Australian Government
Bureau of Meteorology

AWRA Landscape Community Modelling System User Guide

A guide to setting up and running the AWRA Landscape Community Modelling System (AWRA-CMS)

Avijeet Ramchurn, Andrew Frost, Chantal Donnelly, Stuart Baron-Hay, Alison Oke, Wendy Sharples and David Wright

Revision history

Date	Version	Description	Author
22/9/2016	V1	Final User Guide	Alison Oke
29/08/2017	Draft V2.0	Working draft	Avijeet Ramchurn
22/11/2017	Draft V2.1	Draft for MODISM Training	Alison Oke
21/12/2017	V2.6	Full draft of Guide	Alison Oke
17/1/2018	V2.7	Cleaned draft of guide	Chantal Donnelly
14/02/2018	V2.8	Final clean up and check	Wendy Sharples
08/03/2018	V2.9	Final check	David Wright

Review Status

Reviewer	Date Reviewed	Version Reviewed

Release Signatories

Approval	Name	Signature	Date

Contact details

Dr. Chantal Donnelly

Manager - Water Resources Modelling Unit

Bureau of Meteorology

GPO Box 1289 MELBOURNE VIC 3001

Phone: 0732398767
Email: awracms@bom.gov.au

Dr. Amgad Elmahdi

Section Head, Water Resources Assessment (WRA)

Bureau of Meteorology

GPO Box 1289 MELBOURNE VIC 3001

Phone: +61 3 8638 8274
Email: Amgad.Elmahdi@bom.gov.au

CONTENTS

1. Introduction 1
1.1. Background 1
1.2. The AWRA Community Modelling System (AWRA-CMS) 2
1.2.1. Purpose 2
1.3. The AWRA-CMS package 2
1.4. Input and test datasets 4
1.5. User registration 5
1.5.1. Licensing 5
1.6. Understanding this User Guide 6
1.6.1. Purpose 6
2. Installing AWRA-CMS 7
2.1. System overview 7
2.2. System Setup 7
2.2.1. Download the AWRA-CMS 8
2.2.2. Install conda and build the AWRA-CMS conda environment 8
2.2.3. Activate the AWRA-CMS conda environment 9
2.2.4. Install AWRA-CMS into the conda environment 9
2.2.5. For Windows: Interface with the Windows SubSystem for Linux (WSL) 9
2.2.6. For Linux: Interface 9
2.3. Getting started with AWRA-CMS 9
2.3.1. Directories 9
2.3.2. User Manual Notebooks 10
2.3.3. Test Input Data 10
3. AWRA-CMS concepts and classes 11
3.1. Configuring AWRA-L simulations and calibrations 11
3.2. Loading the AWRA-L model 11
3.3. Model inputs 12
3.4. Model configuration and the nodegraph 13
3.4.1. Nodegraphs 13
3.4.2. Default Input Mapping 14
3.4.3. Input daily climate (forcing) data: 14
3.4.4. Climate data supplied with the AWRA-CMS 15
3.4.5. Input starting states 16
3.4.6. Output configuration 16
3.5. Defining the spatial extent of the model to be run 17

3.6. Period specification 18
4. Running a model simulation 20
4.1. On-Demand Simulation 20
4.1.1. Import required libraries 20
4.1.2. Modifying the model configuration 20
4.1.3. Put the model run specification together 25
4.1.4. Exploring model outputs 27
4.1.6. Save outputs to file 30
4.1.7. Quickly viewing model run outputs held in memory 30
4.1.8. More examples 33
4.1.9. Run with a uniform rain input across the country 34
4.2. Basic Server Simulation 35
4.2.1. Import required libraries and set data paths 35
4.2.2. Modify the model configuration 35
4.2.3. Put the model run specification together 37
4.2.4. Visualise outputs 37
4.2.5. How to change the initial states 38
4.2.6. Fill gaps in forcing data with climatology 40
5. Visualising AWRA-L inputs and outputs as maps and time-series 42
5.1. Setup 42
5.1.1. Import required modules 42
5.1.2. Load inputs and results 42
5.1.3. Inspect the variables present in results 43
5.1.4. Spatial plots for different data slices 43
5.1.5. Display all variables, for a single day, at the default continental extent 43
5.1.6. Aggregate over a month for a specified region 44
5.1.7. Change aggregation method from default of mean to sum 44
5.1.8. Accessing the underlying data 45
5.1.9. Specifying a catchment extent 45
5.1.10. Changing the format of the plots 48
5.2. Time-series plots for selected variables, locations or regions 49
5.2.1. Plotting a time-series for a single location 49
5.2.2. Plotting a time-series of aggregated values over a catchment 50
5.2.3. Formatting time-series graphs 51
5.2.4. More formatting options 51
6. Extract point or catchment average data from AWRA-L grids 53

7. Calibration 56
7.1. Import required libraries 56
7.2. Calibration configuration 57
7.2.1. Specify the calibration sites 58
7.2.2. Specify running and calibration period 59
7.2.3. Amending the input nodegraph 59
7.2.4. Specifying the optimiser options 60
7.2.5. Defining the objective function 61
7.2.6. Build specifications dictionary for the calibration 63
7.3. Run the calibration 64
7.4. Extract best parameter set 65
7.5. Visualise the calibration outputs 66
7.6. Comparing modelled outputs to observations 68
8. Benchmarking 70
8.1. Comparing AWRA-L Input data with data from other sources 71
8.1.1. Observation data format 71
8.1.2. AWRA-L data 71
8.2. Benchmarking Example: Qtot and Streamflow Observations 71
8.2.1. Import required libraries 71
8.2.2. Set up benchmarking configuration 71
8.2.3. Create the benchmark object 72
8.2.4. Add models to be benchmarked 73
8.3. View the statistics tables 74
8.4. View the statistic plots [time-series, regression, cumulative exceedance] 77
8.5. Statistics plotting 78
9. References 80
Appendix A. AWRA-L model overview 81
Appendix B. AWRA-L parameters 83
Appendix C. Python version of AWRA-L 87
Appendix D. Contributing to AWRA-CMS 88
Appendix E. AWRAMS Licence 89
Appendix F. AWRAMS Contributor Licence 92

1. Introduction

This Chapter provides background to and introduces the Australian Water Resource Assessment Community Modelling Systems (AWRA -CMS) and User Guide.

1.1. Background

Water availability across the Australian continent is limited by varying climate zones and high year-to-year variability. The Australian Government through the Commonwealth Water Act 2007, has given the Bureau of Meteorology (the Bureau), responsibility for compiling and delivering comprehensive water information for Australia. Specifically, to collate information and assess and report on the availability, condition and use of water resources in Australia. To fulfil its legislative responsibilities, the Bureau requires a water balance modelling system that quantifies water flux and storage terms and their respective uncertainties using a combination of data sets (on-ground metering, remotely sensed data) and modelled outputs. The system needs to be applicable across the continent and flexible enough to be able to use all available data sources (when modelling data rich and data limited regions) to provide nationally consistent and robust estimates (Hafeez et al., 2015).

The Australian Water Resources Assessment modelling system (AWRAMS) has been developed through the Water Information R&D Alliance (WIRADA) between the Bureau of Meteorology (the Bureau) and CSIRO since 2008 – towards fulfilling the Bureau's water reporting requirements as specified in the Water Act (2007). AWRAMS has been developed and tested to provide continental-to-regional scale water balance estimates for water resources assessment and water accounting purposes. The science underpinning AWRAMS has now reached maturity and been implemented as an operational system within the Bureau. AWRAMS ongoing development is being supported into the future by the Bureau and CSIRO.

The AWRAMS (Hafeez et al., 2015) has two modelling components:

- AWRA-L to estimate landscape water balance fluxes, and
- AWRA-R to estimate river water balance fluxes

To enable further development and other uses of the system, the AWRA-L component has been bundled into a framework for pre and post-processing, calibration and benchmarking to the Australian context and has been released to the community as the AWRA-CMS (Community Modelling System) package. This document describes the AWRA-CMS.

1.2. The AWRA Community Modelling System (AWRA-CMS)

1.2.1. Purpose

Based on active engagement with a wide range of stakeholders, it is apparent that users have different requirements for applications of AWRA-L modelled ranging from water resources assessment and planning to various other sectors (including agriculture, natural resources management, flood and groundwater) at continental to regional and catchment scales. The release of the AWRA MS as a community model provides the best means to engage the most qualified experts in each discipline in the process of refining and applying AWRA-L to these modelling applications.

1.3. The AWRA-CMS package

The AWRA-CMS is presented as a publicly available python package called **awrams**. The package is designed to provide easy functionality to use, analyse and modify the AWRA-L model code and its inputs and outputs. While the model kernel code is kept in C language, everything else is in Python. The AWRA-CMS is currently limited to Landscape (AWRA-L) model and related functionality.

The package is purpose built using open-source software such as Python, C to facilitate adoption. Model content and functionality are divided into a number of linked components, shown conceptually in Figure 2. These components are expanded on below:

1. awrams.models - The AWRA-L model code

The current version of the operational model (AWRA-L v5) is the default hydrological model (code and parameters) provided as part of the package. For the purposes of speed and efficiency, the model's spatio-temporal loops are coded in the C-language with Python used to provide the supporting architecture to feed in user-specified input and run settings. Advanced users are able to modify the model code or add other models.

2. awrams.simulation- Simulation tools to run the model

The simulation component contains the required functionality to interact with the run settings of the model to specify its extent, period, and outputs. The inputs and outputs to the model are user configurable. Notably, the model can be run in two modes:

On-demand, i.e. keeping the inputs and outputs stored in memory for fast interactive use,

Server, i.e. writes output directly to disk and is used for spatio-temporally large runs (producing data larger than available memory).

3. awrams.calibration – Tools to calibrate the parameters of the model

This component contains tools for calibration and evaluation of sensitivity to model parameters. It contains the code to calibrate the AWRA-L model with the Shuffled Complex Evolution (SCE-UA) algorithm (Duan et al.,1992). Calibration can be

undertaken on a local computer or on a supercomputer (if computational time becomes critical when applying over large spatial and temporal scales, using multiple processors). The user can specify the calibration objective functions and calibrate to multiple observations (and observation types) simultaneously.

The Bureau has made use of this capability to fit the AWRA-L v5 model simultaneously to streamflow (specific runoff), soil moisture and evapotranspiration (ET) to approximately 300 unimpaired catchments from across Australia with a further 300 catchments reserved for validation.

4. awrams.benchmarking – Benchmarking model outputs against observations

The benchmarking functionality facilitates the repeatable comparison of AWRA-L model outputs against observations of landscape water stores and fluxes. The idea is that the benchmarking can be used to compare old and new model versions. The package generates key model-fit statistics (for example Nash-Sutcliffe Efficiency, relative bias and correlation) and plots (i.e. time-series, scatterplots, box-whisker plots, and CDF) of the outputs (i.e. catchment aggregated gridded runoff, soil moisture, ET and deep drainage) which can be viewed graphically or in tables.

5. awrams.visualisation – Visualisation of inputs and outputs

The visualisation tools enable the visual inspection of AWRA-L model inputs and outputs both spatially and temporally, and comparison of different sets of results from the model. The package allows the plotting of maps or time-series for any specified region such as Australia, a state, a catchment, a bounding box, or a pixel. The package gives the user spatial and temporal aggregation abilities, including the aggregation of gridded model inputs and outputs over ad-hoc polygon areas such as catchments.

6. awrams.utils – Utilities to support the system

The utils component contains functionality that is required in more than one of the above packages. It contains support code for file-handling, geospatial operations and data manipulation such as fast extraction from gridded outputs; and grid time-series conversion.

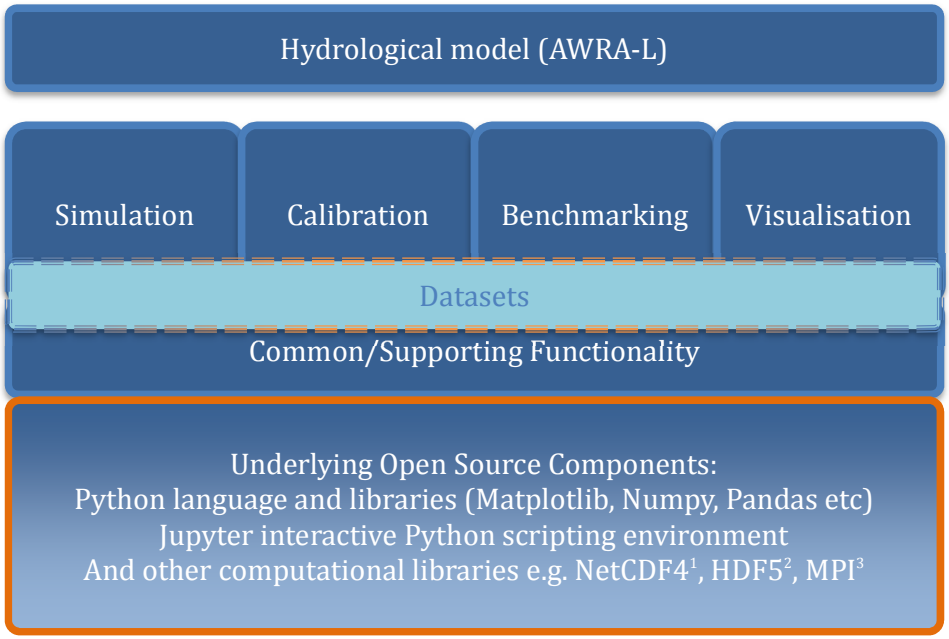


Figure 1. Components of the Community Modelling System

1.4. Input and test datasets

A set of sample datasets are also provided with the CMS package for testing purposes. A fuller set of data is made available to registered users of the AWRA-CMS (see Section 1.5). The datasets contain:

1. Forcing inputs,
2. Spatial parameters,
3. Observed data for calibration and benchmarking.

These datasets were used during the AWRA-L development and are set up within the package for model calibration and benchmarking purposes. Users must also abide by individual dataset licences contained with the data. Details of the data sources are as follows:

- streamflow from unimpaired catchments across Australia (Zhang et al., 2013),
- remotely sensed catchment averaged soil moisture (AMSR-E; Owe et al., 2008),
- remotely sensed, catchment averaged ET derived from MODIS satellite observations (CMRSET; Guerschman et al., 2009),
- sample observations of in-situ soil moisture from two Australian networks in the Upper Hunter (SASMAS; Rüdiger et al., 2007) and Murrumbidgee (OzNet ; Smith et al., 2012) and

- gap filled site ET from an Australian flux tower network (OzFlux; Beringer, Hutley et al., 2016; infilled according to the processing outlined in Beringer, McHugh et al., 2016).

1.5. User registration

There are two types of Users with the AWRA-CMS: ***Users*** and ***Registered Users***. Any ***User*** can access the AWRA-CMS package and documentation via GitHub, and are bound by the modified BSD license agreement (See Section 1.4.1). The downloaded data consists of:

- User Guide (this document),
- Technical Model Description Report (Frost, Ramchurn and Smith, 2016),
- Benchmarking Report (Frost, Ramchurn and Hafeez, 2016),
- Limited observation and forcing data for calibration and benchmarking.

However, to gain access to the evaluation datasets a user must become an AWRA-CMS ***Registered User***. Reasons to become an AWRA-CMS ***Registered User*** are:

- Access to complete calibration datasets (catchment-average timeseries),
- Access to complete benchmarking datasets (catchment-average timeseries),
- Solar radiation "climatology" gridded dataset for input prior to 1990,
- Allowing the user to create a supported fork of github repository further allowing the opportunity to contribute to the ongoing development of the AWRA-CMS code (Appendix E),
- Ongoing support and updates to package datasets.

To become a Registered User send an email to awracms@bom.gov.au and the Bureau will send you the licence agreement and instructions on how to access the datasets. At this point information on using the GitHub repository and how Bureau will manage updating the master copy of the AWRA-CMS repository will be provided.

1.5.1. Licensing

The Bureau has chosen the Berkeley Software Distribution (BSD) license modified according to the Bureau's legal advice and requirements (see Appendix C), including restricting commercial use of the modelling system. The adoption of a modified BSD-based license is expected to simplify the adoption decision for many potential users as it is simple, well-known and open source. Similarly, the Bureau has opted for the Contributor License Agreement (CLA; see Appendix D) for users who wish to submit enhancements to the modelling system they have authored to the Bureau. The CLA will either a) require the contributor to assign copyright in the contribution to the lead organisation, or b) require the contributor to license the lead organisation to use the contribution, including the right to on-license the contribution.

[1] NetCDF4 – network common data format
[2] HDF5 – file format and software library
[3] MPI – message passing interface

1.6. Understanding this User Guide

1.6.1. Purpose

The remainder of this User Guide describes how you set yourself up to use AWRA-CMS, and provides detailed guidance on how to undertake common hydrological modelling tasks (i.e. simulation, calibration, benchmarking, visualisation, and data extraction) using the tools provided.

This User Guide takes the approach of prioritising the description of common hydrological modelling workflows for each of the main capabilities of the package rather than explaining all the commands available in each of the package. The material is thus aimed at educating the user about the essential function calls needed to achieve typical hydrological modelling tasks, and the user is encouraged build their understanding of the package capabilities through both accessing the help available and exploring the package code. Documentation of the package (using docstrings) is ongoing so users are encouraged to update the package regularly.

Note that scientific description of the model is not given here, this is provided in the Bureau's Operational AWRA-L Model technical description document (Frost, Ramchurn and Smith, 2016). This document is accessible from the following URL: http://www.bom.gov.au/water/landscape/static/publications/Frost__Model_Description_Report.pdf.

2. Installing AWRA-CMS

Using an open-source Python-based Jupiter notebook solution the AWRA-CMS offers the flexibility that the system can be installed on various **Linux** operating systems, **OS X** (Apple Macintosh Unix based operating systems) and Windows Subsystem for Linux (a virtual Linux environment which can be setup on a windows 10 machine). The simplest mechanism to install AWRA-CMS is by utilising a conda environment. This section describes the steps to setup a conda environment and install the required python packages into the environment.

2.1. System overview

The Python programming language (Python 3 and Jupyter notebook) underpins the AWRA-CMS. Python is a widely used, platform independent programming language. A notebook is an interactive document containing code, text, and other elements. Jupyter notebooks provide an interface with which you can easily interact with the python code. It is recommended that the user complete some tutorials and familiarise themselves with the basics of Python before setting up the AWRA-CMS (e.g. see Appendix F).

The AWRA-L model kernel is coded in C. C was chosen to ensure the fastest computation time for elements of the code (i.e. the models) that get run repeatedly at each time step. The user is not expected to know C programming for routine model interaction. Knowledge of these languages is however necessary if the user wishes to alter internal model processes.

The user interface to the modelling system departs from a traditional rigid GUI template, and instead makes use of the Jupyter Notebooks (Learn more about notebooks at: http://jupyter.org/) to facilitate experimentation with code, user-driven data analysis and visualisation of outputs.

2.2. System Setup

The modelling system has been developed, tested and used on a **Linux OS**. Installation has been tested on several flavours of **Linux** (latest versions of Ubuntu, Fedora, Centos, Scientific Linux, Debian, Mint and “Bash on Ubuntu on Windows”).

The modelling system is designed to be run with Python 3 (specifically has been developed and tested with v3.4.5) and relies on the following libraries and packages:

- C compiler to build the core model code,
- NetCDF library (version 4.3, NOTE: 4.4 creates files that are unreadable with *h5py*) and Python bindings (*netCDF4*),
- HDF5 library (Tested on 1.8 and up) and Python bindings (*h5py*),
- Python *numpy* and *pandas* packages.

- IPython/Jupyter notebook recommended for the interactive use of the modelling system
- and various Python packages including:
    - *cffi* – for building the model Python bindings,
    - *pyzmq* – for inter-process communication,
    - *matplotlib* – for image and graph display,
    - *nose* – for running tests.

Certain spatial analysis functions also rely on the osgeo/GDAL libraries and corresponding Python bindings. To use ESRI-type shapefiles, GDAL needs to be installed. However, this is optional, as the simulation package will work without it.

The following sections give guidance for setting up a conda environment and installing the AWRA-CMS.

2.2.1. Download the AWRA-CMS

The AWRA-CMS can be downloaded from github as a compressed archive or by cloning the repository.

```
# download a zip file from github
wget https://github.com/awracms/awra_cms/archive/master.zip
unzip master.zip
cd awra_cms-master

# OR download a tarball from github
wget https://github.com/awracms/awra_cms/archive/master.tar.gz
tar zxf master.tar.gz
cd awra_cms-master

# OR clone the repository on github
git clone https://github.com/awracms/awra_cms.git
cd awra_cms
```

2.2.2. Install conda and build the AWRA-CMS conda environment

A conda package list is contained in the AWRA-CMS repository (conda_install_env.lst). Building an environment from this list ensures all package versions are compatible with the AWRA-CMS.

```
wget https://repo.continuum.io/miniconda/Miniconda3-latest-Linux-x86_64.sh
chmod +x Miniconda3-latest-Linux-x86_64.sh
./Miniconda3-latest-Linux-x86_64.sh
```

```
conda create -n awra-cms --file conda_install_env.lst
```

2.2.3. Activate the AWRA-CMS conda environment

To use the AWRA-CMS conda environment in a bash shell it must be activated first.

```
source activate awra-cms
```

2.2.4. Install AWRA-CMS into the conda environment

With the AWRA-CMS conda environment activated you will then need to install the AWRA-CMS packages. A shell script is provided with the code bundle to perform the following steps (install-awra-cms.sh). Execute the shell script: ./install-awra-cms.sh.

2.2.5. For Windows: Interface with the Windows SubSystem for Linux (WSL)

AWRA-CMS has been successfully tested on WSL. WSL will run on a 64-bit version of Windows 10 Anniversary Update build 14393 or later. To enable WSL, follow the instructions at: https://msdn.microsoft.com/en-au/commandline/wsl/install_guide.

Then open a command prompt and type *bash* and complete the installation process as above. To start a notebook server, type at the bash command prompt

```
jupyter notebook --no-browser --ip='*'
```

then open a browser and point to the ip address Jupyter returns, such as: http://localhost:8888/notebooks, where localhost is the server name and 8888 is the port.

2.2.6. For Linux: Interface

Following successful installation, Jupyter Notebooks are initiated from the command line prompt using the same command:

```
jupyter notebook --no-browser --ip='*'
```

2.3. Getting started with AWRA-CMS

2.3.1. Directories

The directories of the package are structured around the five major components of the AWRA-CMS Package:

- Simulation,
- Visualisation,
- Calibration,
- Benchmarking,
- Utilities.

Each of these components consists of multiple modules that define and run these sections of the AWRA-CMS.

2.3.2. User Manual Notebooks

The five AWRA-CMS components each have at least one Jupyter Notebook that demonstrates the functionality of the package and aligns with the example code presented in this User Guide. These notebooks are located in the 'userman' folder under the root directory of the package. It is recommended that the user opens and uses these userman Notebooks to get started with the AWRA-CMS.

2.3.3. Test Input Data

Each of the five AWRA-CMS components has a set of test data associated with it that provide an example of the input data format and feeds into the notebooks to demonstrate package functionality. The data is located under test_data in the top-level directory. To gain access to the full datasets become a AWRA-CMS Registered User (see Section 1.5) or an AWAP registered user for the full forcing data sets. .

3. AWRA-CMS concepts and classes

This section introduces the essential CMS concepts and shows how to configure the model.

3.1. Configuring AWRA-L simulations and calibrations

Setting up a simulation or calibration run starts by defining:

1. The **model** (in this case AWRA-L),
2. The inputs and outputs via the nodegraph (the nodegraph is a dictionary used to configure a model run):
    a. Inputs:
        i. the forcing inputs (rain, temperature, radiation),
        ii. the spatial inputs (grids of various landscape properties),
        iii. model parameters (calibrated or fixed).
    b. Outputs:
        i. the model outputs to be written out (e.g. flow, evaporation, soil moisture),
        ii. the grid resolution (default is 0.05 degree).
3. The spatial **extent,**
4. The modelling time **period.**

3.2. Loading the AWRA-L model

The AWRA-CMS has a generic way of defining models, i.e. all models (objects of 'Model' Class[4]) will conform to a certain way of being defined and have predetermined properties and methods. The *Model* module (awrams.models.model.Model) is specified within the python code at awrams/models/model.py. A few notes on this module:

- This generic model definition (class) provides an interface for configuring and using models.
- All models must derive from this class.
- Models conforming to this interface specification can be simulated and calibrated through AWRA-CMS functionality.
- A model definition needs to provide information about itself:
    - names of inputs, parameters, outputs
    - complete methods for how the various inputs and parameters are produced or calculated from the raw data available (input data pre-processing)

The AWRA-L model is the only model currently set up within the system. The user can inspect the components of its definition at: path_to_awracms/awra_cm/awrams/models/awral.

[4] Learn about python 'Class' at: https://www.learnpython.org/en/Classes_and_Objects

Note that this file is within the awrams package file within your conda environment install. If the user wants to create their own model module, the user is advised to create a copy of the folder in a working location and point to that version.

```
## A. Load the awral model class from the models module
from awrams.models import awral
# see the components of the awral class by tab completion (model, runner, settings, support, solar, template, transforms)
e.g. output for  awral.model below.
awral.model
```

```
# Create a working instance of the the AWRA-L model called awralmod. Multiple awral models can be created with differ ent configurations if required.
awralmod = awral.model.AWRALModel()
```

The AWRA-L model instanced (awralmod) comes with the default settings associated with the model, i.e. for parameters, spatial grids, climate inputs, and outputs to be produced from a run.

3.3. Model inputs

The model inputs can be divided into three types

- Forcing inputs: These are the daily varying climate inputs, e.g. rain, maximum and minimum temperature and solar radiation are provided as netcdf (*.nc) files.
- Spatial grids: static soil, vegetation and topography related spatial datasets are stored in a single HDF5 file **spatial_parameters.h5.** https://en.wikipedia.org/wiki/Hierarchical_Data_Format. Currently these are static spatial grids.
- Calibratable model parameters: These are stored in a *.json format file: **DefaultParameters.json** https://en.wikipedia.org/wiki/JSON

Typically, constants, non-gridded variables and non-forcing arguments to functions are variables defined in the parameter set. The user can modify parameter values at source (i.e. in the file) or on the fly within a notebook prior to setting off the model run.

```
# A list of the complete set of model inputs is available in INPUT_JSON_FILE=data/model_inputs.json file
import json
import pandas as pd
model_inputs = json.load(open(awral.settings.INPUT_JSON_FILE,'r'))
model_inputs
```

```
{'INPUTS_CELL': {'avpt': 'Vapour pressure',
  'k0sat': 'Hydraulic saturation (top)',
  'k_gw': 'Groundwater drainage coefficient',
  'k_rout': 'k_rout',
  'kdsat': 'Hydraulic saturation (deep)',
  'kr_0s': 'Interlayer saturation (top/shallow)',
  'kr_coeff': 'kr_coeff',
  'kr_sd': 'Interlayer saturation (shallow/deep)',
```

```
 'kssat': 'Hydraulic saturation (shallow)',
 'pair': 'Air pressure (pA)',
 'prefr': 'prefr',
 'pt': 'Precipitation (mm/day)',
 'radcskyt': 'radcskyt',
 'rgt': 'Solar radiation (MJ m^-2)',
 's0max': 'Maximum soil moisture (top)',
 'sdmax': 'Maximum soil moisture (deep)',
 'slope': 'slope',
 'slope_coeff': 'slope_coeff',
 'ssmax': 'Maximum soil moisture (shallow)',
 'tat': 'Temperature average (degC)',
 'u2t': 'Windspeed (m/s)'},
'INPUTS_HRU': {'alb_dry': 'Dry Soil Albedo',
 'alb_wet': 'Wet Soil Albedo',
 'cgsmax': 'Conversion Coefficient From Vegetation Photosynthetic Capacity Index to
    Maximum Stomatal Conductance',
 'er_frac_ref': 'Ratio of Average Evaporation Rate Over Average Rainfall Intensity
   During Storms Per Unit Canopy Cover',
 'fhru': 'Fraction of Simulation Cell with <HRU-type> Vegetation',
 'fsoilemax': 'Soil Evaporation Scaling Factor When Soil Water Supply is Not Limiti
   ng Evaporation',
 'hveg': 'Height of Vegetation Canopy',
 'laimax': 'Maximum Leaf Area Index',
 'lairef': 'Reference Leaf Area Index (at which fveg = 0.63)',
 'rd': 'Root depth',
 's_sls': 'Specific Canopy Rainfall Storage Capacity Per Unit Leaf Area',
 'sla': 'Specific Leaf Area',
 'tgrow': 'Characteristic Time Scale for Vegetation Growth Towards Equilibrium',
 'tsenc': 'Characteristic Time Scale for Vegetation Senescence Towards Equilibrium'
   ,
 'ud0': 'Maximum Root Water Uptake Rates From Deep Soil',
 'us0': 'Maximum Root Water Uptake Rates From Shallow Soil',
 'vc': 'Vegetation Photosynthetic Capacity Index Per Unit Canopy Cover',
 'w0lime': 'Relative Top Soil Water Content at Which Evaporation is Reduced',
 'w0ref_alb': 'Reference Value of w0 Determining the Rate of Albedo Decrease With W
   etness',
 'wdlimu': 'Deep Water-Limiting Relative Water Content',
 'wslimu': 'Shallow Water-Limiting Relative Water Content'},
'INPUTS_HYPSO': {'height': 'height', 'hypsperc': 'hypsperc', 'ne': 'ne'},
'STATES_CELL': {'sg': 'Groundwater storage (mm)',
 'sr': 'Surface storage (mm)'},
'STATES_HRU': {'mleaf': 'Vegetation index',
 's0': 'Top soil moisture (mm)',
 'sd': 'Deep soil moisture (mm)',
 'ss': 'Shallow soil moisture (mm)'}}
```

3.4. Model configuration and the nodegraph

In the AWRA-CMS, loading up the default model (AWRA-L) brings up the default configuration of how all the model input/parameters are combined and what outputs are produced. How these come together for the model run (with any transformation or scaling) is dependent on a couple of python scripts found in the models folder: i.e. description.py and transforms.py. The configuration is organised by way of a mapping system, which links a variable used by the system to the mechanism by which it is generated. This mapping system, which ultimately defines the data flow from raw input to model parameter, is presented to the model in a python dictionary (each parameter is a key and the transforms that apply is the value) used to describe the data-flow.

3.4.1. Nodegraphs

In AWRA-CMS, this mapping dictionary can be called up and edited through '**nodegraph**' functionality, essentially linking (*graph*ing) each piece of data processing (*node*) needed by the system and transforming the configuration information into runnable code, telling the model how to configure the various components required for/by the run. The nodegraph is devised

as a python class, and hence comes with certain attributes and applicable methods that facilitate input and output manipulation (awrams.utils.nodegraph).

At runtime, the **input nodegraph** controls the loading of files and any infilling and unit conversion at run time (in memory), hence it provides flexibility in the sense that inputs or outputs can be defined through functions and arguments within a user notebook just prior to a run. It can be customised in many ways to reflect changes the user wants to make to the inputs of the AWRA-L model. These changes can include anything from changing the path to the forcing data to defining the initial states to be used in the simulation.

For example, AWRA-L model uses a single temperature input, but AWAP supplies two (minimum and maximum temperature). The default AWRA-L input mapping loads the AWAP inputs then rectifies and computes a weighted average of these, before passing the single value on to the core model code. Before setting off a model run, the user could modify the calculation of the weighted average by modifying the nodegraph.

Similarly, the **output nodegraph** defines which variables are to be output where, and any transformations to be completed upon output.

3.4.2. Default Input Mapping

The AWRA-CMS comes with default settings for the AWRA-L model which contains paths to the test data provided and then applies the conditions and parameter sets in AWRA-L V5. The default settings are converted to a mapping (or nodegraph) upon loading up the model, providing a starting point for understanding how the AWRA-L model works in the AWRA-CMS. As highlighted earlier, this nodegraph can be edited within a notebook to apply different parameters set, spatial input files or transformations and gap filling to the forcing data. A user can view the output mapping in the same way as the input mapping (Section 3.3)

```
# Convenience functions in the awral.utils.nodegraph module allow you to inspect what goes into creating a parameter
from awrams.utils.nodegraph import nodes, graph
default_config = awralmod.get_default_mapping()
graph.get_input_tree(['k_rout'], default_config)
```

3.4.3. Input daily climate (forcing) data:

Daily climate data required for running the AWRA-L model includes:

- Rainfall,
- Temperature (an average of maximum and minimum values),
- Solar radiation.

The Bureau of Meteorology uses 0.05 degree gridded daily data based on 9am-9am observations and satellite data across Australia; see http://www.bom.gov.au/jsp/awap/. This data covers 1911 until yesterday, with the exception of solar radiation based on satellite data, where climatologies are used prior to 1990. You can use your own data provided you can format it into netcdf input file format, preferably at the desired grid resolution, however there are simple upscaling and downscaling functions built-in for user convenience.

The input climate data and default base path is specified within models/settings.py according to the following pattern:

- *{input variable name for the model: (path, netcdf variable name)},*

For example: {rain_day: ./rain_day/, rain*}

The current input file settings can be seen by looking at awral.model.FORCING and awral.model.CLIMATE_DATA

```
# climate forcing as specified in settings.py
awral.model.FORCING
```

```
{'precip': ('rain_day/rain*', 'rain_day'),
 'solar': ('solar_exposure_day/solar*', 'solar_exposure_day'),
 'tmax': ('temp_max_day/temp_max*', 'temp_max_day'),
 'tmin': ('temp_min_day/temp_min*', 'temp_min_day')}
```

```
# climate data path as specified in
awral.model.CLIMATE_DATA
```

```
'path_to_awracms/awra_cm/test_data/climate/BOM_climate/'
```

The nodegraph functionality can be used to extract and modify this information without looking at the source code. The example below shows how to view the input configuration for the rainfall forcing input.

```
# the following path defines where the precipitation forcing data is located
graph.get_input_tree(['precip_f'], default_config)
```

```
{'precip_f': forcing_from_ncfiles([]):{'path': '/data/cwd_awra_data/AWRACMS/test_dat
    a/climate/BOM_climate/', 'cache': False, 'nc_var': 'rain_day', 'pattern': 'rain
    _day/rain*'}}
```

3.4.4. Climate data supplied with the AWRA-CMS

The AWRA-CMS package contains limited coverage sample input climate and observational test data in path_to_awracms/awrams_cm/test_data/simulation. This data is arranged according to how it is used in the package.

Simulation:

The following data is provided in national 0.05 degree grids:

- rainfall: rain_day_2000/2001.nc and climatology_daily_rain_day.nc,
- temperature: temp_max/min_2000/2001.nc and climatology_daily_temp_min/max_day.nc,
- solar radiation: solar_exposure_2000/2001.nc and climatology_daily_temp_min/max_day.nc.

(where climatologies are the long term means for the period 1911-2017.)

If you wish to download a longer period dataset please contact us at ***awracms@bom.gov.au*** for assistance.

Calibration:

A longer forcing data set for a limited number of catchments is provided for demonstrating the calibration of the system (1990-2010):

- rain, temp_max/min, solar exposure: [variable_name]_1990/2010.nc

3.4.5. Input starting states

The AWRA-L model has the following model states that change from one time-step to the next:

- **mleaf** : vegetation mass (mm) [for deep rooted (dr) and shallow rooted (sr) HRUs]
- **s0** : top layer 0-10cm soil moisture (mm) [for deep rooted (dr) and shallow rooted (sr) HRUs]
- **ss** : upper layer 10cm-100cm soil moisture (mm) [for deep rooted (dr) and shallow rooted (sr) HRUs]
- **sd** : deep layer 100cm-600cm soil moisture (mm) [for deep rooted (dr) and shallow rooted (sr) HRUs]
- **sg** : saturated groundwater storage (mm)
- **sr** : surface water storage (mm)

These states are saved as part of model simulation, for potential use in initialising model runs. These initial states are required to be specified when starting a simulation or calibration, otherwise default parameters will be used. The default settings assume all soil and ground stores are half-full.

To view the currently available files for updating the input state, list the files in the simulation directory:

```
# list files in the awracms test_data simulation directory
!ls ../../test_data/simulation
```

```
bom-rad_day-20101201-20101231.nc          rain_day_2011.nc
bom-rad_day-20110101-20110131.nc          s0_dr.nc
bom-rain_day-20101201-20101231.nc         s0_sr.nc
bom-rain_day-20110101-20110131.nc         sd_dr.nc
bom-tmax_day-20101201-20101231.nc         sd_sr.nc
bom-tmax_day-20110101-20110131.nc         sg_bal.nc
bom-tmin_day-20101201-20101231.nc         solar_exposure_day_2010.nc
bom-tmin_day-20110101-20110131.nc         solar_exposure_day_2011.nc
climatology_daily_rain_day.nc             sr_bal.nc
climatology_daily_solar_exposure_day.nc   ss_dr.nc
climatology_daily_temp_max_day.nc         ss_sr.nc
climatology_daily_temp_min_day.nc         temp_max_day_2010.nc
mleaf_dr.nc                               temp_max_day_2011.nc
mleaf_sr.nc                               temp_min_day_2010.nc
outputs                                           temp_min_day_2011.nc
rain_day_2010.nc
```

Examples of how to modify the default input configuration are available in the python notebooks in the Training/Simulation folder in awram_cm.

3.4.6. Output configuration

The nodegraph for the model output is configured in a similar way to the input with a set of default paths, variables and arguments. To view the default output mapping:

```
# default outputs
```

```
awralmod.get_output_mapping()
```

```
{'dd': output_variable([]):{'var_name': 'dd'},
 'e0': output_variable([]):{'var_name': 'e0'},
 'etot': output_variable([]):{'var_name': 'etot'},
 'mleaf_hrudr': output_variable([]):{'var_name': 'mleaf_hrudr'},
 'mleaf_hrusr': output_variable([]):{'var_name': 'mleaf_hrusr'},
 'qtot': output_variable([]):{'var_name': 'qtot'},
 's0': output_variable([]):{'var_name': 's0'},
 's0_hrudr': output_variable([]):{'var_name': 's0_hrudr'},
 's0_hrusr': output_variable([]):{'var_name': 's0_hrusr'},
 'sd': output_variable([]):{'var_name': 'sd'},
 'sd_hrudr': output_variable([]):{'var_name': 'sd_hrudr'},
 'sd_hrusr': output_variable([]):{'var_name': 'sd_hrusr'},
 'sg': output_variable([]):{'var_name': 'sg'},
 'sr': output_variable([]):{'var_name': 'sr'},
 'ss': output_variable([]):{'var_name': 'ss'},
 'ss_hrudr': output_variable([]):{'var_name': 'ss_hrudr'},
 'ss_hrusr': output_variable([]):{'var_name': 'ss_hrusr'}}
```

Examples of how to modify the default output configuration are available in the python notebooks in the Training/Simulation folder in awram_cm.

3.5. Defining the spatial extent of the model to be run

The area over which the simulation/calibration/extraction occurs is the model extent. This can either be all of Australia, a single grid cell, a region (such as a state) or a catchment.

A number of predefined extents are available in the routine, extents.

```
# load extents
from awrams.utils import extents
```

For example, the default extent, all of Australia, can be extracted from the extents routine.

```
# default extent - all of Australia
DefExt = extents.get_default_extent()  #   Australia, set as reference extent
```

Alternatively, coordinates can be input to define a rectangular extent:

```
## Any rectangular extent
my_extent = DefExt.icoords[-39.5:-44, 143.5: 149]   # Tasmania
my_extent
```

```
origin: -39.5,143.5, shape: (91, 111), cell_size: 0.05
```

It is also possible to simulate a single grid square which can be defined by lat and long.

```
## a single point based on [lat, lon] pair
my_extent  = DefExt.icoords[-34,117]
```

```
print(my_extent)
print(my_extent.cell_count)
```

```
origin: -34.0,117.0, shape: (1, 1), cell_size: 0.05
```

To run a particular catchment, you can define the catchment extent in the model by inputting a shapefile of the catchment boundary. In the example below, an existing shapefile containing all the calibration and validation catchments is opened and the attributes of these extracted into a dataframe.

```
## extents from a shapefile (CATCHMENT_SHAPEFILE contains all calibration and validation catchments)
from awrams.utils.gis import ShapefileDB
print('CATCHMENT_SHAPEFILE=',awrams.utils.gis.CATCHMENT_SHAPEFILE)
calvalshapefile = ShapefileDB(awrams.utils.gis.CATCHMENT_SHAPEFILE)
catchments = calvalshapefile.get_records_df()
```

View the names of the catchments to find those of interest.

```
# view only the first 5 rows of the shapefile table by using .head()
catchments.head()
```

To run the model for 2 catchments, you can specify the StationID from the shapefile (e.g. here, for 204007 and 421103)

```
# Example when you need to combine multiple catchments
extent_dict = {'204007':calvalshapefile.get_extent_by_field('StationID','204007',DefExt),
        '421103':calvalshapefile.get_extent_by_field('StationID','421103',DefExt)}
```

3.6. Period specification

Here we look at specifying the time period for which the simulation will be run. To define a run from 1/1/2000 to 31/12/2010: (note that if only the year is given, it is assumed the full year is to be run)

```
from awrams.utils import datetools as dt
myperiod = dt.dates('2000', '2010')
myperiod
```

```
DatetimeIndex(['2000-01-01', '2000-01-02', '2000-01-03', '2000-01-04',
               '2000-01-05', '2000-01-06', '2000-01-07', '2000-01-08',
               '2000-01-09', '2000-01-10',
               ...
               '2010-12-22', '2010-12-23', '2010-12-24', '2010-12-25',
               '2010-12-26', '2010-12-27', '2010-12-28', '2010-12-29',
               '2010-12-30', '2010-12-31'],
              dtype='datetime64[ns]', length=4018, freq='D', tz=None)
```

Alternatively, to define a period from 1/12/2010 to 3/1/2011 (note that if only the month is given, it is assumed the full month is to be run):

```
period = dt.dates('dec 2010 - 3 jan 2011')
```

```
period
```

```
DatetimeIndex(['2010-12-01', '2010-12-02', '2010-12-03', '2010-12-04',
               '2010-12-05', '2010-12-06', '2010-12-07', '2010-12-08',
               '2010-12-09', '2010-12-10', '2010-12-11', '2010-12-12',
               '2010-12-13', '2010-12-14', '2010-12-15', '2010-12-16',
               '2010-12-17', '2010-12-18', '2010-12-19', '2010-12-20',
               '2010-12-21', '2010-12-22', '2010-12-23', '2010-12-24',
               '2010-12-25', '2010-12-26', '2010-12-27', '2010-12-28',
               '2010-12-29', '2010-12-30', '2010-12-31', '2011-01-01',
               '2011-01-02', '2011-01-03'],
              dtype='datetime64[ns]', freq='D', tz=None)
. . .
```

```
# For help on defining the period
dt.dates?
```

4. Running a model simulation

There are two ways to run a simulation of the AWRA-L model in the AWRA MS:

- Using the On-Demand Simulator, or
- Using the Server Simulator.

4.1. On-Demand Simulation

The AWRA-CMS On-Demand Simulator is designed to run the model for a few years over a small spatial extent. It allows the user to quickly and efficiently assess the impact of changes made to the model or inputs before running a full simulation and writing all of the outputs to file. The user can write the results out but it is generally designed to hold the results in memory for visualisation and checking. *It is therefore recommended only for limited model extents and short model runs as it is limited by the available RAM.*

This section outlines a basic On-Demand Simulation of the AWRA-L model as well as some of the options and functionality available from the package to modify a model run and inspect the outputs.

4.1.1. Import required libraries

```
# Standard Python packages
import numpy as np
from matplotlib import pyplot as plt
# AWRAMS utilities
from awrams.utils import extents
from awrams.utils import datetools as dt
# AWRAMS input nodegraph. The nodegraph is created when building the input mapping
from awrams.utils.nodegraph import nodes
# Select simulation option
from awrams.simulation import ondemand
# Select AWRA model
from awrams.models import awral
```

4.1.2. Modifying the model configuration

1. Read in default input nodegraph

```
input_map = awral.get_default_mapping()
# View full input nodegraph
input_map
# View single element of nodegraph
input_map['hveg_hrusr']
```

```
parameter([]):{'max': 50, 'min': 0.1, 'value': 0.5, 'fixed': True}
```

7. Change forcing inputs

Changing the inputs to the model can be done by changing the values of the items in the nodegraph. Functions are used for taking in an input map (basically a dictionary in the form key: value) and replacing the values (i.e. to change an input)

```
# Create a function than will take in an existing configuration and modify the forcing items only.
def change_path_to_forcing (imap):
    from awrams.utils.nodegraph import nodes  # the nodes library contains various functions that are useful for nodegraph changes
    imap = imap.copy()  # done to ensure that the version of the input_map in memory is not modified within the function
    data_path = '../../test_data/simulation/' # we have a smaller set of data provided as part of the repo
    FORCING = {
        'tmin': (data_path,'temp_min*.nc','temp_min_day'),
        'tmax': (data_path,'temp_max*.nc','temp_max_day'),
        'precip': (data_path,'rain_day*.nc','rain_day'),
        'solar': (data_path,'solar*.nc','solar_exposure_day') #,
    }
    for k,v in FORCING.items():
        imap[k+'_f'] = nodes.forcing_from_ncfiles(v[0],v[1],v[2])  # function nodes.forcing_from_ncfiles reads in the
                                                                   # path, file name and variable name
    return imap
```

To check the updated input:

```
input_map['tmax_f']
```

```
forcing_from_ncfiles([]):{'nc_var': 'temp_max_day', 'cache': False, 'path': '/data/cwd_awra_data/AWRACMS/Training/test_data/climate/BOM_climate/', 'pattern': 'temp_max_day/temp_max*'}
```

8. Changing the spatial input grids

```
# this instruction returns the location hdf5 file containing the stack of spatial parameters
awral.settings.SPATIAL_FILE
```

```
'/data/cwd_awra_data/AWRACMS/MINICONDA3/envs/awra-cms/lib/python3.4/site-packages/awrams/models/awral/data/spatial_parameters.h5'
```

```
# Load and view spatial parameters
import h5py
h = h5py.File(awral.settings.SPATIAL_FILE,'r+')
list(h['parameters'].keys())
```

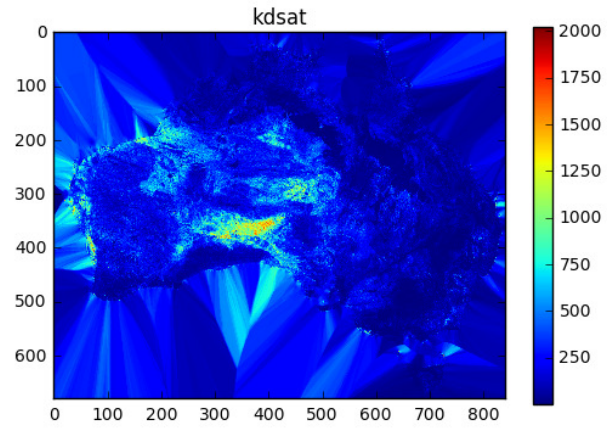
```
['f_tree',
 'height',
 'hveg_dr',
```

```
'k0sat_v5',
'k_gw',
'kdsat_v5',
'kssat_v5',
'lai_max',
'meanPET',
'ne',
'pref',
's0fracAWC',
'slope',
'ssfracAWC',
'windspeed']
```

To create a variable showing all values (across the grid) for only one of the spatial parameters (in this case kdsat) and plot these values across Australia:

```
# Isolate one of the spatial parameter (saturated hydraulic conductivity (kdsat))
kdsat_grid = h['parameters']['kdsat_v5'][:]
m = plt.imshow(kdsat_grid)
plt.title("kdsat")
plt.colorbar(im)
# Don't panic at the disco effect. Grids have been infilled to cater for potential edge effects
```

```
<matplotlib.colorbar.Colorbar at 0x7f7dc6e4d160>
```




```
# Both these instructions will allow you to look at the  mapping for a particular grid
input_map['kdsat_v5_grid']
input_map.kdsat_v5_grid
```

To point the model to an alternative spatial input (in this case for ftree or kdsat) there are a couple of different methods. Note that any spatial input data format recognised by gdal is ok (e.g. *.nc, *.flt).

```
# If you want to modify any of the arguments, in this case the filename to be used for the kd_sat grid
input_map.kdsat_v5_grid.args['filename']
# The nodes library can also be used to perform the same action, i.e. to allow you to point to a different input grid file (.nc, .flt, anything recognised by gdal) for any particular input grid
input_map.kdsat_v5_grid = nodes.spatial_from_file(PATH_TO_NEW_KdSat_FILE)
```

9. How to change a parameter

The model input nodegraph picks up the properties of the model parameters (using its member name in lower case as a key), such as display name, value, whether it is used for calibration, and range for calibration from the available parameter file, a .json type file whose location is defined in the *settings* module:

```
from awrams.models.awral.settings import DEFAULT_PARAMETER_FILE
DEFAULT_PARAMETER_FILE
```

The proposed approach to changing any of the properties of a particular paremeter is to load up the default parameter properties into a Python dataframe, select and modify the desired parameter and save the changes into the nodegraph.

The first step is to import the default parameter file into a dataframe (*param_df*) as shown below.

```
# view default parameter set
import json
import pandas as pd
model_paramset = json.load(open(DEFAULT_PARAMETER_FILE,'r'))
model_paramset
param_df = pd.DataFrame(model_paramset)
param_df
```

	Display Name	Fixed	Max	MemberName	Min	Value
0	HRU:DR Dry Soil Albedo	True	0.50	alb_dry_hruDR	0.100	0.260000
1	HRU:SR Dry Soil Albedo	True	0.50	alb_dry_hruSR	0.100	0.260000
2	HRU:DR Wet Soil Albedo	True	0.50	alb_wet_hruDR	0.100	0.160000
3	HRU:SR Wet Soil Albedo	True	0.50	alb_wet_hruSR	0.100	0.160000
4	HRU:DR Conversion Coefficient From Vegetation ...	False	0.05	cGsmax_hruDR	0.020	0.032005
5	HRU:SR Conversion Coefficient From Vegetation ...	False	0.05	cGsmax_hruSR	0.020	0.023711
6	HRU:DR Ratio of Average Evaporation Rate Over ...	False	0.25	ER_frac_ref_hruDR	0.040	0.073556
7	HRU:DR Soil Evaporation Scaling Factor When So...	False	1.00	FsoilEmax_hruDR	0.200	0.227466
8	HRU:SR Soil Evaporation Scaling Factor When So...	False	1.00	FsoilEmax_hruSR	0.200	0.929708
9	HRU:DR Reference Soil Cover Fraction That Dete...	True	0.25	fvegref_G_hruDR	0.100	0.150000
10	HRU:SR Reference Soil Cover Fraction That Dete...	True	0.25	fvegref_G_hruSR	0.100	0.150000
11	HRU:DR Fraction of Daytime Net Radiation Lost ...	True	0.50	Gfrac_max_hruDR	0.250	0.300000
12	HRU:SR Fraction of Daytime Net Radiation Lost ...	True	0.50	Gfrac_max_hruSR	0.250	0.300000
13	HRU:SR Height of Vegetation Canopy	True	50.00	hveg_hruSR	0.100	0.500000
14	K_gw_scale	False	10.00	K_gw_scale	0.100	0.502238
15	K_rout_int	False	3.00	K_rout_int	0.050	0.157715
16	K_rout_scale	False	3.00	K_rout_scale	0.050	0.050785

17	Scale for saturated hydraulic conductivity sur...	False	10.00	K0sat_scale	0.100	2.872808
18	Scale for saturated hydraulic conductivity dee...	False	1.00	Kdsat_scale	0.010	0.010036
19	Kr_coeff	False	1.00	Kr_coeff	0.010	0.074055
20	Scale for saturated hydraulic conductivity sha...	False	10.00	Kssat_scale	0.010	0.020238
21	HRU:DR Reference Leaf Area Index (at which fve...	True	2.50	LAIref_hruDR	1.300	2.500000
22	HRU:SR Reference Leaf Area Index (at which fve...	True	2.50	LAIref_hruSR	1.300	1.400000
23	Scale for effective porosity	False	1.00	ne_scale	0.010	0.055189
24	Pref_gridscale	False	3.00	Pref_gridscale	0.100	1.815294
25	HRU:DR Rooting Depth	True	20.00	RD_hruDR	3.000	6.000000
26	HRU:SR Rooting Depth	True	2.00	RD_hruSR	0.500	1.000000
27	HRU:DR Specific Canopy Rainfall Storage Capaci...	False	0.80	S_sls_hruDR	0.030	0.094597
28	HRU:SR Specific Canopy Rainfall Storage Capaci...	False	0.80	S_sls_hruSR	0.030	0.042743
29	Scale for Maximum water storage surface layer ...	False	3.00	S0max_scale	0.500	2.995850
30	Scale for Maximum water storage deep layer (Deep)	False	1.00	Sdmax_scale	0.500	0.795122
31	HRU:DR Specific Leaf Area	True	70.00	SLA_hruDR	0.700	3.000000
32	HRU:SR Specific Leaf Area	True	70.00	SLA_hruSR	0.700	10.000000
33	slope_coeff	False	1.00	slope_coeff	0.010	0.951766
34	Scale for Maximum water storage shallow layer ...	False	3.00	Ssmax_scale	0.500	2.433261
35	HRU:DR Characteristic Time Scale for Vegetatio...	True	1000	Tgrow_hruDR	20.000	1000.00000
36	HRU:SR Characteristic Time Scale for Vegetatio...	True	1000	Tgrow_hruSR	20.000	150.000000
37	HRU:DR Characteristic Time Scale for Vegetatio...	True	200.0	Tsenc_hruDR	10.000	60.000000
38	HRU:SR Characteristic Time Scale for Vegetatio...	True	200.0	Tsenc_hruSR	10.000	10.000000
39	HRU:DR Maximum Root Water Uptake Rates From De...	False	10.00	Ud0_hruDR	0.001	7.136380
40	HRU:SR Maximum Root Water Uptake Rates From De...	True	10.00	Ud0_hruSR	0.000	0.000000
41	HRU:DR Maximum Root Water Uptake Rates From Sh...	True	7.00	Us0_hruDR	1.000	6.000000
42	HRU:SR Maximum Root Water Uptake Rates From Sh...	True	7.00	Us0_hruSR	1.000	6.000000
43	HRU:DR Vegetation Photosynthetic Capacity Inde...	True	1.00	Vc_hruDR	0.050	0.350000
44	HRU:SR Vegetation Photosynthetic Capacity Inde...	True	1.00	Vc_hruSR	0.050	0.650000
45	HRU:DR Relative Top Soil Water Content at Whic...	True	0.90	w0limE_hruDR	0.600	0.850000
46	HRU:SR Relative Top Soil Water Content at Whic...	True	0.90	w0limE_hruSR	0.600	0.850000
47	HRU:DR Reference Value of w0 Determining the R...	True	0.50	w0ref_alb_hruDR	0.200	0.300000
48	HRU:SR Reference Value of w0 Determining the R...	True	0.50	w0ref_alb_hruSR	0.200	0.300000
49	HRU:DR Deep Water-Limiting Relative Water Content	True	0.50	wdlimU_hruDR	0.150	0.300000
50	HRU:SR Deep Water-Limiting Relative Water Content	True	0.50	wdlimU_hruSR	0.150	0.300000
51	HRU:DR Shallow Water-Limiting Relative Water C...	True	0.50	wslimU_hruDR	0.150	0.300000
52	HRU:SR Shallow Water-Limiting Relative Water C...	True	0.50	wslimU_hruSR	0.150	0.300000

Then we can use the function below to replace default parameters, parameter ranges and whether or not the parameter is to be calibrated or fixed.

```
# This function basically allows us to replace default parameters by new values
# we will place parameters (the ones that can potentially be calibrated) defined in a pandas dataframe into the model input_map
def change_model_parameters(imap, param_df):
    """
    the intent of this function is to replace the values in imap (your current configuration)
    by new properties (value, fixed, min, max) in a new parameter dataframe (param_df)
    """
    import pandas as pd
    variable_names = param_df['MemberName']
    variable_value = param_df['Value']
    variable_fixed = param_df['Fixed']
    variable_max = param_df['Max']
    variable_min = param_df['Min']

    # Convert to lower case for the input_mapping
    for k,v,f,vmax,vmin in zip(variable_names,variable_value, variable_fixed, variable_max, variable_min):
        imap[k.lower()].args['value'] = v # here we access the "variable_names" key in imap and replace its value argument.
                                          #As the key is in lower case, the .lower() function is applied to it.
        imap[k.lower()].args['fixed'] = f
        imap[k.lower()].args['max'] = vmax
        imap[k.lower()].args['min'] = vmin
```

Then, for example, we can change the value of the default rooting depth in the shallow root HRU ('RD_hruSR') from 0.5 to 2 as follows:

```
# first change the value in the dataframe
param_df.loc[param_df['MemberName']=='RD_hruSR','Value'] = 2
# then use the function to import the new set of variable properties (not just the value) into the nodegraph
change_model_parameters(input_map, param_df)
```

4.1.3. Put the model run specification together

1. Initialise the simulator

At this step, we create a simulator object (*OnDemandSim*) as a simulator of the AWRA-L model using the inputs in *input_map*. This simulator can flexibly be applied to various extents and periods.

```
OnDemandSim = ondemand.OnDemandSimulator(awralmod,input_map)
```

10. Define the required period and spatial extent

Set the period to December 1st 2010 to January 31st 2011.

```
period = dt.dates('dec 2010 - jan 2011')
```

Start with the default extent for all of Australia, *edef*.

```
# Set the starting extent as the extent of the AWAP grid
# in the background this picks up the geospatial references associated with the grid set as default in the configuration file
edef = extents.get_default_extent()
```

and choose a subset of this, e.g. the Perth region defined by the grid cell coordinates below.

```
# Select a sub-area of that grid, say the Perth region [450 grid cells south, 20 grid cells across]
esub = edef.ioffset[400:450,50:100]
esub
```

```
origin: -30.0,114.5, shape: (50, 50), cell_size: 0.05
```

11. Run the model for the defined extent and period

The model can now be run using the below command, and the outputs stored in the variable *r*.

```
r = OnDemandSim.run(period,extent=esub)
```

A switch is available in the model run command that allows the algorithm to also capture the inputs and their attributes, i,e. forcing data and grid values that apply specifically to the extent run. In this example, the outputs are stored in *r*, and the inputs in *ires*.

```
# 4.3.1 Option to capture model inputs
r, ires = OnDemandSim.run(period,esub,True) # r holds  model outputs, ires holds model gridded inputs/parameters
```

Because the inputs were captured, we can now view some of the details of the forcings for this particular run. For example here, we see that for this run, there are 62 points in the time-series (i.e. days between 1/12/2010 and 31/1/2011) and 1602 grid cells in the Perth region.

View the contents of the model inputs for the Perth region in the variable, *ires*:

```
ires
```

```
{0.5: 0.5,
...,  182.2794342 ,
        193.22906494,  195.96000671], dtype=float32)}
```

```
# E.g. view forcings
forcing = ('tmin_f','tmax_f','solar_f','precip_f', 'u2t')
# Use a dictionary comprehension command to capture each forcing as a key in the clm dictionary
clm = {k:ires[k] for k in forcing}
```

```
clm['precip_f'].shape
```

```
(62, 1602)
```

4.1.4. Exploring model outputs

Here we can see what model outputs were produced in the run *r* above. As the output configuration has not been modified, the outputs are those defined as the default output mapping plus model states captured to be able to hotstart (i.e. start the model from the previous run finish date).

```
r.keys() # this is to see what's in the model run outputs
```

```
dict_keys(['sd_hrusr', 'mleaf_hrudr', 's0_hrudr', 'ss', 'qtot', 'mleaf_hrusr', 's0_h
    rusr', 'dd', 'sd', 'ss_hrusr', 'sr', 'final_states', 'ss_hrudr', 'e0', 'sd_hrud
    r', 'etot', 's0', 'sg'])
```

For example, to view the results of shallow soil moisture for the shallow rooted vegetation HRU:

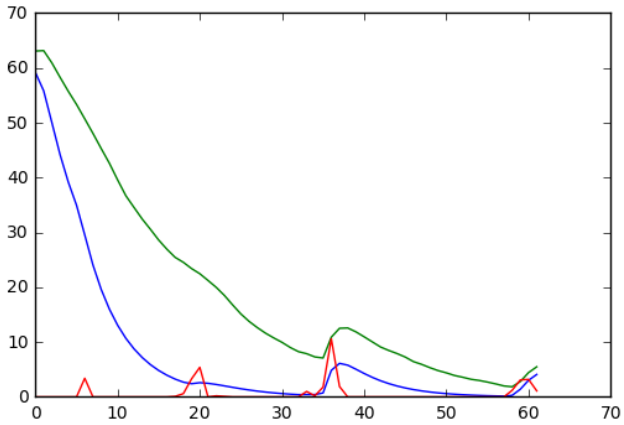
```
r['ss_hrusr']  # this returns an array consisting of a time series for each grid cell
```

```
array([[  59.14972151,   52.39277049,   51.61301026, ...,  106.43728888,
         106.28906634,  106.09083665],
       [  55.84062366,   47.10506228,   44.7127814 , ...,  101.49080397,
         100.86090823,  101.07974464],
       [  50.03172435,   40.75269457,   37.302985  , ...,   97.48582264,
          96.43379343,   96.98550532],
       ...,
       [   1.40551594,    2.50933683,    2.83167797, ...,    4.32575599,
           4.1813607 ,    3.86894543],
       [   3.03523288,    4.14886125,    4.2654218 , ...,   12.33673316,
          11.80357185,   10.21500349],
       [   4.10244692,    4.94945026,    5.14585548, ...,   13.18006557,
          12.49617596,   10.98374063]])
```

To plot a time-series of how the shallow soil moisture (for the shallow and deep routed HRUs) and the precipitation varies during the model run for a single model cell.

```
# For the first cell of the extent over the entire period of modelling
# the starting point is a function of the initial state used.
plt.plot(r['ss_hrusr'][:,0])  # in blue
plt.plot(r['ss_hrudr'][:,0])  # in green
# can also add the model inputs
plt.plot(ires['precip_f'][:,0]) # in red
# Note these commands produce a raw plot of the arrays with no labels for axes nor legend.
# These could be added with further matplotlib commands.
# In this case, the x-axis is the number of days since start of simulation, and the y-axis is the magnitude of the soil
# moisture  or precipitation (mm)
```

```
[<matplotlib.lines.Line2D at 0x7f7dc41e6358>]
```



4.1.5. Configuring further outputs to the model

If you require output of results not in the default mapping, this can be specified prior to running the model. Here, we show an example of adding grid cell averaged leaf biomass (*mleaf*) and HRU specific potential evapotranspiration, *e0*, to the model outputs, then running a new simulation with the newly specified outputs. As previously, the model is run by the command *run*()..

```
# See the currently selected set of model outputs
awralmod.OUTPUTS
```

```
{'OUTPUTS_AVG': ['e0', 'etot', 'dd', 's0', 'ss', 'sd'],
 'OUTPUTS_CELL': ['qtot', 'sr', 'sg'],
 'OUTPUTS_HRU': ['s0', 'ss', 'sd', 'mleaf']}
```

```
# Add mleaf, which is available per hru, to the averaged outputs
awralmod.OUTPUTS['OUTPUTS_AVG'].append('mleaf')
# Turn on the individual HRU outputs for e0
awralmod.OUTPUTS['OUTPUTS_HRU'].append('e0')
# Run a simulation with the new outputs
OnDemandSim_newOutputs = ondemand.OnDemandSimulator(awralmod,input_map) # Initialise simulator
r = OnDemandSim_newOutputs.run(period,extent=esub)  # run the simulator and store outputs in r
```

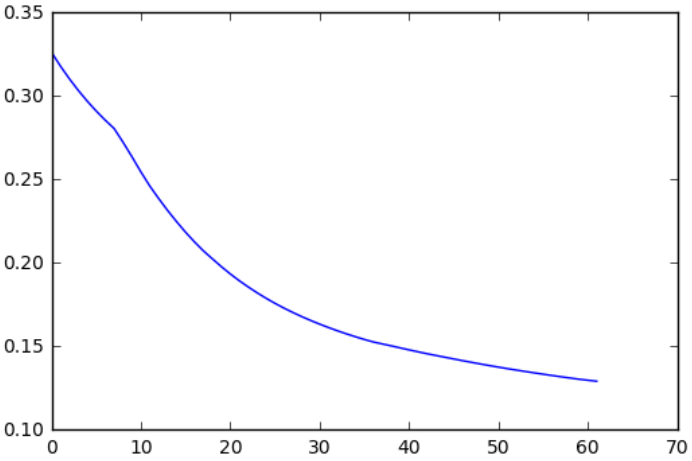
We can now look at and plot our new results for grid averaged mleaf:

```
r['mleaf'].shape
```

```
(62, 1602)
```

```
# Examine mleaf
plt.plot(r['mleaf'][:,0])  # in blue
```

```
[<matplotlib.lines.Line2D at 0x7f7dbfca7358>]
```



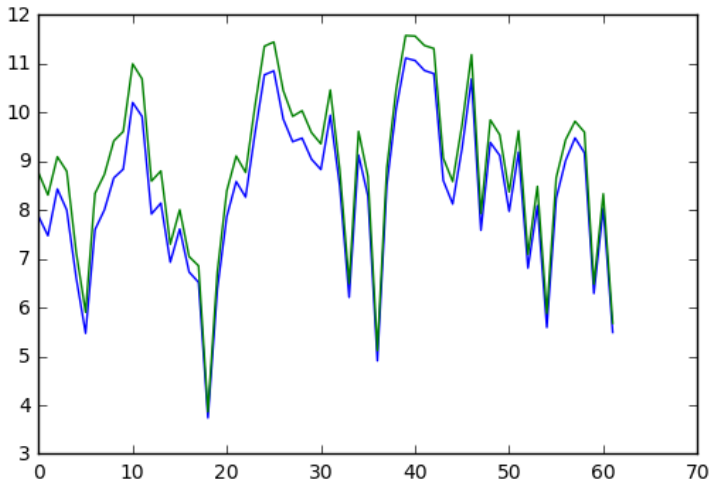
where:

- x-axis = days since simulation start
- y-axis = mleaf

And we can now look at and plot our new results for HRU specific *e0*:

```
# Examine e0
plt.plot(r['e0_hrusr'][:,0]) # in blue
plt.plot(r['e0_hrudr'][:,0]) # in green
```

```
[<matplotlib.lines.Line2D at 0x7f7dc40715f8>]
```



where:

- x-axis = days since simulation start
- y-axis = e0 in mm

4.1.6. Save outputs to file

The OnDemand simulator is not set up by default with an output mapping, so does not write out anything to file using default settings. However, writing outputs to file can readily be done by building a new output mapping for the run as per below. Here, we create a new function, *build_output_mapping*, that can be used to specify any output to write to netcdf file. In this case s0, ss and sd outputs will be saved as netcdf files with daily data in annual files.

```
def build_output_mapping(model):
    from awrams.utils.nodegraph import nodes

    output_map = model.get_output_mapping()

    outpath = './_results/'

    output_map['s0_ncsave'] = nodes.write_to_annual_ncfile(outpath,'s0')
    output_map['ss_ncsave'] = nodes.write_to_annual_ncfile(outpath,'ss')
    output_map['sd_ncsave'] = nodes.write_to_annual_ncfile(outpath,'sd')
    return output_map
```

We then re-initiatlise the function, *build_output_mapping*, to create a new output map in the variable, *omap.*

```
omap = build_output_mapping(awralmod)
# Re-initialise a simulator, this time with an output map
OnDemandSim_write_outputs = ondemand.OnDemandSimulator(awralmod,input_map,omapping=omap)
```

Now, we can rerun the model with our specified dates, extents and saving of input data using the output mapping from the example above:

```
# Specify period of model
period = dt.dates('dec 2010 - jan 2011')
# Run the model writing outputs to file, and also returning model outputs in r and model inputs in ires
r, ires = OnDemandSim_write_outputs.run(period,esub,True)
```

Additional things you can do with the model configuration, such as changing initial states and infilling gaps in the forcing inputs are presented in the SimulationServer notebook

4.1.7. Quickly viewing model run outputs held in memory

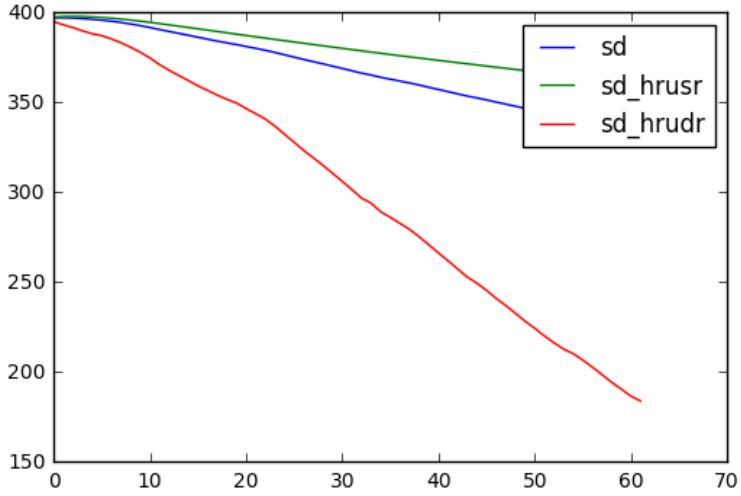
In the previous example we showed how to plot some of the model outputs using the plt.plot() command. The model outputs can be gathered into numpy arrays by using functionality available in the nodes library. That way, 1D and 2D arrays can be visualised using usual matplotlib commands (plt.plot, plt.imshow)

1. View 1-D ouputs

For example, to view catchment averaged outputs of multiple variables on a single time-series graph:

```
for v in ['sd','sd_hrusr','sd_hrudr']:
    plt.plot(r[v].mean(axis=1),label=v)
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f7dbf935d30>
```



where:

- x-axis = days since simulation start,
- y-axis = output in mm.

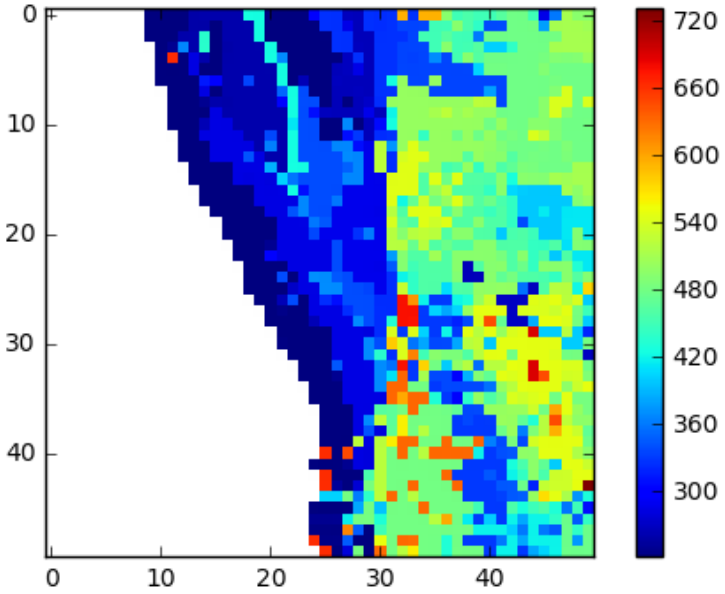
12. View 2-D outputs

To view maps of model output for a particular time-step and area, the nodes.get_expanded() function can be used to create a grid for one time-step from the model outputs.

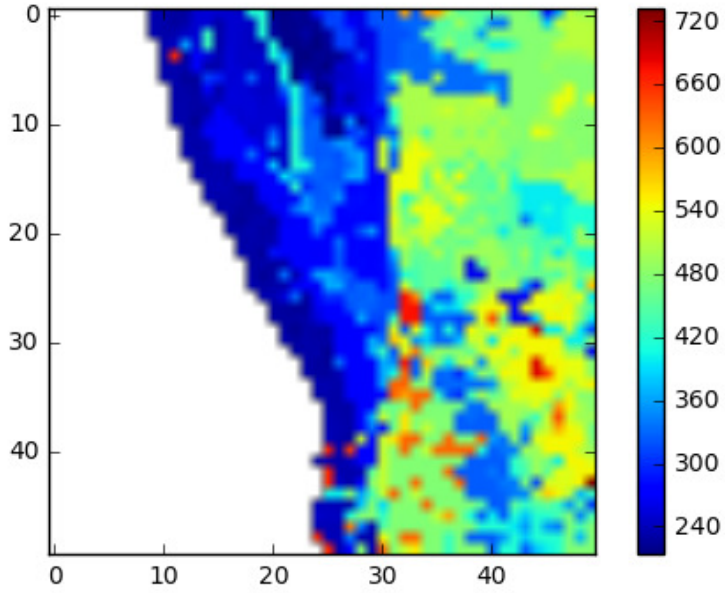
```
# This command creates a gridded array for the first time-step of the r[ 'sd'] output.
nodes.get_expanded(r['sd'][0],esub.mask) # [0] means the first time step, and
                                         # esub.mask specifies that we are only looking for ex
                                         # tent esub,  i.e. the Perth region (Sect. 4.1.3)
```

```
masked_array(data =
 [[-- -- -- ..., 511.81616993368476 495.4767197655839 495.3851941328383]
 [-- -- -- ..., 511.8023464627362 511.8817058641211 508.7091123796555]
 [-- -- -- ..., 511.9534756873185 511.84697057477035 512.1667294371091]
 ...,
 [-- -- -- ..., 477.1237510847895 440.85104128410893 477.18283880381387]
 [-- -- -- ..., 477.0540413099633 375.5022237494674 477.2681763168161]
 [-- -- -- ..., 477.21548728982475 477.33359844003735 477.122465003769431],
             mask =
 [[ True  True  True ..., False False False]
 [ True  True  True ..., False False False]
 [ True  True  True ..., False False False]
 ...,
 [ True  True  True ..., False False False]
 [ True  True  True ..., False False False]
 [ True  True  True ..., False False False]],
       fill_value = nan)
```

```
im = plt.imshow(nodes.get_expanded(r['sd'][0],esub.mask),interpolation='None')
plt.colorbar(im)
```




```
im = plt.imshow(nodes.get_expanded(r['sd'][10],esub.mask)) #,interpolation='None')
plt.colorbar(im)
```



4.1.8. More examples

1. Run over the entire continent.

Here we show an example of running all of Australia (extent *edef*, see Section 4.1.3), just for one day. You are likely to have memory issues if want to run for longer periods on your PC as you need to allow for 2MB per output variable per day.

```
period = dt.dates('jan 1 2011')
r, ires = runner.run(period,extent = edef, True) # r holds  model outputs, ires holds model gridded inputs/parameters
```

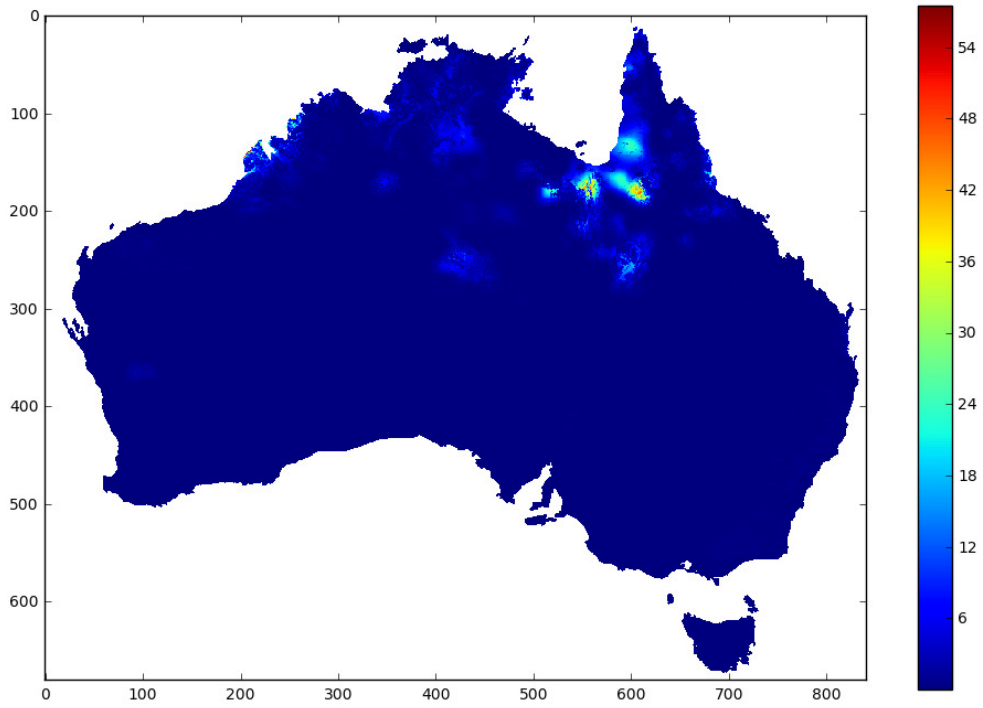
Before plotting a map of the results across Australia, specify a different figure size than the default.

```
from matplotlib import rcParams
rcParams['figure.figsize'] = [12.,8.]
```

Plot a map of Australia with the runoff results for 1 Jan 2011.

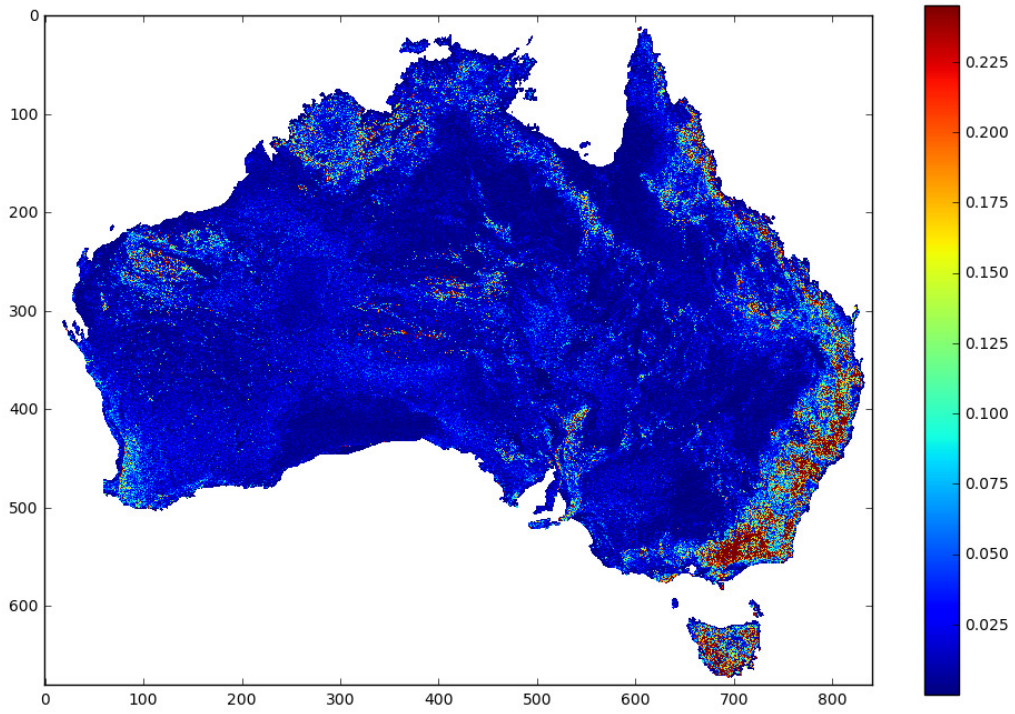
```
im = plt.imshow(nodes.get_expanded(r['qtot'][0],edef.mask),interpolation='None')
plt.colorbar(im)
```

(Note that while this was a day with high runoff across Eastern Qld this can't been seen on this map due to the relatively high runoff in Northern Australia and the fact that default initial states were used in this run – i.e. the initial soil moisture levels were not historically accurate).



Finally, we show an example of plotting the inputs saved across the continent in ires. Here we show the average slope per grid cell:

```
# Also  view slope gridded input
im = plt.imshow(nodes.get_expanded(ires['slope'],edef.mask),interpolation='None')
plt.colorbar(im)
```



Try the following plots yourself:

```
# Grid cell elevation range (highest point of hypsometric curve  - lowest point)
im = plt.imshow(nodes.get_expanded(ires['height'][-1]-ires['height'][0],edef.mask),interpolation='None')
plt.colorbar(im)
```

```
# Rainfall input on the first day of simulation
im = plt.imshow(nodes.get_expanded(ires['pt'][0],edef.mask),interpolation='None')
plt.colorbar(im)
```

4.1.9. Run with a uniform rain input across the country

To further show how model input data can be manipulated, here we show an experiment of running the model with a uniform value (1000 mm per day) of rainfall across the country. The model run is setup here:

```
input_map.pt = nodes.const(1000)
runner = ondemand.OnDemandSimulator(awralmod,input_map)
```

And the model is run here, with the outputs written to variable, fixedpt,

```
fixedpt = runner.run(period,edef)
```

Try plotting the resulting runoff in a map yourself:

```
im = plt.mshow(nodes.get_expanded(fixedpt['qtot'][0],edef.mask),interpolation='None')
plt.colorbar(im)
```

4.2. Basic Server Simulation

To run the AWRA model for more than a couple of years or for a significant extent requires that results are sequentially written and not stored in memory. For this reason the server version of the simulator should be used. The Server Simulation is set up very similarly to the On Demand Simulator, except it needs to write the outputs to disk as the model run progresses, as it does not hold the results in memory. This allows the Server Simulator to be the more efficient choice for longer runs over a larger extent. The same output and input maps can be used for either simulator.

4.2.1. Import required libraries and set data paths

```
from awrams.simulation import server
from awrams.models.awral.model import AWRALModel, CLIMATE_DATA, FORCING
from awrams.models.settings import TRAINING_DATA_PATH

from awrams.utils import extents
from awrams.utils import datetools
```

```
TEST_DATA_PATH = '../../test_data/simulation/'
```

```
/data/cwd_awra_data/AWRACMS/Training/test_data/
```

```
CLIMATE_DATA  = TEST_DATA_PATH
```

```
# Initialise the model
awral = AWRALModel()
```

4.2.2. Modify the model configuration

1. Read in default configuration

```
input_map = awral.get_default_mapping()
```

```
# Here we set the path to tmax, tmin, precip and solar
input_map.tmax_f.args['path'] = CLIMATE_DATA
input_map.tmin_f.args['path'] = CLIMATE_DATA
input_map.precip_f.args['path'] = CLIMATE_DATA
```

```
input_map.solar_f.args['path'] = CLIMATE_DATA
```

13. Create model output map and add nodes for saving some model outputs to netcdf files

```
# These are the default model outputs
# OUTPUTS_AVG : Weighted average of HRU outputs
# OUTPUTS_CELL : Cell level outputs (ie processes happening outside of HRU; groundwater store etc)
# OUTPUTS_HRU : HRU level outputs (ie separate output per HRU)
awral.OUTPUTS
```

```
{OUTPUTS_AVG: [e0, etot, dd, s0, ss, sd],
 OUTPUTS_CELL: [qtot, sr, sg],
 OUTPUTS_HRU: [s0, ss, sd, mleaf]}
```

```
# Get output map
omap = awral.get_output_mapping()
```

Change the default output mapping to write model outputs of annual ss, sd, qto and etot to file. This is done by first defining a function, *build_output_mapping*, then executing that function for a certain output mapping and path to results.

```
# Add some file output nodes to our mapping
def build_output_mapping(output_map, outpath): # outpath is the user-specified path to results
    from awrams.utils.nodegraph import nodes

    FILE_OUTPUT_VARS = ['ss','sd','qtot','etot']

    for f in FILE_OUTPUT_VARS:
        output_map[f+'_ncsave'] = nodes.write_to_annual_ncfile(outpath,f)

    return output_map
```

Use the above function to change the variables being output and the output path

```
outpath = './_results_ServerSim/' # Specify the output path
omap = build_output_mapping(omap, outpath) # update the output mapping with the output path
# The updated output map contains write_to_annual_ncfile nodes for the variables we specified above
```

14. Define the required period and spatial extent

Because we are using the Server Simulator we can now define large extents and longer run-times. Here we define a run with the full continent (default) for 2 months.

```
period = datetools.dates('dec 2010 - jan 2011')
edef = extents.get_default_extent()
```

4.2.3. Put the model run specification together

1. Initialise the server simulator

```
sim = server.SimulationServer(awral)
```

15. Run the simulation

In this step our newly defined simulation is run on the simulation server. Note that the simulation results are not saved to a variable like they were in the on-demand simulation. Instead we look to the path specified in the building of the output mapping.

```
sim.run(input_map,omap,period,edef)
```

16. Note about the outputs

The model run outputs will be located at the path specified earlier by *outpath.* The AWRAL results are output in netcdf format with a single .nc file containing all of the daily grids in a year for a single variable. For example the file qtot_avg_2016.nc contains the 365 daily runoff grids for 2016. A continental simulation output is in the order of 250 MB per year per variable. The size of the files changes with the spatial extent of the simulation.

The best way to handle, extract and manipulate these output files is to use the AWRA-CMS tools. These tools are explored in the Visualisation, Calibration, Benchmarking and Utility sections of this guide. A brief look at some of the Visualisation utilities is provided in the following section.

4.2.4. Visualise outputs

Because outputs are written out to file, we can use the process described in the Visualisation section.

1. Import visualisation libraries

```
import awrams.visualisation.vis as vis
import awrams.visualisation.results as res
import awrams.utils.extents as extents
```

17. Load model outputs

Create a variable in memory with the model results read from file and list the variables available.

```
results = res.load_results('./_results_ServerSim')
results.variables
```

```
OrderedDict([(etot, etot), (ss, ss), (sd, sd), (qtot, qtot)])
```

18. Look at a time slice

This command visualises the results for a particular day for all of Australia. Running this command results in an image file being output.

```
results[:,'1 jan 2011',:].spatial()
```

4.2.5. How to change the initial states

The following functionality applies to both OnDemand and Simulator runs. Start by getting the default initial states.

```
# Get the default initial states path for the supplied training data
from awrams.models.awral.settings import INITIAL_STATES_PATH
INITIAL_STATES_PATH
```

```
/data/cwd_awra_data/AWRACMS/Training/test_data/simulation/initial_states/
```

```
# Override if need be
 INITIAL_STATES_PATH = '/data/cwd_awra_data/AWRACMS/Training/test_data/simulation/initial_states/'
```

Define a function that allows you to input a folder with initial values of sr and sg and update the mapping.

```
# Loading initial states from netCDF files, a fixed set of initial states independent of date
# These variables you have in your file can have any name, these are from a previous model version so have slightly differ
ent variable names
def initial_states_from_files(imap):
    from awrams.utils.nodegraph import nodes


    data_path = INITIAL_STATES_PATH


    mapping = imap.copy()


    mapping['init_sr'] = nodes.init_state_from_ncfile(data_path,'sr_bal*','sr_bal')
    mapping['init_sg'] = nodes.init_state_from_ncfile(data_path,'sg_bal*','sg_bal')


    HRU = {'_hrusr':'_sr','_hrudr':'_dr'} # Conversion table for old->new state naming conventions
    for hru in ('_hrusr','_hrudr'):
        for state in ["s0","ss","sd",'mleaf']:
            mapping['init_'+state+hru] = nodes.init_state_from_ncfile(data_path,state+HRU[hru]+'*',state+HRU[hru])
    return mapping
```

Update the *input_map* with the new initial states by running the function above with the location (path) of the new initial state data as a function argument.

```
# Update the input_map
input_map = initial_states_from_files(input_map)
# Print out new init_states nodes...
dict([(k,v) for k,v in input_map.items() if k.startswith('init')])
```

```
#  Initialise model
sim = server.SimulationServer(awral)


# Change output path by modifying the  output mapping
outpath = './_results_ServerSim_initstates/'  # set a new output path or clear previous outputs. Errors are likely if files exist
.
omap = build_output_mapping(omap, outpath)


# Run the model. Output files are written out and no output is held in memory
sim.run(input_map,omap,period,edef)
```

Alternatively, you might want to compute custom states. The function below offers the basic capability to hold the state data as arrays in memory. These arrays can be manipulated as per the user's requirements before being returned back to the nodegraph.

```
# Using the same data as above, but this time load the states into memory, and store in a python dictionary
# Use this when you want to compute custom states, for example
def initial_states_from_dict(imap,period,extent):
    from awrams.utils.io.data_mapping import SplitFileManager
    from awrams.utils.nodegraph import nodes

    mapping = imap.copy()

    data_path = INITIAL_STATES_PATH

    node_names = {'mleaf_dr': 'init_mleaf_hrudr',
                  'mleaf_sr': 'init_mleaf_hrusr',
                  's0_dr': 'init_s0_hrudr',
                  's0_sr': 'init_s0_hrusr',
                  'ss_dr': 'init_ss_hrudr',
                  'ss_sr': 'init_ss_hrusr',
                  'sd_dr': 'init_sd_hrudr',
                  'sd_sr': 'init_sd_hrusr',
                  'sg_bal': 'init_sg',
                  'sr_bal': 'init_sr'}

    data_map = {}
    period = [period[0] - 1] # Use states from the previous day

    for k in node_names:
        sfm = SplitFileManager.open_existing(data_path,k+'*nc',k)
```

```
        data_map[node_names[k]] = sfm.get_data(period,extent)[0]     # At this point, the data is read in  and it can be modified as needed


    # nodes.init_states_from_dict is a convenience function to update a mapping in-place
    nodes.init_states_from_dict(mapping,data_map,extent) # this reads the new data_map and replaces the orginal mapping
    return mapping
```

Update the input nodegraph with the new initial states

```
input_map = initial_states_from_dict(input_map,period,edef)
```

Run the simulation

```
# Initialise the simulator
sim = server.SimulationServer(awral)
# Change the output path by modifying the output mapping
outpath = './_results_ServerSim_initstatesfromdate/'
omap = build_output_mapping(omap, outpath)
#Run the model
sim.run(input_map,omap,period,edef)
```

4.2.6. Fill gaps in forcing data with climatology

Sometimes, the input forcing data may contain missing values or be completely unavailable. Here, we show an example case where the solar radiation inputs were unavailable in 2009, and prior to 1990. So that we can still run the model, we use a monthly climatology (average for each month) to replace the missing data. This data is provided in a netcdf file (Rad_1990_2009.nc) file within the registered user data folder.

The function below is defined to insert a climatology into the solar radiation data when data is missing.

```
def insert_solar_climatology(imap):
    from awrams.utils.nodegraph import nodes
   # Create a CLIMATOLOGIES dictionary with the forcing name as key, and the correct path to the climatology file and variable names as tuple components
    CLIMATOLOGIES = {'solar': (TRAINING_DATA_PATH+'simulation/climatology/Rad_1990_2009.nc','solar_exposure_day')}
    imap = imap.copy()
    imap['solar_f_orig'] = imap['solar_f'] # 'Move' the forcing node to a new name.
                                           # We want to keep  'solar_f' as the final output name for the nodegraph,
                                           # but we need to first combine it with the climatology
    imap['solar_climatology_f'] = nodes.monthly_climatology(*CLIMATOLOGIES['solar']) # The nodes module has the .
                                          #monthly_climatology() function that is able to add climatologies to the nodegraph
```

```
    # Replace 'solar_f' with infilled data. This ensures that any other nodes in the graph who use solar_f as input will autom
     # atically receive the infilled data

    # nodes.gap_filler takes the first argument as 'gappy' data, identified as -9999, and infills with data from the second ar
      # gument, i.e. the solar_climatology in this case

    imap['solar_f']  = nodes.gap_filler('solar_f_orig','solar_climatology_f')

    return imap
```

The input nodegraph (*input_map*) is updated by running the function above .

```
input_map = insert_solar_climatology(input_map)
```

Upon checking the contents of the input nodegraph for the solar radiation forcing, it can be seen that it is now no longer the original file path, but a result of the gap_filler function being applied to it with the climatology provided.

```
input_map['solar_f']
```

```
gap_filler([solar_f_orig, solar_climatology_f]):{}
```

Run the simulation with the gap-filled solar radiation forcing.

```
# Initialise the simulator

sim = server.SimulationServer(awral)

# Specify the output path by changing the output mapping

outpath = './_results_ServerSim_solarclim/'

omap = build_output_mapping(omap, outpath)

# Run the simulation

sim.run(input_map,omap,period,edef)
```

5. Visualising AWRA-L inputs and outputs as maps and time-series

The inputs to and results of an AWRA-L simulation can be viewed either as maps or time-series using the tools of the AWRA MS. This section demonstrates the available functionality to visualise outputs or inputs that are in the format used by the model, i.e. in CF compliant netcdf files that contain a year's worth of daily data for a single variable. In addition, it will be shown how other tools can be used for extracting subsets of data from the overall outputs.

The following steps are covered:

1. Import required library,
2. Load inputs or results,
   2.1 Inspect which variables are available,
3. Spatial plots for different data slices,
   3.1 A single day, whole extents,
   3.2 Aggregated over a month for specified region,
   3.3 Change aggregation method,
   3.4 Accessing the underlying data,
   3.5 Specifying a catchment extent,
   3.6 Show location of catchment,
   3.7 Manipulating matplotlib settings,
4. Time-series plots for selected locations/regions,
   4.1 Time-series for a single location,
   4.2 Time-series for a catchment,
5. Comparing different simulations.

5.1. Setup

5.1.1. Import required modules

```
import awrams.visualisation.vis as vis
import awrams.visualisation.results as res
import awrams.utils.extents as extents
from awrams.utils.gis import ShapefileDB, CATCHMENT_SHAPEFILE
from awrams.utils.settings import TRAINING_DATA_PATH
```

5.1.2. Load inputs and results

Load the results from the server simulation run in Section 4 into a variable called *results*.

```
# Results path is the relative path that the Simulation notebook puts results when run. The code expects to find netcdf files in this folder
```

```
results = res.load_results('../Simulation/_results_ServerSim')
inputs  = res.load_results(training_data_path+ '/climate/BOM_climate/rain_day/')
results.path
```

5.1.3. Inspect the variables present in results

```
results.variables
```

```
Output: OrderedDict([('s0', s0), ('etot', etot), ('ss', ss), ('qtot', qtot), ('sd', sd)])
```

Alternatively you can use tab complete after typing 'results.variables.' or 'inputs.variables.' to see a list of available output and input variables.

```
# User Tab-complete to access the variable list
results.variables.qtot
inputs.variables
```

```
OrderedDict([('rain_day', rain_day)])
```

5.1.4. Spatial plots for different data slices

The visualisation commands require the definition of three parameters: the variables to be plotted, the period and the extent of interest for creating a map in a list: [variables, period, extent].

The variables slice can consist of:

- a single variable - results.variables.qtot
- multiple variables as a tuple – (results.variables.qtot,results.variables.ss)
- all variables using standard slicing syntax - ":"

The period slice can consist of:

- a single day - "1 jul 2010"
- a period - "jul 2010" or "jul 2010 - jun 2011"
- data will be aggregated over the period using the specified method (pass aggregate_method='average' or aggregate_method='sum') or the default method for a variable

The extent slice can consist of

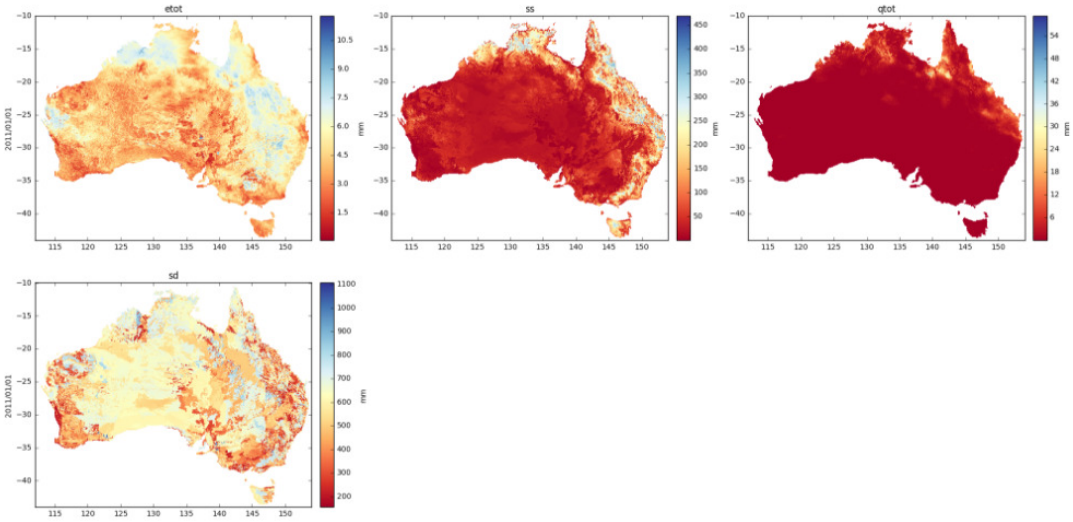
- entire spatial extents - ":" or extents.get_default_extent()
- a bounding box - results.extent.icoords[-39.5:-44,143.5:149]
- a catchment - catchments.get_extent_by_field('GaugeName','Gunning',results.extent)

  (Note that the field name and catchment are valid for the supplied calibration/validation catchments shapefile)

5.1.5. Display all variables, for a single day, at the default continental extent

The required result slice is accessed and plotted using the command '*results[variables,period,extent].spatial()*'. In the example below, all variables and all grid points are specified for the 1st January 2011.

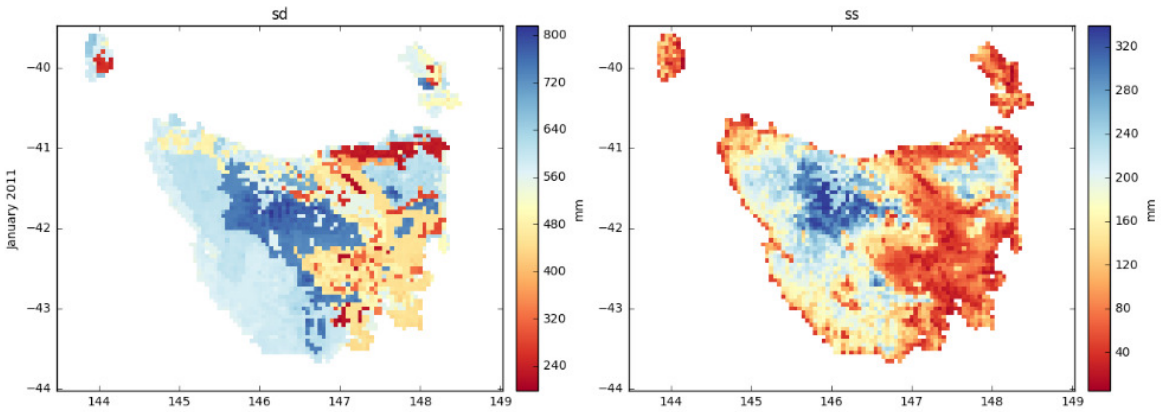
```
results[:,'1 jan 2011',:].spatial()
```



5.1.6. Aggregate over a month for a specified region

In this example, two maps showing the average deep layer and shallow soil moisture across Tasmania will be plotted for the month of January. The initial step is to extract the deep and shallow soil moisture results into the variable *v*, define the period as *January 2011* (and as mean is the default aggregation for soil moistures, the mean January 2011 soil moisture is shown) and define the spatial extent of Tasmania using coordinates. The final command specifies the name and format (.png) of the image file to be saved.

```
v = results.variables.sd,results.variables.ss
results[v,'jan 2011',results.extent.icoords[-39.5:-44,143.5:149]].spatial()
vis.plt.savefig('map_of_tasmania.png', format='png', dpi=120)
```

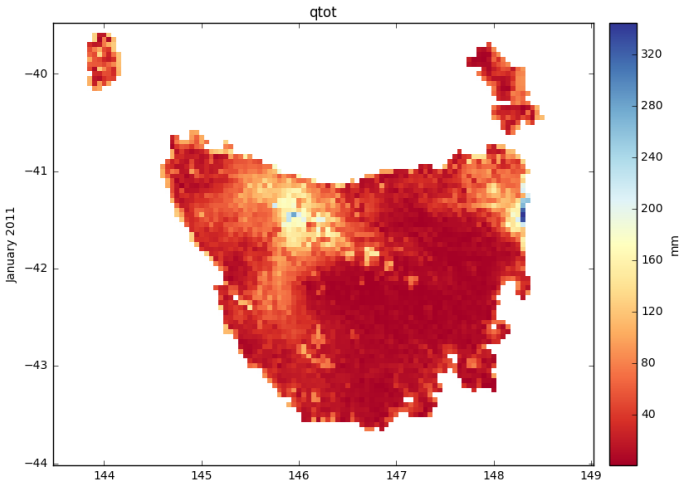


5.1.7. Change aggregation method from default of mean to sum

The temporal aggregation method for the soil moisture variables can be changed, for example to 'sum'.

Here, the aggregation method for flow is specified as 'sum' prior to plotting.

```
v = results.variables.qtot
v.agg_method = 'sum'
results[v,'jan 2011',results.extent.icoords[-39.5:-44,143.5:149]].spatial()
```



5.1.8. Accessing the underlying data

Raw data cube

The data for one variable is a 3-dimensional array (time,x,y)=(31,91,111)

```
ss_data = results.variables.ss.data
```

Data shape

```
results.variables.ss.data.shape
```

```
(31, 91, 111)
```

Temporally aggregated data

```
results.variables.ss.agg_data.shape
```

```
(91, 111)
```

5.1.9. Specifying a catchment extent

The AWRA-CMS is supplied with a shapefile outlining all (782) calibration and validation catchments used in the AWRA-L v5. This file is called CATCHMENT_SHAPEFILE

```
# By default catchments used in AWRA calibration/validation are read from a supplied shapefile
from awrams.utils.gis import CATCHMENT_SHAPEFILE
CATCHMENT_SHAPEFILE
```

Features in the shapefile are referenced ("keyed") by the field: 'StationID' and named with the fields: ['GaugeName', 'RiverName'].

The following commands might be useful to inspect the default shapefile:

```
# A quick way to inspect the contents of the shapefile
default_catchments = ShapefileDB(CATCHMENT_SHAPEFILE)
default_catchments_df = calvalshapefile.get_records_df()
default_catchments_df.head()
```

To supply a different shapefile call:

```
catchments = ShapefileDB(shp_file="shape_file_name")
```

MeanP	QComplete	RiverName	SlopeMean	SlopeRange	State	StationID	Whpk1_Mean	Whpk2_Mean	WrscID
556.338623	93.998816	Barcoo	1.01441	11.2756	QLD	003303	32.0211	70.6515	003303A
241.771332	29.554536	Hamilton Ck	3.66823	18.2268	SA	004502	35.0000	77.0000	AW0045
265.750458	35.540921	Mt McKinlay Ck	4.07388	22.4773	SA	004508	34.9490	76.5210	AW0045
314.112885	99.822406	Hugh	2.11346	28.3030	NT	005115	27.2579	81.0000	G005011
352.356567	82.618026	Trephina Ck	2.64208	24.1171	NT	006005	22.8571	0.0000	G006000

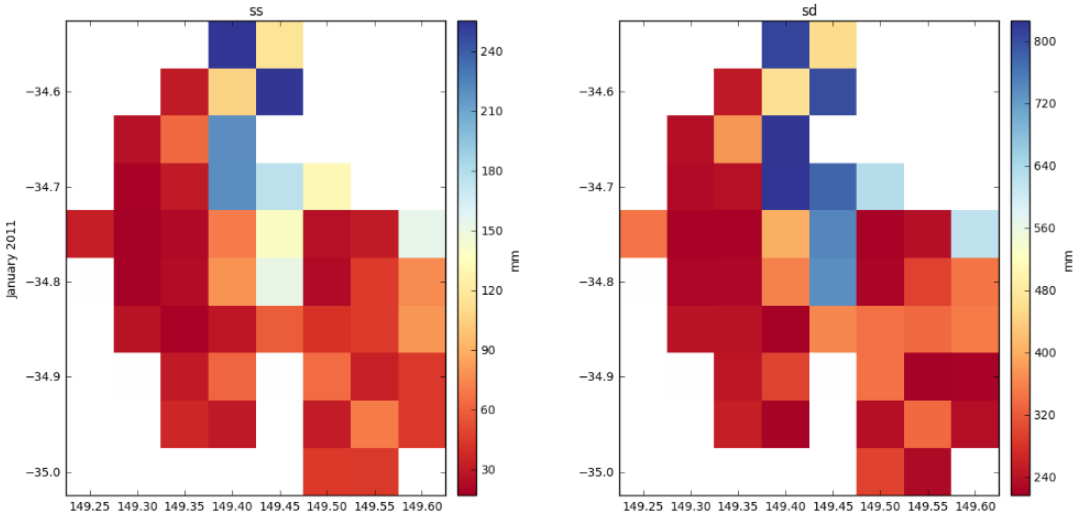
To look at an individual catchment/gauge in more detail (e.g. Gunning catchment), extract the catchment extent into the variable, gunning.

```
gunning = default_catchments.get_extent_by_field('GaugeName','Gunning',results.extent)
gunning
```

```
origin: -34.55,149.25, shape: (10, 8), cell_size: 0.05
```

This variable can then be used to visualise the results only for that catchment.

```
v = results.variables.ss,results.variables.sd
results[v,'jan 2011',gunning].spatial(interpolation=None) # interpolation="bilinear"
```



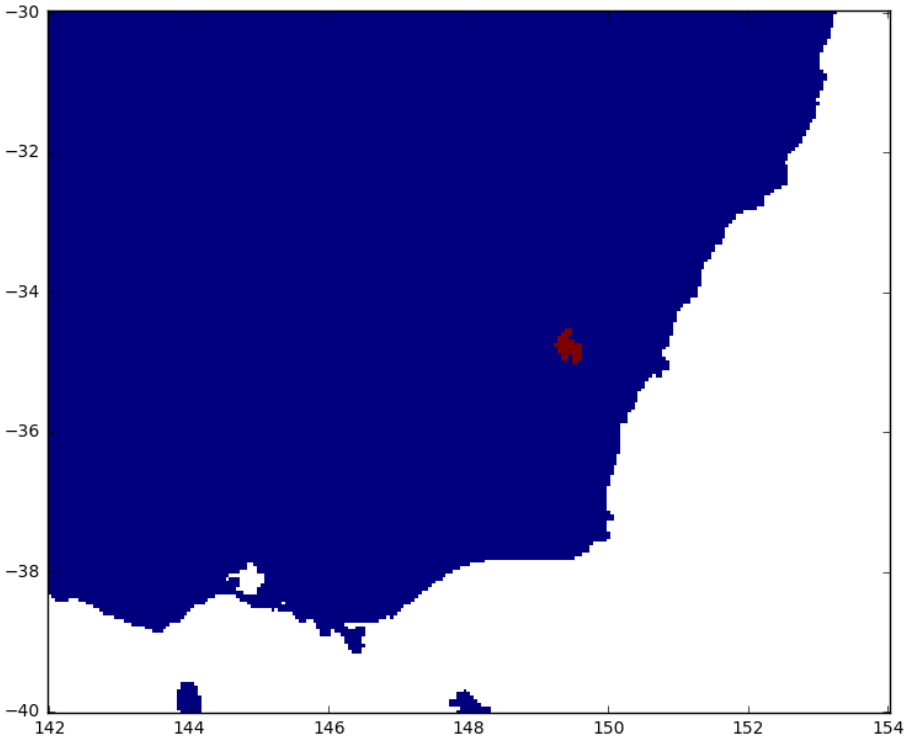

```
# QUICK TIP TO INSPECT
# Create an extent from a shapefile
from awrams.utils.gis import ShapefileDB
# Create a shapefile database from the file
myShapefile = "INSERT PATH TO YOUR SHAPEFILE"
mySDB = ShapefileDB(myShapefile)
# Inspect contents of shapefile
shapefile_df = mySDB.get_records_df()
shapefile_df


my_extent = mySDB.get_extent_by_field('key_field','field_value', parent_extent=edef)
```

Show location of Catchment

Another useful functionality is the ability to visualise the location of the catchment of interest relative to a recognisable extent of Australia. Here we use the function '*show_extent()*' together with a bounding box to show where in NSW the Gunning catchment is located.

```
# Display catchment within the contaxt of a bounding box
vis.show_extent(gunning,results.extent.icoords[-30:-40,142:154])
# Display catchment with reference to the full continent
vis.show_extent(gunning,results.extent)
# Display just catchment grids
vis.show_extent(gunning,gunning)
```



5.1.10. Changing the format of the plots

Formatting the plot axes

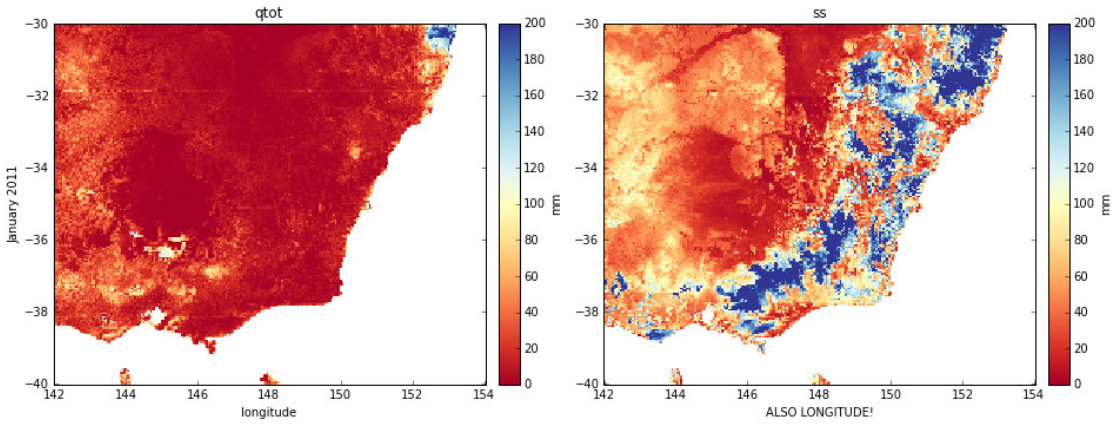
The following commands give examples of how to manipulate the appearance of maps.

Get the range of data for the selection using the function 'get_data_limits'

```
# Capture the variables in v
v = results.variables.qtot,results.variables.ss
q = results[v,'jan 2011',results.extent.icoords[-30:-40,142:154]]
q.get_data_limits()
```

```
(0.0, 412.8)
```

```
# Set colour range limits and horizontal axis labels
q.spatial(clim=(0,200),xlabel="longitude")
# Access underlying axes to modify one plot separately
# Capture grid view plotting settings through the .mpl property of qgridview = q.mpl
# Select the plot settings of the second plot on the first row i.e. [0,1]
view = gridview.children[0,1]
# Change the label on that plot only
view.ax.set_xlabel("ALSO LONGITUDE!")
vis.plt.show()
```

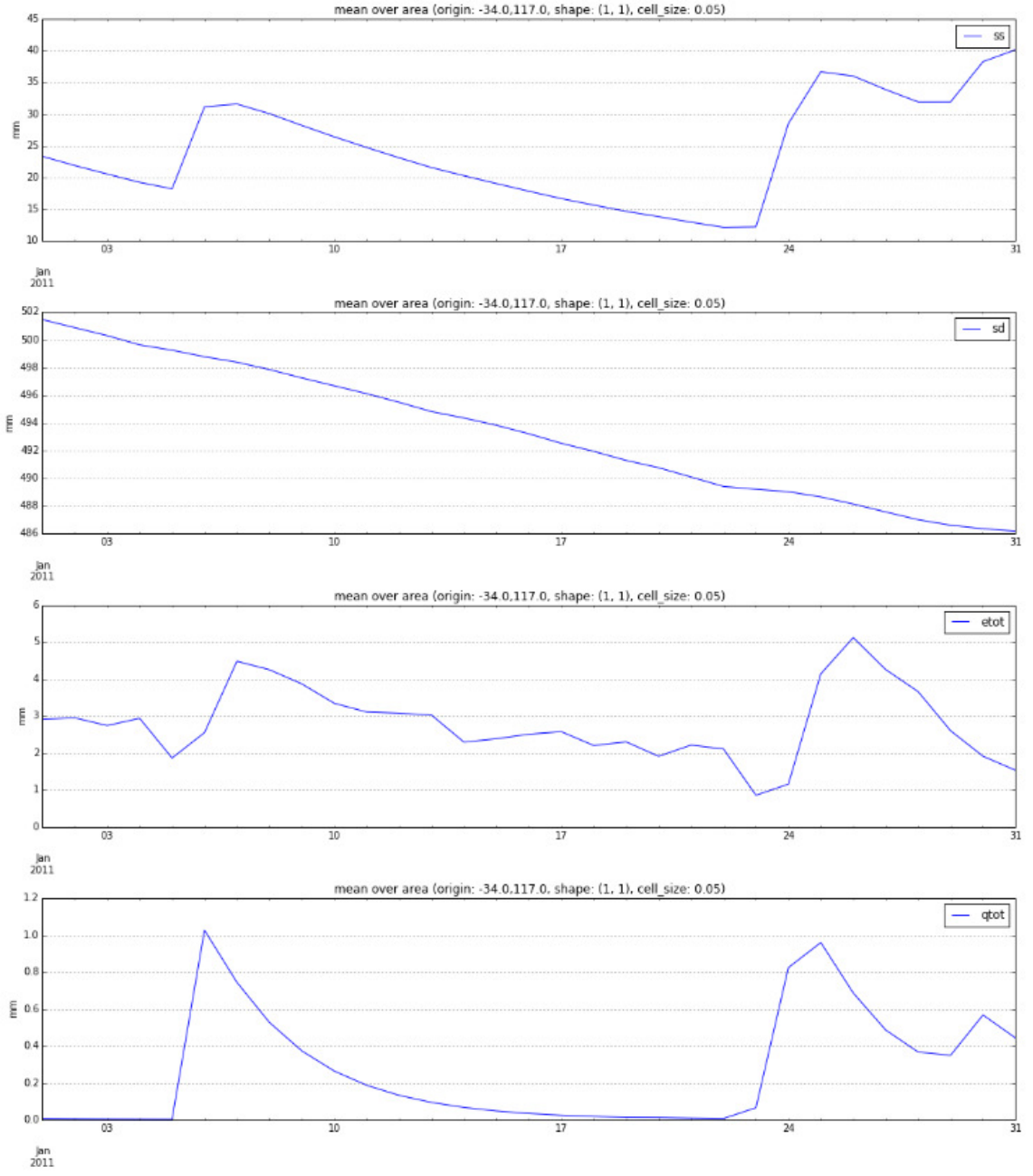


5.2. Time-series plots for selected variables, locations or regions

5.2.1. Plotting a time-series for a single location

To generate a time series plot of an output for an individual grid cell, specify the coordinates of the grid cell within the command, results[variable,time,location].timeseries().

```
p = 'jan 2011'
e = results.extent.icoords[-34,117]
results[:,p,e].timeseries()
```



5.2.2. Plotting a time-series of aggregated values over a catchment

By default, the average over a catchment will be produced if you provide a catchment extent

```
edef = results.extent
v = results.variables.qtot,results.variables.ss
p = 'jan 2011'
e = default_catchments.get_extent_by_field('GaugeName','Mittagang Crossing',results.extent)


results.variables.qtot.data.shape,results.variables.qtot.agg_data.shape
```

5.2.3. Formatting time-series graphs

The following code shows some examples of how graphs can be formatted and appearance of graphs can be changed. First, we plot daily runoff for a single cell:

```
v = results.variables.qtot
e = vis.extents.from_cell_coords(-34,117)
p = jan 2011
q = results[v,p,e]
q.timeseries()
```

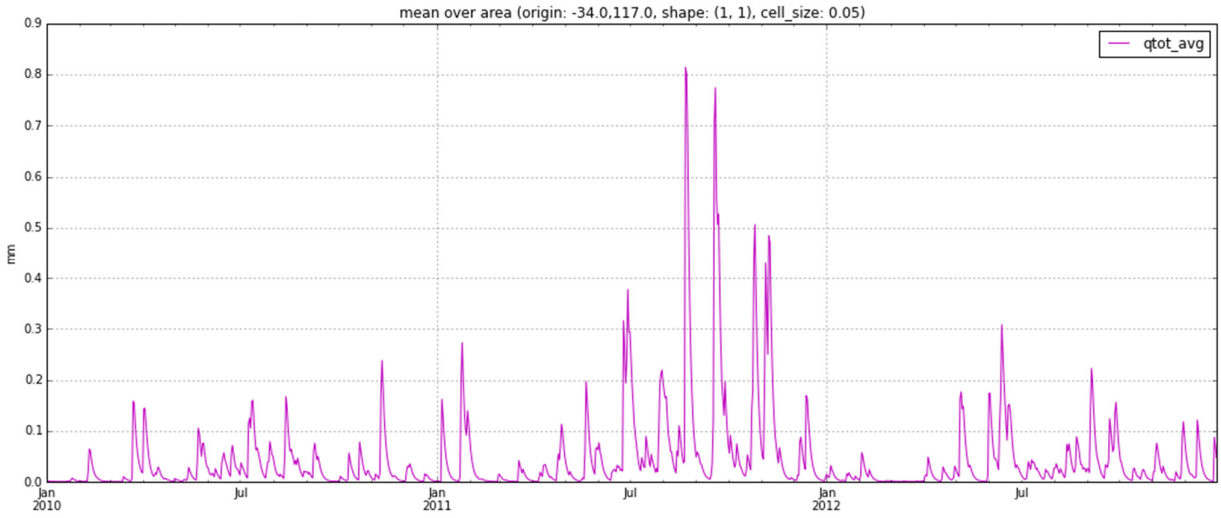
Additional settings for the graph above are shown below:

5.2.4. More formatting options

Sometimes, the default appearance might not be satisfactory. This example shows how to modify one of the plotted lines if we are not happy with the default colours.

```
v = results.variables.qtot
e = results.extent.icoords[-34,117]
p = 'jan 2011'
q = results[v,p,e]
q.timeseries()
```

```
q.timeseries() #ylim=(0,20))  # q is an object that carries data as well as matplotlib configurations
ax = q.mpl.children[0,0].ax  # here we individually access the axis of the first plot generated
lines = ax.get_lines()
line = lines[0]
line.set_color('m') # changing the colour of the first line of the plot
leg = ax.get_legend()  # getting the legend specs
legline = leg.get_lines()[0] # get the specifications of the first line in the legend
legline.set_color('m') # set the color of the legend line
```


mean over area (origin: -34.0,117.0, shape: (1, 1), cell_size: 0.05)
qtot_avg
mm
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Jan
2010
Jul
Jan
2011
Jul
Jan
2012
Jul

6. Extract point or catchment average data from AWRA-L grids

The following section provides some commands to quickly extract catchment averaged data or other regional subsets of gridded data from the model inputs or outputs.

1. Import required libraries

```
from awrams.utils.processing.extract import extract_from_filemanager
from awrams.utils.io.data_mapping import SplitFileManager

from awrams.utils.gis import ShapefileDB, CATCHMENT_SHAPEFILE
catchments = ShapefileDB(CATCHMENT_SHAPEFILE)

import awrams.utils.datetools as dt
from awrams.utils import extents

from awrams.utils.settings import TRAINING_DATA_PATH
import os

# Plotting
from matplotlib import pyplot as plt
%matplotlib inline
```

19. Extract and spatially aggregate catchments

In this example, we will extract the daily rainfall for July 2010-June 2011 for 3 catchments identified by StationIDs 204007, 421103, and 003303.

```
# Capture files of the same variable
var_name = 'rain_day'
data_path = TRAINING_DATA_PATH + '/climate/BOM_climate/rain_day'
pattern = data_path + '/%s*' % var_name
```

SplitFileManager is a Class object that allows reading and storing data from file. The function *open_existing()* is able to collate the data from multiple netcdf files, given a certain naming pattern and variable name.

```
sfm = SplitFileManager.open_existing(data_path,pattern,var_name) # The sfm tool needs the full path to work
```

Specify the period and extents of interest (after first extracting the default extent)

```
# Specify period, parent extent  and collate all extents
```

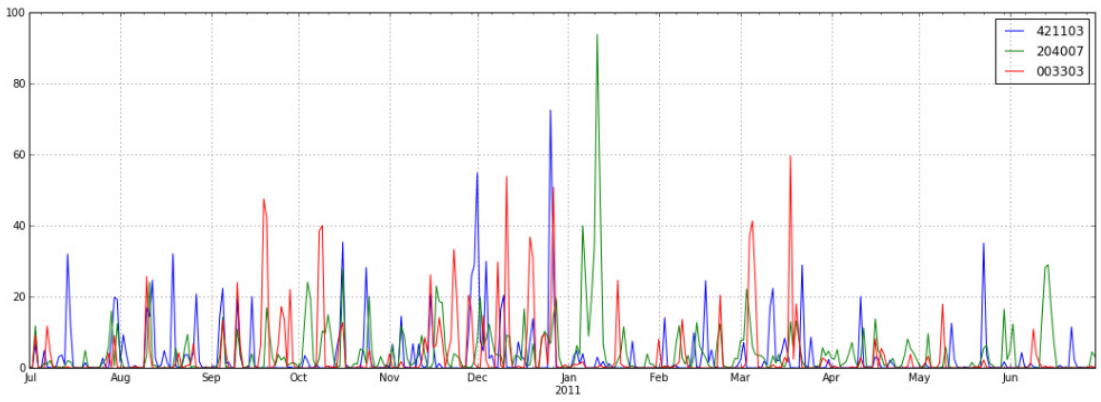
```
period = dt.dates('jul 2010 - jun 2011')
georef = sfm.get_extent()
extent_map = {'204007':catchments.get_extent_by_field('StationID','204007',georef),
              '421103':catchments.get_extent_by_field('StationID','421103',georef),
              '003303':catchments.get_extent_by_field('StationID','003303',georef)}
```

Now we have the extents and period we can extract the data into a dataframe. The *extract_from_filemanager()* function is able to read in the data stored in the *sfm* variable, extract the data over the required spatio-temporal extent and aggregate to a single time series per element of the extents list (*extent_map*). The default aggregation method is the area-weighted mean of all the values within an extent.

```
# Extract the data
df = extract_from_filemanager(sfm,
        extent_map,
        period)
df
```

Finally, plot the daily precipitation for the 3 catchments as a time-series.

```
ax = plt.figure(figsize=(18,6)).gca()
df.plot(ax=ax)
```



20. Extract catchment gridded data

Rather than extracting spatially aggregated data, this example looks at extracting the data for all grid cells in one catchment of interest (here StationID 421103).

Note that the libraries osgeo and gdal are required to process shapefiles. These should be included if you used the conda install for the AWRA-CMS.

First, the period and catchment (extent) of interest are defined.

```
# Specify date
period = dt.dates('jul 2010')
```

```
# Specify catchment
catchment = extent_map['421103']
catchment.cell_count
```

```
9
```

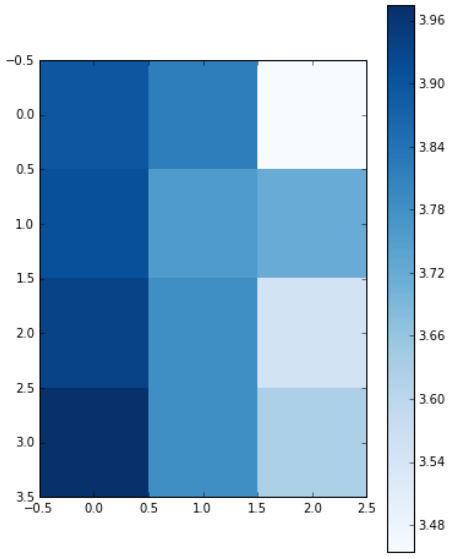
The data for the period (July 2010) and catchment (421103) of interest is then extracted from the *sfm* variable using the sfm.getdata() command. Note that this gives a 4 dimensional array of data (time,x,y,variable). Also note that the data extracted is for the rectangle containing the catchment.

```
data = sfm.get_data(period,catchment)
sfm.close_all()
data.shape,catchment.cell_count # You will note that the data extracted is for the rectangle containing the catchment
```

```
((31, 4, 3), 9)
```

To plot a map of the results for mean daily rain in July 2010 for catchment 421103:

```
plt.figure(figsize=(6,8))
im = plt.imshow(data.mean(axis=0),interpolation='None',cmap=plt.get_cmap('Blues'))
plt.colorbar(im)
```



7. Calibration

As a conceptual model, AWRA-L includes a large number of parameters whose optimal values will depend not only on the objective of the modelling run (i.e. best estimates of any of flow, evaporation or soil moisture, or combination thereof) but also on the sites and periods being modelled and the quality of the available data. Hence, for any given purpose, it is essential to be able to calibrate the values of these parameters. The AWRA-CMS package provides functionality to undertake the required multi-criteria calibration process. This can be done using an automated procedure which compares modelled and observed responses and seeks to minimise the differences between them through the use of an objective function. This section guides the user through an example of calibrating the AWRA-L model so as to optimise the fit to discharge at multiple gauged locations according to their averaged Nash-Sutcliffe Efficiency score.

Reflecting the calibration approach used during the AWRA-L model development (Frost et al. 2016a, Frost el al. 2016b), the calibration functionality in AWRA-CMS is specifically set up to easily calibrate the model for an optimal fit at *multiple sites* and to *multiple observed datasets* (e.g. observed streamflow, remotely sensed surface soil moisture and remotely sensed actual evapotranspiration) with a *single set of parameters*. This (automated) calibration process is flexible and allows the user to specify aspects such as the optimisation routine and constraints, and the objective function. Furthermore, the required ability to calibrate the model to 100+ catchments using very large datasets also means that the functionality has been designed to allow for deployment into high-performance computing environments such as the NCI (i.e by utilising multiple processors).

Note: Only test soil moisture and ET datasets are provided within the AWRA-CMS package. Upon registration, full version 5 calibration and benchmarking data are separately made available to users.

The following steps are covered in this section:

1. Import required libraries
2. Set up calibration configuration
    - 2.1. Specify the catchments to be calibrated
    - 2.2. Specify running and calibration periods
    - 2.3. Amend the nodegraph (forcings and parameter ranges)
    - 2.4. Select Optimiser, e,g shuffled complex evolution
    - 2.5. Define objective functions and observed data
    - 2.6. Combine into a specification dictionary
3. Run model calibration
4. Extract best parameter set
5. Visualise calibration process outputs
6. Compare modelled outputs to observations

7.1. Import required libraries

```
import pandas as pd
```

```
from os.path import join
import h5py

from awrams.models.awral.model import AWRALModel
from awrams.models.settings import TRAINING_DATA_PATH

from awrams.utils.nodegraph import nodes, graph

from awrams.utils import gis
from awrams.utils import datetools as dt
from awrams.utils import extents
from awrams.utils.settings_manager import get_settings
from awrams.utils.gis import ShapefileDB, CATCHMENT_SHAPEFILE
from awrams.calibration.optimizers import sce
from awrams.calibration.objectives import test_objectives as tobj
from awrams.calibration.support import *
from awrams.calibration import cluster
from awrams.calibration.launch_calibration import run_from_pickle
```

```
# Instantiate the AWRA-L model
awral = AWRALModel()
```

7.2. Calibration configuration

A calibration run is configured by defining five fundamental aspects: the extent, the period, the input nodegraph, the optimisation specifications and the objective function definition.

Extents: This is the list of catchments or sites to use in calibration, converted into an extents dictionary as previously described in Section 3.5 The model will effectively run only over the specified extents.

Period: The periods over which the model is run and the objective function evaluated (if different) need to be specified

Input nodegraph: This contains the model settings as described in Section 3.4. The forcings, initial states, gap-filling methods, ranges of parameters to be calibrated are all set in the nodegraph.

Optimisation specifications: The optimiser settings are one part of the calibration specific settings, i.e. the optimiser to be used and specifications for using that optimiser

Objective function definition: The objective function to be used and the observations against which the model performance is optimised (for example discharge gauge data) are provided here.

These five components are then combined into a single set (implemented as a python dictionary) which is the complete configuration for the calibration run.

An example of how these components are put together is provided hereunder. In this simple calibration experiment, the model runoff outputs for two catchments are fit to the observed daily flows at the corresponding gauge locations using the average NSE as the objective function.

Note that AWRA-L v5.0 is calibrated simultaneously to remotely sensed, catchment averaged, top-layer soil moisture and evapotranspiration as well as observed specific discharge (i.e. discharge normalised by catchment area). Details of the calibration can be found in Viney et al (2015) and Frost, Ramchurn and Smith (2016).

7.2.1. Specify the calibration sites

The first step is to specify the extents of the calibration catchments, which can be any combination of point locations and catchments in Australia. In this example, two catchments for streamflow optimisation are chosen from the dataset of 782 Australian catchments (Zhang et al. 2013) included in the package.

Note that for calibration of the current operational version of AWRA-L (v5.0), a set of 586 catchments (295 for calibration, 291 for validation) of area less than 5000 km^2, with sufficient length of record, minimal land-use change, damming or extractions affecting the catchment and independence from other gauged catchments were used.

Of course the user is able to specify their own shapefile to define catchment extents and their own observation data instead.

```
# Get the default Australian extent, which is the extent from which the catchments are defined.
def_extent = extents.get_default_extent()

# Import the shapefile containing locations to be calibrated
# In this instance, CATCHMENT_SHAPEFILE can be replaced by the user's preferred shapefile
calvalshapefile = ShapefileDB(CATCHMENT_SHAPEFILE)
```

To optimise multiple catchments simultaneously (the approach used for AWRALv5.0), a dictionary of the multiple extents needs to be created. Here, this is done by converting a list of catchment IDs (*cal_catchments*) to be calibrated into extent objects using the *.get_extent_by_field* function, where "StationID" is in this case a unique value identifier of catchments within the shapefile, and storing it in the *cal_dict* dictionary.

```
# Create a dict with multiple extents
cal_dict = {}
cal_catchments = ['105001','145003']

for catchment in cal_catchments:
    cal_dict[catchment] = calvalshapefile.get_extent_by_field('StationID', catchment.zfill(6), parent_extent=def_extent)
cal_dict
```

Alternatively, a single catchment extent could be defined from the default catchment data set as follows:

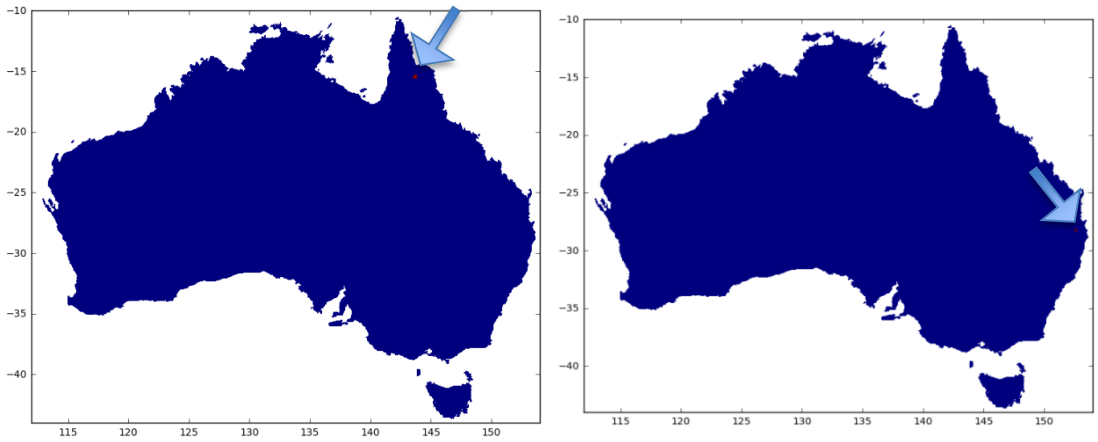
```
# Single catchment
my_extent = calvalshapefile.get_extent_by_field('StationID', '003303', parent_extent=def_extent)
my_extent
```

The visualisation tools can be used to check the locations of the selected catchments in Australia.

Note that the chosen catchments are very small and hard to see on the resulting maps. One is in far north Qld and the other on the Qld/NSW border.

```
# Check out where the catchments are
import awrams.visualisation.vis as vis
import awrams.visualisation.results as res

import awrams.utils.extents as extents
vis.show_extent(cal_dict['105001'],def_extent)
vis.show_extent(cal_dict['145003'],def_extent)
```



7.2.2. Specify running and calibration period

For calibration, both a run period (*run_period*) and an evaluation period (*eval_period*) can be chosen. For example, the run period might start 1 year earlier than the evaluation period if a model spin-up period is required. The objective function for calibration will only be calculated for the evaluation period. Both these periods need to be defined by the user, as per below:

```
run_period = dt.dates('2008 - 2010')
eval_period = dt.dates('2009 - 2010')
```

7.2.3. Amending the input nodegraph

Firstly, the default input nodegraph *input_map* is created.

```
input_map = awral.get_default_mapping()
```

As for simulation (Section 4), the nodegraph specifies the model inputs and parameters, as follows:

1. **the model inputs** : Section 4.1.2 shows how to confirm the path to the forcing inputs.

```
change_path_to_forcing(input_map)
```

2. **the model infilling data**: Section 4.2.6 shows how to ensure that any missing data in the forcing is appropriately infilled. In this case, 2009 contains missing solar radiation, so the data will be infilled with climatological estimates.

```
input_map = insert_solar_climatology(input_map)
```

3. **the model initial states**: Section 4.1.6 shows how to initialise the model run with more appropriate starting states than the default option (half full). The initial states of the various soil moisture stores exert a strong influence on the model response especially in short calibration runs. In this case the model can for example be initialised with the stored values for 30th November 2009, which are the default data used by the *initial_states_from_dict* function.

```
period = dt.date(1-dec-2009)
input_map = initial_states_from_dict(input_map,period,edef)
```

4. **the model parameter ranges**: Section 4.1.2 shows how to adjust the parameter ranges and to confirm which parameters need to be calibrated or are fixed.

```
change_model_parameters(input_map, param_df)
```

7.2.4. Specifying the optimiser options

Given the large amount of parameters, multiple objectives and sites to calibrate, calibration using an automated optimiser is recommended. The AWRA-L calibration implementation in the AWRA-CMS is focused on a framework enabling computational efficiency via parallel processing (i.e. using separate threads/cores on a computer). For large datasets the calibration functionality allows simulating points/catchments independently of one another (on separate nodes, e.g. on a high performance environment like the NCI).

The AWRA-CMS currently implements both the SOBOL and Shuffled Complex evolution (SCE) (Duan et al. 1993, https://doi.org/10.1007/BF00939380) optimisers. For the purposes of AWRA-L model calibration, the SCE optimiser is recommended. The SCE method starts off with a user-defined number of complexes, where a complex is a subpopulation of parameters. The process then first "evolves" each complex through a number of specified evolutions (searching outside of the original complexes for offsprings and replacing the original values) and then "shuffles" the returning complexes (remixing available complexes). This process is iterated until the best value functions obtained from the complexes stops improving according to the criteria provided.

Simple implementations of AWRA-CMS evaluators are parallelized by the number of complexes. Thus, each complex has a number of iterations and each complex in a shuffled population (specified by n_complexes in the ShuffledOptimizer constructor), is launched in its own process.

Note that the AWRA-CMS code is modular and allows expert users to implement other optimisation strategies within the existing parallelisation framework. The calibration process uses an approach that allows the user to utilise a specifiable number of computational nodes on high performance computing architectures.

To use the SCE optimiser, there are some options to be configured.

- n_offspring is number of offspring meaning the number of new points generated from an existing complex (number of unique new parameter sets). Generally, should be between 1 and 5, usually 1.
- n_evolutions is the number of evaluations done on each offspring. Generally, should be between 1 and 5, usually 5.
- elitism is usually left at the default setting, 2.0. See paper for details (Duan et al. 1993).
- n_complexes is the number of complexes. In practice we tend to use around 14
- max_nsni is the max number of shuffles with no improvement. We have used values of 10 up to 1000. If the calibration terminates without converging to a solution, you need to increase this.
- Min_complexes is the minimum number of complexes to be run. In practice this could be higher than 1, particularly for single catchment calibration.
- Max_eval is the maximum number of evaluations that can be run. We tend to use around 20000 in practice.

There is also a default setting for tolerance (i.e. how much an iteration must improve by to continue running the optimisation). The default is set at 1 % improvement over 5 shuffles of the complexes. There are also a number of other things that can configured to use the SCE. Please see the code and Duan et al.(1993).

```
evolver_spec = EvolverSpec(sce.CCEvolver,
                    evolver_run_args=dict(n_offspring=1,n_evolutions=5,elitism=2.0))


optimizer_spec = OptimizerSpec(sce.ShuffledOptimizer,
                        evolver_spec=evolver_spec,
                        n_complexes=5,
                        max_nsni=500,
                        min_complexes=1,
                        max_eval=2000)
```

7.2.5. Defining the objective function

The definition of the objective function requires the observation data to be used, as well as the calculation (e.g. bias, NSE, etc.) through which to establish the model fit (i.e. the objective function).

1. Specify the observation data

The observed data to be compared with the simulated data needs to be specified. This data can be contained in a single file (if comparing a single variable) or multiple files for multi-objective formulations. In the example below, the path to a .csv file with observed specific runoff (river discharge divided by the catchment area) is specified together with the variable name, *qtot*, and put into a dictionary called *observations*. Note that the units of the observation data need to be consistent with the

model outputs being compared, e.g. in the sample data provided, discharge data is in mm (specific runoff), soil moisture data is expressed in percent full and evapotranspiration data in mm.

```
observations = dict(qtot = '../../test_data/calibration/q_obs.csv' )
```

2. Define or select the objective function

The calibration process undertakes the minimisation of the objective score, which is a single value calculated from the comparison of the simulated and observed data. The objective function (which calculates this score) definition thus depends on the whether the calibration uses one or multiple sites, and how many types of observations are used.

For example as Table 1 indicates, calibrating single catchment to one type of observation requires the definition of a "local_single" objective function. On the other hand, calibrating multiple catchments to multiple types of data will require the definition of a "global_multi" objective function, which needs to be a combination of "global_single" or "local_multi" objective functions, which themselves depend on how the local_single objective function is defined for each particular type of data.

Table 1: Objective function definition requirements

Type of objective functions needed		Number of catchments	
		One (local)	Many (Global)
Types of observation data	One (single)	local_single →	global_single
	Many (Multiple)	local_multi →	global_multi

In the current example, where the model is calibrated singarly to flow at two sites, both "local_single" and "global_single" type objective functions are needed. Samples of such functions have been packaged in the Test_objectives module e.g. NSE, bias, Pearson's correlation, Viney's F score (described further below).

As shown hereunder the local_single and global_single objective functions are respectively specified as *local_objspec* and *global_objspec*. This information is then combined into the *objective_spec* object. Note that in this case, the *TestLocalSingle* function is the NSE and *TestGlobalSingle* is simply 1 – the mean of NSE of all the sites.

```
from awrams.calibration.objectives import test_objectives as tobj
local_objfspec = ObjectiveFunctionSpec(tobj.TestLocalSingle)  # this function loads up the ObjFunc which comes from a wrams.calibration.support
global_objfspec = tobj.TestGlobalSingle


objective_spec = ObjectiveSpec(global_objfspec,
                               local_objfspec,
                               observations,
                               eval_period)
```

Note that the operational AWRA-L model is calibrated to flow (at 295 gauging stations), soil moisture (AMSRE satellite estimates averaged across 294 catchments) and actual evaporation (CMRS ET averaged across 295 catchments). Each type of data has its own "local_single" evaluation.

For flow (qtot), the objective function combines daily and monthly specific runoff variability (Nash-Sutcliffe Efficiency, *E*) and bias, *B*:

$$F = (E_d + E_m)/2 - 5 \left| \ln(1 + B) \right|^{2.5}$$

where E_d and E_m are respectively the Nash-Sutcliffe efficiencies of daily and monthly streamflow, and *B* is the bias (total prediction error divided by total observed streamflow) – see definitions in Section 8.3 and Viney et al (2009).

For soil moisture (wunsat):

$$F = r_d$$

where r_d is the Pearson's correlation coefficient for daily data (see Section 8.3 for definition).

For evapotranspiration (etot):

$$F = r_m$$

where r_m is the Pearson's correlation coefficient for monthly data (see Section 8.3 for definition).

For all three variables, the "global_single" approach to obtaining a single value is to take the mean of the 25th, 50th, 75th and 100th percentiles of the "local_single" (*F)* values of the calibration catchments.

$$F_{grand} = (F_{25} + F_{50} + F_{75} + F_{100})/4$$

Finally, the "global_multi" formulation is to perform a weighted sum of the "global_single" (F_{grand}) scores for the three variables:

Objective Function value = $0.7*F_{grand}$ *(qtot)* + $0.15*F_{grand}$ *(wunsat)* +$0.15*$ F_{grand} *(etot)*

This objective function value is then maximised in calibration.

For further details of the model, parameters and calibration approach see Viney et al (2015) and Frost, Ramchurn and Smith (2016).

7.2.6. Build specifications dictionary for the calibration

After putting together the building blocks above, the input configuration to run the calibration is assembled into a specification dictionary for the calibration. The entire calibration process needs to be saved in an hdf type log file (*calres.h5*, in this example) for future analyses.

```
# Create the calibration specification dictionary
'''
User specifiable calibration description
'''
cal_spec = {}
cal_spec['optimizer_spec'] = optimizer_spec
cal_spec['objective_spec'] = objective_spec
cal_spec['extent_map'] = cal_dict
cal_spec['run_period'] = run_period
cal_spec['model'] = awral
cal_spec['node_mapping'] = input_map
cal_spec['logfile'] = './calres.h5'
```

The calibration configuration can then be saved in a binary file (called a pickle file with extension *.pkl*). Being able to save the specification into a standalone file provides the advantages that: 1) it facilitates the reproduction of this calibration experiment at a later stage, and 2) enables the calibration specifications to be transferred to other locations (e.g. NCI).

```
# Build the calibration configuration to be run on a computer cluster and save it to a file containing all the calibration specifications
import sys
import mpi4py
from awrams.calibration import cluster
from awrams.calibration.launch_calibration import run_from_pickle
cluster.build_pickle_from_spec(cal_spec,1,'test_cal.pkl')
```

7.3. Run the calibration

Finally, the function *'run_from_pickle*' is used to pick up the pickle file containing the configurations for the calibration run (saved in the previous step) and start the calibration process. As the calibration runs, progress messages will be displayed.

```
cal = run_from_pickle('./test_cal.pkl')
```

```
Output file:
n
s:  1975

 evaluating initial population
{}
...
 running
{'n_eval': 264, 'n_shuffle': 2, 'best_params': Score: 0.787215807067, Params: [ 0.0
    3941318  0.03789217  0.07214855  0.82408072  0.92199983  0.96857639
  3.08393983  1.5451794   2.00603256  0.049638    0.46869477  8.9847855
  0.47285554  2.04910222  1.67613136  0.34570536  0.34886143  0.62934172
  0.12895572  1.62575523  1.93201036], Meta: None}

 running
```

```
{'n_eval': 336, 'n_shuffle': 8, 'best_params': Score: 0.787215807067, Params: [ 0.0
    3941318  0.03789217  0.07214855  0.82408072  0.92199983  0.96857639
  3.08393983  1.5451794   2.00603256  0.049638    0.46869477  8.9847855
  0.47285554  2.04910222  1.67613136  0.34570536  0.34886143  0.62934172
  0.12895572  1.62575523  1.93201036], Meta: None}

 running
{'n_eval': 388, 'n_shuffle': 14, 'best_params': Score: 0.787215807067, Params: [ 0.
    03941318  0.03789217  0.07214855  0.82408072  0.92199983  0.96857639
  3.08393983  1.5451794   2.00603256  0.049638    0.46869477  8.9847855
  0.47285554  2.04910222  1.67613136  0.34570536  0.34886143  0.62934172
  0.12895572  1.62575523  1.93201036], Meta: None}

......
```

7.4. Extract best parameter set

The calibration process creates a file containing all the parameters and objective scores of each iteration of the calibration. This file (*calres.h5*) can now be opened and analysed by loading the data into the *cr* object.

```
cr = CalibrationResults('./calres.h5')
```

The cr object has many useful functions and properties (accessible via dot completion). For example, to see the list of parameters calibrated:

```
# These are the parameters we calibrated
cr.parameter_names
```

```
['cgsmax_hrudr',
 'cgsmax_hrusr',
 'er_frac_ref_hrudr',
 'fsoilemax_hrudr',
 'fsoilemax_hrusr',
 'k0sat_scale',
 'k_gw_scale',
 'k_rout_int',
 'k_rout_scale',
 'kdsat_scale',
 'kr_coeff',
 'kssat_scale',
 'ne_scale',
 'pref_gridscale',
 's0max_scale',
 's_sls_hrudr',
 's_sls_hrusr',
 'sdmax_scale',
 'slope_coeff',
 'ssmax_scale',
 'ud0_hrudr']
```

To see how the parameter values changed over the evolution of the calibration, a dataframe of the values can be extracted from the variable, *cr*, to create the data frame, *params_all*. From this, the best parameter set can be selected (*best_params*).

```
# Get a DataFrame of the parameter values over time (iterations)
params_all = cr.get_parameter_values()
```

```
# Obtain the best (minimum score) parameter set
best_params = params_all.iloc[cr.best_param_index()]
best_params


best_params.to_csv('cal_params_test.csv')
```

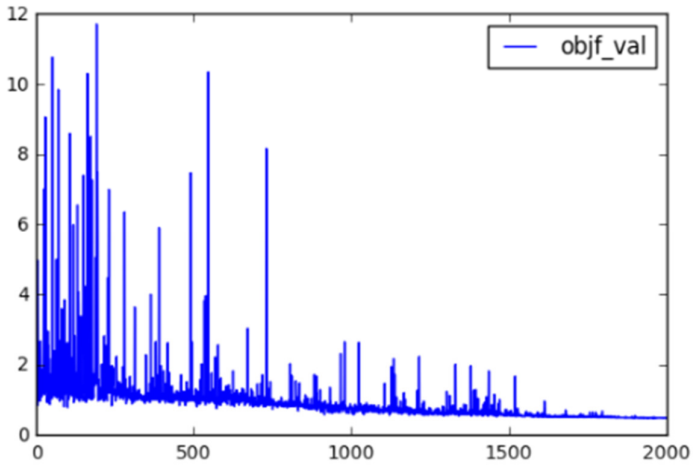
Output:

```
cgsmax_hrudr          0.020237
cgsmax_hrusr          0.032958
er_frac_ref_hrudr     0.066848
fsoilemax_hrudr       0.906176
fsoilemax_hrusr       0.980338
k0sat_scale           6.067147
k_gw_scale            0.808467
k_rout_int            2.585721
k_rout_scale          0.760175
kdsat_scale           0.023155
kr_coeff              0.575623
kssat_scale           0.467066
ne_scale              0.859348
pref_gridscale        2.923311
s0max_scale           2.314816
s_sls_hrudr           0.044190
s_sls_hrusr           0.454831
sdmax_scale           0.906147
slope_coeff           0.745054
ssmax_scale           1.177899
ud0_hrudr             1.783924
Name: 1999, dtype: float64
```

7.5. Visualise the calibration outputs

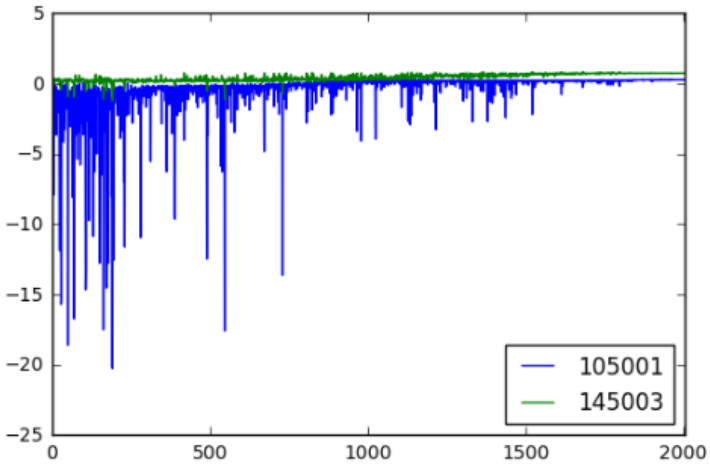
The evolution in the objective function as the calibration progresses can be inspected by plotting the global scores (objective functions) against the number of iterations. As can be seen, for this calibration, the maximum number of iterations, 2000, was run, but the change in the objective function was minimal for the last 400 or so iterations.

```
# Examine the change in global objective function values over time
gscores = cr.get_global_scores()
gscores.plot()
```



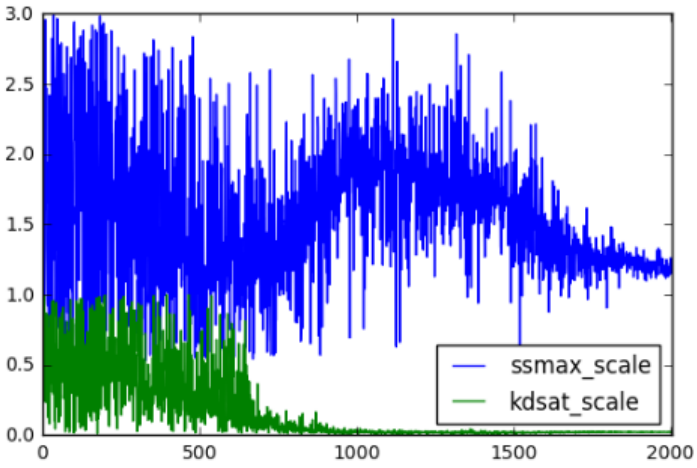
While the previous scores were the overall combined score for all catchments (F_{grand}), the evolution of individual catchment scores can also be viewed:

```
#Get some local (catchment level) scores
qnse_l = cr.get_local_scores('qtot_nse')
```



It is also possible to see how parameters evolve during a calibration.

```
#Observe the evolution in parameter values over iterations
#Great for examining behaviour of parameter space for different optimizers!
params_all[['ssmax_scale','kdsat_scale']].plot()
```



7.6. Comparing modelled outputs to observations

The outcome of any one of the calibration iterations can be used to quickly undertake a model run, which can then be plotted directly against the observed data using Python Pandas functionality.

The OnDemand simulator (Section 4) is used to load up any parameter set. In this case the *best_params* variable already contains the best parameter set obtained from calibration (Section 7.4).

```
#Load up an OnDemand simulator to run the model
from awrams.simulation import ondemand
sim = ondemand.OnDemandSimulator(model,node_mapping)
```

Here we specify that the parameter set to use in the simulation is the best parameter set from the calibration run.

```
# Set the new (calibrated) parameters
sim.input_runner.set_parameters(best_params)
```

Run the model only for catchment with ID 105001 and capture the results in the *sim_res* variable.

```
sim_res = sim.run(run_period,cal_dict['105001'])
```

Show the results in a table. This table will actually contain the time series deep drainage (*dd*) of each grid cell located in the catchment.

```
pd.DataFrame(sim_res['dd'])
```

If so desired, the simulated results for catchment 105001 can be saved as a csv file.

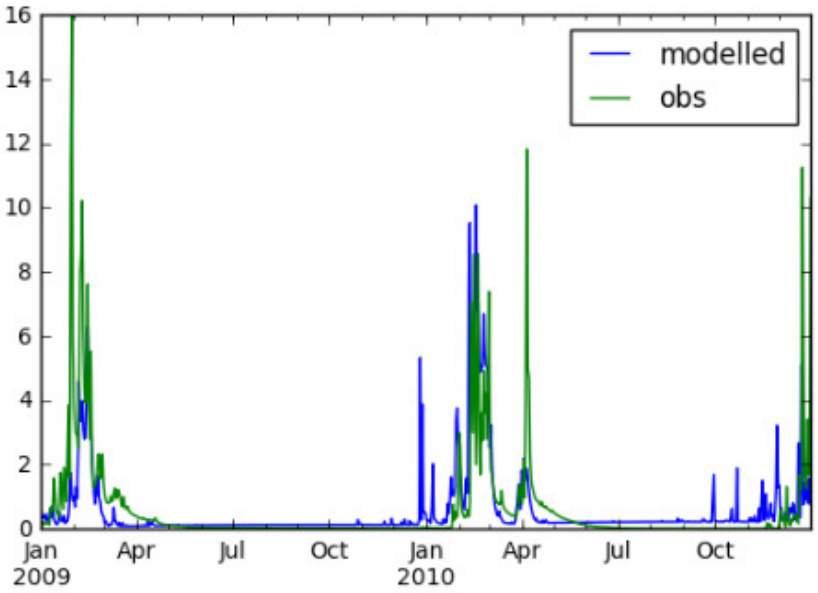
```
sim_res.to_csv('/data/cwd_awra_data/AWRACMS/Training/test_data/105001sim.csv')
```

The above dataframe needs to be converted to a single catchment-averaged value to facilitate comparison with the observed data. This can be done using the function defined below.

```
#Simulations output gridded data; get some convenience aggregation functions
from awrams.calibration.support import aggregate, flattened_areas
def mean_by_extent(in_data,extent):
    weights = flattened_areas(extent)
    return aggregate(in_data,weights,weights.sum())
cat_average_qtot = pd.Series(mean_by_extent(sim_res['qtot'],cal_dict['105001']),index=run_period)
```

The final step is to create a dataframe (*comp_df*) with the modelled and observed time-series for station 105001 side by side and plot the time-series.

```
comp_df = pd.DataFrame()
# Bring in modelled data
comp_df['modelled'] = pd.Series(mean_by_extent(sim_res['qtot'],cal_dict['105001']),index=run_period)
# Load observations for comparison and bring into the comparison dataframe
obs_df = pd.DataFrame.from_csv(observations['qtot'])
comp_df['obs'] = obs_df['105001'][run_period]
# Plot dataframe
comp_df.plot()
```



8. Benchmarking

Benchmarking is the term we use to define testing changes in model performance (i.e. when compared to observations) against current and previous versions of output created by AWRA-L as well as comparisons of model performance from alternative models (e.g. other continental land-surface or conceptual hydrological models). A suite of benchmarking tools are available within the AWRA-CMS. A small set of example datasets are provided with this package but the benchmarking tools are designed to work with comparing any observation dataset with the relevant AWRA-L variable. The current benchmarking data and benchmarking methodology used to evaluate the AWRA-L model are detailed in Frost, Ramchurn and Hafeez (2016). In that document, the following national datasets have been compared with the relevant AWRA-L variables for the latest AWRA-L model version. Subsets of these datasets, with data for only ten catchments, have been provided with the AWRA-CMS package to demonstrate the tools. The benchmarking of the model over these ten catchments is presented in a series of notebooks relevant to each dataset, with just the runoff presented here. The full benchmarking data can be requested by registered users.

AWRA-L Variable	**Description**	**AWRA-CMS Dataset**	**Demo Notebook**
Runoff (Qtot)	Observed streamflow: 786 catchment (Zhang et al., 2013)	runoff	BenchmarkingDemo_QObs
Actual Evapo-transpiration (Etot)	Satellite derived data: CMRSET(Guerschman et al., 2009) Flux tower data: OzFlux (Beringer et al., 2016)	cmrset ozflux	BenchmarkingDemo_CMRS_ET BenchmarkingDemo_OzFlux_ET
Upper Soil Moisture (S0)	Satellite derived soil moisture data: AMSRE (Owe et al., 2008) ASCAT (Bartalis et al., 2007)	amsre ascat	BenchmarkingDemo_wunsat_AMSRE BenchmarkingDemo_wunsat_ASCAT
Surface (S0) and Shallow (SS) Soil Moisture	Soil moisture monitoring networks: SASMAS (Rüdiger et al., 2007) OzNET (Smith et al., 2012)	sasmas oznet	BenchmarkingDemo_SM_SASMAS BenchmarkingDemo_SM_OZNET

The User is alerted to the fact that the soil moisture benchmarking components (SASMAS and OzNET) may further necessitate some configuration settings and metadata adjustment, if for example the file naming convention is different or the observed data is to be modified. The relevant files are located at the following locations: awrams_cm\benchmarking\awrams\benchmarking\config.py, and the folder awrams_cm\benchmarking\awrams\benchmarking\meta

8.1. Comparing AWRA-L Input data with data from other sources

8.1.1. Observation data format

The benchmarking system is designed with the flexibility to compare multiple datasets in the simple format of a csv format. The observation data just needs to be arranged according to an ID and date and there can be gaps in the data, e.g.:

	432157	42434	462374	426258	167210	439704
31/10/2001	76.7064	107.8738			31.7719	
30/11/2001	94.98	114.063			38.598	85.284
31/12/2001	110.4778	138.5948			72.6547	

8.1.2. AWRA-L data

AWRA-L data needs to be extracted at the desired location for comparison with the observed datasets. The user can run the extraction commands detailed in Section 6 to extract the AWRA-L data from the netcdf grids output generated by a simulation.

8.2. Benchmarking Example: Qtot and Streamflow Observations

Compare AWRA-L runoff (Qtot) against observed streamflow (normalised by catchment area to specific runoff). Note that the observed streamflow is provided in csv format. You can use your own data in csv form similar to the example provided. It just needs to have column names matching the names used in extracting AWRA data.

8.2.1. Import required libraries

```
from awrams.benchmarking.benchmark import Benchmark
from awrams.benchmarking.utils import read_id_csv
from awrams.utils import datetools as dt
```

8.2.2. Set up benchmarking configuration

Catchments to be benchmarked

```
data_path = '/data/cwd_awra_data/AWRACMS/Training/test_data/benchmarking/'


catchment_csv = data_path+'/catchment_ids.csv'
```

```
id_list=read_id_csv(catchment_csv)
```

Observations to use

Here we show an example using catchment specific runoff (discharge/Area) for benchmarking.

```
obs_csv = data_path+ '/runoff/Catchment_Qobs.csv'
```

8.2.3. Create the benchmark object

An object of "Benchmark" class is created by defining what variable is to be benchmarked. Everything else gets progressively added, and statistics are calculated when the observation and model outputs are added.

```
q = Benchmark("QObs", "runoff")


# Specify benchmarking period
q.period = dt.dates("1981", "30/12/2011")


# Add observations and catchment subset [the id list needs to be present in the column names of the observation file]
q.load(obs_csv, id_list=id_list)
```

```
q.sites
```

```
['113004',
 '111101',
 '4508',
 '109001',
 '108003',
 '112102',
 '107002',
 '105001',
 '116008']
```

```
from awrams.utils.gis import ShapefileDB, CATCHMENT_SHAPEFILE
from awrams.utils import extents
```

```
extents.get_default_extent()
```

```
origin: -10.0,112.0, shape: (681, 841), cell_size: 0.05
```

```
calvalshapefile = ShapefileDB(CATCHMENT_SHAPEFILE)
cal_dict = {}
for catchment in q.sites:
    cal_dict[catchment] = calvalshapefile.get_extent_by_field('StationID', catchment.zfill(6), parent_extent=extents.get_default_extent())


cal_dict
```

```
import awrams.visualisation.vis as vis
import awrams.visualisation.results as res

import awrams.utils.extents as extents

for catchment in cal_dict.keys():
    vis.show_extent(cal_dict[catchment],extents.get_default_extent())
```

8.2.4. Add models to be benchmarked

The model data to be benchmarked originates from pre-processed AWRA-L data extracted from the simulation grids at the observation locations using the AWRA-CMS extract tool.

```
# Reading data from pre-processed csv
csv_data = data_path+'/runoff/awral_qtot_avg.csv'
q.add_model("AWRAMSI_v4_0_AWRAL", data_csv=csv_data)
```

```
csv_data = data_path+'/runoff/AWRAMSI_v5QES_AWRAL_qtot_avg.csv'
q.add_model("AWRAMSI_v5_0_AWRAL", data_csv=csv_data)
```

```
# Sample code here to extract data on any catchment in the default catchment dataset
# A similar approach would work with a user defined catchment shapefile.

import pandas as pd
from awrams.utils import extents
from awrams.utils.io.data_mapping import SplitFileManager
from awrams.utils.processing.extract import extract_from_filemanager
from awrams.utils.gis import ShapefileDB, CATCHMENT_SHAPEFILE

catchments = ShapefileDB(CATCHMENT_SHAPEFILE)

var_name = 'qtot_avg'
model_data_path = '/data/cwd_awra_data/awra_test_outputs/Scheduled_v5_awraprod1/'
period = dt.dates('jul 2010 - jun 2011')

 pattern = data_path + '/%s*' % var_name
 sfm = SplitFileManager.open_existing(model_data_path,pattern,var_name)
 georef = sfm.get_extent()
 extent_map = {}
```

```
for site in q.sites:
    extent_map[site] = catchments.get_extent_by_field('StationID',site.zfill(6),georef)

df = extract_from_filemanager(sfm, extent_map, period)
df

df.to_csv('./for_streamflow/'+ var_name+ '.csv'
```

Show list of loaded or selected models

The list of loaded models is available with activated dropdown by typing "q.models.

```
q.benchmark.selection
```

```
['AWRAMSI_v4_0_AWRAL', 'AWRAMSI_v5_0_AWRAL']
```

You can "select" or "unselect" models for display

```
q.benchmark.selection.AWRAMSI_v4_0_AWRAL.unselect()
q.benchmark.selection.AWRAMSI_v4_0_AWRAL.select()
```

8.3. View the statistics tables

A full explanation of the benchmarking statistics used to assess model performance is provided in Frost, Ramchurn and Hafeez (2016). In summary the following statistics are built into the AWRA-CMS benchmarking statistics and are used to evaluate AWRA-L.

Relative bias (B):

$$B_i = \sum_{t=1}^{T_i} \frac{Q_o^{ti} - Q_m^{ti}}{\bar{Q}_o^i}$$

Monthly and daily Nash-Sutcliffe Efficiency (NSE):

$$NSE_i = 1 - \frac{\sum_{t=1}^{T_i} \left(Q_o^{ti} - Q_m^{ti}\right)^2}{\sum_{t=1}^{T_i} \left(Q_o^{ti} - \bar{Q}_o^i\right)^2}$$

Pearson's correlation coefficient (r):

$$r = \frac{\Sigma_{t=1}^{T_i}(Q_o^{ti} - \overline{Q_o^t})(Q_m^{ti} - \overline{Q_m^t})}{\sqrt{\Sigma_{t=1}^{T_i}(Q_o^{ti} - \overline{Q_o^t})^2}\sqrt{\Sigma_{t=1}^{T_i}(Q_m^{ti} - \overline{Q_m^t})^2}}$$

where Q_o^{ti} and Q_m^{ti} are the observed and modelled value for site i and time step t, for a total of T_i available observations, and $\overline{Q_o^i}$ is the mean observed and $\overline{Q_m^i}$ the modelled data for site *i*.

The bias and monthly NSE statistics in particular are seen as good metrics for judging the model's performance for specific runoff. Pearson's correlation coefficient is a good indicator for variables where the bias (and absolute value) of the variable is not as important as matching the variability (e.g. soil moisture and actual ET).

The objective function can also be assessed through the benchmarking package by calling 'grand_f'.

Summary percentiles can be printed out by specifying a statistic to the 'stat_percentiles' function.

By default the timeframe used for statistics is monthly, but can be specified by "freq=d" for daily, "freq=m" for monthly, "freq=y" for yearly. 'Stat' functions refer to statistical comparisons between simulation data and observational data and 'data' functions refer to statistical comparisons within the simulation data.

Objective function (grand_f)

```
q.benchmark.stat_percentiles('grand_f', freq='monthly')
```

	grand_f
AWRAMSI_v4_0_AWRAL	0.079414

Nash-Sutcliffe Efficiency (nse)

```
q.benchmark.stat_percentiles('nse', freq='m')
```

Relative bias

```
q.benchmark.stat_percentiles('bias_relative')
```

Pearson's correlation coefficient (pearsons_r)

```
q.benchmark.stat_percentiles('pearsons_r',freq='d')
```

Compare raw observations and modelled data

```
q.benchmark.stat_percentiles('mean',freq='d')
```

```
q.benchmark.data_percentiles(freq='d')
```

```
q.benchmark.data_percentiles(freq='m')
```

	0%	5%	25%	50%	75%	95%	100%
QObs	0.223122	1.006759	24.369122	118.908425	167.596434	255.380894	277.203081
AWRAMSI_v4_0_AWRAL	0.839098	7.674550	54.521343	92.887510	110.854524	165.543277	179.176569

```
q.benchmark.data_percentiles(freq='y')
```

	0%	5%	25%	50%	75%	95%	100%
QObs	2.45434	11.958391	264.210174	1394.824112	1967.992576	2840.950169	3169.355227
AWRAMSI_v4_0_AWRAL	10.05198	85.375847	610.631375	1069.948122	1205.007902	1863.253305	2000.805022

```
q.benchmark.stat(statistic='nse')
```

	AWRAMSI_v4_0_AWRAL
105001	0.628121
107002	0.874442
113004	0.834034
111101	0.795395
109001	0.549784
116008	0.658929
112102	0.914612
4508	0.271805
108003	0.677263
all	0.781663

```
mean_flow = q.benchmark.stat() # default stat is mean monthly value, which also adds the obs to the table
mean_flow
```

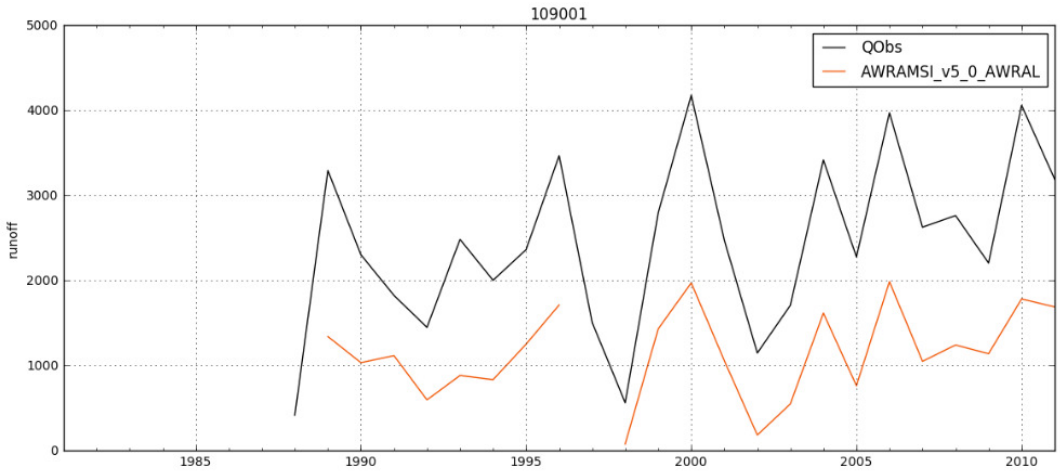
	AWRAMSI_v4_0_AWRAL	QObs
109001	110.854524	238.360273
105001	17.927729	11.517143
112102	145.093339	160.542532
111101	179.176569	285.815689
4508	0.839098	0.284997
116008	54.521343	102.399044
108003	92.887510	170.918889
107002	55.337463	63.690433
113004	94.130333	138.964279
all	92.298593	144.322406

8.4. View the statistic plots [time-series, regression, cumulative exceedance]

Display the modelled and observed runoff for benchmarked locations as a time-series. The frequency can be specified by "freq=d" for daily, "freq=m" for monthly, "freq=y" for yearly

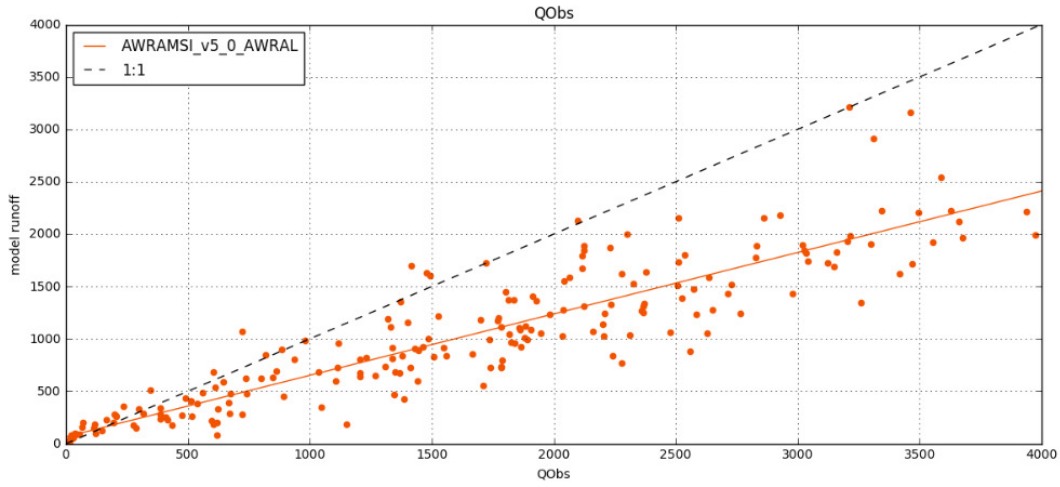
The titles, labels, scales etc can all be customised using standard matplotlib keyword arguments similar to the visualisation tools.

```
p = q.benchmark.plot_timeseries('109001',ylim=[0,5000], freq = 'y')
```



Create a scatter plot of the simulated daily, monthly or annually aggregated data against the observed values. If more than one model is present it will plot in a different colour on the same graph. A linear regression curve is also fit to the data.

```
p = q.benchmark.plot_regression(title="QObs", freq='y', xlim=[0,4000],ylim=[0,4000])
```




```
p = q.benchmark.plot_regression(title="QObs", freq='y', yscale='log', xscale='log', ylim=[10,10000], xlim=[10,10000])
```

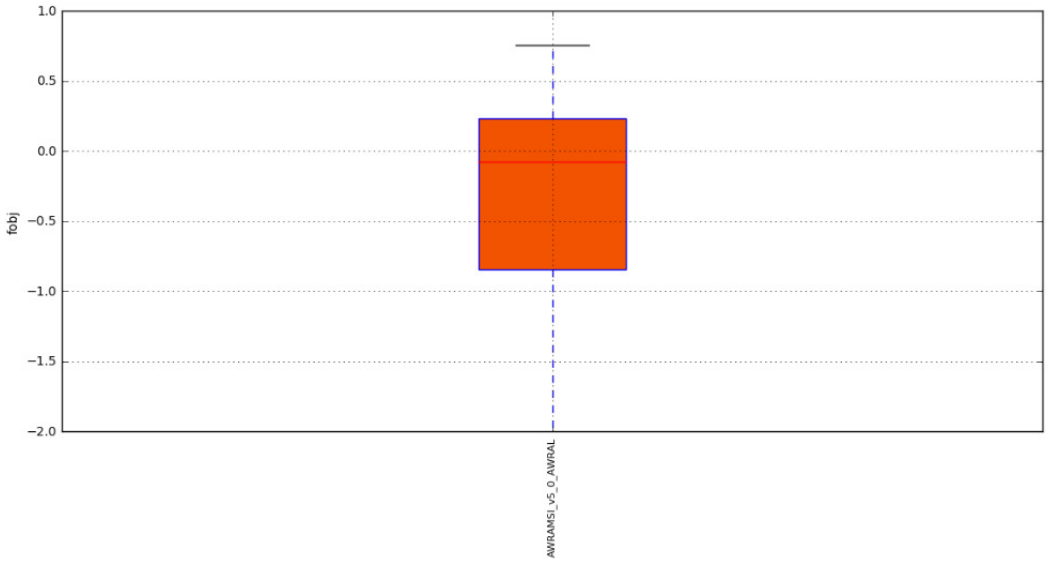
```
p = q.benchmark.plot_regression(title="QObs", freq='m', xlim=[0,2000],ylim=[0,2000])
```

```
p = q.benchmark.plot_regression(title="QObs", freq='d', xlim=[0,600],ylim=[0,600])
```

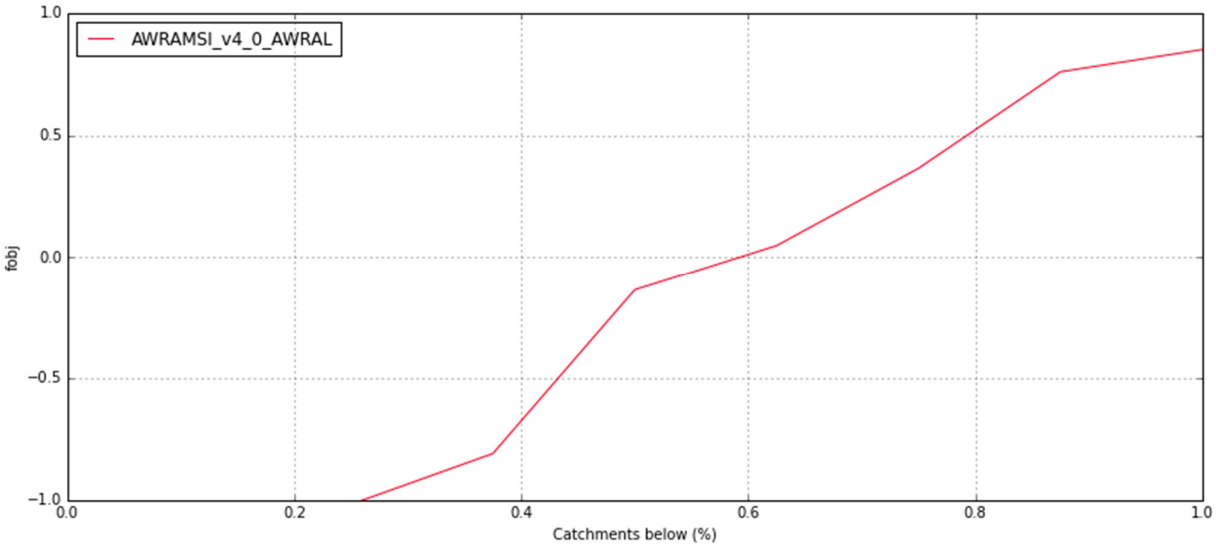
8.5. Statistics plotting

Typically, the functionality provided in the statistical plotting consists of box-whisker plots of the selected statistic for the population of catchment being benchmarked. The statistic type can be selected from "fobj", "nse", "rmse", "bias_relative", "pearsons_r" (default), "mean" and frequency from 'd', 'm', 'y' to create plots of simulation data . As an example, we provide plots for the "fobj" statistic only:

```
# fobj is the combined objective function used by Viney  et al. in calibration
p = q.benchmark.plot_box('fobj', freq='d', ylim=[-2,1])
```




```
p = q.benchmark.plot_cdf('fobj',freq='d', ylim=[-1,1])
```


AWRAMSI_v4_0_AWRAL
fobj
Catchments below (%)
1.0
0.5
0.0
−0.5
−1.0
0.0
0.2
0.4
0.6
0.8
1.0

9. References

Bartalis, Z., Wagner, W., Naeimi, V., Hasenauer, S., Scipal, K., Bonekamp, H., Figa, J. and Anderson, C., 2007. Initial soil moisture retrievals from the METOP-A Advanced Scatterometer (ASCAT). Geophysical Research Letters, 34(20): L20401.

Beringer, J., Hutley, L. B., McHugh, I., Arndt, S. K., Campbell, D., Cleugh, H. A., Cleverly, J., Resco de Dios, V., Eamus, D., Evans, B., Ewenz, C., Grace, P., Griebel, A., Haverd, V., Hinko-Najera, N., Huete, A., Isaac, P., Kanniah, K., Leuning, R., Liddell, M. J., Macfarlane, C., Meyer, W., Moore, C., Pendall, E., Phillips, A., Phillips, R. L., Prober, S. M., Restrepo-Coupe, N., Rutledge, S., Schroder, I., Silberstein, R., Southall, P., Yee, M. S., Tapper, N. J., van Gorsel, E., Vote, C., Walker, J., and Wardlaw, T, 2016. An introduction to the Australian and New Zealand flux tower network – OzFlux, Biogeosciences, 13, 5895-5916.

Beringer, J., McHugh, I., Hutley, L.B., Isaac, P. and Kljun, N., 2016. Dynamic INtegrated Gap-filling and partitioning for OzFlux (DINGO). Biogeosciences Discuss., 2016: 1-36.

Duan, Q., Sorooshian, S. and Gupta, V. 1992. Effective and efficient global optimization for conceptual rainfall-runoff models. Water Resources Research, 28, 1015-1031.

Frost, A, Ramchurn, A. and Hafeez, M, 2016, Evaluation of the Bureau's Operational AWRA-L Model. Bureau of Meteorology Technical Report.

Frost, A, Ramchurn, A. and Smith, A., 2016. The Bureau's Operational AWRA-L Model: Technical description of AWRA-L version 5. Bureau of Meteorology Technical Report.

Guerschman, J.P., Van Dijk, A.I.J.M., Mattersdorf, G., Beringer, J., Hutley, L.B., Leuning, R., Pipunic, R.C. and Sherman, B.S., 2009. Scaling of potential evapotranspiration with MODIS data reproduces flux observations and catchment water balance observations across Australia. Journal of Hydrology, 369(1-2): 107-119.

Hafeez, M., Frost, A., Vaze, J., Dutta, D., Smith, A. and Elmahdi, A., 2015. A New Integrated Continental Hydrological Simulation System: An overview of the Australian Water Resource Assessment Modelling System (AWRAMS). Water: Journal of the Australian Water Association, 42(3): 75-82.

Owe, M., de Jeu, R. and Holmes, T., 2008. Multisensor historical climatology of satellite-derived global land surface moisture. Journal of Geophysical Research: Earth Surface, 113(F1): F01002.

Rüdiger, C., Hancock, G., Hemakumara, H.M., Jacobs, B., Kalma, J.D., Martinez, C., Thyer, M., Walker, J.P., Wells, T. and Willgoose, G.R., 2007. Goulburn River experimental catchment data set. Water Resources Research, 43(10): W10403.

Smith, A.B., Walker, J.P., Western, A.W., Young, R.I., Ellett, K.M., Pipunic, R.C., Grayson, R.B., Siriwardena, L., Chiew, F.H.S. and Richter, H., 2012. The Murrumbidgee soil moisture monitoring network data set. Water Resources Research, 48(7): W07701.

Viney, N., Vaze, J., Crosbie, R., Wang, B., Dawes, W. and Frost, A., 2015. AWRA-L v5.0: technical description of model algorithms and inputs, CSIRO, Australia.

Zhang, Y.Q., Viney, N., Frost, A., Oke, A., Brooks, M., Chen, Y. and Campbell, N., 2013. Collation of streamflow and catchment attribute dataset for 780 unregulated Australian catchments, CSIRO: Water for a Healthy Country National Research Flagship.

Appendix A. AWRA-L model overview

AWRA-L is a one-dimensional, 0.05° grid based water balance model over the Australian continent that has a semi-distributed representation of the soil, groundwater and surface water stores. For soil moisture accounting and runoff generation purposes, each grid cell in AWRA-L is divided into three soil layers (top: 0-10cm, shallow: 10cm-100cm, deep: 100cm-600cm) and two hydrological response units (shallow rooted versus deep-rooted vegetation) (with model description provided in Frost, Ramchurn and Smith, 2016). Within each grid cell the water balance is calculated independently for the 2 HRUs (deep and shallow rooted vegetation) with outputs from the model representing the grid cell average or total (depending on variable). Climatological forcing for AWRA-L model is taken from the operational Australian Water Availability Project (AWAP) climate data set which is available for the same 0.05° grid as AWRA-L (approximately 5 km resolution) and runs with a daily time step from 1911 to 'today' (a solar radiation climatology is used prior to 1990). The model also requires spatial inputs of soil, geological and terrain properties as well as the distribution of the HRUs in each grid cell.

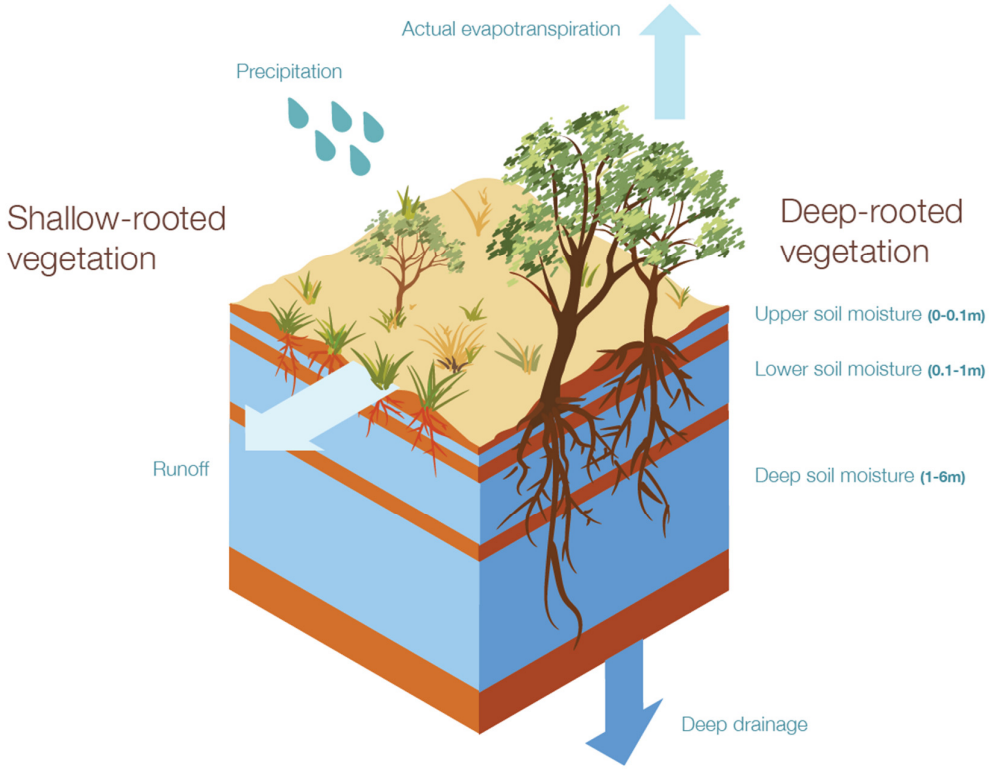


Figure A1. Conceptual AWRA-L grid cell with key water stores and fluxes shown

The model provides daily outputs of runoff from the landscape (surface and subsurface), actual and potential evapotranspiration (ET), soil moisture in each layer, and deep drainage at high-resolution, seamlessly across Australia from the past to the present (100 + years). These outputs can be aggregated spatially or temporally to provide daily, monthly and annual, regional or continental estimates of the landscape water balance.

The model is under constant development, and the current operational version at the Bureau of Meteorology is v5. Frost, Ramchurn and Hafeez (2016) evaluated AWRA-L v5 outputs against a range of data sources including streamflow, probe based estimates of soil moisture, flux tower estimates of evapotranspiration and a national groundwater recharge data set, satellite observations of soil moisture and evapotranspiration. AWRA-L v5 performed relatively well for runoff (compared with streamflow) and rootzone soil moisture compared to peer models. Benchmark statistics are provided in that report over which future versions of AWRA-L and other models can be compared. A key component of the community modelling system is the benchmarking datasets allowing easy comparison to current AWRA-L performance.

Appendix B. AWRA-L parameters

Parameter	Description
alb	Surface albedo (dimensionless)
alb_dry	Dry soil albedo (dimensionless)
alb_soil	Albedo of soil surface (dimensionless)
alb_veg	Albedo of vegetated surfaces (dimensionless)
alb_wet	Wet soil albedo (dimensionless)
cGsmax	Coefficient relating vegetation photosynthetic capacity to maximum stomatal conductance (m s^{-1})
D0	Vertical drainage from the bottom of the surface soil layer (mm)
Dd	Vertical drainage from the bottom of the deep soil layer (mm)
delta	Slope of the saturation vapour pressure curve (Pa K^{-1})
Ds	Vertical drainage from the bottom of the shallow soil layer (mm)
E0	Potential evaporation (mm d^{-1})
Eg	Evaporation flux from the groundwater store (mm d^{-1})
Ei	Evaporation flux from canopy interception (mm d^{-1})
ER_frac_ref	Ratio of the mean evaporation rate and the mean rainfall intensity during storms (dimensionless)
Es	Evaporation flux from the surface soil store (mm d^{-1})
Et	Actual total transpiration flux (mm d^{-1})
Etmax	Potential transpiration rate (mm d^{-1})
Etot	Total evapotranspiration (mm d^{-1})
f_tree	Fraction of tree cover within each grid cell (dimensionless)
fEgt	Fraction of the grid cell that is accessible for transpiration from groundwater (dimensionless)
fsat	Fraction of the grid cell that is saturated at the surface (dimensionless)
FsoilEmax	Soil evaporation scaling factor corresponding to unlimited soil water supply (dimensionless)
fveg	Fractional canopy cover (dimensionless)
fvegref_G	Reference Soil Cover Fraction That Determines The Rate of Decline in Soil Heat Flux With Increasing Canopy Cover
fveq	Equilibrium canopy cover (dimensionless)
ga	Aerodynamic conductance (m s^{-1})
gamma	Psychrometric constant (Pa K^{-1})
Gfrac_max	Fraction of Daytime Net Radiation Lost To Soil Heat Storage for an Unvegetated Surface
gs	Canopy conductance (m s^{-1})
hruDR	Hydrological Response Unit: Deep Rooted
hruSR	Hydrological Response Unit: Shallow Rooted
hveg	Height of Vegetation Canopy (m)
I	Infiltration (mm)
IAL	Interaquifer leakage (mm)

IF0	Interflow draining laterally from the surface soil layer (mm)
IFs	Interflow draining laterally from the shallow soil layer (mm)
Init_	Initial state of variable
K_gw	Groundwater drainage coefficient (d^{-1})
K_gw_grid	Groundwater drainage coefficient obtained from continental mapping (d^{-1})
K_gw_scale	Scaling factor for groundwater drainage coefficient (dimensionless)
K_rout	Rate coefficient controlling discharge to stream (dimensionless)
K_rout_scale	Scale coefficient for calculating K_r ($d\ mm^{-1}$)
K0sat	Saturated hydraulic conductivity of surface soil layer (mm d^{-1})
K0sat_grid	Saturated hydraulic conductivity of surface soil layer from pedtransfer (mm d^{-1})
K0sat_scale	Scaling factor for hydraulic conductivity of surface soil layer (dimensionless)
Kdsat	Saturated hydraulic conductivity of deep soil layer (mm d^{-1})
Kdsat_grid	Saturated hydraulic conductivity of deep soil layer from pedotransfer (mm d^{-1})
Kdsat_scale	Scaling factor for hydraulic conductivity of deep soil layer (dimensionless)
Kr_coeff	Scaling factor for ratio of saturated hydraulic conductivity (dimensionless)
Krout_int	Intercept coefficient for calculating K_r (dimensionless)
Kssat	Saturated hydraulic conductivity of shallow soil layer (mm d^{-1})
Kssat_grid	Saturated hydraulic conductivity of shallow soil layer from pedotransfer (mm d^{-1})
Kssat_scale	Scaling factor for hydraulic conductivity of shallow soil layer (dimensionless)
LAI	Leaf area index (LAI) (dimensionless)
LAImax	Maximum achievable LAI value (dimensionless)
LAIref	Reference LAI value corresponding to $f_v = 0.63$ (dimensionless)
lambda	Latent heat of vaporisation (MJ kg^{-1})
latitude	Latitude (radians), and is negative in the southern hemisphere
Mleaf	Leaf biomass (kg m^{-2})
Mleafnet	Change in leaf biomass at each time step (kg $m^{-2}\ d^{-1}$)
ne	Effective porosity (dimensionless)
ne_scale	Scaling factor for effective porosity (dimensionless)
pair	Air pressure (Pa)
pe	Actual vapour pressure (Pa)
pes	Saturation vapour pressure (Pa)
Pg	Precipitation (mm)
PI	Sunset hour angle (radians)
Pn	Net precipitation – precipitation minus interception (mm)
Pref_gridscale	Scaling factor for reference precipitation (dimensionless)
PrefR	Reference value for precipitation (mm)
Pwet	Reference threshold precipitation amount (mm)
Q0	Function of the day of the year (radians)
Qg	Groundwater discharge to the surface water (mm)
QIF	Interflow (mm)
QR	Surface runoff (mm)
Qtot	Total discharge to stream (mm)

RadClearSky	Expected downwelling shortwave radiation on a cloudless day (MJ m^{-2} d^{-1})
RD	Rooting depth (m)
Rgeff	Daily downwelling shortwave (solar) radiation (MJ m^{-2} d^{-1})
Rh_0s	Partitioning factor for vertical and lateral drainage from the surface soil layer (dimensionless)
Rh_sd	Partitioning factor for vertical and lateral drainage from the shallow soil layer (dimensionless)
Rhof	Infiltration-excess runoff component (mm)
RLin	Daily downwelling longwave radiation (MJ m^{-2} d^{-1})
RLout	Daily upwelling longwave radiation (MJ m^{-2} d^{-1})
Rneff	Daily net radiation (MJ m^{-2} d^{-1})
Rsof	Saturation-excess runoff component (mm)
S	Storativity of the confined aquifer (mm)
S_sls	Specific canopy rainfall storage per unit leaf area (mm)
S0	Water storage in the surface soil layer (mm)
S0fracAWC_grid	Available water holding capacity in the surface soil (dimensionless)
S0max	Maximum storage of the surface soil layer (mm)
S0max_scale	Scaling parameter for maximum storage of the surface soil layer (dimensionless)
Sd	Water content of the deep soil store (mm)
Sdmax	Maximum storage of the deep soil layer (mm)
Sdmax_scale	Scale for Maximum water storage deep layer (Deep) (dimensionless)
Sg	Groundwater storage in the unconfined aquifer (mm)
SLA	Specific leaf area (m^2 kg^{-1})
slope	Slope of the land surface (percent)
slope_coeff	Scaling factor for slope (dimensionless)
Sr	Volume of water in the surface water store (mm)
Ss	Water content of the shallow soil store (mm)
SsfracAWC_grid	Available water holding capacity in the shallow soil (dimensionless)
Ssmax	Maximum storage of the shallow soil layer (mm)
Ssmax_scale	Scale for Maximum water storage shallow layer (Shallow) (dimensionless)
StefBolz	Stefan-Boltzmann constant (MJ m^{-2} d^{-1} K^{-4})
Ta	Daily mean temperature (°C)
Tgrow	Characteristic time scale for vegetation growth towards equilibrium (d)
Tmax	Maximum air temperature (°C)
Tmin	Minimum air temperature (°C)
Tsenc	Characteristic time scale for vegetation senescence towards equilibrium (d)
U0	Maximum root water uptake (mm d^{-1})
u2	Wind speed at a height of 2 m (m s^{-1})
Ud	Root water uptake (transpiration) from the deep soil store (mm d^{-1})
Ud0	Maximum possible root water uptake from the deep soil store (mm d^{-1})

Udmax	Maximum root water uptake from the deep soil store at prevailing moisture content (mm d^{-1})
Us	Root water uptake (transpiration) from the shallow soil store (mm d^{-1})
Us0	Maximum possible root water uptake from the shallow soil store (mm d^{-1})
Usmax	Maximum root water uptake from the shallow soil store at prevailing moisture content (mm d^{-1})
Vc	Greenness index per unit canopy cover
w0	Relative soil moisture content of the top soil layer (dimensionless)
w0limE	Limiting the value of w_0 at which evaporation is reduced (dimensionless)
w0ref_alb	Reference value of w_0 that determines the rate of albedo decrease with wetness (dimensionless)
wd	Relative water content of the deep soil store (dimensionless)
wdlimU	Water-limiting relative water content of the deep soil store (dimensionless)
ws	Relative water content of the shallow soil store (dimensionless)
wslimU	Water-limiting relative water content of the shallow soil store (dimensionless)
Y	Root water uptake (transpiration) from the groundwater store via capillary rise (mm d^{-1})

Appendix C. Python version of AWRA-L

The model code has been translated from C into Python towards understanding the code more easily as compilation is not required. AWRA is run in two ways here:

- Component-by-component
- All components put together

The below code is commented with reference back to the model documentation and the model is run for a year for a chosen point to understand the dynamics.

C.1. The AWRA-L Workflow

AWRA-L broadly follows the following workflow:

- Pre-calculate and convert constants for use in the model
- Initialise models states
- Loop over Time steps
    A. Calculate saturated fraction of cell/catchment for a given groundwater storage (Sg) [a single store - no HRUs
    B. Calculate potential evaporation (E0) according to Energy balance [for each HRU]
    C. Calculate actual ET based on E0, water storage and vegatation state [for each HRU]
    D. Calculate Water balance and flows [for each HRU]
    E. Calculate Vegetation growth and senescence [for each HRU]
    F. Get cell values of variables weighting shallow and deep rooted components depending on fraction of each

These components are coded in Python below (rather than C) to facilitate understanding and testing the AWRA-L model.

C.1.1.AWRA-L model processes and algorithms overview

Run a model simulation to get input data and AWRA-L parameters i

AWRA-L calculations deconstructed - run for time period and parameters supplied

the broad steps listed above are undertaken for demonstration without time step interaction between the energy, water and vegetation components of the model. This is to show th equations used in calculation.

Note there is no dynamic interaction of variables in this case. e.g. fraction saturated, relative top layer soil content kept constant for E0 estimation

The subsequent model run with the components put into the same cell (including time step looping) incorporates this interaction.

Appendix D. Contributing to AWRA-CMS

One of the main motivations of releasing AWRA as a community model is to gain the benefits of the scientific community in contributing improvements to the system. We hope that other research scientists and agencies will be interested in:

- altering model process descriptions (algorithms) towards better performance,
- increasing functionality of the modelling system
- testing alternative input datasets to the system.
- Improving the calibration and validation procedure
- And more

To make changes to the AWRA-L community model source code the User needs to create their own 'fork' (altered copy of the community model code) of the code on the GitHub repository.

After ensuring the robustness of the changes locally, developers are able to contribute to the community modelling system by proposing their alterations to the Bureau according to the following steps:

1. The User proposes changes to the Bureau, for example by submitting an email outlining the proposed changes to the Bureau
2. User submits their fork (new code) and any documentation and testing that has been conducted
3. The Bureau assesses conceptually the proposed change the submission of fork/code provided in step 2.
4. The Bureau assesses that code against various performance criteria including performance against benchmark data, system performance, code complexity and maintainability. If successful according to that testing the Bureau releases the new version of the CMS. Note that the bureau is also open to improvements to the benchmarking process and data sets if this can be scientifically justified.

The above list is an outline of the process, once users contact the Bureau we will be able to give instructions and guidance in greater depth.

Appendix E. AWRAMS Licence

AWRAMS LICENCE

(Variation of BSD 3 licence)

By accessing or using the AWRAMS software, code, data (input or validation data) or documentation ("AWRAMS Material"), You agree to be bound by these licence terms including any terms or notices incorporated by reference (as updated from time to time) on and from the date You first access or use of the AWRAMS Material. You acknowledge that Your agreement to these licence terms constitutes a legally binding agreement between You and the Commonwealth of Australia as represented by the Bureau of Meteorology ("copyright holder"). If You do not agree to be bound by any of these licence terms, You are not authorised to access and use the AWRAMS Material.

The reference to "You" ("or Your") is a reference to an individual or legal entity exercising permissions granted under these licence terms.

Redistribution and use of AWRAMS Material in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

* Redistributions of AWRAMS Material in source code must retain this AWRAMS licence in its entirety.

* Redistributions of AWRAMS Material in binary form must reproduce this AWRAMS licence in its entirety.

* Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from AWRAMS Material without specific prior written permission.

* Redistribution and use of any part of the AWRAMS Material for commercial purposes must not occur without prior written permission from the Bureau of Meteorology (email: awracms@bom.gov.au). For the avoidance of doubt, 'commercial purposes' includes without limitation developing and supplying a product or service to another party for profit or other commercial gain.

* You will make any and all of Your modifications to the AWRAMS Material (including modifications to code, data or documentation) available to the Bureau of Meteorology for possible use by it in any way, including, but not limited to, incorporating modifications into the existing AWRAMS Material if they meet the relevant criteria and making these modifications available to third parties.

* You will complete and sign and return to the Bureau of Meteorology (by email to awracms@bom.gov.au) the Contributor Licence Agreement ("CLA") which accompanies these licence terms in relation to any and all of Your modifications to the AWRAMS Material referred to above.

TO THE FULLEST EXTENT PERMITTED BY LAW, THE AWRAMS MATERIAL IS PROVIDED BY THE COPYRIGHT HOLDER AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED OR EXCLUDED. TO THE FULLEST EXTENT PERMITTED BY LAW, NEITHER THE COPYRIGHT HOLDER NOR ANY CONTRIBUTOR WILL BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWSOEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THE AWRAMS MATERIAL, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH CLAIM, LIABILITY, LOSS OR DAMAGE.

ACCESS TO INPUT/VALIDATION DATA

You will need to complete and sign this AWRAMS licence and the CLA and return signed copies to the Bureau of Meteorology (by email to awracms@bom.gov.au) to enable the Bureau of Meteorology to assess whether to issue You with a link to the input data and validation data for AWRAMS together with a user name and password for access to the data. You agree that the Bureau of Meteorology may allow or deny You access to the input data or validation data for AWRAMS in its absolute discretion and without giving reasons following receipt of the completed and signed AWRAMS licence and CLA. If the Bureau of Meteorology grants you access to the input and validation data you agree that Your access to and use of this data will be governed by the terms of this AWRAMS licence.

Irrespective of whether or not these licence terms and the CLA are signed and submitted to the Bureau as required above, You agree that the licence obligations remain legally binding and enforceable to the extent permitted by law as a consequence of You accessing or using the AWRAMS Material and on and from the date on which You first access or use the AWRAMS

USER DETAILS (please complete)

If signing as an individual and not on behalf of an organisation

Name (individual signing in their own right):

Email address:

Address:

Phone number:

If signing on behalf of an organisation

Name (individual representative signing on behalf of organisation):

Name of Organisation (legal entity):

ABN

Email address:

Address:

Phone number:

Please state the nature and purpose of proposed use of AWRAMS Material:

……

……

Signed:

……

Date:……

Appendix F. AWRAMS Contributor Licence

AWRAMS CONTRIBUTOR LICENCE AGREEMENT

(FOR INDIVIDUALS AND LEGAL ENTITIES)

This contributor agreement ("Agreement") documents the rights granted by contributors to the Commonwealth of Australia as represented by the Bureau of Meteorology ("Us"). To make this document effective, please sign it and send it to Us (by email to awracms@bom.gov.au).

1. Definitions

"You" means as applicable the individual ('natural person') or organisation ('legal entity') who Submits a Contribution to Us.

"Contribution" means any work of authorship or invention that is Submitted by You to Us in which You own or assert ownership of any intellectual property rights (**IP**), including without limitation any copyright, patent rights or related rights. If You do not own IP in the entire work of authorship or invention, please clarify the basis on which the work of authorship or invention was created, contributed or used by You (where indicated in clause 3 below).

"Material" means the work of authorship or invention which is made available by Us to third parties. When this Agreement covers more than one software project, the Material means the work of authorship or invention to which the Contribution was Submitted. After You Submit the Contribution, it may be included in the Material.

"Submit" means any form of electronic, verbal, or written communication sent to Us or our representatives, including but not limited to electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, Us for the purpose of discussing and improving the Material.

"Effective Date" means the date You execute this Agreement or the date You first Submit a Contribution to Us, whichever is earlier.

2. Grant of Rights

2.1 Copyright License

(a) You retain ownership of the copyright in Your Contribution and have the same rights to use or license the Contribution which You would have had without entering into the Agreement.

(b) To the maximum extent permitted by the relevant law, You grant to Us a perpetual, worldwide, non-exclusive, transferable, royalty-free, irrevocable copyright licence covering the Contribution, with the right to sublicense such rights through multiple tiers of sublicensees, to reproduce, modify, display, perform and distribute the Contribution as part of the Material.

2.2 Patent License

For patent claims including, without limitation, method, process, and apparatus claims which You own, control or have the right to grant, now or in the future, You grant to Us a perpetual, worldwide, non-exclusive, transferable, royalty-free, irrevocable patent licence, with the right to sublicense these rights to multiple tiers of sublicensees, to make, have made, use, sell, offer for sale, import and otherwise transfer the Contribution and the Contribution in combination with the Material (and portions of such combination). This licence is granted only to the extent that the exercise of the licensed rights infringes such patent claims.

2.3 Outbound License

Based on the grant of rights in clauses 2.1 and 2.2, if We include Your Contribution in Material, We may license the Contribution under any license, including copyleft, permissive, commercial, or proprietary licenses.

2.4 Moral Rights

If moral rights apply to the Contribution, to the maximum extent permitted by law, You waive and agree not to assert such moral rights against Us or our successors in interest, or any of our licensees, either directly or indirectly. In addition, You consent to Us or our successors in interest or licensees using the Contribution consistently with the terms of this Agreement for moral rights purposes. If You are an organisation or not the moral rights holder, You agree to obtain written consents from the authors of the relevant parts of the Contribution to allow Us or our successors in interest or licensees to use the Contribution consistently with the terms of this Agreement for moral rights purposes.

2.5 Our Rights

You acknowledge that We are not obliged to use Your Contribution as part of the Material and may decide to include any Contribution We consider appropriate.

2.6 Reservation of Rights

Any rights not expressly licensed under this clause are expressly reserved by You.

3. Agreement

You warrant and represent that:
(a) You have the legal authority to enter into this Agreement.
(b) You own or otherwise have rights in respect of the copyright and patent claims covering the Contribution which are required to grant the rights under clause 2.
(c) the grant of rights under clause 2 does not violate any grant of rights which You have made to third parties, including Your employer. If You are an employee, You have had Your employer approve this Agreement or sign another copy of this document. If You are less than eighteen years old, please have Your parents or guardian sign the Agreement.
(d) the Contribution does not contain any malicious code, virus or any intentionally deleterious modification.
(d) If You do not own the IP in the entire work of authorship or invention Submitted, the legal basis on which You have created or used the work of authorship or invention and on which You have legal authority to enter into this Agreement and to grant the licence rights under this Agreement is as follows:

[insert details]

4. Disclaimer

EXCEPT FOR THE EXPRESS WARRANTIES IN CLAUSE 3, THE CONTRIBUTION IS PROVIDED "AS IS". MORE PARTICULARLY, ALL EXPRESS OR IMPLIED WARRANTIES INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT ARE EXPRESSLY DISCLAIMED BY YOU TO US. TO THE EXTENT THAT ANY SUCH WARRANTIES CANNOT BE DISCLAIMED, SUCH WARRANTY IS LIMITED IN DURATION TO THE MINIMUM PERIOD PERMITTED BY LAW.

5. Miscellaneous

5.1 This Agreement will be governed by and construed in accordance with the laws of the Australian Capital Territory excluding its conflicts of law provisions. Under certain circumstances, the governing law in this clause might be superseded by the United Nations Convention on Contracts for the International Sale of Goods ("UN Convention") and the parties intend to avoid the application of the UN Convention to this Agreement and, thus, exclude the application of the UN Convention in its entirety to this Agreement.

5.2 This Agreement sets out the entire agreement between You and Us for Your Contributions to Us and overrides all other agreements or understandings.
5.3 If You or We assign the rights or obligations received through this Agreement to a third party, as a condition of the assignment, that third party must agree in writing to abide by all the rights and obligations in the Agreement.
5.4 The failure of either party to require performance by the other party of any provision of this Agreement in one situation shall not affect the right of a party to require such performance at any time in the future. A waiver of performance under a provision in one situation shall not be considered a waiver of the performance of the provision in the future or a waiver of the provision in its entirety.
5.5 If any provision of this Agreement is found void and unenforceable, such provision will be replaced to the extent possible with a provision that comes closest to the meaning of the original provision and which is enforceable. The terms and conditions set forth in this Agreement shall apply notwithstanding any failure of essential purpose of this Agreement or any limited remedy to the maximum extent possible under law.

You

If signing as an individual and not on behalf of an organisation

Name: _____
Address: _____ _____
Signature:
_____
Date:…

If signing on behalf of an organisation

Name (organisation): _____
Address: _____ _____
Signature (duly authorised representative):
_____
Date:…"

Us

_____
Name: _____
Title: _____
Address: _____
_____
Signature (duly authorised representative):
_____
Date: