# lorentz

October 25, 2024

### 0.0.1 Applied EMT

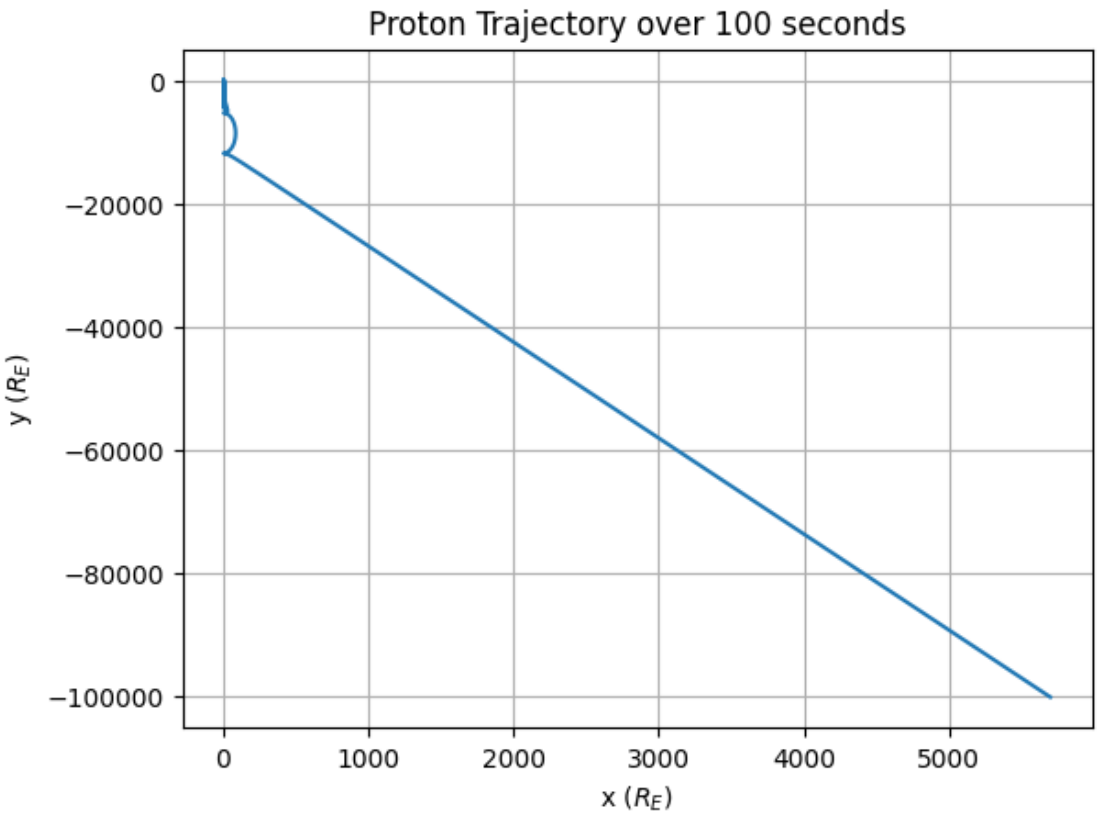
## 0.1 Dennies Bor

[2]:
```
import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm
```

[22]:
```
# Initial conditions
# Constants
q = 1.602e-19  # charge of proton (C)
m = 1.67e-27  # mass of proton (kg)
B0 = 3e-5  # magnetic field strength (T)
RE = 6.371e6  # Earth's radius (m)
x0 = 2 * RE  # initial position (m)
KE = 10 * 1.602e-13  # 10 MeV in joules
dt = 1e-6  # time step (s)
T = 100  # total time (s)
steps = int(T / dt)
v0x = np.sqrt(2 * KE / m)
x = x0
y = 0
vx = v0x
vy = 0

# Arrays to store trajectory
x_traj = np.zeros(steps)
y_traj = np.zeros(steps)

# Euler method using derived equations
for i in tqdm(range(steps)):

    # Store the current position
    x_traj[i] = x
    y_traj[i] = y

    # Compute magnetic field at the current position (only z-component matters␣
↪for   = 0)
```

```
    Bz = 2 * B0 / (x**3 / RE**3)

    # Update velocities
    vx_new = vx + (q / m) * vy * (Bz) * dt
    vy_new = vy + (q / m) * vx * (-Bz) * dt

    # Update positions
    x_new = x + vx * dt
    y_new = y + vy * dt

    # Update for the next iteration
    vx, vy = vx_new, vy_new
    x, y = x_new, y_new


# Plot trajectory
plt.plot(x_traj / RE, y_traj / RE)
plt.xlabel('x ($R_E$)')
plt.ylabel('y ($R_E$)')
plt.title('Proton Trajectory over 100 seconds')
plt.grid(True)
plt.show()
```

```
100%|
                                | 100000000/100000000
[01:22<00:00, 1211514.63it/s]
```




### 0.2 RK4

For velocity $\vec{v} = (v_x, v_y, 0)$, the magnetic field $\vec{B}$ is along z-axis:

$$\vec{B} = \left(0, 0, \frac{2B_0}{(x/R_E)^3}\right)$$

The Lorentz force cross product $\vec{v} \times \vec{B}$ is:

$$\vec{v} \times \vec{B} = \begin{vmatrix} \hat{i} & \hat{j} & \hat{k} \\ v_x & v_y & 0 \\ 0 & 0 & \frac{2B_0}{(x/R_E)^3} \end{vmatrix} = \left(v_y \cdot \frac{2B_0}{(x/R_E)^3}, -v_x \cdot \frac{2B_0}{(x/R_E)^3}, 0\right)$$

Velocity derivatives (force per mass):

$$\frac{dv_x}{dt} = \frac{q}{m} \cdot v_y \cdot \frac{2B_0}{(x/R_E)^3}$$

$$\frac{dv_y}{dt} = -\frac{q}{m} \cdot v_x \cdot \frac{2B_0}{(x/R_E)^3}$$

For vx derivative:

$$f(v_x, v_y) = \frac{q}{m} v_y \cdot \frac{2B_0}{(x/R_E)^3}$$

For vy derivative:

$$g(v_x, v_y) = -\frac{q}{m} v_x \cdot \frac{2B_0}{(x/R_E)^3}$$

For vx:

$$k_1 = f(v_x, v_y) \cdot \Delta t$$
$$k_2 = f(v_x + \frac{k_1}{2}, v_y + \frac{k_1}{2}) \cdot \Delta t$$
$$k_3 = f(v_x + \frac{k_2}{2}, v_y + \frac{k_2}{2}) \cdot \Delta t$$
$$k_4 = f(v_x + k_3, v_y + k_3) \cdot \Delta t$$
$$v_x(t + \Delta t) = v_x(t) + \frac{k_1 + 2k_2 + 2k_3 + k_4}{6}$$

For vy:

$$k_1 = g(v_x, v_y) \cdot \Delta t$$
$$k_2 = g(v_x + \frac{k_1}{2}, v_y + \frac{k_1}{2}) \cdot \Delta t$$
$$k_3 = g(v_x + \frac{k_2}{2}, v_y + \frac{k_2}{2}) \cdot \Delta t$$
$$k_4 = g(v_x + k_3, v_y + k_3) \cdot \Delta t$$
$$v_y(t + \Delta t) = v_y(t) + \frac{k_1 + 2k_2 + 2k_3 + k_4}{6}$$

Position Updates:

$$\frac{dx}{dt} = v_x, \quad \frac{dy}{dt} = v_y$$

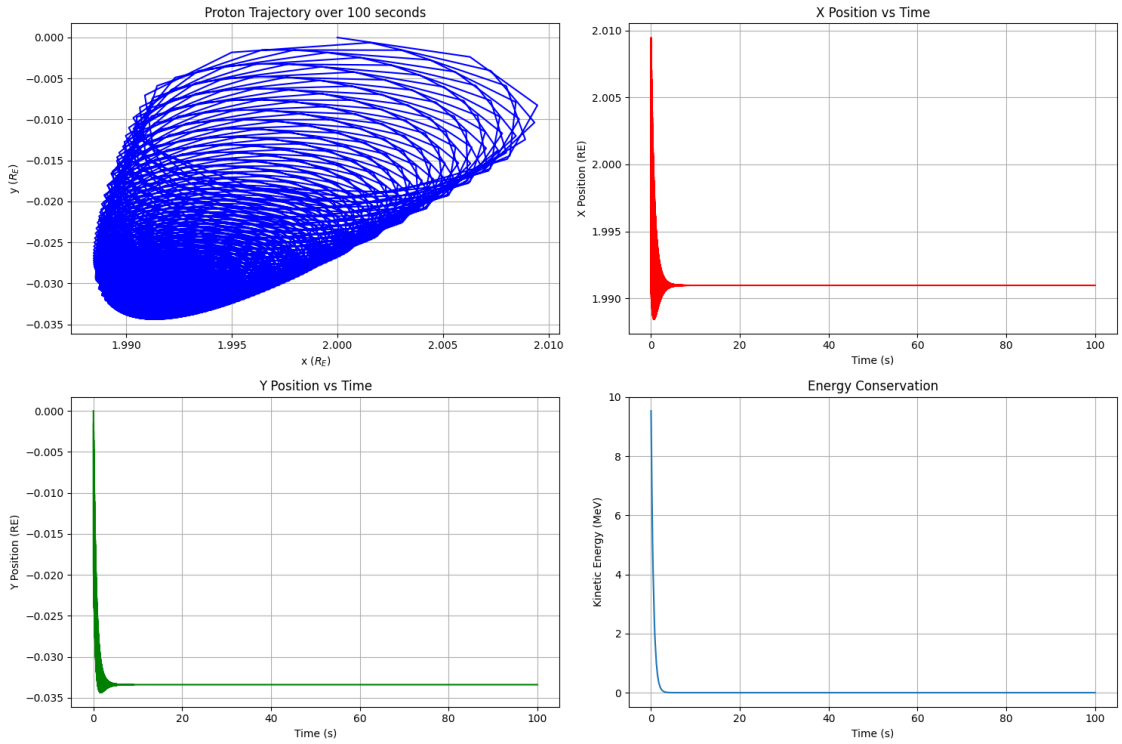
[25]:
```
# RK4
# Initial conditions
# Constants
q = 1.602e-19  # charge of proton (C)
m = 1.67e-27  # mass of proton (kg)
B0 = 3e-5  # magnetic field strength (T)
RE = 6.371e6  # Earth's radius (m)
x0 = 2 * RE  # initial position (m)
KE = 10 * 1.602e-13  # 10 MeV in joules
dt = 1e-3  # time step (s)
T = 100  # total time (s)
steps = int(T / dt)
v0x = np.sqrt(2 * KE / m)
```

```
x = x0
y = 0
vx = v0x
vy = 0

def B_field(x):
    """Calculate magnetic field strength at position x"""
    return 2 * B0 / (x/RE)**3

def f(vx, vy, x):
    """Derivative function for vx"""
    return (q/m) * vy * B_field(x)

def g(vx, vy, x):
    """Derivative function for vy"""
    return -(q/m) * vx * B_field(x)

def h(vx):
    """Derivative function for x"""
    return vx

def k(vy):
    """Derivative function for y"""
    return vy

def rk4_step(x, y, vx, vy, dt):
    """Perform one RK4 step and return new positions and velocities"""
    # RK4 for velocities
    k1_vx = f(vx, vy, x) * dt
    k1_vy = g(vx, vy, x) * dt

    k2_vx = f(vx + k1_vx/2, vy + k1_vy/2, x) * dt
    k2_vy = g(vx + k1_vx/2, vy + k1_vy/2, x) * dt

    k3_vx = f(vx + k2_vx/2, vy + k2_vy/2, x) * dt
    k3_vy = g(vx + k2_vx/2, vy + k2_vy/2, x) * dt

    k4_vx = f(vx + k3_vx, vy + k3_vy, x) * dt
    k4_vy = g(vx + k3_vx, vy + k3_vy, x) * dt

    # Update velocities
    vx_new = vx + (k1_vx + 2*k2_vx + 2*k3_vx + k4_vx) / 6
    vy_new = vy + (k1_vy + 2*k2_vy + 2*k3_vy + k4_vy) / 6

    # RK4 for positions
    k1_x = h(vx) * dt
    k1_y = k(vy) * dt
```

```
    k2_x = h(vx + k1_vx/2) * dt
    k2_y = k(vy + k1_vy/2) * dt

    k3_x = h(vx + k2_vx/2) * dt
    k3_y = k(vy + k2_vy/2) * dt

    k4_x = h(vx + k3_vx) * dt
    k4_y = k(vy + k3_vy) * dt

    # Update positions
    x_new = x + (k1_x + 2*k2_x + 2*k3_x + k4_x) / 6
    y_new = y + (k1_y + 2*k2_y + 2*k3_y + k4_y) / 6

    return x_new, y_new, vx_new, vy_new

# trajectory
x_traj = np.zeros(steps)
y_traj = np.zeros(steps)
vx_traj = np.zeros(steps)
vy_traj = np.zeros(steps)

# initial conditions
x_traj[0] = x0
y_traj[0] = 0
vx_traj[0] = v0x
vy_traj[0] = 0

# Iterate
for i in tqdm(range(1, steps)):
    x_traj[i], y_traj[i], vx_traj[i], vy_traj[i] = rk4_step(
        x_traj[i-1], y_traj[i-1], vx_traj[i-1], vy_traj[i-1], dt
    )

t = np.arange(steps) * dt
plt.figure(figsize=(15, 10))

# Trajectory plot
plt.subplot(221)
plt.plot(x_traj/RE, y_traj/RE, 'b-', linewidth=1.5)
plt.xlabel('x ($R_E$)')
plt.ylabel('y ($R_E$)')
plt.title('Proton Trajectory over 100 seconds')
plt.grid(True)

# X position vs time
plt.subplot(222)
```

```
plt.plot(t, x_traj/RE, 'r-', linewidth=1.5)
plt.xlabel('Time (s)')
plt.ylabel('X Position (RE)')
plt.title('X Position vs Time')
plt.grid(True)

# Y position vs time
plt.subplot(223)
plt.plot(t, y_traj/RE, 'g-', linewidth=1.5)
plt.xlabel('Time (s)')
plt.ylabel('Y Position (RE)')
plt.title('Y Position vs Time')
plt.grid(True)

# Energy conservation
plt.subplot(224)
KE = 0.5 * m * (x_traj[1:]/dt - x_traj[:-1]/dt)**2 + 0.5 * m * (y_traj[1:]/dt -␣
 ↪y_traj[:-1]/dt)**2
plt.plot(t[:-1], KE/1.602e-13)  # Convert to MeV
plt.xlabel('Time (s)')
plt.ylabel('Kinetic Energy (MeV)')
plt.title('Energy Conservation')
plt.grid(True)

plt.tight_layout()
plt.show()
```

```
100%|
                                    | 99999/99999
[00:00<00:00, 222729.05it/s]
```




[26]:
```
# RK4
# Initial conditions
# Constants
q = 1.602e-19  # charge of proton (C)
m = 1.67e-27  # mass of proton (kg)
B0 = 3e-5  # magnetic field strength (T)
RE = 6.371e6  # Earth's radius (m)
x0 = 2 * RE  # initial position (m)
KE = 10 * 1.602e-13  # 10 MeV in joules
dt = 1e-3  # time step (s)
T = 100  # total time (s)
steps = int(T / dt)
v0x = 1e7 * np.cos(10)
x = x0
y = 0
vx = v0x
vy = 1e7 * np.sin(10)

def B_field(x):
    """Calculate magnetic field strength at position x"""
    return 2 * B0 / (x/RE)**3

def f(vx, vy, x):
```

```
    """Derivative function for vx"""
    return (q/m) * vy * B_field(x)

def g(vx, vy, x):
    """Derivative function for vy"""
    return -(q/m) * vx * B_field(x)

def h(vx):
    """Derivative function for x"""
    return vx

def k(vy):
    """Derivative function for y"""
    return vy

def rk4_step(x, y, vx, vy, dt):
    """Perform one RK4 step and return new positions and velocities"""
    # RK4 for velocities
    k1_vx = f(vx, vy, x) * dt
    k1_vy = g(vx, vy, x) * dt

    k2_vx = f(vx + k1_vx/2, vy + k1_vy/2, x) * dt
    k2_vy = g(vx + k1_vx/2, vy + k1_vy/2, x) * dt

    k3_vx = f(vx + k2_vx/2, vy + k2_vy/2, x) * dt
    k3_vy = g(vx + k2_vx/2, vy + k2_vy/2, x) * dt

    k4_vx = f(vx + k3_vx, vy + k3_vy, x) * dt
    k4_vy = g(vx + k3_vx, vy + k3_vy, x) * dt

    # Update velocities
    vx_new = vx + (k1_vx + 2*k2_vx + 2*k3_vx + k4_vx) / 6
    vy_new = vy + (k1_vy + 2*k2_vy + 2*k3_vy + k4_vy) / 6

    # RK4 for positions
    k1_x = h(vx) * dt
    k1_y = k(vy) * dt

    k2_x = h(vx + k1_vx/2) * dt
    k2_y = k(vy + k1_vy/2) * dt

    k3_x = h(vx + k2_vx/2) * dt
    k3_y = k(vy + k2_vy/2) * dt

    k4_x = h(vx + k3_vx) * dt
    k4_y = k(vy + k3_vy) * dt
```

```
    # Update positions
    x_new = x + (k1_x + 2*k2_x + 2*k3_x + k4_x) / 6
    y_new = y + (k1_y + 2*k2_y + 2*k3_y + k4_y) / 6

    return x_new, y_new, vx_new, vy_new

# trajectory
x_traj = np.zeros(steps)
y_traj = np.zeros(steps)
vx_traj = np.zeros(steps)
vy_traj = np.zeros(steps)

# initial conditions
x_traj[0] = x0
y_traj[0] = 0
vx_traj[0] = v0x
vy_traj[0] = 0

# Iterate
for i in tqdm(range(1, steps)):
    x_traj[i], y_traj[i], vx_traj[i], vy_traj[i] = rk4_step(
        x_traj[i-1], y_traj[i-1], vx_traj[i-1], vy_traj[i-1], dt
    )

t = np.arange(steps) * dt
plt.figure(figsize=(15, 10))

# Trajectory plot
plt.subplot(221)
plt.plot(x_traj/RE, y_traj/RE, 'b-', linewidth=1.5)
plt.xlabel('x ($R_E$)')
plt.ylabel('y ($R_E$)')
plt.title('Proton Trajectory over 100 seconds')
plt.grid(True)

# X position vs time
plt.subplot(222)
plt.plot(t, x_traj/RE, 'r-', linewidth=1.5)
plt.xlabel('Time (s)')
plt.ylabel('X Position (RE)')
plt.title('X Position vs Time')
plt.grid(True)

# Y position vs time
plt.subplot(223)
plt.plot(t, y_traj/RE, 'g-', linewidth=1.5)
plt.xlabel('Time (s)')
```

```
plt.ylabel('Y Position (RE)')
plt.title('Y Position vs Time')
plt.grid(True)

# Energy conservation
plt.subplot(224)
KE = 0.5 * m * (x_traj[1:]/dt - x_traj[:-1]/dt)**2 + 0.5 * m * (y_traj[1:]/dt -␣
 ↪y_traj[:-1]/dt)**2
plt.plot(t[:-1], KE/1.602e-13)  # Convert to MeV
plt.xlabel('Time (s)')
plt.ylabel('Kinetic Energy (MeV)')
plt.title('Energy Conservation')
plt.grid(True)

plt.tight_layout()
plt.show()
```

```
100%|                                                                                | 99999/99999
[00:00<00:00, 218332.43it/s]
```