SuPy Documentation

Release 2019.2.8

Dr Ting Sun and Prof Sue Grimmond

Feb 18, 2019

Contents

> **Caution:** This site is under construction. Some information might NOT be accurate and are subject to rapid change.

- **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.

- **How to get SuPy?**

  SuPy is available on all major platforms (macOS, Windows, Linux) for Python 3.5+ 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.

- **How to contribute to SuPy?**
  - Add your development via Pull Request
  - Report issues via the GitHub page.
  - Provide suggestions and feedback.

CHAPTER 1

Tutorials

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.

```
In [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'
```

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.

```
In [2]: path_runcontrol = Path('../sample_run') / 'RunControl.nml'
```

```
In [3]: df_state_init = sp.init_supy(path_runcontrol)
```

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

```
In [4]: df_state_init.filter(like='method').T
```

```
Out[4]: grid                                 98
        var                         ind_dim
        aerodynamicresistancemethod 0         2
        evapmethod                  0         2
        emissionsmethod             0         2
        netradiationmethod          0         3
        roughlenheatmethod          0         2
        roughlenmommethod           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.

```
In [5]: grid = df_state_init.index[0]
        df_forcing = sp.load_forcing_grid(path_runcontrol, grid)
```

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

```
In [6]: df_state_init, df_forcing = sp.load_SampleData()
```

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:

```
In [30]: df_state_init.loc[:,['bldgh','evetreeh','dectreeh']]
```

```
Out[30]: var      bldgh dectreeh evetreeh
         ind_dim      0        0        0
         grid
         98        22.0     13.1     13.1
```

```
In [7]: df_state_init.filter(like='sfr')
```

```
Out[7]: 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.

```
In [28]: 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}}$C)',
             'RH': r'Relative Humidity (%)',
             'pres': 'Air Pressure (hPa)',
             'rain': 'Rainfall (mm)',
             'U': 'Wind Speed (m $\mathrm{s^{-1}}$)'
         }
         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
         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])
```

1.1.2 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.

```
In [9]: df_output, df_state_final = sp.run_supy(df_forcing, df_state_init)
```

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

```
In [10]: df_output.columns.levels[0]
```

```
Out[10]: Index(['SUEWS', 'snow', '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 staring at the timestamp as in `df_state_final`.

Detailed description of variables in `df_state_final` refers to SuPy output

```
In [11]: df_state_final.T.head()
```

```
Out[11]: grid                                                    98
         datetime                            2012-01-01 00:05:00 2013-01-01 00:05:00
         var                        ind_dim
         aerodynamicresistancemethod 0                       2.0                 2.0
         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
```

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).

Ouptut structure

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

```
In [12]: df_output.head()
```

```
Out[12]: 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                                          ...      DailyState  \
var                               QH        QE ... DensSnow_Paved
grid datetime                                  ...
98   2012-01-01 00:05:00  62.420064  3.576493 ...            NaN
     2012-01-01 00:10:00  61.654096  3.492744 ...            NaN
     2012-01-01 00:15:00  60.885968  3.411154 ...            NaN
     2012-01-01 00:20:00  60.115745  3.331660 ...            NaN
     2012-01-01 00:25:00  59.343488  3.254200 ...            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 218 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.

```
In [13]: df_output_suews = df_output['SUEWS']
```

Statistics Calculation

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

```
In [14]: df_output_suews.loc[:, ['QN', 'QS', 'QH', 'QE', 'QF']].describe()

Out[14]: var                QN             QS             QH             QE  \
         count  105408.000000  105408.000000  105408.000000  105408.000000
         mean       42.574626      -2.128683     101.666596      22.880054
         std       134.685026      83.616791      64.426005      28.535854
         min       -84.389073     -81.747551     -44.370665      -0.649114
```

```
25%      -41.096052   -54.493597    52.976865     1.918672
50%      -24.869018   -43.957916    82.984095    13.057008
75%       77.405862    20.563903   140.933003    31.672138
max      689.067820   387.412149   338.167114   272.755143

var               QF
count  105408.000000
mean       79.047033
std        31.237533
min        26.333882
25%        50.066249
50%        82.896007
75%       104.858241
max       160.076122
```

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`.

```
In [15]: # 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:

```
In [16]: 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')
         ax_output.set_ylabel('Flux ($ \mathrm{W \ m^{-2}}$)')
         ax_output.legend()

Out[16]: <matplotlib.legend.Legend at 0x1263f4f98>
```

More examples

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

```
In [17]: # 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()
```

```
Out[17]: <matplotlib.legend.Legend at 0x12642e8d0>
```

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.

```
In [18]: rsmp_1d = df_output_suews.loc[grid].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.

```
In [19]: # 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()
```

```
Out[19]: <matplotlib.legend.Legend at 0x126dccb70>
```

Then we use the hourly results for other analyses.

```
In [20]: # radiation balance
         ax_output = df_1d_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()
```

```
Out[20]: <matplotlib.legend.Legend at 0x1272d6358>
```

```
In [21]: # water balance
         ax_output = df_1d_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()
```

```
Out[21]: <matplotlib.legend.Legend at 0x127610668>
```

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

```
In [22]: # get a monthly Resampler
         df_plot=df_output_suews.loc[grid].copy()
         df_plot.index=df_plot.index.set_names('Month')
         rsmp_1M = df_plot\
             .shift(-1)\
             .dropna(how='all')\
             .resample('1M', kind='period')
         # mean values
         df_1M_mean = rsmp_1M.mean()
         # sum values
         df_1M_sum = rsmp_1M.sum()
```

```
In [23]: # month names
         name_mon = [x.strftime('%b') for x in rsmp_1M.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
                 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()
```

```
Out[23]: <matplotlib.legend.Legend at 0x127add320>
```

Output

The resampled output can be outputed for a smaller file.

```
In [24]: df_1d_mean.to_csv(
             'suews_1d_mean.txt',
             sep='\t',
             float_format='%8.2f',
             na_rep=-999,
         )
```

For a justified format, we use the `to_string` for better format controlling and write the formatted string out to a file.

```
In [25]: str_out = df_1d_mean.to_string(
             float_format='%8.2f',
             na_rep='-999',
             justify='right',
         )
         with open('suews_sample.txt', 'w') as file_out:
             print(str_out, file=file_out)
```

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 in Parallel Mode

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

```
In [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

        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'
        print('version info:')
        print('supy:', sp.__version__)
        print('supy_driver:', sp.__version_driver__)
```

```
version info:
supy: 2019.2.8
supy_driver: 2018c5
```

```
In [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)
```

Paralell setup for `supy` using `dask`

In addition to the above packages, we also load `dask` to enable `supy` run in a parallel mode. Specifically, we will use `dask.dataframe` <http://docs.dask.org/en/latest/dataframe.html>`__, a specialized `dataframe` extending `pandas.DataFrame`'s ability in parallel operations, to implement a parallel `supy` for the impact studies in this tutorial.

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.

Internally, for a given forcing dataset `df_forcing`, `supy` loops over the grids in a `df_state_init` to conduct simulations. In this case, we can adapt the `df_state_init` to a `dask`-ed version 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.

```
In [5]: 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 : ('10.14.3', ('', '', ''), '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

```
In [6]: # just run for 30 days
        df_forcing_part = df_forcing.iloc[:288*30]
        df_state_init_mgrids = df_state_init.copy()
        # construct a multi-grid `df_state_init`
        for i in range(24-1):
            df_state_init_mgrids = df_state_init_mgrids.append(
                df_state_init, ignore_index=True)
        # perform a serial run
        t0 = time()
        xx = sp.run_supy(df_forcing_part, df_state_init_mgrids)
        t1 = time()
        t_ser = t1-t0
        print(f'Execution time: {t_ser:.2f} s')
```

```
Execution time: 23.06 s
```

Parallel run

```
In [7]: # convert `pandas.DataFrame` to `dask.dataframe` to enable parallelization
        dd_state_init = dd.from_pandas(
```

```
        df_state_init_mgrids,
        npartitions=os.cpu_count()*2)

    # perform a parallel run using `map_partitions`
    t0 = time()
    xx_mp = dd_state_init\
        .map_partitions(
            lambda x: sp.run_supy(df_forcing_part, x)[0],
            meta=df_output)\
        .compute(scheduler='processes')
    t1 = time()
    t_par = t1-t0
    print(f'Execution time: {t_par:.2f} s')
```

```
Execution time: 7.05 s
```

Check the data structure of `xx_mp`:

```
In [6]: xx_mp.head()
```

```
Out[6]: group                           SUEWS                                       \
        var                             Kdown        Kup       Ldown           Lup
        grid datetime
        0    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
        0    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                                              ...      DailyState  \
        var                                   QH        QE ... DensSnow_Paved
        grid datetime                                      ...
        0    2012-01-01 00:05:00  62.420064  3.576493 ...             NaN
             2012-01-01 00:10:00  61.654096  3.492744 ...             NaN
             2012-01-01 00:15:00  60.885968  3.411154 ...             NaN
             2012-01-01 00:20:00  60.115745  3.331660 ...             NaN
             2012-01-01 00:25:00  59.343488  3.254200 ...             NaN

        group                                                                        \
        var                      DensSnow_Bldgs DensSnow_EveTr DensSnow_DecTr
        grid datetime
        0    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
        0    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
   0    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 218 columns]
```

Perform a parallel run using `apply`:

```
In [8]: # perform a parallel run using `apply`
        t0 = time()
        xx_apply = dd_state_init\
            .apply(
                lambda x: sp.run_supy(df_forcing_part, x.to_frame().T)[0],
                axis=1,
                meta=df_output.iloc[0],
            )\
            .compute(scheduler='processes')
        t1 = time()
        t_par = t1 - t0
        print(f'Execution time: {t_par:.2f} s')
```

```
Execution time: 9.98 s
```

Check the data structure of `xx_apply`. Note the difference in resulted data structure between `xx_apply` and `xx_mp`:

```
In [9]: xx_apply.head()
```

```
Out[9]: 0    group                            SUEWS              ...
        1    group                            SUEWS              ...
        2    group                            SUEWS              ...
        3    group                            SUEWS              ...
        4    group                            SUEWS              ...
        Name: (98, 2012-01-01 00:05:00), dtype: object
```

Wrap up the above code into a function for easier use in multi-grid simulations

```
In [10]: # function for multi-grid `run_supy` using map_partitions for better performance
         def run_supy_mgrids(df_state_init_mgrids, df_forcing):
             dd_state_init = dd.from_pandas(
                 df_state_init_mgrids,
                 npartitions=os.cpu_count()*2)
             df_output_mgrids = dd_state_init\
                 .map_partitions(
                     lambda x: sp.run_supy(df_forcing, x)[0],
                     meta=df_output)\
                 .compute(scheduler='processes')
             return df_output_mgrids
```

Benchmark test

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

```
In [10]: # 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]
             print('Sim days:', sim_len / 288)
             print('No. of grids:', df_state_init_m.shape[0])
             # serial run
             print('serial:')
             t0 = time()
             sp.run_supy(df_forcing_part, df_state_init_m)
             t1 = time()
             t_test = t1 - t0
             print(f'Execution time: {t_test:.2f} s')
             #     print()
             dict_time_ser.update({sim_len: t_test})

             # parallel run
             print('parallel:')
             t0 = time()
             run_supy_mgrids(df_state_init_m, df_forcing_part)
             t1 = time()
             t_test = t1 - t0
             print(f'Execution time: {t_test:.2f} s')
             print()
             dict_time_par.update({sim_len: t_test})
```

```
Sim days: 30.0
No. of grids: 12
serial:
Execution time: 10.62 s
parallel:
Execution time: 3.99 s

Sim days: 90.0
No. of grids: 12
```

```
serial:
Execution time: 37.36 s
parallel:
Execution time: 19.63 s

Sim days: 120.0
No. of grids: 12
serial:
Execution time: 51.14 s
parallel:
Execution time: 28.22 s

Sim days: 150.0
No. of grids: 12
serial:
Execution time: 58.08 s
parallel:
Execution time: 35.20 s

Sim days: 180.0
No. of grids: 12
serial:
Execution time: 67.24 s
parallel:
Execution time: 50.90 s

Sim days: 270.0
No. of grids: 12
serial:
Execution time: 97.64 s
parallel:
Execution time: 63.56 s

Sim days: 365.0
No. of grids: 12
serial:
Execution time: 125.39 s
parallel:
Execution time: 66.33 s

Sim days: 730.0
No. of grids: 12
serial:
Execution time: 250.16 s
parallel:
Execution time: 97.39 s

Sim days: 1095.0
No. of grids: 12
serial:
Execution time: 381.80 s
parallel:
Execution time: 147.22 s
```

```
In [11]: df_benchmark = pd.DataFrame([
             dict_time_par,
             dict_time_ser,
         ]).T.rename(columns={
```

```
        0: 'parallel',
        1: 'serial',
    })
    df_benchmark.index = (df_benchmark.index / 288).astype(int).set_names(
        'Length of Simulation Period (day)')
    # df_benchmark.columns.set_names('Execution Time (s)',inplace=True)
    df_benchmark = df_benchmark\
        .assign(
            ratio=df_benchmark['parallel'] / df_benchmark['serial']
        )\
        .rename(columns={'ratio': 'ratio (=p/s, right)'})
    # df_benchmark = df_benchmark.drop(index=[1,7,240])
    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='upper center')

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

    # ax.legend(patches,labels, loc='upper center')
```

```
Out[11]: Text(0, 0.5, 'Execution Ratio (=p/s)')
```

1.2.3 Surface properties: surface albedo

Examine the default albedo values loaded from the sample dataset

```
In [6]: df_state_init.alb
```

```
Out[6]: 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`

```
In [7]: df_state_init_test = df_state_init.copy()
```

Set the `Bldg` land cover to 100% for this study

```
In [8]: df_state_init_test.sfr = 0
        df_state_init_test.loc[:, ('sfr', '(1,)')] = 1
        df_state_init_test.sfr
```

```
Out[8]: 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

```
In [9]: # create a `df_state_init_x` with different surface properties
        n_test = 24
        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`

```
In [14]: df_forcing_part = df_forcing.loc['2012 01':'2012 07']
         df_res_alb_test = run_supy_mgrids(df_state_init_x, df_forcing_part)
```

```
In [15]: df_forcing_part.iloc[[0,-1]]
```

```
Out[15]:                        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
         2012-07-31 23:55:00  2012  213  23    55 -999.0 -999.0 -999.0 -999.0 -999.0

                                    U  ...   snow  ldown   fcld    Wuh   xsmd    lai  \
         2012-01-01 00:05:00  4.51500  ... -999.0 -999.0 -999.0 -999.0 -999.0 -999.0
         2012-07-31 23:55:00  3.46125  ... -999.0 -999.0 -999.0 -999.0 -999.0 -999.0

                              kdiff   kdir   wdir  isec
         2012-01-01 00:05:00 -999.0 -999.0 -999.0   0.0
         2012-07-31 23:55:00 -999.0 -999.0 -999.0   0.0

         [2 rows x 25 columns]
```

```
In [19]: df_res_alb_test_x=.head()
```

```
Out[19]: var                     Kdown                                              \
         alb                      0.10      0.13      0.16      0.19      0.22
         datetime
         2012-01-01 00:05:00  0.153333  0.153333  0.153333  0.153333  0.153333
         2012-01-01 00:10:00  0.153333  0.153333  0.153333  0.153333  0.153333
         2012-01-01 00:15:00  0.153333  0.153333  0.153333  0.153333  0.153333
         2012-01-01 00:20:00  0.153333  0.153333  0.153333  0.153333  0.153333
         2012-01-01 00:25:00  0.153333  0.153333  0.153333  0.153333  0.153333

         var                                                                 ...  \
         alb                      0.25      0.28      0.31      0.34      0.37  ...
         datetime                                                            ...
         2012-01-01 00:05:00  0.153333  0.153333  0.153333  0.153333  0.153333  ...
         2012-01-01 00:10:00  0.153333  0.153333  0.153333  0.153333  0.153333  ...
         2012-01-01 00:15:00  0.153333  0.153333  0.153333  0.153333  0.153333  ...
         2012-01-01 00:20:00  0.153333  0.153333  0.153333  0.153333  0.153333  ...
         2012-01-01 00:25:00  0.153333  0.153333  0.153333  0.153333  0.153333  ...

         var                       U10                                              \
         alb                      0.53      0.56      0.59      0.62      0.65
```

```
datetime
2012-01-01 00:05:00  4.504253  4.504254  4.504254  4.504255  4.504255
2012-01-01 00:10:00  4.504464  4.504465  4.504465  4.504466  4.504466
2012-01-01 00:15:00  4.504675  4.504676  4.504676  4.504677  4.504678
2012-01-01 00:20:00  4.504887  4.504887  4.504888  4.504888  4.504889
2012-01-01 00:25:00  4.505098  4.505099  4.505099  4.505100  4.505101

var
alb                      0.68      0.71      0.74      0.77      0.80
datetime
2012-01-01 00:05:00  4.504256  4.504256  4.504257  4.504258  4.504258
2012-01-01 00:10:00  4.504467  4.504467  4.504468  4.504469  4.504469
2012-01-01 00:15:00  4.504678  4.504679  4.504679  4.504680  4.504680
2012-01-01 00:20:00  4.504889  4.504890  4.504891  4.504891  4.504892
2012-01-01 00:25:00  4.505101  4.505102  4.505102  4.505103  4.505103

[5 rows x 1920 columns]
```

In [20]:
```
ind_alb = df_res_alb_test\
    .index\
    .set_levels(list_alb_test, level=0)\
    .set_names('alb', level=0)
df_res_alb_test.index = ind_alb
df_res_alb_test = df_res_alb_test.SUEWS.unstack(0)
df_res_alb_test_july=df_res_alb_test.loc['2012 7']
```

Examine the simulation results

In [21]:
```
df_res_alb_test_july.T2.describe()
```

Out[21]:
```
alb           0.10         0.13         0.16         0.19         0.22  \
count  8928.000000  8928.000000  8928.000000  8928.000000  8928.000000
mean     17.228250    17.221805    17.215353    17.208898    17.202437
std       3.329098     3.324287     3.319480     3.314680     3.309883
min      11.170041    11.169539    11.169037    11.168535    11.168033
25%      15.056289    15.055676    15.052872    15.051240    15.049831
50%      16.567200    16.563258    16.559877    16.558056    16.553386
75%      18.539557    18.528776    18.517567    18.508257    18.505134
max      29.949275    29.906529    29.863626    29.820563    29.777337

alb           0.25         0.28         0.31         0.34         0.37  ...   \
count  8928.000000  8928.000000  8928.000000  8928.000000  8928.000000  ...
mean     17.195971    17.189499    17.183020    17.176535    17.170043  ...
std       3.305090     3.300301     3.295519     3.290738     3.285962  ...
min      11.167531    11.167028    11.166526    11.166024    11.165522  ...
25%      15.047780    15.044683    15.041799    15.039364    15.037256  ...
50%      16.549466    16.548026    16.545437    16.538896    16.533971  ...
75%      18.498565    18.491591    18.483813    18.476377    18.468075  ...
max      29.733945    29.690383    29.646649    29.612174    29.587775  ...

alb           0.53         0.56         0.59         0.62         0.65  \
count  8928.000000  8928.000000  8928.000000  8928.000000  8928.000000
mean     17.135316    17.128783    17.122244    17.115696    17.109142
std       3.260573     3.255821     3.251075     3.246331     3.241592
min      11.162844    11.162341    11.161839    11.161337    11.160835
25%      15.025955    15.023428    15.021041    15.019697    15.017206
50%      16.508681    16.504471    16.497391    16.491112    16.483990
75%      18.426674    18.420512    18.411558    18.405740    18.393170
max      29.455759    29.431154    29.406500    29.381796    29.357040
```

```
alb           0.68         0.71         0.74         0.77         0.80
count  8928.000000  8928.000000  8928.000000  8928.000000  8928.000000
mean     17.102582    17.096015    17.089442    17.082861    17.076274
std       3.236857     3.232125     3.227395     3.222669     3.217943
min      11.160333    11.159830    11.159328    11.158826    11.158323
25%      15.015011    15.010614    15.004378    14.996300    14.993286
50%      16.480801    16.477759    16.472855    16.467660    16.462985
75%      18.384248    18.376006    18.371998    18.363307    18.354587
max      29.332234    29.307376    29.282467    29.257506    29.232492

[8 rows x 24 columns]
```

```
In [ ]: 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 = df_res_alb_T2_diff.columns-df_res_alb_T2_diff.columns[0]
```

```
In [32]: 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)
```

1.2.4 Background climate: air temperature

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

```
In [3]: 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 ($^\degree$C)')
        ax_temp.set_xlabel('Month')
        ax_temp
```

```
Out[3]: <matplotlib.axes._subplots.AxesSubplot at 0x12b4604e0>
```

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.

```
In [4]: # save loaded sample datasets
        df_forcing_part_test = df_forcing.loc['2012 1':'2012 7'].copy()
        df_state_init_test = df_state_init.copy()
```

```
In [5]: # 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:
            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'],
        )
```

```
In [6]: # test the performance of a parallel run
        t0 = time()
        df_res_tairdiff_test = df_res_tairdiff_test0\
            .compute(scheduler='processes')\
            .reset_index('grid', drop=True)
        t1 = time()
        t_par = t1 - t0
        print(f'Execution time: {t_par:.2f} s')
```

```
Execution time: 31.86 s
```

```
In [7]: # 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='processes')
            df_output_mclims = df_output_mclims0.reset_index('grid', drop=True)

            return df_output_mclims
```

Construct `dict_df_forcing_x` with multiple forcing `DataFrames`

```
In [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

```
In [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: 126.18 s
```

Examine the results

```
In [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
```

```
In [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')
```

```
In [ ]:
```

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

```
In [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
        %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('retina')
        print('version info:')
        print('supy:',sp.__version__)
        print('supy_driver:',sp.__version_driver__)
```

```
version info:
supy: 2019.2.8
supy_driver: 2018c5
```

run SUEWS with default settings

```
In [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']
```

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:

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

$$where : math : `T_C`(: math : `T_H`)isthecooling(heating)threshold$$

temperature of buildings, $_B$ ($_B$) is the building cooling (heating) rate, and $_{F0}$ is the baseline anthropogenic heat. The parameters used are: $_C$ ($_H$) set as 20 °C (10 °C), $_B$ ($_B$) set as 1.5 $\mathrm{W\ m^{-2}\ K^{-1}}$ (3 $\mathrm{W\ m^{-2}\ K^{-1}}$) and Q_{F0} is set as 0 $\mathrm{W\ m^{-2}}$, implying other building activities (e.g. lightning, water heating, computers) are zero and therefore do not change the temperature or change with temperature.

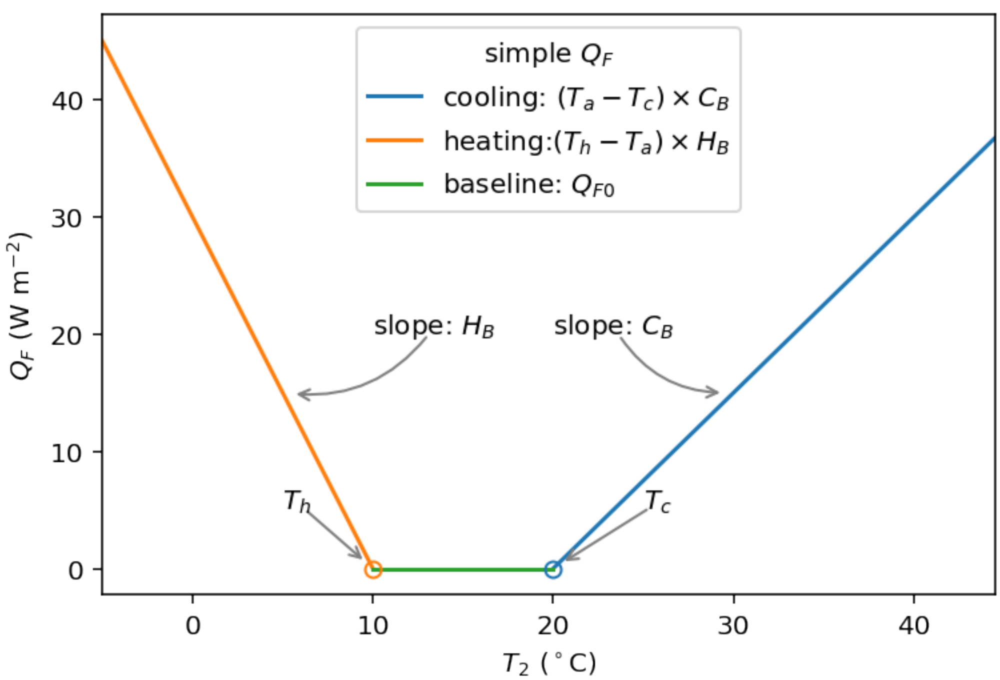
implementation

```
In [3]: 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
```

```
In [4]: 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('$T_2$ ($^\circ$C)')
        ax_qf_func.set_ylabel('$Q_F$ ($ \mathrm{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="->", #linestyle="dashed",
                                            color="0.5",
                                            shrinkA=5, shrinkB=5,
                                            patchA=None,
                                            patchB=None,
                                            connectionstyle='arc3',
                                            ),
                            )

        ax_qf_func.annotate("$T_h$",
                            xy=(10, 0), xycoords='data',
                            xytext=(5, 5), textcoords='data',
                            arrowprops=dict(arrowstyle="->", #linestyle="dashed",
```

```
                                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="->", #linestyle="dashed",
                                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="->", #linestyle="dashed",
                                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.

```
In [5]: # load extra low-level functions from supy to construct interactive functions
        from supy.supy_post import pack_df_output, pack_df_state
        from supy.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 hearders
                    columns={
                        '%' + 'iy': 'iy',
                        'id': 'id',
                        'it': 'it',
                        'imin': 'imin',
                        'qn': 'qn1_obs',
                        'qh': 'qh_obs',
                        'qe': 'qe',
                        'qs': 'qs_obs',
                        'qf': 'qf_obs',
                        'U': 'avu1',
                        'RH': 'avrh',
                        'Tair': 'temp_c',
                        'pres': 'press_hpa',
                        'rain': 'precip',
                        'kdown': 'avkdn',
                        'snow': 'snow_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 tstep
    t_start = df_forcing_test.index[0]
    # convert df to dict with `itertuples` for better performance
    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)
        t2_ext = dict_output_tstep['dataoutlinesuews'][-3]
        qf_ext = QF_simple(t2_ext)

        dict_output.update({(tstep, grid): dict_output_tstep})
        dict_state.update({(tstep + 1, 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 $\mathrm{W\ m^{-2}}$ 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 $T_{2[2]}$.

spinup run (January to June) for summer simulation

```
In [6]: 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]]
```

spinup run (July to October) for winter simulation

```
In [7]: 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]]
```

coupled simulation

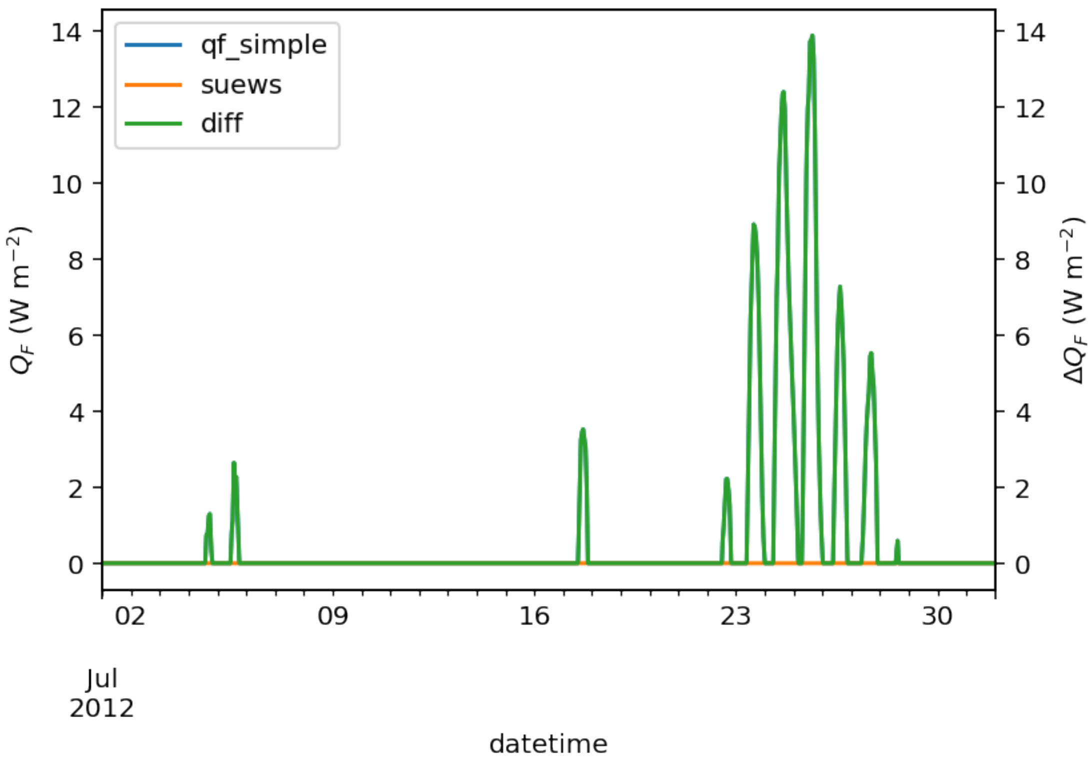
```
In [8]: df_output_test_summer, df_state_summer_test = run_supy_qf(
            df_forcing.loc['2012 7'], df_state_jul_init.copy())
        df_output_test_winter, df_state_winter_test = run_supy_qf(
            df_forcing.loc['2012 12'], df_state_dec_init.copy())
```

examine the results

sumer

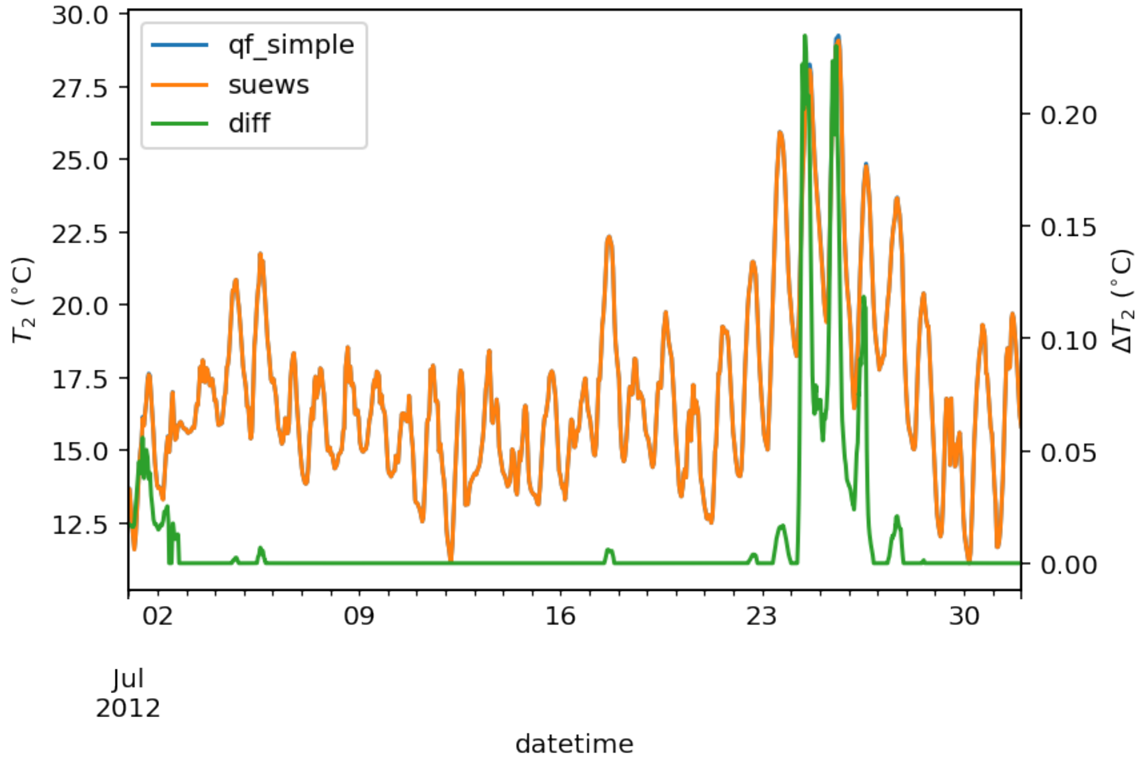
```
In [9]: var = 'QF'
        var_label = '$Q_F$ ($ \mathrm{W \ m^{-2}}$)'
        var_label_right = '$\Delta Q_F$ ($ \mathrm{W \ m^{-2}}$)'
        period = '2012 7'
        df_test = df_output_test_summer
        y1 = df_test.loc[period, var].rename('qf_simple')
        y2 = df_output_def.loc[period, 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')
```

```
Out[9]: <matplotlib.legend.Legend at 0x130455e48>
```

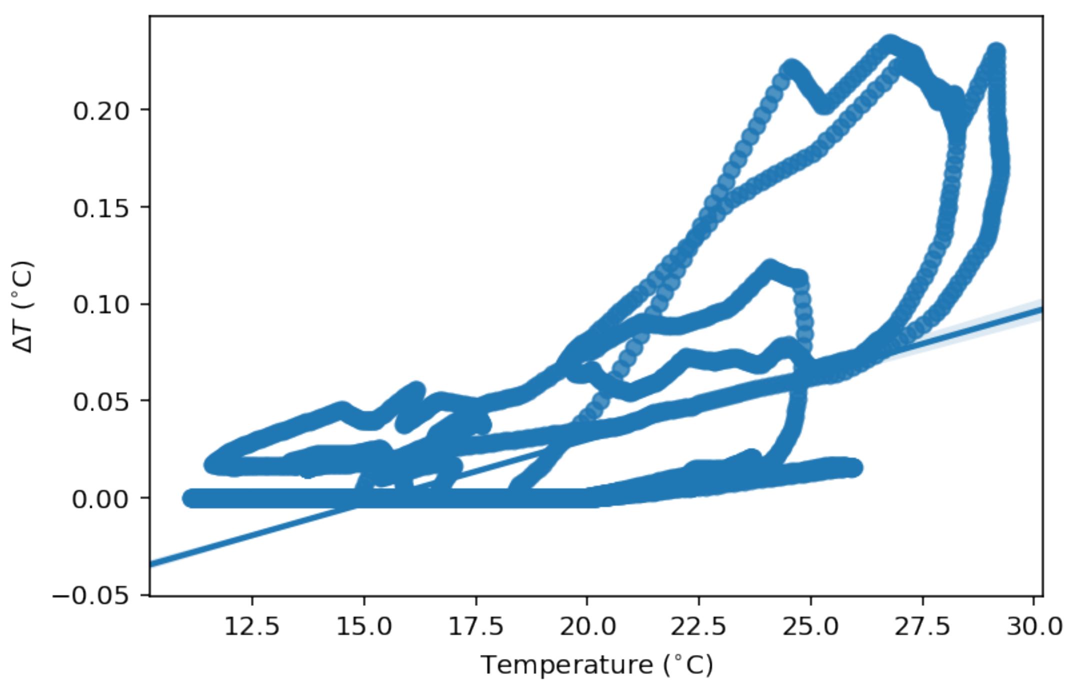



```
In [10]: var = 'T2'
         var_label = '$T_2$ ($^{\circ}$C)'
         var_label_right = '$\Delta T_2$ ($^{\circ}$C)'
         period = '2012 7'
         df_test = df_output_test_summer
         y1 = df_test.loc[period, var].rename('qf_simple')
         y2 = df_output_def.loc[period, var].rename('suews')
         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)
         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')

Out[10]: <matplotlib.legend.Legend at 0x1303b0208>
```



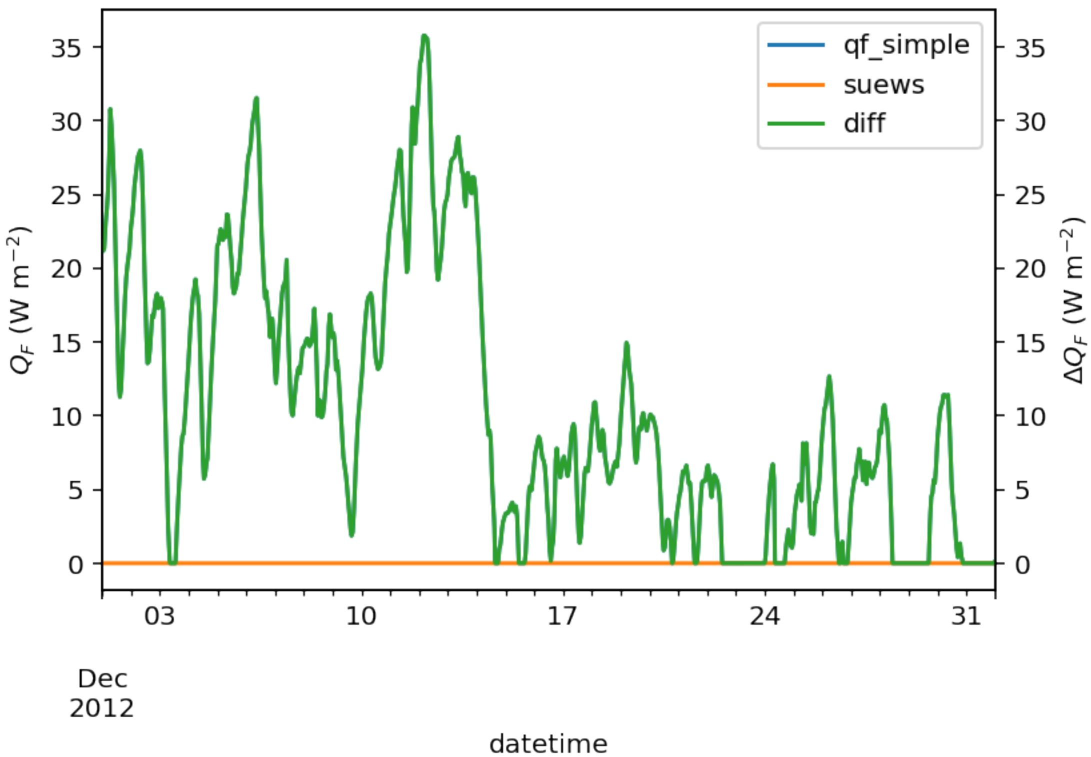

```
In [11]: ax_t2diff = sns.regplot(
             data=df_plot.loc[df_plot['diff'] != 0], x='qf_simple', y='diff')
         ax_t2diff.set_ylabel('$\Delta T$ ($^{\circ}$C)')
         _=ax_t2diff.set_xlabel('Temperature ($^{\circ}$C)')
```



winter

```
In [12]: 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')
```

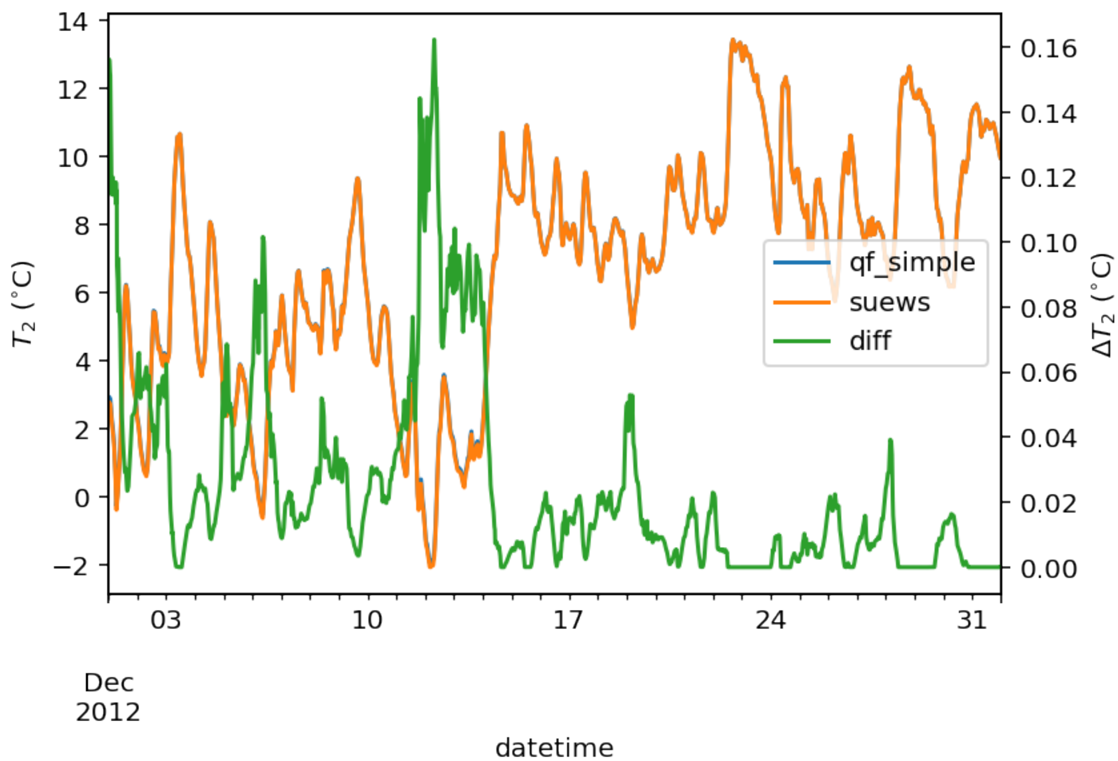
```
Out[12]: <matplotlib.legend.Legend at 0x10cdea828>
```




```
In [13]: 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)
```

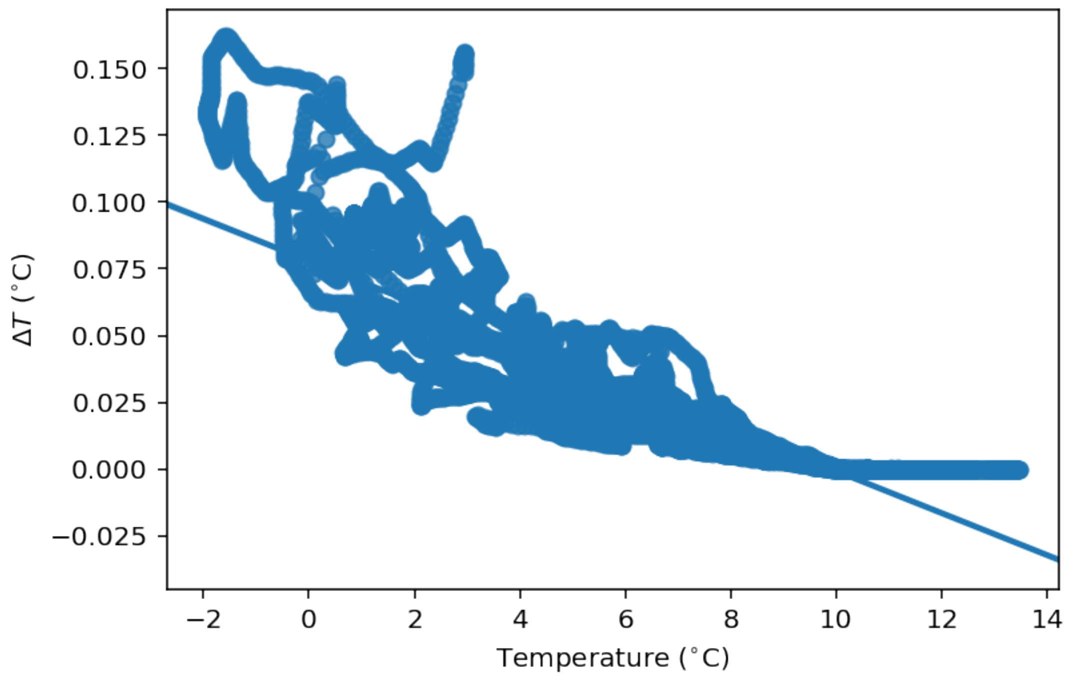
```
    ax.right_ax.set_ylabel(var_label_right)
    lines = ax.get_lines() + ax.right_ax.get_lines()
    ax.legend(lines, [l.get_label() for l in lines], loc='center right')
```

Out[13]: <matplotlib.legend.Legend at 0x141a85940>




```
In [14]: %config InlineBackend.figure_format='retina'
         ax_t2diff = sns.regplot(
             data=df_plot.loc[df_plot['diff'] > 0],
             x='qf_simple', y='diff')
         ax_t2diff.set_ylabel('$\Delta T$ ($^{\circ}$C)')
         ax_t2diff.set_xlabel('Temperature ($^{\circ}$C)')
```

Out[14]: Text(0.5, 0, 'Temperature ($^{\\circ}$C)')



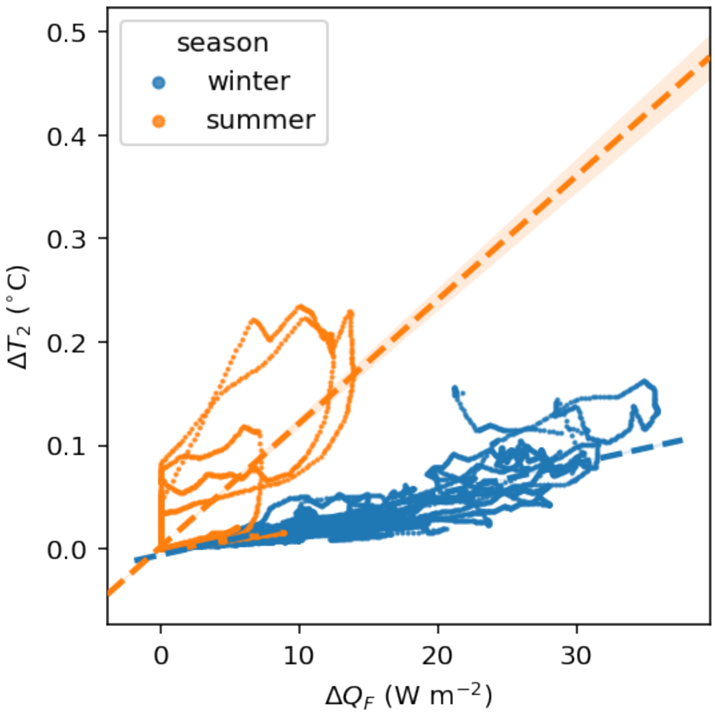
comparison in ΔQ_F-$\Delta T2$ feedback between summer and winter

```
In [15]: # set_matplotlib_formats('retina')
         df_diff_summer = (df_output_test_summer -
                           df_output_def).dropna(how='all', axis=0)
         df_diff_winter = (df_output_test_winter -
                           df_output_def).dropna(how='all', axis=0)
```

```
In [16]: # 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',
         #     col='season',
             hue='season',
             height=4,
             truncate=False,
             markers='o',
             legend_out=False,
             scatter_kws={
         #         'facecolor':'none',
                 's':1,
                 'zorder':0,
                 'alpha':0.8,
             },
             line_kws={
         #         'facecolor':'none',
                 'zorder':6,
```

```
                'linestyle':'--'
            },
        )
        g.set_axis_labels(
            '$\Delta Q_F$ ($ \mathrm{W \ m^{-2}}$)',
            '$\Delta T_2$ ($^{\circ}$C)',
        )
        g.ax.legend(markerscale=4,title='season')
        g.despine(top=False, right=False)
```

```
Out[16]: <seaborn.axisgrid.FacetGrid at 0x13d10d828>
```



The above figure indicate a positive feedback, as Q_F is increased there is an elevated T_2 but with different magnitudes. 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/data-structure/supy-io.ipynb`

CHAPTER 2

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:

```
In [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

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

```
Out[2]: var      aerodynamicresistancemethod ah_min       ah_slope_cooling       \
        ind_dim                            0    (0,)   (1,)             (0,)  (1,)
        grid
```

```
1                                2.0   15.0  15.0             2.7  2.7

var      ah_slope_heating      ahprof_24hr                ...   wuprofm_24hr  \
ind_dim              (0,)  (1,)       (0, 0)  (0, 1)  (1, 0)  ...        (20, 1)
grid                                                      ...
1                     2.7   2.7         0.57    0.65    0.45  ...         -999.0

var                                                               z  z0m_in  zdm_in
ind_dim (21, 0)  (21, 1)  (22, 0)  (22, 1)  (23, 0)  (23, 1)      0       0       0
grid
1        -999.0   -999.0   -999.0   -999.0   -999.0   -999.0   10.0    0.01     0.2

[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.

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

```
Out[3]: ind_dim  (0, 0, 0)  (0, 0, 1)  (0, 0, 2)  (0, 1, 0)  (0, 1, 1)  (0, 1, 2)  \
        grid
        1            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)    ...      (7, 0, 2)  \
        grid                                                   ...
        1            0.719      0.194      -36.6      0.719    ...          -30.0

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

        ind_dim  (7, 3, 0)  (7, 3, 1)  (7, 3, 2)
        grid
        1             0.25        0.6      -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.

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

```
Out[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  ...    snow  ldown   fcld    Wuh   xsmd    lai  \
2012-01-01 00:05:00  4.515  ...  -999.0 -999.0 -999.0 -999.0 -999.0 -999.0
2012-01-01 00:10:00  4.515  ...  -999.0 -999.0 -999.0 -999.0 -999.0 -999.0
2012-01-01 00:15:00  4.515  ...  -999.0 -999.0 -999.0 -999.0 -999.0 -999.0
2012-01-01 00:20:00  4.515  ...  -999.0 -999.0 -999.0 -999.0 -999.0 -999.0
2012-01-01 00:25:00  4.515  ...  -999.0 -999.0 -999.0 -999.0 -999.0 -999.0

                     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]
```

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

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

```
Out[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*.

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

```
Out[6]: group                       SUEWS                                              \
        var                         Kdown     Kup       Ldown         Lup      Tsurf
        grid datetime
        1    2012-01-01 00:05:00  0.153333  0.0184  344.310184  372.270369  11.775916
             2012-01-01 00:10:00  0.153333  0.0184  344.310184  372.270369  11.775916
             2012-01-01 00:15:00  0.153333  0.0184  344.310184  372.270369  11.775916
             2012-01-01 00:20:00  0.153333  0.0184  344.310184  372.270369  11.775916
             2012-01-01 00:25:00  0.153333  0.0184  344.310184  372.270369  11.775916

        group                                                              ...   \
        var                              QN   QF         QS         QH   QE ...
        grid datetime                                                      ...
        1    2012-01-01 00:05:00 -27.825251  0.0 -59.305405  31.480154  0.0 ...
             2012-01-01 00:10:00 -27.825251  0.0 -59.305405  31.480154  0.0 ...
             2012-01-01 00:15:00 -27.825251  0.0 -59.305405  31.480154  0.0 ...
```

```
          2012-01-01 00:20:00 -27.825251  0.0 -59.305405  31.480154  0.0 ...
          2012-01-01 00:25:00 -27.825251  0.0 -59.305405  31.480154  0.0 ...

group                          DailyState                               \
var                        DensSnow_Paved DensSnow_Bldgs DensSnow_EveTr
grid datetime
1    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_DecTr DensSnow_Grass DensSnow_BSoil
grid datetime
1    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_Water  a1  a2  a3
grid datetime
1    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

[5 rows x 218 columns]
```

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

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

```
Out[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`, which facilitates the use 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*.

```
In [8]: df_state_final.head()
```

```
Out[8]: var                        aerodynamicresistancemethod ah_min        \
        ind_dim                                              0    (0,)   (1,)
        grid datetime
        1    2012-01-01 00:05:00                             2    15.0   15.0
             2013-01-01 00:05:00                             2    15.0   15.0

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

```
     2013-01-01 00:05:00              2.7  2.7              2.7  2.7

var                      ahprof_24hr                 ...   wuprofm_24hr  \
ind_dim                       (0, 0)  (0, 1)  (1, 0)  ...        (20, 1)
grid datetime                                        ...
1    2012-01-01 00:05:00        0.57   0.65   0.45  ...         -999.0
     2013-01-01 00:05:00        0.57   0.65   0.45  ...         -999.0

var                                                                        \
ind_dim                  (21, 0) (21, 1) (22, 0) (22, 1) (23, 0) (23, 1)
grid datetime
1    2012-01-01 00:05:00  -999.0  -999.0  -999.0  -999.0  -999.0  -999.0
     2013-01-01 00:05:00  -999.0  -999.0  -999.0  -999.0  -999.0  -999.0

var                          z  z0m_in  zdm_in
ind_dim                      0      0      0
grid datetime
1    2012-01-01 00:05:00  10.0   0.01    0.2
     2013-01-01 00:05:00  10.0   0.01    0.2

[2 rows x 1200 columns]
```

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

CHAPTER 3

API reference

3.1 Top-level Functions

init_supy(path_runcontrol)	Initialise supy by loading initial model states.
load_forcing_grid(path_runcontrol, grid)	Load forcing data for a specific grid included in the index of `df_state_init`.
run_supy(df_forcing, df_state_init[, save_state])	Perform supy simulaiton.
load_SampleData()	Load sample data for quickly starting a demo run.

3.1.1 supy.init_supy

supy.**init_supy**(*path_runcontrol: str*) → pandas.core.frame.DataFrame

Initialise supy by loading initial model states.

Parameters path_runcontrol (*str*) – Path to SUEWS RunControl.nml

Returns df_state_init – Initial model states. See *df_state 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)
```

3.1.2 supy.load_forcing_grid

supy.**load_forcing_grid**(*path_runcontrol: str*, *grid: int*) → 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*) → Tuple[pandas.core.frame.DataFrame, pandas.core.frame.DataFrame]

Perform supy simulaiton.

Parameters

- **df_forcing** (*pandas.DataFrame*) – forcing data.
- **df_state_init** (*pandas.DataFrame*) – initial model states.
- **save_state** (*bool, optional*) – flag for saving model states at each timestep, which can be useful in diagnosing model runtime performance or performing a restart run. (the default is False, which intructs supy not to save runtime model states).

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.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()
```

3.2 Key Data Structures

3.2.1 `df_state` variables

`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 Used for both the radiation and water flow between grids.

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 sensesance 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. [umol m^{-2} 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`

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 k|m^-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 [kg km^{-1}]

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables FcEF_v_kgkm

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 [°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

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) [umol m^{-2} s^{-1}].

Dimensionality (3,)

Dimensionality Remarks 3: { EveTr, DecTr, Grass}

SUEWS-related variables `min_respi`

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`

ohmincqf

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 0

Dimensionality Remarks Scalar

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

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables PrecipLimSnow

preciplimitalb

Description Limit for hourly precipitation when the ground is fully covered with snow. Then snow albedo is reset to AlbedoMax [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 [mm W^{-1} 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

roughlenmommethod

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

Dimensionality 0

Dimensionality Remarks Scalar

SUEWS-related variables `RoughLenMomMethod`

runofftowater

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`

sddfull

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 ΔQS.

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`

tl

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`

3.2.2 `df_forcing` variables

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 [m^3]

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 `NetRadiationMethod` = 0.

qs

Description Storage heat flux [W m^{-2}]

rain

Description Rainfall [mm]

snow

Description Snow [mm]Required if `SnowUse` = 1

wdir

Description Wind direction [°] **Not available in this version.**

xsmd

Description Observed soil moisture [m^3 m^{-3}] or [kg kg^{-1}]

3.2.3 `df_output` variables

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

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 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 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_Bldgs

Description Snow density – building surface [kg m^{-3}]

Group DailyState

DensSnow_Bldgs

Description Snow density - building surface [kg m^{-3}]

Group snow

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_DecTr

Description Snow density – deciduous surface [kg m^{-3}]

Group DailyState

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 snow

DensSnow_EveTr

Description Snow density - evergreen surface [kg m^{-3}]

Group DailyState

DensSnow_Grass

Description Snow density – grass surface [kg m^{-3}]

Group DailyState

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 snow

`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 snow

`DensSnow_Water`

Description Snow density – water surface [kg m^{-3}]

Group snow

`DensSnow_Water`

Description Snow density – water surface [kg m^{-3}]

Group DailyState

`DensSnow_Water`

Description Snow density - water surface [kg m^{-3}]

Group DailyState

`Drainage`

Description Drainage [mm]

Group SUEWS

`Evap`

Description Evaporation [mm]

Group SUEWS

Fc

Description CO2 flux [umol m^{-2} s^{-1}] **Not available in this version.**

Group SUEWS

FcBuild

Description CO2 flux from buildings [umol m^{-2} s^{-1}] **Not available in this version.**

Group SUEWS

FcMetab

Description CO2 flux from metabolism [umol m^{-2} s^{-1}] **Not available in this version.**

Group SUEWS

FcPhoto

Description CO2 flux from photosynthesis [umol m^{-2} s^{-1}] **Not available in this version.**

Group SUEWS

FcRespi

Description CO2 flux from respiration [umol m^{-2} s^{-1}] **Not available in this version.**

Group SUEWS

FcTraff

Description CO2 flux from traffic [umol m^{-2} s^{-1}] **Not available in this version.**

Group SUEWS

Fcld

Description Cloud fraction [-]

Group SUEWS

FlowCh

Description Additional flow into water body [mm]

Group SUEWS

GDD1_g

Description Growing degree days for leaf growth [°C]

Group DailyState

GDD2_s

Description Growing degree days for senescence [°C]

Group DailyState

GDD3_Tmin

Description Daily minimum temperature [°C]

Group DailyState

GDD4_Tmax

Description Daily maximum temperature [°C]

Group DailyState

`GDD5_DLHrs`

Description Day length [h]

Group DailyState

`HDD1_h`

Description Heating degree days [°C]

Group DailyState

`HDD2_c`

Description Cooling degree days [°C]

Group DailyState

`HDD3_Tmean`

Description Average daily air temperature [°C]

Group DailyState

`HDD4_T5d`

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

Group DailyState

`Irr`

Description Irrigation [mm]

Group SUEWS

`Kdown`

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

Group SUEWS

`Kup`

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

Group SUEWS

`LAI`

Description Leaf area index [m 2 m^{-2}]

Group SUEWS

`LAI_DecTr`

Description Leaf area index of deciduous trees [m^{-2} m^{-2}]

Group DailyState

`LAI_EveTr`

Description Leaf area index of evergreen trees [m^{-2} m^{-2}]

Group DailyState

`LAI_Grass`

Description Leaf area index of grass [m^{-2} 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

Lob

Description Obukhov length [m]

Group SUEWS

Lup

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

Group SUEWS

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

QMFreeze

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

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

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

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

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 cofficient a1 - [-]

Group DailyState

a2

Description OHM cofficient a2 [W m^{-2} h^{-1}]

Group DailyState

a3

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

Group DailyState

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

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

z0m

Description Roughness length for momentum [m]

Group SUEWS

zdm

Description Zero-plane displacement height [m]

Group SUEWS

CHAPTER 4

Version History

4.1 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

4.2 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.

4.3 Version 2019.2.8 (under development)

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

Index

D

E

F

G

M

N

O

P

Q

R

S

T

U

V

W

X

Z