THE HEBREW UNIVERSITY OF JERUSALEM

Electricity and Magnetism

Numerical Analysis of Wien Filter Velocity Selector

Amit Roth
Amit Harel

1. Introduction

We will solve at first the following problem.

Suppose we have a charged particle with a charge q, moving under the influence of a constant electric field and a constant magnetic field.

$$\begin{cases} \overrightarrow{E}(\vec{r}) = E\hat{y} \\ \overrightarrow{B}(\vec{r}) = -B\hat{x} \end{cases}$$

In $t = 0$, the particle has the following velocity, $v(t = 0) = u\hat{z}$ and $u = \dfrac{3E}{B}$.

Solution

Let's start calculating the force on such a particle. The magnetic field points in $\hat{x}$ and the velocity in $\hat{z}$. We can reduce our problem in to 2-D problem on the zy plane. We will get:

$$\begin{cases} \overrightarrow{v} = v_y\hat{y} + v_z\hat{z} \\ \overrightarrow{B} = -B\hat{x} \end{cases} \quad \Rightarrow \quad q\overrightarrow{v} \times \overrightarrow{B} = qv_yB\hat{z} - qv_zB\hat{y}$$

So the net force will be

$$\overrightarrow{F} = (qE - qv_zB)\hat{y} + qv_yB\hat{z}$$

And from newton's second law

$$\begin{cases} m\dot{v}_y = qE - qv_zB \\ m\dot{v}_z = qBv_y \end{cases}$$

We can derive from the second equation

$$m\ddot{v}_z = qB\dot{v}_y \Rightarrow \dot{v}_y = \frac{m}{qB}\ddot{v}_z$$

We will substitute our result for $\dot{v}_z$ in the first equation and we will get

$$\frac{m^2}{q^2B^2}\ddot{v}_z = \frac{E}{B} - v_z$$

We got an harmonic oscillator for v_z, the solution is

$$v_z = Acos(\omega t + \phi) + \frac{E}{B}$$

We know that $v(t = 0) = u\hat{z}$, and we know that $\dot{v}_z(t = 0) = 0$. So the full solution for v_z is

$$v_z = \frac{2E}{B}cos(\frac{qB}{m}t) + \frac{E}{B}$$

From here we can derive v_y instantly using:

$$v_y = \frac{m}{qB}\dot{v}_z = \frac{m}{qB}\frac{-2E}{B}\frac{qB}{m}sin(\frac{qB}{m}t)$$

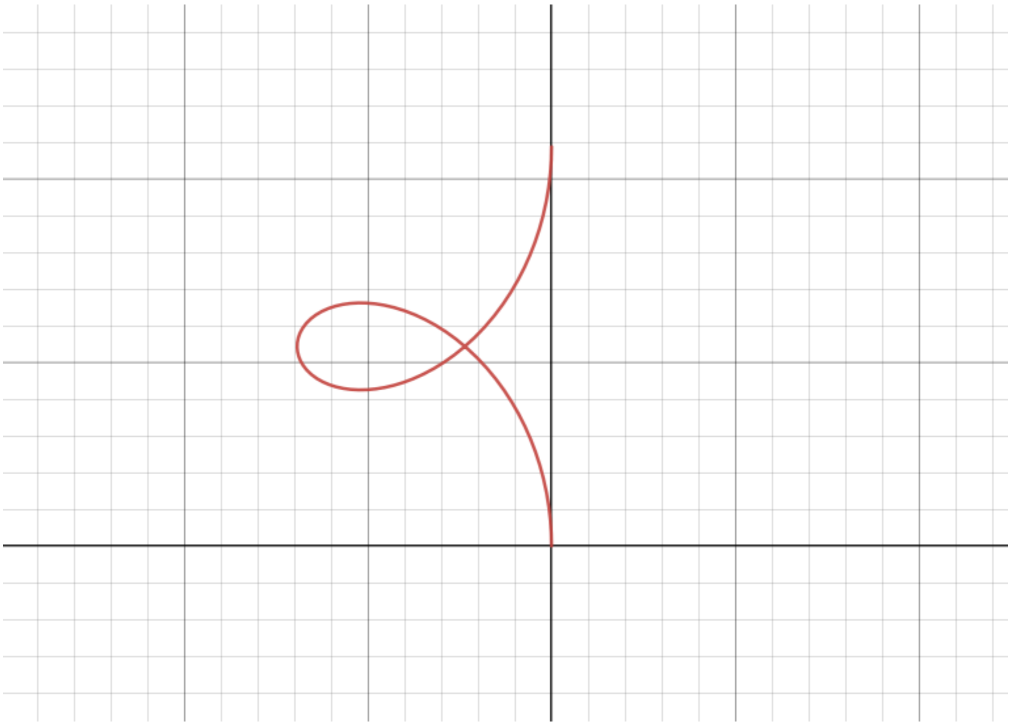
$$\Rightarrow v_y = \frac{-2E}{B}sin(\frac{qB}{m}t)$$

Now, in order to get r_y and r_z we will just integrate the velocities, using the given $\vec{r}(t=0) = 0$.

$$r_y = \int v_y dt = \frac{2mE}{qB^2} cos(\frac{qB}{m}t) - \frac{2mE}{qB^2}$$

$$r_z = \int v_z dt = \frac{2mE}{qB^2} sin(\frac{qB}{m}t) + \frac{E}{B}t$$

The graph for $\vec{r}(t)$ for $0 \leq t \leq \frac{2\pi}{\omega}$ is



2. Numerical Integration

Taylor First Order

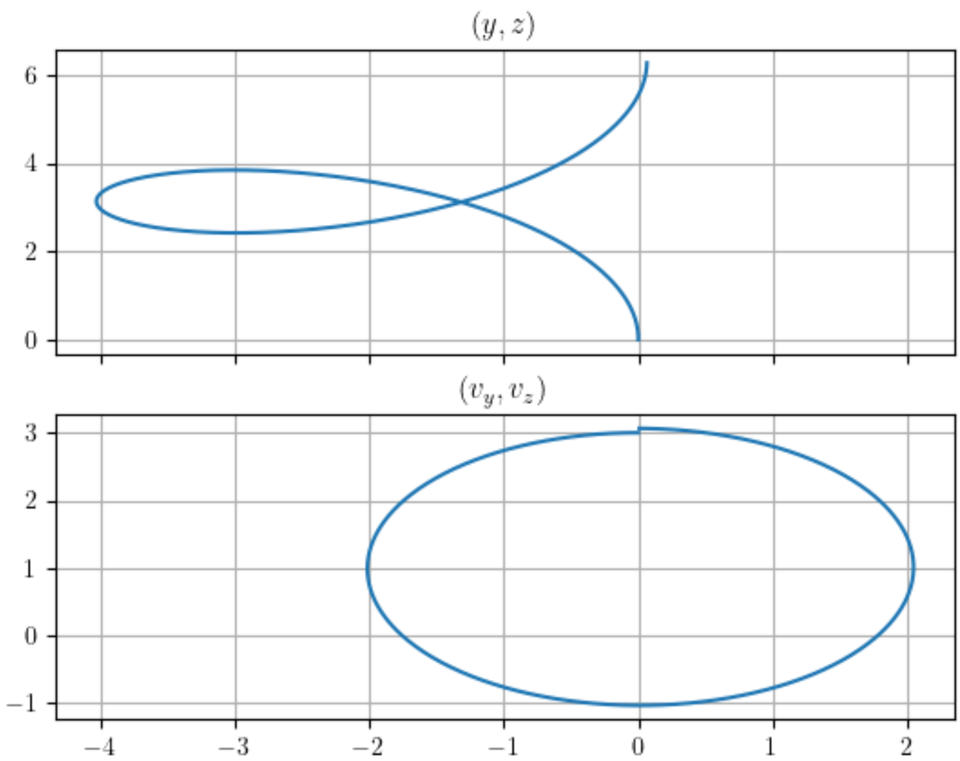
We will now solve our problem using numerical methods. Using first order Taylor approximation we will get the following relations:

$$\boldsymbol{r}(t+\Delta t) = \mathbf{r}(t) + \frac{d\boldsymbol{r}}{dt}\Delta t + O(\Delta t^2) = \mathbf{r}(t) + \boldsymbol{v}(t)\Delta t + O(\Delta t^2)$$

$$\boldsymbol{v}(t+\Delta t) = \boldsymbol{v}(t) + \frac{d\boldsymbol{v}}{dt}\Delta t + O(\Delta t^2) = \boldsymbol{v}(t) + \boldsymbol{a}(t)\Delta t + O(\Delta t^2)$$

The code is implemented in the file numerical_integration.py, and also appended below in the code appendix.

The output graphs are:



We can see that the graph (y, z) has the same form as the graph that we got from the analytical solution.

Midpoint

We will now use a more precise approximation. Using the midpoint technique, with:

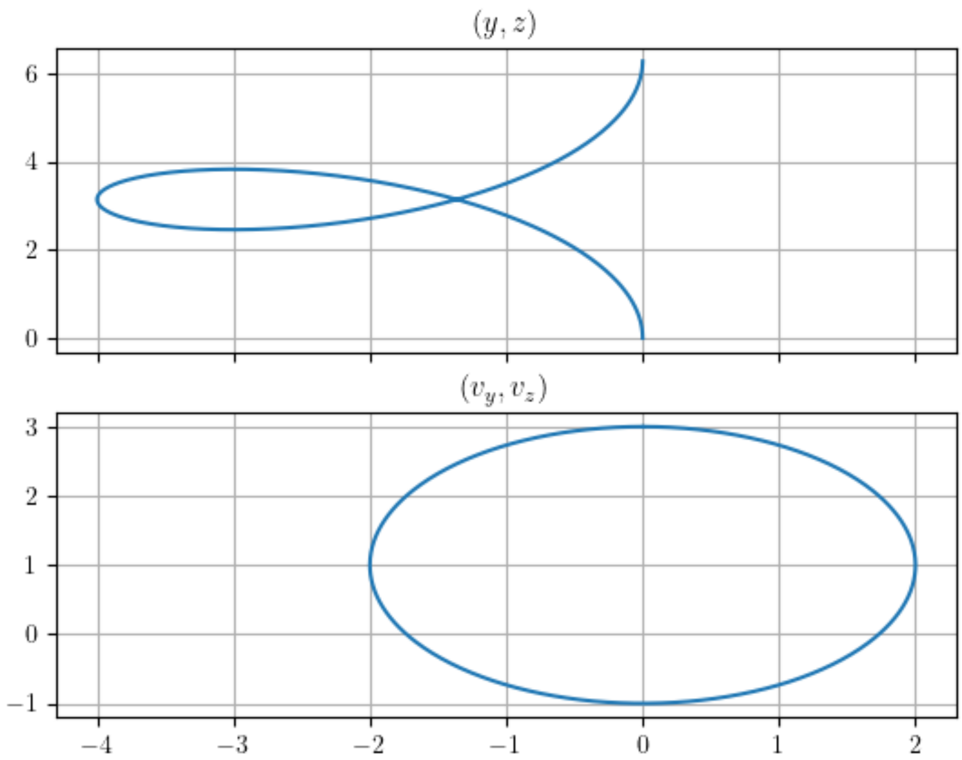
$$\begin{cases} f(t, y) = \frac{\partial f}{\partial t} \\ k_1 = \Delta t \cdot f(t, y(t)) \\ k_2 = \Delta t \cdot f(t + \frac{\Delta t}{2}, y(t + \frac{k_1}{2})) \end{cases}$$

And we will calculate the result using

$$y_{n+1} = y_n + k_2 + O(\Delta t^3)$$

The code is implemented in the file numerical_integration.py, and also appended below in the code appendix.

The output graphs are here:



We can see that the graphs are smoother.

Runge-Kutta

A more accurate technique can be implemented using:

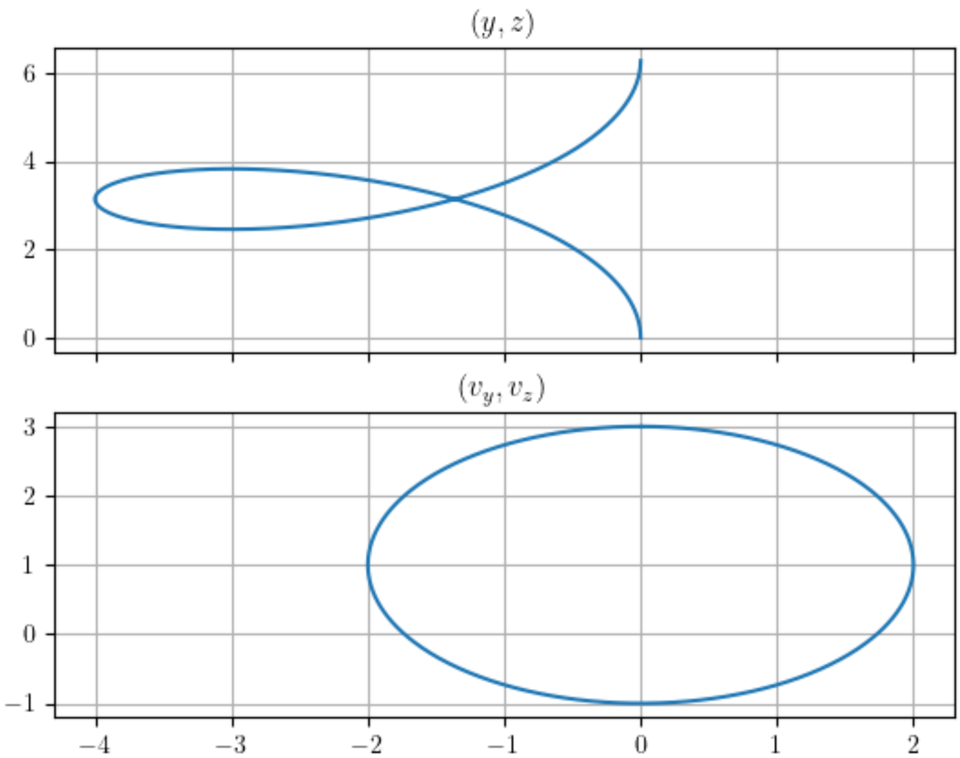
$$\begin{cases} k_1 = \Delta t \cdot f(t, y(t)) \\ k_2 = \Delta t \cdot f(t + \frac{\Delta t}{2}, y(t + \frac{k_1}{2})) \\ k_3 = \Delta t \cdot f(t + \frac{\Delta t}{2}, y(t + \frac{k_2}{2})) \\ k_3 = \Delta t \cdot f(t + \Delta t, y(t + k_3)) \end{cases}$$

And we will get

$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) + O(\Delta t^5)$$

The code is implemented as well in the file numerical_integration.py, and also appended below in the code appendix.

The output graphs are similar to the midpoint graphs:



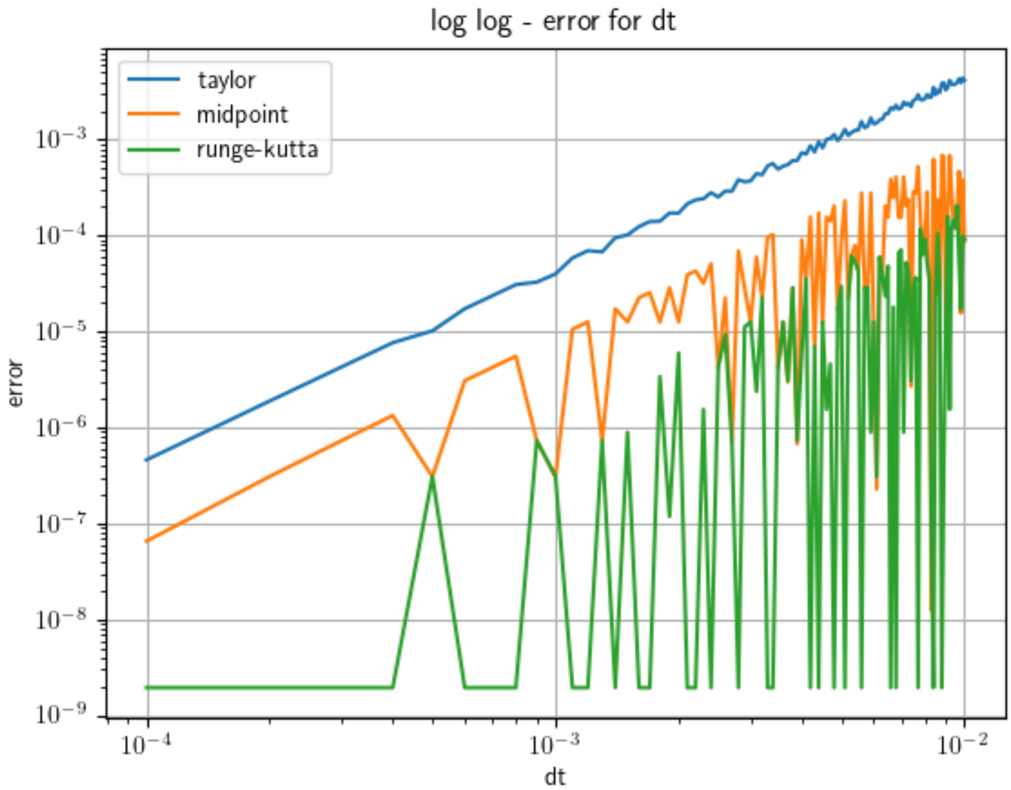
Benchmakrs

We will calculate the point that the particle will be in $T = \dfrac{2\pi}{\omega}$.

In the analytical solution we get:

$$\begin{cases} r_z(T) = \frac{2mE\pi}{qB^2} \\ r_y(T) = 0 \end{cases}$$

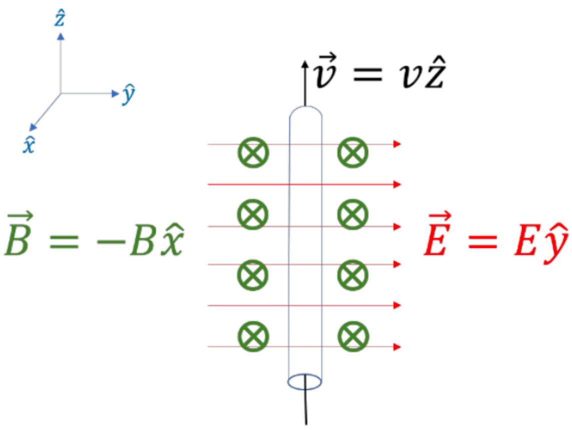
We will calculate the error (euclidian distance from the analytical to the numeric) and plot it against dt. The graph in log log scale is:



We can see that midpoint and range have similar results, but Taylor is much worse.

3. Wien Filter Velocity Selector

We will now deal with the following problem of Wien filter velocity selector.



We already solved the problem analytically in the first section.

Assume we have beam of protons traveling with average kinetic energy $E_0 = 5Mev = 8.0109 \cdot 10^{-13}J$, and pipe of length $l = 1m$ and radius $r = 3mm$.

The ratio $\frac{E}{B}$

The initial velocity we need to set in order to let the protons' beam to pass can be derived instantly from our solution and is $v_0 = \frac{E}{B}\hat{z}$

Solving for the path of the particles

We will solve the same problem, but taking into consideration with the initial energy and the initial coordinates in the pipe.

Assume for all particles $E_{initial} \in [E_0 - \delta E, E_0 + \delta E]$ for $\delta E = 0.25[Mev]$ and $E_0 = 5[Mev]$ and $y_0 \in [-R, R]$ for $R = 0.003[m]$. We can derive the velocity from the energy using $v_0 = \sqrt{\frac{2E_0}{m}}$. We will get:

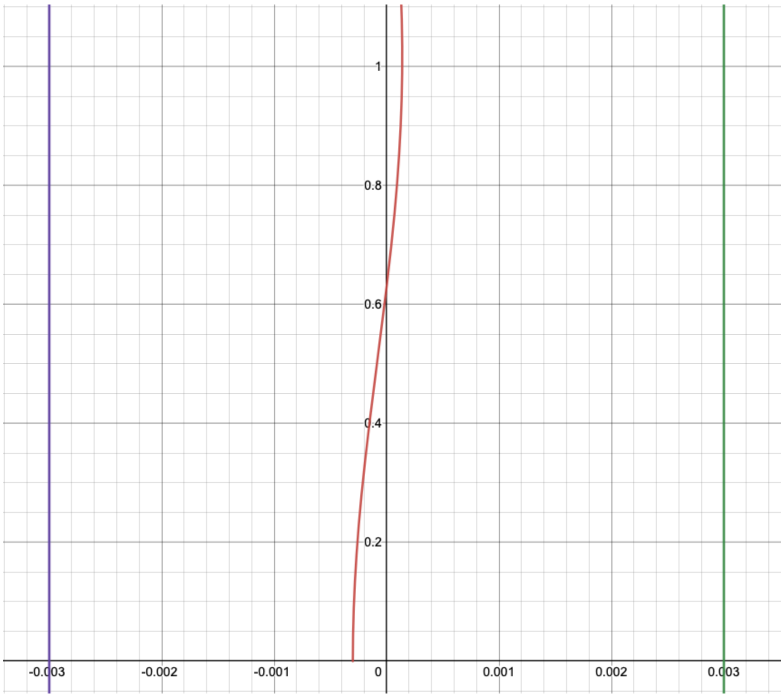
$$\begin{cases} v_z = (v_0 - \frac{E}{B})cos(\frac{qB}{m}t) + \frac{E}{B} \\ v_y = (\frac{E}{B} - v_0)sin(\frac{qB}{m}t) \end{cases}$$

And after integration,

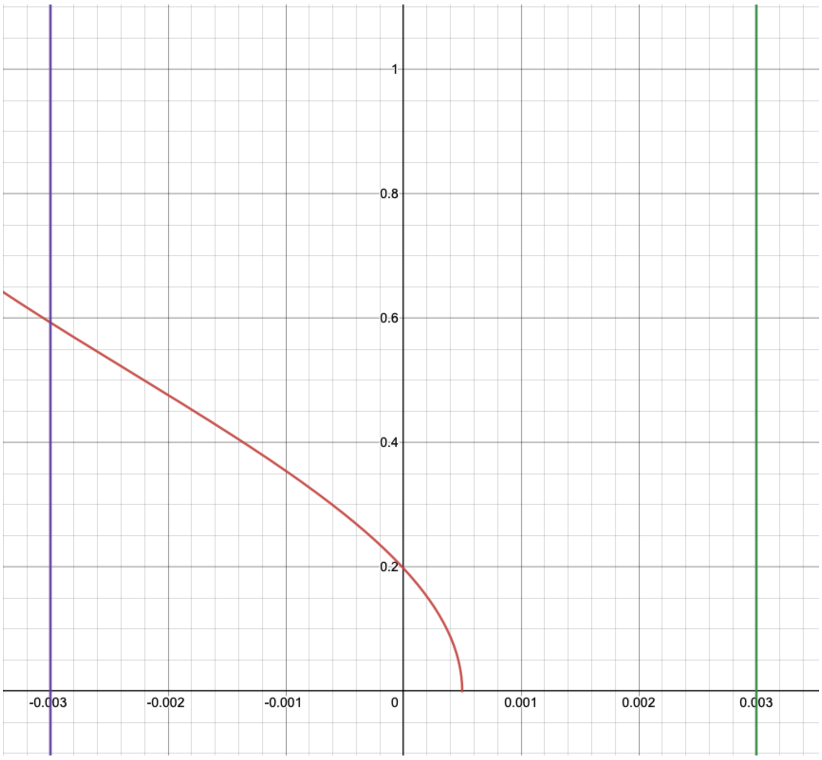
$$\begin{cases} r_z = (v_0 - \frac{E}{B})\frac{m}{qB}sin(\frac{qB}{m}t) + \frac{E}{B}t \\ r_y = \frac{m}{qB}(v_0 - \frac{E}{B})(cos(\frac{qB}{m}t) - 1) + y_0 \end{cases}$$

Before we will show the numerical solution, we plotted the analytical solution in desmos. The link to the graph is in the appendix, it is very beautiful to change the parameters and observe how the graph changes.

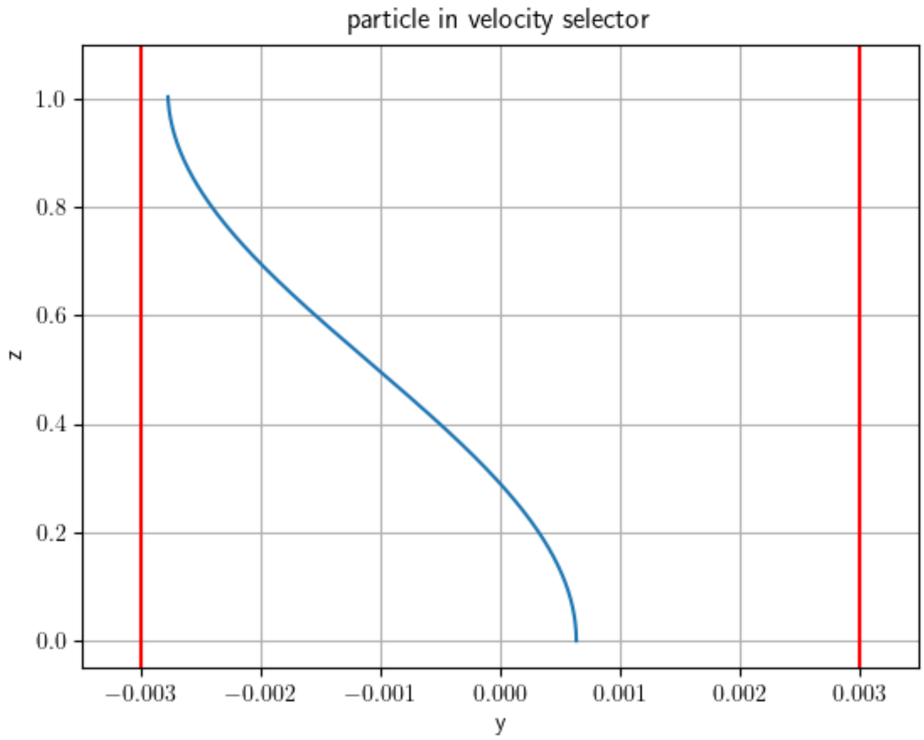
The first graph is for a particle with $E_0 = 0.8 \cdot 10^{-13}[J]$ and $y_0 = -3 \cdot 10^{-4}[m]$, we can see that the particle passes the velocity selector.

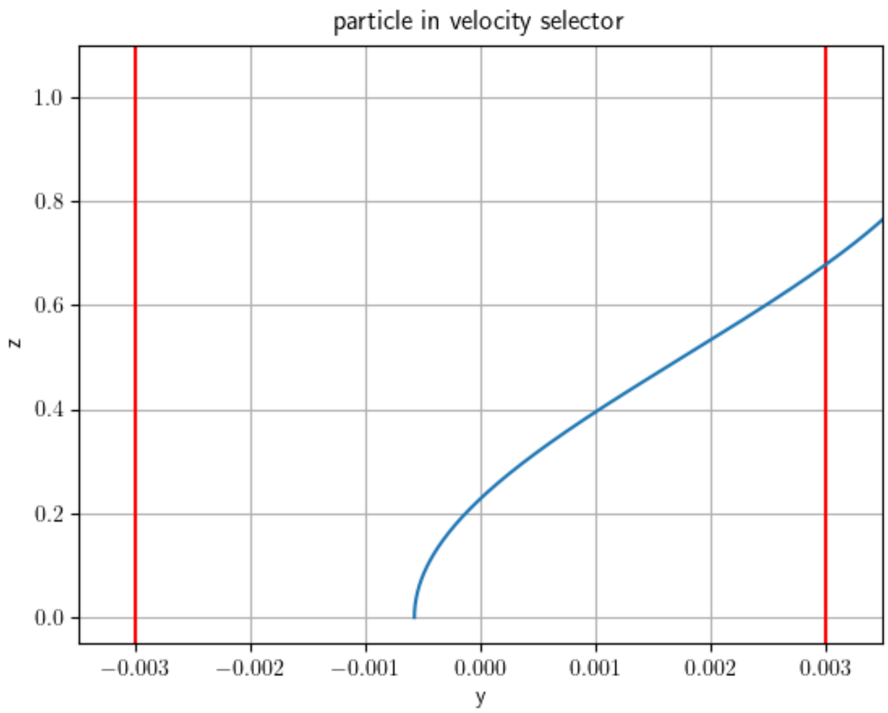


And a particle with $E_0 = 0.815 \cdot 10^{-13}[J]$ and $y_0 = 5 \cdot 10^{-4}[m]$ won't pass the velocity selector.



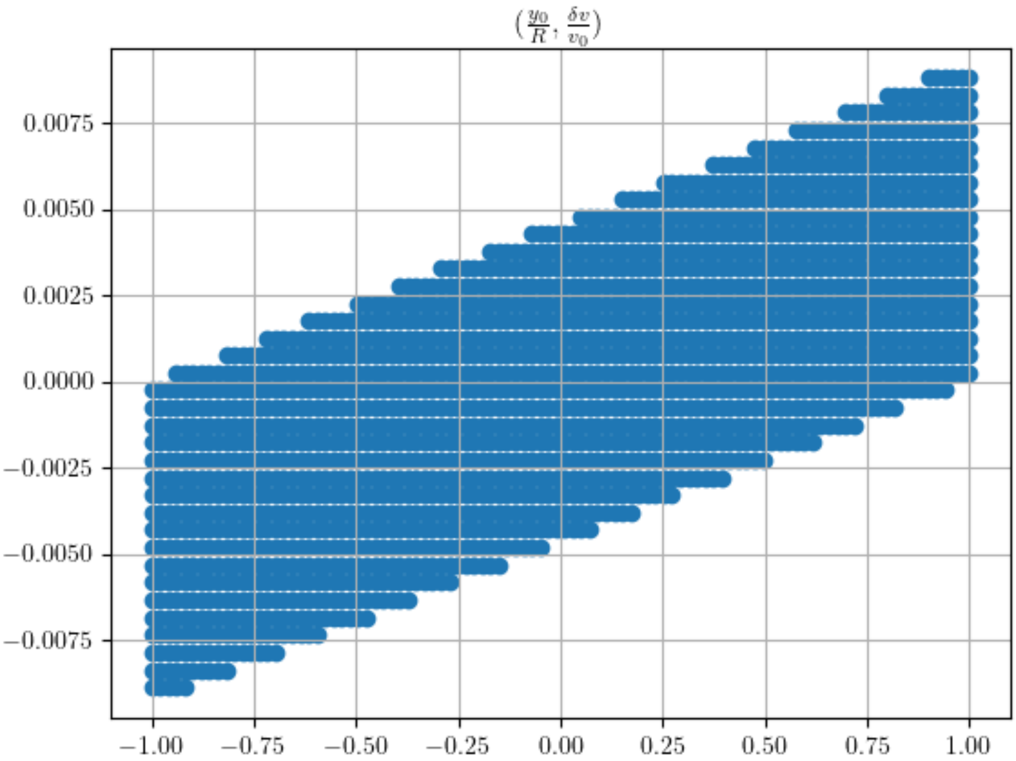
We will plot this equations in wien_filter_numerical_integration.py and observe the routes. The red lines are standing as the pipe boundaries and the blue line is the route of the particle.





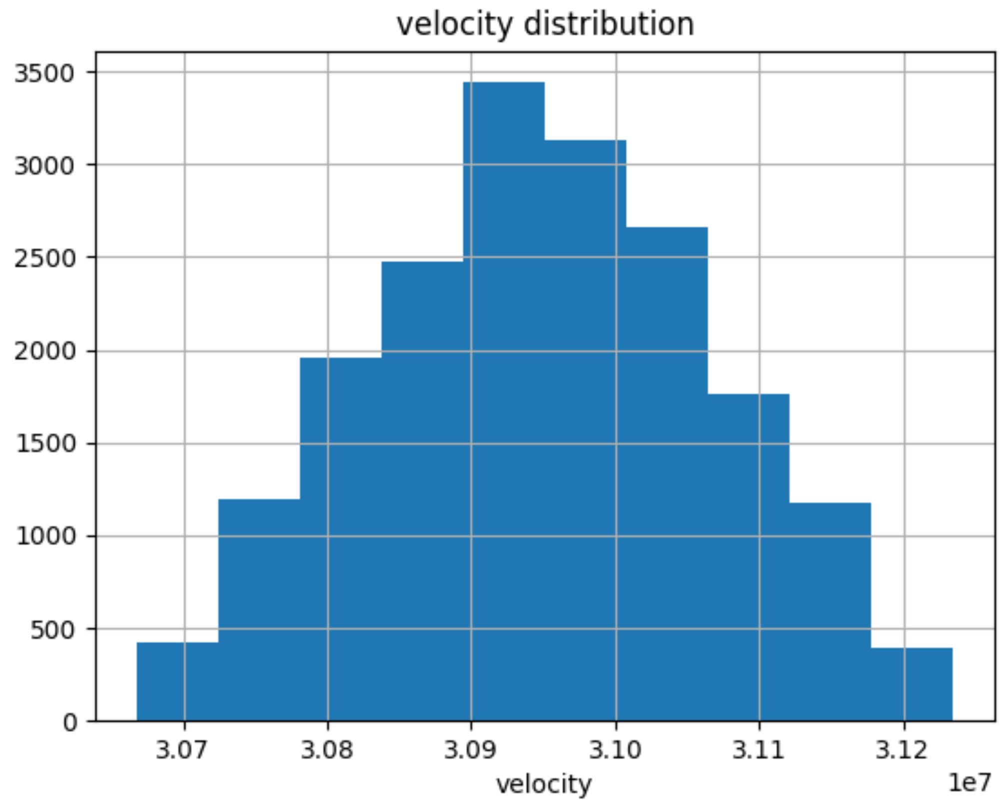
The plane $(\frac{y_0}{R}, \frac{\delta v}{v_0})$

We will observe the plane $(\frac{y_0}{R}, \frac{\delta v}{v_0})$, and plot the dots that the particle will pass the velocity selector. We got from the graph a parallelogram:



The protons beam

We will take a beam of 10^5 protons with $E_{initial} \in [E_0 - \delta E, E_0 + \delta E]$ for $\delta E = 0.25[Mev]$ and $E_0 = 5[Mev]$ and $y_0 \in [-R, R]$ for $R = 0.003[m]$ distributed evenly. We will calculate the distribution of the velocities of the particles that pass the velocity selector.



We got the following distribution of particles. We can see that the most of the particles who passed is with velocity v_0, and particles with similar velocities are passing with a lower rate. We can also see the symmetry between velocities which bigger than v_0 to velocities which are smaller then v_0.

Percent of particles which pass

We can calculate the particles that passed the velocity selector by summing the particles that have been passed, and divide by the total particles in the beam. We saw that the number of particles that passed is $n = 19{,}000$ and the total number of particles is $n_{tot} = 10^5$.

So the percent of particles which have been passed is

$$\frac{n}{n_{tot}} \cdot 100\,\% = \frac{19{,}000}{100{,}000} \cdot 100\,\% = 19\,\%\,.$$

Code Appendix

The full code can be found in the GitHub page.

Link for the desmos file with the analytic solution.

```
import math

E_0 = 8.0109 * 10**(-13)
delta_E = E_0/20

m = 1.672621898 * 10**(-27)  # [Kg]
q = 1.602176634 * 10**(-19)  # [C]
B = 1
E = math.sqrt((2*E_0)/m)*B  # [N/C]
R = 0.003  # [m]
```

```python
import math
import numpy as np
import matplotlib.pyplot as plt

import constants as c


def taylor_first_order(dt, gen_graph=False):
    """
        r(t+dt) = r(t) + v(t)dt
        v(t+dt) = v(t) + a(t)dt
    """
    omega = (c.q * c.B) / c.m
    T = (2 * math.pi) / omega
    num_of_time_intervals = math.ceil(T / dt)

    rz = np.zeros(num_of_time_intervals)
    ry = np.zeros(num_of_time_intervals)
    vz = np.zeros(num_of_time_intervals)
    vy = np.zeros(num_of_time_intervals)

    rz[0] = 0
    ry[0] = 0
    vz[0] = (3 * c.E) / c.B
    vy[0] = 0

    for i in range(1, num_of_time_intervals):
        rz[i] = rz[i-1] + vz[i-1] * dt
        ry[i] = ry[i - 1] + vy[i - 1] * dt
        vz[i] = vz[i - 1] + ((c.q * c.B * vy[i-1]) / c.m) * dt
        vy[i] = vy[i - 1] + ((c.q * c.E - c.q * c.B * vz[i-1]) / c.m) * dt

    if gen_graph:
        plt.rcParams['text.usetex'] = True

        figure, axis = plt.subplots(2, 1, sharex=True)

        axis[0].plot(ry, rz)
        axis[0].set_title(r"$(y, z)$")

        axis[1].plot(vy, vz)
        axis[1].set_title(r"$(v_y, v_z)$")

        axis[0].grid(True)
        axis[1].grid(True)
        plt.savefig('taylor_first_order.png')
        plt.show()

    return ry[num_of_time_intervals-1], rz[num_of_time_intervals-1]
```

```python
def midpoint(dt, gen_graph=False):
    omega = (c.q * c.B) / c.m
    T = (2 * math.pi) / omega
    num_of_time_intervals = math.ceil(T / dt)

    rz = np.zeros(num_of_time_intervals)
    ry = np.zeros(num_of_time_intervals)
    vz = np.zeros(num_of_time_intervals)
    vy = np.zeros(num_of_time_intervals)

    rz[0] = 0
    ry[0] = 0
    vz[0] = (3 * c.E) / c.B
    vy[0] = 0

    def az(vy):
        return (c.q * vy * c.B) / c.m

    def ay(vz):
        return (c.q * c.E - c.q * c.B * vz) / c.m

    for i in range(1, num_of_time_intervals):
        k1vz = az(vy[i-1]) * dt
        k1vy = ay(vz[i - 1]) * dt
        k2vz = az(vy[i-1] + 0.5 * k1vy) * dt
        k2vy = ay(vz[i - 1] + 0.5 * k1vz) * dt

        # k1rz = vz[i - 1] * dt
        # k1ry = vy[i - 1] * dt
        k2rz = (vz[i-1] + 0.5 * k1vz) * dt
        k2ry = (vy[i-1] + 0.5 * k1vy) * dt

        rz[i] = rz[i-1] + k2rz
        ry[i] = ry[i - 1] + k2ry
        vz[i] = vz[i - 1] + k2vz
        vy[i] = vy[i - 1] + k2vy

    if gen_graph:
        plt.rcParams['text.usetex'] = True

        figure, axis = plt.subplots(2, 1, sharex=True)

        axis[0].plot(ry, rz)
        axis[0].set_title(r"$(y, z)$")

        axis[1].plot(vy, vz)
        axis[1].set_title(r"$(v_y, v_z)$")

        axis[0].grid(True)
        axis[1].grid(True)
        plt.savefig('midpoint.png')
        plt.show()

    return ry[num_of_time_intervals-1], rz[num_of_time_intervals-1]
```

```python
def runge_kutta(dt, gen_graph=False):
    omega = (c.q * c.B) / c.m
    T = (2 * math.pi) / omega
    num_of_time_intervals = math.ceil(T / dt) + 1

    rz = np.zeros(num_of_time_intervals)
    ry = np.zeros(num_of_time_intervals)
    vz = np.zeros(num_of_time_intervals)
    vy = np.zeros(num_of_time_intervals)

    rz[0] = 0
    ry[0] = 0
    vz[0] = (3 * c.E) / c.B
    vy[0] = 0

    def az(vy):
        return (c.q * vy * c.B) / c.m

    def ay(vz):
        return (c.q * c.E - c.q * c.B * vz) / c.m


    for i in range(1, num_of_time_intervals):
        k1vz = az(vy[i-1]) * dt
        k1vy = ay(vz[i - 1]) * dt
        k2vz = az(vy[i-1] + 0.5 * k1vy) * dt
        k2vy = ay(vz[i - 1] + 0.5 * k1vz) * dt
        k3vz = az(vy[i - 1] + 0.5 * k2vy) * dt
        k3vy = ay(vz[i - 1] + 0.5 * k2vz) * dt
        k4vz = az(vy[i - 1] + k3vy) * dt
        k4vy = ay(vz[i - 1] + k3vz) * dt

        k1rz = vz[i - 1] * dt
        k1ry = vy[i - 1] * dt
        k2rz = (vz[i - 1] + 0.5 * k1vz) * dt
        k2ry = (vy[i - 1] + 0.5 * k1vy) * dt
        k3rz = (vz[i - 1] + 0.5 * k2vz) * dt
        k3ry = (vy[i - 1] + 0.5 * k2vy) * dt
        k4rz = (vz[i - 1] + k3vz) * dt
        k4ry = (vy[i - 1] + k3vy) * dt

        rz[i] = rz[i - 1] + (k1rz + 2 * k2rz + 2 * k3rz + k4rz) / 6
        ry[i] = ry[i - 1] + (k1ry + 2 * k2ry + 2 * k3ry + k4ry) / 6
        vz[i] = vz[i - 1] + (k1vz + 2 * k2vz + 2 * k3vz + k4vz) / 6
        vy[i] = vy[i - 1] + (k1vy + 2 * k2vy + 2 * k3vy + k4vy) / 6

    if gen_graph:
        plt.rcParams['text.usetex'] = True

        figure, axis = plt.subplots(2, 1, sharex=True)

        axis[0].plot(ry, rz)
        axis[0].set_title(r"$(y, z)$")

        axis[1].plot(vy, vz)
        axis[1].set_title(r"$(v_y, v_z)$")

        axis[0].grid(True)
        axis[1].grid(True)
        plt.savefig('runge_kutta.png')
        plt.show()

    if abs(ry[num_of_time_intervals-1]) > abs(ry[num_of_time_intervals-2]):
        return ry[num_of_time_intervals-2], rz[num_of_time_intervals-2]

    return ry[num_of_time_intervals-1], rz[num_of_time_intervals-1]
```

```python
def plot_error_graph(num_of_intervals, step):
    times = np.zeros(num_of_intervals)
    taylor = np.zeros(num_of_intervals)
    mid = np.zeros(num_of_intervals)
    runge = np.zeros(num_of_intervals)

    for i in range(num_of_intervals):
        times[i] = (i+1)*step
        taylor[i] = error(taylor_first_order(times[i]), c.analytic)
        mid[i] = error(midpoint(times[i]), c.analytic)
        runge[i] = error(runge_kutta(times[i]), c.analytic)

    print(times)
    print(taylor)
    print(mid)
    print(runge)

    plt.rcParams['text.usetex'] = True
    plt.plot(times, taylor, label="taylor")
    plt.plot(times, mid, label="midpoint")
    plt.plot(times, runge, label="runge-kutta")

    plt.ylabel("error")
    plt.xlabel("dt")
    plt.xscale("log")
    plt.yscale("log")
    plt.title("log log - error for dt")

    plt.grid(True)
    plt.legend()
    plt.savefig('error.png')

    plt.show()
```

```python
import math
import numpy as np
import matplotlib.pyplot as plt

import constants
import constants as c


def runge_kutta_passes_filter(dt, E_0, y_0, gen_graph=False):
    omega = (c.q * c.B) / c.m
    T = (2 * math.pi) / omega
    num_of_time_intervals = math.ceil(T / dt)

    rz = np.zeros(1)
    ry = np.zeros(1)
    vz = np.zeros(1)
    vy = np.zeros(1)

    rz[0] = 0
    ry[0] = y_0
    vz[0] = math.sqrt((2 * E_0)/c.m)
    vy[0] = 0

    def az(vy):
        print(vy)
        return (c.q * vy * c.B) / c.m

    def ay(vz):
        return (c.q * c.E - c.q * c.B * vz) / c.m

    i = 0
    while rz[i] <= 1:
        i += 1

        k1vz = az(vy[i - 1]) * dt
        k1vy = ay(vz[i - 1]) * dt
        k2vz = az(vy[i - 1] + 0.5 * k1vy) * dt
        k2vy = ay(vz[i - 1] + 0.5 * k1vz) * dt
        k3vz = az(vy[i - 1] + 0.5 * k2vy) * dt
        k3vy = ay(vz[i - 1] + 0.5 * k2vz) * dt
        k4vz = az(vy[i - 1] + k3vy) * dt
        k4vy = ay(vz[i - 1] + k3vz) * dt

        k1rz = vz[i - 1] * dt
        k1ry = vy[i - 1] * dt
        k2rz = (vz[i - 1] + 0.5 * k1vz) * dt
        k2ry = (vy[i - 1] + 0.5 * k1vy) * dt
        k3rz = (vz[i - 1] + 0.5 * k2vz) * dt
        k3ry = (vy[i - 1] + 0.5 * k2vy) * dt
        k4rz = (vz[i - 1] + k3vz) * dt
        k4ry = (vy[i - 1] + k3vy) * dt

        rz = np.append(rz, [rz[i - 1] + (k1rz + 2 * k2rz + 2 * k3rz + k4rz) / 6])
        ry = np.append(ry, ry[i - 1] + (k1ry + 2 * k2ry + 2 * k3ry + k4ry) / 6)
        vz = np.append(vz, vz[i - 1] + (k1vz + 2 * k2vz + 2 * k3vz + k4vz) / 6)
        vy = np.append(vy, vy[i - 1] + (k1vy + 2 * k2vy + 2 * k3vy + k4vy) / 6)

```

```
    if gen_graph:
        plt.rcParams['text.usetex'] = True

        plt.ylim(-0.05, 1.1)
        plt.xlim(-0.0035, 0.0035)

        plt.axvline(x=c.R, color='r', ymin= 0, ymax=1)
        plt.axvline(x=-c.R, color='r', ymin= 0, ymax=1)

        plt.plot(ry, rz)

        plt.ylabel("z")
        plt.xlabel("y")
        plt.title("particle in velocity selector")
        plt.grid(True)
        plt.savefig('runge_kutta_wien_filter.png')
        plt.show()

    return -c.R < ry[i] < c.R, vz[-1]



def error_plane():
    energy = np.linspace(c.E_0 - c.delta_E, c.E_0 + c.delta_E, num=100)
    radius = np.linspace(-c.R, c.R, num=100)

    output_velocity = []
    output_radius = []

    for e in energy:
        for r in radius:
            if runge_kutta_passes_filter(10**(-10), e, r,)[0]:
                output_velocity.append(math.sqrt(e/c.E_0)-1)
                output_radius.append(r/c.R)

    plt.rcParams['text.usetex'] = True
    plt.scatter(output_radius, output_velocity)
    plt.grid(True)
    plt.title(r"$(\frac{y_0}{R}, \frac{\delta v}{v_0})$")
    plt.grid(True)
    plt.savefig('error_plane.png')
    plt.show()
```

```python
def velocity_distribution(num_of_particles):
    energy = np.linspace(c.E_0 - c.delta_E, c.E_0 + c.delta_E, num=math.ceil(math.sqrt(num_of_particles)))
    radius = np.linspace(-c.R, c.R, num=math.ceil(math.sqrt(num_of_particles)))

    velocities = []
    i=0
    for e in energy:
        for r in radius:
            i+=1
            passes, vel = runge_kutta_passes_filter(10 ** (-9), e, r)
            if passes:
                velocities.append(vel)

            print(i)

    print(velocities)
    plt.hist(velocities)
    plt.rcParams['text.usetex'] = True
    plt.title(r"velocity distribution")
    plt.xlabel("velocity")
    plt.grid(True)
    plt.savefig('velocity_distribution.png')
    plt.show()
```