Computational Modelling: Week 4

Task 1

In this task, the path of an electron moving in the x-y plane under a uniform magnetic field, B = 1 T in the z-direction, was calculated. The initial conditions for the electron were

- At time t = 0 s, position (x, y, z) = (0, 0, 0) m
- Momentum, (p_x, p_y, p_z) = (1000, 0, 0) MeV
- Electron mass, m = 0.511 MeV, negative charge

Task 1.1: Initial Calculations

Firstly, some simple calculations were performed in order to better understand the task. The speed of the electron, the bending radius and the time taken for the electron to turn a full circle were calculated.

The relativistic speed (in units of c), β, is given by,

$$\beta = \frac{\bar{p}}{E_{tot}}, \qquad \text{Equation 1}$$

where E_{tot} is the total relativistic energy,

$$E_{tot} = \sqrt{m^2 + p_x^2 + p_y^2 + p_z^2}. \qquad \text{Equation 2}$$

Combining Equations 1 and 2 gave an expression for β,

$$\beta = \frac{\bar{p}}{\sqrt{m^2+p_x^2+p_y^2+p_z^2}}. \qquad \text{Equation 3}$$

This gave β = 0.9999998674c ≈ 1c = 3 x 10^8 m/s

The radius of curvature of a particle travelling in a magnetic field, B is,

$$R = \frac{p\ [MeV]}{300 \cdot B\ [T]}, \qquad \text{Equation 4}$$

which gave R = 3.33 m.

Using R, the time taken for the electron to turn a full circle, T_{turn}, was calculated,

$$T_{turn} = \frac{2\pi \cdot R}{c} = 6.91 \times 10^{-8} s. \qquad \text{Equation 5}$$

Task 1.2: Numerical solution

The classes `RunSimulation.java`, `Particle.java`, `particleTracker.java` and `Track.java` were loaded onto NetBeans and the programs run. Figure 1 shows the x-y scatter plot obtained using the unedited programs where the cut of time t_{max} = 1 x 10^{-7} s and the number of steps N = 100. It can be observed that the particle does not end its track in the same position as it starts (0, 0, 0) m. This was because t_{max} was greater than T_{turn}, so the particle travelled more than one cycle. Furthermore, the Euler integration method always under predicts the curvature of the path, with a global error proportional to Δt. Thus, Figure 1 shows the path spiralling outwards. Increasing N would increase the accuracy and bring the start and end points closer together. The Runge-Kutta 4th order integration method has global error proportional to $(\Delta t)^4$, making it more accurate. I used both of these algorithms and compared the relationship between N and the separation between the start and end points.

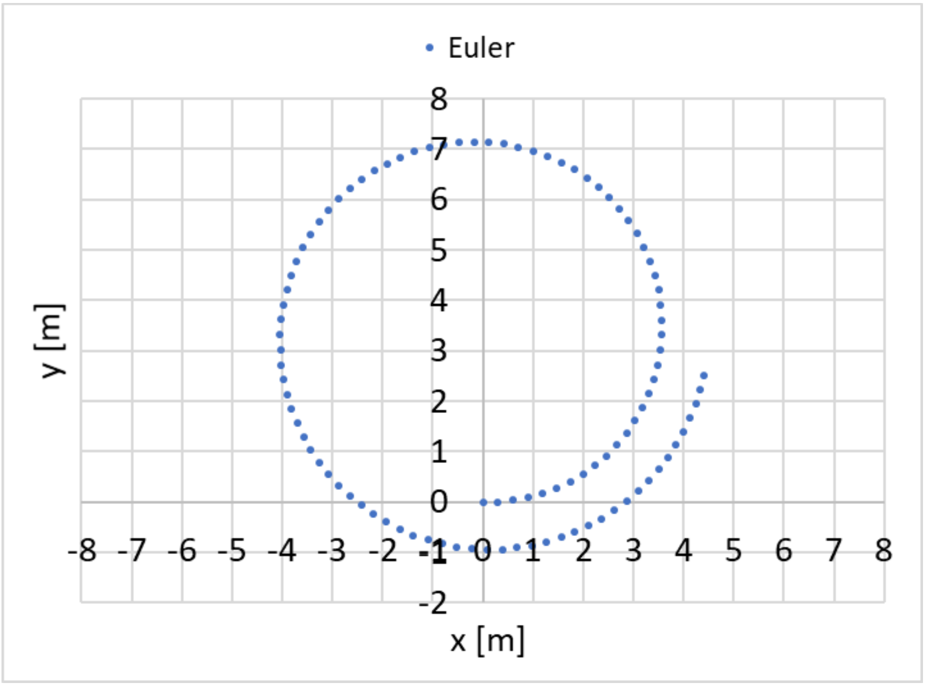

Figure 1: x-y scatter plot of the csv file produced using the unedited code with N = 100 and $t_{max} = 1 \times 10^{-7}$ s.

Firstly, I changed the cut off time to $T_{turn} = 6.91 \times 10^{-8}$ s. I then added a for loop which calculated the separation between the start position (0, 0, 0) m, and end position the particle for $100 \leq N \leq 1{,}000{,}000$. Each N and the corresponding separation (`distance`) were saved to `distances.csv`. The additional code is shown below:

```
//the array of N vales for which the separation between the start and end points ('
distance') will be calculated:
int [] N = {100,200,300,400,500,600,700,800,900, 1000, 5000, 10000, 100000, 200000,
 400000, 600000, 800000, 1000000};
double[] distance = new double[N.length]; //array to store the distances, same leng
th as N[]

for (int i = 0; i < N.length; i++){
    //ParticleTracker tracker = new ParticleTracker(p, time, nsteps, useRungeKutta4
);
    ParticleTracker tracker2 = new ParticleTracker(p, time, N[i], useRungeKutta4);

    Particle stunning = tracker2.track(); //the particle
    double d = Math.sqrt(stunning.x*stunning.x + stunning.y*stunning.y); //calculat
es the distance between the start position (0,0) and end position (x,y)
    distance[i] = d; //saves the distance to the array
}

PrintWriter outputFile; //saving the distance vs N data as a csv
try {
    outputFile = new PrintWriter("distances.csv");
} catch (IOException e) {
    System.err.println("Failed to open file distances.csv Track data was not saved.
");
```

```
    return;
}

//A loop which writes the contents of the arrays to disk, one number at a time
outputFile.println("N, d [m]");
for (int n = 0; n < N.length; n++) {
    outputFile.println(N[n] + "," + distance[n]); //comma separated values
}
outputFile.close(); // close the output file
```

Figure 2 shows the outcome using the Euler and Runge-Kutta 4th order algorithms. To use the Runge-Kutta 4th order integration method, `useRungeKutta4` was set to `True`.

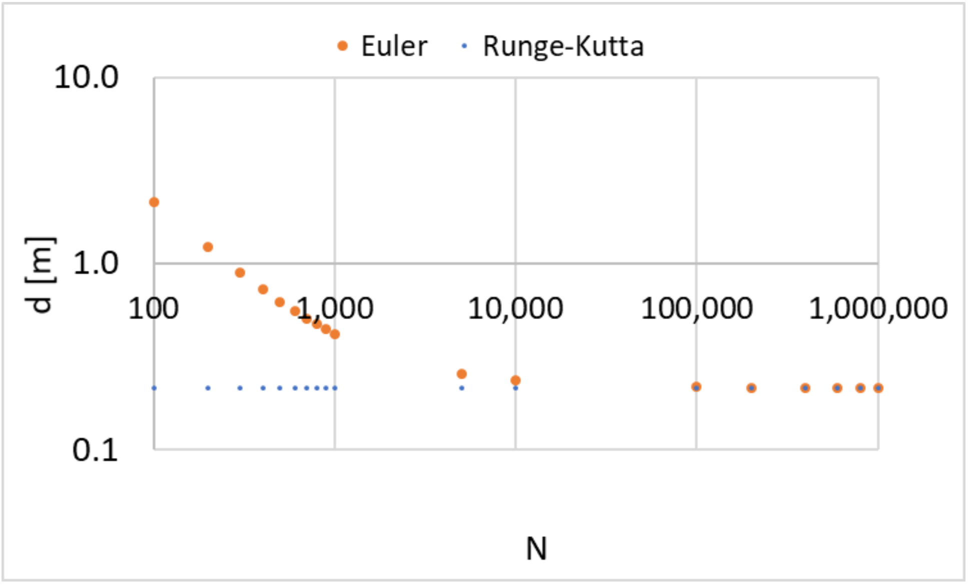

Figure 2: Euler and Runge-Kutta integration methods for increasing N.

For the Euler method, as N was doubled, the distanced approximately halved, which agrees with the expectation that the global error is proportional to Δt. The Runge-Kutta method converged to the physical solution in fewer integration steps, showing that the global error dropped off faster with increasing N. Throughout the rest of this report, the Runge-Kutta method was used.

Task 2: A proton in E and B fields

In this task, the movement of a proton in electric and magnetic fields was simulated. The particle was initially at rest (zero momentum) in position (x, y, z) = (0, 0, 0) m. The rest mass of a proton is m = 938 MeV and it has positive charge.

Task 2.1: Acceleration in an electric field

An electric field was defined as E_1 = (1.0, 0, 0) MV/m in the range 0 m ≤ x < 1 m for all y,z and zero everywhere else. This was done by writing an if statement which checked whether x was in the range 0 m ≤ x < 1 m and assigned it the required electric field if it was. If not, the electric field was set to zero. The if-statement is shown below:

```
public double [] Efield(double t, double x, double y, double z)
{
    // define vectorial E-field in V/m at position t, x, y, z

    if (x >= 0 && x <1){
        double[] E = {1.0e6,0.,0.};
        return E;
    }
    else {
        double[] E = {0.,0.,0.};
        return E;
    }

}
```

The B-field was set to B = (0, 0, 0) T everywhere, t_{max} was changed to 1 x 10^{-6} s so that the particle travelled a sufficient distance in x, and 10,000 integration steps were used to create a smooth plot. The x vs t results are shown in Figure 3. Figure 3 shows the particle accelerating in x under the electric field until it exits the field at x = 1 m. It then travels with constant velocity (constant gradient).

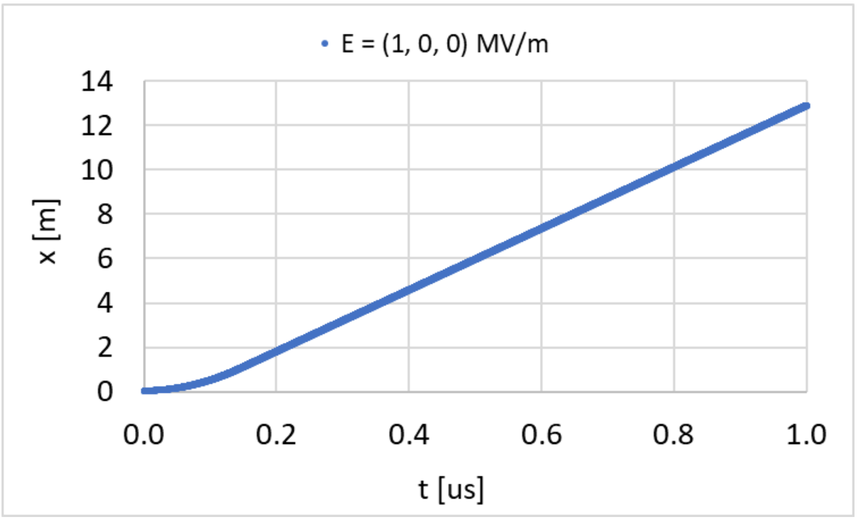

Figure 3: The change in x position with time for a proton under an electric field E_1 = (1.0, 0, 0) MV/m in the range 0 m ≤ x ≤ 1 m and zero elsewhere. The magnetic field was zero throughout space.

Task 2.2: deflection in a magnetic field

Next, a B field was defined which covered the range 2 m ≤ x < 3 m for any y and z, with (B_x, B_y, B_z) = 0, 0, 0.2) T using the same method as used for the E field in Task 2.1. The additional code is shown below:

```
public double [] Bfield(double t, double x, double y, double z)
{
    // define vectorial B-field in Tesla at position t, x, y, z
    if (x>=2 && x<3){
        double[] B = {0.,0.,0.2};
        return B;
    }
    else {
        double[] B = {0.,0.,0.};
        return B;
    }
}
```

The program was run using 10,000 steps and $t_{max} = 1 \times 10^{-6}$ s. Figure 5 shows the y vs x trajectory of the proton under the influence of the magnetic and electric fields. Initially, the proton only travels in x as it is accelerated by the electric field (which is in x). At x = 2 m, the proton reaches the magnetic field and is deflected. The proton travels in a circle until it exits the magnetic field at x = 2 m. It then travels solely in the negative x direction. In the region 0 m ≤ x ≤ 1 m, the electric field exerts a force on the particle in the positive x direction and it decelerates to velocity = 0 at x = 0. The particle is then accelerated back in the positive x direction and the cycle repeats. Conservation of energy and momentum mean that it is sent along the same trajectory as before.

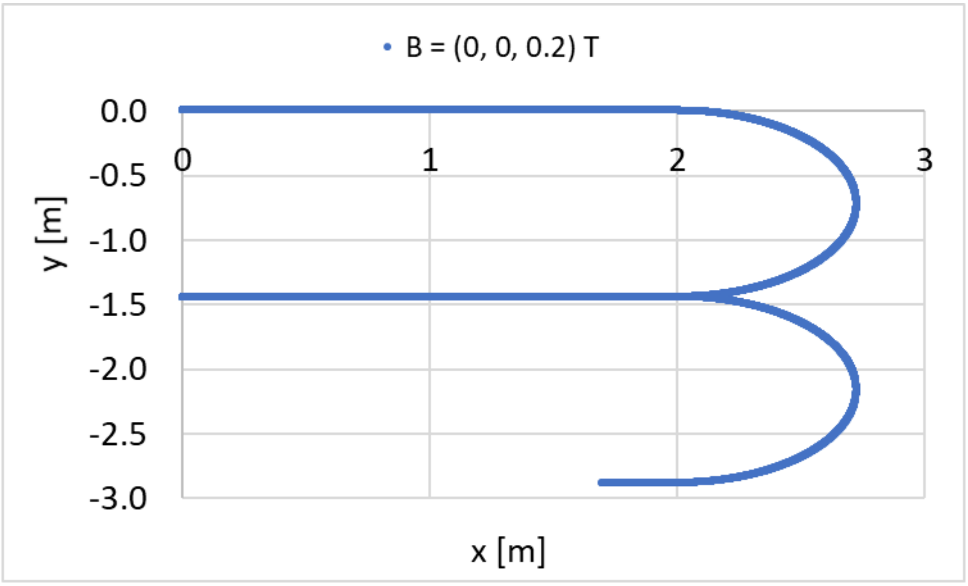

Figure 4: The trajectory of a proton under E_1 = (1.0, 0, 0) MV/m in the range 0 m ≤ x ≤ 1 m and (B_x, B_y, B_z) = 0, 0, 0.2) T in 2 m ≤ x < 3 m.

Task 2.3: Selecting particles with a certain velocity

In this last task, an additional E field was defined such that it cancelled the effect of the B field in order to simulate a Wien Filter which selects particles by their velocity. Thus, the electric field covered the range 2 m ≤ x < 3 m and all y and z. Its magnitude and direction were derived using the Lorentz equation,

$$F = q(\bar{E} + \bar{v} \times \bar{B}) \quad \text{Equation 6}$$

where F is the net force on the particle, q is the charge, and v is the velocity. In order to find an E which would cancel B, Equation 6 was set equal to zero and rearranged for E. As v only had components in the x direction, v_x, and B was only in the z direction, B_z, this reduced to,

$$\bar{E} = \bar{v} \times \bar{B} = (0,\ v_x B_z,\ 0)V/m = (0,\ \beta c B_z,\ 0)V/m \quad \text{Equation 7}$$

where β is the relativistic speed of the proton. This was implemented using the code below:

```
public double [] Efield(double t, double x, double y, double z)
{
    // define vectorial E-field in V/m at position t, x, y, z
    if (x >= 0 && x < 1){
        double[] E = {1.0e6,0.,0.};
        return E;
    }
    else if (x >= 2 && x < 3){ //filter field
9.          double[] E = {0., output.beta()*0.2*c, 0.};   // E_y = beta*c*B_z
10.         return E;
11.     }
12.     else {
13.         double[] E = {0.,0.,0.};
14.         return E;
15.     }
16. }
```

The program was run using the original electric field, E_1 = (1.0, 0.0, 0.0) MV/m, and the output value of β was 0.046128884816926466 c. The distance travelled by the proton vs time was plotted in Figure 5. Figure 5 is identical to Figure 3 which shows x vs t before the Wien Filter was implemented. This shows that the particle had the same magnitude acceleration.

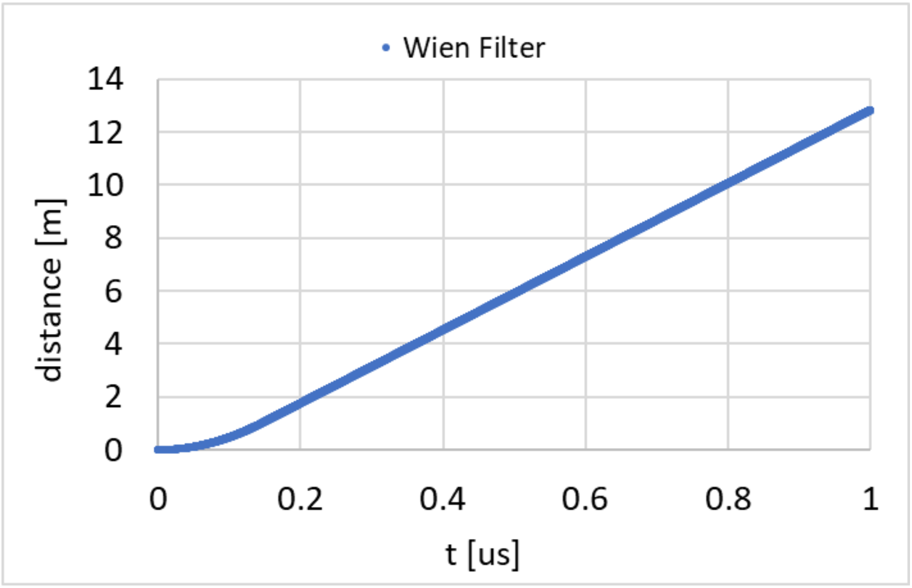

Figure 5: The distance travelled by a proton over time once a Wien filter was applied when E_1 = (1.0, 0.0, 0.0) MV/m.

Line 9 of the code above was edited so that the β value for E_1 = (1.0, 0.0, 0.0) MV/m was hardcoded into the Wien Filter. Thus, the filter would only select particles β = 0.046128884816926466 c. those with a different β would be deflected. Finally, E_1 was varied to test the filter. The trajectories of the particle for various E_1 are plotted in Figure 6. For E_1 = (0.9, 0, 0) MV/m, the proton was deflected in the positive y direction and for E_1 = (1.1, 0, 0) MV/m it was deflected in the negative y direction. For E_1 = (1.0, 0, 0) MV/m, the proton travelled in an approximately straight path along y = 0 m, as expected. The path was not perfectly straight because of the limited precision of the program. This is shown more clearly in Figure 7. These results showed that the filter was successfully selecting particles with β = 0.046128884816926466 c.

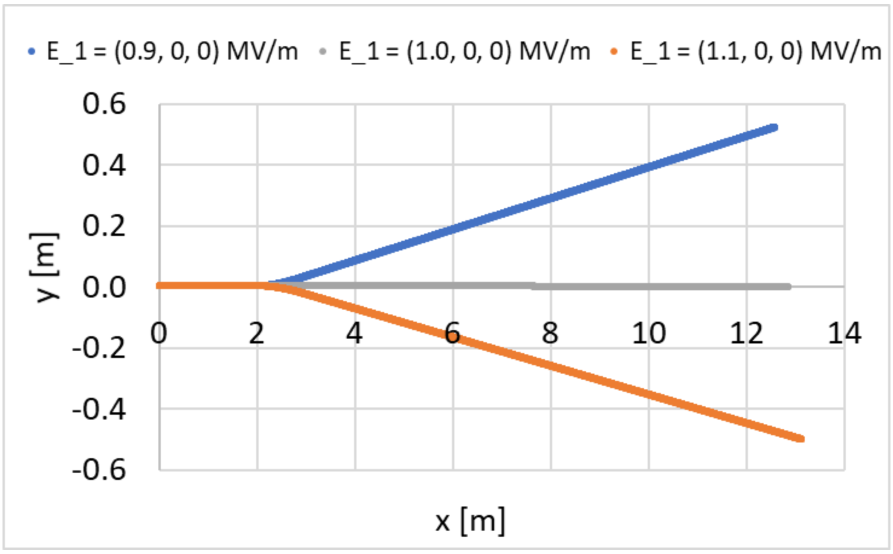

Figure 6: The Wein Filter was tested for various E_1 and performed as expected, deflecting particles which were not travelling at the selected speed.

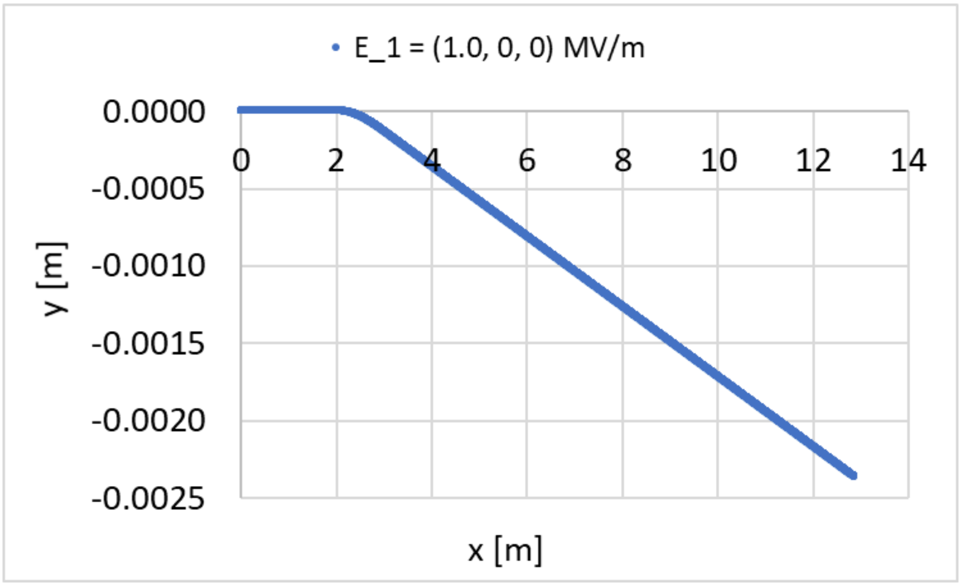

Figure 7: For E_1 = (1.0, 0, 0) MV/m, the particle was deflected slightly due to the limited precision of the program.