variables in `df_forcing` refers to [SuPy input](#).

Below is an overview of forcing variables of the sample data set used in the following simulations.

```
[10]: list_var_forcing = [
        'kdown',
        'Tair',
        'RH',
        'pres',
        'U',
        'rain',
    ]
dict_var_label = {
    'kdown': 'Incoming Solar\n Radiation ( $\mathrm{W \ m^{-2}}$ )',
    'Tair': 'Air Temperature ( $^{\circ}\mathrm{C}$ )',
    'RH': r'Relative Humidity (%)',
    'pres': 'Air Pressure (hPa)',
    'rain': 'Rainfall (mm)',
    'U': 'Wind Speed ( $\mathrm{m \ s^{-1}}$ )'
}
df_plot_forcing_x = df_forcing.loc[:, list_var_forcing].copy().shift(
    -1).dropna(how='any')
df_plot_forcing = df_plot_forcing_x.resample('1h').mean()
df_plot_forcing['rain'] = df_plot_forcing_x['rain'].resample('1h').sum()

axes = df_plot_forcing.plot(
    subplots=True,
    figsize=(8, 12),
    legend=False,
)
fig = axes[0].figure
```

(continues on next page)

(continued from previous page)

```
fig.tight_layout()
fig.autofmt_xdate(bottom=0.2, rotation=0, ha='center')
for ax, var in zip(axes, list_var_forcing):
    ax.set_ylabel(dict_var_label[var])
```

Modification of SuPy input

Given `pandas.DataFrame` as the core data structure of SuPy, all operations, including modification, output, demonstration, etc., on SuPy inputs (`df_state_init` and `df_forcing`) can be done using pandas-based functions/methods.

Specifically, for modification, the following operations are essential:

locating data

Data can be located in two ways, namely: 1. by name via `.loc` <http://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#selection-by-label>‘__’; 2. by position via `.iloc` <http://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#selection-by-position>‘__’.

```
[11]: # view the surface fraction variable: `sfr`
df_state_init.loc[:, 'sfr']
```

```
[11]: ind_dim  (0,)  (1,)  (2,)  (3,)  (4,)  (5,)  (6,)
grid
98          0.43  0.38  0.001  0.019  0.029  0.001  0.14
```

```
[12]: # view the second row of `df_forcing`, which is a pandas Series
df_forcing.iloc[1]
```

```
[12]: iy          2012.000000
id              1.000000
it              0.000000
imin           10.000000
qn            -999.000000
qh            -999.000000
qe            -999.000000
qs            -999.000000
qf            -999.000000
U              4.515000
RH             85.463333
Tair           11.773750
pres          1001.512500
rain           0.000000
kdown          0.153333
snow          -999.000000
ldown          -999.000000
fcld           -999.000000
Wuh            -999.000000
xsmid          -999.000000
lai            -999.000000
kdiff          -999.000000
kdir           -999.000000
wdir           -999.000000
```

(continues on next page)

(continued from previous page)

```
isec          0.000000
Name: 2012-01-01 00:10:00, dtype: float64
```

```
[13]: # view a particular position of `df_forcing`, which is a value
df_forcing.iloc[8,9]
```

```
[13]: 4.455
```

setting new values

Setting new values is very straightforward: after locating the variables/data to modify, just set the new values accordingly:

```
[14]: # modify surface fractions
df_state_init.loc[:, 'sfr'] = [.1, .1, .2, .3, .25, .05, 0]
# check the updated values
df_state_init.loc[:, 'sfr']
```

```
[14]: ind_dim  (0,)  (1,)  (2,)  (3,)  (4,)  (5,)  (6,)
grid
98         0.1   0.1   0.2   0.3  0.25  0.05   0.0
```

1.1.2 Run simulations

Once met-forcing (via `df_forcing`) and initial conditions (via `df_state_init`) are loaded in, we call `sp.run_supy` to conduct a SUEWS simulation, which will return two pandas DataFrames: `df_output` and `df_state`.

```
[15]: df_output, df_state_final = sp.run_supy(df_forcing, df_state_init)
```

```
INFO:root:=====
INFO:root:Simulation period:
INFO:root:  Start: 2012-01-01 00:05:00
INFO:root:  End: 2013-01-01 00:00:00
INFO:root:
INFO:root:No. of grids: 1
INFO:root:SuPy is running in serial mode
INFO:root:Execution time: 3.2 s
INFO:root:=====
```

df_output

`df_output` is an ensemble output collection of major SUEWS output groups, including:

- SUEWS: the essential SUEWS output variables
- DailyState: variables of daily state information
- snow: snow output variables (effective when `snowuse = 1` set in `df_state_init`)

Detailed description of variables in `df_output` refers to [SuPy output](#)

```
[16]: df_output.columns.levels[0]
[16]: Index(['SUEWS', 'snow', 'RSL', 'DailyState'], dtype='object', name='group')
```

df_state_final

df_state_final is a DataFrame for holding:

1. all model states if save_state is set to True when calling sp.run_supy and supy may run significantly slower for a large simulation;
2. or, only the final state if save_state is set to False (the default setting) in which mode supy has a similar performance as the standalone compiled SUEWS executable.

Entries in df_state_final have the same data structure as df_state_init and can thus be used for other SUEWS simulations starting at the timestamp as in df_state_final.

Detailed description of variables in df_state_final refers to [SuPy output](#)

```
[17]: df_state_final.T.head()
[17]: datetime          2012-01-01 00:05:00  2013-01-01 00:05:00
grid                    98                    98
var                    ind_dim
ah_min                (0,)          15.0          15.0
                    (1,)          15.0          15.0
ah_slope_cooling      (0,)           2.7           2.7
                    (1,)           2.7           2.7
ah_slope_heating      (0,)           2.7           2.7
```

1.1.3 Examine results

Thanks to the functionality inherited from pandas and other packages under the [PyData](#) stack, compared with the standard SUEWS simulation workflow, supy enables more convenient examination of SUEWS results by statistics calculation, resampling, plotting (and many more).

Output structure

df_output is organised with MultiIndex (grid,timestamp) and (group,variable) as index and columns, respectively.

```
[18]: df_output.head()
[18]: group          SUEWS          Kup          Ldown          Lup          \
var          Kdown          Kup          Ldown          Lup
grid datetime
98  2012-01-01 00:05:00  0.153333  0.021237  344.310184  372.417244
    2012-01-01 00:10:00  0.153333  0.021237  344.310184  372.417244
    2012-01-01 00:15:00  0.153333  0.021237  344.310184  372.417244
    2012-01-01 00:20:00  0.153333  0.021237  344.310184  372.417244
    2012-01-01 00:25:00  0.153333  0.021237  344.310184  372.417244

group          Tsurf          QN          QF          QS          \
var          Tsurf          QN          QF          QS
grid datetime
```

(continues on next page)

(continued from previous page)

```

98  2012-01-01 00:05:00  11.775859 -27.974963  40.569300 -45.253674
    2012-01-01 00:10:00  11.775859 -27.974963  39.719681 -45.070905
    2012-01-01 00:15:00  11.775859 -27.974963  38.870062 -44.895750
    2012-01-01 00:20:00  11.775859 -27.974963  38.020443 -44.727894
    2012-01-01 00:25:00  11.775859 -27.974963  37.170824 -44.567032

group
var          QH          QE  ...  DailyState  \
grid datetime
98  2012-01-01 00:05:00  57.807360  0.040651  ...          NaN
    2012-01-01 00:10:00  56.775225  0.040398  ...          NaN
    2012-01-01 00:15:00  55.750704  0.040145  ...          NaN
    2012-01-01 00:20:00  54.733480  0.039895  ...          NaN
    2012-01-01 00:25:00  53.723248  0.039645  ...          NaN

group
var          DensSnow_Bldgs DensSnow_EveTr DensSnow_DecTr  \
grid datetime
98  2012-01-01 00:05:00          NaN          NaN          NaN
    2012-01-01 00:10:00          NaN          NaN          NaN
    2012-01-01 00:15:00          NaN          NaN          NaN
    2012-01-01 00:20:00          NaN          NaN          NaN
    2012-01-01 00:25:00          NaN          NaN          NaN

group
var          DensSnow_Grass DensSnow_BSoil DensSnow_Water  a1  a2  \
grid datetime
98  2012-01-01 00:05:00          NaN          NaN          NaN NaN NaN
    2012-01-01 00:10:00          NaN          NaN          NaN NaN NaN
    2012-01-01 00:15:00          NaN          NaN          NaN NaN NaN
    2012-01-01 00:20:00          NaN          NaN          NaN NaN NaN
    2012-01-01 00:25:00          NaN          NaN          NaN NaN NaN

group
var          a3
grid datetime
98  2012-01-01 00:05:00 NaN
    2012-01-01 00:10:00 NaN
    2012-01-01 00:15:00 NaN
    2012-01-01 00:20:00 NaN
    2012-01-01 00:25:00 NaN

[5 rows x 340 columns]

```

Here we demonstrate several typical scenarios for SUEWS results examination.

The essential SUEWS output collection is extracted as a separate variable for easier processing in the following sections. More [advanced slicing techniques](#) are available in [pandas documentation](#).

```
[19]: df_output_suews = df_output['SUEWS']
```

Statistics Calculation

We can use `.describe()` method for a quick overview of the key surface energy balance budgets.

```
[20]: df_output_suews.loc[:, ['QN', 'QS', 'QH', 'QE', 'QF']].describe()

[20]: var      QN      QS      QH      QE  \
count  105408.000000  105408.000000  105408.000000  105408.000000
mean     39.319914    -15.810252     88.755915     45.857651
std     130.797388     53.953592     69.057335     54.363737
min     -86.212629    -87.482114    -114.375930     0.000081
25%     -42.028676    -48.084784     41.334831     1.266435
50%     -25.694495    -40.948527     75.221473     22.980817
75%      73.254869     -2.433109     126.971057     75.607932
max      662.453669     239.033524     371.051513     378.152626

var      QF
count  105408.000000
mean     79.068259
std     30.855099
min     26.506045
25%     50.520548
50%     82.815455
75%    104.577731
max    162.947824
```

Plotting

Basic example

Plotting is very straightforward via the `.plot` method bounded with `pandas.DataFrame`. Note the usage of `loc` for to slices of the output `DataFrame`.

```
[21]: # a dict for better display variable names
dict_var_disp = {
    'QN': '$Q^{*}$',
    'QS': r'$\Delta Q_{S}$',
    'QE': '$Q_{E}$',
    'QH': '$Q_{H}$',
    'QF': '$Q_{F}$',
    'Kdown': r'$K_{\downarrow}$',
    'Kup': r'$K_{\uparrow}$',
    'Ldown': r'$L_{\downarrow}$',
    'Lup': r'$L_{\uparrow}$',
    'Rain': '$P$',
    'Irr': '$I$',
    'Evap': '$E$',
    'RO': '$R$',
    'TotCh': '$\Delta S$',
}
```

Quick look at the simulation results:

```
[22]: ax_output = df_output_suews\
    .loc[grid]\
    .loc['2012 6 1':'2012 6 7',
        ['QN', 'QS', 'QE', 'QH', 'QF']]\
    .rename(columns=dict_var_disp)\
    .plot()
ax_output.set_xlabel('Date')
```

(continues on next page)

(continued from previous page)

```
ax_output.set_ylabel('Flux ($ \mathrm{W \ m^{-2}}$)')
ax_output.legend()
```

```
[22]: <matplotlib.legend.Legend at 0x7f8ec14566a0>
```

More examples

Below is a more complete example for examination of urban energy balance over the whole summer (June to August).

```
[23]: # energy balance
ax_output = df_output_suews.loc[grid]\
    .loc['2012 6':'2012 8', ['QN', 'QS', 'QE', 'QH', 'QF']]\
    .rename(columns=dict_var_disp)\
    .plot(
        figsize=(10, 3),
        title='Surface Energy Balance',
    )
ax_output.set_xlabel('Date')
ax_output.set_ylabel('Flux ($ \mathrm{W \ m^{-2}}$)')
ax_output.legend()
```

```
[23]: <matplotlib.legend.Legend at 0x7f8ed08c3278>
```

Resampling

The suggested runtime/simulation frequency of SUEWS is 300 s, which usually results a large output and may be over-weighted for storage and analysis. Also, you may feel apparent slowdown in producing the above figure as a large amount of data were used for the plotting. To slim down the result size for analysis and output, we can resample the default output very easily.

```
[24]: rsmpld = df_output_suews.loc[grid].resample('1d')
# daily mean values
df_1d_mean = rsmpld.mean()
# daily sum values
df_1d_sum = rsmpld.sum()
```

We can then re-examine the above energy balance at hourly scale and plotting will be significantly faster.

```
[25]: # energy balance
ax_output = df_1d_mean\
    .loc[:, ['QN', 'QS', 'QE', 'QH', 'QF']]\
    .rename(columns=dict_var_disp)\
    .plot(
        figsize=(10, 3),
        title='Surface Energy Balance',
    )
ax_output.set_xlabel('Date')
ax_output.set_ylabel('Flux ($ \mathrm{W \ m^{-2}}$)')
ax_output.legend()
```

```
[25]: <matplotlib.legend.Legend at 0x7f8ec1623908>
```

Then we use the hourly results for other analyses.

```
[26]: # radiation balance
ax_output = df_ld_mean\
    .loc[:, ['QN', 'Kdown', 'Kup', 'Ldown', 'Lup']]\
    .rename(columns=dict_var_disp)\
    .plot(
        figsize=(10, 3),
        title='Radiation Balance',
    )
ax_output.set_xlabel('Date')
ax_output.set_ylabel('Flux ($ \mathrm{W \ m^{-2}}$)')
ax_output.legend()
```

```
[26]: <matplotlib.legend.Legend at 0x7f8eb149a0b8>
```

```
[27]: # water balance
ax_output = df_ld_sum\
    .loc[:, ['Rain', 'Irr', 'Evap', 'RO', 'TotCh']]\
    .rename(columns=dict_var_disp)\
    .plot(
        figsize=(10, 3),
        title='Surface Water Balance',
    )
ax_output.set_xlabel('Date')
ax_output.set_ylabel('Water amount (mm)')
ax_output.legend()
```

```
[27]: <matplotlib.legend.Legend at 0x7f8ef208bc18>
```

Get an overview of partitioning in energy and water balance at monthly scales:

```
[28]: # get a monthly Resampler
df_plot=df_output_suews.loc[grid].copy()
df_plot.index=df_plot.index.set_names('Month')
rsmplM = df_plot\
    .shift(-1)\
    .dropna(how='all')\
    .resample('1M', kind='period')
# mean values
df_1M_mean = rsmplM.mean()
# sum values
df_1M_sum = rsmplM.sum()
```

```
[29]: # month names
name_mon = [x.strftime('%b') for x in rsmplM.groups]
# create subplots showing two panels together
fig, axes = plt.subplots(2, 1, sharex=True)
# surface energy balance
df_1M_mean\
    .loc[:, ['QN', 'QS', 'QE', 'QH', 'QF']]\
    .rename(columns=dict_var_disp)\
    .plot(
        ax=axes[0], # specify the axis for plotting
        figsize=(10, 6), # specify figure size
        title='Surface Energy Balance',
        kind='bar',
```

(continues on next page)

(continued from previous page)

```

    )
    # surface water balance
    df_1M_sum\
        .loc[:, ['Rain', 'Irr', 'Evap', 'RO', 'TotCh']]\
        .rename(columns=dict_var_disp)\
        .plot(
            ax=axes[1], # specify the axis for plotting
            title='Surface Water Balance',
            kind='bar'
        )

    # annotations
    axes[0].set_ylabel('Mean Flux ($ \mathrm{W \ m^{-2}}$)')
    axes[0].legend()
    axes[1].set_xlabel('Month')
    axes[1].set_ylabel('Total Water Amount (mm)')
    axes[1].xaxis.set_ticklabels(name_mon, rotation=0)
    axes[1].legend()

```

```
[29]: <matplotlib.legend.Legend at 0x7f8ec195a9b0>
```

Output

The supy output can be saved as txt files for further analysis using supy function `save_supy`.

```
[30]: list_path_save = sp.save_supy(df_output, df_state_final, path_runcontrol=path_
    ↪runcontrol)
```

```
[31]: for file_out in list_path_save:
    print(file_out.name)

Kc98_2012_SUEWS_5.txt
Kc98_2012_snow_5.txt
Kc98_2012_RSL_5.txt
Kc98_2012_DailyState.txt
Kc98_2012_SUEWS_60.txt
Kc98_2012_snow_60.txt
Kc98_2012_RSL_60.txt
InitialConditionsKc98_2013_EndofRun.nml

```

End of doc/tutorial/quick-start.ipynb

The following section was generated from docs/source/tutorial/impact-studies-parallel.ipynb

1.2 Impact Studies Using SuPy

1.2.1 Aim

In this tutorial, we aim to perform sensitivity analysis using `supy` in a parallel mode to investigate the impacts on urban climate of

1. surface properties: the physical attributes of land covers (e.g., albedo, water holding capacity, etc.)

2. background climate: longterm meteorological conditions (e.g., air temperature, precipitation, etc.)

1.2.2 Prepare supy for the parallel mode

load supy and sample dataset

```
[1]: from dask import delayed
from dask import dataframe as dd
import os
import supy as sp
import seaborn as sns
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
from time import time
import logging
logging.basicConfig()
logging.getLogger().setLevel(logging.WARNING)

get_ipython().run_line_magic('matplotlib', 'inline')
# produce high-quality figures, which can also be set as one of ['svg', 'pdf', 'retina', 'png']
# 'svg' produces high quality vector figures
%config InlineBackend.figure_format = 'svg'
# show supy version info
sp.show_version()

supy: 2019.8.30dev
supy_driver: 2019a4
```

```
[2]: # load sample datasets
df_state_init, df_forcing = sp.load_SampleData()
# perform an example run to get output samples for later use
df_output, df_state_final = sp.run_supy(df_forcing, df_state_init)
```

Parallel setup for supy using dask

Given the nature of impact studies that requires multiple independent models with selected parameters/variables varying across the setups, such simulations well fall into the scope of so-called **embarrassingly parallel computation** that is fully supported by dask. Also, as supy is readily built on the data structure `pandas.DataFrame`, we can fairly easily transfer it to the dask framework for parallel operations thanks to `dask.dataframe` <<http://docs.dask.org/en/latest/dataframe.html>>__, a specialized dataframe extending `pandas.DataFrame`'s ability in parallel operations.

Prior to version 2019.5, for a given forcing dataset `df_forcing`, supy would loop over the grids in a `df_state_init` to conduct simulations. Since version 2019.5, supy has been using the `dask.dataframe` to gain the parallel benefits through its parallelized `apply` method.

`dask.dataframe` essentially divides the work into pieces for parallel operations. As such, depending on the number of processors in your computer, it would be more efficient to set the partition number as the multipliers of CPU numbers.

```
[3]: import platform
import psutil
list_info=['machine','system','mac_ver','processor']
for info in list_info:
    info_x=getattr(platform,info)()
    print(info,':',info_x)
cpu_count=psutil.cpu_count()
print('number of CPU processors:',cpu_count)
mem_size=psutil.virtual_memory().total/1024**3
print('memory size (GB):',mem_size)
```

```
machine : x86_64
system : Darwin
mac_ver : ('10.14.6', ('', '', ''), 'x86_64')
processor : i386
number of CPU processors: 12
memory size (GB): 32.0
```

To demonstrate the parallelization, we simply duplicate the contents in `df_state_init` to make it seemingly large. Note we intentionally choose 24 as the number for copies to accompany the power of CPU.

Before we move on to the parallel mode, we perform a simulation in the traditional serial way to see the baseline performance.

Baseline serial run

```
[4]: # just run for 30 days
df_forcing_part = df_forcing.iloc[:288*30]
df_state_init_mgrid = df_state_init.copy()
# construct a multi-grid `df_state_init`
for i in range(24-1):
    df_state_init_mgrid = df_state_init_mgrid.append(
        df_state_init, ignore_index=True)
# perform a serial run
t0 = time()
for i in range(24-1):
    xx = sp.run_supy(df_forcing_part, df_state_init_mgrid.iloc[[i]])
t1 = time()
t_ser = t1-t0
logging.warning(f'Execution time: {t_ser:.2f} s')
```

```
WARNING:root:Execution time: 7.61 s
```

Parallel run

```
[5]: # parallel run is enabled in supy by default
t0 = time()
xx = sp.run_supy(df_forcing_part, df_state_init_mgrid)
t1 = time()
t_par = t1-t0
logging.warning(f'Execution time: {t_par:.2f} s')
```

```
WARNING:root:Execution time: 4.13 s
```

Benchmark test

Note: this test may take a considerably long time depending on the machine performance

```
[6]: # different running length
list_sim_len = [
    day * 288 for day in [30, 90, 120, 150, 180, 270, 365, 365 * 2, 365 * 3]
]

# number of test grids
n_grid = 12

# construct a multi-grid `df_state_init`
df_state_init_m = df_state_init.copy()
for i in range(n_grid - 1):
    df_state_init_m = df_state_init_m.append(df_state_init, ignore_index=True)

# construct a longer `df_forcing` for three years
df_forcing_m = pd.concat([df_forcing for i in range(3)])
df_forcing_m.index = pd.date_range(df_forcing.index[0],
                                    freq=df_forcing.index.freq,
                                    periods=df_forcing_m.index.size)

dict_time_ser = dict()
dict_time_par = dict()
for sim_len in list_sim_len:
    df_forcing_part = df_forcing_m.iloc[:sim_len]
    logging.warning(f'Sim days: {sim_len / 288}')
    logging.warning(f'No. of grids: {df_state_init_m.shape[0]}')
    # serial run
    logging.warning('serial:')
    t0 = time()
    for i in range(df_state_init_m.shape[0]):
        sp.run_supy(df_forcing_part, df_state_init_m.iloc[[i]])
    t1 = time()
    t_test = t1 - t0
    logging.warning(f'Execution time: {t_test:.2f} s')

    dict_time_ser.update({sim_len: t_test})

    # parallel run
    logging.warning('parallel:')
    t0 = time()
    sp.run_supy(df_forcing_part, df_state_init_m)
    t1 = time()
    t_test = t1 - t0
    logging.warning(f'Execution time: {t_test:.2f} s\n')

    dict_time_par.update({sim_len: t_test})
```

```
WARNING:root:Sim days: 30.0
WARNING:root:No. of grids: 12
WARNING:root:serial:
WARNING:root:Execution time: 4.07 s
WARNING:root:parallel:
WARNING:root:Execution time: 2.17 s

WARNING:root:Sim days: 90.0
```

(continues on next page)

(continued from previous page)

```
WARNING:root:No. of grids: 12
WARNING:root:serial:
WARNING:root:Execution time: 8.99 s
WARNING:root:parallel:
WARNING:root:Execution time: 4.07 s

WARNING:root:Sim days: 120.0
WARNING:root:No. of grids: 12
WARNING:root:serial:
WARNING:root:Execution time: 10.88 s
WARNING:root:parallel:
WARNING:root:Execution time: 5.08 s

WARNING:root:Sim days: 150.0
WARNING:root:No. of grids: 12
WARNING:root:serial:
WARNING:root:Execution time: 13.29 s
WARNING:root:parallel:
WARNING:root:Execution time: 5.80 s

WARNING:root:Sim days: 180.0
WARNING:root:No. of grids: 12
WARNING:root:serial:
WARNING:root:Execution time: 15.47 s
WARNING:root:parallel:
WARNING:root:Execution time: 6.70 s

WARNING:root:Sim days: 270.0
WARNING:root:No. of grids: 12
WARNING:root:serial:
WARNING:root:Execution time: 22.23 s
WARNING:root:parallel:
WARNING:root:Execution time: 9.84 s

WARNING:root:Sim days: 365.0
WARNING:root:No. of grids: 12
WARNING:root:serial:
WARNING:root:Execution time: 29.06 s
WARNING:root:parallel:
WARNING:root:Execution time: 13.65 s

WARNING:root:Sim days: 730.0
WARNING:root:No. of grids: 12
WARNING:root:serial:
WARNING:root:Execution time: 67.05 s
WARNING:root:parallel:
WARNING:root:Execution time: 28.66 s

WARNING:root:Sim days: 1095.0
WARNING:root:No. of grids: 12
WARNING:root:serial:
WARNING:root:Execution time: 98.66 s
WARNING:root:parallel:
WARNING:root:Execution time: 48.68 s
```

```
[7]: df_benchmark = pd.DataFrame([
        dict_time_par,
        dict_time_ser])\
    .transpose()\
    .rename(columns={0: 'parallel', 1: 'serial'})
idx_bmk = (df_benchmark.index / 288).astype(int)
df_benchmark.index = idx_bmk.set_names('Length of Simulation Period (day)')

# calculate execution time ratio between parallel and serial runs
ser_ratio = df_benchmark['parallel'] / df_benchmark['serial']
df_benchmark = df_benchmark.assign(ratio=ser_ratio)\
    .rename(columns={'ratio': 'ratio (=p/s, right)'})

# show execution times and ratio on plot
ax = df_benchmark.plot(secondary_y='ratio (=p/s, right)',
                        marker='o',
                        fillstyle='none')
ax.set_ylabel('Execution Time (s)')

lines = ax.get_lines() + ax.right_ax.get_lines()
ax.legend(lines, [l.get_label() for l in lines], loc='best')

ax.right_ax.set_ylabel('Execution Ratio (=p/s)', color='C2')
ax.right_ax.spines['right'].set_color('C2')
ax.right_ax.tick_params(axis='y', colors='C2')
```

1.2.3 Surface properties: surface albedo

Examine the default albedo values loaded from the sample dataset

```
[8]: df_state_init.alb
```

[8]:	ind_dim	(0,)	(1,)	(2,)	(3,)	(4,)	(5,)	(6,)
	grid							
	98	0.12	0.15	0.12	0.18	0.21	0.21	0.1

Copy the initial condition DataFrame to have a *clean slate* for our study

Note: `DataFrame.copy()` defaults to deepcopy

```
[9]: df_state_init_test = df_state_init.copy()
```

Set the Bldg land cover to 100% for this study

```
[10]: df_state_init_test.sfr = 0
df_state_init_test.loc[:, ('sfr', '(1,)')] = 1
df_state_init_test.sfr
```

```
[10]: ind_dim  (0,)  (1,)  (2,)  (3,)  (4,)  (5,)  (6,)
      grid
      98          0      1      0      0      0      0      0
```

Construct a `df_state_init_x` dataframe to perform `supy` simulation with specified albedo

```
[11]: # create a `df_state_init_x` with different surface properties
      n_test = 48
      list_alb_test = np.linspace(0.1, 0.8, n_test).round(2)
      df_state_init_x = df_state_init_test.append(
          [df_state_init_test]*(n_test-1), ignore_index=True)

      # here we modify surface albedo
      df_state_init_x.loc[:, ('alb', '(1,)')] = list_alb_test
```

Conduct simulations with `supy`

```
[12]: df_forcing_part = df_forcing.loc['2012 01':'2012 07']
      df_res_alb_test, df_state_final_x = sp.run_supy(df_forcing_part, df_state_init_x)
```

Examine the simulation results

```
[13]: # choose results of July 2012 for analysis
      df_res_alb_test_july = df_res_alb_test.SUEWS.unstack(0).loc['2012 7']
      df_res_alb_T2_stat = df_res_alb_test_july.T2.describe()
      df_res_alb_T2_diff = df_res_alb_T2_stat.transform(
          lambda x: x - df_res_alb_T2_stat.iloc[:, 0])
      df_res_alb_T2_diff.columns = list_alb_test - list_alb_test[0]

[14]: ax_temp_diff = df_res_alb_T2_diff.loc[['max', 'mean', 'min']].T.plot()
      ax_temp_diff.set_ylabel('$\Delta T_2$ ($^{\circ}$C)')
      ax_temp_diff.set_xlabel(r'$\Delta\alpha$')
      ax_temp_diff.margins(x=0.2, y=0.2)
```

Why a bi-linear $\Delta\alpha - \Delta T_{2,max}$ relationship?

Although the relations for mean and minimum T_2 demonstrate single linear patterns, the one for maximum T_2 , interestingly, consists of two linear sections.

```
[15]: df_t2 = df_res_alb_test_july.T2
      df_t2.columns = list_alb_test

      df_t2.idxmax().unique()

[15]: array(['2012-07-25T13:35:00.000000000', '2012-07-25T15:30:00.000000000'],
      dtype='datetime64[ns]')
```

By looking into the peaking times of $T_{2,max}$, we see a shift in the peaking times from 13:35 to 15:30 on 2012-07-25 as albedo increases. Taking the two ending cases, $\alpha = 0.1$ and $\alpha = 0.8$, we see diurnal cycles of T_2 evolves according to the albedo: peak is delayed as albedo increases.

```
[16]: df_t2.loc['2012-07-25'].iloc[:, [0, -1]].plot()
[16]: <matplotlib.axes._subplots.AxesSubplot at 0x7f9630da73c8>
```

Furthermore, when the $\Delta\alpha - \Delta T_2$ relations at the two peaking times are shown below, we can see the bi-linear relation based on the $T_{2,max}$ values for the July 2012 is actually composed of two linear relations at different times under different peaking scenarios.

```
[17]: ax_t2_max=df_t2.loc['2012-07-25 13:35':'2012-07-25 15:30'].iloc[[0, -1]].T.plot()
ax_t2_max.set_xlabel(r'$\alpha$')
ax_t2_max.set_ylabel('$T_{2,max}$ ($^{\circ}$C)')
[17]: Text(0, 0.5, '$T_{2,max}$ ($^{\circ}$C)')
```

1.2.4 Background climate: air temperature

Examine the monthly climatology of air temperature loaded from the sample dataset

```
[18]: df_plot = df_forcing.Tair.iloc[: -1].resample('1m').mean()
ax_temp = df_plot.plot.bar(color='tab:blue')
ax_temp.set_xticklabels(df_plot.index.strftime('%b'))
ax_temp.set_ylabel('Mean Air Temperature ($^{\circ}$C)')
ax_temp.set_xlabel('Month')
ax_temp
[18]: <matplotlib.axes._subplots.AxesSubplot at 0x7f9630d7f668>
```

Construct a function to perform parallel supy simulation with specified `diff_airtemp_test`: the difference in air temperature between the one used in simulation and loaded from sample dataset.

Note: forcing data “df_forcing” has different data structure from “df_state_init”; so we need to modify “run_supy_mgrids” to implement a “run_supy_mclims” for different climate scenarios

Let’s start the implementation of `run_supy_mclims` with a small problem of four forcing groups (i.e., climate scenarios), where the air temperatures differ from the baseline scenario with a constant bias.

```
[19]: # save loaded sample datasets
df_forcing_part_test = df_forcing.loc['2012 1':'2012 7'].copy()
df_state_init_test = df_state_init.copy()

[20]: # create a dict with four forcing conditions as a test
n_test = 4
list_TairDiff_test = np.linspace(0., 2, n_test).round(2)
dict_df_forcing_x = {
    tairdiff: df_forcing_part_test.copy()
    for tairdiff in list_TairDiff_test
}
for tairdiff in dict_df_forcing_x:
```

(continues on next page)

(continued from previous page)

```

    dict_df_forcing_x[tairdiff].loc[:, 'Tair'] += tairdiff

dd_forcing_x = {
    k: delayed(sp.run_supy)(df, df_state_init_test)[0]
    for k, df in dict_df_forcing_x.items()}

df_res_tairdiff_test0 = delayed(pd.concat)(
    dd_forcing_x,
    keys=list_TairDiff_test,
    names=['tairdiff'],
)

```

```

[21]: # test the performance of a parallel run
t0 = time()
df_res_tairdiff_test = df_res_tairdiff_test0\
    .compute(scheduler='threads')\
    .reset_index('grid', drop=True)
t1 = time()
t_par = t1 - t0
print(f'Execution time: {t_par:.2f} s')

```

Execution time: 6.83 s

```

[22]: # function for multi-climate `run_supy`
# wrapping the above code into one
def run_supy_mclims(df_state_init, dict_df_forcing_mclims):
    dd_forcing_x = {
        k: delayed(sp.run_supy)(df, df_state_init_test)[0]
        for k, df in dict_df_forcing_x.items()}
    df_output_mclims0 = delayed(pd.concat)(
        dd_forcing_x,
        keys=list(dict_df_forcing_x.keys()),
        names=['clm'],
    ).compute(scheduler='threads')
    df_output_mclims = df_output_mclims0.reset_index('grid', drop=True)

    return df_output_mclims

```

Construct dict_df_forcing_x with multiple forcing DataFrames

```

[23]: # save loaded sample datasets
df_forcing_part_test = df_forcing.loc['2012 1':'2012 7'].copy()
df_state_init_test = df_state_init.copy()

# create a dict with a number of forcing conditions
n_test = 24 # can be set with a smaller value to save simulation time
list_TairDiff_test = np.linspace(0., 2, n_test).round(2)
dict_df_forcing_x = {
    tairdiff: df_forcing_part_test.copy()
    for tairdiff in list_TairDiff_test}
for tairdiff in dict_df_forcing_x:
    dict_df_forcing_x[tairdiff].loc[:, 'Tair'] += tairdiff

```

Perform simulations

```
[24]: # run parallel simulations using `run_supy_mclims`
t0 = time()
df_airtemp_test_x = run_supy_mclims(df_state_init_test, dict_df_forcing_x)
t1 = time()
t_par = t1-t0
print(f'Execution time: {t_par:.2f} s')

Execution time: 38.52 s
```

Examine the results

```
[25]: df_airtemp_test = df_airtemp_test_x.SUEWS.unstack(0)
df_temp_diff=df_airtemp_test.T2.transform(lambda x: x - df_airtemp_test.T2[0.0])
df_temp_diff_ana=df_temp_diff.loc['2012 7']
df_temp_diff_stat=df_temp_diff_ana.describe().loc[['max', 'mean', 'min']].T

[26]: ax_temp_diff_stat=df_temp_diff_stat.plot()
ax_temp_diff_stat.set_ylabel('$\\Delta T_2$ ($^{\\circ}$C)')
ax_temp_diff_stat.set_xlabel('$\\Delta T_a$ ($^{\\circ}$C)')
ax_temp_diff_stat.set_aspect('equal')
```

The T_2 results indicate the increased T_a has different impacts on the T_2 metrics (minimum, mean and maximum) but all increase linearly with T_a . The maximum T_2 has the stronger response compared to the other metrics.

End of doc/tutorial/impact-studies-parallel.ipynb

The following section was generated from docs/source/tutorial/external-interaction.ipynb

1.3 Interaction between SuPy and external models

1.3.1 Introduction

SUEWS can be coupled to other models that provide or require forcing data using the SuPy single timestep running mode. We demonstrate this feature with a simple online anthropogenic heat flux model.

Anthropogenic heat flux (Q_F) is an additional term to the surface energy balance in urban areas associated with human activities (Gabey et al., 2018; Grimmond, 1992; Nie et al., 2014; 2016; Sailor, 2011). In most cities, the largest emission source is from buildings (Hamilton et al., 2009; Iamarino et al., 2011; Sailor, 2011) and is high dependent on outdoor ambient air temperature.

load necessary packages

```
[1]: import supy as sp
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import seaborn as sns
```

(continues on next page)

(continued from previous page)

```
%matplotlib inline
# produce high-quality figures, which can also be set as one of ['svg', 'pdf', 'retina
→', 'png']
# 'svg' produces high quality vector figures
from IPython.display import set_matplotlib_formats
set_matplotlib_formats('svg')
sp.show_version()

supy: 2019.8.30dev
supy_driver: 2019a4
```

run SUEWS with default settings

```
[2]: # load sample run dataset
df_state_init, df_forcing = sp.load_SampleData()
df_state_init_def=df_state_init.copy()
# set QF as zero for later comparison
df_forcing_def=df_forcing.copy()
grid=df_state_init_def.index[0]
df_state_init_def.loc[:, 'emissionsmethod']=0
df_forcing_def['qf']=0
# run supy
df_output, df_state = sp.run_supy(df_forcing_def, df_state_init_def)
df_output_def = df_output.loc[grid, 'SUEWS']
```

```
INFO:root:All cache cleared.
INFO:root:=====
INFO:root:Simulation period:
INFO:root:  Start: 2012-01-01 00:05:00
INFO:root:  End: 2013-01-01 00:00:00
INFO:root:
INFO:root:No. of grids: 1
INFO:root:SuPy is running in serial mode
INFO:root:Execution time: 3.0 s
INFO:root:=====
```

```
[3]: df_output_def.columns
```

```
[3]: Index(['Kdown', 'Kup', 'Ldown', 'Lup', 'Tsurf', 'QN', 'QF', 'QS', 'QH', 'QE',
        'QHlumps', 'QElumps', 'QHresis', 'Rain', 'Irr', 'Evap', 'RO', 'TotCh',
        'SurfCh', 'State', 'NWtrState', 'Drainage', 'SMD', 'FlowCh', 'AddWater',
        'ROSoil', 'ROPipe', 'ROImp', 'ROVeg', 'ROWater', 'WUInt', 'WUEveTr',
        'WUDecTr', 'WUGrass', 'SMDPaved', 'SMDBldgs', 'SMDEveTr', 'SMDDecTr',
        'SMDGrass', 'SMDBSoil', 'StPaved', 'StBldgs', 'StEveTr', 'StDecTr',
        'StGrass', 'StBSoil', 'StWater', 'Zenith', 'Azimuth', 'AlbBulk', 'Fcld',
        'LAI', 'z0m', 'zdm', 'UStar', 'Lob', 'RA', 'RS', 'Fc', 'FcPhoto',
        'FcRespi', 'FcMetab', 'FcTraff', 'FcBuild', 'FcPoint', 'QNSnowFr',
        'QNSnow', 'AlbSnow', 'QM', 'QMFreeze', 'QMRain', 'SWE', 'MeltWater',
        'MeltWStore', 'SnowCh', 'SnowRPaved', 'SnowRBldgs', 'Ts', 'T2', 'Q2',
        'U10', 'RH2'],
        dtype='object', name='var')
```

1.3.2 a simple QF model: QF_simple

model description

For demonstration purposes we have created a very simple model instead of using the SUEWS Q_F (Järvi et al. 2011) with feedback from outdoor air temperature. The simple Q_F model considers only building heating and cooling:

$$Q_F = \begin{cases} (T_2 - T_C) \times C_B, & T_2 > T_C \\ (T_H - T_2) \times H_B, & T_2 < T_H \\ Q_{F0} & \end{cases}$$

where T_C (T_H) is the cooling (heating) threshold temperature of buildings, C_B (H_B) is the building cooling (heating) rate, and Q_{F0} is the baseline anthropogenic heat. The parameters used are: C_H set as 20 °C (10 °C), C_B set as 1.5 W m⁻² K⁻¹ (3 W m⁻² K⁻¹) and Q_{F0} is set as 0 W m⁻², implying other building activities (e.g. lightning, water heating, computers) are zero and therefore do not change the temperature or change with temperature.

implementation

```
[4]: def QF_simple(T2):
    qf_cooling = (T2-20)*5 if T2 > 20 else 0
    qf_heating = (10-T2)*10 if T2 < 10 else 0
    qf_res = np.max([qf_heating, qf_cooling])*0.3
    return qf_res
```

Visualise the QF_simple model:

```
[6]: ser_temp = pd.Series(np.arange(-5, 45, 0.5),
                          index=np.arange(-5, 45, 0.5)).rename('temp_C')
ser_qf_heating = ser_temp.loc[-5:10].map(QF_simple).rename(
    r'heating:$(T_H-T_a) \times H_B$')
ser_qf_cooling = ser_temp.loc[20:45].map(QF_simple).rename(
    r'cooling: $(T_a-T_C) \times C_B$')
ser_qf_zero = ser_temp.loc[10:20].map(QF_simple).rename('baseline: $Q_{F0}$')
df_temp_qf = pd.concat([ser_temp, ser_qf_cooling, ser_qf_heating, ser_qf_zero],
                       axis=1).set_index('temp_C')
ax_qf_func = df_temp_qf.plot()
ax_qf_func.set_xlabel('$T_2$ ($^\circ\text{C}$)')
ax_qf_func.set_ylabel('$Q_F$ ($\text{W m}^{-2}$)')
ax_qf_func.legend(title='simple $Q_F$')
ax_qf_func.annotate(
    "$T_C$",
    xy=(20, 0),
    xycoords='data',
    xytext=(25, 5),
    textcoords='data',
    arrowprops=dict(
        arrowstyle="->",
        color="0.5",
        shrinkA=5,
        shrinkB=5,
        patchA=None,
        patchB=None,
        connectionstyle='arc3',
    ),
)

ax_qf_func.annotate(
    "$T_H$",
```

(continues on next page)

(continued from previous page)

```

xy=(10, 0),
xycoords='data',
xytext=(5, 5),
textcoords='data',
arrowprops=dict(
    arrowstyle="->",
    color="0.5",
    shrinkA=5,
    shrinkB=5,
    patchA=None,
    patchB=None,
    connectionstyle='arc3',
),
),
ax_qf_func.annotate(
    "slope: $C_B$",
    xy=(30, QF_simple(30)),
    xycoords='data',
    xytext=(20, 20),
    textcoords='data',
    arrowprops=dict(
        arrowstyle="->",
        color="0.5",
        shrinkA=5,
        shrinkB=5,
        patchA=None,
        patchB=None,
        connectionstyle='arc3, rad=0.3',
    ),
),
ax_qf_func.annotate(
    "slope: $H_B$",
    xy=(5, QF_simple(5)),
    xycoords='data',
    xytext=(10, 20),
    textcoords='data',
    arrowprops=dict(
        arrowstyle="->",
        color="0.5",
        shrinkA=5,
        shrinkB=5,
        patchA=None,
        patchB=None,
        connectionstyle='arc3, rad=-0.3',
    ),
),
ax_qf_func.plot(10, 0, 'o', color='C1', fillstyle='none')
_ = ax_qf_func.plot(20, 0, 'o', color='C0', fillstyle='none')

```

1.3.3 communication between supy and QF_simple

construct a new coupled function

The coupling between the simple Q_F model and SuPy is done via the low-level function `suews_cal_tstep`, which is an interface function in charge of communications between SuPy frontend and the calculation kernel. By setting

SuPy to receive external Q_F as forcing, at each timestep, the simple Q_F model is driven by the SuPy output T_2 and provides SuPy with Q_F , which thus forms a two-way coupled loop.

```
[7]: # load extra low-level functions from supy to construct interactive functions
from supy.post import pack_df_output, pack_df_state
from supy.run import suews_cal_tstep, pack_grid_dict

def run_supy_qf(df_forcing_test, df_state_init_test):
    grid = df_state_init_test.index[0]
    df_state_init_test.loc[grid, 'emissionsmethod'] = 0

    df_forcing_test = df_forcing_test\
        .assign(
            metforcingdata_grid=0,
            ts5mindata_ir=0,
        )\
        .rename(
            # remanae is a workaround to resolve naming inconsistency between
            # suews fortran code interface and input forcing file headers
            columns={
                '%' + 'iy': 'iy',
                'id': 'id',
                'it': 'it',
                'imin': 'imin',
                'qn': 'qn1_obs',
                'qh': 'qh_obs',
                'qe': 'qe',
                'qs': 'qs_obs',
                'qf': 'qf_obs',
                'U': 'avul',
                'RH': 'avrh',
                'Tair': 'temp_c',
                'pres': 'press_hpa',
                'rain': 'precip',
                'kdown': 'avkdn',
                'snow': 'snowfrac_obs',
                'ldown': 'ldown_obs',
                'fcld': 'fcld_obs',
                'Wuh': 'wu_m3',
                'xsmd': 'xsmd',
                'lai': 'lai_obs',
                'kdiff': 'kdiff',
                'kdir': 'kdir',
                'wdir': 'wdir',
            }
        )

    t2_ext = df_forcing_test.iloc[0].temp_c
    qf_ext = QF_simple(t2_ext)

    # initialise dicts for holding results
    dict_state = {}
    dict_output = {}

    # starting timestep
    t_start = df_forcing_test.index[0]
    # convert df to dict with `itertuples` for better performance
```

(continues on next page)

(continued from previous page)

```

dict_forcing = {
    row.Index: row._asdict()
    for row in df_forcing_test.itertuples()
}
# dict_state is used to save model states for later use
dict_state = {(t_start, grid): pack_grid_dict(series_state_init)
              for grid, series_state_init in df_state_init_test.iterrows()}

# just use a single grid run for the test coupling
for tstep in df_forcing_test.index:
    # load met forcing at `tstep`
    met_forcing_tstep = dict_forcing[tstep]
    # inject `qf_ext` to `met_forcing_tstep`
    met_forcing_tstep['qf_obs'] = qf_ext

    # update model state
    dict_state_start = dict_state[(tstep, grid)]

    dict_state_end, dict_output_tstep = suews_cal_tstep(
        dict_state_start, met_forcing_tstep)
    # the fourth to the last is `T2` stored in the result array
    t2_ext = dict_output_tstep['dataoutlinesuews'][-4]
    qf_ext = QF_simple(t2_ext)

    dict_output.update({(tstep, grid): dict_output_tstep})
    dict_state.update({(tstep + tstep.freq, grid): dict_state_end})

# pack results as easier DataFrames
df_output_test = pack_df_output(dict_output).swaplevel(0, 1)
df_state_test = pack_df_state(dict_state).swaplevel(0, 1)
return df_output_test.loc[grid, 'SUEWS'], df_state_test

```

simulations for summer and winter months

The simulation using SuPy coupled is performed for London 2012. The data analysed are a summer (July) and a winter (December) month. Initially Q_F is 0 W m^{-2} the T_2 is determined and used to determine $Q_{F[1]}$ which in turn modifies $T_{2[1]}$ and therefore modifies $Q_{F[2]}$ and the diagnosed $T_{2[2]}$.

spin-up run (January to June) for summer simulation

```
[8]: df_output_june, df_state_jul = sp.run_supy(
    df_forcing.loc[:, '2012 6'], df_state_init)
df_state_jul_init = df_state_jul.reset_index('datetime', drop=True).iloc[[-1]]
```

```

INFO:root:=====
INFO:root:Simulation period:
INFO:root:  Start: 2012-01-01 00:05:00
INFO:root:  End: 2012-06-30 23:55:00
INFO:root:
INFO:root:No. of grids: 1
INFO:root:SuPy is running in serial mode
INFO:root:Execution time: 1.6 s
INFO:root:=====

```

spin-up run (July to October) for winter simulation

```
[9]: df_output_oct, df_state_dec = sp.run_supy(
      df_forcing.loc['2012 7':'2012 11'], df_state_jul_init)
df_state_dec_init = df_state_dec.reset_index('datetime', drop=True).iloc[[-1]]
```

```
INFO:root:=====
INFO:root:Simulation period:
INFO:root:  Start: 2012-07-01 00:00:00
INFO:root:  End: 2012-11-30 23:55:00
INFO:root:
INFO:root:No. of grids: 1
INFO:root:SuPy is running in serial mode
INFO:root:Execution time: 1.3 s
INFO:root:=====
```

coupled simulation

```
[10]: df_output_test_summer, df_state_summer_test = run_supy_qf(
      df_forcing.loc['2012 7'], df_state_jul_init.copy())
df_output_test_winter, df_state_winter_test = run_supy_qf(
      df_forcing.loc['2012 12'], df_state_dec_init.copy())
```

examine the results

sumer

```
[11]: var = 'QF'
var_label = '$Q_F$ ($ \mathrm{W} \ m^{-2})$)'
var_label_right = '$\Delta Q_F$ ($ \mathrm{W} \ m^{-2})$)'
period = '2012 7'
df_test = df_output_test_summer
y1 = df_test.loc[period, var].rename('qf_simple')
y2 = df_output_def.loc[period, var].rename('suews')
y3 = (y1-y2).rename('diff')
df_plot = pd.concat([y1, y2, y3], axis=1)
ax = df_plot.plot(secondary_y='diff')
ax.set_ylabel(var_label)
# sns.lmplot(data=df_plot, x='qf_simple', y='diff')
ax.right_ax.set_ylabel(var_label_right)
lines = ax.get_lines() + ax.right_ax.get_lines()
ax.legend(lines, [l.get_label() for l in lines], loc='best')
```

```
[11]: <matplotlib.legend.Legend at 0x7fa3853c1b00>
```

```
[12]: var = 'T2'
var_label = '$T_2$ ($^{\circ}\mathrm{C})$)'
var_label_right = '$\Delta T_2$ ($^{\circ}\mathrm{C})$)'
period = '2012 7'
df_test = df_output_test_summer
```

(continues on next page)

(continued from previous page)

```

y1 = df_test.loc[period, var].rename('qf_simple')
y2 = df_output_def.loc[period, var].rename('suews')
y3 = (y1-y2).rename('diff')
df_plot = pd.concat([y1, y2, y3], axis=1)
ax = df_plot.plot(secondary_y='diff')
ax.set_ylabel(var_label)
ax.right_ax.set_ylabel(var_label_right)
lines = ax.get_lines() + ax.right_ax.get_lines()
ax.legend(lines, [l.get_label() for l in lines], loc='best')

```

```
[12]: <matplotlib.legend.Legend at 0x7fa3852ccb70>
```

winter

```

[13]: var = 'QF'
var_label = '$Q_F$ ($ \mathrm{W} \ m^{-2})$'
var_label_right = '$\Delta Q_F$ ($ \mathrm{W} \ m^{-2})$'
period = '2012 12'
df_test = df_output_test_winter
y1 = df_test.loc[period, var].rename('qf_simple')
y2 = df_output_def.loc[period, var].rename('suews')
y3 = (y1-y2).rename('diff')
df_plot = pd.concat([y1, y2, y3], axis=1)
ax = df_plot.plot(secondary_y='diff')
ax.set_ylabel(var_label)
# sns.lmplot(data=df_plot,x='qf_simple',y='diff')
ax.right_ax.set_ylabel(var_label_right)
lines = ax.get_lines() + ax.right_ax.get_lines()
ax.legend(lines, [l.get_label() for l in lines], loc='best')

```

```
[13]: <matplotlib.legend.Legend at 0x7fa3852cc0b8>
```

```

[14]: var = 'T2'
var_label = '$T_2$ ($^{\circ}C)$'
var_label_right = '$\Delta T_2$ ($^{\circ}C)$'
period = '2012 12'
df_test = df_output_test_winter
y1 = df_test.loc[period, var].rename('qf_simple')
y2 = df_output_def.loc[period, var].rename('suews')
y3 = (y1-y2).rename('diff')
df_plot = pd.concat([y1, y2, y3], axis=1)
ax = df_plot.plot(secondary_y='diff')
ax.set_ylabel(var_label)
ax.right_ax.set_ylabel(var_label_right)
lines = ax.get_lines() + ax.right_ax.get_lines()
ax.legend(lines, [l.get_label() for l in lines], loc='center right')

```

```
[14]: <matplotlib.legend.Legend at 0x7fa2d0e6f940>
```

comparison in ΔQ_F - ΔT_2 feedback between summer and winter

```
[15]: # filter results using `where` to choose periods when `QF_simple` is effective
# (i.e. activated by outdoor air temperatures)
df_diff_summer = (df_output_test_summer - df_output_def)\
    .where(df_output_def.T2 > 20, np.nan)\
    .dropna(how='all', axis=0)
df_diff_winter = (df_output_test_winter - df_output_def)\
    .where(df_output_test_winter.T2 < 10, np.nan)\
    .dropna(how='all', axis=0)

set_matplotlib_formats('svg')
# set_matplotlib_formats('retina')
df_diff_season = pd.concat([
    df_diff_winter.assign(season='winter'),
    df_diff_summer.assign(season='summer'),
]).loc[:, ['season', 'QF', 'T2']]
g = sns.lmplot(
    data=df_diff_season,
    x='QF',
    y='T2',
    hue='season',
    height=4,
    truncate=False,
    markers='o',
    legend_out=False,
    scatter_kws={
        's': 1,
        'zorder': 0,
        'alpha': 0.8,
    },
    line_kws={
        'zorder': 6,
        'linestyle': '--'
    },
)
g.set_axis_labels(
    '$\Delta Q_F$ ($\mathrm{W \ m^{-2}}$)',
    '$\Delta T_2$ ($^{\circ}\mathrm{C}$)',
)
g.ax.legend(markerscale=4)
g.despine(top=False, right=False)
```

```
[15]: <seaborn.axisgrid.FacetGrid at 0x7fa35b350748>
```

The above figure indicate a positive feedback, as Q_F is increased there is an elevated T_2 but with different magnitudes given the non-linearity in the SUEWS modelling system. Of particular note is the positive feedback loop under warm air temperatures: the anthropogenic heat emissions increase which in turn elevates the outdoor air temperature causing yet more anthropogenic heat release. Note that London is relatively cool so the enhancement is much less than it would be in warmer cities.

End of doc/tutorial/external-interaction.ipynb

The following section was generated from docs/source/tutorial/AMF-sim.ipynb

1.4 Modelling Surface Energy Balance at an AmeriFlux Site Using SuPy

This tutorial aims to demonstrate how to use an advanced land surface model (SuPy, SUEWS in Python) to better understand the surface energy balance (SEB) features by conducting simulation at an AmeriFlux site. This would be particularly useful after building your own model: as you will learn how sophisticated models could be developed from those simpler ones.

SuPy is a Python-enhanced urban climate model with SUEWS, **Surface Urban Energy and Water Balance Scheme**, as its computation core. More SuPy tutorials are available [here](#).

In this tutorial the workflow to model the surface energy balance (SEB) at a chosen AmeriFlux (AMF) site using SuPy/SUEWS is undertaken. The steps, consist of

1. *Preparing the input data;*
2. *Running a simulation;*
3. *Examination of results;* and
4. *Further exploration*

Before starting, you need to install SuPy and load the following necessary packages.

```
pip install supy==2019.11.18.dev0
```

```
[1]: # !pip install supy==2019.11.18.dev0 &> install.log
```

```
[2]: import matplotlib.pyplot as plt
import supy as sp
import pandas as pd
import numpy as np
from pathlib import Path
%matplotlib inline
```

```
[3]: %load_ext autoreload
%autoreload 2
```

```
[4]: sp.show_version()

supy: 2019.11.18dev
supy_driver: 2019a18
```

1.4.1 Prepare input data

Overview of SuPy input

Load sample data:

To ease the preparation of model input, a helper function `load_SampleData` is provided to get the sample input for SuPy simulations, which will later be used as template to populate your specific model configurations and forcing input.

```
[5]: df_state_init, df_forcing = sp.load_SampleData()
2019-11-22 09:16:28,573 -- SuPy -- INFO -- All cache cleared.
```

df_state_init

df_state_init includes model Initial state consisting of:

- surface characteristics (e.g., albedo, emissivity, land cover fractions, etc.; full details refer to [SUEWS documentation](#)).
- model configurations (e.g., stability; full details refer to [SUEWS documentation](#)).

Detailed description of variables in df_state_init refers to [SuPy input](#).

- Surface land cover fraction information in the sample input dataset:

```
[6]: df_state_init.filter(like='sfr')
[6]: var      sfr
ind_dim  (0,)  (1,)  (2,)  (3,)  (4,)  (5,)  (6,)
grid
98          0.43  0.38  0.001  0.019  0.029  0.001  0.14
```

- Heights of bluff-bodies (m):

```
[7]: df_state_init.loc[:, ['bldgh', 'evetreeh', 'dectreeh']]
[7]: var      bldgh dectreeh evetreeh
ind_dim      0          0          0
grid
98          22.0          13.1          13.1
```

df_forcing

df_forcing includes meteorological and other external forcing information.

Detailed description of variables in df_forcing refers to [SuPy input](#).

Below is a view of heading lines of the forcing variables.

```
[8]: df_forcing.head()
[8]:
      iy  id  it  imin  qn  qh  qe  qs  qf  \
2012-01-01 00:05:00  2012  1  0    5 -999.0 -999.0 -999.0 -999.0 -999.0
2012-01-01 00:10:00  2012  1  0   10 -999.0 -999.0 -999.0 -999.0 -999.0
2012-01-01 00:15:00  2012  1  0   15 -999.0 -999.0 -999.0 -999.0 -999.0
2012-01-01 00:20:00  2012  1  0   20 -999.0 -999.0 -999.0 -999.0 -999.0
2012-01-01 00:25:00  2012  1  0   25 -999.0 -999.0 -999.0 -999.0 -999.0

      U  ...  snow  ldown  fclld  Wuh  xsmd  lai  \
2012-01-01 00:05:00  4.5225  ... -999.0 -999.0 -999.0 -999.0 -999.0 -999.0
2012-01-01 00:10:00  4.5225  ... -999.0 -999.0 -999.0 -999.0 -999.0 -999.0
2012-01-01 00:15:00  4.5225  ... -999.0 -999.0 -999.0 -999.0 -999.0 -999.0
2012-01-01 00:20:00  4.5225  ... -999.0 -999.0 -999.0 -999.0 -999.0 -999.0
2012-01-01 00:25:00  4.5225  ... -999.0 -999.0 -999.0 -999.0 -999.0 -999.0
```

(continues on next page)

(continued from previous page)

```

          kdiff  kdir  wdir  isec
2012-01-01 00:05:00 -999.0 -999.0 -999.0  0.0
2012-01-01 00:10:00 -999.0 -999.0 -999.0  0.0
2012-01-01 00:15:00 -999.0 -999.0 -999.0  0.0
2012-01-01 00:20:00 -999.0 -999.0 -999.0  0.0
2012-01-01 00:25:00 -999.0 -999.0 -999.0  0.0

```

```
[5 rows x 25 columns]
```

Site-specific configuration of surface parameters

Given `pandas.DataFrame` as the core data structure of SuPy, all operations, including modification, output, demonstration, etc., on SuPy inputs (`df_state_init` and `df_forcing`) can be done using pandas-based functions/methods. Please see [SuPy quickstart](#) for methods to do so.

Below we will modify several key properties of the chosen site with appropriate values to run SuPy. First, we copy the `df_state_init` to have a new `DataFrame` for manipulation.

```
[9]: df_state_amf = df_state_init.copy()
```

```
[10]: # site identifier
name_site = 'US-AR1'
```

location

```
[11]: # latitude
df_state_amf.loc[:, 'lat'] = 41.37
# longitude
df_state_amf.loc[:, 'lng'] = -106.24
# altitude
df_state_amf.loc[:, 'alt'] = 611.
```

land cover fraction

Land covers in SUEWS

```
[12]: # view the surface fraction variable: `sfr`
df_state_amf.loc[:, 'sfr'] = 0
df_state_amf.loc[:, ('sfr', '(4,)')] = 1
df_state_amf.loc[:, 'sfr']
```

```
[12]: ind_dim  (0,)  (1,)  (2,)  (3,)  (4,)  (5,)  (6,)
grid
98          0.0  0.0  0.0  0.0  1.0  0.0  0.0
```

albedo

```
[13]: # we only set values for grass as the modelled site has a single land cover type:↵
↵grass.
df_state_amf.albmax_grass = 0.19
df_state_amf.albmin_grass = 0.14
```

```
[14]: # initial albedo value
df_state_amf.loc[:, 'albgrass_id'] = 0.14
```

LAI/phenology

```
[15]: df_state_amf.filter(like='lai')
```

```
[15]: var      laimax      laimin      laipower      ... \
ind_dim  (0,) (1,) (2,)   (0,) (1,) (2,)   (0, 0) (0, 1) (0, 2) (1, 0) ...
grid
98        5.1  5.5  5.9    4.0  1.0  1.6    0.03  0.03  0.03  0.0005 ...

var      laitype      laicalcyes lai_id
ind_dim  (3, 0) (3, 1) (3, 2)   (0,) (1,) (2,)   0   (0,) (1,) (2,)
grid
98        0.0005  0.0005  0.0005    0.0  0.0  0.0    1   4.0  1.0  1.6

[1 rows x 25 columns]
```

```
[16]: # properties to control vegetation phenology
# you can skip the details for and just set them as provided below

# LAI paramters
df_state_amf.loc[:, ('laimax', '(2,)')] = 1
df_state_amf.loc[:, ('laimin', '(2,)')] = 0.2
# initial LAI
df_state_amf.loc[:, ('lai_id', '(2,)')] = 0.2

# BaseT
df_state_amf.loc[:, ('baset', '(2,)')] = 5
# BaseTe
df_state_amf.loc[:, ('basete', '(2,)')] = 20

# SDDFull
df_state_amf.loc[:, ('sddfull', '(2,)')] = -1000
# GDDFull
df_state_amf.loc[:, ('gddfull', '(2,)')] = 1000
```

surface resistance

```
[17]: # parameters to model surface resistance
df_state_amf.maxconductance = 18.7
df_state_amf.g1 = 1
df_state_amf.g2 = 104.215
df_state_amf.g3 = 0.424
df_state_amf.g4 = 0.814
df_state_amf.g5 = 36.945
df_state_amf.g6 = 0.025
```

measurement height

```
[18]: # height where forcing variables are measured/collected
df_state_amf.z = 2.84
```

urban feature

```
[19]: # disable anthropogenic heat by setting zero population
df_state_amf.popdensdaytime = 0
df_state_amf.popdensnighttime = 0
```

check df_state

```
[20]: # this procedure is to double-check proper values are set in `df_state_amf`
sp.check_state(df_state_amf)

2019-11-22 09:16:30,954 -- SuPy -- INFO -- SuPy is validating `df_state`...
2019-11-22 09:16:31,120 -- SuPy -- INFO -- All checks for `df_state` passed!
```

prepare forcing conditions

Here we use the a SuPy utility function `read_forcing` to read in forcing data from an external file in the format of SUEWS input. Also note, this `read_forcing` utility will also resample the forcing data to a proper temporal resolution to run SuPy/SUEWS, which is usually 5 min (300 s).

load and resample forcing data

```
[21]: # load forcing data from an external file and resample to a resolution of 300 s.
# Note this dataset has been gap-filled.
df_forcing_amf = sp.util.read_forcing('./data/US-AR1_2010_data_60.txt',
                                     timestep_mod=300)

# this procedure is to double-check proper forcing values are set in `df_forcing_amf`
_ = sp.check_forcing(df_forcing_amf)

2019-11-22 09:16:32,059 -- SuPy -- INFO -- SuPy is validating `df_forcing`...
2019-11-22 09:16:34,374 -- SuPy -- ERROR -- Issues found in `df_forcing`:
`kdown` should be between [0, 1400] but -1.3057500000000002 is found at 2010-01-01_
↪00:05:00
```

The checker detected invalid values in variable `kdown`: negative incoming solar radiation is found. We then need to fix this as follows:

```
[22]: # modify invalid values
df_forcing_amf.kdown = df_forcing_amf.kdown.where(df_forcing_amf.kdown > 0, 0)
```

```
[23]: # check `df_forcing` again
_ = sp.check_forcing(df_forcing_amf)
```

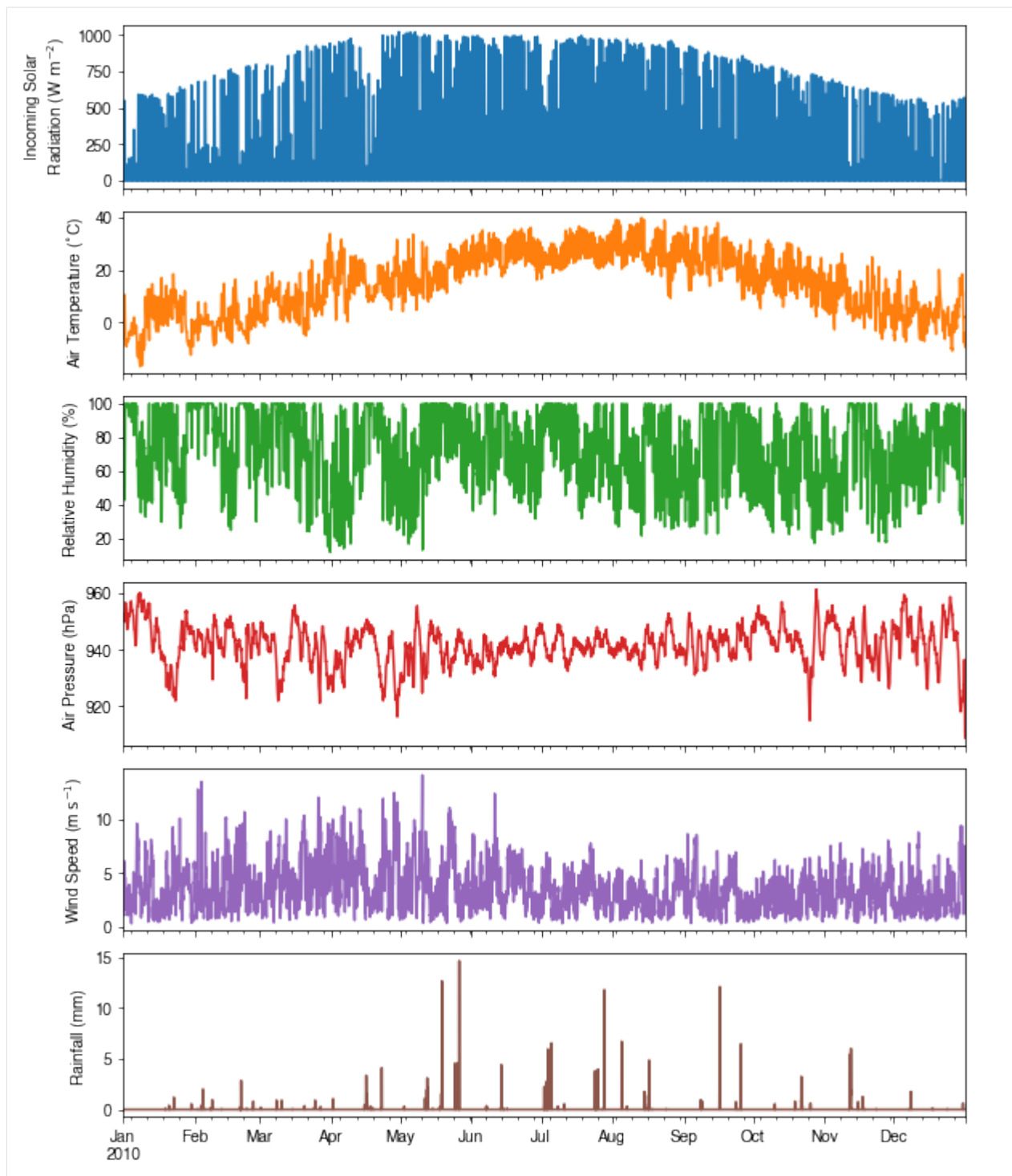
```
2019-11-22 09:16:34,466 -- SuPy -- INFO -- SuPy is validating `df_forcing`...
2019-11-22 09:16:36,946 -- SuPy -- INFO -- All checks for `df_forcing` passed!
```

examine forcing data

We can examine the forcing data:

```
[24]: list_var_forcing = [
        'kdown',
        'Tair',
        'RH',
        'pres',
        'U',
        'rain',
    ]
dict_var_label = {
    'kdown': 'Incoming Solar\n Radiation ( $\mathrm{W \ m^{-2}}$ )',
    'Tair': 'Air Temperature ( $^{\circ}\mathrm{C}$ )',
    'RH': r'Relative Humidity (%)',
    'pres': 'Air Pressure (hPa)',
    'rain': 'Rainfall (mm)',
    'U': 'Wind Speed ( $\mathrm{m \ s^{-1}}$ )'
}
df_plot_forcing_x = df_forcing_amf.loc[:, list_var_forcing].copy().shift(
    -1).dropna(how='any')
df_plot_forcing = df_plot_forcing_x.resample('1h').mean()
df_plot_forcing['rain'] = df_plot_forcing_x['rain'].resample('1h').sum()

axes = df_plot_forcing.plot(
    subplots=True,
    figsize=(8, 12),
    legend=False,
)
fig = axes[0].figure
fig.tight_layout()
fig.autofmt_xdate(bottom=0.2, rotation=0, ha='center')
for ax, var in zip(axes, list_var_forcing):
    _ = ax.set_ylabel(dict_var_label[var])
```

1.4.2 Run simulations

Once met-forcing (via `df_forcing_amf`) and initial conditions (via `df_state_amf`) are loaded in, we call `sp.run_supy` to conduct a SUEWS simulation, which will return two pandas DataFrames: `df_output` and `df_state_final`.

```
[25]: df_output, df_state_final = sp.run_supy(df_forcing_amf, df_state_amf)

2019-11-22 09:16:40,809 -- SuPy -- INFO -- =====
2019-11-22 09:16:40,810 -- SuPy -- INFO -- Simulation period:
2019-11-22 09:16:40,811 -- SuPy -- INFO --      Start: 2010-01-01 00:05:00
2019-11-22 09:16:40,811 -- SuPy -- INFO --      End: 2011-01-01 00:00:00
2019-11-22 09:16:40,812 -- SuPy -- INFO --
2019-11-22 09:16:40,813 -- SuPy -- INFO -- No. of grids: 1
2019-11-22 09:16:40,813 -- SuPy -- INFO -- SuPy is running in serial mode
2019-11-22 09:16:54,304 -- SuPy -- INFO -- Execution time: 13.5 s
2019-11-22 09:16:54,305 -- SuPy -- INFO -- =====
```

df_output

df_output is an ensemble output collection of major SUEWS output groups, including:

- SUEWS: the essential SUEWS output variables
- DailyState: variables of daily state information
- snow: snow output variables (effective when snowuse = 1 set in df_state_init)
- RSL: profile of air temperature, humidity and wind speed within roughness sub-layer.

Detailed description of variables in df_output refers to [SuPy output](#)

```
[26]: df_output.columns.levels[0]
```

```
[26]: Index(['SUEWS', 'snow', 'RSL', 'DailyState'], dtype='object', name='group')
```

df_state_final

df_state_final is a DataFrame for holding:

1. all model states if save_state is set to True when calling sp.run_supy and supy may run significantly slower for a large simulation;
2. or, only the final state if save_state is set to False (the default setting) in which mode supy has a similar performance as the standalone compiled SUEWS executable.

Entries in df_state_final have the same data structure as df_state_init and can thus be used for other SUEWS simulations starting at the timestamp as in df_state_final.

Detailed description of variables in df_state_final refers to [SuPy output](#)

```
[27]: df_state_final.T.head()
```

```
[27]: datetime          2010-01-01 00:05:00 2011-01-01 00:05:00
grid                                98                                98
var          ind_dim
ah_min      (0,)          15.0          15.0
             (1,)          15.0          15.0
ah_slope_cooling (0,)          2.7          2.7
             (1,)          2.7          2.7
ah_slope_heating (0,)          2.7          2.7
```

1.4.3 Examine results

Thanks to the functionality inherited from `pandas` and other packages under the `PyData` stack, compared with the standard SUEWS simulation workflow, `supy` enables more convenient examination of SUEWS results by statistics calculation, resampling, plotting (and many more).

Output structure

`df_output` is organised with `MultiIndex (grid,timestamp)` and `(group,variable)` as index and columns, respectively.

```
[28]: df_output.head()
```

```
[28]: group          SUEWS          \
var          Kdown  Kup          Ldown          Lup          Tsurf
grid datetime
98  2010-01-01 00:05:00    0.0    0.0  265.638676  305.413842 -1.587667
    2010-01-01 00:10:00    0.0    0.0  265.638676  305.413842 -1.587667
    2010-01-01 00:15:00    0.0    0.0  265.638676  305.413842 -1.587667
    2010-01-01 00:20:00    0.0    0.0  265.638676  305.413842 -1.587667
    2010-01-01 00:25:00    0.0    0.0  265.638676  305.413842 -1.587667

group          QN  QF          QS          QH          QE  ...  \
var                                     QN  QF          QS          QH          QE  ...
grid datetime
98  2010-01-01 00:05:00 -39.775166    0.0 -50.989269  11.054221  0.159883  ...
    2010-01-01 00:10:00 -39.775166    0.0 -50.729902  10.795477  0.159259  ...
    2010-01-01 00:15:00 -39.775166    0.0 -50.481342  10.547515  0.158661  ...
    2010-01-01 00:20:00 -39.775166    0.0 -50.243138  10.309886  0.158086  ...
    2010-01-01 00:25:00 -39.775166    0.0 -50.014860  10.082159  0.157534  ...

group          DailyState          \
var          DensSnow_Paved  DensSnow_Bldgs  DensSnow_EveTr
grid datetime
98  2010-01-01 00:05:00          NaN          NaN          NaN
    2010-01-01 00:10:00          NaN          NaN          NaN
    2010-01-01 00:15:00          NaN          NaN          NaN
    2010-01-01 00:20:00          NaN          NaN          NaN
    2010-01-01 00:25:00          NaN          NaN          NaN

group          DensSnow_DecTr  DensSnow_Grass  DensSnow_BSoil  \
var
grid datetime
98  2010-01-01 00:05:00          NaN          NaN          NaN
    2010-01-01 00:10:00          NaN          NaN          NaN
    2010-01-01 00:15:00          NaN          NaN          NaN
    2010-01-01 00:20:00          NaN          NaN          NaN
    2010-01-01 00:25:00          NaN          NaN          NaN

group          DensSnow_Water  a1  a2  a3
var
grid datetime
98  2010-01-01 00:05:00          NaN  NaN  NaN  NaN
    2010-01-01 00:10:00          NaN  NaN  NaN  NaN
    2010-01-01 00:15:00          NaN  NaN  NaN  NaN
    2010-01-01 00:20:00          NaN  NaN  NaN  NaN
    2010-01-01 00:25:00          NaN  NaN  NaN  NaN
```

(continues on next page)

(continued from previous page)

[5 rows x 345 columns]

Here we demonstrate several typical scenarios for SUEWS results examination.

The essential SUEWS output collection is extracted as a separate variable for easier processing in the following sections. More [advanced slicing techniques](#) are available in [pandas documentation](#).

```
[29]: grid = df_state_amf.index[0]
      df_output_suews = df_output.loc[grid, 'SUEWS']
```

Statistics Calculation

We can use `.describe()` method for a quick overview of the key surface energy balance budgets.

```
[30]: df_output_suews.loc[:, ['QN', 'QS', 'QH', 'QE', 'QF']].describe()
```

```
[30]: var      QN      QS      QH      QE      QF
count  105120.000000  105120.000000  105120.000000  105120.000000  105120.0
mean    118.291222     9.117453    53.611159    56.302547     0.0
std     213.150698    82.433076    72.044586    93.959856     0.0
min     -98.895138   -103.332153   -136.770097   -10.592811     0.0
25%     -32.252113   -46.922708     9.790903     0.633106     0.0
50%      -1.216564   -33.883630    25.756834     4.240341     0.0
75%     247.458581    55.231281    75.382465    67.207728     0.0
max      746.187700    262.700179    403.492712    445.496829     0.0
```

Plotting

Basic example

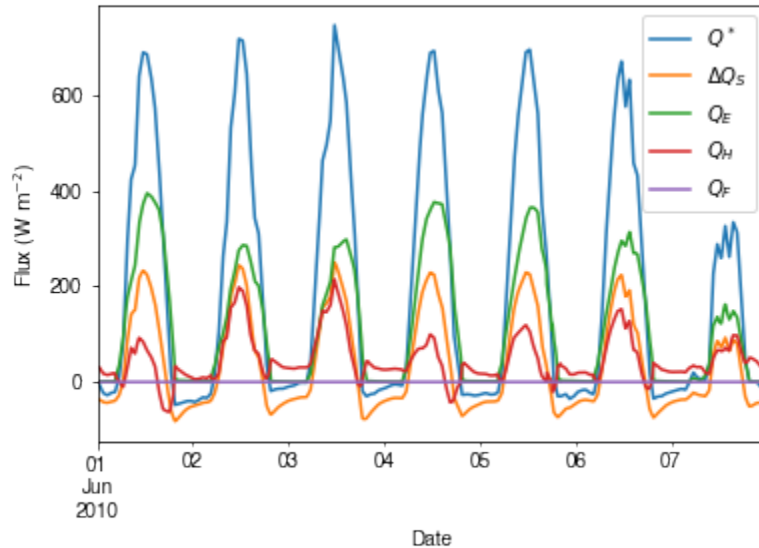
Plotting is very straightforward via the `.plot` method bounded with `pandas.DataFrame`. Note the usage of `loc` for to slices of the output `DataFrame`.

```
[31]: # a dict for better display variable names
dict_var_disp = {
    'QN': '$Q^*$',
    'QS': r'$\Delta Q_S$',
    'QE': '$Q_E$',
    'QH': '$Q_H$',
    'QF': '$Q_F$',
    'Kdown': r'$K_{\downarrow}$',
    'Kup': r'$K_{\uparrow}$',
    'Ldown': r'$L_{\downarrow}$',
    'Lup': r'$L_{\uparrow}$',
    'Rain': '$P$',
    'Irr': '$I$',
    'Evap': '$E$',
    'RO': '$R$',
    'TotCh': '$\Delta S$',
}
```

Peek at the simulation results:

```
[32]: grid = df_state_init.index[0]
```

```
[33]: ax_output = df_output_suews\
      .loc['2010-06-01':'2010-06-07',
           ['QN', 'QS', 'QE', 'QH', 'QF']]\
      .rename(columns=dict_var_disp)\
      .plot()
_ = ax_output.set_xlabel('Date')
_ = ax_output.set_ylabel('Flux ($ \mathrm{W \ m^{-2}}$)')
_ = ax_output.legend()
```



Plotting after resampling

The suggested runtime/simulation frequency of SUEWS is 300 s, which usually results a large output and may be over-weighted for storage and analysis. Also, you may feel apparent slowdown in producing the above figure as a large amount of data were used for the plotting. To slim down the result size for analysis and output, we can resample the default output very easily.

```
[34]: rsmp_1d = df_output_suews.resample('1d')
# daily mean values
df_1d_mean = rsmp_1d.mean()
# daily sum values
df_1d_sum = rsmp_1d.sum()
```

We can then re-examine the above energy balance at hourly scale and plotting will be significantly faster.

```
[35]: # energy balance
ax_output = df_1d_mean\
      .loc[:, ['QN', 'QS', 'QE', 'QH', 'QF']]\
      .rename(columns=dict_var_disp)\
      .plot(
          figsize=(10, 3),
          title='Surface Energy Balance',
      )
_ = ax_output.set_xlabel('Date')
```

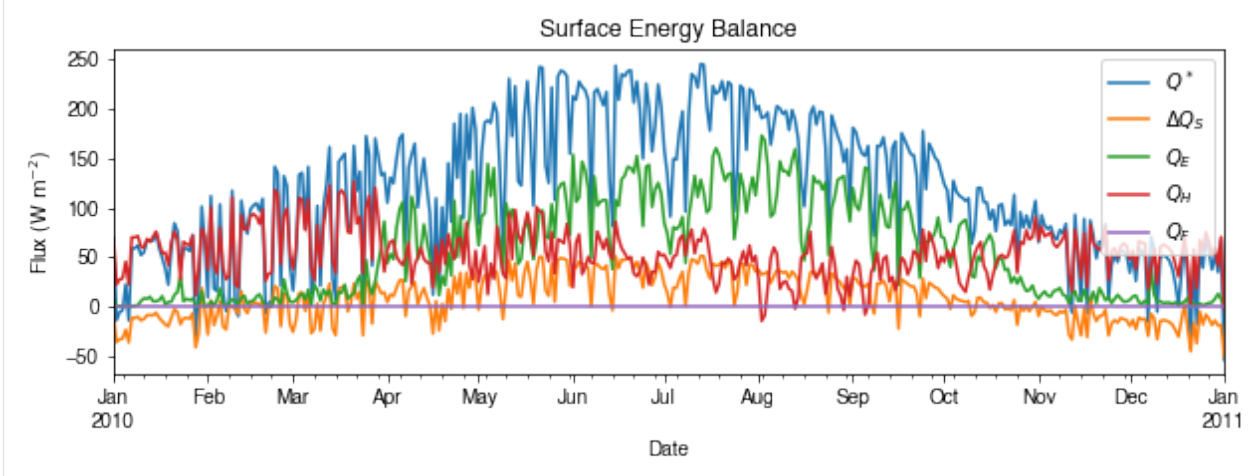
(continues on next page)

(continued from previous page)

```

_ = ax_output.set_ylabel('Flux ($ \mathrm{W \ m^{-2}}$)')
_ = ax_output.legend()

```

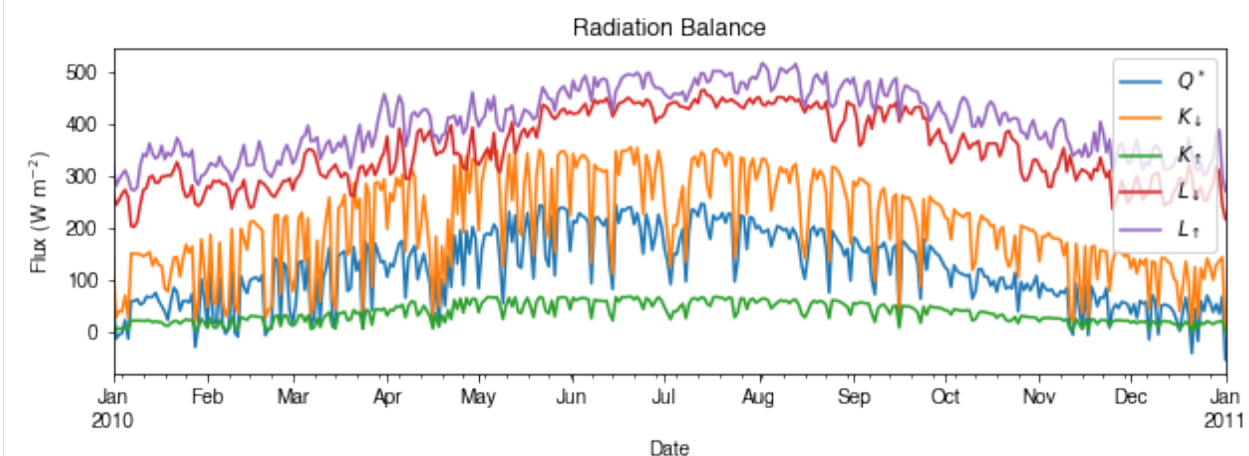


Then we use the hourly results for other analyses.

```

[36]: # radiation balance
ax_output = df_ld_mean\
    .loc[:, ['QN', 'Kdown', 'Kup', 'Ldown', 'Lup']]\
    .rename(columns=dict_var_disp)\
    .plot(
        figsize=(10, 3),
        title='Radiation Balance',
    )
_ = ax_output.set_xlabel('Date')
_ = ax_output.set_ylabel('Flux ($ \mathrm{W \ m^{-2}}$)')
_ = ax_output.legend()

```



```

[37]: # water balance
ax_output = df_ld_sum\
    .loc[:, ['Rain', 'Irr', 'Evap', 'RO', 'TotCh']]\
    .rename(columns=dict_var_disp)\
    .plot(
        figsize=(10, 3),

```

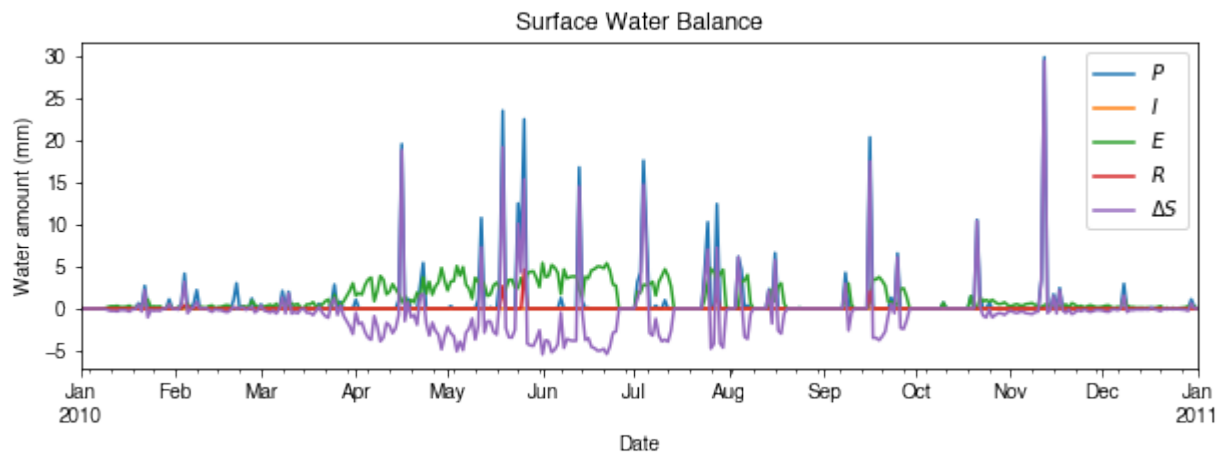
(continues on next page)

(continued from previous page)

```

        title='Surface Water Balance',
    )
    _ = ax_output.set_xlabel('Date')
    _ = ax_output.set_ylabel('Water amount (mm)')
    _ = ax_output.legend()

```



Get an overview of partitioning in energy and water balance at monthly scales:

```

[38]: # get a monthly Resampler
df_plot = df_output_suews.copy()
df_plot.index = df_plot.index.set_names('Month')
rsmplM = df_plot\
    .shift(-1)\
    .dropna(how='all')\
    .resample('1M', kind='period')
# mean values
df_1M_mean = rsmplM.mean()
# sum values
df_1M_sum = rsmplM.sum()

[39]: # month names
name_mon = [x.strftime('%b') for x in rsmplM.groups]
# create subplots showing two panels together
fig, axes = plt.subplots(2, 1, sharex=True)
# surface energy balance
_ = df_1M_mean\
    .loc[:, ['QN', 'QS', 'QE', 'QH', 'QF']]\
    .rename(columns=dict_var_disp)\
    .plot(
        ax=axes[0], # specify the axis for plotting
        figsize=(10, 6), # specify figure size
        title='Surface Energy Balance',
        kind='bar',
    )
# surface water balance
_ = df_1M_sum\
    .loc[:, ['Rain', 'Irr', 'Evap', 'RO', 'TotCh']]\
    .rename(columns=dict_var_disp)\
    .plot(
        ax=axes[1], # specify the axis for plotting

```

(continues on next page)

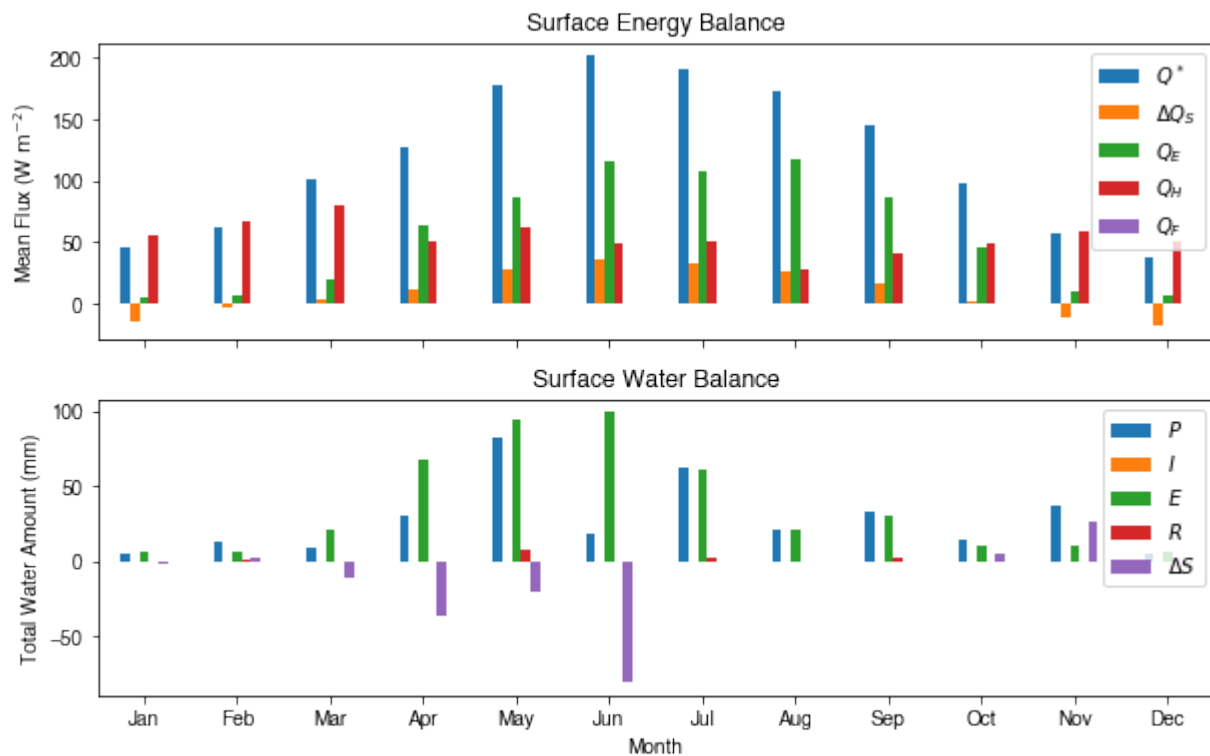
(continued from previous page)

```

    title='Surface Water Balance',
    kind='bar'
)

# annotations
_ = axes[0].set_ylabel('Mean Flux ($ \mathrm{W \ m^{-2}}$)')
_ = axes[0].legend()
_ = axes[1].set_xlabel('Month')
_ = axes[1].set_ylabel('Total Water Amount (mm)')
_ = axes[1].xaxis.set_ticklabels(name_mon, rotation=0)
_ = axes[1].legend()

```



Save results to external files

The supy output can be saved as txt files for further analysis using supy function `save_supy`.

```
[40]: list_path_save = sp.save_supy(df_output, df_state_final)
```

```
[41]: for file_out in list_path_save:
      print(file_out.name)
```

```

98_2010_SUEWS_5.txt
98_2010_snow_5.txt
98_2010_RSL_5.txt
98_2010_DailyState.txt
98_2010_SUEWS_60.txt
98_2010_snow_60.txt
98_2010_RSL_60.txt
df_state.csv

```


1.4.4 More explorations into simulation results

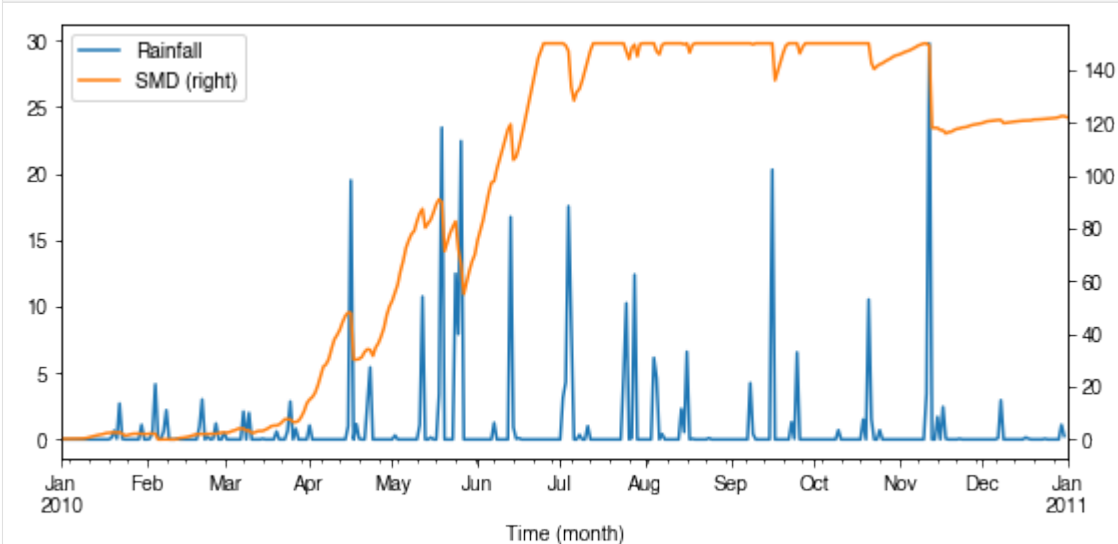
In this section, we will use the simulation results to explore more features revealed by SuPy/SUEWS simulations but *unavailable in your simple model*.

Dynamics in rainfall and soil moisture deficit (SMD)

```
[42]: df_dailystate = df_output.loc[grid, 'DailyState'].dropna(
      how='all').resample('1d').mean()
```

```
[43]: # daily rainfall
ser_p = df_dailystate.P_day.rename('Rainfall')
ser_smd = df_output_suews.SMD
ser_smd_dmax = ser_smd.resample('1d').max().rename('SMD')

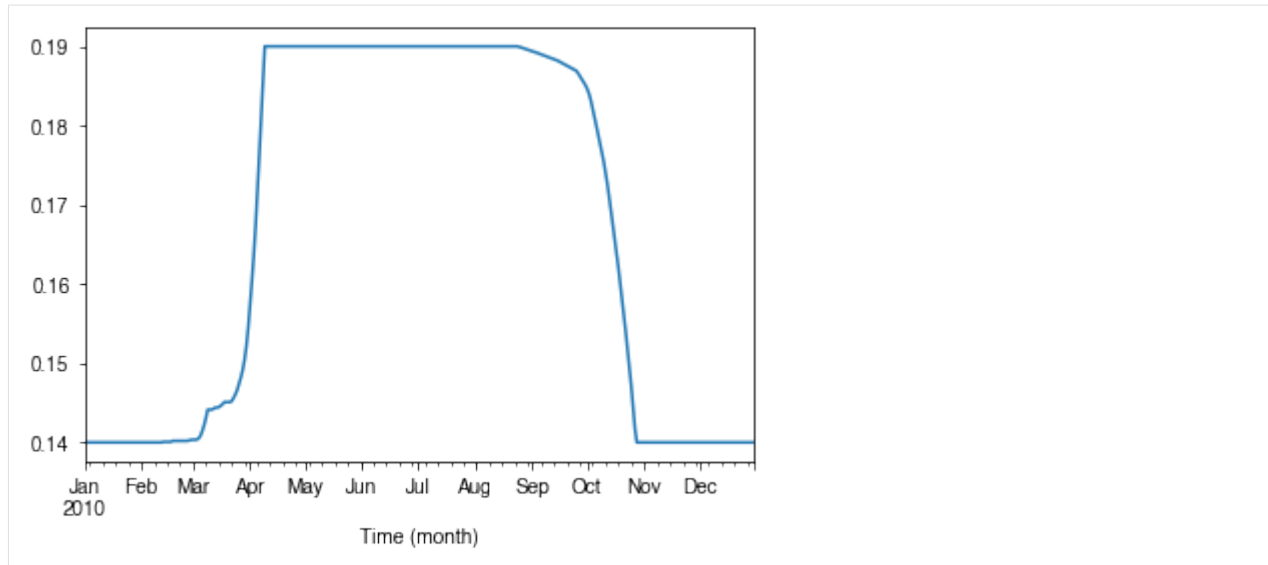
ax = pd.concat([ser_p, ser_smd_dmax], axis=1).plot(secondary_y='SMD',
      figsize=(9, 4))
_ = ax.set_xlabel('Time (month)')
```



Variability in albedo

How does albedo change over time?

```
[44]: ser_alb = df_dailystate.AlbGrass
ax = ser_alb.plot()
_ = ax.set_xlabel('Time (month)')
```

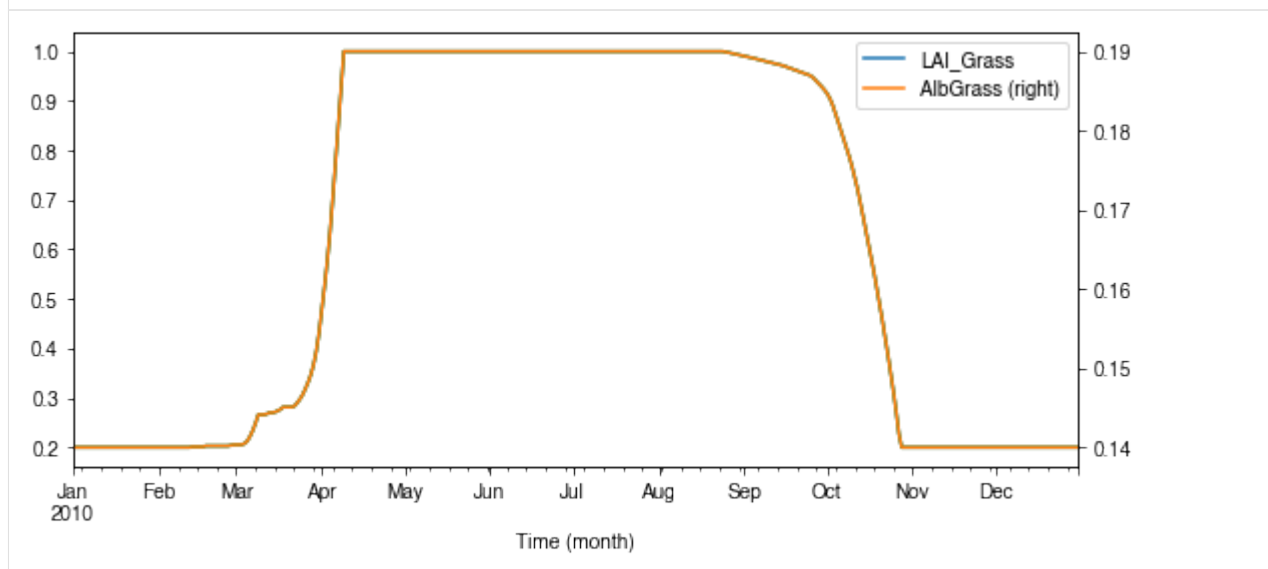


How is albedo associated with vegetation phenology?

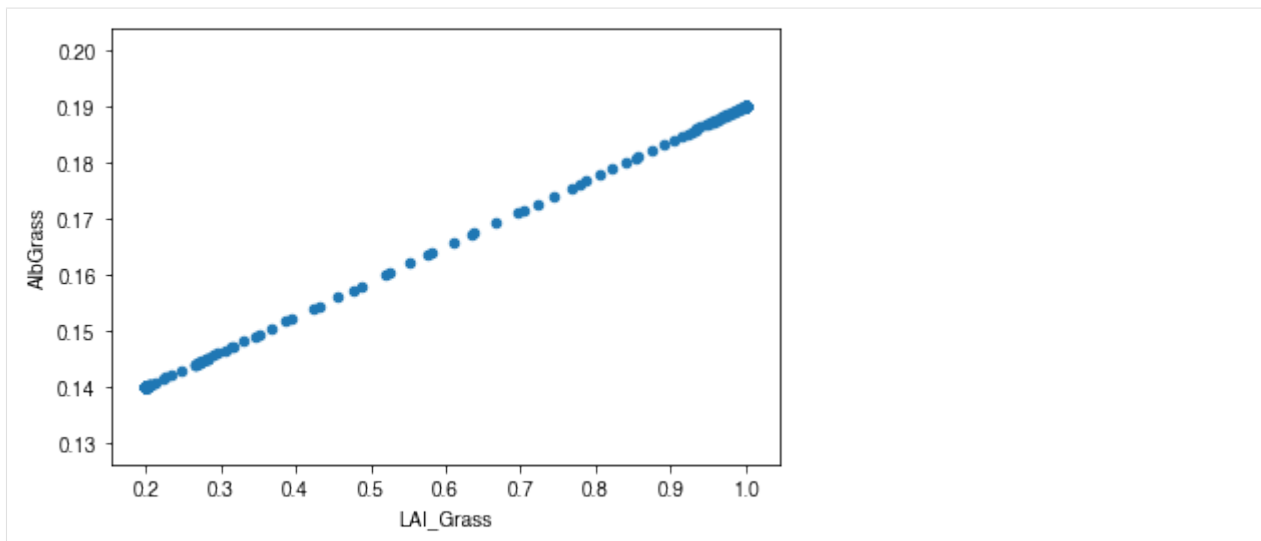
```
[45]: ser_lai = df_dailystate.LAI_Grass
pd.concat([ser_lai, ser_alb], axis=1).plot(secondary_y='AlbGrass',
                                           figsize=(9, 4))

ax = ser_lai.plot()
_ = ax.set_xlabel('Time (month)')
```

```
[45]: <matplotlib.axes._subplots.AxesSubplot at 0x7fc9989c9ac8>
```



```
[46]: ax_alb_lai = df_dailystate[['LAI_Grass', 'AlbGrass']].plot.scatter(
    x='LAI_Grass',
    y='AlbGrass',
)
ax_alb_lai.set_aspect('auto')
```



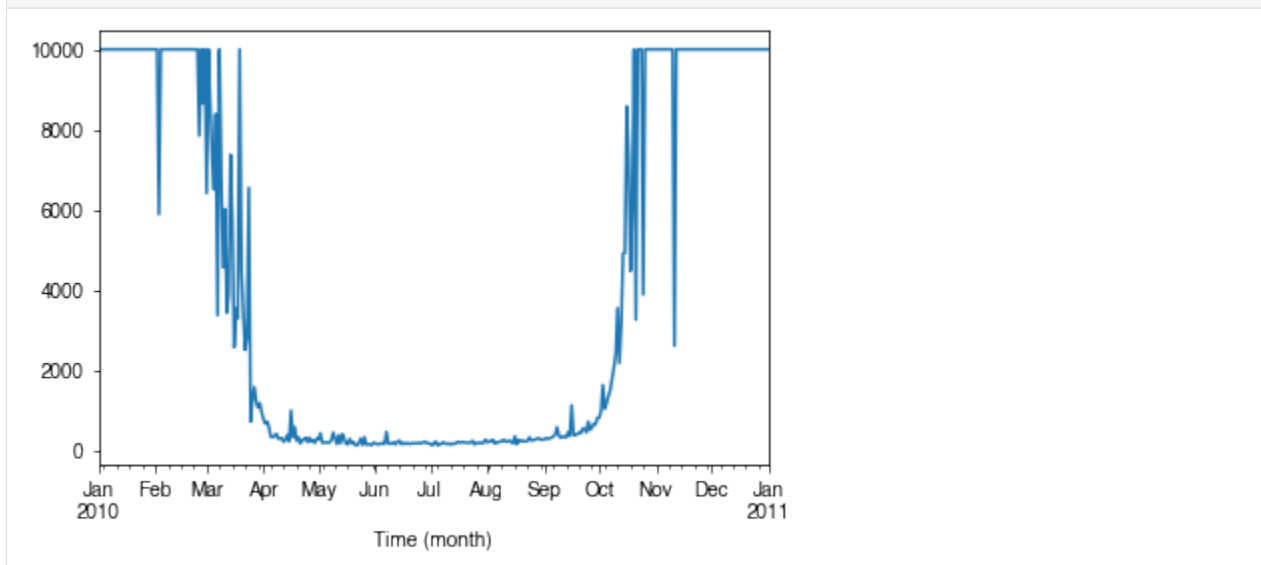
Variability in surface resistance

How does surface resistance vary over time?

```
[47]: ser_rs = df_output_suews.RS
```

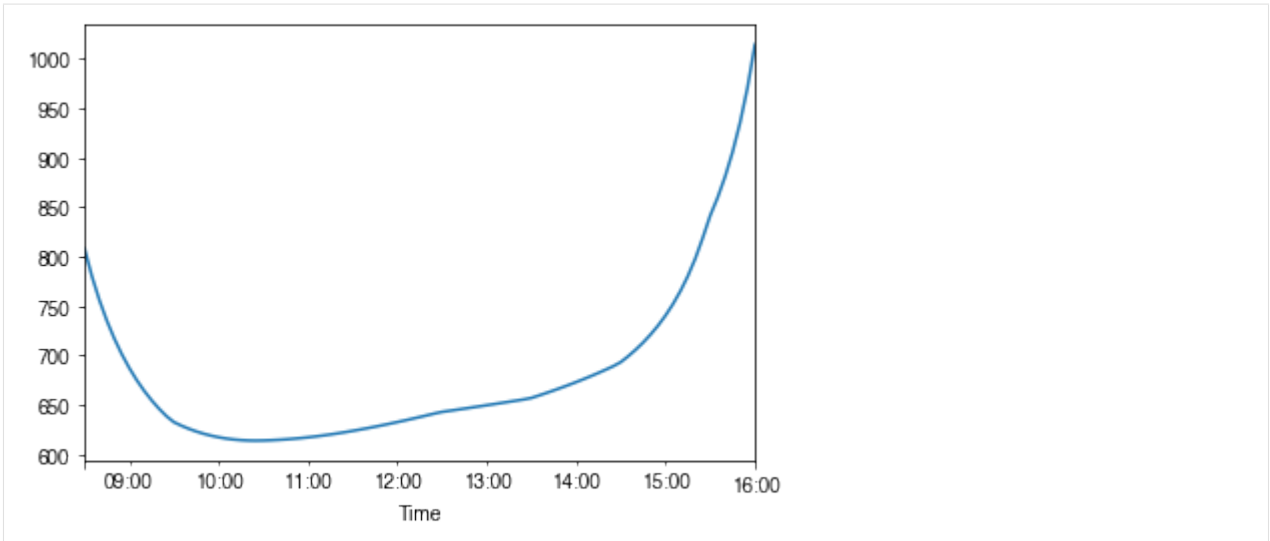
- intra-annual

```
[48]: ax = ser_rs.resample('1d').median().plot()
_ = ax.set_xlabel('Time (month)')
```

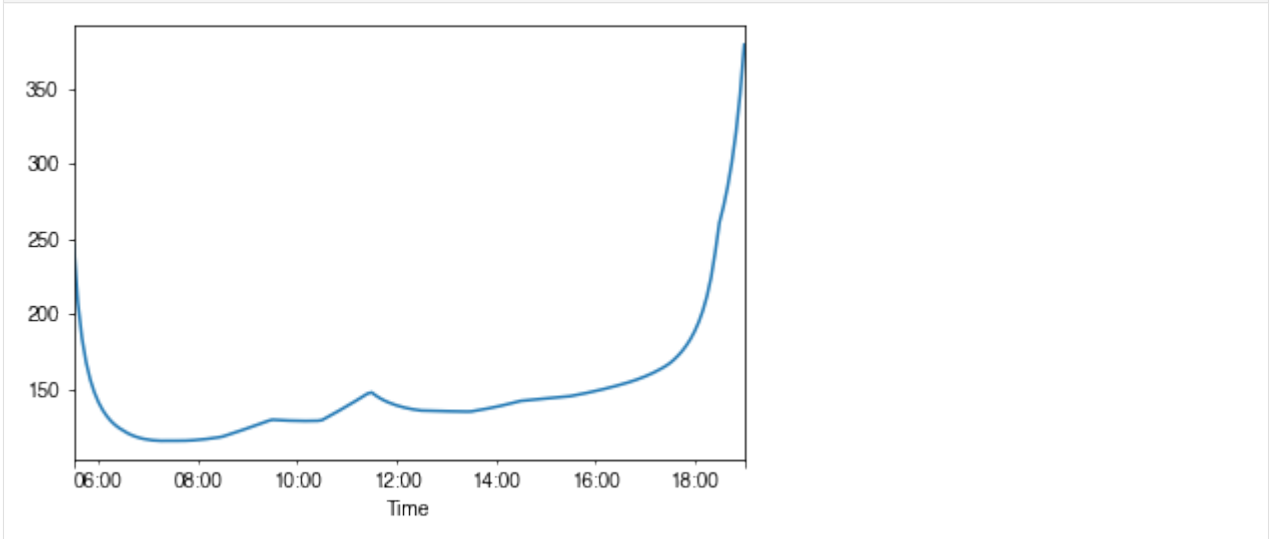


- intra-daily

```
[49]: # a winter day
ax = ser_rs.loc['2010-01-22'].between_time('0830', '1600').plot()
_ = ax.set_xlabel('Time')
```

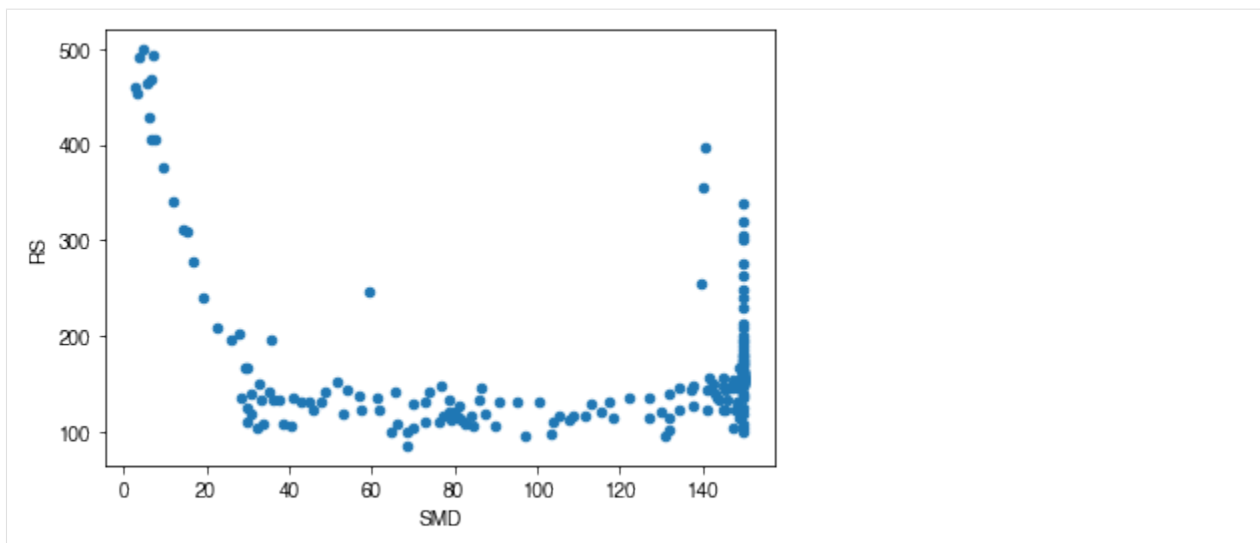


```
[50]: # a summer day
ax = ser_rs.loc['2010-07-01'].between_time('0530', '1900').plot()
_ = ax.set_xlabel('Time')
```

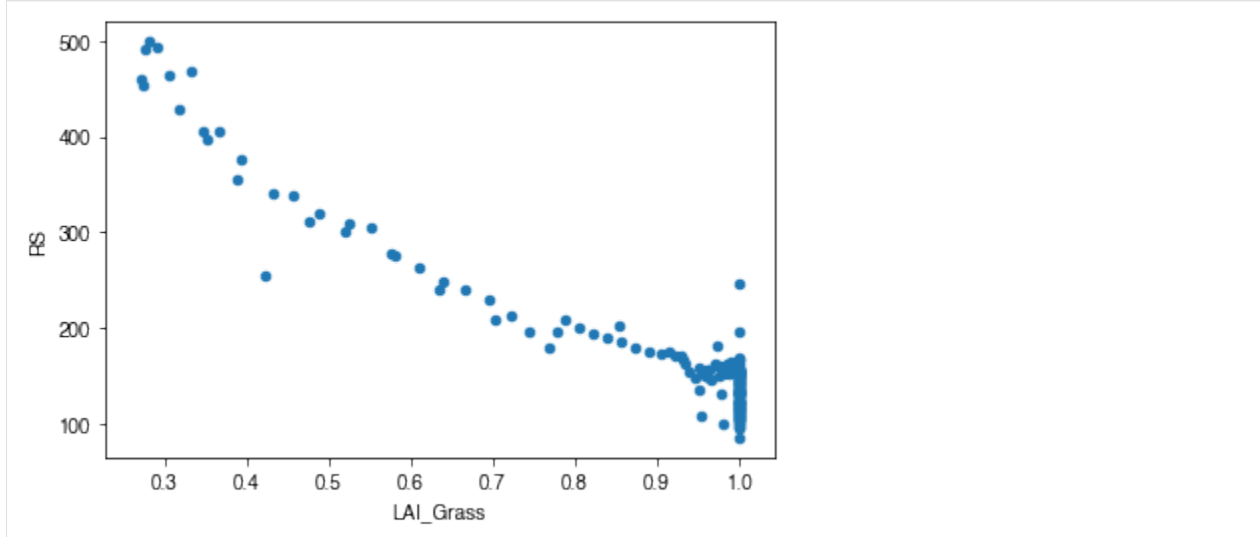


How is surface resistance associated with other surface properties?

```
[51]: # SMD
ser_smd = df_output_suews.SMD
df_x = pd.concat([ser_smd, ser_rs],
                  axis=1).between_time('1000', '1600').resample('1d').mean()
df_x = df_x.loc[df_x.RS < 500]
_ = df_x.plot.scatter(
    x='SMD',
    y='RS',
)
```



```
[52]: # LAI
df_x = pd.concat(
    [ser_lai,
     ser_rs.between_time('1000', '1600').resample('1d').mean()],
    axis=1)
df_x = df_x.loc[df_x.RS < 500]
_ = df_x.plot.scatter(
    x='LAI_Grass',
    y='RS',
)
```



How is surface resistance dependent on meteorological conditions?

```
[53]: cmap_sel = plt.cm.get_cmap('RdBu', 12)
```

```
[54]: # solar radiation
# colour by season
```

(continues on next page)

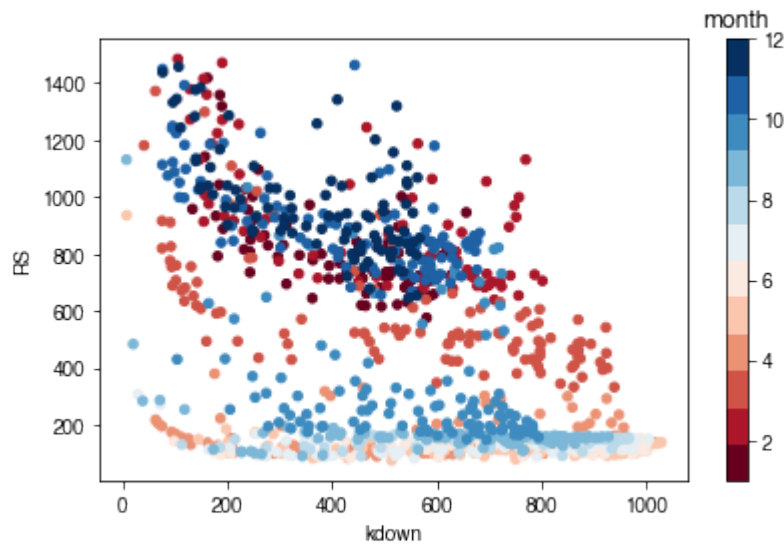
(continued from previous page)

```

ser_kdown = df_forcing_amf.kdown
df_x = pd.concat([ser_kdown, ser_rs], axis=1).between_time('1000', '1600')
df_x = df_x.loc[df_x.RS < 1500]
df_plot = df_x.iloc[::20]
ax = df_plot.plot.scatter(x='kdown',
                          y='RS',
                          c=df_plot.index.month,
                          cmap=cmap_sel,
                          sharex=False)

fig = ax.figure
_ = fig.axes[1].set_title('month')
fig.tight_layout()

```

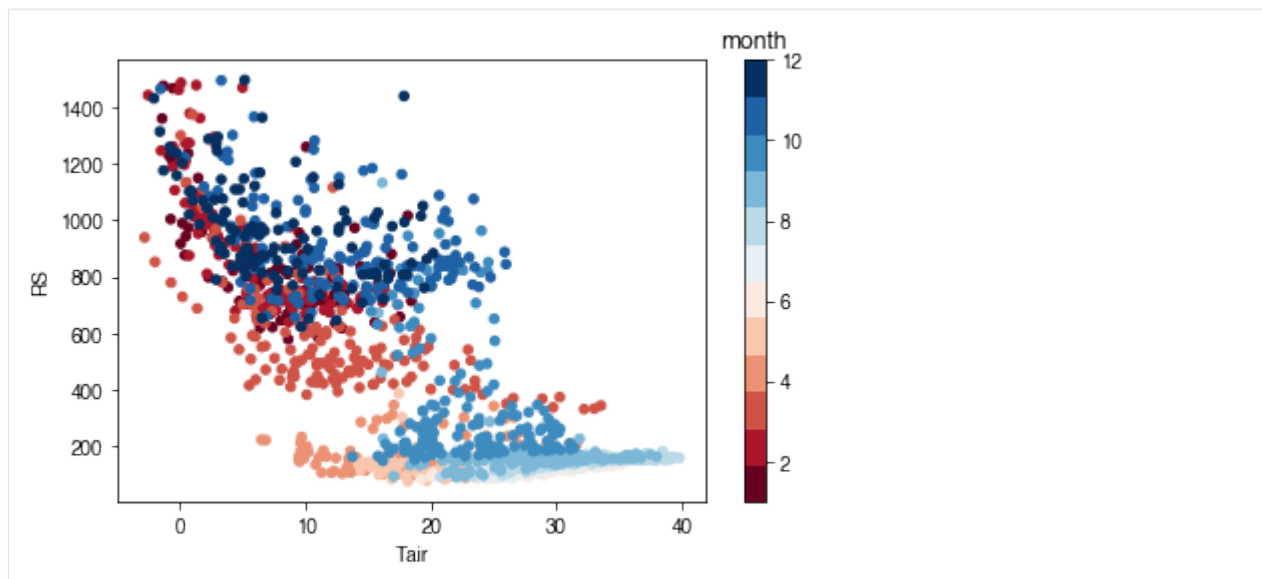


```

[55]: # air temperature
ser_ta = df_forcing_amf.Tair
df_x = pd.concat([ser_ta, ser_rs], axis=1).between_time('1000', '1600')
df_x = df_x.loc[df_x.RS < 1500]
df_plot = df_x.iloc[::15]
ax = df_plot.plot.scatter(x='Tair',
                          y='RS',
                          c=df_plot.index.month,
                          cmap=cmap_sel,
                          sharex=False)

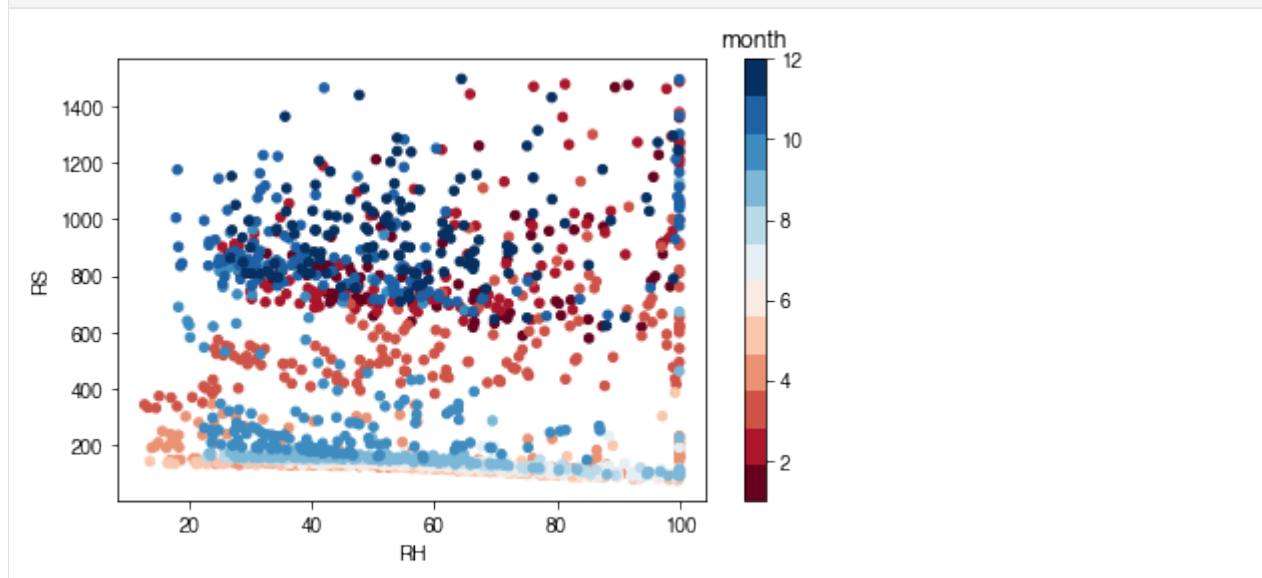
fig = ax.figure
_ = fig.axes[1].set_title('month')
fig.tight_layout()

```



```
[56]: # air humidity
ser_rh = df_forcing_amf.RH
df_x = pd.concat([ser_rh, ser_rs], axis=1).between_time('1000', '1600')
df_x = df_x.loc[df_x.RS < 1500]
df_plot = df_x.iloc[:, :15]
ax = df_plot.plot.scatter(x='RH',
                          y='RS',
                          c=df_plot.index.month,
                          cmap=cmap_sel,
                          sharex=False)

fig = ax.figure
_ = fig.axes[1].set_title('month')
fig.tight_layout()
```



- **Task:**

Based on the above plots showing RS vs. met. conditions, explore these relationships again at the intra-daily scales.

Note:

1. The Anaconda distribution is suggested as the scientific Python 3 environment for its completeness in necessary packages. Please follow the official guide for its [installation](#).
 2. Users with less experience in Python are suggested to go through the following section first before using SuPy.
-

1.5 Python 101 before SuPy

Admittedly, this header is somewhat misleading: given the enormity of Python, it's more challenging to get this section *correct* than coding SuPy per se. As such, here a collection of data analysis oriented links to useful Python resources is provided to help novices start using Python and **then** SuPy.

- [The gist of Python](#): a quick introductory blog that covers Python basics for data analysis.
- **Jupyter Notebook**: Jupyter Notebook provides a powerful notebook-based data analysis environment that SuPy users are strongly encouraged to use. Jupyter notebooks can run in browsers (desktop, mobile) either by easy local configuration or on remote servers with pre-set environments (e.g., [Google Colaboratory](#), [Microsoft Azure Notebooks](#)). In addition, Jupyter notebooks allow great shareability by incorporating source code and detailed notes in one place, which helps users to organise their computation work.

- Installation

Jupyter notebooks can be installed with pip on any desktop/server system and open .ipynb notebook files locally:

```
python3 -m pip install jupyter -U
```

- Extensions: To empower your Jupyter Notebook environment with better productivity, please check out the [Unofficial Jupyter Notebook Extensions](#). Quick introductory blogs can be found [here](#) and [here](#).
- **pandas**: [pandas](#) is heavily used in SuPy and thus better understanding of pandas is essential in SuPy workflows.
 - Introductory blogs:
 - * [Quick dive into Pandas for Data Science](#): introduction to pandas.
 - * [Basic Time Series Manipulation with Pandas](#): pandas-based time series manipulation.
 - * [Introduction to Data Visualization in Python](#): plotting using pandas and related libraries.
 - A detailed tutorial in Jupyter Notebooks:
 - * [Introduction to pandas](#)
 - * [pandas fundamentals](#)
 - * [Data Wrangling with pandas](#)

The following section was generated from `docs/source/data-structure/supy-io.ipynb`

Key IO Data Structures in SuPy

2.1 Introduction

The cell below demonstrates a minimal case of SuPy simulation with all key IO data structures included:

```
[1]: import supy as sp
df_state_init, df_forcing = sp.load_SampleData()
df_output, df_state_final = sp.run_supy(df_forcing, df_state_init)
```

- Input: SuPy requires two DataFrames to perform a simulation, which are:

- df_state_init: model initial states;
- df_forcing: forcing data.

These input data can be loaded either through calling `load_SampleData()` as shown above or using `init_supy`. Or, based on the loaded sample DataFrames, you can modify the content to create new DataFrames for your specific needs.

- Output: The output data by SuPy consists of two DataFrames:

- df_output: model output results; this is usually the basis for scientific analysis.
- df_state_final: model final states; any of its entries can be used as a df_state_init to start another SuPy simulation.

2.2 Input

2.2.1 df_state_init: model initial states

```
[2]: df_state_init.head()
```

```
[2]: var      ah_min      ah_slope_cooling      ah_slope_heating      ahprof_24hr \
ind_dim  (0,)  (1,)      (0,)  (1,)      (0,)  (1,)      (0, 0)
grid
98      15.0  15.0      2.7  2.7      2.7  2.7      0.57

var
ind_dim  (0, 1) (1, 0) (1, 1) (2, 0) (2, 1) (3, 0) (3, 1) (4, 0) ... tair24hr \
grid
98      0.65  0.45  0.49  0.43  0.46  0.4  0.47  0.4 ... (275,)
...
273.15

var
ind_dim  (276,) (277,) (278,) (279,) (280,) (281,) (282,) (283,) \
grid
98      273.15 273.15 273.15 273.15 273.15 273.15 273.15 273.15

var
ind_dim  (284,) (285,) (286,) (287,)      numcapita gridiv
grid
98      273.15 273.15 273.15 273.15      204.58      98

[1 rows x 1200 columns]
```

`df_state_init` is organised with ***grids*** in **rows** and ***their states*** in **columns**. The details of all state variables can be found in [the description page](#).

Please note the properties are stored as *flattened values* to fit into the tabular format due to the nature of DataFrame though they may actually be of higher dimension (e.g. `ahprof_24hr` with the dimension {24, 2}). To indicate the variable dimensionality of these properties, SuPy use the `ind_dim` level in columns for indices of values:

- 0 for scalars;
- (`ind_dim1`, `ind_dim2`, ...) for arrays (for a generic sense, vectors are 1D arrays).

Take `ohm_coef` below for example, it has a dimension of {8, 4, 3} according to [the description](#), which implies the actual values used by SuPy in simulations are passed in a layout as an array of the dimension {8, 4, 3}. As such, to get proper values passed in, users should follow the dimensionality requirement to prepare/modify `df_state_init`.

```
[3]: df_state_init.loc[:, 'ohm_coef']

[3]: ind_dim  (0, 0, 0)  (0, 0, 1)  (0, 0, 2)  (0, 1, 0)  (0, 1, 1)  (0, 1, 2) \
grid
98      0.719      0.194      -36.6      0.719      0.194      -36.6

ind_dim  (0, 2, 0)  (0, 2, 1)  (0, 2, 2)  (0, 3, 0)  (0, 3, 1)  (0, 3, 2) \
grid
98      0.719      0.194      -36.6      0.719      0.194      -36.6

ind_dim  (1, 0, 0)  (1, 0, 1)  (1, 0, 2)  ...  (6, 3, 0)  (6, 3, 1) \
grid
98      0.238      0.427      -16.7  ...      0.5      0.21

ind_dim  (6, 3, 2)  (7, 0, 0)  (7, 0, 1)  (7, 0, 2)  (7, 1, 0)  (7, 1, 1) \
grid
98      -39.1      0.25      0.6      -30.0      0.25      0.6

ind_dim  (7, 1, 2)  (7, 2, 0)  (7, 2, 1)  (7, 2, 2)  (7, 3, 0)  (7, 3, 1) \
grid
98      -30.0      0.25      0.6      -30.0      0.25      0.6
```

(continues on next page)

(continued from previous page)

```
ind_dim (7, 3, 2)
grid
98      -30.0

[1 rows x 96 columns]
```

2.2.2 df_forcing: forcing data

df_forcing is organised with ***temporal records*** in rows and ***forcing variables*** in columns. The details of all forcing variables can be found in [the description page](#).

The missing values can be specified with -999s, which are the default NaNs accepted by SuPy and its backend SUEWS.

```
[4]: df_forcing.head()

[4]:
```

	iy	id	it	imin	qn	qh	qe	qs	qf	\
2012-01-01 00:05:00	2012	1	0	5	-999.0	-999.0	-999.0	-999.0	-999.0	
2012-01-01 00:10:00	2012	1	0	10	-999.0	-999.0	-999.0	-999.0	-999.0	
2012-01-01 00:15:00	2012	1	0	15	-999.0	-999.0	-999.0	-999.0	-999.0	
2012-01-01 00:20:00	2012	1	0	20	-999.0	-999.0	-999.0	-999.0	-999.0	
2012-01-01 00:25:00	2012	1	0	25	-999.0	-999.0	-999.0	-999.0	-999.0	

	U	RH	Tair	pres	rain	kdown	\
2012-01-01 00:05:00	4.515	85.463333	11.77375	1001.5125	0.0	0.153333	
2012-01-01 00:10:00	4.515	85.463333	11.77375	1001.5125	0.0	0.153333	
2012-01-01 00:15:00	4.515	85.463333	11.77375	1001.5125	0.0	0.153333	
2012-01-01 00:20:00	4.515	85.463333	11.77375	1001.5125	0.0	0.153333	
2012-01-01 00:25:00	4.515	85.463333	11.77375	1001.5125	0.0	0.153333	

	snow	ldown	fcld	Wuh	xsmc	lai	kdiff	kdir	\
2012-01-01 00:05:00	-999.0	-999.0	-999.0	-999.0	-999.0	-999.0	-999.0	-999.0	
2012-01-01 00:10:00	-999.0	-999.0	-999.0	-999.0	-999.0	-999.0	-999.0	-999.0	
2012-01-01 00:15:00	-999.0	-999.0	-999.0	-999.0	-999.0	-999.0	-999.0	-999.0	
2012-01-01 00:20:00	-999.0	-999.0	-999.0	-999.0	-999.0	-999.0	-999.0	-999.0	
2012-01-01 00:25:00	-999.0	-999.0	-999.0	-999.0	-999.0	-999.0	-999.0	-999.0	

	wdir	isec
2012-01-01 00:05:00	-999.0	0.0
2012-01-01 00:10:00	-999.0	0.0
2012-01-01 00:15:00	-999.0	0.0
2012-01-01 00:20:00	-999.0	0.0
2012-01-01 00:25:00	-999.0	0.0

Note:

The index of df_forcing **SHOULD BE** strictly of DatetimeIndex type if you want create a df_forcing for SuPy simulation. The SuPy runtime time-step size is instructed by the df_forcing with its index information.

The information below indicates SuPy will run at a 5 min (i.e. 300 s) time-step if driven by this specific df_forcing:

```
[5]: freq_forcing=df_forcing.index.freq
freq_forcing
```

```
[5]: <300 * Seconds>
```

2.3 Output

2.3.1 df_output: model output results

df_output is organised with ***temporal records of grids*** in rows and ***output variables of different groups*** in columns. The details of all forcing variables can be found in [the description page](#).

```
[6]: df_output.head()
```

```
[6]: group          SUEWS          \
var          Kdown          Kup          Ldown          Lup
grid datetime
98  2012-01-01 00:05:00  0.153333  0.018279  344.310184  371.986259
    2012-01-01 00:10:00  0.153333  0.018279  344.310184  371.986259
    2012-01-01 00:15:00  0.153333  0.018279  344.310184  371.986259
    2012-01-01 00:20:00  0.153333  0.018279  344.310184  371.986259
    2012-01-01 00:25:00  0.153333  0.018279  344.310184  371.986259

group          \
var          Tsurf          QN          QF          QS
grid datetime
98  2012-01-01 00:05:00  11.775615 -27.541021  40.574001 -46.53243
    2012-01-01 00:10:00  11.775615 -27.541021  39.724283 -46.53243
    2012-01-01 00:15:00  11.775615 -27.541021  38.874566 -46.53243
    2012-01-01 00:20:00  11.775615 -27.541021  38.024849 -46.53243
    2012-01-01 00:25:00  11.775615 -27.541021  37.175131 -46.53243

group          \
var          QH          QE          QHlumps          QElumps          QHresis
grid datetime
98  2012-01-01 00:05:00  62.420064  3.576493  49.732605  9.832804  0.042327
    2012-01-01 00:10:00  61.654096  3.492744  48.980360  9.735333  0.042294
    2012-01-01 00:15:00  60.885968  3.411154  48.228114  9.637861  0.042260
    2012-01-01 00:20:00  60.115745  3.331660  47.475869  9.540389  0.042226
    2012-01-01 00:25:00  59.343488  3.254200  46.723623  9.442917  0.042192

group          ... DailyState          \
var          Rain  Irr          ...  WU_Grass2  WU_Grass3  deltaLAI
grid datetime          ...
98  2012-01-01 00:05:00  0.0  0.0          ...          NaN          NaN          NaN
    2012-01-01 00:10:00  0.0  0.0          ...          NaN          NaN          NaN
    2012-01-01 00:15:00  0.0  0.0          ...          NaN          NaN          NaN
    2012-01-01 00:20:00  0.0  0.0          ...          NaN          NaN          NaN
    2012-01-01 00:25:00  0.0  0.0          ...          NaN          NaN          NaN

group          \
var          LAIlumps  AlbSnow  DensSnow_Paved  DensSnow_Bldgs
grid datetime
98  2012-01-01 00:05:00          NaN          NaN          NaN          NaN
    2012-01-01 00:10:00          NaN          NaN          NaN          NaN
    2012-01-01 00:15:00          NaN          NaN          NaN          NaN
    2012-01-01 00:20:00          NaN          NaN          NaN          NaN
    2012-01-01 00:25:00          NaN          NaN          NaN          NaN
```

(continues on next page)

(continued from previous page)

```

group
var          DensSnow_EveTr DensSnow_DecTr DensSnow_Grass \
grid datetime
98  2012-01-01 00:05:00      NaN          NaN          NaN
    2012-01-01 00:10:00      NaN          NaN          NaN
    2012-01-01 00:15:00      NaN          NaN          NaN
    2012-01-01 00:20:00      NaN          NaN          NaN
    2012-01-01 00:25:00      NaN          NaN          NaN

group
var          DensSnow_BSoil DensSnow_Water  a1  a2  a3
grid datetime
98  2012-01-01 00:05:00      NaN          NaN NaN NaN NaN
    2012-01-01 00:10:00      NaN          NaN NaN NaN NaN
    2012-01-01 00:15:00      NaN          NaN NaN NaN NaN
    2012-01-01 00:20:00      NaN          NaN NaN NaN NaN
    2012-01-01 00:25:00      NaN          NaN NaN NaN NaN

[5 rows x 218 columns]

```

`df_output` are recorded at the same temporal resolution as `df_forcing`:

```
[7]: freq_out = df_output.index.levels[1].freq
      (freq_out, freq_out == freq_forcing)
```

```
[7]: (<300 * Seconds>, True)
```

2.3.2 `df_state_final`: model final states

`df_state_final` has the identical data structure as `df_state_init` except for the extra level `datetime` in index, which stores the temporal information associated with model states. Such structure can facilitate the reuse of it as initial model states for other simulations (e.g., diagnostics of runtime model states with `save_state=True` set in `run_supy`; or simply using it as the initial conditions for future simulations starting at the ending times of previous runs).

The meanings of state variables in `df_state_final` can be found in [the description page](#).

```

[8]: df_state_final.head()

[8]: var          aerodynamicresistancemethod ah_min \
ind_dim
datetime      grid
2012-01-01 00:05:00 98      2    15.0  15.0
2013-01-01 00:05:00 98      2    15.0  15.0

var          ah_slope_cooling      ah_slope_heating \
ind_dim          (0,) (1,)          (0,) (1,)
datetime      grid
2012-01-01 00:05:00 98      2.7  2.7          2.7  2.7
2013-01-01 00:05:00 98      2.7  2.7          2.7  2.7

var          ahprof_24hr \
ind_dim          (0, 0) (0, 1) (1, 0) (1, 1) (2, 0) (2, 1)
datetime      grid
2012-01-01 00:05:00 98      0.57  0.65  0.45  0.49  0.43  0.46

```

(continues on next page)

(continued from previous page)

```

2013-01-01 00:05:00 98      0.57  0.65  0.45  0.49  0.43  0.46

var
ind_dim      (3, 0) (3, 1) ... wuprofm_24hr \
datetime      grid      ...
2012-01-01 00:05:00 98      0.4  0.47 ...      -999.0 -999.0 -999.0
2013-01-01 00:05:00 98      0.4  0.47 ...      -999.0 -999.0 -999.0

var
ind_dim      (19, 1) (20, 0) (20, 1) (21, 0) (21, 1) (22, 0) \
datetime      grid
2012-01-01 00:05:00 98      -999.0 -999.0 -999.0 -999.0 -999.0 -999.0
2013-01-01 00:05:00 98      -999.0 -999.0 -999.0 -999.0 -999.0 -999.0

var
ind_dim      (22, 1) (23, 0) (23, 1)      z z0m_in zdm_in
datetime      grid
2012-01-01 00:05:00 98      -999.0 -999.0 -999.0 49.6      1.9      14.2
2013-01-01 00:05:00 98      -999.0 -999.0 -999.0 49.6      1.9      14.2

[2 rows x 1200 columns]

```

End of doc/data-structure/supy-io.ipynb

3.1 Top-level Functions

<code>init_supy(path_init[, force_reload, check_input])</code>	Initialise supy by loading initial model states.
<code>load_forcing_grid(path_runcontrol, grid[, ...])</code>	Load forcing data for a specific grid included in the index of <code>df_state_init</code> .
<code>run_supy(df_forcing, df_state_init[, ...])</code>	Perform supy simulation.
<code>save_supy(df_output, df_state_final, freq_s, ...)</code>	Save SuPy run results to files
<code>load_SampleData()</code>	Load sample data for quickly starting a demo run.
<code>show_version()</code>	print SuPy and supy_driver version information.

3.1.1 supy.init_supy

`supy.init_supy(path_init: str, force_reload=True, check_input=False) → pandas.core.frame.DataFrame`
 Initialise supy by loading initial model states.

Parameters

- **path_init** (*str*) –
Path to a file that can initialise SuPy, which can be either of the follows:
 - SUEWS `RunControl.nml`: a namelist file for SUEWS configurations
 - SuPy `df_state.csv`: a CSV file including model states produced by a SuPy run via `supy.save_supy()`
- **force_reload** (*boolean, optional*) – Flag to force reload all initialisation files by clearing all cached states, with default value `True` (i.e., force reload all files). Note: If the number of simulation grids is large (e.g., > 100), `force_reload=False` is strongly recommended for better performance.

Returns `df_state_init` – Initial model states. See *df_state variables* for details.

Return type `pandas.DataFrame`

Examples

1. Use `RunControl.nml` to initialise SuPy

```
>>> path_init = "~/SUEWS_sims/RunControl.nml"
>>> df_state_init = supy.init_supy(path_init)
```

2. Use `df_state.csv` to initialise SuPy

```
>>> path_init = "~/SuPy_res/df_state_test.csv"
>>> df_state_init = supy.init_supy(path_init)
```

3.1.2 supy.load_forcing_grid

`supy.load_forcing_grid(path_runcontrol: str, grid: int, check_input=False) → pandas.core.frame.DataFrame`
Load forcing data for a specific grid included in the index of `df_state_init`.

Parameters

- **path_runcontrol** (*str*) – Path to SUEWS `RunControl.nml`
- **grid** (*int*) – Grid number

Returns `df_forcing` – Forcing data. See *df_forcing variables* for details.

Return type `pandas.DataFrame`

Examples

```
>>> path_runcontrol = "~/SUEWS_sims/RunControl.nml" # a valid path to
↳ `RunControl.nml`
>>> df_state_init = supy.init_supy(path_runcontrol) # get `df_state_init`
>>> grid = df_state_init.index[0] # first grid number included in `df_state_init`
>>> df_forcing = supy.load_forcing_grid(path_runcontrol, grid) # get df_forcing
```

3.1.3 supy.run_supy

`supy.run_supy(df_forcing: pandas.core.frame.DataFrame, df_state_init: pandas.core.frame.DataFrame, save_state=False, chunk_day=3660, logging_level=20, check_input=False, serial_mode=False) → Tuple[pandas.core.frame.DataFrame, pandas.core.frame.DataFrame]`

Perform supy simulation.

Parameters

- **df_forcing** (*pandas.DataFrame*) – forcing data for all grids in `df_state_init`.
- **df_state_init** (*pandas.DataFrame*) – initial model states; or a collection of model states with multiple timestamps, whose last temporal record will be used as the initial model states.
- **save_state** (*bool, optional*) – flag for saving model states at each time step, which can be useful in diagnosing model runtime performance or performing a restart run. (the default is `False`, which instructs supy not to save runtime model states).

- **chunk_day** (*int, optional*) – chunk size (`chunk_day` days) to split simulation periods so memory usage can be reduced. (the default is 3660, which implies ~10-year forcing chunks used in simulations).
- **logging_level** (*logging level*) – one of these values [50 (CRITICAL), 40 (ERROR), 30 (WARNING), 20 (INFO), 10 (DEBUG)]. A lower value informs SuPy for more verbose logging info.
- **check_input** (*bool, optional*) – flag for checking validity of input: `df_forcing` and `df_state_init`. If set to `True`, any detected invalid input will stop SuPy simulation; a `False` flag will bypass such validation and may incur kernel error if any invalid input. *Note: such checking procedure may take some time if the input is large.* (the default is `False`, which bypasses the validation).
- **serial_mode** (*bool, optional*) – If set to `True`, SuPy simulation will be conducted in serial mode; a `False` flag will try parallel simulation if possible (Windows not supported, i.e., always serial). (the default is `False`).

Returns

df_output, df_state_final –

- `df_output`: *output results*
- `df_state_final`: *final model states*

Return type `Tuple[pandas.DataFrame, pandas.DataFrame]`

Examples

```
>>> df_output, df_state_final = supy.run_supy(df_forcing, df_state_init)
```

3.1.4 supy.save_supy

`supy.save_supy(df_output: pandas.core.frame.DataFrame, df_state_final: pandas.core.frame.DataFrame, freq_s: int = 3600, site: str = "", path_dir_save: str = PosixPath('.'), path_runcontrol: str = None, save_tstep=False, logging_level=50, output_level=1, debug=False) → list`

Save SuPy run results to files

Parameters

- **df_output** (*pandas.DataFrame*) – DataFrame of output
- **df_state_final** (*pandas.DataFrame*) – DataFrame of final model states
- **freq_s** (*int, optional*) – Output frequency in seconds (the default is 3600, which indicates hourly output)
- **site** (*str, optional*) – Site identifier (the default is "", which indicates site identifier will be left empty)
- **path_dir_save** (*str, optional*) – Path to directory to saving the files (the default is `Path('.')`, which indicates the current working directory)
- **path_runcontrol** (*str, optional*) – Path to SUEWS `RunControl.nml`, which, if set, will be preferably used to derive `freq_s`, `site` and `path_dir_save`. (the default is `None`, which is unset)

- **save_tstep** (*bool*, *optional*) – whether to save results in temporal resolution as in simulation (which may result very large files and slow progress), by default False.
- **logging_level** (*logging level*) – one of these values [50 (CRITICAL), 40 (ERROR), 30 (WARNING), 20 (INFO), 10 (DEBUG)]. A lower value informs SuPy for more verbose logging info.
- **output_level** (*integer*, *optional*) – option to determine selection of output variables, by default 1. Notes: 0 for all but snow-related; 1 for all; 2 for a minimal set without land cover specific information.
- **debug** (*bool*, *optional*) – whether to enable debug mode (e.g., writing out in serial mode, and other debug uses), by default False.

Returns a list of paths of saved files

Return type `list`

Examples

1. save results of a supy run to the current working directory with default settings

```
>>> list_path_save = supy.save_supy(df_output, df_state_final)
```

2. save results according to settings in `RunControl.nml`

```
>>> list_path_save = supy.save_supy(df_output, df_state_final, path_runcontrol=
↳ 'path/to/RunControl.nml')
```

3. save results of a supy run at resampling frequency of 1800 s (i.e., half-hourly results) under the site code Test to a customised location 'path/to/some/dir'

```
>>> list_path_save = supy.save_supy(df_output, df_state_final, freq_s=1800, site=
↳ 'Test', path_dir_save='path/to/some/dir')
```

3.1.5 supy.load_SampleData

`supy.load_SampleData()` → `Tuple[pandas.core.frame.DataFrame, pandas.core.frame.DataFrame]`

Load sample data for quickly starting a demo run.

Returns

df_state_init, df_forcing –

- `df_state_init`: *initial model states*
- `df_forcing`: *forcing data*

Return type `Tuple[pandas.DataFrame, pandas.DataFrame]`

Examples

```
>>> df_state_init, df_forcing = supy.load_SampleData()
```

3.1.6 supy.show_version

`supy.show_version()`
print SuPy and supy_driver version information.

3.2 Utility Functions

3.2.1 ERA-5 Data Downloader

<code>download_era5(lat_x, lon_x, start, end[, ...])</code>	Generate ERA-5 cdsapi-based requests and download data for area of interests.
<code>gen_forcing_era5(lat_x, lon_x, start, end[, ...])</code>	Generate SUEWS forcing files using ERA-5 data.

supy.util.download_era5

`supy.util.download_era5(lat_x: float, lon_x: float, start: str, end: str, dir_save=PosixPath('.'), grid=[0.125, 0.125], scale=0) → dict`
Generate ERA-5 cdsapi-based requests and download data for area of interests.

Parameters

- **lat_x** (*float*) – Latitude of centre at the area of interest.
- **lon_x** (*float*) – Longitude of centre at the area of interest.
- **start** (*str*) – Any datetime-like string that can be parsed by `pandas.daterange()`.
- **end** (*str*) – Any datetime-like string that can be parsed by `pandas.daterange()`.
- **grid** (*list, optional*) – grid size used in CDS request API, by default `[0.125, 0.125]`.
- **scale** (*int, optional*) – scaling factor that determines the area of interest (i.e., `area=grid[0]*scale`), by default 0.
- **dir_save** (*Path or path-like string*) – path to directory for saving downloaded ERA5 netCDF files.

Returns key: name of downloaded file. value: CDS API request used for downloading the file named by the corresponding key.

Return type `dict`

Note: This function uses CDS API to download ERA5 data; follow this for configuration first: <https://cds.climate.copernicus.eu/api-how-to>

supy.util.gen_forcing_era5

`supy.util.gen_forcing_era5(lat_x: float, lon_x: float, start: str, end: str, dir_save=PosixPath('.'), grid=[0.125, 0.125], hgt_agl_diag=100.0, scale=0, simple_mode=True) → list`
Generate SUEWS forcing files using ERA-5 data.

Parameters

- **lat_x** (*float*) – Latitude of centre at the area of interest.

- **lon_x** (*float*) – Longitude of centre at the area of interest.
- **start** (*str*) – Any datetime-like string that can be parsed by `pandas.daterange()`.
- **end** (*str*) – Any datetime-like string that can be parsed by `pandas.daterange()`.
- **dir_save** (*Path or path-like string*) – path to directory for saving downloaded ERA5 netCDF files.
- **grid** (*list, optional*) – grid size used in CDS request API, by default `[0.125, 0.125]`.
- **hgt_agl_diag** (*float*) – height above ground level to diagnose forcing variables, by default 0; the ground level is taken from ERA5 grid altitude.
- **scale** (*int, optional*) – scaling factor that determines the area of interest (i.e., `area=grid[0]*scale`), by default 0
- **simple_mode** (*boolean*) – if use the *simple* mode for diagnosing the forcing variables, by default `True`. In the simple mode, temperature is diagnosed using environmental lapse rate 6.5 K/km and wind speed using MOST under neutral condition. If `False`, MOST with consideration of stability conditions will be used to diagnose forcing variables.

Returns A list of files in SUEWS forcing input format.

Return type List

Note:

1. This function uses CDS API to download ERA5 data; follow this for configuration first: <https://cds.climate.copernicus.eu/api-how-to>
2. The generated forcing files can be imported using `supy.util.read_forcing` to get simulation-ready `pandas.DataFrame`'s.
3. See Section 3.10.2 and 3.10.3 in the reference for details of diagnostics calculation.

ECMWF, S. P. (2016). In IFS documentation CY41R2 Part IV: Physical Processes. ECMWF: Reading, UK, 111-113. <https://www.ecmwf.int/en/eLibrary/16648-part-iv-physical-processes>

3.2.2 Typical Meteorological Year

<code>gen_epw(df_output[, path_epw, ratio_dif_dir])</code>	Generate an epw file of uTMY (urbanised Typical Meteorological Year) using SUEWS simulation results
<code>read_epw(path_epw)</code>	Read in epw file as a DataFrame

`supy.util.gen_epw`

`supy.util.gen_epw(df_output: pandas.core.frame.DataFrame, path_epw=PosixPath('TMY.epw'), ratio_dif_dir=0.15) → Tuple[pandas.core.frame.DataFrame, str, pathlib.Path]`
Generate an epw file of uTMY (urbanised Typical Meteorological Year) using SUEWS simulation results

Parameters

- **df_output** (*pd.DataFrame*) – SUEWS simulation results.
- **path_epw** (*Path, optional*) – Path to store generated epw file, by default `Path('./uTMY.epw')`

- **ratio_dif_dir** (*float*, *optional*) – Ratio between direct and diffuse solar radiation, by default 0.15

Returns**df_epw, text_meta, path_epw** –

- **df_epw**: uTMY result
- **text_meta**: meta-info text
- **path_epw**: path to generated epw file

Return type Tuple[pd.DataFrame, str, Path]**supy.util.read_epw**`supy.util.read_epw(path_epw: pathlib.Path) → pandas.core.frame.DataFrame`

Read in epw file as a DataFrame

Parameters **path_epw** (*Path*) – path to epw file**Returns** **df_tmy** – TMY results of epw file**Return type** pd.DataFrame**3.2.3 Gap Filling**`fill_gap_all(ser_to_fill[, freq])`

Fill all gaps in a time series using data from neighbouring divisions of 'freq'

supy.util.fill_gap_all`supy.util.fill_gap_all(ser_to_fill: pandas.core.series.Series, freq='1D') → pandas.core.series.Series`

Fill all gaps in a time series using data from neighbouring divisions of 'freq'

Parameters

- **ser_to_fill** (*pd.Series*) – Time series to gap-fill
- **freq** (*str*, *optional*) – Frequency to identify gapped divisions, by default '1D'

Returns

- **ser_test_filled** (*pd.Series*) – Gap-filled time series.
- *Patterns*
- *_____*
- **010** (*missing data in division between others with no missing data*)
- **01** (*missing data in division after one with no missing data*)
- **10** (*division with missing data before one with no missing data*)

3.2.4 OHM

<code>derive_ohm_coef</code> (<code>ser_QS</code> , <code>ser_QN</code>)	A function to linearly fit two independant variables to a dependent one.
<code>sim_ohm</code> (<code>ser_qn</code> , <code>a1</code> , <code>a2</code> , <code>a3</code>)	Calculate QS using OHM (Objective Hysteresis Model).

`supy.util.derive_ohm_coef`

`supy.util.derive_ohm_coef`(`ser_QS`, `ser_QN`)

A function to linearly fit two independant variables to a dependent one. Input params: `QS_Ser`: The dependent variable QS (Surface heat storage). Pandas Series.

`QN_Ser`: The first independent variable (Net all wave radiation). Pandas Series. `dt`: The time interval with which the rate of change of QN is calculated. Float (hours).

Returns: `a1`, `a2` coefficients and `a3` (intercept)

`supy.util.sim_ohm`

`supy.util.sim_ohm`(`ser_qn`: `pandas.core.series.Series`, `a1`: `float`, `a2`: `float`, `a3`: `float`) → `pandas.core.series.Series`
Calculate QS using OHM (Objective Hysteresis Model).

Parameters

- `ser_qn` (`pd.Series`) – net all-wave radiation.
- `a1` (`float`) – `a1` of OHM coefficients.
- `a2` (`float`) – `a2` of OHM coefficients.
- `a3` (`float`) – `a3` of OHM coefficients.

Returns heat storage flux calculated by OHM.

Return type `pd.Series`

3.2.5 Surface Conductance

<code>cal_gs_mod</code> (<code>kd</code> , <code>ta_k</code> , <code>rh</code> , <code>pa</code> , <code>smd</code> , <code>lai</code> , <code>g_cst</code>)	Model surface conductance/resistance using phenology and atmospheric forcing conditions.
<code>cal_gs_obs</code> (<code>qh</code> , <code>qe</code> , <code>ta</code> , <code>rh</code> , <code>pa</code>)	Calculate surface conductance based on observations, notably turbulent fluxes.
<code>calib_g</code> (<code>df_fc_suews</code> [, <code>g_max</code> , <code>lai_max</code> , <code>s1</code> , ...])	Calibrate parameters for modelling surface conductance over vegetated surfaces using LMFIT.

`supy.util.cal_gs_mod`

`supy.util.cal_gs_mod`(`kd`, `ta_k`, `rh`, `pa`, `smd`, `lai`, `g_cst`, `g_max=30.0`, `lai_max=6.0`, `s1=5.56`)

Model surface conductance/resistance using phenology and atmospheric forcing conditions.

Parameters

- `kd` (`numeric`) – Incoming solar radiation [`W m-2`]
- `ta_k` (`numeric`) – Air temperature [`K`]
- `rh` (`numeric`) – Relative humidity [%]

- **pa** (*numeric*) – Air pressure
- **smd** (*numeric*) – Soil moisture deficit [mm]
- **lai** (*numeric*) – Leaf area index [m² m⁻²]
- **g_cst** (*size-6 array*) – Parameters to determine surface conductance/resistance: g1 (LAI related), g2 (solar radiation related), g3 (humidity related), g4 (humidity related), g5 (air temperature related), g6 (soil moisture related)
- **g_max** (*numeric, optional*) – Maximum surface conductance [mm s⁻¹], by default 30
- **lai_max** (*numeric, optional*) – Maximum LAI [m² m⁻²], by default 6
- **s1** (*numeric, optional*) – Wilting point (WP=s1/g6, indicated as deficit [mm]) related parameter, by default 5.56

Returns Modelled surface conductance [mm s⁻¹]

Return type numeric

supy.util.cal_gs_obs

`supy.util.cal_gs_obs(qh, qe, ta, rh, pa)`

Calculate surface conductance based on observations, notably turbulent fluxes.

Parameters

- **qh** (*numeric*) – Sensible heat flux [W m⁻²]
- **qe** (*numeric*) – Latent heat flux [W m⁻²]
- **ta** (*numeric*) – Air temperature [K]
- **rh** (*numeric*) – Relative humidity [%]
- **pa** (*numeric*) – Air pressure [Pa]

Returns Surface conductance based on observations [mm s⁻¹]

Return type numeric

supy.util.calib_g

`supy.util.calib_g(df_fc_suews, g_max=33.1, lai_max=5.9, s1=5.56, method='cobyla', prms_init=None, debug=False)`

Calibrate parameters for modelling surface conductance over vegetated surfaces using LMFIT.

Parameters

- **df_fc_suews** (*pandas.DataFrame*) – DataFrame in SuPy forcing format
- **g_max** (*numeric, optional*) – Maximum surface conductance [mm s⁻¹], by default 30
- **lai_max** (*numeric, optional*) – Maximum LAI [m² m⁻²], by default 6
- **s1** (*numeric, optional*) – Wilting point (WP=s1/g6, indicated as deficit [mm]) related parameter, by default 5.56
- **method** (*str, optional*) – Method used in minimisation by `lmfit.minimize`: details refer to its method.

- **prms_init** (*lmfit.Parameters*) – Initial parameters for calibration
- **debug** (*bool*, *optional*) – Option to output final calibrated `ModelResult`, by default `False`

Returns

dict, or 'ModelResult <lmfit –

1. dict: {parameter_name -> best_fit_value}
2. `ModelResult`

Note: Parameters for surface conductance: g1 (LAI related), g2 (solar radiation related), g3 (humidity related), g4 (humidity related), g5 (air temperature related), g6 (soil moisture related)

Return type `ModelResult` if `debug==True`

Note: For calibration validity, turbulent fluxes, QH and QE, in `df_fc_suews` should ONLY be observations, i.e., interpolated values should be avoided. To do so, please place `np.nan` as missing values for QH and QE.

3.2.6 WRF-SUEWS

<code>extract_reclassification(path_nml)</code>	Extract reclassification info from <code>path_nml</code> as a <code>DataFrame</code> .
<code>plot_reclassification(path_nml[, path_save, ...])</code>	Produce Sankey Diagram to visualise the reclassification specified in <code>path_nml</code>

supy.util.extract_reclassification

`supy.util.extract_reclassification(path_nml: str) → pandas.core.frame.DataFrame`

Extract reclassification info from `path_nml` as a `DataFrame`.

Parameters `path_nml` (*str*) – Path to `namelist.suews`

Returns Reclassification `DataFrame` with rows for WRF land covers while columns for SUEWS.

Return type `pd.DataFrame`

supy.util.plot_reclassification

`supy.util.plot_reclassification(path_nml: str, path_save='LC-WRF-SUEWS.png', width=800, height=360, top=10, bottom=10, left=260, right=60)`

Produce Sankey Diagram to visualise the reclassification specified in `path_nml`

Parameters

- **path_nml** (*str*) – Path to `namelist.suews`
- **path_save** (*str*, *optional*) – Path to save Sankey diagram, by default 'LC-WRF-SUEWS.png'
- **width** (*int*, *optional*) – Width of diagram, by default 800
- **height** (*int*, *optional*) – Height of diagram, by default 360

- **top** (*int*, *optional*) – Top margin of diagram, by default 10
- **bottom** (*int*, *optional*) – Bottom margin of diagram, by default 10
- **left** (*int*, *optional*) – Left margin of diagram, by default 260
- **right** (*int*, *optional*) – Right margin of diagram, by default 60

Returns Sankey Diagram showing the reclassification.

Return type Sankey Diagram

3.2.7 Plotting

<code>plot_comp(df_var[, fig, ax])</code>	Produce a scatter plot with linear regression line to compare simulation results and observations.
<code>plot_day_clm(df_var[, fig, ax])</code>	Produce a ensemble diurnal climatologies with uncertainties shown in inter-quartile ranges.

supy.util.plot_comp

`supy.util.plot_comp(df_var, fig=None, ax=None)`

Produce a scatter plot with linear regression line to compare simulation results and observations.

Parameters **df_var** (*pd.DataFrame*) – DataFrame containing variables to plot with datetime as index. Two columns, ‘Obs’ and ‘Sim’ for observations and simulation results, respectively, must exist.

Returns figure showing 1:1 line plot

Return type *MPL.figure*

supy.util.plot_day_clm

`supy.util.plot_day_clm(df_var, fig=None, ax=None)`

Produce a ensemble diurnal climatologies with uncertainties shown in inter-quartile ranges.

Parameters **df_var** (*pd.DataFrame*) – DataFrame containing variables to plot with datetime as index

Returns figure showing median lines and IQR in shadings

Return type *MPL.figure*

3.3 Command-Line Tools

3.3.1 suews-run

Run SUEWS simulation using settings in `PATH_RUNCONTROL` (default: “./RunControl.nml”, i.e., the RunControl namelist file in the current directory).

```
suews-run [OPTIONS] [PATH_RUNCONTROL]
```

Arguments

`PATH_RUNCONTROL`

Optional argument

3.3.2 suews-convert

Convert SUEWS input tables from older versions to newer ones (one-way only).

```
suews-convert [OPTIONS]
```

Options

-f, --from <fromVer>

Version to convert from [required]

Options 2019a|2018c|2018b|2018a|2017a|2016a

-t, --to <toVer>

Version to convert to [required]

Options 2019b|2019a|2018c|2018b|2018a|2017a

-i, --input <fromDir>

Original directory to convert [required]

-o, --output <toDir>

New directory to create for converted tables [required]

3.4 Key Data Structures

3.4.1 `df_state` variables

Note: Data structure of `df_state` is explained [here](#).

`aerodynamicresistancemethod`

Description Internal use. Please DO NOT modify

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables None

`ah_min`

Description Minimum QF values.

Dimensionality (2,)

Dimensionality Remarks 2: {Weekday, Weekend}

SUEWS-related variables `AHMin_WD`, `AHMin_WE`

`ah_slope_cooling`

Description Cooling slope of QF calculation.

Dimensionality (2,)

Dimensionality Remarks 2: {Weekday, Weekend}

SUEWS-related variables `AHSlope_Cooling_WD`, `AHSlope_Cooling_WE`

ah_slope_heating

Description Heating slope of QF calculation.

Dimensionality (2,)

Dimensionality Remarks 2: {Weekday, Weekend}

SUEWS-related variables `AHSlope_Heating_WD`, `AHSlope_Heating_WE`

ahprof_24hr

Description Hourly profile values used in energy use calculation.

Dimensionality (24, 2)

Dimensionality Remarks 24: hours of a day

2: {Weekday, Weekend}

SUEWS-related variables `EnergyUseProfWD`, `EnergyUseProfWE`

alb

Description Effective surface albedo (middle of the day value) for summertime.

Dimensionality (7,)

Dimensionality Remarks 7: { `Paved`, `Bldgs`, `EveTr`, `DecTr`, `Grass`, `BSoil`, `Water` }

SUEWS-related variables `AlbedoMax`

albdectr_id

Description Albedo of deciduous surface `DecTr` on day 0 of run

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `albDecTr0`

albevetr_id

Description Albedo of evergreen surface `EveTr` on day 0 of run

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `albEveTr0`

albgrass_id

Description Albedo of grass surface `Grass` on day 0 of run

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `albGrass0`

albmax_dectr

Description Effective surface albedo (middle of the day value) for summertime.

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables [AlbedoMax](#)

albmax_evetr

Description Effective surface albedo (middle of the day value) for summertime.

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables [AlbedoMax](#)

albmax_grass

Description Effective surface albedo (middle of the day value) for summertime.

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables [AlbedoMax](#)

albmin_dectr

Description Effective surface albedo (middle of the day value) for wintertime (not including snow).

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables [AlbedoMin](#)

albmin_evetr

Description Effective surface albedo (middle of the day value) for wintertime (not including snow).

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables [AlbedoMin](#)

albmin_grass

Description Effective surface albedo (middle of the day value) for wintertime (not including snow).

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables [AlbedoMin](#)

alpha_bioco2

Description The mean apparent ecosystem quantum. Represents the initial slope of the light-response curve.

Dimensionality (3,)

Dimensionality Remarks 3: { [EveTr](#), [DecTr](#), [Grass](#) }

SUEWS-related variables [alpha](#)

alpha_enh_bioco2

Description Part of the `alpha` coefficient related to the fraction of vegetation.

Dimensionality (3,)

Dimensionality Remarks 3: { `EveTr`, `DecTr`, `Grass` }

SUEWS-related variables `alpha_enh`

`alt`

Description Altitude of grids [m].

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `Alt`

`baset`

Description Base Temperature for initiating growing degree days (GDD) for leaf growth. [°C]

Dimensionality (3,)

Dimensionality Remarks 3: { `EveTr`, `DecTr`, `Grass` }

SUEWS-related variables `BaseT`

`basete`

Description Base temperature for initiating senescence degree days (SDD) for leaf off. [°C]

Dimensionality (3,)

Dimensionality Remarks 3: { `EveTr`, `DecTr`, `Grass` }

SUEWS-related variables `BaseTe`

`basethdd`

Description Base temperature for heating degree days [°C]

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `BaseTHDD`

`beta_bioco2`

Description The light-saturated gross photosynthesis of the canopy. [$\mu\text{mol m}^{-2} \text{s}^{-1}$]

Dimensionality (3,)

Dimensionality Remarks 3: { `EveTr`, `DecTr`, `Grass` }

SUEWS-related variables `beta`

`beta_enh_bioco2`

Description Part of the `beta` coefficient related to the fraction of vegetation.

Dimensionality (3,)

Dimensionality Remarks 3: { `EveTr`, `DecTr`, `Grass` }

SUEWS-related variables `beta_enh`

`bldgh`

Description Mean building height [m]

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `H_Bldgs`

capmax_dec

Description Maximum water storage capacity for upper surfaces (i.e. canopy)

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `StorageMax`

capmin_dec

Description Minimum water storage capacity for upper surfaces (i.e. canopy).

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `StorageMin`

chanohm

Description Bulk transfer coefficient for this surface to use in AnOHM [-]

Dimensionality (7,)

Dimensionality Remarks 7: { `Paved`, `Bldgs`, `EveTr`, `DecTr`, `Grass`, `BSoil`, `Water` }

SUEWS-related variables `AnOHM_Ch`

co2pointsource

Description CO2 emission factor [kg km^{-1}]

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `CO2PointSource`

cpanohm

Description Volumetric heat capacity for this surface to use in AnOHM [J m^{-3}]

Dimensionality (7,)

Dimensionality Remarks 7: { `Paved`, `Bldgs`, `EveTr`, `DecTr`, `Grass`, `BSoil`, `Water` }

SUEWS-related variables `AnOHM_Cp`

crwmax

Description Maximum water holding capacity of snow [mm]

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `CRWMax`

crwmin

Description Minimum water holding capacity of snow [mm]

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `CRWMin`

daywat

Description Irrigation flag: 1 for on and 0 for off.

Dimensionality (7,)

Dimensionality Remarks 7: {Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday}

SUEWS-related variables `DayWat (1)`, `DayWat (2)`, `DayWat (3)`, `DayWat (4)`,
`DayWat (5)`, `DayWat (6)`, `DayWat (7)`

daywatper

Description Fraction of properties using irrigation for each day of a week.

Dimensionality (7,)

Dimensionality Remarks 7: {Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday}

SUEWS-related variables `DayWatPer (1)`, `DayWatPer (2)`, `DayWatPer (3)`,
`DayWatPer (4)`, `DayWatPer (5)`, `DayWatPer (6)`, `DayWatPer (7)`

decidcap_id

Description Storage capacity of deciduous surface `DecTr` on day 0 of run.

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `decidCap0`

dectreeh

Description Mean height of deciduous trees [m]

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `H_DecTr`

diagnose

Description Internal use. Please DO NOT modify

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables None

diagqn

Description Internal use. Please DO NOT modify

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables None

diagqs

Description Internal use. Please DO NOT modify

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables None

drainrt

Description Drainage rate of bucket for LUMPS [mm h⁻¹]

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `LUMPS_DrRate`

ef_umolco2perj

Description Emission factor for fuels used for building heating.

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `EF_umolCO2perJ`

emis

Description Effective surface emissivity.

Dimensionality (7,)

Dimensionality Remarks 7: { `Paved`, `Bldgs`, `EveTr`, `DecTr`, `Grass`, `BSoil`, `Water`}

SUEWS-related variables `Emissivity`

emissionsmethod

Description Determines method for QF calculation.

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `EmissionsMethod`

enddls

Description End of the day light savings [DOY]

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `EndDLS`

enef_v_jkm

Description Emission factor for heat [J klm⁻¹].

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `EnEF_v_Jkm`

evapmethod

Description Internal use. Please DO NOT modify

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables None

evetreeh

Description Mean height of evergreen trees [m]

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `H_EveTr`

faibldg

Description Frontal area index for buildings [-]

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `FAI_Bldgs`

faidectree

Description Frontal area index for deciduous trees [-]

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `FAI_DecTr`

faievetree

Description Frontal area index for evergreen trees [-]

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `FAI_EveTr`

faut

Description Fraction of irrigated area that is irrigated using automated systems

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `Faut`

fcef_v_kgkm

Description CO2 emission factor for weekdays [kg km^{-1}];;CO2 emission factor for weekends [kg km^{-1}]

Dimensionality (2,)

Dimensionality Remarks 2: {Weekday, Weekend}

SUEWS-related variables `FcEF_v_kgkmWD`, `FcEF_v_kgkmWE`

flowchange

Description Difference in input and output flows for water surface [mm h^{-1}]

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `FlowChange`

`frfossilfuel_heat`

Description Fraction of fossil fuels used for building heating [-]

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `FrFossilFuel_Heat`

`frfossilfuel_nonheat`

Description Fraction of fossil fuels used for building energy use [-]

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `FrFossilFuel_NonHeat`

`g1`

Description Related to maximum surface conductance [mm s^{-1}]

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `G1`

`g2`

Description Related to Kdown dependence [W m^{-2}]

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `G2`

`g3`

Description Related to VPD dependence [units depend on `gsModel`]

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `G3`

`g4`

Description Related to VPD dependence [units depend on `gsModel`]

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `G4`

`g5`

Description Related to temperature dependence [$^{\circ}\text{C}$]

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `G5`

g6

Description Related to soil moisture dependence [mm⁻¹]

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `G6`

gddfull

Description The growing degree days (GDD) needed for full capacity of the leaf area index (LAI) [°C].

Dimensionality (3,)

Dimensionality Remarks 3: { `EveTr`, `DecTr`, `Grass` }

SUEWS-related variables `GDDFull`

gsmodel

Description Formulation choice for conductance calculation.

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `gsModel`

humactivity_24hr

Description Hourly profile values used in human activity calculation.

Dimensionality (24, 2)

Dimensionality Remarks 24: hours of a day

2: {Weekday, Weekend}

SUEWS-related variables `ActivityProfWD`, `ActivityProfWE`

ie_a

Description Coefficient for automatic irrigation model.

Dimensionality (3,)

Dimensionality Remarks 3: { `EveTr`, `DecTr`, `Grass` }

SUEWS-related variables `Ie_a1`, `Ie_a2`, `Ie_a3`

ie_end

Description Day when irrigation ends [DOY]

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `Ie_end`

ie_m

Description Coefficient for manual irrigation model.

Dimensionality (3,)

Dimensionality Remarks 3: { `EveTr`, `DecTr`, `Grass` }

SUEWS-related variables `Ie_m1, Ie_m2, Ie_m3`

ie_start

Description Day when irrigation starts [DOY]

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `Ie_start`

internalwateruse_h

Description Internal water use [mm h⁻¹]

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `InternalWaterUse`

irrfracconif

Description Fraction of evergreen trees that are irrigated [-]

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `IrrFr_EveTr`

irrfracdecid

Description Fraction of deciduous trees that are irrigated [-]

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `IrrFr_DecTr`

irrfracgrass

Description Fraction of Grass that is irrigated [-]

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `IrrFr_Grass`

kkanohm

Description Thermal conductivity for this surface to use in AnOHM [W m K⁻¹]

Dimensionality (7,)

Dimensionality Remarks 7: { `Paved`, `Bldgs`, `EveTr`, `DecTr`, `Grass`, `BSoil`, `Water` }

SUEWS-related variables `AnOHM_Kk`

kmax

Description Maximum incoming shortwave radiation [W m⁻²]

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `Kmax`

lai_id

Description Initial LAI values.

Dimensionality (3,)

Dimensionality Remarks 3: { EveTr, DecTr, Grass}

SUEWS-related variables LAIinitialDecTr, LAIinitialEveTr, LAIinitialGrass

laicalcyes

Description Internal use. Please DO NOT modify

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables None

laimax

Description full leaf-on summertime value

Dimensionality (3,)

Dimensionality Remarks 3: { EveTr, DecTr, Grass}

SUEWS-related variables LAIMax

laimin

Description leaf-off wintertime value

Dimensionality (3,)

Dimensionality Remarks 3: { EveTr, DecTr, Grass}

SUEWS-related variables LAIMin

laipower

Description parameters required by LAI calculation.

Dimensionality (4, 3)

Dimensionality Remarks 4: {LeafGrowthPower1, LeafGrowthPower2,
LeafOffPower1, LeafOffPower2}

3: { EveTr, DecTr, Grass}

SUEWS-related variables LeafGrowthPower1, LeafGrowthPower2, LeafOffPower1,
LeafOffPower2

laitype

Description LAI calculation choice.

Dimensionality (3,)

Dimensionality Remarks 3: { EveTr, DecTr, Grass}

SUEWS-related variables LAIEq

lat

Description Latitude [deg].

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `lat`

lng

Description longitude [deg]

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `lng`

maxconductance

Description The maximum conductance of each vegetation or surface type. [mm s^{-1}]

Dimensionality (3,)

Dimensionality Remarks 3: { `EveTr`, `DecTr`, `Grass` }

SUEWS-related variables `MaxConductance`

maxfcmetab

Description Maximum (day) CO₂ from human metabolism. [W m^{-2}]

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `MaxFCMetab`

maxqfmetab

Description Maximum value for human heat emission. [W m^{-2}]

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `MaxQFMetab`

min_res_bioco2

Description Minimum soil respiration rate (for cold-temperature limit) [$\text{umol m}^{-2} \text{s}^{-1}$].

Dimensionality (3,)

Dimensionality Remarks 3: { `EveTr`, `DecTr`, `Grass` }

SUEWS-related variables `min_respi`

minfcmetab

Description Minimum (night) CO₂ from human metabolism. [W m^{-2}]

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `MinFCMetab`

minqfmetab

Description Minimum value for human heat emission. [W m^{-2}]

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `MinQFMetab`

narp_emis_snow

Description Effective surface emissivity.

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `Emissivity`

narp_trans_site

Description Atmospheric transmissivity for NARP [-]

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `NARP_Trans`

netradiationmethod

Description Determines method for calculation of radiation fluxes.

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `NetRadiationMethod`

ohm_coef

Description Coefficients for OHM calculation.

Dimensionality (8, 4, 3)

Dimensionality Remarks 8: { `Paved`, `Bldgs`, `EveTr`, `DecTr`, `Grass`, `BSoil`, `Water`, one extra land cover type (currently NOT used)}

4: {`SummerWet`, `SummerDry`, `WinterWet`, `WinterDry`}

3: {`a1`, `a2`, `a3`}

SUEWS-related variables `a1`, `a2`, `a3`

ohm_threshsw

Description Temperature threshold determining whether summer/winter OHM coefficients are applied [°C]

Dimensionality (8,)

Dimensionality Remarks 8: { `Paved`, `Bldgs`, `EveTr`, `DecTr`, `Grass`, `BSoil`, `Water`, one extra land cover type (currently NOT used)}

SUEWS-related variables `OHMThresh_SW`

ohm_threshwd

Description Soil moisture threshold determining whether wet/dry OHM coefficients are applied [-]

Dimensionality (8,)

Dimensionality Remarks 8: { `Paved`, `Bldgs`, `EveTr`, `DecTr`, `Grass`, `BSoil`, `Water`, one extra land cover type (currently NOT used)}

SUEWS-related variables `OHMThresh_WD`

ohmincqh

Description Determines whether the storage heat flux calculation uses Q^* or $(Q^* + QF)$.

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `OHMIncQF`

pipecapacity

Description Storage capacity of pipes [mm]

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `PipeCapacity`

popdensdaytime

Description Daytime population density (i.e. workers, tourists) [people ha⁻¹]

Dimensionality (2,)

Dimensionality Remarks 2: {Weekday, Weekend}

SUEWS-related variables `PopDensDay`

popdensnighttime

Description Night-time population density (i.e. residents) [people ha⁻¹]

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `PopDensNight`

popprof_24hr

Description Hourly profile values used in dynamic population estimation.

Dimensionality (24, 2)

Dimensionality Remarks 24: hours of a day

2: {Weekday, Weekend}

SUEWS-related variables `PopProfWD`, `PopProfWE`

pormax_dec

Description full leaf-on summertime value Used only for `DecTr` (can affect roughness calculation)

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `PorosityMax`

pormin_dec

Description leaf-off wintertime value Used only for `DecTr` (can affect roughness calculation)

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `PorosityMin`

porosity_id

Description Porosity of deciduous vegetation on day 0 of run.

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `porosity0`

preciplimit

Description Temperature limit when precipitation falls as snow [°C]

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `PrecipLimSnow`

preciplimitalb

Description Limit for hourly precipitation when the ground is fully covered with snow [mm]

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `PrecipLimAlb`

qf0_beu

Description Building energy use [W m^{-2}]

Dimensionality (2,)

Dimensionality Remarks 2: {Weekday, Weekend}

SUEWS-related variables `QF0_BEU_WD`, `QF0_BEU_WE`

qf_a

Description Base value for QF calculation.

Dimensionality (2,)

Dimensionality Remarks 2: {Weekday, Weekend}

SUEWS-related variables `QF_A_WD`, `QF_A_WE`

qf_b

Description Parameter related to heating degree days.

Dimensionality (2,)

Dimensionality Remarks 2: {Weekday, Weekend}

SUEWS-related variables `QF_B_WD`, `QF_B_WE`

qf_c

Description Parameter related to heating degree days.

Dimensionality (2,)

Dimensionality Remarks 2: {Weekday, Weekend}

SUEWS-related variables `QF_C_WD`, `QF_C_WE`

radmeltfact

Description Hourly radiation melt factor of snow [$\text{mm W}^{-1} \text{h}^{-1}$]

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `RadMeltFactor`

raincover

Description Limit when surface totally covered with water for LUMPS [mm]

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `LUMPS_Cover`

rainmaxres

Description Maximum water bucket reservoir [mm] Used for LUMPS surface wetness control.

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `LUMPS_MaxRes`

resp_a

Description Respiration coefficient a.

Dimensionality (3,)

Dimensionality Remarks 3: { `EveTr`, `DecTr`, `Grass` }

SUEWS-related variables `resp_a`

resp_b

Description Respiration coefficient b - related to air temperature dependency.

Dimensionality (3,)

Dimensionality Remarks 3: { `EveTr`, `DecTr`, `Grass` }

SUEWS-related variables `resp_b`

roughlenheatmethod

Description Determines method for calculating roughness length for heat.

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `RoughLenHeatMethod`

roughlenmommethode

Description Determines how aerodynamic roughness length (`z0m`) and zero displacement height (`zdm`) are calculated.

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `RoughLenMomMethod`

runofftewater

Description Fraction of above-ground runoff flowing to water surface during flooding [-]

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `RunoffToWater`

s1

Description A parameter related to soil moisture dependence [-]

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `s1`

s2

Description A parameter related to soil moisture dependence [mm]

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `s2`

sathydraulicconduct

Description Hydraulic conductivity for saturated soil [mm s⁻¹]

Dimensionality (7,)

Dimensionality Remarks 7: { `Paved`, `Bldgs`, `EveTr`, `DecTr`, `Grass`, `BSoil`, `Water` }

SUEWS-related variables `SatHydraulicCond`

sddfll

Description The sensesence degree days (SDD) needed to initiate leaf off. [°C]

Dimensionality (3,)

Dimensionality Remarks 3: { `EveTr`, `DecTr`, `Grass` }

SUEWS-related variables `SDDFull`

sfr

Description Surface cover fractions.

Dimensionality (7,)

Dimensionality Remarks 7: { `Paved`, `Bldgs`, `EveTr`, `DecTr`, `Grass`, `BSoil`, `Water` }

SUEWS-related variables `Fr_Bldgs`, `Fr_Bsoil`, `Fr_DecTr`, `Fr_EveTr`, `Fr_Grass`,
`Fr_Paved`, `Fr_Water`

smdmethod

Description Determines method for calculating soil moisture deficit (SMD).

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `SMDMethod`

snowalb

Description Initial snow albedo

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `SnowAlb0`

snowalbmax

Description Effective surface albedo (middle of the day value) for summertime.

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `AlbedoMax`

snowalbmin

Description Effective surface albedo (middle of the day value) for wintertime (not including snow).

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `AlbedoMin`

snowdens

Description Initial snow density of each land cover.

Dimensionality (7,)

Dimensionality Remarks 7: { `Paved`, `Bldgs`, `EveTr`, `DecTr`, `Grass`, `BSoil`, `Water` }

SUEWS-related variables `SnowDensBldgs`, `SnowDensPaved`, `SnowDensDecTr`,
`SnowDensEveTr`, `SnowDensGrass`, `SnowDensBSoil`, `SnowDensWater`

snowdensmax

Description Maximum snow density [kg m^{-3}]

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `SnowDensMax`

snowdensmin

Description Fresh snow density [kg m^{-3}]

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `SnowDensMin`

snowfrac

Description Initial plan area fraction of snow on each land cover

Dimensionality (7,)

Dimensionality Remarks 7: { `Paved`, `Bldgs`, `EveTr`, `DecTr`, `Grass`, `BSoil`, `Water` }

SUEWS-related variables `SnowFracBldgs`, `SnowFracPaved`, `SnowFracDecTr`,
`SnowFracEveTr`, `SnowFracGrass`, `SnowFracBSoil`, `SnowFracWater`

snowlimbldg

Description Limit of the snow water equivalent for snow removal from roads and roofs [mm]

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `SnowLimRemove`

snowlimpaved

Description Limit of the snow water equivalent for snow removal from roads and roofs [mm]

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `SnowLimRemove`

snowpack

Description Initial snow water equivalent on each land cover

Dimensionality (7,)

Dimensionality Remarks 7: { `Paved`, `Bldgs`, `EveTr`, `DecTr`, `Grass`, `BSoil`, `Water` }

SUEWS-related variables `SnowPackBldgs`, `SnowPackPaved`, `SnowPackDecTr`,
`SnowPackEveTr`, `SnowPackGrass`, `SnowPackBSoil`, `SnowPackWater`

snowpacklimit

Description Limit for the snow water equivalent when snow cover starts to be patchy [mm]

Dimensionality (7,)

Dimensionality Remarks 7: { `Paved`, `Bldgs`, `EveTr`, `DecTr`, `Grass`, `BSoil`, `Water` }

SUEWS-related variables `SnowLimPatch`

snowprof_24hr

Description Hourly profile values used in snow clearing.

Dimensionality (24, 2)

Dimensionality Remarks 24: hours of a day

2: {Weekday, Weekend}

SUEWS-related variables `SnowClearingProfWD`, `SnowClearingProfWE`

snowuse

Description Determines whether the snow part of the model runs.

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `SnowUse`

snowwater

Description Initial amount of liquid water in the snow on each land cover

Dimensionality (7,)

Dimensionality Remarks 7: { `Paved`, `Bldgs`, `EveTr`, `DecTr`, `Grass`, `BSoil`, `Water` }

SUEWS-related variables `SnowWaterBldgsState`, `SnowWaterPavedState`,
`SnowWaterDecTrState`, `SnowWaterEveTrState`, `SnowWaterGrassState`,
`SnowWaterBSoilState`, `SnowWaterWaterState`

soildepth

Description Depth of soil beneath the surface [mm]

Dimensionality (7,)

Dimensionality Remarks 7: { Paved, Bldgs, EveTr, DecTr, Grass, BSoil, Water}

SUEWS-related variables SoilDepth

soilstore_id

Description Initial water stored in soil beneath each land cover

Dimensionality (7,)

Dimensionality Remarks 7: { Paved, Bldgs, EveTr, DecTr, Grass, BSoil, Water}

SUEWS-related variables SoilstoreBldgsState, SoilstorePavedState,
SoilstoreDecTrState, SoilstoreEveTrState, SoilstoreGrassState,
SoilstoreBSoilState

soilstorecap

Description Limit value for SoilDepth [mm]

Dimensionality (7,)

Dimensionality Remarks 7: { Paved, Bldgs, EveTr, DecTr, Grass, BSoil, Water}

SUEWS-related variables SoilStoreCap

stabilitymethod

Description Defines which atmospheric stability functions are used.

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables StabilityMethod

startdls

Description Start of the day light savings [DOY]

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables StartDLS

state_id

Description Initial wetness condition on each land cover

Dimensionality (7,)

Dimensionality Remarks 7: { Paved, Bldgs, EveTr, DecTr, Grass, BSoil, Water}

SUEWS-related variables BldgsState, PavedState, DecTrState, EveTrState,
GrassState, BSoilState, WaterState

statelimit

Description Upper limit to the surface state. [mm]

Dimensionality (7,)

Dimensionality Remarks 7: { Paved, Bldgs, EveTr, DecTr, Grass, BSoil, Water}

SUEWS-related variables `StateLimit`

`storageheatmethod`

Description Determines method for calculating storage heat flux ΔQ_S .

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `StorageHeatMethod`

`storedrainprm`

Description Coefficients used in drainage calculation.

Dimensionality (6, 7)

Dimensionality Remarks 6: { `StorageMin`, `DrainageEq`, `DrainageCoef1`,
`DrainageCoef2`, `StorageMax`, current storage }

7: { `Paved`, `Bldgs`, `EveTr`, `DecTr`, `Grass`, `BSoil`, `Water` }

SUEWS-related variables `DrainageCoef1`, `DrainageCoef2`, `DrainageEq`,
`StorageMax`, `StorageMin`

`surfacearea`

Description Area of the grid [ha].

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `SurfaceArea`

`t_critic_cooling`

Description Critical cooling temperature.

Dimensionality (2,)

Dimensionality Remarks 2: {Weekday, Weekend}

SUEWS-related variables `TCritic_Cooling_WD`, `TCritic_Cooling_WE`

`t_critic_heating`

Description Critical heating temperature.

Dimensionality (2,)

Dimensionality Remarks 2: {Weekday, Weekend}

SUEWS-related variables `TCritic_Heating_WD`, `TCritic_Heating_WE`

`tau_a`

Description Time constant for snow albedo aging in cold snow [-]

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `tau_a`

`tau_f`

Description Time constant for snow albedo aging in melting snow [-]

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `tau_f`

tau_r

Description Time constant for snow density ageing [-]

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `tau_r`

tempmeltfact

Description Hourly temperature melt factor of snow [mm K⁻¹ h⁻¹]

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `TempMeltFactor`

th

Description Upper air temperature limit [°C]

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `TH`

theta_bioco2

Description The convexity of the curve at light saturation.

Dimensionality (3,)

Dimensionality Remarks 3: { `EveTr`, `DecTr`, `Grass` }

SUEWS-related variables `theta`

timezone

Description Time zone [h] for site relative to UTC (east is positive). This should be set according to the times given in the meteorological forcing file(s).

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `Timezone`

t1

Description Lower air temperature limit [°C]

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `TL`

trafficrate

Description Traffic rate used for CO2 flux calculation.

Dimensionality (2,)

Dimensionality Remarks 2: {Weekday, Weekend}

SUEWS-related variables `TrafficRate_WD`, `TrafficRate_WE`

trafficunits

Description Units for the traffic rate for the study area. Not used in v2018a.

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `TrafficUnits`

traffprof_24hr

Description Hourly profile values used in traffic activity calculation.

Dimensionality (24, 2)

Dimensionality Remarks 24: hours of a day

2: {Weekday, Weekend}

SUEWS-related variables `TraffProfWD`, `TraffProfWE`

tstep

Description Specifies the model time step [s].

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `Tstep`

veg_type

Description Internal use. Please DO NOT modify

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables None

waterdist

Description Fraction of water redistribution

Dimensionality (8, 6)

Dimensionality Remarks 8: { `Paved`, `Bldgs`, `EveTr`, `DecTr`, `Grass`, `BSoil`, `Water`, one extra land cover type (currently NOT used)}

6: { `Paved`, `Bldgs`, `EveTr`, `DecTr`, `Grass`, `BSoil`}

SUEWS-related variables `ToBSoil`, `ToBldgs`, `ToDecTr`, `ToEveTr`, `ToGrass`, `ToPaved`, `ToRunoff`, `ToSoilStore`, `ToWater`

waterusemethod

Description Defines how external water use is calculated.

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `WaterUseMethod`

wetthresh

Description Depth of water which determines whether evaporation occurs from a partially wet or completely wet surface [mm].

Dimensionality (7,)

Dimensionality Remarks 7: { Paved, Bldgs, EveTr, DecTr, Grass, BSoil, Water}

SUEWS-related variables `WetThreshold`

wuprofa_24hr

Description Hourly profile values used in automatic irrigation.

Dimensionality (24, 2)

Dimensionality Remarks 24: hours of a day

2: {Weekday, Weekend}

SUEWS-related variables `WaterUseProfAutoWD`, `WaterUseProfAutoWE`

wuprofm_24hr

Description Hourly profile values used in manual irrigation.

Dimensionality (24, 2)

Dimensionality Remarks 24: hours of a day

2: {Weekday, Weekend}

SUEWS-related variables `WaterUseProfManuWD`, `WaterUseProfManuWE`

z

Description Measurement height [m].

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `z`

z0m_in

Description Roughness length for momentum [m]

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `z0`

zdm_in

Description Zero-plane displacement [m]

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `zd`

3.4.2 df_forcing variables

Note: Data structure of `df_forcing` is explained [here](#).

RH**Description** Relative Humidity [%]**Tair****Description** Air temperature [°C]**U****Description** Wind speed [m s⁻¹] Height of the wind speed measurement (z) is needed in `SUEWS_SiteSelect.txt`.**Wuh****Description** External water use [m³]**fcld****Description** Cloud fraction [tenths]**id****Description** Day of year [DOY]**imin****Description** Minute [M]**isec****Description** Second [S]**it****Description** Hour [H]**iy****Description** Year [YYYY]**kdiff****Description** Diffuse radiation [W m⁻²] **Recommended in this version.** if `SOLWEIGUse = 1`**kdir****Description** Direct radiation [W m⁻²] **Recommended in this version.** if `SOLWEIGUse = 1`**kdown****Description** Incoming shortwave radiation [W m⁻²] Must be > 0 W m⁻².**lai****Description** Observed leaf area index [m⁻² m⁻²]**ldown****Description** Incoming longwave radiation [W m⁻²]**pres****Description** Barometric pressure [kPa]**qe****Description** Latent heat flux [W m⁻²]**qf**

	Description Anthropogenic heat flux [W m^{-2}]
qh	
	Description Sensible heat flux [W m^{-2}]
qn	
	Description Net all-wave radiation [W m^{-2}] Required if <code>NetRadiationMethod = 0</code> .
qs	
	Description Storage heat flux [W m^{-2}]
rain	
	Description Rainfall [mm]
snow	
	Description Snow cover fraction (0 – 1) [-] Required if <code>SnowUse = 1</code>
wdir	
	Description Wind direction [$^{\circ}$] Not available in this version.
xsmc	
	Description Observed soil moisture [$\text{m}^3 \text{ m}^{-3}$] or [kg kg^{-1}]

3.4.3 df_output variables

Note: Data structure of `df_output` is explained [here](#).

AddWater

Description Additional water flow received from other grids [mm]

Group SUEWS

AlbBulk

Description Bulk albedo [-]

Group SUEWS

AlbDecTr

Description Albedo of deciduous trees [-]

Group DailyState

AlbEveTr

Description Albedo of evergreen trees [-]

Group DailyState

AlbGrass

Description Albedo of grass [-]

Group DailyState

AlbSnow

Description Snow albedo [-]

Group SUEWS

AlbSnow

Description Snow albedo [-]

Group DailyState

Azimuth

Description Solar azimuth angle [°]

Group SUEWS

CI

Description clearness index for Ldown (Lindberg et al. 2008)

Group SOLWEIG

DLHrs

Description Day length [h]

Group DailyState

DaysSR

Description Days since rain [days]

Group DailyState

DecidCap

Description Moisture storage capacity of deciduous trees [mm]

Group DailyState

DensSnow_BSoil

Description Snow density – bare soil surface [kg m⁻³]

Group DailyState

DensSnow_BSoil

Description Snow density - bare soil surface [kg m⁻³]

Group snow

DensSnow_BSoil

Description Snow density - bare soil surface [kg m⁻³]

Group DailyState

DensSnow_BSoil

Description Snow density – bare soil surface [kg m⁻³]

Group snow

DensSnow_Bldgs

Description Snow density - building surface [kg m⁻³]

Group DailyState

DensSnow_Bldgs

Description Snow density - building surface [kg m⁻³]

Group snow

DensSnow_Bldgs

Description Snow density – building surface [kg m⁻³]

Group snow

DensSnow_Bldgs

Description Snow density – building surface [kg m⁻³]

Group DailyState

DensSnow_DecTr

Description Snow density – deciduous surface [kg m⁻³]

Group DailyState

DensSnow_DecTr

Description Snow density - deciduous surface [kg m⁻³]

Group DailyState

DensSnow_DecTr

Description Snow density – deciduous surface [kg m⁻³]

Group snow

DensSnow_DecTr

Description Snow density - deciduous surface [kg m⁻³]

Group snow

DensSnow_EveTr

Description Snow density - evergreen surface [kg m⁻³]

Group snow

DensSnow_EveTr

Description Snow density – evergreen surface [kg m⁻³]

Group snow

DensSnow_EveTr

Description Snow density - evergreen surface [kg m⁻³]

Group DailyState

DensSnow_EveTr

Description Snow density – evergreen surface [kg m⁻³]

Group DailyState

DensSnow_Grass

Description Snow density - grass surface [kg m⁻³]

Group snow

DensSnow_Grass

Description Snow density – grass surface [kg m⁻³]

Group DailyState

DensSnow_Grass

Description Snow density – grass surface [kg m⁻³]

Group snow

DensSnow_Grass

Description Snow density - grass surface [kg m⁻³]

Group DailyState

DensSnow_Paved

Description Snow density - paved surface [kg m⁻³]

Group snow

DensSnow_Paved

Description Snow density – paved surface [kg m⁻³]

Group snow

DensSnow_Paved

Description Snow density - paved surface [kg m⁻³]

Group DailyState

DensSnow_Paved

Description Snow density – paved surface [kg m⁻³]

Group DailyState

DensSnow_Water

Description Snow density – water surface [kg m⁻³]

Group DailyState

DensSnow_Water

Description Snow density - water surface [kg m⁻³]

Group DailyState

DensSnow_Water

Description Snow density – water surface [kg m⁻³]

Group snow

DensSnow_Water

Description Snow density - water surface [kg m⁻³]

Group snow

DiffuseRad

Description Diffuse shortwave radiation

Group SOLWEIG

DirectRad

Description Direct shortwave radiation

Group SOLWEIG

Drainage

Description Drainage [mm]

Group SUEWS

Evap

Description Evaporation [mm]

Group SUEWS

Fc

Description CO₂ flux [$\mu\text{mol m}^{-2} \text{s}^{-1}$]

Group SUEWS

FcBuild

Description CO₂ flux from buildings [$\mu\text{mol m}^{-2} \text{s}^{-1}$]

Group SUEWS

FcMetab

Description CO₂ flux from metabolism [$\mu\text{mol m}^{-2} \text{s}^{-1}$]

Group SUEWS

FcPhoto

Description CO₂ flux from photosynthesis [$\mu\text{mol m}^{-2} \text{s}^{-1}$]

Group SUEWS

FcPoint

Description CO₂ flux from point source [$\mu\text{mol m}^{-2} \text{s}^{-1}$]

Group SUEWS

FcRespi

Description CO₂ flux from respiration [$\mu\text{mol m}^{-2} \text{s}^{-1}$]

Group SUEWS

FcTraff

Description CO₂ flux from traffic [$\mu\text{mol m}^{-2} \text{s}^{-1}$]

Group SUEWS

Fcld

Description Cloud fraction [-]

Group SUEWS

FlowCh

Description Additional flow into water body [mm]

Group SUEWS

GDD_DecTr

Description Growing degree days for deciduous tree [$^{\circ}\text{C d}$]

Group DailyState

GDD_EveTr

Description Growing degree days for evergreen eree [°C d]

Group DailyState

GDD_Grass

Description Growing degree days for grass [°C d]

Group DailyState

GlobalRad

Description Input Kdn

Group SOLWEIG

HDD1_h

Description Heating degree days [°C d]

Group DailyState

HDD2_c

Description Cooling degree days [°C d]

Group DailyState

HDD3_Tmean

Description Average daily air temperature [°C]

Group DailyState

HDD4_T5d

Description 5-day running-mean air temperature [°C]

Group DailyState

I0

Description theoretical value of maximum incoming solar radiation

Group SOLWEIG

Irr

Description Irrigation [mm]

Group SUEWS

Kdown

Description Incoming shortwave radiation [W m^{-2}]

Group SUEWS

Kdown2d

Description Incoming shortwave radiation at POI

Group SOLWEIG

Keast

Description Shortwave radiation from east at POI

	Group SOLWEIG
Knorth	
	Description Shortwave radiation from north at POI
	Group SOLWEIG
Ksouth	
	Description Shortwave radiation from south at POI
	Group SOLWEIG
Kup	
	Description Outgoing shortwave radiation [W m^{-2}]
	Group SUEWS
Kup2d	
	Description Outgoing shortwave radiation at POI
	Group SOLWEIG
Kwest	
	Description Shortwave radiation from west at POI
	Group SOLWEIG
LAI	
	Description Leaf area index [$\text{m}^2 \text{m}^{-2}$]
	Group SUEWS
LAI_DecTr	
	Description Leaf area index of deciduous trees [$\text{m}^2 \text{m}^{-2}$]
	Group DailyState
LAI_EveTr	
	Description Leaf area index of evergreen trees [$\text{m}^2 \text{m}^{-2}$]
	Group DailyState
LAI_Grass	
	Description Leaf area index of grass [$\text{m}^2 \text{m}^{-2}$]
	Group DailyState
LAIlumps	
	Description Leaf area index used in LUMPS (normalised 0-1) [-]
	Group DailyState
Ldown	
	Description Incoming longwave radiation [W m^{-2}]
	Group SUEWS
Ldown2d	
	Description Incoming longwave radiation at POI

Least
Group SOLWEIG
Description Longwave radiation from east at POI
Group SOLWEIG

Lnorth
Description Longwave radiation from north at POI
Group SOLWEIG

Lob
Description Obukhov length [m]
Group SUEWS

Lsouth
Description Longwave radiation from south at POI
Group SOLWEIG

Lup
Description Outgoing longwave radiation [W m^{-2}]
Group SUEWS

Lup2d
Description Outgoing longwave radiation at POI
Group SOLWEIG

Lwest
Description Longwave radiation from west at POI
Group SOLWEIG

MeltWStore
Description Meltwater store [mm]
Group SUEWS

MeltWater
Description Meltwater [mm]
Group SUEWS

MwStore_BSoil
Description Melt water store – bare soil surface [mm]
Group snow

MwStore_Bldgs
Description Melt water store – building surface [mm]
Group snow

MwStore_DecTr
Description Melt water store – deciduous surface [mm]

Group snow

MwStore_EveTr

Description Melt water store – evergreen surface [mm]

Group snow

MwStore_Grass

Description Melt water store – grass surface [mm]

Group snow

MwStore_Paved

Description Melt water store – paved surface [mm]

Group snow

MwStore_Water

Description Melt water store – water surface [mm]

Group snow

Mw_BSoil

Description Meltwater – bare soil surface [mm h⁻¹]

Group snow

Mw_Bldgs

Description Meltwater – building surface [mm h⁻¹]

Group snow

Mw_DecTr

Description Meltwater – deciduous surface [mm h⁻¹]

Group snow

Mw_EveTr

Description Meltwater – evergreen surface [mm h⁻¹]

Group snow

Mw_Grass

Description Meltwater – grass surface [mm h⁻¹ 1]

Group snow

Mw_Paved

Description Meltwater – paved surface [mm h⁻¹]

Group snow

Mw_Water

Description Meltwater – water surface [mm h⁻¹]

Group snow

NWtrState

Description Surface wetness state (for non-water surfaces) [mm]

	Group SUEWS
P_day	
	Description Daily total precipitation [mm]
	Group DailyState
Porosity	
	Description Porosity of deciduous trees [-]
	Group DailyState
Q2	
	Description Air specific humidity at 2 m agl [g kg ⁻¹]
	Group SUEWS
QE	
	Description Latent heat flux (calculated using SUEWS) [W m ⁻²]
	Group SUEWS
QElumps	
	Description Latent heat flux (calculated using LUMPS) [W m ⁻²]
	Group SUEWS
QF	
	Description Anthropogenic heat flux [W m ⁻²]
	Group SUEWS
QH	
	Description Sensible heat flux (calculated using SUEWS) [W m ⁻²]
	Group SUEWS
QHlumps	
	Description Sensible heat flux (calculated using LUMPS) [W m ⁻²]
	Group SUEWS
QHresis	
	Description Sensible heat flux (calculated using resistance method) [W m ⁻²]
	Group SUEWS
QM	
	Description Snow-related heat exchange [W m ⁻²]
	Group SUEWS
QFreeze	
	Description Internal energy change [W m ⁻²]
	Group SUEWS
QMRain	
	Description Heat released by rain on snow [W m ⁻²]

	Group SUEWS
QN	
	Description Net all-wave radiation [W m^{-2}]
	Group SUEWS
QNSnow	
	Description Net all-wave radiation for snow area [W m^{-2}]
	Group SUEWS
QNSnowFr	
	Description Net all-wave radiation for snow-free area [W m^{-2}]
	Group SUEWS
QS	
	Description Storage heat flux [W m^{-2}]
	Group SUEWS
Qa_BSoil	
	Description Advective heat – bare soil surface [W m^{-2}]
	Group snow
Qa_Bldgs	
	Description Advective heat – building surface [W m^{-2}]
	Group snow
Qa_DecTr	
	Description Advective heat – deciduous surface [W m^{-2}]
	Group snow
Qa_EveTr	
	Description Advective heat – evergreen surface [W m^{-2}]
	Group snow
Qa_Grass	
	Description Advective heat – grass surface [W m^{-2}]
	Group snow
Qa_Paved	
	Description Advective heat – paved surface [W m^{-2}]
	Group snow
Qa_Water	
	Description Advective heat – water surface [W m^{-2}]
	Group snow
QmFr_BSoil	
	Description Heat related to freezing of surface store – bare soil surface [W m^{-2}]

Group snow

QmFr_Bldgs

Description Heat related to freezing of surface store – building surface [W m^{-2}]

Group snow

QmFr_DecTr

Description Heat related to freezing of surface store – deciduous surface [W m^{-2}]

Group snow

QmFr_EveTr

Description Heat related to freezing of surface store – evergreen surface [W m^{-2}]

Group snow

QmFr_Grass

Description Heat related to freezing of surface store – grass surface [W m^{-2}]

Group snow

QmFr_Paved

Description Heat related to freezing of surface store – paved surface [W m^{-2}]

Group snow

QmFr_Water

Description Heat related to freezing of surface store – water [W m^{-2}]

Group snow

Qm_BSoil

Description Snowmelt-related heat – bare soil surface [W m^{-2}]

Group snow

Qm_Bldgs

Description Snowmelt-related heat – building surface [W m^{-2}]

Group snow

Qm_DecTr

Description Snowmelt-related heat – deciduous surface [W m^{-2}]

Group snow

Qm_EveTr

Description Snowmelt-related heat – evergreen surface [W m^{-2}]

Group snow

Qm_Grass

Description Snowmelt-related heat – grass surface [W m^{-2}]

Group snow

Qm_Paved

Description Snowmelt-related heat – paved surface [W m^{-2}]

	Group snow
Qm_Water	
	Description Snowmelt-related heat – water surface [W m^{-2}]
	Group snow
RA	
	Description Aerodynamic resistance [s m^{-1}]
	Group SUEWS
RH2	
	Description Relative humidity at 2 m agl [%]
	Group SUEWS
RO	
	Description Runoff [mm]
	Group SUEWS
ROImp	
	Description Above ground runoff over impervious surfaces [mm]
	Group SUEWS
ROPipe	
	Description Runoff to pipes [mm]
	Group SUEWS
ROSoil	
	Description Runoff to soil (sub-surface) [mm]
	Group SUEWS
ROVeg	
	Description Above ground runoff over vegetated surfaces [mm]
	Group SUEWS
ROWater	
	Description Runoff for water body [mm]
	Group SUEWS
RS	
	Description Surface resistance [s m^{-1}]
	Group SUEWS
Rain	
	Description Rain [mm]
	Group SUEWS
RainSn_BSoil	
	Description Rain on snow – bare soil surface [mm]

Group snow

RainSn_Bldgs

Description Rain on snow – building surface [mm]

Group snow

RainSn_DecTr

Description Rain on snow – deciduous surface [mm]

Group snow

RainSn_EveTr

Description Rain on snow – evergreen surface [mm]

Group snow

RainSn_Grass

Description Rain on snow – grass surface [mm]

Group snow

RainSn_Paved

Description Rain on snow – paved surface [mm]

Group snow

RainSn_Water

Description Rain on snow – water surface [mm]

Group snow

SDD_DecTr

Description Senescence degree days for deciduous tree [°C d]

Group DailyState

SDD_EveTr

Description Senescence degree days for evergreen tree [°C d]

Group DailyState

SDD_Grass

Description Senescence degree days for grass [°C d]

Group DailyState

SMD

Description Soil moisture deficit [mm]

Group SUEWS

SMDBSoil

Description Soil moisture deficit for bare soil surface [mm]

Group SUEWS

SMDBldgs

Description Soil moisture deficit for building surface [mm]

Group SUEWS

SMDDecTr

Description Soil moisture deficit for deciduous surface [mm]

Group SUEWS

SMDEveTr

Description Soil moisture deficit for evergreen surface [mm]

Group SUEWS

SMDGrass

Description Soil moisture deficit for grass surface [mm]

Group SUEWS

SMDPaved

Description Soil moisture deficit for paved surface [mm]

Group SUEWS

SWE

Description Snow water equivalent [mm]

Group SUEWS

SWE_BSoil

Description Snow water equivalent – bare soil surface [mm]

Group snow

SWE_Bldgs

Description Snow water equivalent – building surface [mm]

Group snow

SWE_DecTr

Description Snow water equivalent – deciduous surface [mm]

Group snow

SWE_EveTr

Description Snow water equivalent – evergreen surface [mm]

Group snow

SWE_Grass

Description Snow water equivalent – grass surface [mm]

Group snow

SWE_Paved

Description Snow water equivalent – paved surface [mm]

Group snow

SWE_Water

Description Snow water equivalent – water surface [mm]

Group snow

Sd_BSoil

Description Snow depth – bare soil surface [mm]

Group snow

Sd_Bldgs

Description Snow depth – building surface [mm]

Group snow

Sd_DecTr

Description Snow depth – deciduous surface [mm]

Group snow

Sd_EveTr

Description Snow depth – evergreen surface [mm]

Group snow

Sd_Grass

Description Snow depth – grass surface [mm]

Group snow

Sd_Paved

Description Snow depth – paved surface [mm]

Group snow

Sd_Water

Description Snow depth – water surface [mm]

Group snow

SnowCh

Description Change in snow pack [mm]

Group SUEWS

SnowRBldgs

Description Snow removed from building surface [mm]

Group SUEWS

SnowRPaved

Description Snow removed from paved surface [mm]

Group SUEWS

StBSoil

Description Surface wetness state for bare soil surface [mm]

Group SUEWS

StBldgs

Description Surface wetness state for building surface [mm]

Group SUEWS

StDecTr

Description Surface wetness state for deciduous tree surface [mm]

Group SUEWS

StEveTr

Description Surface wetness state for evergreen tree surface [mm]

Group SUEWS

StGrass

Description Surface wetness state for grass surface [mm]

Group SUEWS

StPaved

Description Surface wetness state for paved surface [mm]

Group SUEWS

StWater

Description Surface wetness state for water surface [mm]

Group SUEWS

State

Description Surface wetness state [mm]

Group SUEWS

SurfCh

Description Change in surface moisture store [mm]

Group SUEWS

T2

Description Air temperature at 2 m agl [°C]

Group SUEWS

T_1

Description Air temperature at level 1 [°C]

Group RSL

T_10

Description Air temperature at level 10 [°C]

Group RSL

T_11

Description Air temperature at level 11 [°C]

Group RSL

T_12

Description Air temperature at level 12 [°C]

	Group RSL
T_13	
	Description Air temperature at level 13 [°C]
	Group RSL
T_14	
	Description Air temperature at level 14 [°C]
	Group RSL
T_15	
	Description Air temperature at level 15 [°C]
	Group RSL
T_16	
	Description Air temperature at level 16 [°C]
	Group RSL
T_17	
	Description Air temperature at level 17 [°C]
	Group RSL
T_18	
	Description Air temperature at level 18 [°C]
	Group RSL
T_19	
	Description Air temperature at level 19 [°C]
	Group RSL
T_2	
	Description Air temperature at level 2 [°C]
	Group RSL
T_20	
	Description Air temperature at level 20 [°C]
	Group RSL
T_21	
	Description Air temperature at level 21 [°C]
	Group RSL
T_22	
	Description Air temperature at level 22 [°C]
	Group RSL
T_23	
	Description Air temperature at level 23 [°C]

	Group RSL
T_24	
	Description Air temperature at level 24 [°C]
	Group RSL
T_25	
	Description Air temperature at level 25 [°C]
	Group RSL
T_26	
	Description Air temperature at level 26 [°C]
	Group RSL
T_27	
	Description Air temperature at level 27 [°C]
	Group RSL
T_28	
	Description Air temperature at level 28 [°C]
	Group RSL
T_29	
	Description Air temperature at level 29 [°C]
	Group RSL
T_3	
	Description Air temperature at level 3 [°C]
	Group RSL
T_30	
	Description Air temperature at level 30 [°C]
	Group RSL
T_4	
	Description Air temperature at level 4 [°C]
	Group RSL
T_5	
	Description Air temperature at level 5 [°C]
	Group RSL
T_6	
	Description Air temperature at level 6 [°C]
	Group RSL
T_7	
	Description Air temperature at level 7 [°C]

	Group RSL
T_8	
	Description Air temperature at level 8 [°C]
	Group RSL
T_9	
	Description Air temperature at level 9 [°C]
	Group RSL
Ta	
	Description Air temperature
	Group SOLWEIG
Tg	
	Description Surface temperature
	Group SOLWEIG
Tmax	
	Description Daily maximum temperature [°C]
	Group DailyState
Tmin	
	Description Daily minimum temperature [°C]
	Group DailyState
Tmrt	
	Description Mean Radiant Temperature
	Group SOLWEIG
TotCh	
	Description Change in surface and soil moisture stores [mm]
	Group SUEWS
Ts	
	Description Skin temperature [°C]
	Group SUEWS
Tsnow_BSoil	
	Description Snow surface temperature – bare soil surface [°C]
	Group snow
Tsnow_Bldgs	
	Description Snow surface temperature – building surface [°C]
	Group snow
Tsnow_DecTr	
	Description Snow surface temperature – deciduous surface [°C]

Group snow

Tsnow_EveTr

Description Snow surface temperature – evergreen surface [°C]

Group snow

Tsnow_Grass

Description Snow surface temperature – grass surface [°C]

Group snow

Tsnow_Paved

Description Snow surface temperature – paved surface [°C]

Group snow

Tsnow_Water

Description Snow surface temperature – water surface [°C]

Group snow

Tsurf

Description Bulk surface temperature [°C]

Group SUEWS

U10

Description Wind speed at 10 m agl [m s⁻¹]

Group SUEWS

U_1

Description Wind speed at level 1 [m s⁻¹]

Group RSL

U_10

Description Wind speed at level 10 [m s⁻¹]

Group RSL

U_11

Description Wind speed at level 11 [m s⁻¹]

Group RSL

U_12

Description Wind speed at level 12 [m s⁻¹]

Group RSL

U_13

Description Wind speed at level 13 [m s⁻¹]

Group RSL

U_14

Description Wind speed at level 14 [m s⁻¹]

	Group RSL
U_15	
	Description Wind speed at level 15 [m s ⁻¹]
	Group RSL
U_16	
	Description Wind speed at level 16 [m s ⁻¹]
	Group RSL
U_17	
	Description Wind speed at level 17 [m s ⁻¹]
	Group RSL
U_18	
	Description Wind speed at level 18 [m s ⁻¹]
	Group RSL
U_19	
	Description Wind speed at level 19 [m s ⁻¹]
	Group RSL
U_2	
	Description Wind speed at level 2 [m s ⁻¹]
	Group RSL
U_20	
	Description Wind speed at level 20 [m s ⁻¹]
	Group RSL
U_21	
	Description Wind speed at level 21 [m s ⁻¹]
	Group RSL
U_22	
	Description Wind speed at level 22 [m s ⁻¹]
	Group RSL
U_23	
	Description Wind speed at level 23 [m s ⁻¹]
	Group RSL
U_24	
	Description Wind speed at level 24 [m s ⁻¹]
	Group RSL
U_25	
	Description Wind speed at level 25 [m s ⁻¹]

	Group RSL
U_26	
	Description Wind speed at level 26 [m s ⁻¹]
	Group RSL
U_27	
	Description Wind speed at level 27 [m s ⁻¹]
	Group RSL
U_28	
	Description Wind speed at level 28 [m s ⁻¹]
	Group RSL
U_29	
	Description Wind speed at level 29 [m s ⁻¹]
	Group RSL
U_3	
	Description Wind speed at level 3 [m s ⁻¹]
	Group RSL
U_30	
	Description Wind speed at level 30 [m s ⁻¹]
	Group RSL
U_4	
	Description Wind speed at level 4 [m s ⁻¹]
	Group RSL
U_5	
	Description Wind speed at level 5 [m s ⁻¹]
	Group RSL
U_6	
	Description Wind speed at level 6 [m s ⁻¹]
	Group RSL
U_7	
	Description Wind speed at level 7 [m s ⁻¹]
	Group RSL
U_8	
	Description Wind speed at level 8 [m s ⁻¹]
	Group RSL
U_9	
	Description Wind speed at level 9 [m s ⁻¹]

Group RSL

WUDecTr

Description Water use for irrigation of deciduous trees [mm]

Group SUEWS

WUEveTr

Description Water use for irrigation of evergreen trees [mm]

Group SUEWS

WUGrass

Description Water use for irrigation of grass [mm]

Group SUEWS

WUInt

Description Internal water use [mm]

Group SUEWS

WU_DecTr1

Description Total water use for deciduous trees [mm]

Group DailyState

WU_DecTr2

Description Automatic water use for deciduous trees [mm]

Group DailyState

WU_DecTr3

Description Manual water use for deciduous trees [mm]

Group DailyState

WU_EveTr1

Description Total water use for evergreen trees [mm]

Group DailyState

WU_EveTr2

Description Automatic water use for evergreen trees [mm]

Group DailyState

WU_EveTr3

Description Manual water use for evergreen trees [mm]

Group DailyState

WU_Grass1

Description Total water use for grass [mm]

Group DailyState

WU_Grass2

Description Automatic water use for grass [mm]

Group DailyState

WU_Grass3

Description Manual water use for grass [mm]

Group DailyState

Zenith

Description Solar zenith angle [°]

Group SUEWS

a1

Description OHM coefficient a1 - [-]

Group DailyState

a2

Description OHM coefficient a2 [$\text{W m}^{-2} \text{h}^{-1}$]

Group DailyState

a3

Description OHM coefficient a3 - [W m^{-2}]

Group DailyState

altitude

Description Altitude angle of the Sun

Group SOLWEIG

azimuth

Description Azimuth angle of the Sun

Group SOLWEIG

deltaLAI

Description Change in leaf area index (normalised 0-1) [-]

Group DailyState

frMelt_BSoil

Description Amount of freezing melt water – bare soil surface [mm]

Group snow

frMelt_Bldgs

Description Amount of freezing melt water – building surface [mm]

Group snow

frMelt_DecTr

Description Amount of freezing melt water – deciduous surface [mm]

Group snow

frMelt_EveTr

Description Amount of freezing melt water – evergreen surface [mm]

Group snow

frMelt_Grass

Description Amount of freezing melt water – grass surface [mm]

Group snow

frMelt_Paved

Description Amount of freezing melt water – paved surface [mm]

Group snow

frMelt_Water

Description Amount of freezing melt water – water surface [mm]

Group snow

fr_Bldgs

Description Fraction of snow – building surface [-]

Group snow

fr_DecTr

Description Fraction of snow – deciduous surface [-]

Group snow

fr_EveTr

Description Fraction of snow – evergreen surface [-]

Group snow

fr_Grass

Description Fraction of snow – grass surface [-]

Group snow

fr_Paved

Description Fraction of snow – paved surface [-]

Group snow

gvf

Description Ground view factor (Lindberg and Grimmond 2011)

Group SOLWEIG

kup_BSoilSnow

Description Reflected shortwave radiation – bare soil surface [W m^{-2}]

Group snow

kup_BldgsSnow

Description Reflected shortwave radiation – building surface [W m^{-2}]

Group snow

kup_DecTrSnow

Description Reflected shortwave radiation – deciduous surface [W m^{-2}]

Group snow

kup_EveTrSnow

Description Reflected shortwave radiation – evergreen surface [W m^{-2}]

Group snow

kup_GrassSnow

Description Reflected shortwave radiation – grass surface [W m^{-2}]

Group snow

kup_PavedSnow

Description Reflected shortwave radiation – paved surface [W m^{-2}]

Group snow

kup_WaterSnow

Description Reflected shortwave radiation – water surface [W m^{-2}]

Group snow

q_1

Description Specific humidity at level 1 [g kg^{-1}]

Group RSL

q_10

Description Specific humidity at level 10 [g kg^{-1}]

Group RSL

q_11

Description Specific humidity at level 11 [g kg^{-1}]

Group RSL

q_12

Description Specific humidity at level 12 [g kg^{-1}]

Group RSL

q_13

Description Specific humidity at level 13 [g kg^{-1}]

Group RSL

q_14

Description Specific humidity at level 14 [g kg^{-1}]

Group RSL

q_15

Description Specific humidity at level 15 [g kg^{-1}]

Group RSL

q_16

Description Specific humidity at level 16 [g kg^{-1}]

	Group RSL
q_17	
	Description Specific humidity at level 17 [g kg ⁻¹]
	Group RSL
q_18	
	Description Specific humidity at level 18 [g kg ⁻¹]
	Group RSL
q_19	
	Description Specific humidity at level 19 [g kg ⁻¹]
	Group RSL
q_2	
	Description Specific humidity at level 2 [g kg ⁻¹]
	Group RSL
q_20	
	Description Specific humidity at level 20 [g kg ⁻¹]
	Group RSL
q_21	
	Description Specific humidity at level 21 [g kg ⁻¹]
	Group RSL
q_22	
	Description Specific humidity at level 22 [g kg ⁻¹]
	Group RSL
q_23	
	Description Specific humidity at level 23 [g kg ⁻¹]
	Group RSL
q_24	
	Description Specific humidity at level 24 [g kg ⁻¹]
	Group RSL
q_25	
	Description Specific humidity at level 25 [g kg ⁻¹]
	Group RSL
q_26	
	Description Specific humidity at level 26 [g kg ⁻¹]
	Group RSL
q_27	
	Description Specific humidity at level 27 [g kg ⁻¹]

	Group RSL
q_28	
	Description Specific humidity at level 28 [g kg ⁻¹]
	Group RSL
q_29	
	Description Specific humidity at level 29 [g kg ⁻¹]
	Group RSL
q_3	
	Description Specific humidity at level 3 [g kg ⁻¹]
	Group RSL
q_30	
	Description Specific humidity at level 30 [g kg ⁻¹]
	Group RSL
q_4	
	Description Specific humidity at level 4 [g kg ⁻¹]
	Group RSL
q_5	
	Description Specific humidity at level 5 [g kg ⁻¹]
	Group RSL
q_6	
	Description Specific humidity at level 6 [g kg ⁻¹]
	Group RSL
q_7	
	Description Specific humidity at level 7 [g kg ⁻¹]
	Group RSL
q_8	
	Description Specific humidity at level 8 [g kg ⁻¹]
	Group RSL
q_9	
	Description Specific humidity at level 9 [g kg ⁻¹]
	Group RSL
shadow	
	Description Shadow value (0= shadow, 1 = sun)
	Group SOLWEIG
svf	
	Description Sky View Factor from ground and buildings

Group SOLWEIG

svfbuveg

Description Sky View Factor from ground, buildings and vegetation

Group SOLWEIG

z0m

Description Roughness length for momentum [m]

Group SUEWS

z_1

Description Height at level 1 [m]

Group RSL

z_10

Description Height at level 10 [m]

Group RSL

z_11

Description Height at level 11 [m]

Group RSL

z_12

Description Height at level 12 [m]

Group RSL

z_13

Description Height at level 13 [m]

Group RSL

z_14

Description Height at level 14 [m]

Group RSL

z_15

Description Height at level 15 [m]

Group RSL

z_16

Description Height at level 16 [m]

Group RSL

z_17

Description Height at level 17 [m]

Group RSL

z_18

Description Height at level 18 [m]

	Group RSL
z_19	
	Description Height at level 19 [m]
	Group RSL
z_2	
	Description Height at level 2 [m]
	Group RSL
z_20	
	Description Height at level 20 [m]
	Group RSL
z_21	
	Description Height at level 21 [m]
	Group RSL
z_22	
	Description Height at level 22 [m]
	Group RSL
z_23	
	Description Height at level 23 [m]
	Group RSL
z_24	
	Description Height at level 24 [m]
	Group RSL
z_25	
	Description Height at level 25 [m]
	Group RSL
z_26	
	Description Height at level 26 [m]
	Group RSL
z_27	
	Description Height at level 27 [m]
	Group RSL
z_28	
	Description Height at level 28 [m]
	Group RSL
z_29	
	Description Height at level 29 [m]

	Group RSL
z_3	
	Description Height at level 3 [m]
	Group RSL
z_30	
	Description Height at level 30 [m]
	Group RSL
z_4	
	Description Height at level 4 [m]
	Group RSL
z_5	
	Description Height at level 5 [m]
	Group RSL
z_6	
	Description Height at level 6 [m]
	Group RSL
z_7	
	Description Height at level 7 [m]
	Group RSL
z_8	
	Description Height at level 8 [m]
	Group RSL
z_9	
	Description Height at level 9 [m]
	Group RSL
zdm	
	Description Zero-plane displacement height [m]
	Group SUEWS

Contents

- *I cannot install SuPy following the docs, what is wrong there?*
- *How do I know which version of SuPy I am using?*
- *A kernel may have died exception happened, where did I go wrong?*
- *How can I upgrade SuPy to an up-to-date version?*

4.1 I cannot install SuPy following the docs, what is wrong there?

please check if your environment meets the following requirements:

1. Operating system (OS):

- a. is it 64 bit? only 64 bit systems are supported.
- b. is your OS up to date? only recent desktop systems are supported:
 - Windows 10 and above
 - macOS 10.13 and above
 - Linux: no restriction; If SuPy cannot run on your specific Linux distribution, please report it to us.

You can get the OS information with the following code:

```
import platform
platform.platform()
```

2. Python interpreter:

- a. is your Python interpreter 64 bit?

Check running mode with the following code:

```
import struct
struct.calcsize('P') * 8
```

- b. is your Python version above 3.5?

Check version info with the following code:

```
import sys
sys.version
```

If your environment doesn't meet the requirement by SuPy, please use a proper environment; otherwise, [please report your issue](#).

4.2 How do I know which version of SuPy I am using?

Use the following code:

```
import supy
supy.show_version()
```

Note: `show_version` is only available after v2019.5.28.

4.3 A kernel may have died exception happened, where did I go wrong?

The issue is highly likely due to invalid input to SuPy and SUEWS kernel. We are trying to avoid such exceptions, but unfortunately they might happen in some edge cases.

Please [report such issues to us](#) with your input files for debugging. Thanks!

4.4 How can I upgrade SuPy to an up-to-date version?

Run the following code in your terminal:

```
python3 -m pip install supy --upgrade
```

5.1 Version Dev

- **New**
 1. Update supy-driver to 2020a iteration.
- **Improvement**

None.
- **Changes**

None.
- **Fix**

None.
- **Known issue**
 1. ESTM is not supported yet.
 2. BLUEWS, a CBL modules in SUEWS, is not supported yet.
 3. Simulation in parallel mode is NOT supported on Windows due to system limitation.

5.2 Version 20200202

- **New**
 1. A checker to validate input DataFrame`s. See option ``check_input`` in `run_supy`.
 2. Utilities to generate forcing data using ERA-5 data. See `download_era5` and `gen_forcing_era5`.
- **Improvement**
 1. Improved performance of the parallel mode.

- **Changes**

None.

- **Fix**

None.

- **Known issue**

1. ESTM is not supported yet.
2. BLUEWS, a CBL modules in SUEWS, is not supported yet.
3. Simulation in parallel mode is NOT supported on Windows due to system limitation.

5.3 Version 20190829

- **New**

1. added WRF-SUEWS related functions.
2. added [diagnostics of canyon profiles](#).

- **Improvement**

None.

- **Changes**

1. synchronised with v2019a interface: minimum supy_driver v2019a2.

- **Fix**

None.

- **Known issue**

1. ESTM is not supported yet.
2. BLUEWS, a CBL modules in SUEWS, is not supported yet.
3. Performance in parallel mode can be worse than serial mode sometimes due to heavy (de)-serialisation loads.

5.4 Version 2019.7.17

- **New**

1. added OHM related functions.
2. added surface conductance related functions.

- **Improvement**

None.

- **Changes**

None.

- **Fix**

1. Fixed a bug in unit conversion for TMY data generation.

- **Known issue**

ESTM is not supported yet.

5.5 Version 2019.6.8

- **New**

None.

- **Improvement**

None.

- **Changes**

None.

- **Fix**

1. Fixed a bug in rescaling Kdown when loading forcing data.

- **Known issue**

ESTM is not supported yet.

5.6 Version 2019.5.28

Spring house cleaning with long-await command line tools (more on the way!).

- **New**

1. Added version info function: `show_version`.
2. Added command line tools:
 - `suews-run`: SuPy wrapper to mimic SUEWS-binary-based simulation.
 - `suews-convert`: convert input tables from older versions to newer ones (one-way only).

- **Improvement**

None.

- **Changes**

None.

- **Fix**

1. Fixed a bug in writing out multi-grid output files caused by incorrect dropping of temporal information by pandas .

- **Known issue**

ESTM is not supported yet.

5.7 Version 2019.4.29

Parallel run.

- **New**
Added support for parallel run on the fly.
- **Improvement**
None.
- **Changes**
None.
- **Fix**
None.
- **Known issue**
None

5.8 Version 2019.4.17

UMEP compatibility tweaks.

- **New**
None.
- **Improvement**
None.
- **Changes**
`Error messages: problems.txt` will be written out in addition to the console error message similarly as SUEWS binary.
- **Fix**
Incorrect caching of input libraries.
- **Known issue**
None

5.9 Version 2019.4.15

ERA-5 download.

- **New**
Added experimental support for downloading and processing ERA-5 data to force supy simulations.
- **Improvement**
Improved compatibility with earlier `pandas` version in resampling output.

- **Changes**

None.

- **Fix**

None.

- **Known issue**

None

5.10 Version 2019.3.21

TMY generation.

- **New**

Added preliminary support for generating TMY dataset with SuPy output.

- **Improvement**

None.

- **Changes**

None.

- **Fix**

None.

- **Known issue**

None

5.11 Version 2019.3.14

This release improved memory usage.

- **New**

None.

- **Improvement**

Optimised memory consumption for longterm simulations.

- **Changes**

None.

- **Fix**

None.

- **Known issue**

None

5.12 Version 2019.2.25

This release dropped support for Python 3.5 and below.

- **New**
None.
- **Improvement**
None.
- **Changes**
Dropped support for Python 3.5 and below.
- **Fix**
None.
- **Known issue**
None

5.13 Version 2019.2.24

This release added the ability to save output files.

- **New**
 1. Added support to save output files. See: `supy.save_supy()`
 2. Added support to initialise SuPy from saved `df_state.csv`. See: `supy.init_supy()`
- **Improvement**
None.
- **Changes**
None.
- **Fix**
None.
- **Known issue**
None

5.14 Version 2019.2.19

This is a release that improved the exception handling due to fatal error in `supy_driver`.

- **New**
Added support to handle python kernel crash caused by fatal error in `supy_driver` kernel; so python kernel won't crash any more even `supy_driver` is stopped.
- **Improvement**
None.

- **Changes**

None

- **Fix**

None.

- **Known issue**

None

5.15 Version 2019.2.8

This is a release that fixes recent bugs found in SUEWS that may lead to abnormal simulation results of storage heat flux, in particular when `SnowUse` is enabled (i.e., `snowuse=1`).

- **New**

None.

- **Improvement**

Improved the performance in loading initial model state from a large number of grids (>1k)

- **Changes**

Updated `SampleRun` dataset by: 1. setting surface fractions (*sfr*) to a more realistic value based on London KCL case; 2. enabling snow module (`snowuse=1`).

- **Fix**

1. Fixed a bug in the calculation of storage heat flux.
2. Fixed a bug in loading `popdens` for calculating anthropogenic heat flux.

- **Known issue**

None

5.16 Version 2019.1.1 (preview release, 01 Jan 2019)

- **New**

1. Slimmed the output groups by excluding unsupported `ESTM` results
2. SuPy documentation
 - Key IO data structures documented:
 - *df_output variables* (GH9)
 - *df_state variables* (GH8)
 - *df_forcing variables* (GH7)
 - Tutorial of parallel SuPy simulations for impact studies

- **Improvement**

1. Improved calculation of OHM-related radiation terms

- **Changes**

None.

- **Fix**

None

- **Known issue**

None

5.17 Version 2018.12.15 (internal test release in December 2018)

- **New**

1. Preview release of SuPy based on the computation kernel of SUEWS 2018b

- **Improvement**

1. Improved calculation of OHM-related radiation terms

- **Changes**

None.

- **Fix**

None

- **Known issue**

1. The heat storage modules AnOHM and ESTM are not supported yet.

Symbols

-f, -from <fromVer>
 suews-convert command line option,
 70

-i, -input <fromDir>
 suews-convert command line option,
 70

-o, -output <toDir>
 suews-convert command line option,
 70

-t, -to <toVer>
 suews-convert command line option,
 70

A

a1
 command line option, 120

a2
 command line option, 120

a3
 command line option, 120

AddWater
 command line option, 96

aerodynamicresistancemethod
 command line option, 70

ah_min
 command line option, 70

ah_slope_cooling
 command line option, 70

ah_slope_heating
 command line option, 71

ahprof_24hr
 command line option, 71

alb
 command line option, 71

AlbBulk
 command line option, 96

AlbDecTr
 command line option, 96

albdectr_id
 command line option, 71

AlbEveTr
 command line option, 96

albevetr_id
 command line option, 71

AlbGrass
 command line option, 96

albgrass_id
 command line option, 71

albmax_dectr
 command line option, 71

albmax_evetr
 command line option, 72

albmax_grass
 command line option, 72

albmin_dectr
 command line option, 72

albmin_evetr
 command line option, 72

albmin_grass
 command line option, 72

AlbSnow
 command line option, 96, 97

alpha_bioco2
 command line option, 72

alpha_enh_bioco2
 command line option, 72

alt
 command line option, 73

altitude
 command line option, 120

Azimuth
 command line option, 97

azimuth
 command line option, 120

B

baset
 command line option, 73

basete
 command line option, 73
 basethdd
 command line option, 73
 beta_bioco2
 command line option, 73
 beta_enh_bioco2
 command line option, 73
 bldgh
 command line option, 73

C

cal_gs_mod() (in module *supy.util*), 66
 cal_gs_obs() (in module *supy.util*), 67
 calib_g() (in module *supy.util*), 67
 capmax_dec
 command line option, 74
 capmin_dec
 command line option, 74
 chanohm
 command line option, 74
 CI
 command line option, 97
 co2pointsource
 command line option, 74
 command line option
 a1, 120
 a2, 120
 a3, 120
 AddWater, 96
 aerodynamicresistancemethod, 70
 ah_min, 70
 ah_slope_cooling, 70
 ah_slope_heating, 71
 ahprof_24hr, 71
 alb, 71
 AlbBulk, 96
 AlbDecTr, 96
 albdectr_id, 71
 AlbEveTr, 96
 albevetr_id, 71
 AlbGrass, 96
 albgrass_id, 71
 albmax_dectr, 71
 albmax_evetr, 72
 albmax_grass, 72
 albmin_dectr, 72
 albmin_evetr, 72
 albmin_grass, 72
 AlbSnow, 96, 97
 alpha_bioco2, 72
 alpha_enh_bioco2, 72
 alt, 73
 altitude, 120

Azimuth, 97
 azimuth, 120
 baset, 73
 basete, 73
 basethdd, 73
 beta_bioco2, 73
 beta_enh_bioco2, 73
 bldgh, 73
 capmax_dec, 74
 capmin_dec, 74
 chanohm, 74
 CI, 97
 co2pointsource, 74
 cpanohm, 74
 crwmax, 74
 crwmin, 74
 DaysSR, 97
 daywat, 75
 daywatper, 75
 DecidCap, 97
 decidcap_id, 75
 dectreeh, 75
 deltaLAI, 120
 DensSnow_Bldgs, 97, 98
 DensSnow_BSoil, 97
 DensSnow_DecTr, 98
 DensSnow_EveTr, 98
 DensSnow_Grass, 98, 99
 DensSnow_Paved, 99
 DensSnow_Water, 99
 diagnose, 75
 diagqn, 75
 diagqs, 75
 DiffuseRad, 99
 DirectRad, 99
 DLHrs, 97
 Drainage, 100
 drainrt, 76
 ef_umolco2perj, 76
 emis, 76
 emissionsmethod, 76
 enddls, 76
 enef_v_jkm, 76
 Evap, 100
 evapmethod, 76
 evetreeh, 77
 faibldg, 77
 faidectree, 77
 faievetree, 77
 faut, 77
 Fc, 100
 FcBuild, 100
 fcef_v_kgkm, 77
 Fcld, 100

fcld, 95
FcMetab, 100
FcPhoto, 100
FcPoint, 100
FcRespi, 100
FcTraff, 100
FlowCh, 100
flowchange, 77
fr_Bldgs, 121
fr_DecTr, 121
fr_EveTr, 121
fr_Grass, 121
fr_Paved, 121
frfossilfuel_heat, 78
frfossilfuel_nonheat, 78
frMelt_Bldgs, 120
frMelt_BSoil, 120
frMelt_DecTr, 120
frMelt_EveTr, 120
frMelt_Grass, 121
frMelt_Paved, 121
frMelt_Water, 121
g1, 78
g2, 78
g3, 78
g4, 78
g5, 78
g6, 78
GDD_DecTr, 100
GDD_EveTr, 101
GDD_Grass, 101
gddfull, 79
GlobalRad, 101
gsmodel, 79
gvf, 121
HDD1_h, 101
HDD2_c, 101
HDD3_Tmean, 101
HDD4_T5d, 101
humactivity_24hr, 79
I0, 101
id, 95
ie_a, 79
ie_end, 79
ie_m, 79
ie_start, 80
imin, 95
internalwateruse_h, 80
Irr, 101
irrfracconif, 80
irrfracdecid, 80
irrfracgrass, 80
isec, 95
it, 95
iy, 95
kdiff, 95
kdir, 95
Kdown, 101
kdown, 95
Kdown2d, 101
Keast, 101
kkanohm, 80
kmax, 80
Knorth, 102
Ksouth, 102
Kup, 102
Kup2d, 102
kup_BldgsSnow, 121
kup_BSoilSnow, 121
kup_DecTrSnow, 121
kup_EveTrSnow, 122
kup_GrassSnow, 122
kup_PavedSnow, 122
kup_WaterSnow, 122
Kwest, 102
LAI, 102
lai, 95
LAI_DecTr, 102
LAI_EveTr, 102
LAI_Grass, 102
lai_id, 80
laicalcyes, 81
LAIlumps, 102
laimax, 81
laimin, 81
laipower, 81
laitype, 81
lat, 81
Ldown, 102
ldown, 95
Ldown2d, 102
Least, 103
lng, 82
Lnorth, 103
Lob, 103
Lsouth, 103
Lup, 103
Lup2d, 103
Lwest, 103
maxconductance, 82
maxfcmetab, 82
maxqfmetab, 82
MeltWater, 103
MeltWStore, 103
min_res_bioco2, 82
minfcmetab, 82
minqfmetab, 82
Mw_Bldgs, 104

Mw_BSoil, 104
Mw_DecTr, 104
Mw_EveTr, 104
Mw_Grass, 104
Mw_Paved, 104
Mw_Water, 104
MwStore_Bldgs, 103
MwStore_BSoil, 103
MwStore_DecTr, 103
MwStore_EveTr, 104
MwStore_Grass, 104
MwStore_Paved, 104
MwStore_Water, 104
narp_emis_snow, 82
narp_trans_site, 83
netradiationmethod, 83
NWtrState, 104
ohm_coef, 83
ohm_threshsw, 83
ohm_threshwd, 83
ohmincqf, 83
P_day, 105
pipecapacity, 84
popdensdaytime, 84
popdensnighttime, 84
popprof_24hr, 84
pormax_dec, 84
pormin_dec, 84
Porosity, 105
porosity_id, 84
preciplimit, 85
preciplimitalb, 85
pres, 95
Q2, 105
q_1, 122
q_10, 122
q_11, 122
q_12, 122
q_13, 122
q_14, 122
q_15, 122
q_16, 122
q_17, 123
q_18, 123
q_19, 123
q_2, 123
q_20, 123
q_21, 123
q_22, 123
q_23, 123
q_24, 123
q_25, 123
q_26, 123
q_27, 123
q_28, 124
q_29, 124
q_3, 124
q_30, 124
q_4, 124
q_5, 124
q_6, 124
q_7, 124
q_8, 124
q_9, 124
Qa_Bldgs, 106
Qa_BSoil, 106
Qa_DecTr, 106
Qa_EveTr, 106
Qa_Grass, 106
Qa_Paved, 106
Qa_Water, 106
QE, 105
qe, 95
QElumps, 105
QF, 105
qf, 95
qf0_beu, 85
qf_a, 85
qf_b, 85
qf_c, 85
QH, 105
qh, 96
QHlumps, 105
QHresis, 105
QM, 105
Qm_Bldgs, 107
Qm_BSoil, 107
Qm_DecTr, 107
Qm_EveTr, 107
Qm_Grass, 107
Qm_Paved, 107
Qm_Water, 108
QmFr_Bldgs, 107
QmFr_BSoil, 106
QmFr_DecTr, 107
QmFr_EveTr, 107
QmFr_Grass, 107
QmFr_Paved, 107
QmFr_Water, 107
QMFreeze, 105
QMRain, 105
QN, 106
qn, 96
QNSnow, 106
QNSnowFr, 106
QS, 106
qs, 96
RA, 108

radmeltfact, 85
 Rain, 108
 rain, 96
 raincover, 86
 rainmaxres, 86
 RainSn_Bldgs, 109
 RainSn_BSoil, 108
 RainSn_DecTr, 109
 RainSn_EveTr, 109
 RainSn_Grass, 109
 RainSn_Paved, 109
 RainSn_Water, 109
 resp_a, 86
 resp_b, 86
 RH, 94
 RH2, 108
 RO, 108
 ROImp, 108
 ROPipe, 108
 ROSoil, 108
 roughlenheatmethod, 86
 roughlenmommmethod, 86
 ROVeg, 108
 ROWater, 108
 RS, 108
 runofftowater, 86
 s1, 87
 s2, 87
 sathydraulicconduct, 87
 Sd_Bldgs, 111
 Sd_BSoil, 111
 Sd_DecTr, 111
 Sd_EveTr, 111
 Sd_Grass, 111
 Sd_Paved, 111
 Sd_Water, 111
 SDD_DecTr, 109
 SDD_EveTr, 109
 SDD_Grass, 109
 sddfull, 87
 sfr, 87
 shadow, 124
 SMD, 109
 SMD_Bldgs, 109
 SMD_BSoil, 109
 SMD_DecTr, 110
 SMD_EveTr, 110
 SMD_Grass, 110
 smdmethod, 87
 SMD_Paved, 110
 snow, 96
 snowalb, 87
 snowalbmax, 88
 snowalbmin, 88
 SnowCh, 111
 snowdens, 88
 snowdensmax, 88
 snowdensmin, 88
 snowfrac, 88
 snowlimbldg, 88
 snowlimpaved, 89
 snowpack, 89
 snowpacklimit, 89
 snowprof_24hr, 89
 SnowRBldgs, 111
 SnowRPaved, 111
 snowuse, 89
 snowwater, 89
 soildepth, 89
 soilstore_id, 90
 soilstorecap, 90
 stabilitymethod, 90
 startdls, 90
 State, 112
 state_id, 90
 statelimit, 90
 StBldgs, 111
 StBSoil, 111
 StDecTr, 112
 StEveTr, 112
 StGrass, 112
 storageheatmethod, 91
 storedrainprm, 91
 StPaved, 112
 StWater, 112
 surfacearea, 91
 SurfCh, 112
 svf, 124
 svfbuveg, 125
 SWE, 110
 SWE_Bldgs, 110
 SWE_BSoil, 110
 SWE_DecTr, 110
 SWE_EveTr, 110
 SWE_Grass, 110
 SWE_Paved, 110
 SWE_Water, 110
 T2, 112
 T_1, 112
 T_10, 112
 T_11, 112
 T_12, 112
 T_13, 113
 T_14, 113
 T_15, 113
 T_16, 113
 T_17, 113
 T_18, 113

T_19, 113
T_2, 113
T_20, 113
T_21, 113
T_22, 113
T_23, 113
T_24, 114
T_25, 114
T_26, 114
T_27, 114
T_28, 114
T_29, 114
T_3, 114
T_30, 114
T_4, 114
T_5, 114
T_6, 114
T_7, 114
T_8, 115
T_9, 115
t_critic_cooling, 91
t_critic_heating, 91
Ta, 115
Tair, 95
tau_a, 91
tau_f, 91
tau_r, 92
tempmeltfact, 92
Tg, 115
th, 92
theta_bioco2, 92
timezone, 92
tl, 92
Tmax, 115
Tmin, 115
Tmrt, 115
TotCh, 115
traffirate, 92
trafficunits, 93
traffprof_24hr, 93
Ts, 115
Tsnow_Bldgs, 115
Tsnow_BSoil, 115
Tsnow_DecTr, 115
Tsnow_EveTr, 116
Tsnow_Grass, 116
Tsnow_Paved, 116
Tsnow_Water, 116
tstep, 93
Tsurf, 116
U, 95
U10, 116
U_1, 116
U_10, 116
U_11, 116
U_12, 116
U_13, 116
U_14, 116
U_15, 117
U_16, 117
U_17, 117
U_18, 117
U_19, 117
U_2, 117
U_20, 117
U_21, 117
U_22, 117
U_23, 117
U_24, 117
U_25, 117
U_26, 118
U_27, 118
U_28, 118
U_29, 118
U_3, 118
U_30, 118
U_4, 118
U_5, 118
U_6, 118
U_7, 118
U_8, 118
U_9, 118
veg_type, 93
waterdist, 93
waterusemethod, 93
wdir, 96
wetthresh, 93
WU_DecTr1, 119
WU_DecTr2, 119
WU_DecTr3, 119
WU_EveTr1, 119
WU_EveTr2, 119
WU_EveTr3, 119
WU_Grass1, 119
WU_Grass2, 119
WU_Grass3, 120
WUDecTr, 119
WUEveTr, 119
WUGrass, 119
Wuh, 95
WUInt, 119
wuprofa_24hr, 94
wuprofm_24hr, 94
xsmd, 96
z, 94
z0m, 125
z0m_in, 94
z_1, 125

z_10, 125
 z_11, 125
 z_12, 125
 z_13, 125
 z_14, 125
 z_15, 125
 z_16, 125
 z_17, 125
 z_18, 125
 z_19, 126
 z_2, 126
 z_20, 126
 z_21, 126
 z_22, 126
 z_23, 126
 z_24, 126
 z_25, 126
 z_26, 126
 z_27, 126
 z_28, 126
 z_29, 126
 z_3, 127
 z_30, 127
 z_4, 127
 z_5, 127
 z_6, 127
 z_7, 127
 z_8, 127
 z_9, 127
 zdm, 127
 zdm_in, 94
 Zenith, 120
 cpanohm
 command line option, 74
 crwmax
 command line option, 74
 crwmin
 command line option, 74
D
 DaysSR
 command line option, 97
 daywat
 command line option, 75
 daywatper
 command line option, 75
 DecidCap
 command line option, 97
 decidcap_id
 command line option, 75
 dectreeh
 command line option, 75
 deltaLAI
 command line option, 120

DensSnow_Bldgs
 command line option, 97, 98
 DensSnow_BSoil
 command line option, 97
 DensSnow_DecTr
 command line option, 98
 DensSnow_EveTr
 command line option, 98
 DensSnow_Grass
 command line option, 98, 99
 DensSnow_Paved
 command line option, 99
 DensSnow_Water
 command line option, 99
 derive_ohm_coef() (*in module supy.util*), 66
 diagnose
 command line option, 75
 diagqn
 command line option, 75
 diagqs
 command line option, 75
 DiffuseRad
 command line option, 99
 DirectRad
 command line option, 99
 DLHrs
 command line option, 97
 download_era5() (*in module supy.util*), 63
 Drainage
 command line option, 100
 drainrt
 command line option, 76

E

ef_umolco2perj
 command line option, 76
 emis
 command line option, 76
 emissionsmethod
 command line option, 76
 enddls
 command line option, 76
 enef_v_jkm
 command line option, 76
 Evap
 command line option, 100
 evapmethod
 command line option, 76
 evetreeh
 command line option, 77
 extract_reclassification() (*in module supy.util*), 68

F

`faibldg`
 command line option, 77
`faidectree`
 command line option, 77
`faievetre`
 command line option, 77
`faut`
 command line option, 77
`Fc`
 command line option, 100
`FcBuild`
 command line option, 100
`fcef_v_kgkm`
 command line option, 77
`Fcld`
 command line option, 100
`fcld`
 command line option, 95
`FcMetab`
 command line option, 100
`FcPhoto`
 command line option, 100
`FcPoint`
 command line option, 100
`FcRespi`
 command line option, 100
`FcTraff`
 command line option, 100
`fill_gap_all()` (*in module supy.util*), 65
`FlowCh`
 command line option, 100
`flowchange`
 command line option, 77
`fr_Bldgs`
 command line option, 121
`fr_DecTr`
 command line option, 121
`fr_EveTr`
 command line option, 121
`fr_Grass`
 command line option, 121
`fr_Paved`
 command line option, 121
`frfossilfuel_heat`
 command line option, 78
`frfossilfuel_nonheat`
 command line option, 78
`frMelt_Bldgs`
 command line option, 120
`frMelt_BSoil`
 command line option, 120
`frMelt_DecTr`
 command line option, 120

`frMelt_EveTr`
 command line option, 120
`frMelt_Grass`
 command line option, 121
`frMelt_Paved`
 command line option, 121
`frMelt_Water`
 command line option, 121

G

`g1`
 command line option, 78
`g2`
 command line option, 78
`g3`
 command line option, 78
`g4`
 command line option, 78
`g5`
 command line option, 78
`g6`
 command line option, 78
`GDD_DecTr`
 command line option, 100
`GDD_EveTr`
 command line option, 101
`GDD_Grass`
 command line option, 101
`gddfull`
 command line option, 79
`gen_epw()` (*in module supy.util*), 64
`gen_forcing_era5()` (*in module supy.util*), 63
`GlobalRad`
 command line option, 101
`gsmodel`
 command line option, 79
`gvf`
 command line option, 121

H

`HDD1_h`
 command line option, 101
`HDD2_c`
 command line option, 101
`HDD3_Tmean`
 command line option, 101
`HDD4_T5d`
 command line option, 101
`humactivity_24hr`
 command line option, 79

I

`I0`
 command line option, 101

id
 command line option, 95
 ie_a
 command line option, 79
 ie_end
 command line option, 79
 ie_m
 command line option, 79
 ie_start
 command line option, 80
 imin
 command line option, 95
 init_supy() (*in module supy*), 59
 internalwateruse_h
 command line option, 80
 Irr
 command line option, 101
 irrfracconif
 command line option, 80
 irrfracdecid
 command line option, 80
 irrfracgrass
 command line option, 80
 isec
 command line option, 95
 it
 command line option, 95
 iy
 command line option, 95

K

kdiff
 command line option, 95
 kdir
 command line option, 95
 Kdown
 command line option, 101
 kdown
 command line option, 95
 Kdown2d
 command line option, 101
 Keast
 command line option, 101
 kkanohm
 command line option, 80
 kmax
 command line option, 80
 Knorth
 command line option, 102
 Ksouth
 command line option, 102
 Kup
 command line option, 102
 Kup2d

 command line option, 102
 kup_BldgsSnow
 command line option, 121
 kup_BSoilSnow
 command line option, 121
 kup_DecTrSnow
 command line option, 121
 kup_EveTrSnow
 command line option, 122
 kup_GrassSnow
 command line option, 122
 kup_PavedSnow
 command line option, 122
 kup_WaterSnow
 command line option, 122
 Kwest
 command line option, 102

L

LAI
 command line option, 102
 lai
 command line option, 95
 LAI_DecTr
 command line option, 102
 LAI_EveTr
 command line option, 102
 LAI_Grass
 command line option, 102
 lai_id
 command line option, 80
 laicalcyes
 command line option, 81
 LAIlumps
 command line option, 102
 laimax
 command line option, 81
 laimin
 command line option, 81
 laipower
 command line option, 81
 laitype
 command line option, 81
 lat
 command line option, 81
 Ldown
 command line option, 102
 ldown
 command line option, 95
 Ldown2d
 command line option, 102
 Least
 command line option, 103
 lng

- command line option, 82
- Lnorth
 - command line option, 103
- load_forcing_grid() (*in module supy*), 60
- load_SampleData() (*in module supy*), 62
- Lob
 - command line option, 103
- Lsouth
 - command line option, 103
- Lup
 - command line option, 103
- Lup2d
 - command line option, 103
- Lwest
 - command line option, 103

M

- maxconductance
 - command line option, 82
- maxfcmstab
 - command line option, 82
- maxqfmetab
 - command line option, 82
- MeltWater
 - command line option, 103
- MeltWStore
 - command line option, 103
- min_res_bioco2
 - command line option, 82
- minfcmstab
 - command line option, 82
- minqfmetab
 - command line option, 82
- Mw_Bldgs
 - command line option, 104
- Mw_BSoil
 - command line option, 104
- Mw_DecTr
 - command line option, 104
- Mw_EveTr
 - command line option, 104
- Mw_Grass
 - command line option, 104
- Mw_Paved
 - command line option, 104
- Mw_Water
 - command line option, 104
- MwStore_Bldgs
 - command line option, 103
- MwStore_BSoil
 - command line option, 103
- MwStore_DecTr
 - command line option, 103
- MwStore_EveTr

- command line option, 104
- MwStore_Grass
 - command line option, 104
- MwStore_Paved
 - command line option, 104
- MwStore_Water
 - command line option, 104

N

- narp_emis_snow
 - command line option, 82
- narp_trans_site
 - command line option, 83
- netradiationmethod
 - command line option, 83
- NWtrState
 - command line option, 104

O

- ohm_coef
 - command line option, 83
- ohm_threshsw
 - command line option, 83
- ohm_threshwd
 - command line option, 83
- ohmincqf
 - command line option, 83

P

- P_day
 - command line option, 105
- PATH_RUNCONTROL
 - suews-run command line option, 70
- pipecapacity
 - command line option, 84
- plot_comp() (*in module supy.util*), 69
- plot_day_clm() (*in module supy.util*), 69
- plot_reclassification() (*in module supy.util*), 68
- popdensdaytime
 - command line option, 84
- popdensnighttime
 - command line option, 84
- popprof_24hr
 - command line option, 84
- pormax_dec
 - command line option, 84
- pormin_dec
 - command line option, 84
- Porosity
 - command line option, 105
- porosity_id
 - command line option, 84
- preciplimit

command line option, 85
 preciplimitalb
 command line option, 85
 pres
 command line option, 95

Q

Q2
 command line option, 105
 q_1
 command line option, 122
 q_10
 command line option, 122
 q_11
 command line option, 122
 q_12
 command line option, 122
 q_13
 command line option, 122
 q_14
 command line option, 122
 q_15
 command line option, 122
 q_16
 command line option, 122
 q_17
 command line option, 123
 q_18
 command line option, 123
 q_19
 command line option, 123
 q_2
 command line option, 123
 q_20
 command line option, 123
 q_21
 command line option, 123
 q_22
 command line option, 123
 q_23
 command line option, 123
 q_24
 command line option, 123
 q_25
 command line option, 123
 q_26
 command line option, 123
 q_27
 command line option, 123
 q_28
 command line option, 124
 q_29
 command line option, 124
 q_3

command line option, 124
 q_30
 command line option, 124
 q_4
 command line option, 124
 q_5
 command line option, 124
 q_6
 command line option, 124
 q_7
 command line option, 124
 q_8
 command line option, 124
 q_9
 command line option, 124
 Qa_Bldgs
 command line option, 106
 Qa_BSoil
 command line option, 106
 Qa_DecTr
 command line option, 106
 Qa_EveTr
 command line option, 106
 Qa_Grass
 command line option, 106
 Qa_Paved
 command line option, 106
 Qa_Water
 command line option, 106
 QE
 command line option, 105
 qe
 command line option, 95
 QElumps
 command line option, 105
 QF
 command line option, 105
 qf
 command line option, 95
 qf0_beu
 command line option, 85
 qf_a
 command line option, 85
 qf_b
 command line option, 85
 qf_c
 command line option, 85
 QH
 command line option, 105
 qh
 command line option, 96
 QHlumps
 command line option, 105
 QHresis

command line option, 105

QM

 command line option, 105

Qm_Bldgs

 command line option, 107

Qm_BSoil

 command line option, 107

Qm_DecTr

 command line option, 107

Qm_EveTr

 command line option, 107

Qm_Grass

 command line option, 107

Qm_Paved

 command line option, 107

Qm_Water

 command line option, 108

QmFr_Bldgs

 command line option, 107

QmFr_BSoil

 command line option, 106

QmFr_DecTr

 command line option, 107

QmFr_EveTr

 command line option, 107

QmFr_Grass

 command line option, 107

QmFr_Paved

 command line option, 107

QmFr_Water

 command line option, 107

QMFreeze

 command line option, 105

QMRain

 command line option, 105

QN

 command line option, 106

qn

 command line option, 96

QNSnow

 command line option, 106

QNSnowFr

 command line option, 106

QS

 command line option, 106

qs

 command line option, 96

R

RA

 command line option, 108

radmeltfact

 command line option, 85

Rain

 command line option, 108

rain

 command line option, 96

raincover

 command line option, 86

rainmaxres

 command line option, 86

RainSn_Bldgs

 command line option, 109

RainSn_BSoil

 command line option, 108

RainSn_DecTr

 command line option, 109

RainSn_EveTr

 command line option, 109

RainSn_Grass

 command line option, 109

RainSn_Paved

 command line option, 109

RainSn_Water

 command line option, 109

read_epw() (*in module supy.util*), 65

resp_a

 command line option, 86

resp_b

 command line option, 86

RH

 command line option, 94

RH2

 command line option, 108

RO

 command line option, 108

ROImp

 command line option, 108

ROPipe

 command line option, 108

ROSoil

 command line option, 108

roughlenheatmethod

 command line option, 86

roughlenmommethode

 command line option, 86

ROVeg

 command line option, 108

ROWater

 command line option, 108

RS

 command line option, 108

run_supy() (*in module supy*), 60

runofftowater

 command line option, 86

S

s1

command line option, 87
 s2
 command line option, 87
 sathydraulicconduct
 command line option, 87
 save_supy() (*in module supy*), 61
 Sd_Bldgs
 command line option, 111
 Sd_BSoil
 command line option, 111
 Sd_DecTr
 command line option, 111
 Sd_EveTr
 command line option, 111
 Sd_Grass
 command line option, 111
 Sd_Paved
 command line option, 111
 Sd_Water
 command line option, 111
 SDD_DecTr
 command line option, 109
 SDD_EveTr
 command line option, 109
 SDD_Grass
 command line option, 109
 sddfull
 command line option, 87
 sfr
 command line option, 87
 shadow
 command line option, 124
 show_version() (*in module supy*), 63
 sim_ohm() (*in module supy.util*), 66
 SMD
 command line option, 109
 SMDBldgs
 command line option, 109
 SMDBSoil
 command line option, 109
 SMDDecTr
 command line option, 110
 SMDEveTr
 command line option, 110
 SMDGrass
 command line option, 110
 smdmethod
 command line option, 87
 SMDPaved
 command line option, 110
 snow
 command line option, 96
 snowalb
 command line option, 87
 snowalbmax
 command line option, 88
 snowalbmin
 command line option, 88
 SnowCh
 command line option, 111
 snowdens
 command line option, 88
 snowdensmax
 command line option, 88
 snowdensmin
 command line option, 88
 snowfrac
 command line option, 88
 snowlimbldg
 command line option, 88
 snowlimpaved
 command line option, 89
 snowpack
 command line option, 89
 snowpacklimit
 command line option, 89
 snowprof_24hr
 command line option, 89
 SnowRBldgs
 command line option, 111
 SnowRPaved
 command line option, 111
 snowuse
 command line option, 89
 snowwater
 command line option, 89
 soildepth
 command line option, 89
 soilstore_id
 command line option, 90
 soilstorecap
 command line option, 90
 stabilitymethod
 command line option, 90
 startdls
 command line option, 90
 State
 command line option, 112
 state_id
 command line option, 90
 statelimit
 command line option, 90
 StBldgs
 command line option, 111
 StBSoil
 command line option, 111
 StDecTr
 command line option, 112

StEveTr
 command line option, 112

StGrass
 command line option, 112

storageheatmethod
 command line option, 91

storedrainprm
 command line option, 91

StPaved
 command line option, 112

StWater
 command line option, 112

suews-convert command line option
 -f, -from <fromVer>, 70
 -i, -input <fromDir>, 70
 -o, -output <toDir>, 70
 -t, -to <toVer>, 70

suews-run command line option
 PATH_RUNCONTROL, 70

surfacearea
 command line option, 91

SurfCh
 command line option, 112

svf
 command line option, 124

svfbuveg
 command line option, 125

SWE
 command line option, 110

SWE_Bldgs
 command line option, 110

SWE_BSoil
 command line option, 110

SWE_DecTr
 command line option, 110

SWE_EveTr
 command line option, 110

SWE_Grass
 command line option, 110

SWE_Paved
 command line option, 110

SWE_Water
 command line option, 110

T

T2
 command line option, 112

T_1
 command line option, 112

T_10
 command line option, 112

T_11
 command line option, 112

T_12
 command line option, 112

T_13
 command line option, 113

T_14
 command line option, 113

T_15
 command line option, 113

T_16
 command line option, 113

T_17
 command line option, 113

T_18
 command line option, 113

T_19
 command line option, 113

T_2
 command line option, 113

T_20
 command line option, 113

T_21
 command line option, 113

T_22
 command line option, 113

T_23
 command line option, 113

T_24
 command line option, 114

T_25
 command line option, 114

T_26
 command line option, 114

T_27
 command line option, 114

T_28
 command line option, 114

T_29
 command line option, 114

T_3
 command line option, 114

T_30
 command line option, 114

T_4
 command line option, 114

T_5
 command line option, 114

T_6
 command line option, 114

T_7
 command line option, 114

T_8
 command line option, 115

T_9
 command line option, 115

t_critic_cooling

command line option, 91
 t_critic_heating
 command line option, 91
 Ta
 command line option, 115
 Tair
 command line option, 95
 tau_a
 command line option, 91
 tau_f
 command line option, 91
 tau_r
 command line option, 92
 tempmeltfact
 command line option, 92
 Tg
 command line option, 115
 th
 command line option, 92
 theta_bioco2
 command line option, 92
 timezone
 command line option, 92
 tl
 command line option, 92
 Tmax
 command line option, 115
 Tmin
 command line option, 115
 Tmrt
 command line option, 115
 TotCh
 command line option, 115
 trafficrate
 command line option, 92
 trafficunits
 command line option, 93
 traffprof_24hr
 command line option, 93
 Ts
 command line option, 115
 Tsnow_Bldgs
 command line option, 115
 Tsnow_BSoil
 command line option, 115
 Tsnow_DecTr
 command line option, 115
 Tsnow_EveTr
 command line option, 116
 Tsnow_Grass
 command line option, 116
 Tsnow_Paved
 command line option, 116
 Tsnow_Water

command line option, 116
 tstep
 command line option, 93
 Tsurf
 command line option, 116

U

U
 command line option, 95
 U10
 command line option, 116
 U_1
 command line option, 116
 U_10
 command line option, 116
 U_11
 command line option, 116
 U_12
 command line option, 116
 U_13
 command line option, 116
 U_14
 command line option, 116
 U_15
 command line option, 117
 U_16
 command line option, 117
 U_17
 command line option, 117
 U_18
 command line option, 117
 U_19
 command line option, 117
 U_2
 command line option, 117
 U_20
 command line option, 117
 U_21
 command line option, 117
 U_22
 command line option, 117
 U_23
 command line option, 117
 U_24
 command line option, 117
 U_25
 command line option, 117
 U_26
 command line option, 118
 U_27
 command line option, 118
 U_28
 command line option, 118
 U_29

- command line option, 118
- U_3
 - command line option, 118
- U_30
 - command line option, 118
- U_4
 - command line option, 118
- U_5
 - command line option, 118
- U_6
 - command line option, 118
- U_7
 - command line option, 118
- U_8
 - command line option, 118
- U_9
 - command line option, 118

V

- veg_type
 - command line option, 93

W

- waterdist
 - command line option, 93
- waterusemethod
 - command line option, 93
- wdir
 - command line option, 96
- wetthresh
 - command line option, 93
- WU_DecTr1
 - command line option, 119
- WU_DecTr2
 - command line option, 119
- WU_DecTr3
 - command line option, 119
- WU_EveTr1
 - command line option, 119
- WU_EveTr2
 - command line option, 119
- WU_EveTr3
 - command line option, 119
- WU_Grass1
 - command line option, 119
- WU_Grass2
 - command line option, 119
- WU_Grass3
 - command line option, 120
- WUDecTr
 - command line option, 119
- WUEveTr
 - command line option, 119
- WUGrass

- command line option, 119
- Wuh
 - command line option, 95
- WUInt
 - command line option, 119
- wuprofa_24hr
 - command line option, 94
- wuprofm_24hr
 - command line option, 94

X

- xsmd
 - command line option, 96

Z

- z
 - command line option, 94
- z0m
 - command line option, 125
- z0m_in
 - command line option, 94
- z_1
 - command line option, 125
- z_10
 - command line option, 125
- z_11
 - command line option, 125
- z_12
 - command line option, 125
- z_13
 - command line option, 125
- z_14
 - command line option, 125
- z_15
 - command line option, 125
- z_16
 - command line option, 125
- z_17
 - command line option, 125
- z_18
 - command line option, 125
- z_19
 - command line option, 126
- z_2
 - command line option, 126
- z_20
 - command line option, 126
- z_21
 - command line option, 126
- z_22
 - command line option, 126
- z_23
 - command line option, 126
- z_24

- command line option, [126](#)
- z_25
 - command line option, [126](#)
- z_26
 - command line option, [126](#)
- z_27
 - command line option, [126](#)
- z_28
 - command line option, [126](#)
- z_29
 - command line option, [126](#)
- z_3
 - command line option, [127](#)
- z_30
 - command line option, [127](#)
- z_4
 - command line option, [127](#)
- z_5
 - command line option, [127](#)
- z_6
 - command line option, [127](#)
- z_7
 - command line option, [127](#)
- z_8
 - command line option, [127](#)
- z_9
 - command line option, [127](#)
- zdm
 - command line option, [127](#)
- zdm_in
 - command line option, [94](#)
- Zenith
 - command line option, [120](#)