PHYS2600: Homework 4*

Due on Friday, Mar, 9th 2018

Yangrui Hu

*This Latex template is from http://www.latextemplates.com

Contents

Problem 1

Problem: Consider the one-dimensional, time-independent Schrodinger equation for a harmonic oscillator with potential $V(x) = V_0x^2/a^2$ where V_0 and a are constants.
(a) Convert the Schrodinger equation from a second order equation to two first-order ones, as we did class. Write a program using the shooting method with a root finding algorithm of your choice to find the energies of the ground state and the first two excited states with m the electron mass ($9.1 \times 10^{-31}kg$), $V_0 = 50eV$, and $a = 10^{-11}m$. Note that strictly speaking the wavefunction is nonzero even at infinity but it is a reasonable approximation to use a large but finite interval, e.g., $x \leq 10a$, with the wavefunction $\psi = 0$ at both boundaries. The quantum harmonic oscillator is known to have energy states that are equally spaced. Check that this is true numerically. (Note: The ground state has energy in the range 100 to $200eV$. To keep your units consistent write $\frac{d\psi}{dx} = e(2m/\hbar^2)(V(x) - E)\psi$ with m, $\hbar$ and e in SI units, V and E are in electron volts.)
(b) Now modify your program to calculate the same three energies for the anharmonic oscillator with $V(x) = V_0x^4/a^4$, with the same parameter values as in part a.
(c) Modify your program further to calculate the wavefunctions of the anharmonic oscillator for the three states and make a plot of them, all on the same axes, as a function of x over a modest range near the originsay $x = 5a$ to $x = 5a$. You do not need to normalize the wavefunctions. Scale them so that all three curves fit on the same plot.

Solution: (a)As we did in class, the Schrodinger equation can be converted to two first-order ones as:

$$v = \frac{du}{dx} \tag{1}$$

$$\frac{dv}{dx} = e\frac{2m}{\hbar^2}(V(x) - E)u \tag{2}$$

where u is the wavefunction which we are looking for, $V(x)$ is the potential function, and E is the eigen-energy. Listing 1 is my program to solve this problem. Figure 1 shows the output of eigen-energies and I find the first three energy levels:

- Ground State: $E_0 = 138.210388\ eV$
- The First Excited State: $E_1 = 414.31167\ eV$
- The Second Excited State: $E_2 = 690.51943\ eV$

The quantum harmonic oscillator is known to have energy states that are equally spaced. According to my calculation results, $E_2 - E_1 = 276.20776\ eV$ and $E_1 - E_0 = 276.101282\ eV$. So the error is 0.03855%, and we can state that the energy levels of the quantum harmonic oscillator are equally spaced. Figure 2 shows the wavefunctions of these three states.

```
crystal@crystaldembp [11:57:00] [~/Google Drive/PHYS 2600/Homework/hw4]
-> % ipython problem1.py
Energy found: 690.51943
Energy found: 414.31167
Energy found: 138.10388
```

Figure 1: The output of energies for part(a)

(b) When the potential function is $V(x) = V_0x^4/a^4$, Figure 3 shows the output and the first three energy levels are:

- Ground State: $E_0 = 205.46543\ eV$

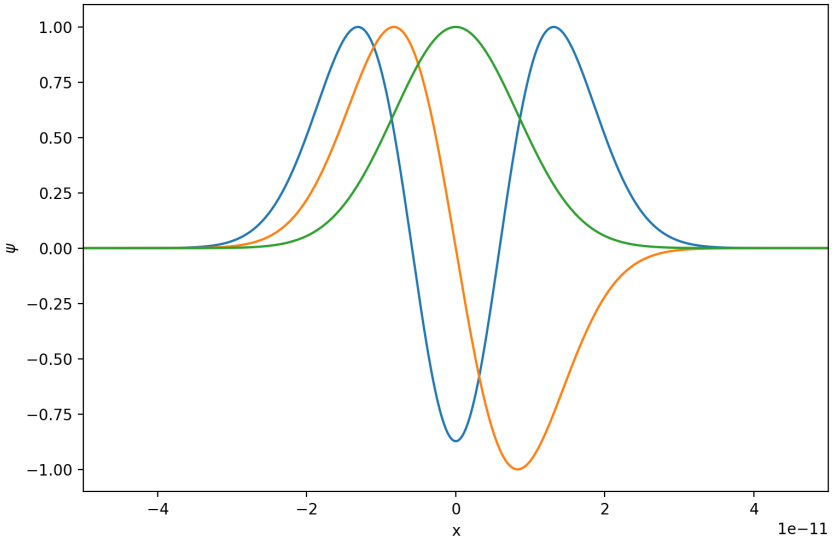

Figure 2: Wavefunctions of the first three states for part(a)

- The First Excited State: $E_1 = 736.25931\ eV$
- The Second Excited State: $E_2 = 1444.68405\ eV$

```
crystal@crystaldembp [12:00:52] [~/Google Drive/PHYS 2600/Homework/hw4]
-> % ipython problem1.py
Energy found: 1444.68405
Energy found: 736.25931
Energy found: 205.46543
```

Figure 3: The output of energies for part(b)

(c) Figure 4 shows the wavefunctions of the anharmonic oscillator.

Problem 1 continued on next page. . .

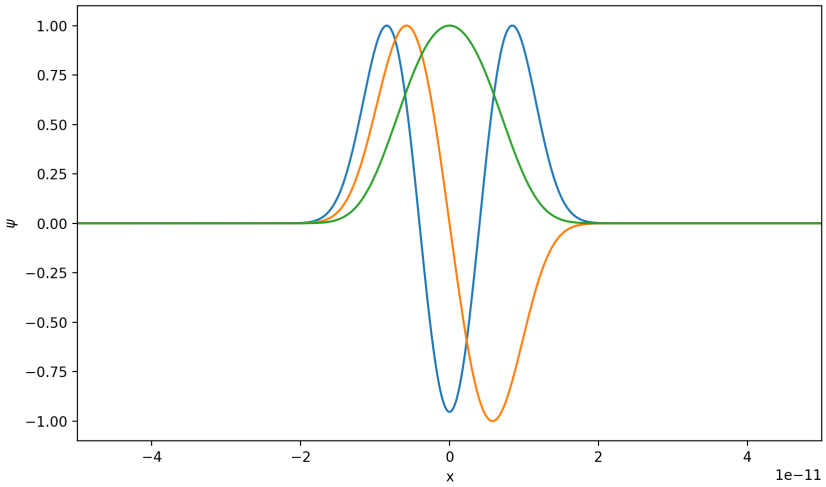

Figure 4: Wavefunctions of the first three states for part(c)

Listing 1: Original Script of Problem 1

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.linalg import eigh
from scipy.integrate import odeint

class QM(object):

    def __init__(self, E, npoints, x_start, x_end = 0, u0 = 0.0, v0 = 0.01):
        self.E = E
        self.npoints = npoints
        self.uv0 = np.array([u0, v0])
        self.xarray = np.linspace(x_start, x_end, npoints, endpoint = True)
        self.V0 = 50.0

        self.a = 10.0**11
        #self.m = 9.1*np.power(10,-31)
        #self.hbar = 1.0546*np.power(10, -34)
        #self.e = 1.602*np.power(10, -19)
        self.c = 2.62155*10.0**19#2.*9.1*1.602*10.**18/(1.0546**2)

    def scipy_trajectory(self):
        self.uv = odeint(self.sch, self.uv0, self.xarray)
        self.uv_end = self.uv[-1]

    def sch(self, uv, x):
        v = uv[1]
        acce = self.c*(self.potential(x)-self.E)*uv[0]
        return np.ravel(np.array([v, acce]))

    def potential(self,x):
        #return self.V0*x*x*self.a*self.a
```

```
            return self.V0*(x**4)*(self.a**4)

def match(En):
    p_up = QM(E = En, npoints = 500, x_start = -10.0/(10**11))
    p_down = QM(E = En, npoints = 500, x_start = 10.0/(10**11))
    p_up.scipy_trajectory()
    p_down.scipy_trajectory()

    return p_down.uv_end[0]*p_up.uv_end[1] - p_down.uv_end[1]*p_up.uv_end[0]

def bisect(f, a, b, eps = 1.e-6):
    fa, fb, gap  = f(a), f(b), abs(b-a)

    if (fa*fb > 0.0):
        print('Bisection error: no root bracketed')
        return None
    elif fa == 0.0: return a
    elif fb == 0.0: return b

    while(True):
        xmid = 0.5*(a+b)
        fmid = f(xmid)

        if (fa*fmid > 0.0):
            a, fa = xmid, fmid
        else :b = xmid

        if (fmid == 0.0 or abs (b-a) < eps*gap): break

    return xmid

E1 = 2000.0
dE = 1.0
npoints = 1000

fig = plt.figure()
ax = fig.add_subplot(111)

while (E1 > 0+dE):
    if match(E1)*match(E1-dE) < 0 :
        E = bisect(match, E1, E1-dE, 1e-8)
        print('Energy found: %.5f'%(E))


        p_up = QM(E, npoints = npoints, x_start = -10.0/(10**11), x_end = 0.)
        p_down = QM(E, npoints = npoints, x_start = 10.0/(10**11), x_end = 0.)
        p_up.scipy_trajectory()
        p_down.scipy_trajectory()

        # scale factor
        #print(p_up.uv[-1][0])
        if (abs(p_up.uv[-1][0]) <= 1e30):
          scale = abs(p_up.uv[-2][0])/abs(p_down.uv[-2][0])
```

Problem 1 continued on next page...

```
        else:
          #print("here")
          scale = p_up.uv[-2][0]/p_down.uv[-2][0]

        # full solution
        psi_x = np.concatenate((p_up.uv[:-1,0], scale*p_down.uv[::-1,0]))
        xa = np.linspace(-5.0/(10**11),5.0/(10**11), 2*npoints-1, endpoint = True)

        # plot the scaled solution
        ax.plot(xa, psi_x/max(psi_x))

    E1 = E1-dE

ax.set_xlim([-5.0/(10**11), 5.0/(10**11)])
ax.set_xlabel('x')
ax.set_ylabel('$\psi$')
plt.show()
```

Problem 2

Problem: (a) Use the Jacobi method to compute the electric potential between two infinitely long concentric square cylinders. Assume the cylinders have sides of 5.0 and 25.0. Assume that the inner conductor is held at $V = 1$, and the outer at $V = 0$. You can use a grid spacing of one in your Jacobi method. Make a contour plot of the potential and some one dimensional slices across rows of the grid.
(b) Compute the electric potential and field for a point charge located at the center of a parallel plate capacitor in two dimensions. Make a contour plot of the potential and field. Study how the equipotential contours are affected by the proximity of the charge to one of the capacitor places.

Solution: (a) Figure 5 shows the contour plot of the electric potential and field. Figure 6 shows some one dimensional slices of the potential. When the slice across $y = 0$ and $y = -2$, the slice will pass through two square cylinders and we can see that there are two horizontal segments corresponding to $V = 0$ (outer cylinder B.C.) and $V = 1$ (inner cylinder B.C.).

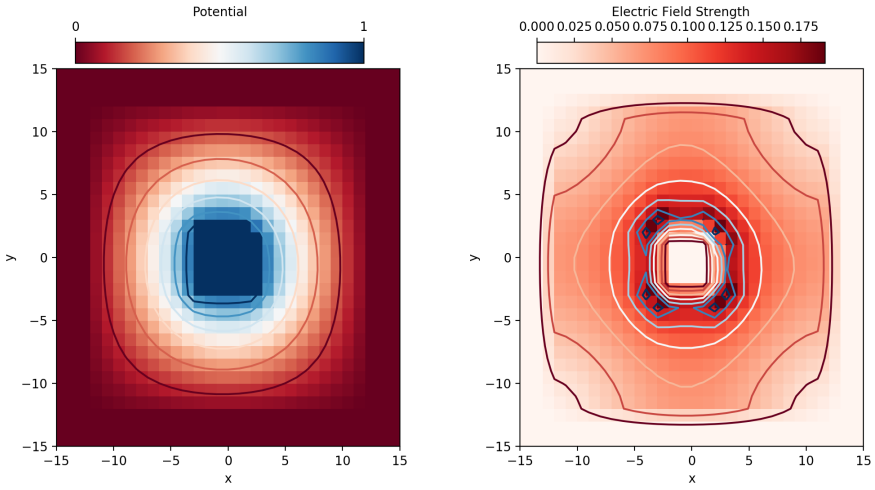

Figure 5: The plot of the electric potential and field for part (a)

(b) Figure 10 shows the contour plot of the potential and field when the point charge is located at the center of a parallel plate capacitor. From Figure 7 to Figure 13, the position of the point charge is moved to right. I assume the charge is positive, so the positive charge is located near the negative charged plate at the beginning as Figure 7 shown. When the charge moving to the positive plate, the contour plots are gradually like the parallel plate capacitor case, as Figure 14 shown.

Problem 2 continued on next page...

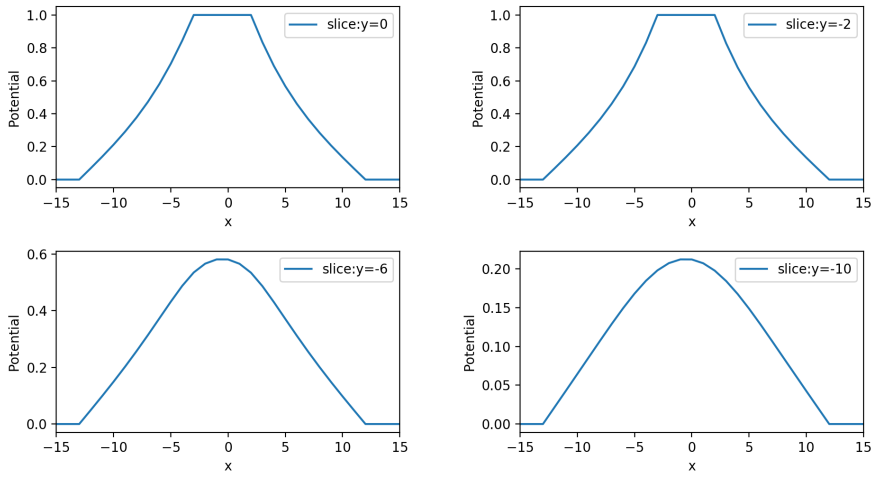

Figure 6: One dimensional slices of the electric potential for part (a)

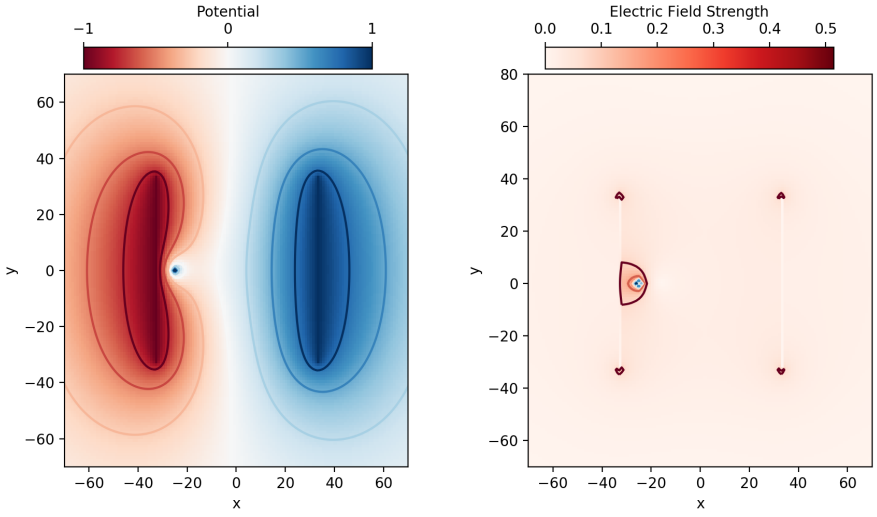

Figure 7: The plot of the electric potential and field for part (b): the position of the point charge is at (-25,0)

Problem 2 continued on next page. . .

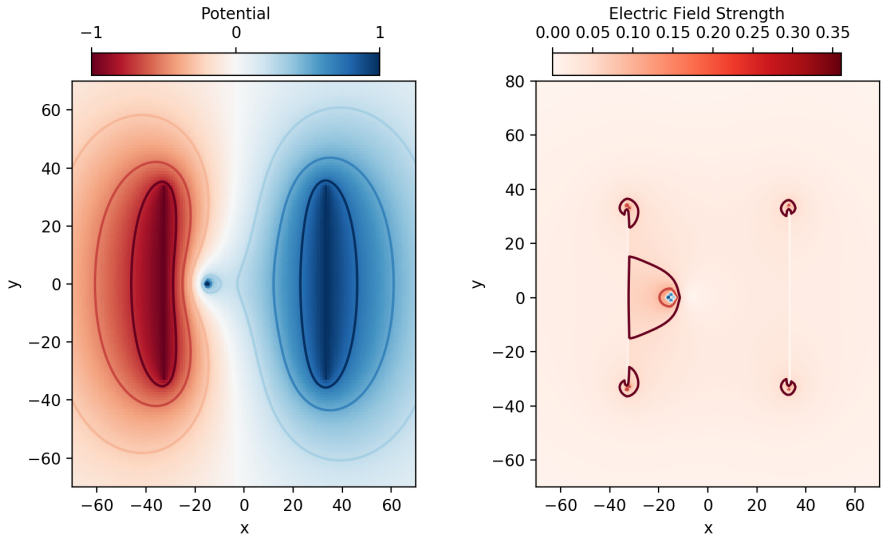

Figure 8: The plot of the electric potential and field for part (b): the position of the point charge is at (-15,0)

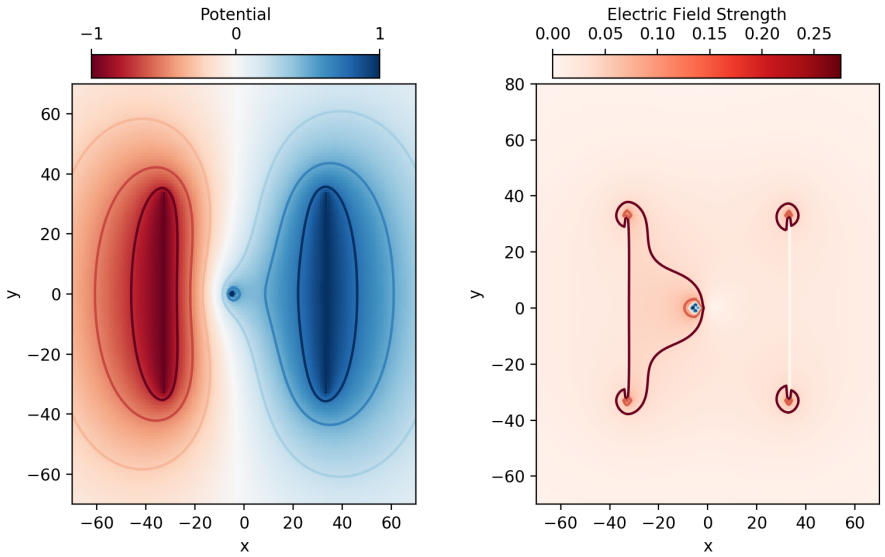

Figure 9: The plot of the electric potential and field for part (b): the position of the point charge is at (-5,0)

Problem 2 continued on next page. . .

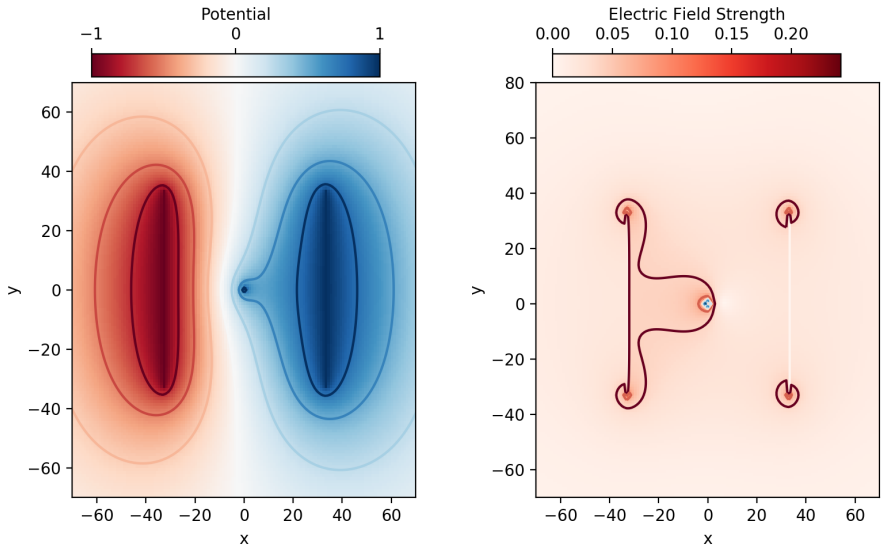

Figure 10: The plot of the electric potential and field for part (b): the position of the point charge is at (0,0)

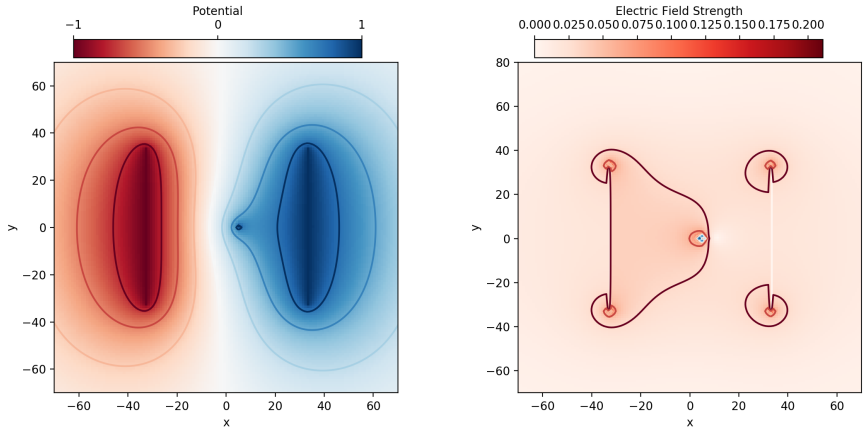

Figure 11: The plot of the electric potential and field for part (b): the position of the point charge is at (5,0)

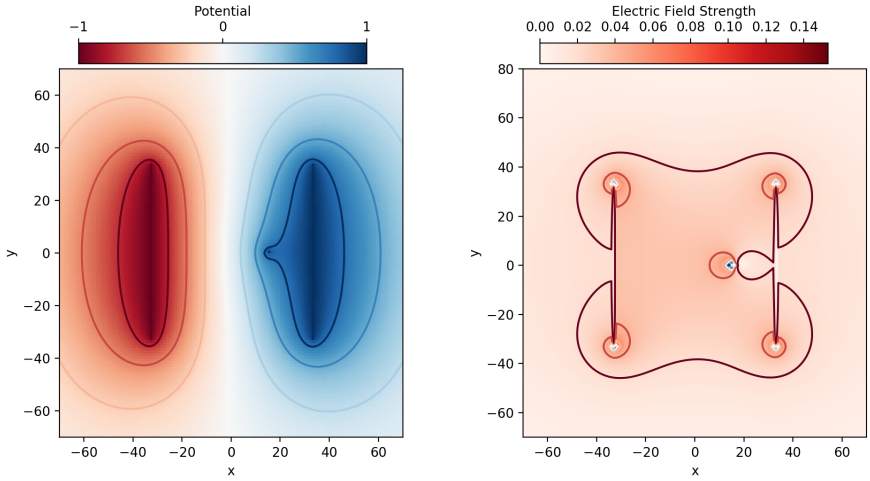

Figure 12: The plot of the electric potential and field for part (b): the position of the point charge is at (15,0)

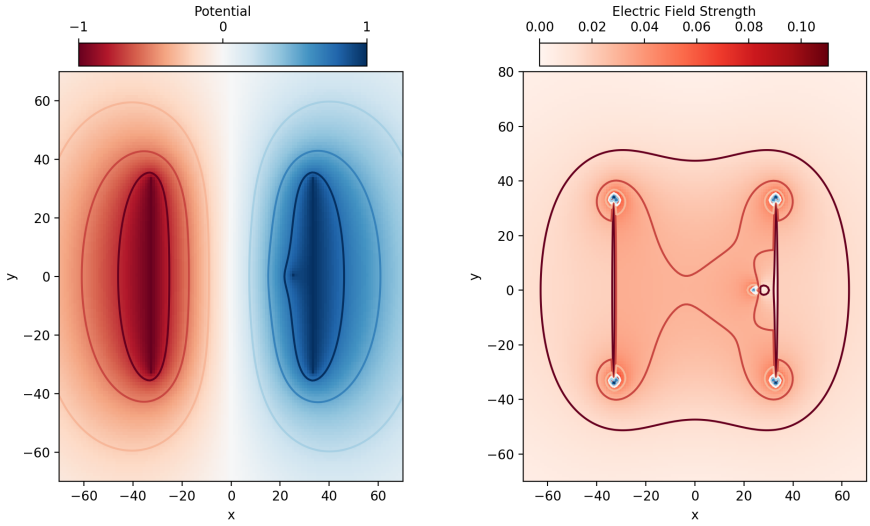

Figure 13: The plot of the electric potential and field for part (b): the position of the point charge is at (25,0)

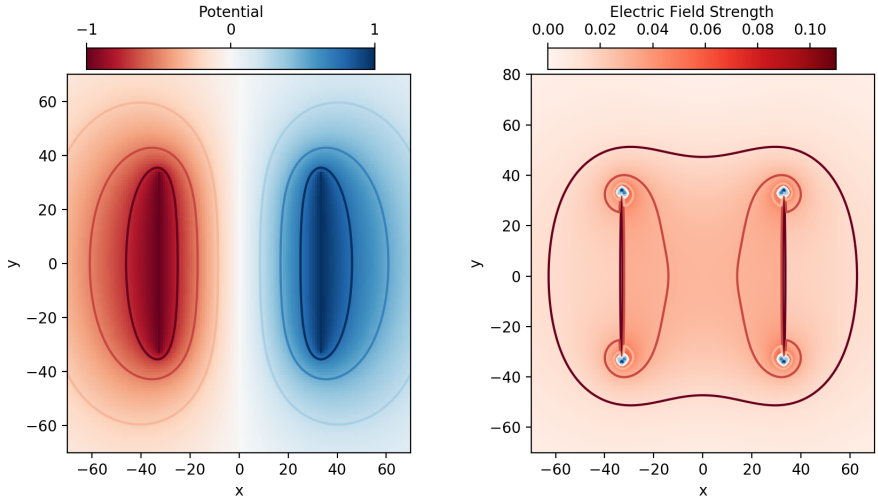

Figure 14: The plot of the electric potential and field for a parallel plate capacitor

Listing 2: Original Script of Problem 2

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

def Jacobi2D(b, boundary,  dx=1.0, converge = 1e-6):

    V = boundary(np.zeros(b.shape, float))
    Vnew = np.zeros(b.shape, float)
    Npoints = len(b.flatten())

    t = 0
    dV = converge*10

    while(dV > converge):
        Vnew[1:-1, 1:-1] = (V[1:-1,2:] + V[1:-1,:-2] + V[2:,1:-1] + V[:-2,1:-1])/4
        Vnew = boundary(Vnew)
        if t > 100 :
            dV = np.sum(abs(Vnew - V))/Npoints
        V[:,:] = Vnew
        t+=1

    return Vnew

def parallel_plate(phi):
    n, m = np.shape(phi)
    midx = n//2
    midy = m//2
    h = n//3
    right = midx + h//2
    left = midx - h//2
    phi[h:2*h, left] = - 1
    phi[h:2*h, right] =  1
    # point charge potetial
```

```
    phi[midx,midy+25] = 1

    return phi

def concentric_square_cylinder(phi):
    n, m = np.shape(phi)
    midx = n//2
    midy = m//2
    top25 = midx -13
    bot25 = midx +12
    top5 = midx - 3
    bot5 = midx + 2
    left25 = midy -13
    right25 = midy +12
    left5 = midy - 3
    right5 = midy + 2
    phi[top25:bot25,left25] = 0
    phi[top25:bot25,right25] = 0
    phi[top25,left25:right25] = 0
    phi[bot25,left25:right25] = 0
    phi[top5:bot5,left5] = 1
    phi[top5:bot5,right5] = 1
    phi[top5,left5:right5] = 1
    phi[bot5,left5:right5] = 1
    return phi

def plot(phi, X, Y, EY, EX, magE):

    fig = plt.figure()
    ax1 = fig.add_subplot(121)
    ax2 = fig.add_subplot(122)

    #plot potential
    m1 = ax1.pcolormesh(X, Y, phi, cmap = 'RdBu')
    cax1 = fig.add_axes([0.14,0.87,0.3,0.04])
    cbar = fig.colorbar(m1,cax1, orientation = 'horizontal', ticklocation = 'top',
     ticks = [-1,0,1])
    cbar.set_label('Potential')
    ax1.contour(X,Y,phi, cmap = 'RdBu')
    ax1.set_xlabel("x")
    ax1.set_ylabel("y")
    ax1.set_xlim([-70,70])
    ax1.set_ylim([-70,70])

    #plot E-field
    m2 = ax2.pcolormesh(X,Y,magE, cmap = 'Reds')
    cax2 = fig.add_axes([0.62,0.87,0.3,0.04])
    cbar2 = fig.colorbar(m2,cax2,orientation = 'horizontal', ticklocation = 'top' )
    cbar2.set_label('Electric Field Strength')
    ax2.contour(X,Y,magE, cmap = 'RdBu')
    ax2.set_xlabel("x")
    ax2.set_ylabel("y")
    ax2.set_xlim([-70,70])
```

Problem 2 continued on next page...

```
    ax2.set_ylim([-70,80])

    fig.subplots_adjust(left = 0.12,right = 0.96, bottom = 0.14, top = 0.86,
        wspace =0.35, hspace = 0.35)
    plt.show()

def plot_slices(phi):
    n, m = np.shape(phi)
    midy = n//2
    sli_1 = phi[midy,:]
    sli_2 = phi[midy-2,:]
    sli_3 = phi[midy-6,:]
    sli_4 = phi[midy-10,:]
    xarray = np.linspace(-100,100,201,endpoint = True)

    fig = plt.figure()
    ax1 = fig.add_subplot(221)
    ax2 = fig.add_subplot(222)
    ax3 = fig.add_subplot(223)
    ax4 = fig.add_subplot(224)
    ax1.plot(xarray,sli_1, label='slice:y=0')
    ax2.plot(xarray,sli_2, label='slice:y=-2')
    ax3.plot(xarray,sli_3, label='slice:y=-6')
    ax4.plot(xarray,sli_4, label='slice:y=-10')

    ax1.set_xlabel("x")
    ax1.set_ylabel("Potential")
    ax1.set_xlim([-15,15])
    ax2.set_xlabel("x")
    ax2.set_ylabel("Potential")
    ax2.set_xlim([-15,15])
    ax3.set_xlabel("x")
    ax3.set_ylabel("Potential")
    ax3.set_xlim([-15,15])
    ax4.set_xlabel("x")
    ax4.set_ylabel("Potential")
    ax4.set_xlim([-15,15])
    ax1.legend()
    ax2.legend()
    ax3.legend()
    ax4.legend()
    fig.subplots_adjust(left = 0.12,right = 0.96, bottom = 0.14, top = 0.86,
        wspace =0.35, hspace = 0.35)
    plt.show()

def main(boundary):

    N = 200
    dx = 1/N
    X,Y = np.meshgrid(range(-N//2, N//2+1), range(-N//2, N//2+1))
    grid = np.zeros([N+1, N+1])

    phi = Jacobi2D(grid, boundary, dx)
```

```
    EY, EX = -1*np.array(np.gradient(phi))
    magE = np.sqrt(EX**2 + EY**2)

    plot(phi,X,Y,EY,EX,magE)
    #plot_slices(phi)

main(parallel_plate)
```