15_PDEs-2

March 30, 2017

1 15 Partial Differential Equations — 2

1.1 Solving Laplace's or Poisson's equation

Still solving **Poisson's equation** for the electric potential $\Phi(\mathbf{r})$ and the charge density $\rho(\mathbf{r})$:

$$\nabla^2\Phi(x,y,z) = -4\pi\rho(x,y,z)$$

For a region of space without charges ($\rho = 0$) this reduces to **Laplace's equation**

$$\nabla^2\Phi(x,y,z) = 0$$

General solution by iteration:

$$\Phi_{i,j} = \frac{1}{4}\Big(\Phi_{i+1,j} + \Phi_{i-1,j} + \Phi_{i,j+1} + \Phi_{i,j-1}\Big) + \pi\rho_{i,j}\Delta^2$$

Jacobi method Do not change $\Phi_{i,j}$ until a complete sweep has been completed.

```
In [1]: def Laplace_Jacobi_slow(Phi):
            # Don't use, very slow AND inefficient
            Phi_new = Phi.copy()
            Nx, Ny = Phi.shape
            for xi in range(1, Nx-2):
                for yj in range(1, Ny-2):
                    Phi_new[xi, yj] = 0.25*(Phi[xi+1, yj] + Phi[xi-1, yj]
                                            + Phi[xi, yj+1] + Phi[xi, yj-1])
            Phi[:, :] = Phi_new
            return Phi
```

Fast implementation using numpy array operations (vectorized, run at C speed, not Python speed) (see the board notes (PDF) for an explanation for how to set up the numpy array operations that give you a 100-fold speed up over `Laplace_Jacobi_slow()`):

```
In [2]: def Laplace_Jacobi(Phi):
            """One update in the Jacobi algorithm"""
            Phi[1:-1, 1:-1] = 0.25*(Phi[2:, 1:-1] + Phi[0:-2, 1:-1] + Phi[1:-1, 2:]
            return Phi
```

Gauss-Seidel method Immediately use updated new values for $\Phi_{i-1,j}$ and $\Phi_{i,j-1}$ (if starting from $\Phi_{1,1}$).

Leads to *accelerated convergence* and therefore *less round-off error* (but distorts symmetry of boundary conditions... hopefully irrelevant when converged but check!)

```
In [3]: def Laplace_Gauss_Seidel(Phi):
            """One update in the Gauss-Seidel algorithm"""
            Nx, Ny = Phi.shape
            for xi in range(1, Nx-2):
                for yj in range(1, Ny-2):
                    Phi[xi, yj] = 0.25*(Phi[xi+1, yj] + Phi[xi-1, yj]
                                        + Phi[xi, yj+1] + Phi[xi, yj-1])
            return Phi
```

Fast Gauss-Seidel-like Divide the lattice into a checkerboard of black and white cells. Update the odd cells first (like Jacobi) but then use the odd cells to update the even ones (Gauss-Seidel-like; see the board notes (PDF)). This leads to faster convergence *and* can be done with numpy array operations.

```
In [4]: def Laplace_Gauss_Seidel_odd_even(Phi):
            """One update in the Gauss-Seidel algorithm on odd or even fields"""
            # odd update (uses old even)
            Phi[1:-2:2, 1:-2:2] = 0.25*(Phi[2::2, 1:-2:2] + Phi[0:-2:2, 1:-2:2] + P
            Phi[2:-1:2, 2:-1:2] = 0.25*(Phi[3::2, 2:-1:2] + Phi[1:-2:2, 2:-1:2] + P

            # even update (uses new odd)
            Phi[1:-2:2, 2:-1:2] = 0.25*(Phi[2::2, 2:-1:2] + Phi[0:-2:2, 2:-1:2] + P
            Phi[2:-1:2, 1:-2:2] = 0.25*(Phi[3::2, 1:-2:2] + Phi[1:-2:2, 1:-2:2] + P
            return Phi
```

1.1.1 *Converged* solution of the wire-in-a-box problem

Solve the box-wire problem and **make sure that the solution is converged to `tol = 1e-3`.**

```
In [5]: import numpy as np
        import matplotlib.pyplot as plt
        from mpl_toolkits.mplot3d import Axes3D
        #%matplotlib inline
        %matplotlib notebook
```

Only execute the next line if you *don't* want interactive plotting (e.g., when exporting to LaTeX/PDF or html):

```
In [6]: %matplotlib inline
```

Convience plotting functions. If you provide a filename then output is only written to a file and figures are close to conserve memory. This allows you to plot files in loops and later assemble them into movies using other programs such as ffmpeg, ImageMagick, mencoder, QuickTime 7, ...

In [7]:
```
def plot_contour(Phi, filename=None):
    """Plot Phi as a contour plot.

    Arguments
    ---------
    Phi : 2D array
          potential on lattice
    filename : string or None, optional (default: None)
          If `None` then show the figure and return the axes object.
          If a string is given (like "contour.png") it will only plot
          to the filename and close the figure but return the filename.
    """
    fig = plt.figure(figsize=(5,4))
    ax = fig.add_subplot(111)

    x = np.arange(Phi.shape[0])
    y = np.arange(Phi.shape[1])
    X, Y = np.meshgrid(x, y)
    Z = Phi[X, Y]
    cset = ax.contourf(X, Y, Z, 20, cmap=plt.cm.coolwarm)
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_aspect(1)

    cb = fig.colorbar(cset, shrink=0.5, aspect=5)
    cb.set_label(r"potential $\Phi$ (V)")

    if filename:
        fig.savefig(filename)
        plt.close(fig)
        return filename
    else:
        return ax


def plot_surf(Phi, filename=None, offset=-20, zlabel=r'potential $\Phi$ (V)
              elevation=40, azimuth=20):
    """Plot Phi as a 3D plot with contour plot underneath.

    Arguments
    ---------
    Phi : 2D array
          potential on lattice
    filename : string or None, optional (default: None)
          If `None` then show the figure and return the axes object.
          If a string is given (like "contour.png") it will only plot
          to the filename and close the figure but return the filename.
    offset : float, optional (default: 20)
```

```
        position the 2D contour plot by offset along the Z direction
        under the minimum Z value
    zlabel : string, optional
        label for the Z axis and color scale bar
    elevation : float, optional
        choose elevation for initial viewpoint
    azimuth : float, optional
        chooze azimuth angle for initial viewpoint
    """

    x = np.arange(Phi.shape[0])
    y = np.arange(Phi.shape[1])
    X, Y = np.meshgrid(x, y)
    Z = Phi[X, Y]

    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    surf = ax.plot_surface(X, Y, Z, cmap=plt.cm.coolwarm, rstride=2, cstrid
    cset = ax.contourf(X, Y, Z, 20, zdir='z', offset=offset+Z.min(), cmap=p

    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel(zlabel)
    ax.set_zlim(offset + Z.min(), Z.max())

    ax.view_init(elev=elevation, azim=azimuth)

    cb = fig.colorbar(surf, shrink=0.5, aspect=5)
    cb.set_label(zlabel)

    if filename:
        fig.savefig(filename)
        plt.close(fig)
        return filename
    else:
        return ax
```

The following solution can also write out intermediate solution steps as PNG graphics (commented out for speed). Filenames and directory names are hard-coded.(Important: `plt.ioff()` to disable interactive plotting in the notebook frontend, otherwise it is very slow.)

Put all images into a directory `wire`:

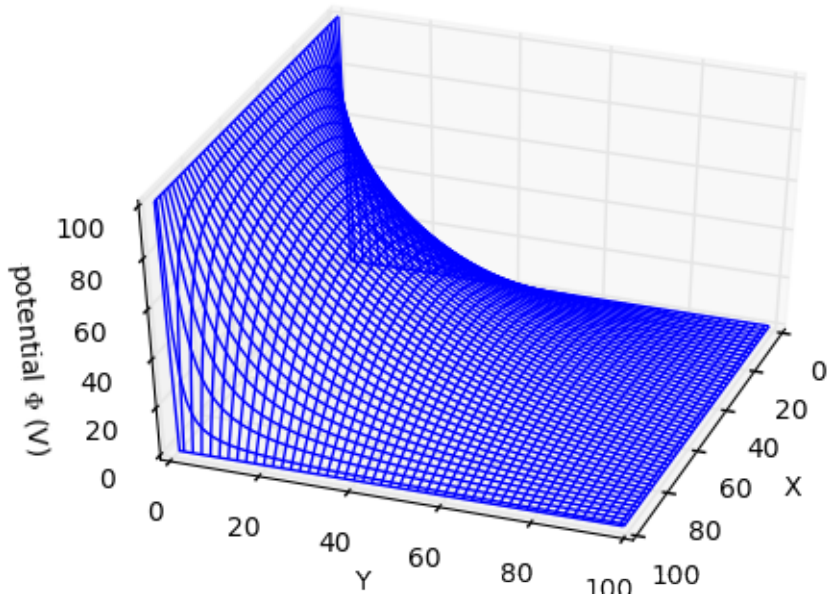
```
In [8]: import os
        os.mkdir("./wire")

In [9]: plt.ioff()  # suppress interactive plotting

        Max_iter=30000
```

```
        tol = 1e-3
        Nmax = 100
        Phi = np.zeros((Nmax, Nmax), dtype=np.float64)
        Phi_old = np.zeros_like(Phi)

        # initialize boundaries
        # everything starts out zero so nothing special for the grounded wires
        Phi[:, 0] = 100     # wire at y=0 at 100 V

        for n_iter in range(Max_iter):
            Phi_old[:, :] = Phi
            Phi = Laplace_Gauss_Seidel_odd_even(Phi)
            DeltaPhi = np.linalg.norm(Phi - Phi_old)
            if DeltaPhi < tol:
                print("Laplace_Gauss_Seidel_odd_even converged in {0} iterations to
                plot_contour(Phi, filename="wire/phi_{0:08d}.png".format(n_iter))
                break
            if n_iter % 1000 == 0:
                print("Iteration {0}".format(n_iter), end="\r")
                plot_contour(Phi, filename="wire/phi_{0:08d}.png".format(n_iter))
        else:
            print("Laplace_Gauss_Seidel_odd_even did NOT converge in {0} iterations

        plt.ion()
```

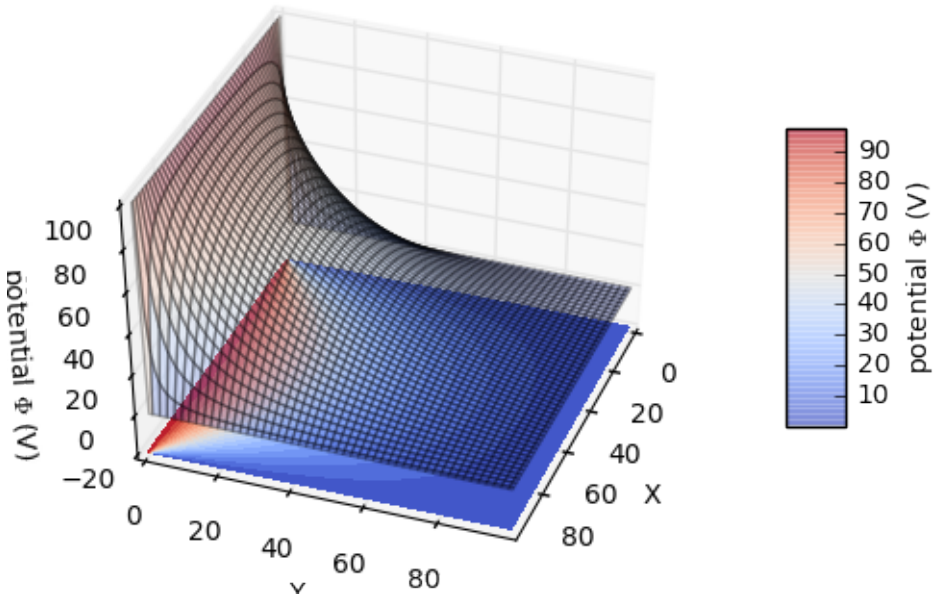
```
Laplace_Gauss_Seidel_odd_even converged in 7558 iterations to 0.0009995267090299163
```

Wireframe plot as usual:

```
In [10]: # plot Phi(x,y)
         x = np.arange(Nmax)
         y = np.arange(Nmax)
         X, Y = np.meshgrid(x, y)

         Z = Phi[X, Y]

         fig = plt.figure()
         ax = fig.add_subplot(111, projection='3d')
         ax.plot_wireframe(X, Y, Z, rstride=2, cstride=2)

         ax.set_xlabel('X')
         ax.set_ylabel('Y')
         ax.set_zlabel(r'potential $\Phi$ (V)')

         ax.view_init(elev=40, azim=20)
```




```
In [11]: ax = plot_surf(Phi, elevation=40, azimuth=20)
```



1.2 Successive Over Relaxation (SOR)

Accelerate convergence with the scheme

$$r_{i,j} = \Phi^{\text{new}}_{i,j} - \Phi^{\text{old}}_{i,j} \tag{1}$$

$$\Phi^{\text{new}}_{i,j} = \Phi^{\text{old}}_{i,j} + \omega r_{i,j} \tag{2}$$

where the new solution is computed with the Gauss-Seidel scheme.
Values of $1 \leq \omega \leq 2$ may work well, $\omega > 2$ can lead to numerical instabilities. Experiment!
Create images in directory `./wire_SOR`

In [12]:
```
os.mkdir("./wire_SOR")
```

In [13]:
```
# %%timeit
plt.ioff()

Max_iter=10000
tol = 1e-3
Nmax = 100
omega = 1.99

Phi = np.zeros((Nmax, Nmax), dtype=np.float64)
Phi_old = np.zeros_like(Phi)
residual = np.zeros_like(Phi)

# initialize boundaries
# everything starts out zero so nothing special for the grounded wires
Phi[:, 0] = 100      # wire at y=0 at 100 V

for n_iter in range(Max_iter):
    Phi_old[:, :] = Phi
    Phi = Laplace_Gauss_Seidel_odd_even(Phi)
    residual[:, :] = Phi - Phi_old
    DeltaPhi = np.linalg.norm(residual)
    if DeltaPhi < tol:
        print("SOR converged in {0} iterations to {1}".format(n_iter+1, De
        plot_contour(Phi, "wire_SOR/phi_{0:08d}.png".format(n_iter))
        break
    # SOR
    Phi[:, :] = Phi_old + omega*residual # = omega*Phi + (1-omega)*Phi_old

    if n_iter % 100 == 0:
        print("Iteration {0}".format(n_iter), end="\r")
        plot_contour(Phi, "wire_SOR/phi_{0:08d}.png".format(n_iter))
else:
    print("SOR did NOT converge in {0} iterations, DeltaPhi={1}".format(n_

plt.ion()
```

```
SOR converged in 3797 iterations to 0.0009987117944037717
```

Run to convergence (tol = 1e-03). Record the number of steps required (and visually check the solution). * Start with $\omega = 1$. What should you get? * Change ω and try to get faster convergence? Who can get it to converge in the fewest number of steps?

Tuning the code Alternative implementation that uses more in-place array operations. (However, only marginally faster (~5%) than the previous solution.)

```
In [14]: %%timeit
         plt.ioff()

         Max_iter=10000
         tol = 1e-3
         Nmax = 100
         omega = 1.99

         Phi = np.zeros((Nmax, Nmax), dtype=np.float64)
         Phi_old = np.zeros_like(Phi)

         # initialize boundaries
         # everything starts out zero so nothing special for the grounded wires
         Phi[:, 0] = 100     # wire at y=0 at 100 V

         for n_iter in range(Max_iter):
             Phi_old[:, :] = Phi
             Phi = Laplace_Gauss_Seidel_odd_even(Phi)
             DeltaPhi = np.linalg.norm(Phi - Phi_old)
             if DeltaPhi < tol:
                 print("SOR converged in {0} iterations to {1}".format(n_iter+1, De
                 #plot_contour(Phi, "wire_SOR/phi_{0:08d}.png".format(n_iter))
                 break
             # SOR
             # Phi[:, :] = Phi_old + omega*residual  = omega*Phi + (1-omega)*Phi_ol
             Phi *= omega
             Phi += (1-omega)*Phi_old

             if n_iter % 100 == 0:
                 print("Iteration {0}".format(n_iter), end="\r")
                 #plot_contour(Phi, "wire_SOR/phi_{0:08d}.png".format(n_iter))
         else:
             print("SOR did NOT converge in {0} iterations, DeltaPhi={1}".format(n_

         plt.ion()

SOR converged in 3797 iterations to 0.0009998711794429458 8
SOR converged in 3797 iterations to 0.0009998711794429458 8
SOR converged in 3797 iterations to 0.0009998711794429458 8
SOR converged in 3797 iterations to 0.0009998711794429458 8
```

```
1 loop, best of 3: 499 ms per loop
```

1.3 Simple example of Poisson's equation: Wire and charge

Add a positive charge and a negative charge in the box.

Now we need to solve **Poisson's equation**.

$$\nabla^2\Phi(x,y,z) = -4\pi\rho(x,y,z)$$

Finite difference solution:

$$\Phi_{i,j} = \frac{1}{4}\Big(\Phi_{i+1,j} + \Phi_{i-1,j} + \Phi_{i,j+1} + \Phi_{i,j-1}\Big) + \pi\rho_{i,j}\Delta^2$$

1.3.1 Poisson Solver

Modify the Laplace solvers to now solve Poisson's equation:

```
In [15]: import numpy as np

         def Poisson_Jacobi(Phi, rho, Delta=1.):
             """One update in the Jacobi algorithm for Poisson's equation"""
             Phi[1:-1, 1:-1] = 0.25*(Phi[2:, 1:-1] + Phi[0:-2, 1:-1] + Phi[1:-1, 2:
                               + np.pi * Delta**2 * rho[1:-1, 1:-1]
             return Phi

         def Poisson_Gauss_Seidel(Phi, rho, Delta=1.):
             """One update in the Gauss-Seidel algorithm for Poisson's equation"""
             Nx, Ny = Phi.shape
             for xi in range(1, Nx-2):
                 for yj in range(1, Ny-2):
                     Phi[xi, yj] = 0.25*(Phi[xi+1, yj] + Phi[xi-1, yj]
                                         + Phi[xi, yj+1] + Phi[xi, yj-1]) \
                                   + np.pi * Delta**2 * rho[xi, yj]
             return Phi

         def Poisson_Gauss_Seidel_odd_even(Phi, rho, Delta=1.):
             """One update in the Gauss-Seidel algorithm on odd or even fields"""
             a = np.pi * Delta**2
             # odd update (uses old even)
             Phi[1:-2:2, 1:-2:2] = 0.25*(Phi[2::2, 1:-2:2] + Phi[0:-2:2, 1:-2:2]
                                         + Phi[1:-2:2, 2::2] + Phi[1:-2:2, 0:-2:2])
             Phi[2:-1:2, 2:-1:2] = 0.25*(Phi[3::2, 2:-1:2] + Phi[1:-2:2, 2:-1:2]
                                         + Phi[2:-1:2, 3::2] + Phi[2:-1:2, 1:-2:2])

             # even update (uses new odd)
             Phi[1:-2:2, 2:-1:2] = 0.25*(Phi[2::2, 2:-1:2] + Phi[0:-2:2, 2:-1:2]
                                         + Phi[1:-2:2, 3::2] + Phi[1:-2:2, 1:-1:2])
             Phi[2:-1:2, 1:-2:2] = 0.25*(Phi[3::2, 1:-2:2] + Phi[1:-2:2, 1:-2:2]
```

```
                                                + Phi[2:-1:2, 2::2] + Phi[2:-1:2, 0:-2:2])
            return Phi
```

The following code can write partial solutions as images. To make it a bit more comfortable, we generate a directory for the images from Python:

In [16]:
```
import os

def makedir(dirname):
    try:
        os.mkdir(dirname)
    except OSError:
        pass
    return dirname
```

Solve the Poisson problem with SOR (as above):

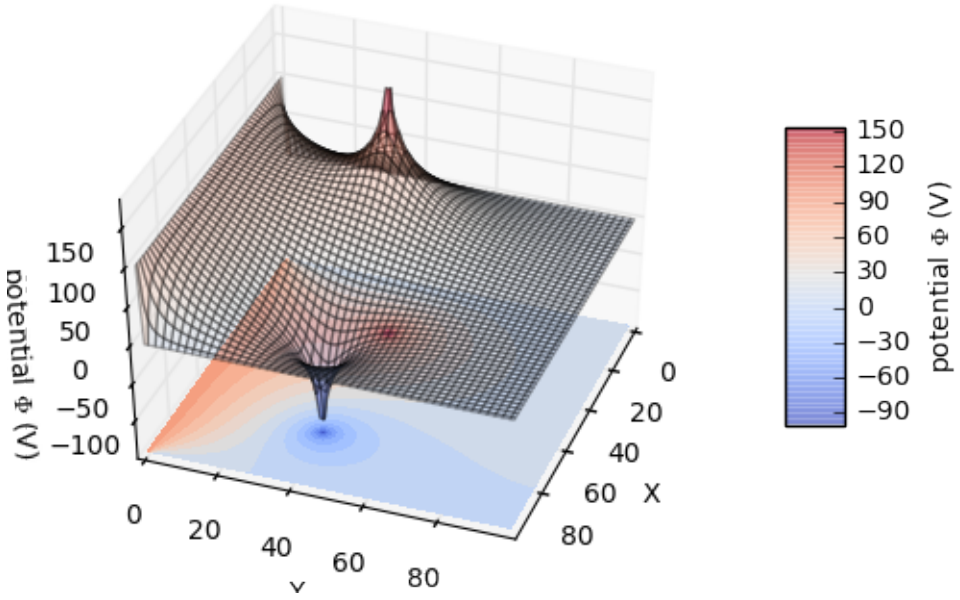
In [17]:
```
plt.ioff()

Nmax = 100
Max_iter = 10000
omega = 1.99

dirname = makedir("dipole_wire")
filename = os.path.join(dirname, "phi_{0:08d}.png")

Phi = np.zeros((Nmax, Nmax), dtype=np.float64)
Phi_old = np.zeros_like(Phi)
rho = np.zeros_like(Phi)

# initialize boundaries
# everything starts out zero so nothing special for the grounded wires
Phi[:, 0] = 100      # wire at y=0 at 100 V
rho[25:27, 39:41] = 5.0
rho[75:77, 39:41] = -5.0

Delta = 1.0

for n_iter in range(Max_iter):
    Phi_old[:, :] = Phi
    Phi = Poisson_Gauss_Seidel_odd_even(Phi, rho, Delta=Delta)
    residual[:, :] = Phi - Phi_old
    DeltaPhi = np.linalg.norm(residual)
    if DeltaPhi < tol:
        print("SOR converged in {0} iterations to {1}".format(n_iter+1, De
        plot_contour(Phi, filename=filename.format(n_iter))
        break
    # SOR
    Phi[:, :] = Phi_old + omega*residual # = omega*Phi + (1-omega)*Phi_old
```

```
            if n_iter % 100 == 0:
                print("Iteration {0}".format(n_iter), end="\r")
                plot_contour(Phi, filename=filename.format(n_iter))
        else:
            print("SOR did NOT converge in {0} iterations, DeltaPhi={1}".format(n_

        print("Images in {0}".format(dirname))
        plt.ion()
```

```
SOR converged in 3823 iterations to 0.0009991605868815044
Images in dipole_wire
```

```
In [18]: plot_surf(Phi, elevation=40, azimuth=20)
```

```
Out[18]: <matplotlib.axes._subplots.Axes3DSubplot at 0x10e0de2e8>
```



1.3.2 Charge density

Given a potential Φ, Poisson's equation gives the charge density:

$$\rho(\mathbf{x}) = -\frac{1}{4\pi}\nabla^2\Phi(\mathbf{x})$$

Discretized Laplacian

$$\nabla^2\Phi \approx \frac{\Phi(x+\Delta x, y) + \Phi(x-\Delta x, y) + \Phi(x, y+\Delta y) + \Phi(x, y-\Delta y) - 4\Phi(x, y)}{\Delta^2}$$

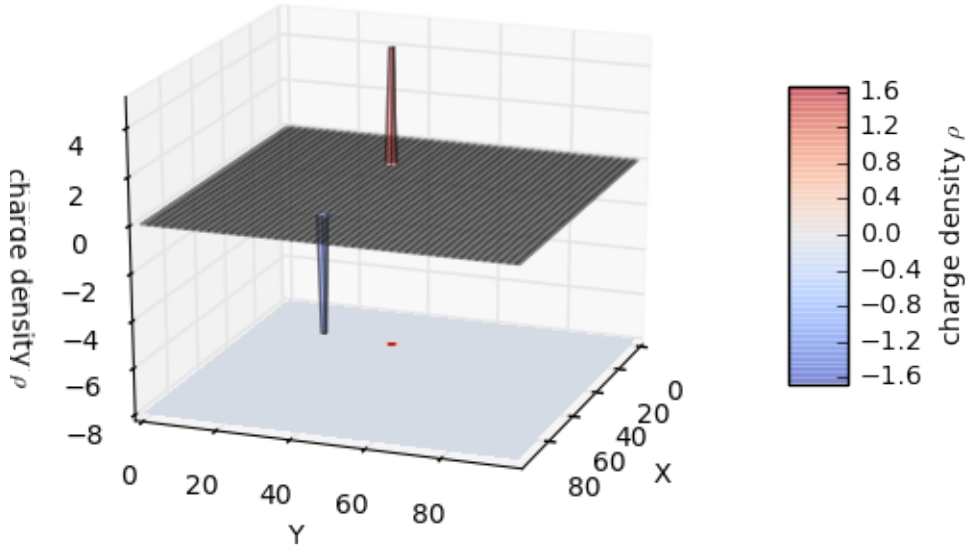
```
In [19]: def laplacian2d(f, Delta=1):
             """Finite difference approximation of Del^2 f.

             Arguments
             ---------
             f : M x N matrix
             Delta : float

             Returns
             -------
             M x N matrix, boundaries set to 0
             """

             L = np.zeros_like(f, dtype=np.float64)
             L[1:-1, 1:-1] = f[1:-1, 2:] + f[1:-1, 0:-2] + f[2:, 1:-1] + f[0:-2, 1:
             return L/Delta**2

         def laplacian2dsimple(f, Delta=1):
             L = np.zeros_like(f, dtype=np.float64)
             for i in range(1, L.shape[0]-1):
                 for j in range(1, L.shape[1]-1):
                     L[i, j] = f[i, j+1] + f[i, j-1] + f[i+1, j] + f[i-1, j] - 4*f[
             return L/Delta**2

In [20]: rhox = - laplacian2d(Phi)/(4*np.pi)
```

Does `rhox` show the charges of $+5$ and -5 that we introduced with the charge density ρ?

```
In [21]: print(rhox.min())
         print(rhox.max())

-5.00000418672
5.0
```

Plot the charge density:

```
In [22]: plot_surf(rhox, zlabel=r"charge density $\rho$", elevation=20, azimuth=20,

Out[22]: <matplotlib.axes._subplots.Axes3DSubplot at 0x10d1d72b0>
```



The point charges are found correctly but the charge on the wire appear to be absent. The problem is that the Laplacian is not defined on the boundary.

(Also, if the charges are only defined on a single grid cell then the visualization is sometimes not able to show it correctlty.)

Testing the implementation of Laplacian

```
In [23]: import numpy as np

         def test_laplacian2d():
             ftest = np.random.random((200, 200))
             assert np.allclose(laplacian2d(ftest), laplacian2dsimple(ftest))

         test_laplacian2d()

In [ ]:
```