```python
import numpy as np
from scipy.optimize    import curve_fit
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import pyplot as plt
from matplotlib.backends.backend_pdf import PdfPages
import matplotlib.cm as cm
import matplotlib.mlab as mlab
from  math import pi


size = 80

def init_operator( harmonic = False ):
    ret = np.zeros( shape = ( size*size , size*size ))

    for raw in range(  size*size  ):

        if raw % size != 0 or harmonic:
            ret[raw][raw-1] = 1
        if raw % size != size - 1 or harmonic:
            ret[raw][(raw+1) % (size*size)] = 1
        if raw >= size :
            ret[raw][raw - size] = 1
        if raw < size * (size - 1):
            ret[raw][raw + size] = 1
        if raw < size:
            ret[raw][raw] = 1.
        if raw > size * (size - 1):
            ret[raw][raw] = 1.

    return ret

def init_E_fieldX_oprator( harmonic = False ):
    ret = np.zeros( shape = ( size*size , size*size ))

    for raw in range(  size*size  ):
        if not harmonic :
            if raw % size != 0 and raw % size != size - 1:
                ret[raw][raw] = -1
                ret[raw][raw+1] = 1
        else:
            ret[raw][raw] = -1
            ret[raw][(raw+1) % size*size] = 1


    return ret

def init_E_fieldY_oprator():
    ret = np.zeros( shape = ( size*size , size*size ))

    for raw in range(  size*size  ):
        if raw < size * (size - 1):
            ret[raw][raw] = -1
            ret[raw][raw + size] = 1

    return ret

laplace_matrix = []
EX_operator = []  #init_E_fieldX_oprator()
EY_operator = []  #init_E_fieldY_oprator()


def calc_iteration(laplace_vec, laplace_matrix):
    ret = 0.25 * laplace_matrix @ laplace_vec
    diff = abs( ret - laplace_vec )
    maxdiff = np.max( diff )
    return ret , maxdiff

def calc( laplace_vec, eps, harmonic=False):
    laplace_matrix = init_operator(harmonic)

    laplace_vec, maxdiff = calc_iteration(laplace_vec, laplace_matrix)
```

```python
    fn = [ maxdiff ]
    while maxdiff > eps :
        laplace_vec, maxdiff = calc_iteration(laplace_vec, laplace_matrix)
        fn.append( maxdiff )
    return laplace_vec , fn

def reso_matrix (matrix, nx, ny):

    reso = np.zeros( shape = (nx, ny) )

    reso[0,0] = abs(matrix[0,0])

    for j in range(1 , ny ):
        reso[0 , j] = reso[0 , j-1] + abs(matrix[0, j])

    for i in range(1 , nx ):
        reso[i , 0] = reso[i-1 , 0] + abs(matrix[i, 0])


    for i in range(1 , nx):
        for j in range(1 , ny):
            reso[i , j]  =  abs(matrix[i, j]) + reso[i, j - 1] + \
            reso[i-1, j] - reso[i-1, j-1]

    reso = reso / ( nx * ny )

    plotreso = np.zeros( shape = (nx, ny) )

    eps = 10

    for i in range(1 , nx):
        for j in range(1 , ny):
            if abs(reso[i , j]  - reso[i // 2 , j // 2]) < eps:
                plotreso[i,j] = reso[i,j]

    return reso

def electricfield(laplace_vec, harmonic = False) :

    global laplace_matrix
    laplace_matrix = None

    EX_operator = init_E_fieldX_oprator(harmonic)
    EY_operator = init_E_fieldY_oprator()

    Ey = EY_operator @ laplace_vec * size
    Ex = EX_operator @ laplace_vec * size

    U , V = ( np.zeros( shape=(size,size) ) for _ in range(2) )

    for i in range(size):
        for j in range(size):
            V[i,j] = Ex[i*size + j%size]
            U[i,j] = Ey[i*size + j%size]

    EX_operator = None
    EY_operator = None

    return V, U

def chargeDist(laplace_vec):
    global laplace_matrix
    laplace_matrix = None

    V = np.zeros( size )
    U = np.zeros( size )

    EY_operator = init_E_fieldY_oprator()
    Ey = EY_operator @ laplace_vec * size

    for i in range(size):
        V[i] = Ey[i]
        U[i] = Ey[-i]
    return V , U
```

```python

def calculateEnergySurface(laplace_vec, charge1 , charge2):
    ret = 0.0

    for i in range(10 ,size-10):
        ret += laplace_vec[i]* charge1[i]
        ret += laplace_vec[-i]*charge2[i]

    return ret / 2

def calculateEnergyVolume(Ex ,Ey):
    ret = 0.0

    for i in range(10 , size-10):
        for j in range(size):
            ret += Ex[i,j]**2 + Ey[i,j]**2

    return ret / (8 * pi)

if __name__ == '__main__':

    #print(laplace_matrix)
    # X = np.arange(-5, 5, 0.25)
    # Y = np.arange(-5, 5, 0.25)
    # X, Y = np.meshgrid(X, Y)
    # R = np.sqrt(X**2 + Y**2)
    # print(R)

    section = [ False, False, False, False , True, True, True, True ]

    state = np.zeros(size*size)

    for i in range(size):
        state[i] = 1.
        state[-i] = -1.

    state , fn = calc( state, 0.0001 )

    V = np.zeros(shape =(size,size))

    for i in range(size):
        for j in range(size):
            V[i,j] = state[i*size + j%size]
            print(  "{0: f}".format(state[i*size + j%size])  , end = "  ")
        print()


    x_range = 1
    delta_x = 1.0 / size
    delta_y = delta_x
    y_range = x_range
    xvec = np.arange(delta_x  ,x_range+delta_x,delta_x)
    yvec = np.arange(delta_y ,y_range+delta_y,delta_y)
    Xgrid, Ygrid = np.meshgrid(xvec,yvec)

    if section[0] :
        plt.figure()
        xp = np.exp(np.arange(-5., 0.5, 0.25))
        xm = [-x for x in xp[::-1]]
        zz = [0.]
        levels = np.concatenate((xm,zz,xp))
        # plt.streamplot(Xgrid, Ygrid, Ex, Ey,
        #                color='r',linewidth=1.5,density=[0.75,1.]
        #                ,maxlength=10.)
        CS = plt.contour(V, levels,
                         origin='lower',colors='grey',
                         linestyles='solid',
                         linewidths=1.,
                         extent=(0, x_range, 0, y_range))
        plt.axis([0,x_range,0,y_range])
        plt.axes().set_aspect('equal','box')
        plt.show()

    if section[1]:
```

```python
        fig = plt.figure()
        ax = fig.gca(projection='3d')
        print(V)
        print(Xgrid)
        surf = ax.plot_surface(Xgrid, Ygrid, V,
                               linewidth=0, antialiased=False)
        plt.show()

    if section[2]:
        fig = plt.figure()
        plt.plot( [ i for i in range( len(fn) ) ] , fn )
        plt.show()

    if section[3]:
        fig = plt.figure()
        ax = fig.gca(projection='3d')
        surf = ax.plot_surface(Xgrid, Ygrid, reso_matrix(V , size, size),
                               linewidth=0, antialiased=False)
        plt.show()

    if section[4]:
        Ex, Ey =  electricfield(state)

        fig, ax = plt.subplots()
        # q = ax.quiver(X, Y, U, V)
        # ax = fig.gca(projection='2d')
        q = ax.quiver(Xgrid, Ygrid, Ex, Ey,
                               linewidth=0, antialiased=False)
        plt.show()


    if section[5]:
        V , U = chargeDist(state)
        plt.plot( xvec , V )
        plt.show()


    if section[6]:

        XX = []
        YY = []
        for i in range(40 , 80 , 3):
            size = i
            state = np.zeros(size*size)
            for j in range(size):
                state[j] = 1.
                state[-j] = -1.
            state , fn = calc( state, 0.00001 )
            Ex, Ey =  electricfield(state)
            V , U = chargeDist(state)


            YY.append(calculateEnergyVolume(Ex, Ey))
            XX.append(calculateEnergySurface(state, V , U ))
        fig = plt.figure()
        print(XX)
        print(YY)
        #plt.plot( [ i for i in range(len(XX)) ] ,  [ x / y for x , y in zip(XX, YY) ]  )
        ZZ = [ i for i in range(40 , 80 , 3) ]
        plt.plot(ZZ, XX)
        plt.plot(ZZ, YY)
        plt.show()

    if section[7]:
        size = 80
        state = np.zeros(size*size)
        for i in range(size):
            state[i] = 1.
            state[-i] = -1.

        state , fn = calc( state, 0.00001, harmonic=True )
        Ex, Ey =  electricfield(state, harmonic=True)

        V = np.zeros(shape =(size,size))
```

```
        for i in range(size):
            for j in range(size):
                V[i,j] = state[i*size + j%size]
                print(  "{0: f}".format(state[i*size + j%size])  , end = "  ")
            print()

        plt.figure()
        xp = np.exp(np.arange(-5., 0.5, 0.25))
        xm = [-x for x in xp[::-1]]
        zz = [0.]
        levels = np.concatenate((xm,zz,xp))
        # plt.streamplot(Xgrid, Ygrid, Ex, Ey,
        #                 color='r',linewidth=1.5,density=[0.75,1.]
        #                 ,maxlength=10.)
        CS = plt.contour(V, levels,
                         origin='lower',colors='grey',
                         linestyles='solid',
                         linewidths=1.,
                         extent=(0, x_range, 0, y_range))
        plt.axis([0,x_range,0,y_range])
        plt.axes().set_aspect('equal','box')
        plt.show()

        XX = []
        YY = []
        for i in range(40 , 80 , 3):
            size = i
            state = np.zeros(size*size)
            for j in range(size):
                state[j] = 1.
                state[-j] = -1.
            state , fn = calc( state, 0.00001, harmonic=True)
            Ex, Ey =  electricfield(state, harmonic=True)
            V , U = chargeDist(state)


            YY.append(calculateEnergyVolume(Ex, Ey))
            XX.append(calculateEnergySurface(state, V , U ))
        fig = plt.figure()
        print(XX)
        print(YY)
        #plt.plot( [ i for i in range(len(XX)) ] ,  [ x / y for x , y in zip(XX, YY) ]  )
        ZZ = [ i for i in range(40 , 80 , 3) ]
        plt.plot(ZZ, XX)
        plt.plot(ZZ, YY)
        plt.show()
```