Computational Physics – Coding Report – Exercise 3

Caspar Fuller 1310006

Free-fall with fixed or varying drag

The aim of this program was to simulate the free-fall of an object from a determined height, investigating how the velocity and displacement changes with time during the fall. With the correct initial conditions the free-fall carried out by Felix Baumgartner can be simulated testing to see whether Newtonian mechanics accurately predict his maximum velocity. This is carried using the Euler’s method shown in equations (1) and (2).

(1)

; (2)

This solving method can be applied to the one-dimensional differential equation of motion found for vertical free-fall to give a scheme for finding the velocity and displacement of an object via a program at a certain time as seen in equations (3-5).

(3)

(4)

(5)

Analysis of the simulation can take place as well by computing the analytical values (equations 6+7) for the free-fall and comparing Euler’s method to these investigating the accuracy as the limits were changed.

(6)

(7)

The program written took user inputs for each variable, with conditions on each input ensuring the correct typecast was used. Equations 3-5 were then employed iteratively to find the velocity and displacement as t increased until the program detected the object had reached the ground. This results were written to a file that also stored the analytical results for the free-fall with the same inputted conditions.

The program was then extended in various ways. A main extension given to the user was the choice of using a constant drag or whether to vary it. A varying drag force more accurately represents free-fall from a great height as the air density thins. The variation in air density was taken as an exponential decay, with code written to change this value upon each iteration so it is consist with the current displacement. The program was instructed to ignore this however if the user had chosen a constant air density.

A further extension was that the user is prompted to give a percentage of terminal velocity from which the program will then calculate the time taken for the object to reach this. This could only take place during a constant drag force simulation, as the terminal velocity was calculated assuming a constant air density shown in equation 8.

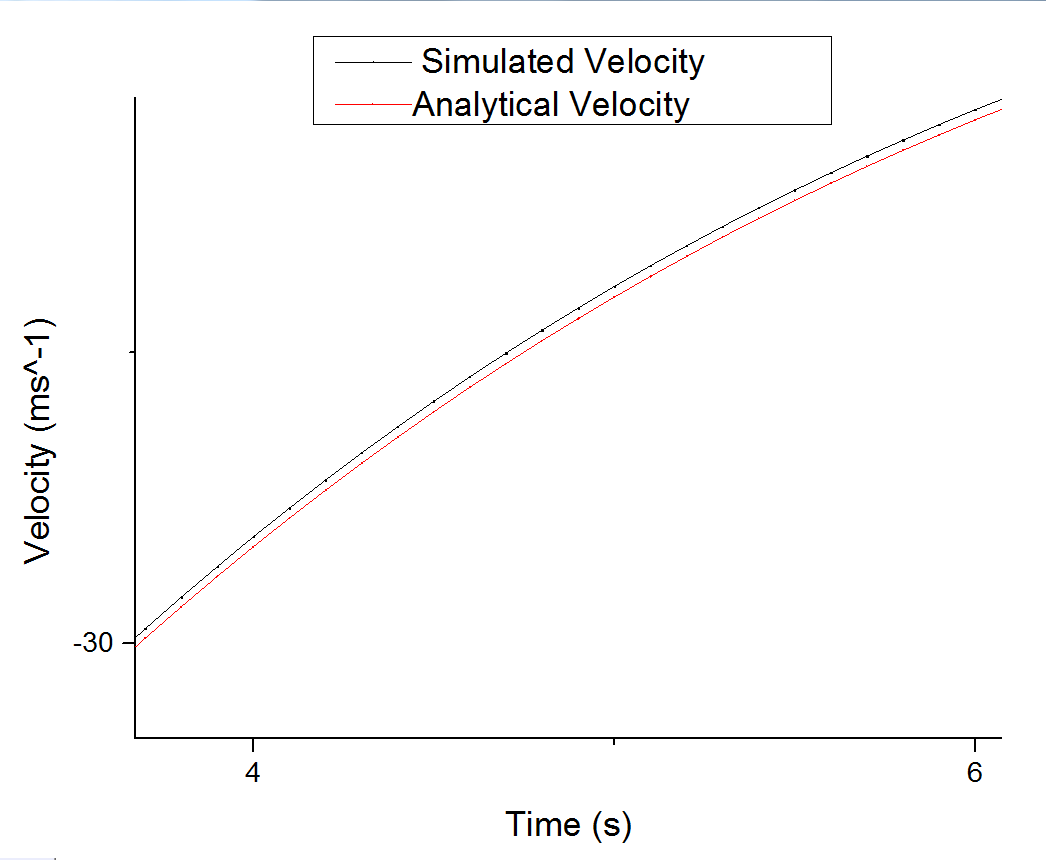
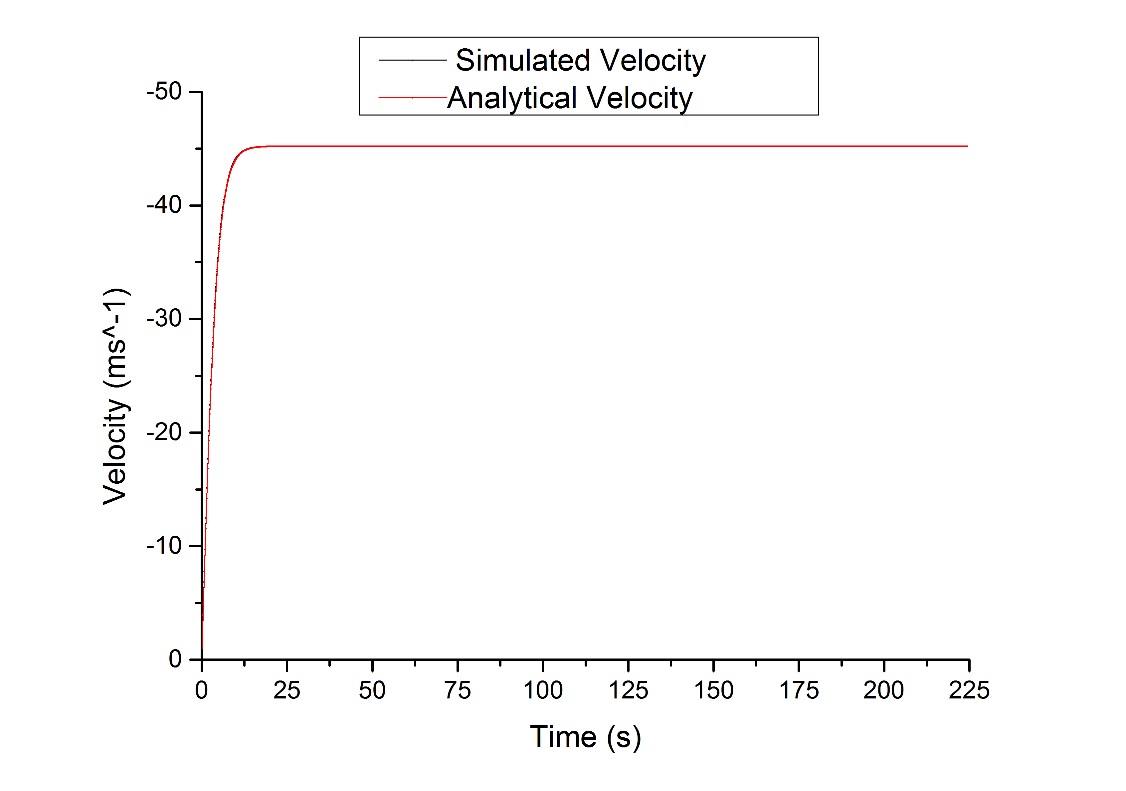
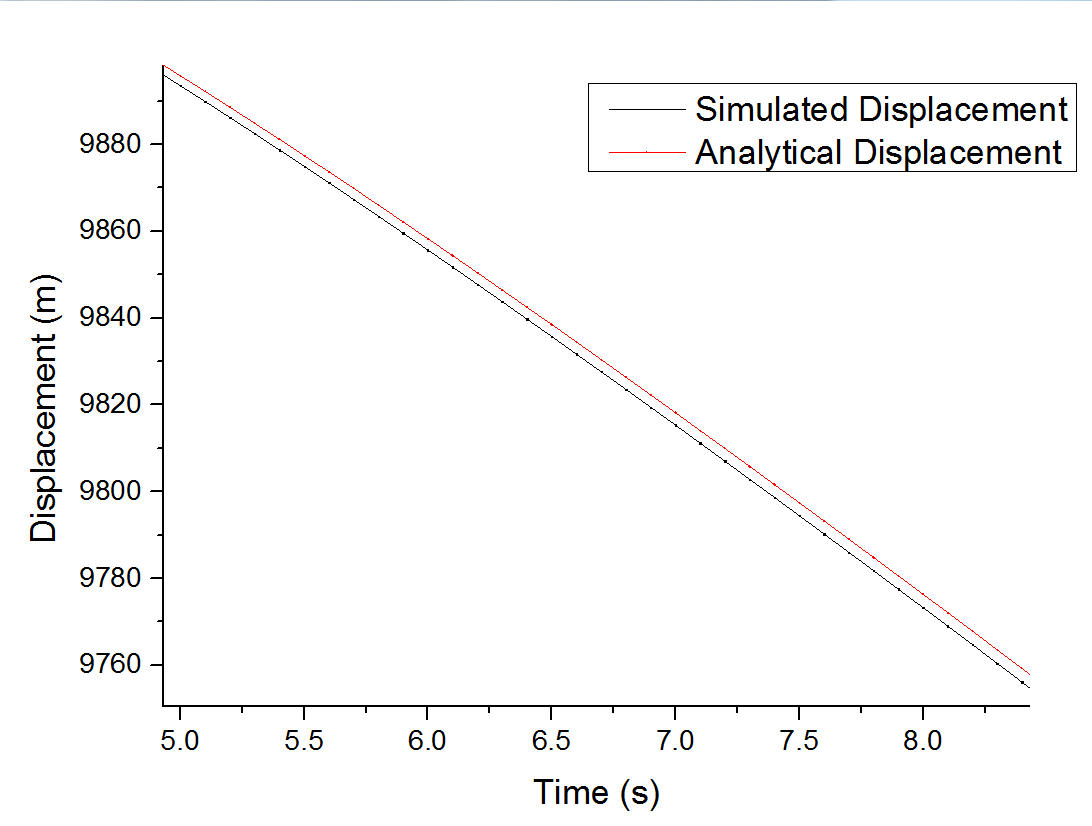
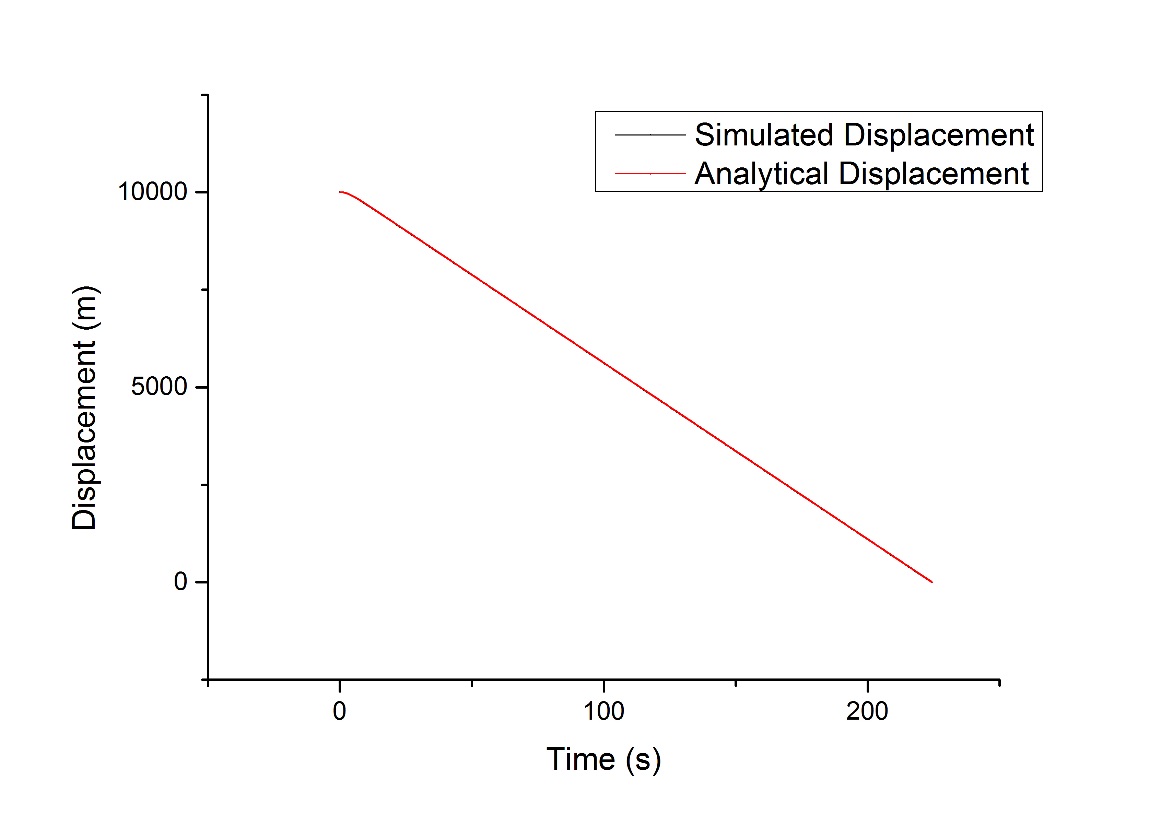
(8)

The program computes the percentage of this terminal velocity that was inputted and runs the iterations as usual. When the desired velocity has been reached there is a conditional statement that records the current value of time at this point, then increases an arbitrary counter ensuring the code does not run through the conditional statement again and overwriting this time.

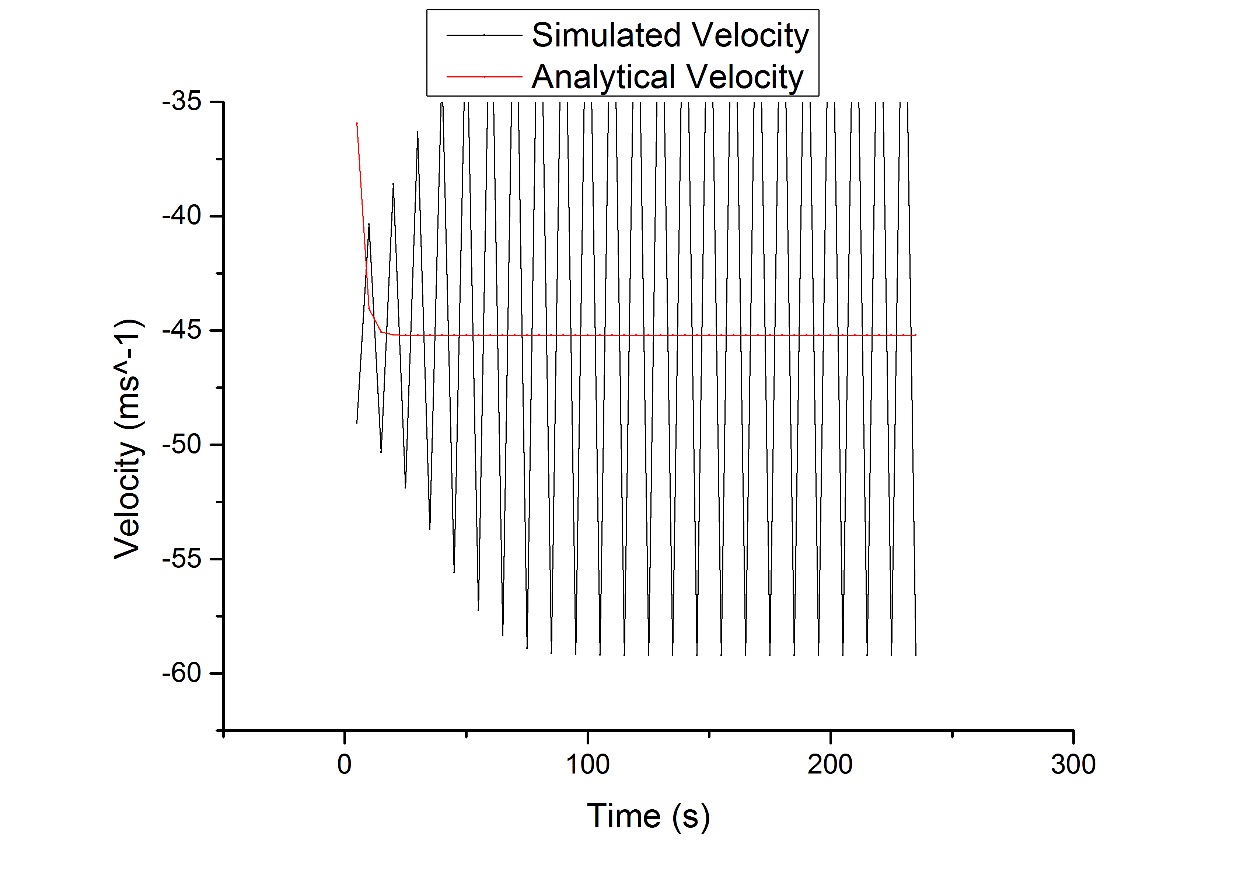
After initial testing a final extension was made to the code as it was found that by using iterative steps the final iteration would always occur below ground therefore not giving a precise time of impact. However the program could be made to account from that using the simple equation . The velocity was approximated by taking the average from the last two iterations and the displacement below ground was given by the program therefore it was possible to find an approximation for the time taken to travel that far. If this time was subtracted from the final time shown an accurate approximation can be given for impact time. To get the correct velocities to average a variable must be set storing the previous velocity at the start of each iteration.

The program was compared to the analytical results for various different values of Δt to see if there was any change in accuracy. All other initial conditions were held constant and a jump from 10,000 metres at constant air density was simulated. Figure 1 and 2 shows the difference between analytical and simulated values for a step size of 0.1 seconds. They are seen to be approximately accurate with differences in displacement seen to be held at a constant of about 3 metres. It is only upon zooming in on the graph can the difference be seen as shown on the insets on each figure.

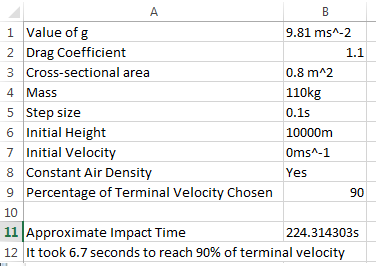
*Figure 1: Plot of displacement against time for the conditions shown in figure 4*



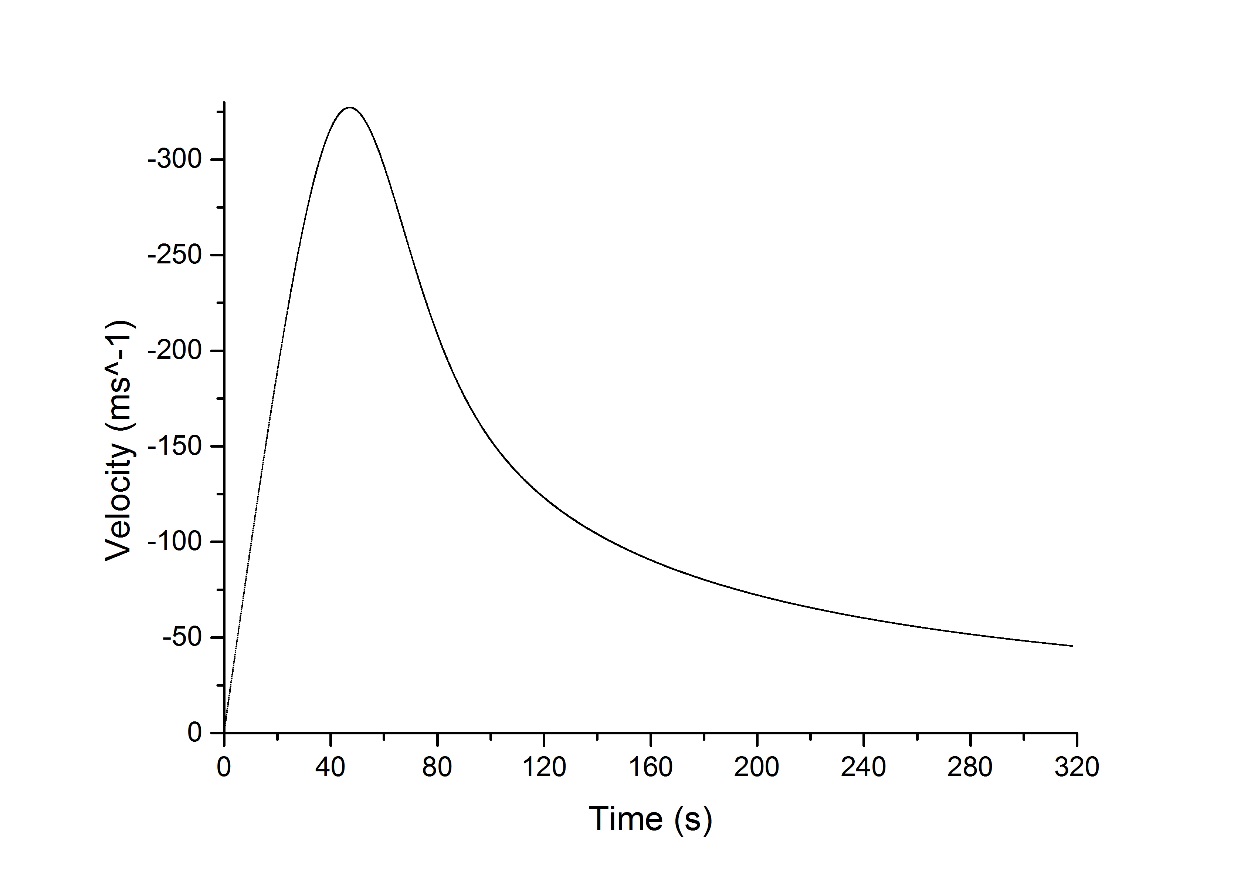
*Figure 2: Plot of velocity against time for the conditions shown in figure 4*

When the step size was decreased to smaller values of step size the results converged upon the analytical results. For example a step size of 0.001 seconds gave results correct to 3 decimal places, showing a high level of accuracy. However when the time interval was made larger, say 5 seconds, the limitations in the Euler method became very apparent. Figure 3 shows a comparison in simulated velocity with analytical velocity and the errors in the Euler method become clear here. Although the method is still correct however the error within it is found to be proportional to the step size chosen, therefore inaccurate results are returned.

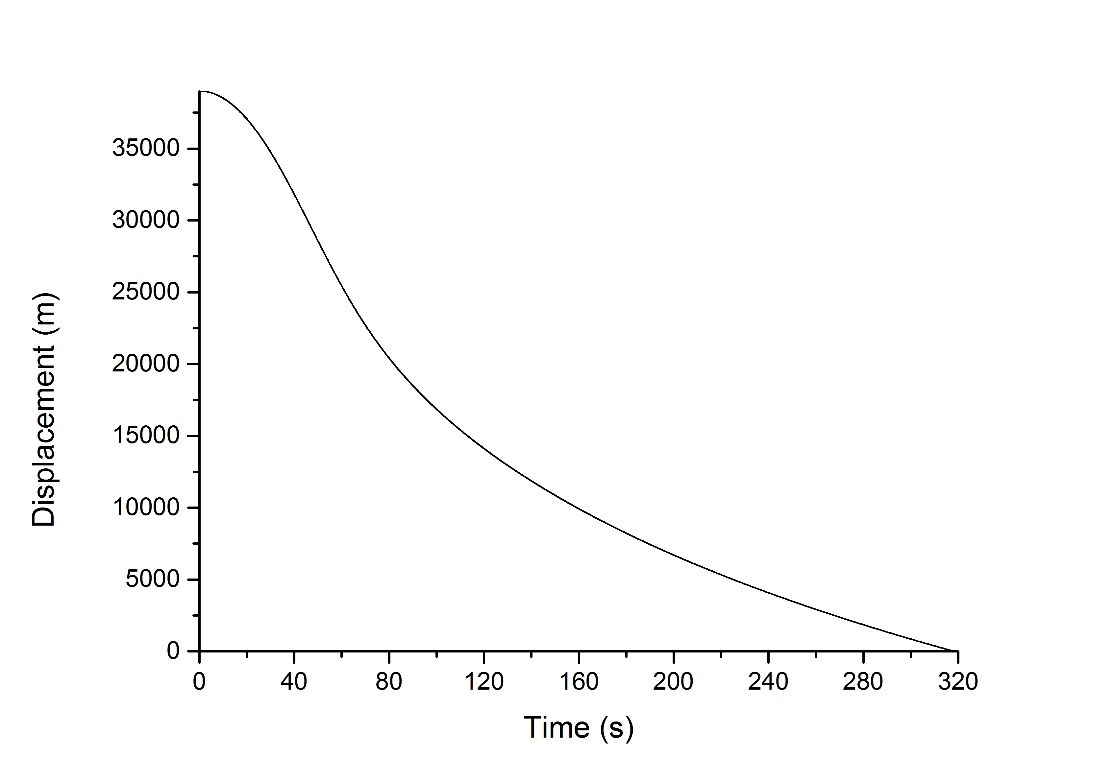
*Figure 3: Plot of velocity against time for a time step of 5 seconds*

To test whether the extensions made for impact time and the percentage of terminal velocity time the program was run with the conditions shown in figure 4. The results are also shown in figure 4, and correspond to results found with a calculator.

*Figure 4: Initial conditions for testing the program and varying the step size*

Finally the program was set up to model Felix Baumgartner’s free-fall jump. The initial conditions were found online and inputted and the program was run with varying air density to test whether he theoretically broke the sound barrier. A plot of his velocity with time is shown in figure 5 showing clearly the great velocity at higher altitudes which is reduced as the object returns to the thicker lower atmosphere and is slowed. This can also be deduced from figure 6 from the change in gradient on the curve showing a reduction in velocity. Analysing the data it was seen the simulation had a maximum velocity at 344.12m therefore it did break the sound barrier of 343 m.

*Figure 5: Plot of velocity against time for Felix Baumgartner’s 39,000m simulation free-fall*



*Figure 6*: *Plot of velocity against time for Felix Baumgartner’s 39,000m simulation free-fall*

One problem with the program was that during the impact time calculation the average of the last 2 velocities may not always lead to an accurate result. If the object has reached terminal velocity then the results will be accurate however if it is still accelerating to terminal velocity, this result will only be an estimate of the impact time. To calculate it more accurately a backwards Euler method could be employed resetting the last values used, then by decreasing the step size an impact time can be found correct to a desired accuracy.

Also it was found that if the simulation did not reach the percentage of terminal velocity inputted it would return a random value for how long it took to reach there. This was rectified with a conditional statement when the program was about to write the result to the file. If the counter set previously had not changed then the percentage of terminal velocity had not been reached, so the program skips this part.

A final problem was that although the air density was varied the value for g was kept constant at 9.81ms^-2. This value would change at altitude however when investigated it only changed by approximately 1% at 39,000m therefore this change could be ignored as it is negligible.

BLANK PAGE – PLEASE IGNORE

Report for problem 2 on next page.

Electric Fields and Potentials

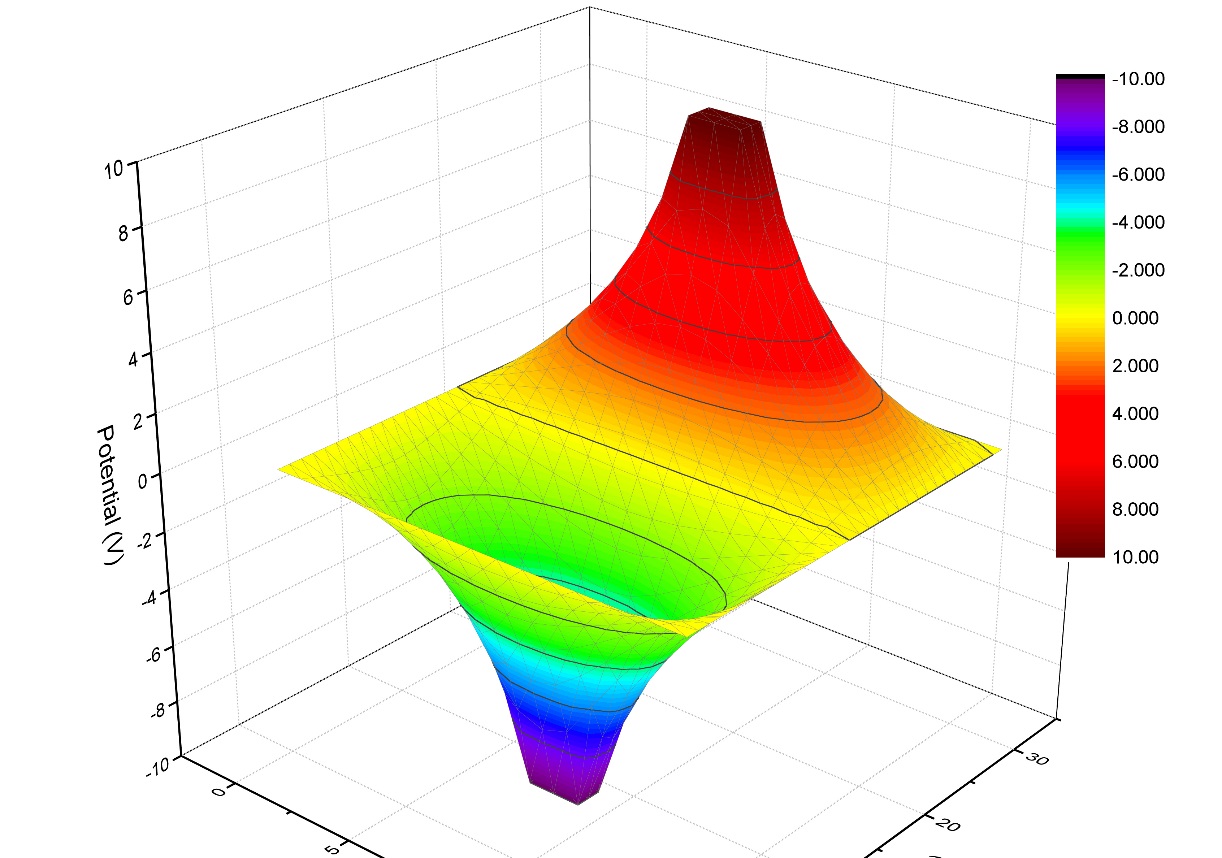
This problem was designed to find the electrostatic potential round two conductors and investigate the relationship between the electric field and the potential. The potential is found via a relaxation method shown in equation 9 for a 2-dimensional 2 core wire cross section shown in figure 7, this is then used to find the electric field around the wire. The relaxation equation used to find potential is shown below, the z term can be ignored as the situation is simplified by taking the cross-section of the wire:

(9)

Where *h* corresponds to a step size taken as 1 in this exercise. The wire was set up to have 2 cores held at a constant voltage and to have constantly zero potential outside the wire.

The program was set up so a 2D array represented figure 7, with each segment set up with a width of *h.* The array was initialised and the 2 core potentials were set as in the diagram. The relaxation method is then applied to the array and this method iterates over the whole array before resetting the core potentials and repeating. A second array was needed to fill the values from the iteration to allow the program to evolve. I wanted this method to repeat until it reached an accuracy of 0.001V on each point in the array. To ensure this accuracy, code was written to add a count to a dummy variable every time a difference greater than 0.001V was found between Vold and Vnew. This counter was reset to 0 after each full iteration and with the use of a conditional statement at the end of the iteration, if the count was still zero when the iteration had been completed the program could detect the required accuracy had been reached. An important note for the code here was to make sure the outer zero potential wall was included in the array as if it is omitted the potentials found at the edges of the wire are very inaccurate as this outer wall is assigned random values from the computer and therefore breaks the program. The results of each iteration were written to a text file where the evolution of the potential could be studied and a plot for the final potential could be found, shown in figure 8.

*Figure 7: Set-up of 2-core wire system*



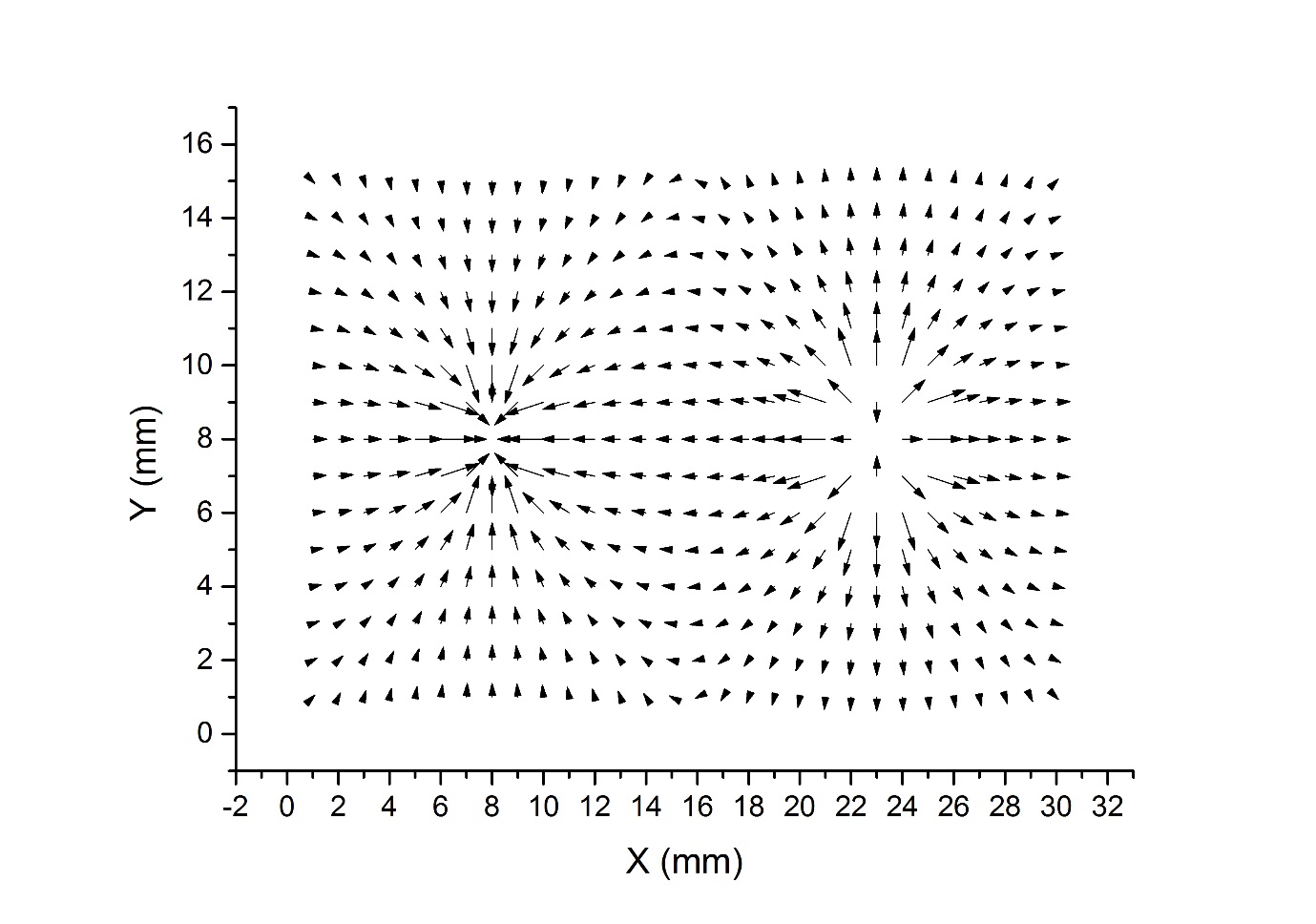
*Figure 8: 3D surface plot of the electrostatic potential within the cross-section of the wire.*

Next the field was evaluated for each point via the central difference formula this was done as the value for the electric field at a point is shown in equation 10.

(10)

From this an angle and magnitude could be found for the electric field at each point in the array allowing for a vector plot of the E-field to be created shown in figure 9. These results were also written to the same file shown at the bottom. Here it was found that if the field component in the x-direction was negative then the calculation to find the angle of the vector needs to be shifted by π to account for this.

This program seemed to run efficiently however it was all contained in the main function. Ideally it would be split into separate functions, for example, one for the relaxation method and one for the electric field. However I had problems with setting arrays as arguments for these functions. This would mean creating a new array in each function which is less space efficient. After investigation I found that if I had used a ‘struct’ function then a structure could have been set before the main and allows me to use the same arrays in separate functions so if I were to improve on my code this would be the first thing to do.



*Figure 9: Vector field plot of the electric field*