WP 1.2 Establishing Methodology

J. G. P. Vermeulen

May 26, 2021

1 Contents

After establishing the necessary mathematics, it is good to give the methodology some thought before committing to start coding. This is important as some of the structuring of the data and the program flow will be a lot of work to change later down the line.

This work package contains detailed descriptions of the methodology for the thesis work. The contents are as follows:

- Concepts for metrics and data structuring.
- A note on parallelization
- Numerical methods
- Extended Kalman filter
- Dense neural network
- Recurrrent neural network
- Other techniques?

2 Concepts for metrics and data structuring

First, the structure of the data and the metrics to be used will be set out. This will be annoying to change later on. For structuring the data, the obvious choice is using NumPy arrays: Tensorflow works natively with NumPy, NumPy provides good matrix algebra functions, and allows vectorization of functions for fast processing. Additionally, large clumps of data can be quickly saved in Python's serialized Pickle format for quick and efficient storage. Disadvantage of NumPy is that data will have to be kept in memory, but I assume for now that 16 GB will be enough to accomplish most tasks.

As most operations are in the form of $y = f(x)$, saving it as a (x, y) tuple makes sense. It is then easily unwrapped using (x, y) = data. The y value represents the target metric, the x values are a NumPy array of inputs. When

using only angles as inputs (skipping the image processing), the data for a single observation is structured as follows:

$$\left(\begin{bmatrix} \begin{bmatrix} \alpha_0 & \delta_0 & m_0 \end{bmatrix} \\ \begin{bmatrix} \alpha_1 & \delta_1 & m_1 \end{bmatrix} \\ \vdots \\ \begin{bmatrix} \alpha_n & \delta_n & m_n \end{bmatrix} \end{bmatrix}, y \right) \tag{1}$$

These observation tuples are then stored sequentially in an array if they are randomly generated, or alternatively, in a predetermined shape for studying the effects of specific parameters.

For storing "images" of observations, the x arrays of α, δ, m are replaced by uncompressed bitmaps of the images. This can then also be easily visualized using e.g. plt.imshow().

The next point of disucssion is the impact hazard metric. It might be an obvious idea to simply calculate the chance of impact / closest approach to Earth / etc in the next (e.g.) 100 years. This would correspond with known metrics such as the Torino scale. However, I consider this metric to be inadequate for the research goal: the question is whether or not the impact hazardous asteroid can be **identified**, not whether we can perform some hyper-accurate orbit determination. This leads us to the concept of minimum orbit intersection distance. I.e. what is the closest approach of the asteroids orbit to Earth's orbit. This will of course not account for phasing differences (which is convenient, as it avoids having to take any Earth emphemeris into account and essentially makes the problem circularly symmetrical around the barycenter of the solar system, if we assume Earth's orbit to be circular). Additionally, such a single metric allows using regression-based solutions.

The MOID, however, fails to fully quantify the risk. Because of imprecisions in the determination of the MOID, we can not accurately assess the actual impact probability, only whether or not the object has any impact probability. Additionally, the actual effect of the impact (big rock = more damage) is also neglected. These problems could be resolved using a more detailed metric. For example, finding an accuracy in the value of the MOID and assessing the risk using a normal distribution, or actually doing orbit determination. I consider these options outside of the scope of the research however. The option I do think that might be worth considering is the size of the asteroid. This is however offset by the fact that a larger asteroid will have a larger SNR owing to its brightness. This might skew the results towards heavier asteroids. Therefore, initially, I think the MOID is the best candidate for a metric.

3 A note on parallelization

Next to being a very convient and flexible tool for processing scientific datasets, python is known for another thing: it is painfully slow. The flexibility of an interpreted language comes at a significant penalty to performce. There are solutions to this proble, however: mainly in the form of parallelization. Tools like

Tensorflow and SciKit Learn incorporate this principle natively, and we should strive to set up our own functions to facilitate this. The idea of parallelization is that instead of processing a single floating point operation and looping through all the data:

1. Get data in the form of $(y_0, x_0), (y_1, x_1), \ldots, (y_n, x_n)$
2. Calculate $y_0 = f(x_0)$ and record result
3. Calculate $y_1 = f(x_1)$ and record result
4. Calculate $y_2 = f(x_2)$ and record result
5. etc.

It is more efficient to do:

1. Get data in the form of $(y_0, x_0), (y_1, x_1), \ldots, (y_n, x_n)$
2. Set vectors $\vec{y} = [y_0, y_1, \ldots, y_n]$, $\vec{x} = [x_0, x_1, \ldots, x_n]$
3. Calculate $\vec{y} = f(\vec{x})$ and record result

As the speed of python is generally limited by processor clock speed (most operations are very simple), the latter is clearly much faster. This means all functions should (where applicable) be constructed as vector functions. Note that this is only possible if all x's and y's are independent of eachother.

If necessary, we could speed up the process even more by using e.g. CUDA to do the calculations on the cores of a GPU. GPU's often have several orders of magnitude more cores than CPU's and can therefore perform (simple) calculations at a significantly higher rate. I think that, for now, this is not a necessity.

4 Numerical Methods

5 Extended Kalman Filter

The core work on the Kalman Filter was already performed in the mathematical foundations work package: the equations needed for the system were developed. However, the EKF still has some loose ends. Firstly, like the numerical methods, we are faced with the question on how to determine the MOID. As the state vector does not include the MOID, and there is no other logical place to include it either, it is probably easiest to do a simple orbit determination based on $\vec{x}$ and $\dot{\vec{x}}$.

The second problem of the EKF is the initial state estimation. As the observation matrix is dependent on the guess of the state vector, the problem might take a very long time to converge (or might not even converge at all) for an inaccurate initial state estimation. Initially, it should be possible to do a very rough estimation using the apparent magnitude. If this proves too inaccurate, we might need to either perform a more complex state estimation (e.g. using the method of very short arcs) or implement an Iterated Extended Kalman Filter,

where the steps of doing the observation and determining the Kalman gain are iterated. This allows converging faster at minimal loss of processing time.

The last issue involves the number of observations allowed to the system. There are two approaches that might be taken here: firstly, the system can be allowed a limited number of observations. The upside is that this might be more realistic and allows a better comparison with other techniques. The downside is that the slow convergence of the EKF might give really bad results. The second approach is continuing the observation until the system converges. In this case it would be important to monitor the number of observations necessary to obtain convergence. A last possibility would be to continue until convergence, or until observation is not longer possible (e.g. based on apparent magnitude).

6 Dense Neural Network

7 Recurrent Neural Network

8 Classical Machine Learning

As fancy techniques are not necessarily the best way of doing something, it might be worth considering more classical machine learning techniques. Specifically, as the metric used is a single value, we can consider some regression techniques based on classical machine learning. Even though these techniques will probably not give good results considering the high non-linearity of the problem, they are a good check on whether or not the problem will generalize. If such a simple technique performs unnaturally well, it might indicate that the training data is too good and the solution is not robust to generalization.

Specifically, it might be worth it to quickly check three different techniques: Firstly, regression trees / regression forests, secondly, regression using support vector machines, and lastly, a nearest-neighbour algorithm. I think especially the latter will give some interesting solutions: it might be possible to just "learn" the system *all* possible observations and just fit the new observation to the known (gigantic) dataset. The size of the dataset isnt necessarily an issue as it can be processed very efficiently. However, a good performance on this will probably generalize very badly and could serve as a robustness check on the altogether test procedure.