A Python-enhanced urban land surface model SuPy (SUEWS in Python, v2019.2): development, deployment and demonstration

Ting Sun and Sue Grimmond
Department of Meteorology, University of Reading, Reading, RG6 6BB, UK

Correspondence: Ting Sun (ting.sun@reading.ac.uk) and Sue Grimmond (c.s.grimmond@reading.ac.uk)

Received: 10 February 2019 – Discussion started: 21 March 2019
Revised: 12 June 2019 – Accepted: 18 June 2019 – Published: 9 July 2019

Abstract. Accurate and agile modelling of cities weather, climate, hydrology and air quality is essential for integrated urban services. The Surface Urban Energy and Water balance Scheme (SUEWS) is a state-of-the-art widely used urban land surface model (ULSM) which simulates urban–atmospheric interactions by quantifying the energy, water and mass fluxes. Using SUEWS as the computation kernel, SuPy (SUEWS in Python) uses a Python-based data stack to streamline the pre-processing, computation and post-processing that are involved in the common modelling-centred urban climate studies. This paper documents the development of SuPy, including the SUEWS interface modification, F2PY (Fortran to Python) configuration and Python front-end implementation. In addition, the deployment of SuPy via PyPI (Python Package Index) is introduced along with the automated workflow for cross-platform compilation. This makes SuPy available for all mainstream operating systems (Windows, Linux and macOS). Three online tutorials in Jupyter Notebook are provided to users of different levels to become familiar with SuPy urban climate modelling. The SuPy package represents a significant enhancement that supports existing and new model applications, reproducibility and enhanced functionality.

1 Introduction

Cities need to be resilient to weather, climate, hydrological and air quality hazards given their large and ever-increasing populations (Baklanov et al., 2018). One prerequisite to building resilience is information at various spatio-temporal scales, e.g. to understand energy partitioning over urban surfaces (D. Li et al., 2015; Sun et al., 2017; Wang et al., 2015; Ward and Grimmond, 2017; Zhao et al., 2014), pedestrian level meteorology to diagnose thermal comfort (Bar et al., 2011; Erell et al., 2013; Krayenhoff et al., 2018; Sun et al., 2016; Tan et al., 2009), or ambient radiation and wind conditions to assist building design (Chen, 2004; Jentsch et al., 2013; B. Li et al., 2015; Lindberg and Grimmond, 2011; Reinhart and Cerezo Davila, 2016; Santamouris et al., 2001). To obtain such information, accurate and agile modelling capacity of the urban weather and climate are essential.

Urban land surface models (ULSMs) are widely used to simulate urban–atmosphere interactions by quantifying the energy, water and mass fluxes between the surface and urban atmosphere (Best and Grimmond, 2015; Chen et al., 2011; Wang et al., 2012). These models require information on urban morphology (e.g. heights, spacings of buildings, etc.) and anthropogenic dynamics (e.g. building-operation-related heat release, emissions of heat by traffic) to be included.

One widely used and tested ULSM, the Surface Urban Energy and Water balance Scheme (SUEWS; Table 1), requires basic meteorological data and surface information to characterize essential urban features (i.e. urban surface heterogeneity and anthropogenic dynamics). SUEWS enables long-term urban climate simulations without specialized computing facilities (Järvi et al., 2011, 2014; Ward et al., 2016). SUEWS is regularly enhanced (Grimmond et al., 1991, 1986; Grimmond and Oke, 1991; Järvi et al., 2011, 2014, 2019; Offerle et al., 2003; Ward et al., 2016) and tested in cities under a range of climates worldwide (Table 1). Although operationally simple and scientifically robust, SUEWS still requires some skill for application (e.g. computing environment setup, parameter configuration), which may limit uptake for broader applications in urban planning and design.
Table 1. Recent studies using SUEWS. D&E – development and evaluation. n/a – not applicable

<table>
<thead>
<tr>
<th>Topic</th>
<th>City</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Impact of haze on the urban water balance</td>
<td>Beijing, China</td>
<td>Kokkonen et al. (2019)</td>
</tr>
<tr>
<td>CO₂ emissions module: D&amp;E</td>
<td>Helsinki, Finland</td>
<td>Järvi et al. (2019)</td>
</tr>
<tr>
<td>Impacts of anthropogenic heat and irrigation on surface energy balance</td>
<td>Shanghai, China</td>
<td>Ao et al. (2018)</td>
</tr>
<tr>
<td>Sensitivity of SUEWS to forcing variables</td>
<td>Vancouver, Canada</td>
<td>Kokkonen et al. (2018)</td>
</tr>
<tr>
<td>SUEWS a core processor of the UMEP system</td>
<td>London, UK (n/a)</td>
<td>Lindberg et al. (2018)</td>
</tr>
<tr>
<td>Impacts of changes in surface cover, human behaviour and climate on energy partitioning</td>
<td>London, UK</td>
<td>Ward and Grimmond (2017)</td>
</tr>
<tr>
<td>Offline evaluation of SUEWS driven by WRF output</td>
<td>Porto, Portugal</td>
<td>Rafael et al. (2017)</td>
</tr>
<tr>
<td>Implications of warming to cold climate cities</td>
<td>High-latitude cities</td>
<td>Järvi et al. (2017)</td>
</tr>
<tr>
<td>Evaluation in Singapore and comparison with other urban land surface models</td>
<td>Singapore</td>
<td>Demuzere et al. (2017)</td>
</tr>
<tr>
<td>Evaluation in four cities with different climates</td>
<td>Dublin, Ireland</td>
<td>Alexander et al. (2016)</td>
</tr>
<tr>
<td></td>
<td>Hamburg, Germany</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Melbourne, Australia</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Phoenix, USA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>London, UK</td>
<td>Ward et al. (2016)</td>
</tr>
<tr>
<td></td>
<td>Swindon, UK</td>
<td></td>
</tr>
<tr>
<td>Evaluation of SUEWS in two UK cities</td>
<td>Shanghai, China</td>
<td>Ao et al. (2016)</td>
</tr>
<tr>
<td>Evaluation of radiation flux in Shanghai</td>
<td>Sacramento, USA</td>
<td>Onomura et al. (2015)</td>
</tr>
<tr>
<td>Boundary layer modelling and coupling with SUEWS</td>
<td>Dublin, Ireland</td>
<td>Alexander et al. (2015)</td>
</tr>
<tr>
<td>Evaluation with Local Climate Zone information as surface characteristics</td>
<td>Helsinki, Finland</td>
<td>Karsisto et al. (2015)</td>
</tr>
<tr>
<td>Model inter-comparison for sensible and latent heat fluxes</td>
<td>Helsinki, Finland</td>
<td>Järvi et al. (2014)</td>
</tr>
<tr>
<td>Snow melt model D&amp;E</td>
<td>Montreal, Canada</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Vancouver, Canada</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Los Angeles, USA</td>
<td>Järvi et al. (2011)</td>
</tr>
</tbody>
</table>

Reproducibility and open science principles are increasingly important (Peng, 2011). Although climate scientists by convention publish detailed model configurations used in their research, minor inconsistencies or lack of transparency of code often hampers efforts to reproduce simulation results. In addition, new users may lack prerequisite knowledge in low-level compilation and scripting to undertake initial model runs and interpretation of simulation results (Lin, 2012).

Today Python is used extensively by the atmospheric sciences community for data analyses and numerical modelling (Lin, 2012; Perkel, 2015) thanks to its simplicity and the large scientific Python ecosystem (e.g. PyData community: https://pydata.org, last access: 12 June 2019). Recent Python-based endeavours include global climate system models (Monteiro et al., 2018), stochastic geological models (de la Varga et al., 2019) and hydrological models (Hamman et al., 2018) to cite just a few.

In this paper, we present a Python-enhanced urban climate system based on the popular Fortran-coded SUEWS – SuPy (SUEWS in Python). The development of SuPy (Sect. 2), the essential workflow in its cross-platform deployment (Sect. 3), and three demonstration tutorials for users of different levels (Sect. 4) are presented.

2 Development

The following are considered within the design process of SuPy:

1. Data preparation. Climate simulations typically require extensive pre-processing of data (loading input data, reformatting to conform with standards, etc.) and post-processing (conversion of output, graphical and cartographic plotting, etc.). Python has a vast array of utilities to support this; notably, NumPy (the fundamental package for scientific computing with Python, https://www.numpy.org, last access: 12 June 2019) and pandas (a tabular-format-centred data analysis tool) are two cornerstone libraries in Python-based scientific computing.

2. Performance. Python, as a scripting language, has poorer performance than compiled languages (e.g. C, Fortran; Kouatchou, 2018). For this reason, Fortran is used extensively for weather and climate-related software (e.g. WRF – Skamarock and Klemp, 2008; GFDL AM3 – Donner et al., 2011; etc.). Therefore, by using different languages their individual strengths can be utilized.
3. **Cross-platform ability.** Given the range of computer environments, it is important that software can be easily used across operating systems with ease. Python and Fortran are both easily used on most operating systems. However, for performance reasons as noted, it is preferable to have a compiled back end for intensive simulations, where platform-specific compilations are mandatory. Thus, we adopt the Microsoft Azure Pipelines to allow for a cross-platform ability for SuPy (Sect. 3.1).

4. **Extendibility.** It is desirable, possibly even essential, for the scientific model to interact with other models and data sources to extend the overall capacity and to explore questions related to urban climate beyond the climate science.

To address these four considerations, SuPy’s architecture uses Python’s data processing and Fortran’s computational efficiency. SuPy consists of three parts (Fig. 1):

1. **SuPy.** A Python-based front-end processor based on the pandas DataFrame with functionality for data analysis and simulation management (Appendix A).

2. **SuPy_driver.** Calculation kernel compiled by F2PY (Fortran to Python, part of the NumPy package) (Peterson, 2009) to facilitate the transfer of SUEWS’ Fortran modelling ability to Python and guarantee the computational performance.

3. **SUEWS.** A Fortran-coded local-scale urban land surface model of moderate complexity that can simulate the urban surface energy balance in combination with the complete urban hydrological cycle, considering irrigation and runoff processes (Grimmond and Oke, 1991; Järvi et al., 2011, 2014; Offerle et al., 2003; Ward et al., 2016).

Development of SuPy (Sun, 2019) started with SUEWS v2017b (Ward and Grimmond, 2017). SUEWS has three distinct groups of subroutines: model physics, input and output (I/O). To help generalize coupling, the use of Fortran as the central data structure to simplify the pre- and post-processing as required by the original SUEWS. The overall data structures used in SuPy are described in the documentation (https://supy.readthedocs.io/en/latest/data-structure/supy-io.html, last access: 12 June 2019). The pre-processor is designed to load existing SUEWS input files, which consists of the following (Appendix A):

1. **init_supy.** This loads surface characteristics (e.g. albedo, emissivity, land cover fractions; full details given in the SUEWS documentation: https://suews-docs.readthedocs.io/en/latest/input_files/SUEWS_SiteInfo/SUEWS_SiteInfo.html, last access: 12 June 2019) and model configurations (e.g. stability correction option chosen; https://suews-docs.readthedocs.io/en/latest/input_files/RunControl/RunControl.html, last access: 12 June 2019). The data go into df_state_init (a pandas DataFrame; https://supy.readthedocs.io/en/latest/data-structure/df_state.html, last access: 12 June 2019). Two auxiliary json files are used with the look-up hierarchies for loading this information from SUEWS library files (https://suews-docs.readthedocs.io/en/latest/input_files/input_files.html, last access: 12 June 2019) in a consistent file-code-variable way.

2. **load_forcing.** Meteorological and other external forcing information are loaded into df_forcing (a pandas DataFrame; https://supy.readthedocs.io/en/latest/data-structure/df_forcing.html, last access: 12 June 2019) to drive the SuPy simulations with time-step size inferred from its DatetimeIndex (i.e. the freq attribute). SUEWS should be run at short time steps (e.g. 5 min) as precipitation or irrigation runoff from impervious surfaces becomes too large if the water arrives as one large hourly (or longer) amount (Grimmond and Oke, 1991; Ward et al., 2018). As such, load_forcing is implemented with the ability to downscale the raw forcing data to finer time steps (5 min by default). The temporal resolution of raw forcing data can be between 5 and 360 min, with 30–60 min being the most common.

Detailed guidance is provided in SUEWS documentation for preparing input files (https://suews-docs.readthedocs.io/en/latest/input_files/input_files.html, last access: 12 June 2019).

The SUEWS kernel (v2018c) is compiled by F2PY to generate the Python-compliant SuPy_driver package. Using the two subroutines allows better computational performance. The SuPy_driver calls the two subroutines depending on time-step simulation type: single- (sd_cal_tstep) or multi-time-step mode (sd_cal_multitstep, Fig. 1b). The former is useful in flexible manipulation of SuPy runtime behaviours (application in Sect. 4.3), while the latter has much better performance because of the much lower computational overheads with the F2PY wrapper. Therefore, sd_cal_multitstep is the default executer in run_supy, the SuPy core processor performing simulations, for regular runs without runtime manipulation.

In addition to run_supy, SuPy uses pandas DataFrame as the central data structure to simplify the pre- and post-processing as required by the original SUEWS. The overall data structures used in SuPy are described in the documentation (https://supy.readthedocs.io/en/latest/data-structure/supy-io.html, last access: 12 June 2019). The pre-processor is designed to load existing SUEWS input files, which consists of the following (Appendix A):

1. **init_supy.** This loads surface characteristics (e.g. albedo, emissivity, land cover fractions; full details given in the SUEWS documentation: https://suews-docs.readthedocs.io/en/latest/input_files/SUEWS_SiteInfo/SUEWS_SiteInfo.html, last access: 12 June 2019) and model configurations (e.g. stability correction option chosen; https://suews-docs.readthedocs.io/en/latest/input_files/RunControl/RunControl.html, last access: 12 June 2019). The data go into df_state_init (a pandas DataFrame; https://supy.readthedocs.io/en/latest/data-structure/df_state.html, last access: 12 June 2019). Two auxiliary json files are used with the look-up hierarchies for loading this information from SUEWS library files (https://suews-docs.readthedocs.io/en/latest/input_files/input_files.html, last access: 12 June 2019) in a consistent file-code-variable way.

2. **load_forcing.** Meteorological and other external forcing information are loaded into df_forcing (a pandas DataFrame; https://supy.readthedocs.io/en/latest/data-structure/df_forcing.html, last access: 12 June 2019) to drive the SuPy simulations with time-step size inferred from its DatetimeIndex (i.e. the freq attribute). SUEWS should be run at short time steps (e.g. 5 min) as precipitation or irrigation runoff from impervious surfaces becomes too large if the water arrives as one large hourly (or longer) amount (Grimmond and Oke, 1991; Ward et al., 2018). As such, load_forcing is implemented with the ability to downscale the raw forcing data to finer time steps (5 min by default). The temporal resolution of raw forcing data can be between 5 and 360 min, with 30–60 min being the most common.
Figure 1. SuPy’s three major components (left to right): (a) SuPy, a Python-based front-end processor; (b) SuPy_driver, the calculation kernel compiled by F2PY; (c) SUEWS, a Fortran-coded local-scale urban land surface model. Note that not all physics schemes are listed.

3 Deployment

To achieve cross-platform compatibility, SuPy has two parts:

1. SuPy_driver (calculation kernel): the F2PY generated binaries of SUEWS are platform-dependent because of compilation being necessary for assurance of performance.

2. SuPy (front-end processor): this platform-independent Python code allows rapid iteration in functionality enhancement and bug fixing thanks to the powerful ecosystem of Python utilities.

As software compilation can be frustrating and/or prone to operator errors, this procedure is automated using two online services: Microsoft Azure Pipeline (https://azure.microsoft.com/en-us/services/devops/pipelines/, last access: 12 June 2019) for continuous integration (CI) and PyPI (Python Package Index; https://pypi.org, last access: 12 June 2019) for distribution. Microsoft Azure Pipeline has good cross-platform support (https://docs.microsoft.com/en-gb/azure/devops/pipelines/agents/agents?view=azure-devops, last access: 1 July 2019) and easy connection with code repositories (e.g. GitHub: https://github.com, last access: 12 June 2019; Bitbucket: https://bitbucket.com, last access: 12 June 2019) and supports automated compilation for different platforms. The Azure Pipeline build workflow permits a variety of functionalities to facilitate compilation and publishing to other online services (e.g. PyPI, GitHub pages, etc.). Currently, this is set up for three major platforms (Windows, macOS and Linux) with three Python 3 configurations (3.5, 3.6 and 3.7) to conduct automated compilation of SuPy back-end files: SUEWS binaries and SuPy_driver, the product of which is directly pushed to PyPI and released in real time.
To build the SuPy_driver two crucial steps to allow cross-platform deployment (full details refer to configuration file setup.py in SuPy_driver) are the following.

1. **Static linking**: to eliminate the issue of missing dynamic libraries, the calculation kernels are pre-built using static linking and therefore run directly after downloading.

2. **manylinux tagging**: given the many Linux distributions and their different runtime libraries that often require distribution-specific compilation, we use the manylinux docker image (for details refer to https://github.com/pypa/manylinux) to compile SuPy_driver.

In addition to the cross-platform compilation, to guarantee delivery quality we perform automatic code tests of four preset configurations for every build:

1. **Connectivity between SuPy and SuPy_driver**: checks if the front-end processor and back-end calculation core can communicate with correct input and output.

2. **Success in single-time-step mode**: checks SuPy can produce correct simulation results in the single-time-step mode.

3. **Success in multi-time-step mode**: checks SuPy can produce correct simulation results in the multi-time-step mode and does a quick benchmark of computation speed.

4. **Compare simulation results between single- and multi-time-steps modes**: checks SuPy can produce identical simulation results as designed.

All build and test output is logged in detail (see all logs here: https://dev.azure.com/sunt05/SuPy/_build, last access: 12 June 2019) and the results are reported to developers in real time. This feature is used for all code and underpins a commitment for timely support to SuPy development.

The Python Package Index (PyPI: https://pypi.org, last access: 12 June 2019) is the official third-party software repository for Python. As it is supported by the pip toolchain it provides Python users easy worldwide access to packages and frees Python developers from maintaining indexing and distribution servers. By using the PyPI channel, SuPy can be easily installed by users with a one-line input in a command line tool on any desktop/server system (Listing 1).

```bash
python3 -m pip install supy -U
```

**Listing 1.** Command line code for installing SuPy using pip. Note 64-bit Python 3.5+ is required for SuPy installation.

4 Demonstration: SuPy tutorials

To familiarize users with SuPy urban climate modelling and to demonstrate the functionality of SuPy, we provide three tutorials (Table 2, access at https://supy.readthedocs.io/en/latest/tutorial/tutorial.html, last access: 12 June 2019) in Jupyter notebooks (https://jupyter.org/, last access: 12 June 2019).

They can run in browsers (e.g. desktop, tablet) either by local configuration or on remote servers with pre-set environments (e.g. Google Colaboratory, https://colab.research.google.com, last access: 12 June 2019; Microsoft Azure Notebooks, https://notebooks.azure.com, last access: 12 June 2019). As Jupyter notebooks allow source code to be incorporated with detailed notes, users can organize their analyses (Shen, 2014). Jupyter notebooks can be installed with pip on any desktop/server system and open .ipynb notebook files locally (Listing 2). We note that running SuPy in browsers is not implemented by SuPy per se but allowed by the Jupyter environment where Python 3 is supported. The reason for running SuPy (and many other python applications) on mobile devices (e.g. mobile phone, tablet) is simple: working seamlessly across different devices is a natural need.

```bash
python3 -m pip install jupyter -U
jupyter-notebook path_to_your_notebook
```

**Listing 2.** Command line code for installing Jupyter Notebook with pip and open an existing local .ipynb notebook file (i.e. path_to_your_notebook).

These are made available to SUEWS by calling the load_SampleData function. This produces pandas DataFrames with the initial model state (df_state_init) and the forcing variables (df_forcing). These are used in all three tutorials.

4.1 SuPy quick-start

In this tutorial, we demonstrate the key steps in using SuPy to undertake the core task to simulate energy and water balance in an urban context using SUEWS. Here the runs are for a central London area in 2012.

The urban surface energy balance (SEB) can be expressed as

\[
Q^* + Q_f = Q_H + Q_E + \Delta Q_S. \tag{1}
\]

where the flux densities (W m\(^{-2}\)) are \(Q^*\) net all-wave radiation, \(Q_f\) anthropogenic heat, \(Q_H\) turbulent sensible heat, \(Q_E\) latent heat and \(\Delta Q_S\) the net storage heat flux. Through \(Q_E\), the SEB characteristics can be linked to the water balance:

\[
P + I = E + R + \Delta S, \tag{2}
\]

where each term is a depth of water per unit of time (e.g. mm d\(^{-1}\)). \(P\) is precipitation, \(I\) irrigation, \(E\) evapotrans-

...
Table 2. Three SuPy tutorials. Note that the website links redirect to online Jupyter notebooks for SuPy simulation without any configuration by users.

<table>
<thead>
<tr>
<th>Name</th>
<th>Aim</th>
<th>Audience</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quick-start SuPy (<a href="https://mybinder.org/v2/gh/sunt05/SuPy/master?filepath=docs/source/tutorial/quick-start.ipynb">https://mybinder.org/v2/gh/sunt05/SuPy/master?filepath=docs/source/tutorial/quick-start.ipynb</a>, last access: 1 July 2019)</td>
<td>Essential workflow to conduct SuPy simulations</td>
<td>New users, students</td>
</tr>
<tr>
<td>Impact studies using SuPy in parallel mode (<a href="https://mybinder.org/v2/gh/sunt05/SuPy/master?filepath=docs/source/tutorial/impact-studies-parallel.ipynb">https://mybinder.org/v2/gh/sunt05/SuPy/master?filepath=docs/source/tutorial/impact-studies-parallel.ipynb</a>, last access: 1 July 2019)</td>
<td>Impacts on urban climate from varying surface characteristics and forcing conditions</td>
<td>Urban climate researchers with experience in land surface simulations</td>
</tr>
<tr>
<td>Interaction between SuPy and external models (<a href="https://mybinder.org/v2/gh/sunt05/SuPy/master?filepath=docs/source/tutorial/external-interaction.ipynb">https://mybinder.org/v2/gh/sunt05/SuPy/master?filepath=docs/source/tutorial/external-interaction.ipynb</a>, last access: 1 July 2019)</td>
<td>Couple SuPy and external models for agile development</td>
<td>Model developers with background in climate modelling</td>
</tr>
</tbody>
</table>

Table 3. Default settings in the sample dataset provided with SuPy for (a) physics scheme and (b) basic site characteristics. For full SuPy variable setting details refer to the online documentation: https://supy.readthedocs.io/en/latest/data-structure/df_state.html (last access: 12 June 2019).

<table>
<thead>
<tr>
<th>(a) Basic site characteristics</th>
<th>SuPy parameter (unit)</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Land cover fractions: building, paved, evergreen tree, deciduous tree, grass, bare soil and water</td>
<td>sfr (--)</td>
<td>[0.43, 0.38, 0.001, 0.019, 0.029, 0.001, 0.14]</td>
</tr>
<tr>
<td>Building height</td>
<td>bldgh (m)</td>
<td>22.0</td>
</tr>
<tr>
<td>Evergreen tree height</td>
<td>evetreeh (m)</td>
<td>13.1</td>
</tr>
<tr>
<td>Deciduous tree height</td>
<td>dectreeh (m)</td>
<td>13.1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(b) Physics scheme</th>
<th>Code</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Radiation</td>
<td>radiationmethod</td>
<td>3</td>
</tr>
<tr>
<td>Heat storage</td>
<td>storageheatmethod</td>
<td>1</td>
</tr>
<tr>
<td>Anthropogenic heat</td>
<td>emissionsmethod</td>
<td>2</td>
</tr>
<tr>
<td>Snow</td>
<td>snowuse</td>
<td>1</td>
</tr>
<tr>
<td>Roughness length for momentum</td>
<td>roughlenmommethod</td>
<td>2</td>
</tr>
<tr>
<td>Roughness length for heat</td>
<td>roughlenheatmethod</td>
<td>2</td>
</tr>
<tr>
<td>Atmospheric stability</td>
<td>stabilitymethod</td>
<td>3</td>
</tr>
</tbody>
</table>
spiration ($= Q_E/L_v$ where $L_v$ is the latent heat of vaporization), $R$ runoff and $\Delta S$ the net change in water storage.

The fundamental steps to use SuPy after the software environment has been installed (see Listings 1, 2) are (1) load input, (2) run a simulation and (3) examine the results. With everything ready, three lines of python code are needed.

SuPy is run by calling `run_supy` after `df_state_init` and `df_forcing` have been loaded. After the simulation the two DataFrames provide major SUEWS outputs (`df_output`) and the model state (`df_state_final`) at the end of the run. The latter can be used as initial conditions for other SuPy runs.

```
# import supy package
import supy as sp
# import sample data
df_state_init, df_forcing = sp.load_SampleData()
# run supy simulation
df_output, df_state_final = sp.run_supy(df_forcing, df_state_init)
```

Listing 3. Python code for a minimal SuPy simulation with comments (green).

The post-processing uses pandas functions to resample, plot and write out the model output. The default output

Figure 2. Intra-annual (2012) variation in forcing variables in the sample dataset (from top to bottom): incoming solar radiation, air temperature, relative humidity, air pressure, wind speed and rainfall in central London. All variables are hourly averages except for total hourly rainfall (source of data: Kotthaus and Grimmond, 2014; gap filled: Ward et al., 2016).
Dataframe of 5 min resolution can be upcaled to the month for an overview of intra-annual dynamics of surface energy and water balances (Fig. 3).

This workflow using SuPy for urban climate modelling can be easily adapted to existing SUEWS tutorials under the UMEP framework (https://tutorial-docs.readthedocs.io, last access: 12 June 2019) by replacing the conventional SUEWS binary executable with the python SuPy package. Given the central role of Python in the UMEP framework, it is expected the adoption of SuPy will further streamline the workflows for urban climate simulations in UMEP.

4.2 Impacts of the urban area on local climate

A major application of urban climate models is to study the impacts on urban climate from design scenarios that change surface characteristics or the climate (atmospheric forcing). In this tutorial both scenario types are explored: we provide one example of modification of albedo for surface characteristics, while another of air temperature alteration for climate conditions.

Technically, this requires several configuration files to be prepared for a suite of independent model runs. These could be run consecutively (i.e. no interactions between runs are needed) or in parallel, so-called “embarrassingly parallel computation” (Bailey et al., 1991), with multiple independent runs with sufficient CPUs. In this tutorial, we first demonstrate how SuPy can be easily set up to efficiently complete multiple simulations in parallel.

We use *dask* (https://dask.org, last access: 12 June 2019) to parallelize the SuPy simulations given its close coherence with NumPy and pandas, in particular its almost identical DataFrame interfaces as pandas. Specifically, we use the apply method of *dask.DataFrame* to improve the simulation performance by distributing the SuPy computations across different configurations. Compared with the serial mode, the *dask*-based parallel mode takes only ~30% of the execution time of the serial mode for simulations longer than 1000 d for 12 grids (Fig. 4). The parallel configuration for running SuPy, *run_supyngrids*, is then used in the following two cases for more efficient parallel simulations.

To explore the effect of changes to surface properties, the DataFrame *df_state_init* needs to be modified. The surface albedo of different materials impacts the outgoing shortwave (solar) radiation and thus the surface energy balance fluxes and other atmospheric variables. Modifying roof albedo has been suggested extensively as a method to cool urban areas (e.g. Santamouris et al., 2011; Li et al., 2014; Ramamurthy et al., 2015). In the example, we conduct simulations from January 2012 to July 2012 with the first 6 months as the spin-up period. The building roof albedo is incrementally increased from 0.1 to 0.8 (e.g. a change from a very dark to a very light surface). The near-surface temperature $T_2$, an indicator of thermal state at pedestrian level, is analysed using the monthly maximum, mean and minimum (Fig. 5). It would be expected that the maximum and mean values of $T_2$ are greatly reduced as they are directly influenced by the altered net solar radiation, while impacts on the minimum $T_2$ might be expected to be minimal.

In this tutorial we demonstrate some starting cases rather than a complete research cycle. Notably, limitations are imposed in the configuration used (e.g. length of the model run, spin-up period, feedbacks permitted) and thus the relations shown should be interpreted with caution.

To explore changes in atmospheric forcing, the DataFrame *df_forcing* is modified. In this example, we investigate the impact of increased local-scale (constant flux layer) air temperature $T_a$ on the near-surface air temperature $T_2$. Air temperature $T_a$ is increased over 24 runs from 0°C (no change) to +2°C. The upper limit (+2°C) represents a highly possible average global warming scenario for the near future (IPCC, 2014). The SuPy simulations are conducted from January to July 2012 and July data are analysed. 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 (Fig. 6).

This tutorial demonstrates the simplicity of using SuPy to conduct impact studies of both surface characteristics and background climates. These can be easily adapted by users to their specific application interests. Thus, as various effects are combined the net impacts becomes more realistic.

4.3 Interaction between SuPy and external models

SUEWS can be coupled to other models that provide or require forcing data using the SuPy single-time-step running mode (Sect. 2). 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; Lamarino et al., 2011; Sailor, 2011) and is highly dependent on outdoor ambient air temperature. 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 (Fig. 7). 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 $T_C$ ($T_H$) set as 20°C (10°C), $C_B$ ($H_B$) set as 1.5 W m$^{-2}$ K$^{-1}$ (3 W m$^{-2}$ K$^{-1}$) and $Q_{F0}$ is set as 0 W m$^{-2}$.
Figure 3. SuPy simulated monthly (a) surface energy and (b) water balance for London 2012. Assuming no irrigation.

Figure 4. Comparison of execution times (seconds) between serial and parallel modes when 12 grids are simulated for different periods (days): 30, 90, 120, 150, 180, 270, 365, 730 and 1095. Simulations performed with macOS 10.14.3 running on 2.9 GHz Intel Core i9 with 32 GB memory. The model configuration is the same as Quick-start SuPy (Table 2).

implying other building activities (e.g. lightning, water heating, computers) are zero and therefore do not change the temperature or change with temperature.

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 the SuPy front end and the calculation kernel. By setting SuPy to receive external $Q_F$ as forcing, at each time step, 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.

Here we replace the SUEWS $Q_F$ (Table 2) with the simpler $Q_F$ model (Fig. 7, Eq. 3) to explore the question of the impact of $Q_F$ on $T_2$ and its feedback on $Q_F$. The simulation using SuPy coupled is performed for London in 2012. The data analysed are a summer (July) and a winter (December) month. Initially, $Q_F$ is 0 W m$^{-2}$ and 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 $T_{2[2]}$. Results indicate a positive feedback, as $Q_F$ increases $T_2$ is elevated but with different magnitudes (Fig. 8). 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 (Fig. 8). Note that London is relatively cool (cf. air temperature in Fig. 2) so the enhancement is much less than it would be in warmer cities.
In this case the anthropogenic heat flux model is simple, but a more complex model could be coupled to SUEWS in the same way. This can facilitate development of climate service tools that are both agile and responsive.

5 Concluding remarks

The development and delivery of a Python-enhanced urban climate model SuPy is introduced with tutorials (Table 2) to demonstrate typical applications and some new SUEWS features (e.g. surface diagnostics calculation). The Python code and tutorials are freely and openly available online (Appendix B). Users are encouraged to explore more intriguing urban-climate-related questions with SuPy. Notable features of SuPy include the following:

1. **Version consistency via PyPI**: SuPy is distributed via the well managed Python package repository PyPI with all history versions stored. This allows for clear version consistency for reproducing simulation results.

2. **Simplicity in input/output sharing**: SuPy uses pandas DataFrame as its core data structure and thus draws on a powerful data analysis toolchain, which can facilitate the ease with which urban climate research outcomes can be communicated.

3. **Ease of scientific development**: given the importance of meteorological forcing data in running climate simulations, SuPy will shortly be equipped with the ability to retrieve forcing variables from global reanalysis datasets. We anticipate data analyses and model development will be added more conveniently within the Python data ecosystem.

4. **An open source tool**: we welcome all kinds of contributions, e.g. incorporation of a new feature (pull requests), submission of issues or development of new tutorials.

In addition to SuPy in data analysis and communication features, the computation kernel is SUEWS so all physics schemes development will remain in the Fortran stack for computational performance and compatibility with a large
cohort of scientific code. In one application software, UMEP (Lindberg et al., 2018) written in Python, the SUEWS binary executable will shortly be updated to SuPy for better connectivity to other UMEP components.

We expect SuPy will help guide future development of SUEWS (and similar urban climate models) and enable new applications of the model. For example, the parallel set up of SuPy will allow large-scale simulations of urban climate across larger domains with greater surface heterogeneity. Moreover, the improvement in the SUEWS model structure and deployment process introduced by the development of SuPy paves the way to a more robust workflow of SUEWS for its sustainable success.

Code availability. Appendices B and C describe the locations and licence information for SuPy (https://doi.org/10.5281/zenodo.2574405; Sun, 2019) and SUEWS (https://doi.org/10.5281/zenodo.3267306; Sun et al., 2019), respectively.
Appendix A: SuPy functions

The utility of the six SuPy functions are

- **init_supy.** Initialize SuPy by loading initial model states.

- **load_forcing_grid.** Load forcing data for a specific grid included in the index of df_state_init.

- **run_supy.** Perform SuPy simulation.

- **save_supy.** Save SuPy run results to files.

- **load_SampleData.** Load sample data for quickly starting a demo run.

- **show_version.** Print supy and supy_driver version information.

For detailed usage of the included functions see: https://supy.readthedocs.io/en/latest/api.html#top-level-functions (last access: 1 July 2019).

Appendix B: SuPy model source code and documentation

Code repository

- Name: GitHub

- Identifier: https://github.com/sunt05/SuPy (last access: 12 June 2019)

- Licence: GNU GPL v3.0

- Date published: 10 February 2019

Versioned documentation

- Name: ReadTheDocs

- Identifier: https://supy.readthedocs.io (last access: 1 July 2019)

- Licence: GNU GPL v3.0

- Date published: 21 February 2019

Appendix C: SUEWS model source code and documentation

Code repository

- Name: GitHub

- Identifier: https://github.com/Urban-Meteorology-Reading/SUEWS

  (last access: 1 July 2019)

- Licence: GNU GPL v3.0

- Date published: 21 February 2019

Versioned documentation

- Name: ReadTheDocs

- Identifier: https://suews-docs.readthedocs.io

  (last access: 1 July 2019)

- Licence: GNU GPL v3.0

- Date published: 21 February 2019
Author contributions. TS led the development of SuPy and core enhancements of SUEWS since v2017b. SG provided overall oversight of the SUEWS development. TS and SG wrote the paper.

Competing interests. The authors declare that they have no conflict of interest.

Acknowledgements. We thank the two anonymous reviewers and the editor Tim Butler for their constructive comments. We appreciate the numerous people who have and continue to contribute to the development of SUEWS and the users who identify issues. We thank all those who have contributed to the field observations maintenance, analysis, provided sites, access and funding.

Financial support. This research has been supported by NERC Independent Research Fellowship (grant no. NE/P018637/1; Ting Sun); Newton Fund/Met Office CSSP China (Sue Grimmond, Ting Sun); RS Newton Mobility funding (Sue Grimmond, Ting Sun) and EPSRC LoHiCool (grant no. EP/N009797/1) (Sue Grimmond, Ting Sun). Field observations have been funded by grants from NERC (grant no. NE/H003231/1), European Commission, FP7 (grant no. 211345), the European Commission, H2020-EO-2014 (Urban-Fluxes (grant no. 637519)), EPSRC (grant nos. EP/I00159X/1, EP/I00159X/2), KCL (Sue Grimmond), Belmont Forum (grant nos. TRUC NE/L008971/1, G8MUREFU3FP-2201-075), Newton Fund and the Met Office (CSSP grant).

Review statement. This paper was edited by Tim Butler and reviewed by two anonymous referees.

References


Erell, E., Pearlmutter, D., Boneh, D. and Kutiel, P. B.: The Nas Parallel Benchmarks, In-


