How to output UM data on predetermined flight tracks using ISO_simulator

These instructions explain how to produce UM model output on a set of predetermined flight tracks to make it easier and quicker to compare model data to observations. This is done by interpolating gridded, hourly (or higher frequency) model data in space and time onto observations coordinates.

ISO_simulator can be run on existing model output (postprocessing) or it can be embedded into a UM rose suite. The step by step instructions work for both cases. To run ISO_simulator in postprocessing mode you can omit point 3 below:

1. Ensure the correct python environment is available. Instructions for Monsoon are different to instructions for other platforms, including Archer2 and JASMIN (see 1A and 1B).
2. Produce flight track input files in the appropriate netcdf format
3. Modify a UM suite to include a new app which calls ISO_simulator
4. Select appropriate input options for the interpolation code
5. Ensure the UM has the appropriate model output in the required format to be read by the interpolation code

1. A) <u>Installing a Python environment (on JASMIN, Archer2, local clusters)</u>

The interpolation code requires general python packages plus CIS, Iris and cf-python. To install a suitableconda environment follow the instructions below:

a) From the home directory, install a *vision* conda environment by typing these lines

```
conda create --name vision --file vision_environment.txt
```

The vision_environment.txt file is provided here and it ensures that package dependencies are not disrupted.

b) If you are running in postprocessing you will need to activate the conda environment:

```
conda activate vision
```

1. B) <u>Python environment (on Monsoon)</u>

There are currently some issues with installing python directly on Monsoon, this is because Monsoon has some old C libraries and more recent versions of python will not work (this should be resolved once moving to Monsoon2).
Therefore, in order to access the right python environment, you will have to make a copy of the current python installation (see instruction below).

a) Make a copy of `/home/d05/marus/miniconda3` and place it in your home directory
b) Modify your .bash_profile (or .bashrc etc.) by adding the two lines below to define the python path:

```
export PATH=~/miniconda3/bin:$PATH
export UDUNITS2_XML_PATH=~/miniconda3/share/udunits/udunits2.xml
```

2. Produce flight track input files in the appropriate netcdf format

Input files containing the flight track information are currently read in using cis; this requires the file to be in a CF-compliant, NetCDF format.

The filename is used to extract information on the date for which the data is valid. Therefore files must comply to the following requirements:

- Data must be stored in daily files
- The date needs to be added to the filename in the form YYYYMMDD
- The date cannot be at the beginning of the filename and needs to be preceded by an underscore '_': for example flight_data20010101.nc (will work but can occasionally crash); flight_data_20010101.nc (OK)
- There should be no digits in the filename before the date (you can add digits after): for example CVAO_O3_20070101.nc (not OK); CVAO_20070101_O3.nc (OK)

The following variables defining time and location are required: 'time', 'latitude', 'longitude' 'air_pressure' and/or 'altitude' and the variable names must be as specified (standard cf-compliant names). The flight input file can also contain additional fields for convenience (any observed variable of interest, e.g. ozone, CO, aerosol, etc.): these extra fields are ignored by the interpolation code, but it might be useful to have them in the files for comparison with model data after interpolation.

3. Modify a UM suite to include the interpolation code

For a list of changes required to embed the interpolation code into the UM at runtime, you can compare the following suites that show before/after embedding the simulator code: u-cn535 and u-cn586

Specifically, check relevant differences in:

a) `app/fcm_make_pp/rose-app.conf` (this change is necessary to archive the resulting NetCDF files)
b) a new directory is added, `app/flight_track_sim`. This contains `rose_app.conf` and `bin/ISO_simulator.py`. Please use the latest versions of the above, which is provided with this document.
   `rose_app.conf` contains input variables and a command line argument to invoke `ISO_simulator.py`.
c) `site/monsoon.rc` or `site/archer2.rc` (these changes are to define resources for the flight_track code)
d) `suite.rc` (these changes define the new UM task and point to the python libraries)

4. Select appropriate input options for the interpolation code

A list of input variables required to run `ISO_simulator.py`, their description and usage is shown in the table below. 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, e.g. on existing model data, (if 'postprocessing' is selected). The postprocessing mode is useful to test the code and observational input files before

running within a UM job. 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.
When running within a UM suite, the input variables and command line arguments can be accessed/modified through `app/flight_track_sim/rose_app.conf` and/or through the 'flight_track_sim' menu in the rosie job gui.

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, 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)
-v --vertical_coord	*choose coordinate for vertical interpolation: 'air_pressure' or 'altitude'; (optional; default=altitude)*
-e --extra_file	*Filename (including full path) of model orography file'; (optional, only required if vertical_coord=altitude; default=Directory_ft/orography.pp)*
-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 space separated stashcodes (optional; default = process all)

5. Ensure the UM has the appropriate model output in the required format to be read by the interpolation code

The interpolation code can use either 'air_pressure' or 'altitude' as the vertical coordinate. If this is not specified it will use altitude by default.

For air_pressure interpolation

Required model variables for comparison with flight data need to be output on selected pressure levels. You can specify the pressure levels required depending on the aircraft data used and which areas in the atmosphere it samples the most.
Since the UM has a hybrid sigma-height vertical coordinate system, we also need to output the Heaviside function on pressure levels to account for model missing data where a pressure level near the surface falls below the orography for that gridbox. The Heaviside functions associated to the variables of interest (the UM has different Heaviside functions for different groups of variables) should be output in the same file and at the same time-resolution as the variables we want to interpolate. For an example of setting up the right model output go to 'STASH Request' section for u-cu445 and filter for 'UPO' in usage name.

For altitude interpolation

Required model variables are output on the model hybrid sigma-height levels (or theta levels). A subset of these levels can be defined in the UM stash section (e.g. removing levels in the stratosphere or outside our vertical area of interest) to reduce the size of the output (if required). When interpolating in altitude, the interpolation code will require the model orography field to convert model levels to altitude. The name and path of the orography file can be defined using the -e input variable. If this is not specified, the code will look for a file named 'orography.pp' (if running in postprocessing) or 'orography' (if embedded within a UM suite) located in the same directory as the observational input files. The difference between the two files is that the former is in UM pp format and the latter is in UM fieldsfile (ff) format. It is necessary that the orography file is in the same format as the model files (UM pp when postprocessing after retrieving from MASS and UM ff when the code is embedded within a UM suite).

All model variables should be output as hourly (or higher frequency) values and written to daily files on a single output stream. This is because all model data required for interpolation is read from a single file. Therefore, users need to set up the right 'domain', 'time' and 'usage' profiles, output stream and file re-initialisation for the model output containing the fields to be interpolated. For an example of setting up the right model output go to 'STASH Request' section for u-cu445 and filter for 'UPP' in usage name.