Exploring Taylor Methods for Fast and Precise Particle Tracking

Ibai Ceberio

July 4, 2019

Bachelor Research Project Physics

Supervisors:

Dr.ir. C.J.G. Onderwater

Prof. dr. S. Hoekstra

university of groningen

Van Swinderen Institute

University of Groningen

The Netherlands

Exploring Taylor Methods for Fast and Precise Particle Tracking

Ibai Ceberio

Abstract

The LHCb experiment is set up to explore what happened after the Big Bang that allowed matter to build the universe we inhabit today. In this paper an alternative particle tracking method is described. The potential of this method resides in the capacity of working in parallel computing. At the same time allows to calculate the ending parameters without the constant need of numerical integration methods which are time consuming. The method is being encoded in a c++ script.

Contents

List of Figures

Chapter 1

Introduction

Scientist have tried to give explanation to several phenomena through years. As the technology has improved we have being able to explore smaller and smaller interactions of matter. Particle physics is a branch of physics that works with the subatomic particles which constitute matter and radiation. Moreover, The Standard Model of particle physics is the theory enclosing the interaction of particles under the fundamental forces (gravity is excluded).

However, there are so phenomena that the Standard Model can't explain. Some of those are the neutrino oscillations, matter-antimatter asymmetry or the nature of dark matter and dark energy. Despite of these problems there is a bigger and more important discussion in the mathematical framework. Both the Standard Model and general relativity are incompatible under certain conditions. In conclusion the Standard Model is far for being a complete theory of fundamental interactions.

To discover and develop a new model physicist work in the LHC (Large Hadron Collider). It is the worlds most powerful particle accelerator and the largest. The LHC is a 27 km ring of superconducting magnets (see figure 1.1). To boost energy to the particles involved there are some accelerating structures along the way. Using magnetic dipoles particles are bend, such that they can collide several times. Thank to the magnets placed alongside the ring the particle beams can be curved and keep them in track. As it has being said the magnets are superconducting at temperatures such as -271.3 °C [2]. Therefore all the magnets are cooled down via a system of liquid helium. On the other hand there are other type of magnets which focus on making the beam as narrow and focus as possible so it can collide with the beam coming from the opposite direction. For the LHCb experiment high precision in particle tracking is really necessary. Also, because of the high velocities of the detected particles the bending is going to be really small.

Once the particles collide there is a cascade of smaller particles which decay and must be detected and analyzed later on. For that several detectors are used. Generally the data comes at a rate of millions per second (for LHCb). The Worlwide LHC computing grid deals with all the data in two main steps. In the first one the data amount is reduced via an algorithm that selects interesting events for physicist. That narrows the event number to around 100,000 per second. In the second step the digital reconstruction must be done. Using more specific algorithms the event number of interest is reduced to only 100-200 events per second [6]. Nevertheless Physicist continue to refine algorithms to detect even more interesting events.

As we see a fast particle tracking method is needed. Numerical methods have

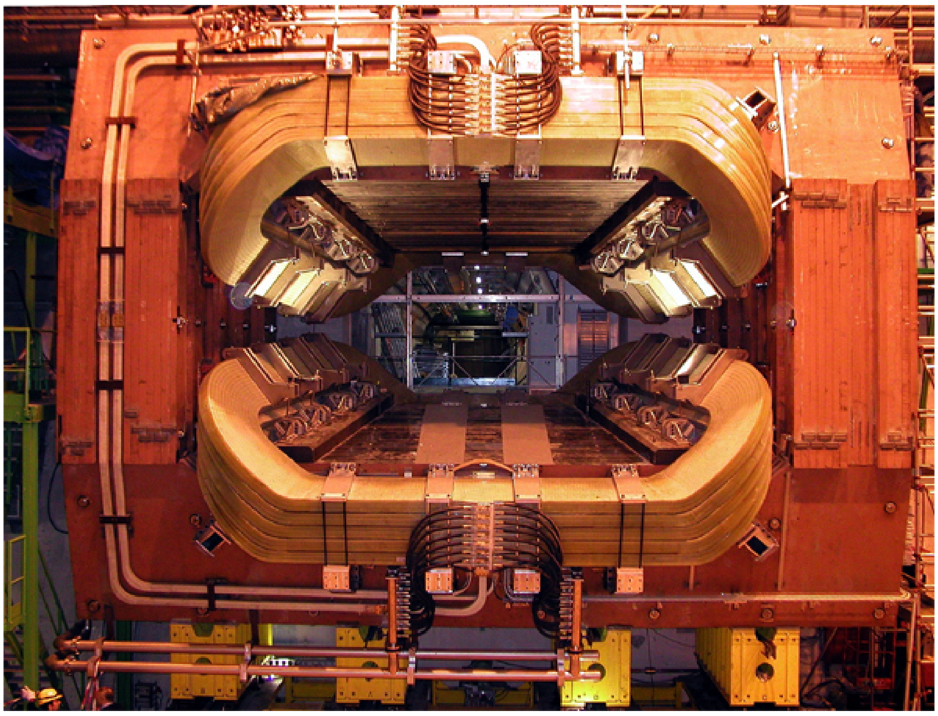


Figure 1.1: Dipole Magnet of LHCb. [4]

proven to be able to determine the tracks with high precision and reliability but there are time consuming methods. The tracking routines used nowadays, such as Euler or Runge Kutta, work but they have a problem. As they work sequentially quite a lot of time is lost in the determination of middle points which not that important.

The LHCb magnet create an inhomogeneous magnetic fields due to its wide spectrum (see figure 1.2). Taking into account that the LHCb magnet stay unaltered for long periods of time we can assume that the magnetic field will remain stable during the time of the experiment running. Consequently we have the chance of Taylor expanding the magnetic field so we can track the particle in a later stage. Knowing the coefficients of the Taylor expansion would lead to a decrease in the computation time since the same parameters can be used again and again while the LHCb stays unaltered.

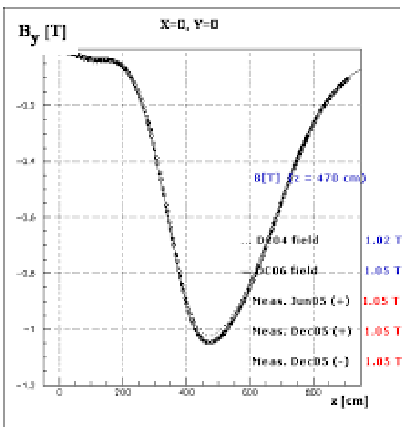


Figure 1.2: Magnetic Field map in LHCb. [5]

The aim of this bachelor research project is to **calculate the minimum order of the Taylor expansion for a nice particle tracking in arbitrary magnetic fields**. With the calculated coefficients particle position in the end of the magnet can be determined with the only need of the initial parameters.

Chapter 2

Theoretical base

To begin with the report some basic theory will be introduced for a better understanding of the following chapters.

2.1 Electromagnetism

First of all, electromagnetism is a branch of Physics which focuses on the study of the electromagnetic force, one of the four fundamental interactions. The electromagnetic force describes the interaction that occurs between electrically charged particles. At high energies the weak and electromagnetic forces are unified into what is called the electroweak force.

Primarily both electricity and magnetism are different manifestations of the same phenomena. This is due to the fact that the particles involved must obey the Maxwell's equations (see subsection 2.1.1). Even though electricity and magnetism are related to each other we will see that for at certain circumstances the electric field is negligible.

2.1.1 Maxwell Equations

Maxwell's equations represent one of the most concise ways to state the relationship between electricity and magnetism. Taking them as a starting point one can explain almost all the developed relations in the field. Maxwell's equations are a group of 4 equations which relate distinct aspects of the electromagnetism. First we have Gauss' law for electrcity:

$$\nabla \cdot \mathbf{E} = \frac{\rho}{\epsilon_0} \tag{2.1}$$

meaning that the electric flux out of any closed surface is proportional to the total charge enclosed within the surface. Secondly we have Gauss' law for magnetism:

$$\nabla \cdot \mathbf{B} = 0. \tag{2.2}$$

The physical meaning of the law is that the net magnetic flux out of any closed surface is 0. After that we come to Faraday's law of induction:

$$\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t}. \tag{2.3}$$

In this case the line integral of the electric field around a closed loop is equal to the negative of the rate of change of the magnetic flux through the area enclosed by the loop. The last of these set of equations is Ampere's law. It means that in the case of static electric field, the line integral of the magnetic field around a closed loop is proportional to the electric current flowing through the loop and is represented in the following way:

$$\nabla \times \mathbf{B} = \mu_0 \mathbf{J} + \mu_0 \epsilon_0 \frac{\partial \mathbf{E}}{\partial t}. \tag{2.4}$$

The bases of the equations will be used for simplicity when the equations of motion are set.

2.1.2 Magnetic Rigidity and x_3

The magnetic rigidity is the influence on the motion of charged particles. With units of $\mathrm{T} \cdot \mathrm{m}$ it is a measure of the momentum of a particle. Additionally, it gives an explanation to the fact that higher momentum particles tend to bend less than same particles with smaller momentum.

A particle traveling through a uniform magnetic field describes a circular orbit of radius ρ and the Lorentz force equalling the centrifugal force leads to

$$R = B\rho = \frac{p}{q} \tag{2.5}$$

where B is the magnetic field, ρ is the gyroradius of the particle under the influence of the field, p the momentum of the particle and q its electric charge.

The inverse of the bending radius is x_3 and it sums up the effects of the momentum, charge and magnetic field in just one variable. This can be useful in further calculations. At some reference point x_3 is defined as:

$$x_3 = \frac{q}{p} \cdot B(x_0, y_0, z_0). \tag{2.6}$$

2.2 From classic to relativistic view

What we are mainly concerned with the LHCb experiment is that particles go almost as fast as the speed of light. Bearing this in mind it is important to take into account the relativistic effects that arise in this environment. Our method will have to take into account this corrections in order to work out.

The Lorentz factor

$$\gamma = \frac{1}{\sqrt{1 - \frac{v^2}{c^2}}} \tag{2.7}$$

is used in the calculation of the relativistic mass which we must coupled in the equations of motion or magnetic rigidity. Taking the limit of $v \ll c$, $\gamma \approx 1$ (classical mechanics). In contrast, at high velocities the Lorentz factor starts to gain importance and that is why we must consider it.

As a result the momentum of the particle can be expressed in terms of the rest mass and the velocity of the incoming particle.

$$p = \gamma m_0 v = \frac{m_0 v}{\sqrt{1 - \frac{v^2}{c^2}}} \tag{2.8}$$

and consequently

$$x_3 = \frac{Bq}{p} = \frac{Bq}{\gamma m_0 v}. \tag{2.9}$$

2.3 Equations of motion

One of the most striking aspects of this problem is to use the correct equations of motion. For our purpose we will we using cartesian coordinates in a way that the transverse axes are going to be x (horizontal) and z (vertical). What is more, the longitudinal axis, the one along the magnetic field, will be defined by y.

We start from the Lorentz force equation:

$$\mathbf{F} = m\mathbf{a} = q\mathbf{E} + q\mathbf{v} \times \mathbf{B}. \tag{2.10}$$

As it has been mentioned in the introduction (see chapter 1) the LHCb experiments runs with a magnetic field of around 1 T. When a particle with a velocity $v \approx c$ enters the magnetic region it gets curved. In case we wanted the particle to be affected significantly by an electric field we would need to create one of at least several million V/m because $Bv \approx 3 \times 10^8$ V/m. With this in mind the force exerted by any electric field will be neglected:

$$\mathbf{F} = m\mathbf{a} \approx q\mathbf{v} \times \mathbf{B}. \tag{2.11}$$

Expanding the equation of motion into the coordinates fixed before we get:

$$\ddot{x} = \frac{q}{\gamma m}(\dot{y}B_z - \dot{z}B_y), \tag{2.12}$$

$$\ddot{y} = \frac{q}{\gamma m}(\dot{z}B_x - \dot{x}B_z), \tag{2.13}$$

$$\ddot{z} = \frac{q}{\gamma m}(\dot{x}B_y - \dot{y}B_x). \tag{2.14}$$

2.4 Taylor series expansion

In this section an introduction to the Taylor series is given. After that we will explain how the expansions are going to be useful in our method.

The Taylor series expansion is a representation of a function as an infinite sum of terms. For a function, $f(x)$ that is infinitely differentiable at a real or complex number a the Taylor expansion is:

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2) + \frac{f'''(a)}{3!}(x-a)^3 + ... \tag{2.15}$$

In addition we have the possibility of expanding the serie into more than one variable. The 2-D Taylor expansion is defined as:

$$\begin{aligned} f(x,y) &= \sum_{n=0}^{\infty} \left[\frac{1}{n!} \sum_{k=0}^{n} \begin{pmatrix} n \\ k \end{pmatrix} \frac{\partial^n f}{\partial x^{n-k} \partial y^k}\bigg|_{(a,b)} (x-a)^{n-k}(y-b)^k \right] \qquad (2.16) \\ &= f(a,b) + f_x(a,b)(x-a) + f_y(a,b)(y-b) \\ &\quad + \frac{1}{2!}[f_{xx}(a,b)(x-a)^2 + 2f_{xy}(a,b)(x-a)(y-b) + f_{yy}(a,b)(y-b)^2] + ... \end{aligned}$$

where

$$\begin{pmatrix} n \\ k \end{pmatrix} = \frac{n!}{(n-k)!k!}$$

are the binomial coefficients and

$$f_{ij} = \frac{\partial^2 f}{\partial i\, \partial j}.$$

.

Now that an introduction has been given we can explore another useful property of the Taylor series. They can be used to solve differential equations. We are going to make use of this property to fit a polynomial sum to the trajectory of the particle.

To give an example let's say we have the next equation which needs to be solved:

$$\frac{d^2y}{dx^2} - y = 0. \tag{2.17}$$

Then we start with a series expansion around x=0 with the form of

$$y(x) = \sum_{n=0}^{\infty} a_n x^n. \tag{2.18}$$

Taking the second derivative of the expression 2.18 and plugging in into our equation 2.17 leads to

$$\sum_{n=2}^{\infty} n(n-1)a_n x^{n-2} - \sum_{n=0}^{\infty} a_n x^n = 0. \tag{2.19}$$

Coming to the realization that the first sum does not start from 0 we can transform it into a sum starting from 0. If we make the variable change of $n' = n - 2$ the first term of equation 2.19 becomes

$$\sum_{n'=0}^{\infty} (n'+2)(n'+1)a_{n'+2} x^{n'}. \tag{2.20}$$

Moreover, we want also the exponent of x to be the same in both cases. In resume we end up having the next equation:

$$\sum_{n=0}^{\infty} [(n+2)(n+1)a_{n+2} - a_n]x^n = 0. \tag{2.21}$$

For this to hold along x, all terms in the sum have to be 0. So

$$(n+2)(n+1)a_{n+2} - a_n = 0,\ \forall n \tag{2.22}$$

Solving the equation leaves us with two relations for a_k: $a_{2k} = \frac{a_0}{(2k)!}, k = 0, 1, 2, ...$ and $a_{2k+1} = \frac{a_0}{(2k+1)!}, k = 0, 1, 2, ...$ To conclude we will put these results into the general form of $y(x)$ of equation 2.18:

$$y(x) = a_0 \sum_{k=0}^{\infty} \frac{x^{2k}}{(2k)!} + a_1 \sum_{k=0}^{\infty} \frac{x^{2k+1}}{(2k+1)!}. \tag{2.23}$$

As the general solution of the equation is $y(x) = c_1 \cosh(x) + c_2 \sinh(x)$ and as we know that

$$cosh(x) = \sum_{n=0}^{\infty} \frac{x^{2n}}{(2n)!} \tag{2.24}$$

and

$$sinh(x) = \sum_{n=0}^{\infty} \frac{x^{2n+1}}{(2n+1)!} \tag{2.25}$$

we can conclude that the solution found via Taylor series is correct. To determine the coefficients, either the analytical solution needs to be known, some technique needs to be developed to relate the coefficients to the differential equation, or we could fit the polynomials led by numerical solutions.

Chapter 3

Particle Tracking

In this chapter we explore the numerical solution of a particle crossing a magnetic field. We will first explore a constant magnetic field and then a more complex position depended field which we will later use for Taylor approximation.

3.1 Constant Magnetic Field

When we start to work with different numerical methods one of the first challenges is to find a discretization stepsize. As we want to have a precise solution we need a maximum number of points available. Moreover, we know that if we use to many steps the solution is going to be the most precise possible. However, inserting more steps means that the computation time is going to be longer also. In this perspective, a decision has to be made. Do we want a really precise but slow method or do we want a less precise but faster one instead?

To obtain some experience we will investigate a well known case. If we make a particle go through a constant magnetic field, $\overrightarrow{B}(x, y, z) = B_z\,\widehat{\mathrm{k}} = \mathrm{const}$, we expect it to start making circles. $\overrightarrow{F} = q\overrightarrow{v} \times \overrightarrow{B}$ then becomes

$$\gamma m\,(\dot{v}_x\,\hat{\imath} + \dot{v}_y\,\hat{\jmath}) = q\,(v_y B_z\,\hat{\imath} - v_x B_z\,\hat{\jmath}).$$

Applying the Euler method for the differential equation and taking the relativistic effects into account we finish with the two sets showed next:

$$v_{x+1} = v_x + h\frac{q}{\gamma\, m}\,v_y B_z, \tag{3.1}$$

$$v_{y+1} = v_y - h\frac{q}{\gamma\, m}\,v_x B_z. \tag{3.2}$$

Hence, implementing the method would give us one of the first answers we need. During the process different number of steps have been taken and plotted to show that the more points we use the more precise the solution is going to be. Particularly in this case the way to see if the points are enough is to try to get the most constant radius possible. Results are showed in figure (3.1).

Needs to be said that we can't use all the steps we want. Despite being theoretically possible we need to take into account the processing power of standard computers. If it is not said otherwise and we use double type variables we will be working with numbers that have a maximum storage capacity of 8 bytes. As a result if at some point we require more memory the program will crash down. As it can

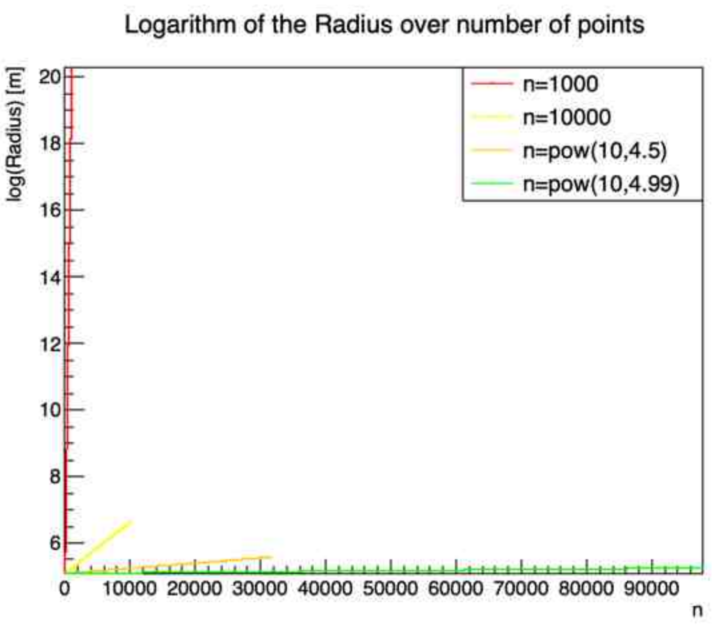


Figure 3.1: Logarithm of the radius over the number of steps taken.

be seeing we reach this point when we get in the neighbourhood of 10^5 steps. After the exact value of 10^5 being tried we have being given back an error and leaded us to use almost that quantity of steps.

3.2 Inhomogeneous Magnetic Field

In general the magnetic field is not constant. The rare case is the one where the applied magnetic field is constant indeed. Our next step is going to be to manipulate the code to include an inhomogeneous magnetic field. Taking as a reference the magnetic field of the LHCb (see figure 1.2) we will use a Gaussian shaped magnetic field:

$$\overrightarrow{B}(x,y,z) = B_z(x,y)\,\hat{k} = 1.05\,e^{\left(-\frac{x^2+(y-5)^2}{5.5}\right)}\hat{k}\text{ T}. \tag{3.3}$$

A plot of the magnetic field over the distance can be seeing in figure 3.2. For our purpose we will be focusing on the symmetry plane because if $z \neq 0$ the magnetic field would also have components in the x and y directions ($B_x \neq 0, B_y \neq 0$).

Using the data we have from the LHCb [4] we have set up the amplitude of the magnetic field in 1.05 Tesla and the range of the exponential to the dimensions of one of the magnets in the LHCb experiment. That means we will have a Gaussian distributed magnetic field which main value of 1.05 Tesla is going to be in the $(x_0, y_0) = (0, 5\text{ m})$ position. Essentially we are satisfying the boundary conditions given by the dimensions of the magnet.

In connection with the incident electron momentum we have that in order to reach the detector after exiting the magnets a minimum momentum of 1.5 GeV/c [3] is required. On the other hand the maximum momentum allowed is 100 GeV/c. Below is shown the deflection of the different momentum electrons (see figure 3.3). As we can see the faster the particle the less it is going to be affected by the magnetic field. The code used for the plot can be found in Appendix A.

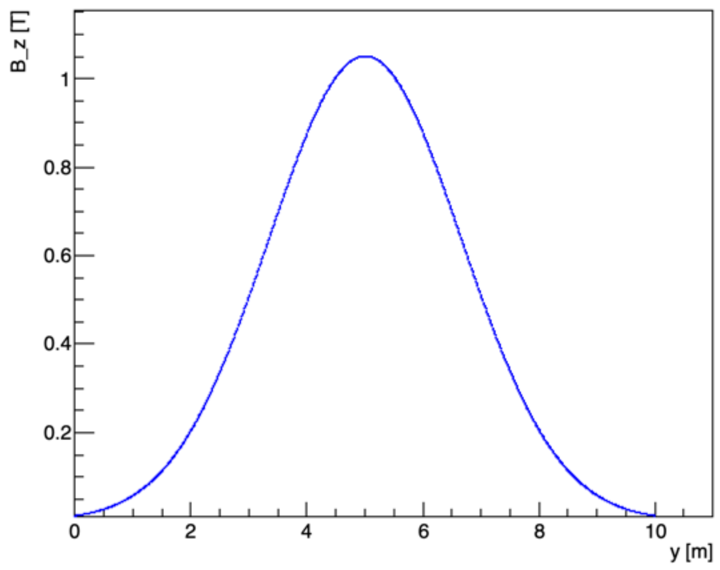


Figure 3.2: The magnetic field along the y direction. x=0 has been chosen for the image.

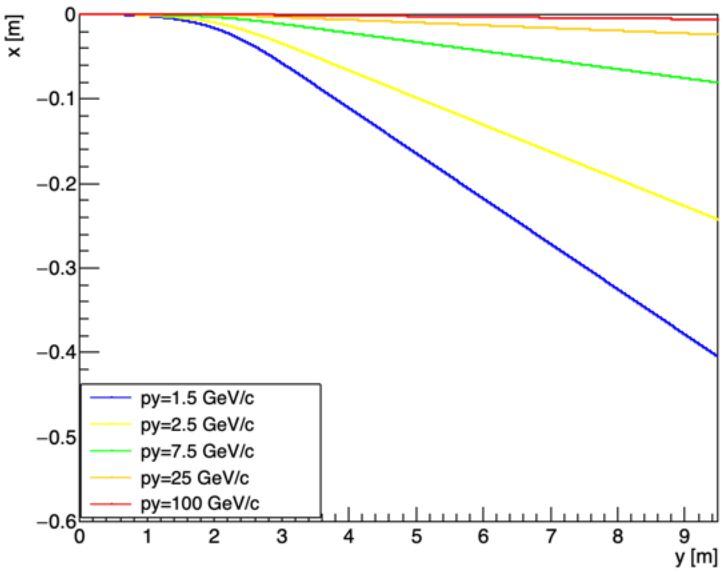


Figure 3.3: Different momentum electron going through a Gaussian distributed magnetic field. All the particles enter the magnet in perpendicular to the x-plane and from the (0,0) position.

With this implementation, we can proceed to the next step; finding the Taylor amplitudes. This is necessary, because although the Euler method can be very precise, it is very slow. Also Euler like methods, such as Runge-Kutta method will be similarly slow.

Chapter 4

Taylor Expansion

In this chapter a description for the determination of the polynomial coefficients of the x direction is given. First particle tracking is carried out for different v_x values using the Euler method coding (see Appendix B.1) and the polynomial order is determined. Secondly, curve fitting is done for all the velocities with the previous calculated polynomial order. After that stage the coefficients of the equations are plotted versus the initial v_x velocity and curve fitting is done for each order. In the end the whole equation is written together and some results are obtained.

4.1 Polynomial Order

The aim of this polynomial approach is to define an expression for x_{out} dependable in x_{in} and Vx_{in}. In addition, we want to be able to repeat the same procedure with Vx_{out}, Vy_{out}, etc.

We start plotting the particle tracks for different initial velocities. At least for now we are going to be focusing in how a variation of the initial v_x is going to affect the track this particle is going to have. In accordance with that we need to calculate the x_{out} for a certain x_{in}. Moreover, we want our polynomial function to be able to give us the end position for any initial x position as long as we are inside the magnet. After computing the code for different velocities the results obtained is shown in the following graph (see figure 4.2). Additionally, the tracks are also shown in figure 4.1. Analyzing the plot we see that, as expected, the initial v_x plays an important role in the upcoming particle track.

The next step we are taking is to see which will be the minimum polynomial order to fit the curves. We begin with the second order polynomial fitting because it is clear that we don't have a linear response. What's more, we can use the same reasoning for the second order polynomial function because the curve is not a parabola neither. After increasing the polynomial order of our function the development of the fitting curve is shown in figure 4.3.

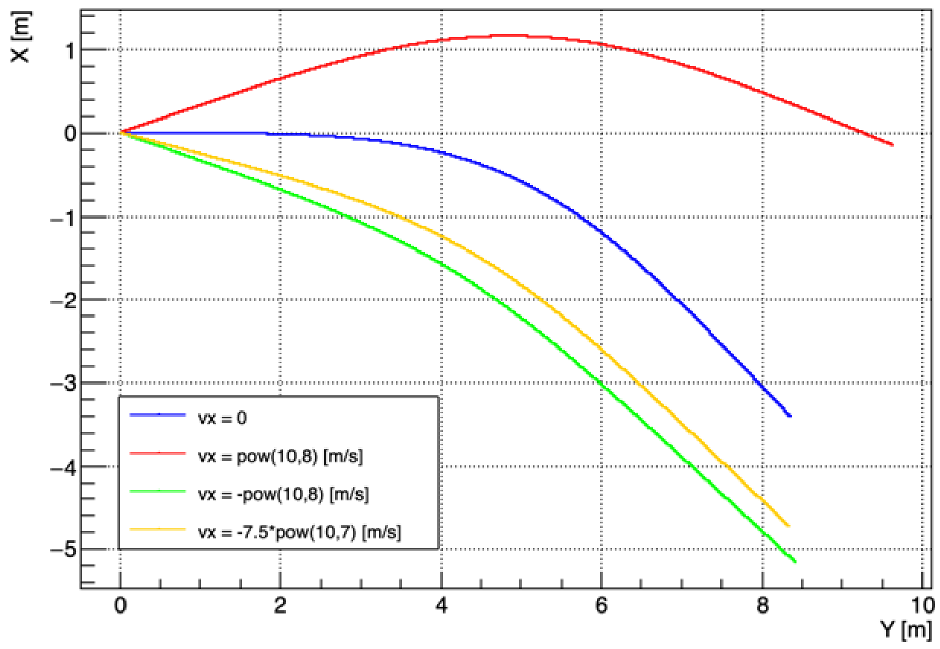


Figure 4.1: x vs y. Different track for each of the v_x velocities implemented with a starting point of $x = 0$ and $y = 0$. The plotted curves are for an electron with $p_y = 1.5$ GeV/c coming through a Gaussian distributed magnetic field with a maximum amplitude of 1.05T.

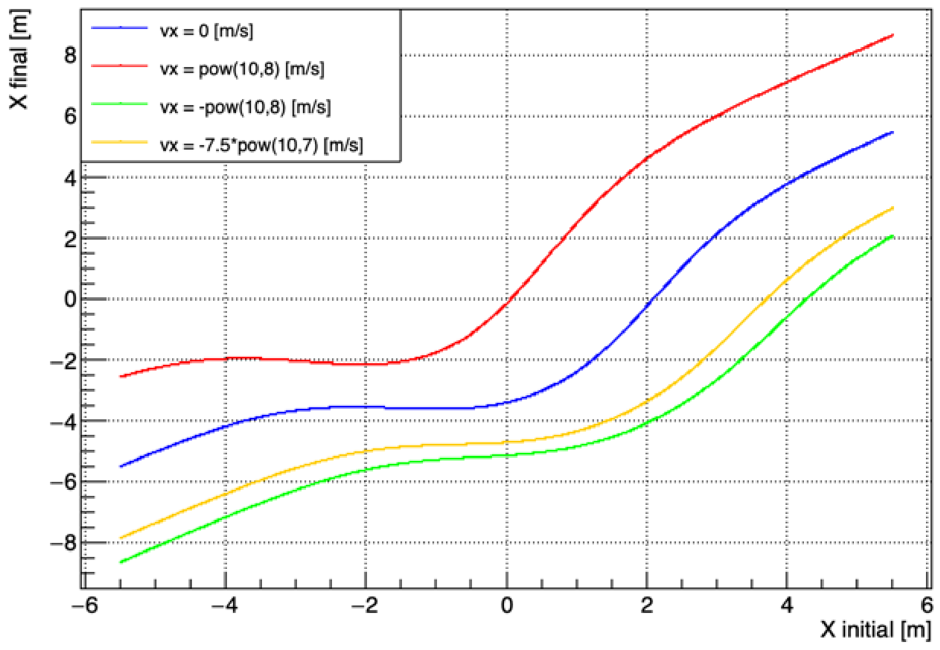


Figure 4.2: x_{out} vs x_{in}. The final x position over all the possible inital x positions.

In conclusion, we have to reach the seventh order of the polynomial equation if we want to fit the curve with enough precision and even though the extreme points

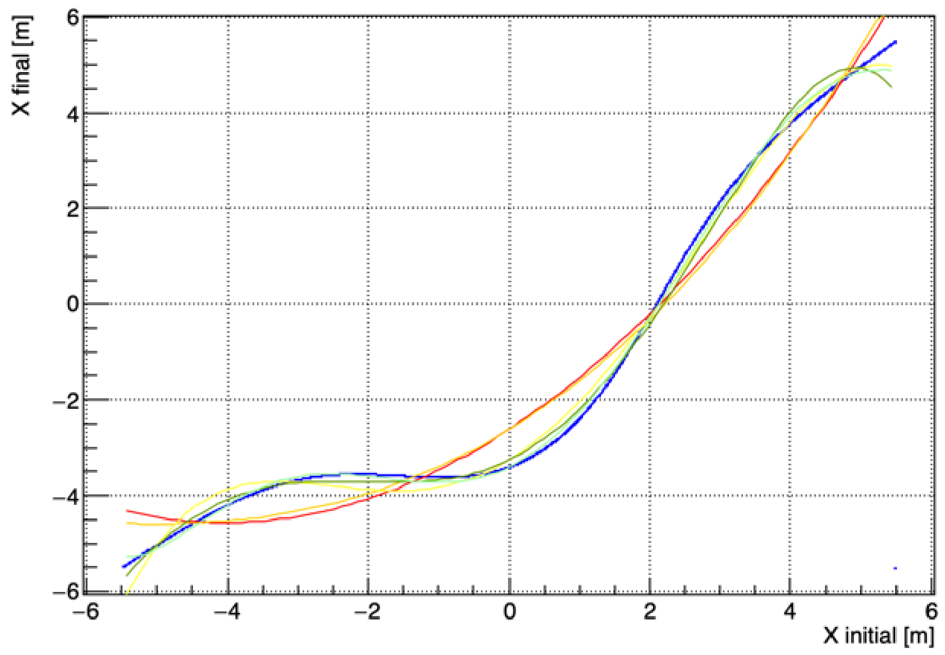


Figure 4.3: Increasing order polynomial fitting, from second to sixth order, over the real curve (blue). The curve shows the x_{out} vs x_{in} for a $v_x = 0$ velocity.

don’t seem to fit correctly as we can see in figure 4.4.

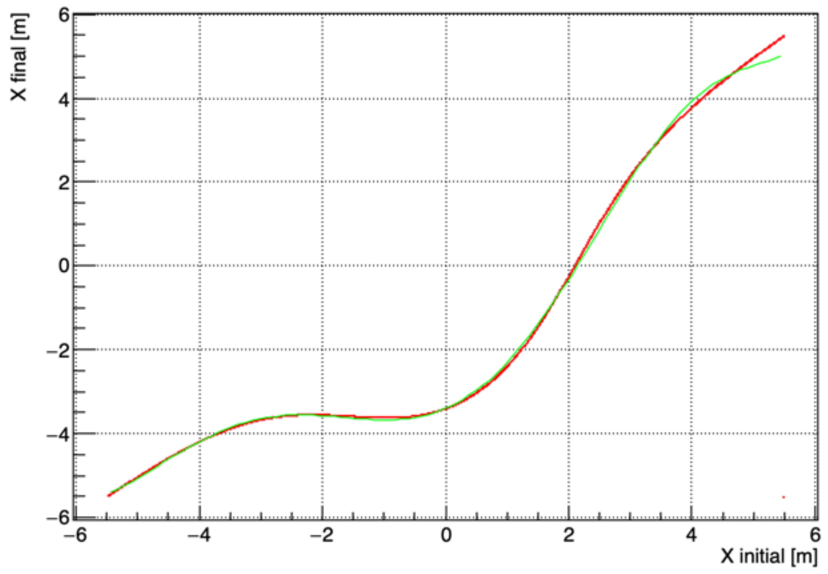


Figure 4.4: Seventh order polynomial fitting(green) over the real curve (red). The curve shows the x_{out} vs x_{in} for a $v_x = 0$ velocity.

Once this part is done we come back to figure 4.2. Now that it is known the order we need we can do the curve fitting for all the 4 different v_x velocities. Consequently we will have 4 different polynomial equations. The reason for that is the expected dependence of the coefficients on v_x. What first was a function of the form

$$x_{out} = [a + bx + cx^2 + dx^3 + ex^4 + fx^5 + gx^6 + hx^7] \times x_{in} \tag{4.1}$$

with all the coefficients being constant has become

$$x_{out} = [a(x') + b(x')x + c(x')x^2 + d(x')x^3 + e(x')x^4 + f(x')x^5 + g(x')x^6 + h(x')x^7] \times x_{in} \tag{4.2}$$

. Inevitably, this means that we will need to do the same polynomial fit for each of the coefficients. The next table summarizes the data available from the figures:

x' [m/s]	a	b	c	d	e	f	g	h
0	-3.426	0.6032	0.4548	0.06625	-0.02081	-0.003468	0.0003182	-8.719·10^{-5}
10^8	-0.01346	2.181	0.3467	-0.1383	-0.01533	0.005912	0.0002437	-8.719·10^{-5}
-10^8	-5.147	0.1487	0.06422	0.06466	0.00196	-0.001602	-6.93·10^{-5}	1.205·10^{-5}
$-7.5 \cdot 10^7$	-4.733	0.1329	01442	0.07987	-0.001441	-0.002509	-2.949·10^{-5}	2.642·10^{-5}

We start again plotting the values of a over the velocities in which they were measured and after the fitting we obtain:

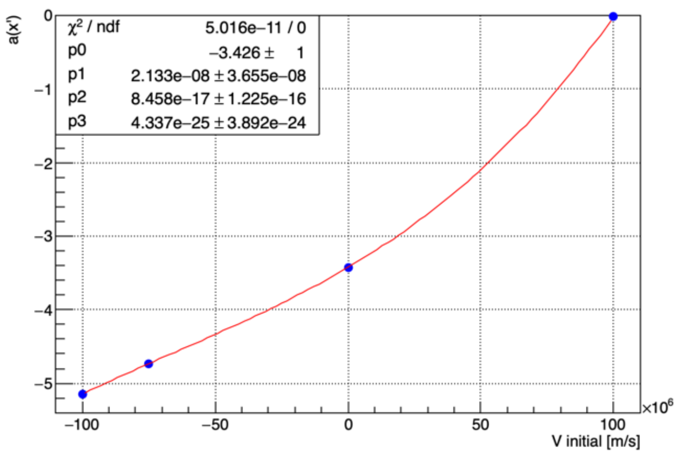
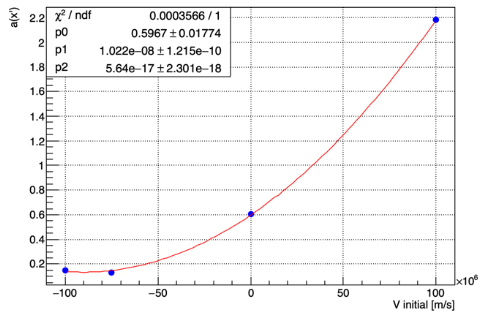


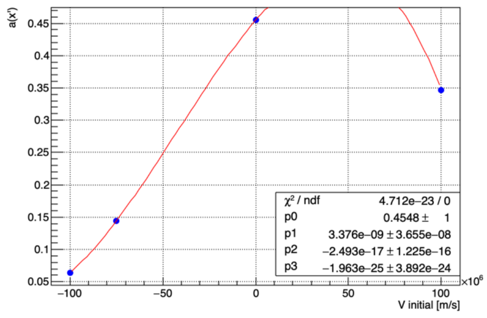
Figure 4.5: Coefficient a over v_x.

We will do the same for each of the coefficients. In each of the cases the polynomial order can be different mainly because it might not be the need of more. It is important to mention that for the case in figure 4.9 the polynomial fitting from Cern ROOT [1] did not work from the third order on. As a solution the KaleidaGraph program has been used. In each of the graphs we see how the initial v_x affects the polynomial coefficient. Shall we say that $a(x') = a(v_x)$. That is the reason why we have plotted the coefficients over the initial v_x.



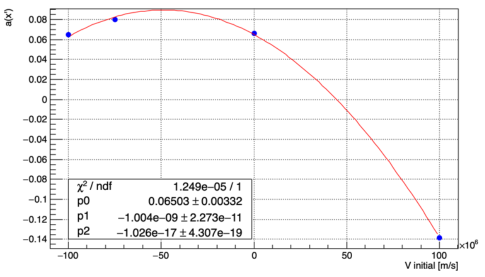
```
χ² / ndf    0.0003566 / 1
p0          0.5967 ± 0.01774
p1          1.022e−08 ± 1.215e−10
p2          5.64e−17 ± 2.301e−18
```

(a) Coefficient b over v_x.



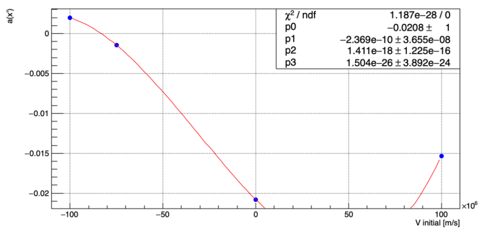
```
χ² / ndf    4.712e−23 / 0
p0          0.4548 ±      1
p1          3.376e−09 ± 3.655e−08
p2          −2.493e−17 ± 1.225e−16
p3          −1.963e−25 ± 3.892e−24
```

(b) Coefficient c over v_x.



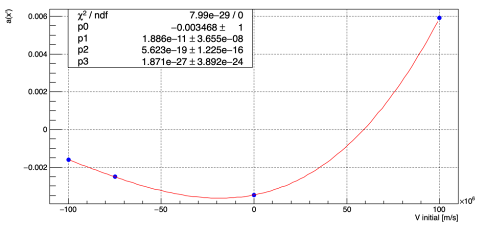
```
χ² / ndf    1.249e−05 / 1
p0          0.06503 ± 0.00332
p1          −1.004e−09 ± 2.273e−11
p2          −1.026e−17 ± 4.307e−19
```

(a) Coefficient d over v_x.



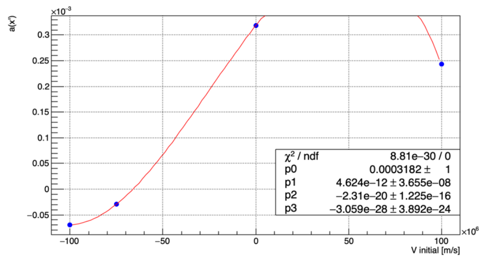
```
χ² / ndf    1.187e−28 / 0
p0          −0.0208 ±      1
p1          −2.369e−10 ± 3.655e−08
p2          1.411e−18 ± 1.225e−16
p3          1.504e−26 ± 3.892e−24
```

(b) Coefficient e over v_x.



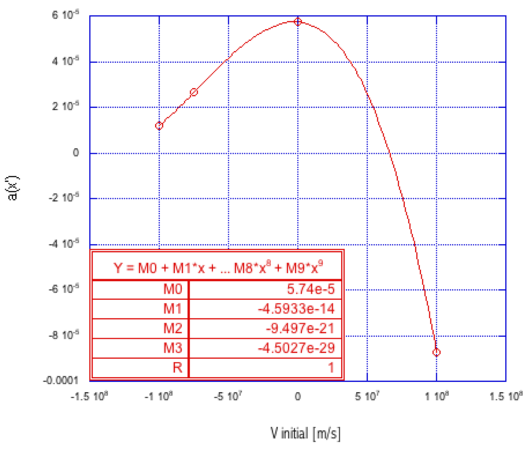
```
χ² / ndf    7.99e−29 / 0
p0          −0.003468 ±      1
p1          1.886e−11 ± 3.655e−08
p2          5.623e−19 ± 1.225e−16
p3          1.871e−27 ± 3.892e−24
```

(a) Coefficient f over v_x.



```
χ² / ndf    8.81e−30 / 0
p0          0.0003182 ±      1
p1          4.624e−12 ± 3.655e−08
p2          −2.31e−20 ± 1.225e−16
p3          −3.059e−28 ± 3.892e−24
```

(b) Coefficient g over v_x.



Y = M0 + M1*x + ... M8*x⁸ + M9*x⁹	
M0	5.74e-5
M1	-4.5933e-14
M2	-9.497e-21
M3	-4.5027e-29
R	1

Figure 4.9: Coefficient h over v_x.

4.2 Coupling everything

Summing up we have ended up with the next big polynomial equation where x_{out} not only depends in the initial x but in the v_x too.

$$
\begin{aligned}
x_o &= -3.43 + 2.13 \cdot 10^{-8}\,\dot{x}_i + 8.46 \cdot 10^{-17}\,\dot{x}_i^2 + 4.33 \cdot 10^{-17}\,\dot{x}_i^3 \\
&+ (0.59 + 1.02 \cdot 10^{-8}\,\dot{x}_i + 5.64 \cdot 10^{-17}\,\dot{x}_i^2)\,x_i \\
&+ (0.45 + 3.37 \cdot 10^{-9}\,\dot{x}_i - 2.49 \cdot 10^{-17}\,\dot{x}_i^2 - 1.96 \cdot 10^{-25}\,\dot{x}_i^3)\,x_i^2 \\
&+ (0.06 - 1.00 \cdot 10^{-9}\,\dot{x}_i - 1.03 \cdot 10^{-17}\,\dot{x}_i^2)\,x_i^3 \\
&+ (-0.02 - 2.37 \cdot 10^{-10}\,\dot{x}_i + 1.41 \cdot 10^{-18}\,\dot{x}_i^2 - 1.51 \cdot 10^{-26}\,\dot{x}_i^3)\,x_i^4 \\
&+ (1.88 \cdot 10^{-11}\,\dot{x}_i + 5.62 \cdot 10^{-19}\,\dot{x}_i^2 + 1.87 \cdot 10^{-27}\,\dot{x}_i^3)\,x_i^5 \\
&+ (4.62 \cdot 10^{-12}\,\dot{x}_i - 2.31 \cdot 10^{-20}\,\dot{x}_i^2 - 3.06 \cdot 10^{-28}\,\dot{x}_i^3)\,x_i^6 \\
&+ (5.74 \cdot 10^{-5} - 4.59 \cdot 10^{-14}\,\dot{x}_i - 9.49 \cdot 10^{-21}\,\dot{x}_i^2 - 4.50 \cdot 10^{-29}\,\dot{x}_i^3)\,x_i^7
\end{aligned}
$$

4.3 Validation: $v_x = 10000\,\mathrm{m/s}$

In this chapter a confirmation that the polynomial function works is proven. The polynomial equation versus the real tracking is shown below. We shall remember that our particle is an electron with $p_y = 1.5\mathrm{GeV/c}$ and $v_x = 10000$. This value for the horizontal velocity is near to the maximum limit we could use for a particle with such an energy. In case we tried higher velocity the total velocity of the particle would have passed the speed of light and it is well known that no particles go faster than that.

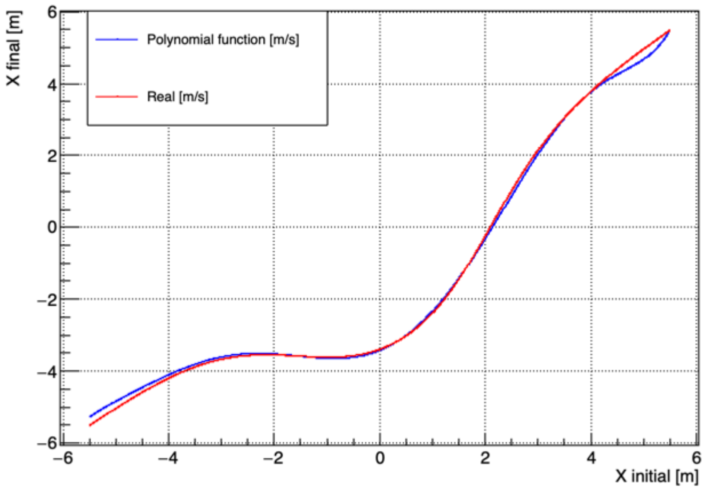


Figure 4.10: Polynomial function(blue) versus Euler tracking method(red).

Overall we see that the polynomial works magnificently in the center values and starts to fail in the edges. The come up with two main reasons for that. First, the magnetic field in the magnet limits is almost zero and consequently it is hard to calculate how the particle is going to respond. On the other hand, we settled the polynomial order in 7th order. If we want to get better results we just need to increase the polynomial order to get as much precision as we want. Although the fitting has being done by eye during this research, comparing with the LHCb experiment we would like to have a precision of the order of some micrometers [3].

Chapter 5

Conclusion

We have developed a possible method to track charged particles through arbitrary magnetic fields. We wanted to make use of the method to track the particles in the LHCb experiment. We have implemented a Taylor expansion based polynomial function. The initial idea was to find a similarly precise and faster tracking method. Currently used numerical methods need to compute the whole trajectory of the particle. However, the momenta values which are needed are only the ones saved in the initial and final positions. Most of the implementation time necessary would be reduced with a method such as the Taylor expanded function.

For a constant magnetic field the method shows acceptable results. The main reason is that it doesn't matter which is the initial position of the particle because the track will always be circles. Even though we could use the developed method to get the solution it would be time wasting because we can figure out the solutions without any computaional calculation.

In the next stage we have started to work with a Gaussian shape magnetic field. Given the amount of time we have had for this bachelor project we are not being able to do the following implementations in all three coordinates. That's why we have focused in the most interesting one of the 3. The x direction of the experiment is the most important because the x_{in} fixes how much the particle is going to bend and consequently, at what x position is going to come out (x_{out}). It wasn't possible to compute the polynomial fitting for the y and z directions but having the time it would be something not to complicated to do.

After that, calculation of the track for several x_{in} values (all the x_{in} inside the magnet) has being computed using the Euler method shown in appendix A. Next we have fitted the curve x_{out} versus x_{in}. As we expected the polynomial order needed wasn't small and we have being forced to use up to the seventh order to get a similar curve. As we already knew from experimental results, when the initial velocity changes the polynomial function coefficients change too. This has led us to the idea that the coefficients are not constant but they change with the initial v_x. In order to have this solved we have repeated the process for some different values of the initial velocity. Due to having a seventh order polynomial function we have done a fitting for each of the coefficients over the velocities.

In the end we have finished with a polynomial function which changes in terms of initial x position and velocity.

Overall, we have come up with a method that works for arbitrary fields. Unfortunately it is hard to say if it is going to be faster or not. As it has been said we need to compute any numerical method before we can figure out the polynomial. As a result we will need at least the same amount of computational time in the beginning. I believe there is potential for cases in which the set up is not changed for long periods. We could do the calculations in the first time periods and then we could get the next results with ease and rapidly. One of the benefits it has is that it can be used in parallel programming since all the measurements are independent.

Summing up, we have a method that works but needs of long periods of unaltered set up in order to be profitable.

Appendix A

Euler Method

When trying to solve ordinary differential equations with a given initial value one of the first and more simple methods that comes to the mind is the Euler Method. It is a first order method and therefore the local error or error per step is going to be proportional to the step size. At the same time the error at a given time (global error) is proportional to the step size. If a small enough step size is used highly precise tracks can be determined.

We need an ODE of the next form to start with the Euler Method:

$$\begin{cases} y'(t) = f(t, y(t)) \\ y(t_0) = y_0 \end{cases}$$

where y_0 is the initial value. For a fixed step size we can define the time as $t_n = t_0 + nh$. Moreover, if the that time expression is applied we can put down the distance between to neighbour points in the next way:

$$y_{n+1} = y_n + hf(t_n, y_n)$$

To translate this into our problem with the charged particle we will use the Lorentz forces (sartu ekuazioa):

$$\begin{cases} \dot{\mathbf{v}}(t) = \frac{q}{\gamma m}\left[\mathbf{v}(t) \times \mathbf{B}(t)\right] \\ \mathbf{v}(t_0) = \mathbf{v_0} \end{cases}, \begin{cases} \dot{\mathbf{x}}(t) = \mathbf{v}(t) \\ \mathbf{x}(t_0) = \mathbf{x_0} \end{cases}$$

Translating it to the Euler method we have

$$\mathbf{v}_{n+1} = \mathbf{v}_n + h \cdot \frac{q}{\gamma m}\left[\mathbf{v}(t) \times \mathbf{B}(t)\right] \tag{A.1}$$

and

$$\mathbf{x}_{n+1} = \mathbf{x}_n + h \cdot \mathbf{v}_n \tag{A.2}$$

Both equations are solved in the following c++ code in order to track the trajectory of the electron.

...

```
#include <iostream>
#include <iomanip>
#include <cmath>
#include <TNtuple.h>
#include <TGraph.h>
#include <TMultiGraph.h>
#include <TAxis.h>
#include <TCanvas.h>
```

```
#include <TLegend.h>

using namespace std;

// MAGNETIC FIELD:

Double_t Bzx(Double_t x){
    return exp(-pow(x,2)/(5.5));
}
Double_t Bzy(Double_t y){
    return exp(-pow(y-5,2)/(5.5));
}

void MF()
{

    // CONSTANTS:

    Double_t clight=299792458; // In m/s
    Double_t mass=9.10938356*pow(10,-31); // In kg
    Double_t con=5.344286*pow(10,-19); // To go from Gev/c to kg m/s
    Double_t q=-1.602*pow(10,-19); // In C

    Double_t Bz[10000]; // We are gonna use 1.05 T

    // MOMENTUM=1.5 AND INITIAL VELOCITY:

    Double_t py=1.5; // In GeV/c
    Double_t vx[10000], vy[10000], vz[10000];
    vx[0]=0;
    vy[0]=py*con/(mass*sqrt(1+pow(py*con,2)/(pow(mass,2)*pow(clight,2))));
    vz[0]=0;

    Double_t gamma=1/sqrt(1-pow(sqrt(pow(vy[0],2)+pow(vx[0],2)),2)/pow(clight,2));

    // INITIAL POSITION AND MAGNETIC FIELD:

    Double_t xx[100000], xy[100000], xz[100000];
    xx[0]=0;
    xy[0]=0;
    xz[0]=0.0;
    Bz[0]=1.05*Bzx(xx[0])*Bzy(xy[0]);

    // STEP SIZE:

    Double_t time = pow(10,-7.5); // In seconds
    int steps = pow(10,4.99);
    Double_t h = time/steps;

    // EULER METHOD AND PLOTTING:


    int i=0;

    while(i<steps){
        vx[i+1]=vx[i]+h*q/(mass*gamma)*vy[i]*Bz[i];
        vy[i+1]=vy[i]-h*q/(mass*gamma)*vx[i]*Bz[i];
        vz[i+1]=vz[i];

        xx[i+1]=xx[i]+h*vx[i];
        xy[i+1]=xy[i]+h*vy[i];
        xz[i+1]=xz[i]+h*vz[i];

        i=i+1;

        Bz[i]=1.05*Bzx(xx[i])*Bzy(xy[i]);
        gamma=1/sqrt(1-pow(sqrt(pow(vy[i],2)+pow(vx[i],2)),2)/pow(clight,2));
    }

    TGraph* tg1=new TGraph(steps,xy,xx);
    tg1->SetTitle("py=1.5 GeV/c");
    tg1->SetMarkerStyle(7);
    tg1->SetLineColor(kBlue);
    tg1->SetMarkerColor(kBlue);

    // MOMENTUM=2.5 AND INITIAL VELOCITY:

    py=2.5; // In GeV/c
    vx[0]=0;
    vy[0]=py*con/(mass*sqrt(1+pow(py*con,2)/(pow(mass,2)*pow(clight,2))));
    vz[0]=0;

    gamma=1/sqrt(1-pow(sqrt(pow(vy[0],2)+pow(vx[0],2)),2)/pow(clight,2));

    // INITIAL POSITION AND MAGNETIC FIELD:

    xx[0]=0;
    xy[0]=0;
    xz[0]=0.0;
    Bz[0]=1.05*Bzx(xx[0])*Bzy(xy[0]);

    // EULER METHOD AND PLOTTING:

    i=0;

    while(i<steps){
        vx[i+1]=vx[i]+h*q/(mass*gamma)*vy[i]*Bz[i];
        vy[i+1]=vy[i]-h*q/(mass*gamma)*vx[i]*Bz[i];
        vz[i+1]=vz[i];
        xx[i+1]=xx[i]+h*vx[i];
        xy[i+1]=xy[i]+h*vy[i];
        xz[i+1]=xz[i]+h*vz[i];

        i=i+1;

        Bz[i]=1.05*Bzx(xx[i])*Bzy(xy[i]);
        gamma=1/sqrt(1-pow(sqrt(pow(vy[i],2)+pow(vx[i],2)),2)/pow(clight,2));
}

TGraph* tg2=new TGraph(steps,xy,xx);
tg2->SetTitle("py=2.5 GeV/c");
tg2->SetMarkerStyle(7);
tg2->SetLineColor(kYellow);
tg2->SetMarkerColor(kYellow);

// MOMENTUM=7.5 AND INITIAL VELOCITY:

 py=7.5; // In GeV/c
vx[0]=0;
vy[0]=py*con/(mass*sqrt(1+pow(py*con,2)/(pow(mass,2)*pow(clight,2))));
vz[0]=0;

gamma=1/sqrt(1-pow(sqrt(pow(vy[0],2)+pow(vx[0],2)),2)/pow(clight,2));

// INITIAL POSITION AND MAGNETIC FIELD:

xx[0]=0;
xy[0]=0;
xz[0]=0.0;
Bz[0]=1.05*Bzx(xx[0])*Bzy(xy[0]);

// EULER METHOD AND PLOTTING:

i=0;

while(i<steps){
        vx[i+1]=vx[i]+h*q/(mass*gamma)*vy[i]*Bz[i];
        vy[i+1]=vy[i]-h*q/(mass*gamma)*vx[i]*Bz[i];
        vz[i+1]=vz[i];

        xx[i+1]=xx[i]+h*vx[i];
        xy[i+1]=xy[i]+h*vy[i];
        xz[i+1]=xz[i]+h*vz[i];

        i=i+1;

        Bz[i]=1.05*Bzx(xx[i])*Bzy(xy[i]);
        gamma=1/sqrt(1-pow(sqrt(pow(vy[i],2)+pow(vx[i],2)),2)/pow(clight,2));
}

TGraph* tg3=new TGraph(steps,xy,xx);
tg3->SetTitle("py=7.5 GeV/c");
tg3->SetMarkerStyle(7);
tg3->SetLineColor(kGreen);
tg3->SetMarkerColor(kGreen);

// MOMENTUM=25 AND INITIAL VELOCITY:

py=25; // In GeV/c
vx[0]=0;
vy[0]=py*con/(mass*sqrt(1+pow(py*con,2)/(pow(mass,2)*pow(clight,2))));
vz[0]=0;

gamma=1/sqrt(1-pow(sqrt(pow(vy[0],2)+pow(vx[0],2)),2)/pow(clight,2));

// INITIAL POSITION AND MAGNETIC FIELD:

xx[0]=0;
xy[0]=0;
xz[0]=0.0;
Bz[0]=1.05*Bzx(xx[0])*Bzy(xy[0]);

// EULER METHOD AND PLOTTING:

i=0;

while(i<steps){
        vx[i+1]=vx[i]+h*q/(mass*gamma)*vy[i]*Bz[i];
        vy[i+1]=vy[i]-h*q/(mass*gamma)*vx[i]*Bz[i];
        vz[i+1]=vz[i];

        xx[i+1]=xx[i]+h*vx[i];
        xy[i+1]=xy[i]+h*vy[i];
        xz[i+1]=xz[i]+h*vz[i];

        i=i+1;

        Bz[i]=1.05*Bzx(xx[i])*Bzy(xy[i]);
        gamma=1/sqrt(1-pow(sqrt(pow(vy[i],2)+pow(vx[i],2)),2)/pow(clight,2));
}

TGraph* tg4=new TGraph(steps,xy,xx);
tg4->SetTitle("py=25 GeV/c");
tg4->SetMarkerStyle(7);
tg4->SetLineColor(kOrange);
tg4->SetMarkerColor(kOrange);
```

```
    // MOMENTUM=100 AND INITIAL VELOCITY:

    py=100; // In GeV/c
    vx[0]=0;
    vy[0]=py*con/(mass*sqrt(1+pow(py*con,2)/(pow(mass,2)*pow(clight,2))));
    vz[0]=0;

    gamma=1/sqrt(1-pow(sqrt(pow(vy[0],2)+pow(vx[0],2)),2)/pow(clight,2));

    // INITIAL POSITION AND MAGNETIC FIELD:

    xx[0]=0;
    xy[0]=0;
    xz[0]=0.0;
    Bz[0]=1.05*Bzx(xx[0])*Bzy(xy[0]);

    // EULER METHOD AND PLOTTING:

    i=0;

    while(i<steps){
        vx[i+1]=vx[i]+h*q/(mass*gamma)*vy[i]*Bz[i];
        vy[i+1]=vy[i]-h*q/(mass*gamma)*vx[i]*Bz[i];
        vz[i+1]=vz[i];

        xx[i+1]=xx[i]+h*vx[i];
        xy[i+1]=xy[i]+h*vy[i];
        xz[i+1]=xz[i]+h*vz[i];

        i=i+1;

        Bz[i]=1.05*Bzx(xx[i])*Bzy(xy[i]);
        gamma=1/sqrt(1-pow(sqrt(pow(vy[i],2)+pow(vx[i],2)),2)/pow(clight,2));
    }

    TGraph* tg5=new TGraph(steps,xy,xx);
    tg5->SetTitle("py=100 GeV/c");
    tg5->SetMarkerStyle(7);
    tg5->SetLineColor(kRed);
    tg5->SetMarkerColor(kRed);

    //FINAL PLOT

    auto c1 = new TCanvas("c", "c", 600,500);
    TMultiGraph* mg=new TMultiGraph();

    mg->Add(tg1);
    mg->Add(tg2);
    mg->Add(tg3);
    mg->Add(tg4);
    mg->Add(tg5);

    mg->SetTitle(" ");
    mg->GetXaxis()->SetTitle("y [m]");
    mg->GetYaxis()->SetTitle("x [m]");
    mg->SetMinimum(-0.6);
    mg->Draw("AP");
    c1->BuildLegend();
}
```

...

Appendix B

c++ implementation code

B.1 Particle Tracking for some arbitrary v_x

...

```
#include <iostream>
#include <iomanip>
#include <cmath>
#include <TNtuple.h>
#include <TGraph.h>
#include <TMultiGraph.h>
#include <TAxis.h>
#include <TCanvas.h>
#include <TLegend.h>
#include <TStyle.h>

using namespace std;

Double_t Bzx(Double_t x){
    return exp(-pow(x,2)/(5.5));
}
Double_t Bzy(Double_t y){
    return exp(-pow(y-5,2)/(5.5));
}

void Poly()
{

    TCanvas *c1 = new TCanvas("c1","xout_vs_xin",700,500);
    c1->SetGrid();

    TMultiGraph *mg = new TMultiGraph();

    // CONSTANTS:

    Double_t clight=299792458; // In m/s
    Double_t mass=9.10938356*pow(10,-31); // In kg
    Double_t con=5.344286*pow(10,-19); // To go from Gev/c to kg m/s
    Double_t q=-1.602*pow(10,-19); // In C

    // STEP SIZE:

    Double_t time = pow(10,-7.5); // In seconds
    int steps = pow(10,4.99);
    Double_t h = time/steps;

    Double_t Bz[steps]; // We are gonna use 1.05 T



    // MOMENTUM=1.5 AND INITIAL VELOCITY:

    Double_t py=1.5; // In GeV/c

    Double_t vx[steps], vy[steps], vz[steps];
    vx[0]=0; // The maximum is 102140,96
    vy[0]=(py*con*clight)/sqrt(pow(py*con,2)+(pow(mass*clight,2)));
    vz[0]=0;

    Double_t gamma=1/sqrt(1-(pow(vx[0],2)+pow(vy[0],2))/pow(clight,2));

    // INITIAL POSITION AND MAGNETIC FIELD:

    Int_t n=1000; // NUumber of x positions wanted
    Double_t h2=11.0/n;


    Double_t xx[steps], xy[steps], xz[steps], xin1[n], xout1[n];
    Double_t xin2[n], xout2[n], xin3[n], xout3[n], xin4[n], xout4[n];
    xx[0]=-5.5;
    xy[0]=0;
    xz[0]=0;
    Bz[0]=1.05*Bzx(xx[0])*Bzy(xy[0]);
    xin1[0]=xx[0];
    xout1[0]=0;
    xin2[0]=xx[0];
    xout2[0]=0;
    xin3[0]=xx[0];
    xout3[0]=0;
    xin4[0]=xx[0];
    xout4[0]=0;

    int i=0;
    int j=0;

    // EULER METHOD AND PLOTTING:

    for (int k=0; k<=3; k++) {
        while (j<n) {
            if (k==1) {
                vx[0]=pow(10,8);
            }
            else if (k==2){
                vx[0]=-pow(10,8);

            }
            else if (k==3){
                vx[0]=-7.5*pow(10,7);
            }
            while(i<steps){
                vx[i+1]=vx[i]+h*q/(mass*gamma)*vy[i]*Bz[i];
                vy[i+1]=vy[i]-h*q/(mass*gamma)*vx[i]*Bz[i];
                vz[i+1]=vz[i];

                xx[i+1]=xx[i]+h*vx[i];
                xy[i+1]=xy[i]+h*vy[i];
                xz[i+1]=xz[i]+h*vz[i];
                i=i+1;

                Bz[i]=1.05*Bzx(xx[i])*Bzy(xy[i]);
            }

            if (k==0) {
                xin1[j]=xx[0];
                xout1[j]=xx[steps];
                cout<<"␣k␣=␣"<<k<<"␣working␣for␣j␣=␣"<<j<<endl;
            }
            else if (k==1){
                xin2[j]=xx[0];
                xout2[j]=xx[steps];
                cout<<"␣k␣=␣"<<k<<"␣working␣for␣j␣=␣"<<j<<endl;
            }
            else if (k==2){
                xin3[j]=xx[0];
                xout3[j]=xx[steps];
                cout<<"␣k␣=␣"<<k<<"␣working␣for␣j␣=␣"<<j<<endl;
            }
            else if (k==3){
                xin4[j]=xx[0];
                xout4[j]=xx[steps];
                cout<<"␣k␣=␣"<<k<<"␣working␣for␣j␣=␣"<<j<<endl;
            }


            xx[0]=-5.5+h2*(j+1);
            i=0;
            j=j+1;
        }

        j=0;
    }

    // Creating the graphs:

    //gStyle->SetOptFit(1);

    TGraph *g1 = new TGraph(n,xin1,xout1);
    g1->SetTitle("vx␣=␣0␣[m/s]");
    g1->SetLineColor(kBlue);
    g1->SetMarkerColor(kBlue);
    g1->SetMarkerStyle(7);
    mg->Add(g1);

    TGraph *g2 = new TGraph(n,xin2,xout2);
    g2->SetTitle("vx␣=␣pow(10,8)␣[m/s]");
    g2->SetLineColor(kRed);
    g2->SetMarkerColor(kRed);
    g2->SetMarkerStyle(7);
    mg->Add(g2);

    TGraph *g3 = new TGraph(n,xin3,xout3);
    g3->SetTitle("vx␣=␣-pow(10,8)␣[m/s]");
    g3->SetLineColor(kGreen);
    g3->SetMarkerColor(kGreen);
    g3->SetMarkerStyle(7);
    mg->Add(g3);

    TGraph *g4 = new TGraph(n,xin4,xout4);
    g4->SetTitle("vx␣=␣-7.5*pow(10,7)␣[m/s]");
    g4->SetLineColor(kOrange);
    g4->SetMarkerColor(kOrange);
    g4->SetMarkerStyle(7);
    mg->Add(g4);

    mg->Draw("ap");
    mg->GetXaxis()->SetTitle("X_initial_[m]");
    mg->GetYaxis()->SetTitle("X_final_[m]");

    c1->BuildLegend();
    //mg->Fit("pol4","FQ","",-5.5,5.5);
    gPad->Update();
    gPad->Modified();

}
```

...

B.2 Fitting Curves

...

```
#include <iostream>
#include <iomanip>
#include <cmath>
#include <TNtuple.h>
#include <TGraph.h>
#include <TMultiGraph.h>
#include <TAxis.h>
#include <TCanvas.h>
#include <TLegend.h>
#include <TStyle.h>

using namespace std;

void Fit(){

    TCanvas *c1 = new TCanvas("c1","xout_vs_xin",700,500);
    c1->SetGrid();

    TMultiGraph *mg = new TMultiGraph();

    Double_t a[4]={-0.0208,-0.01533,0.00196,-0.00144};
    Double_t velocity[4]={0,pow(10,8),-pow(10,8),-7.5*pow(10,7)};


    gStyle->SetOptFit(1);

    TGraph *g = new TGraph(4,velocity,a);
    g->SetMarkerColor(kBlue);
    g->SetMarkerStyle(kFullCircle);
    mg->Add(g);


    mg->Draw("ap");
    mg->GetXaxis()->SetTitle("V_initial_[m/s]");
    mg->GetYaxis()->SetTitle("a(x')");
    mg->Fit("pol3","FQ","",-pow(10,8),pow(10,8));
    gPad->Update();
    gPad->Modified();

}
```

...

Bibliography

[1] CERN. *Cern Root*. 2018. URL: https://root.cern.ch/.

[2] CERN. *Large Hadron Collider*. 2019. URL: https://home.cern/science/accelerators/large-hadron-collider.

[3] CERN. *LHCb Detector Performance*. 2014. URL: https://arxiv.org/pdf/1412.6352.pdf.

[4] CERN. *LHCb Dipole Magnet*. URL: http://lhcb-magnet.web.cern.ch/lhcb-magnet/.

[5] CERN. *LHCb Magnetic Field*. URL: http://lhcb-magnet.web.cern.ch/lhcb-magnet/.

[6] CERN. *Processing data in LHC*. 2019. URL: https://home.cern/science/computing/processing-what-record.