User Guide: Applying Optimisation Algorithms to Climate Models with OptClimSO

Sophy Oliver
Department of Earth Sciences, University of Oxford
sophy.oliver@oriel.ox.ac.uk

June 3, 2019

Contents

1 Introduction **1**

2 Inheritance and Acknowledgements **1**

3 Source **1**

4 Installation **1**

5 Workflow **1**

6 OptClim Scripts **2**

6.1 Optscript 2

6.2 JSON File 3

6.3 runOptimise.py 4

7 Differences to S.Tett's OptClimV2 **5**

8 Adding a New Climate Model **6**

8.1 Local machine 6

8.1.1 Step 1 6

8.1.2 Step 2 6

8.1.3 Step 3 6

8.2 Computing Cluster 7

8.2.1 Step 1 7

8.2.2 Step 2: If using OptClim/OptClimV2 8

8.2.3 Step 2: If using OptClimSO 8

9 Optimisation Algorithms **8**

9.1 Incorporated Optimisers 8

9.2 Adding a new algorithm to OptClim 9

10 OptClimSO Plotting Scripts **9**

11 References **15**

1 Introduction

OptClim is a Python-based control system which allows any computational model to be optimised (or "tuned") by any optimisation algorithm. Neither the climate model or the optimisation algorithm need know much about the other, only their inputs and outputs, which the OptClim system reads, monitors and distributes. This way, the climate model, which are usually very computationally expensive and complex, can be treated as a "black box".

2 Inheritance and Acknowledgements

OptClim (and the next version OptClimV2) was created and written by Professor Simon Tett, University of Edinburgh, and used in the optimisation studies Tett et al. (2013) and Tett et al. (2017). In 2018 OptClimV2 was edited by Sophy Oliver, University of Oxford. The code package has now diverged from OptClimV2 significantly (see Section 7) and is now known as OptClimSO.

3 Source

https://github.com/SophyOliver/OptClim

4 Installation

Copy all of OptClim across. This currently requires Python 2 with all necessary packages (e.g. pandas, numpy, scipy). To compile the optimisation algorithm packages do the following. Within DFOLS and Py_BOBYQA subfolders type "pip install ." to install the algorithms. Within cmaes subfolder, type "make" into the command line to compile.

5 Workflow

OptClim allows a climate model to be run multiple times to iteratively improve the model parameters. It does this by using a provided optimisation algorithm to iteratively minimise the model error (or "misfit"). As shown in the workflow diagram in Figure 1, the continuous loop of running a model then an optimiser is controlled by a JSON file in OptClim. The climate model in this example requires the input file "parameters_input.txt" and is run by the script "runscript". This model then runs, internally using any observational files, initial conditions files or scripts it needs to, and calculates it's misfit error to be minimised. The optimiser then reads in the previous history of parameter values and associated misfits, and determines the next parameter set to try which is likely

to minimise the misfit. The model is then run again with this parameter set, and the loop continues until the optimiser determines that the misfit has been sufficiently minimised.

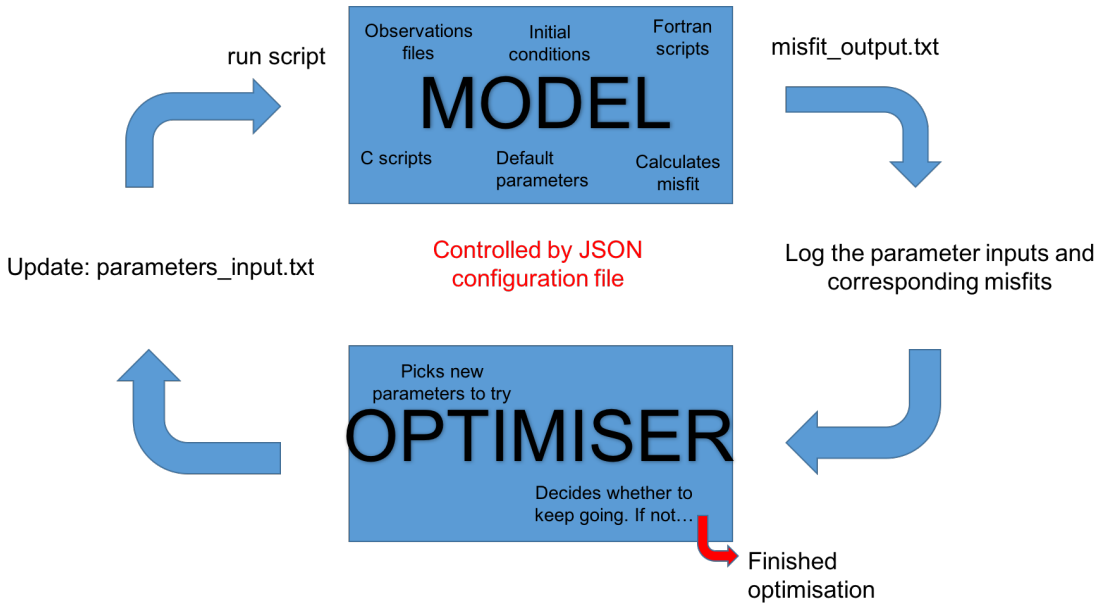

Figure 1: Schematic of OptClim's workflow.

6 OptClim Scripts

6.1 Optscript

This bash script to be submitted to the cluster job submission system contains a loop to repeatedly call runOptimise.py, which runs OptClim. This script is only required for OptClimSO, not for OptClimV2 (see Section 7).

Example:

```
#!/bin/bash
# set the number of nodes and processes per node
#SBATCH --nodes=6
# set the number of tasks (processes) per node.
#SBATCH --ntasks-per-node=16
# set max wallclock time
#SBATCH --time=120:00:00

module load python/anaconda2/5.0.1
source activate myenv

finished="False"
iRun=1 # change this if optimisation process crashed half way
   last time. At least don't have as 1 if you have previous runs
```

```
   saved, as will re
start the whole thing and erase previous runs.
iMax=80 # max number of iterations that can fit in the set wall
   clock time.

# Loop through optimisation iterations while we're below the set
   max iterations and the algorithm is not finished
while [ $iRun -le $iMax ] && [ "$finished" == "False" ]; do

        if [ $iRun -eq 1 ]; then
                # Need to restart
                python runOptimise.py --restart -dir twin6_dfo_r
                   -v ../Configurations/OxfordMOPS_Configs/
                   TWIN_Configs/OxfordMOPS_dfols_twin_6p.json &>
                   prog_t6d_r.txt

        else
                # Can carry on from previous iteration
                python runOptimise.py -dir twin6_dfo_r -v ../
                   Configurations/OxfordMOPS_Configs/TWIN_Configs
                   /OxfordMOPS_dfols_twin_6p.json &>prog_t6d_r.
                   txt

        fi

        # Update counter
        let iRun++
        # Get last line from progress text, which should be
           either True or False, determining whether the
           optimisation process is finished or not.
        finished=`tail -1 prog_t6d_r.txt`

        echo The iRun is $iRun
        echo "$finished"

done
```

6.2 JSON File

The JSON (JavaScript Object Notation) file is written in a way that is easy for machines to parse, as well as for us to read and write. Here the JSON file contains all the information provided by the user regarding their optimisation study, model parameters, location of reference files and scripts, optimiser to use, optimisation settings, etc. For examples see folder OptClim/Configurations/OxfordMOPS_Configs/TWIN_Configs.

6.3 runOptimise.py

runOptimise.py is a Python script which begins the workflow loop shown in Figure 1. Inputs it requires include an output directory to save each iteration's climate model results to, and the JSON file specifying the OptClim settings. Below is an example of how to call runOptimise, with "–restart" specifying it to delete any previous iterations and start the optimisation study from scratch, and "&>prog_t6d_r.txt" specifying all text output to the command line to be printed in a text file.

Example:

```
python runOptimise.py --restart -dir twin6_dfo_r -v ../JSON_Files
    /OxfordMOPS_dfols_twin_6p.json &>progress.txt
```

CODE DESCRIPTION: RUNOPTIMISE.PY

INPUT: Output directory ("rootDir"), JSON file location, and if it is to RESTART.

1. Import packages.
2. Define functions.
3. Define the function for model creation using ModelSimulation.py and the class specific to the climate model.
4. Use StudyConfig.py to load in the config information from the JSON file.
5. Create MODELRUN to store all information on models previously run.
6. **if** RESTART is True **do**
7. Delete all files and folders in the output directory.
8. **end if**
9. **for** every subfolder in rootDir **do**
10. Read in that model run's information and store in MODELRUN.
11. **end do**
12. Read algorithm name from config.
13. **if** algorithm name == "algorithmX" **do**
14. Retrieve user's settings specific to this optimisation algorithm.
15. Run the optimiser.
16. **if** optimiser states it's not yet at a solution **do**
17. Append MODELRUN['modelsToMake'] with a new model run(s).
18. Raise "LinAlgError": Go to line 22.
19. **end if**
20. **end if**
21. Print and save solution status.
22. **if** LinAlgError **do**
23. **for** every model in MODELRUN['modelsToMake'] **do**
24. Create the next subfolder in rootDir.
25. Create the model for this subfolder using ModelSimulation.py and the class specific to the climate model.
26. Submit the model run using Submit.py.
27. **end do**
28. **end if**
29. Store optimisation history and updated configuration file.

30. Write optimisation history to the "final" JSON file.

7 Differences to S.Tett's OptClimV2

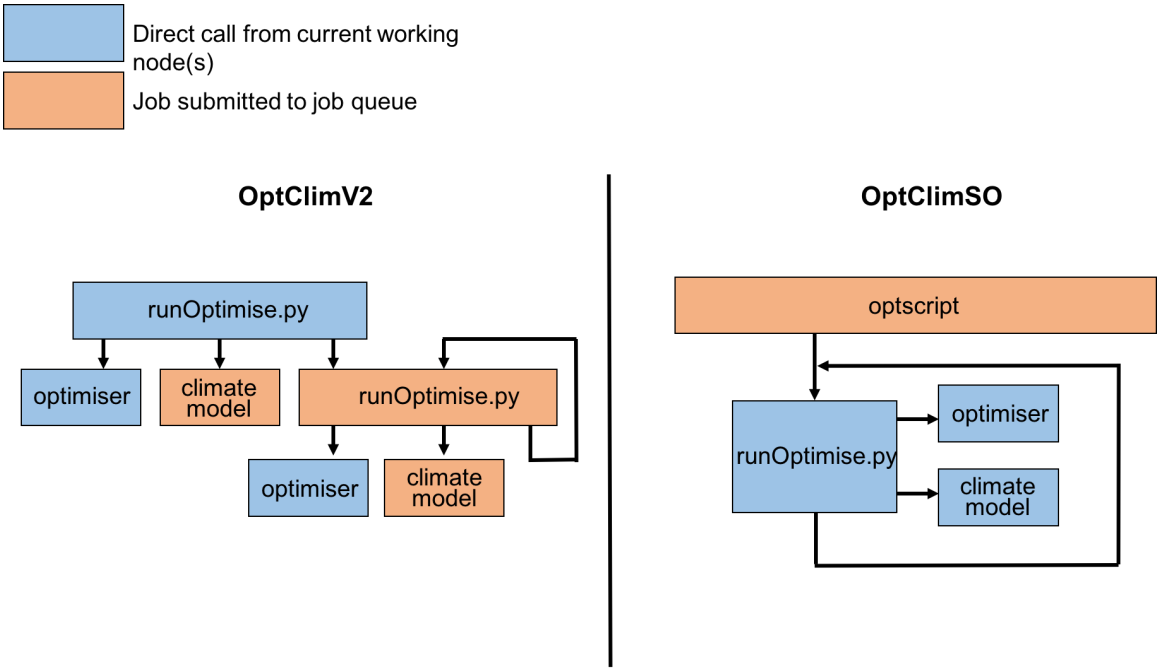

Figure 2: Schematic of OptClimV2 and OptClimSO workflow.

To reduce the total time spent in the job submission queue, the workflow of OptClimV2 was edited for OptClimSO, as shown in Figure 2.

In OptClimV2, the script runOptimise.py (when not run in test -t mode) will follow this workflow:

1. From the command line: python runOptimise.py ....
2. runOptimise directly runs the optimiser.
3. runOptimise submits to the job submission queue the climate model job(s).
4. runOptimise submits the runOptimise script again, with a dependency to wait for the job(s) above to finish.

The workflow was edited for OptClimSO:

1. Submit optscript to the job submission system.
2. Optscript calls runOptimise in a for loop.
3. runOptimise directly runs the optimiser.
4. runOptimise directly runs the climate model.
5. The next iteration of the optscript loop calls runOptimise again.

8 Adding a New Climate Model

Inputs (e.g. parameter values) and outputs (e.g. misfit error to minimise) specific to the climate model need to be sorted by a Python class. For example see HadCM3.py, MiniMOPS.py, or MOPS.py. This class should use ModelSimulation.py to inherit all necessary files from the rootDir and place copies in the new model folder. It could also, for example, have functions to write the model input parameter file for this model run ("setParams"), read the misfit output file ("readObs"), and submit the script to run the climate model. To introduce a new climate model to OptClim, both on your local machine and a supercomputing cluster, see the following instructions.

8.1 Local machine

8.1.1 Step 1

1. In .bash_profile or .bashrc make sure $OPTCLIMTOP is location to this OptClim folder, and that OPTCLIMTOP is in your PYTHONPATH.
2. Under OptClim/Configurations create a new _Configs folder named appropriately for the model e.g. OxofrdMiniMOPS_Configs.
3. Within a folder in here e.g. optTestExample place all the code, runscripts and files needed to run the model. This is the ROOTDIR, and will get replicated for every algorithm iteration.
4. Create a JSON configuration file specific to this model and optimisation e.g. OxfordMiniMOPS_dfols.json. Don't have this in ROOTDIR, but just above, and labelled helpfully. This file should contain the initial conditions of the parameters, their ranges etc.

8.1.2 Step 2

1. Create a new folder (which is now TESTDIR) under OptClim labelled according to the model you're using (e.g. OxfordMiniMOPS) and add it to your PYTHONPATH in .bash_profile.
2. Create a python script creating the model class with outputs to a temporary directory e.g. MiniMOPS.py.
3. Create a coinciding test class e.g. test_MiniMOPS.py with the correct ROOTDIR and TESTDIR, to check each function within the model class works correctly.
4. Run this test case and check all works.

8.1.3 Step 3

1. Go into OptClimVn2 or OptClimSO (whichever version you want to use) and run all test cases to check all working on your machine (if you need to change any of the python scripts in here then copy them to your TESTDIR for editing, and make a coinciding test script to check it. This is likely to be needed for runOptimise.py for a new model, and Submit.py for a new cluster e.g. ARC.

2. Edit runOptimise and/or Submit python scripts if necessary, specifically where they call the modules from (OptClimV2 or OptClimSO).
3. Run at the command line:
   python runOptimise.py –restart -dir outputTest -v -t
   ../Configurations/OxfordMiniMOPS_Configs/OxfordMiniMOPS_dfols.json &
   >outputTest/progress.txt - Specify correct JSON file
   - This runs the first optimisation iteration by copying everything from ROOTDIR into a test0 folder within a folder called outputTest.
   - The restart command clears every test folder within outputTest and starts from the beginning
   - Check all works and output test folder contains the right stuff and has right parameter_inputs.txt and misfit_outputs.txt
   - If getting errors do python -m pdb ... etc then put in breakpoints (b 354)r=run, c=continue, n=next
   - The & >.txt at the end outputs all the terminal statements to a text file
4. Run at the command line:
   python runOptimise.py -dir outputTest -v -t
   ../Configurations/OxfordMiniMOPS_Configs/OxfordMiniMOPS_dfols.json
   - This carries on from whichever iteration it got to in this outputTest folder
   - Check test1 in outputTest contains right stuff
5. Repeat 4 many times
6. Repeat 4 but with -m at the end of the command
   - this plots the progress of the optimisation
   - this -m does not have to be run with every runOptimise command - it will plot all info available in the json file

8.2 Computing Cluster

8.2.1 Step 1

1. Copy all to appropriate place on cluster
2. Add $OPTCLIMTOP and necessary paths to .bashrc
3. Load anaconda (module load python/anaconda2/5.0.1)
4. Create environment (conda create -n myenv) or activate if previously made (source activate myenv)
5. Install necessary packages like pandas numpy scipy netCDF4 xarray f90nl ext (last 2 not working at the mo - seem to be fine without)
6. Install pip
7. Use pip to install dfols and bobyqa like before
8. Run test as in Step 2.3
9. Run tests as in Step 3.1
10. Run the model in ROOTDIR to check works on cluster (may need to re-create the compiled scripts using gcc runtest.c -o runtest -lm)
11. runOptmise as in Step 3.4 (no cluster submissions yet)

8.2.2 Step 2: If using OptClim/OptClimV2

If using OptClim/OptClimV2 (Submits a new job for each optimisation iteration and model run).

1. Edit the runscript in ROOTDIR to include anything needed for the job submission at the top of the runscript e.g. #sbatch nodes = 1, #SBATCH –ntasks-per-node=1, #SBATCH –time=00:00:30
2. Actually submit the updated runscript in ROOTDIR and check it works (sbatch –export=All –error="fred%j.out" runscript)
3. Edit arcSubmit or eddieSubmit in Submit.py if necessary (or make new one for your cluster) and make sure runOptimise will now go through the correct def to submit
4. runOptimise as in Step 3.3 but without the -t (no longer a test)
5. Once completed run again (without restart!) but with -m on the end so that it plots everything (but won't submit any model runs as doesn't need to).

8.2.3 Step 2: If using OptClimSO

If using OptClimSO (Subits a single large job, and runs each optimisation iteration and model run within this main job).

1. Edit the optscript (e.g. $OPTCLIMTOP/OxfordMOPS/optscript_t6b) at the top of the runscript to suit the job submission system e.g. #sbatch nodes = 1, #SBATCH –ntasks-per-node=1, #SBATCH –time=00:00:30.
2. Edit the optscript set the number of optimisation iterations to loop through, the output directory, and the output progress text file to write to.

9 Optimisation Algorithms

9.1 Incorporated Optimisers

There are currently 4 optimisation algorithms incorporated into OptClim. Py_BOBYQA (Cartis et al, 2018) is a Python-based implementation of BOBYQA (Bound Optimisation BY Quadratic Approximation), which is an iterative method of minimising a function $f(x)$ with bounds on the variables, by Powell (2009). The Python implementation of derivative-free optimisation using least squares (DFOLS) algorithm (Cartis et al, 2018) works very similar to Py-BOBYQA, but takes more of the problem structure into account. Most recent version of Py_BOBYQA can be found here: https://github.com/numericalalgorithmsgroup/pybobyqa. Most recent versions of DFOLS can be found here: https://github.com/numericalalgorithmsgroup/dfols. Both Py_Bobyqa and DFO-LS are derivative-free optimisation algorithms, while Gauss-Newton requires derivatives. These 3 optimisers were incorporated into OptClim by Prof. Simon Tett, and are deterministic local optimisers. A stochastic global optimisation method requiring more function evaluations is the Covariance matrix adaptation evolution strategy (CMA-ES) (Hansen, 2016), which is nicely explained in application by Kriest et al (2017).

9.2 Adding a new algorithm to OptClim

Download and compile the new optimiser to it's own sub directory within OptClim and add the necessary paths. Code another branch to the **if** statement in the main block of code of runOptimise.py, to be executed in the case of the user specifying this new optimiser. This branch should collect the inputs required for the optimiser, run the optimiser, then read the information output by the optimiser needed for OptClim to continue.

10 OptClimSO Plotting Scripts

GitHub Source:

Here are several plotting MATLAB scripts to visualise the progress of optimisation. Some edits will need to be made for different cases.

Top Script **— Function Called**	**Purpose**
plotAll.m	Plot the progress of OptClim in this output directory, if using BOBYQA or DFO-LS.
— checkOptMatches.m	Check that the model run inputs/outputs match the text printed by BOBYQA / DFO-LS.
— plotParamTraject.m	Plot the parameter values throughout the optimisation by BOBYQA / DFO-LS. See example in Figure 3.
— plotTracerMisfit.m	Plot the total and individual misfit values throughout the optimisation by BOBYQA / DFO-LS. See example in Figure 4.
— plotParamPairMisfits.m	In one figure plot all parameter and total misfit values throughout the optimisation by BOBYQA / DFO-LS for each parameter pair. See example in Figure 5.
— plotParamPairProgress.m	In one figure per iteration, plot the parameter and total misfit values during the optimisation by BOBYQA / DFO-LS for each parameter pair. See example of a single iteration in Figure 5.
plotAllCmaes.m	Plot the progress of OptClim in this output directory, if using CMA-ES.
— plotParamTrajectCmaes.m	Plot the parameter values throughout the optimisation by CMA-ES.
— plotTracerMisfitCmaes.m	Plot the total and individual misfit values throughout the optimisation by CMA-ES.
— plotParamPairMisfitsCmaes.m	In one figure plot all parameter and total misfit values throughout the optimisation by CMA-ES for each parameter pair.
plotAllCompare.m	Compare the progress of OptClim by different optimisers in these output directories.

— plotCompare.m	Create 2 plots. (1) Plot on top of each other the total misfit values throughout the optimisation by different optimisers. See example in Figure 6. (2) Plot on top of each other the parameter values throughout the optimisation by different optimisers. See example in Figure 7.
— plotCompareMisfits.m	Compare the individual misfit values throughout the optimisation by different optimisers. See example in Figure 8.
— plotParamRelations.m	Create one figure per parameter pair comparing the parameter tuning of one parameter pair by 2 different optimisers. This helps spot any non-independent parameters which may influence the tuning process. See example in Figure 9.

Table 1: OptClimSO Plotting Scripts.

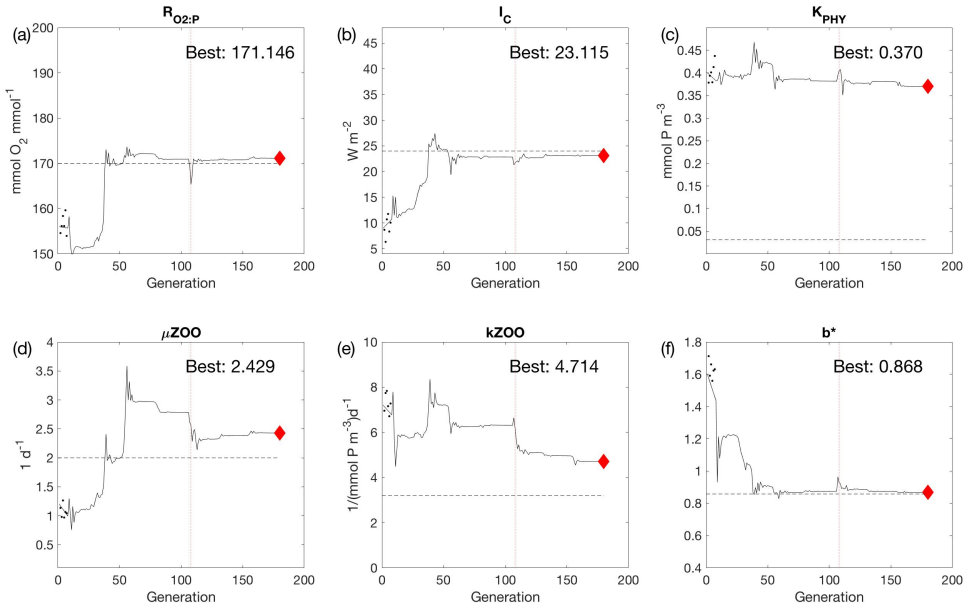

Figure 3: Tuning 6 parameters by DFOLS, created by plotParamTraject.m.

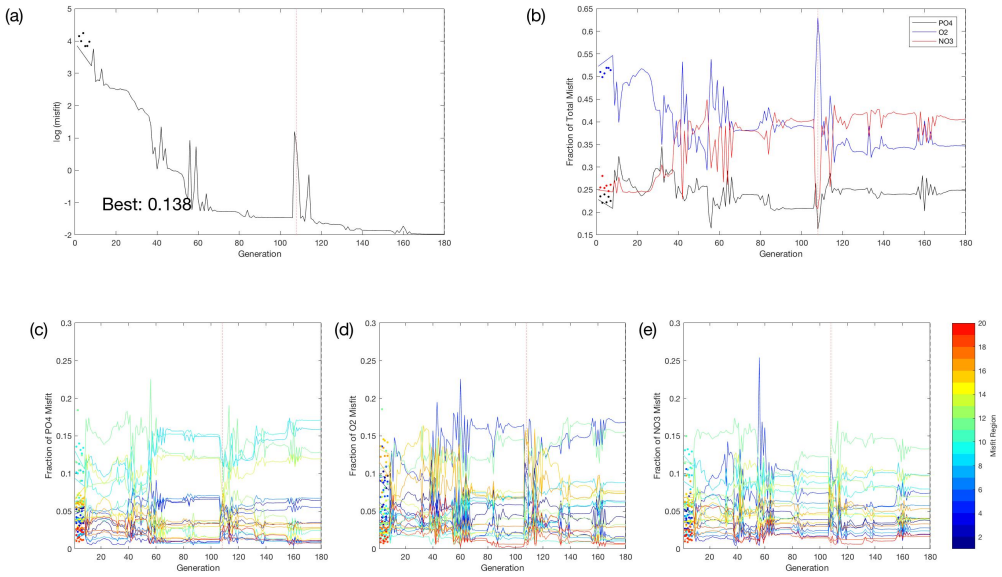

Figure 4: Minimising the misfit with DFOLS, created by plotTracerMisfit.m. Misfits have been broken down into different tracers/regions, which is very specific to this case.

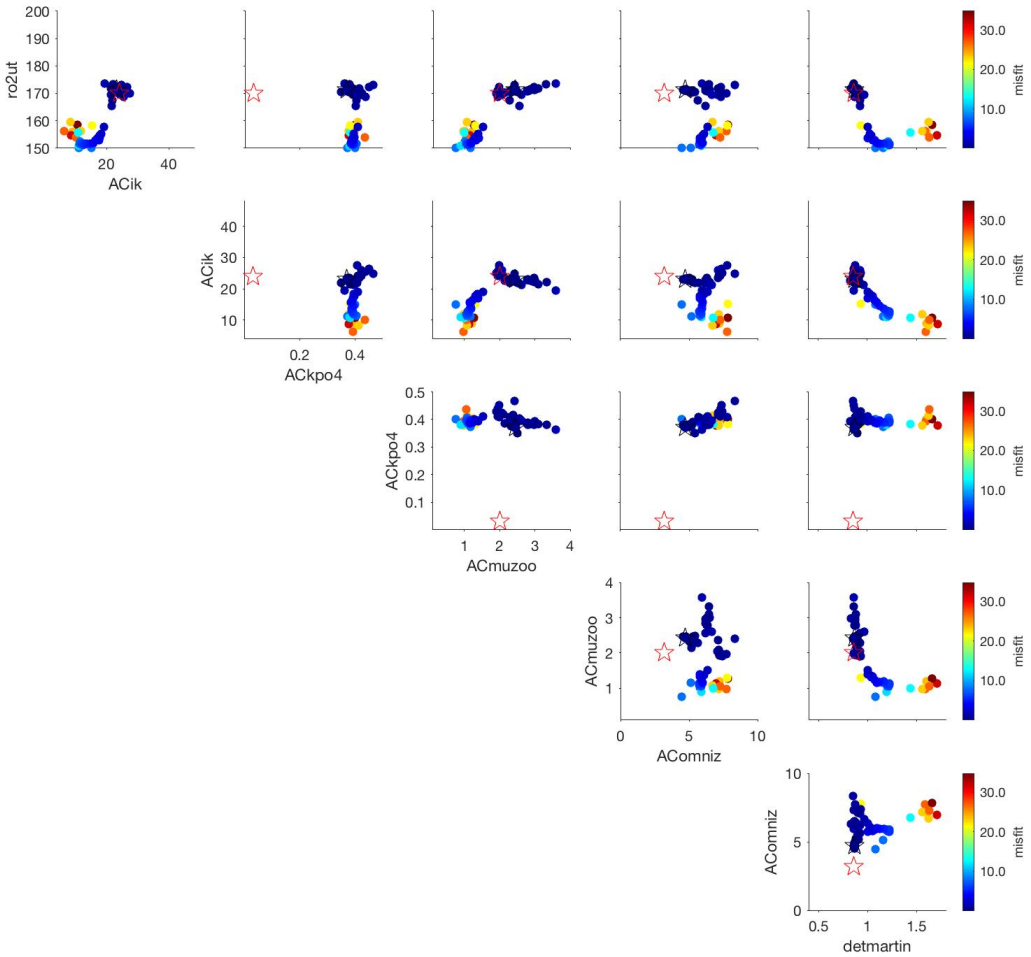

Figure 5: How the misfit is reduced in relation to each parameter pair being tuned by DFOLS, created by plotParamPairMisfits.m.

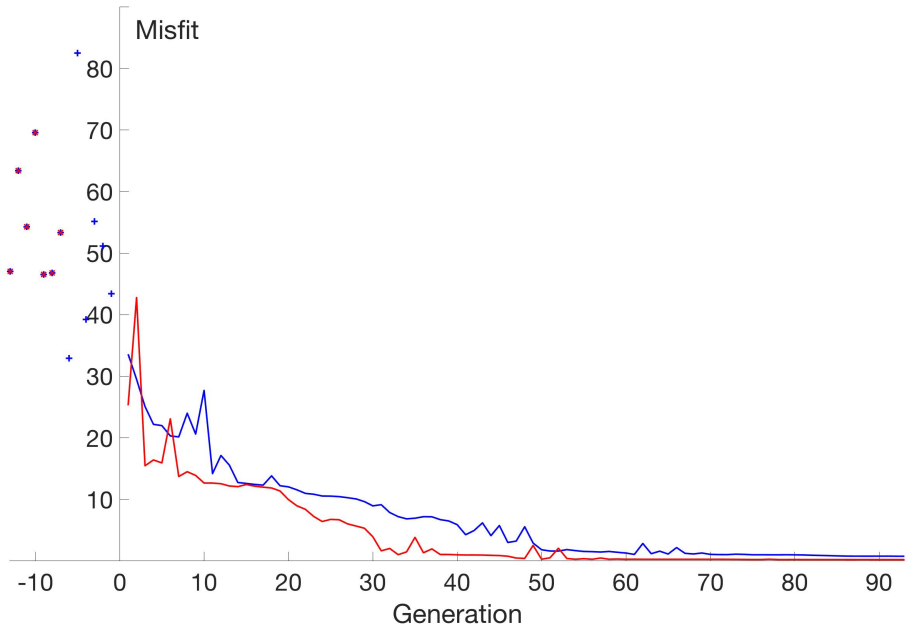

Figure 6: Minimising the misfit with DFOLS and BOBYQA, created by plotCompare.m.

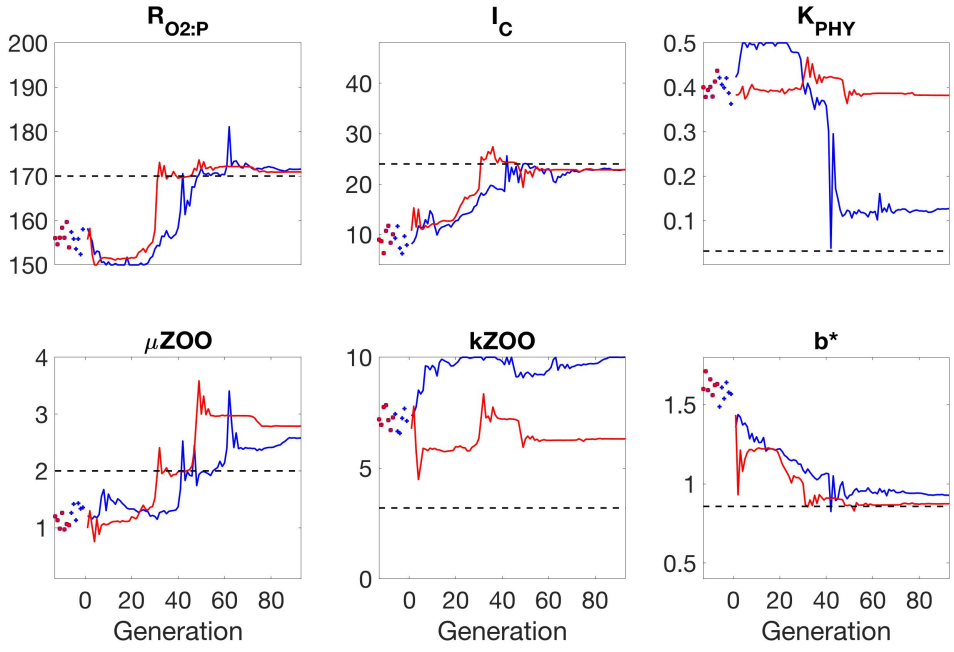

Figure 7: Tuning 6 parameters by DFOLS and BOBYQA, created by plotCompare.m (similar to plot created by plotParamTraject.m).

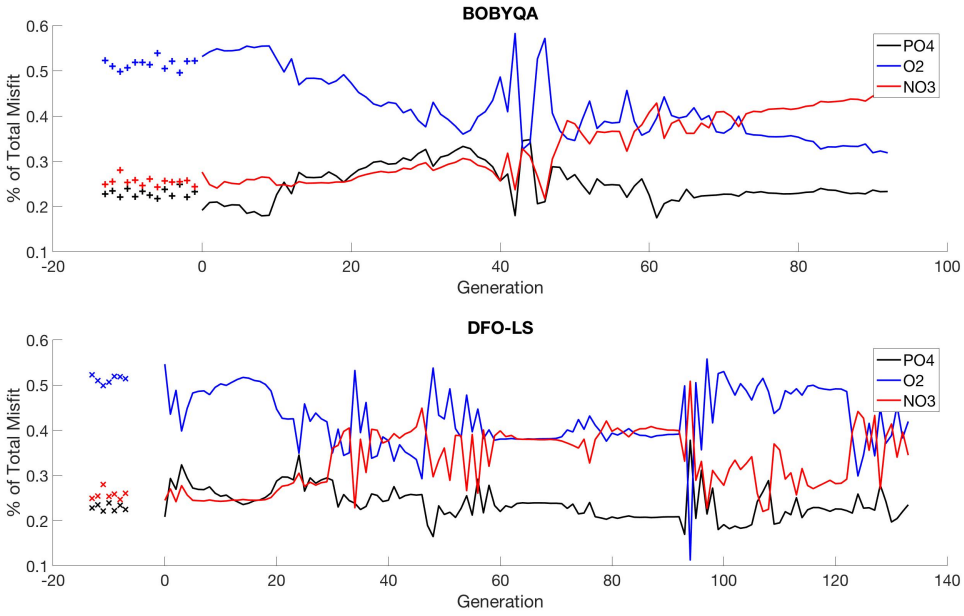

Figure 8: Minimising 3 individual misfits with DFOLS and BOBYQA, created by plotCompareMisfits.m. Misfits have been broken down into different tracers, which is very specific to this case.

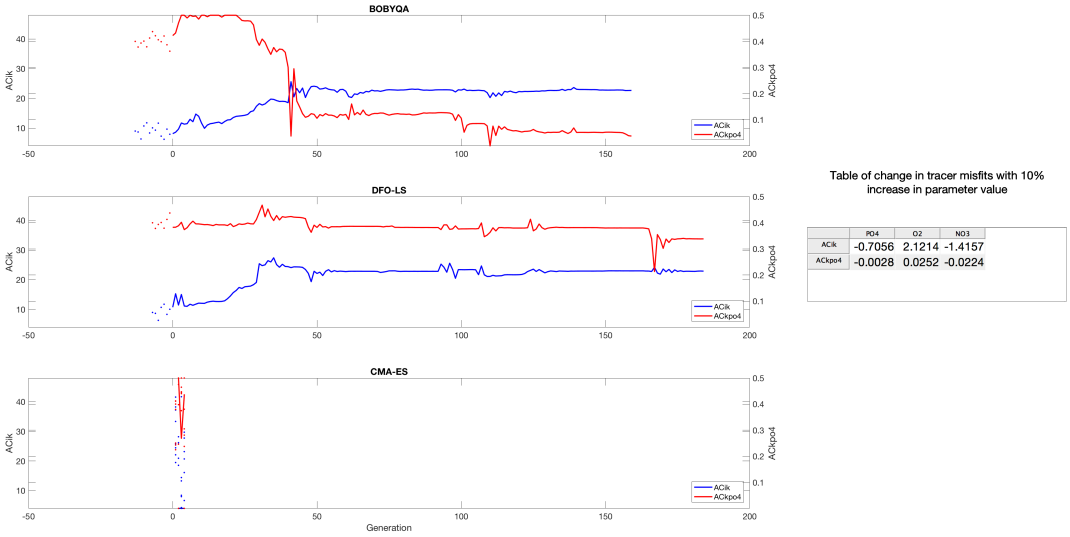

Figure 9: Compare the tuning of each parameter pair by multiple different optimisation algorithms, created by plotParamRelations.m.

11 References

Cartis, C., Fiala, J., Marteau, B. & Roberts, L. (2018), 'Improving the Flexibility and Robustness of Model- Based Derivative-Free Optimization Solvers', arXiv preprint arXiv:1804.00154 pp. 1-35. URL: http://arxiv.org/abs/1804.00154

Hansen, N. (2016), 'The CMA Evolution Strategy: A Tutorial', arXiv preprint arXiv:1604.00772 . URL: http://arxiv.org/abs/1604.00772

Kriest, I., Sauerland, V., Khatiwala, S., Srivastav, A. & Oschlies, A. (2017), 'Calibrating a global three- dimensional biogeochemical ocean model (MOPS-1.0)', Geoscientific Model Development 10(1), 127-154.

Powell, M. (2009), 'The BOBYQA algorithm for bound constrained optimization without derivatives', NA Report NA2009/06 p. 39. URL: http://www6.cityu.edu.hk/rcms/publications/preprint26.pdf