CA , End -sem , 2020 , Part -II

Shivam Kumaran , SC17B122 28/12/2020

1 Question 01

1.1 Code

```
import numpy as np
from matplotlib import pyplot as plt


def acc(theta, beta):
    num = 1-beta*np.cos(theta)
    val = 1/(num**3)
    return val

theta = np.linspace(-np.pi,np.pi, 100)
beta = np.linspace( 0.0 , 0.9, 100)
k = []
for th in theta:
    temp = []
    for bt in beta:
        val = acc(th , bt)
        temp.append(val)
    k.append(temp)

b , t = np.meshgrid(beta, theta)


fig , ax = plt.subplots()
color_plot = ax.contourf((b), np.degrees(t), k, 100 , cmap=plt.cm.magma)
#line_plot = ax.contour(color_plot, levels=color_plot.levels[::8], colors='cyan' , linewidths=[0.3
cbar = fig.colorbar(color_plot)
cbar.ax.set_ylabel(r'$\alpha\times|\vec E_{acc}|$')
ax.set_xlabel(r'$\beta$')
ax.set_ylabel(r'$\theta (deg)$')

plt.savefig('acc_m_p9_v2.png')
plt.show()
#print(k)
```

1.2 Plot

Notes :

- θ is taken from $[-\pi, \pi]$
- Range of values for β is $[0.0, 0.5]$ and $[0.0, 0.9]$ respectively for two plots.
- plotted for different ranges of β for better inference
- in the plots α is constant of proportionality.
- For plots upto 0.9C , log plot is taken for clarity.

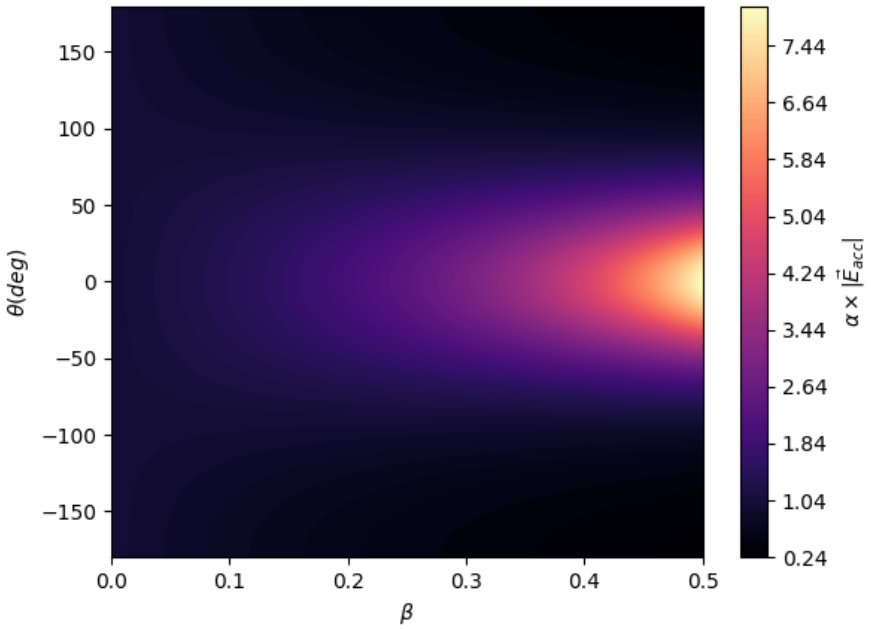

Figure 1: Filled contour plot for Acceleration component of electric field , for velocity upto 0.5C

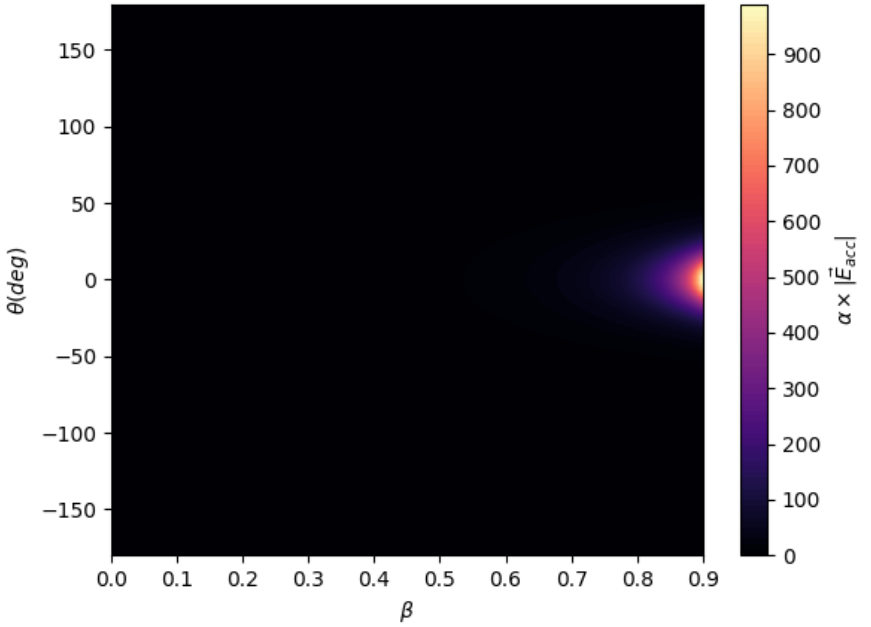

Figure 2: Filled contour plot for Acceleration component of electric field , for velocity upto 0.9C

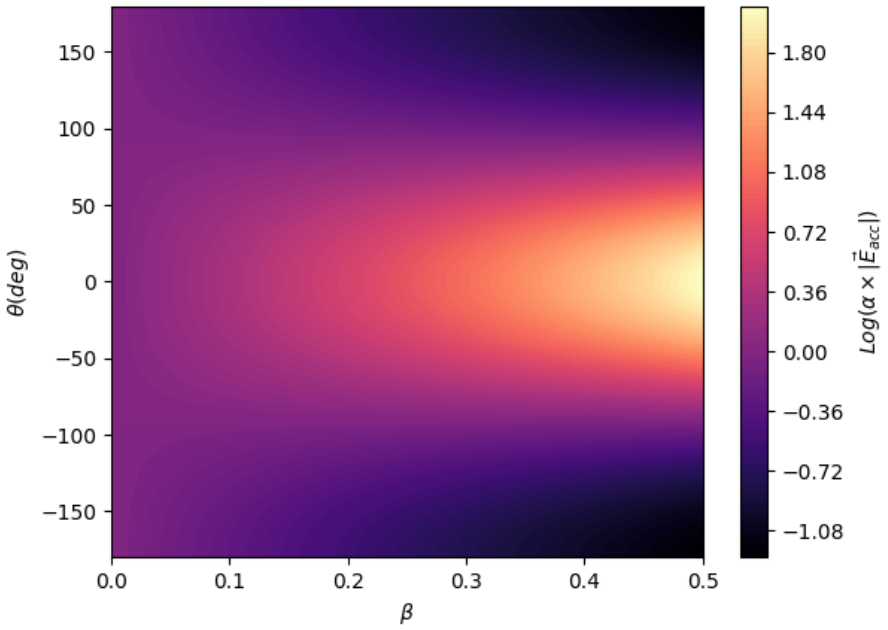

Figure 3: Filled contour plot for Acceleration component of electric field , for velocity upto 0.5C , plotted on log scale for clarity

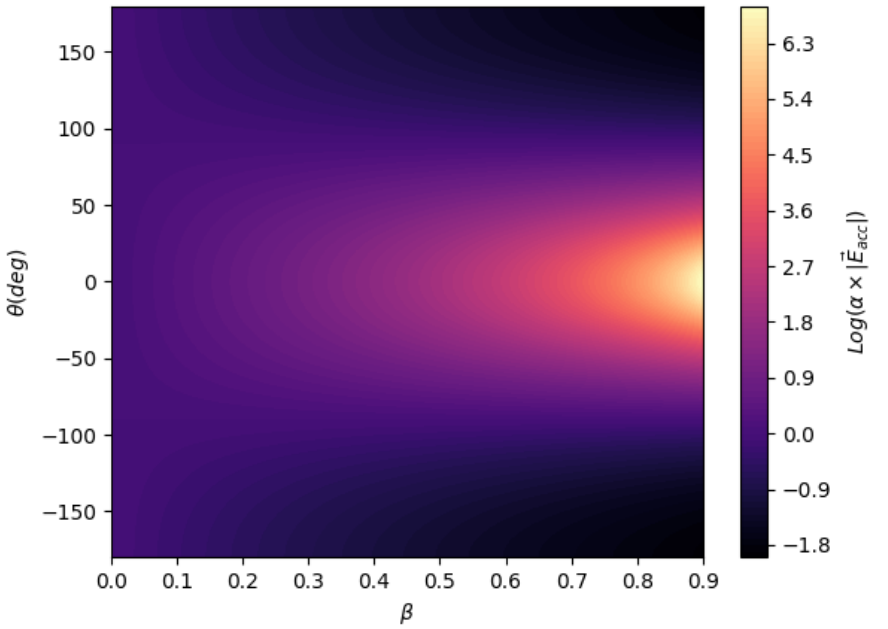

Figure 4: Filled contour plot for Acceleration component of electric field , for velocity upto 0.9C , plotted on log scale for clarity

1.3 Inference

1. For nonrelativistic particles , ($\beta \approx 0$) , magnitude of electic field is almost evenly distributed.

2. As velocity increases , magnitude of electric field increases and electric fields gets concentrated towards $theta \approx 0$
3. for ($\beta \approx 0.9$) electric field is almost confined between $\theta \approx \pm 30^o$
4. One interesting observation, when we plot contour lines also , (below plot) , that for $\theta = \pm 90^o$ contour plot become parallel.

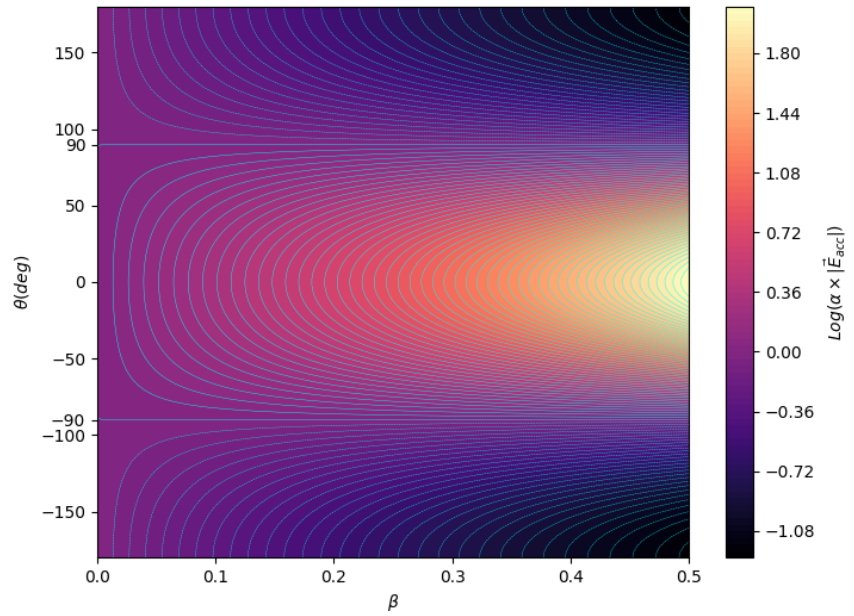

Figure 5: Filled contour plot with line contour for Acceleration component of electric field , for velocity upto 0.5C , plotted on log scale for clarity

2 Question 02

2.1 Formulation

converting given problem into matrix equation .

$$\epsilon_n = \epsilon_\infty + (c_1/n) + (c_2/n^2) + (c_3/n^3) + (c_4/n^4)$$

$$\begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{2}^2 & \frac{1}{2}^3 & \frac{1}{2}^4 \\ 1 & \frac{1}{4} & \frac{1}{4}^2 & \frac{1}{4}^3 & \frac{1}{4}^4 \\ 1 & \frac{1}{6} & \frac{1}{6}^2 & \frac{1}{6}^3 & \frac{1}{6}^4 \\ 1 & \frac{1}{8} & \frac{1}{8}^2 & \frac{1}{8}^3 & \frac{1}{8}^4 \\ 1 & \frac{1}{10} & \frac{1}{10}^2 & \frac{1}{10}^3 & \frac{1}{10}^4 \end{bmatrix} \times \begin{bmatrix} \epsilon_\infty \\ c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} = \begin{bmatrix} -2.0 \\ -1.5 \\ -1.4343 \\ -1.4128 \\ -1.4031 \end{bmatrix}$$

2.2 Code

2.2.1 Solving matrix equation

Using Gauss Elimination method

```
def update_row(r2,r1):
    coeff = r2[0]/r1[0]
    r_temp = r2 - coeff*r1
    return(r_temp)

def update_sub_mat(mat):
    m_temp = []
    m_temp.append(mat[0])
    for i in range(1,np.shape(mat)[0]):
        m_temp.append(update_row(mat[i],mat[0]))
    return (np.asarray(m_temp) )

def calc_ecl(m):
    temp_mat = np.copy(m)
    for i in range(len(temp_mat)):
        temp_mat[i:,i:] = update_sub_mat(temp_mat[i:,i:])
    return(temp_mat)

def calc_row_sol(mat_vec,sol_vec):
    sol = (mat_vec[-1] - np.dot(mat_vec[1:-1],sol_vec))/mat_vec[0]
    return(sol)

def solve_mat_eqn(m,b_vec):

    if(np.shape(m)[0]!=len(b_vec)):
        raise ValueError('Matrix and vector dim mismatch')
    else:
        mat_given = np.column_stack((m,b_vec))
        mat = calc_ecl(mat_given)
        sol_vec = []
        for i in reversed(range(np.shape(mat)[0])):
            sol_vec.insert(0,calc_row_sol(mat[i,i:],sol_vec))
        return(sol_vec)
```

2.2.2 Solving for ϵ_∞

```
import numpy as np
from linear_solver_v2 import solve_mat_eqn
```

```
e2 = -2.0
e4 = -1.5
e6 = -1.4343
e8 = -1.4128
e10 = -1.4031

const_vec = [e2,e4,e6,e8,e10]

coeff = []
for j in range(1,6):
    c = [(1/(2*j)**i) for i in range(5)]
    coeff.append(c)

coeff = np.asarray(coeff)

solution = solve_mat_eqn(coeff,const_vec)
eps_inf = solution[0]
print('Epsilon infinity : {:.4f}'.format(eps_inf))
```

Epsilon infinity : -1.3875