AIMS Version Update (14/11/2018)

Remaining updates:

- Make selection of evolutionary state generic
- Provide specific entry criteria for this to remove duplicate functions in model.py

AIMS_configure.py:

Line 36 - number of steps to be added if MCMC process does not converge.

```
35 nsteps      = 200    # number of steps
36 add_steps   = 500    # number of steps to add if convergence isn't achieved
37 thin        = 10     # thinning parameter (1 out of thin steps will be kept ...)
```

Line 198 - detemines which evolutionary parameter is to be used.

```
196 ########################   Interpolation    ###############################
197 scale_age = True                  # use a scaled age when interpolating
198 interp_type = "Age"               # options to use either "age" or "mHe" for interpolation. Should only be
199                          # changed if using a grid with mHe values > 0
```

AIMS.py:

Line 64: now import ptemcee to use instead of emcee.PTsampler

```
64 import ptemcee
```

Lines 1786-1800, *load_binary_data* - allows the user to extract information about a specific model based on mass, age and log(Z) value.

```
1786    # for track in grid.tracks:
1787    #     for model in track.models:
1788    #         # print model.glb
1789    #         # sys.exit()
1790    #         # print(model.glb[1]/constants.solar_mass)
1791    #         if (model.glb[1] == 2.5253600000000000E+033) & (model.glb[3] == 0.01) & (model.glb[0] == 4098.5500000000002):
1792    #             print model.glb[1]/constants.solar_mass
1793    #             print model.glb
1794    #             print model.get_freq()
1795    #             model.write_file_simple('test')
1796    #             print model.find_large_separation()
1797    #             print model.numax
1798    #             print model.FeH
1799    #             model.print_me()
1800    # sys.exit()
```

Lines 1930-44 (*find_best_model_in_track*): edit to the calculation of the log probability to find the best model along a track. Edit made by Gael therefore for further clarification, please ask him.

```
    for model in grid.tracks[ntrack].models:
        result, rc, rs, rp = prob.evaluate(model)
        reject_classic += rc
        reject_seismic += rs
        reject_prior   += rp
        if (config.interp_type == "mHe"):
            istart = max(i-1,0)
            istop  = min(i+1,nmodels-1)
            log_slope = math.log(abs((grid.tracks[ntrack].models[istop].string_to_param("Age")   \
                                    - grid.tracks[ntrack].models[istart].string_to_param("Age")) \
                                    /(grid.tracks[ntrack].models[istop].string_to_param("mHe")   \
                                    - grid.tracks[ntrack].models[istart].string_to_param("mHe"))))
        else:
            log_slope = 0.0
        result += log_slope
```

Run_emcee: have replaced emcee.PTsampler with the purpose specific package ptsampler. Have also included convergence criteria and looping based on whether the threshold set for the Gelman-Rubin statistic is satisfied.

```
if (config.PT):
    ''' v1 of emcee sampler '''
#     print("Number of temp.:       "+str(config.ntemps))
#     sampler = emcee.PTSampler(config.ntemps, config.nwalkers, ndims, prob.likelihood, prob.priors, pool=pool)
#
#     # initial burn-in:
#     for p, lnprob, lnlike in tqdm(sampler.sample(p0, iterations = config.nsteps0),total=config.nsteps0): pass
#
#     # production run:
#     sampler.reset()
#     for p, lnprob, lnlike in tqdm(sampler.sample(p, lnprob0 = lnprob, lnlike0 = lnlike, iterations = config.nsteps),total=config.nsteps):
#         pass
#
# else:
#     sampler = emcee.EnsembleSampler(config.nwalkers, ndims, prob, pool=pool)
#
#     # initial burn-in:
#     p, new_prob, state = sampler.run_mcmc(p0,config.nsteps0)
#
#     # production run:
#     sampler.reset()
#     p, new_prob, state = sampler.run_mcmc(p,config.nsteps)
#
# # Print acceptance fraction
# print("Mean acceptance fraction: {0:.5f}".format(np.mean(sampler.acceptance_fraction)))
# # Estimate the integrated autocorrelation time for the time series in each parameter.
# try:
#     autocorr_time =  sampler.get_autocorr_time(c=1.0)
#     print("Autocorrelation time: "+str(autocorr_time))
# except emcee.autocorr.AutocorrError:
#     print("Autocorrelation time not available")
    ''' v2 of emcee sampler '''
    print "Number of temp.:       ", config.ntemps
    sampler = ptemcee.Sampler(config.nwalkers, ndims, prob.likelihood, prob.priors, ntemps=config.ntemps, threads=config.nprocesses)
    # initial burn-in:
```

Lines 2174-78 (*find_a_blob*): now provides options depending on the evolutionary parameter used.

```
2174    if config.interp_type == "Age":
2175        my_model = model.interpolate_model(grid,params[0:ndims-nsurf],grid.tessellation,grid.ndx)
2176    elif config.interp_type == "mHe":
2177        my_model, slope = model.interpolate_model_mHe(grid,params[0:ndims-nsurf],grid.tessellation,grid.ndx)
2178    return utilities.my_map(my_model.string_to_param,config.output_params)
```

Lines 2280-307 (*write_percentiles*): includes additional columns with the sigma values computed. Lenght of percentiles array increased accordingly.

```
# initialisation
(m,n) = np.shape(samples)
percentiles = np.empty((n,7), dtype=np.float64)
for i in range(n):
    percentiles[i,0] = np.percentile(samples[:,i],100.0*(0.5-two_sigma/2.0))
    percentiles[i,4] = np.percentile(samples[:,i],100.0*(0.5+two_sigma/2.0))
    percentiles[i,1] = np.percentile(samples[:,i],100.0*(0.5-one_sigma/2.0))
    percentiles[i,3] = np.percentile(samples[:,i],100.0*(0.5+one_sigma/2.0))
    percentiles[i,2] = np.percentile(samples[:,i],50.0)
    percentiles[i,5] = abs((percentiles[i,0]-percentiles[i,4])/2.)
    percentiles[i,6] = abs((percentiles[i,1]-percentiles[i,3])/2.)

# write results to file
output_file = open(filename,"w")
output_file.write("Percentiles\n===========\n\n");
output_file.write('{0:25} '.format("Quantity"))
output_file.write('{0:25} '.format("-2sigma"))
output_file.write('{0:25} '.format("-1sigma"))
output_file.write('{0:25} '.format("Median"))
output_file.write('{0:25} '.format("+1sigma"))
output_file.write('{0:25}\n'.format("+2sigma"))
output_file.write('{0:25} '.format("1sig"))
output_file.write('{0:25}\n'.format("2sig"))

for i in range(n):
    output_file.write('{0:25} '.format(labels[i]))
    for j in range(7):
        output_file.write('{0:25.15e} '.format(percentiles[i,j]))
    output_file.write('\n')
output_file.close()
```

Lines 2505-12 (*write_combinations*): Addition of other evolutionary parameter to perform model interpolation with.

```
2505    output_file.write("{0:s} {1:e}\n".format(grid.prefix,constants.G))
2506    for params in samples:
2507        results = model.find_combination(grid,params[0:ndims-nsurf])
2508        if config.interp_type == "Age":
2509            my_model = model.interpolate_model(grid,params[0:ndims-nsurf],grid.tessellation,grid.ndx)
2510        if config.interp_type == "mHe":
2511            my_model, slope = model.interpolate_model_mHe(grid,params[0:ndims-nsurf],grid.tessellation,grid.ndx)
2512
```

Line 3342: changed “Age” to config.interp_type to access the relevant evolutionary parameter functions.

```
3339    # load grid and associated quantities
3340    grid = load_binary_data(config.binary_grid)
3341    plot_frequencies(grid) # this will stop the program
3342    grid_params_MCMC = grid.grid_params + (config.interp_type,)
3343    grid_params_MCMC_with_surf = grid_params_MCMC \
```

Lines 3464-67: functions selected based on the evolutionary parameter chosen to find best MCMC model.

```
3459    # find best MCMC model
3460    ndx_max = lnprob.argmax()
3461    best_MCMC_result = lnprob[ndx_max,0]
3462    best_MCMC_params = samples[ndx_max,1:]
3463    best_MCMC_params.reshape(ndims)
3464    if config.interp_type == "Age":
3465        best_MCMC_model  = model.interpolate_model(grid,best_MCMC_params[0:ndims-nsurf],grid.tessellation,grid.ndx)
3466    elif config.interp_type == "mHe":
3467        best_MCMC_model, slope  = model.interpolate_model_mHe(grid,best_MCMC_params[0:ndims-nsurf],grid.tessellation,grid.ndx)
```

Lines 3479-82: functions selected based on the evolutionary parameter chosen to find best statistical model.

```
3475     # find statistical model
3476     statistical_params = samples[:,1:].sum(axis=0, dtype=np.float64)/(1.0*config.nwalkers*config.nsteps)
3477     statistical_params.reshape(ndims)
3478     statistical_result = prob(statistical_params)
3479     if config.interp_type == "Age":
3480         statistical_model  = model.interpolate_model(grid,statistical_params[0:ndims-nsurf],grid.tessellation,grid.ndx)
3481     elif config.interp_type == "mHe":
3482         statistical_model, slope  = model.interpolate_model_mHe(grid,statistical_params[0:ndims-nsurf],grid.tessellation,grid.ndx)
```

model.py:

- Many of the changes in this section are replications of current functions for when age is used as the evolutionary parameter. In these cases, just the name of the function is included as the core code remains the same, only age is changed for mHe

Series of changes to global parameter numbers. These all depend upon whether the core helium mass is being included in as a grid parameter. grid_SUN_DIFF2 does not include this, hence the '-1' included. When He-core mass is included, the '-1' disappears. imHe is included as a global parameter for all other grids used in the AIMS paper.

```
94 nglb          = 9 + len(config.user_params) - 1
95 """ total number of global quantities in a model (see :py:data:`Model.glb`)."""
```

```
124 # imHe          = 5
125 """ index of the parameter corresponding to He core mass in the :py:data:`Model.glb` array """
126
127 ifreq_ref     = 6 + len(config.user_params) - 1
128 """
129 index of the parameter corresponding to the reference frequency
130 (used to non-dimensionalise the pulsation frequencies of the model)
131 in the :py:data:`Model.glb` array
132 """
133
134 iradius       = 7 + len(config.user_params) - 1
135 """ index of the parameter corresponding to radius in the :py:data:`Model.glb` array """
136
137 iluminosity   = 8 + len(config.user_params) - 1
138 """ index of the parameter corresponding to luminosity in the :py:data:`Model.glb` array """
```

string_to_latexl: Inclusion of mHe.

```
166     if (string == "mHe"):        return r'Mass of He core, $%sM_{He-core}/M_{\odot}%s$'%(prefix,postfix)
```

Model:

string_to_param: Inclusion of mHe.

```
237         if (string == "mHe"):        return self.glb[imHe]/constants.solar_mass
```

Lines 284-5: Assertion of mHe to be non-zero and positive if used as the evolutionary parameter. Value can be zero if an MS grid is used that includes the He-core mass.

```
284        if config.interp_type == "mHe":
285            assert (_glb[imHe] >= 0.0),          "A star cannot have a negative He core mass! M = %f, %s"%(_glb[imHe],self.name)
```

Lines 511-20: function to return core helium mass.

```
511     def get_mHe(self):
```

Line 897 (*print_me*): Allows for direct inclusion of core helium mass in model properties printed to the screen.

```
897         # print "Mass He core (in M_Sun):      %.4f" % self.glb[imHe]
```

Track:

Line 1214: Beginning of the core helium mass functions in Track class. What follows are the function names unless there are any major changes to the included code.

```
1214        ''' Mass of He core versions '''

1215    def sort_mHe(self):

1220    def is_sorted_mHe(self):

1231    def duplicate_mHe(self):

1253    def interpolate_model_mHe(self, mHe):

1291    def find_combination_mHe(self, mHe, coef):

1326    def find_modes_mHe(self, ntarget, ltarget):

1353    def test_interpolation_mHe(self, nincr):
```

Model_grid:

Lines 1494-1496 (*read_model_list*): hard coded inclusion of imHe to global parameter list and extension of the array length for the addition of any other parameters.

```
1494            # glb[imHe]          = utilities.to_float(columns[8])
1495
1496            i = 9 - 1
```

Lines 1527-32: Sort tracks depending upon the evolutionary parameter selected.

```
1527        # sort tracks:
1528        for track in self.tracks: track.sort()
1529        if config.interp_type == 'Age':
1530            for track in self.tracks: track.sort_age()
1531        elif config.interp_type == 'mHe':
1532            for track in self.tracks: track.sort_mHe()
```

- *Test_interpolation* (line 1668) contains multiple changes to allow original model and interpolated model properties to be saved out. These are currently commented out in the code, but can be reinstated accordingly.

```
1685        # initialisation
1686        results = []
1687        ndim = self.ndim+1
1688        # print ndim
1689        # output_folder = '/home/buldgen/AIMS-master_New/AIMS_BEN/'
1690        # filename = os.path.join(output_folder,"combinations_Delaunay.txt")
1691        # f2 = os.path.join(output_folder,"Delaunay_MS_Models_mHe.txt")
1692        # f3 = os.path.join(output_folder,"Delaunay_MS_Mod_vals_mHe.txt")
1693        # output_file = open(filename,"w")
1694        # out2 = open(f2,"w")
1695        # out3 = open(f3,"w")
1696
```

```
1702            for i in range(nmodels):
1703                aModel1 = self.tracks[j].models[i]
1704                if config.interp_type == "Age":
1705                    pt[-1] = aModel1.glb[iage]
1706                    aModel2 = interpolate_model(self,pt,tessellation,ndx2) #,aModel1.name,aModel1,out2,out3)
1707                elif config.interp_type == "mHe":
1708                    pt[-1] = aModel1.glb[imHe]
1709                    aModel2, slope = interpolate_model_mHe(self,pt,tessellation,ndx2)
1710                aResult[i,0:ndim] = pt
1711
```

```
1720                if (res is None): continue  # filter out combinations outside the grid
1721                # output_file.write("{0:d} {1:.2f} {2:.3f} {3:.3f} {4:.4f} {5:.3f} {6:.3f} {7:.2f}\n".format( \
1722                #                     len(res), aModel1.glb[imass]/constants.solar_mass, aModel1.glb[iradius]/constants.solar_radius, \
1723                #                     aModel1.glb[iluminosity]/constants.solar_luminosity,aModel1.glb[iz0], \
1724                #                     aModel1.glb[ix0],aModel1.glb[iage], \
1725                #                     aModel1.glb[itemperature], aModel1.glb[imHe]/constants.solar_mass))
1726                #
1727                # output_file.write("{0:d} {1:.2f} {2:.3f} {3:.3f} {4:.4f} {5:.3f} {6:.3f} {7:.2f}\n".format( \
1728                #                     len(res), aModel2.glb[imass]/constants.solar_mass, aModel2.glb[iradius]/constants.solar_radius, \
1729                #                     aModel2.glb[iluminosity]/constants.solar_luminosity,aModel2.glb[iz0], \
1730                #                     aModel2.glb[ix0],aModel2.glb[iage], \
1731                #                     aModel2.glb[itemperature], aModel1.glb[imHe]/constants.solar_mass))
1732                # for (coef,model_name) in res:
1733                #     output_file.write("{0:.15f} {1:s}\n".format(coef, model_name))
1734                # output_file.write("\n")
1735            results.append(aResult)
1736
1737        # output_file.close()
1738        # out2.close()
1739        # out3.close()
1740
```

(sorry it kept getting smaller)

Final interpolation copy-cats for mHe:

```python
def find_mHes(coefs, tracks, mHe):
    """
    Find mHes to which each track needs to be interpolated for a specified
    mHe.  Follows the same structure as find_ages() and uses the same scaled and
    fixed parameter for determining the method used to calculate the values.
    """
```

```python
def interpolate_model_mHe(grid,pt,tessellation,ndx):
    """
    Interpolate model in grid using provided parameters.
```

```python
def find_combination_mHe(grid,pt):
    """
    Find linear combination of models which corresponds to interpolating
    the model based on the provided parameters.
```