

Instructions for ADJULES

Catherine Luke and Tim Jupp

August 28, 2012

This document provides instructions for the ADJULES parameter estimation system. By following these instructions you should be able to compile the code, optimise parameters, evaluate and plot results.

For licensing reasons, recalculation of the adjoint following code changes can only be done by us here at Exeter. However, we are able to distribute adjoint code after we have created it. Thus, you can compile and run the *current* version ADJULES but you will not be able to propagate any code changes you make through to ADJULES.

N.B. some instructions, usually identified in bold type, given here rely on the TAF licence only available to certain users. If you do not think you have this licence, please do not carry out these steps as they will not work!

We include in this distribution a precompiled version of the shared object file `adjules.so`. This was compiled on a 32-bit Windows Vista laptop. Thus, Windows users should be able to skip the compilation steps below for ADJULES. Linux/Mac users will have to create the shared object files afresh.

This distribution also includes code to create R-JULES, which allows for JULES to be run through an interface in R. This is described in another set of instructions within this directory.

1 Retrieving ADJULES

The ADJULES bundle is available online, including the files and data required to run the JULES benchmarking sites. To obtain the code:

1. Open a web browser and navigate to <http://empslocal.ex.ac.uk/people/staff/tej202/rjules/>. Download 'rjules.tar.gz' by right-clicking on the link and saving to your computer or homespace.

2. Unpack the download by executing `tar -xzvf rjules.tar.gz`.

You are now set up with all the source code, files and data required for running ADJULES.

1.1 For TAF users only

If you have access to the FastOpt JULES repository, you can use this to get the most up to date code. This will only give you test data, you will need the `jules_verify_data` directory available with the online distribution.

1. Create an empty directory (`mkdir ADJULES`).
2. Navigate to this directory and execute
`svn co svn+ssh://jules@fastopt.net/jules/2.2/trunk ./`

2 Compiling ADJULES

In order to run ADJULES you must compile the relevant code. The most comprehensive version of ADJULES is called `fhjules`, where FH stands for ‘forward hessian’. This refers to the second derivative of JULES which allows us to estimate uncertainty and the relative importance of parameters.

1. If you have the appropriate licence (see above) you can generate the derivative source code by executing `make adsrcs vtsrcs fhsrcs`. **If you do not have the appropriate licence proceed to the next step**, which relies on the previously generated adjoint source code included in this distribution.
2. Compile the shared object file `fhjules.so` by executing (something like) `make fhjules COMPILER=gfortran`. Note that you may wish to reset the `COMPILER` option as appropriate for your local machine. You should now be ready to run ADJULES.

3 Running ADJULES

ADJULES is run from within the statistical package R, which can be downloaded for free from the web.

1. Begin a session in R by executing R at the command prompt.

2. Choose a location for your R-JULES run by executing `source("setup.R")`.

By following the on-screen instructions you can select the site you wish to use and the observation data against which you would like to calibrate your parameters:

- Choose the site (location) you wish to use by typing the appropriate number indicated on screen at the command prompt. Bondville (site 1) is set up for a week's test run.
- Select the option to optimise parameters by typing `y` at the appropriate prompt.
- Select (by typing `y` or `n` as appropriate) the observables you wish to use to calibrate the model.
- If you wish, you can choose to optimise averaged time-series (for example you may wish to calibrate monthly mean fluxes). Make your choice at the appropriate prompt by typing `y` or `n`.
- If you chose to optimise averaged time-series you must decide how many time steps to average over. Enter the number of time steps at the appropriate prompt.
- You can choose to mask out nighttime fluxes if you wish. Make your choice at the appropriate prompt by typing `y` or `n`.

3. To initialise ADJULES execute `source("fhjules.R")`.

- ADJULES recognises the parameters to which the cost function is sensitive and removes all others from the optimisation. You can find out which parameters are allowed to vary by typing `znames`.

4. To run the optimisation execute `source("findopt.R")`. This uses R minimisation routines and derivative information from the adjoint to search for the set of parameters that minimises the cost function.

You have now run ADJULES and (if all has worked smoothly) generated a locally optimum parameter set.

4 Visualising your results

ADJULES includes a number of built-in functions to allow you to visualise your results. The following list describes the basic functionality of each one.

plotadts This function allows you to plot time series of R-JULES output for initial parameter values and optimised parameter values along with time series of observed data (where available). You can select observed variable (**its**), range of points to plot (**imin** and **imax**) and y-axis limits (**ylo** and **yhi**).

Example: `plotadts(its=1,imin=1,imax=1000,ylo=-200)`.

plotttserr This function allows you to plot a time series of errors for initial parameter values and optimised parameter values in ADJULES output with respect to observed data (where available). You can select observed variable (**its**) and range of points to plot (**imin** and **imax**).

Example: `plotadtserr(its=1,imin=1,imax=1000)`.

ploterr This function allows you to plot the change in errors in a time series after optimisation. You can select a time series (**its**).

Example: `ploterr(its=1)`.

plotslice This function allows you to plot the cost function for a slice through parameter space, varying one parameter. You can select a location in z-space (**zz**), a parameter to vary (**zw**), range over which to vary the parameter (**zlo** and **zhi**) and the number of points to evaluate at (**n**).

Example: `plotslice(zz=z, zw=1, zlo=0.1, zhi=1)`.

plotadcorr This function allow you to plot error correlations in time series of R-JULES output with respect to observations for initial parameter values and optimised parameter values. You can select time series to compare (**its1** and **its2**).

Example: `plotadcorr(its1=2,its2=3)`.

5 Generating and using second derivative information

The second derivative of the cost function can be used to obtain information on the curvature of the cost function, which can be used to generate parameter uncertainties and associated statistics.

The theory behind the statistics is based on using the second derivative to calculate the curvature of the cost function at the locally optimal parameter set generated in the optimisation. A truncated multivariate normal distribution is generated based on the curvature of the cost function for each parameter. This allows us to look at parameter uncertainties and other relevant information.

1. To generate the second derivative (forward hessian) and associated statistics, execute `source("hessutils.R")`. Please note that this does take a long time to run! You are now ready to investigate parameter uncertainties and associated statistics.
2. Three useful objects have been created to help understand the relative importance, magnitude of change and correlation of the optimised parameters.

orderchange This object lists the parameters in order of the magnitude of their deviation from their original values. To look at this object, execute **orderchange**.

ordersharp This object lists the parameters in order of the ‘sharpness’ of the second derivative at the local optimum. This tells us about our confidence in parameter values, or how important the parameter is in cost reduction: a very sharp second derivative shows that we are confident that the parameter value should lie close to its optimum value (or that it is very important for cost reduction), while a flat second derivative shows that the parameter value is relatively unimportant in comparison to other parameters for the observables chosen. To look at this object, execute **ordersharp**.

ordercorr This object lists pairs of parameters sorted by the correlation between them. To look at this object, execute **ordercorr**.

3. To visualise your second derivative results, you can use the following plotting functions:

plotmarg This function allows you to plot marginal parameter densities. You can select a parameter (**ivar**) and choose whether to add scatter points from Gibbs sampling (**lsample**).

Example: `plotmarg(ivar=1, lsample=FALSE)`.

plotmarg2 This function allows you to plot the correlation between parameters. You can select parameters (**ivar**) and choose whether

to overlay contours of the two-dimensional marginal distribution (`lcontour`).

Example: `plotmarg(ivar=c(1,2), lcontour=TRUE)`.

6 Contact details

The ADJULES system is a work in progress. Please do not hesitate to contact us if you would like to suggest additions or changes, or require help:

- Tim Jupp: `T.E.Jupp@exeter.ac.uk`.
- Catherine Luke: `C.M.Luke@exeter.ac.uk`.