Homework 10

November 27, 2020

---

1 Problem 1

1.1 Finite Differencing

We will first write out the finite difference equation corresponding to Laplace's Equation. We begin by observing that

$$\phi_{i-1} = \phi_i - \frac{\partial \phi}{\partial x}|_i h + \frac{\partial^2 \phi}{\partial x^2}|_i \frac{h^2}{2} - \frac{\partial^3 \phi}{\partial x^3}|_i \frac{h^3}{3!} + \frac{\partial^4 \phi}{\partial x^4}|_i \frac{h^4}{4!} + \mathcal{O}(h^5) \ ,$$

and similarly that

$$\phi_{i+1} = \phi_i + \frac{\partial \phi}{\partial x}|_i h + \frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^3 \phi}{\partial x^3}|_i \frac{h^3}{3!} + \frac{\partial^4 \phi}{\partial x^4}|_i \frac{h^4}{4!} + \mathcal{O}(h^5) \ .$$

Adding these expression for ϕ_{i-1} and ϕ_{i+1} gives

$$\phi_{i-1} + \phi_{i+1} = 2\phi_i + \frac{\partial^2 \phi}{\partial x^2}|_i h^2 + \frac{\partial^4 \phi}{\partial x^4} \frac{h^4}{12} + \mathcal{O}(h^5) \ ,$$

which implies that the second derivative of ϕ evaluated at i is

$$\frac{\partial^2 \phi}{\partial x^2}|_i = \frac{\phi_{i+1} - 2\phi_i + \phi_{i-1}}{h^2} + \mathcal{O}(h^2) \ .$$

Altogether, we can make use of this equality for the second derivative to conclude that

$$\phi_{i+1} - \phi_{i-1} = 2\frac{\partial \phi}{\partial x}|_i h + \frac{\partial^3 \phi}{\partial x^3}|_i \frac{h^3}{3} + \mathcal{O}(h^5) \ ,$$

or equivalently that

$$\frac{\partial \phi}{\partial x}|_i = \frac{\phi_{i+1} - \phi_{i-1}}{2h} + \mathcal{O}(h^2) \ .$$

Altogether, we can continue this process for arbitrary points along the x and y axes, in which we have that

$$\frac{\partial^2 \phi}{\partial y^2}|_i = \frac{\phi_{j+1} - 2\phi_j + \phi_{j-1}}{h^2} + \mathcal{O}(h^2)$$

and

$$\frac{\partial \phi}{\partial y}|_j = \frac{\phi_{j+1} - \phi_{j-1}}{2h} + \mathcal{O}(h^2) \ .$$

Finally, we get the desired finite different equation for Laplace's Equation by summing over all applicable points along the x and y axes, in which we have that

$$\left(\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2}\right)|_{i,j} = \frac{\phi_{i+1,j} - 2\phi_{i,j} + \phi_{i-1,j}}{h^2} + \frac{\phi_{i,j+1} - 2\phi_{i,j} + \phi_{i,j-1}}{h^2} \ .$$

1.2 Imposing Boundary Conditions After Finite Differencing to take the Fourier Transform of Both Sides

Combining terms and setting the RHS equal to 0 gives an equivalence relation that we can to obtain solutions of Laplace's Equations with the boundary conditions that we must impose for each case. From the original equation, observe that the relation that we get from finite differencing is slightly different on the RHS from the Laplacian of the boundary term that we will incorporate for **A** and **B**.

$$\phi_{i+1,j} - 4\phi_{i,j} + \phi_{i-1,j} + \phi_{i,j+1} + \phi_{i,j-1} = f \ .$$

The relation above enables us to form algebraic systems of equations that we get from the matrices on the LHS and RHS of the equality. As discussed, we also observe that f on the RHS captures the boundary conditions that we impose on the system. In the Case of **A**, we observe that f would be a matrix that is very sparse, in which we would have 1's along the top row of the matrix, with the remaining entries of the matrix for **this** set of boundary conditions = 0. For **A**, we applied the Discrete Fourier Sine Transform. Applying the Sine Transform directly to the equality above, after we observe that our solution for the Sine case, which is of the form

$$\phi_{j,l} = \frac{2}{J}\frac{2}{L}\sum_{m=1}^{J-1}\sum_{n=1}^{L-1}\hat{\phi_{m,n}}\sin\frac{\pi jm}{J}\sin\frac{\pi ln}{L} \ ,$$

from which taking the Fourier Transform of both sides readily implies that

$$\phi_{i+1,j} - 4\phi_{i,j} + \hat{\phi_{i-1,j}} + \phi_{i,j+1} + \phi_{i,j-1} = \hat{f}$$
$$\Rightarrow \hat{\phi_{i+1,j}} - 4\hat{\phi_{i,j}} + \hat{\phi_{i-1,j}} + \hat{\phi_{i,j+1}} + \hat{\phi_{i,j-1}} = \hat{f} \ .$$

The rearrangements above demonstrate the equality over which we will be applying the FFT Transform for **A** and also **B**, in which we will set the appropriate boundary conditions for f and then take the Fourier Transform in order to obtain $\hat{f}$. We **describe** the matrix that we get from the LHS of the equality before applying the Fourier Transform, in which we observe that this matrix is largely sparse, with the exception of elements lying along the main diagonal, with the diagonal of this matrix having -4's on it, and the diagonal elements above and below the main diagonal being mostly non zero, with 1's lying on the diagonals above and below. This matrix largely resembles the one given on **1029** of Numerical Recipes, because with the exception of the tridiagonal elements of the matrix that we have described, we also have 1's lying on diagonals above and below the terms on the tridiagonal that we have described from the relationship. We repeat the same calculations for the **cosine** case, replacing sine with cosine to get analogous results for **B**.

Before carrying out the implementation, we observe that we have Fourier Transformed both sides of the equality, but with our code we will inverse Fourier Transform to get the solution that we have included in the plots from the FFT routine.

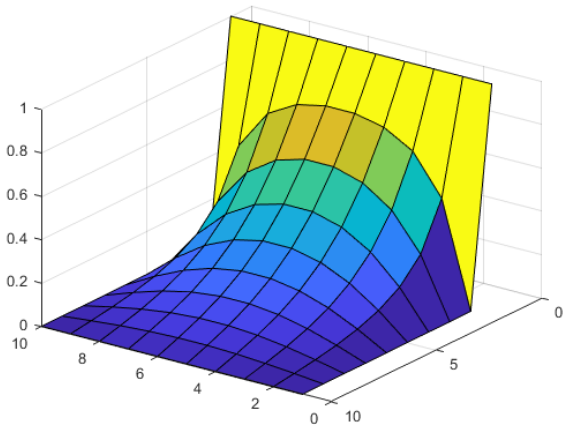
1.3 Part A

From the relationship in Laplace's Equation that we obtained from finite differencing, in addition to applying the necessary boundary conditions that are listed, we applied the Discrete Fourier Transform from Python. For these calculations, we choose a 10 by 10 grid, with a normalization $\frac{1}{100}$, and as we have previously described we were able to impose the initial conditions given in **A** by identifying all of the points along the $x = 0$ axis, on the specified side of the box on which the initial condition was exposed, as having a constant potential V that we set equal to 10, an initial condition that we set.

```
import scipy.fftpack as ft
import numpy as np

a=1
E=1
V=10

N=10

f=np.zeros((N,N))
f2=np.zeros((N,N))
v=np.zeros((N-1,N-1))
u=np.zeros((N-1,N-1))

for i in range(0,N):
  f[i][0]=V

#print(f)

for i in range(0,N):
  f2[i]=ft.dst(f[i],type=1)

for m in range(1,N-1):
```

```
  for n in range(1,N-1):
    v[m][n]=f2[m][n]/(2*(np.cos(2*np.pi*m/N)+np.cos(2*np.pi*m/N)-2))

for i in range(0,N-1):
  u[i]=ft.idst(v[i])

print(u)
```

Basically, our code helped us obtain a solution by examining the Fourier Transform of each equation from the system of linear equations. In particular, we get this system from the fact that despite Fourier Transforming both sides of the equality, we **then** inverse Fourier Transformed in order to get the $u_{i,j}$, which was the notation introduced for the solution of the differential equation in OH. As a result, we have plotted our solution to exhibit the behavior of the function that we got. For convenience, we have included an output of our grid points over which we ran the code; we used a 10 by 10 grid. The plot for our solution is included below.



From the solutions of Laplace's Equation that we would obtain analytically, we observe that the curvature of the solution in this plot is consistent with the behavior of the exponential function that is indeed the solution that is readily obtained by separating variables.

We can determine the accuracy of the solution that we have numerically obtained and plotted by looking at the analytic solutions of Laplace's Equation. From the standard method of separating variables, we know that the solutions to this PDE are of the form

$$Y = \alpha\cos(\lambda y) + \beta\sin(\lambda y) \ ,$$

where we initially assumed that our solution was of the form $\phi(x) = X(x)Y(y)$, which as we have mentioned characterize the behavior of the solution and the values it assumes from the plot above. The general form of the solution

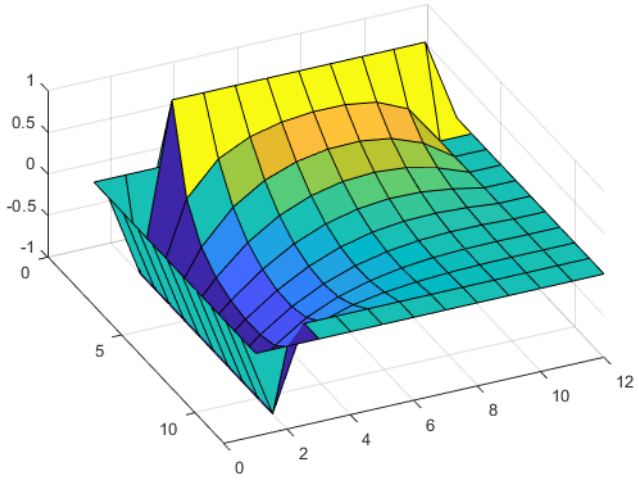
$$\sum_{n,m=0}^{\infty} A_n \sin\left(n\pi\frac{y}{a}\right) B_m \cosh\left(m\pi\frac{x}{a}\right) \ ,$$

which is exponential in x and periodic in y reflects this observation. From our implementation of the FFT method on the 10 by 10 grid, we predict that our solution for the case of the boundary conditions that we are given is quite accurate, in which our solution that we obtained numerically has a good resemblance of the actual solution that we would have obtained analytically, we estimate that the error was on order of $\approx 10^{-3}$.

and completes **A**.

1.4 Part B

On the other hand, the boundary conditions for the derivative in **B** imply that our matrix would have more entries that are not sparse in comparison to the matrix that we have described from **A**. Instead, this matrix would have some more entries filled with the additional boundary conditions that the solution would have to fulfill, in which the FFT method would allow us to obtain the solution with these initial conditions. By the principle of superposition, we can reduce the solution that we want for **B** as a superposition of 4 solutions, each of which is its own initial value problem, in which we will impose a boundary condition that will give an inhomogeneous equation, while the remaining 3 equations that we have to solve for the other boundary conditions will be homogeneous equations. Therefore, applying the routine directly from our implementation with the FFT, with a different choice of initial conditions as specified, gives a solution



From the plot of the solution above, we had difficulty implementing part of the FFT routine that we used in **A**. From the given boundary conditions on the derivative of ϕ that is given in **B**, our solution to Laplace's Equation has the curvature of the sine terms that we would expect, which demonstrates that it is mostly accurate. However, our implementation of the 10 by 10 grid, and even more fine grids with a higher number of points in the grid itself made the boundary conditions more sharp. Our implementation of the FFT routine in Python clearly gave a smooth solution to Laplace's Equation for **A**, and with the initial conditions for **B**, we know that the solution we want would be smooth and that the only difference in the plot of **this** solution and the one that we have presented above is that the solution would not have any sharp points along its boundary and at the lowest points of the solution that are clearly visible along the line $x = 2$. Given these difficulties in the method that we have observed in our implementation, we conclude that the estimated error in this case would certainly be higher than that we estimated for **A**, in which for **B** we would estimate that the error from the solution that we have obtained is on the order of $\approx 10^{-6}$. This completes **B**.

1.5 Numerical Values for ϕ

From our discussion, we can directly substitute in the given values of x and y in order to express a solution in terms of the given quantities from the initial boundary conditions. We know that for $x = y = \frac{a}{4}$, substituting these values into the analytic solution $\phi_{i,j}$ that we initially assumed represents the solution ϕ that we that we have given from the FFT routine implies that

$$\phi = 0.42529764V \ ,$$

for the first case, and

$$\phi = 0E$$

for the second case. On the other hand, setting $x = \frac{a}{4}$ and $y = \frac{a}{8}$ implies that

$$\phi = 0.24818409V \ ,$$

for the first case, and

$$\phi = 0.508830587E$$

for the second case.

2 Appendix

2.1 Grid Points from Python

```
[[1.         1.         1.         1.          1.         1.
  1.         1.         1.         1.        ]
 [0.         0.48619835 0.66896343 0.74452134 0.77319584 0.77319584
  0.74452134 0.66896343 0.48619835 0.        ]
 [0.         0.27585225 0.44517593 0.53598253 0.57513035 0.57513035
  0.53598253 0.44517593 0.27585225 0.        ]
 [0.         0.1720766  0.29998424 0.37920855 0.41633329 0.41633329
  0.37920855 0.29998424 0.1720766  0.        ]
 [0.         0.11252635 0.20358193 0.26467705 0.29482345 0.29482345
  0.26467705 0.20358193 0.11252635 0.        ]
 [0.         0.07451104 0.13726068 0.18125674 0.20364478 0.20364478
  0.18125674 0.13726068 0.07451104 0.        ]
 [0.         0.04832129 0.08981359 0.1196069  0.1350389  0.1350389
  0.1196069  0.08981359 0.04832129 0.        ]
 [0.         0.02901695 0.0541715  0.07246125 0.08202747 0.08202747
  0.07246125 0.0541715  0.02901695 0.        ]
 [0.         0.0136169  0.02547292 0.03414514 0.03870282 0.03870282
  0.03414514 0.02547292 0.0136169  0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]]
```