Technical Report for ARCHER2-eCSE02-02: Reducing UM-UKCA data output using flight-track simulation

Maria R. Russo, Nathan Luke Abraham
National Centre for Atmospheric Science, Department of Chemistry, University of Cambridge, Cambridge, U.K.

Abstract
The United Kingdom Chemistry and Aerosols model (UKCA) [1] is a component of the Met Office Unified Model (UM) [2]. In this project we have developed a stand-alone flight-track simulator code that can be embedded into the UM-UKCA workflow. This code outputs high resolution data on a number of pre-defined flight tracks by interpolating modelled dynamical, chemical, and aerosol fields from the model grid to the flight times and locations. This data is then written to a CF-compliant NetCDF file that can be efficiently stored, due to its small size, and more easily compared to flight observations. This method achieves a large reduction in the size of model data being produced for comparison with flight data. By interpolating global, hourly files onto flight-track locations, we reduce data output for a typical climate resolution run, from ~3 Gb per model variable per month to ~15 Mb per model variable per month (a 200 times reduction in data).

1. Introduction
The UM-UKCA model allows users to run interactive chemistry and aerosol schemes with the UM. As a whole-atmosphere composition-climate model, UM-UKCA simulates large numbers of chemical and aerosol tracers using its stratosphere-troposphere (StratTrop) chemistry [3] and GLOMAP-mode aerosol [4] schemes. Other chemistry schemes under development include CRI-Strat [5,6]. As well as chemical and aerosol tracers, model variables include numerous dynamical fields and further diagnostics.

In order to validate the chemistry schemes used in UM-UKCA, a wide variety of observational datasets are used, such as ozone sondes, satellite observations, surface stations, and observations from research aircrafts. Evaluating model configurations against flight data is a useful tool to understand model biases and to help improve model performance; moreover, model simulations can be used effectively to help interpret data from flight campaigns. However, comparison between climate or weather forecast models and aircraft data is not straightforward. Data from research aircrafts is defined on the aeroplane flight track; this means that each point in time for the observations corresponds to a fixed point in 3D space (associated to the longitude, latitude and pressure of the research aircraft at that time), with observations taken at a high temporal frequency. However, each point in time for modelled data corresponds to a large number of longitudes, latitudes and pressure, defined by the model 3D grid. Model data is also typically only available every 20 minutes or every hour due to the dynamical and chemical timesteps used.

To be able to compare to flight-track data correctly, users in the past have output hourly (or higher frequency) fields over a large part of the atmospheric domain (typically limiting only in height). As well as using a large amount of storage space, handling such large files is very time-intensive: extracting the hourly data from a tape archive typically requires ~3 hours per model year, followed by several hours to read and interpolate these fields down to the single-point of the flight track. This past method led to orders of magnitude more data being generated, stored, and processed than is actually required, and significant amounts of time and computer resources spent to extract, read and interpolate model data onto flight tracks.

A previous attempt at producing model data on flight track code in a more efficient way was made several years ago, by embedding a flight_track routine (using Fortran programming language) within the UKCA source code [7]. However, this approach leads to a number of problems:

1) The additional code will add a computational burden to the running of the UM-UKCA main code, potentially slowing down the model integrations.
2) Model diagnostic related to some UM dynamical fields are not easily available within the UKCA code.
3) Embedding code within the UM-UKCA model trunk requires strict coding practices and model tests have to be designed and run to ensure the code runs properly and remains effective as the structure of the UM-UKCA code evolves over time.
4) The format of output files is restricted to the UM internal file formats (e.g. UM fieldsfile) which is not an easy-to-handle and internationally recognised format.
5) Due to strict licensing of the UM-UKCA code, any code developed which is embedded within UM-UKCA cannot be easily made available.

As a result of the above points, the flight_track routine used in Telford et al. [7] was never ported to further versions of UM-UKCA.

2. Method Overview

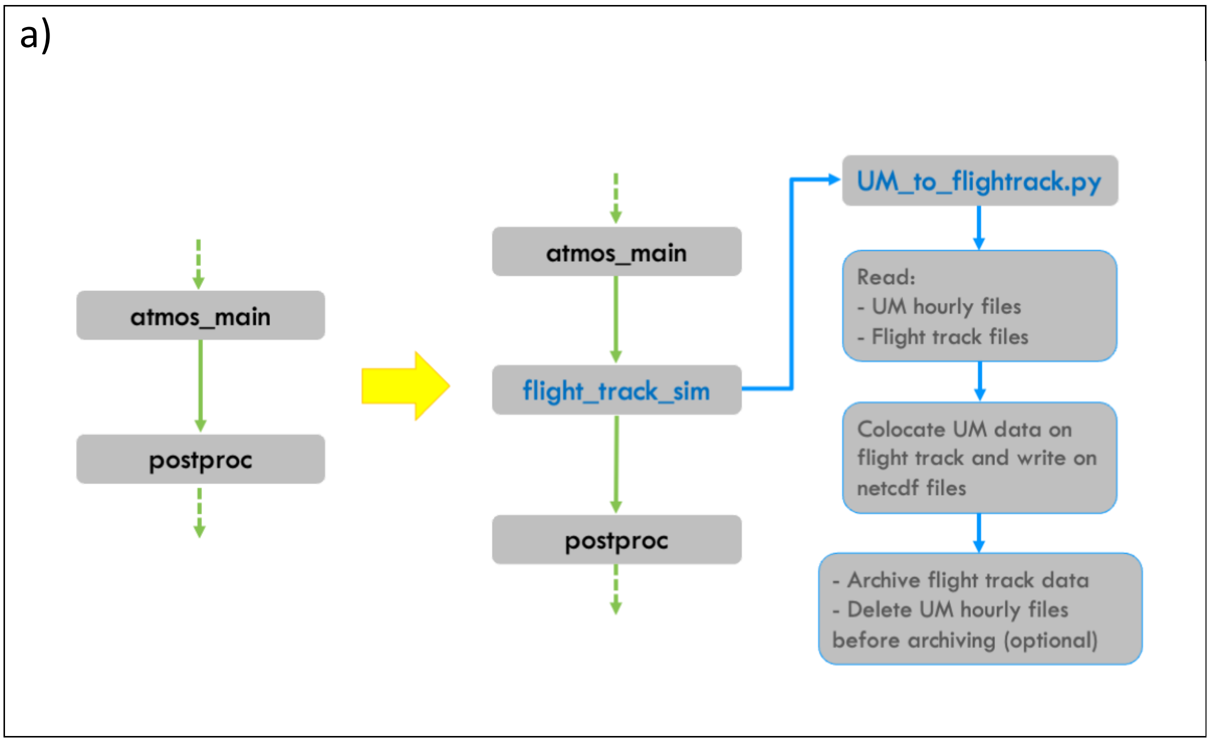
This section describes the method developed to allow model users to easily produce model data on defined flight tracks as the model runs.

Instead of outputting and archiving large, hourly, 3D, gridded model fields, of which only a small fraction is used, the code we have developed produces and archives much smaller data files on flight tracks coordinates, therefore greatly reducing data storage requirements and the time and resources necessary to process the data.

A flight track code (UM_to_flightrack.py) was developed using the python programming language. This python code makes use of the Community Intercomparison Suite (CIS) python library [8] interfaced to the cf-python library [9] for reading-in UM-format data files. This new code is embedded into the UM-UKCA run-time workflow, by creating a new Rose/Cylc 'app', flight_track_sim, which is inserted after the model timestep is completed and before the postprocessing step. Rose[10] is the graphical user interface (GUI) for the UM, and the Cylc[11] workflow engine is used to schedule the UM and other necessary tasks on the HPC batch system. A schematic of the UM workflow with the flight track simulator step (flight_track_sim) is shown in Fig 1.

Since this code is inserted into the UM-UKCA run-time workflow, it does not have any of the problems listed in the previous section, which are typically associated with code developed within the main UM-UKCA source code (e.g. similar to Telford et al. [7]). This method also has the following advantages:

1) Model data on the flight track is output using the internationally recognised NetCDF, CF-compliant format, making handling and analysis quicker and easier for users.
2) As well as being inserted into the UM-UKCA workflow, the code has optional logical input that allow it to run as a postprocessing tool, for example to analyse data from older simulations or to test the impact of different interpolation methods.
3) The code can be easily customised to process any model data (not just UM-UKCA), therefore making it useful to the wider atmospheric science community.


a)
atmos_main
postproc
atmos_main
flight_track_sim
postproc
UM_to_flightrack.py
Read:
- UM hourly files
- Flight track files
Colocate UM data on flight track and write on netcdf files
- Archive flight track data
- Delete UM hourly files before archiving (optional)

b)

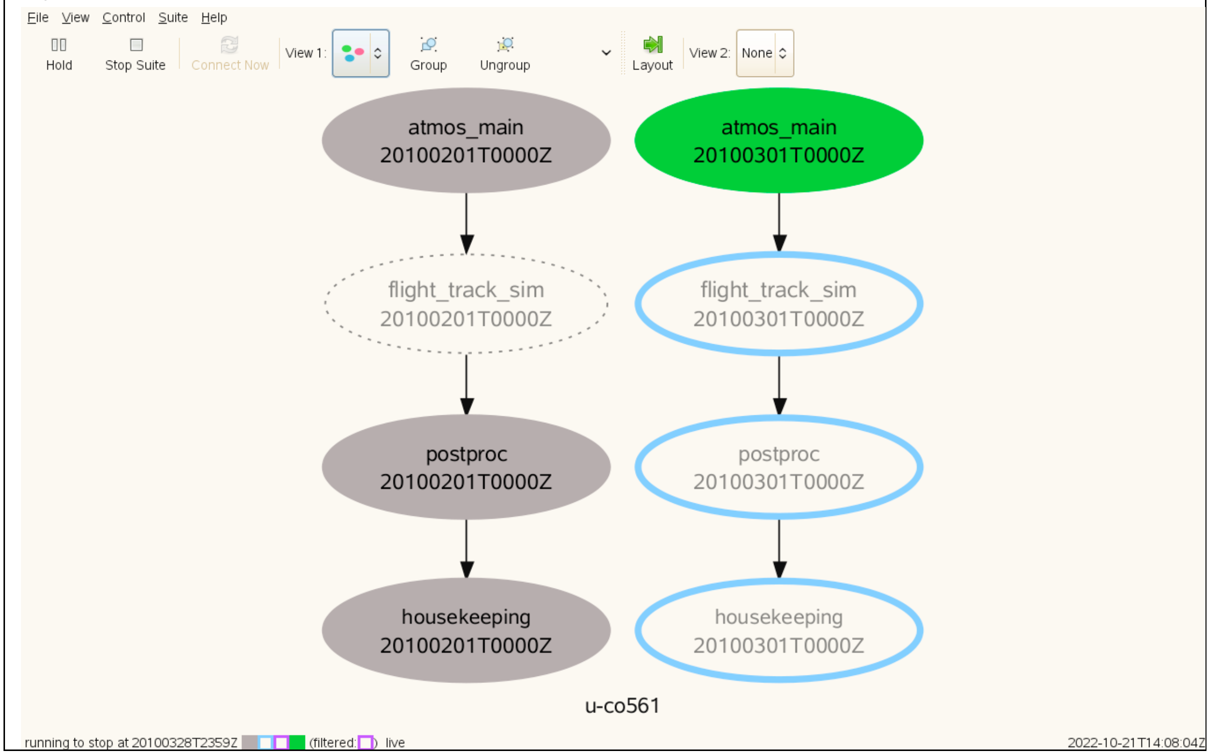

File View Control Suite Help
Hold
Stop Suite
Connect Now
View 1:
Group
Ungroup
Layout
View 2: None
atmos_main 20100201T0000Z
atmos_main 20100301T0000Z
flight_track_sim 20100201T0000Z
flight_track_sim 20100301T0000Z
postproc 20100201T0000Z
postproc 20100301T0000Z
housekeeping 20100201T0000Z
housekeeping 20100301T0000Z
u-co561
running to stop at 20100328T2359Z
(filtered:) live
2022-10-21T14:08:04Z

Figure 1. a) shows a schematic of the UM workflow indicating where the flight track simulator step fits: flight_track_sim reads input variables and logicals provided by the user through the Rose GUI and runs the python script UM_to_flightrack.py; b) shows the UM run-time control GUI for an example run which includes the flight_track_sim step: these include running the main UM-UKCA code (atmos_main), interpolating model output on predefined flight tracks (flight_track_sim), data handling and archiving (postproc) and deleting files from the user workspace (housekeeping). Grey colours indicate that a task has been completed, green shows a task that is running and blue shows tasks that are awaiting to run.

3. Changes to the UM Rose suite

This section describes in more details how the flight-track simulator code is embedded into the UM-UKCA workflow.

We created a new Rose/Cylc app, ‘flight_track_sim’, which reads input variables and logicals provided by the user through the Rose suite GUI. File changes required to include flight_track_sim into a UM Rose suite are described below:

1) Create a new directory within the UM runid directory, roses/*$runid*/app/flight_track_sim (where *$runid* is the UM job identifier, eg u-co561). This directory contains a rose-app.conf file and a bin subdirectory containing the python script UM_to_flightrack.py.
2) rose-app.conf defines input variables and launches UM_to_flightrack.py. An example of rose-app.conf file is shown below:

```
[command]
default=python3
${UM_data_dir}/../../../app/flight_track_sim/bin/UM_to_flightrack.py --outdir
= ${Additional_output_dir} --inputdir = ${UM_data_dir}  --trackdir =
${Flight_dir} –ppstream = ${hourly_data_ppstream}     --jobid = ${jobid} --
date = $YEAR_MONTH batch --archive_hourly = ${archive_hourly_ppstream}

[env]
Additional_output_dir=/work/n02/n02/mrr32/Flight_output/
```

```
  Flight_dir=/work/n02/n02/mrr32/AeroCom_flight_track_source/
  UM_data_dir=$DATAM/
  archive_hourly_ppstream=True
  hourly_data_ppstream=1
  jobid=$RUNID
```

3) The UM_to_flightrack.py file can be downloaded from GitHub (https://github.com/MariaRusso/CF-CIS-Iris_python_tools) and copied to roses/*$runid*/app/flight_track_sim/bin/.
4) Include a flight_track_resource 'block' in roses/$runid/site/archer2.rc. See below, which shows the added text highlighted in blue:

```
    [[PPBUILD_RESOURCE]]
        inherit = HPC_SERIAL
        [[[job]]]
            execution time limit = PT5M

    [[FLIGHT_TRACK_RESOURCE]]
        inherit = HPC_SERIAL, RETRIES
        [[[job]]]
            execution retry delays = PT5M, PT5M, PT10M, PT10M
            execution time limit = PT1H

    [[POSTPROC_RESOURCE]]
        inherit = HPC_SERIAL
        pre-script = """module load postproc
                        module list 2>&1
                        ulimit -s unlimited
                     """
```

5) Modify roses/$runid/suite.rc by adding the highlighted text as shown below:

```
{# Command for UM must make sure using main executable #}
{% set UM_TASK_RUN_COMMAND = TASK_RUN_COMMAND ~ ' --
path="share/fcm_make_um/build-*/bin"' %}

{# Set rose date command and associated print-format options #}
{% set ROSEDATE = "rose date -c --calendar=" ~ EXPT_CALENDAR %}
{% set PFMT_YR = "--print-format='%Y'" %}
{% set PFMT_FLIGHTMONTH = "--print-format='%Y%m'" %}
{% set PFMT_MONTH = "--print-format='%Y%b'" %}
{% set PFMT_DUMP = "--print-format='%Y%m%d_%H'" %}
{% set PFMT_UM_PT = "--print-format='%Y,%m,%d,%H,%M,%S'" %}
{% set PFMT_UM_DUR = "--print-format='y,m,d,h,M,s'" %}

{# Set jinja2 variables based on values from rose-suite.conf file #}
{% set CONFIG_OPT = '(' ~ EXPT_CONFIG ~ ') (' ~ EXPT_HORIZ ~ ') (' ~
EXPT_CALENDAR ~ ') ' ~ EXPT_AEROSOLS %}
```

```
…………
      {# Set up cycling graph #}
      {% set RESUB_GRAPH = '' %}
      {% set RESUB_GRAPH = RESUB_GRAPH ~ 'flight_track_sim => ' %}
      {% set RESUB_GRAPH = RESUB_GRAPH ~ 'postproc => ' if TASK_POSTPROC else
RESUB_GRAPH %}
      {% set RESUB_GRAPH = RESUB_GRAPH ~ 'pptransfer => ' if TASK_PPTRANSFER else
RESUB_GRAPH %}
      {% set RESUB_GRAPH = RESUB_GRAPH ~ 'supermean => ' if TASK_SUPERMEAN else
RESUB_GRAPH %}
      {% set RESUB_GRAPH = RESUB_GRAPH ~ 'rose_arch_logs => ' if TASK_ARCH_LOG else
RESUB_GRAPH %}
      {% set RESUB_GRAPH = RESUB_GRAPH ~ 'housekeeping' %}

      [[[ {{EXPT_RESUB}} ]]]
           graph = atmos_main => {{ RESUB_GRAPH }}
…………
```

```
 [[atmos_main]]
     inherit = RUN_MAIN, ATMOS_RESOURCE, ATMOS
     post-script = save_wallclock.sh {{EXPT_RESUB}}

 [[fcm_make_pp]]
     inherit = RUN_MAIN, EXTRACT_RESOURCE
 [[fcm_make2_pp]]
     inherit = RUN_MAIN, PPBUILD_RESOURCE

 [[flight_track_sim]]
     inherit = RUN_MAIN, FLIGHT_TRACK_RESOURCE
     pre-script = '''
       export PATH=/home/n02/n02/mrr32/miniconda3/bin:$PATH
       export
UDUNITS2_XML_PATH=/home/n02/n02/mrr32/miniconda3/share/udunits/udunits2.xml
                    '''
      [[[environment]]]
          ROSE_TASK_APP = flight_track_sim
          YEAR_MONTH = $({{ROSEDATE}} {{PFMT_FLIGHTMONTH}})
          CYCLEPERIOD = {{EXPT_RESUB}}

 [[POSTPROC]]
      [[[environment]]]
          CYCLEPERIOD = {{EXPT_RESUB}}
```

4. Description of UM_to_flightrack.py

UM_to_flightrack.py performs the following steps:

1) Read air pressure and campaign name from flight track files, using CIS python libraries.
2) Read all model variables and Heaviside functions on model pressure levels, from hourly UM-format files, using cf-python libraries, and remove grid points for which the Heaviside function is zero.
3) Colocate model variables onto flight track (using CIS python libraries).
4) Write daily NetCDF file containing model variables colocated onto flight track (using CIS python libraries).
5) Read daily NetCDF files and write monthly NetCDF, CF-compliant files (one file per model variable per month).
6) If archiving of hourly UM-format files is set to False, delete hourly files before the postproc step.

4.1 Input

Command line arguments

The flight track simulator app, fligh_track_sim, provides input variables from the Rose GUI and invokes UM_to_flightrack.py parsing such variables as command line arguments. A list of input variables required to run UM_to_flightrack.py, their description and usage is shown in Table 1.

ARGUMENT	DESCRIPTION
-i --inputdir *Directory_in*	*Directory_in* is the full path to the directory containing hourly pp files
-t --trackdir *Directory_ft*	*Directory_ft* is the full path to the directory containing flight track files
-d --cycle_date *YearMonth*	*YearMonth* is a six digit tag to identify the start time of the analysis
-n --n_months *N*	*N* is the number of months to process from/including *YearMonth* (optional; default 1)
-r --runid *UM_jobid*	*UM_jobid* is the unique identifier associated to a UM integration

-p --ppstream *Single_char*	*Single_char* is a single character identifying the hourly data ppstream as defined in Rose, e.g. k
-m --method *Interpolation*	*Interpolation* is “lin”/“nn” for linear or nearest neighbour interpolation (optional; default lin)
-c --climatology *True/False*	*True* produces a multi-year climatology for each flight (optional; default False)
-o --outdir *Directory_out*	*Directory_out* is the location to write output NetCDF files (optional). If *batch* is selected, output files are always written to *Directory_in* and additionally copied to *Directory_out* if present. If *postprocessing* is selected, output files are written to the current directory (./) or to *Directory_out* if present)
batch	Indicates the python script is running within the UM run-time workflow
-a --archive_hourly	*True* to archive hourly files instead of deleting them (optional; default True)
postprocessing	Indicates the python script is running outside the UM run-time workflow
-s --select_stash *Code*	*Code* is a list of stashcodes to be processed (optional; default = process all)

Table 1. Description of command line arguments used to run UM_to_flightrack.py

A subparser argument, ‘jobtype’, is used to indicate whether the code is running within the UM-UKCA run-time workflow (if ‘batch’ is selected) or as a standalone postprocessing tool, eg on existing model data, (if ‘postprocessing’ is selected). These subparser arguments also unlock specific conditional arguments: --archive_hourly can be used only if ‘batch’ is selected and --select_stash can only be used if ‘postprocessing’ is selected.

Input files

Suitable formats for model input are NetCDF, UM pp format and UM fieldsfile format, while input files containing flight track information are required to be in NetCDF, CF-compliant format. The ability to read different formats of model input files gives extra flexibility to the code as it allows to read other model data as well as UM-UKCA data.

4.1 Code Optimisation

There are several python libraries that can deal with reading/writing of large, gridded data files (e.g. CIS, cf-python, Iris [12] etc.). The choice to use CIS python libraries stems from their ability to handle ungridded data (such as data on a flight track) and the ease of performing colocation from gridded to ungridded data. However, preliminary tests showed that reading model input files using CIS was significantly slower than reading the same file with Iris or cf-python. For a typical climate resolution file, containing 24 hourly values for 7 variables on 19 vertical levels, read times where ~30 minutes for CIS, ~6 minutes for Iris and ~30 seconds for cf-python. Given that potentially many such files would need to be read in each model month, using CIS to read model data would be unfeasible. For this reason, cf-python was chosen to read the model data. However, CIS and cf-python use very different data structures for the gridded variables they read. In order to overcome this problem, a python function was developed to convert the cf-python gridded data structure to the CIS gridded data structure. This work was then extended to produce similar functions which convert cf-python gridded data structure to Iris or xarray data structure. These functions are more widely useful as they allow users to efficiently read large datasets using the fast cf-python libraries and then convert to the desired gridded data structure to interface with python code using CIS, Iris or xarray libraries. These functions can be found in the convert_CFvars.py python module available on GitHub (https://github.com/MariaRusso/CF-CIS-Iris_python_tools).

Since reading model data and dividing by the Heaviside function are the slowest steps in UM_to_flightrack.py, we further optimized the code by only reading model days for which a flight track input file exists.

4.2 Output

The model data output on flight track is generated in a NetCDF, CF-compliant format. The size of monthly output generated by UM_to_flightrack.py depends on the number and size of the flight track files on which the model data is colocated and therefore can vary each month.

Figure 2 shows an example of comparison between modelled ozone and ozone measured by the FAAM research aircraft in Jan 2010. This type of comparison can help to identify and improve model biases.

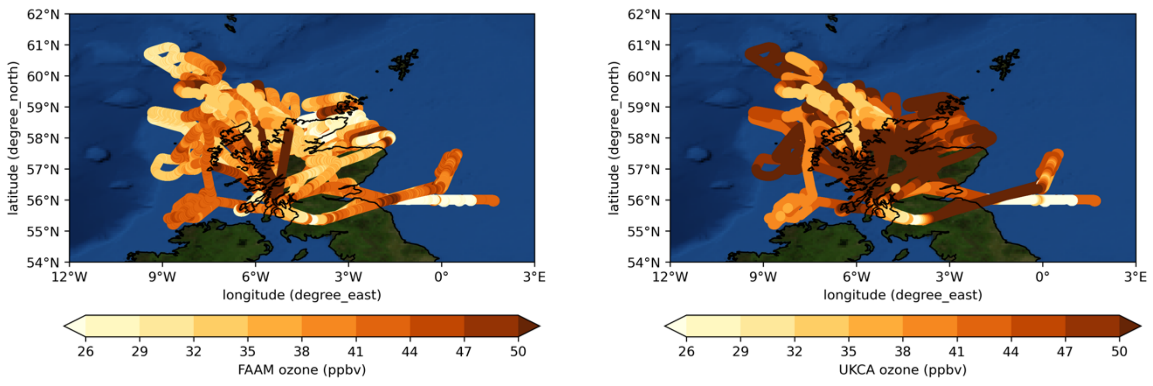


Figure 2. Comparison between observed (left) and modelled (right) ozone for 12 flights in January 2010.

Analysis of ozone as a function of air pressure is shown in Figure 3: the largest bias between UKCA and FAAM ozone is found for air pressure values lower than 400 hPa (or altitudes greater than 7-8 km); such pressure values are close to the tropopause for mid-latitude winter months and therefore the high modelled ozone bias could be due a model underestimate in tropopause height, leading to modelled stratospheric ozone being sampled.

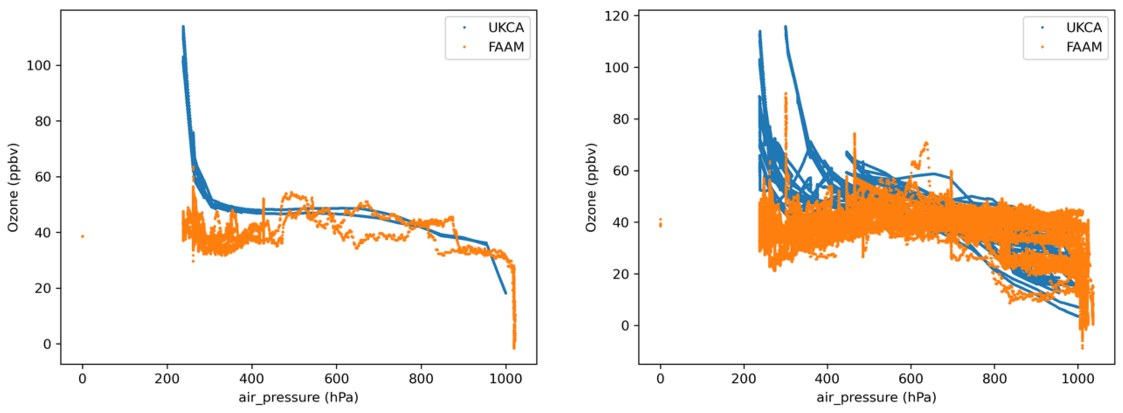


Figure 3. Comparison between modelled and observed ozone for one flight on 18th of January (left) and all flights in January 2010 (right).

5. Conclusions

The ability to sample Unified Model output along observed flight tracks allows for better model evaluation. However, to do this usually requires the processing of large volumes of high frequency gridded model data. By interfacing with the CIS python library, we are able to automate this step, greatly reducing post-processing time and the volume of data that needs to be saved following a model simulation. This method is also transferable to other atmospheric models, and the code is provided on GitHub under an open-source license. The use of the cf-python library to read-in the UM-format files significantly decreases the time take to read these files when compared to the Iris or CIS libraries.

6. Code availability

The code is available under a permissive BSD-3 license. See link for details: https://github.com/MariaRusso/CF-CIS-Iris_python_tools

Acknowledgements

This work was funded under the embedded CSE programme of the ARCHER2 UK National Supercomputing Service (http://www.archer2.ac.uk).

References

1. The United Kingdom Chemistry and Aerosol model: https://www.ukca.ac.uk/
2. The Met Office Unified Model: https://www.metoffice.gov.uk/research/approach/modelling-systems/unified-model/index
3. Archibald, A. T., et al.: Description and evaluation of the UKCA stratosphere–troposphere chemistry scheme (StratTrop vn 1.0) implemented in UKESM1, Geosci. Model Dev., 13, 1223–1266, https://doi.org/10.5194/gmd-13-1223-2020, 2020.
4. Mann, G. W., et al.: Description and evaluation of GLOMAP-mode: a modal global aerosol microphysics model for the UKCA composition-climate model, Geosci. Model Dev., 3, 519–551, https://doi.org/10.5194/gmd-3-519-2010, 2010.
5. Archer-Nicholls, S., et al.: The Common Representative Intermediates Mechanism Version 2 in the United Kingdom Chemistry and Aerosols Model, Journal of Advances in Modeling Earth Systems, 13, e2020MS002420, https://doi.org/10.1029/2020MS002420
6. Weber, J., et al.: Improvements to the representation of BVOC chemistry–climate interactions in UKCA (v11.5) with the CRI-Strat 2 mechanism: incorporation and evaluation, Geosci. Model Dev., 14, 5239–5268 (2021). https://doi.org/10.5194/gmd-14-5239-2021
7. Telford, P. J., et al.: Implementation of the Fast-JX Photolysis scheme (v6.4) into the UKCA component of the MetUM chemistry-climate model (v7.3), Geosci. Model Dev., 6, 161–177, https://doi.org/10.5194/gmd-6-161-2013, 2013.
8. The Community Intercomparison Suite: http://www.cistools.net/
9. cf-python: An Earth Science data analysis library that is built on a complete implementation of the CF data model. https://ncas-cms.github.io/cf-python/
10. Rose: a framework for managing and running meteorological suites. https://github.com/metomi/rose
11. The Cylc general purpose workflow engine. https://github.com/cylc/cylc-flow

12. Iris: A powerful, format-agnostic, community-driven Python package for analysing and visualising Earth science data. https://scitools-iris.readthedocs.io/en/stable/