Interpolation Test Changes to AIMS

The following is a list of changes and additions to the AIMS interpolation capabilities with instructions on how and in what order to run the new scripts. In cases where changes were made to existing scripts to extend functionality, care was taken to ensure that the original functionality has been preserved and thus a user unfamiliar to the changes made should be able to execute scripts as before with the same outcome as before.

Frequency plotting

Frequencies within a track (for radial-only model grids) can be plotted to see if the model frequencies appear as they should. This can be achieved by executing:

```
$ python plot_track_freqs.py grid_file n
```

from the command line with `grid_file` being the binary grid file to extract modes from and `n` being an optional integer argument for the number of samples. The plots generated show the dimensionless scaled frequencies against age for a particular evolutionary track whose grid parameters are printed as the plot title. Each series represents a particular value of radial order. The tracks to plot are chosen based on their variance and come in `n` pairs so there will be `2n` plots generated for the `n` highest and `n` lowest variances. If a value of `n` is not given, the default plots are just the tracks with highest and lowest variance. Variance of a track is calculated as the variance of frequencies of a particular radial order summed over all available orders.

Frequency plotting incorporated into interpolation test plotting

The frequency plotter has been incorporated into the interpolation test plotter as an optional command line argument. This means that the plotter should now be executed via:

```
$ python plt_interp_MS.py interp_file grid_file
```

where the `interp_file` and `grid_file` are the binary files for the interpolation test results generated through AIMS the usual way and the grid that the interpolation test was applied to respectively. The grid file is an optional argument that extends the functionality of `plt_interp_MS` to also plot the frequencies against age of the highest and lowest variance tracks. Omitting the optional argument will not execute the frequency plotter and recovers the original functionality of the script.

Interpolation test plotting for more than two parameters

The interpolation test plotter has been extended to allow presentation of interpolation test data for more than two parameters. Plotting can be achieved via the command line call:

```
$ python plot_multi_interp interp_file
```

where the `interp_file` is the binary file with interpolation test results. This script extracts the user defined parameters that define a track and allows the user to input a selection of two parameters for the axes of the resulting plots. The script has been made with the possible future extension into more than 3 parameters in mind but at the moment can only be used to plot 3. The third parameter is kept constant and the `plt_interp_MS.py` script is imported and used to plot the results for every value of the third parameter separately. If the number of different values of the third parameter exceeds 10, the user is alerted with the total number of plots that will be generated (preventing the user from accidentally generating hundreds of plots) and asked to input a value for how many plots they actually want generated. This number of different values is then selected at random from those available.

Testing AIMS by providing a randomly chosen grid model as input

The AIMS program can be tested by running a series of scripts that extract a random model from the grid file, perturb its frequencies and constraints, and generate an input file for AIMS.

LEGACY files

When a random model is chosen from the model grid, its frequencies are perturbed by sampling a Gaussian distribution with a mean at the original grid value and a standard deviation representative of likely 'real-world' uncertainties. For main sequence grids, these uncertainties are taken from those found in the LEGACY data, which has been saved in the `LEGACY_mode_list` file. Individual uncertainties have been extracted using `legacy_uncertainty_grid_extractor.py` and then saved in the `legacy_grid` binary file. This file is then used when selecting appropriate uncertainties for the randomly selected model.

Testing randomly selected model using AIMS

Executing the `random_model_extractor.py` script will randomly select a model from the grid file found in `AIMS_configure.py` rather than accepting a grid file as a command line argument. This is to prevent AIMS (which selects the grid file from `AIMS_configure.py`) from using a different grid to the one the model was extracted from. The user is prompted to select whether the grid is for a main sequence or red giant population as different uncertainties are applied in the two different cases. The user is then asked how many model input files they would like to generate. The first model, labelled 'Model_0', contains unperturbed frequency and constraint values and the all further models are perturbed by the aforementioned Gaussian distribution. 'Model_params' contains the main model parameters that AIMS should recover. AIMS can be run automatically for each input file by running `multi_model.py`. The results can be viewed as usual by navigating to the relevant model directory in the results directory that is created by AIMS but a comparison between the values of model parameters given by AIMS and the true values found in 'Model_params' can be more directly viewed by running `plot_random_test.py`. The user should ensure prior to running the extractor script and any subsequent scripts that rely on it that any model input files from previous runs have been removed from the main directory. This is because `multi_model.py` will attempt to run AIMS on every file labelled 'Model_x' in the main directory.

Concise order of business:

- If `legacy_grid` is not present in main directory, run:

```
$ python legacy_uncertainty_grid_extractor.py
```

- Ensure the desired grid file is assigned in `AIMS_configure.py` and run the following list of commands each in order after the previous has finished:

```
$ python random_model_extractor.py
$ python multi_model.py
$ python plot_random_test.py
```