

---

# **SuPy Documentation**

***Release 2021.2.26***

**Dr Ting Sun, Dr Hamidreza Omidvar and Prof Sue Grimmond**

**Feb 26, 2021**



<b>1</b>	<b>Tutorials</b>	<b>3</b>
1.1	Quickstart of SuPy . . . . .	3
1.2	Impact Studies Using SuPy . . . . .	20
1.3	Interaction between SuPy and external models . . . . .	25
1.4	Set up SuPy for Your Own Site . . . . .	35
1.5	Python 101 before SuPy . . . . .	55
<b>2</b>	<b>Key IO Data Structures in SuPy</b>	<b>57</b>
2.1	Introduction . . . . .	57
2.2	Input . . . . .	57
2.3	Output . . . . .	60
<b>3</b>	<b>API reference</b>	<b>63</b>
3.1	Top-level Functions . . . . .	63
3.2	Utility Functions . . . . .	67
3.3	Command-Line Tools . . . . .	76
3.4	Key Data Structures . . . . .	76
<b>4</b>	<b>FAQ</b>	<b>135</b>
4.1	I cannot install SuPy following the docs, what is wrong there? . . . . .	135
4.2	How do I know which version of SuPy I am using? . . . . .	136
4.3	A <code>kernel may have died exception</code> happened, where did I go wrong? . . . . .	136
4.4	How can I upgrade SuPy to an up-to-date version? . . . . .	136
<b>5</b>	<b>Version History</b>	<b>137</b>
5.1	Version 2020.11.3 . . . . .	137
5.2	Version 2020.5.29 . . . . .	137
5.3	Version 2020.2.2 . . . . .	138
5.4	Version 2019.8.29 . . . . .	138
5.5	Version 2019.7.17 . . . . .	139
5.6	Version 2019.6.8 . . . . .	139
5.7	Version 2019.5.28 . . . . .	140
5.8	Version 2019.4.29 . . . . .	140
5.9	Version 2019.4.17 . . . . .	140
5.10	Version 2019.4.15 . . . . .	141
5.11	Version 2019.3.21 . . . . .	141
5.12	Version 2019.3.14 . . . . .	142

5.13	Version 2019.2.25 . . . . .	142
5.14	Version 2019.2.24 . . . . .	142
5.15	Version 2019.2.19 . . . . .	143
5.16	Version 2019.2.8 . . . . .	143
5.17	Version 2019.1.1 (preview release, 01 Jan 2019) . . . . .	144
5.18	Version 2018.12.15 (internal test release in December 2018) . . . . .	144
<b>Index</b>		<b>145</b>

- **What is SuPy?**

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

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

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

- **How to get SuPy?**

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

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

- **How to use SuPy?**

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

- **How to contribute to SuPy?**

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



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

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

---

## 1.1 Quickstart of SuPy

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

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

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

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

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

%matplotlib inline
```

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

SuPy versions
-----
supy: 2020.7.1dev
supy_driver: 2020b1
```

(continues on next page)

(continued from previous page)

```
=====
SYSTEM DEPENDENCY

INSTALLED VERSIONS
-----
commit           : None
python           : 3.7.3.final.0
python-bits      : 64
OS               : Darwin
OS-release       : 19.5.0
machine          : x86_64
processor        : i386
byteorder        : little
LC_ALL           : None
LANG             : en_US.UTF-8
LOCALE           : en_US.UTF-8

pandas           : 1.0.3
numpy            : 1.17.5
pytz             : 2019.3
dateutil         : 2.8.1
pip              : 19.3.1
setuptools       : 45.1.0.post20200119
Cython           : None
pytest           : 5.3.1
hypothesis       : None
sphinx           : 3.1.1
blosc            : None
feather          : None
xlsxwriter       : None
lxml.etree       : 4.5.0
html5lib         : None
pymysql          : None
psycopg2         : None
jinja2           : 2.10.3
IPython          : 7.11.1
pandas_datareader: None
bs4              : 4.8.2
bottleneck       : None
fastparquet      : None
gcsfs            : None
lxml.etree       : 4.5.0
matplotlib       : 3.1.2
numexpr          : 2.7.1
odfpy            : None
openpyxl         : None
pandas_gbq       : None
pyarrow          : None
pytables         : None
pytest           : 5.3.1
pyxlsb           : None
s3fs             : None
scipy            : 1.4.1
sqlalchemy       : None
tables           : 3.6.1
tabulate         : 0.8.6
xarray           : 0.14.1
```

(continues on next page)



(continued from previous page)

```
xlrd          : None
xlwt          : None
xlsxwriter    : None
numba         : 0.46.0
```

### 1.1.1 Load input files

#### For existing SUEWS users:

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

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

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

```
2020-07-05 22:59:45,696 - SuPy - INFO - All cache cleared.
```

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

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

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

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

```
[6]: grid = df_state_init.index[0]
df_forcing = sp.load_forcing_grid(path_runcontrol, grid)
# by default, two years of forcing data are included;
# to save running time for demonstration, we only use one year in this demo
df_forcing=df_forcing.loc['2012'].iloc[1:]
```

```
2020-07-05 22:59:47,526 - SuPy - INFO - All cache cleared.
```

#### For new users to SUEWS/SuPy:

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

```
[7]: df_state_init, df_forcing = sp.load_SampleData()
grid = df_state_init.index[0]
# by default, two years of forcing data are included;
# to save running time for demonstration, we only use one year in this demo
df_forcing=df_forcing.loc['2012'].iloc[1:]
```

```
2020-07-05 22:59:50,754 - SuPy - INFO - All cache cleared.
```

## Overview of SuPy input

### `df_state_init`

`df_state_init` includes model Initial state consisting of:

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

Detailed description of variables in `df_state_init` refers to *SuPy input*

Surface land cover fraction information in the sample input dataset:

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

```
[8]: var      bldgh dectreeh evetreeh
ind_dim      0          0          0
grid
1           22.0       13.1       13.1
```

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

```
[9]: var      sfr
ind_dim  (0,)  (1,)  (2,)  (3,)  (4,)  (5,)  (6,)
grid
1           0.43  0.38  0.0  0.02  0.03  0.0  0.14
```

### `df_forcing`

`df_forcing` includes meteorological and other external forcing information.

Detailed description of variables in `df_forcing` refers to *SuPy input*.

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

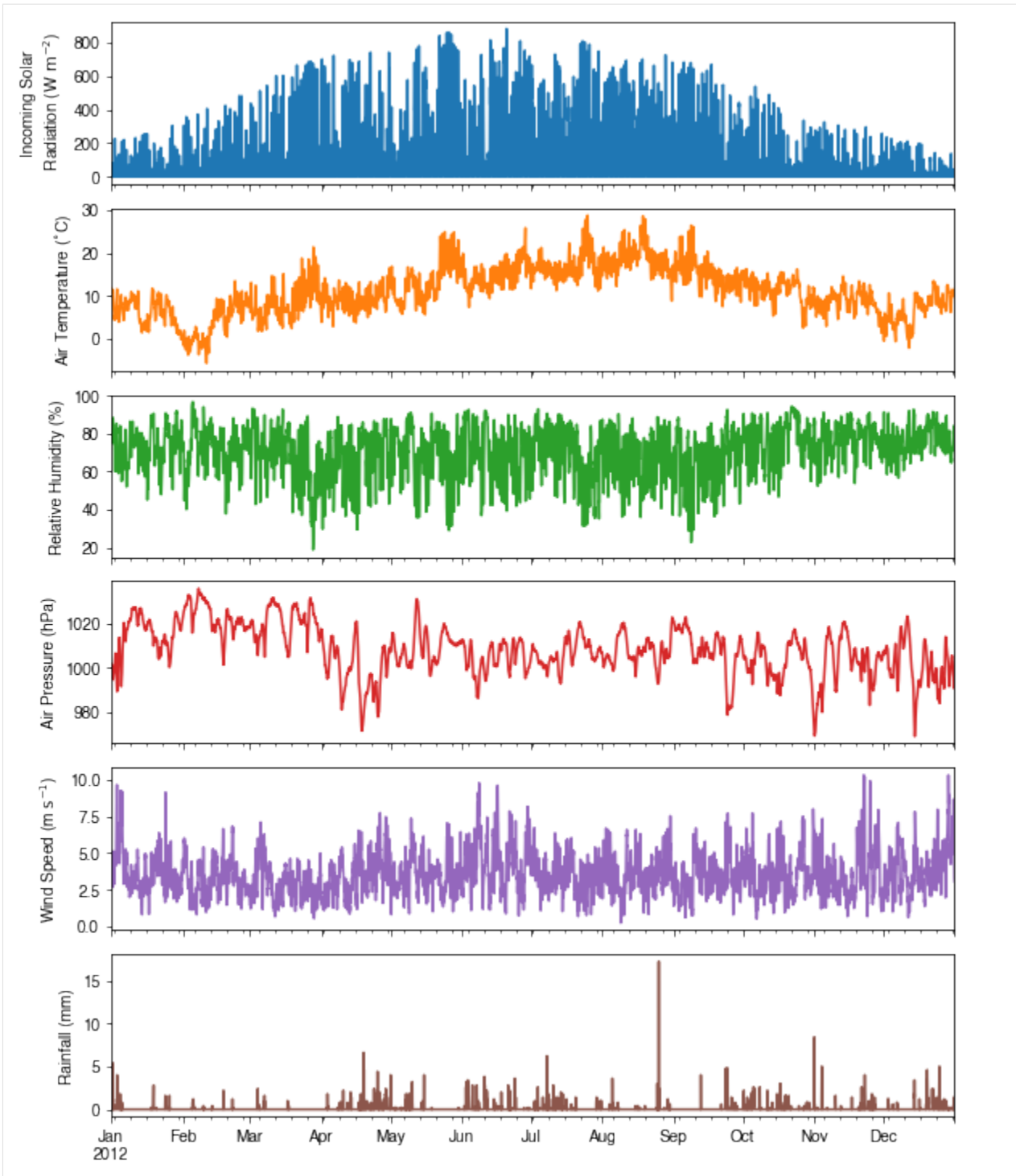
```
[10]: list_var_forcing = [
        "kdown",
        "Tair",
        "RH",
        "pres",
        "U",
        "rain",
    ]
dict_var_label = {
    "kdown": "Incoming Solar\n Radiation ( $\mathrm{W \ m^{-2}}$ )",
    "Tair": "Air Temperature ( $^{\circ}\mathrm{C}$ )",
    "RH": "Relative Humidity (%)",
    "pres": "Air Pressure (hPa)",
    "rain": "Rainfall (mm)",
    "U": "Wind Speed ( $\mathrm{m \ s^{-1}}$ )",
}
df_plot_forcing_x = (
```

(continues on next page)

(continued from previous page)

```
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])
```



### Modification of SuPy input

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

Specifically, for modification, the following operations are essential:

### locating data

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

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

```
[11]: ind_dim  (0,)  (1,)  (2,)  (3,)  (4,)  (5,)  (6,)
grid
1          0.43  0.38  0.0  0.02  0.03  0.0  0.14
```

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

```
[12]: iy          2012.000000
id              1.000000
it              0.000000
imin           10.000000
qn            -999.000000
qh            -999.000000
qe            -999.000000
qs            -999.000000
qf            -999.000000
U               5.176667
RH              86.195000
Tair            11.620000
pres           1001.833333
rain            0.000000
kdown           0.173333
snow           -999.000000
ldown           -999.000000
fcld           -999.000000
Wuh              0.000000
xsmid          -999.000000
lai            -999.000000
kdiff          -999.000000
kdir           -999.000000
wdir           -999.000000
isec              0.000000
Name: 2012-01-01 00:10:00, dtype: float64
```

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

```
[13]: 4.78
```

### setting new values

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

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

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

## 1.1.2 Run simulations

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

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

2020-07-05 22:59:56,659 - SuPy - INFO - =====
2020-07-05 22:59:56,660 - SuPy - INFO - Simulation period:
2020-07-05 22:59:56,660 - SuPy - INFO -   Start: 2012-01-01 00:05:00
2020-07-05 22:59:56,661 - SuPy - INFO -   End: 2012-12-31 23:55:00
2020-07-05 22:59:56,662 - SuPy - INFO -
2020-07-05 22:59:56,662 - SuPy - INFO - No. of grids: 1
2020-07-05 22:59:56,663 - SuPy - INFO - SuPy is running in serial mode
2020-07-05 23:00:15,586 - SuPy - INFO - Execution time: 18.9 s
2020-07-05 23:00:15,587 - SuPy - INFO - =====
```

### `df_output`

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

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

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

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

```
[16]: Index(['SUEWS', 'snow', 'RSL', 'SOLWEIG', '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` (supy may run significantly slower for a large simulation);
2. or, only the final state if `save_state` is set to `False` (the default setting), in which mode `supy` has a similar performance as the standalone compiled SUEWS executable.

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

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

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

### 1.1.3 Examine results

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

#### Ouptut structure

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

```
[18]: df_output.head()
[18]: group
      var      SUEWS
      grid datetime Kdown      Kup      Ldown      Lup \
1      2012-01-01 00:05:00 0.176667 0.021459 344.179805 371.680316
      2012-01-01 00:10:00 0.173333 0.046164 344.190048 372.637243
      2012-01-01 00:15:00 0.170000 0.045271 344.200308 372.715137
      2012-01-01 00:20:00 0.166667 0.044378 344.210586 372.793044
      2012-01-01 00:25:00 0.163333 0.043485 344.220882 372.870963

      group
      var      Tsurf      QN      QF      QS \
1      2012-01-01 00:05:00 11.607207 -27.345303 40.574001 -5.886447
      2012-01-01 00:10:00 11.620000 -28.320026 39.724283 -1.013319
      2012-01-01 00:15:00 11.635000 -28.390100 38.874566 -1.001900
      2012-01-01 00:20:00 11.650000 -28.460168 38.024849 -0.989860
      2012-01-01 00:25:00 11.665000 -28.530232 37.175131 -0.977988

      group
      var      QH      QE      ...      DailyState \
1      2012-01-01 00:05:00 15.276915 -7.777741 ...      DensSnow_Paved
      2012-01-01 00:10:00 -22.518257 -81.748807 ...      NaN
      2012-01-01 00:15:00 -23.450672 -82.273388 ...      NaN
      2012-01-01 00:20:00 -24.350304 -82.818868 ...      NaN
      2012-01-01 00:25:00 -25.191350 -83.410146 ...      NaN

      group
      var      DensSnow_Bldgs DensSnow_EveTr DensSnow_DecTr \
1      2012-01-01 00:05:00 15.276915 -7.777741 ...      NaN
      2012-01-01 00:10:00 -22.518257 -81.748807 ...      NaN
      2012-01-01 00:15:00 -23.450672 -82.273388 ...      NaN
      2012-01-01 00:20:00 -24.350304 -82.818868 ...      NaN
      2012-01-01 00:25:00 -25.191350 -83.410146 ...      NaN
```

(continues on next page)

(continued from previous page)

```

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_Grass DensSnow_BSoil DensSnow_Water  a1  a2
grid datetime
1      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
1      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 371 columns]
```

Here we demonstrate several typical scenarios for SUEWS results examination.

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

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

## Statistics Calculation

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

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

```

[20]: var      QN      QS      QH      QE  \
count  105407.000000  105407.000000  105407.000000  105407.000000
mean      39.375516      5.729435      66.614072      46.798096
std      131.952334      48.981924      71.535234      70.441795
min      -86.331686     -75.287258     -98.890985     -84.805997
25%      -42.635690     -27.871115      20.680393       0.960748
50%      -26.001734     -7.830453      48.672443      14.846743
75%       73.479667      18.009734      91.152469      65.817674
max       679.848644     237.932439     480.602696     532.281922

var      QF
count  105407.000000
mean      79.024549
std      31.231867
min      26.327536
25%      50.058031
50%      82.883410
```

(continues on next page)



(continued from previous page)

75%	104.812507
max	160.023207

## Plotting

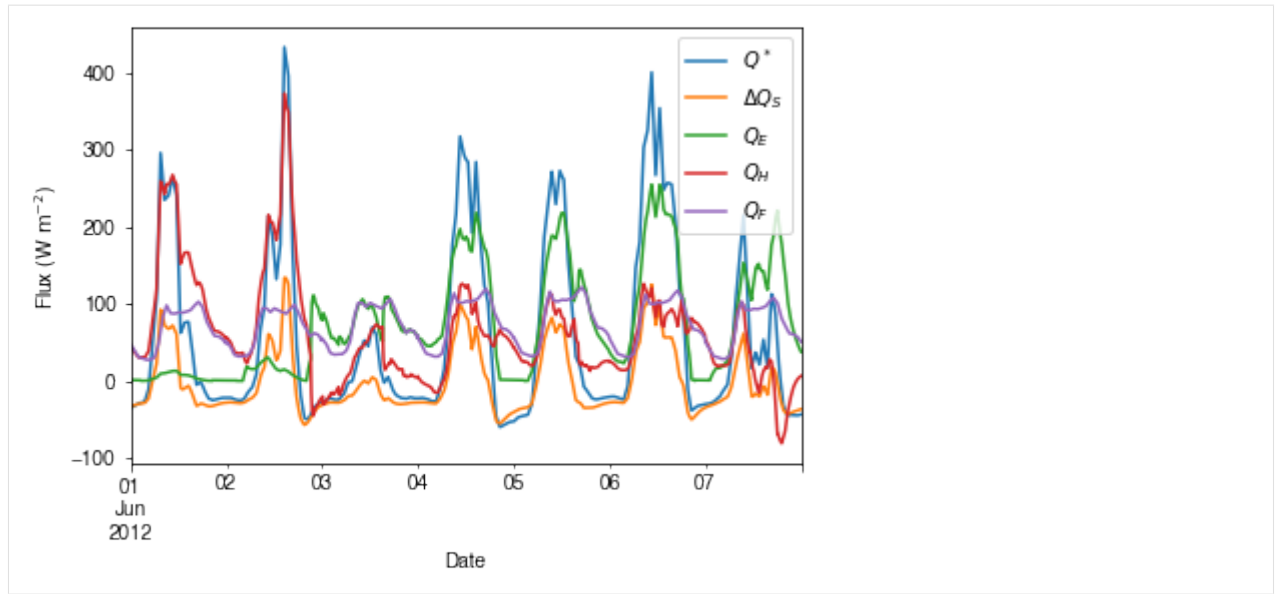
### Basic example

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

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

Quick look at the simulation results:

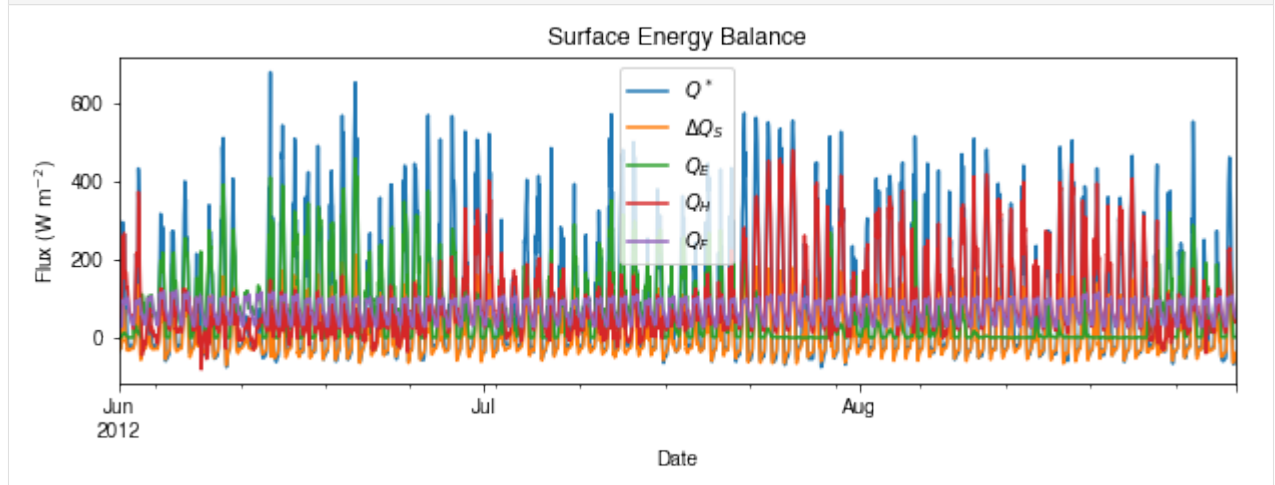
```
[22]: ax_output = df_output_suews\
    .loc[grid]\
    .loc['2012 6 1':'2012 6 7',
        ['QN', 'QS', 'QE', 'QH', 'QF']]\
    .rename(columns=dict_var_disp)\
    .plot()
_ = ax_output.set_xlabel('Date')
_ = ax_output.set_ylabel('Flux ($ \mathrm{W \ m^{-2}}$)')
_ = ax_output.legend()
```



### More examples

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

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



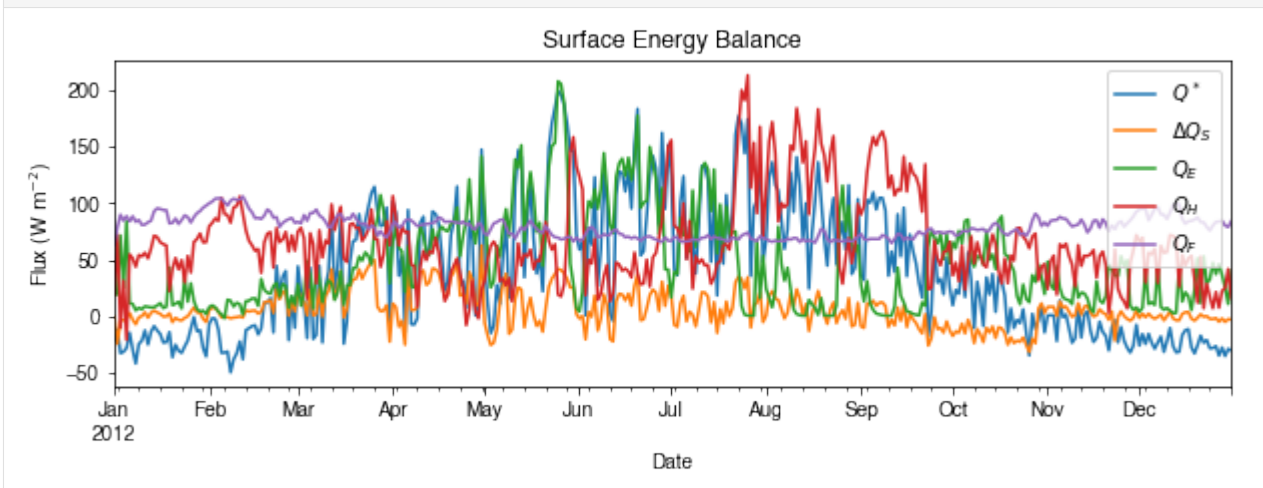
## Resampling

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

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

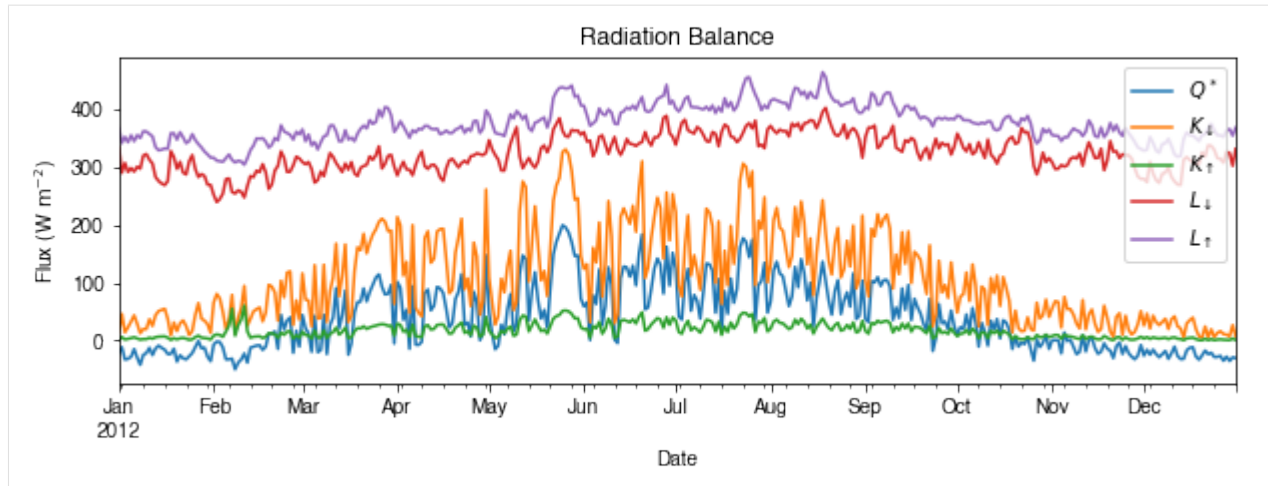
We can then re-examine the above energy balance at hourly scale and plotting will be significantly faster.

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

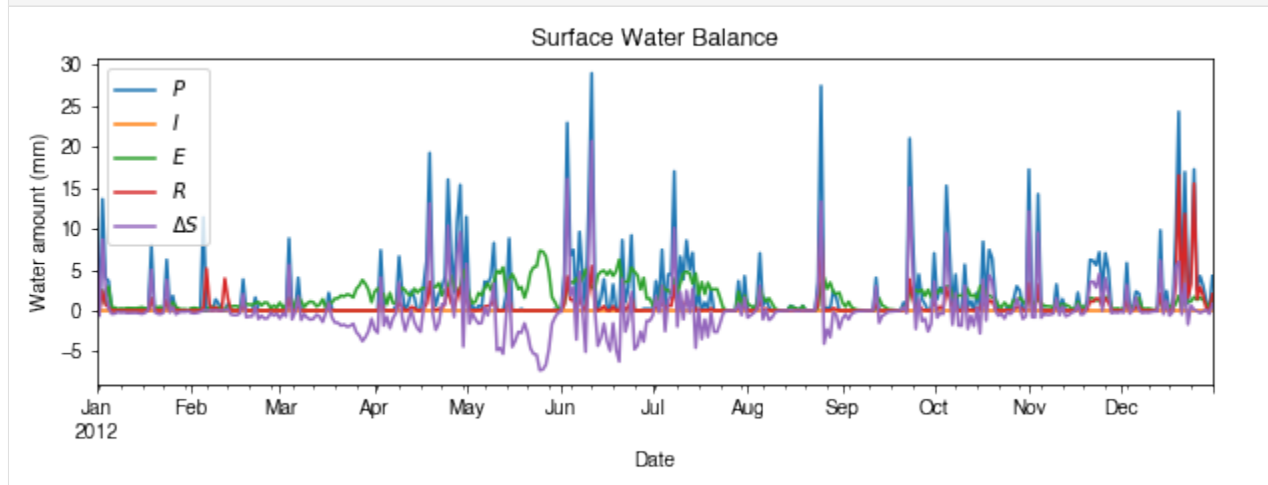


Then we use the hourly results for other analyses.

```
[26]: # 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()
```



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



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

```
[28]: # get a monthly Resampler
df_plot = df_output_suews.loc[grid].copy()
df_plot.index = df_plot.index.set_names("Month")
rsmpl_1M = df_plot.shift(-1).dropna(how="all").resample("1M", kind="period")
# mean values
df_1M_mean = rsmpl_1M.mean()
# sum values
df_1M_sum = rsmpl_1M.sum()
```

```
[29]: # month names
name_mon = [x.strftime("%b") for x in rsmpl_1M.groups]
```

(continues on next page)

(continued from previous page)

```

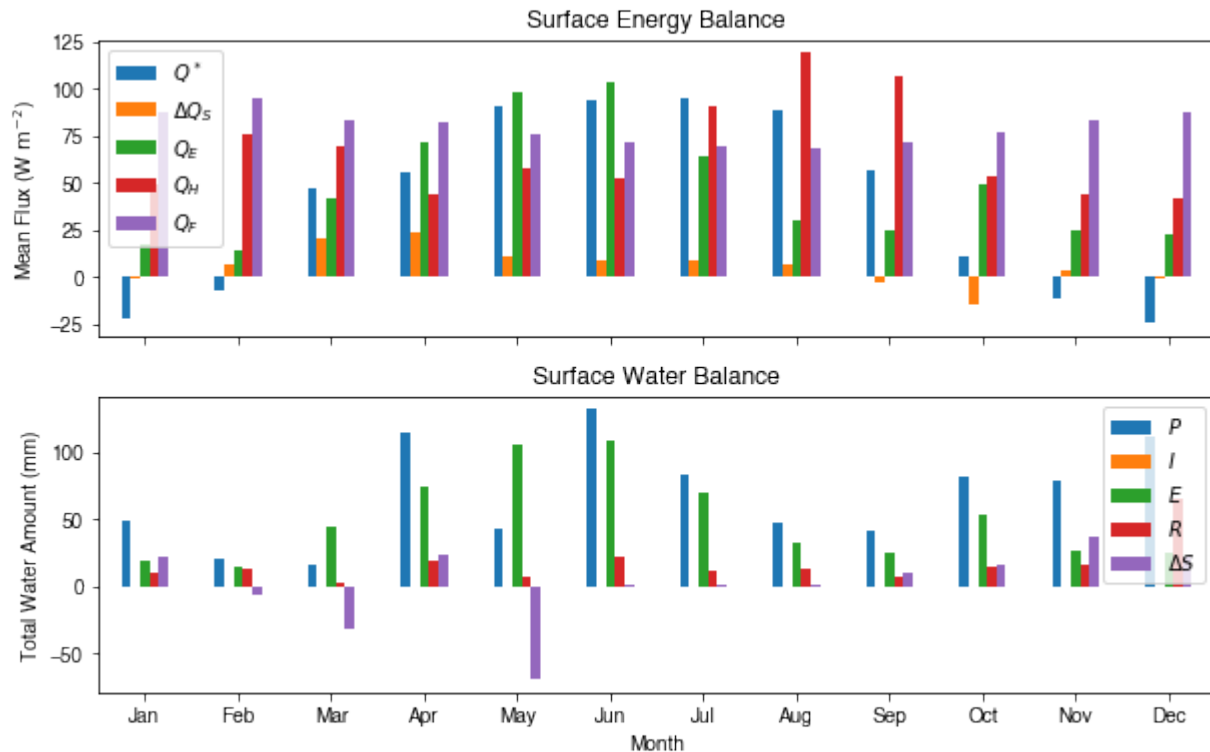
# create subplots showing two panels together
fig, axes = plt.subplots(2, 1, sharex=True)
# surface energy balance
df_1M_mean.loc[:, ["QN", "QS", "QE", "QH", "QF"]].rename(columns=dict_var_disp).plot(
    ax=axes[0], # specify the axis for plotting
    figsize=(10, 6), # specify figure size
    title="Surface 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()

```

[29]: <matplotlib.axes.\_subplots.AxesSubplot at 0x7fb081241128>

[29]: <matplotlib.axes.\_subplots.AxesSubplot at 0x7fb0942eb5f8>



## Output

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

```
[30]: df_output
```

```
[30]: group          SUEWS          \
      var          Kdown          Kup          Ldown          Lup
      grid datetime
1    2012-01-01 00:05:00  0.176667  0.021459  344.179805  371.680316
      2012-01-01 00:10:00  0.173333  0.046164  344.190048  372.637243
      2012-01-01 00:15:00  0.170000  0.045271  344.200308  372.715137
      2012-01-01 00:20:00  0.166667  0.044378  344.210586  372.793044
      2012-01-01 00:25:00  0.163333  0.043485  344.220882  372.870963
      ...
      2012-12-31 23:35:00  0.000000  0.000000  330.263407  363.676342
      2012-12-31 23:40:00  0.000000  0.000000  330.263407  363.676342
      2012-12-31 23:45:00  0.000000  0.000000  330.263407  363.676342
      2012-12-31 23:50:00  0.000000  0.000000  330.263407  363.676342
      2012-12-31 23:55:00  0.000000  0.000000  330.263407  363.676342

      group          \
      var          Tsurf          QN          QF          QS
      grid datetime
1    2012-01-01 00:05:00  11.607207 -27.345303  40.574001 -5.886447
      2012-01-01 00:10:00  11.620000 -28.320026  39.724283 -1.013319
      2012-01-01 00:15:00  11.635000 -28.390100  38.874566 -1.001900
      2012-01-01 00:20:00  11.650000 -28.460168  38.024849 -0.989860
      2012-01-01 00:25:00  11.665000 -28.530232  37.175131 -0.977988
      ...
      2012-12-31 23:35:00  10.140000 -33.412935  53.348682 -4.399144
      2012-12-31 23:40:00  10.140000 -33.412935  52.422737 -4.397669
      2012-12-31 23:45:00  10.140000 -33.412935  51.496792 -4.395831
      2012-12-31 23:50:00  10.140000 -33.412935  50.570847 -4.393681
      2012-12-31 23:55:00  10.140000 -33.412935  46.174492 -4.391264

      group          ...    DailyState \
      var          QH          QE    ... DensSnow_Paved
      grid datetime
1    2012-01-01 00:05:00  15.276915  -7.777741  ...          NaN
      2012-01-01 00:10:00 -22.518257 -81.748807  ...          NaN
      2012-01-01 00:15:00 -23.450672 -82.273388  ...          NaN
      2012-01-01 00:20:00 -24.350304 -82.818868  ...          NaN
      2012-01-01 00:25:00 -25.191350 -83.410146  ...          NaN
      ...
      2012-12-31 23:35:00  2.559974  21.774918  ...          NaN
      2012-12-31 23:40:00  2.178582  21.228889  ...          NaN
      2012-12-31 23:45:00  1.797190  20.682498  ...          NaN
      2012-12-31 23:50:00  1.436708  20.114885  ...          NaN
      2012-12-31 23:55:00 -0.234230  17.387051  ...          100.0

      group          \
      var          DensSnow_Bldgs DensSnow_EveTr DensSnow_DecTr
      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
```

(continues on next page)

(continued from previous page)

```

2012-01-01 00:25:00      NaN      NaN      NaN
...
2012-12-31 23:35:00      NaN      NaN      NaN
2012-12-31 23:40:00      NaN      NaN      NaN
2012-12-31 23:45:00      NaN      NaN      NaN
2012-12-31 23:50:00      NaN      NaN      NaN
2012-12-31 23:55:00    100.0    100.0    100.0

group
var          DensSnow_Grass DensSnow_BSoil DensSnow_Water \
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
...
   2012-12-31 23:35:00      NaN      NaN      NaN
   2012-12-31 23:40:00      NaN      NaN      NaN
   2012-12-31 23:45:00      NaN      NaN      NaN
   2012-12-31 23:50:00      NaN      NaN      NaN
   2012-12-31 23:55:00    100.0    100.0    449.702073

group
var          a1      a2      a3
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
...
   2012-12-31 23:35:00      NaN      NaN      NaN
   2012-12-31 23:40:00      NaN      NaN      NaN
   2012-12-31 23:45:00      NaN      NaN      NaN
   2012-12-31 23:50:00      NaN      NaN      NaN
   2012-12-31 23:55:00    0.36935  0.3242  8.0995

[105407 rows x 371 columns]
```

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

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

```

1_2012_DailyState.txt
1_2012_SUEWS_60.txt
1_2012_snow_60.txt
1_2012_RSL_60.txt
1_2012_SOLWEIG_60.txt
df_state.csv
```

End of doc/tutorial/quick-start.ipynb

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

---

## 1.2 Impact Studies Using SuPy

### 1.2.1 Aim

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

1. surface properties: the physical attributes of land covers (e.g., albedo, water holding capacity, etc.)
2. background climate: longterm meteorological conditions (e.g., air temperature, precipitation, etc.)

#### load `supy` and sample dataset

```
[1]: from dask import dataframe as dd
import supy as sp

import pandas as pd
import numpy as np

from time import time
```

```
[2]: # load sample datasets
df_state_init, df_forcing = sp.load_SampleData()

# by default, two years of forcing data are included;
# to save running time for demonstration, we only use one year in this demo
df_forcing=df_forcing.loc['2012'].iloc[1:]

# perform an example run to get output samples for later use
df_output, df_state_final = sp.run_supy(df_forcing, df_state_init)
```

```
2020-07-06 00:35:30,550 - SuPy - INFO - All cache cleared.
2020-07-06 00:35:33,162 - SuPy - INFO - =====
2020-07-06 00:35:33,163 - SuPy - INFO - Simulation period:
2020-07-06 00:35:33,164 - SuPy - INFO -     Start: 2012-01-01 00:05:00
2020-07-06 00:35:33,164 - SuPy - INFO -     End: 2012-12-31 23:55:00
2020-07-06 00:35:33,165 - SuPy - INFO -
2020-07-06 00:35:33,166 - SuPy - INFO - No. of grids: 1
2020-07-06 00:35:33,166 - SuPy - INFO - SuPy is running in serial mode
2020-07-06 00:35:46,945 - SuPy - INFO - Execution time: 13.8 s
2020-07-06 00:35:46,946 - SuPy - INFO - =====
```

### 1.2.2 Surface properties: surface albedo

#### Examine the default albedo values loaded from the sample dataset

```
[3]: df_state_init.alb
```



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

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

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

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

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

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

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

### Construct a `df_state_init_x` dataframe to perform `supy` simulations with specified albedo

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

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

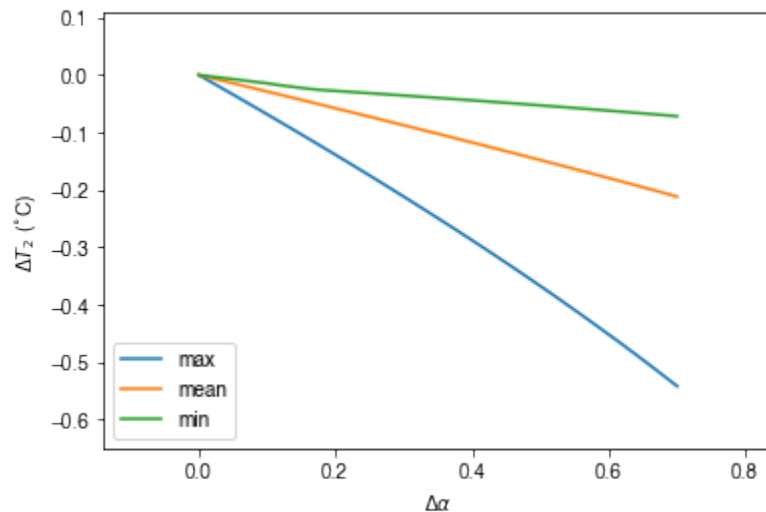
### Conduct simulations with `supy`

```
[7]: df_forcing_part = df_forcing.loc['2012 01':'2012 07']
      df_res_alb_test, df_state_final_x = sp.run_supy(df_forcing_part, df_state_init_x,
      ↪ logging_level=90)
```

### Examine the simulation results

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

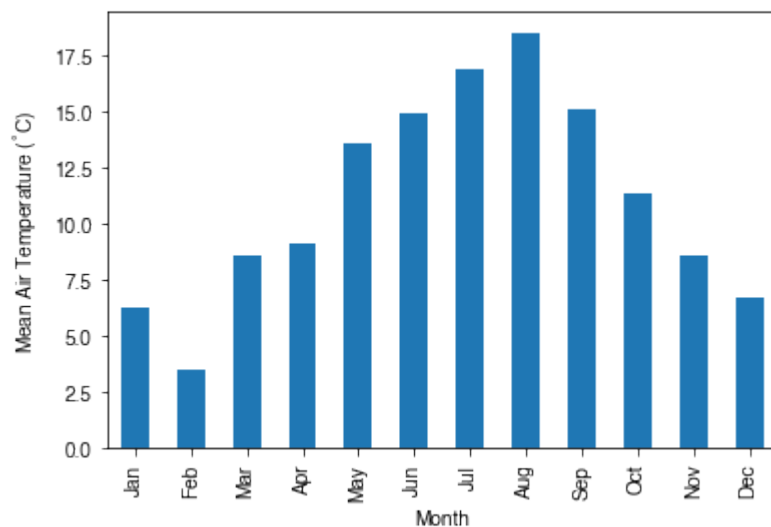
```
[9]: ax_temp_diff = df_res_alb_T2_diff.loc[["max", "mean", "min"]].T.plot()
_ = ax_temp_diff.set_ylabel("$\Delta T_2$ ($^\circ\text{C}$)")
_ = ax_temp_diff.set_xlabel(r"$\Delta\alpha$")
ax_temp_diff.margins(x=0.2, y=0.2)
```



### 1.2.3 Background climate: air temperature

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

```
[10]: df_plot = df_forcing.Tair.loc["2012"].resample("1m").mean()
ax_temp = df_plot.plot.bar(color="tab:blue")
_ = ax_temp.set_xticklabels(df_plot.index.strftime("%b"))
_ = ax_temp.set_ylabel("Mean Air Temperature ($^\circ\text{C}$)")
_ = ax_temp.set_xlabel("Month")
```



**Construct a function to perform parallel supy simulations 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.

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

[13]: from dask import delayed
# 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, logging_level=90)[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'],
)
```

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

Execution time: 29.80 s

```
[15]: # 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, logging_level=90)[0]
        for k, df in dict_df_forcing_x.items()}
    df_output_mclims0 = delayed(pd.concat)(
        dd_forcing_x,
```

(continues on next page)

(continued from previous page)

```

        keys=list(dict_df_forcing_x.keys()),
        names=['clm'],
    ).compute(scheduler='threads')
    df_output_mclims = df_output_mclims0.reset_index('grid', drop=True)

    return df_output_mclims

```

### Construct dict\_df\_forcing\_x with multiple forcing DataFrames

```

[17]: # 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 = 12 # 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

```

[18]: # 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: 183.60 s

### Examine the results

```

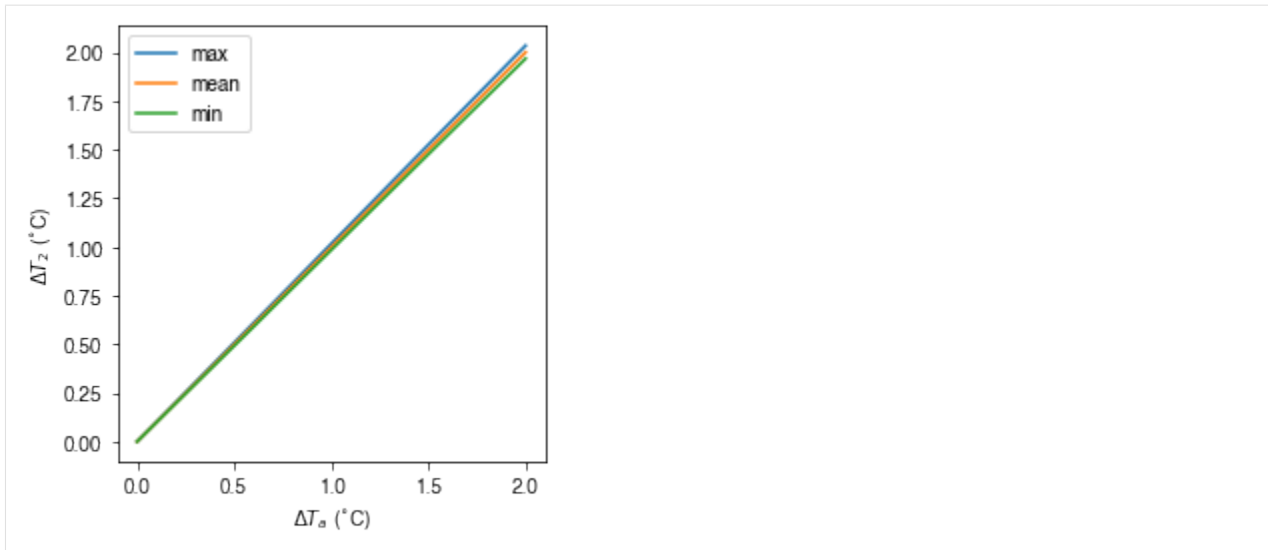
[19]: df_airtemp_test = df_airtemp_test_x.SUEWS.unstack(0)
df_temp_diff = df_airtemp_test.T2.transform(lambda x: x - df_airtemp_test.T2[0.0])
df_temp_diff_ana = df_temp_diff.loc["2012 7"]
df_temp_diff_stat = df_temp_diff_ana.describe().loc[["max", "mean", "min"]].T

```

```

[20]: ax_temp_diff_stat=df_temp_diff_stat.plot()
_=ax_temp_diff_stat.set_ylabel('$\Delta T_2$ ($^{\circ}$C)')
_=ax_temp_diff_stat.set_xlabel('$\Delta T_{(a)}$ ($^{\circ}$C)')
ax_temp_diff_stat.set_aspect('equal')

```



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

End of doc/tutorial/impact-studies.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 highly dependent on outdoor ambient air temperature.

#### load necessary packages

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

%matplotlib inline
# sp.show_version()
```

### run SUEWS with default settings

```
[2]: # load sample run dataset
df_state_init, df_forcing = sp.load_SampleData()
# turn off the snow module as unnecessary at the sample site
df_state_init.loc[:, "snowuse"] = 0

# copy `df_state_init` as the basis for later simulations
df_state_init_def = df_state_init.copy()

# by default, two years of forcing data are included;
# to save running time for demonstration, we only use one year in this demo
df_forcing = df_forcing.loc["2012"].iloc[1:]

# 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"]

2020-07-06 10:55:01,875 - SuPy - INFO - All cache cleared.
2020-07-06 10:55:05,017 - SuPy - INFO - =====
2020-07-06 10:55:05,018 - SuPy - INFO - Simulation period:
2020-07-06 10:55:05,019 - SuPy - INFO -   Start: 2012-01-01 00:05:00
2020-07-06 10:55:05,019 - SuPy - INFO -   End: 2012-12-31 23:55:00
2020-07-06 10:55:05,020 - SuPy - INFO -
2020-07-06 10:55:05,021 - SuPy - INFO - No. of grids: 1
2020-07-06 10:55:05,021 - SuPy - INFO - SuPy is running in serial mode
2020-07-06 10:55:20,390 - SuPy - INFO - Execution time: 15.4 s
2020-07-06 10:55:20,391 - SuPy - INFO - =====
```

## 1.3.2 a simple QF model: `QF_simple`

### model description

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

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

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

## implementation

```
[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
```

Visualise the QF\_simple model:

```
[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('$T_2$ ($^\circ\text{C}$)')
_=ax_qf_func.set_ylabel('$Q_F$ ($\text{W m}^{-2}$)')
_=ax_qf_func.legend(title='simple $Q_F$')
_=ax_qf_func.annotate(
    "$T_C$",
    xy=(20, 0),
    xycoords='data',
    xytext=(25, 5),
    textcoords='data',
    arrowprops=dict(
        arrowstyle="->",
        color="0.5",
        shrinkA=5,
        shrinkB=5,
        patchA=None,
        patchB=None,
        connectionstyle='arc3',
    ),
)

_=ax_qf_func.annotate(
    "$T_H$",
    xy=(10, 0),
    xycoords='data',
    xytext=(5, 5),
    textcoords='data',
    arrowprops=dict(
        arrowstyle="->",
        color="0.5",
        shrinkA=5,
        shrinkB=5,
        patchA=None,
        patchB=None,
        connectionstyle='arc3',
    ),
)
_=ax_qf_func.annotate(
```

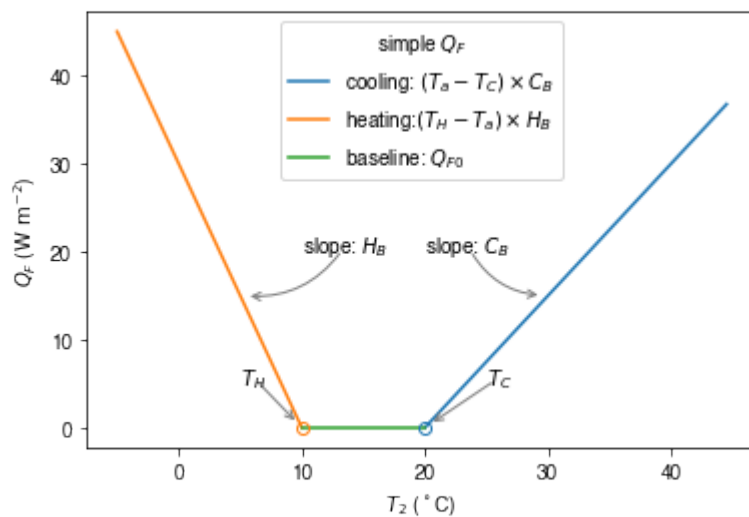
(continues on next page)

(continued from previous page)

```

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

```



### 1.3.3 communication between supy and QF\_simple



### construct a new coupled function

The coupling between the simple  $Q_F$  model and SuPy is done via the low-level function `suews_cal_tstep`, which is an interface function in charge of communications between SuPy frontend and the calculation kernel. By setting SuPy to receive external  $Q_F$  as forcing, at each timestep, the simple  $Q_F$  model is driven by the SuPy output  $T_2$  and provides SuPy with  $Q_F$ , which thus forms a two-way coupled loop.

```
[5]: # load extra low-level functions from supy to construct interactive functions
from supy._post import pack_df_output, pack_df_state
from supy._run import suews_cal_tstep, pack_grid_dict

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

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

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

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

(continues on next page)

(continued from previous page)

```

# starting timestep
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 timestep in df_forcing_test.index:
    # load met forcing at `timestep`
    met_forcing_timestep = dict_forcing[timestep]
    # inject `qf_ext` to `met_forcing_timestep`
    met_forcing_timestep['qf_obs'] = qf_ext

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

    dict_state_end, dict_output_timestep = suews_cal_timestep(
        dict_state_start, met_forcing_timestep)
    # the fourth to the last is `T2` stored in the result array
    t2_ext = dict_output_timestep['dataoutline_suews'][-4]
    qf_ext = QF_simple(t2_ext)

    dict_output.update({(timestep, grid): dict_output_timestep})
    dict_state.update({(timestep + timestep.freq, grid): dict_state_end})

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

```

## simulations for summer and winter months

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

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

```

[6]: df_output_june, df_state_jul = sp.run_supy(
    df_forcing.loc[:, '2012 6'], df_state_init)

2020-07-06 10:55:20,909 - SuPy - INFO - =====
2020-07-06 10:55:20,909 - SuPy - INFO - Simulation period:
2020-07-06 10:55:20,910 - SuPy - INFO - Start: 2012-01-01 00:05:00
2020-07-06 10:55:20,911 - SuPy - INFO - End: 2012-06-30 23:55:00
2020-07-06 10:55:20,911 - SuPy - INFO -
2020-07-06 10:55:20,912 - SuPy - INFO - No. of grids: 1
2020-07-06 10:55:20,913 - SuPy - INFO - SuPy is running in serial mode

```

(continues on next page)

(continued from previous page)

```
2020-07-06 10:55:27,168 - SuPy - INFO - Execution time: 6.3 s
2020-07-06 10:55:27,169 - SuPy - INFO - =====
```

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

```
[7]: df_output_oct, df_state_dec = sp.run_supy(
      df_forcing.loc['2012 7':'2012 11'], df_state_jul)

2020-07-06 10:55:27,176 - SuPy - INFO - =====
2020-07-06 10:55:27,177 - SuPy - INFO - Simulation period:
2020-07-06 10:55:27,178 - SuPy - INFO -   Start: 2012-07-01 00:00:00
2020-07-06 10:55:27,179 - SuPy - INFO -   End: 2012-11-30 23:55:00
2020-07-06 10:55:27,179 - SuPy - INFO -
2020-07-06 10:55:27,180 - SuPy - INFO - No. of grids: 1
2020-07-06 10:55:27,181 - SuPy - INFO - SuPy is running in serial mode
2020-07-06 10:55:33,139 - SuPy - INFO - Execution time: 6.0 s
2020-07-06 10:55:33,140 - SuPy - INFO - =====
```

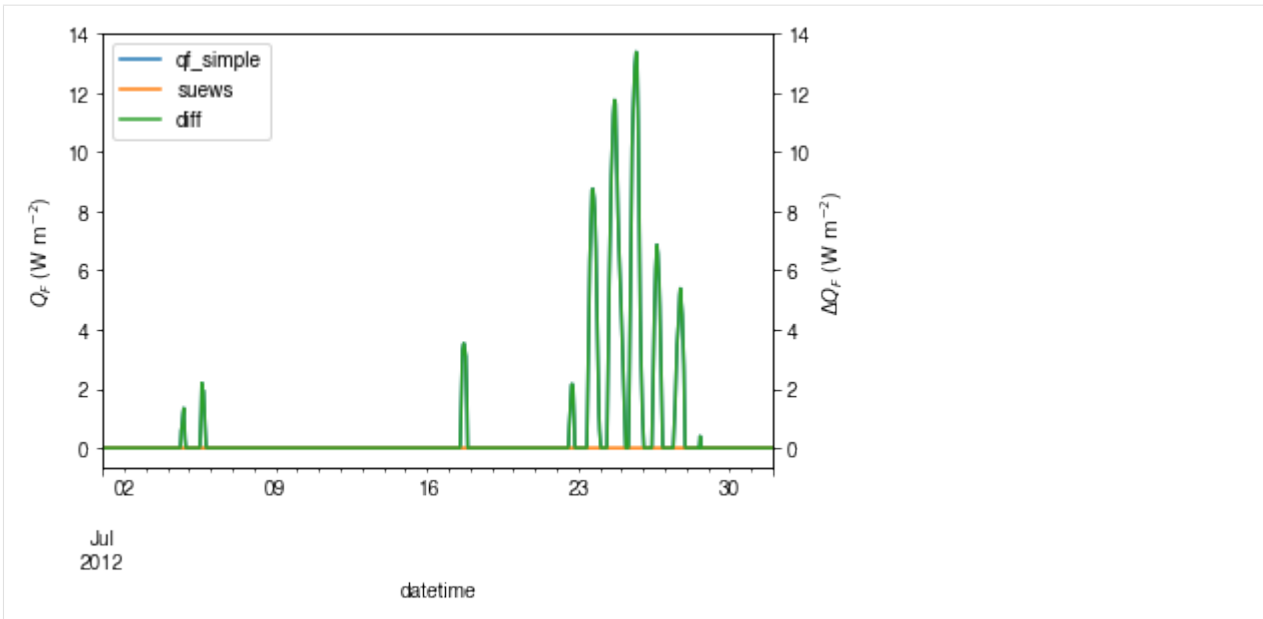
### coupled simulation

```
[8]: df_output_test_summer, df_state_summer_test = run_supy_qf(
      df_forcing.loc["2012-07"], df_state_jul.copy()
    )
df_output_test_winter, df_state_winter_test = run_supy_qf(
      df_forcing.loc["2012-12"], df_state_dec.copy()
    )
```

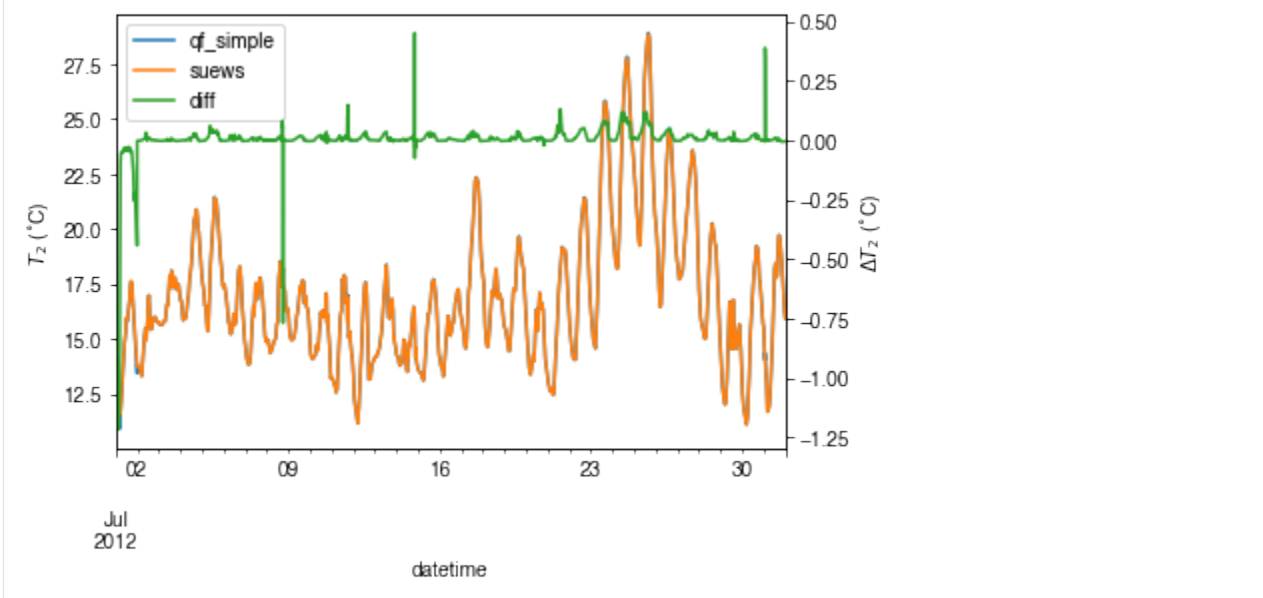
### examine the results

#### sumer

```
[9]: var = "QF"
var_label = "$Q_F$ ($ \mathrm{W} \ m^{-2})$"
var_label_right = "$\Delta Q_F$ ($ \mathrm{W} \ m^{-2})$"
period = "2012-07"
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")
```

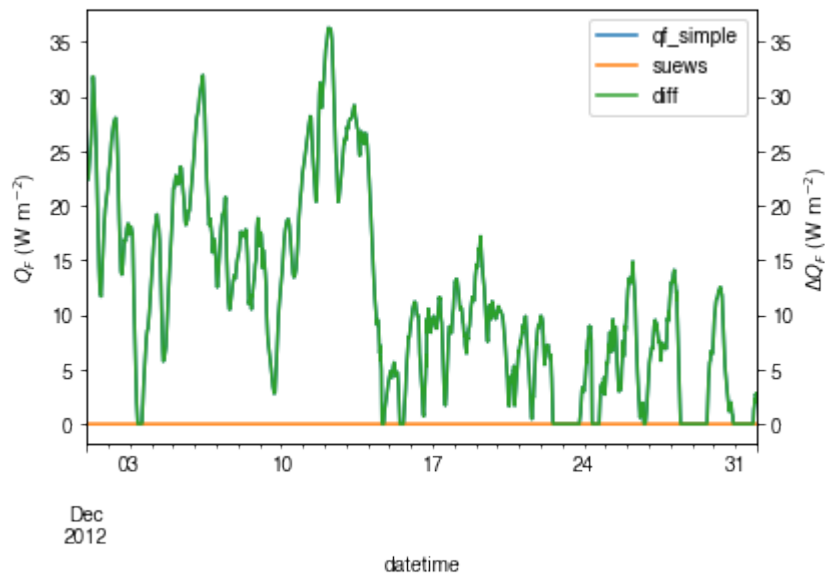


```
[10]: var = "T2"
var_label = "$T_2$ ($^{\circ}$C)"
var_label_right = "$\Delta T_2$ ($^{\circ}$C)"
period = "2012-07"
df_test = df_output_test_summer
y1 = df_test.loc[period, var].rename("qf_simple")
y2 = df_output_def.loc[period, var].rename("suews")
y3 = (y1 - y2).rename("diff")
df_plot = pd.concat([y1, 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")
```

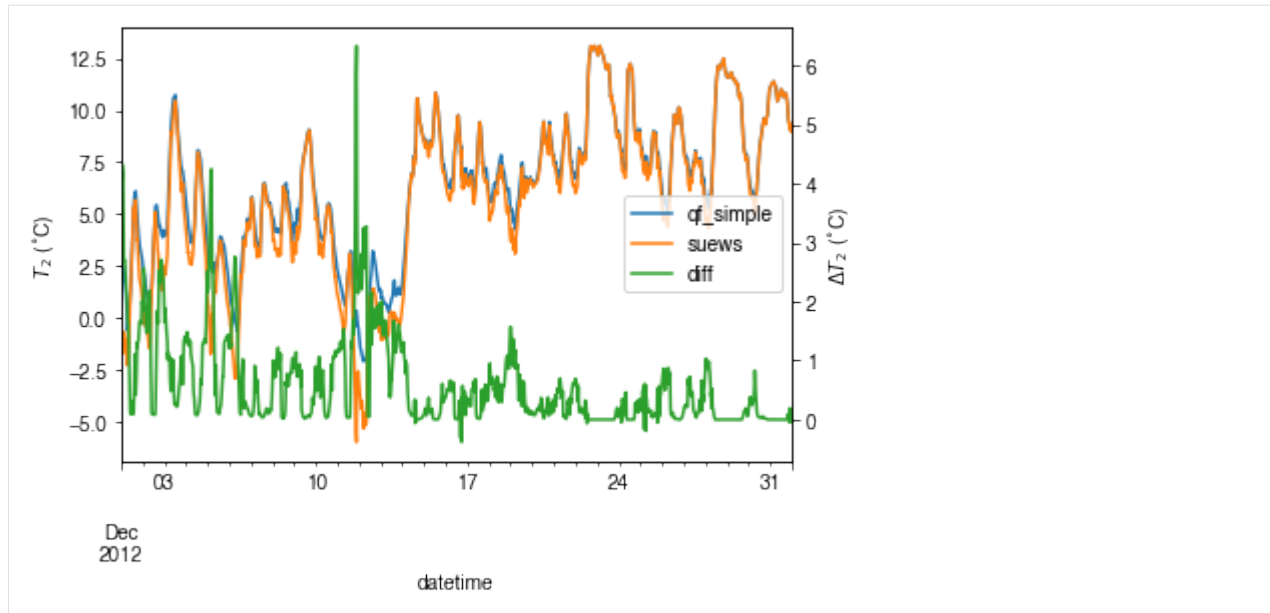


## winter

```
[11]: 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)
_ = ax.right_ax.set_ylabel(var_label_right)
lines = ax.get_lines() + ax.right_ax.get_lines()
_ = ax.legend(lines, [l.get_label() for l in lines], loc="best")
```



```
[12]: var = "T2"
var_label = "$T_2$ ($^{\circ}\text{C}$)"
var_label_right = "$\Delta T_2$ ($^{\circ}\text{C}$)"
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")
```



### comparison in $\Delta Q_F$ - $\Delta T_2$ feedback between summer and winter

```
[13]: # filter results using `where` to choose periods when `QF_simple` is effective
# (i.e. activated by outdoor air temperatures)
df_diff_summer = (
    (df_output_test_summer - df_output_def)
    .where(df_output_def.T2 > 20, np.nan)
    .dropna(how="all", axis=0)
)
df_diff_winter = (
    (df_output_test_winter - df_output_def)
    .where(df_output_test_winter.T2 < 10, np.nan)
    .dropna(how="all", axis=0)
    .loc["20121215":]
)

df_diff_season = pd.concat(
    [df_diff_winter.assign(season="winter"), df_diff_summer.assign(season="summer"),]
).loc[:, ["season", "QF", "T2"]]
g = sns.lmplot(
    data=df_diff_season,
    x="QF",
    y="T2",
    hue="season",
    height=4,
    truncate=False,
    markers="o",
    legend_out=False,
    scatter_kws={"s": 1, "zorder": 0, "alpha": 0.8,},
    line_kws={"zorder": 6, "linestyle": "--"},
)
_ = g.set_axis_labels(
```

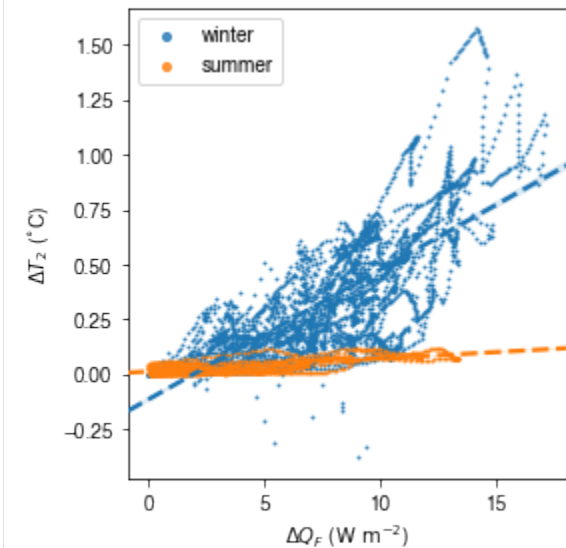
(continues on next page)

(continued from previous page)

```

"$\Delta Q_F$ ($ \mathrm{W \ m^{-2}}$)", "$\Delta T_2$ ($^{\circ}\mathrm{C}$)",
)
_ = g.ax.legend(markerscale=4)
_ = g.despine(top=False, right=False)

```



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

End of doc/tutorial/external-interaction.ipynb

The following section was generated from docs/source/tutorial/setup-own-site.ipynb

## 1.4 Set up SuPy for Your Own Site

This tutorial aims to demonstrate how to set up SuPy for your own site to model the surface energy balance (SEB).

Please note: **SuPy** is a Python-enhanced urban climate model with **SUEWS**, *\*Surface Urban Energy and Water Balance Scheme\**, as its computation core.

**We thus strongly recommend/encourage users to have a good understanding of SUEWS first before diving into the SuPy world.**

In this tutorial, We will use an **AmeriFlux** site **US-AR1** as example:

starting by preparation of input data, we show how to specify site characteristics and choose proper scheme options, then conduct simulations, finally provide some demo figures to help understand the simulation results.

A brief structure is as follows:

1. *Preparing the input data;*
2. *Running a simulation;*

3. *Examination of results*; and
4. *Further exploration*

### 1.4.1 Boilerplate code

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

### 1.4.2 Prepare input data

#### Site-specific configuration of surface parameters

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

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

```
[2]: df_state_init, df_forcing = sp.load_SampleData()
df_state_amf = df_state_init.copy()

2020-07-06 11:24:40,102 - SuPy - INFO - All cache cleared.
```

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

Details for determining the proper values of selected physical parameters can be found [here](#).

#### location

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

#### land cover fraction

Land covers in SUEWS

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



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

## albedo

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

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

## LAI/phenology

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

```
[8]: var      laimax      laimin      laipower      ... \
      ind_dim (0,) (1,) (2,) (0,) (1,) (2,) (0, 0) (0, 1) (0, 2) (1, 0) ...
      grid
      1      5.1  5.5  5.9      4.0  1.0  1.6      0.04  0.04  0.04  0.001 ...

      var      laitype      laicalcyes lai_id
      ind_dim (3, 0) (3, 1) (3, 2) (0,) (1,) (2,)      0 (0,) (1,) (2,)
      grid
      1      0.0015  0.0015  0.0015      1.0  1.0  1.0      1      4.0  1.0  1.6

      [1 rows x 25 columns]
```

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

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

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

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

## surface resistance

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

## measurement height

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

## urban feature

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

## check df\_state

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

2020-07-06 11:24:43,372 - SuPy - INFO - SuPy is validating `df_state`...
2020-07-06 11:24:43,574 - SuPy - INFO - All checks for `df_state` passed!
```

## prepare forcing conditions

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

## load and resample forcing data

---

UMEP workshop users: please note the AMF file path might be DIFFERENT from yours; please set it to the location where your downloaded file is placed.

---

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

(continues on next page)

(continued from previous page)

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

2020-07-06 11:24:44,453 - SuPy - INFO - SuPy is validating `df_forcing`...
2020-07-06 11:24:46,299 - SuPy - ERROR - Issues found in `df_forcing`:
`kdown` should be between [0, 1400] but `-1.298` is found at 2010-01-01 00:05:00
```

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

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

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

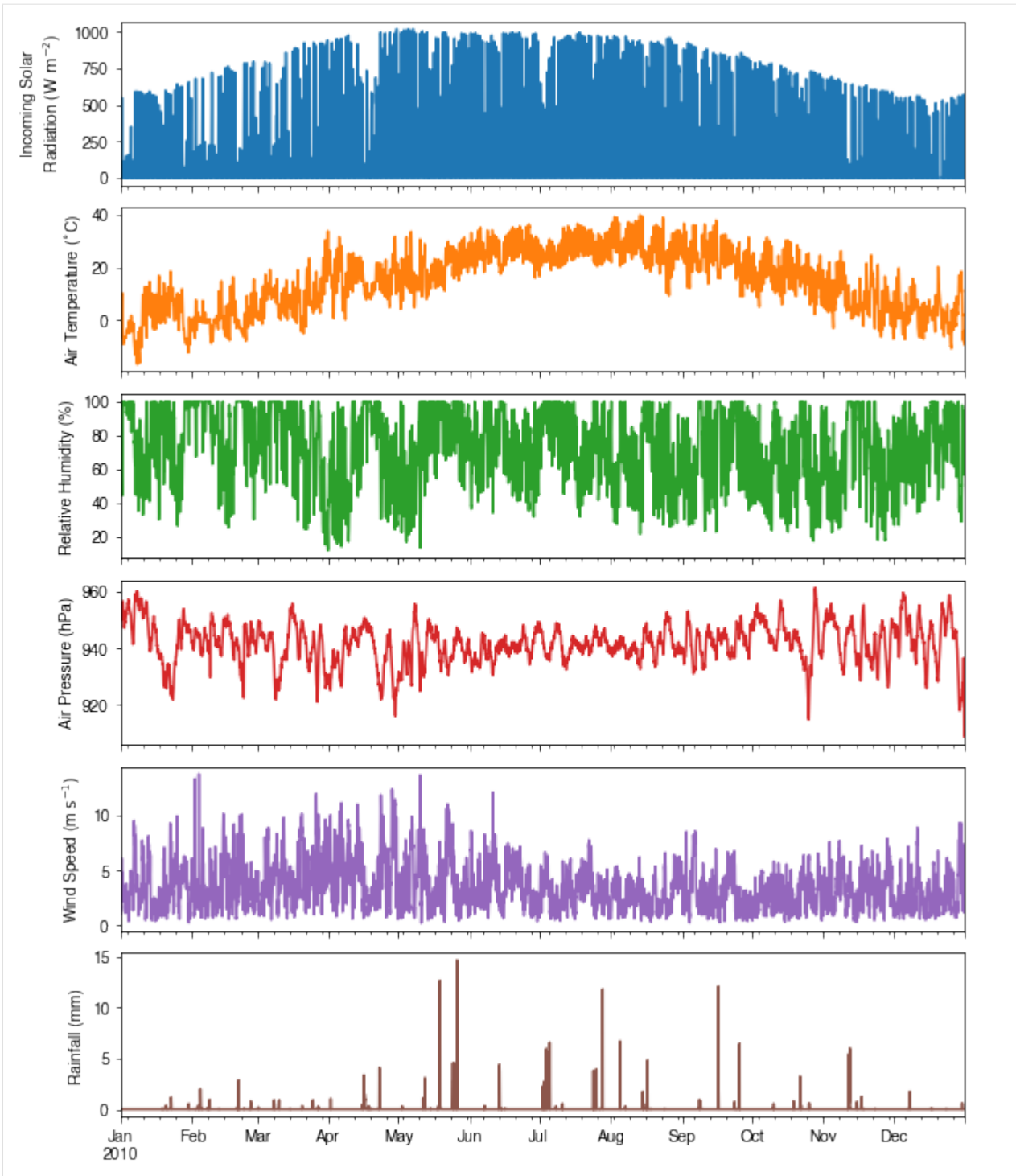
2020-07-06 11:24:46,312 - SuPy - INFO - SuPy is validating `df_forcing`...
2020-07-06 11:24:48,523 - SuPy - INFO - All checks for `df_forcing` passed!
```

## examine forcing data

We can examine the forcing data:

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

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



### 1.4.3 Run simulations

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

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

2020-07-06 11:24:51,973 - SuPy - INFO - =====
2020-07-06 11:24:51,974 - SuPy - INFO - Simulation period:
2020-07-06 11:24:51,975 - SuPy - INFO -   Start: 2010-01-01 00:05:00
2020-07-06 11:24:51,975 - SuPy - INFO -   End: 2011-01-01 00:00:00
2020-07-06 11:24:51,976 - SuPy - INFO -
2020-07-06 11:24:51,977 - SuPy - INFO - No. of grids: 1
2020-07-06 11:24:51,977 - SuPy - INFO - SuPy is running in serial mode
2020-07-06 11:25:01,975 - SuPy - INFO - Execution time: 10.0 s
2020-07-06 11:25:01,976 - SuPy - INFO - =====
```

## df\_output

df\_output is an ensemble output collection of major SUEWS output groups, including:

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

Detailed description of variables in df\_output refers to [SuPy output](#)

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

[20]: Index(['SUEWS', 'snow', 'RSL', 'SOLWEIG', '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 (supy may run significantly slower for a large simulations);
2. or, only the final state if save\_state is set to False (the default setting) in which mode supy has a similar performance as the standalone compiled SUEWS executable.

Entries in df\_state\_final have the same data structure as df\_state\_init and can thus be used for other SUEWS simulations starting at the timestamp as in df\_state\_final.

Detailed description of variables in df\_state\_final refers to [SuPy output](#)

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

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

## 1.4.4 Examine results

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

### Output structure

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

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

```
[22]: group
var          SUEWS
      Kdown  Kup      Ldown      Lup  Tsurf      QN
grid datetime
1  2010-01-01 00:05:00  0.0  0.0  265.492652  305.638434 -1.593 -40.145783
   2010-01-01 00:10:00  0.0  0.0  265.492652  305.638434 -1.593 -40.145783
   2010-01-01 00:15:00  0.0  0.0  265.492652  307.825865 -1.593 -42.333213
   2010-01-01 00:20:00  0.0  0.0  265.492652  307.825865 -1.593 -42.333213
   2010-01-01 00:25:00  0.0  0.0  265.492652  307.825865 -1.593 -42.333213

group
var          QF      QS      QH      QE      ...
grid datetime
1  2010-01-01 00:05:00  0.0 -9.668746 -24.387976  1.284400 ...
   2010-01-01 00:10:00  0.0 -9.424108  -6.676973  1.618190 ...
   2010-01-01 00:15:00  0.0 -0.545992  16.458627  11.833592 ...
   2010-01-01 00:20:00  0.0 -0.536225  15.988621  11.830741 ...
   2010-01-01 00:25:00  0.0 -0.525680  15.537087  11.827934 ...

group
var          DailyState
      DensSnow_Paved DensSnow_Bldgs DensSnow_EveTr
grid datetime
1  2010-01-01 00:05:00      NaN      NaN      NaN
   2010-01-01 00:10:00      NaN      NaN      NaN
   2010-01-01 00:15:00      NaN      NaN      NaN
   2010-01-01 00:20:00      NaN      NaN      NaN
   2010-01-01 00:25:00      NaN      NaN      NaN

group
var          DensSnow_DecTr DensSnow_Grass DensSnow_BSoil
grid datetime
1  2010-01-01 00:05:00      NaN      NaN      NaN
   2010-01-01 00:10:00      NaN      NaN      NaN
   2010-01-01 00:15:00      NaN      NaN      NaN
   2010-01-01 00:20:00      NaN      NaN      NaN
   2010-01-01 00:25:00      NaN      NaN      NaN

group
var          DensSnow_Water  a1  a2  a3
grid datetime
1  2010-01-01 00:05:00      NaN NaN NaN NaN
   2010-01-01 00:10:00      NaN NaN NaN NaN
   2010-01-01 00:15:00      NaN NaN NaN NaN
   2010-01-01 00:20:00      NaN NaN NaN NaN
   2010-01-01 00:25:00      NaN NaN NaN NaN
```

(continues on next page)

(continued from previous page)

```
[5 rows x 371 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](#).

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

## Statistics Calculation

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

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

```
[24]: var          QN          QS          QH          QE          QF
count  105120.000000  105120.000000  105120.000000  105120.000000  105120.0
mean    118.207887    19.047648    38.349672    62.790798    0.0
std     214.335328    61.955598    85.050755    112.585643    0.0
min    -104.566267   -81.170768   -212.925432   -15.483971    0.0
25%    -33.437969   -23.174678   -15.992876    0.341017    0.0
50%    -1.894385    -2.603727    9.862241    3.042328    0.0
75%    248.960723    52.299898    68.130871    65.272384    0.0
max     749.868243    218.450452    414.514498    559.472107    0.0
```

## Plotting

### Basic example

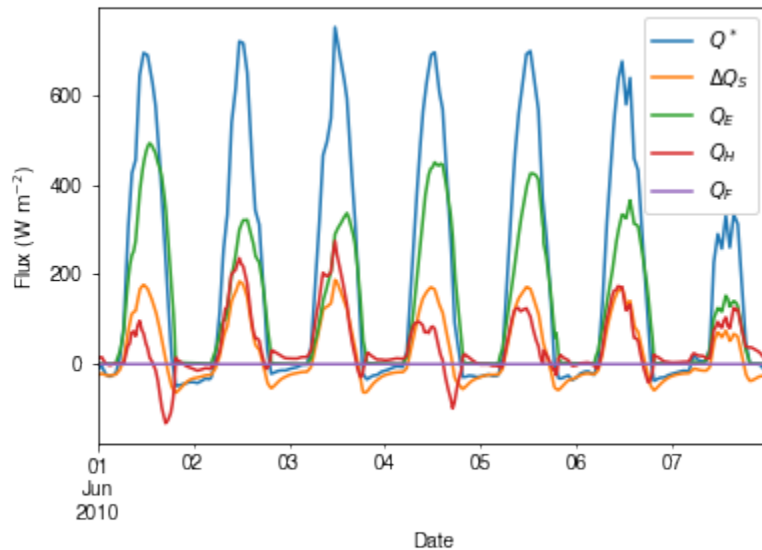
Plotting is very straightforward via the `.plot` method bounded with `pandas.DataFrame`. Note the usage of `loc` for to slices of the output `DataFrame`.

```
[25]: # 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": r"$\Delta S$",
}
```

Peek at the simulation results:

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

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



### Plotting after resampling

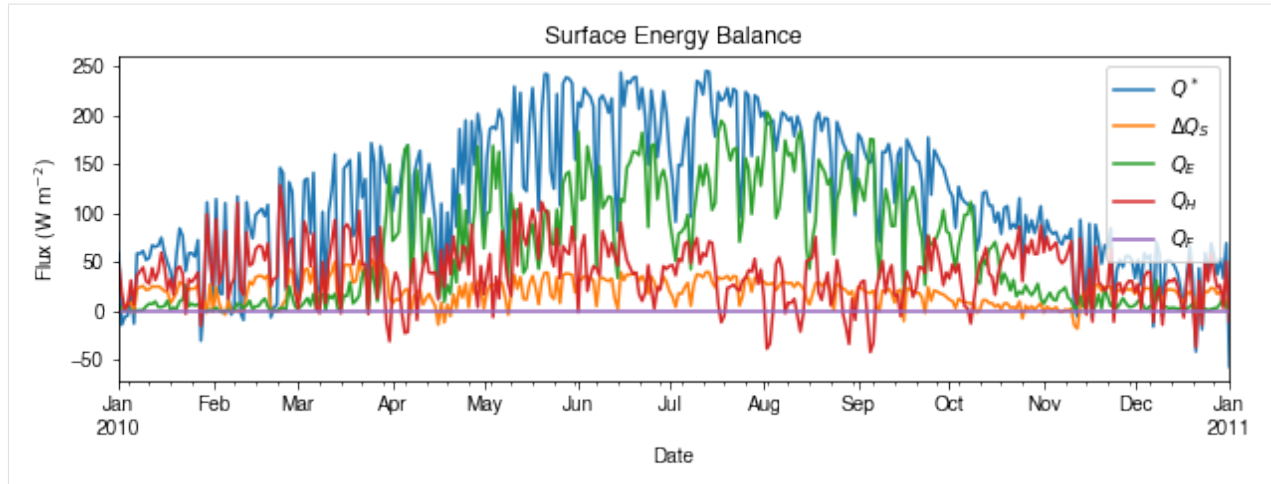
The suggested runtime/simulation frequency of SUEWS is 300 s, which usually results in a large output and may be over-weighted for storage and analysis. Also, you may feel an apparent slowdown in producing the above figure as a large amount of data were used for the plotting. To slim down the result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

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

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

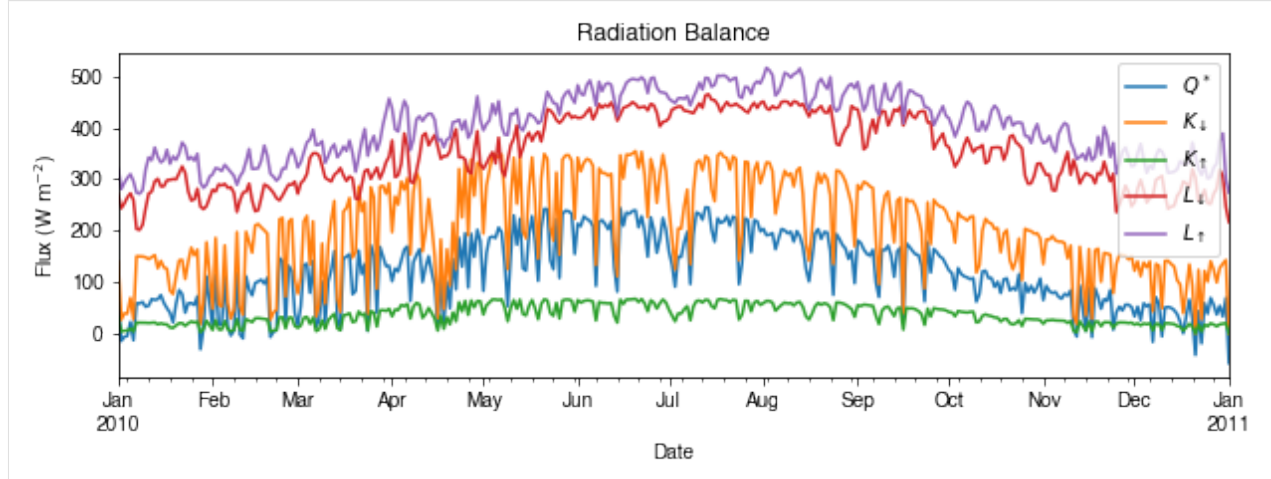
```
[29]: # energy balance
ax_output = (
    df_1d_mean.loc[:, ["QN", "QS", "QE", "QH", "QF"]]
    .rename(columns=dict_var_disp)
    .plot(figsize=(10, 3), title="Surface Energy Balance",)
)
_ = ax_output.set_xlabel("Date")
_ = ax_output.set_ylabel("Flux ($ \mathrm{W \ m^{-2}})$")
_ = ax_output.legend()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



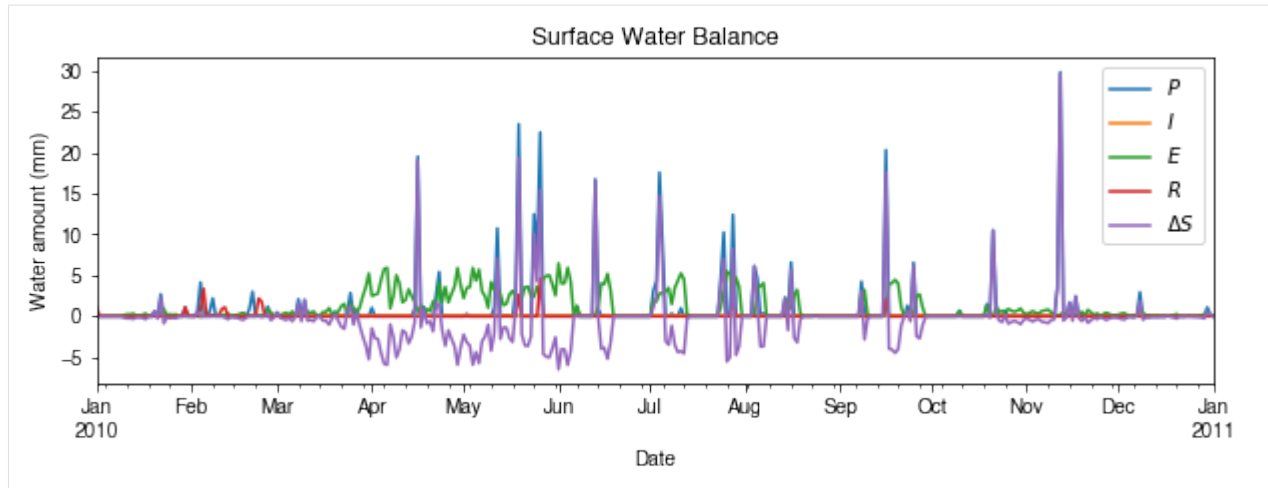


Then we use the hourly results for other analyses.

```
[30]: # 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 ( $\text{W m}^{-2}$ )")
_ = ax_output.legend()
```



```
[31]: # 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()
```



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

```
[32]: # get a monthly Resampler
df_plot = df_output_suews.copy()
df_plot.index = df_plot.index.set_names("Month")
rsmpl_1M = df_plot.shift(-1).dropna(how="all").resample("1M", kind="period")
# mean values
df_1M_mean = rsmpl_1M.mean()
# sum values
df_1M_sum = rsmpl_1M.sum()
```

```
[33]: # month names
name_mon = [x.strftime("%b") for x in rsmpl_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()
```

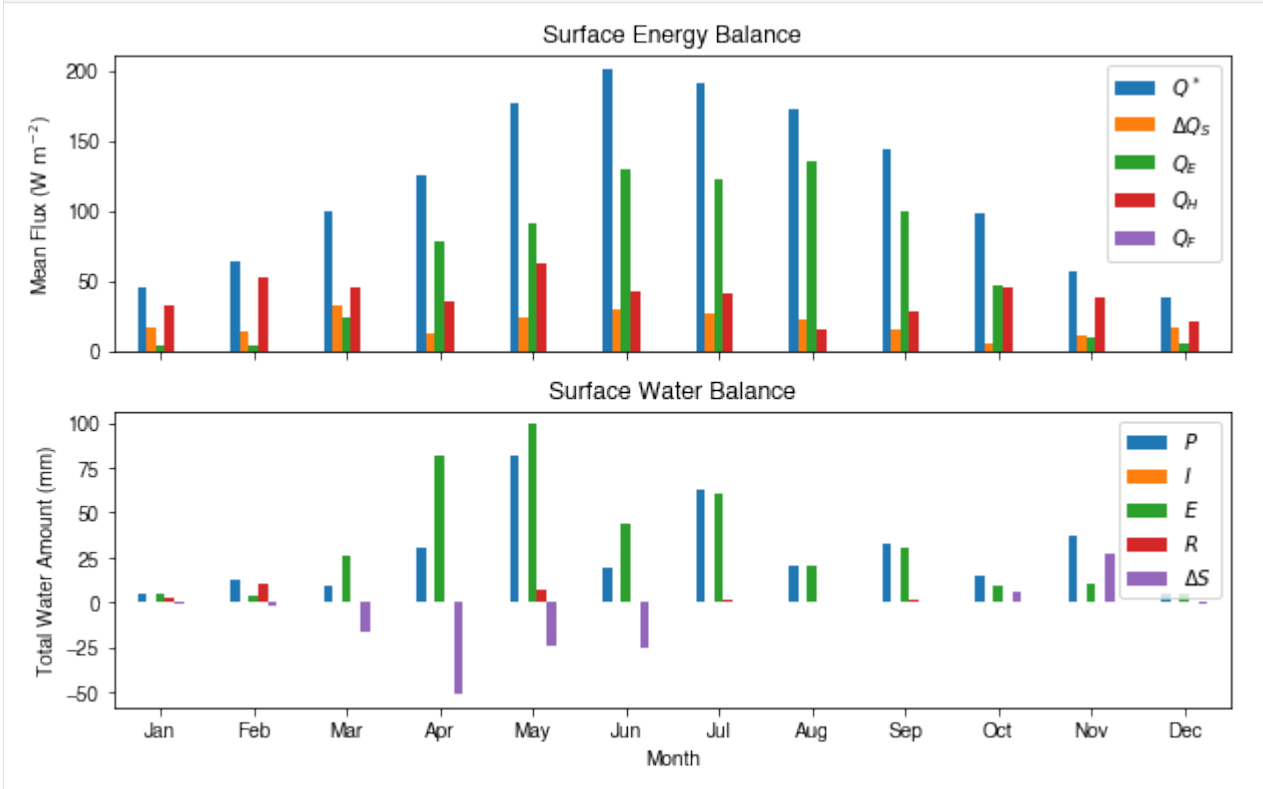
(continues on next page)

(continued from previous page)

```

_ = axes[1].set_xlabel("Month")
_ = axes[1].set_ylabel("Total Water Amount (mm)")
_ = axes[1].axis.set_ticklabels(name_mon, rotation=0)
_ = axes[1].legend()

```



## Save results to external files

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

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

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

```

1_2010_DailyState.txt
1_2010_SUEWS_60.txt
1_2010_snow_60.txt
1_2010_RSL_60.txt
1_2010_SOLWEIG_60.txt
df_state.csv

```

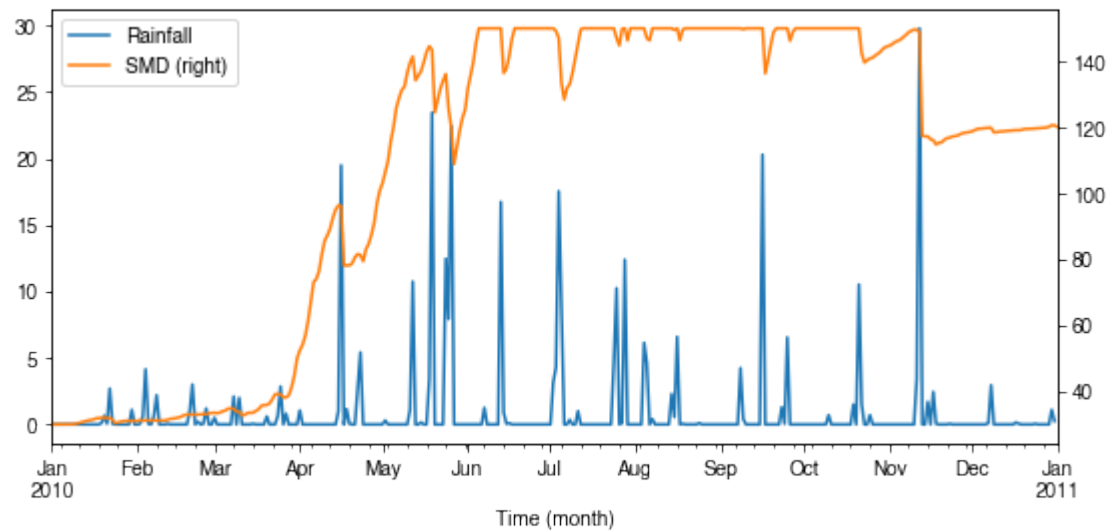
## 1.4.5 More explorations into simulation results

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

## Dynamics in rainfall and soil moisture deficit (SMD)

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

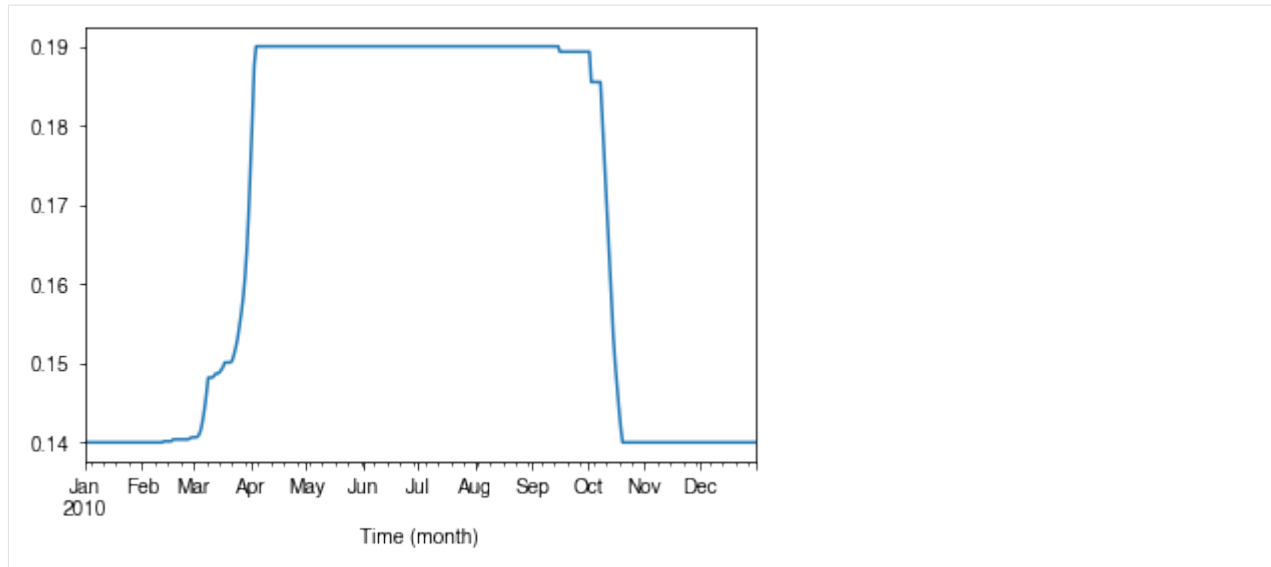
```
[37]: # daily rainfall  
ser_p = df_dailystate.P_day.rename("Rainfall")  
ser_smd = df_output_suews.SMD  
ser_smd_dmax = ser_smd.resample("1d").max().rename("SMD")  
  
ax = pd.concat([ser_p, ser_smd_dmax], axis=1).plot(secondary_y="SMD", figsize=(9, 4))  
_ = ax.set_xlabel("Time (month)")
```



## Variability in albedo

### How does albedo change over time?

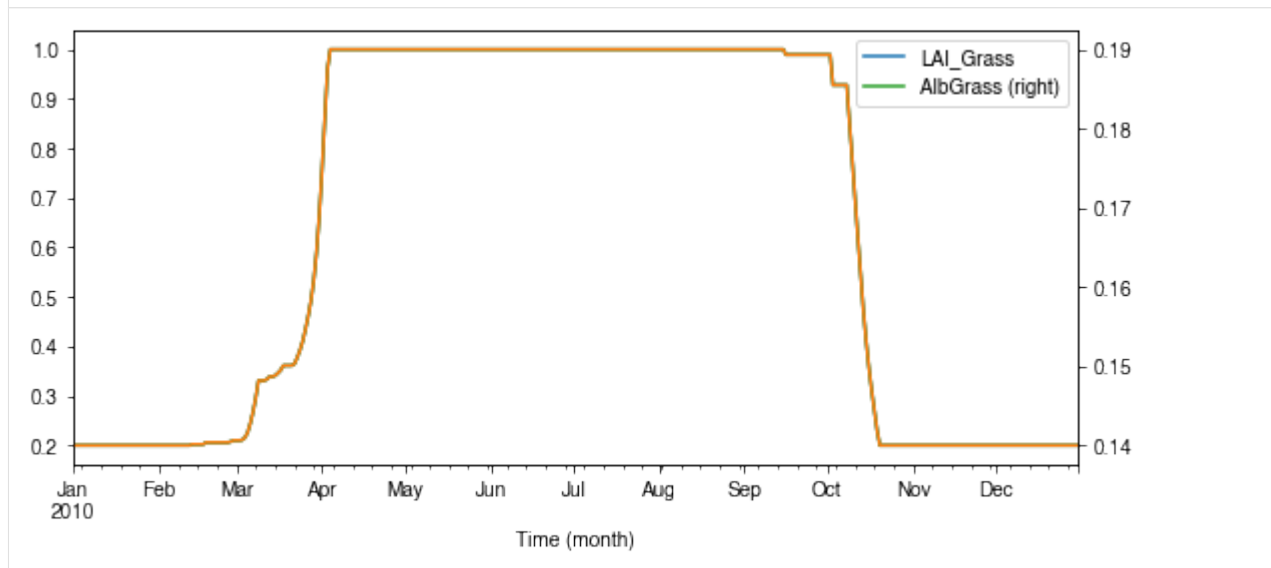
```
[38]: ser_alb = df_dailystate.AlbGrass  
ax = ser_alb.plot()  
_ = ax.set_xlabel("Time (month)")
```



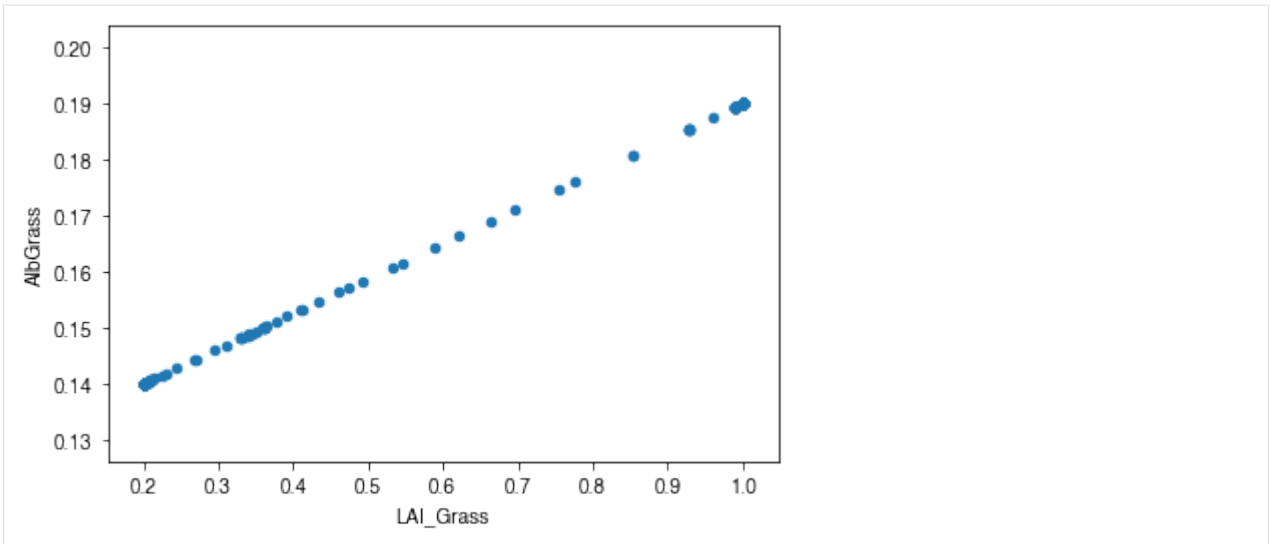
### How is albedo associated with vegetation phenology?

```
[39]: ser_lai = df_dailystate.LAI_Grass
pd.concat([ser_lai, ser_alb], axis=1).plot(secondary_y="AlbGrass", figsize=(9, 4))
ax = ser_lai.plot()
_ = ax.set_xlabel("Time (month)")
```

```
[39]: <matplotlib.axes._subplots.AxesSubplot at 0x7f8969449978>
```



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



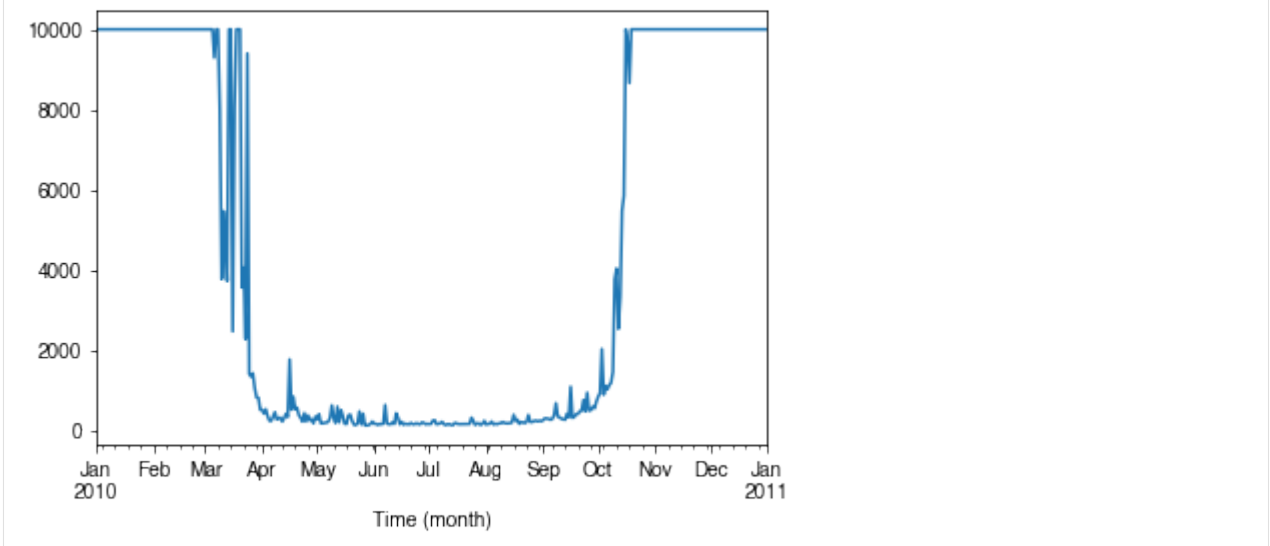
## Variability in surface resistance

### How does surface resistance vary over time?

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

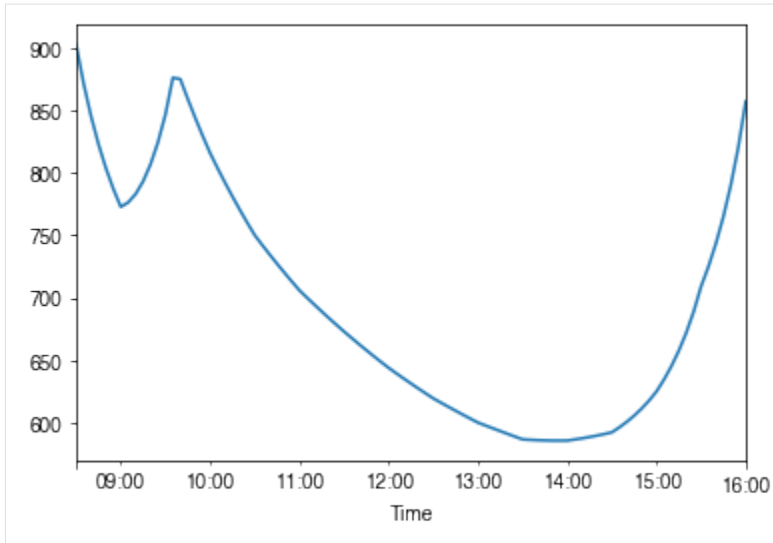
- intra-annual

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

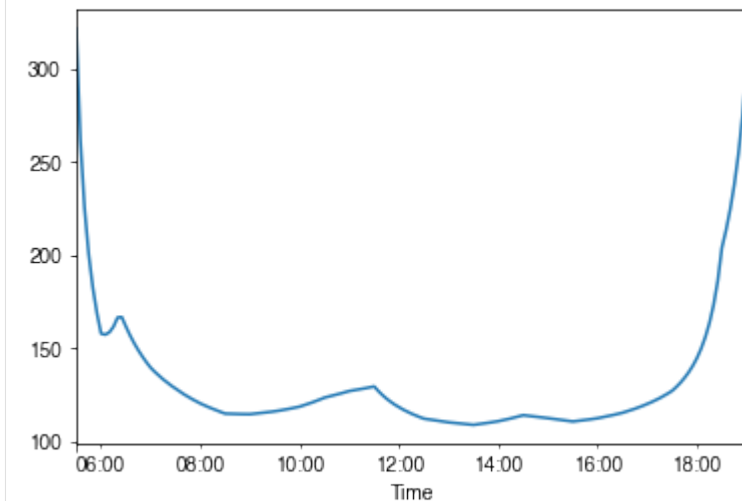


- intra-daily

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

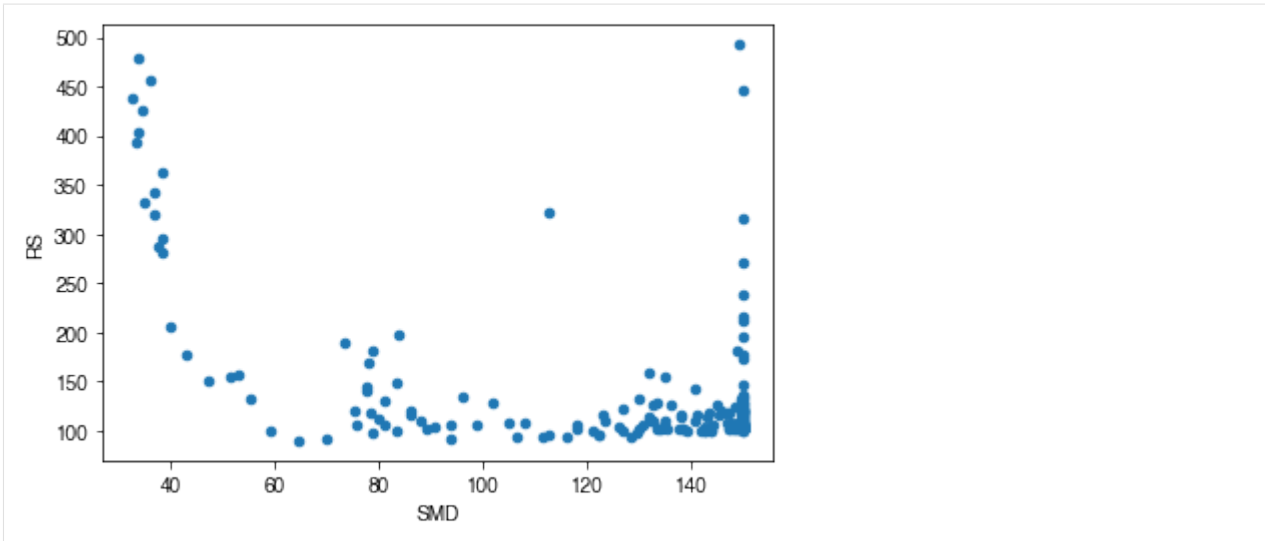


```
[44]: # a summer day
ax = ser_rs.loc["2010-07-01"].between_time("0530", "1900").plot()
_ = ax.set_xlabel("Time")
```

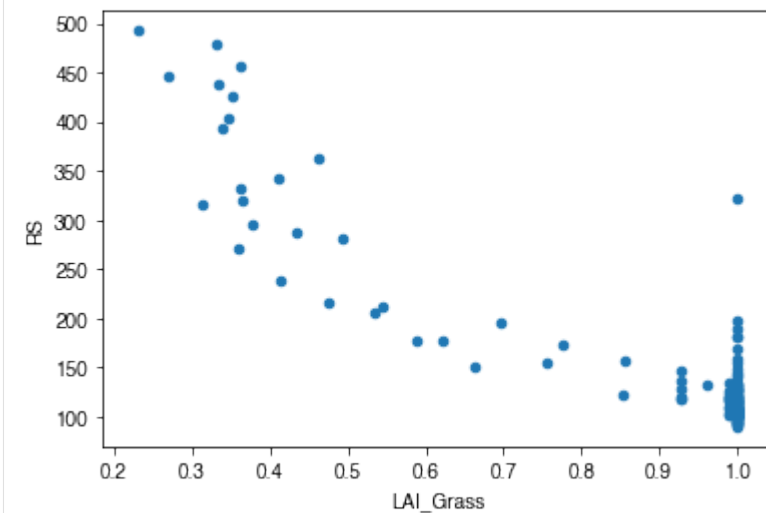


### How is surface resistance associated with other surface properties?

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



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



**How is surface resistance dependent on meteorological conditions?**

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

```
[48]: # solar radiation
# colour by season
ser_kdown = df_forcing_amf.kdown
df_x = pd.concat([ser_kdown, ser_rs], axis=1).between_time('1000', '1600')
df_x = df_x.loc[df_x.RS < 1500]
```

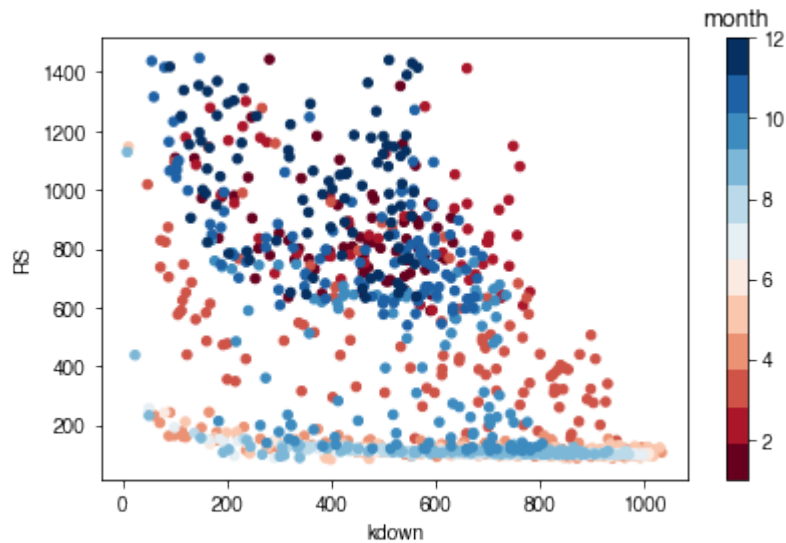
(continues on next page)



(continued from previous page)

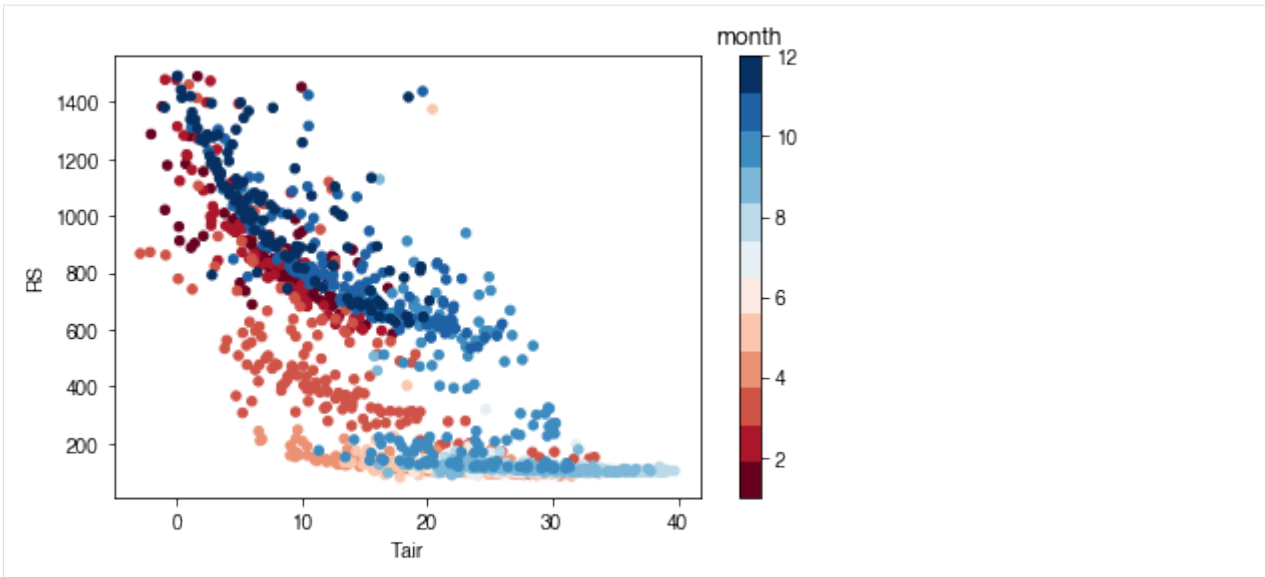
```
df_plot = df_x.iloc[::20]
ax = df_plot.plot.scatter(x='kdown',
                          y='RS',
                          c=df_plot.index.month,
                          cmap=cmap_sel,
                          sharex=False)

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



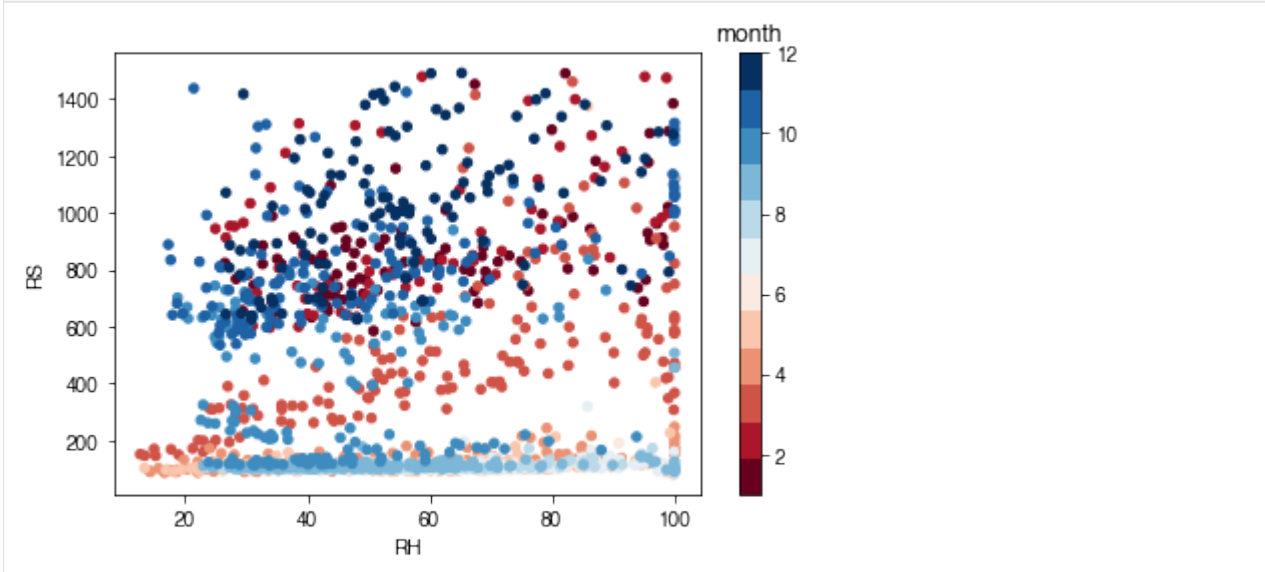
```
[49]: # air temperature
ser_ta = df_forcing_amf.Tair
df_x = pd.concat([ser_ta, ser_rs], axis=1).between_time('1000', '1600')
df_x = df_x.loc[df_x.RS < 1500]
df_plot = df_x.iloc[::15]
ax = df_plot.plot.scatter(x='Tair',
                          y='RS',
                          c=df_plot.index.month,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

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



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

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



- **Task:**

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

---

End of doc/tutorial/setup-own-site.ipynb

---

**Note:**

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

## 1.5 Python 101 before SuPy

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

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

- Installation

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

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

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

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

---



---

## Key IO Data Structures in SuPy

---

### 2.1 Introduction

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

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

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

- df\_state\_init: model initial states;
- df\_forcing: forcing data.

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

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

- df\_output: model output results; this is usually the basis for scientific analysis.
- df\_state\_final: model final states; any of its entries can be used as a df\_state\_init to start another SuPy simulation.

### 2.2 Input

#### 2.2.1 df\_state\_init: model initial states

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

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

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

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

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

[1 rows x 1200 columns]
```

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

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

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

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

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

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

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

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

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

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

(continues on next page)

(continued from previous page)

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

[1 rows x 96 columns]
```

## 2.2.2 df\_forcing: forcing data

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

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

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

```
[4]:
```

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

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

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

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

### Note:

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

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

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

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

## 2.3 Output

### 2.3.1 df\_output: model output results

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

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

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

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

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

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

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

(continues on next page)



(continued from previous page)

```

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

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

[5 rows x 218 columns]

```

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

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

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

## 2.3.2 `df_state_final`: model final states

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

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

```

[8]: df_state_final.head()

[8]: var          aerodynamicresistancemethod ah_min      \
ind_dim          0  (0,)  (1,)
datetime      grid
2012-01-01 00:05:00 98      2  15.0  15.0
2013-01-01 00:05:00 98      2  15.0  15.0

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

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

```

(continues on next page)

(continued from previous page)

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

var                                ... wuprofm_24hr          \
ind_dim          (3, 0) (3, 1) ...      (18, 0) (18, 1) (19, 0)
datetime          grid          ...
2012-01-01 00:05:00 98          0.4  0.47 ...      -999.0 -999.0 -999.0
2013-01-01 00:05:00 98          0.4  0.47 ...      -999.0 -999.0 -999.0

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

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

[2 rows x 1200 columns]
```

---

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

## 3.1 Top-level Functions

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

### 3.1.1 supy.init\_supy

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

#### Parameters

- **path\_init** (*str*) –  
**Path to a file that can initialise SuPy, which can be either of the follows:**
  - SUEWS `RunControl.nml`: a namelist file for SUEWS configurations
  - SuPy `df_state.csv`: a CSV file including model states produced by a SuPy run via `supy.save_supy()`
- **force\_reload** (*boolean, optional*) – Flag to force reload all initialisation files by clearing all cached states, with default value `True` (i.e., force reload all files). Note: If the number of simulation grids is large (e.g., > 100), `force_reload=False` is strongly recommended for better performance.
- **check\_input** (*boolean, optional*) – flag for checking validity of input: `df_forcing` and `df_state_init`. If set to `True`, any detected invalid input will

stop SuPy simulation; a `False` flag will bypass such validation and may incur kernel error if any invalid input. *Note: such checking procedure may take some time if the input is large.* (the default is `False`, which bypasses the validation).

**Returns** `df_state_init` – Initial model states. See *df\_state variables* for details.

**Return type** `pandas.DataFrame`

## Examples

1. Use `RunControl.nml` to initialise SuPy

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

2. Use `df_state.csv` to initialise SuPy

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

### 3.1.2 supy.load\_forcing\_grid

`supy.load_forcing_grid(path_runcontrol: str, grid: int, check_input=False, force_reload=True) → 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
- **check\_input** (*bool, optional*) – flag for checking validity of input: `df_forcing` and `df_state_init`. If set to `True`, any detected invalid input will stop SuPy simulation; a `False` flag will bypass such validation and may incur kernel error if any invalid input. *Note: such checking procedure may take some time if the input is large.* (the default is `False`, which bypasses the validation).

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

**Return type** `pandas.DataFrame`

## Examples

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

### 3.1.3 supy.run\_supy

`supy.run_supy` (*df\_forcing*: *pandas.core.frame.DataFrame*, *df\_state\_init*: *pandas.core.frame.DataFrame*, *save\_state*=*False*, *chunk\_day*=3660, *logging\_level*=20, *check\_input*=*False*, *serial\_mode*=*False*) → *Tuple*[*pandas.core.frame.DataFrame*, *pandas.core.frame.DataFrame*]

Perform supy simulation.

#### Parameters

- **df\_forcing** (*pandas.DataFrame*) – forcing data for all grids in *df\_state\_init*.
- **df\_state\_init** (*pandas.DataFrame*) – initial model states; or a collection of model states with multiple timestamps, whose last temporal record will be used as the initial model states.
- **save\_state** (*bool*, *optional*) – flag for saving model states at each time step, which can be useful in diagnosing model runtime performance or performing a restart run. (the default is *False*, which instructs supy not to save runtime model states).
- **chunk\_day** (*int*, *optional*) – chunk size (*chunk\_day* days) to split simulation periods so memory usage can be reduced. (the default is 3660, which implies ~10-year forcing chunks used in simulations).
- **logging\_level** (*logging level*) – one of these values [50 (CRITICAL), 40 (ERROR), 30 (WARNING), 20 (INFO), 10 (DEBUG)]. A lower value informs SuPy for more verbose logging info.
- **check\_input** (*bool*, *optional*) – flag for checking validity of input: *df\_forcing* and *df\_state\_init*. If set to *True*, any detected invalid input will stop SuPy simulation; a *False* flag will bypass such validation and may incur kernel error if any invalid input. *Note: such checking procedure may take some time if the input is large.* (the default is *False*, which bypasses the validation).
- **serial\_mode** (*bool*, *optional*) – If set to *True*, SuPy simulation will be conducted in serial mode; a *False* flag will try parallel simulation if possible (Windows not supported, i.e., always serial). (the default is *False*).

#### Returns

**df\_output, df\_state\_final** –

- *df\_output*: *output results*
- *df\_state\_final*: *final model states*

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

#### Examples

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

### 3.1.4 supy.save\_supy

`supy.save_supy` (*df\_output*: *pandas.core.frame.DataFrame*, *df\_state\_final*: *pandas.core.frame.DataFrame*, *freq\_s*: *int* = 3600, *site*: *str* = "", *path\_dir\_save*: *str* = *PosixPath('.')*, *path\_runcontrol*: *str* = *None*, *save\_tstep*=*False*, *logging\_level*=50, *output\_level*=1, *debug*=*False*) → *list*

Save SuPy run results to files

### Parameters

- **df\_output** (*pandas.DataFrame*) – DataFrame of output
- **df\_state\_final** (*pandas.DataFrame*) – DataFrame of final model states
- **freq\_s** (*int, optional*) – Output frequency in seconds (the default is 3600, which indicates hourly output)
- **site** (*str, optional*) – Site identifier (the default is '', which indicates site identifier will be left empty)
- **path\_dir\_save** (*str, optional*) – Path to directory to saving the files (the default is Path('.'), which indicates the current working directory)
- **path\_runcontrol** (*str, optional*) – Path to SUEWS [RunControl.nml](#), which, if set, will be preferably used to derive `freq_s`, `site` and `path_dir_save`. (the default is None, which is unset)
- **save\_tstep** (*bool, optional*) – whether to save results in temporal resolution as in simulation (which may result very large files and slow progress), by default False.
- **logging\_level** (*logging level*) – one of these values [50 (CRITICAL), 40 (ERROR), 30 (WARNING), 20 (INFO), 10 (DEBUG)]. A lower value informs SuPy for more verbose logging info.
- **output\_level** (*integer, optional*) – option to determine selection of output variables, by default 1. Notes: 0 for all but snow-related; 1 for all; 2 for a minimal set without land cover specific information.
- **debug** (*bool, optional*) – whether to enable debug mode (e.g., writing out in serial mode, and other debug uses), by default False.

**Returns** a list of paths of saved files

**Return type** `list`

### Examples

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

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

2. save results according to settings in [RunControl.nml](#)

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

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

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

### 3.1.5 supy.load\_SampleData

`supy.load_SampleData()` → `Tuple[pandas.core.frame.DataFrame, pandas.core.frame.DataFrame]`  
Load sample data for quickly starting a demo run.

#### Returns

**df\_state\_init, df\_forcing** –

- **df\_state\_init**: *initial model states*
- **df\_forcing**: *forcing data*

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

#### Examples

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

### 3.1.6 supy.show\_version

`supy.show_version()`  
print SuPy and supy\_driver version information.

## 3.2 Utility Functions

### 3.2.1 ERA-5 Data Downloader

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

#### supy.util.download\_era5

`supy.util.download_era5(lat_x: float, lon_x: float, start: str, end: str, dir_save=PosixPath('.'), grid=None, scale=0, logging_level=20)` → `dict`  
Generate ERA-5 cdsapi-based requests and download data for area of interests.

#### Parameters

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

- **logging\_level** (*logging level*) – one of these values [50 (CRITICAL), 40 (ERROR), 30 (WARNING), 20 (INFO), 10 (DEBUG)]. A lower value informs SuPy for more verbose logging info.

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

**Return type** `dict`

---

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

---

### supy.util.gen\_forcing\_era5

```
supy.util.gen_forcing_era5(lat_x: float, lon_x: float, start: str, end: str, dir_save=PosixPath('.'),
                           grid=None, hgt_agl_diag=100.0, scale=0, force_download=True,
                           simple_mode=True, logging_level=20) → list
```

Generate SUEWS forcing files using ERA-5 data.

#### Parameters

- **lat\_x** (*float*) – Latitude of centre at the area of interest.
- **lon\_x** (*float*) – Longitude of centre at the area of interest.
- **start** (*str*) – Any datetime-like string that can be parsed by `pandas.daterange()`.
- **end** (*str*) – Any datetime-like string that can be parsed by `pandas.daterange()`.
- **dir\_save** (*Path or path-like string*) – path to directory for saving downloaded ERA5 netCDF files.
- **grid** (*list, optional*) – grid size used in CDS request API, by default [0.125, 0.125].
- **hgt\_agl\_diag** (*float*) – height above ground level to diagnose forcing variables, by default 100; the ground level is taken from ERA5 grid altitude.
- **scale** (*int, optional*) – scaling factor that determines the area of interest (i.e., `area=grid[0]*scale`), by default 0
- **force\_download** (*boolean, optional*) – flag to determine whether to download required ERA5 netCDF files; if `False`, all ERA5-related nc files in `dir_save` will be picked up for generation. by default `True`.
- **simple\_mode** (*boolean*) – if use the *simple* mode for diagnosing the forcing variables, by default `True`. In the simple mode, temperature is diagnosed using environmental lapse rate 6.5 K/km and wind speed using MOST under neutral condition. If `False`, MOST with consideration of stability conditions will be used to diagnose forcing variables.
- **logging\_level** (*logging level*) – one of these values [50 (CRITICAL), 40 (ERROR), 30 (WARNING), 20 (INFO), 10 (DEBUG)]. A lower value informs SuPy for more verbose logging info.

**Returns** A list of files in SUEWS forcing input format.

**Return type** `List`

---

**Note:**



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

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

4. For `start/end`, it is recommended to use the format `YYYY-MM-DD` to avoid confusion in day/month-first conversions (an [upstream known issue](#) due to the `dateutil` behavior)

### 3.2.2 Typical Meteorological Year

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

#### `supy.util.gen_epw`

`supy.util.gen_epw(df_output: pandas.core.frame.DataFrame, lat, lon, tz=0, path_epw=PosixPath('uTMY.epw')) → Tuple[pandas.core.frame.DataFrame, str, pathlib.Path]`

Generate an `epw` file of uTMY (urbanised Typical Meteorological Year) using SUEWS simulation results

##### Parameters

- **df\_output** (`pd.DataFrame`) – SUEWS simulation results.
- **path\_epw** (`Path`, *optional*) – Path to store generated `epw` file, by default `Path('./uTMY.epw')`.
- **lat** (`float`) – Latitude of the site, used for calculating solar angle.
- **lon** (`float`) – Longitude of the site, used for calculating solar angle.
- **tz** (`float`) – time zone represented by time difference from UTC+0 (e.g., 8 for UTC+8), by default 0 (i.e., UTC+0)

##### Returns

**df\_epw, text\_meta, path\_epw** –

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

**Return type** `Tuple[pd.DataFrame, str, Path]`

#### `supy.util.read_epw`

`supy.util.read_epw(path_epw: pathlib.Path) → pandas.core.frame.DataFrame`  
Read in `epw` file as a `DataFrame`

**Parameters** `path_epw` (*Path*) – path to epw file

**Returns** `df_tmy` – TMY results of epw file

**Return type** `pd.DataFrame`

### 3.2.3 Gap Filling

---

<code>fill_gap_all</code> ( <code>ser_to_fill</code> [, <code>freq</code> ])	Fill all gaps in a time series using data from neighbouring divisions of 'freq'
--	---

---

#### `supy.util.fill_gap_all`

`supy.util.fill_gap_all` (`ser_to_fill`: `pandas.core.series.Series`, `freq='1D'`) → `pd.Series`  
Fill all gaps in a time series using data from neighbouring divisions of 'freq'

**Parameters**

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

**Returns**

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

### 3.2.4 OHM

---

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

---

#### `supy.util.derive_ohm_coef`

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

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

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

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

**supy.util.sim\_ohm**

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

**Parameters**

- **ser\_qn** (*pd.Series*) – net all-wave radiation.
- **a1** (*float*) – a1 of OHM coefficients.
- **a2** (*float*) – a2 of OHM coefficients.
- **a3** (*float*) – a3 of OHM coefficients.

**Returns** heat storage flux calculated by OHM.

**Return type** *pd.Series*

**3.2.5 Surface Conductance**

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

**supy.util.cal\_gs\_mod**

`supy.util.cal_gs_mod`(*kd*, *ta\_c*, *rh*, *pa*, *smd*, *lai*, *g\_cst*, *g\_max*, *lai\_max*, *wp\_smd*, *debug=False*)  
 Model surface conductance/resistance using phenology and atmospheric forcing conditions.

**Parameters**

- **kd** (*numeric*) – Incoming solar radiation [W m<sup>-2</sup>]
- **ta\_c** (*numeric*) – Air temperature [degC]
- **rh** (*numeric*) – Relative humidity [%]
- **pa** (*numeric*) – Air pressure [Pa]
- **smd** (*numeric*) – Soil moisture deficit [mm]
- **lai** (*numeric*) – Leaf area index [m<sup>2</sup> m<sup>-2</sup>]
- **g\_cst** (*size-6 array*) – Parameters to determine surface conductance/resistance: *g\_lai* (LAI related), *g2* (solar radiation related), *g\_q1* (humidity related), *g\_q2* (humidity related), *g\_ta* (air temperature related), *g\_smd* (soil moisture related)
- **g\_max** (*numeric*) – Maximum surface conductance [mm s<sup>-1</sup>]
- **lai\_max** (*numeric*) – Maximum LAI [m<sup>2</sup> m<sup>-2</sup>]
- **wp\_smd** (*numeric*) – Wilting point indicated as soil moisture deficit [mm]

**Returns** Modelled surface conductance [mm s<sup>-1</sup>]

**Return type** *numeric*

### supy.util.cal\_gs\_obs

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

Calculate surface conductance based on observations, notably turbulent fluxes.

#### Parameters

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

**Returns** Surface conductance based on observations [mm s-1]

**Return type** numeric

### supy.util.calib\_g

`supy.util.calib_g(df_fc_suews, ser_ra, g_max, lai_max, wp_smd, method='cobyta', prms_init=None, debug=False)`

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

#### Parameters

- **df\_fc\_suews** (*pandas.DataFrame*) – DataFrame in SuPy forcing format
- **ser\_ra** (*pandas.Series*) – Series with RA, aerodynamic resistance, [s m-1]
- **g\_max** (*numeric*) – Maximum surface conductance [mm s-1]
- **lai\_max** (*numeric*) – Maximum LAI [m2 m-2]
- **wp\_smd** (*numeric*) – Wilting point indicated as soil moisture deficit [mm]
- **method** (*str, optional*) – Method used in minimisation by `lmfit.minimize`: details refer to its method.
- **prms\_init** (*lmfit.Parameters, optional*) – Initial parameters for calibration
- **debug** (*bool, optional*) – Option to output final calibrated `ModelResult`, by default False

#### Returns

dict, or 'ModelResult <lmfit –

1. dict: {parameter\_name -> best\_fit\_value}
2. `ModelResult`

**Note:** Parameters for surface conductance: `g_lai` (LAI related), `g2` (solar radiation related), `g_q1` (humidity related), `g_q2` (humidity related), `g_ta` (air temperature related), `g_smd` (soil moisture related)

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

---

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

---

### 3.2.6 WRF-SUEWS

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

#### `supy.util.extract_reclassification`

`supy.util.extract_reclassification(path_nml: str) → pandas.core.frame.DataFrame`  
 Extract reclassification info from `path_nml` as a DataFrame.

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

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

**Return type** `pd.DataFrame`

#### `supy.util.plot_reclassification`

`supy.util.plot_reclassification(path_nml: str, path_save='LC-WRF-SUEWS.png', width=800, height=360, top=10, bottom=10, left=260, right=60)`  
 Produce Sankey Diagram to visualise the reclassification specified in `path_nml`

**Parameters**

- `path_nml (str)` – Path to `namelist.suews`
- `path_save (str, optional)` – Path to save Sankey diagram, by default 'LC-WRF-SUEWS.png'
- `width (int, optional)` – Width of diagram, by default 800
- `height (int, optional)` – Height of diagram, by default 360
- `top (int, optional)` – Top margin of diagram, by default 10
- `bottom (int, optional)` – Bottom margin of diagram, by default 10
- `left (int, optional)` – Left margin of diagram, by default 260
- `right (int, optional)` – Right margin of diagram, by default 60

**Returns** Sankey Diagram showing the reclassification.

**Return type** Sankey Diagram

### 3.2.7 Plotting

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

### supy.util.plot\_comp

`supy.util.plot_comp(df_var, scatter_kws={'alpha': 0.1, 'color': 'k', 's': 0.3}, kde_kws={'levels': 4, 'shade': True, 'shade_lowest': False}, show_pdf=False, fig=None, ax=None)`  
Produce a scatter plot with linear regression line to compare simulation results and observations.

#### Parameters

- **df\_var** (*pd.DataFrame*) – DataFrame containing variables to plot with datetime as index. Two columns, ‘Obs’ and ‘Sim’ for observations and simulation results, respectively, must exist.
- **scatter\_kws** (*dict*) – keyword arguments passed to `sns.regplot`. By default, `{ "alpha": 0.1, "s": 0.3, "color": "k" }`.
- **show\_pdf** (*boolean*) – if a PDF overlay should be added. By default, `False`.
- **kde\_kws** (*dict*) – `kde_kws` passed to `sns.kdeplot` when `show_pdf=True`

**Returns** figure showing 1:1 line plot

**Return type** `MPL.figure`

### supy.util.plot\_day\_clm

`supy.util.plot_day_clm(df_var, fig=None, ax=None, show_dif=False, col_ref='Obs')`  
Produce a ensemble diurnal climatologies with uncertainties shown in inter-quartile ranges.

#### Parameters

- **df\_var** (*pd.DataFrame*) – DataFrame containing variables to plot with datetime as index.
- **show\_dif** (*boolean*) – flag to determine if differences against `col_ref` should be plotted.
- **col\_ref** (*str*) – name of column that is used as reference to show differences instead of original values.

**Returns** figure showing median lines and IQR in shadings

**Return type** `MPL.figure`

### supy.util.plot\_rsl

`supy.util.plot_rsl(df_output, var=None, fig=None, ax=None)`  
Produce a quick plot of RSL results

#### Parameters

- **df\_output** (*pandas.DataFrame*) – SuPy output dataframe with RSL results.

- **var** (*str*, *optional*) – Variable to plot; must be one of ‘U’, ‘T’, or ‘q’; or use `None` to plot all; by default `None`

**Returns** (*fig*, *ax*) of plot.

**Return type** `tuple`

**Raises** `issue` – If an invalid variable is specified, an issue will be raised.

### 3.2.8 Roughness Calculation

<code>optimize_MO(df_val, z_meas, h_sfc)</code>	Calculates surface roughness and zero plane displacement height.
<code>cal_neutral(df_val, z_meas, h_sfc)</code>	Calculates the rows associated with neutral condition (threshold=0.01)

#### `supy.util.optimize_MO`

`supy.util.optimize_MO(df_val, z_meas, h_sfc)`

Calculates surface roughness and zero plane displacement height. Refer to <https://suews-parameters-docs.readthedocs.io/en/latest/steps/roughness-SuPy.html> for example

##### Parameters

- **df\_val** (*pd.DataFrame*) – Index should be time with columns: ‘H’, ‘USTAR’, ‘TA’, ‘RH’, ‘PA’, ‘WS’
- **z\_meas** – measurement height in m
- **h\_sfc** – vegetation height in m

##### Returns

- *z0* – surface roughness
- *d* – zero displacement height
- **ser\_ws** (*pd.series*) – observation time series of WS (Neutral conditions)
- **ser\_ustar** (*pd.series*) – observation time series of  $u^*$  (Neutral conditions)

#### `supy.util.cal_neutral`

`supy.util.cal_neutral(df_val, z_meas, h_sfc)`

Calculates the rows associated with neutral condition (threshold=0.01)

##### Parameters

- **df\_val** (*pd.DataFrame*) – Index should be time with columns: ‘H’, ‘USTAR’, ‘TA’, ‘RH’, ‘PA’, ‘WS’
- **z\_meas** – measurement height in m
- **h\_sfc** – vegetation height in m

##### Returns

- **ser\_ws** (*pd.series*) – observation time series of WS (Neutral conditions)
- **ser\_ustar** (*pd.series*) – observation time series of  $u^*$  (Neutral conditions)

## 3.3 Command-Line Tools

### 3.3.1 suews-run

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

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

#### Arguments

**PATH\_RUNCONTROL**

Optional argument

### 3.3.2 suews-convert

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

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

#### Options

**-f, --from <fromVer>**

**Required** Version to convert from

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

**-t, --to <toVer>**

**Required** Version to convert to

**Options** 2020a | 2019b | 2019a | 2018c | 2018b | 2018a | 2017a

**-i, --input <fromDir>**

**Required** Original directory to convert, which must have the RunControl.nml file

**-o, --output <toDir>**

**Required** New directory to create for converted tables. Note: the created directory will have the same structure as the original one; however, forcing files and output folder won't be included.

## 3.4 Key Data Structures

### 3.4.1 df\_state variables

---

**Note:** Data structure of df\_state is explained [here](#).

---

**aerodynamicresistancemethod**

**Description** Internal use. Please DO NOT modify

**Dimensionality** 0



**Dimensionality Remarks** Scalar

**SUEWS-related variables** None

#### **ah\_min**

**Description** Minimum QF values.

**Dimensionality** (2,)

**Dimensionality Remarks** 2: {Weekday, Weekend}

**SUEWS-related variables** AHMin\_WD, AHMin\_WE

#### **ah\_slope\_cooling**

**Description** Cooling slope of QF calculation.

**Dimensionality** (2,)

**Dimensionality Remarks** 2: {Weekday, Weekend}

**SUEWS-related variables** AHSlope\_Cooling\_WD, AHSlope\_Cooling\_WE

#### **ah\_slope\_heating**

**Description** Heating slope of QF calculation.

**Dimensionality** (2,)

**Dimensionality Remarks** 2: {Weekday, Weekend}

**SUEWS-related variables** AHSlope\_Heating\_WD, AHSlope\_Heating\_WE

#### **ahprof\_24hr**

**Description** Hourly profile values used in energy use calculation.

**Dimensionality** (24, 2)

**Dimensionality Remarks** 24: hours of a day

2: {Weekday, Weekend}

**SUEWS-related variables** EnergyUseProfWD, EnergyUseProfWE

#### **alb**

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

**Dimensionality** (7,)

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

**SUEWS-related variables** AlbedoMax

#### **albdectr\_id**

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

**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** albDecTr0

#### **albevetr\_id**

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

**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** `albEveTr0`

#### `albgrass_id`

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

**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** `albGrass0`

#### `albmax_dectr`

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

**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** `AlbedoMax`

#### `albmax_evetr`

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

**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** `AlbedoMax`

#### `albmax_grass`

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

**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** `AlbedoMax`

#### `albmin_dectr`

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

**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** `AlbedoMin`

#### `albmin_evetr`

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

**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** `AlbedoMin`

#### `albmin_grass`

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

**Dimensionality** 0

**Dimensionality Remarks** Scalar

SUEWS-related variables `AlbedoMin`

#### `alpha_bioco2`

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

**Dimensionality** (3,)

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

SUEWS-related variables `alpha`

#### `alpha_enh_bioco2`

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

**Dimensionality** (3,)

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

SUEWS-related variables `alpha_enh`

#### `alt`

**Description** Altitude of grids [m].

**Dimensionality** 0

**Dimensionality Remarks** Scalar

SUEWS-related variables `Alt`

#### `baset`

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

**Dimensionality** (3,)

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

SUEWS-related variables `BaseT`

#### `baset_cooling`

**Description** Critical cooling temperature.

**Dimensionality** (2,)

**Dimensionality Remarks** 2: { Weekday, Weekend }

SUEWS-related variables `TCritic_Cooling_WD`, `TCritic_Cooling_WE`

#### `baset_hc`

**Description** Base temperature for heating degree days [°C]

**Dimensionality** 0

**Dimensionality Remarks** Scalar

SUEWS-related variables `BaseT_HC`

#### `baset_heating`

**Description** Critical heating temperature.

**Dimensionality** (2,)

**Dimensionality Remarks** 2: { Weekday, Weekend }

SUEWS-related variables `TCritic_Heating_WD`, `TCritic_Heating_WE`

#### **basete**

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

**Dimensionality** (3,)

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

SUEWS-related variables `BaseTe`

#### **basetmethod**

**Description** Determines method for base temperature used in HDD/CDD calculations.

**Dimensionality** 0

**Dimensionality Remarks** Scalar

SUEWS-related variables `BaseTMethod`

#### **beta\_bioco2**

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

**Dimensionality** (3,)

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

SUEWS-related variables `beta`

#### **beta\_enh\_bioco2**

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

**Dimensionality** (3,)

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

SUEWS-related variables `beta_enh`

#### **bldgh**

**Description** Mean building height [m]

**Dimensionality** 0

**Dimensionality Remarks** Scalar

SUEWS-related variables `H_Bldgs`

#### **capmax\_dec**

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

**Dimensionality** 0

**Dimensionality Remarks** Scalar

SUEWS-related variables `StorageMax`

#### **capmin\_dec**

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

**Dimensionality** 0

**Dimensionality Remarks** Scalar

SUEWS-related variables `StorageMin`

**chanohm**

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

**Dimensionality** (7,)

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

**SUEWS-related variables** `AnOHM_Ch`

**co2pointsource**

**Description** CO2 emission factor [ $\text{kg km}^{-1}$ ]

**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** `CO2PointSource`

**cpanohm**

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

**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<sup>-1</sup>]

**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 [ $\text{J km}^{-1}$ ].

**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** `EnEF_v_Jkm`

#### **evapmethod**

**Description** Internal use. Please DO NOT modify

**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** None

#### **evetreeh**

**Description** Mean height of evergreen trees [m]

**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** `H_EveTr`

#### **faibldg**

**Description** Frontal area index for buildings [-]

**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** `FAI_Bldgs`

#### **faidectree**

**Description** Frontal area index for deciduous trees [-]

**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** `FAI_DecTr`

#### **faievetree**

**Description** Frontal area index for evergreen trees [-]

**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** `FAI_EveTr`

#### **faut**

**Description** Fraction of irrigated area that is irrigated using automated systems

**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** `Faut`

#### **fcef\_v\_kgkm**

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

**Dimensionality** (2,)

**Dimensionality Remarks** 2: {Weekday, Weekend}

**SUEWS-related variables** `FcEF_v_kgkmWD`, `FcEF_v_kgkmWE`

#### **flowchange**

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

**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 [ $\text{mm s}^{-1}$ ]



**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** [G1](#)

**g2**

**Description** Related to Kdown dependence [ $\text{W m}^{-2}$ ]

**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** [G2](#)

**g3**

**Description** Related to VPD dependence [units depend on [gsModel](#)]

**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** [G3](#)

**g4**

**Description** Related to VPD dependence [units depend on [gsModel](#)]

**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** [G4](#)

**g5**

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

**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** [G5](#)

**g6**

**Description** Related to soil moisture dependence [ $\text{mm}^{-1}$ ]

**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) [ $^{\circ}\text{C}$ ].

**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`

#### **h\_maintain**

**Description** water depth to maintain used in automatic irrigation (e.g., ponding water due to flooding irrigation in rice crop-field) [mm].

**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** `H_maintain`

#### **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<sup>-1</sup>]

**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** `InternalWaterUse`

#### **irrfracbldgs**

**Description** Fraction of Bldgs that is irrigated [-]

**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** `IrrFr_Bldgs`

#### **irrfracbsoil**

**Description** Fraction of BSoil that is irrigated [-]

**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** `IrrFr_BSoil`

#### **irrfracdectr**

**Description** Fraction of DecTr that is irrigated [-]

**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** `IrrFr_DecTr`

#### **irrfacevetr**

**Description** Fraction of EveTr that is irrigated [-]

**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** `IrrFr_EveTr`

#### **irrfracgrass**

**Description** Fraction of Grass that is irrigated [-]

**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** `IrrFr_Grass`

#### **irrfracpaved**

**Description** Fraction of Paved that is irrigated [-]

**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** `IrrFr_Paved`

#### **irrfracwater**

**Description** Fraction of Water that is irrigated [-]

**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** `IrrFr_Water`

#### **kkanohm**

**Description** Thermal conductivity for this surface to use in AnOHM [ $\text{W m K}^{-1}$ ]

**Dimensionality** (7,)

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

**SUEWS-related variables** `AnOHM_Kk`

#### **kmax**

**Description** Maximum incoming shortwave radiation [ $\text{W m}^{-2}$ ]

**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<sup>-1</sup>]

**Dimensionality** (3,)

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

**SUEWS-related variables** MaxConductance

#### maxfcmetab

**Description** Maximum (day) CO<sub>2</sub> from human metabolism. [W m<sup>-2</sup>]

**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** MaxFCMetab

#### maxqfmetab

**Description** Maximum value for human heat emission. [W m<sup>-2</sup>]

**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** MaxQFMetab

#### min\_res\_bioco2

**Description** Minimum soil respiration rate (for cold-temperature limit) [umol m<sup>-2</sup> s<sup>-1</sup>].

**Dimensionality** (3,)

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

**SUEWS-related variables** `min_respi`

#### **minfcmetab**

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

**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** `MinFCMetab`

#### **minqfmetab**

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

**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** `MinQFMetab`

#### **narp\_emis\_snow**

**Description** Effective surface emissivity.

**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** `Emissivity`

#### **narp\_trans\_site**

**Description** Atmospheric transmissivity for NARP [-]

**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** `NARP_Trans`

#### **netradiationmethod**

**Description** Determines method for calculation of radiation fluxes.

**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** `NetRadiationMethod`

#### **ohm\_coef**

**Description** Coefficients for OHM calculation.

**Dimensionality** (8, 4, 3)

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

4: { SummerWet, SummerDry, WinterWet, WinterDry }

3: { a1, a2, a3 }

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

**ohm\_threshsw**

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

**Dimensionality** (8,)

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

**SUEWS-related variables** `OHMThresh_SW`

**ohm\_threshwd**

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

**Dimensionality** (8,)

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

**SUEWS-related variables** `OHMThresh_WD`

**ohmincqh**

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

**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** `OHMIncQF`

**pipecapacity**

**Description** Storage capacity of pipes [mm]

**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** `PipeCapacity`

**popdensdaytime**

**Description** Daytime population density (i.e. workers, tourists) [people ha<sup>-1</sup>]

**Dimensionality** (2,)

**Dimensionality Remarks** 2: {Weekday, Weekend}

**SUEWS-related variables** `PopDensDay`

**popdensnighttime**

**Description** Night-time population density (i.e. residents) [people ha<sup>-1</sup>]

**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** `PopDensNight`

**popprof\_24hr**

**Description** Hourly profile values used in dynamic population estimation.

**Dimensionality** (24, 2)

**Dimensionality Remarks** 24: hours of a day

2: {Weekday, Weekend}

**SUEWS-related variables** `PopProfWD`, `PopProfWE`

#### **pormax\_dec**

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

**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** `PorosityMax`

#### **pormin\_dec**

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

**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** `PorosityMin`

#### **porosity\_id**

**Description** Porosity of deciduous vegetation on day 0 of run.

**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** `porosity0`

#### **preciplimit**

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

**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** `PrecipLimSnow`

#### **preciplimitalb**

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

**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** `PrecipLimAlb`

#### **qf0\_beu**

**Description** Building energy use [ $\text{W m}^{-2}$ ]

**Dimensionality** (2,)

**Dimensionality Remarks** 2: {Weekday, Weekend}

**SUEWS-related variables** `QF0_BEU_WD`, `QF0_BEU_WE`

#### **qf\_a**

**Description** Base value for QF calculation.

**Dimensionality** (2,)



**Dimensionality Remarks** 2: {Weekday, Weekend}

**SUEWS-related variables** `QF_A_WD`, `QF_A_WE`

#### **qf\_b**

**Description** Parameter related to heating degree days.

**Dimensionality** (2,)

**Dimensionality Remarks** 2: {Weekday, Weekend}

**SUEWS-related variables** `QF_B_WD`, `QF_B_WE`

#### **qf\_c**

**Description** Parameter related to heating degree days.

**Dimensionality** (2,)

**Dimensionality Remarks** 2: {Weekday, Weekend}

**SUEWS-related variables** `QF_C_WD`, `QF_C_WE`

#### **radmeltfact**

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

**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** `RadMeltFactor`

#### **raincover**

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

**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** `LUMPS_Cover`

#### **rainmaxres**

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

**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** `LUMPS_MaxRes`

#### **resp\_a**

**Description** Respiration coefficient a.

**Dimensionality** (3,)

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

**SUEWS-related variables** `resp_a`

#### **resp\_b**

**Description** Respiration coefficient b - related to air temperature dependency.

**Dimensionality** (3,)

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

SUEWS-related variables `resp_b`

#### `roughlenheatmethod`

**Description** Determines method for calculating roughness length for heat.

**Dimensionality** 0

**Dimensionality Remarks** Scalar

SUEWS-related variables `RoughLenHeatMethod`

#### `roughlenmommmethod`

**Description** Determines how aerodynamic roughness length ( $z_0$ ) and zero displacement height ( $z_{dm}$ ) are calculated.

**Dimensionality** 0

**Dimensionality Remarks** Scalar

SUEWS-related variables `RoughLenMomMethod`

#### `runofftewater`

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

**Dimensionality** 0

**Dimensionality Remarks** Scalar

SUEWS-related variables `RunoffToWater`

#### `s1`

**Description** A parameter related to soil moisture dependence [-]

**Dimensionality** 0

**Dimensionality Remarks** Scalar

SUEWS-related variables `S1`

#### `s2`

**Description** A parameter related to soil moisture dependence [mm]

**Dimensionality** 0

**Dimensionality Remarks** Scalar

SUEWS-related variables `S2`

#### `sathydraulicconduct`

**Description** Hydraulic conductivity for saturated soil [ $\text{mm s}^{-1}$ ]

**Dimensionality** (7,)

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

SUEWS-related variables `SatHydraulicCond`

#### `sddf11`

**Description** The sensesence degree days (SDD) needed to initiate leaf off. [ $^{\circ}\text{C}$ ]

**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 [ $\text{kg m}^{-3}$ ]

**Dimensionality** 0

**Dimensionality Remarks** Scalar

SUEWS-related variables `SnowDensMax`

#### **snowdensmin**

**Description** Fresh snow density [ $\text{kg m}^{-3}$ ]

**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  $\Delta Q_S$ .

**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** `StorageHeatMethod`

**storedrainprm**

**Description** Coefficients used in drainage calculation.

**Dimensionality** (6, 7)

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

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

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

**surfacearea**

**Description** Area of the grid [ha].

**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** `SurfaceArea`

**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 [ $\text{mm K}^{-1} \text{h}^{-1}$ ]

**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** `TempMeltFactor`

**th**

**Description** Upper air temperature limit [ $^{\circ}\text{C}$ ]

**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** `TH`

**theta\_bioco2**

**Description** The convexity of the curve at light saturation.

**Dimensionality** (3,)

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

**SUEWS-related variables** `theta`

**timezone**

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

**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** `Timezone`

**t1**

**Description** Lower air temperature limit [ $^{\circ}\text{C}$ ]

**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** `TL`

#### **trafficrate**

**Description** Traffic rate used for CO2 flux calculation.

**Dimensionality** (2,)

**Dimensionality Remarks** 2: {Weekday, Weekend}

**SUEWS-related variables** `TrafficRate_WD`, `TrafficRate_WE`

#### **trafficunits**

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

**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** `TrafficUnits`

#### **traffprof\_24hr**

**Description** Hourly profile values used in traffic activity calculation.

**Dimensionality** (24, 2)

**Dimensionality Remarks** 24: hours of a day

2: {Weekday, Weekend}

**SUEWS-related variables** `TraffProfWD`, `TraffProfWE`

#### **tstep**

**Description** Specifies the model time step [s].

**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** `Tstep`

#### **veg\_type**

**Description** Internal use. Please DO NOT modify

**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** None

#### **waterdist**

**Description** Fraction of water redistribution

**Dimensionality** (8, 6)

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

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

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

#### **waterusemethod**



**Description** Defines how external water use is calculated.

**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** `WaterUseMethod`

#### **wetthresh**

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

**Dimensionality** (7,)

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

**SUEWS-related variables** `WetThreshold`

#### **wuprofa\_24hr**

**Description** Hourly profile values used in automatic irrigation.

**Dimensionality** (24, 2)

**Dimensionality Remarks** 24: hours of a day

2: { Weekday, Weekend }

**SUEWS-related variables** `WaterUseProfAutoWD`, `WaterUseProfAutoWE`

#### **wuprofm\_24hr**

**Description** Hourly profile values used in manual irrigation.

**Dimensionality** (24, 2)

**Dimensionality Remarks** 24: hours of a day

2: { Weekday, Weekend }

**SUEWS-related variables** `WaterUseProfManuWD`, `WaterUseProfManuWE`

#### **z**

**Description** Measurement height [m].

**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** `z`

#### **z0m\_in**

**Description** Roughness length for momentum [m]

**Dimensionality** 0

**Dimensionality Remarks** Scalar

**SUEWS-related variables** `z0`

#### **zdm\_in**

**Description** Zero-plane displacement [m]

**Dimensionality** 0

**Dimensionality Remarks** Scalar

SUEWS-related variables `zd`

### 3.4.2 `df_forcing` variables

---

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

---

**RH**

**Description** Relative Humidity [%]

**Tair**

**Description** Air temperature [°C]

**U**

**Description** Wind speed [m s<sup>-1</sup>] Height of the wind speed measurement (z) is needed in `SUEWS_SiteSelect.txt`.

**Wuh**

**Description** External water use [m<sup>3</sup>]

**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<sup>-2</sup>] **Recommended in this version.** if `SOLWEIGUse = 1`

**kdir**

**Description** Direct radiation [W m<sup>-2</sup>] **Recommended in this version.** if `SOLWEIGUse = 1`

**kdown**

**Description** Incoming shortwave radiation [W m<sup>-2</sup>] Must be > 0 W m<sup>-2</sup>.

**lai**

**Description** Observed leaf area index [m<sup>2</sup> m<sup>-2</sup>]

**ldown**

**Description** Incoming longwave radiation [W m<sup>-2</sup>]

**pres**

**Description** Barometric pressure [kPa]

**qe**

**Description** Latent heat flux [ $\text{W m}^{-2}$ ]

**qf**

**Description** Anthropogenic heat flux [ $\text{W m}^{-2}$ ]

**qh**

**Description** Sensible heat flux [ $\text{W m}^{-2}$ ]

**qn**

**Description** Net all-wave radiation [ $\text{W m}^{-2}$ ] Required if `NetRadiationMethod = 0`.

**qs**

**Description** Storage heat flux [ $\text{W m}^{-2}$ ]

**rain**

**Description** Rainfall [mm]

**snow**

**Description** Snow cover fraction (0 – 1) [-] Required if `SnowUse = 1`

**wdir**

**Description** Wind direction [ $^{\circ}$ ] **Not available in this version.**

**xsmd**

**Description** Observed soil moisture [ $\text{m}^3 \text{m}^{-3}$ ] or [ $\text{kg kg}^{-1}$ ]

### 3.4.3 df\_output variables

---

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

---

**AddWater**

**Description** Additional water flow received from other grids [mm]

**Group** SUEWS

**AlbBulk**

**Description** Bulk albedo [-]

**Group** SUEWS

**AlbDecTr**

**Description** Albedo of deciduous trees [-]

**Group** DailyState

**AlbEveTr**

**Description** Albedo of evergreen trees [-]

**Group** DailyState

**AlbGrass**

**Description** Albedo of grass [-]

**Group** DailyState

**AlbSnow**

**Description** Snow albedo [-]

**Group** DailyState

**AlbSnow**

**Description** Snow albedo [-]

**Group** SUEWS

**Azimuth**

**Description** Solar azimuth angle [°]

**Group** SUEWS

**CI**

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

**Group** BEERS

**DLHrs**

**Description** Day length [h]

**Group** DailyState

**DaysSR**

**Description** Days since rain [days]

**Group** DailyState

**DecidCap**

**Description** Moisture storage capacity of deciduous trees [mm]

**Group** DailyState

**DensSnow\_BSoil**

**Description** Snow density - bare soil surface [ $\text{kg m}^{-3}$ ]

**Group** DailyState

**DensSnow\_BSoil**

**Description** Snow density – bare soil surface [ $\text{kg m}^{-3}$ ]

**Group** DailyState

**DensSnow\_BSoil**

**Description** Snow density - bare soil surface [ $\text{kg m}^{-3}$ ]

**Group** snow

**DensSnow\_BSoil**

**Description** Snow density – bare soil surface [ $\text{kg m}^{-3}$ ]

**Group** snow

**DensSnow\_Bldgs**

**Description** Snow density - building surface [kg m<sup>-3</sup>]

**Group** DailyState

**DensSnow\_Bldgs**

**Description** Snow density – building surface [kg m<sup>-3</sup>]

**Group** DailyState

**DensSnow\_Bldgs**

**Description** Snow density - building surface [kg m<sup>-3</sup>]

**Group** snow

**DensSnow\_Bldgs**

**Description** Snow density – building surface [kg m<sup>-3</sup>]

**Group** snow

**DensSnow\_DecTr**

**Description** Snow density – deciduous surface [kg m<sup>-3</sup>]

**Group** snow

**DensSnow\_DecTr**

**Description** Snow density - deciduous surface [kg m<sup>-3</sup>]

**Group** snow

**DensSnow\_DecTr**

**Description** Snow density – deciduous surface [kg m<sup>-3</sup>]

**Group** DailyState

**DensSnow\_DecTr**

**Description** Snow density - deciduous surface [kg m<sup>-3</sup>]

**Group** DailyState

**DensSnow\_EveTr**

**Description** Snow density - evergreen surface [kg m<sup>-3</sup>]

**Group** DailyState

**DensSnow\_EveTr**

**Description** Snow density – evergreen surface [kg m<sup>-3</sup>]

**Group** snow

**DensSnow\_EveTr**

**Description** Snow density - evergreen surface [kg m<sup>-3</sup>]

**Group** snow

**DensSnow\_EveTr**

**Description** Snow density – evergreen surface [kg m<sup>-3</sup>]

**Group** DailyState

**DensSnow\_Grass**

**Description** Snow density - grass surface [kg m<sup>-3</sup>]

**Group** DailyState

**DensSnow\_Grass**

**Description** Snow density – grass surface [kg m<sup>-3</sup>]

**Group** DailyState

**DensSnow\_Grass**

**Description** Snow density - grass surface [kg m<sup>-3</sup>]

**Group** snow

**DensSnow\_Grass**

**Description** Snow density – grass surface [kg m<sup>-3</sup>]

**Group** snow

**DensSnow\_Paved**

**Description** Snow density – paved surface [kg m<sup>-3</sup>]

**Group** snow

**DensSnow\_Paved**

**Description** Snow density - paved surface [kg m<sup>-3</sup>]

**Group** snow

**DensSnow\_Paved**

**Description** Snow density – paved surface [kg m<sup>-3</sup>]

**Group** DailyState

**DensSnow\_Paved**

**Description** Snow density - paved surface [kg m<sup>-3</sup>]

**Group** DailyState

**DensSnow\_Water**

**Description** Snow density – water surface [kg m<sup>-3</sup>]

**Group** snow

**DensSnow\_Water**

**Description** Snow density - water surface [kg m<sup>-3</sup>]

**Group** DailyState

**DensSnow\_Water**

**Description** Snow density - water surface [kg m<sup>-3</sup>]

**Group** snow

**DensSnow\_Water**

**Description** Snow density – water surface [kg m<sup>-3</sup>]

**Group** DailyState

**DiffuseRad**

**Description** Diffuse shortwave radiation

**Group** BEERS

**DirectRad**

**Description** Direct shortwave radiation

**Group** BEERS

**Drainage**

**Description** Drainage [mm]

**Group** SUEWS

**Evap**

**Description** Evaporation [mm]

**Group** SUEWS

**Fc**

**Description** CO<sub>2</sub> flux [ $\mu\text{mol m}^{-2} \text{s}^{-1}$ ]

**Group** SUEWS

**FcBuild**

**Description** CO<sub>2</sub> flux from buildings [ $\mu\text{mol m}^{-2} \text{s}^{-1}$ ]

**Group** SUEWS

**FcMetab**

**Description** CO<sub>2</sub> flux from metabolism [ $\mu\text{mol m}^{-2} \text{s}^{-1}$ ]

**Group** SUEWS

**FcPhoto**

**Description** CO<sub>2</sub> flux from photosynthesis [ $\mu\text{mol m}^{-2} \text{s}^{-1}$ ]

**Group** SUEWS

**FcPoint**

**Description** CO<sub>2</sub> flux from point source [ $\mu\text{mol m}^{-2} \text{s}^{-1}$ ]

**Group** SUEWS

**FcRespi**

**Description** CO<sub>2</sub> flux from respiration [ $\mu\text{mol m}^{-2} \text{s}^{-1}$ ]

**Group** SUEWS

**FcTraff**

**Description** CO<sub>2</sub> flux from traffic [ $\mu\text{mol m}^{-2} \text{s}^{-1}$ ]

**Group** SUEWS

**Fcld**

**Description** Cloud fraction [-]

**Group** SUEWS

**FlowCh**

**Description** Additional flow into water body [mm]

**Group** SUEWS

**GDD\_DecTr**

**Description** Growing degree days for deciduous tree [°C d]

**Group** DailyState

**GDD\_EveTr**

**Description** Growing degree days for evergreen tree [°C d]

**Group** DailyState

**GDD\_Grass**

**Description** Growing degree days for grass [°C d]

**Group** DailyState

**GlobalRad**

**Description** Input Kdn

**Group** BEERS

**HDD1\_h**

**Description** Heating degree days [°C d]

**Group** DailyState

**HDD2\_c**

**Description** Cooling degree days [°C d]

**Group** DailyState

**HDD3\_Tmean**

**Description** Average daily air temperature [°C]

**Group** DailyState

**HDD4\_T5d**

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

**Group** DailyState

**I0**

**Description** theoretical value of maximum incoming solar radiation

**Group** BEERS

**Irr**

**Description** Irrigation [mm]

**Group** SUEWS

**Kdown**

**Description** Incoming shortwave radiation [ $\text{W m}^{-2}$ ]

**Group** SUEWS



**Kdown2d**

**Description** Incoming shortwave radiation at POI

**Group** BEERS

**Keast**

**Description** Shortwave radiation from east at POI

**Group** BEERS

**Knorth**

**Description** Shortwave radiation from north at POI

**Group** BEERS

**Ksouth**

**Description** Shortwave radiation from south at POI

**Group** BEERS

**Kup**

**Description** Outgoing shortwave radiation [ $\text{W m}^{-2}$ ]

**Group** SUEWS

**Kup2d**

**Description** Outgoing shortwave radiation at POI

**Group** BEERS

**Kwest**

**Description** Shortwave radiation from west at POI

**Group** BEERS

**LAI**

**Description** Leaf area index [ $\text{m}^2 \text{m}^{-2}$ ]

**Group** SUEWS

**LAI\_DecTr**

**Description** Leaf area index of deciduous trees [ $\text{m}^2 \text{m}^{-2}$ ]

**Group** DailyState

**LAI\_EveTr**

**Description** Leaf area index of evergreen trees [ $\text{m}^2 \text{m}^{-2}$ ]

**Group** DailyState

**LAI\_Grass**

**Description** Leaf area index of grass [ $\text{m}^2 \text{m}^{-2}$ ]

**Group** DailyState

**LAIlumps**

**Description** Leaf area index used in LUMPS (normalised 0-1) [-]

**Group** DailyState

**Ldown**

**Description** Incoming longwave radiation [ $\text{W m}^{-2}$ ]  
**Group** SUEWS

**Ldown2d**

**Description** Incoming longwave radiation at POI  
**Group** BEERS

**Least**

**Description** Longwave radiation from east at POI  
**Group** BEERS

**Lnorth**

**Description** Longwave radiation from north at POI  
**Group** BEERS

**Lob**

**Description** Obukhov length [m]  
**Group** SUEWS

**Lsouth**

**Description** Longwave radiation from south at POI  
**Group** BEERS

**Lup**

**Description** Outgoing longwave radiation [ $\text{W m}^{-2}$ ]  
**Group** SUEWS

**Lup2d**

**Description** Outgoing longwave radiation at POI  
**Group** BEERS

**Lwest**

**Description** Longwave radiation from west at POI  
**Group** BEERS

**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<sup>-1</sup>]

**Group** snow

**Mw\_Bldgs**

**Description** Meltwater – building surface [mm h<sup>-1</sup>]

**Group** snow

**Mw\_DecTr**

**Description** Meltwater – deciduous surface [mm h<sup>-1</sup>]

**Group** snow

**Mw\_EveTr**

**Description** Meltwater – evergreen surface [mm h<sup>-1</sup>]

**Group** snow

**Mw\_Grass**

**Description** Meltwater – grass surface [mm h<sup>-1</sup> 1]

**Group** snow

**Mw\_Paved**

**Description** Meltwater – paved surface [mm h<sup>-1</sup>]

**Group** snow

**Mw\_Water**

**Description** Meltwater – water surface [mm h<sup>-1</sup>]

**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<sup>-1</sup>]

**Group** SUEWS

**QE**

**Description** Latent heat flux (calculated using SUEWS) [W m<sup>-2</sup>]

**Group** SUEWS

**QElumps**

**Description** Latent heat flux (calculated using LUMPS) [W m<sup>-2</sup>]

**Group** SUEWS

**QF**

**Description** Anthropogenic heat flux [W m<sup>-2</sup>]

**Group** SUEWS

**QH**

**Description** Sensible heat flux (calculated using SUEWS) [W m<sup>-2</sup>]

**Group** SUEWS

**QHlumps**

**Description** Sensible heat flux (calculated using LUMPS) [W m<sup>-2</sup>]

**Group** SUEWS

**QHresis**

**Description** Sensible heat flux (calculated using resistance method) [W m<sup>-2</sup>]

**Group** SUEWS

**QM**

**Description** Snow-related heat exchange [W m<sup>-2</sup>]

**Group** SUEWS

**QMFreeze**

**Description** Internal energy change [ $\text{W m}^{-2}$ ]

**Group** SUEWS

**QMRain**

**Description** Heat released by rain on snow [ $\text{W m}^{-2}$ ]

**Group** SUEWS

**QN**

**Description** Net all-wave radiation [ $\text{W m}^{-2}$ ]

**Group** SUEWS

**QNSnow**

**Description** Net all-wave radiation for snow area [ $\text{W m}^{-2}$ ]

**Group** SUEWS

**QNSnowFr**

**Description** Net all-wave radiation for snow-free area [ $\text{W m}^{-2}$ ]

**Group** SUEWS

**QS**

**Description** Storage heat flux [ $\text{W m}^{-2}$ ]

**Group** SUEWS

**Qa\_BSoil**

**Description** Advective heat – bare soil surface [ $\text{W m}^{-2}$ ]

**Group** snow

**Qa\_Bldgs**

**Description** Advective heat – building surface [ $\text{W m}^{-2}$ ]

**Group** snow

**Qa\_DecTr**

**Description** Advective heat – deciduous surface [ $\text{W m}^{-2}$ ]

**Group** snow

**Qa\_EveTr**

**Description** Advective heat – evergreen surface [ $\text{W m}^{-2}$ ]

**Group** snow

**Qa\_Grass**

**Description** Advective heat – grass surface [ $\text{W m}^{-2}$ ]

**Group** snow

**Qa\_Paved**

**Description** Advective heat – paved surface [ $\text{W m}^{-2}$ ]

**Group** snow

**Qa\_Water**

**Description** Advective heat – water surface [ $\text{W m}^{-2}$ ]

**Group** snow

**QmFr\_BSoil**

**Description** Heat related to freezing of surface store – bare soil surface [ $\text{W m}^{-2}$ ]

**Group** snow

**QmFr\_Bldgs**

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

**Group** snow

**QmFr\_DecTr**

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

**Group** snow

**QmFr\_EveTr**

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

**Group** snow

**QmFr\_Grass**

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

**Group** snow

**QmFr\_Paved**

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

**Group** snow

**QmFr\_Water**

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

**Group** snow

**Qm\_BSoil**

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

**Group** snow

**Qm\_Bldgs**

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

**Group** snow

**Qm\_DecTr**

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

**Group** snow

**Qm\_EveTr**

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

**Group** snow

**Qm\_Grass**

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

**Group** snow

**Qm\_Paved**

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

**Group** snow

**Qm\_Water**

**Description** Snowmelt-related heat – water surface [ $\text{W m}^{-2}$ ]

**Group** snow

**RA**

**Description** Aerodynamic resistance [ $\text{s m}^{-1}$ ]

**Group** SUEWS

**RA**

**Description** Aerodynamic resistance [ $\text{s m}^{-1}$ ]

**Group** debug

**RH2**

**Description** Relative humidity at 2 m agl [%]

**Group** SUEWS

**RO**

**Description** Runoff [mm]

**Group** SUEWS

**ROImp**

**Description** Above ground runoff over impervious surfaces [mm]

**Group** SUEWS

**ROPipe**

**Description** Runoff to pipes [mm]

**Group** SUEWS

**ROSoil**

**Description** Runoff to soil (sub-surface) [mm]

**Group** SUEWS

**ROVeg**

**Description** Above ground runoff over vegetated surfaces [mm]

**Group** SUEWS

**ROWater**

**Description** Runoff for water body [mm]

**Group** SUEWS

**RS**

**Description** Surface resistance [ $\text{s m}^{-1}$ ]

**Group** SUEWS

**RS**

**Description** Surface resistance [ $\text{s m}^{-1}$ ]

**Group** debug

**Rain**

**Description** Rain [mm]

**Group** SUEWS

**RainSn\_BSoil**

**Description** Rain on snow – bare soil surface [mm]

**Group** snow

**RainSn\_Bldgs**

**Description** Rain on snow – building surface [mm]

**Group** snow

**RainSn\_DecTr**

**Description** Rain on snow – deciduous surface [mm]

**Group** snow

**RainSn\_EveTr**

**Description** Rain on snow – evergreen surface [mm]

**Group** snow

**RainSn\_Grass**

**Description** Rain on snow – grass surface [mm]

**Group** snow

**RainSn\_Paved**

**Description** Rain on snow – paved surface [mm]

**Group** snow

**RainSn\_Water**

**Description** Rain on snow – water surface [mm]

**Group** snow

**SDD\_DecTr**

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

**Group** DailyState

**SDD\_EveTr**

**Description** Senescence degree days for evergreen tree [ $^{\circ}\text{C d}$ ]

**Group** DailyState



**SDD\_Grass**

**Description** Senescence degree days for grass [°C d]

**Group** DailyState

**SMD**

**Description** Soil moisture deficit [mm]

**Group** SUEWS

**SMDBSoil**

**Description** Soil moisture deficit for bare soil surface [mm]

**Group** SUEWS

**SMDBldgs**

**Description** Soil moisture deficit for building surface [mm]

**Group** SUEWS

**SMDDecTr**

**Description** Soil moisture deficit for deciduous surface [mm]

**Group** SUEWS

**SMDEveTr**

**Description** Soil moisture deficit for evergreen surface [mm]

**Group** SUEWS

**SMDGrass**

**Description** Soil moisture deficit for grass surface [mm]

**Group** SUEWS

**SMDPaved**

**Description** Soil moisture deficit for paved surface [mm]

**Group** SUEWS

**SWE**

**Description** Snow water equivalent [mm]

**Group** SUEWS

**SWE\_BSoil**

**Description** Snow water equivalent – bare soil surface [mm]

**Group** snow

**SWE\_Bldgs**

**Description** Snow water equivalent – building surface [mm]

**Group** snow

**SWE\_DecTr**

**Description** Snow water equivalent – deciduous surface [mm]

**Group** snow

**SWE\_EveTr**

**Description** Snow water equivalent – evergreen surface [mm]

**Group** snow

**SWE\_Grass**

**Description** Snow water equivalent – grass surface [mm]

**Group** snow

**SWE\_Paved**

**Description** Snow water equivalent – paved surface [mm]

**Group** snow

**SWE\_Water**

**Description** Snow water equivalent – water surface [mm]

**Group** snow

**Sd\_BSoil**

**Description** Snow depth – bare soil surface [mm]

**Group** snow

**Sd\_Bldgs**

**Description** Snow depth – building surface [mm]

**Group** snow

**Sd\_DecTr**

**Description** Snow depth – deciduous surface [mm]

**Group** snow

**Sd\_EveTr**

**Description** Snow depth – evergreen surface [mm]

**Group** snow

**Sd\_Grass**

**Description** Snow depth – grass surface [mm]

**Group** snow

**Sd\_Paved**

**Description** Snow depth – paved surface [mm]

**Group** snow

**Sd\_Water**

**Description** Snow depth – water surface [mm]

**Group** snow

**SnowCh**

**Description** Change in snow pack [mm]

**Group** SUEWS

**SnowRBldgs**

**Description** Snow removed from building surface [mm]

**Group** SUEWS

**SnowRPaved**

**Description** Snow removed from paved surface [mm]

**Group** SUEWS

**StBSoil**

**Description** Surface wetness state for bare soil surface [mm]

**Group** SUEWS

**StBldgs**

**Description** Surface wetness state for building surface [mm]

**Group** SUEWS

**StDecTr**

**Description** Surface wetness state for deciduous tree surface [mm]

**Group** SUEWS

**StEveTr**

**Description** Surface wetness state for evergreen tree surface [mm]

**Group** SUEWS

**StGrass**

**Description** Surface wetness state for grass surface [mm]

**Group** SUEWS

**StPaved**

**Description** Surface wetness state for paved surface [mm]

**Group** SUEWS

**StWater**

**Description** Surface wetness state for water surface [mm]

**Group** SUEWS

**State**

**Description** Surface wetness state [mm]

**Group** SUEWS

**SurfCh**

**Description** Change in surface moisture store [mm]

**Group** SUEWS

**T2**

**Description** Air temperature at 2 m agl [°C]

**Group** SUEWS

**T\_1****Description** Air temperature at level 1 [°C]**Group** RSL**T\_10****Description** Air temperature at level 10 [°C]**Group** RSL**T\_11****Description** Air temperature at level 11 [°C]**Group** RSL**T\_12****Description** Air temperature at level 12 [°C]**Group** RSL**T\_13****Description** Air temperature at level 13 [°C]**Group** RSL**T\_14****Description** Air temperature at level 14 [°C]**Group** RSL**T\_15****Description** Air temperature at level 15 [°C]**Group** RSL**T\_16****Description** Air temperature at level 16 [°C]**Group** RSL**T\_17****Description** Air temperature at level 17 [°C]**Group** RSL**T\_18****Description** Air temperature at level 18 [°C]**Group** RSL**T\_19****Description** Air temperature at level 19 [°C]**Group** RSL**T\_2****Description** Air temperature at level 2 [°C]**Group** RSL

**T\_20****Description** Air temperature at level 20 [°C]**Group** RSL**T\_21****Description** Air temperature at level 21 [°C]**Group** RSL**T\_22****Description** Air temperature at level 22 [°C]**Group** RSL**T\_23****Description** Air temperature at level 23 [°C]**Group** RSL**T\_24****Description** Air temperature at level 24 [°C]**Group** RSL**T\_25****Description** Air temperature at level 25 [°C]**Group** RSL**T\_26****Description** Air temperature at level 26 [°C]**Group** RSL**T\_27****Description** Air temperature at level 27 [°C]**Group** RSL**T\_28****Description** Air temperature at level 28 [°C]**Group** RSL**T\_29****Description** Air temperature at level 29 [°C]**Group** RSL**T\_3****Description** Air temperature at level 3 [°C]**Group** RSL**T\_30****Description** Air temperature at level 30 [°C]**Group** RSL

**T\_4****Description** Air temperature at level 4 [°C]**Group** RSL**T\_5****Description** Air temperature at level 5 [°C]**Group** RSL**T\_6****Description** Air temperature at level 6 [°C]**Group** RSL**T\_7****Description** Air temperature at level 7 [°C]**Group** RSL**T\_8****Description** Air temperature at level 8 [°C]**Group** RSL**T\_9****Description** Air temperature at level 9 [°C]**Group** RSL**Ta****Description** Air temperature**Group** BEERS**Tg****Description** Surface temperature**Group** BEERS**Tmax****Description** Daily maximum temperature [°C]**Group** DailyState**Tmin****Description** Daily minimum temperature [°C]**Group** DailyState**Tmrt****Description** Mean Radiant Temperature**Group** BEERS**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<sup>-1</sup>]

**Group** SUEWS

**U\_1**

**Description** Wind speed at level 1 [m s<sup>-1</sup>]

**Group** RSL

**U\_10**

**Description** Wind speed at level 10 [m s<sup>-1</sup>]

**Group** RSL

**U\_11****Description** Wind speed at level 11 [ $\text{m s}^{-1}$ ]**Group** RSL**U\_12****Description** Wind speed at level 12 [ $\text{m s}^{-1}$ ]**Group** RSL**U\_13****Description** Wind speed at level 13 [ $\text{m s}^{-1}$ ]**Group** RSL**U\_14****Description** Wind speed at level 14 [ $\text{m s}^{-1}$ ]**Group** RSL**U\_15****Description** Wind speed at level 15 [ $\text{m s}^{-1}$ ]**Group** RSL**U\_16****Description** Wind speed at level 16 [ $\text{m s}^{-1}$ ]**Group** RSL**U\_17****Description** Wind speed at level 17 [ $\text{m s}^{-1}$ ]**Group** RSL**U\_18****Description** Wind speed at level 18 [ $\text{m s}^{-1}$ ]**Group** RSL**U\_19****Description** Wind speed at level 19 [ $\text{m s}^{-1}$ ]**Group** RSL**U\_2****Description** Wind speed at level 2 [ $\text{m s}^{-1}$ ]**Group** RSL**U\_20****Description** Wind speed at level 20 [ $\text{m s}^{-1}$ ]**Group** RSL**U\_21****Description** Wind speed at level 21 [ $\text{m s}^{-1}$ ]**Group** RSL



**U\_22****Description** Wind speed at level 22 [ $\text{m s}^{-1}$ ]**Group** RSL**U\_23****Description** Wind speed at level 23 [ $\text{m s}^{-1}$ ]**Group** RSL**U\_24****Description** Wind speed at level 24 [ $\text{m s}^{-1}$ ]**Group** RSL**U\_25****Description** Wind speed at level 25 [ $\text{m s}^{-1}$ ]**Group** RSL**U\_26****Description** Wind speed at level 26 [ $\text{m s}^{-1}$ ]**Group** RSL**U\_27****Description** Wind speed at level 27 [ $\text{m s}^{-1}$ ]**Group** RSL**U\_28****Description** Wind speed at level 28 [ $\text{m s}^{-1}$ ]**Group** RSL**U\_29****Description** Wind speed at level 29 [ $\text{m s}^{-1}$ ]**Group** RSL**U\_3****Description** Wind speed at level 3 [ $\text{m s}^{-1}$ ]**Group** RSL**U\_30****Description** Wind speed at level 30 [ $\text{m s}^{-1}$ ]**Group** RSL**U\_4****Description** Wind speed at level 4 [ $\text{m s}^{-1}$ ]**Group** RSL**U\_5****Description** Wind speed at level 5 [ $\text{m s}^{-1}$ ]**Group** RSL

**U\_6**

**Description** Wind speed at level 6 [m s<sup>-1</sup>]

**Group** RSL

**U\_7**

**Description** Wind speed at level 7 [m s<sup>-1</sup>]

**Group** RSL

**U\_8**

**Description** Wind speed at level 8 [m s<sup>-1</sup>]

**Group** RSL

**U\_9**

**Description** Wind speed at level 9 [m s<sup>-1</sup>]

**Group** RSL

**WUDecTr**

**Description** Water use for irrigation of deciduous trees [mm]

**Group** SUEWS

**WUEveTr**

**Description** Water use for irrigation of evergreen trees [mm]

**Group** SUEWS

**WUGrass**

**Description** Water use for irrigation of grass [mm]

**Group** SUEWS

**WUInt**

**Description** Internal water use [mm]

**Group** SUEWS

**WU\_DecTr1**

**Description** Total water use for deciduous trees [mm]

**Group** DailyState

**WU\_DecTr2**

**Description** Automatic water use for deciduous trees [mm]

**Group** DailyState

**WU\_DecTr3**

**Description** Manual water use for deciduous trees [mm]

**Group** DailyState

**WU\_EveTr1**

**Description** Total water use for evergreen trees [mm]

**Group** DailyState

**WU\_EveTr2**

**Description** Automatic water use for evergreen trees [mm]

**Group** DailyState

**WU\_EveTr3**

**Description** Manual water use for evergreen trees [mm]

**Group** DailyState

**WU\_Grass1**

**Description** Total water use for grass [mm]

**Group** DailyState

**WU\_Grass2**

**Description** Automatic water use for grass [mm]

**Group** DailyState

**WU\_Grass3**

**Description** Manual water use for grass [mm]

**Group** DailyState

**Zenith**

**Description** Solar zenith angle [°]

**Group** SUEWS

**a1**

**Description** OHM coefficient a1 - [-]

**Group** DailyState

**a2**

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

**Group** DailyState

**a3**

**Description** OHM coefficient a3 - [ $\text{W m}^{-2}$ ]

**Group** DailyState

**altitude**

**Description** Altitude angle of the Sun

**Group** BEERS

**azimuth**

**Description** Azimuth angle of the Sun

**Group** BEERS

**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 [ $\text{W m}^{-2}$ ]

**Group** snow

**kup\_BldgsSnow**

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

**Group** snow

**kup\_DecTrSnow**

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

**Group** snow

**kup\_EveTrSnow**

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

**Group** snow

**kup\_GrassSnow**

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

**Group** snow

**kup\_PavedSnow**

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

**Group** snow

**kup\_WaterSnow**

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

**Group** snow

**q\_1**

**Description** Specific humidity at level 1 [ $\text{g kg}^{-1}$ ]

**Group** RSL

**q\_10**

**Description** Specific humidity at level 10 [ $\text{g kg}^{-1}$ ]

**Group** RSL

**q\_11**

**Description** Specific humidity at level 11 [ $\text{g kg}^{-1}$ ]

**Group** RSL

**q\_12**

**Description** Specific humidity at level 12 [ $\text{g kg}^{-1}$ ]

**Group** RSL

**q\_13**

**Description** Specific humidity at level 13 [ $\text{g kg}^{-1}$ ]

**Group** RSL

**q\_14**

**Description** Specific humidity at level 14 [g kg<sup>-1</sup>]

**Group** RSL

**q\_15**

**Description** Specific humidity at level 15 [g kg<sup>-1</sup>]

**Group** RSL

**q\_16**

**Description** Specific humidity at level 16 [g kg<sup>-1</sup>]

**Group** RSL

**q\_17**

**Description** Specific humidity at level 17 [g kg<sup>-1</sup>]

**Group** RSL

**q\_18**

**Description** Specific humidity at level 18 [g kg<sup>-1</sup>]

**Group** RSL

**q\_19**

**Description** Specific humidity at level 19 [g kg<sup>-1</sup>]

**Group** RSL

**q\_2**

**Description** Specific humidity at level 2 [g kg<sup>-1</sup>]

**Group** RSL

**q\_20**

**Description** Specific humidity at level 20 [g kg<sup>-1</sup>]

**Group** RSL

**q\_21**

**Description** Specific humidity at level 21 [g kg<sup>-1</sup>]

**Group** RSL

**q\_22**

**Description** Specific humidity at level 22 [g kg<sup>-1</sup>]

**Group** RSL

**q\_23**

**Description** Specific humidity at level 23 [g kg<sup>-1</sup>]

**Group** RSL

**q\_24**

**Description** Specific humidity at level 24 [g kg<sup>-1</sup>]

**Group** RSL

**q\_25**

**Description** Specific humidity at level 25 [g kg<sup>-1</sup>]

**Group** RSL

**q\_26**

**Description** Specific humidity at level 26 [g kg<sup>-1</sup>]

**Group** RSL

**q\_27**

**Description** Specific humidity at level 27 [g kg<sup>-1</sup>]

**Group** RSL

**q\_28**

**Description** Specific humidity at level 28 [g kg<sup>-1</sup>]

**Group** RSL

**q\_29**

**Description** Specific humidity at level 29 [g kg<sup>-1</sup>]

**Group** RSL

**q\_3**

**Description** Specific humidity at level 3 [g kg<sup>-1</sup>]

**Group** RSL

**q\_30**

**Description** Specific humidity at level 30 [g kg<sup>-1</sup>]

**Group** RSL

**q\_4**

**Description** Specific humidity at level 4 [g kg<sup>-1</sup>]

**Group** RSL

**q\_5**

**Description** Specific humidity at level 5 [g kg<sup>-1</sup>]

**Group** RSL

**q\_6**

**Description** Specific humidity at level 6 [g kg<sup>-1</sup>]

**Group** RSL

**q\_7**

**Description** Specific humidity at level 7 [g kg<sup>-1</sup>]

**Group** RSL

**q\_8**

**Description** Specific humidity at level 8 [g kg<sup>-1</sup>]

**Group** RSL

**q\_9****Description** Specific humidity at level 9 [g kg<sup>-1</sup>]**Group** RSL**z0m****Description** Roughness length for momentum [m]**Group** SUEWS**z\_1****Description** Height at level 1 [m]**Group** RSL**z\_10****Description** Height at level 10 [m]**Group** RSL**z\_11****Description** Height at level 11 [m]**Group** RSL**z\_12****Description** Height at level 12 [m]**Group** RSL**z\_13****Description** Height at level 13 [m]**Group** RSL**z\_14****Description** Height at level 14 [m]**Group** RSL**z\_15****Description** Height at level 15 [m]**Group** RSL**z\_16****Description** Height at level 16 [m]**Group** RSL**z\_17****Description** Height at level 17 [m]**Group** RSL**z\_18****Description** Height at level 18 [m]**Group** RSL



**z\_19**

**Description** Height at level 19 [m]

**Group** RSL

**z\_2**

**Description** Height at level 2 [m]

**Group** RSL

**z\_20**

**Description** Height at level 20 [m]

**Group** RSL

**z\_21**

**Description** Height at level 21 [m]

**Group** RSL

**z\_22**

**Description** Height at level 22 [m]

**Group** RSL

**z\_23**

**Description** Height at level 23 [m]

**Group** RSL

**z\_24**

**Description** Height at level 24 [m]

**Group** RSL

**z\_25**

**Description** Height at level 25 [m]

**Group** RSL

**z\_26**

**Description** Height at level 26 [m]

**Group** RSL

**z\_27**

**Description** Height at level 27 [m]

**Group** RSL

**z\_28**

**Description** Height at level 28 [m]

**Group** RSL

**z\_29**

**Description** Height at level 29 [m]

**Group** RSL

**z\_3**

**Description** Height at level 3 [m]

**Group** RSL

**z\_30**

**Description** Height at level 30 [m]

**Group** RSL

**z\_4**

**Description** Height at level 4 [m]

**Group** RSL

**z\_5**

**Description** Height at level 5 [m]

**Group** RSL

**z\_6**

**Description** Height at level 6 [m]

**Group** RSL

**z\_7**

**Description** Height at level 7 [m]

**Group** RSL

**z\_8**

**Description** Height at level 8 [m]

**Group** RSL

**z\_9**

**Description** Height at level 9 [m]

**Group** RSL

**zdm**

**Description** Zero-plane displacement height [m]

**Group** SUEWS

### Contents

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

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

please check if your environment meets the following requirements:

### 1. Operating system (OS):

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

You can get the OS information with the following code:

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

### 2. Python interpreter:

- a. is your Python interpreter 64 bit?

Check running mode with the following code:

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

- b. is your Python version above 3.5?

Check version info with the following code:

```
import sys
sys.version
```

If your environment doesn't meet the requirement by SuPy, please use a proper environment; otherwise, **'please report your issue'\_\_**.

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

Use the following code:

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

---

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

---

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

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

Please **'report such issues to us'\_\_** with your input files for debugging. Thanks!

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

Run the following code in your terminal:

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

### 5.1 Version 2020.11.3

- **New**
  1. Update supy-driver to 2020b iteration.
  2. Add function for plotting RSL variables `supy.util.plot_rsl`.
- **Improvement**
  1. The RSL related functions are more robust in dealing with broader urban morphology settings.
  2. Internal changes to conform with recent upgrades in `pandas`.
- **Changes**

None.
- **Fix**
  1. Fix an issue in `supy.util.read_forcing` that improper resampling could be conducted if input temporal resolution is the same as the desirable resampling time step `tstep_mod`.
- **Known issue**
  1. ESTM is not supported yet.
  2. BLUEWS, a CBL modules in SUEWS, is not supported yet.
  3. Simulation in parallel mode is NOT supported on Windows due to system limitation.

### 5.2 Version 2020.5.29

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

2. Add function for plotting RSL variables `supy.util.plot_rsl`.

- **Improvement**

None.

- **Changes**

None.

- **Fix**

1. Fix the humidity variable in ERA5-based forcing generation.
2. Fix the impact study tutorial.

- **Known issue**

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

## 5.3 Version 2020.2.2

- **New**

1. A checker to validate input `DataFrame`'s. See option ``check_input`` in `run_supy`.
2. Utilities to generate forcing data using ERA-5 data. See `download_era5` and `gen_forcing_era5`.

- **Improvement**

1. Improved performance of the parallel mode.

- **Changes**

None.

- **Fix**

None.

- **Known issue**

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

## 5.4 Version 2019.8.29

- **New**

1. added WRF-SUEWS related functions.
2. added `diagnostics of canyon profiles`.

- **Improvement**

None.

- **Changes**

1. synchronised with v2019a interface: minimum supy\_driver v2019a2.

- **Fix**

None.

- **Known issue**

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

## 5.5 Version 2019.7.17

- **New**

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

- **Improvement**

None.

- **Changes**

None.

- **Fix**

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

- **Known issue**

ESTM is not supported yet.

## 5.6 Version 2019.6.8

- **New**

None.

- **Improvement**

None.

- **Changes**

None.

- **Fix**

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

- **Known issue**

ESTM is not supported yet.

## 5.7 Version 2019.5.28

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

- **New**

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

- **Improvement**

None.

- **Changes**

None.

- **Fix**

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

- **Known issue**

ESTM is not supported yet.

## 5.8 Version 2019.4.29

Parallel run.

- **New**

Added support for parallel run on the fly.

- **Improvement**

None.

- **Changes**

None.

- **Fix**

None.

- **Known issue**

None

## 5.9 Version 2019.4.17

UMEP compatibility tweaks.

- **New**

None.



- **Improvement**

None.

- **Changes**

Error messages: `problems.txt` will be written out in addition to the console error message similarly as SUEWS binary.

- **Fix**

Incorrect caching of input libraries.

- **Known issue**

None

## 5.10 Version 2019.4.15

ERA-5 download.

- **New**

Added experimental support for downloading and processing ERA-5 data to force supy simulations.

- **Improvement**

Improved compatibility with earlier `pandas` version in resampling output.

- **Changes**

None.

- **Fix**

None.

- **Known issue**

None

## 5.11 Version 2019.3.21

TMY generation.

- **New**

Added preliminary support for generating TMY dataset with SuPy output.

- **Improvement**

None.

- **Changes**

None.

- **Fix**

None.

- **Known issue**

None

## 5.12 Version 2019.3.14

This release improved memory usage.

- **New**  
None.
- **Improvement**  
Optimised memory consumption for longterm simulations.
- **Changes**  
None.
- **Fix**  
None.
- **Known issue**  
None

## 5.13 Version 2019.2.25

This release dropped support for Python 3.5 and below.

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

## 5.14 Version 2019.2.24

This release added the ability to save output files.

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

- **Changes**

None.

- **Fix**

None.

- **Known issue**

None

## 5.15 Version 2019.2.19

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

- **New**

Added support to handle python kernel crash caused by fatal error in `supy_driver` kernel; so python kernel won't crash any more even `supy_driver` is stopped.

- **Improvement**

None.

- **Changes**

None

- **Fix**

None.

- **Known issue**

None

## 5.16 Version 2019.2.8

This is a release that fixes recent bugs found in SUEWS that may lead to abnormal simulation results of storage heat flux, in particular when `SnowUse` is enabled (i.e., `snowuse=1`).

- **New**

None.

- **Improvement**

Improved the performance in loading initial model state from a large number of grids (>1k)

- **Changes**

Updated `SampleRun` dataset by: 1. setting surface fractions (`sfr`) to a more realistic value based on London KCL case; 2. enabling snow module (`snowuse=1`).

- **Fix**

1. Fixed a bug in the calculation of storage heat flux.
2. Fixed a bug in loading `popdens` for calculating anthropogenic heat flux.

- **Known issue**

None

## 5.17 Version 2019.1.1 (preview release, 01 Jan 2019)

- **New**
  1. Slimmed the output groups by excluding unsupported **ESTM** results
  2. SuPy documentation
    - Key IO data structures documented:
    - *df\_output variables* (GH9)
    - *df\_state variables* (GH8)
    - *df\_forcing variables* (GH7)
    - Tutorial of parallel SuPy simulations for impact studies
- **Improvement**
  1. Improved calculation of OHM-related radiation terms
- **Changes**

None.
- **Fix**

None
- **Known issue**

None

## 5.18 Version 2018.12.15 (internal test release in December 2018)

- **New**
  1. Preview release of SuPy based on the computation kernel of SUEWS 2018b
- **Improvement**
  1. Improved calculation of OHM-related radiation terms
- **Changes**

None.
- **Fix**

None
- **Known issue**
  1. The heat storage modules AnOHM and ESTM are not supported yet.

## Symbols

-f, -from <fromVer>  
    suews-convert command line option,  
    76

-i, -input <fromDir>  
    suews-convert command line option,  
    76

-o, -output <toDir>  
    suews-convert command line option,  
    76

-t, -to <toVer>  
    suews-convert command line option,  
    76

## A

a1  
    command line option, 127

a2  
    command line option, 127

a3  
    command line option, 127

AddWater  
    command line option, 103

aerodynamicresistancemethod  
    command line option, 76

ah\_min  
    command line option, 77

ah\_slope\_cooling  
    command line option, 77

ah\_slope\_heating  
    command line option, 77

ahprof\_24hr  
    command line option, 77

alb  
    command line option, 77

AlbBulk  
    command line option, 103

AlbDecTr  
    command line option, 103

albdectr\_id  
    command line option, 77

AlbEveTr  
    command line option, 103

albevetr\_id  
    command line option, 77

AlbGrass  
    command line option, 104

albgrass\_id  
    command line option, 78

albmax\_dectr  
    command line option, 78

albmax\_evetr  
    command line option, 78

albmax\_grass  
    command line option, 78

albmin\_dectr  
    command line option, 78

albmin\_evetr  
    command line option, 78

albmin\_grass  
    command line option, 78

AlbSnow  
    command line option, 104

alpha\_bioco2  
    command line option, 79

alpha\_enh\_bioco2  
    command line option, 79

alt  
    command line option, 79

altitude  
    command line option, 127

Azimuth  
    command line option, 104

azimuth  
    command line option, 127

## B

baset  
    command line option, 79

baset\_cooling  
     command line option, 79  
 baset\_hc  
     command line option, 79  
 baset\_heating  
     command line option, 79  
 basete  
     command line option, 80  
 basetmethod  
     command line option, 80  
 beta\_bioco2  
     command line option, 80  
 beta\_enh\_bioco2  
     command line option, 80  
 bldgh  
     command line option, 80

## C

cal\_gs\_mod() *(in module supy.util)*, 71  
 cal\_gs\_obs() *(in module supy.util)*, 72  
 cal\_neutral() *(in module supy.util)*, 75  
 calib\_g() *(in module supy.util)*, 72  
 capmax\_dec  
     command line option, 80  
 capmin\_dec  
     command line option, 80  
 chanohm  
     command line option, 80  
 CI  
     command line option, 104  
 co2pointsource  
     command line option, 81  
 command line option  
     a1, 127  
     a2, 127  
     a3, 127  
     AddWater, 103  
     aerodynamicresistancemethod, 76  
     ah\_min, 77  
     ah\_slope\_cooling, 77  
     ah\_slope\_heating, 77  
     ahprof\_24hr, 77  
     alb, 77  
     AlbBulk, 103  
     AlbDecTr, 103  
     albdectr\_id, 77  
     AlbEveTr, 103  
     albevetr\_id, 77  
     AlbGrass, 104  
     albgrass\_id, 78  
     albmax\_dectr, 78  
     albmax\_evetr, 78  
     albmax\_grass, 78  
     albmin\_dectr, 78

albmin\_evetr, 78  
 albmin\_grass, 78  
 AlbSnow, 104  
 alpha\_bioco2, 79  
 alpha\_enh\_bioco2, 79  
 alt, 79  
 altitude, 127  
 Azimuth, 104  
 azimuth, 127  
 baset, 79  
 baset\_cooling, 79  
 baset\_hc, 79  
 baset\_heating, 79  
 basete, 80  
 basetmethod, 80  
 beta\_bioco2, 80  
 beta\_enh\_bioco2, 80  
 bldgh, 80  
 capmax\_dec, 80  
 capmin\_dec, 80  
 chanohm, 80  
 CI, 104  
 co2pointsource, 81  
 cpanohm, 81  
 crwmax, 81  
 crwmin, 81  
 DaysSR, 104  
 daywat, 81  
 daywatper, 81  
 DecidCap, 104  
 decidcap\_id, 81  
 dectreeh, 82  
 deltaLAI, 127  
 DensSnow\_Bldgs, 104, 105  
 DensSnow\_BSoil, 104  
 DensSnow\_DecTr, 105  
 DensSnow\_EveTr, 105  
 DensSnow\_Grass, 105, 106  
 DensSnow\_Paved, 106  
 DensSnow\_Water, 106  
 diagnose, 82  
 diagqn, 82  
 diagqs, 82  
 DiffuseRad, 106  
 DirectRad, 107  
 DLHrs, 104  
 Drainage, 107  
 drainrt, 82  
 ef\_umolco2perj, 82  
 emis, 82  
 emissionsmethod, 83  
 enddls, 83  
 enef\_v\_jkm, 83  
 Evap, 107

---

evapmethod, 83  
 evetreeh, 83  
 faibldg, 83  
 faidectree, 83  
 faievetree, 84  
 faut, 84  
 Fc, 107  
 FcBuild, 107  
 fcef\_v\_kgkm, 84  
 Fcld, 107  
 fcld, 102  
 FcMetab, 107  
 FcPhoto, 107  
 FcPoint, 107  
 FcRespi, 107  
 FcTraff, 107  
 FlowCh, 107  
 flowchange, 84  
 fr\_Bldgs, 128  
 fr\_DecTr, 128  
 fr\_EveTr, 128  
 fr\_Grass, 128  
 fr\_Paved, 128  
 frfossilfuel\_heat, 84  
 frfossilfuel\_nonheat, 84  
 frMelt\_Bldgs, 128  
 frMelt\_BSoil, 127  
 frMelt\_DecTr, 128  
 frMelt\_EveTr, 128  
 frMelt\_Grass, 128  
 frMelt\_Paved, 128  
 frMelt\_Water, 128  
 g1, 84  
 g2, 85  
 g3, 85  
 g4, 85  
 g5, 85  
 g6, 85  
 GDD\_DecTr, 108  
 GDD\_EveTr, 108  
 GDD\_Grass, 108  
 gddfull, 85  
 GlobalRad, 108  
 gsmodel, 85  
 h\_maintain, 86  
 HDD1\_h, 108  
 HDD2\_c, 108  
 HDD3\_Tmean, 108  
 HDD4\_T5d, 108  
 humactivity\_24hr, 86  
 I0, 108  
 id, 102  
 ie\_a, 86  
 ie\_end, 86  
 ie\_m, 86  
 ie\_start, 86  
 imin, 102  
 internalwateruse\_h, 86  
 Irr, 108  
 irrfracbldgs, 87  
 irrfracbsoil, 87  
 irrfracdectr, 87  
 irrfracevetr, 87  
 irrfracgrass, 87  
 irrfracpaved, 87  
 irrfracwater, 87  
 isec, 102  
 it, 102  
 iy, 102  
 kdiff, 102  
 kdir, 102  
 Kdown, 108  
 kdown, 102  
 Kdown2d, 108  
 Keast, 109  
 kkanohm, 88  
 kmax, 88  
 Knorth, 109  
 Ksouth, 109  
 Kup, 109  
 Kup2d, 109  
 kup\_BldgsSnow, 129  
 kup\_BSoilsSnow, 128  
 kup\_DecTrSnow, 129  
 kup\_EveTrSnow, 129  
 kup\_GrassSnow, 129  
 kup\_PavedSnow, 129  
 kup\_WaterSnow, 129  
 Kwest, 109  
 LAI, 109  
 lai, 102  
 LAI\_DecTr, 109  
 LAI\_EveTr, 109  
 LAI\_Grass, 109  
 lai\_id, 88  
 laicalcyes, 88  
 LAIlumps, 109  
 laimax, 88  
 laimin, 88  
 laipower, 88  
 laitype, 89  
 lat, 89  
 Ldown, 109  
 ldown, 102  
 Ldown2d, 110  
 Least, 110  
 lng, 89  
 Lnorth, 110

Lob, 110  
Lsouth, 110  
Lup, 110  
Lup2d, 110  
Lwest, 110  
maxconductance, 89  
maxfcmetab, 89  
maxqfmetab, 89  
MeltWater, 110  
MeltWStore, 110  
min\_res\_bioco2, 89  
minfcmetab, 90  
minqfmetab, 90  
Mw\_Bldgs, 111  
Mw\_BSoil, 111  
Mw\_DecTr, 111  
Mw\_EveTr, 111  
Mw\_Grass, 111  
Mw\_Paved, 111  
Mw\_Water, 111  
MwStore\_Bldgs, 110  
MwStore\_BSoil, 110  
MwStore\_DecTr, 111  
MwStore\_EveTr, 111  
MwStore\_Grass, 111  
MwStore\_Paved, 111  
MwStore\_Water, 111  
narp\_emis\_snow, 90  
narp\_trans\_site, 90  
netradiationmethod, 90  
NWtrState, 112  
ohm\_coef, 90  
ohm\_threshsw, 90  
ohm\_threshwd, 91  
ohmincqf, 91  
P\_day, 112  
pipecapacity, 91  
popdensdaytime, 91  
popdensnighttime, 91  
popprof\_24hr, 91  
pormax\_dec, 92  
pormin\_dec, 92  
Porosity, 112  
porosity\_id, 92  
preciplimit, 92  
preciplimitalb, 92  
pres, 102  
Q2, 112  
q\_1, 129  
q\_10, 129  
q\_11, 129  
q\_12, 129  
q\_13, 129  
q\_14, 129  
q\_15, 130  
q\_16, 130  
q\_17, 130  
q\_18, 130  
q\_19, 130  
q\_2, 130  
q\_20, 130  
q\_21, 130  
q\_22, 130  
q\_23, 130  
q\_24, 130  
q\_25, 130  
q\_26, 131  
q\_27, 131  
q\_28, 131  
q\_29, 131  
q\_3, 131  
q\_30, 131  
q\_4, 131  
q\_5, 131  
q\_6, 131  
q\_7, 131  
q\_8, 131  
q\_9, 131  
Qa\_Bldgs, 113  
Qa\_BSoil, 113  
Qa\_DecTr, 113  
Qa\_EveTr, 113  
Qa\_Grass, 113  
Qa\_Paved, 113  
Qa\_Water, 113  
QE, 112  
qe, 103  
QElumps, 112  
QF, 112  
qf, 103  
qf0\_beu, 92  
qf\_a, 92  
qf\_b, 93  
qf\_c, 93  
QH, 112  
qh, 103  
QHlumps, 112  
QHresis, 112  
QM, 112  
Qm\_Bldgs, 114  
Qm\_BSoil, 114  
Qm\_DecTr, 114  
Qm\_EveTr, 114  
Qm\_Grass, 114  
Qm\_Paved, 115  
Qm\_Water, 115  
QmFr\_Bldgs, 114  
QmFr\_BSoil, 114



QmFr\_DecTr, 114  
 QmFr\_EveTr, 114  
 QmFr\_Grass, 114  
 QmFr\_Paved, 114  
 QmFr\_Water, 114  
 QMFreeze, 112  
 QMRain, 113  
 QN, 113  
 qn, 103  
 QNSnow, 113  
 QNSnowFr, 113  
 QS, 113  
 qs, 103  
 RA, 115  
 radmeltfact, 93  
 Rain, 116  
 rain, 103  
 raincover, 93  
 rainmaxres, 93  
 RainSn\_Bldgs, 116  
 RainSn\_BSoil, 116  
 RainSn\_DecTr, 116  
 RainSn\_EveTr, 116  
 RainSn\_Grass, 116  
 RainSn\_Paved, 116  
 RainSn\_Water, 116  
 resp\_a, 93  
 resp\_b, 93  
 RH, 102  
 RH2, 115  
 RO, 115  
 ROImp, 115  
 ROPipe, 115  
 ROSoil, 115  
 roughlenheatmethod, 94  
 roughlenmommmethod, 94  
 ROVeg, 115  
 ROWater, 115  
 RS, 115, 116  
 runofftowater, 94  
 s1, 94  
 s2, 94  
 sathydraulicconduct, 94  
 Sd\_Bldgs, 118  
 Sd\_BSoil, 118  
 Sd\_DecTr, 118  
 Sd\_EveTr, 118  
 Sd\_Grass, 118  
 Sd\_Paved, 118  
 Sd\_Water, 118  
 SDD\_DecTr, 116  
 SDD\_EveTr, 116  
 SDD\_Grass, 116  
 sddfull, 94  
 sfr, 95  
 SMD, 117  
 SMDBldgs, 117  
 SMDBSoil, 117  
 SMDDecTr, 117  
 SMDEveTr, 117  
 SMDGrass, 117  
 smdmethod, 95  
 SMDPaved, 117  
 snow, 103  
 snowalb, 95  
 snowalbmax, 95  
 snowalbmin, 95  
 SnowCh, 118  
 snowdens, 95  
 snowdensmax, 95  
 snowdensmin, 96  
 snowfrac, 96  
 snowlimbldg, 96  
 snowlimpaved, 96  
 snowpack, 96  
 snowpacklimit, 96  
 snowprof\_24hr, 96  
 SnowRBldgs, 118  
 SnowRPaved, 119  
 snowuse, 97  
 snowwater, 97  
 soildepth, 97  
 soilstore\_id, 97  
 soilstorecap, 97  
 stabilitymethod, 97  
 startdls, 97  
 State, 119  
 state\_id, 98  
 statelimit, 98  
 StBldgs, 119  
 StBSoil, 119  
 StDecTr, 119  
 StEveTr, 119  
 StGrass, 119  
 storageheatmethod, 98  
 storedrainprm, 98  
 StPaved, 119  
 StWater, 119  
 surfacearea, 98  
 SurfCh, 119  
 SWE, 117  
 SWE\_Bldgs, 117  
 SWE\_BSoil, 117  
 SWE\_DecTr, 117  
 SWE\_EveTr, 117  
 SWE\_Grass, 118  
 SWE\_Paved, 118  
 SWE\_Water, 118

T2, 119  
T\_1, 119  
T\_10, 120  
T\_11, 120  
T\_12, 120  
T\_13, 120  
T\_14, 120  
T\_15, 120  
T\_16, 120  
T\_17, 120  
T\_18, 120  
T\_19, 120  
T\_2, 120  
T\_20, 120  
T\_21, 121  
T\_22, 121  
T\_23, 121  
T\_24, 121  
T\_25, 121  
T\_26, 121  
T\_27, 121  
T\_28, 121  
T\_29, 121  
T\_3, 121  
T\_30, 121  
T\_4, 121  
T\_5, 122  
T\_6, 122  
T\_7, 122  
T\_8, 122  
T\_9, 122  
Ta, 122  
Tair, 102  
tau\_a, 98  
tau\_f, 99  
tau\_r, 99  
tempmeltfact, 99  
Tg, 122  
th, 99  
theta\_bioco2, 99  
timezone, 99  
tl, 99  
Tmax, 122  
Tmin, 122  
Tmrt, 122  
TotCh, 122  
traffirate, 100  
traffunits, 100  
traffprof\_24hr, 100  
Ts, 122  
Tsnow\_Bldgs, 123  
Tsnow\_BSoil, 123  
Tsnow\_DecTr, 123  
Tsnow\_EveTr, 123  
Tsnow\_Grass, 123  
Tsnow\_Paved, 123  
Tsnow\_Water, 123  
tstep, 100  
Tsurf, 123  
U, 102  
U10, 123  
U\_1, 123  
U\_10, 123  
U\_11, 123  
U\_12, 124  
U\_13, 124  
U\_14, 124  
U\_15, 124  
U\_16, 124  
U\_17, 124  
U\_18, 124  
U\_19, 124  
U\_2, 124  
U\_20, 124  
U\_21, 124  
U\_22, 124  
U\_23, 125  
U\_24, 125  
U\_25, 125  
U\_26, 125  
U\_27, 125  
U\_28, 125  
U\_29, 125  
U\_3, 125  
U\_30, 125  
U\_4, 125  
U\_5, 125  
U\_6, 125  
U\_7, 126  
U\_8, 126  
U\_9, 126  
veg\_type, 100  
waterdist, 100  
waterusemethod, 100  
wdir, 103  
wetthresh, 101  
WU\_DecTr1, 126  
WU\_DecTr2, 126  
WU\_DecTr3, 126  
WU\_EveTr1, 126  
WU\_EveTr2, 126  
WU\_EveTr3, 127  
WU\_Grass1, 127  
WU\_Grass2, 127  
WU\_Grass3, 127  
WUDecTr, 126  
WUEveTr, 126  
WUGrass, 126

- Wuh, 102
- WUInt, 126
- wuprofa\_24hr, 101
- wuprofm\_24hr, 101
- xsmd, 103
- z, 101
- z0m, 132
- z0m\_in, 101
- z\_1, 132
- z\_10, 132
- z\_11, 132
- z\_12, 132
- z\_13, 132
- z\_14, 132
- z\_15, 132
- z\_16, 132
- z\_17, 132
- z\_18, 132
- z\_19, 132
- z\_2, 133
- z\_20, 133
- z\_21, 133
- z\_22, 133
- z\_23, 133
- z\_24, 133
- z\_25, 133
- z\_26, 133
- z\_27, 133
- z\_28, 133
- z\_29, 133
- z\_3, 133
- z\_30, 134
- z\_4, 134
- z\_5, 134
- z\_6, 134
- z\_7, 134
- z\_8, 134
- z\_9, 134
- zdm, 134
- zdm\_in, 101
- Zenith, 127
- cpanohm
  - command line option, 81
- crwmax
  - command line option, 81
- crwmin
  - command line option, 81
- D**
- DaysSR
  - command line option, 104
- daywat
  - command line option, 81
- daywatper
  - command line option, 81
- DecidCap
  - command line option, 104
- decidcap\_id
  - command line option, 81
- dectreeh
  - command line option, 82
- deltaLAI
  - command line option, 127
- DensSnow\_Bldgs
  - command line option, 104, 105
- DensSnow\_BSoil
  - command line option, 104
- DensSnow\_DecTr
  - command line option, 105
- DensSnow\_EveTr
  - command line option, 105
- DensSnow\_Grass
  - command line option, 105, 106
- DensSnow\_Paved
  - command line option, 106
- DensSnow\_Water
  - command line option, 106
- derive\_ohm\_coef() (*in module supy.util*), 70
- diagnose
  - command line option, 82
- diagqn
  - command line option, 82
- diagqs
  - command line option, 82
- DiffuseRad
  - command line option, 106
- DirectRad
  - command line option, 107
- DLHrs
  - command line option, 104
- download\_era5() (*in module supy.util*), 67
- Drainage
  - command line option, 107
- drainrt
  - command line option, 82
- E**
- ef\_umolco2perj
  - command line option, 82
- emis
  - command line option, 82
- emissionsmethod
  - command line option, 83
- enddls
  - command line option, 83
- enef\_v\_jkm
  - command line option, 83
- Evap

command line option, 107  
evapmethod  
command line option, 83  
evetreeh  
command line option, 83  
extract\_reclassification() (in module *supy.util*), 73

## F

faibldg  
command line option, 83  
faidectree  
command line option, 83  
faievetree  
command line option, 84  
faut  
command line option, 84  
Fc  
command line option, 107  
FcBuild  
command line option, 107  
fcef\_v\_kgkm  
command line option, 84  
Fcld  
command line option, 107  
fcld  
command line option, 102  
FcMetab  
command line option, 107  
FcPhoto  
command line option, 107  
FcPoint  
command line option, 107  
FcRespi  
command line option, 107  
FcTraff  
command line option, 107  
fill\_gap\_all() (in module *supy.util*), 70  
FlowCh  
command line option, 107  
flowchange  
command line option, 84  
fr\_Bldgs  
command line option, 128  
fr\_DecTr  
command line option, 128  
fr\_EveTr  
command line option, 128  
fr\_Grass  
command line option, 128  
fr\_Paved  
command line option, 128  
frfossilfuel\_heat  
command line option, 84

frfossilfuel\_nonheat  
command line option, 84  
frMelt\_Bldgs  
command line option, 128  
frMelt\_BSoil  
command line option, 127  
frMelt\_DecTr  
command line option, 128  
frMelt\_EveTr  
command line option, 128  
frMelt\_Grass  
command line option, 128  
frMelt\_Paved  
command line option, 128  
frMelt\_Water  
command line option, 128

## G

g1  
command line option, 84  
g2  
command line option, 85  
g3  
command line option, 85  
g4  
command line option, 85  
g5  
command line option, 85  
g6  
command line option, 85  
GDD\_DecTr  
command line option, 108  
GDD\_EveTr  
command line option, 108  
GDD\_Grass  
command line option, 108  
gddfull  
command line option, 85  
gen\_epw() (in module *supy.util*), 69  
gen\_forcing\_era5() (in module *supy.util*), 68  
GlobalRad  
command line option, 108  
gsmodel  
command line option, 85

## H

h\_maintain  
command line option, 86  
HDD1\_h  
command line option, 108  
HDD2\_c  
command line option, 108  
HDD3\_Tmean  
command line option, 108

HDD4\_T5d  
     command line option, 108  
 humactivity\_24hr  
     command line option, 86

## I

I0  
     command line option, 108  
 id  
     command line option, 102  
 ie\_a  
     command line option, 86  
 ie\_end  
     command line option, 86  
 ie\_m  
     command line option, 86  
 ie\_start  
     command line option, 86  
 imin  
     command line option, 102  
 init\_supy() (*in module supy*), 63  
 internalwateruse\_h  
     command line option, 86  
 Irr  
     command line option, 108  
 irrfracbldgs  
     command line option, 87  
 irrfracbsoil  
     command line option, 87  
 irrfracdectr  
     command line option, 87  
 irrfracevetr  
     command line option, 87  
 irrfracgrass  
     command line option, 87  
 irrfracpaved  
     command line option, 87  
 irrfracwater  
     command line option, 87  
 isec  
     command line option, 102  
 it  
     command line option, 102  
 iy  
     command line option, 102

## K

kdiff  
     command line option, 102  
 kdir  
     command line option, 102  
 Kdown  
     command line option, 108  
 kdown

    command line option, 102  
 Kdown2d  
     command line option, 108  
 Keast  
     command line option, 109  
 kkanohm  
     command line option, 88  
 kmax  
     command line option, 88  
 Knorth  
     command line option, 109  
 Ksouth  
     command line option, 109  
 Kup  
     command line option, 109  
 Kup2d  
     command line option, 109  
 kup\_BldgsSnow  
     command line option, 129  
 kup\_BSoilSnow  
     command line option, 128  
 kup\_DecTrSnow  
     command line option, 129  
 kup\_EveTrSnow  
     command line option, 129  
 kup\_GrassSnow  
     command line option, 129  
 kup\_PavedSnow  
     command line option, 129  
 kup\_WaterSnow  
     command line option, 129  
 Kwest  
     command line option, 109

## L

LAI  
     command line option, 109  
 lai  
     command line option, 102  
 LAI\_DecTr  
     command line option, 109  
 LAI\_EveTr  
     command line option, 109  
 LAI\_Grass  
     command line option, 109  
 lai\_id  
     command line option, 88  
 laicalcyes  
     command line option, 88  
 LAIlumps  
     command line option, 109  
 laimax  
     command line option, 88  
 laimin

- command line option, 88
- laipower
  - command line option, 88
- laitype
  - command line option, 89
- lat
  - command line option, 89
- Ldown
  - command line option, 109
- ldown
  - command line option, 102
- Ldown2d
  - command line option, 110
- Least
  - command line option, 110
- lng
  - command line option, 89
- Lnorth
  - command line option, 110
- load\_forcing\_grid() (*in module supy*), 64
- load\_SampleData() (*in module supy*), 67
- Lob
  - command line option, 110
- Lsouth
  - command line option, 110
- Lup
  - command line option, 110
- Lup2d
  - command line option, 110
- Lwest
  - command line option, 110

## M

- maxconductance
  - command line option, 89
- maxfcmetab
  - command line option, 89
- maxqfmetab
  - command line option, 89
- MeltWater
  - command line option, 110
- MeltWStore
  - command line option, 110
- min\_res\_bioco2
  - command line option, 89
- minfcmetab
  - command line option, 90
- minqfmetab
  - command line option, 90
- Mw\_Bldgs
  - command line option, 111
- Mw\_BSoil
  - command line option, 111
- Mw\_DecTr

- command line option, 111
- Mw\_EveTr
  - command line option, 111
- Mw\_Grass
  - command line option, 111
- Mw\_Paved
  - command line option, 111
- Mw\_Water
  - command line option, 111
- MwStore\_Bldgs
  - command line option, 110
- MwStore\_BSoil
  - command line option, 110
- MwStore\_DecTr
  - command line option, 111
- MwStore\_EveTr
  - command line option, 111
- MwStore\_Grass
  - command line option, 111
- MwStore\_Paved
  - command line option, 111
- MwStore\_Water
  - command line option, 111

## N

- narp\_emis\_snow
  - command line option, 90
- narp\_trans\_site
  - command line option, 90
- netradiationmethod
  - command line option, 90
- NWtrState
  - command line option, 112

## O

- ohm\_coef
  - command line option, 90
- ohm\_threshsw
  - command line option, 90
- ohm\_threshwd
  - command line option, 91
- ohmincqw
  - command line option, 91
- optimize\_MO() (*in module supy.util*), 75

## P

- P\_day
  - command line option, 112
- PATH\_RUNCONTROL
  - suews-run command line option, 76
- pipecapacity
  - command line option, 91
- plot\_comp() (*in module supy.util*), 74
- plot\_day\_clm() (*in module supy.util*), 74

`plot_reclassification()` (in module *supy.util*),  
 73  
`plot_rsl()` (in module *supy.util*), 74  
`popdensdaytime`  
 command line option, 91  
`popdensnighttime`  
 command line option, 91  
`popprof_24hr`  
 command line option, 91  
`pormax_dec`  
 command line option, 92  
`pormin_dec`  
 command line option, 92  
`Porosity`  
 command line option, 112  
`porosity_id`  
 command line option, 92  
`preciplimit`  
 command line option, 92  
`preciplimitalb`  
 command line option, 92  
`pres`  
 command line option, 102  
**Q**  
`Q2`  
 command line option, 112  
`q_1`  
 command line option, 129  
`q_10`  
 command line option, 129  
`q_11`  
 command line option, 129  
`q_12`  
 command line option, 129  
`q_13`  
 command line option, 129  
`q_14`  
 command line option, 129  
`q_15`  
 command line option, 130  
`q_16`  
 command line option, 130  
`q_17`  
 command line option, 130  
`q_18`  
 command line option, 130  
`q_19`  
 command line option, 130  
`q_2`  
 command line option, 130  
`q_20`  
 command line option, 130  
`q_21`  
 command line option, 130  
`q_22`  
 command line option, 130  
`q_23`  
 command line option, 130  
`q_24`  
 command line option, 130  
`q_25`  
 command line option, 130  
`q_26`  
 command line option, 131  
`q_27`  
 command line option, 131  
`q_28`  
 command line option, 131  
`q_29`  
 command line option, 131  
`q_3`  
 command line option, 131  
`q_30`  
 command line option, 131  
`q_4`  
 command line option, 131  
`q_5`  
 command line option, 131  
`q_6`  
 command line option, 131  
`q_7`  
 command line option, 131  
`q_8`  
 command line option, 131  
`q_9`  
 command line option, 131  
`Qa_Bldgs`  
 command line option, 113  
`Qa_BSoil`  
 command line option, 113  
`Qa_DecTr`  
 command line option, 113  
`Qa_EveTr`  
 command line option, 113  
`Qa_Grass`  
 command line option, 113  
`Qa_Paved`  
 command line option, 113  
`Qa_Water`  
 command line option, 113  
`QE`  
 command line option, 112  
`qe`  
 command line option, 103  
`QElumps`  
 command line option, 112  
`QF`

- command line option, 112
- qf
  - command line option, 103
- qf0\_beu
  - command line option, 92
- qf\_a
  - command line option, 92
- qf\_b
  - command line option, 93
- qf\_c
  - command line option, 93
- QH
  - command line option, 112
- qh
  - command line option, 103
- QHlumps
  - command line option, 112
- QHresis
  - command line option, 112
- QM
  - command line option, 112
- Qm\_Bldgs
  - command line option, 114
- Qm\_BSoil
  - command line option, 114
- Qm\_DecTr
  - command line option, 114
- Qm\_EveTr
  - command line option, 114
- Qm\_Grass
  - command line option, 114
- Qm\_Paved
  - command line option, 115
- Qm\_Water
  - command line option, 115
- QmFr\_Bldgs
  - command line option, 114
- QmFr\_BSoil
  - command line option, 114
- QmFr\_DecTr
  - command line option, 114
- QmFr\_EveTr
  - command line option, 114
- QmFr\_Grass
  - command line option, 114
- QmFr\_Paved
  - command line option, 114
- QmFr\_Water
  - command line option, 114
- QMFreeze
  - command line option, 112
- QMRain
  - command line option, 113
- QN

- command line option, 113
- qn
  - command line option, 103
- QNSnow
  - command line option, 113
- QNSnowFr
  - command line option, 113
- QS
  - command line option, 113
- qs
  - command line option, 103

## R

- RA
  - command line option, 115
- radmeltfact
  - command line option, 93
- Rain
  - command line option, 116
- rain
  - command line option, 103
- raincover
  - command line option, 93
- rainmaxres
  - command line option, 93
- RainSn\_Bldgs
  - command line option, 116
- RainSn\_BSoil
  - command line option, 116
- RainSn\_DecTr
  - command line option, 116
- RainSn\_EveTr
  - command line option, 116
- RainSn\_Grass
  - command line option, 116
- RainSn\_Paved
  - command line option, 116
- RainSn\_Water
  - command line option, 116
- read\_epw() (*in module supy.util*), 69
- resp\_a
  - command line option, 93
- resp\_b
  - command line option, 93
- RH
  - command line option, 102
- RH2
  - command line option, 115
- RO
  - command line option, 115
- ROImp
  - command line option, 115
- ROPipe
  - command line option, 115



ROSoil  
     command line option, 115  
 roughlenheatmethod  
     command line option, 94  
 roughlenmommethode  
     command line option, 94  
 ROVeg  
     command line option, 115  
 ROWater  
     command line option, 115  
 RS  
     command line option, 115, 116  
 run\_supy() (*in module supy*), 65  
 runofftowater  
     command line option, 94

## S

s1  
     command line option, 94  
 s2  
     command line option, 94  
 sathydraulicconduct  
     command line option, 94  
 save\_supy() (*in module supy*), 65  
 Sd\_Bldgs  
     command line option, 118  
 Sd\_BSoil  
     command line option, 118  
 Sd\_DecTr  
     command line option, 118  
 Sd\_EveTr  
     command line option, 118  
 Sd\_Grass  
     command line option, 118  
 Sd\_Paved  
     command line option, 118  
 Sd\_Water  
     command line option, 118  
 SDD\_DecTr  
     command line option, 116  
 SDD\_EveTr  
     command line option, 116  
 SDD\_Grass  
     command line option, 116  
 sddfull  
     command line option, 94  
 sfr  
     command line option, 95  
 show\_version() (*in module supy*), 67  
 sim\_ohm() (*in module supy.util*), 71  
 SMD  
     command line option, 117  
 SMDBldgs  
     command line option, 117  
 SMDBSoil  
     command line option, 117  
 SMDDecTr  
     command line option, 117  
 SMDEveTr  
     command line option, 117  
 SMDGrass  
     command line option, 117  
 smdmethod  
     command line option, 95  
 SMDPaved  
     command line option, 117  
 snow  
     command line option, 103  
 snowalb  
     command line option, 95  
 snowalbmax  
     command line option, 95  
 snowalbmin  
     command line option, 95  
 SnowCh  
     command line option, 118  
 snowdens  
     command line option, 95  
 snowdensmax  
     command line option, 95  
 snowdensmin  
     command line option, 96  
 snowfrac  
     command line option, 96  
 snowlimbldg  
     command line option, 96  
 snowlimpaved  
     command line option, 96  
 snowpack  
     command line option, 96  
 snowpacklimit  
     command line option, 96  
 snowprof\_24hr  
     command line option, 96  
 SnowRBldgs  
     command line option, 118  
 SnowRPaved  
     command line option, 119  
 snowuse  
     command line option, 97  
 snowwater  
     command line option, 97  
 soildepth  
     command line option, 97  
 soilstore\_id  
     command line option, 97  
 soilstorecap  
     command line option, 97

stabilitymethod  
    command line option, 97  
startdls  
    command line option, 97  
State  
    command line option, 119  
state\_id  
    command line option, 98  
statelimit  
    command line option, 98  
StBldgs  
    command line option, 119  
StBSoil  
    command line option, 119  
StDecTr  
    command line option, 119  
StEveTr  
    command line option, 119  
StGrass  
    command line option, 119  
storageheatmethod  
    command line option, 98  
storedrainprm  
    command line option, 98  
StPaved  
    command line option, 119  
StWater  
    command line option, 119  
suews-convert command line option  
    -f, -from <fromVer>, 76  
    -i, -input <fromDir>, 76  
    -o, -output <toDir>, 76  
    -t, -to <toVer>, 76  
suews-run command line option  
    PATH\_RUNCONTROL, 76  
surfacearea  
    command line option, 98  
SurfCh  
    command line option, 119  
SWE  
    command line option, 117  
SWE\_Bldgs  
    command line option, 117  
SWE\_BSoil  
    command line option, 117  
SWE\_DecTr  
    command line option, 117  
SWE\_EveTr  
    command line option, 117  
SWE\_Grass  
    command line option, 118  
SWE\_Paved  
    command line option, 118  
SWE\_Water

    command line option, 118

## T

T2  
    command line option, 119  
T\_1  
    command line option, 119  
T\_10  
    command line option, 120  
T\_11  
    command line option, 120  
T\_12  
    command line option, 120  
T\_13  
    command line option, 120  
T\_14  
    command line option, 120  
T\_15  
    command line option, 120  
T\_16  
    command line option, 120  
T\_17  
    command line option, 120  
T\_18  
    command line option, 120  
T\_19  
    command line option, 120  
T\_2  
    command line option, 120  
T\_20  
    command line option, 120  
T\_21  
    command line option, 121  
T\_22  
    command line option, 121  
T\_23  
    command line option, 121  
T\_24  
    command line option, 121  
T\_25  
    command line option, 121  
T\_26  
    command line option, 121  
T\_27  
    command line option, 121  
T\_28  
    command line option, 121  
T\_29  
    command line option, 121  
T\_3  
    command line option, 121  
T\_30  
    command line option, 121  
T\_4

command line option, [121](#)  
 T\_5  
 command line option, [122](#)  
 T\_6  
 command line option, [122](#)  
 T\_7  
 command line option, [122](#)  
 T\_8  
 command line option, [122](#)  
 T\_9  
 command line option, [122](#)  
 Ta  
 command line option, [122](#)  
 Tair  
 command line option, [102](#)  
 tau\_a  
 command line option, [98](#)  
 tau\_f  
 command line option, [99](#)  
 tau\_r  
 command line option, [99](#)  
 tempmeltfact  
 command line option, [99](#)  
 Tg  
 command line option, [122](#)  
 th  
 command line option, [99](#)  
 theta\_bioco2  
 command line option, [99](#)  
 timezone  
 command line option, [99](#)  
 tl  
 command line option, [99](#)  
 Tmax  
 command line option, [122](#)  
 Tmin  
 command line option, [122](#)  
 Tmrt  
 command line option, [122](#)  
 TotCh  
 command line option, [122](#)  
 trafficrate  
 command line option, [100](#)  
 trafficunits  
 command line option, [100](#)  
 traffprof\_24hr  
 command line option, [100](#)  
 Ts  
 command line option, [122](#)  
 Tsnow\_Bldgs  
 command line option, [123](#)  
 Tsnow\_BSoil  
 command line option, [123](#)  
 Tsnow\_DecTr

command line option, [123](#)  
 Tsnow\_EveTr  
 command line option, [123](#)  
 Tsnow\_Grass  
 command line option, [123](#)  
 Tsnow\_Paved  
 command line option, [123](#)  
 Tsnow\_Water  
 command line option, [123](#)  
 tstep  
 command line option, [100](#)  
 Tsurf  
 command line option, [123](#)

## U

U  
 command line option, [102](#)  
 U10  
 command line option, [123](#)  
 U\_1  
 command line option, [123](#)  
 U\_10  
 command line option, [123](#)  
 U\_11  
 command line option, [123](#)  
 U\_12  
 command line option, [124](#)  
 U\_13  
 command line option, [124](#)  
 U\_14  
 command line option, [124](#)  
 U\_15  
 command line option, [124](#)  
 U\_16  
 command line option, [124](#)  
 U\_17  
 command line option, [124](#)  
 U\_18  
 command line option, [124](#)  
 U\_19  
 command line option, [124](#)  
 U\_2  
 command line option, [124](#)  
 U\_20  
 command line option, [124](#)  
 U\_21  
 command line option, [124](#)  
 U\_22  
 command line option, [124](#)  
 U\_23  
 command line option, [125](#)  
 U\_24  
 command line option, [125](#)  
 U\_25

- command line option, [125](#)
- U\_26
  - command line option, [125](#)
- U\_27
  - command line option, [125](#)
- U\_28
  - command line option, [125](#)
- U\_29
  - command line option, [125](#)
- U\_3
  - command line option, [125](#)
- U\_30
  - command line option, [125](#)
- U\_4
  - command line option, [125](#)
- U\_5
  - command line option, [125](#)
- U\_6
  - command line option, [125](#)
- U\_7
  - command line option, [126](#)
- U\_8
  - command line option, [126](#)
- U\_9
  - command line option, [126](#)

## V

- veg\_type
  - command line option, [100](#)

## W

- waterdist
  - command line option, [100](#)
- waterusemethod
  - command line option, [100](#)
- wdir
  - command line option, [103](#)
- wetthresh
  - command line option, [101](#)
- WU\_DecTr1
  - command line option, [126](#)
- WU\_DecTr2
  - command line option, [126](#)
- WU\_DecTr3
  - command line option, [126](#)
- WU\_EveTr1
  - command line option, [126](#)
- WU\_EveTr2
  - command line option, [126](#)
- WU\_EveTr3
  - command line option, [127](#)
- WU\_Grass1
  - command line option, [127](#)
- WU\_Grass2
  - command line option, [127](#)

- command line option, [127](#)
- WU\_Grass3
  - command line option, [127](#)
- WUDecTr
  - command line option, [126](#)
- WUEveTr
  - command line option, [126](#)
- WUGrass
  - command line option, [126](#)
- Wuh
  - command line option, [102](#)
- WUInt
  - command line option, [126](#)
- wuprofa\_24hr
  - command line option, [101](#)
- wuprofm\_24hr
  - command line option, [101](#)

## X

- xsmd
  - command line option, [103](#)

## Z

- z
  - command line option, [101](#)
- z0m
  - command line option, [132](#)
- z0m\_in
  - command line option, [101](#)
- z\_1
  - command line option, [132](#)
- z\_10
  - command line option, [132](#)
- z\_11
  - command line option, [132](#)
- z\_12
  - command line option, [132](#)
- z\_13
  - command line option, [132](#)
- z\_14
  - command line option, [132](#)
- z\_15
  - command line option, [132](#)
- z\_16
  - command line option, [132](#)
- z\_17
  - command line option, [132](#)
- z\_18
  - command line option, [132](#)
- z\_19
  - command line option, [132](#)
- z\_2
  - command line option, [133](#)
- z\_20

- command line option, 133
- z\_21
  - command line option, 133
- z\_22
  - command line option, 133
- z\_23
  - command line option, 133
- z\_24
  - command line option, 133
- z\_25
  - command line option, 133
- z\_26
  - command line option, 133
- z\_27
  - command line option, 133
- z\_28
  - command line option, 133
- z\_29
  - command line option, 133
- z\_3
  - command line option, 133
- z\_30
  - command line option, 134
- z\_4
  - command line option, 134
- z\_5
  - command line option, 134
- z\_6
  - command line option, 134
- z\_7
  - command line option, 134
- z\_8
  - command line option, 134
- z\_9
  - command line option, 134
- zdm
  - command line option, 134
- zdm\_in
  - command line option, 101
- Zenith
  - command line option, 127