README

This document is intended to guide the interested reader through the exact sequence of calculations used in preparation of the journal manuscript “A measurement of the Kuiper Belt’s midplane from objects classified by machine learning,” by Ian Matheson and Renu Malhotra, submitted to *Astronomical Journal*. In theory, the journal paper itself should be sufficient to let any interested researcher reproduce the work. Today’s expectations for data sharing and code sharing make it desirable to provide supplemental information for reproducibility as attachments on Arxiv or a repository on GitHub. Any researcher who downloads this GitHub repository should be able to reproduce the results in AJ nearly exactly, allowing for some small random variation due to the use of different random number seeds or small changes in the underlying software.

SOFTWARE

This work is done on a 2018 Macbook Pro running MacOS Monterey 12.0.1. Cluster computations are done on the University of Arizona Puma cluster, which uses Slurm as its batch submission scripting language. Both local and cluster calculations are done partly in Fortran 95 using the **gfortran** compiler and partly in Python 3.8.2. Python packages include **Astroquery, rebound, scikit-learn, urllib, shapely,** and **json**. A small amount of code is written in MATLAB. Someday I should take the time to translate it to Python, but I (Matheson) haven’t yet done so.

STEP-BY-STEP INSTRUCTIONS FOR REPRODUCING CALCULATIONS

First, go to the JPL Solar System Dynamics Small Body Database query at https://ssd.jpl.nasa.gov/tools/sbdb_query.html. Expand the “Custom Object/Orbit Constraints” menu. Select “Define New Custom Constraint,” then “Orbit and Model Parameter Fields.” In the “Select field orbit” menu, choose “e”, then “<”, and finally enter “1” in the field. This is a constraint $e < 1$. Repeat for constraints $a > 34$, $a \leq 150$, and $q > 30.07$. These elements are heliocentric. Later the selection will be more refined through Python scripting. Expand the menu “Output Selection Controls,” click “select all”, then “add.” Not all available fields will be used later, but it is easier to select all and then remove unneeded fields through Python scripting than to tediously select individual fields. In the "Output Fields" menu box, click "Full Precision." Then click the green "Get Results" button. Click "Download (CSV-format)". In a new working directory, save the csv as **sbdb_query_results.csv**. This is the raw list of KBOs that will be the base of all later computation. Skip this step if you plan to use the existing file in the GitHub repository.

Next, go to the Minor Planet Center database at https://www.minorplanetcenter.net/data. Right-click to save **MPCORB.DAT** in a convenient directory, as **MPCORB_YYYYMMDD.DAT**. This will later be used to cross-reference against the Small Body database to eliminate some objects from consideration. Skip this step if you plan to use the existing file in the GitHub repository, but still save the MPCORB_20211202 files from the GitHub repository in a separate directory.

Next, I need to retrieve barycentric elements for the planets at a common epoch. This is done by running **horizons_save_planets.py**. In the file as archived, this epoch is 2022-01-22 00:00:00, corresponding to the epoch in the archived **sbdb_query_results.csv**. If you wish to repeat the calculations using an updated data set instead of the archived SBDB query and MPCORB, you may wish to consult the SBDB query results to see what epoch they use, then find the corresponding Julian date using any convenient online Julian data calculator (I use https://www.aavso.org/jd-calculator) and substitute that on line 11 (or thereabouts) in the Python script.

Having retrieved the appropriate planetary elements, I compute the Laplace plane in MATLAB by running **laplace_plane_driver.m**. This will create several files that contain the Laplace plane for different semimajor axis bins.

After computing the Laplace plane, I run **mpcorb_dat2csv.py** to convert the MPC catalog from an unwieldy DAT format to a more convenient CSV format.

Next, **sbdb_reduce.py** is run to trim the MPC catalog to a more manageable size, retrieve barycentric elements for KBOs, trim the KBO catalog by barycentric semimajor axis and semimajor axis uncertainty, remove objects with fewer than three observed oppositions by cross-referencing with the MPC catalog, retrieve JSON files with a covariance matrix for each remaining object from the Small Body Database API, save the barycentric elements and covariance matrix to **sbdb_query_results_addcov.csv**, and finally generate 300 clones for each object and save the clones to file.

Next, **sbdb_classify_clones.py** is run through line 235 (approx.) to generate **sim_template_20220122.bin**, which is used as the template for all Rebound integrations performed by the machine learning classifier. Lines 138-235 (approx.) are then commented out, as in the archived code.

Having generated the simulation template for Rebound, we need to classify each KBO and its clones. I generate 275 Python files for doing so on the Puma cluster by running **sbdb_generate_classify_clones.py**, which makes copies of **sbdb_classify_clones.py** differing only in a single number so that Puma can run them individually. This is because there are 2748 sets of clones to classify, and I can run up to 300 jobs on the cluster.

Next, I upload the entire folder to the cluster and submit the 275 jobs as a batch using the command **sbatch slurm_classify_clones.slurm**, which runs the batch submission script **slurm_classify_clones.slurm**. After the batch job has completed, I delete the local folder from my computer and download the entire folder, now containing cluster results, from the storage space on the cluster.

To read and count the resulting classifications of the clones, I then run locally on my computer **sbdb_read_clone_classifications.py**. This takes the **.out** files from the cluster standard output, reads the classification for each clone, and appends counts of how many clones were put in each category to the existing SBDB query spreadsheet, which is then saved as **sbdb_query_results_addcloneclass.csv**.

As previously stated, it was easier to save all possible SBDB output to **sbdb_query_results.csv** and drop the unneeded information in a Python script than to tediously select which outputs to keep in the browser window. I now drop unneeded information by running **sbdb_delcols.py** in preparation for later Fortran computations. This produces **sbdb_query_results_delcols.csv**.

Although I need the cluster to run many repetitions of synthetic samples and find their midplanes, I do not need it to find the midplanes of the observed samples. I compute the midplanes of the observed samples locally by running **fortran_mean_plane_nominal.f95** using **gfortran -fortran_mean_plane_nominal.f95 -o test** and then **./test**.

I then run **sbdb_generate_fortran.py** to generate 300 separate Fortran files to run on the cluster.

I delete the existing folder from the cluster and reupload the folder as it now exists locally, with the results of all previous computations and the Fortran files to run. I then use **sbatch slurm_fortran_mean_planes.slurm** to run the batch submission script **slurm_fortran_mean_planes.slurm**. When the batch job has completed, I delete the local folder and download the folder on the cluster, now containing the 40,200 synthetic midplanes for each semimajor axis bin.

It is possible that some of the real objects may prove difficult to make a synthetic match for, so I check whether any object has not been matched within the maximum iteration count of 10^7 by running **sbdb_mean_plane_consolidate.py**. Since no objects failed to match when preparing the manuscript, I did not prepare any code to check how much that would affect the computed mean planes. This script also consolidates the outputs of the 300 cluster jobs into separate files for each semimajor axis bin containing all 40,200 synthetic midplanes for that bin.

The plots in the manuscript showing the inclination and node of the mean plane by semimajor axis bin, as well as the q, p covariance ellipses for each semimajor axis bin, are generated by running **mean_plane_plots.py**.

The plot in the manuscript showing convergence of the inclination lower bound for a semimajor axis bin over 40,200 repetitions, as well as similar plots for the inclination upper bound and for the longitude of node upper and lower bounds, are generated by running **show_convergence.py**. Note that the plots for the node lower bound cannot always be generated as written, because for some bins the lower bound is zero and that is a singularity of the fractional error format of the plot. However, when the lower bound is zero the upper bound is 360 degrees, so in those cases the upper bound plot can be examined for convergence.

To produce the table of classified objects included in the manuscript and electronically from *AJ*, open **sbdb_query_results_delcols.csv** in Excel and delete the unnecessary columns, saving the remainder as **table_for_paper.xlsx**.

In summary, the sequence of actions to take is:

1. Save **sbdb_query_results.csv**.
2. Save **MPCORB.DAT**.
3. Run **horizons_save_planets.py**.
4. Run **laplace_plane_driver.m**.
5. Run **mpcorb_dat2csv.py**.
6. Run **sbdb_reduce.py**.
7. Run lines 1-235 of **sbdb_classify_clones.py**, then comment out lines 138-235.
8. Run **sbdb_generate_classify_clones.py**.
9. Upload local folder to cluster.
10. On cluster, run **sbatch slurm_classify_clones.slurm**.
11. Delete local folder and download folder from cluster.
12. Run **sbdb_read_clone_classifications.py**.
13. Run **sbdb_delcols.py**.
14. Run **fortran_mean_plane_nominal.f95**.
15. Run **sbdb_generate_fortran.py**.
16. Delete cluster folder and upload local folder to cluster.
17. On cluster, run **sbatch slurm_fortran_mean_planes.slurm**.
18. Delete local folder and download folder from cluster.
19. Run **sbdb_mean_plane_consolidate.py**.
20. Run **mean_plane_plots.py**.
21. Run **show_convergence.py**.
22. Delete unnecessary columns from **sbdb_query_results_delcols.csv**.

I'm afraid this isn't a well-oiled one-click checklist to reproduce the work in the journal manuscript, but I haven't the training in software development to boil it down to one click.

Before running any of these scripts on your local machine, make sure to replace all references to **/Users/iggymatheson/Documents_off_iCloud/mm22_v3** to whatever local directory you're working in (call it **mm22_v3** for convenience). Also replace references to **/Users/iggymatheson/Documents_off_iCloud/mm22 mpcorb databases** with a separate directory in which to save the **MPCORB_20211202.DAT** and **MPCORB_202211202.csv** files, because they are individually too large to upload to the Puma cluster if you want to keep everything in one folder and just upload and download the entire folder for convenience.

HOW TO RE-RUN THESE CALCULATIONS WITH NEW DATA

If you want to rerun the calculations with new downloads from the Small Body Database and the Minor Planet Center instead of using the SBDB and MPC databases in the GitHub repository, make sure to save the MPC database as **MPCORB_YYYYMMDD.DAT** and replace all references in all Python scripts to **MPCORB_20211202** with **MPCORB_YYYYMMDD**. If the Small Body Database results use a different epoch than 2022 January 22, replace all Python, Fortran, and MATLAB references to **20220122** or **20210121** with **YYYYMMDD**, whatever the epoch of the SBDB download may be. Also, make sure to replace all Python references to **2022-01-21 00:00:00** with the appropriate epoch.

The position of the Laplace plane may shift slightly when you run **laplace_plane_driver.m** with planetary elements from a different epoch, so replace all Python and Fortran references to **34.79** and **40.524** with whatever the appropriate bin boundaries may be as seen in the output files from MATLAB – for example, **a_p0q0_i0Omega0_34.79_40.524_20220122.txt**.)

When running **slurm_classify_clones.slurm** on your cluster, change the **mail-user, account,** and **job-name** fields as appropriate, as also the **source** path for your cluster account's Python environment.

Then in **sbdb_read_clone_classifications.py**, change **ianmatheson-sbdb_classify_clones-2689905-** to the appropriate stem for the **.out** files produced by your cluster. Similarly, change those fields in **slurm_fortran_mean_planes.slurm**.

If you desire (for instance, if there are more than 2748 objects to clone or you can run more than 300 jobs on your cluster), you can change the number of batch jobs in **sbdb_generate_classify_clones.py** and **sbdb_generate_fortran.py** from 275 and 300, respectively, to whatever you desire, but you then need to change the **array** fields in **slurm_classify_clones.slurm** and **slurm_fortran_mean_planes.slurm** accordingly, as well as all Python references to **Njobs = 275** and **Njobs = 300**, such as in **sbdb_read_clone_classifications.py** and in **sbdb_mean_plane_consolidate.py**.

In **sbdb_mean_plane_consolidate.py**, you also need to change **rootstr** to the appropriate job name as in the numbered **.out** files produced by your cluster when computing the mean planes with Fortran.

To use the results of the calculations in plots, make the following changes in **mean_plane_plots.py**. I only mention changes that have not already been made if you've been following step-by-step, changing all Fortran and Python references as you go along. The object counts per bin can be seen in filenames such as **sbdb_query_results_delcols__objct215_amin42_amax43_nominal.txt.** Change all references to **__objct####** to the appropriate object count per bin as seen in the filenames, eg **__objct215**. Similarly, change the counts in each appearance of **objct_list** to the appropriate values.

Make the same changes in **show_convergence.py** to get convergence plots for the inclination and nodal bounds for your new calculations.

CONTACT INFORMATION

For any questions, please email ianmatheson AT email DOT arizona DOT edu. I'll be happy to help with any questions about the work or problems trying to reproduce the calculations.