

**School of Physics and Astronomy**

**Year 3 Project Report**

**Session 2016-2017**

|  |  |
| --- | --- |
| **Name:** | **Michael Norman** |
| **Student Number:** | **1325126** |
| **Degree Programme:** | **BSc Physics** |
|  |  |
| **Project Title:** | **Simulating the possibility of a transfer of complex life from Venus to Earth.** |
|  |  |
| **Supervisor:** | **Dr A Cartwright** |
| **Assessor:** | **Dr M Matsuura** |

Declaration:

I have read and understand Appendix 2 in the Student Handbook: “Some advice on the avoidance of plagiarism”.

I hereby declare that the attached report is exclusively my own work, that no part of the work has previously been submitted for assessment (although it may re-use material from the interim report for **this project** as it is considered part of the same assessment), and that I have not knowingly allowed it to be copied by another person.

.

# Abstract

Lithopanspermia hypothesises that material ejected from the surface of a planet via sufficiently violent events can transport life between bodies within the solar system. There is a school of thought that the planet Venus may once have been capable of supporting life. N-body simulations were run to assess the best possible conditions for a transfer of ejecta from the surface of Venus to Earth as well as the likelihood of such a transfer. Estimations of the probability of such a transfer were found for a 1, 10 and 100 year time frame. Particular interest was placed around the coincidental timing of certain astronomical events, namely, the resurfacing of Venus, the Cambrian explosion, and the L-Condrite asteroid breakup. These fairly short time frames were used as the possibility of the transfer of complex life was considered over simple microbes.

Contents

[Introduction 4](#_Toc481929462)

[Method 8](#_Toc481929463)

[Simulation 8](#_Toc481929464)

[Testing 9](#_Toc481929465)

[Ideal Transfer Condition Simulation 9](#_Toc481929466)

[Probability Determination Simulation 10](#_Toc481929467)

[Results 12](#_Toc481929468)

[Ideal Transfer Simulation Results 12](#_Toc481929469)

[Probability Simulation Results 13](#_Toc481929470)

[Conclusion 14](#_Toc481929471)

[Discussion 14](#_Toc481929472)

[Reference List 15](#_Toc481929473)

[Appendix I – Development of simulation 15](#_Toc481929474)

[Appendix II – Reflective Statement 16](#_Toc481929475)

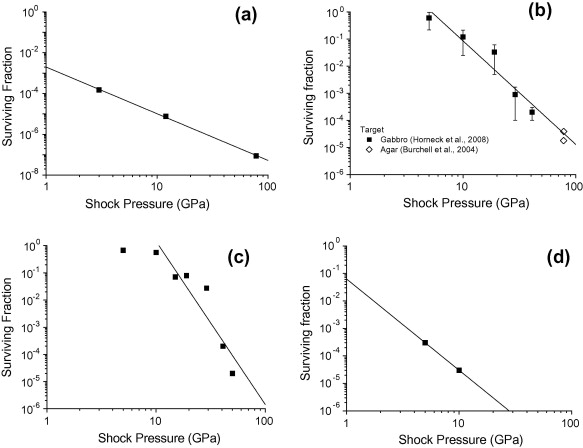
[Appendix III – Code 17](#_Toc481929476)

# Introduction

The subject of Panspermia has been a much discussed one in recent decades, often, however, it is the more glamourous idea of interstellar panspermia that gets the spotlight. The notion of alien microbes carried in on some wayfaring rock to kick start things on Earth is certainly a romantic one. That’s not to say that there isn’t a lot to be said on the idea that life, even within the confines of the solar system, earth originated or not, may have spread itself around on material ejected from a planet’s surface during one or many of the violent collisions that were so commonplace in the early solar system. Certainly, such an inter-solar transfer seems a lot more likely when compared with the vast distances involved with an interstellar version.

Possibly among the first to delve into the technicalities of such a transfer was an article in Nature published in 1988 by Melosh [1], entitled “The rocky road to panspermia,” it discussed the details of how much material could be ejected in an impact, as well as the feasibility that any microbial life could have survived the event. It has however, little to say on the likelihood of such an orbital transfer from occurring once the ejecta has reached escape velocity. I would postulate that this is majorly in part to the lack of computation power readily available at the time, though in reality it is possible that it was just beyond the scope of this particular investigation.

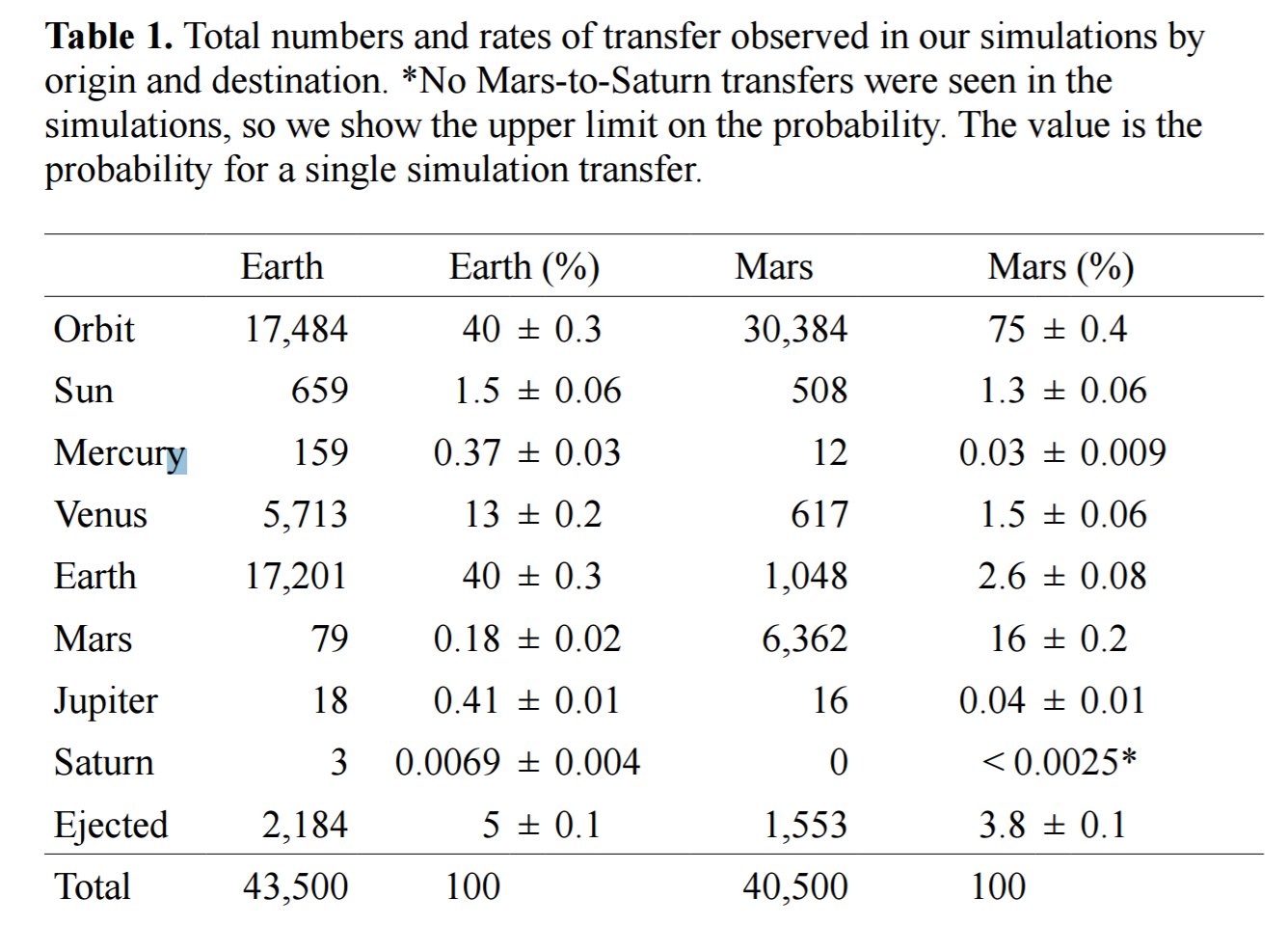
Another thing I would note is that whilst the article makes an estimate on the pressure and temperatures at which life could no longer survive, there have been some modern studies showing that microorganisms can survive at pressures far in excess of the 0.1GPa quoted, though the number of organisms that survive such pressures has been found to decrease at an approximately logarithmic pattern [2], see **figure 1**.



**Figure 1:** Survivability of Microorganisms under differing pressure. Figure taken from [2], with numerous sources as describe in figure description taken from source, “(a) Rhodococcus erythroplisimpacting agar ( [Burchell et al., 2004](http://www.sciencedirect.com/science/article/pii/S0019103512004447#b0035)). (b) Bacillus subtilis in gabbro (solid squares are data originally from[Horneck et al. (2008)](http://www.sciencedirect.com/science/article/pii/S0019103512004447#b0080)) and impacting on agar (open diamonds, [Burchell et al., 2004](http://www.sciencedirect.com/science/article/pii/S0019103512004447#b0035)). (c) Xanthoria elegansin gabbro ( [Horneck et al., 2008](http://www.sciencedirect.com/science/article/pii/S0019103512004447#b0080)). (d) Chroococcidiopsis in gabbro ( [Horneck et al., 2008](http://www.sciencedirect.com/science/article/pii/S0019103512004447#b0080)). General note: the data from [Horneck et al. (2008)](http://www.sciencedirect.com/science/article/pii/S0019103512004447#b0080) are from flyer plate experiments with the biological material mounted on the target which, was then shocked. Their values are restated here with newly calculated shock pressures from [Meyer et al., 2011](http://www.sciencedirect.com/science/article/pii/S0019103512004447#b0125) (and translated to surviving fractions rather than percent). The data from [Burchell et al. (2004)](http://www.sciencedirect.com/science/article/pii/S0019103512004447#b0035) are from impact experiments where the biological material was carried on a projectile and impacted the stated target material.”

Since that time there have been many but not extensive papers on the subject which have included progressively more complex and adept simulations, especially relating the complex often million yearlong simulations required for the simulation of ejecta transfer to the outer planets and moons.

One such study (R. J. Worth, Steinn Sigurdsson, Christopher H. House) [3], looked at the possibility of life from Earth or Mars spreading throughout the solar system with a particular interest in planets thought capable or once capable of sustaining life. Earth, Mars and some moons of the outer solar system such as Titan and Europa. Notably, Venus was only included in their study as a non-discussed location that the ejecta could eventuate at.



**Table 1:** Table taken from [3] depicting number of transfers from Earth and Mars to the outer solar system within the 10 million year duration of the simulation.

As can be seen from table 1, though there was no simulation of objects ejected from Venus, there was a significant percentage of ejecta, 13%, that travelled from Earth to Venus. The inverse, a Venus to Earth transfer, would be an entirely different endeavour seen as the gravitational potential is inversed. However, even to decrease its orbit the ejecta would have had to utilise a certain velocity.

The research that will be carried out relates primarily to the potential transfer of life from Venus to Earth, more specifically, in relation to both the resurfacing event that may have occurred sometime around 500 Myr [4], quite interestingly around the time of both the L-chondrite asteroid breakup, approx. 450 to 520 Myr [9] , which potentially lead to a large increase in the number of impacts on Earth surface, and the Cambrian explosion, approx. 543 Myr [8], where in the diversity of life on earth greatly increased in magnitude in just a few million years, a short event on evolutionary timescales. [8] The coincidental timing of the three events is something that warrants investigation, if nothing else, since the possibility of life on Earth origination from Venus would be such a fascinating one.

There have been few if any confirmed Venus originated meteorites found on the surface of earth, (no information could be found on the subject, though it is presumed to exist somewhere), presumably because of Venus’s thick atmosphere and its depth within the gravitational well compared to earth. However, meteorites from Venus that impacted during the time period that this investigation is interested in should be significantly harder to locate than those from other bodies that have impacted much closer to the present. Natural processes such as sedimentation and erosion and general changes to the Earth’s crust will have long since buried or destroyed such evidence, even to the point where there would be virtually negligible evidence of Venus ejecta today despite the fact that in the past it may have been more commonplace.

The change in this likelihood is hypothesised to arise from a few key events, namely, the resurfacing of Venus and a possible drastic change in atmospheric composition, whether independent or independent on resurfacing events. During resurfacing such violent events may have occurred that ejecta may have gained sufficient velocities to reach earth orbit and thus impact its surface. Also noteworthy, is the idea that Venus’s thick dense atmosphere may not have existed prior to the resurfacing event.

The idea of a catastrophic resurfacing of Venus’s surface, first derives from the density and distribution of craters on Venus’s surface, were Venus’s surface to have existed in its current state for a much longer period than 750 million years or so, the densities of cratering would be expected to be much higher along with an increased amount of large impacts. The surface of Venus as it stands, has a distribution of craters very close to random, with volcanism seemingly not having modified many craters at all. In a large multi-billion year history, the volcanic activity present on the surface should have ensured a relatively non-random distribution.

However, what is observed of its surface does not seem to match what would be expected, it has far less cratering in a distribution that is not indicative of a long existent static surface. [5] One of the idea’s proposed, in the cited paper along with several others, as a solution to this discrepancy is that the total, or at least majority of the crust was violently altered and wiped clean of any previous impact records.

In the article, “Resurfacing of Venus,” [5] the idea is inspected with some detail, further building on precious hypotheses. Monty Carlo simulations were used to compare the ending distributions of different scenario’s with the evidence seen today varying multiple factors during the simulation such as resurfacing time and volcanisity of the surface after the resurfacing event, assuming that 40% of the surface experienced moderate volcanic activity, they were able to reproduce a distribution similar to what is seen today assuming that a catastrophic event occurred approximately 1 billion years ago.

Such an event may perhaps have been violent enough to produce eject with the required velocities to reach and intersect Earth’s orbit. Although the timings in the cited paper gave evidence for an event one billion years ago which would fall outside of the time frame the coincidence inspected would fall in the time frame suggested by many other papers. Such as this older paper [5], suggest a time frame, of between 200 and 500 million years ago, this is the more commonly investigated time frame and one that aligns nicely with the time frame of the coincidence being strongly investigated. However it should be noted that as evidenced by the previously cited articles 4 and 5, there is still much debate on the time frame of the event or even if the event happened at all, though it is seen by many as more likely that the alternative [4]. If the resurfacing and hence next discussed topic of habitability were indeed pushed back to a billion year time frame. It would not invalidate the investigation, simply, it would negate some of the more pleasant coincidences in timings, the possibility of a transfer of like occurring is still there even when earlier in time.

In a few recent papers, new hypotheses about the previous condition of Venus’s surface and atmosphere have been drawn [6]. They suggest that Venus may have been previously habitable some time before or leading up to the resurfacing event. The article states that the ratio of hydrogen isotopes present on Venus indicate that Venus once held a large amount of water which it has obviously subsequently lost. The article also states that though Venus’s rotation rate is extremely slow, a fact previously thought to have arisen to some extent through friction with the atmosphere, an atmosphere of just one bar combined with tidal bulge forces from the sun, could perhaps alone have led to the rotation rate that we see today.

Thus the article predicts that Venus may once have indeed been habitable and a runaway greenhouse effect may have led to its thick atmosphere seen today. Perhaps initiated by its reduction of acceleration or perhaps by the aforementioned resurfacing event. Though it should be noted that the resurfacing event is often hypothesised to be a recursive cycle reliving pressure every 500 million to a billion years, so if the effect was caused by such, then questions about why previous events didn’t cause such a result should be asked.

The primordial Venus though sharing some similarities to Earth would indeed have been a very different place, even at a slightly faster rate than the present, its rotation would have caused dramatic changes in temperature throughout the Venusian day. According to the simulations run in the article [5], going from average equatorial temperatures as low as -22°C during the night to 84°C during the day depending on the simulation parameters inputted.

Whilst quite extreme, such conditions would not be prohibitive of the existence of life even in comparison with many hardy earth lifeforms, and indeed the necessity for such life to evolve with a certain resistance to temperature change may aid it in the challenge of leaping from planet to planet aboard a meteorite. So indeed the possibility remains that Venus may once have been habitable, even hospitable to life and with the scientific significance of such an event further investigation into the subject of Venus to Earth transfer is certainly warranted.

There has, as has been previously noted, been much research into most areas around the subject of lithopansperimia. Most of which it would not be possible to improve on with the limited resources and time frame available. Therefore, survivability of ejection, transit and re-entry as well as the likelihood of germination from the impact point will not be taken into consideration for this first stage of the investigation other than to compare metrics found with results from previously performed investigations.

The largest hole in research falls around the transit of material from Venus to Earth, most other investigations have taken the source of eject from Earth and/or Mars, without consideration of Venus as there had been previously very little evidence for its past habitability. This is thus, the area in which the initial investigation shall be performed.

# Method

## Simulation

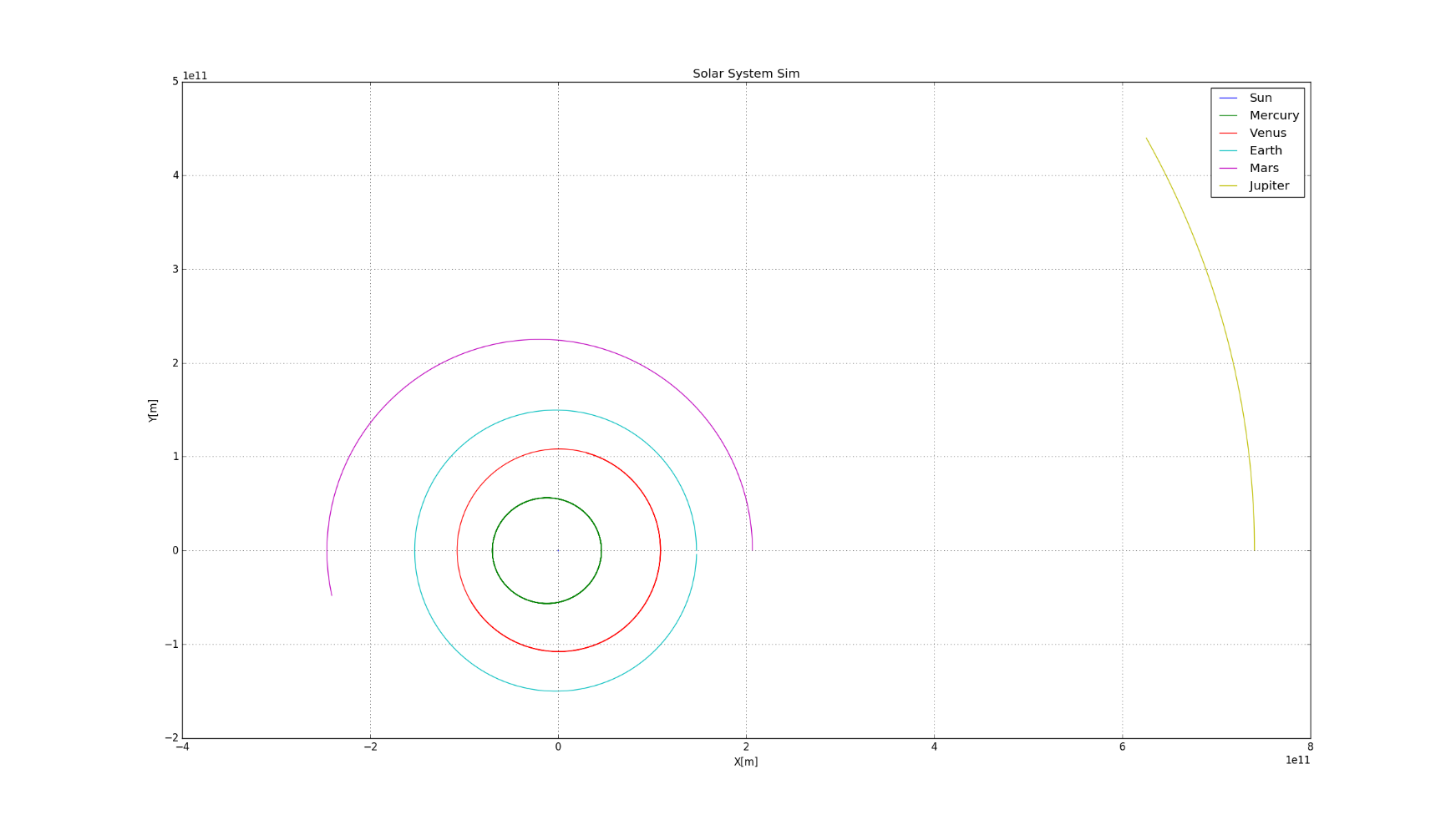
The n-body simulation that was used to predict the paths of the orbital bodies was created specifically for the task.

The simulation used a symplectic integration method, the velocity Verlet method, to evolve the meteoroids with time. It uses the vector form of Newton’s Law of Universal Gravitation (1) to calculate the acceleration on the bodies at each timestep.

|  |  |  |
| --- | --- | --- |
| (1) | = | Where |

The acceleration on each body from each other body was summed at each timestep, then those values of acceleration were used to evolve the system over time.

The Verlet method was chosen over other integration methods for its fairly good accuracy and stability compared to its computation costs. It is also symplectic meaning that total energy of the system should be conserved, and important factor in n-body simulations as it ensures that bodies do not gain or lose energy over time in an unrealistic fashion. Whist Euler’s method simply calculates a new acceleration based on the position of the previous timestep, then calculates a new velocity and hence position based on that, the velocity Verlet method calculates the new position first, based on the previous velocity, then calculates the new acceleration based on the new position, then calculates the new velocity based on an average of the previous and current acceleration, in effect offsetting the acceleration by half a timestep. This leads to a more accurate and stable simulation, see **figure 2.**



**Figure 2:** Evolution of the planetary positions over one year using the simulation.

In each individual simulation thirty six meteoroids were simulated along with the Sun and the first five planets. The planets beyond Jupiter were deemed gravitationally insignificant enough, given the time frame and the accuracy level to which the simulation was working, that they were excluded to save computational time. The meteoroids themselves were also taken as gravitationally insignificant and modelled as massless particles again to save computational time.

Information about the orbits and masses of the sun and planetary bodies included in the simulation was taken from a NASA website [7], see table 2.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Name | Mass [kg] | Radius [m] | Initial Position [m] | | | Initial Velocity [ms-1] | | |
| X | y | Z | x | Y | z |
| Sun | 1.98850E+30 | 6.9570E+08 | 0 | 0 | 0 | 0 | -14.25 | -0.30 |
| Mercury | 3.30110E+23 | 2.4397E+06 | 4.60000E+10 | 0 | 0 | 0 | 5.8540E+04 | 7.188E+03 |
| Venus | 4.86750E+24 | 6.0518E+06 | 1.07480E+11 | 0 | 0 | 0 | 3.5200E+04 | 2.085E+03 |
| Earth | 5.97240E+24 | 6.3781E+06 | 1.47090E+11 | 0 | 0 | 0 | 3.0290E+04 | 0 |
| Mars | 6.41710E+23 | 3.3962E+06 | 2.06620E+11 | 0 | 0 | 0 | 2.6386E+04 | 8.520E+02 |
| Jupiter | 1.89819E+27 | 6.6854E+07 | 7.40520E+11 | 0 | 0 | 0 | 1.4717E+04 | 3.120E+02 |

**Table 2:** Initial conditions inputted into simulation, information retrieved from [7]. *Note I:* velocities split between y and z to simulate orbital inclination, inclination 0 taken as Earth orbit, i.e. the elliptical plane, hence all of Earth’s velocity lies in the x direction. *Note II:* Sun velocity added to ensure total system momentum equal to 0.

## Testing

The simulation was run several times with only planetary bodies in the system to test the accuracy of the simulation. A simple scenario was performed, running the system for exactly one Earth year to see how close the planet would return to its initial starting location in the time that it is known to complete one rotation. The simulation was run in 100 seconds timesteps for 365.24 days with each day lasting 86400 seconds. After this run time the Earth was found to lie within 2.09 million km of its starting position, an error of 0.22% compared with a known value of orbital circumference.

## Ideal Transfer Condition Simulation

In the initial simulations to calculate the ideal transfer conditions the planets were positioned on the same line along the simulations x-axis. The 36 ejecta were positioned equally around Venus in a ring with radius of the Hill sphere, see figure 3. The ring was angled at 0° inclination, i.e in line with the orbit of Earth. A radius of the Hill sphere was chosen to avoid most of the gravitational effects from Venus itself, since the Hill sphere is the radius after which the sun’s gravitational influence dominates. Though in reality the meteoroids would have to have additional velocity to those used to get to that radius, the program could not accurately simulate a launch from the surface. Plus collisions were not dealt with in the simulation leading to strange results as objects would fall though the centre of the planet and gain enormous velocities. The Hill radius used in all simulations was 6.28x108m.



36 Ejecta

**Figure 3:** Initial simulation set up.

Each of the meteoroids was given a velocity equal to that of the initial velocity of Venus plus an additional ejection Velocity away from the centre of Venus. The simulation runtime was 1 year with 1000 second timesteps. To find the minimum required meteoroid velocity at the Hill sphere the simulation was repeated with ejection velocities equally spaced between 1000ms-1 and 5000ms-1 in nine simulations. For each ejecta in each simulation the orbital aphelion, d, the maximum distance from the sun during the simulation was recorded.

Next, a simulation was run to determine the best starting angle for Venus to achieve a one year transfer, in the second test, rather than vary the velocity, the velocity was kept constant at 3000ms-1 and the orbital positon of Venus around the sun was varied between simulations in 12 simulations again each of length one year with a 1000 second timestep with Venus orbital angle varied between 0° and 360°.

To enact this rotation a rotation function was created that could rotate any vector around any point, with a given direction and magnitude of rotation. It used the product of three rotation matrices to produce three equations that could perform any rotation given the previously mentioned inputs.

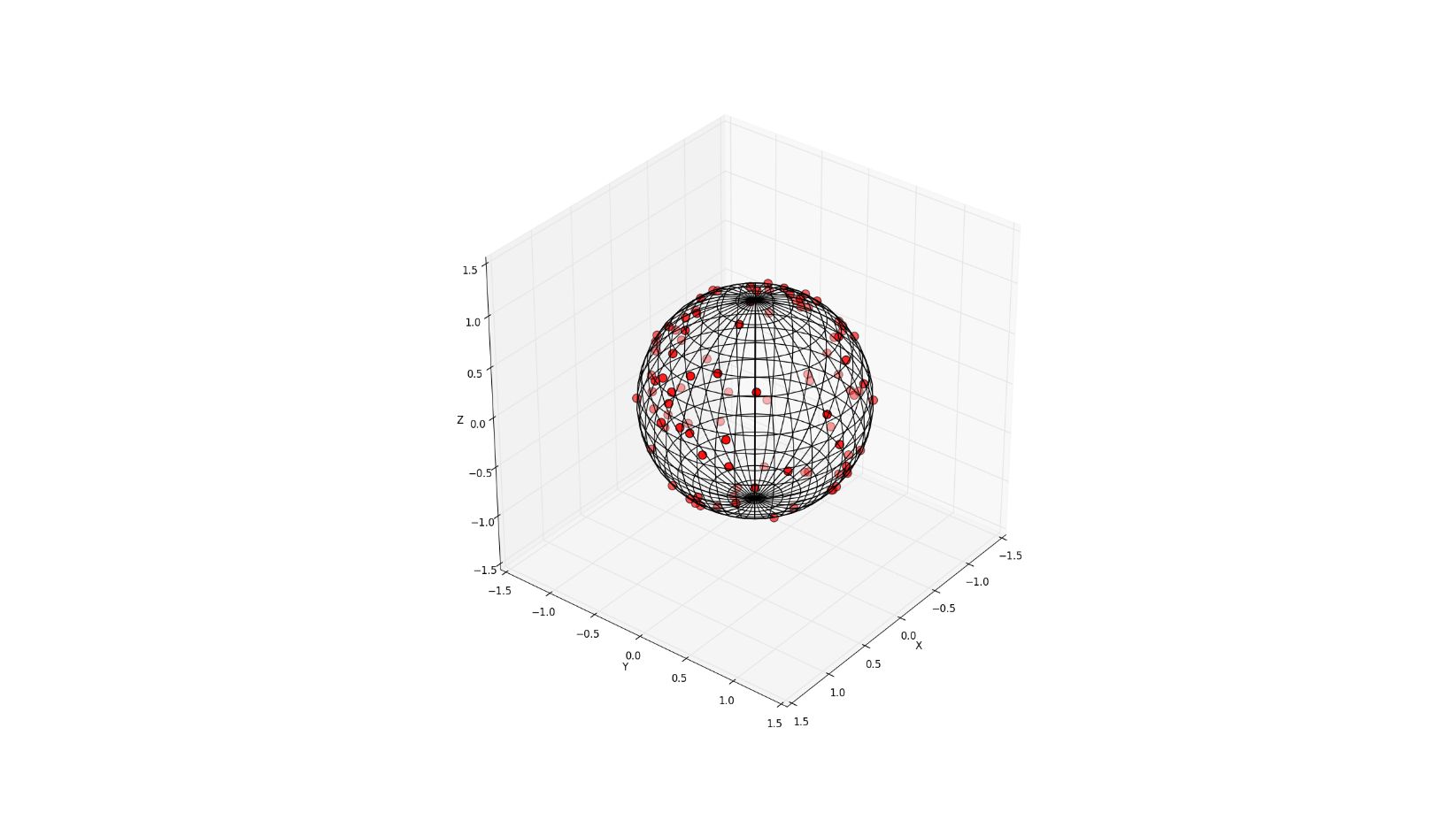
## Probability Determination Simulation

The simulation was run multiple times for each of the one year, ten year and 100 year time periods investigated. For each of the three sets of simulations the timestep was kept at 1000 seconds meaning computational times were longer respectively for each set of simulations. This meant that less simulations could be reasonably performed at higher simulation lengths. 6780 simulations were run for one year, 1600 were run for ten years and 140 were run for 100 years. With each simulation containing thirty-six meteoroids, the total meteoroids run for each length were 244080, 57600 and 5010 respectively.

In each simulation, the orbital positions of all of the planets was randomised. This was achieved by rotating the planets position and velocity vectors around the origin by a random degree in the z direction.

Each of the meteoroids starting positions were randomly generated across the surface of Venus’s Hill sphere around the position of Venus at the start of that simulation. The escape velocity at that radius was used as 1020ms-1.

Their random positions were assigned using an inbuilt CPU function to generate three arrays of random numbers of length 36, i.e. x, y and z arrays, then converting these vectors into unit vectors by dividing each x, y and z by the magnitude of the respective vector. This gave a set of points randomly distributed over a sphere, see **Figure 4**. The vectors can then be multiplied by the required radius to produce a sphere of the correct size.



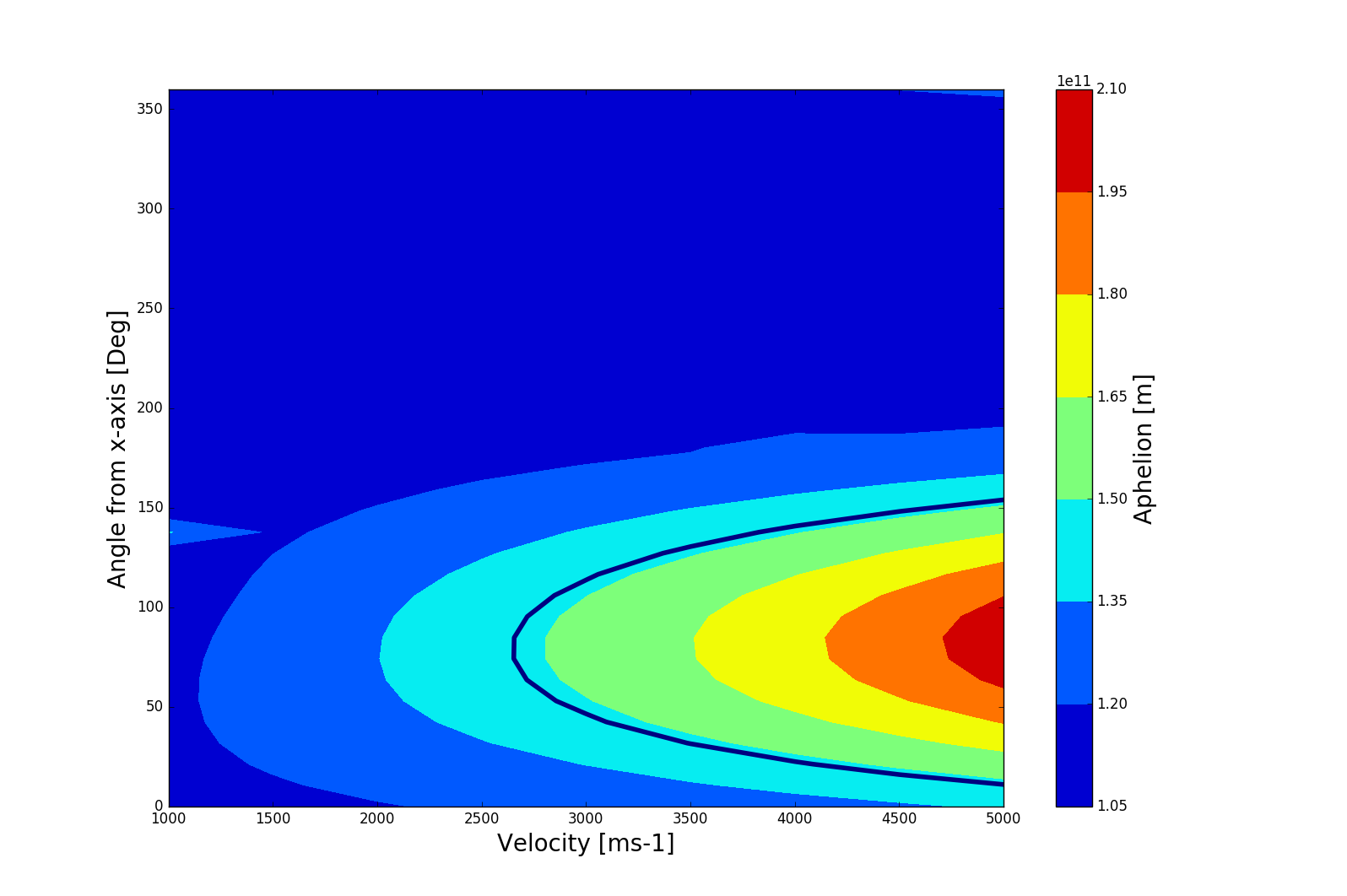
**Figure 4:** Random distribution of 36 points across a sphere using above method. *Note:* Sphere grid just for visualisation and is not involved in the creation of points at all.

The meteoroids given initial velocity was a combination of Venus’s own initial velocity plus one to three times the escape velocity at the Hill sphere in a vector away from the centre of Venus. Meaning that no meteoroids had a velocity lower than that required to at least escape from Venus.

For each timestep a value of the magnitude of displacement from Earth to each meteoroid was calculated. Then for each meteoroid the closest point to Earth over the whole simulation was found. Using this value, the number of meteoroids per simulation that came within Earth’s Hill sphere was calculated. Earth’s Hill sphere was taken to be 1.5 million kilometres.

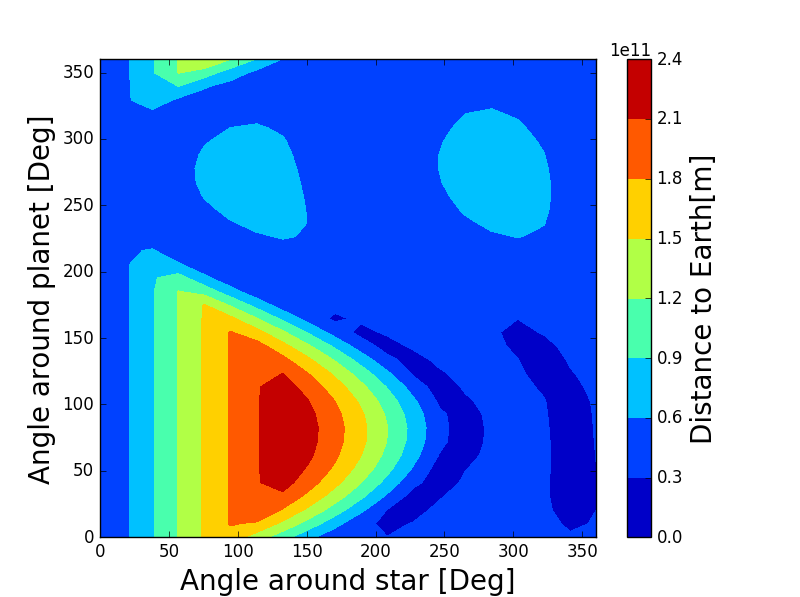
# Results

## Ideal Transfer Simulation Results



**Figure 5:** Meteoroid aphelion (max distance from the sun) vs ejection angle and velocity. The Blue line represents Earth’s orbit, anything within that blue line has some chance to hit Earth. *Note:* An angle of 0° indicates that the ejecta was launched along the x axis in a positive direction, whereas an angle of 90° would indicate the ejecta was launched along the y axis in a positive direction, in this case alone the direction of motion of Venus.

As might be expected the greatest Aphelions are achieved with faster velocities and nearer a prograde trajectory, i.e. in the direction of Venus’s initial velocity vector, see **fi**g**ure 5.** The lowest velocity to intersect Earth’s orbit is at 90° and around 2750ms-1, bearing in mind this is the velocity required at the Hill sphere, meaning that significantly greater velocities would be needed from the surface of Venus.



**Figure 6:** Meteoroid closest distance to Earth vs ejection angle and Venus starting angle.

See **figure 6**,the blue patch in the lower right of the diagram represents the times when Earth started fairly close to Venus in the first place so is largely uninformative. The area that is fairly informative is the one slightly to the left of that, with Venus at around a 270° rotation. This is the best possible place to launch a meteoroid from and have it hit earth. Again, this is as might be expected as the orbital period of an elliptical orbit is the average of the two axis, meaning an orbital period half way between Earth’s and Venus’s. Assuming it would take one half an orbit for a meteoroid to get from Venus to Earth, it makes sense that Earth would have to start a quarter orbit distant from the collision point.

## Probability Simulation Results

There were no recorded collisions of any meteoroids with Earth during any simulations. The closest that any meteoroid came to collision with Earth was 110,000km. A measurement was made of the number of meteoroids that entered the Earth’s Hill sphere at least once during their respective simulation, the hypothesis being that if an object were to pass through the sphere there is some possibility of capture, though none was seen in the simulations. No further analysis or simulation was performed to gauge probability of meteoroid capture.

The results can be seen in full in **table 3**. Approximately 0.01% of all meteoroids passed through the Hill sphere in a period of one year. Over increasing simulation lengths the increase in percentage appears to follow in a linear patter though there is some fall off in the hundred year simulation.

|  |  |  |  |
| --- | --- | --- | --- |
| Duration [Years] | 1 | 10 | 100 |
| Simulations Run/Angles simulated [Number] | 6780 | 1600 | 140 |
| Meteoroids Simulated [Number] | 244080 | 57600 | 5010 |
| Passed through Earths Hill Sphere [Number] | 26  [±6] | 39  [±3] | 38  [±1] |
| Percent [%] | 0.011  [±0.003] | 0.097  [±0.005] | 0.75  [±0.02] |
| Closest Passage to Earth’s centre [Km] | 122000 | 218000 | 110000 |

**Table 3:** Simulation results separating number of meteoroids that passed through the Earth’s Hill sphere. *Note:* Error is order of magnitude estimate arising from not perfectly average random distribution of initial starting conditions calculated using standard error. In reality, lower simulation numbers would have an increasingly large effect on the random distribution not accounted for.

# Conclusion

Given the results of the simulations to predict the ideal transfer conditions, the most probable conditions for a transfer are a meteoroid of approximate speed 2750ms-1 ejected along a prograde trajectory at time when Earth is offset from Venus by 270°.

From the results collected from the probability simulations it is impossible to calculate any reasonable estimate for the collision likelihood for any given meteoroid ejected from Venus with Earth. The results may however serve as upper bounds for such probabilities. For a collision to occur the closest distance to Earth’s centre must be less than the radius of Earth, 6371km. Looking at the closest passage to Earth’s centre, 110 000km, a logical step can be made to say that for the likelihood of a single collision to approach one, either many more simulations would be needed or the simulation would need to be run for a much greater time.

What can be said perhaps, is that the likelihood of a transfer event occurring in such short timescales <100 Yrs, is fairly low, at least less than 0.75%, probably a few orders of magnitude less, given the size of Earth compared with the closest approach and the inverse square relation of area to radius.

# Discussion

There are many factors that may have introduced error into the simulations, most of which can’t be easily mathematically accounted for. One of the largest of which may have been the low variation of angular orbital positions at lower simulation counts of the 100 year simulations in particular. 140 simulations would likely not have been enough to produce as even spread of angular positions as would be preferred. Angular positions were not recorded so there is no way to account for this possibility, however it is possible that this may be one of the main causes of the discrepancy from the linear relation that is seen at the 100 year simulation Hill sphere count. Error should also increase simply with total meteoroid count even not accounting for the angular positions dictated by simulation count.

Values for error were given using a standard error approximation to give approximately the correct magnitude of error, however as was previously stated the reliability of these errors is untested.

Another major factor unaccounted for was material return to Venus, only distance to Earth was calculated at each timestep so it is unclear how large a percentage of material would re-enter Venus’s Hill sphere, looking at previously completed simulations by different authors [3], the source planet is usually the prime destination, so many meteoroids that might have found their way to Earth otherwise might be instead captured back by Venus.

Further study would involve running longer simulations and possibly greater number of simulations at smaller time frames to achieve Earth collision and calculate a probability of transfer with some degree of reliability. This would involve a few further adjustments to the program but ultimately more computational power would be needed to achieve the time frames necessary to find collisions. Also minimum distance to all planets would be recorded, not just Earth, with possibly some mechanism implemented to simulate collision. A smaller simulation to investigate the probability of meteoroid capture based on angle and speed might make the results of the investigation more meaningful.

# Reference List

[1] “The Rocky Road to Panspermia”, Melosh, *Nature,* 1988

[2] “Survival of yeast spores in hypervelocity impact events up to velocities of 7.4 kms-1”, M.C. Price, C. Solscheid, M.J. Burchell, L. Josse, N. Adamek, M.J. Cole, *Icarus,* Received 3 July 2012, Revised 4 October 2012, Accepted: 31 October 2012, Available online: 20 November 2012

[3] “Seeding Life on the Moons of the Outer Planets via Lithopanspermia”, R. J. Worth, Steinn Sigurdsson, Christopher H. House, *Astrobiology,* Nov. 10, 2013.

[4] “Resurfacing on Venus”, Romeo , D.L. Turcotte, *Planetary and Space Science*, Received 16 February 2010 Revised 29 April 2010, Accepted: 30 May 2010, Available online: 4 June 2010

[5] “Catastrophic Resurfacing and Episodic Subduction on Venus”, D. L. Turcotte, G. Morein, D. Roberts, and B. D. Malamud, *Icarus,* May 1999.

[6] “Was Venus the first habitable world of our solar system?”, M. J. Way, Anthony D. Del Genio, Nancy Y. Kiang, Linda E. Sohl, David H. Grinspoon, Igor Aleinov, Maxwell Kelley, Thomas Clune, open access letter, 28 August 2016.

[7] Nasa Planetary Factsheet, address, http://nssdc.gsfc.nasa.gov/planetary/factsheet/, Date Accessed: 2016-12-04, Last Updated: 18 November 2015

[8] “The Cambrian “explosion”: Slow-fuse or megatonnage?”, Simon Conway Morris, *PNAS*, April 25 2000

[9] “L -chondrite asteroid breakup tied to Ordovician meteorite shower by multiple isochron 40Ar-39Ar dating”, Ekaterina V. KOROCHANTSEVA , Mario TRIELOFF, Cyrill A. LORENZ, Alexey I. BUYKIN, Marina A. IVANOVA, Winfried H. SCHWARZ, Jens HOPP, and Elmar K. JESSBERGER, Meteoritics & Planetary Science, Received 22 June 2006; revision accepted 05 November 2006.

# Appendix I – Development of simulation

The majority of the time spent developing the simulation was spent in a trial and error approach to coding. The first versions of the n-body code produced were programmed entirely in pure python. This suited purpose fine for a long time and much developmental time was put into this version, adding features such as adaptive timestep’s and look up tables which were never re-integrated into the program after this point. Later versions of the python code was re-written in an object-oriented code to simplify use. However I think coding in that way became more of an abstraction to those unfamiliar with the code and was perhaps a mistake.

It became clear at the start of the second term that in order to garner any estimate of transfer probability, a larger number of meteoroids were going to need to be simulated. Even over the relatively small periods investigated simulations with large numbers of ejecta would take far too much computational time to be feasible. I did not have access to the large amounts of computing power that would be necessary to run it in any conceivable time. The code had to be sped up dramatically.

The first attempt at increasing the simulation speed was to completely rewrite it using numpy library functions. Numpy arrays and functions are written in c and as such have a significantly reduced performance impact over writing purely in python. This was a partial success, the simulation run time was reduced about ten-fold.

Soon after this I started working on a c program to run the simulation. In reality this was the only way to simulate the required number of meteoroids within the computation capacity and time frame that I had access to. The completed c simulation ran over 1000 times faster than my original messy object oriented python code.

# Appendix II – Reflective Statement

I’ve come a long way over the course of the project, both in terms of my programing and project management skills. Learning c coding from the ground up was an obvious achievement but even without that my knowledge of python and just general programming practice increased significantly.

Most of the time I spent on the project was spent on the code front, which did turn out very well but was perhaps a mistake, as a lot of what I did will probably not make much of a difference to the final scientific outcome, bearing in mind that it was the point of the exercise may have been a better idea, and spending more time on producing a more substantial set of results to extrapolate from.

I enjoyed the freedom to work on an open ended project. For most of the time I had a clear idea of what I was working toward at the time, though admittedly my goals (as would perhaps be expected) shifted over time as things changed. For example, at my original skill level it would have been impossible for me to produce a program capable of simulating the number of meteoroids that I, in the end, ended up doing.

Although the outcome of the project was not a particularly ground-breaking conclusion, I feel that in some way’s I’ve answered the question of probability. I would say that whilst not impossible the transfer of complex life between Venus and Earth on any short timescale would be extremely unlikely, though admittedly I have relatively little information about the amount of meteoroids that might be produced.

I have certainly developed my skills in oral communication, my Viva went pretty well. And my presentation, whilst perhaps lacking in scientific technicalities, was a great learning experience.

Overall whilst I am a little disappointed that I could not get an estimation of the likelihood of collision I am pleased with how the project went and certainly pleased with my personal development over its duration.

# Appendix III – Code

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <math.h>

#include <ctype.h>

#include <string.h>

#include <unistd.h>

#include <limits.h>

#include <float.h>

#include <stdbool.h>

#include <inttypes.h>

#include <immintrin.h>

#define G 6.67e-11

#define RAND\_64\_MAX ~(0ULL)

// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ FUNCTIONS ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ //

void get\_arg(double\* val)

{

char\* t = strtok(NULL, ",");

\*val = atof(t);

}

void get\_argf(double \*val)

{

char\* t = strtok(NULL, ",");

\*val = atof(t);

}

void get\_args(char \*\*val)

{

char\* t = strtok(NULL, ",");

\*val = t;

}

void get\_argi(int \*val)

{

char\* t = strtok(NULL, ",");

\*val = atoi(t);

}

void get\_argzu(size\_t \*val)

{

char\* t = strtok(NULL, ",");

\*val = atoi(t);

}

uint64\_t rand\_64()

{

unsigned long long val;

while(!\_rdrand64\_step(&val));

return (uint64\_t)val;

}

void readConfig(char\* filename, size\_t\* input\_bodies, char\*\* input\_filename, int\* prec, char\*\* output\_filename, int\* time\_steps, double\* max\_time, size\_t\* ring\_assigned, double\* assigned\_angle, size\_t\* ring\_bodies,

double\* ring\_radius, double\* ring\_veloc, size\_t\* target)

{

// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ //

//

// Reads data from input file.

//

// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ //

FILE\* f = fopen(filename, "r");

if (f == NULL)

{

printf("Could not find config file! \"%s\"\n", filename);

return;

}

char\* buffer = NULL;

size\_t lines\_read = 0;

size\_t size;

getline(&buffer, &size, f);

size\_t i = 0;

while (getline(&buffer, &size, f) != EOF)

{

char\* t = strtok(buffer, ",");

get\_argzu(input\_bodies);

get\_args(input\_filename);

get\_argi(prec);

get\_args(output\_filename);

get\_argi(time\_steps);

get\_argf(max\_time);

get\_argzu(ring\_assigned);

get\_argf(assigned\_angle);

get\_argzu(ring\_bodies);

get\_argf(ring\_radius);

get\_argf(ring\_veloc);

get\_argzu(target);

fflush(stdout);

if (i >= 1)

{

exit(1);

}

i++;

}

/\*

if (input\_filename != NULL)

{

char temp\_filename[512];

sprintf(temp\_filename, "%s.csv", input\_filename);

\*input\_filename = temp\_filename;

}

\*/

fclose(f);

}

void readInputArgs(int argc, char\*\* argv, size\_t\* input\_bodies, char\*\* input\_filename, int\* prec, char\*\* output\_filename, int\* time\_steps, double\* max\_time,

size\_t\* ring\_assigned, double\* assigned\_angle, size\_t\* ring\_n\_bodies, double\* ring\_radius, double\* ring\_veloc, size\_t\* target)

{

int c;

opterr = 0;

while ((c = getopt(argc, argv, "n:i:p:o:t:m:a:q:e:r:v:")) != -1)

{

switch (c)

{

case 'n': \*input\_bodies = atoi(optarg); break;

case 'i': \*input\_filename = optarg; break;

case 'p': \*prec = atoi(optarg); break;

case 'o': \*output\_filename = optarg; break;

case 't': \*time\_steps = atoi(optarg); break;

case 'm': \*max\_time = atof(optarg); break;

case 'a': \*ring\_assigned = atoi(optarg); break;

case 'q': \*assigned\_angle = atof(optarg);break;

case 'e': \*ring\_n\_bodies = atoi(optarg); break;

case 'r': \*ring\_radius = atof(optarg); break;

case 'v': \*ring\_veloc = atof(optarg); break;

case 'x': \*target = atoi(optarg); break;

}

}

/\*

if (input\_filename != NULL)

{

char temp\_filename[512];

sprintf(temp\_filename, "%s.csv", input\_filename);

\*input\_filename = temp\_filename;

}

\*/

}

void readFile(char\* filename, double\* mass, double\* radius, double\* posit, double\* veloc, size\_t num\_bodies, size\_t\* bodies\_read)

{

// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ //

//

// Reads data from input file.

//

// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ //

FILE\* f = fopen(filename, "r");

if (f == NULL)

{

printf("Could not find input file! \"%s\"\n", filename);

return;

}

char\* buffer = NULL;

size\_t lines\_read = 0;

size\_t size;

getline(&buffer, &size, f);

size\_t i = 0;

while (getline(&buffer, &size, f) != EOF)

{

char\* t = strtok(buffer, ",");

get\_arg(&mass[i]);

get\_arg(&radius[i]);

get\_arg(&posit[3\*i]);

get\_arg(&posit[(3\*i)+1]);

get\_arg(&posit[(3\*i)+2]);

get\_arg(&veloc[3\*i]);

get\_arg(&veloc[(3\*i)+1]);

get\_arg(&veloc[(3\*i)+2]);

if (i >= num\_bodies)

{

fprintf(stderr, "File bigger than num bodies!\n");

exit(1);

}

i++;

}

\*bodies\_read = i;

fclose(f);

}

double calcAbs(double\* vector)

{

// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ //

//

// Returns absoloute magnitude of 3d vector.

//

// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ //

return sqrt(vector[0]\*vector[0] + vector[1]\*vector[1] + vector[2]\*vector[2]);

}

void Arange(size\_t length, double\* array)

{

for (size\_t i = 0; i < (length); ++i) { array[i] = i; }

}

void calcMax(double\* array, size\_t array\_height, size\_t array\_width, size\_t array\_max\_width, double\* max)

{

for (size\_t i = 0; i < array\_width; ++i)

{

double temp\_max = array[0];

for (size\_t t = 0; t < array\_height; ++t)

{

if (array[t\*array\_max\_width + i] > temp\_max)

{

temp\_max = array[t\*array\_max\_width + i];

}

}

max[i] = temp\_max;

}

}

void calcMin(double\* array, size\_t array\_height, size\_t array\_width, size\_t array\_max\_width, double\* max)

{

for (size\_t i = 0; i < array\_width; ++i)

{

double temp\_max = array[0];

for (size\_t t = 0; t < array\_height; ++t)

{

if (array[t\*array\_max\_width + i] < temp\_max)

{

temp\_max = array[t\*array\_max\_width + i];

}

}

max[i] = temp\_max;

}

}

void calcDifference(double\* final, double\* initial, double\* vector)

{

for (size\_t d = 0; d < 3; ++d) { vector[d] = final[d] - initial[d]; }

}

void calcDisplcaement(double\* posit, double\* n\_posit, size\_t target, size\_t num\_n\_bodies, size\_t max\_n\_bodies, size\_t max\_bodies, size\_t start\_time\_steps, size\_t end\_time\_steps, double\* displace)

{

for (size\_t t = start\_time\_steps; t < (end\_time\_steps + start\_time\_steps); ++t)

{

for (size\_t i = 0; i < num\_n\_bodies ; ++i)

{

double difference[3];

calcDifference(&n\_posit[t\*max\_n\_bodies\*3 + (i\*3)], &posit[t\*max\_bodies\*3 + (target\*3)], difference);

displace[t\*max\_n\_bodies + i] = calcAbs(&difference[0]);

}

}

}

void calcGravAccel(double\* affected\_posit, double\* effector\_posit, double\* effector\_mass, int affected\_no, int num\_effector, double\* accel)

{

// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ //

//

// Returns (accel) gravitational acceleration at point (affected\_posit) given number (num\_effector) of graviational attractors at an array of positions (effector\_postit) with masses (effector\_mass).

// If affected object is also and effector than its number in the position array (effector\_posit), should be given as (affected\_no), otherwise set affected\_no to -1.

//

// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ //

// Assigning current position to variables:

for (size\_t j = 0; j < num\_effector; ++j)

{

if (affected\_no == j) continue; // <-- If comparing body to itself iteration skips to prevent devide by 0 errors.

// Calculating diplacement and assigning to variables:

double displace[3];

calcDifference(&effector\_posit[(j\*3)], affected\_posit, displace );

// Calculating magnitude of gravitational acceleration:

double grav\_mag = effector\_mass[j]/(calcAbs(displace)\*calcAbs(displace)\*calcAbs(displace));

// Calculating gravitational acelleration vector:

for (size\_t d = 0; d < 3; ++d) { accel[d] += grav\_mag\*displace[d]; }

}

}

void calcAllGravAccel(double\* affected\_posit, double\* effector\_posit, double\* effector\_mass, bool same, int num\_affected, int num\_effector, double\* affected\_accel)

{

// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ //

//

// Returns (accel) gravitational acceleration at array of points (affected\_posit) given number (num\_effector) of graviational attractors at an array of positions (effector\_postit) with masses (effector\_mass).

// If affected objects are also and effector than (same) should be set to 1, elsewize 0.

//

// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ //

for (size\_t i = 0; i < num\_affected; ++i) { calcGravAccel(&affected\_posit[(3\*i) + 0], effector\_posit, effector\_mass, i\*same - (1 - same) , num\_effector, &affected\_accel[(3\*i)]); }

}

void calcGravVerlet(size\_t current\_time, double\* posit, double\* veloc, double\* accel, double\* effector\_mass, double\* effector\_posit, double dt, int num\_affected, int num\_effector, int max\_affected, int max\_effector, bool same)

{

// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ //

//

// Returns (posit), acceleration (veloc), velocity (acell) of (num\_affected) objects at (posit) after one time interval duration dt, given graviitational attraction between objects and num\_effector) objects with mass (mass\_effector)

// at positions (pefffector\_posit).

//

// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ //

for (size\_t i = 0; i < num\_affected; ++i)

{

// ~~ Saving old acceleration and resetting to 0:

double prev\_accel[3];

for (size\_t d = 0; d < 3; ++d) { prev\_accel[d] = accel[(3\*i) + d]; }

accel[(3\*i) + 0] = 0, accel[(3\*i) + 1] = 0, accel[(3\*i) + 2] = 0;

// ~~~~ Calculating new position:

for (size\_t d = 0; d < 3; ++d) { posit[(current\_time\*max\_affected\*3) + (i\*3) + d] = posit[((current\_time - 1)\*max\_affected\*3) + (i\*3) + d] + veloc[(3\*i) + d]\*dt + prev\_accel[d]\*dt\*dt\*0.5; }

// ~~~~ Calculating new acceleration:

calcGravAccel(&posit[((current\_time\*max\_affected\*3)) + (i\*3)], &effector\_posit[(current\_time - 1)\*max\_effector\*3], effector\_mass, i\*same - (1 - same), num\_effector, &accel[(i\*3)]); //

// ~~~~ Calcualting new velocity:

for (size\_t d = 0; d < 3; ++d) { veloc[(3\*i) + d] += (prev\_accel[d] + accel[(3\*i) + d])\*0.5\*dt; }

}

}

void calcRotatedVector(double\* init\_vector, double\* centre, double\* rotation, double\* fin\_vector)

{

// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ //

//

// Returns (fin\_vector) the postion of a 3d vector after rotation (rotation) around a point (centre).

//

// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ //

//Assigning values to variables for ease of use:

double x = init\_vector[0], y = init\_vector[1], z = init\_vector[2];

double a = centre[0], b = centre[1], c = centre[2];

double u = rotation[0], v = rotation[1], w = rotation[2], angle = rotation[3];

double cos\_angle = cos((2.\*M\_PI/360.) \* angle); double sin\_angle = sin((2.\*M\_PI/360.) \* angle);

//Calculating rotation:

fin\_vector[0] = (a\*(v\*v + w\*w) - u\*(b\*v + c\*w - u\*x - v\*y - w\*z))\*(1-cos\_angle) + x\*cos\_angle + (- c\*v + b\*w - w\*y + v\*z)\*sin\_angle;

fin\_vector[1] = (b\*(u\*u + w\*w) - v\*(a\*u + c\*w - u\*x - v\*y - w\*z))\*(1-cos\_angle) + y\*cos\_angle + ( c\*u - a\*w + w\*x - u\*z)\*sin\_angle;

fin\_vector[2] = (c\*(u\*u + v\*v) - w\*(a\*u + b\*v - u\*x - v\*y - w\*z))\*(1-cos\_angle) + z\*cos\_angle + (- b\*u + a\*v - v\*x + u\*y)\*sin\_angle;

}

void rotateBody(double\* posit, double\* veloc, double\* accel, double\* effector\_posit, double\* effector\_mass, double\* centre, double\* rotation, size\_t body\_no, size\_t max\_affected, size\_t max\_effector, size\_t num\_effector,

size\_t current\_time, bool same)

{

calcRotatedVector(&posit[(current\_time\*max\_affected\*3 + body\_no\*3)], centre, rotation, &posit[(current\_time\*max\_affected\*3 + body\_no\*3)]);

calcRotatedVector(&veloc[(body\_no\*3)], centre, rotation, &veloc[(body\_no\*3)]);

// Calculating initial acceleration:

calcGravAccel(&posit[3\*max\_affected\*current\_time + 3\*body\_no], &effector\_posit[3\*max\_effector\*current\_time], effector\_mass, body\_no\*same - (1 - same), num\_effector, &accel[3\*body\_no]);

}

void calcPolygonVertices(double\* centre, double\* rotation, double radius, int no\_of\_vertices, double\* vertices\_positions)

{

// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ //

//

// Returns (vertices\_position) 3d vector positions of the vertices of a regular polygon with radius (radius) and number of sides (no\_of\_vertices),

// centered around a point (centre) with rotation (rotation).

//

// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ //

for (size\_t i = 0; i < (no\_of\_vertices+1); ++i)

{

vertices\_positions[(i\*3) + 0] = centre[0] + cos((2\*M\_PI)/no\_of\_vertices\*i)\*radius;

vertices\_positions[(i\*3) + 1] = centre[1] + sin((2\*M\_PI)/no\_of\_vertices\*i)\*radius;

vertices\_positions[(i\*3) + 2] = centre[2];

calcRotatedVector(&vertices\_positions[i\*3], centre, rotation, &vertices\_positions[(i\*3)]);

}

}

void calcSphereRandPoints(double\* centre, double radius, int no\_of\_vertices, double\* vertices\_positions)

{

for (int i = 0; i < no\_of\_vertices; ++i)

{

for (size\_t d = 0; d < 3; ++d) { vertices\_positions[3\*i + d] = (double)rand\_64()/(double)RAND\_64\_MAX;}

double mag = calcAbs(&vertices\_positions[3\*i]);

for (size\_t d = 0; d < 3; ++d) { vertices\_positions[3\*i + d] /= mag; vertices\_positions[3\*i + d] \*= radius; vertices\_positions[3\*i + d] += centre[d]; }

}

}

void calcVelocityFromPoint(double\* posit, double\* centre, double\* init\_veloc, double extra\_veloc, int num\_n\_bodies, double\* veloc)

{

// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ //

//

// Returns (veloc) velocities of an array of vectors (posit) of number (num\_n\_bodies) given a constant velocity (extra\_veloc) away from a point (centre) plus an initial

// velocity (init\_veloc)

//

// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ //

for (size\_t i = 0; i < (num\_n\_bodies+1); ++i)

{

// Calculating diplacement from point and assigning to variables:

double displace[3];

calcDifference(&posit[(i\*3)], centre, displace);

// Calculating magnetude of diplacement:

double mag = calcAbs(displace);

// Assigning new velocity:======================

for (size\_t d = 0; d < 3; ++d) { veloc[(i\*3) + d] = init\_veloc[d] + extra\_veloc\*(displace[d]/mag); }

}

}

void addBodiesFromInputFile(double input\_bodies, char\* input\_filename, size\_t\* current\_bodies, size\_t\* effector\_bodies, size\_t current\_time, size\_t max\_bodies, size\_t max\_n\_bodies, double\* mass, double\* radius, double\* posit,

double\* veloc, double\* accel, double\* effector\_posit, double\* effector\_mass, bool same)

{

size\_t val\_current\_bodies = \*current\_bodies;

if (input\_bodies <= max\_bodies - val\_current\_bodies)

{

size\_t bodies\_read = 0;

if (input\_filename != NULL)

{

readFile(input\_filename, &mass[val\_current\_bodies], &radius[val\_current\_bodies], &posit[3\*max\_bodies\*current\_time + 3\*val\_current\_bodies], &veloc[3\*val\_current\_bodies], input\_bodies, &bodies\_read);

printf("Read %d bodies.\n", bodies\_read);

\*current\_bodies += input\_bodies;

// Putting mass in units of G:

for (size\_t i = 0; i < input\_bodies; ++i) { mass[i] = mass[i]\*G; }

// Calculating initial acceleration:

calcAllGravAccel(&posit[3\*max\_bodies\*current\_time + 3\*val\_current\_bodies], &effector\_posit[3\*max\_n\_bodies\*current\_time], effector\_mass, same, \*current\_bodies, \*effector\_bodies, &accel[3\*val\_current\_bodies]);

}

else { printf("Could not find input file!\n"); }

}

else { printf("Max bodies reached, unable to add! Total bodies: %d, Max Bodies: %d, Trying to add: %d. \n", current\_bodies, max\_bodies, input\_bodies); }

}

void addBodiesInRingAroundObject(int ring\_bodies, double ring\_radius, double\* rotation, double ring\_veloc, double\* object\_posit, double\* inital\_veloc, size\_t\* current\_bodies, size\_t\* effector\_bodies, size\_t current\_time,

size\_t max\_bodies, size\_t max\_n\_bodies, double\* posit, double\* veloc, double\* accel, double\* effector\_posit, double\* effector\_mass, bool same)

{

size\_t val\_current\_bodies = \*current\_bodies;

size\_t val\_effector\_bodies = \*effector\_bodies;

if (ring\_bodies <= max\_n\_bodies - \*current\_bodies)

{

calcPolygonVertices(object\_posit, rotation, ring\_radius, ring\_bodies, &posit[3\*max\_n\_bodies\*current\_time + 3\*val\_current\_bodies]);

calcVelocityFromPoint(posit, object\_posit, inital\_veloc, ring\_veloc, ring\_bodies, &veloc[3\*val\_current\_bodies]);

\*current\_bodies += ring\_bodies;

// Calculating initial acceleration:

calcAllGravAccel(&posit[3\*max\_bodies\*current\_time + 3\*val\_current\_bodies], &effector\_posit[3\*max\_n\_bodies\*current\_time], effector\_mass, same, \*current\_bodies, \*effector\_bodies, &accel[3\*val\_current\_bodies]);

}

else { printf("Max bodies reached, unable to add! Total bodies: %d, Max Bodies: %d, Trying to add: %d. \n", current\_bodies, max\_bodies, ring\_bodies); }

}

void addBodiesInRandSphereAroundObject(int sphere\_bodies, double sphere\_radius, double sphere\_veloc, double\* object\_posit, double\* inital\_veloc, size\_t\* current\_bodies, size\_t\* effector\_bodies, size\_t current\_time,

size\_t max\_bodies, size\_t max\_n\_bodies, double\* posit, double\* veloc, double\* accel, double\* effector\_posit, double\* effector\_mass, bool same)

{

size\_t val\_current\_bodies = \*current\_bodies;

size\_t val\_effector\_bodies = \*effector\_bodies;

if (sphere\_bodies <= max\_n\_bodies - \*current\_bodies)

{

calcSphereRandPoints(object\_posit, sphere\_radius, sphere\_bodies, &posit[3\*max\_n\_bodies\*current\_time + 3\*val\_current\_bodies]);

calcVelocityFromPoint(posit, object\_posit, inital\_veloc, sphere\_veloc, sphere\_bodies, &veloc[3\*val\_current\_bodies]);

\*current\_bodies += sphere\_bodies;

//Randomise Velocities

for (size\_t i = 0; i < sphere\_bodies; ++i)

{

double mag = calcAbs(&veloc[3\*i]);

for (size\_t d = 0; d < 3; ++d) { veloc[3\*i + d] += 3\*(veloc[3\*i + d]/mag)\*sphere\_veloc\*(double)rand\_64()/(double)RAND\_64\_MAX;}

}

// Calculating initial acceleration:

calcAllGravAccel(&posit[3\*max\_bodies\*current\_time + 3\*val\_current\_bodies], &effector\_posit[3\*max\_n\_bodies\*current\_time], effector\_mass, same, \*current\_bodies, \*effector\_bodies, &accel[3\*val\_current\_bodies]);

}

else { printf("Max bodies reached, unable to add! Total bodies: %d, Max Bodies: %d, Trying to add: %d. \n", current\_bodies, max\_bodies, sphere\_bodies); }

}

void runSim(size\_t time\_steps, double run\_time, size\_t max\_time\_steps, size\_t\* current\_time, double\* posit, double\* n\_posit, double\* veloc, double\* n\_veloc, double\* accel, double\* n\_accel,

double\* effector\_mass, double\* effector\_posit, int current\_bodies, int current\_n\_bodies, int max\_bodies, int max\_n\_bodies)

{

//Looping over timesteps using velocity verlet intergation:

if ((time\_steps + \*current\_time) > max\_time\_steps)

{

printf("Timesteps exceeds max time steps! Current Timesteps: \n", current\_time);

fflush(stdout);

exit(1);

}

else

{

printf("Running from 0 - %fs, over %d timesteps, with %d bodies and %d ejecta...\n", run\_time, time\_steps, current\_bodies, current\_n\_bodies);

fflush(stdout);

}

//Calculate timestep:

double dt = ((run\_time)/(double)time\_steps);

\*current\_time += 1;

for (int t = \*current\_time; t < \*current\_time + time\_steps; ++t)

{

//Looping over gravitational bodies:

calcGravVerlet(t, posit, veloc, accel, effector\_mass, posit, dt, current\_bodies, current\_bodies, max\_bodies, max\_bodies, true);

//Looping over gravitationally negligable bodies:

calcGravVerlet(t, n\_posit, n\_veloc, n\_accel, effector\_mass, posit, dt, current\_n\_bodies, current\_bodies, max\_n\_bodies, max\_bodies, false);

}

\*current\_time += time\_steps;

//Ensuring loop isn't removed:

printf("%f %f %f\n", posit[(current\_n\_bodies\*3) + (0\*3) + 0], posit[(current\_n\_bodies\*3) + (0\*3) + 1], posit[(current\_n\_bodies\*3) + (0\*3) + 2]);

fflush(stdout);

printf("\nSimulation completed.\n");

}

void resetSim(size\_t max\_time\_steps, size\_t max\_bodies, size\_t max\_n\_bodies, size\_t\* current\_bodies, size\_t\* current\_n\_bodies, size\_t\* current\_time, double\* mass, double\* radius, double\* posit,

double\* n\_posit, double\* veloc, double\* n\_veloc, double\* accel, double\* n\_accel)

{

printf("Reseting simulation...\n");

\*current\_time = 0;

\*current\_bodies = 0;

\*current\_n\_bodies = 0;

for (int i = 0; i < max\_bodies\*max\_time\_steps\*3; ++i) { posit[i] = 0.0; }

for (int i = 0; i < max\_n\_bodies\*max\_time\_steps\*3; ++i) { n\_posit[i] = 0.0; }

for (int i = 0; i < max\_bodies\*3; ++i) { veloc[i] = 0.0; accel[i] = 0.0; }

for (int i = 0; i < max\_n\_bodies\*3; ++i) { n\_veloc[i] = 0.0; n\_accel[i] = 0.0; }

for (int i = 0; i < max\_bodies; ++i) { mass[i] = 0.0; radius[i] = 0.0;}

printf("Reset complete.\n");

}

void clearSim(size\_t max\_bodies, size\_t max\_n\_bodies, size\_t current\_time, size\_t\* current\_bodies, size\_t\* current\_n\_bodies, double\* mass, double\* radius, double\* posit, double\* n\_posit, double\* veloc,

double\* n\_veloc, double\* accel, double\* n\_accel)

{

printf("Removing all bodies from simulation...\n");

\*current\_bodies = 0;

\*current\_n\_bodies = 0;

for (int i = 0; i < max\_bodies\*3; ++i) { posit[(current\_time\*max\_bodies\*3) + i] = 0.0; }

for (int i = 0; i < max\_n\_bodies\*3; ++i) { n\_posit[(current\_time\*max\_n\_bodies\*3) + i] = 0.0; }

for (int i = 0; i < max\_bodies\*3; ++i) { veloc[i] = 0.0; accel[i] = 0.0; }

for (int i = 0; i < max\_n\_bodies\*3; ++i) { n\_veloc[i] = 0.0; n\_accel[i] = 0.0; }

for (int i = 0; i < max\_bodies; ++i) { mass[i] = 0.0; radius[i] = 0.0;}

printf("Bodies removed.\n");

}

void printPosit(char\* output\_filename, size\_t detail, int start\_time\_steps, int end\_time\_steps, int prec, double\* posit, double\* n\_posit, size\_t max\_bodies, size\_t max\_n\_bodies, size\_t current\_bodies, size\_t current\_n\_bodies, int print\_number)

{

int no\_lines = (end\_time\_steps - start\_time\_steps)/detail;

int start\_lines = start\_time\_steps/detail;

if (output\_filename != NULL)

{

char filename\_positions[512];

sprintf(filename\_positions, "%s\_posit\_%d.csv", output\_filename, print\_number);

printf("Printing positions to file: %s\n", filename\_positions);

FILE\* f = fopen(filename\_positions, "w");

for (size\_t t = start\_lines; t < (start\_lines + no\_lines); ++t)

{

for (size\_t i = 0; i < current\_bodies; ++i)

{

fprintf(f, "%.\*e,%.\*e,%.\*e,", prec, posit[(detail\*t\*max\_bodies\*3) + (i\*3) + 0], prec, posit[(detail\*t\*max\_bodies\*3) + (i\*3) + 1], prec, posit[(detail\*t\*max\_bodies\*3) + (i\*3) + 2]);

}

for (size\_t i = 0; i < current\_n\_bodies; ++i)

{

fprintf(f, "%.\*e,%.\*e,%.\*e,", prec, n\_posit[(detail\*t\*max\_n\_bodies\*3) + (i\*3) + 0], prec, n\_posit[(detail\*t\*max\_n\_bodies\*3) + (i\*3) + 1], prec, n\_posit[(detail\*t\*max\_n\_bodies\*3) + (i\*3) + 2] );

}

fprintf(f, "%s\n");

}

fclose(f);

printf("Printing positions completed. \n");

}

else printf("No output selected. Canceling printing. \n" );

}

void printSpecial(char\* output\_filename, size\_t detail, int start\_time\_steps, int end\_time\_steps, int prec, double\* origin\_displace, double\* target\_displace, size\_t max\_n\_bodies, size\_t current\_n\_bodies, int print\_number)

{

int no\_lines = (end\_time\_steps - start\_time\_steps)/detail;

int start\_lines = start\_time\_steps/detail;

if (output\_filename != NULL)

{

char filename\_displace[512];

sprintf(filename\_displace, "%s\_origin\_displace\_%d.csv", output\_filename, print\_number);

printf("Printing origin displacement to file: %s\n", filename\_displace);

FILE\* f = fopen(filename\_displace, "w");

for (size\_t t = start\_lines; t < (start\_lines + no\_lines); ++t)

{

for (size\_t i = 0; i < current\_n\_bodies; ++i)

{

fprintf(f, "%.\*e,", prec, origin\_displace[(detail\*t\*max\_n\_bodies) + i]);

}

fprintf(f, "%s\n");

}

fclose(f);

char filename\_displace\_2[512];

sprintf(filename\_displace\_2, "%s\_target\_displace\_%d.csv", output\_filename, print\_number);

printf("Printing target displacement to file: %s\n", filename\_displace\_2);

FILE\* f2 = fopen(filename\_displace\_2, "w");

for (size\_t t = start\_lines; t < (start\_lines + no\_lines); ++t)

{

for (size\_t i = 0; i < current\_n\_bodies; ++i)

{

fprintf(f2, "%.\*e,", prec, target\_displace[(detail\*t\*max\_n\_bodies) + i]);

}

fprintf(f2, "%s\n");

}

fclose(f2);

printf("Printing special completed. \n");

}

else printf("No output selected. Canceling printing. \n" );

}

void printMax(char\* output\_filename, char\* filename\_suffix, int start\_time\_steps, int end\_time\_steps, int prec, double\* array, size\_t max\_n\_bodies, size\_t current\_n\_bodies, int print\_number)

{

if (output\_filename != NULL)

{

char filename\_displace[512];

sprintf(filename\_displace, "%s\_%s\_%d.csv", output\_filename, filename\_suffix, print\_number);

printf("Printing to file: %s\n", filename\_displace);

FILE\* f = fopen(filename\_displace, "w");

for (size\_t t = start\_time\_steps; t < (start\_time\_steps + end\_time\_steps); ++t)

{

for (size\_t i = 0; i < current\_n\_bodies; ++i)

{

fprintf(f, "%.\*e,", prec, array[(t\*max\_n\_bodies) + i]);

}

fprintf(f, "%s\n");

}

}

else printf("No output selected. Canceling printing. \n" );

}

int main(int argc, char\*\* argv)

{

// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ VARIABLES ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ //

size\_t max\_bodies = 6;

size\_t current\_bodies = 0;

size\_t max\_n\_bodies = 36;

size\_t current\_n\_bodies = 0;

size\_t input\_bodies = 1;

char\* input\_filename = NULL;

int prec = 7;

int detail = 10;

char\* output\_filename = NULL;

size\_t current\_time = 0;

int max\_time\_steps = 4000000;

double run\_time = 100\*365.25\*24\*3600;

int time\_steps = 300000;

size\_t repeat = 40;

size\_t ring\_assigned = 1;

double assigned\_angle = 0;

size\_t ring\_n\_bodies = 0;

double ring\_radius = 621000000;

double ring\_veloc = 1000;

size\_t target = 3;

double veloc\_increase = 200;

//Reading config file:

readConfig("config.csv", &input\_bodies, &input\_filename, &prec, &output\_filename, &time\_steps, &run\_time, &ring\_assigned, &assigned\_angle, &ring\_n\_bodies, &ring\_radius,

&ring\_veloc, &target);

//Reading console arguments

readInputArgs(argc, argv, &input\_bodies, &input\_filename, &prec, &output\_filename, &time\_steps, &run\_time, &ring\_assigned, &assigned\_angle, &ring\_n\_bodies, &ring\_radius,

&ring\_veloc, &target);

// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ SETUP ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ //

//Creating z\_rotation and origin vector

double z\_rotation[4] = {0, 0, 1, assigned\_angle};

double origin[3] = {0,0,0};

//Allocating Memory for gravitational bodies

double\* mass = (double\*)malloc(sizeof(double)\*max\_bodies);

double\* radius = (double\*)malloc(sizeof(double)\*max\_bodies);

double\* posit = (double\*)malloc(sizeof(double)\*max\_bodies\*max\_time\_steps\*3);

double\* veloc = (double\*)calloc(max\_bodies\*3, sizeof(double));

double\* accel = (double\*)calloc(max\_bodies\*3, sizeof(double));

//Allocating Memory for non gravitation bodies

double\* n\_posit = (double\*)malloc(sizeof(double)\*max\_n\_bodies\*max\_time\_steps\*3);

double\* n\_veloc = (double\*)calloc(max\_n\_bodies\*3, sizeof(double));

double\* n\_accel = (double\*)calloc(max\_n\_bodies\*3, sizeof(double));

//Allocating Memory for Special arrays

double\* origin\_displace = (double\*)malloc(sizeof(double)\*max\_n\_bodies\*max\_time\_steps);

double\* target\_displace = (double\*)malloc(sizeof(double)\*max\_n\_bodies\*max\_time\_steps);

double\* max\_origin\_displace = (double\*)malloc(sizeof(double)\*max\_n\_bodies\*repeat);

double\* max\_target\_displace = (double\*)malloc(sizeof(double)\*max\_n\_bodies\*repeat);

//printf("Total memory allocated = %dMB\n", sizeof(double)\*(num\_bodies + num\_n\_bodies)\*((max\_time\_steps\*3 + 5)/1000000));

// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ SIMULATION ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ //

for (size\_t i = 0; i < repeat; ++i)

{

//Adding bodies read from input\_file:

z\_rotation[3] = (360\*(double)rand\_64()/(double)RAND\_64\_MAX); //< --- Rotaty buisness

addBodiesFromInputFile(input\_bodies, input\_filename, &current\_bodies, &current\_bodies, current\_time, max\_bodies, max\_n\_bodies, mass, radius, posit, veloc, accel, posit, mass, true);

//Rotate assigned planet and velocity around origin:

for (size\_t j = 0; j < current\_bodies; ++j)

{

z\_rotation[3] = (360\*(double)rand\_64()/(double)RAND\_64\_MAX);

rotateBody(posit, veloc, accel, posit, mass, origin, z\_rotation, j, max\_bodies, max\_bodies, current\_bodies, current\_time, true);

}

//Assigning ring positions and velocities:

addBodiesInRandSphereAroundObject(ring\_n\_bodies, ring\_radius, ring\_veloc, &posit[(ring\_assigned\*3)], &veloc[(ring\_assigned\*3)], &current\_n\_bodies, &current\_bodies,

current\_time, max\_bodies, max\_n\_bodies, n\_posit, n\_veloc, n\_accel, posit, mass, false);

printf("Angle = %f \n", z\_rotation[3]);

runSim(time\_steps, run\_time, max\_time\_steps, &current\_time, posit, n\_posit, veloc, n\_veloc, accel, n\_accel, mass, posit, current\_bodies, current\_n\_bodies, max\_bodies, max\_n\_bodies);

//calcDisplcaement(posit, n\_posit, 0, current\_n\_bodies, max\_n\_bodies, max\_bodies, 0, time\_steps, origin\_displace);

calcDisplcaement(posit, n\_posit, target, current\_n\_bodies, max\_n\_bodies, max\_bodies, 0, time\_steps, target\_displace);

//calcMax(origin\_displace, time\_steps, current\_n\_bodies, max\_n\_bodies, &max\_origin\_displace[i\*max\_n\_bodies]);

calcMin(target\_displace, time\_steps, current\_n\_bodies, max\_n\_bodies, &max\_target\_displace[i\*max\_n\_bodies]);

if (i != (repeat-1)) { resetSim(max\_time\_steps, max\_bodies, max\_n\_bodies, &current\_bodies, &current\_n\_bodies, &current\_time, mass, radius, posit, n\_posit, veloc, n\_veloc, accel, n\_accel); }

}

// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ PRINTING ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ //

// ~~~~~~~~~~ Printing to file ~~~~~~~~~~ //

//printMax(output\_filename, "\_max\_origin-veloc0-50", 0, repeat, prec, max\_origin\_displace, max\_n\_bodies, current\_n\_bodies, 1);

printMax(output\_filename, "\_max\_target-displace-sphere", 0, repeat, prec, max\_target\_displace, max\_n\_bodies, current\_n\_bodies, 105);

//printPosit(output\_filename, detail, 0, time\_steps, prec, posit, n\_posit, max\_bodies, max\_n\_bodies, current\_bodies, current\_n\_bodies, 1);

//printSpecial(output\_filename, detail, 0, time\_steps, prec, origin\_displace, target\_displace, max\_n\_bodies, current\_n\_bodies, 1);

printf("\nProgram terminated.\n");

return 0;

}