UMEP Workshop

Release 2020-12-23

Sue Grimmond, Fredrik Lindberg, Ting Sun and Hamidreza Omidv

Dec 23, 2020

WORKSHOP STRUCTURE

Welcome to the urbisphere UMEP July 2020 Online Workshop

CHAPTER

ONE

PREPARING FOR THE WORKSHOP

Prior to the workshop, complete the steps in this section.

1.1 Introduction

1.1.1 Topics to be Covered

The three main software tools that will be used in the workshop are:

- **UMEP** - this is an urban climate services tool that has a large number of tools and models. It is an application add-on to QGIS (*Lindberg et al. 2018*)
- **SUEWS** - this is an energy, water and carbon balance model that can be used to simulate urban and non-urban areas (*Järvi et al. 2011*). This is both a standalone model and is also within UMEP. The UMEP environment is a good place to start to use SUEWS.
- **SuPy** - this is a Python interface to SUEWS which allows SUEWS to be rapidly used to address research questions (*Sun and Grimmond 2019*). It can also be used to couple SUEWS to other programs (e.g. DASH. STEBBS, EnergyPlus).

Two user environments that we will use are:

- **QGIS** - a geographic information system for analysing spatial data.
- **Jupyter Notebooks** - this allows for a project to be developed with comments, code, graphics etc, that contains all of the analyses of a project.

There are two main programming languages involved in the software

- **Python** - the QGIS add-ins are written in Python as is SuPy. This has the advantage of having a large number of code libraries for graphics, statistics, *etc*.
- **Fortran** - SUEWS is written in this. This has the advantage of being more computationally efficient for processing larger volumes of data. It is also the language that many weather and climate models are written in.

SUEWS coupling

- SUEWS is in UMEP, SuPy, and coupled to WRF, as well as being a standalone code.
- SUEWS can be forced (i.e. the meteorological data that is needed to run the model) in the following way:
  - Observations
  - ERA5 data (or other reanalysis data)
  - WRF (or other weather/climate model that it is coupled to)
  - Climate projections

SUEWS requires the urban form and function to be characterised

- form (e.g. land cover, building heights)
- function (e.g. population density, energy use diurnal pattern for work days and non-work days, irrigation behaviour, snow clearing behaviour)

These characteristics need to be provided for each grid (or spatial unit) to be modelled. A spatial unit does not need to be a rectangular grid (if not within a weather/climate model), but can be, for example, census data or other governance related spatial units.

Some *background* reading for the Workshop will help you understand the models and provide details options that will be useful later.

1.1.2 People

Hosted By

- Sue Grimmond
- Fredrick Lindberg (lead UMEP developer)
- Ting Sun (lead SuPy developer and current lead on SUEWS)

Contributors to the workshop

- Kit Benjamin (tester of all the Workshop activities)
- Vicky Lucas, Sonja Thoms (and others) at IEA, University of Reading (Videos)
- Hamid Omidvar (Jupyter videos, Python/Jupyter tutorials), Meg Stretton (Jupyter Video)
- Nils Wallenberg and Oskar Bäcklin (GIS and UMEP assistants during the Workshop)

1.1.3 Acknowledgements

- ERC Synergy urbisphere
- Newton Fund/Met Office CSSP China
- NERC
- FORMAS: Swedish Research Council for Sustainable Development

A large number of individuals have worked on the development of the software (SUEWS Grimmond et al. 2020, UMEP Lindberg et al. 2020, SuPy Sun et al. 2020) and the collection of the various datasets that are used in the tutorials. These are all gratefully acknowledged.

We thank the following organisations whose **data** are used, including:

- AmeriFlux
- ECMWF ERA5
- LUMA

1.1.4 Feedback

- If you find problems with the Manuals (Workshop, `UMEP <, SUEWS, SuPy) please raise a GitHub issue so it can be fixed for others in the group (saving everyone time)
- Do you have suggestions for other tutorials?
- What other topics would you like to see covered? Please add to GitHub or contact Sue

1.2 urbisphere Workshop Program: 8 - 10 July 2020

Welcome to the urbisphere **2020 Modelling Workshop**

Given this is the first *urbisphere* Modelling workshop our main goal is to get everyone familiar with several modelling tools that already exist.

As this is an online workshop (rather than being at University of Reading as planned) we will meet twice a day via TEAMS formally but we will keep the TEAMS channels open throughout each session so you can ask questions of each other and the people ready to provide help.

The Teams channel will be open for chat, and you can go into a DG (Discussion Group room)

Table 1.1: We will meet at these times each day on Teams

	UK	DE	GR
am	8:00	9:00	10:00
pm	12:30	13:30	14:30

Table 1.2: Program

Day Time	Topic	Other parallel activities
Before	*Installation or Updating of Software*	
Wednesday am	*Introduction* and *First UMEP Activity*	Observations database meeting
Wednesday pm	*SUEWS*	
Thursday am	*Spatial data*	
Thursday pm	*What else can UMEP do?*	Paris Olympics planning meeting
Friday am	*SuPy*	
Friday pm	*SuPy* and Wrap-up session	

Structure

- Each section has its own page with links to relevant materials.
- Initial material is general, later topics are more focused.
- The amount of time needed to undertake a topic is given.
- If you have gone to a tutorial site (these are hosted separately) make certain you return to the Workshop pages! At the end of each Workshop page the 'Next' button takes you forward.
- On each page there is a link to the *Need help page* where you will see various ways to move forward if someone is not available to help you (suggestions for improving this would be good).

Delivery Beyond the workshop manual there are videos, tutorials, lectures and links to other material (e.g. manuals, articles). This is a hands on workshop. People will be available to help throughout.

1.3 Create a GitHub account

If you do not have a GitHub account please create one.

Please search for GitHub login and create an account.

1.4 New to GitHub

GitHub defines itself as:

> "a code hosting platform for version control and collaboration. It lets you and others work together on projects from anywhere."

A good tutorial for GitHub can be found here. The following tutorials will help you beome familiar with git and GitHub:

1. Git Handbook: Learn about version control—in particular, Git, and how it works with GitHub.
2. GitHub-based workflow: This guide explains how and why GitHub flow works.

1.5 Software Version

As software is frequently updated, here are the versions used in the workshop. **Please update/install QGIS and UMEP before** the workshop. The rest of the software you will install/update during the workshop.

Table 1.3: Software Version and methods to install or update

Software	Version	Update	Installation
QGIS	3.14	Follow QGIS instructions	*Install*
UMEP	Update new version from Teams or GitHub	Update UMEP	*Install*
FUSION	4.00	*Used*	*Install in* tutorial
SuPy	2020.6.30	Update SuPy	*Installed in* tutorial
SUEWS	V2020b	See SuPy or if standalone	*Installed in* tutorial

1.6 Installation of QGIS

Note: You will need administrative rights to your computer to install QGIS on your computer.

This will take: ~10-20 minutes (depending on your internet download speed)

Activity

- Written instructions are available in the UMEP manual
- This Video gives detailed installation instructions for Windows
  - The differences for **Other Operating systems** are also discussed in the video.
- If you need to *update QGIS - go to need help*

Next: You are now ready to install UMEP

1.7 Installation of UMEP

This will this take: ~10 minutes

Prior to this activity: You need to have QGIS installed

Activity

- Follow video for instructions
- See the Getting Started section of the manual for written instructions
- When following these instructions, all Python dependencies should be installed automatically. However if this is not the case then follow the instructions in section 2.3 of the manual and install these packages.
- If you need to *update QGIS - go to need help*

Note: macOS users, if you have issues in UMEP installation due to `supy`, please follow *the instructions here* for a solution.

Next: You are now ready to run UMEP.

1.8 Workshop History

Table 1.4: Past workshops

When	Where	Version	Given to
8-10 July 2020	Reading [Online]	*Program*	urbisphere
August 2018	New York City	UMEP	ICUC10
28 Aug- 8 Sep 2017	Johor, Malaysia	UMEP	WMO Workshop Urban Meteorology, Environment and Climate Services
7-8 Nov 2016	Shanghai	City base Climate Services UMEP	WMO - IUWCS - CSSP China

Background Materials

Take a look through the following so you are familiar with what is avalable if you need help

- *Key papers and manuals and helpful links*
- *Jupyter Notebooks and Python*
- *Meteorology*
- *Metrics for model evaluation*
- *Glossary*

CHAPTER

TWO

FIRST UMEP ACTIVITY

This section is an introduction to GIS and UMEP, exemplified with a shadow casting exercise.

2.1 First QGIS and UMEP activity

This will take: ~30 min

Assumed prior to this

- *Installation of QGIS*
- *Installation of UMEP*

Activity

- Simple example of how to create shadow maps using QGIS and UMEP.
- Watch the Video below
- Tutorial Link

2.2 Basics of UMEP, QGIS and Spatial data

This will take: ~10 min

Assumed prior knowledge

- Completed *A First QGIS and UMEP activity*

Activity

- What is covered
  - QGIS and UMEP basic functionality
  - Spatial data for UMEP (Vector-Raster, DSM, Land cover etc.)
- Link to Video

CHAPTER

THREE

LOCAL SCALE - URBAN ENERGY BALANCE

This section includes an introduction to the SUEWS model and how to use it within UMEP.

3.1 Introduction To SUEWS

This will this take: ~30 minutes

Prior to this:

- QGIS and UMEP should be installed on you computer
- Completed the first UMEP activity

Meteorology Background

- Some brief background
- Introduction to urban surface energy and water balance (See *Introduction to Ward et al. 2016*, Oke et al. 2017)
- Scale in the urban environment (see Cleugh and Grimmond 2012, Oke et al. 2017)

Activity

- Introduction to local scale surface energy and water balance modelling for a specific point location
- Tutorial link

3.2 Sensitivity analysis of SUEWS using geodatasets

This will this take: ~45 minutes

Expected knowledge/activity before this

- *Introduction to SUEWS*

Activity

- Flexible local scale surface energy and water balance modelling for a specific point location using geodatasets
- Tutorial link

3.3 Spatial analysis urban energy and water balance

This will take: ~ 60 minutes

Expected knowledge/activity before this

- *Introduction To SUEWS*
- *Sensitivity analysis of SUEWS using geodatasets*

Activity

- In this tutorial you model spatial variations in the local scale surface energy and water balance
- Tutorial link

3.4 What else can SUEWS do?

- More capabilities of SUEWS will be adressed later in this Workshop when SuPy is *introduced*.
- See Studies that have used or evaluated SUEWS.

CHAPTER

FOUR

SPATIAL AND METEOROLOGICAL DATA FOR UMEP

This section includes examples on how to generate and process appropriate data for UMEP.

4.1 Generate modelling grid in QGIS

This will this take: ~10 minutes

Expected knowledge/activity before this

- *Basics of UMEP, QGIS and Spatial data*
- *Installation of UMEP*

Activity

- Learn how to generate a grid for local scale modelling using built-in functionallity in QGIS.
- Link to Video

4.2 Creating your own geodata

This will this take: ~45 minutes

Expected knowledge/activity before this

- *First QGIS and UMEP activity*
- *Basics of UMEP, QGIS and Spatial data*

Activity

- Create a digital surface model from Open Street Map data.
- Tutorial link

4.3 Meteorological data for UMEP

This will this take: ~10 minutes

Expected knowledge/activity before this

- *First QGIS and UMEP activity*
- *Basics of UMEP, QGIS and Spatial data*

Activity

- Get acquainted with the ERA5 meteorological download tool in UMEP.
- Get acquainted with the Meteorological Pre-processor in UMEP.
- Link to Video

Note: The ERA5 downloader needs CDSAPI configured before using. Please follow the official guide here to finish the configuration.

Constraints

- The actual download of data takes a long time (hours).

4.4 Generating geodata from LiDAR point clouds

Note: As this tutorial uses FUSION/LDV it is restricted to **Windows OS** only. You need administrative rights to your computer to install the software on your computer.

This will this take: ~90 minutes

Expected knowledge/activity before this

- *Basics of UMEP, QGIS and Spatial data*

Activity

- Create a DSM, CDSM and Land Cover from LiDAR point clouds using QGIS and FUSION/LVD.
- Tutorial link

4.5 Making use of online Web Services in QGIS

This will this take: ~10 minutes

Expected knowledge/activity before this

- *Basics of UMEP, QGIS and Spatial data*
- *Installation of UMEP*

Activity

- Learn how to connect to external data sources
- Learn how to acquire population density data from a Web Coverage Service.

- Link to Video

CHAPTER

FIVE

WHAT ELSE CAN UMEP DO?

In this section you will try out and learn about other tools found in UMEP. You will also get acquainted with the UMEP for Processing capabilities.

5.1 Micro scale - Solar radiation modelling

This will this take: ~45 minutes

Expected knowledge/activity before this

- *First QGIS and UMEP activity*
- *Basics of UMEP, QGIS and Spatial data*

Activity

- Solar Energy of Building Envelopes (SEBE).
- Tutorial link

5.2 UMEP - model parameters (z0, h/w etc)

This will this take: ~10 minutes

Expected knowledge/activity before this

- *First QGIS and UMEP activity*
- *Basics of UMEP, QGIS and Spatial data*
- *Introduction To SUEWS*
- *Sensitivity analysis of SUEWS using geodatasets*

Activity

- Get aquainted with tools in UMEP used for estimating morphological parameters.
- Link to Video

5.3 UMEP - Footprint modelling

This will this take: ~45 minutes

Expected knowledge/activity before this

- *First QGIS and UMEP activity*
- *Basics of UMEP, QGIS and Spatial data*
- *Introduction To SUEWS*
- *Sensitivity analysis of SUEWS using geodatasets*

Activity

- Footprint modelling in UMEP.
- Tutorial link

5.4 Micro scale - Pedestrian Radiant load (SOLWEIG)

This will this take: ~60 minutes

Expected knowledge/activity before this

- *First QGIS and UMEP activity*
- *Basics of UMEP, QGIS and Spatial data*

Activity

- Modelling 3D radiation fluxes and mean radiant temperature in urban environments
- Tutorial link

5.5 Workflow patterns - UMEP for the QGIS Processing toolbox

This will this take: ~45 minutes

Expected knowledge/activity before this

- *First QGIS and UMEP activity*
- *Basics of UMEP, QGIS and Spatial data*
- *Micro scale - Solar radiation modelling*

Activity

- Examples of workflow patterns in UMEP and introduction to UMEP for processing.
- Link to Tutorial

5.6 What else can UMEP do?

- Here is the full list of tutorials that exist.
- The Tool Applications section gives a full list of evaluation and application studies using UMEP and related models. If you know of others please raise an issue so we we can update the list.

CHAPTER

SIX

SUPY - SUEWS WHO TALKS PYTHON

This section is an introduction to SuPy, SUEWS that talks Python.

6.1 Preparation for using SuPy

This will take: ~30 mins

Prior to this activity

- Be familiar with SUEWS in UMEP
- *Introduction to Jupyter Notebooks*
- Jupyter Notebook Cheatsheet

Note: It is very important you do the right thing at this point. **Only do one of the following**

1. If you are a **Windows** user without a version of Python on you computer except for QGIS if so go *here*
2. If you already have Jupyter Notebooks installed use that
3. Otherwise follow *these instructions*

Activity

- *Installation of Jupyter Notebooks - follow the appropriate instructions for your operating system*
- If you already have Python3 and Jupyter Notebooks installed you do not need to do this.
- If you are a **Windows** user, *install the QGIS version* **OR** you can go to *Anaconda-based approach* for **General (all platforms)**.

Windows+QGIS: `osgeo`-based approach

- With your already *installed QGIS*, go to the start menu in Windows, locate **OSGeo4W Shell** and open it. If you automatically do not have administrative rights you need to right-click on **OSGeo4W Shell**, *Open file Location*, Right-click on **OSGeo4W Shell** again and choose *Run as Administrator*; type the following two commands:

```
py3_env
pip install notebook
```

- To start Jupyter Notebook type:

```
jupyter notebook
```

(or) General OS installation (all platforms): Anaconda-based approach

- Download Anaconda3:
- Install Anaconda3 (Jupyter Notebook included):

Note: If you use Anaconda for Python, it is better to choose conda-forge channel for package installation to have better compatibility with various scientific libraries.

6.2 Jupyter Notebooks: setting up your research-oriented coding environment

Time taken will be: ~ 20 min

Prior to this

- *Install Jupyter Notebooks*
- Read about *Jupyter Notebooks*

Activity

- Launching Jupyter Notebook

Note:

Only choose **ONE** of the two below that applies to your system

- using **Anaconda Navigator**:

- from **OSGeo4W Shell** (the video below automatically starts at: 2:40)

- Install **Jupyter Notebook Extensions** (Optional, but recommended): We *recommend* these. The method to install the extensions is here
- Jupyter Notebook operations:
- Basic operations (V, L)
  - editing and saving a notebook
  - adding and running cells
  - edit/command mode
  - take notes using Markdown
- Advanced tips (L)
  - add cells above/below
  - stop/restart kernel
  - cut (delete) / copy / paste cells
  - merge/split cells
  - structuring your notebook

  - main storyline
  - scratch pad
  - note-taking in markdown
  - basics
  - equation
  - magic with `line/cell-magic`
  - terminal commands
  - external modules (e.g., Fortran)

6.3 First Activity with SuPy

This will this take: ~30 minutes

Prior to this activity

- Familiarity with SUEWS
- SUEWS vs SuPy vs UMEP: see the SUEWS docs site
- *Jupyter Notebook: setting up your research-oriented coding workshop*

Activity

- Quickstart of SuPy Tutorial

6.4 SuPy - Impact study

This will take: ~45 minutes

Prior to this activity

- *Basics for running SuPy*

Activity

- In this activity you use will use SuPy to conduct an impact study

  You will explore the urban climate response to two scenarios:

  1. modified urban surface properties (e.g., albedo), such as may come about from people's behaviour (e.g. planning regulations, manintenace over time)
  2. varying atmospheric forcing (e.g., air temperature), such as may come about from climate changing

6.5 SuPy - interacting with external models

This will take: ~25 minutes

Prior to this activity

- Understand the basic data structures of SuPy forcing:
  - basics given in the *SuPy impact study*
  - more details see SuPy manual

Activity

- In this activity, you will build a simple model for anthropogenic heat emissions and use it to interact with SuPy for urban climate simulations

Background

- Anthropogenic heat schemes used in SUEWS

6.6 SuPy - calculating own parameters

This will take: ~40 minutes

Prior to this activity

- Understand the basic data structures of SuPy model states:
  - basics given in *SuPy - Impact study*
  - more details see SuPy manual

Activity

- In this activity, you will calculate appropriate parameters for an AmeriFlux site and conduct SuPy simulations using these parameters. This example uses data from an AmeriFlux site *US-AR1 for 2010* (`download data`).

Background

- Introduction to AmeriFlux network

CHAPTER

SEVEN

CONSIDERATIONS WHEN APPLYING SUEWS TO YOUR OWN STUDY AREA

The SUEWS manual (Grimmond et al. 2020) provides some guidance for the running at a new site. In particular read this section. *You can* calculate some parameters from observations.

7.1 Model Inputs

- Files required to run SUEWS
- The data are needed for every grid (can be any shape) area in the domain to be modelled.

7.1.1 Model Physics options

- RunControl
- the selection of which submodels are to be used
- choose after the other decisions

7.1.2 Forcing data

- Meteorological variables needed

Data Sources

- Observations
- Climate data e.g. ERA5, projections
- Coupled Model e.g. WRF

Height consideration of the forcing data

- *Several heights need to be considered*

7.1.3 Site Information

- Site parameters

7.1.4 Initial Conditions

- Initial conditions
- Conducting a model spinup for a number of years allows for the influence of the conditions selected to begin with to become less critical.
- Key conditions to consider
  - leaf area index (leaf-on, leaf-off) - this is critical for evaporation
  - soil moisture state - this is critical for evaporation
- less critical
  - surface state (as long as modelling for many days) as these will change rapidly.
  - it is easy to just set this to be dry (e.g. summer) or wet (e.g. if rain has just occured)
- meteorological variables
  - These can be determined from a wide range of data (e.g. the same source as your forcing data)
- snow conditions
  - This needs to be done very carefully for a short model run. A long model run starting from snow-free conditons would ensure that the snow accumulation occurs at the right time etc

7.2 Heights that need to be considered in SUEWS

A wide number of height needs to be considered before applying SUEWS. The following figure and table help to clarify the differences between these

Table 7.1: Heights important for SUEWS

Name	Symbol	Definition	Comment
Height of Forcing data for Local scale SUEWS application		Height above ground level	This should be above the RSL
Roughness SubLayer	RSL	Layer that is directly influenced by individual roughness elements	includes the UCL but is below the ISL
Urban canopy layer	UCL	Height of the roughness elements (RE)	Building, trees and other RE
Inertial sub layer or Constant flux layer	ISL	The layer where the influence of the individual RE are blended	
Height above ground level	agl		height of RE, height of ISL, mid-height of building
Height above sea level	asl	Altitude of land surface	can be different between model grids (e.g. ERA5 grid and SUEWS grids)

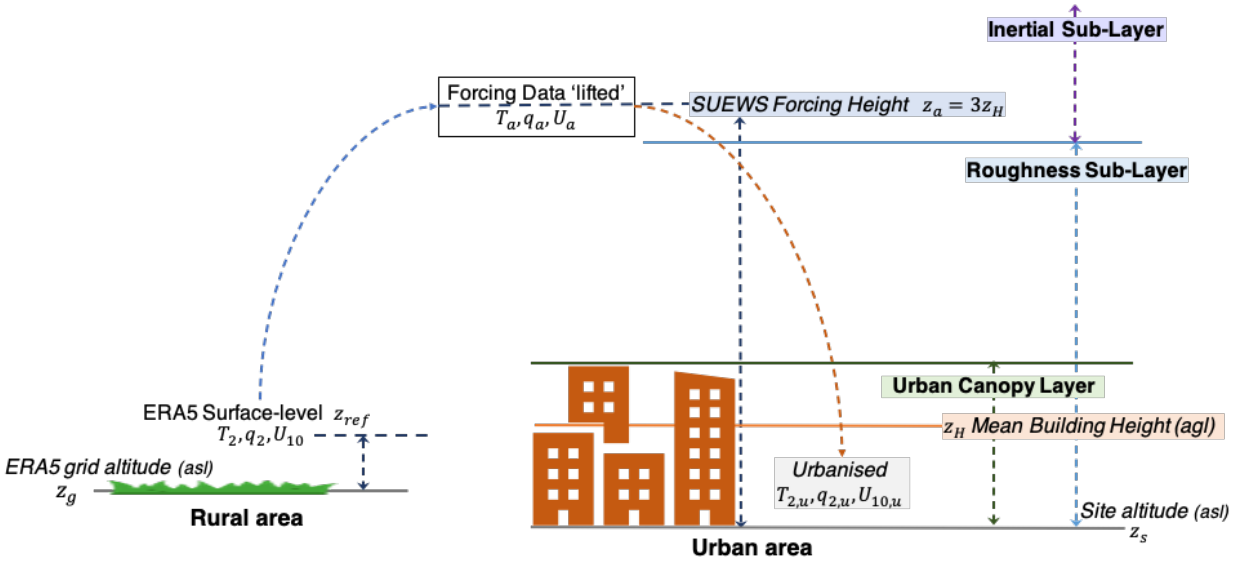


Fig. 7.1: Critical heights that need to be considered: rural ERA5 surface-level data are "lifted" to the SUEWS forcing height z_a after altitude differences (above sea level, asl) are accounted for the forcing height needs to consider the urban canopy height (z_H) to ensure that is within the inertial sub-layer and above the roughness sub-layer (RSL). The SUEWS RSL module is used to determine the variables (air temperature T, specific humidity q and wind speed U) at the height desired within the RSL for an application. Figure not to scale.

7.3 Site parameters

A wide range of parameters can be set to characterise each grid in your study area. The following explains what things you should consider as you decide on the amount of effort needs to be spent on each fo these. Somed data can be obtained from literature and others are very site specifice. Dpending on your goal for applying the model you may want to spend more or less effort ensuring the parameters values are good/reasonable/OK for your site. A good starting point is to concentrate on the parameters that are initially most importnat and update the others once you have some initial runs.

Make certain you have read this section and this section of the SUEWS manual.

Table 7.2: Site parameters

Name	Type
SUEWS_AnthropogenicEmission.txt	*Anthropogenic Heat*
SUEWS_BiogenCO2.txt	*Water within Grid*
SUEWS_Conductance.txt	*Conductances*
SUEWS_Irrigation.txt	*Water within Grid*, *Function*
SUEWS_NonVeg.txt	*Materials*
SUEWS_OHMCoefficients.txt	*Materials*
SUEWS_Profiles.txt	*Function*
SUEWS_SiteSelect.txt	*All*
SUEWS_Snow.txt	*Water within Grid*, *Function*
SUEWS_Soil.txt	*Materials*
SUEWS_Veg.txt	*Materials*
SUEWS_Water.txt	*Water within Grid*
SUEWS_WithinGridWaterDist.txt	*Water within Grid*

7.3.1 Anthropogenic Heat

- This varies significantly across a city and between cities.
- This can be modelled offline (e.g. using LUCY, LQF) and then values supplied in the meterological input. The disadvantage of this is that the results are static (e.g. do not repond to temperature) but may allow for more sophisticated/detailed modeling of the values.
- The offline models can be used to derive parameters for models within SUEWS (e.g. Ward and Grimmond 2017). This allows for the responses to conditions to be captured and then modelling for other conditions to be predicted.

Approaches

- U approach (Ao et al. 2018)
- V approach (Jarvi et al. 2011)
- DASH approach (Capel-Timms et al. 2020)
- LUCY/LQF approach (Allen et al. 2011, Lindberg et al. 2013, Gabey et al. 2019)
- GQF approach (Iamarino et al. 2012, Gabey et al. 2019)

References

- Allen L, F Lindberg, CSB Grimmond 2011: Global to city scale model for anthropogenic heat flux, International J. of Climatology, 31, 1990-2005 10.1002/joc.2210
- Ao Xiangyu, CSB Grimmond, HC Ward, AM Gabey, Jianguo Tan, Xiuqun Yang, Dongwei Liu, Xing Zhi, Hongya Liu, Ning Zhang Evaluation of the Surface Urban Energy and Water balance Scheme (SUEWS) at a dense urban site in Shanghai: Sensitivity to anthropogenic heat and irrigation J Hydrometeorology 19, 1983–2005,https://doi.org/10.1175/JHM-D-18-0057.1
- Capel-Timms I, ST Smith, T Sun, S Grimmond Dynamic Anthropogenic activitieS impacting Heat emissions (DASH v1.0): Development and evaluation. In reivew
- Gabey A, S Grimmond, I Capel-Timms 2019: Anthropogenic Heat Flux: advisable spatial resolutions when input data are scarce Theoretical and Applied Climatology 135 (1-2), 791-807 https://doi.org/10.1007/s00704-018-2367-y

- Iamarino M, Beevers S, CSB Grimmond 2012: High Resolution (Space, Time) Anthropogenic Heat Emissions: London 1970-2025 International J. of Climatology 32, 1754-1767 10.1002/joc.2390
- Järvi L, CSB Grimmond, A Christen 2011: The Surface Urban Energy and Water Balance Scheme (SUEWS): Evaluation in Vancouver and Los Angeles. J. of Hydrology, 411, 219-237 10.1016/j.jhydrol.2011.10.001
- Lindberg F, CSB Grimmond, N Yogeswaran, S Kotthaus, L Allen 2013: Impact of city changes and weather on anthropogenic heat flux in Europe 1995-2015 Urban Climate,4, 1–15 10.1016/j.uclim.2013.03.002
- Ward HC, S Grimmond 2017: Using biophysical modelling to assess the impact of various scenarios on summertime urban climate across Greater London Landscape and Urban Planning 165, 142–161, https://doi.org/10.1016/j.landurbplan.2017.04.001

7.3.2 Conductances

- For urban areas, at the moement, we *recommend* the values that the model come with. However, for areas which are largely vegetated we *recommend* that other values are used. Omidvar et al. (2020) provides values for a range of different vegetation types.

How to determine your own values?

- if you have observations for a long period (e.g. changing phenology) then you can calculate your own values. The following papers explain how:
  - *Omdivar et al. (2020)* - there are Jupyter Notebooks with Python code for many model parameters **start here**
  - `Ward et al. (2016) <Ward2016>
  - *Järvi et al. (2011)*
  - `Grimmond and Oke (1991) <G091>

7.3.3 Materials

For each material type there are different characteristics needed for the radiative, conductive and water behaviour.

Table 7.3: Material related parameters

Type	Comment
Height	for larger roughness elements this is needed
Albedo **important**	This changes with phenology
Emissivity	
Surface water storage capacity	amount of water that is intercepted before drainage occurs
Runoff/drainge of water from the surface	drainage rate after interception water storage is full
Storage Heat	Choice from three sub-models (OHM, AnOHM, ESTM) parameters needed vary with which is used
LAI/Phenology **important**	for vegetation it is necessary to capture the seasonal response of the vegetation.
Infiltration rate	does water pond on the surface or drain into the soil?
Soil	soil density, hydraulid conducitivity, depth,

How to determine your own values?

- If you have observations for a long period (e.g. changing phenology) then you can calculate your own values. The following paper explains how:

  - Omdivar et al. (2020) - there are Jupyter Notebooks with Python code for many model parameters *start here*

7.3.4 Function

- Does day light savings occur?

Table 7.4: Profile types

Type	Comment
Energy Use	When do people do things on work days? non-work days?
External Water use	garden irrigation, car cleaning, street cleaning, dust suppression,automatic or manual, flood irrigation to maintain a soil mositure content
Snow clearing	when does this occur? what is the prioriy order for a region (e.g. major vs minor roads)
Population density	day time densities (work, school relate)(Night - census where people live); DASH model allows fopoy dynamic occupancy (Capel-Timms et al. 2020)

References

- Capel-Timms I, ST Smith, T Sun, S Grimmond Dynamic Anthropogenic activitieS impacting Heat emissions (DASH v1.0): Development and evaluation. In reivew

7.3.5 Water within Grid

- How does water move between areas?
- Does the water from the roof all go in to drains or does some go on to grass?
- All surfaces can be set to move water fronn one land cover to another - the constraint is that is it must add up to 100%
- this will influence the soil moisture available (or not) to plants
- does irrgation all go to vegetation or does flow to the roads if too wet

7.3.6 All

- all the different characteristic types need to be considered

CHAPTER

EIGHT

NEW TUTORIALS

- Post-2020 Workshop tutorials are to:
  - *Calculate your own SUEWS parameters*

CHAPTER

NINE

FUTURE TOPICS

- Using SuPY with EnergyPLus
- DASH
- LUCY/LQF
- Benchmark system
- WRF/SUEWS
- Urban/rural differences
- ESTM
- Suggestions add to GitHub

CHAPTER

TEN

CALCULATE YOUR OWN SUEWS PARAMETERS FROM OBSERVATIONS

This is based on *Omidvar et al. (2020)*.

- leaf area index (LAI)
- Albedo (α)
- Roughness
- Conductances

10.1 Preparing to Calculate your own parameters

This will take: ~ XX minutes (note there are 5 topics)

Prior to this activity

- Install and run notebooks Jupyter Notebooks
- Basics for running SuPy

Activity

In these tutorials you can calculate various SUEWS parameters before running the model for the specific site and vegetation type. The parameters discussed here are:

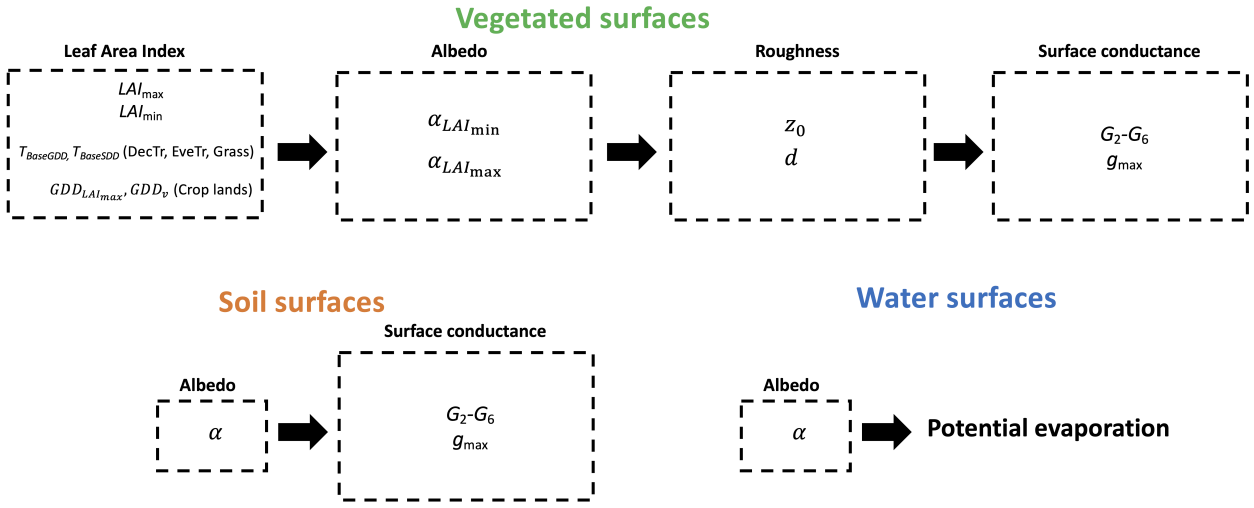
- LAI
- albedo
- surface roughness length and displacement height
- surface conductances.

The Figures below shows the order in which parameters should be derived:

These tutorials are based on the calculations undertaken in Omidvar et al. (2020).

In this example we use meteorological observations from AmeriFlux (https://ameriflux.lbl.gov/) network. The data required are air temperature, incoming shortwave radiation, upwelling shortwave radiation, station pressure, relative humidity, wind speed, precipitation, net all-wave radiation, sensible heat flux and latent heat flux, ideally storage heat flux (os soil heat flux if a simple surface), and the momentum flux. Wind direction is also very helpful.

Reference



- Omidvar H, T Sun, S Grimmond, D Bilesbach, A Black, J Chen, Z Duan, Z Gao, H Iwata, JP McFadden. Surface [Urban] Energy and Water Balance Scheme in non-urban areas: developments, parameters and performance, (in review)

10.1.1 Steps to using these notebooks:

Terms in the notebooks are defined *here*

1- Download the following files and codes (all the files are available here. You need to unzip the folders):

- data : data that are necessary to calculate parameters (put them in the same directory as notebooks).
- Utility functions: these are the utility functions that are called in notebooks. Put them in the same directory as notebooks.
  - Albedo and LAI utility
  - Conductance utility
  - Roughness utility
- CSV files containing site information and parameters. Put them in the same directory as the notebook:
  - site information
  - site parameters . You can then change these parameters if you like to tune the sites.
- runs folder contains SUEWS files which are used by SuPy
- figs folder to write the figure into (it can be empty initially). Put them in the same directory as the notebook.
- outputs folder to write down pickle files. The structure of the file should be as it is in the link, but the folders (LAI, albedo etc.) can be empty initially. Put them in the same directory as notebook.

Note: after downloading the above files an folders, the structure of the directory where you are running the notebooks should be as same as here.

2- Download this environment file. Go to the directory where the file is. Type

```
conda env create -f environment.yml
```

After all the packages are installed, type

```
conda activate SUEWS_parameters
```

This activates the the created environment. In the same environment, open Jupyter notebook.

3- Then you can run each notebook in the same order as the tutorials taht follow. `

10.2 Terminology used in the following Python Notebooks

For Leaf Area Index

Table 10.1: Terms Used in LAI

Name, SUEWS/SuPy	Definition
BaseTe, SuPy	Base temperature for initiating sensence degree days (SDD) for leaf off. [°C]
BaseT1, SUEWS	Base Temperature for initiating growing degree days (GDD) for leaf growth. [°C]
LAI_max, SUEWS	maximum LAI
LAI_min, SUEWS	leaf-off wintertime value
GDD, SUEWS	Growing degree days
SDD, SuPy	Senescence degree days

All or Multiple

Table 10.2: Terms Used in All/Multiple

Name	Definition
DecTr, SUEWS	Deciduous trees and shrubs
EveTr, SUEWS	Evergreen trees and shrubs
Grass, SUEWS	Grass surface
SWIN, SUEWS	Incoming shortwave radiation (Kdown) [W m-2]
SWOUT, SuPY	Outgoing shortwave radiation (Kup) [W m-2]

Albedo

Table 10.3: Terms Used in Albedo

Name	Definition
α_{LAImax}	Albedo of vegetation when LAI is equal to LAI_max
α_{LAImin}	Albedo of vegetation when LAI is equal to LAI_min

Conductances

Table 10.4: Terms Used in Conductances

Name	Definition
SMD, SuPy	Soil moisture deficit for bare soil surface [mm]

Roughness

Table 10.5: Terms Used in Roughness

Name	Definition
z_0, SuPy	Roughness length for momentum [m]
d, SuPy	Zero-plane displacement height [m]
Obukhov length, SuPy	Stability parameter
$USTAR$	Friction velocity

10.3 Calculate Parameter Tutorials

These tutorials demonstrate how to determine SUEWS parameters using Python and Jupyter Notebooks

Note: These tutorials are in Jupyter notebooks and can be executed and viewed online.

The following section was generated from source/Parameters/tutorials/param1-LAI.ipynb

10.3.1 Leaf Area Index related parameters

Necessary packages for analysis:

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from datetime import datetime
from pathlib import Path
import supy as sp
import os
from shutil import copyfile
import geopandas as gpd
import pickle
pd.options.mode.chained_assignment = None
from IPython.display import clear_output
import warnings
warnings.filterwarnings('ignore')
```

This is a custom package specifically designed for this analysis. It contains various functions for reading and computing and plotting.

[2]:
```
from alb_LAI_util import generate_array_dif, generate_array_same, modify_attr,func_
↪parse_date
```

Parameters needed to calcuate LAI for non-crop vegetation (see Background https://umep-workshop.readthedocs.io/en/latest/Parameters/CalcBG.html)

- $BaseT$ -
- $BaseTe$
- LAI_{MAX}
- LAI_{Min}

Necessary functions for analysis

Read in the data (including LAI data)

[3]:
```
def read_data(year, name):

    df = pd.read_csv('data/MODIS_LAI_AmeriFlux/statistics_Lai_500m-' + name +
                     '.csv')
    if name!='crop':
        df.columns = ['product'
                      ] + [i.split(' ')[1] for i in df.columns if i != 'product']

    df = df.filter(['modis_date', 'value_mean'])

    df_period = df[[i.startswith('A' + str(year)) for i in df.modis_date]]
    df_period.loc[:, 'DOY'] = [
        int(i.split('A' + str(year))[1]) for i in df_period.modis_date
```

(continues on next page)

(continued from previous page)

```
    ]
    df_period = df_period.set_index('DOY')

    copyfile("./runs/data/" + name + "_" + str(year) + "_data_60.txt",
             "runs/run/Input/Kc_2012_data_60.txt")
    df_forcing = pd.read_csv('runs/run' + '/Input/' + 'Kc' + '_' + '2012' +
                             '_data_60.txt',
                             sep=' ',
                             parse_dates={'datetime': [0, 1, 2, 3]},
                             keep_date_col=True,
                             date_parser=func_parse_date)

    all_sites_info = pd.read_csv('site_info.csv')
    site_info = all_sites_info[all_sites_info['Site Id'] == name]
    df = pd.DataFrame({
        'Site': [name],
        'Latitude': [site_info['Latitude (degrees)']],
        'Longitude': [site_info['Longitude (degrees)']]
    })

    path_runcontrol = Path('runs/run' + '/') / 'RunControl.nml'
    df_state_init = sp.init_supy(path_runcontrol)

    df_state_init ,level= modify_attr(df_state_init, df, name)

    grid = df_state_init.index[0]
    df_forcing_run = sp.load_forcing_grid(path_runcontrol, grid)

    df_output, df_state_final = sp.run_supy(df_forcing_run,
                                            df_state_init,
                                            save_state=False)

    return df_output, df_state_final, df_state_init, df_period, grid, df_forcing_run,
↪level
```

Calculating some of the variables of the models (LAI,GDD,SDD,Tair)

[4]:
```
def calc_vars(df_output, grid, df_forcing_run,year,level):

    df_output_2 = df_output.loc[grid, :]
    df_output_2 = df_output_2[df_output_2.index.year >= year]

    lai_model = pd.DataFrame(df_output_2.SUEWS.LAI)
    a = lai_model.index.strftime('%j')
    lai_model['DOY'] = [int(b) for b in a]

    if(level==1):
        nameGDD='GDD_DecTr'
        nameSDD='SDD_DecTr'
    elif(level==0):
        nameGDD='GDD_EveTr'
        nameSDD='SDD_EveTr'
```

(continues on next page)

(continued from previous page)

```
    if(level==2):
        nameGDD='GDD_Grass'
        nameSDD='SDD_Grass'

    GDD_model = pd.DataFrame(df_output_2.DailyState[nameGDD])
    a = GDD_model.index.strftime('%j')
    GDD_model['DOY'] = [int(b) for b in a]
    GDD_model = GDD_model.dropna()
    GDD_model = GDD_model.iloc[1:]

    SDD_model = pd.DataFrame(df_output_2.DailyState[nameSDD])
    print(SDD_model)
    a = SDD_model.index.strftime('%j')
    SDD_model['DOY'] = [int(b) for b in a]
    SDD_model = SDD_model.dropna()
    SDD_model = SDD_model.iloc[1:]

    Tair = df_forcing_run.Tair.resample('1d', closed='left',
                                        label='right').mean()
    Tair = pd.DataFrame(Tair)
    a = Tair.index.strftime('%j')
    Tair['DOY'] = [int(b) for b in a]

    return lai_model, GDD_model, SDD_model, Tair,nameGDD,nameSDD
```

Calculate and plot the LAI for the site being analysed. The data are saved to a pickle file.

[5]:
```
def LAI_tune(year,name):
    df_output,df_state_final,df_state_init,df_period,grid,df_forcing_run,level = read_
↪data(year,name)
    lai_model,GDD_model,SDD_model,Tair,nameGDD,nameSDD=calc_vars(df_output,grid,df_
↪forcing_run,year,level)
    clear_output()

    with open('outputs/LAI/'+name+'-'+str(year)+'-MODIS.pkl','wb') as f:
        pickle.dump(df_period, f)
    with open('outputs/LAI/'+name+'-'+str(year)+'-Model.pkl','wb') as f:
        pickle.dump(lai_model, f)

    attrs=[
    df_state_init.loc[:,'laimin'].loc[grid][level],
    df_state_init.loc[:,'laimax'].loc[grid][level],
    df_state_init.loc[:,'basete'].loc[grid][level],
    df_state_init.loc[:,'baset'].loc[grid][level]
    ]

    with open('outputs/LAI/'+name+'-attrs.pkl','wb') as f:
            pickle.dump(attrs, f)

    plt.rcParams.update({'font.size': 15})
    fig,axs=plt.subplots(2,1,figsize=(20,10))
    ax=axs[0]
    ax.plot(lai_model.DOY,lai_model.LAI,color='b',label='SUEWS-LAI')
    df_period.value_mean.plot(color='r',label='MODIS-LAI',ax=ax)
```

(continues on next page)

(continued from previous page)

```
    ax.set_ylabel('LAI')
    ax.legend()
    ax.set_title(name+'-'+str(year))
    ax.set_xlabel('')

    ax=axs[1]
    plt.scatter(Tair.DOY,Tair.Tair,color='k')



    try:
        max_y=30
        for gdd_day in [90,100,110,120,130,140,150,170,180,210,260]:
            a=GDD_model[GDD_model.DOY==gdd_day][nameGDD]
            plt.plot([gdd_day,gdd_day],[-15,max_y],color='r')
            plt.annotate(str(np.round(a.values[0],0)),(gdd_day-5,-14),color='r',
↪rotation=90)


        for sdd_day in [150,170,180,200,250,270,300,310,320,330,340,350,360]:
            a=SDD_model[SDD_model.DOY==sdd_day][nameSDD]
            plt.plot([sdd_day,sdd_day],[-15,max_y],color='b')
            plt.annotate(str(np.round(a.values[0],0)),(sdd_day-5,-14),color='b',
↪rotation=90)
    except:
        pass

    ax.plot([0,365],[df_state_init.basete.iloc[0][0],df_state_init.basete.iloc[0][0]],
↪label='BaseTe',color='k')
    ax.plot([0,365],[df_state_init.baset.iloc[0][0],df_state_init.baset.iloc[0][0]],'-
↪-k',label='BaseT')
    ax.legend()
    plt.ylabel('Tair (C)')
    plt.xlabel('DOY')
    ax.set_xlim(left=0,right=365)
```

Now evaluate the LAI parameters at other 'Test' sites. Calculate and plot LAI (and save data to pickle files)

[6]:
```
def LAI_test(years,name):

    plt.rcParams.update({'font.size': 12})
    fig,axs=plt.subplots(len(years),2,figsize=(20,13))

    counter=-1
    for year in years:
        print(name+'-'+str(year))
        counter=counter+1
        df_output,df_state_final,df_state_init,df_period,grid,df_forcing_run,level =␣
↪read_data(year,name)
        lai_model,GDD_model,SDD_model,Tair,nameGDD,nameSDD=calc_vars(df_output,grid,
↪df_forcing_run,year,level)
        clear_output()
```

(continues on next page)

(continued from previous page)

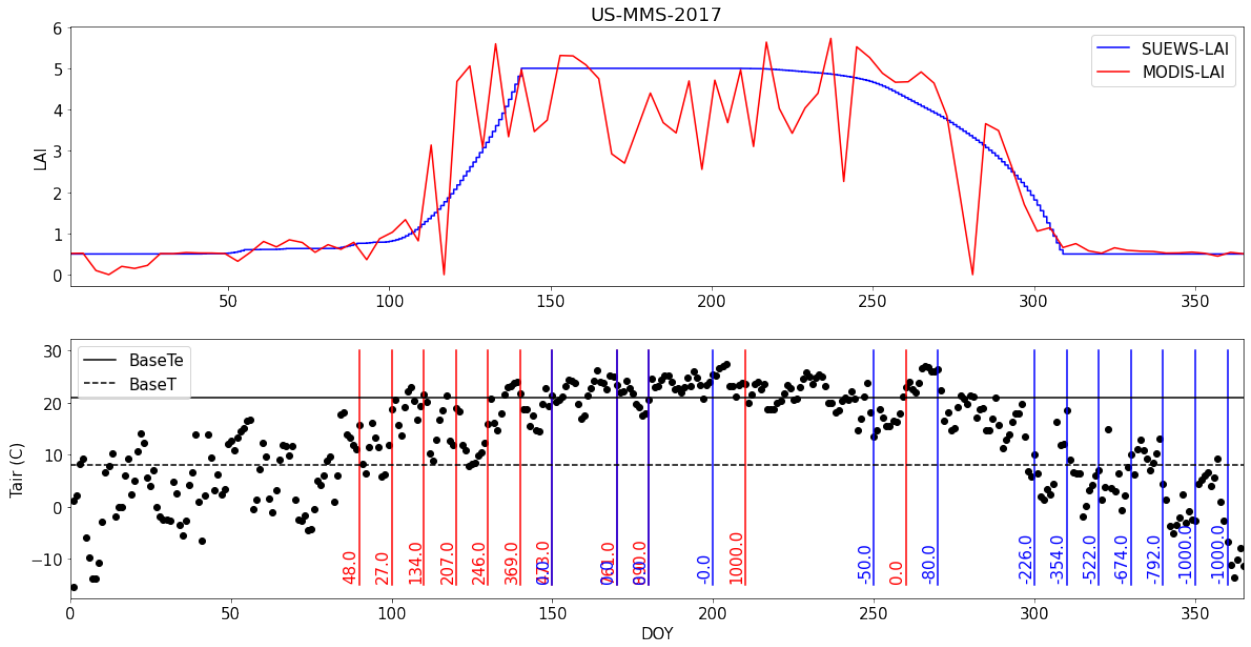
```
        with open('outputs/LAI/'+name+'-'+str(year)+'-MODIS.pkl','wb') as f:
            pickle.dump(df_period, f)
        with open('outputs/LAI/'+name+'-'+str(year)+'-Model.pkl','wb') as f:
            pickle.dump(lai_model, f)


        ax=axs[counter][0]
        ax.plot(lai_model.DOY,lai_model.LAI,color='b',label='SUEWS-LAI')
        df_period.value_mean.plot(color='r',label='MODIS-LAI',ax=ax)
        ax.set_ylabel('LAI')
        if counter==0:
            ax.legend()
        if counter!=len(years)-1:
            ax.set_xlabel('')
        ax.set_title(name+'-'+str(year))




        ax=axs[counter][1]
        ax.scatter(Tair.DOY,Tair.Tair,color='k')
        try:
            max_y=30
            for gdd_day in [90,110,130,150]:
                a=GDD_model[GDD_model.DOY==gdd_day][nameGDD]
                ax.plot([gdd_day,gdd_day],[-15,max_y],color='r')
                ax.annotate(str(np.round(a.values[0],0)),(gdd_day-10,-14),color='r',
↪rotation=90)


            for sdd_day in [300,320,340,360]:
                a=SDD_model[SDD_model.DOY==sdd_day][nameSDD]
                ax.plot([sdd_day,sdd_day],[-15,max_y],color='b')
                ax.annotate(str(np.round(a.values[0],0)),(sdd_day-10,-14),color='b',
↪rotation=90)
        except:
            pass
        ax.plot([0,365],[df_state_init.basete.iloc[0][0],df_state_init.basete.
↪iloc[0][0]],label='BaseTe',color='k')
        ax.plot([0,365],[df_state_init.baset.iloc[0][0],df_state_init.baset.
↪iloc[0][0]],'--k',label='BaseT')
        if counter==0:
            ax.legend()
        ax.set_ylabel('Tair (C)')
        if counter==len(years)-1:
            ax.set_xlabel('DOY')
        ax.set_xlim(left=0,right=365)
        ax.set_title(name+'-'+str(year))

    plt.savefig('figs/'+name+'-LAI.png',dpi=300,bbox_inches = 'tight',pad_inches = 0.
↪01)
```

Plot for all sites

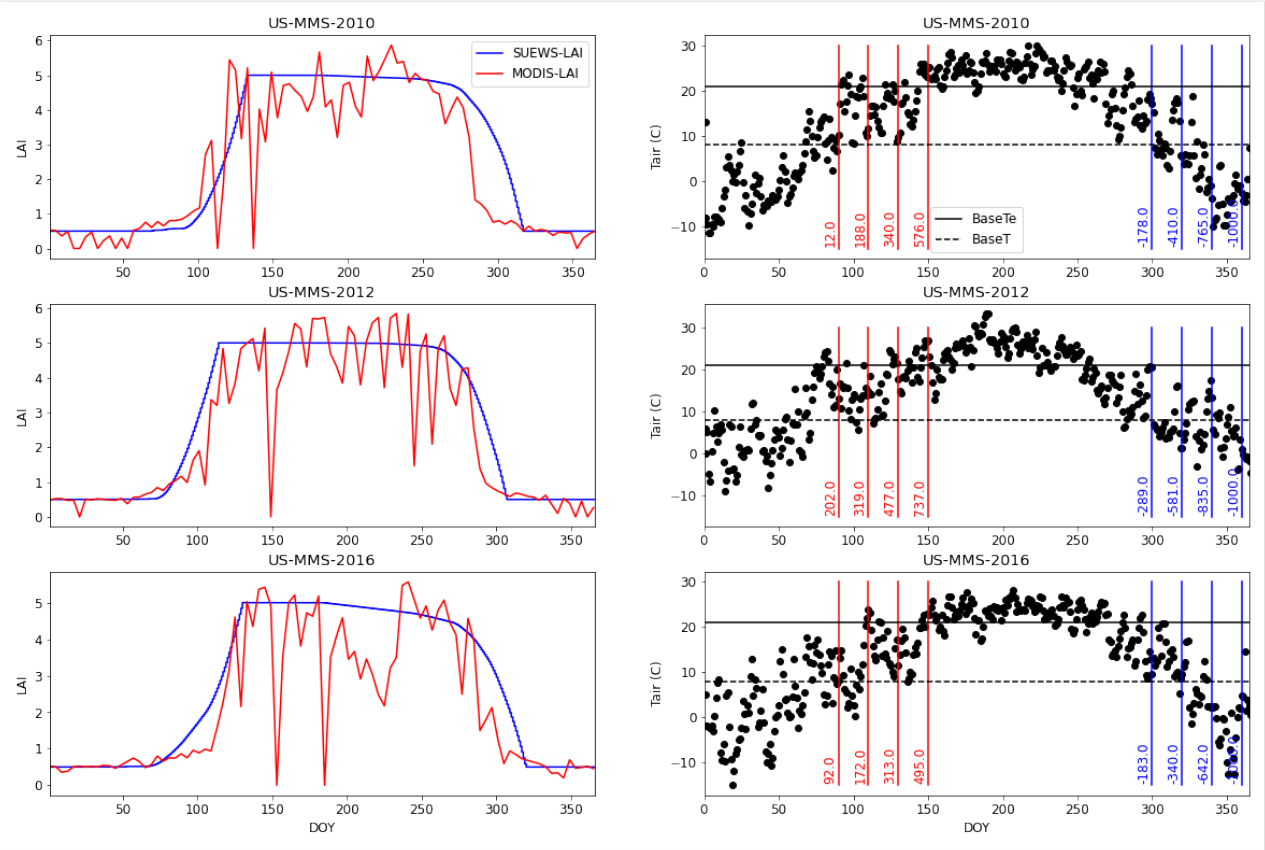
DecTr

US-MMS

[7]:
```
name='US-MMS'
year=2017
LAI_tune(year,name)
```



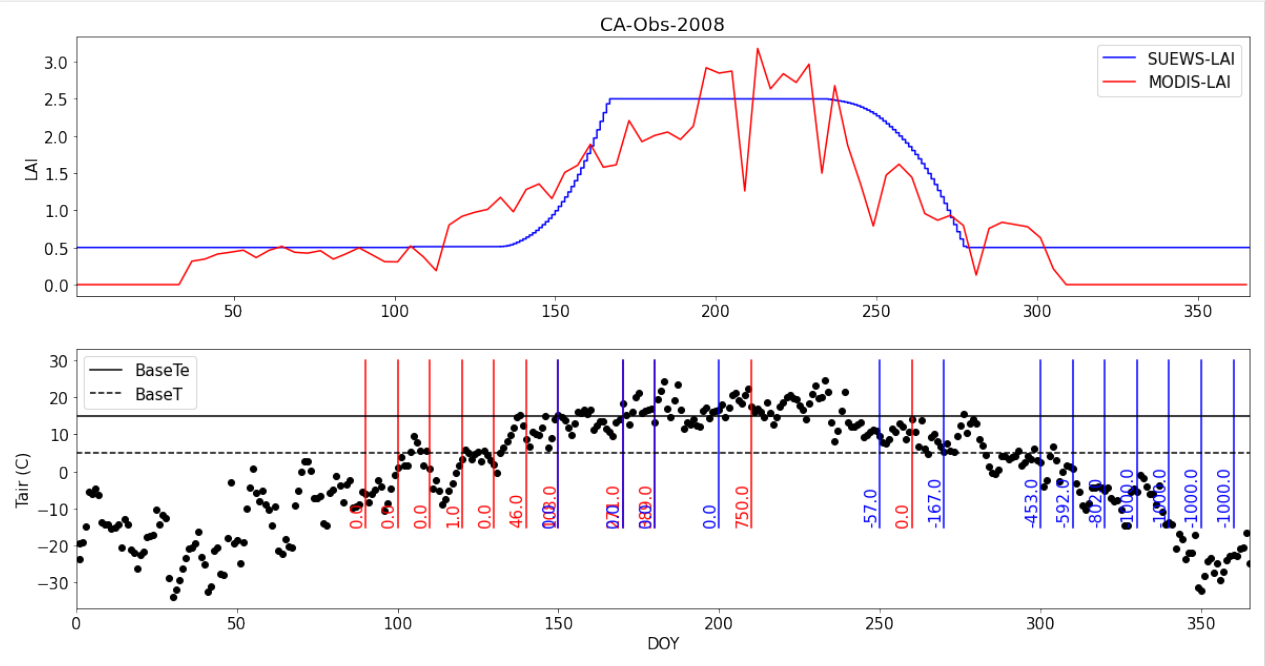
[8]:
```
name='US-MMS'
years=[2010,2012,2016]
LAI_test(years,name)
```



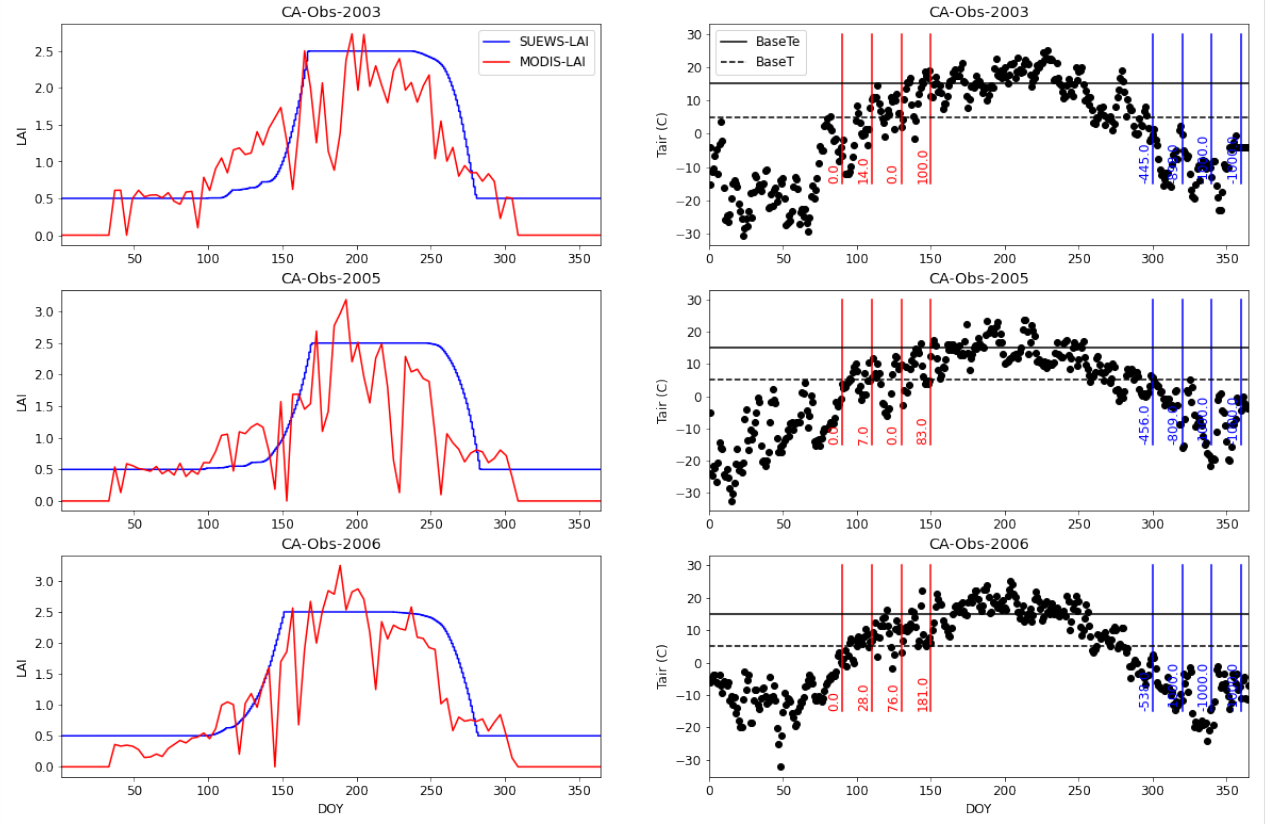
EveTr

CA-Obs

[9]:
```
name = 'CA-Obs'
year = 2008
LAI_tune(year, name)
```



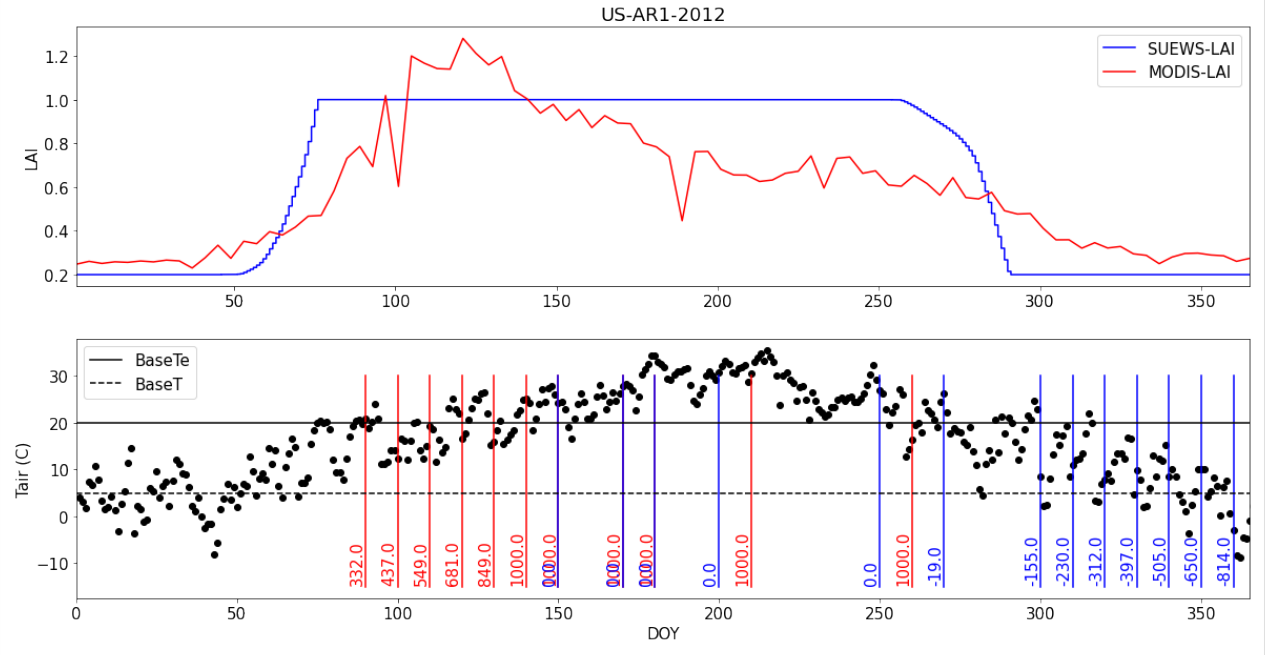
[10]:
```
name = 'CA-Obs'
years = [2003, 2005, 2006]
LAI_test(years, name)
```



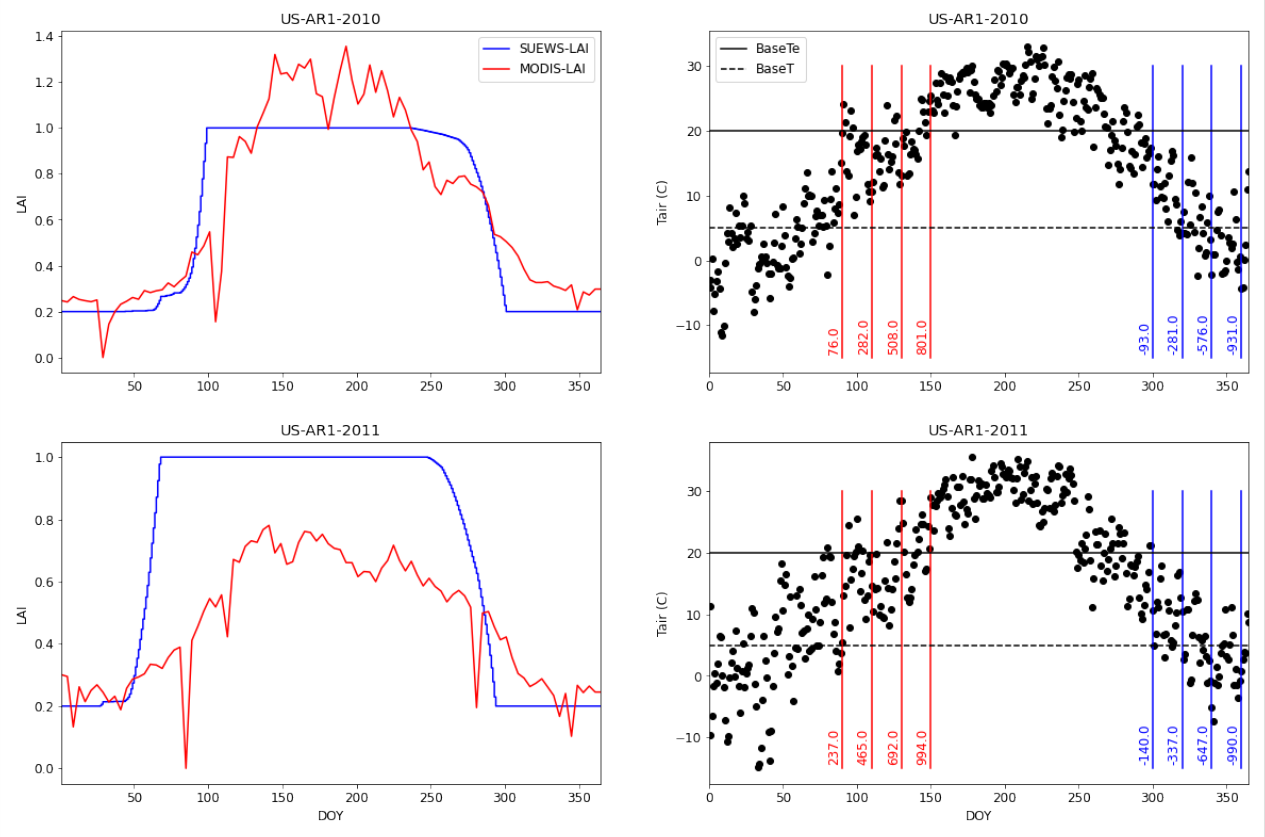
Grass

US-AR1

[11]:
```
name = 'US-AR1'
year = 2012
LAI_tune(year, name)
```



[12]:
```
name = 'US-AR1'
years = [2010, 2011]
LAI_test(years, name)
```



End of `Parameters/tutorials/param1-LAI.ipynb`

The following section was generated from `source/Parameters/tutorials/param2-albedo.ipynb`

10.3.2 Albedo parameters

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from datetime import datetime
from pathlib import Path
import supy as sp
import os
from shutil import copyfile
import geopandas as gpd
import pickle
pd.options.mode.chained_assignment = None
from IPython.display import clear_output
```

This is a custom package specifically designed for this analysis. It contains various functions for reading and computing and plotting.

[3]:
```
from alb_LAI_util import generate_array_dif, generate_array_same, modify_attr,func_
↪parse_date
```

Parameters to tune for α

- $\alpha_{LAI_{max}}$
- $\alpha_{LAI_{min}}$

Necessary functions for analysis

Reading data, calculating α and plotting gainst the observation (saving to pickle files)

[26]:
```
def read_plot(years,name,multiple_run=0):

    for year in years:
        print(name+'-'+str(year))
        df=pd.read_csv('data/data_csv_zip_clean/'+name+'_clean.csv.gz')
        df.time=pd.to_datetime(df.time)
        df=df.set_index(['time'])


        period_start=str(year)+'-01-01'
        period_end=str(year+1)+'-01-01'
        df_period=df[(df.index>=period_start) & (df.index<period_end)]



        df_period=df_period[df_period.SWIN>5]
        df_period=df_period[(df_period.index.hour <=14) & (df_period.index.hour >=10)]
        alb_raw=df_period['SWOUT']/df_period['SWIN']
        alb_raw=alb_raw.resample('1D').mean()
        alb=alb_raw[(alb_raw>0) & (alb_raw<1)]


        alb_avg_day=pd.DataFrame(alb,columns=['alb'])



        a=alb_avg_day.index.strftime('%j')
        alb_avg_day['DOY']=[int(b) for b in a]


        copyfile("./runs/data/"+name+"_"+str(year)+"_data_60.txt", "runs/run/Input/Kc_
↪2012_data_60.txt")
        df_forcing=pd.read_csv('runs/run'+'/Input/'+'Kc'+'_'+'2012'+'_data_60.txt',
↪sep=' ',
                                        parse_dates={'datetime': [0, 1, 2, 3]},
                                        keep_date_col=True,
                                        date_parser=func_parse_date)



        df_forcing.loc[:,'snow']=.8
        rol=df_forcing.Tair.rolling(5).mean()
        snowdetected=1
        for i in range(len(df_forcing)):

            if snowdetected==1:
```

(continues on next page)

(continued from previous page)

```
                if rol.iloc[i]>=5:
                    df_forcing.loc[df_forcing.iloc[i].name,'snow']=0
                    snowdetected=0
                else:
                    df_forcing.loc[df_forcing.iloc[i].name,'snow']=0.8
            else:
                df_forcing.loc[df_forcing.iloc[i].name,'snow']=0


            if (df_forcing.iloc[i].Tair<0) and (df_forcing.iloc[i].rain>0):
                df_forcing.loc[df_forcing.iloc[i].name,'snow']=0.8
                snowdetected=1


        all_sites_info =  pd.read_csv('site_info.csv')
        site_info=all_sites_info[all_sites_info['Site Id'] == name]
        df = pd.DataFrame(
            {'Site': [name],
            'Latitude': [site_info['Latitude (degrees)']],
            'Longitude': [site_info['Longitude (degrees)']]})



        path_runcontrol = Path('runs/run'+'/') / 'RunControl.nml'
        df_state_init = sp.init_supy(path_runcontrol)


        df_state_init,level=modify_attr(df_state_init, df, name)

        if level==1:
            attrs=[
                df_state_init.albmin_dectr,
                df_state_init.albmax_dectr
                ]
        elif level==0:
            attrs=[
                df_state_init.albmin_evetr,
                df_state_init.albmax_evetr
                ]
        elif level ==2:
            attrs=[
                df_state_init.albmin_grass,
                df_state_init.albmax_grass
                ]

        with open('outputs/albedo/'+name+'-attrs_albedo.pkl','wb') as f:
            pickle.dump(attrs, f)

        grid = df_state_init.index[0]
        df_forcing_run = sp.load_forcing_grid(path_runcontrol, grid)

        if multiple_run == 0:
            df_output, df_state_final = sp.run_supy(df_forcing_run, df_state_init,␣
↪save_state=False)

        if multiple_run == 1:
            error=20
```

(continues on next page)

(continued from previous page)

```
            for i in range(10):

                if (error <= 0.1):
                    break
                df_output, df_state_final = sp.run_supy(df_forcing_run, df_state_init,
↪save_state=False)
                final_state = df_state_final[df_state_init.columns.levels[0]].iloc[1]
                df_state_init.iloc[0] = final_state
                soilstore_before = df_state_final.soilstore_id.iloc[0]
                soilstore_after = df_state_final.soilstore_id.iloc[1]
                diff_soil = sum(abs(soilstore_after-soilstore_before))
                error = 100*diff_soil/soilstore_before.mean()
                print(error)

        df_output_2=df_output.SUEWS.loc[grid]
        df_output_2=df_output_2[df_output_2.index.year>=year]

        alb_model=pd.DataFrame(df_output_2.AlbBulk)
        a=alb_model.index.strftime('%j')
        alb_model['DOY']=[int(b) for b in a]


        Tair=df_forcing_run.Tair.resample('1d', closed='left',label='right').mean()
        Tair=pd.DataFrame(Tair)
        a=Tair.index.strftime('%j')
        Tair['DOY']=[int(b) for b in a]

        lai=df_output_2.LAI
        lai=lai.resample('1d', closed='left',label='right').mean()
        lai=pd.DataFrame(lai)
        a=lai.index.strftime('%j')
        lai['DOY']=[int(b) for b in a]

        snow=df_forcing.snow
        snow.index=df_forcing.datetime
        snow=snow.resample('1D').mean()
        snow=pd.DataFrame(snow)
        a=snow.index.strftime('%j')
        snow['DOY']=[int(b) for b in a]

        rain=df_forcing_run.rain
        rain=rain.resample('1d', closed='left',label='right').sum()
        rain=pd.DataFrame(rain)
        a=rain.index.strftime('%j')
        rain['DOY']=[int(b) for b in a]

        SMD=df_output_2.SMD
        SMD=SMD.resample('1d', closed='left',label='right').mean()
        SMD=pd.DataFrame(SMD)
        a=SMD.index.strftime('%j')
        SMD['DOY']=[int(b) for b in a]

        out={'obs':{'x':alb_avg_day.DOY,'y':alb_avg_day.alb},
            'model':{'x':alb_model.DOY,'y':alb_model.AlbBulk},
            'Tair':{'x':Tair.DOY,'y':Tair.Tair},
            'lai':{'x':lai.DOY,'y':lai.LAI},
            'snow':{'x':snow.DOY,'y':snow.snow},
```

(continues on next page)

(continued from previous page)

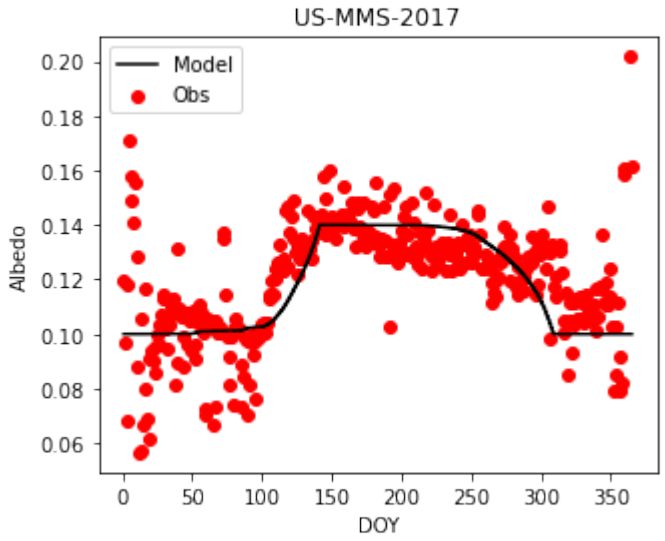
```
            'rain':{'x':rain.DOY,'y':rain.rain},
            'smd':{'x':SMD.DOY,'y':SMD.SMD},
            }
        with open('outputs/albedo/'+name+'-'+str(year)+'-all-albedo.pkl','wb') as f:
            pickle.dump(out, f)

        clear_output()
    fig,axs=plt.subplots(len(years),1,figsize=(5,4*len(years)))
    plt.subplots_adjust(hspace=0.3)
    counter=-1
    for year in years:
        counter += 1
        try:
            ax=axs[counter]
        except:
            ax=axs
        with open('outputs/albedo/'+name+'-'+str(year)+'-all-albedo.pkl','rb') as f:
            out=pickle.load(f)
        ax.scatter(out['obs']['x'],out['obs']['y'],color='r',label='Obs')
        ax.plot(out['model']['x'],out['model']['y'],color='k',label='Model')
        ax.legend()
        ax.set_title(name+'-'+str(year))
        ax.set_xlabel('DOY')
        ax.set_ylabel('Albedo')
```

Plotting for all the sites

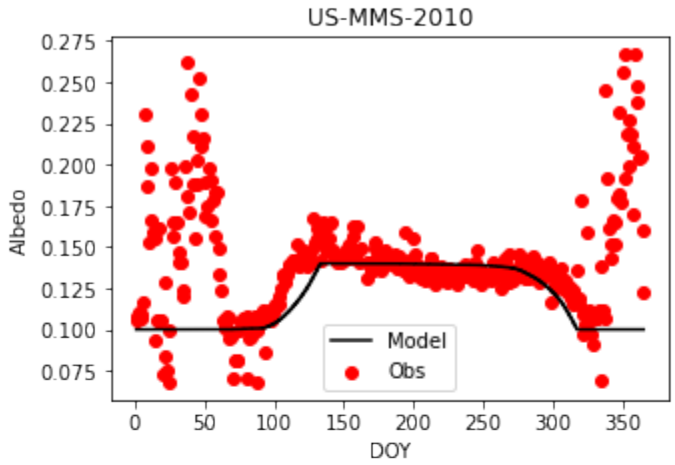
DecTr

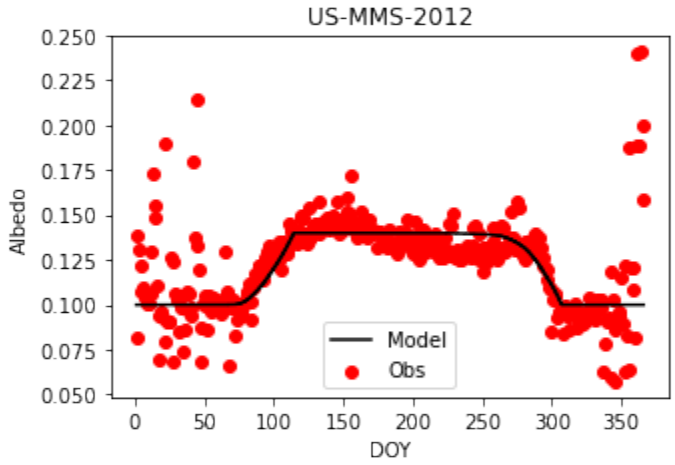
US-MMS

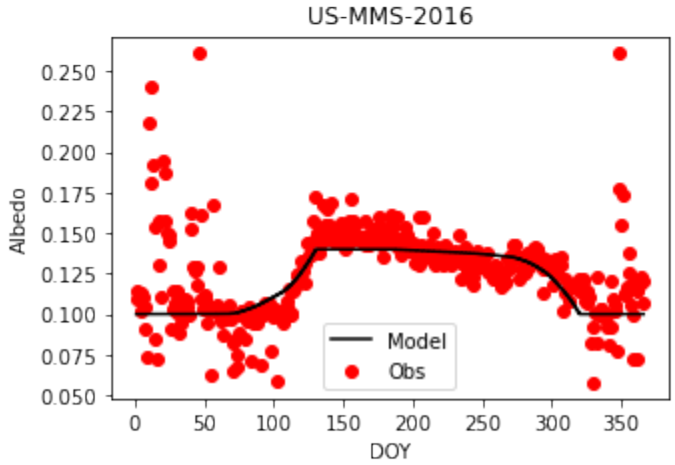
[28]:
```
name='US-MMS'
years=[2017]
read_plot(years,name)
```



[29]:
```
name='US-MMS'
years=[2010,2012,2016]
read_plot(years,name)
```


US-MMS-2010
Albedo
DOY
Model
Obs

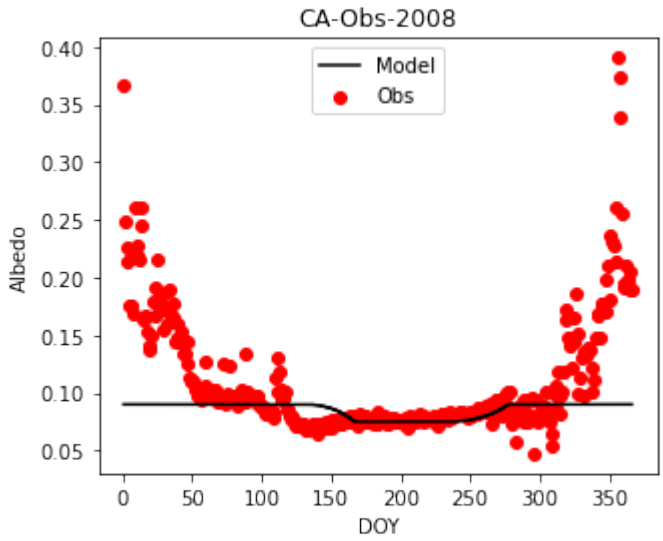

US-MMS-2012
Albedo
DOY
Model
Obs


US-MMS-2016
Albedo
DOY
Model
Obs

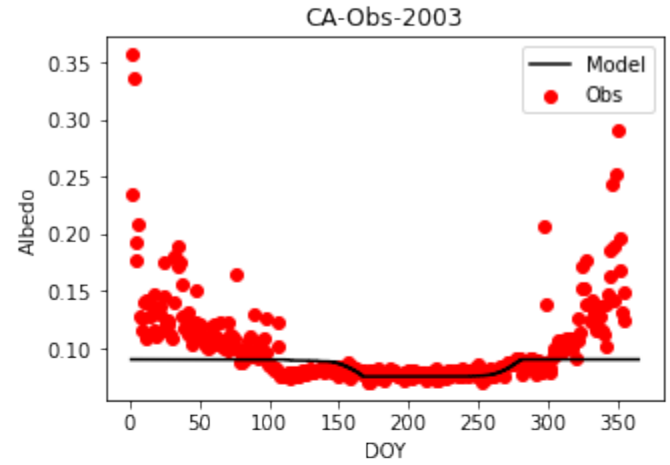
EveTr

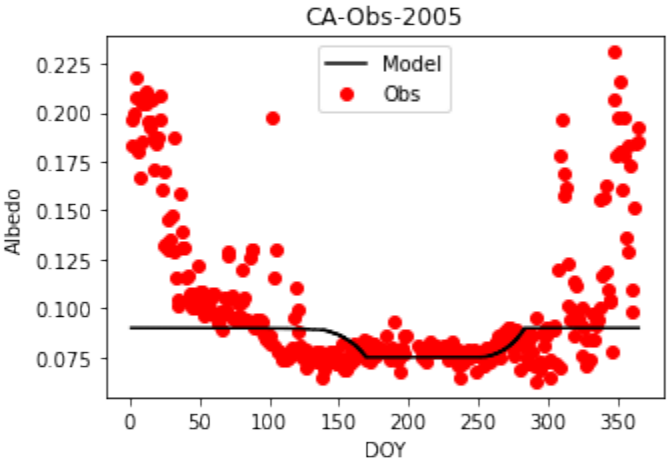
CA-Obs

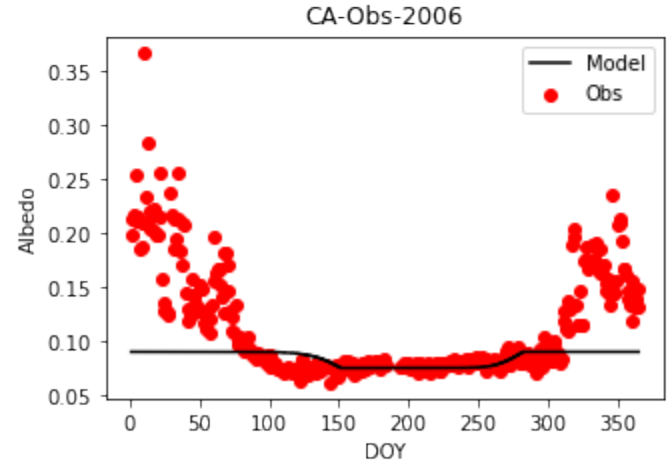
[30]:
```
name='CA-Obs'
years=[2008]
read_plot(years,name)
```



[31]:
```
name='CA-Obs'
years=[2003,2005,2006]
read_plot(years,name)
```


CA-Obs-2003
Model
Obs
Albedo
DOY
0.35
0.30
0.25
0.20
0.15
0.10
0
50
100
150
200
250
300
350

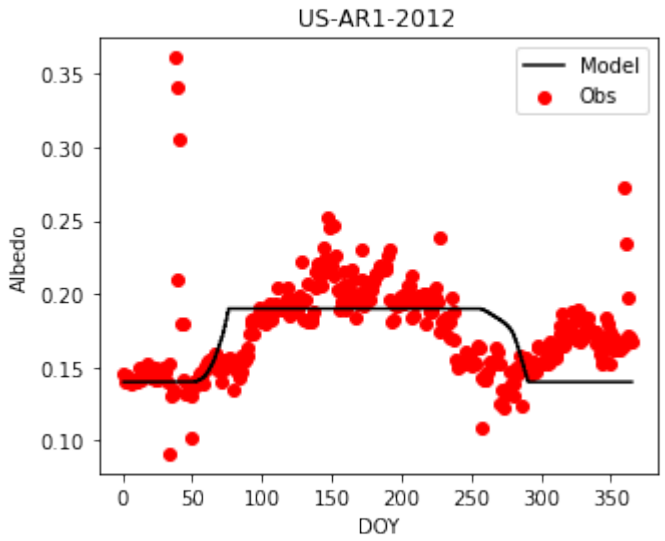

CA-Obs-2005
Model
Obs
Albedo
DOY
0.225
0.200
0.175
0.150
0.125
0.100
0.075
0
50
100
150
200
250
300
350


CA-Obs-2006
Model
Obs
Albedo
DOY
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0
50
100
150
200
250
300
350

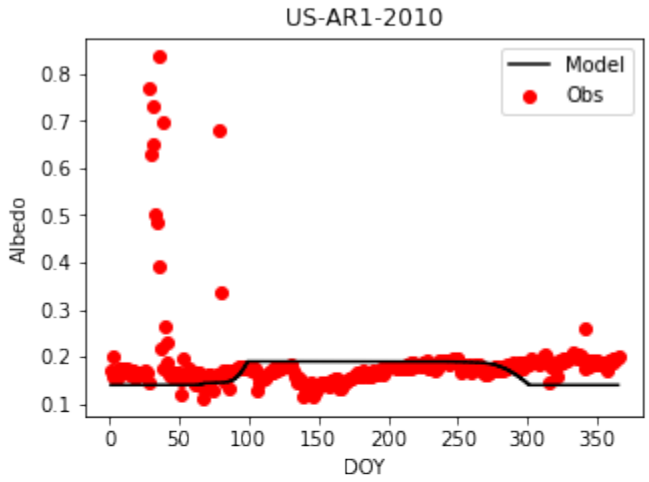
Grass

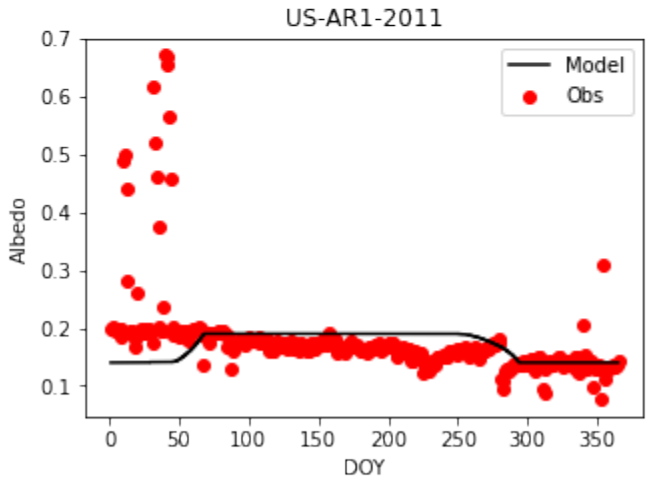
US-AR1

[32]:
```
name='US-AR1'
years=[2012]
read_plot(years,name)
```



[33]:
```
name='US-AR1'
years=[2010,2011]
read_plot(years,name)
```





End of `Parameters/tutorials/param2-albedo.ipynb`

The following section was generated from source/Parameters/tutorials/param3-roughness.ipynb

10.3.3 Roughness parameters

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from pathlib import Path
from platypus.core import Problem
from platypus.types import Real,random
from platypus.algorithms import NSGAIII
import warnings
warnings.filterwarnings('ignore')
```

This is a custom package specifically designed for this analysis. It contains various functions for reading and computing and plotting.

[2]:
```
from z0_util import cal_vap_sat, cal_dens_dry, cal_dens_vap, cal_cpa, cal_dens_air,␣
↪cal_Lob
```

Function to calculate Neutral condition

[13]:
```
def cal_neutral(df_val,z_meas,h_sfc):

    # calculate Obukhov length
    ser_Lob = df_val.apply(
        lambda ser: cal_Lob(ser.H, ser.USTAR, ser.TA, ser.RH, ser.PA * 10), axis=1)

    # zero-plane displacement: estimated using rule f thumb `d=0.7*h_sfc`

    z_d = 0.7 * h_sfc

    if z_d >= z_meas:
        print(
            'vegetation height is greater than measuring height. Please fix this␣
↪before continuing'
        )

    # calculate stability scale
    ser_zL = (z_meas - z_d) / ser_Lob

    # determine periods under quasi-neutral conditions
    limit_neutral = 0.01
    ind_neutral = ser_zL.between(-limit_neutral, limit_neutral)


    ind_neutral=ind_neutral[ind_neutral]
    df_sel = df_val.loc[ind_neutral.index, ['WS', 'USTAR']].dropna()
    ser_ustar = df_sel.USTAR
    ser_ws = df_sel.WS


    return ser_ws,ser_ustar
```

Function to calculate z0 and d using MO optimization

[14]:
```
def optimize_MO(df_val,z_meas,h_sfc):


    ser_ws,ser_ustar=cal_neutral(df_val,z_meas,h_sfc)

    def func_uz(params):
        z0=params[0]
        d=params[1]
        z = z_meas
        k = 0.4
        uz = (ser_ustar / k) * np.log((z - d) / z0)

        o1=abs(1-np.std(uz)/np.std(ser_ws))
        o2=np.mean(abs(uz-ser_ws))/(np.mean(ser_ws))

        return [o1,o2],[uz.min(),d-z0]

    problem = Problem(2,2,2)
    problem.types[0] = Real(0, 10)
    problem.types[1] = Real(0, h_sfc)


    problem.constraints[0] = ">=0"
    problem.constraints[1] = ">=0"

    problem.function = func_uz
    random.seed(12345)
    algorithm=NSGAIII(problem, divisions_outer=50)
    algorithm.run(30000)

    z0s=[]
    ds=[]
    os1=[]
    os2=[]
    for s in algorithm.result:
        z0s.append(s.variables[0])
        ds.append(s.variables[1])
        os1.append(s.objectives[0])
        os2.append(s.objectives[1])

    idx=os2.index(min(os2, key=lambda x:abs(x-np.mean(os2))))
    z0=z0s[idx]
    d=ds[idx]

    return z0,d,ser_ws,ser_ustar
```

Loading data, cleaning and getting ready for optimization

[15]:
```
name_of_site='US-MMS'
years=[2010,2012,2016]


df_attr=pd.read_csv('all_attrs.csv')
z_meas=df_attr[df_attr.site==name_of_site].meas.values[0]
h_sfc=df_zmeas=df_attr[df_attr.site==name_of_site].height.values[0]
folder='data/data_csv_zip_clean_roughness/'
site_file = folder+'/'+name_of_site + '_clean.csv.gz'
df_data = pd.read_csv(site_file, index_col='time', parse_dates=['time'])
# Rain
bb=pd.DataFrame(~np.isin(df_data.index.date,df_data[df_data.P!=0].index.date))
bb.index=df_data.index
df_data=df_data[bb.values]


df_data=df_data[(df_data['WS']!=0)]

df_years=df_data.loc[f'{years[0]}']
for i in years[1:]:
    df_years=df_years.append(df_data.loc[f'{i}'])

df_val = df_years.loc[:, ['H', 'USTAR', 'TA', 'RH', 'PA', 'WS']].dropna()
df_val.head()
```

[15]:

time	H	USTAR	TA	RH	PA	WS
2010-01-01 00:00:00	21.873	0.739	-8.08	70.503	98.836	3.695
2010-01-01 01:00:00	41.819	0.855	-9.17	72.757	98.880	3.928
2010-01-01 02:00:00	-6.078	0.699	-9.63	72.611	98.910	3.088
2010-01-01 03:00:00	-16.788	0.581	-10.03	73.868	98.970	3.623
2010-01-01 04:00:00	5.006	0.562	-10.36	74.242	99.030	3.474

Calculating z0 and d

[16]:
```
z0,d,ser_ws,ser_ustar=optimize_MO(df_val,z_meas,h_sfc)
```

Calculating model wind speed using logarithmic law

[17]:
```
def uz(z0,d,ser_ustar,z_meas):
    z = z_meas
    k = 0.4
    uz = (ser_ustar / k) * np.log((z - d) / z0)
    return uz

ws_model=uz(z0,d,ser_ustar,z_meas)
```

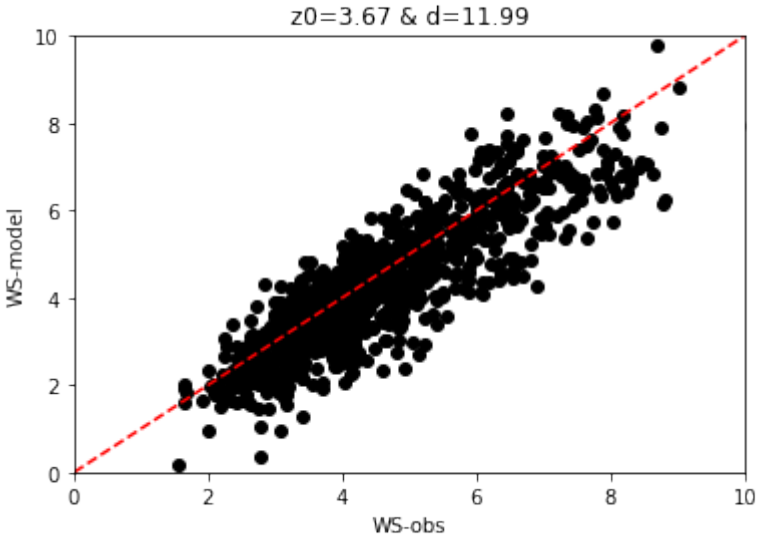
[18]:
```
plt.scatter(ser_ws,ws_model,color='k')
plt.xlabel('WS-obs')
plt.ylabel('WS-model')
plt.title(f'z0={np.round(z0,2)} & d={np.round(d,2)}')
plt.plot([0,10],[0,10],color='r',linestyle='--')
```

(continues on next page)

(continued from previous page)

```
plt.ylim([0,10])
plt.xlim([0,10])
```

[18]:
```
(0.0, 10.0)
```



End of `Parameters/tutorials/param3-roughness.ipynb`

The following section was generated from source/Parameters/tutorials/param4-conductance.ipynb

10.3.4 Surface conductance parameters

[7]:
```
from lmfit import Model, Parameters, Parameter
import numpy as np
import pandas as pd
from atmosp import calculate as ac
import numpy as np
from scipy.optimize import minimize
from pathlib import Path
import supy as sp
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import seaborn as sns
from platypus.core import *
from platypus.types import *
from platypus.algorithms import *
import random
import pickle
import os
from shutil import copyfile
import warnings
warnings.filterwarnings('ignore')
```

This is a custom package specifically designed for this analysis. It contains various functions for reading and computing and plotting.

[4]:
```
from gs_util import read_forcing,modify_attr,cal_gs_obs,IQR_compare,obs_sim,cal_gs_
↪mod,gs_plot_test,modify_attr_2,func_parse_date
```

Preparing the data (obs and model)

[5]:
```
name='US-MMS'
year=2017
df_forcing= read_forcing(name,year)
```

[8]:
```
path_runcontrol = Path('runs/run'+'/') / 'RunControl.nml'
df_state_init = sp.init_supy(path_runcontrol)
df_state_init,level=modify_attr(df_state_init,name)
df_state_init.loc[:,'soilstore_id']=[50,50,50,50,50,50,0]
grid = df_state_init.index[0]
df_forcing_run = sp.load_forcing_grid(path_runcontrol, grid)
```

```
2020-03-26 10:09:25,798 -- SuPy -- INFO -- All cache cleared.
2020-03-26 10:09:26,642 -- SuPy -- INFO -- All cache cleared.
```

Spin up to get soil moisture

[9]:
```
error=10
for i in range(10):

    if (error <= 0.1):
        break
    df_output, df_state_final = sp.run_supy(df_forcing_run, df_state_init, save_
↪state=False)
    final_state = df_state_final[df_state_init.columns.levels[0]].iloc[1]
    df_state_init.iloc[0] = final_state
    soilstore_before = df_state_final.soilstore_id.iloc[0]
    soilstore_after = df_state_final.soilstore_id.iloc[1]
    diff_soil = sum(abs(soilstore_after-soilstore_before))
    error = 100*diff_soil/soilstore_before.mean()
    print(error)
```

```
2020-03-26 10:09:34,245 -- SuPy -- INFO -- ====================
2020-03-26 10:09:34,246 -- SuPy -- INFO -- Simulation period:
2020-03-26 10:09:34,248 -- SuPy -- INFO --   Start: 2016-12-31 23:05:00
2020-03-26 10:09:34,250 -- SuPy -- INFO --   End: 2017-12-31 23:00:00
2020-03-26 10:09:34,251 -- SuPy -- INFO --
2020-03-26 10:09:34,253 -- SuPy -- INFO -- No. of grids: 1
2020-03-26 10:09:34,254 -- SuPy -- INFO -- SuPy is running in serial mode
2020-03-26 10:09:48,656 -- SuPy -- INFO -- Execution time: 14.4 s
2020-03-26 10:09:48,656 -- SuPy -- INFO -- ====================

933.0266258519457
2020-03-26 10:09:49,106 -- SuPy -- INFO -- ====================
2020-03-26 10:09:49,107 -- SuPy -- INFO -- Simulation period:
2020-03-26 10:09:49,107 -- SuPy -- INFO --   Start: 2016-12-31 23:05:00
2020-03-26 10:09:49,108 -- SuPy -- INFO --   End: 2017-12-31 23:00:00
2020-03-26 10:09:49,109 -- SuPy -- INFO --
2020-03-26 10:09:49,111 -- SuPy -- INFO -- No. of grids: 1
2020-03-26 10:09:49,112 -- SuPy -- INFO -- SuPy is running in serial mode
2020-03-26 10:10:02,928 -- SuPy -- INFO -- Execution time: 13.8 s
2020-03-26 10:10:02,929 -- SuPy -- INFO -- ====================

0.0
```

Preparation of the data for model optimization

[10]:
```
df=df_output.SUEWS.loc[grid,:]
df=df.resample('1h',closed='left',label='right').mean()
```

[11]:
```
df_forcing.xsmd=df.SMD
df_forcing.lai=df.LAI
df_forcing = df_forcing[df_forcing.qe > 0]
df_forcing = df_forcing[df_forcing.qh > 0]
df_forcing = df_forcing[df_forcing.kdown > 5]
df_forcing = df_forcing[df_forcing.Tair > -20]
df_forcing.pres *= 1000
df_forcing.Tair += 273.15
gs_obs = cal_gs_obs(df_forcing.qh, df_forcing.qe, df_forcing.Tair,
                    df_forcing.RH, df_forcing.pres,df.RA)
```

(continues on next page)

(continued from previous page)

```
df_forcing=df_forcing[gs_obs>0]
gs_obs=gs_obs[gs_obs>0]
df_forcing=df_forcing.replace(-999,np.nan)
```

[12]:
```
g_max=np.percentile(gs_obs,99)
s1=5.56
```

[13]:
```
print('Initial g_max is {}'.format(g_max))
```

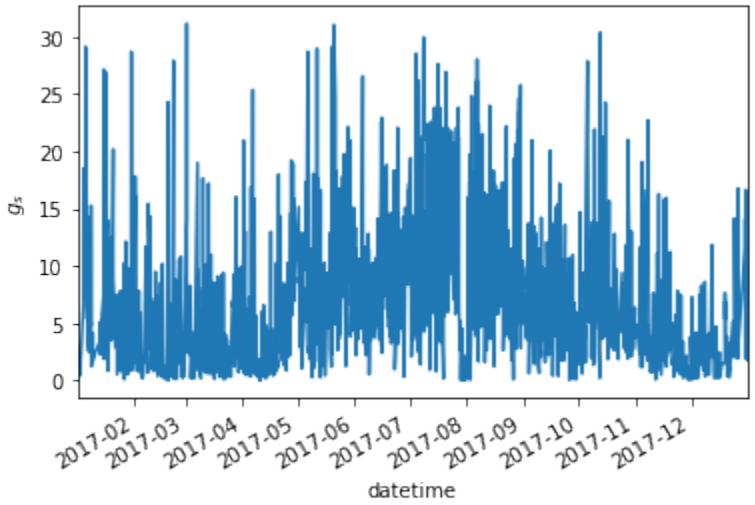
```
Initial g_max is 33.023283412991034
```

[14]:
```
df_forcing=df_forcing[gs_obs<g_max]
lai_max=df_state_init.laimax.loc[grid,:][1]
gs_obs=gs_obs[gs_obs<g_max]
```

Distribution of observed g_s

[16]:
```
gs_obs.plot()
plt.ylabel('$g_s$')
```

[16]:
```
Text(0, 0.5, '$g_s$')
```



[17]:
```
print('lai_max is {}'.format(lai_max))
```

```
lai_max is 5.0
```

Splitting the data to test and train

[19]:
```
df_forcing_train, df_forcing_test, gs_train, gs_test = train_test_split(df_forcing,␣
↪gs_obs, test_size=0.6, random_state=42)
```

[20]:
```
kd=df_forcing_train.kdown
ta=df_forcing_train.Tair
rh=df_forcing_train.RH
pa=df_forcing_train.pres
smd=df_forcing_train.xsmd
lai=df_forcing_train.lai
```

Optimization

More info in here

Function to optimize

[22]:
```
def fun_to_opts(G):
    [g1,g2, g3, g4, g5, g6]=[G[0],G[1],G[2],G[3],G[4],G[5]]
    gs_model,g_lai,g_kd,g_dq,g_ta,g_smd,g_z=cal_gs_mod(kd, ta, rh, pa, smd, lai,
                          [g1, g2, g3, g4, g5, g6],
                          g_max, lai_max, s1)
    gs_obs=gs_train
    o1=abs(1-np.std(gs_model)/np.std(gs_obs)) # normilized std difference
    o2=np.mean(abs(gs_model-gs_obs))/(np.mean(gs_obs))
    return [o1,o2],[gs_model.min()]
```

Problem definition and run

[27]:
```
problem = Problem(6,2,1)
problem.types[0] = Real(.09, .5)
problem.types[1] = Real(100, 500)
problem.types[2] = Real(0, 1)
problem.types[3] = Real(0.4, 1)
problem.types[4] = Real(25, 55)
problem.types[5] = Real(0.02, 0.03)

problem.constraints[0] = ">=0"

problem.function = fun_to_opts
random.seed(12345)
algorithm=CMAES(problem, epsilons=[0.005])
algorithm.run(3000)
```

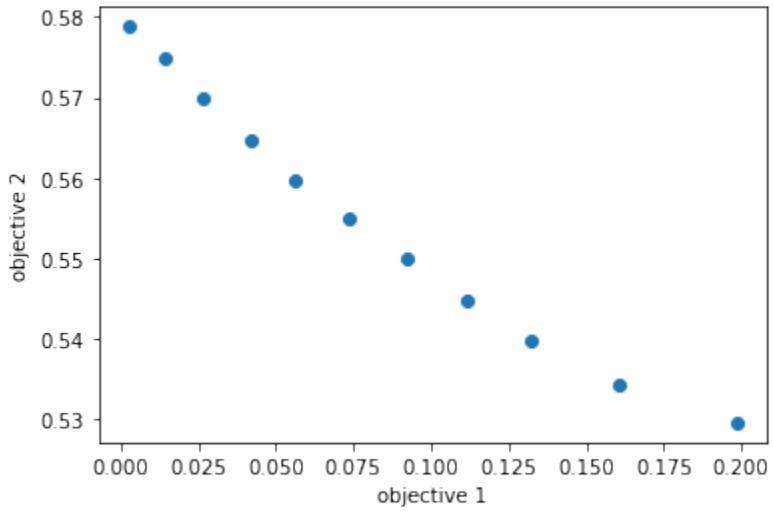
Solutions

[28]:
```
print( " Obj1\t Obj2")

for solution in algorithm.result[:10]:
    print ("%0.3f\t%0.3f" % tuple(solution.objectives))
```

```
 Obj1    Obj2
0.073   0.555
0.112   0.545
0.161   0.534
0.056   0.560
0.003   0.579
0.092   0.550
0.199   0.530
0.133   0.540
0.026   0.570
0.014   0.575
```

[29]:
```
f, ax = plt.subplots(1, 1)
plt.scatter([s.objectives[0] for s in algorithm.result],
            [s.objectives[1] for s in algorithm.result])
plt.xlabel('objective 1')
plt.ylabel('objective 2')
```

[29]:
```
Text(0, 0.5, 'objective 2')
```



[30]:
```
all_std=[s.objectives[0] for s in algorithm.result]
all_MAE=[s.objectives[1] for s in algorithm.result]
all_std=np.array(all_std)
all_MAE=np.array(all_MAE)
```

Choosing between : the median of two objectives, where obj1 is max or where obj2 is max

[31]:
```
method='median' # 'obj1' or 'obj2' or 'median'
colors= ['b','g','r','y']

if method == 'median':
    idx_med=np.where(all_MAE==all_MAE[(all_std<=np.median(all_std))].min())[0][0]
elif method == 'obj1':
    idx_med=np.where(all_MAE==all_MAE[(all_std>=np.min(all_std))].max())[0][0]
elif method == 'obj2':
    idx_med=np.where(all_MAE==all_MAE[(all_std<=np.max(all_std))].min())[0][0]
print(all_std[idx_med])
print(all_MAE[idx_med])
```

```
0.07329333444496633
0.5549996145597578
```

[32]:
```
[g1,g2,g3,g4,g5,g6] = algorithm.result[idx_med].variables
```

Saving the solution

[33]:
```
with open('outputs/g1-g6/'+name+'-g1-g6.pkl','wb') as f:
    pickle.dump([g1,g2,g3,g4,g5,g6], f)
```

[34]:
```
with open('outputs/g1-g6/'+name+'-g1-g6.pkl','rb') as f:
    [g1,g2,g3,g4,g5,g6]=pickle.load(f)
```

[35]:
```
pd.DataFrame([np.round([g1,g2,g3,g4,g5,g6],3)],columns=['g1','g2','g3','g4','g5','g6
↪'],index=[name])
```

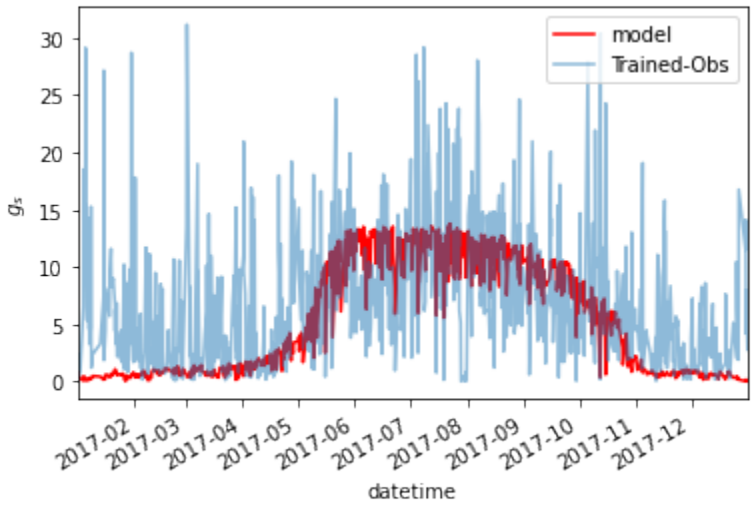
[35]:

	g1	g2	g3	g4	g5	g6
US-MMS	0.431	104.34	0.634	0.683	35.25	0.03

Let's see how the model and observation compare for the training data set:

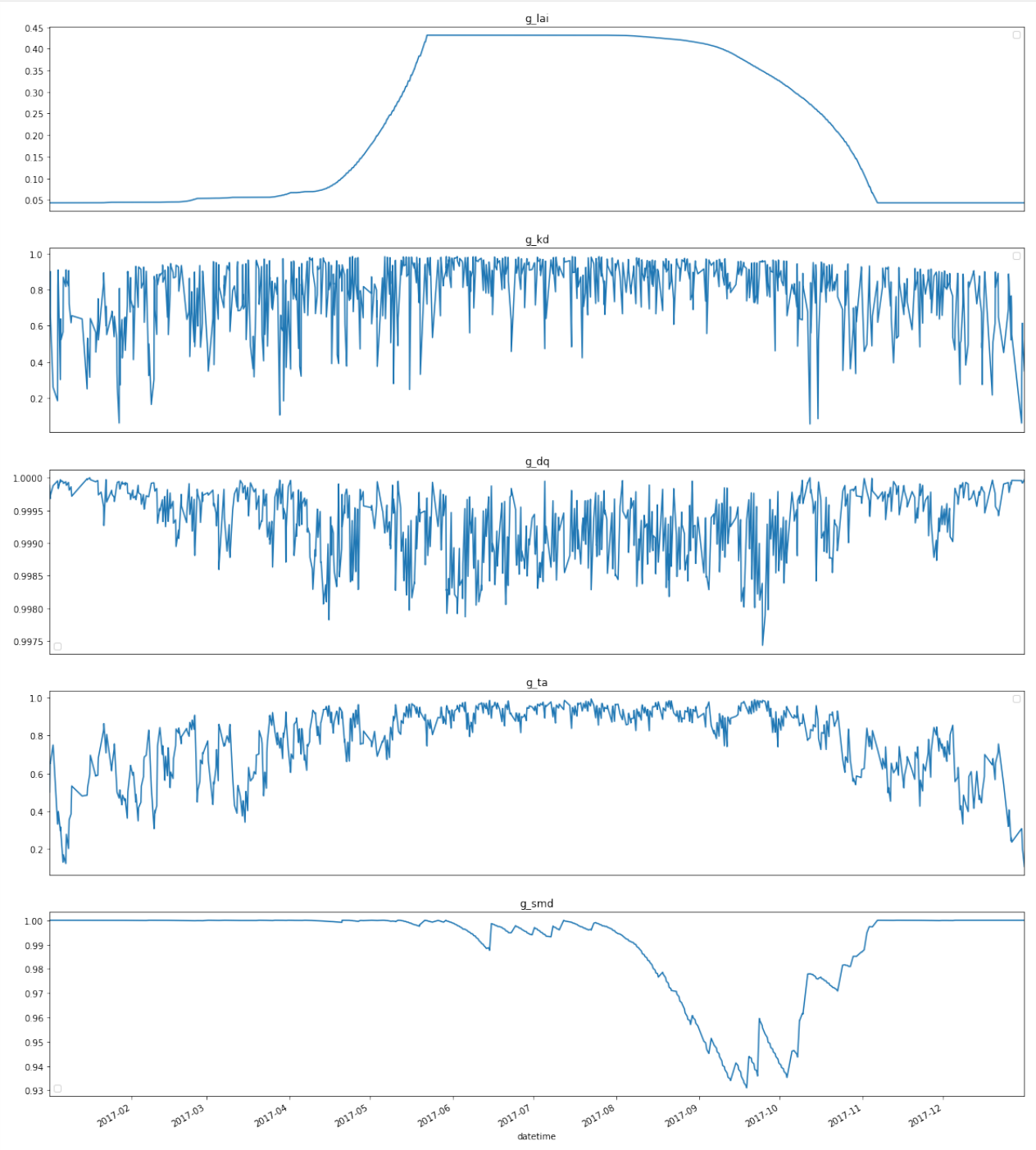
[38]:
```
gs_model,g_lai,g_kd,g_dq,g_ta,g_smd,g_max=cal_gs_mod(kd, ta, rh, pa, smd, lai,
                          [g1, g2, g3, g4, g5, g6],
                          g_max, lai_max, s1)

gs_model.plot(color='r',label='model')
gs_train.plot(alpha=0.5,label='Trained-Obs')
plt.legend()
plt.ylabel('$g_s$')
```

[38]:
```
Text(0, 0.5, '$g_s$')
```



Let's look at each individual g term:

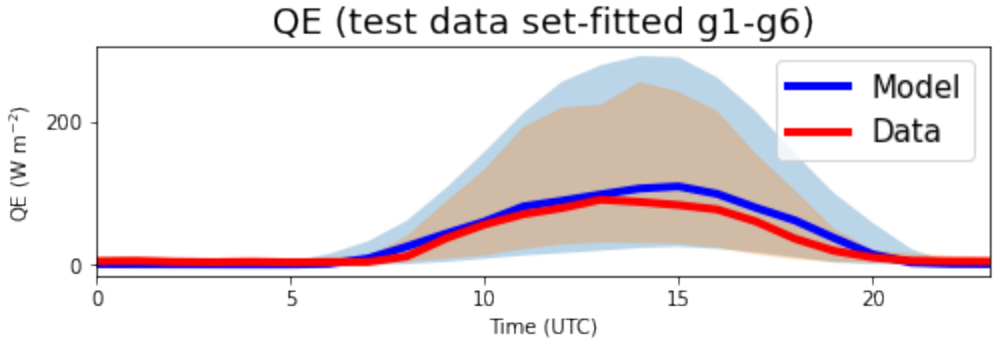
[39]:
```
g_dq=pd.DataFrame(g_dq,index=g_lai.index)
fig,axs=plt.subplots(5,1,figsize=(20,25))
a={'0':g_lai,'1':g_kd,'2':g_dq,'3':g_ta,'4':g_smd}
b={'0':'g_lai','1':'g_kd','2':'g_dq','3':'g_ta','4':'g_smd'}
for i in range(0,5):
    ax=axs[i]
    a[str(i)].plot(ax=ax)
    ax.set_title(b[str(i)])
    ax.legend('')
    if i!=4:
        ax.set_xticks([''])
        ax.set_xlabel('')
```

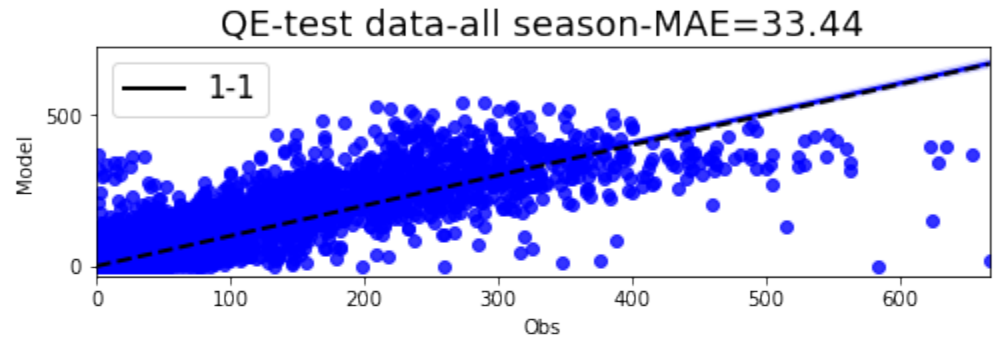

g_lai
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
g_kd
1.0
0.8
0.6
0.4
0.2
g_dq
1.0000
0.9995
0.9990
0.9985
0.9980
0.9975
g_ta
1.0
0.8
0.6
0.4
0.2
g_smd
1.00
0.99
0.98
0.97
0.96
0.95
0.94
0.93
2017-02
2017-03
2017-04
2017-05
2017-06
2017-07
2017-08
2017-09
2017-10
2017-11
2017-12
datetime

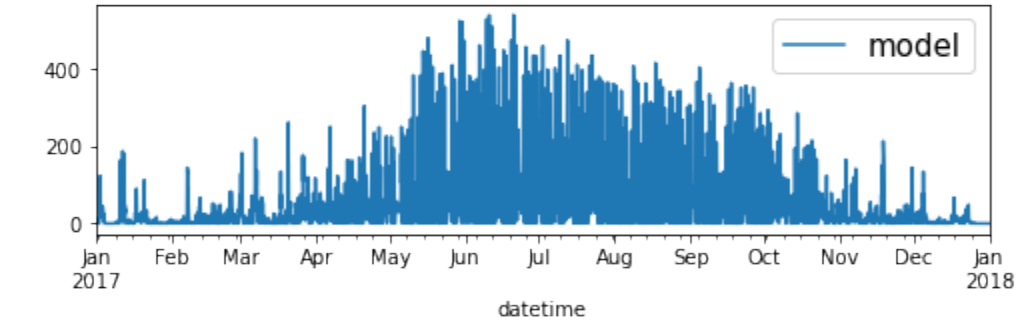
Running Supy with new g1-g6

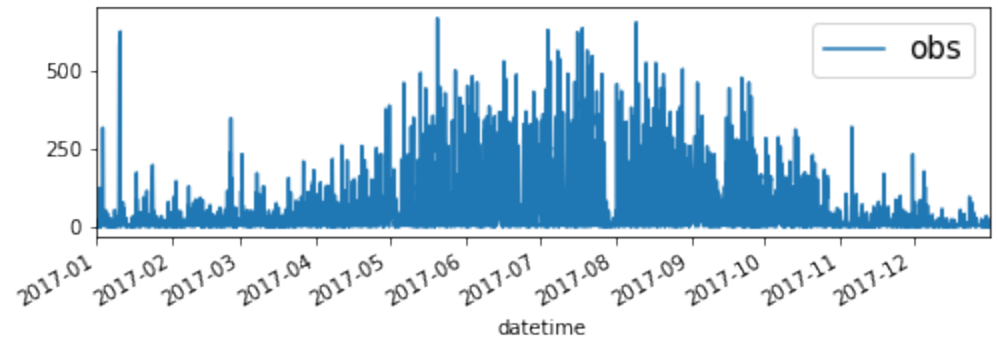
[40]:
```
alpha=2.6 # need to be tuned iteratively
name='US-MMS'
year=year
gs_plot_test(g1,g2,g3,g4,g5,g6,g_max,s1,name,year,alpha)
```

```
2020-03-26 10:38:44,539 -- SuPy -- INFO -- All cache cleared.
2020-03-26 10:38:45,412 -- SuPy -- INFO -- All cache cleared.
2020-03-26 10:38:47,776 -- SuPy -- INFO -- ====================
2020-03-26 10:38:47,776 -- SuPy -- INFO -- Simulation period:
2020-03-26 10:38:47,777 -- SuPy -- INFO --   Start: 2016-12-31 23:05:00
2020-03-26 10:38:47,778 -- SuPy -- INFO --   End: 2017-12-31 23:00:00
2020-03-26 10:38:47,779 -- SuPy -- INFO --
2020-03-26 10:38:47,780 -- SuPy -- INFO -- No. of grids: 1
2020-03-26 10:38:47,782 -- SuPy -- INFO -- SuPy is running in serial mode
2020-03-26 10:39:08,180 -- SuPy -- INFO -- Execution time: 20.4 s
2020-03-26 10:39:08,181 -- SuPy -- INFO -- ====================
```


QE (test data set-fitted g1-g6)
Model
Data
QE (W m^{-2})
200
0
0
5
10
15
20
Time (UTC)


QE-test data-all season-MAE=33.44
1-1
Model
500
0
0
100
200
300
400
500
600
Obs


model
400
200
0
Jan
2017
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
Jan
2018
datetime


obs
500
250
0
2017-01
2017-02
2017-03
2017-04
2017-05
2017-06
2017-07
2017-08
2017-09
2017-10
2017-11
2017-12
datetime

[41]:
```
g1=g1*alpha
g_max=g1*g_max
g1=1
```

[42]:
```
g_max
```

[42]:
```
37.03471400427182
```

Creating the table for all sites here (if more than one site is tuned)

[43]:
```
sites=['US-MMS']
g1_g6_all=pd.DataFrame(columns=['g1','g2','g3','g4','g5','g6'])

for s in sites:
    with open('outputs/g1-g6/'+s+'-g1-g6.pkl','rb') as f:
        g1_g6_all.loc[s,:]=pickle.load(f)
g1_g6_all
```

[43]:

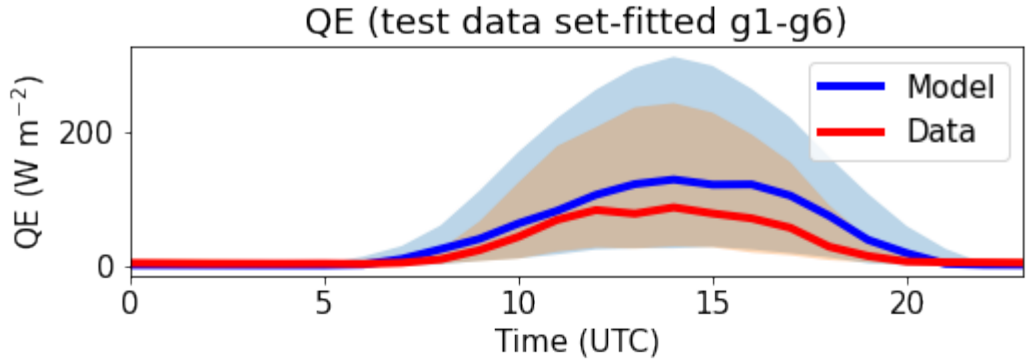
	g1	g2	g3	g4	g5	g6
US-MMS	0.431336	104.34	0.633861	0.682953	35.25	0.0298504

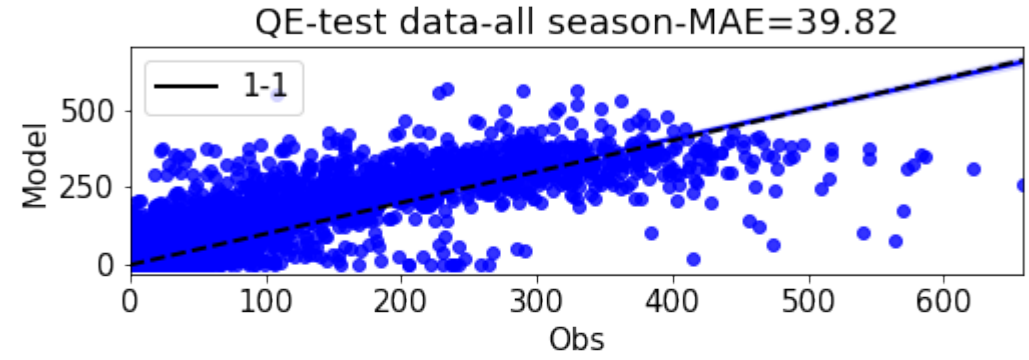
To test

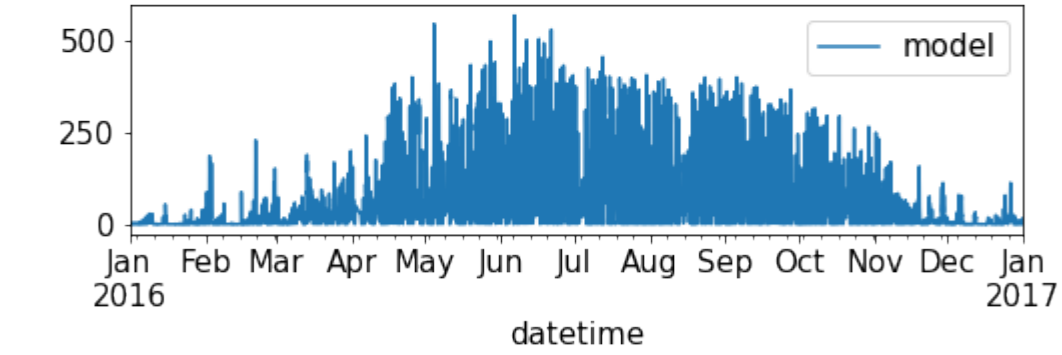
US-MMS-2016

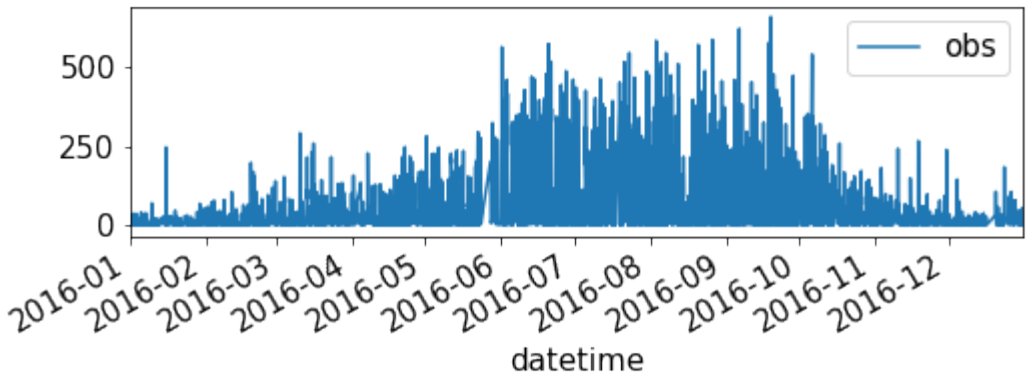
[44]:
```
g1,g2,g3,g4,g5,g6=g1_g6_all.loc['US-MMS',:].values
g_max=g_max
g1=1
s1=5.56


name='US-MMS'
year=2016
df_forcing= read_forcing(name,year)
gs_plot_test(g1,g2,g3,g4,g5,g6,g_max,s1,name,year)
```

```
2020-03-26 10:41:58,297 -- SuPy -- INFO -- All cache cleared.
2020-03-26 10:41:59,122 -- SuPy -- INFO -- All cache cleared.
2020-03-26 10:42:01,787 -- SuPy -- INFO -- ====================
2020-03-26 10:42:01,789 -- SuPy -- INFO -- Simulation period:
2020-03-26 10:42:01,791 -- SuPy -- INFO --   Start: 2015-12-31 23:05:00
2020-03-26 10:42:01,801 -- SuPy -- INFO --   End: 2016-12-31 23:00:00
2020-03-26 10:42:01,802 -- SuPy -- INFO --
2020-03-26 10:42:01,807 -- SuPy -- INFO -- No. of grids: 1
2020-03-26 10:42:01,810 -- SuPy -- INFO -- SuPy is running in serial mode
2020-03-26 10:42:24,960 -- SuPy -- INFO -- Execution time: 23.2 s
2020-03-26 10:42:24,961 -- SuPy -- INFO -- ====================
```


QE (test data set-fitted g1-g6)
QE (W m^{-2})
200
0
0
5
10
15
20
Time (UTC)
Model
Data


QE-test data-all season-MAE=39.82
Model
500
250
0
0
100
200
300
400
500
600
Obs
1-1

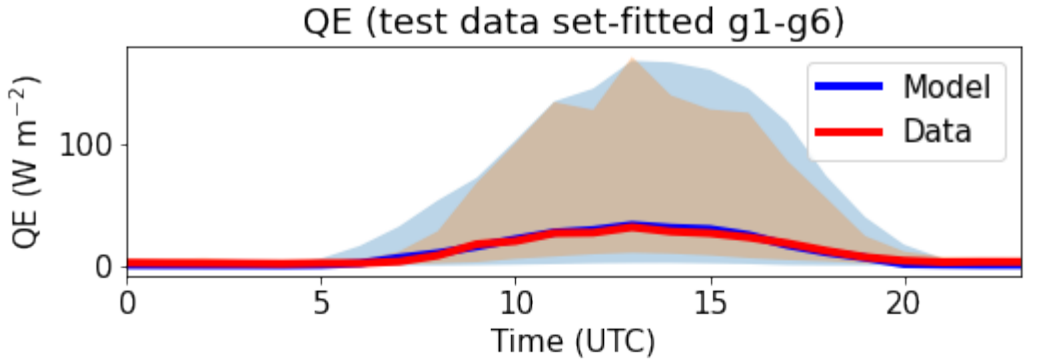

500
250
0
Jan
2016
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
Jan
2017
datetime
model

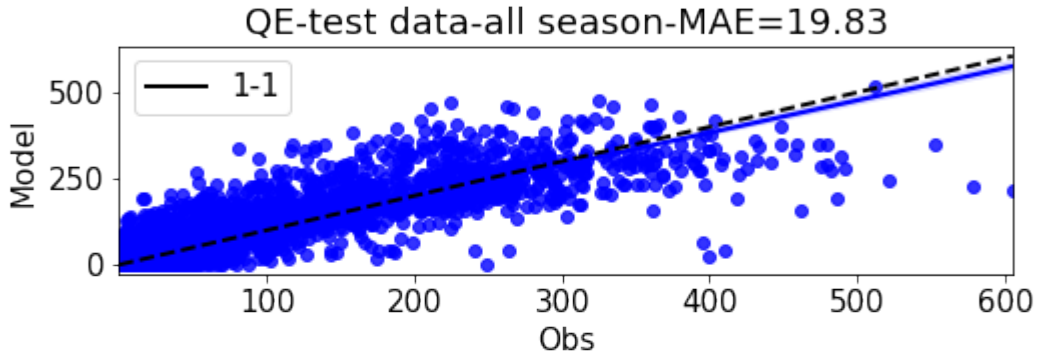

500
250
0
2016-01
2016-02
2016-03
2016-04
2016-05
2016-06
2016-07
2016-08
2016-09
2016-10
2016-11
2016-12
datetime
obs

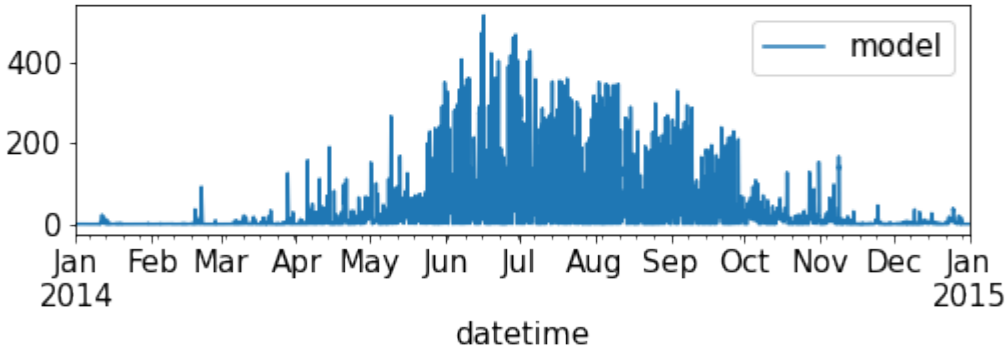
UMB-2014

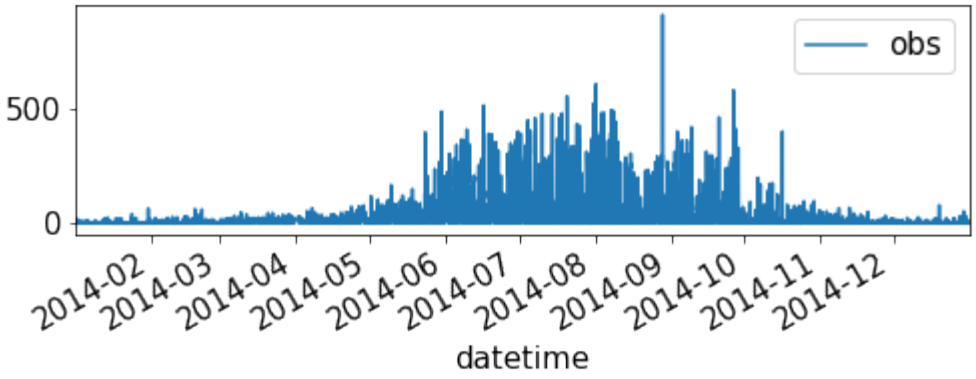
[46]:
```
g1,g2,g3,g4,g5,g6=g1_g6_all.loc['US-MMS',:].values
g_max=g_max
g1=1
s1=5.56

name='US-UMB'
year=2014
df_forcing= read_forcing(name,year)
gs_plot_test(g1,g2,g3,g4,g5,g6,g_max,s1,name,year)
```

```
2020-03-26 10:43:28,350 -- SuPy -- INFO -- All cache cleared.
2020-03-26 10:43:29,145 -- SuPy -- INFO -- All cache cleared.
2020-03-26 10:43:31,505 -- SuPy -- INFO -- ====================
2020-03-26 10:43:31,506 -- SuPy -- INFO -- Simulation period:
2020-03-26 10:43:31,506 -- SuPy -- INFO --   Start: 2013-12-31 23:05:00
2020-03-26 10:43:31,507 -- SuPy -- INFO --   End: 2014-12-31 23:00:00
2020-03-26 10:43:31,509 -- SuPy -- INFO --
2020-03-26 10:43:31,511 -- SuPy -- INFO -- No. of grids: 1
2020-03-26 10:43:31,512 -- SuPy -- INFO -- SuPy is running in serial mode
2020-03-26 10:43:45,433 -- SuPy -- INFO -- Execution time: 13.9 s
2020-03-26 10:43:45,434 -- SuPy -- INFO -- ====================
```


QE (test data set-fitted g1-g6)
QE (W m^{-2})
100
0
0
5
10
15
20
Time (UTC)
Model
Data


QE-test data-all season-MAE=19.83
Model
500
250
0
100
200
300
400
500
600
Obs
1-1

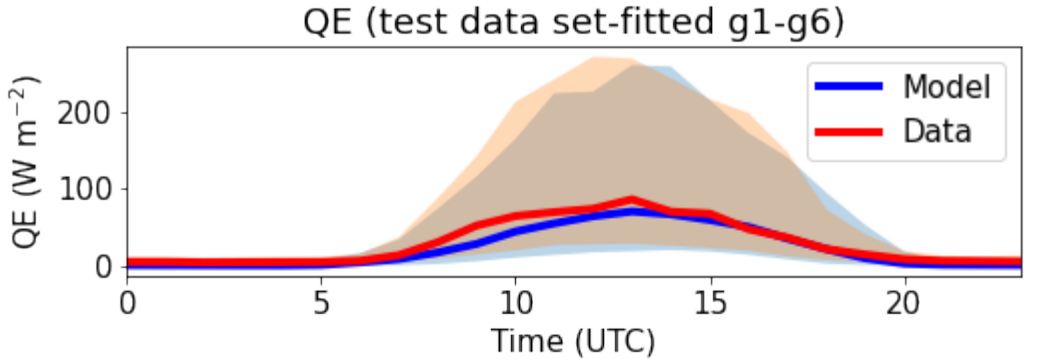

400
200
0
Jan
2014
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
Jan
2015
datetime
model

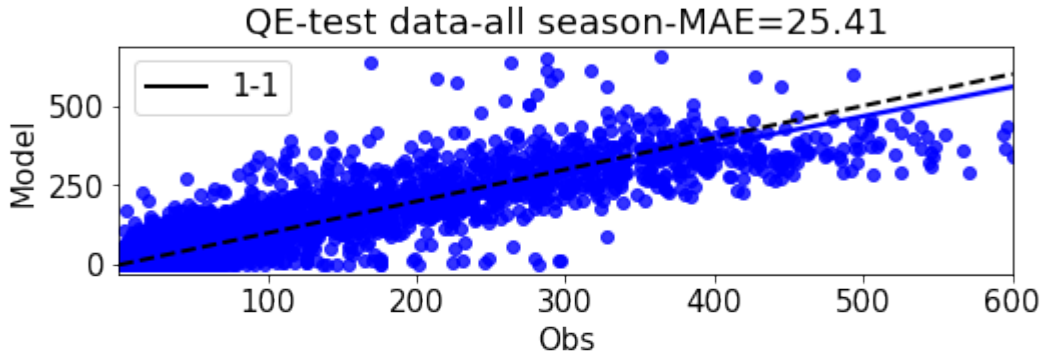

500
0
2014-02
2014-03
2014-04
2014-05
2014-06
2014-07
2014-08
2014-09
2014-10
2014-11
2014-12
datetime
obs

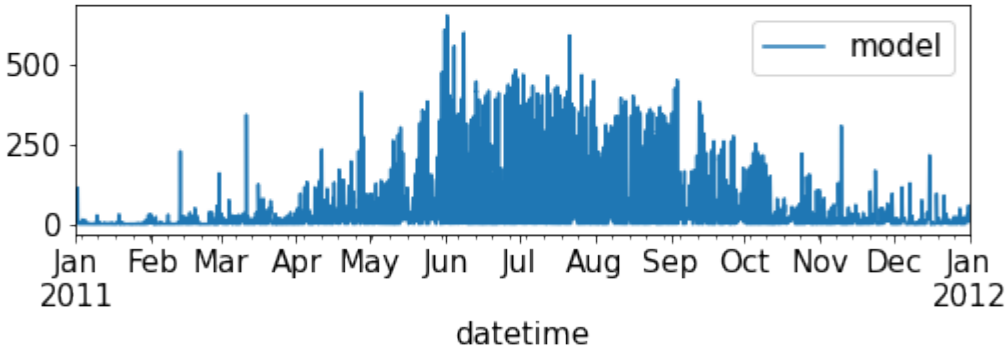
US-Oho-2011

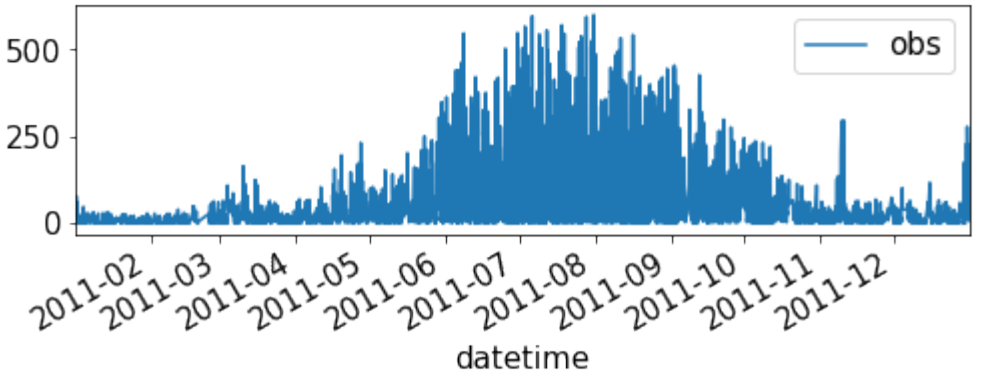
[47]:
```
g1,g2,g3,g4,g5,g6=g1_g6_all.loc['US-MMS',:].values
g_max=g_max
g1=1
s1=5.56

name='US-Oho'
year=2011
df_forcing= read_forcing(name,year)
gs_plot_test(g1,g2,g3,g4,g5,g6,g_max,s1,name,year)
```

```
2020-03-26 10:43:49,647 -- SuPy -- INFO -- All cache cleared.
2020-03-26 10:43:50,493 -- SuPy -- INFO -- All cache cleared.
2020-03-26 10:43:53,625 -- SuPy -- INFO -- ====================
2020-03-26 10:43:53,626 -- SuPy -- INFO -- Simulation period:
2020-03-26 10:43:53,627 -- SuPy -- INFO --   Start: 2010-12-31 23:05:00
2020-03-26 10:43:53,628 -- SuPy -- INFO --   End: 2011-12-31 23:00:00
2020-03-26 10:43:53,629 -- SuPy -- INFO --
2020-03-26 10:43:53,631 -- SuPy -- INFO -- No. of grids: 1
2020-03-26 10:43:53,632 -- SuPy -- INFO -- SuPy is running in serial mode
2020-03-26 10:44:11,821 -- SuPy -- INFO -- Execution time: 18.2 s
2020-03-26 10:44:11,822 -- SuPy -- INFO -- ====================
```


QE (test data set-fitted g1-g6)
QE (W m^{-2})
200
100
0
0
5
10
15
20
Time (UTC)
Model
Data


QE-test data-all season-MAE=25.41
Model
500
250
0
100
200
300
400
500
600
Obs
1-1


500
250
0
Jan
2011
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
Jan
2012
datetime
model


500
250
0
2011-02
2011-03
2011-04
2011-05
2011-06
2011-07
2011-08
2011-09
2011-10
2011-11
2011-12
datetime
obs

End of Parameters/tutorials/param4-conductance.ipynb

The following section was generated from source/Parameters/tutorials/param5-roughness-SuPy.ipynb

10.3.5 Roughness parameters (SuPy)

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import supy as sp
import warnings
warnings.filterwarnings('ignore')
```

Loading data, cleaning and getting ready for optimization

[2]:
```
name_of_site='US-MMS'
years=[2010,2012,2016]


df_attr=pd.read_csv('all_attrs.csv')
z_meas=df_attr[df_attr.site==name_of_site].meas.values[0]
h_sfc=df_zmeas=df_attr[df_attr.site==name_of_site].height.values[0]
folder='data/data_csv_zip_clean_roughness/'
site_file = folder+'/'+name_of_site + '_clean.csv.gz'
df_data = pd.read_csv(site_file, index_col='time', parse_dates=['time'])
# Rain
bb=pd.DataFrame(~np.isin(df_data.index.date,df_data[df_data.P!=0].index.date))
bb.index=df_data.index
df_data=df_data[bb.values]


df_data=df_data[(df_data['WS']!=0)]

df_years=df_data.loc[f'{years[0]}']
for i in years[1:]:
    df_years=df_years.append(df_data.loc[f'{i}'])

df_val = df_years.loc[:, ['H', 'USTAR', 'TA', 'RH', 'PA', 'WS']].dropna()
df_val.head()
```

[2]:

time	H	USTAR	TA	RH	PA	WS
2010-01-01 00:00:00	21.873	0.739	-8.08	70.503	98.836	3.695
2010-01-01 01:00:00	41.819	0.855	-9.17	72.757	98.880	3.928
2010-01-01 02:00:00	-6.078	0.699	-9.63	72.611	98.910	3.088
2010-01-01 03:00:00	-16.788	0.581	-10.03	73.868	98.970	3.623
2010-01-01 04:00:00	5.006	0.562	-10.36	74.242	99.030	3.474

Running supy to calculate z0 and d

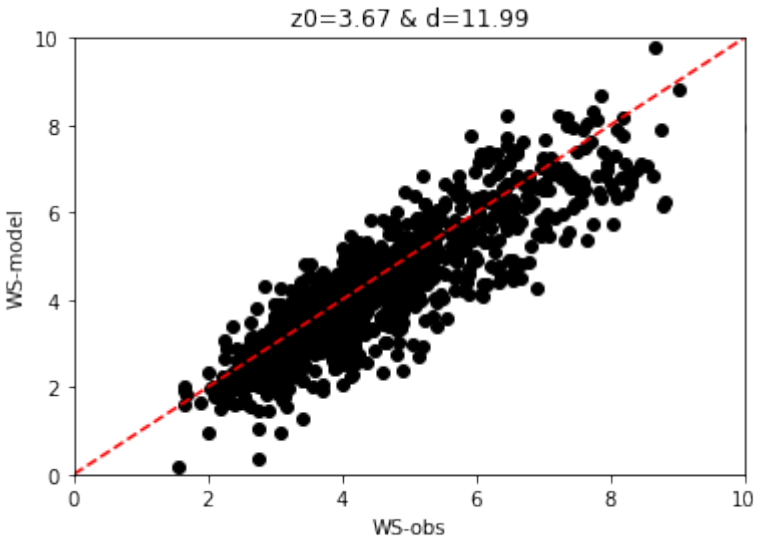
[3]:
```
z0,d,ser_ws,ser_ustar=sp.util.optimize_MO(df_val,z_meas,h_sfc)
```

Calculating model wind speed using logarithmic law

[4]:
```
def uz(z0,d,ser_ustar,z_meas):
    z = z_meas
    k = 0.4
    uz = (ser_ustar / k) * np.log((z - d) / z0)
    return uz

ws_model=uz(z0,d,ser_ustar,z_meas)
```

[5]:
```
plt.scatter(ser_ws,ws_model,color='k')
plt.xlabel('WS-obs')
plt.ylabel('WS-model')
plt.title(f'z0={np.round(z0,2)} & d={np.round(d,2)}')
plt.plot([0,10],[0,10],color='r',linestyle='--')
plt.ylim([0,10])
plt.xlim([0,10])
```

[5]:
```
(0.0, 10.0)
```



End of `Parameters/tutorials/param5-roughness-SuPy.ipynb`

CHAPTER

ELEVEN

NEED HELP AND BACKGROUND RESOURCES

Workshop issue or general help

You can submit an issue and we will try to help you as soon as possible.

UMEP

- UMEP Manual
- UMEP Overview Paper (*Lindberg et al. 2018*)
- UMEP YouTube channel
- To get news and help from other users, join the UMEP email list.
- FAQ and known issues are listed in the UMEP Manual.
- How to update UMEP.
- If you suspect that there is an error in UMEP, e.g. you get an error python message, please report to our bug reporter.

SUEWS

- SUEWS manual
- SUEWS paper (*Järvi et al. 2011*)
- To get news and help from other users, join the SUEWS email list:.
- How to update SUEWS. If SuPy is used (below), then the newest version is always used.

SuPy

- SuPy Manual
- SuPy paper (*Sun and Grimmond 2019*)
- FAQ
- How to update SuPy

QGIS

- QGIS software
- How to Update: QGIS gives a message & directions if a newer version is available
- QGIS manual
- QGIS bug reporter
- A Gentle Introduction to GIS
- QGIS Training Manual

Jupyter Notebooks

- Jupyter Notebook Cheatsheet
- *Recommended extensions*
- Jupyter Notebook Extension: Installation
- Introduction to Jupyter Notebooks *Background reading for Jupyter Notebooks*

Python

- *Introduction to Python*

GitHub

- Git Handbook: Learn about version control—in particular, Git, and how it works with GitHub.
- GitHub-based workflow: This guide explains how and why GitHub flow works.
- GitHub tutorial

Background meteorology

- *Introductory Material*

CHAPTER

TWELVE

GLOSSARY

AmeriFlux The AmeriFlux network is a community of sites and scientists measuring ecosystem carbon, water, and energy fluxes across the Americas, and committed to producing and sharing high quality eddy covariance data.

EC Eddy Covariance - a technique to measure turbulent heat, mass (e.g. water, CO_2) and momentum fluxes.

ERA5 Reanalysis data

Fetch Area influencing a sensor. For EC the area is upwind of the sensor.

Fortran A computer programming language

Jupyter Notebook A tool to help organize a project with progamming involved. An introduction is given here *Jupyter Notebooks: setting up your research-oriented coding environment*

LAI Leaf Area Index

Obukhov Length (L) A parameter with dimension of length that gives a relation between parameters characterizing dynamic, thermal, and buoyant processes. More detailed explanation refers to AMS wiki.

OHM Objective Hysteresis Model. Example coeffcients are given here

Python A computer programming language

SEB *Surface energy balance*. An urban definition

SUEWS Surface [Urban] Energy and Water Balance Scheme Manual

SuPy SUEWS that speaks Python Manual

UMEP Urban Multi-scale Environmental Predictor Manual

CHAPTER

THIRTEEN

BACKGROUND TO PYTHON AND JUPYTER NOTEBOOKS

Note: A tutorial for setting up Jupyter Notebook and its basic operations is given in *Jupyter Notebooks: setting up your research-oriented coding environment*.

13.1 Background reading for Jupyter Notebooks

The Jupyter Notebook is an ideal tool for exploratory data analysis. It provides a place where you can carry out your data processing, computation, plotting and even writing (using Markdown) in one stop. Also, it supports more programming languages than just Python: Julia, R, etc. In this course, we will only use *Python* for its well developed ecosystem in data science and scientific computing.

Below we only cover several key points for using Jupyter Notebooks. A more complete tutorial can be found here.

13.1.1 Jupyter Notebook Extensions

In addition to the Jupyter Notebook itself, we recommend Jupyter Notebook Extensions as well, which can further enhance your productivity in the Jupyter Notebook environment. A good guide can be found here.

Among a variety of extensions, the following are considered essential and particularly useful for this course, and many other data-analysis-oriented ones:

1. *Table of Contents*: this extension helps building well-structured notebooks and allows easier navigation.
2. *Collapsible Headings*: this tool further enhances the power of the above one by allowing collasible sections, particularly useful for large notebooks.
3. *ExecuteTime*: this gadget reports the executation time of each cell so you can always have a measure of the code efficiency.
4. *Code prettify*: this tool quickly formats your Python code with a pretty and tidy layout.

13.1.2 Best Practice

A good guide can be found here.

Structuring your notebook is key. This post helps you set up a Notebook with a good structure.

13.2 Python

13.2.1 Useful Links

There are a lot of excellent tutorials and web pages for helping with Python coding:

- 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 an essential tool for data analysis in Python.
  - 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

13.3 Python Tutorials

These tutorials demonstrate some basic data analysis and visualisation techniques using Python.

Note: These tutorials are in Jupyter notebooks and can be executed and viewed online.

The following section was generated from source/Jupyter/tutorials/tutorial-1-basic-plotting.ipynb

13.3.1 Basic plotting in Python

Here you will learn the basics of plotting in python including:

1. How to make a basic plot
2. How to add labels, legends and title to plot
3. How to change the color, fontsize, and other properties of the lines in the plot
4. How to save plots to files with different formats

Load the necessary packages

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

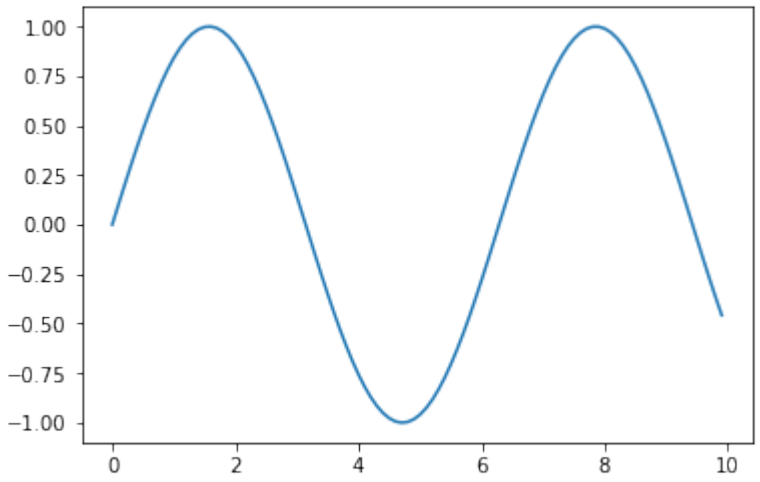
Let's create some sample data to plot

[2]:
```
x=np.arange(0,10,.1)
y=np.sin(x)
```

A first quick plot:

[3]:
```
plt.plot(x,y)
```

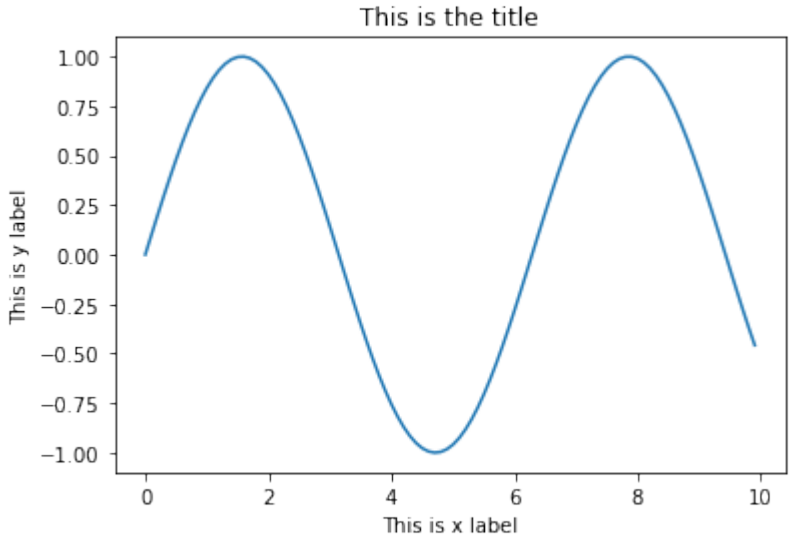
[3]:
```
[<matplotlib.lines.Line2D at 0x1edf2412b70>]
```



We want to add x and y labels and title to the plot:

[4]:
```
plt.plot(x,y)
plt.xlabel('This is x label')
plt.ylabel('This is y label')
plt.title('This is the title')
```

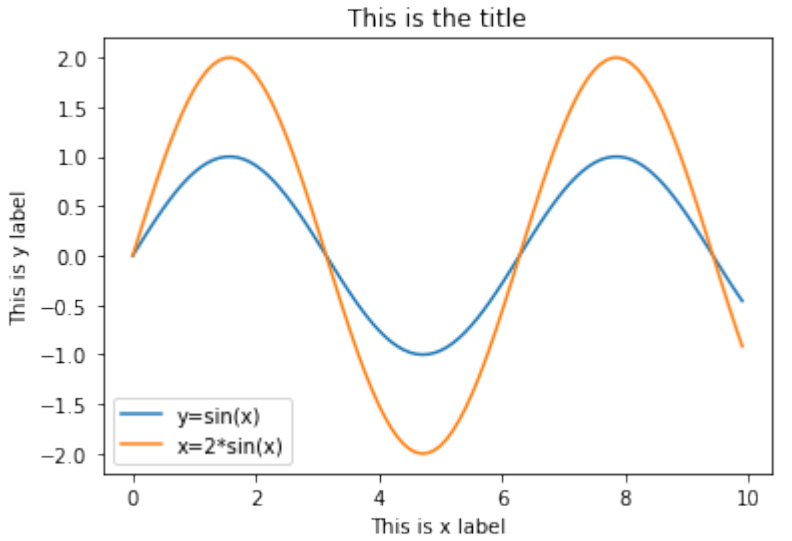
[4]:
```
Text(0.5, 1.0, 'This is the title')
```



Now add more data (lines) to the plot, and make a legend

[5]:
```
plt.plot(x,y,label='y=sin(x)')
plt.plot(x,2*y,label='x=2*sin(x)')
plt.xlabel('This is x label')
plt.ylabel('This is y label')
plt.title('This is the title')
plt.legend()
```

[5]:
```
<matplotlib.legend.Legend at 0x1edf252db70>
```



The mathematical formula can be improved in the legend

You need to use: `$your formula here$`

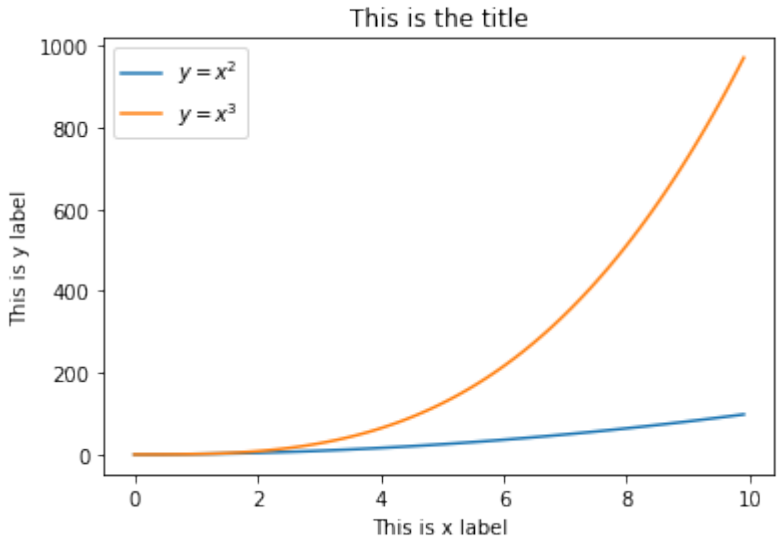
[6]:
```
plt.plot(x,x*x,label='$y=x^2$')
plt.plot(x,x*x*x,label='$y=x^3$')
plt.xlabel('This is x label')
plt.ylabel('This is y label')
```

(continues on next page)

(continued from previous page)

```
plt.title('This is the title')
plt.legend()
```

[6]:
```
<matplotlib.legend.Legend at 0x1edf2597b70>
```



The superscript is made using `^{your superscript}` and subscript is made using `_{your subscript}`

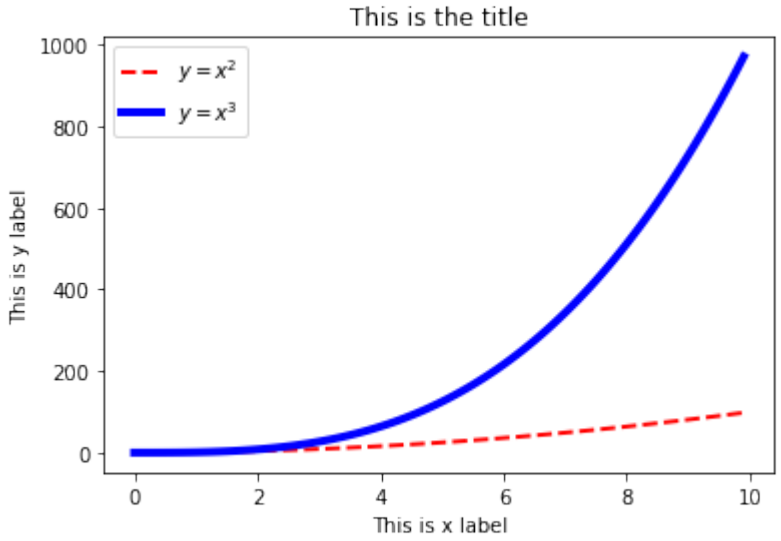
`$x^{-3}$`=x^{-3}

`$x_{data}$`=x_{data}

The line width, line style and line color can all be changed:

[7]:
```
plt.plot(x,x*x,label='$y=x^2$',linewidth=2,color='r',linestyle='--')
plt.plot(x,x*x*x,label='$y=x^3$',linewidth=4,color='b')
plt.xlabel('This is x label')
plt.ylabel('This is y label')
plt.title('This is the title')
plt.legend()
```

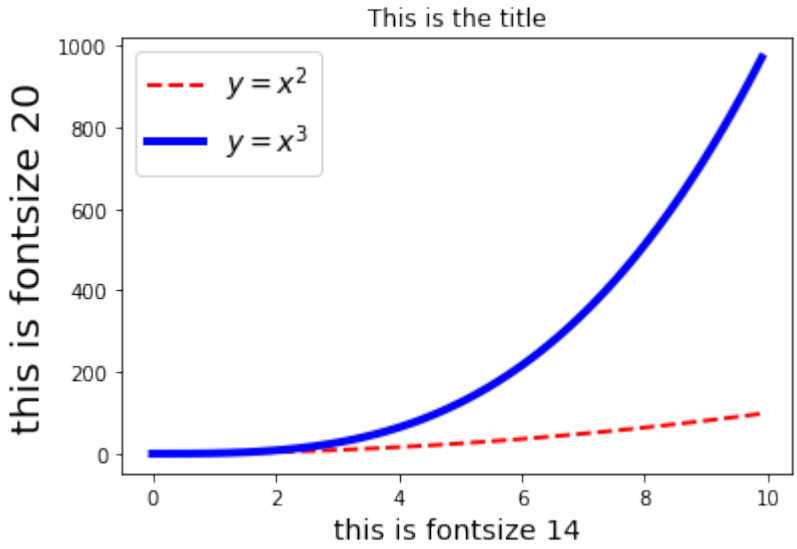
[7]:
```
<matplotlib.legend.Legend at 0x1edf262c048>
```



To change the text fontsize:

[8]:
```
plt.plot(x,x*x,label='$y=x^2$',linewidth=2,color='r',linestyle='--')
plt.plot(x,x*x*x,label='$y=x^3$',linewidth=4,color='b')
plt.xlabel('this is fontsize 14',fontsize=14)
plt.ylabel('this is fontsize 20',fontsize=20)
plt.title('This is the title')
plt.legend(fontsize=14)
```

[8]:
```
<matplotlib.legend.Legend at 0x1edf26c7128>
```



To change the figure size and shape:

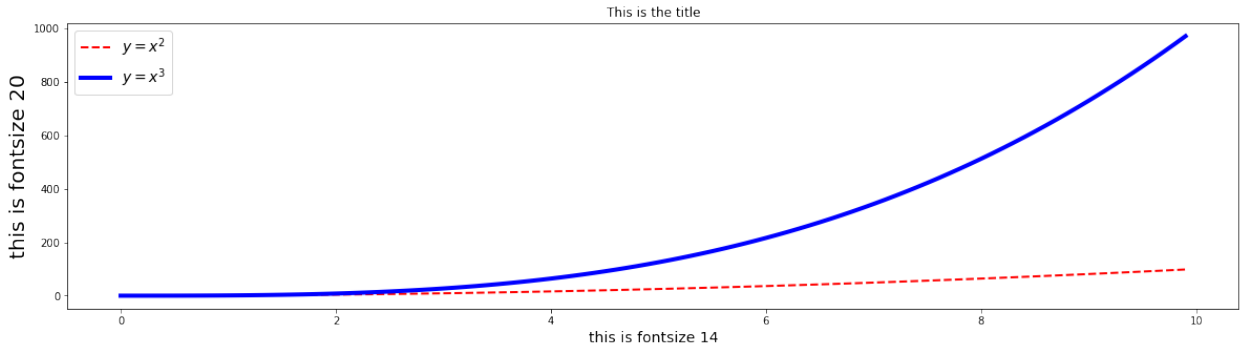
[9]:
```
plt.figure(figsize=(20,5))
plt.plot(x,x*x,label='$y=x^2$',linewidth=2,color='r',linestyle='--')
plt.plot(x,x*x*x,label='$y=x^3$',linewidth=4,color='b')
plt.xlabel('this is fontsize 14',fontsize=14)
plt.ylabel('this is fontsize 20',fontsize=20)
```

(continues on next page)

(continued from previous page)

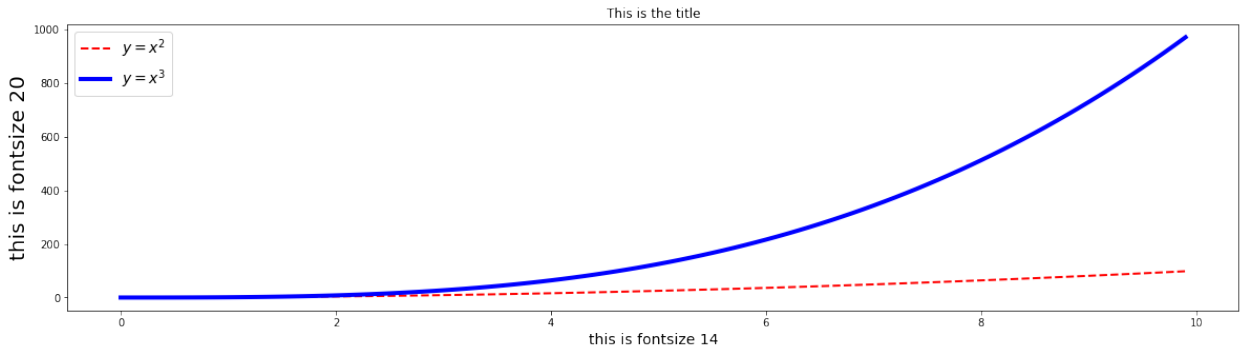
```
plt.title('This is the title')
plt.legend(fontsize=14)
```

[9]:
```
<matplotlib.legend.Legend at 0x1edf276e160>
```



and finally, to save your figure to a file:

[10]:
```
plt.figure(figsize=(20,5))
plt.plot(x,x*x,label='$y=x^2$',linewidth=2,color='r',linestyle='--')
plt.plot(x,x*x*x,label='$y=x^3$',linewidth=4,color='b')
plt.xlabel('this is fontsize 14',fontsize=14)
plt.ylabel('this is fontsize 20',fontsize=20)
plt.title('This is the title')
plt.legend(fontsize=14)
plt.savefig('sample.png',dpi=100)
```



You can use various formats such as jpg, pdf, etc.

You can control the quality of the saved file using `dpi` keyword. Be careful that using large numbers for `dpi` would make the process of saving figures slow and the created file very large.

End of `Jupyter/tutorials/tutorial-1-basic-plotting.ipynb`

The following section was generated from source/Jupyter/tutorials/tutorial-2-advance-plotting.ipynb

13.3.2 More plotting in Python

Some more advanced plotting in Python:

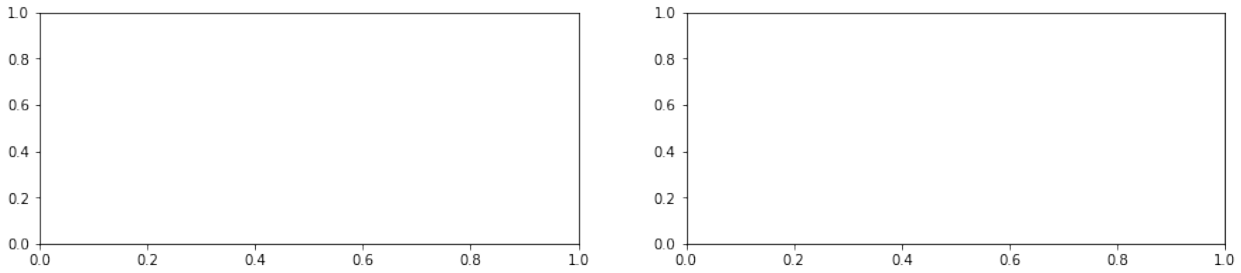
1. How to have multiple plots in one figure (subplots)
2. How to handle different axes in one figure
3. How to position legend
4. How to change x and y ticks

Loading necessary packages

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

We want a figure with two horizontal subplots:

[2]:
```
fig,axs=plt.subplots(1,2,figsize=(15,3))
```



These figures have two axes (left and right). You can see this by printing the contents of `axs`:

[3]:
```
print(axs)
```

```
[<matplotlib.axes._subplots.AxesSubplot object at 0x000002BA219DBEB8>
 <matplotlib.axes._subplots.AxesSubplot object at 0x000002BA21CA0E10>]
```

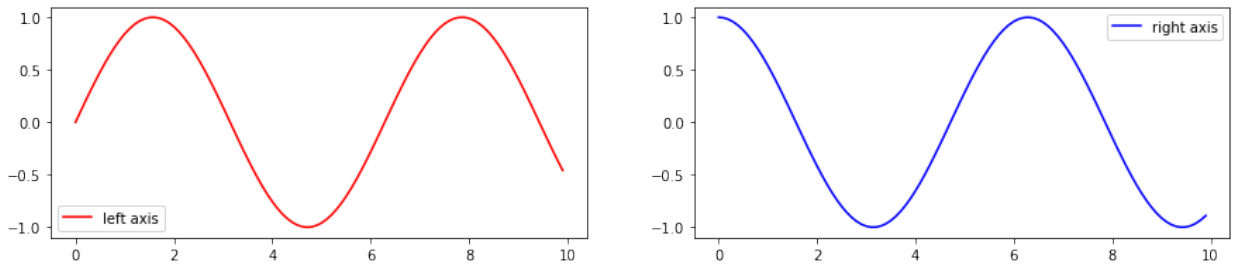
Create some data to plot:

[4]:
```
x=np.arange(0,10,.1)
y1=np.sin(x)
y2=np.cos(x)
```

Plot them :

[5]:
```
fig,axs=plt.subplots(1,2,figsize=(15,3))

ax1=axs[0] #first axis (left one)
ax2=axs[1] #second axis (right one)

ax1.plot(x,y1,color='r',label='left axis')
ax1.legend()

ax2.plot(x,y2,color='b',label='right axis')
ax2.legend()
```

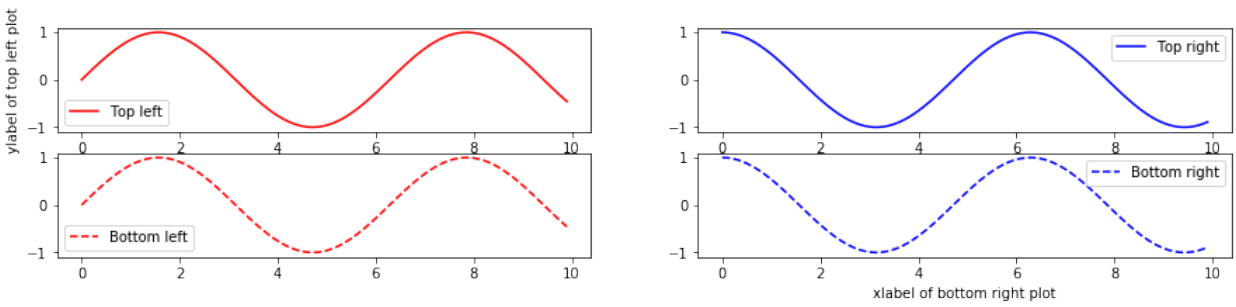
[5]:
```
<matplotlib.legend.Legend at 0x2ba21db06a0>
```



It is easy to define which axis you want to plot, and everything is similar to single plots (almost everything, you see later on why). Now Let's have more subplots:

[6]:
```
fig,axs=plt.subplots(2,2,figsize=(15,3))

ax11=axs[0][0] # Top left
ax12=axs[0][1] # Top right
ax21=axs[1][0] # Bottom left
ax22=axs[1][1] # Bottom right

ax11.plot(x,y1,color='r',label='Top left')
ax11.legend()
ax11.set_ylabel('ylabel of top left plot')

ax12.plot(x,y2,color='b',label='Top right')
ax12.legend()


ax21.plot(x,y1,color='r',linestyle='--',label='Bottom left')
ax21.legend()

ax22.plot(x,y2,color='b',linestyle='--',label='Bottom right')
ax22.legend()
ax22.set_xlabel('xlabel of bottom right plot')
```

[6]:
```
Text(0.5, 0, 'xlabel of bottom right plot')
```



Important note:

In contrast to single plots, when you set a property for the plot, you use `set_{keyword}`.

For example:

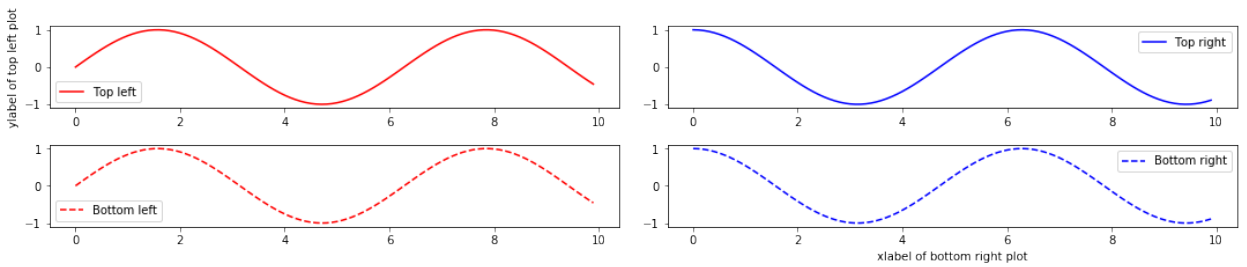
in single plot: `plt.xlabel('your xlabel')`

in subplots plot: `ax.set_xlabel('your xlabel')`

To modify the sub-plot postion we use: `plt.tight_layout()`

[7]:
```
fig,axs=plt.subplots(2,2,figsize=(15,3))
plt.tight_layout()

ax11=axs[0][0] # Top left
ax12=axs[0][1] # Top right
ax21=axs[1][0] # Bottom left
ax22=axs[1][1] # Bottom right

ax11.plot(x,y1,color='r',label='Top left')
ax11.legend()
ax11.set_ylabel('ylabel of top left plot')

ax12.plot(x,y2,color='b',label='Top right')
ax12.legend()


ax21.plot(x,y1,color='r',linestyle='--',label='Bottom left')
ax21.legend()

ax22.plot(x,y2,color='b',linestyle='--',label='Bottom right')
ax22.legend()
ax22.set_xlabel('xlabel of bottom right plot')
```

[7]:
```
Text(0.5, 6.000000000000025, 'xlabel of bottom right plot')
```



To adjust the distance between the subplots, you should use: `fig.subplots_adjust(hspace=)`

For example:

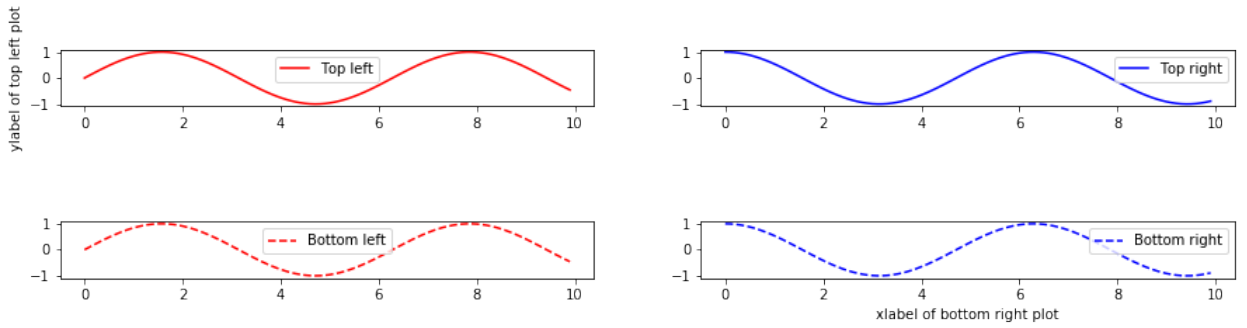
[8]:
```
fig,axs=plt.subplots(2,2,figsize=(15,3))
fig.subplots_adjust(hspace=2)

ax11=axs[0][0] # Top left
ax12=axs[0][1] # Top right
ax21=axs[1][0] # Bottom left
ax22=axs[1][1] # Bottom right

ax11.plot(x,y1,color='r',label='Top left')
ax11.legend()
ax11.set_ylabel('ylabel of top left plot')

ax12.plot(x,y2,color='b',label='Top right')
ax12.legend()


ax21.plot(x,y1,color='r',linestyle='--',label='Bottom left')
ax21.legend()
```

(continues on next page)

(continued from previous page)

```
ax22.plot(x,y2,color='b',linestyle='--',label='Bottom right')
ax22.legend()
ax22.set_xlabel('xlabel of bottom right plot')
```

[8]:
```
Text(0.5, 0, 'xlabel of bottom right plot')
```

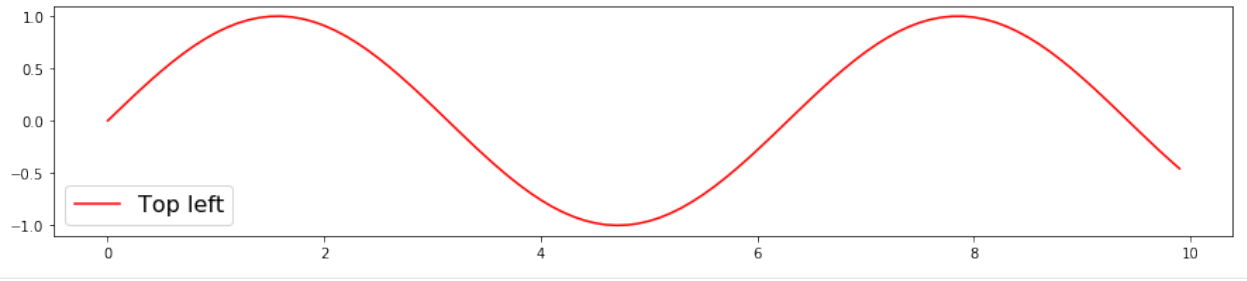


You can change the position of legend using the keyword `loc=`. Possible values for this keyword:

```
best
upper right
upper left
lower left
lower right
right
center left
center right
lower center
upper center
center
```

[9]:
```
fig,axs=plt.subplots(1,1,figsize=(15,3))
axs.plot(x,y1,color='r',label='Top left')
axs.legend(loc='lower left',fontsize=16)
```

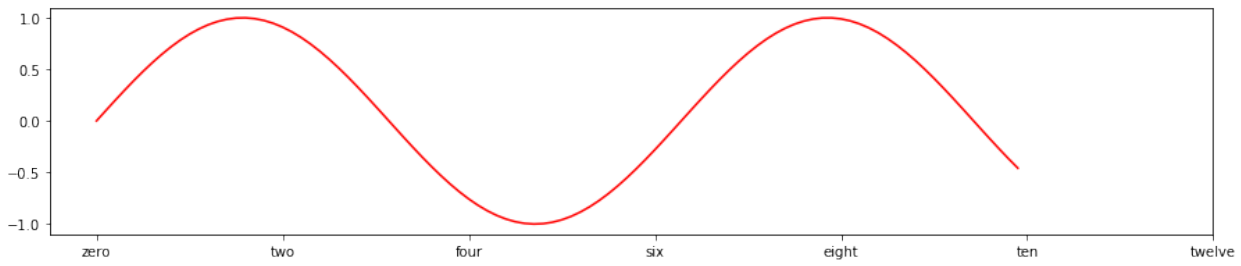
[9]:
```
<matplotlib.legend.Legend at 0x2ba22269eb8>
```



You can can customise x or y ticks using the following:

[10]:
```
fig,axs=plt.subplots(1,1,figsize=(15,3))
axs.plot(x,y1,color='r')
axs.set_xticks([0,2,4,6,8,10,12])

axs.set_xticklabels(['zero','two','four','six','eight','ten','twelve'])
```

[10]:
```
[Text(0, 0, 'zero'),
 Text(0, 0, 'two'),
 Text(0, 0, 'four'),
 Text(0, 0, 'six'),
 Text(0, 0, 'eight'),
 Text(0, 0, 'ten'),
 Text(0, 0, 'twelve')]
```



End of Jupyter/tutorials/tutorial-2-advance-plotting.ipynb

The following section was generated from source/Jupyter/tutorials/tutorial-3-Plotting albedo values in one plot.ipynb

13.3.3 Plotting albedo values in one plot

Here you learn how to plot the albedo of two different days in one plot

First, some packages:

[1]:
```
# These packages are necessary later on. Load all the packages in one place for␣
↪consistency
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from pathlib import Path
import datetime
```

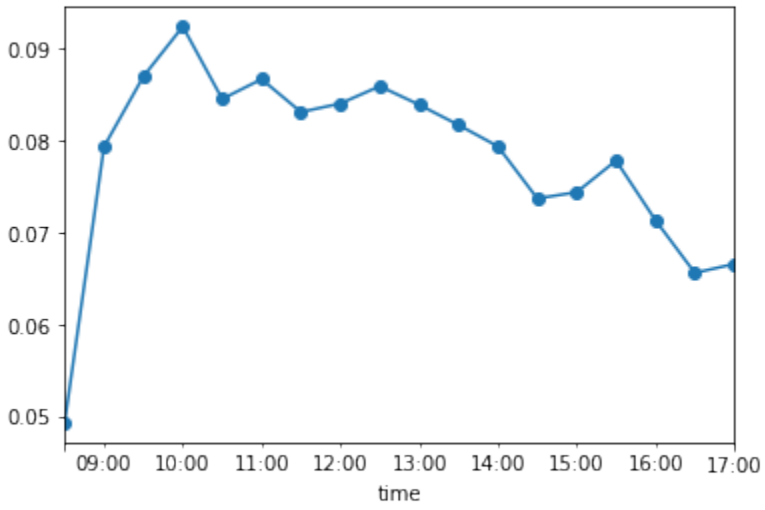
Read a data file and calculate albedos for all the data

[3]:
```
#The path of the directory where all AMF data are
path_dir = Path.cwd()/'data'/'1'
name_of_site = 'CA-Obs_clean.csv.gz'
path_data = path_dir/name_of_site
path_data.resolve()
df_data = pd.read_csv(path_data, index_col='time',parse_dates=['time'])
ser_alb=df_data['SWOUT']/df_data['SWIN']
```

Plot the albedo of one day

[4]:
```
date1 = '2001 10 27'
ser_alb[ser_alb.between(0,1)&(df_data['SWIN']>5)].loc[date1].plot(marker='o')
```

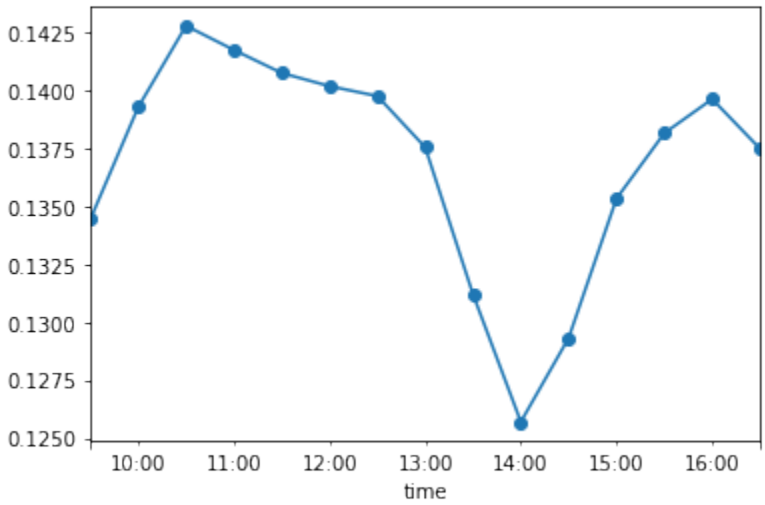
[4]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1f959c23c88>
```



And another day

[5]:
```
date2 = '2001 11 21'
ser_alb[ser_alb.between(0,1)&(df_data['SWIN']>5)].loc[date2].plot(marker='o')
```

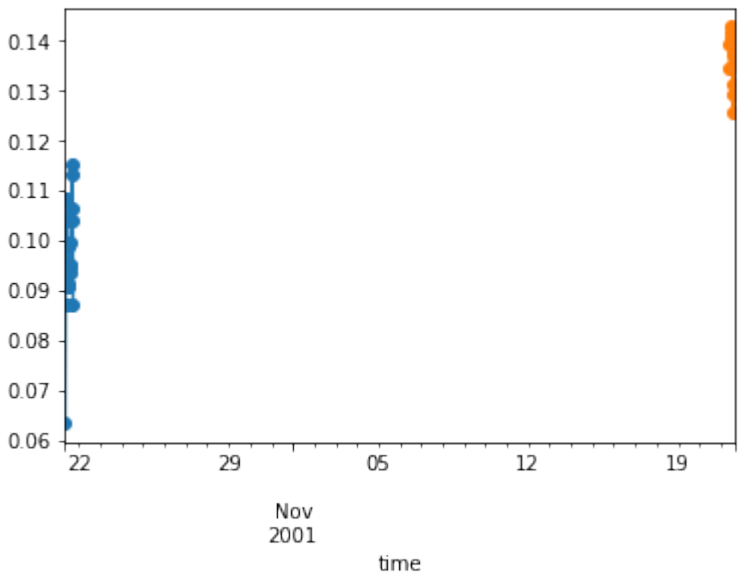
[5]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1f958793240>
```



Plot the albedo of both days on one plot using the previous methods

[6]:
```
date1 = '2001 10 21'
ser_alb[ser_alb.between(0,1)&(df_data['SWIN']>5)].loc[date1].plot(marker='o')
date2 = '2001 11 21'
ser_alb[ser_alb.between(0,1)&(df_data['SWIN']>5)].loc[date2].plot(marker='o')
```

[6]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1f9587ef198>
```



As the days are different, we do not get a good plot.

To fix this we need to find the time of the day each data point is associated with. For example, for date1, we use the time index from `Dataframe.index.time` :

[11]:
```
ser_alb[ser_alb.between(0,1)&(df_data['SWIN']>5)].loc[date1].index.time
```

[11]:
```
array([datetime.time(8, 30), datetime.time(9, 0), datetime.time(9, 30),
       datetime.time(10, 0), datetime.time(10, 30), datetime.time(11, 0),
       datetime.time(11, 30), datetime.time(12, 0), datetime.time(12, 30),
       datetime.time(13, 0), datetime.time(13, 30), datetime.time(14, 0),
       datetime.time(14, 30), datetime.time(15, 0), datetime.time(15, 30),
       datetime.time(16, 0), datetime.time(16, 30), datetime.time(17, 0),
       datetime.time(17, 30)], dtype=object)
```

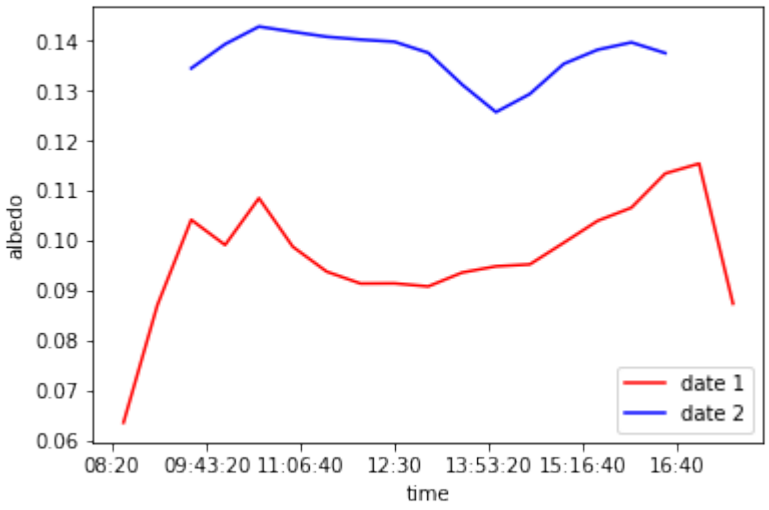
Lets do this for both dates

[12]:
```
X1=ser_alb[ser_alb.between(0,1)&(df_data['SWIN']>5)].loc[date1].index.time
X2=ser_alb[ser_alb.between(0,1)&(df_data['SWIN']>5)].loc[date2].index.time
Y1=ser_alb[ser_alb.between(0,1)&(df_data['SWIN']>5)].loc[date1]
Y2=ser_alb[ser_alb.between(0,1)&(df_data['SWIN']>5)].loc[date2]
```

Now plot them in one figure

[13]:
```
plt.plot(X1,Y1,'r',label='date 1')
plt.plot(X2,Y2,'b',label='date 2')
plt.legend()
plt.ylabel('albedo')
```

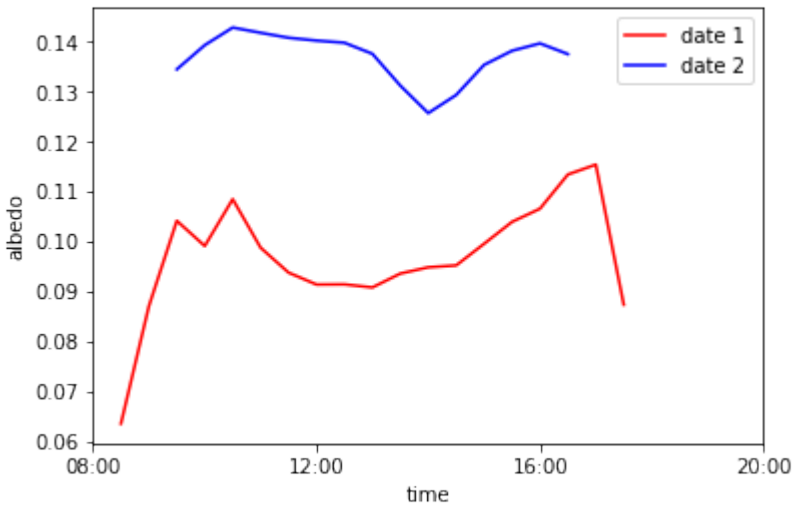
[13]:
```
Text(0, 0.5, 'albedo')
```



Much better! But the xticks are a little messy. We fix this by:

[14]:
```
plt.plot(X1,Y1,'r',label='date 1')
plt.plot(X2,Y2,'b',label='date 2')
plt.legend()
plt.ylabel('albedo')
plt.xticks([datetime.time(x, 0) for x in [8,12,16,20]])
```

[14]:
```
([<matplotlib.axis.XTick at 0x1f958b0cfd0>,
  <matplotlib.axis.XTick at 0x1f958b0c7f0>,
  <matplotlib.axis.XTick at 0x1f958b2db70>,
  <matplotlib.axis.XTick at 0x1f958b2de80>],
 <a list of 4 Text xticklabel objects>)
```



End of Jupyter/tutorials/tutorial-3-Plotting albedo values in one plot.ipynb

The following section was generated from `source/Jupyter/tutorials/tutorial-4-Functions in Python.ipynb`

13.3.4 Functions in Python

What you will learn here:

1. How to write simple functions in Python
2. How to call functions
3. How to pass variables and parameters to functions
4. How to use `return` in functions
5. How to integrate functions together

These packages are necessary later on. Load all the packages in one place for consistency

[1]:
```
import numpy as np
import pandas as pd
```

Let's say you want a function that simply prints `Hello World`, you define the function:

[2]:
```
def your_printer():
    print('Hello World')
```

So it is very simple to define the function but when you run this cell nothing happens because the function has not been called. This is the handy part of functions in Python (or any other programming languages). You define a function once and then you can all it whenever you want. To call the function simply:

[3]:
```
your_printer()
```

```
Hello World
```

Congrats! you just wrote and called the first function in Python.

The application of functions goes beyond printing. You can pass variables into function and they do the job for you. For example, if you modify `your_printer` function to print what we pass to it:

[4]:
```
def your_printer(your_word):
    print(your_word)
```

Now let's call this function:

[5]:
```
your_printer('This is your word!')
```

```
This is your word!
```

You can call a function multiple times, saving you time and space in coding:

[6]:
```
your_printer('first word')
your_printer('second word')
your_printer('third word')
```

```
first word
second word
third word
```

More commonly you pass a variable to a function, the function does some mathematical operations and returns the results for future use. In this case, you use `return` in the function. For example

[8]:
```
def sum_two_number(number1,number2):

    return number1+number2
```

This function, receives two number and returns the sum. You can use it as follows

[9]:
```
sum_numbers=sum_two_number(1,2)
print(sum_numbers)
```

```
3
```

You can use the result of the function to do other stuff. For example:

[10]:
```
print(sum_two_number(sum_numbers,4))
```

```
7
```

Now a more complex example, using the following equations:

$q_1 = x^2 + x$

$q_2 = \sin(q_1) + \frac{1}{q_1+1}$

$q_3 = q_2 * q_1 + x^3$

$q_4 = q_1 + q_2 + q_3 + x$

Given a value for x, you want to find the value of q_4. This is where functions come in handy. Let's implement this:

[11]:
```
############################## q1

def cal_q1(x):

    return x**2+x

############################## q2

def cal_q2(q1):

    return np.sin(q1)+1/(q1+1)

############################## q3

def cal_q3(q1,q2,x):

    return q2*q1+x**3

############################## q4

def cal_q4(x):

    q1=cal_q1(x)
    q2=cal_q2(q1)
    q3=cal_q3(q1,q2,x)

    return q1+q2+q3+x
```

now we can calculate values of q_4 given any arbitrary value for x

[12]:
```
cal_q4(12)
```

[12]:
```
1758.5598148460792
```

[13]:
```
cal_q4(1)
```

[13]:
```
7.727892280477045
```

[14]:
```
cal_q4(5)
```

[14]:
```
130.3710196531213
```

End of Jupyter/tutorials/tutorial-4-Functions in Python.ipynb

The following section was generated from source/Jupyter/tutorials/tutorial-5-Scatter_plot_datetime.ipynb

13.3.5 Getting a subset of a time series and creating a Scatter plot

Here you learn how to:

1. plot a time series using dots (scatter plot)
2. Get subset of time series data and plot them

First, some packages:

[1]:
```
# These packages are necessary later on. Load all the packages in one place for␣
↪consistency
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from pathlib import Path
import datetime
```

Load the data:

[2]:
```
#The path of the directory where all AMF data are
path_dir = Path.cwd()/'data'/'1'
name_of_site = 'CA-Obs_clean.csv.gz'
path_data = path_dir/name_of_site
path_data.resolve()
df_data = pd.read_csv(path_data, index_col='time',parse_dates=['time'])
df_data.head()
```

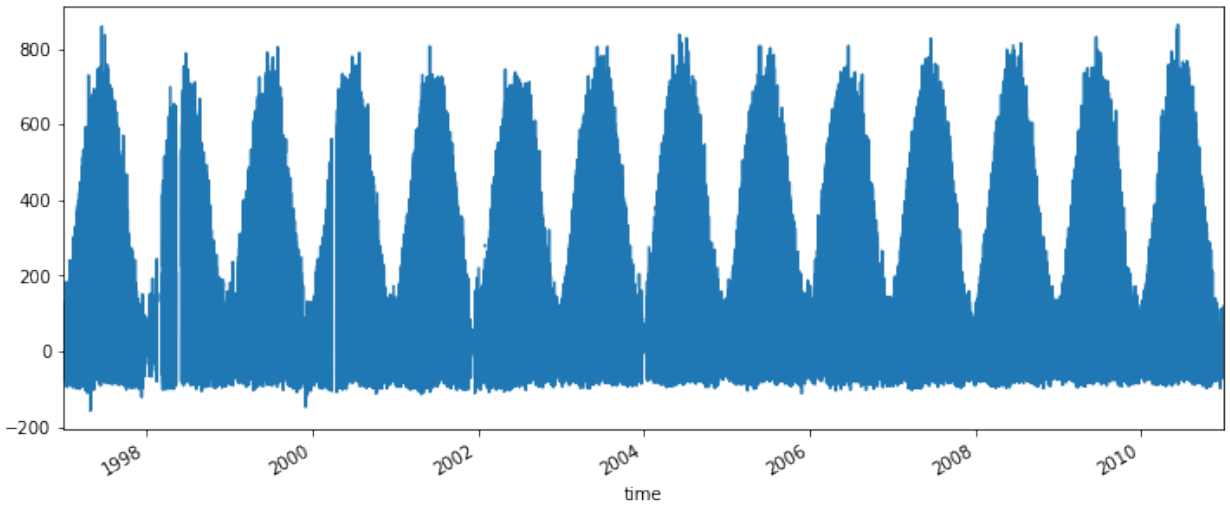
[2]:

time	WS	RH	TA	PA	WD	P	SWIN	LWIN	SWOUT	LWOUT	NETRAD	H	LE
1997-01-01 00:30:00	2.988	73.036	-24.570	93.942	102.84	NaN	-0.14	213.39	0.01	214.83	-1.59	NaN	NaN
1997-01-01 01:00:00	2.671	73.146	-24.562	93.887	96.09	NaN	-0.03	216.73	0.03	215.03	1.64	NaN	NaN
1997-01-01 01:30:00	2.303	73.151	-24.431	93.934	112.34	NaN	0.24	223.46	0.02	215.91	7.77	NaN	NaN
1997-01-01 02:00:00	2.789	73.093	-24.379	93.917	109.16	NaN	-0.04	218.32	0.00	215.72	2.56	NaN	NaN
1997-01-01 02:30:00	2.274	73.140	-24.284	93.977	115.74	NaN	0.14	217.89	0.01	215.99	2.02	NaN	NaN

Plot the net all-wave radiation:

[3]:
```
df_data['NETRAD'].plot(figsize=(12,5))
```

[3]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x26ffae40f28>
```



If you want to focus on a subset of the data, use this feature of pandas package (DataFrame.loc[index1:index2]):

[4]:
```
df_sub=df_data.loc['2002 07 01':'2002 07 30']
df_sub.head()
```

[4]:
```
                        WS      RH      TA      PA      WD    P  SWIN    LWIN  \
time
2002-07-01 00:00:00  3.508  51.637  14.268  93.494  266.48  0.0  0.13  286.54
2002-07-01 00:30:00  3.183  53.221  13.799  93.509  265.86  0.0  0.07  282.77
2002-07-01 01:00:00  3.759  55.664  13.182  93.506  266.07  0.0 -0.11  281.14
2002-07-01 01:30:00  4.518  52.553  13.217  93.500  276.19  0.0  0.02  282.81
2002-07-01 02:00:00  3.940  52.343  13.150  93.484  275.58  0.0 -0.35  282.55

                     SWOUT   LWOUT  NETRAD       H      LE
time
2002-07-01 00:00:00  -0.20  361.24  -74.37  -3.451   0.175
2002-07-01 00:30:00  -0.23  358.10  -75.02     NaN     NaN
2002-07-01 01:00:00  -0.14  356.13  -74.96 -14.770   1.628
2002-07-01 01:30:00   0.15  362.15  -79.47 -69.300  12.360
2002-07-01 02:00:00   0.15  362.48  -80.43 -35.270   3.713
```

[5]:
```
df_sub.index
```

[5]:
```
DatetimeIndex(['2002-07-01 00:00:00', '2002-07-01 00:30:00',
               '2002-07-01 01:00:00', '2002-07-01 01:30:00',
               '2002-07-01 02:00:00', '2002-07-01 02:30:00',
               '2002-07-01 03:00:00', '2002-07-01 03:30:00',
               '2002-07-01 04:00:00', '2002-07-01 04:30:00',
               ...
               '2002-07-30 19:00:00', '2002-07-30 19:30:00',
               '2002-07-30 20:00:00', '2002-07-30 20:30:00',
               '2002-07-30 21:00:00', '2002-07-30 21:30:00',
               '2002-07-30 22:00:00', '2002-07-30 22:30:00',
```

(continues on next page)

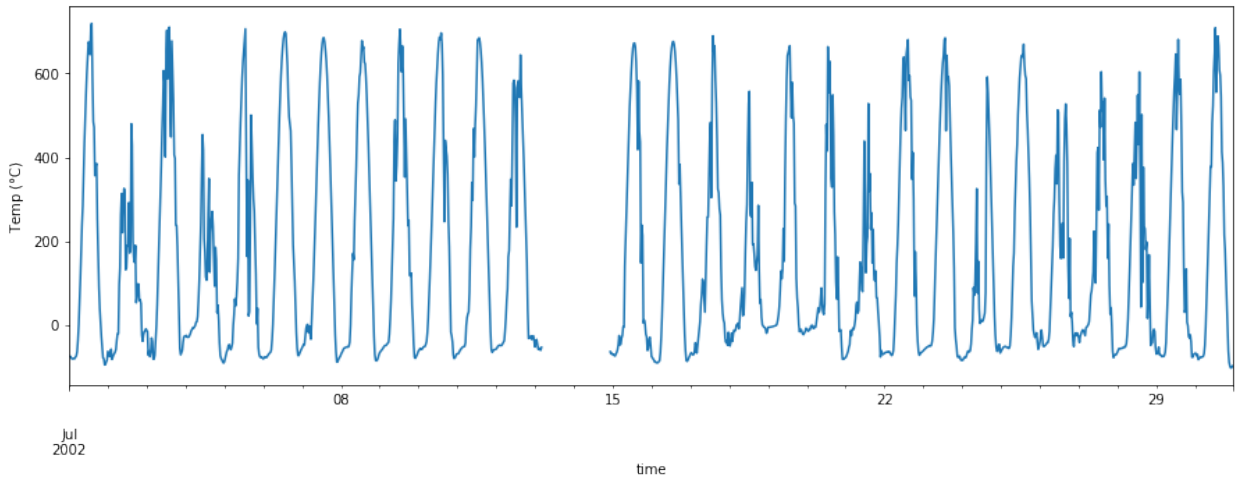
(continued from previous page)

```
               '2002-07-30 23:00:00', '2002-07-30 23:30:00'],
              dtype='datetime64[ns]', name='time', length=1440, freq=None)
```

We have a subset of df_data which contains the data from July 2002. Now let's plot it:

[6]:
```
df_sub['NETRAD'].plot(figsize=(15,5))
plt.ylabel('Temp ($\degree$C)')
```

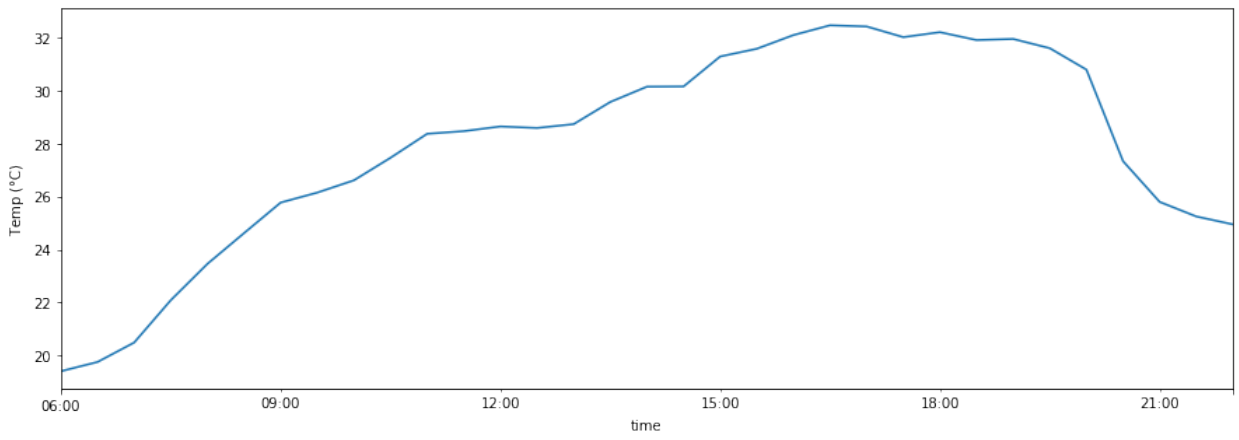
[6]:
```
Text(0, 0.5, 'Temp ($\\degree$C)')
```



You can specify the date and time of interest in the subset:

[7]:
```
df_sub.loc['2002 07 12 6:00:00':'2002 07 12 22:00:00']['TA'].plot(figsize=(15,5))
plt.ylabel('Temp ($\degree$C)')
```

[7]:
```
Text(0, 0.5, 'Temp ($\\degree$C)')
```



Sometimes we want to plot points instead of lines (e.g. if there are missing data, large variations in the data etc). To do this use plt.scatter:

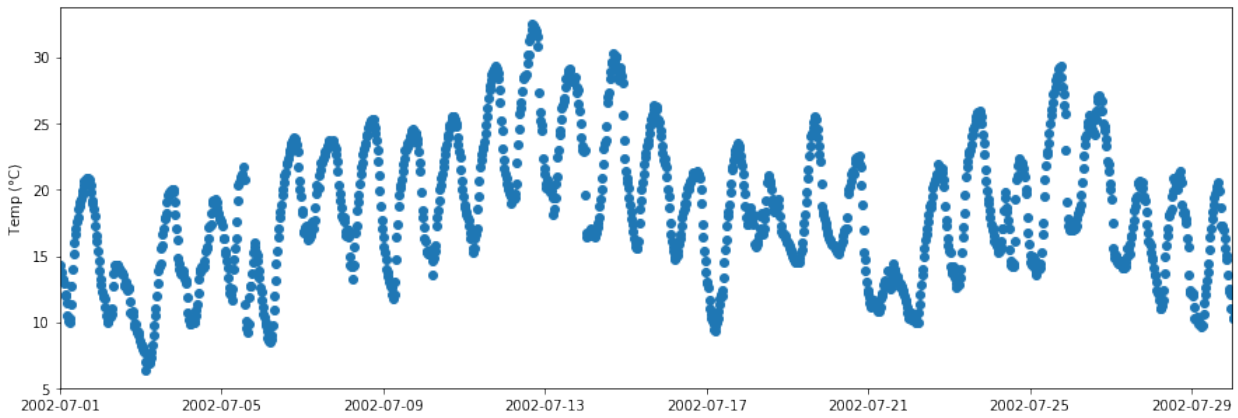
[8]:
```
Y=df_sub['TA']
X=df_sub.index
```

(continues on next page)

(continued from previous page)

```
fig,ax=plt.subplots(1,1,figsize=(15,5))
plt.scatter(X,Y)
plt.xlim([df_sub.index.date.min(),df_sub.index.date.max()])
plt.ylabel('Temp ($\degree$C)')
```

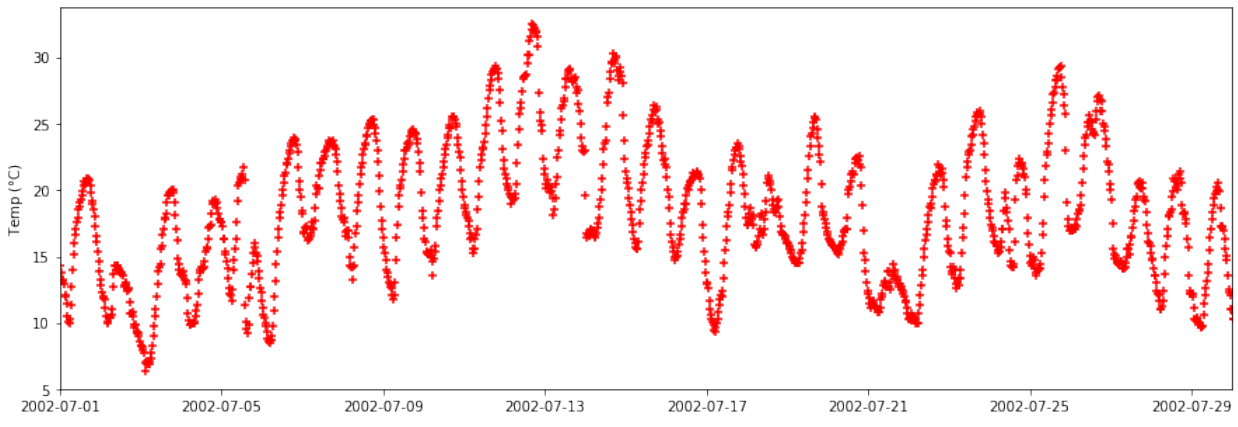
[8]:
```
Text(0, 0.5, 'Temp ($\\degree$C)')
```



The marker color and style can be changed (different types of markers are shown here):

[9]:
```
Y=df_sub['TA']
X=df_sub.index

fig,ax=plt.subplots(1,1,figsize=(15,5))
plt.scatter(X,Y,color='r',marker='+')
plt.xlim([df_sub.index.date.min(),df_sub.index.date.max()])
plt.ylabel('Temp ($\degree$C)')
```

[9]:
```
Text(0, 0.5, 'Temp ($\\degree$C)')
```



End of Jupyter/tutorials/tutorial-5-Scatter$_{p}lot_{d}atetime.ipynb$

CHAPTER

FOURTEEN

METEOROLOGY

Some introductory material, that is covered in many books and papers more thoroughly.

14.1 Fluxes

14.1.1 Surface energy balance

The surface energy balance (SEB) can be defined for an extensive simple surface as (units: $\mathrm{W\ m^{-2}}$):

$$Q^* = Q_H + Q_E + Q_G \tag{14.1}$$

where Q^* is the net all wave radiation, the turbulent sensible heat flux (Q_H) and the turbulent latent heat flux (Q_E) and the soil heat flux (Q_G) (units: $\mathrm{W\ m^{-2}}$).

Note other notation that is commonly used for the SEB are (in the same order as above):

$$R_n = H + LE + G$$

Bowen ratio

The Bowen ratio (β) is the ratio of sensible to latent heat flux:

$$\beta = Q_H / Q_E \tag{14.2}$$

14.1.2 Radiation balance

The Q^* (or R_n) within the *SEB* consists of:

$$Q^* = K_\downarrow - K_\uparrow + L_\downarrow - L_\uparrow \tag{14.3}$$

which includes the incoming (↓) and outgoing (↑) shortwave (K) and longwave radiation (L) fluxes.

SEB and Radiation Balance Measurements

Each of these fluxes can be directly measured or modelled using several methods (and data inputs).

Radiation fluxes

Various types of radiometers are used. For shortwave radiation, pyranometers are used, and for longwave radiation, pyrgeometers are used. The source area or field of view of a radiation sensor is fixed by geometery.

Soil Heat Fluxes

Soil heat flux plates are used with temperature sensors above to determine the heat gain/loss between the plate (e.g. at 0.05 m below the surface) and the surface. In more complex environments the storage heat flux (heating and cooling) of the whole volume needs to be accounted for. For example in a forest, the trees (trunk, branches, leaves, air) as well as the soil itself. So in most environments the soil heat flux is one part of the storage heat flux.

Turbulent heat Fluxes

The turbulent heat fluxes and momentum can be measured using Eddy Covariance techniques which requires a sonic anemometer and an infrared gas analyser. An alternative method is it use scintillometry.

As the source area of EC and scintillometer sensors varies with wind direction, fetch, stability etc, the surface characteristics may change with time if the fetch is not homogeneous.

SEB and Radiation Balance Modelling

Generally, to calculate a convective or conductive flux, data are needed to determine the size of the gradient (e.g. temperature difference) and the ability of the medium (e.g. air, soil) to transfer heat (or mass). The latter may use a resistance scheme, an eddy diffusivity, or other approach, which changes with the state of the medium (e.g. stability if air, moisture state if soil).

14.2 Model Parameters

Land surface models use parameters to describe the surface.

14.2.1 Albedo

From the short-wave radiation (K), within the Eq.14.3 *Radiation balance* the albedo (α) is calculated:

$$\alpha = K_{\uparrow}/K_{\downarrow} \tag{14.4}$$

using the incoming ($\downarrow$) and outgoing ($\uparrow$) shortwave radiation (K) fluxes.

14.2.2 Roughness length (z_0) and displacement height (d)

If the displacement height is known, or is negligible, the logarithmic law equation can be rearranged with observed z_0 and mean wind speed to allow z_0 to be determined. As this may vary we normally take the median of a minimum of 20 results for a wind direction sector. If you have a period with a lot of *neutral* conditions you may be able to get a lot of samples rapidly.

$$z_0 = (z - d) \exp[(U_z \kappa)/u_] \tag{14.5}$$

How does it vary with wind direction?

A rule of thumb for calculating d is to assume it is $0.7h$ where h is the height of the canopy. As the heights may vary with direction you can determine how much this may vary. What are expected to be consistent sectors?

The wind profile can also be used to determine z_0 and d if there are more than 2 levels in the profile. This requires fitting a straight line (linear regression) through the data to determine the intercept, which provides the $z_0 + d$ value. See equations 1-2 in Grimmond et al. (1998)

14.3 LAI (Leaf Area Index)

- Varies with phenology (state of the vegetation e.g. leaf on or leaf off, bud burst).
- Measure of the area of leaf surface relative to the ground area. If you look up at an adult, healthy tree there will be typically many layers of leaves between the ground and the top of the canopy. Hence the LAI values are typically greater than 1 m^2 m^{-2}.
- Units: m^2 m^{-2}

Parameters LAI influences

- Albedo
- Roughness length and displacement
- Aerodynamic resistance
- Surface/ canopy resistance
- Surface interception of water

Porosity

If air can blow through an object (e.g. a tree) it is more porous than those one the that air can not (e.g. buildings). The latter are referred to as *bluff bodies*. Trees porosity may change through the year with phenology. For example, when deciduous plants lose their leaves (fall off) they become more porous.

Modelling of LAI

- Depends on:
  - temperature (growing degree days)

  - solar radiation/day length

Measurement of LAI

- Direct measurements
  - Destructive sampling - measuring the area of all the leaves of a tree
  - Collect all the leaves that fall off and measuring their area
- Indirect measurements
  - Measurements (above and below the canopy) of radiation (e.g. using LAI-2000, LI-COR) which are then used to infer LAI
  - Satellite based products e.g. MODIS

14.4 Stability

Modifications to the logarithmic profile are required in conditions of non-neutral stability, using the results of **Monin-Obukhov theory**. This theory of the surface layer derives relations between the vertical variation of wind speed u(z) and potential temperature $\theta(z)$ (which approximates the measured temperature T close to the surface), the scaling factors for momentum and temperature, u^* and T^*, and the **Monin-Obukhov stability parameter**

$$\frac{z'}{L} = -\frac{k\,(z-d)\,\frac{g}{\theta_0}\frac{H}{\rho c_p}}{u_*^3} \tag{14.6}$$

where L is the **Obukhov length** and $z^{'} = z - d$. NB: the surface temperature θ_0 is an absolute temperature (units: K). The logarithmic profile relation can be rewritten for wind speed to include the stability corrections

$$\bar{u}(z) = \frac{u_*}{k}\left[\ln\left(\frac{z-d}{z_0}\right) - \Psi_m\left(\frac{z-d}{L}\right) + \Psi_m\left(\frac{z_0}{L}\right)\right] \tag{14.7}$$

and similarly, for potential temperature:

$$\bar{\theta}(z) = \theta_0 + \frac{T_*}{k}\left[\ln\left(\frac{z-d}{z_h}\right) - \Psi_h\left(\frac{z-d}{L}\right) + \Psi_h\left(\frac{z_h}{L}\right)\right] \tag{14.8}$$

where the **turbulent temperature scale** T_* is given by $T_* = -\overline{w'T'}/u_* = -Q_H/(\rho C_p u_*)$, Ψ_m is the **integral stability correction function** for momentum and Ψ_h is the integral stability correction function for heat.

There are a number of forms of Ψ_m and Ψ_h; one set of forms from Foken (2008) are as follows:

under unstable conditions:

$$\begin{gathered}\psi_m(\zeta) = \ln\left[\left(\tfrac{1+x^2}{2}\right)\left(\tfrac{1+x}{2}\right)^2\right] - 2\tan^{-1}x + \tfrac{\pi}{2} \quad \text{for} \quad \tfrac{z}{L} < 0 \\ \psi_h(\zeta) = 2\ln\left(\tfrac{1+y}{2}\right) \quad \text{for} \quad \tfrac{z}{L} < 0\end{gathered} \tag{14.9}$$

with $x = (1 - 19.3\zeta)^{1/4} \quad y = 0.95(1 - 11.6\zeta)^{1/2}$.

under stable conditions:

$$\begin{gathered}\psi_m(\zeta) = -6\tfrac{z}{L} \quad \text{for} \quad \tfrac{z}{L} \geq 0 \\ \psi_h(\zeta) = -7.8\tfrac{z}{L} \quad \text{for} \quad \tfrac{z}{L} \geq 0\end{gathered} \tag{14.10}$$

Note that both T_* and z'/L have the opposite sign to Q_H (which is positive in unstable conditions and negative in stable conditions). If $z'/z_0 \gg 1$ then the third term can assumed to be negligible (Garratt 1992).

Other stability metrics include the Richardson number:

- Gradient
- Bulk
- Flux

Bulk Richardson number is the ratio of thermally produced turbulence and turbulence generated by vertical shear or the ratio of free or forced convection (thermal: mechanical)

$$R_B = \frac{(g/T_v)\,\Delta\theta_v \Delta z}{(\Delta U)^2 + (\Delta V)^2} \tag{14.11}$$

where g acceleration due to gravity, T_V virtual temperature, $\Delta\theta_v$ change (difference) in potential temperature, Δz change in height ΔU change in U wind-speed, and ΔV change in V wind-speed.

14.5 Wind Profile

Wind profile in neutral conditions

The variation of mean wind speed $\bar{u}$ with height z (in surface layer, above the RSL) above an aerodynamically rough surface can be represented by a logarithmic relation:

$$\bar{u}(z) = \frac{u_*}{k} \ln\left[\frac{(z-d)}{z_0}\right] \tag{14.12}$$

where u_* is the **friction velocity** ($u_*^2 = -\overline{u'w'}$) rate of vertical transfer by turbulence of horizontal momentum per unit mass of air), z_0 is the roughness length of the surface for momentum, d is the zero-plane displacement and k is von Karman's constant (0.4). The logarithmic law is strictly valid only in **neutral conditions**, i.e. when the effect of buoyancy on turbulence is small compared to the effect of wind shear. In such conditions, the temperature profile in the surface layer will be close to adiabatic (i.e. $dT/dz = \tilde{}9.8\,\mathrm{K\,km^{-1}}$ or potential temperature is constant with height). When the sensible heat flux is significantly different from zero, Monin-Obukhov theory (or other) must be used.

Wind and temperature profile in non-neutral conditions

Modifications to the logarithmic profile are required in conditions of non-neutral stability. Here we use **Monin-Obukhov theory** of the surface layer that derives relations between the vertical variation of wind speed u(z) and potential temperature $\theta(z)$ (which approximates the measured temperature T close to the surface), the scaling factors for momentum and temperature, u^* and T^*, and the **Monin-Obukhov stability parameter**

$$\frac{z'}{L} = -\frac{k\,(z-d)\,\frac{g}{\theta_0}\frac{H}{\rho c_p}}{u_*^3} \tag{14.13}$$

where L is the **Obukhov length** and $z' = z - d$. NB: the surface temperature θ_0 is an absolute temperature (units: K). The logarithmic profile relation can be rewritten for wind speed to include the stability corrections

$$\bar{u}(z) = \frac{u_*}{k}\left[\ln\left(\frac{z-d}{z_0}\right) - \Psi_m\left(\frac{z-d}{L}\right) + \Psi_m\left(\frac{z_0}{L}\right)\right] \tag{14.14}$$

and similarly, for potential temperature:

$$\bar{\theta}(z) = \theta_0 + \frac{T_*}{k}\left[\ln\left(\frac{z-d}{z_h}\right) - \Psi_h\left(\frac{z-d}{L}\right) + \Psi_h\left(\frac{z_h}{L}\right)\right] \tag{14.15}$$

where the **turbulent temperature scale** is given by $T_* = -\overline{w'T'}/u_* = -Q_H/(\rho c_p u_*)$, Ψ_m is the **integral stability correction function** for momentum and Ψ_h is the integral stability correction function for heat. Note that both T_* and z'/L have the opposite sign to Q_H (which is positive in unstable conditions and negative in stable conditions). If $z'/z \gg 1$ then the third term can assumed to be negligible (*Garratt 1994*)

Profiles and fluxes of moisture

Just as surface layer profiles of momentum and temperature follow a logarithmic form in neutral conditions, humidity in the surface layer has the same form, being transported by the same eddies. Thus, the profile is given by

$$\bar{q}(z) - \bar{q}_0 = \frac{q_*}{k}\ln\left(\frac{z-d}{z_q}\right) \tag{14.16}$$

where q is specific humidity, subscript 0 denotes a surface measurement, zq is the equivalent "roughness length" for humidity, and d is the zero-plane displacement height of a plant canopy over which measurements are being made). $q*$ is a scaling variable, defined as

$$q_* = \frac{-\overline{w'q'}}{u_*} \tag{14.17}$$

and thus the dimensionless moisture profile is defined as

$$\phi_q = \frac{k(z-d)}{q_*}\frac{\partial \bar{q}}{\partial z}. \tag{14.18}$$

The moisture flux can be written in various equivalent forms

$$E = \rho\overline{w'q'} = u_* q_* = -\rho K_q \frac{\partial \bar{q}}{\partial z} \tag{14.19}$$

where K_q is the eddy diffusivity for moisture. In neutral conditions it is assumed $K_m = K_h = K_q = k(z-d)u_*$. Moisture follows Monin-Obukhov similarity just as other scalar variables do. This was not established at the Kansas experiments due to limitations in the accuracy of the measurements.

14.6 General Micrometeorology Books

Arya, Pal S. Introduction to Micrometeorology. Volume 79. Elsevier, 2001. ISBN 9780120593545. doi:10.1016/s0074-6142(01)x8014-5. Rigorous, good all-round book

Arya, Pal S. Air Pollution Meteorology. Elsevier, 2002. ISBN 9781898563938. doi:10.1016/c2013-0-18095-6. Particularly surface-layer results

Aubinet, Marc, Vesala, Timo, and Papale, Dario. Eddy Covariance. Springer Netherlands, 2012. ISBN 9789400723504, 9789400723511. doi:10.1007/978-94-007-2351-1.

Bradley, Stuart. Atmospheric acoustic remote sensing: Principles and applications. CRC press, 2007.

Burba, G. Eddy covariance method for scientific, industrial, agricultural, and regulatory applications : A field book on measuring ecosystem gas exchange and areal emission rates. LI-COR Biosciences, 2013. ISBN 061576827X 9780615768274. Practical view of EC

Emeis, Stefan. Surface-Based Remote Sensing of the Atmospheric Boundary Layer. Volume 40. Springer Netherlands, 2011. ISBN 9789048193394, 9789048193400. doi:10.1007/978-90-481-9340-0. Good introduction to remote sensing principles & methods

Foken, Thomas. Micrometeorology. Volume 2. Springer Berlin Heidelberg, 2017. ISBN 9783642254390, 9783642254406. doi:10.1007/978-3-642-25440-6. Very good overview of Micrometeorology

Garratt, J. R. The atmospheric boundary layer. Cambridge University Press, 1994. ISBN 9780521467452. Rigorous mathematical treatment, lots of experimental data

Kaimal, J. C. and Finnigan, J. J. Atmospheric Boundary Layer Flows : Their Structure and Measurement. Oxford University Press, Incorporated, 1994. ISBN 9780195362770. Particularly good for flow over changing surfaces and obervational techniques. Chapters 2 and 6 are strongly recommended.

Leclerc, Monique Y. and Foken, Thomas. Footprints in Micrometeorology and Ecology. Volume 239. Springer Berlin Heidelberg, 2014. ISBN 9783642545443, 9783642545450. doi:10.1007/978-3-642-54545-0. Overview of footprint analysis

LeMone, Margaret A., Angevine, Wayne M., Bretherton, Christopher S., Chen, Fei, Dudhia, Jimy, Fedorovich, Evgeni, Katsaros, Kristina B., Lenschow, Donald H., Mahrt, Larry, Patton, Edward G., Sun, Jielun, Tjernström, Michael, and Weil, Jeffrey. 100 years of progress in boundary layer meteorology. Meteor. Monogr., 59():9.1-9.85, 2018. doi:10.1175/amsmonographs-d-18-0013.1.

Moene, Arnold F. and van Dam, Jos C. Transport in the Atmosphere-Vegetation-Soil Continuum. Cambridge University Press, 2013. ISBN 9781139043137. doi:10.1017/cbo9781139043137.

Oke, T. R. Boundary Layer Climates. Routledge, September 2002. ISBN 9780203407219. doi:10.4324/9780203407219. Good description of processes

Oke, T. R., Mills, G., Christen, A., and Voogt, J. A. Urban Climates. Cambridge University Press, 2017. ISBN 9781139016476. doi:10.1017/9781139016476. Comprehensive and readable

Stull, Roland B. An Introduction to Boundary Layer Meteorology. Volume 13. Springer Netherlands, 1988. ISBN 9789027727695, 9789400930278. doi:10.1007/978-94-009-3027-8. Comprehensive and readable

Wilczak, J. M., Gossard, E. E., Neff, W. D., and Eberhard, W. L. Ground-based remote sensing of the atmospheric boundary layer: 25 years of progress. In Garratt, J. R. and Taylor, P. A., editors, Boundary-Layer Meteorology

25th Anniversary Volume, 1970–1995, pages 321–349. Springer Netherlands, Dordrecht, 1996. doi:10.1007/978-94-017-0944-6_14.

Wyngaard, John C. Turbulence in the Atmosphere. Cambridge University Press, 2009. ISBN 9780511840524. doi:10.1017/cbo9780511840524. More advanced book on turbulence

CHAPTER

FIFTEEN

BACKGROUND TO MODEL EVALAUTION

There is a benchmark system for evaluating the model performance with observations and differences between model versions (for checking developments). The statistics used include some of the following:

15.1 Metrics for Model evaluation

Methods commonly used to evaluate model performance, include:

- Mean absolute error (MAE)

$$\mathrm{MAE} = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| \tag{15.1}$$

where N is number of observations, y_i the actual expected output and $\hat{y}_i$ the model's prediction (same notations below if not indicated otherwise).

- Mean bias error (MBE)

$$\mathrm{MBE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i) \tag{15.2}$$

- Mean square error (MSE)

$$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{15.3}$$

- Root mean square error (RMSE)

$$\mathrm{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2} \tag{15.4}$$

- Coefficient of determination (R^2)

$$\begin{aligned} R^2 &= 1 - \frac{\mathrm{MSE}(\ \mathtt{model}\ )}{\mathrm{MSE}(\ \mathtt{baseline}\ )} \\ \mathrm{MSE}(\ \mathtt{baseline}\ ) &= \frac{1}{N}\sum_{i=1}^{N}(y_i - \overline{y})^2 \end{aligned} \tag{15.5}$$

where $\overline{y}$ is mean of observed y_i.

Combined with plots (e.g. scatter, time series) allows identification of periods when a model performs well/poorly relative to observations. It should be remembered that both the model (e.g. parameters, forcing data) and the evaluation observations have errors.

CHAPTER

SIXTEEN

FAQ (FREQUENTLY ASKED QUESTIONS)

Here are common questions that appear throughout the workshop.

General

- **I have a Mac computer. Can I still participate in the workshop?** Yes, QGIS and UMEP are software independent but some of the tutorials are just for windows users. The tutorial on Lidar processing, section 4.4 is for Windows user only. Otherwise, all other activities is for all OS. Installation of UMEP can be a bit cjallenging on a Mac, but help instructions are included below (see below).

Daily shading

- **I can see shadow patterns during the sunshine hours which is logical. Is it also possible to get a complete dark profile duri** The animation is only for daytime when it comes to using the Shadow casting tool. This is because no data are created at night. You can have a look in the output folder and see the geotiff that are created.
- **On the 21. Dec there is not really a clear difference between Building Shadows and Building+Veg Shadows as there was or** Yes. You are in Sweden with very low sun elevations and 4-5 level buildings. That results in no vegetation shadows reaching the ground.

LIDAR processing

- **I cannot start the FUSION viewer in QGIS. What I have been doing wrong?** One common misstake is that FUSION is not installed on your computer. To use FUSION within QGIS yo first need to install the actual software form http://forsys.sefs.uw.edu/fusion/fusionlatest.html. Then you should install the plugin FUSION for processing and make the settings according to the tutorial instructions.

Software installation - Mac

- **SuPy couldn't be installed automatically on my Mac. What should I do?** The recommended version 2020.6.30 seems to have installation issues due to a third-party package that prevents installation. If unfortunately such issue happens, please manually install a development version of SuPy manually following these steps:

  1. Prepare utility functions in QGIS-python console by running the following:

```
import sys, subprocess, os
from pathlib import Path
import platform
```

(continues on next page)

(continued from previous page)

```
# locate QGIS-python interpreter
def locate_py():
    try:
        # non-Linux
        path_py = os.environ["PYTHONHOME"]
    except Exception:
        # Linux
        path_py = sys.executable

    # convert to Path for eaiser processing
    path_py = Path(path_py)

    # pre-defined paths for python executable
    dict_pybin = {
        "Darwin": path_py / "bin" / "python3",
        "Windows": path_py / "python3.exe",
        "Linux": path_py,
    }

    # python executable
    path_pybin = dict_pybin[platform.system()]

    if path_pybin.exists():
        return path_pybin
    else:
        raise RuntimeError("UMEP cannot locate the Python interpreter used by␣
↪QGIS!")

# install supy
def install_supy(ver=None):

    str_ver = f"=={ver}" if ver else ""
    try:
        path_pybin = locate_py()
        list_cmd = f"{str(path_pybin)} -m pip install supy{str_ver} -U --user
↪".split()
        list_info = subprocess.check_output(list_cmd, encoding="UTF8").split(
↪"\n")
        str_info = list_info[-2].strip()
        str_info = (
            str_info
            if len(str_info) > 0
            else f"supy{str_ver} has already been installed!"
        )
        return str_info
    except Exception:
        raise RuntimeError(f"UMEP couldn't install supy{str_ver}!") from␣
↪Exception
```

This code snippet should determine the actual path of Python interpreter QGIS is using.

2. install a development version of SuPy:

Also in the QGIS-python console, run this:

```
install_supy(ver='2020.7.8dev0')
```

3. restart your QGIS

Note: If this issue persists, please raise an issue in the UMEP repo and let Ting Sun know by @sunt05.

- **Jupyter notebooks CANNOT be launched? What should I do?** Please check the following in your command line tool (e.g., Terminal on macOS, OSGeo4W prompt on Windows given QGIS installed):

Note: if using OSGeo4W prompt, please run py3_env first to switch to your python3 environment.

1. Check if Jupyter notebook is installed:

```
python3 -m pip show notebook
```

if not, please install it:

```
python3 -m pip install notebook --user --upgrade
```

2. Jupyter notebook is installed but cannot be properly launched:

try to re-install it:

uninstall it first:

```
python3 -m pip uninstall notebook -y
```

then install it:

```
python3 -m pip install notebook --user --upgrade
```

Errors related to parameters derivation

- **I am getting errors related to pandas and SuPy, when running the notebooks for parameter derivation.** This can happen because of various reasons when running notebooks in this page; however, the most frequent reason is because Python, SuPy and pandas versions does not match the required ones in the yml file (see instructions in step 2 of this page). If you are using different version than the yml file, you need to upgrade/downgrade the rquired packages.

CHAPTER

SEVENTEEN

REFERENCES

- Allen L, F Lindberg, CSB Grimmond 2011: Global to city scale model for anthropogenic heat flux, *International J. of Climatology*, 31, 1990–2005, https://doi.org/10.1002/joc.2210
- Ao Xiangyu, CSB Grimmond, HC Ward, AM Gabey, Jianguo Tan, Xiuqun Yang, Dongwei Liu, Xing Zhi, Hongya Liu, Ning Zhang 2018: Evaluation of the Surface Urban Energy and Water balance Scheme (SUEWS) at a dense urban site in Shanghai: Sensitivity to anthropogenic heat and irrigation *J Hydrometeorology*, 19, 1983–2005, https://doi.org/10.1175/JHM-D-18-0057.1
- Capel-Timms I, ST Smith, T Sun, S Grimmond Dynamic Anthropogenic activitieS impacting Heat emissions (DASH v1.0): Development and evaluation. *In review*
- Cleugh HA and S Grimmond 2012: Urban Climates and Global Climate Change. in Henderson - Sellers A & Mc Guffie K (ed). *The Future of the World's Climate*. Elsevier Chapter 3, 47–76 + references, https://doi.org/10.1016/B978-0-12-386917-3.00003-8
- Gabey A, S Grimmond, I Capel-Timms 2019: Anthropogenic Heat Flux: advisable spatial resolutions when input data are scarce *Theoretical and Applied Climatology*, 135 (1–2), 791–807 https://doi.org/10.1007/s00704-018-2367-y
- Grimmond CSB, TR Oke 1991: An evaporation-interception model for urban areas. *Water Resources Research*, 27, 1739–1755. https://doi.org/10.1029/91WR00557
- Grimmond CSB, King T, Roth M. Oke T 1998: Aerodynamic Roughness of Urban Areas Derived from Wind Observations. *Boundary-Layer Meteorology* 89, 1–24 https://doi.org/10.1023/A:1001525622213
- Grimmond et al. 2020: SUEWS Documentation Release v2020a micromet@University of Reading https://suews.readthedocs.io/en/latest/#
- Iamarino M, Beevers S, CSB Grimmond 2012: High Resolution (Space, Time) Anthropogenic Heat Emissions: London 1970–2025 *International J. of Climatology*, 32, 1754–1767 https://doi.org/10.1002/joc.2390
- Järvi L, CSB Grimmond, A Christen 2011: The Surface Urban Energy and Water Balance Scheme (SUEWS): Evaluation in Vancouver and Los Angeles. *J. of Hydrology*, 411, 219–237 https://doi.org/10.1016/j.jhydrol.2011.10.001
- Lindberg F, CSB Grimmond, N Yogeswaran, S Kotthaus, L Allen 2013: Impact of city changes and weather on anthropogenic heat flux in Europe 1995–2015 *Urban Climate*, 4, 1–15 https://doi.org/10.1016/j.uclim.2013.03.002

- Lindberg F, Grimmond CSB, Gabey A, Huang B, Kent CW, Sun T, Theeuwes N, Järvi L, Ward H, Capel- Timms I, Chang YY, Jonsson P, Krave N, Liu D, Meyer D, Olofson F, Tan JG, Wästberg D, Xue L, Zhang Z 2018: Urban Multi-scale Environmental Predictor (UMEP) - An integrated tool for city-based climate services. *Environmental Modelling and Software*, 99, 70-87 https://doi.org/10.1016/j.envsoft.2017.09.020
- Lindberg F, T Sun, S Grimmond, Y Tang 2020: UMEP Manual Documentation 7 July 2020 https://umep-docs.readthedocs.io/en/latest/
- Oke TR, Mills G, Christen A, Voogt JA 2017: *Urban Climates*. Cambridge University Press https://doi.org/10.1017/9781139016476
- Omidvar, H., Sun, T., Grimmond, S., Bilesbach, D., Black, A., Chen, J., Duan, Z., Gao, Z., Iwata, H., and McFadden, J. P.: Surface [Urban] Energy and Water Balance Scheme (v2020a) in non-urban areas: developments, parameters and performance, Geosci. Model Dev. Discuss., https://doi.org/10.5194/gmd-2020-148, *in review*, 2020.
- Sun T, Grimmond CSB (2019) A Python-Enhanced Urban Land Surface Model SuPy (SUEWS in Python, v2019.2): Development, Deployment and Demonstration. *Geosci. Model Dev.*, 12, 2781-2795 https://doi.org/10.5194/gmd-12-2781-2019
- Sun T, H Omidvar, S Grimmond 2020: SuPy Documentation Release 2020.7.8dev Jul 10, 2020 https://supy.readthedocs.io/en/latest/#
- Ward HC, S Grimmond 2017: Using biophysical modelling to assess the impact of various scenarios on summertime urban climate across Greater London *Landscape and Urban Planning* 165, 142-161, https://doi.org/10.1016/j.landurbplan.2017.04.001
- Ward HC. S Kotthaus, L Järvi, CSB Grimmond 2016: Surface Urban Energy and Water Balance Scheme (SUEWS): development and evaluation at two UK sites *Urban Climate* 18, 1-32 https://doi.org/10.1016/j.uclim.2016.05.001

INDEX