University of Colorado Boulder

APPM 4660 — Numerical Analysis 2
Final Project

Fast Poisson Solver and Integral Equations for 2D Elliptical PDEs

---

Cole Sturza
Spas Angelov

May 3, 2021

1 Introduction and Background

The Laplace and Poisson equation describe many useful real world phenomenon in electrostatics, mechanical engineering, and theoretical physics. For example, the solution of Poisson's equation can be used to describe the electric field generated by a set of stationary charges. Poisson's equation is a generalization of Laplace's equation, which sets the density term to zero. The ability to solve these equations quickly can help use predict how these physics systems will behave. In this paper we will look at various techniques for quickly finding the solution to these two equations.

2 Project Aims

We aim to look at a faster approach for solving the typical finite difference system of elliptical two dimensional PDEs. The method uses the Fast Fourier Transform (FFT) algorithm to quickly solve the finite difference system. We will also look into if this method can be applied to non-uniform and/or non-square meshes. Following looking into this approach we will see how the results of the former method compare to a approach using integral equations. The two PDEs we will apply the two aforementioned methods to are the Laplace and Poisson equations.

Laplace's Equation:

$$\begin{aligned} -\nabla^2\psi &= 0 \quad \text{or} \quad \text{inside} \quad \Omega \in \mathbb{R}^2 \\ \psi(x,y) &= g(x,y) \quad \text{on} \quad \partial\Omega \end{aligned} \tag{1}$$

Poisson's Equation:

$$\begin{aligned} -\nabla^2\varphi &= f(x,y) \quad \text{inside} \quad \Omega \in \mathbb{R}^2 \\ \varphi(x,y) &= g(x,y) \quad \text{on} \quad \partial\Omega \end{aligned} \tag{2}$$

We will look at a multitude of examples and compare the accuracy between the two methods.

3 Approach/Methods

We will start by setting up the finite difference scheme for Poisson's equation. The same process can be used for Laplace's equation too. To keep things simple we will use a uniform square mesh with N interior points in each direction over the region $\Omega = [0,a] \times [0,b]$. The boundary values for this problem are given as:

$$\varphi(x,0) = g(x,0), \quad \varphi(x,b) = g(x,b), \quad 0 < x < a,$$

$$\varphi(0,y) = g(0,y), \quad \varphi(a,y) = g(a,y), \quad 0 < y < b.$$

Next, we will need to approximate the two second order partial derivatives. To do this we will use a five point stencil with the center difference methods:

$$\frac{\partial^2 \varphi}{\partial x^2} \approx \frac{\varphi_{i+1,j} - 2\varphi_{ij} + \varphi_{i-1,j}}{h^2} \quad \text{and} \quad \frac{\partial^2 \varphi}{\partial y^2} \approx \frac{\varphi_{i,j+1} - 2\varphi_{ij} + \varphi_{i,j-1}}{h^2} \ . \tag{3}$$

Substituting (3) into (2) gives us the following:

$$4\varphi_{ij} - \varphi_{i,j-1} - \varphi_{i-1,j} - \varphi_{i+1,j} - \varphi_{i,j+1} = h^2 f(ih, jh) \ . \tag{4}$$

We can now use (4) to build a system of equations of the form

$$\boldsymbol{K2D}\ \mathbf{U} = \mathbf{F} \tag{5}$$

to solve for the solution to Poisson's equation. For clarity, $\boldsymbol{K2D}$ is a $N^2 \times N^2$ matrix, and $\mathbf{U}$ and $\mathbf{F}$ are $N^2 \times 1$ vectors. The elements of these vectors are written as U_{ij} and F_{ij}. Where i corresponds to a x value in the mesh and j a y value. The equations for the interior points of the mesh are the same as (4). The points on the edges of the mesh need to factor in the Dirichlet boundary conditions and are as follows:

$$4\varphi_{1,j} - \varphi_{1,j-1} - \varphi_{2,j} - \varphi_{1,j+1} = h^2 f(h, jh) + g(h, jh) = F_{1,j}$$

$$4\varphi_{N,j} - \varphi_{N,j-1} - \varphi_{N-1,j} - \varphi_{N,j+1} = h^2 f(Nh, jh) + g(Nh, jh) = F_{N,j}$$

$$4\varphi_{i,1} - \varphi_{i-1,1} - \varphi_{i+1,1} - \varphi_{i,2} = h^2 f(ih, h) + g(ih, h) = F_{i,1}$$

$$4\varphi_{i,N} - \varphi_{i,N-1} - \varphi_{i-1,N} - \varphi_{i+1,N} = h^2 f(ih, Nh) + g(ih, Nh) = F_{i,N}$$

The matrix $\boldsymbol{K2D}$ is sparse and can be created with blocks of size N from the second difference matrix K. K is a tridiagonal sparse matrix found using the finite difference scheme for the Poisson equation in one dimension. It takes the form:

$$K = \begin{bmatrix} 2 & -1 & & \\ -1 & 2 & -1 & \\ & \cdot & \cdot & \cdot \\ & & -1 & 2 \end{bmatrix} \ .$$

Using K, we can write the two dimensional version $\boldsymbol{K2D}$ as a is block tridiagonal matrix with tridiagonal blocks. The form of $\boldsymbol{K2D}$ is as follows:

$$\boldsymbol{K2D} = \begin{bmatrix} K+2I & -I & & \\ -I & K+2I & -I & \\ & \cdot & \cdot & \cdot \\ & & -I & K+2I \end{bmatrix} \ . \tag{6}$$

Figure 1 shows the structure of $\boldsymbol{K2D}$ more explicitly for a system with $N = 5$ interior

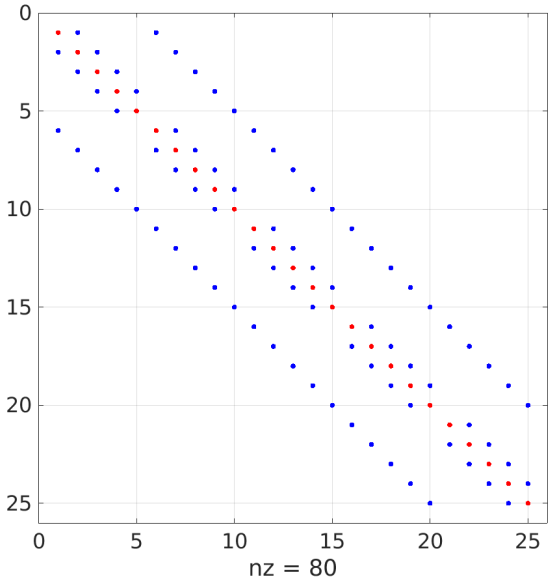


Figure 1: Structure of the $\boldsymbol{K2D}$ matrix with $N = 5$. The blue points represent the value -1 and the red 4.

points. The matrix is very sparse and for $N = 5$ there are only 80 nonzero values.

The simplest method to solve the equation $\boldsymbol{K2D}\ \mathbf{U} = \mathbf{F}$ is by inverting $\boldsymbol{K2D}$ and multiplying on the left to get $\mathbf{U} = \boldsymbol{K2D}^{-1}\mathbf{F}$. Computationally, this is very inefficient because it requires $\mathcal{O}(n^3)$ operations. This is extremely inefficient because $\boldsymbol{K2D}$ is of size $N^2 \times N^2$ and as the mesh becomes finer the time to solve the system will grow exponentially. This is especially bad because the centered difference method has a truncation error of $\mathcal{O}(h^2)$, so to achieve a small error over a large region we will need a significantly large number of points. To combat this we will use the special form of the matrix $\boldsymbol{K2D}$ to solve the system much faster in $\mathcal{O}(n \log(n))$ operations using a Fast Poisson Solver.

3.1 Fast Poisson Solver

Fast Poisson solver's are a class of methods that solve Poisson/Laplace's equation very efficiently. The fast Poisson solver we will look at is one that utilizes the FFT to efficiently solve the $\boldsymbol{K2D}\ \mathbf{U} = \mathbf{F}$ system in $\mathcal{O}(n \log(n))$. Instead of inverting the system, the FFT based method uses the eigenvalues and eigenvectors of $\boldsymbol{K2D}$. We first begin by decomposing the matrix $\boldsymbol{K2D}$ using it's eigenvalues and eigenvectors. A matrix A can be factored with a diagonal matrix of its eigenvalues, Λ, and a matrix of its eigenvectors S:

$$A = S\Lambda S^{-1}\ . \tag{7}$$

This allows us to rewrite (5) as

$$\mathbf{U} = \boldsymbol{S}\boldsymbol{\Lambda}^{-1}\boldsymbol{S}^{-1}\mathbf{F} \ . \tag{8}$$

This method hinges on the fact that we must already know the eigenvalues and eigenvectors of $\boldsymbol{K}\mathbf{2D}$ and that we already know the inverse for $\boldsymbol{S}^{-1}$, otherwise we still pay the same complexity cost of $\mathcal{O}(n^3)$. Also, for the algorithm to be classified as "fast" we need to be able to quickly multiply by $\boldsymbol{S}$ and $\boldsymbol{S}^{-1}$—sub $\mathcal{O}(n^2)$ operations. The matrix $\boldsymbol{S}$ is not sparse like $\boldsymbol{K}\mathbf{2D}$ and typically will require $\mathcal{O}(n^2)$ operations to perform a matrix multiplication. Fortunately, the eigenvalues and eigenvectors are known for the matrix K, and subsequently $\boldsymbol{K}\mathbf{2D}$. The eigenvalues for the K are $\lambda_k = 2 - 2\cos\frac{k\pi}{N+1}$ and the eigenvectors $S_{jk} = \sin\frac{jk\pi}{N+1}$. Where j is the jth component of the kth eigenvector.

The eigenvectors of the matrix K are both orthogonal and symmetric. This presents a couple advantages, an orthogonal matrix has the special property $S^\top = S^{-1}$, but since the matrix S is also symmetric $S = S^{-1}$. This means we no longer need to compute the inverse of S. The eigenvectors of K also follow the form of the Discrete Sine Transform (DST):

$$X_k = \sum_{n=0}^{N-1} x_n \sin\left[\frac{\pi}{N+1}(n+1)(k+1)\right] \quad k = 0, \ldots, N-1 \ . \tag{9}$$

The DST can be computed in $\mathcal{O}(n \log n)$ using the FFT. To do this we must pad the input matrix/vector. Let V be a $n \times n$ matrix we want to perform the Fast Sine Transform (FST) on, and $\boldsymbol{v}_i$ be a column of V. We first pad a zero before each vector and then $n+1$ zeros after each vector. So we end up with a $2(n+1) \times n$ matrix. Since the the sines comprise the complex components of the FFT we only need the complex coefficients. Therefore, after calling FFT on our padded input we only take the complex coefficients. The resulting call in MATLAB would be the following:

$$\text{imag}(\text{fft}(\begin{bmatrix} 0 & \ldots & \ldots & 0 \\ \boldsymbol{v}_1 & \ldots & \ldots & \boldsymbol{v}_n \\ \mathbf{0} & \ldots & \ldots & \mathbf{0} \end{bmatrix})) \ .$$

Since we are using the FST in place of the inverse FST we must normalize with the constant $\sqrt{\frac{2}{N+1}}$. This also happens to be the length of the eigenvectors of S. This is because the inverse FST is equal to $\frac{2}{N+1}$ times the FST. The normalizing constant is multiplied to all transforms to assures that our solution is not scaled by any additional multiplicative factor. The last thing we must do when returning the value produced by calling the FFT is select only the values of the vector V. We do not need the values produced in the indices of the padded zeros.

The eigenvalues and eigenvectors for $\boldsymbol{K}\mathbf{2D}$ follow similar form as the ones for K. The difference is that the eigenvalues and eigenvectors of $\boldsymbol{K}\mathbf{2D}$ are made up of products and sums of sines and cosines. The key point is that the N^2 eigenvectors of $\boldsymbol{K}\mathbf{2D}$ are separable (i.e. each of the eigenvectors denoted y_{kl} can be separated into a product of sines). The two sines correspond to a the second differences in the x and y directions. The eigenvectors of

$\boldsymbol{K2D}$ have a double index k, l and the (i, j) component of y_{kl} is defined as:

$$\sin \frac{ik\pi}{N+1} \sin \frac{jl\pi}{N+1}$$

This notation can be rather confusing. Essentially, we are computing the outer product between the kth eigenvector in the x direction and the lth eigenvector in the y direction. This gives the $N \times N$ matrix of with the corresponding (i, j) components. This matrix can be reshaped into a vector and the corresponding eigenvalue is:

$$\lambda_{kl} = \left(2 - 2\cos\frac{k\pi}{N+1}\right) + \left(2 - 2\cos\frac{l\pi}{N+1}\right)$$

Again the eigenvalue in 2 dimensions is the sum of the eigenvalue in the x and y direction. To further drive this home the following MATLAB code shows how to compute the eigenvalue and eigenvector for $k = 1$ and $l = 3$ for a mesh with $N = 5$ interior points.

Listing 1: Computing the Eigenstuff of $\boldsymbol{K2D}$ Code

```
N = 5;
e = ones(N, 1);

K = diag(-e(1:N-1), -1) + diag(2*e)...
    + diag(-e(1:N-1), 1);
K2D = kron(eye(N), K) + kron(K, eye(N));

k = 1; l = 3;
j = (1:5)'; i = (1:5)';

% Create Eigenvector
S = sin(k*i*pi/(N+1)) * sin(l*j*pi/(N+1))';
S = reshape(S, [N^2, 1]);

% Create Eigenvalue
lam = 2 - 2*cos(k*pi/(N+1))...
    + 2 - 2*cos(l*pi/(N+1));
```

The eigenstuff for the 2 dimensional matrix retains the same properties of the one dimensional case. The main difference is we need to call the FST twice since we have two sines in the eigenvectors now.

The problem in 2 dimensions is often re-framed as (K2D) (U2D) = (F2D). Where (U2D) and (F2D) are $N \times N$ matrices corresponding to the x, y coordinates in the mesh. This simply makes it easier to create the right hand side of the equation and perform our FST calls. The following MATLAB code shows how to compute the FST.

Listing 2: Computing the FST in MATLAB

```
function Y = fast_sine_transform(V)
    [m, n] = size(V);
    % Normalizing term makes eigenvectors
    % orthonormal so we only need one
    % transform for both ifft and fft.
    normalizing_term = sqrt(2/(n+1));
    % Need to pad the values of the matrix
    % to get the DST using FFT.
    V_ext = [zeros(1,n); V; zeros(m+1,n)];
    V_ext = imag(fft(V_ext));
    Y = normalizing_term.*V_ext(2:m+1, :);
end
```

In the code above we take the 2 to $m+1$ elements of the matrix returned by the imaginary coefficients of the the FFT because we no longer care about the values that resulted from the padding. The next code section shows how to solve for the matrix (U2D) in MATLAB.

Listing 3: Fast Poisson Solver MATLAB Code

```
% generate the eigenvalues
lambda = 2*(1-cos(pi*(1:n)/(n+1)));
L = lambda + lambda';

% U = S * inv(\Lambda) * S^{-1} * F
F_prime = fast_sine_transform(F);
F_prime = fast_sine_transform(F_prime');
U_prime = F_prime ./ L;
U_prime = fast_sine_transform(U_prime);
U = fast_sine_transform(U_prime');
```

The FST must be applied to the matrices twice becasue we need to account for both sine terms in the eigenvectors. We take the transpose of the matrix before passing it to the second FST to simulate the row-column dot products that are taken during matrix multiplication. Overall, the FFT based Fast Poisson Solver is a rather short method to implement in yields a vast improvement over taking the inverse of the large $N^2 \times N^2$ $\boldsymbol{K}\mathbf{2D}$ matrix. The time complexity of this method is $\mathcal{O}(n \log n)$. Where n is the number of interior points in the mesh.

3.2 Limitations of the FFT Based Method

Despite the great improvement in computational speed, the FFT method has some limitations that must be addressed. The first being that the method relies on the finite difference scheme for accuracy. Using the centered difference scheme with a 5-point-stencil yields a truncation term of $\mathcal{O}(h^2)$. It is possible to increase the order of the truncation term using the FFT method, there are versions of the method that use a 9-point-stencil (Houstis and Papatheodorou). The 9-point-stencil increases the order of the truncation term to $\mathcal{O}(h^4)$.

Another limitation to the FFT method is that it will not work on non-uniform, or non-square geometries (Clancy). There is scheme that has been presented by Shortley and Weller that can be applied to non-uniform, non-rectangular boundaries. This scheme suffers from a decrease in the truncation error to $\mathcal{O}(h)$. Despite this theoretical short coming, Shortley and Weller show that in practice the scheme performs decent. To solve this the system of equation created by this scheme one can use an iterative solver, such as Jacobi, Gauss–Seidel, or SOR. This would still be faster than inverting the matrix to solve the system. An alternative would be to use Integral Equations, which is the second method discussed in this paper.

3.3 Integral Equations

Integral Equation provide a completely separate formulation of partial differential equation problems than the typical finite difference methods. Instead of trying to approximate the derivative as an operator, problems are formulated in such way that we can take advantage of highly accurate quadrature rules. In order to do this we take advantage of well-known harmonic functions and their super-positions to build up a valid solution. Each contribution is added up in the form of an integral over the boundary of the domain. The problem can be formulated as follows:

$$\psi(\mathbf{x}) = \int_{\partial\Omega} k(\mathbf{x}, \mathbf{y})\sigma(\mathbf{y})ds(\mathbf{y}), \quad \mathbf{x} \in \Omega$$

Where $k(\mathbf{x}, \mathbf{y})$ is a harmonic function away from the origin. Our goal is to find $\sigma(\mathbf{y})$ such that our boundary conditions are met. One natural choice of $k(x, y)$ turns out to be the Green's function associated with the Laplace operator. It is defined as the solution to the following partial differential equation:

$$\nabla^2\phi = \delta(\mathbf{x})$$

Due to the properties of the Dirac delta function $\delta(\mathbf{x})$ it is very easy to see that the function $\phi(\mathbf{x})$ must satisfy Laplace's equation everywhere except at the origin. This function $\phi(\mathbf{x})$ is

well-known analytically and represents the potential of a point charge. In two dimensions it is defined as follows:

$$\phi(\mathbf{x}) = -\frac{1}{2\pi}\log|\mathbf{x}|$$

Due to $\phi(\mathbf{x})$ being harmonic, it is a good candidate function for our integral equation formulation. It turns out that discretizing the problem this way leads to a well defined solution, however, typically this is not the function of choice for integral equations. The reasoning behind this is that discretizing the integral operator leads to a matrix which has a condition number that scales with $O(h^{-}1)$. While this is better than the conditioning of the $\boldsymbol{K2D}$ matrix, a different choice of $k(x, y)$ results in a matrix which has a condition number that converges to a constant (Chapter 10: Integral equation formulations). This is the motivation behind the 'Double-Layer Potential' function which is defined as follows:

$$d(\mathbf{x}, \mathbf{y}) = \mathbf{n}(\mathbf{y}) \cdot \nabla_{\mathbf{y}}(\phi(\mathbf{x} - \mathbf{y}))$$

Here we are taking the dot product of the normal to the boundary with the gradient of $\phi(\mathbf{x} - \mathbf{y})$. For ease we will denote the integral with operator notation:

$$[D\sigma](\mathbf{x}) = \int_{\partial\Omega} d(\mathbf{x}, \mathbf{y})\sigma(y)ds(y)$$

In order to match our boundary conditions $g(\mathbf{x})$ we must ensure that the in the limit as a point goes to the boundary, we get the right expression:

$$g(\mathbf{x}) = \lim_{\mathbf{x}' \to \mathbf{x}, \mathbf{x}' \in \Omega} \int_{\partial\Omega} d(\mathbf{x}', \mathbf{y})\sigma(\mathbf{y})ds(\mathbf{y}), \quad \mathbf{x} \in \partial\Omega$$

It turns out that when we do this we get an extra factor of $\sigma(\mathbf{x})$ that will pop out of the integral. For an interior problem this looks like the following:

$$g(\mathbf{x}) = -\frac{1}{2}\sigma(\mathbf{x}) + \int_{\partial\Omega} d(\mathbf{x}, \mathbf{y})\sigma(\mathbf{y})ds(\mathbf{y}), \quad \mathbf{x} \in \partial\Omega$$

With operator notation this is simply:

$$\left(-\frac{1}{2}I + D\right)\sigma(\mathbf{x}) = g(\mathbf{x})$$

The above equation is what we will need to discretize and invert in order to find the density of potentials which will produce the solution inside of our domain. This type of integral equation is called a Fredholm Integral Equation of the Second Kind, and produces a numerically well behaved non-singular matrix. In order to discretize this operator, we will assume that the parametrization of the boundary, $\mathbf{G}(t)$ is known. Here $\mathbf{G}(t)$ will take parameter t and map some interval I to our boundary. We can now formulate the integral over this interval.

$$[D\sigma](\mathbf{x}) = \int_I d(\mathbf{x}, \mathbf{G}(t))\sigma(\mathbf{G}(t))|\mathbf{G}'(t)|dt, \quad t \in I$$

This is now in the form of a one dimensional integral, and any quadrature can be used to compute this. Typically choices are trapezoidal rule, or Gaussian panel based quadrature. Gaussian panel based quadrature is often nice because it provides a way to refine the integral in areas that are needed. In quadrature form, we approximate the integral with a finite sum:

$$[D\sigma](\mathbf{x}) \approx \sum_i w_i d(\mathbf{x}, \mathbf{G}(t_i))\sigma(\mathbf{G}(t_i))$$

where w_i and t_i are the weights and nodes of the quadrature being used. When we are actually evaluating this operator on the boundary, which we will have to do in order to find the densities $\boldsymbol{\sigma}$, we will have to worry about the singularity of $d(x, y)$. There is an analytic formula which will allow us to resolve this issue when $x = y$ on our discretization (Chapter 12: Discretization of integral equations).

$$\lim_{t' \to t} d(\mathbf{G}(t), \mathbf{G}(t')) = \frac{G_2'(t)G_1''(t) - G_1'(t)G_2''(t)}{4\pi|\mathbf{G}'(t)|^3}$$

Once we know how to discretize the D operator, we can form the matrix $(-\frac{1}{2}I + D)$ and apply the inverse to our boundary conditions. In order to solve the potential at any point, we just reform the operator D which depends on the points we chose to evaluate it at, and apply it to our now known densities.

One more thing we considered with this method is how to handle the Poisson Equation. We did this to so we could compare the results between the FFT method and integral equations. We follow the theory laid out the paper by Askham and Cerfon. We split up the problem into two parts, a homogeneous and particular:

$$\psi(\mathbf{x}) = \psi_h(\mathbf{x}) + \psi_p(\mathbf{x})$$

The particular solution solves the free-space Poisson problem:

$$\nabla^2 \psi_p(\mathbf{x}) = f(\mathbf{x}), \quad \mathbf{x} \in \Omega$$

where $f(\mathbf{x})$ describes the given density. If we have a solution to this problem we can solve the homogeneous problem with the techniques already outlined, as it is just a Laplace problem:

$$\nabla^2 \psi_h(\mathbf{x}) = 0, \quad \mathbf{x} \in \Omega$$
$$\psi_h(\mathbf{x}) = g(\mathbf{x}) - \psi_p(\mathbf{x}), \quad \mathbf{x} \in \partial\Omega$$

The solution is then just the sum of the two parts. There are many ways to solve the particular problem, including doing just FFT on it, however there exists an integral formulation which we will stick with. The formulation is as follows:

$$\psi_p(\mathbf{x}) = \int_\Omega \phi(\mathbf{x} - \mathbf{y}) f(\mathbf{y}) d\Omega$$

This integral is a 2D integral over the entire domain, and it includes a singularity as well. We seek to compute the principle integral over this region in order to find the free space Poisson solution. This is where the domain starts to play a role. Previously, we only considered the boundary, and thus actually computing the line integral was much easier. The paper does outline a way to extend the density $f(\mathbf{y})$ to a square which completely encompasses the original domain Ω. The problem is a little easier when the integration is done over a square. There are many ways to do this integral efficiently, including employing the fast multipole method, however we ran out of time to explore these methods, and instead used a built in 2D quadrature for our numerical examples.

3.4 Limitations of Integral Equations

Solving Laplace's equation with the Integral Equations approach is vastly superior to finite difference schemes. Both methods can exploit fast algorithms to compute the solution, however, Integral Equations does not limit the accuracy you can achieve before hitting floating point approximation errors. The structure of the discretized matrix can be exploited for fast inversions, which makes up for the fact that the matrix is dense, at least when considering computation. Complex boundaries are very easily and elegantly handled for

Laplace's equation, and while the problem is slightly more complex for Poisson's equation, integral formulations can still achieve a stunning level of accuracy when compared to finite difference methods. Overall, integral formulations are just a more elegant way of solving the Laplace and Poisson equations on a domain.

4 Results/Findings

We will now compare the results of a handful of examples with both methods. We will mostly be looking at the error between the two approximations generated with both methods.

4.1 Fast Poisson Solver

The fist example we will look at is Laplace's equation with a point charge located at $[0.95, 1.10]$. Figure 2 shows the plot of the point charge. We will be solving Laplace's equation over the boundary $\Omega = [0, 1] \times [0, 1]$. The approximation using the FFT based Fast Poisson Solver can be seen in figure 3a, and the error in the approximation in figure 3b. The FFT based method does a fair good job approximating the solution, the error at worse is 3×10^{-5}. The error is worse in the corner were the point charge is located. $N = 100$ interior points were used in the mesh.

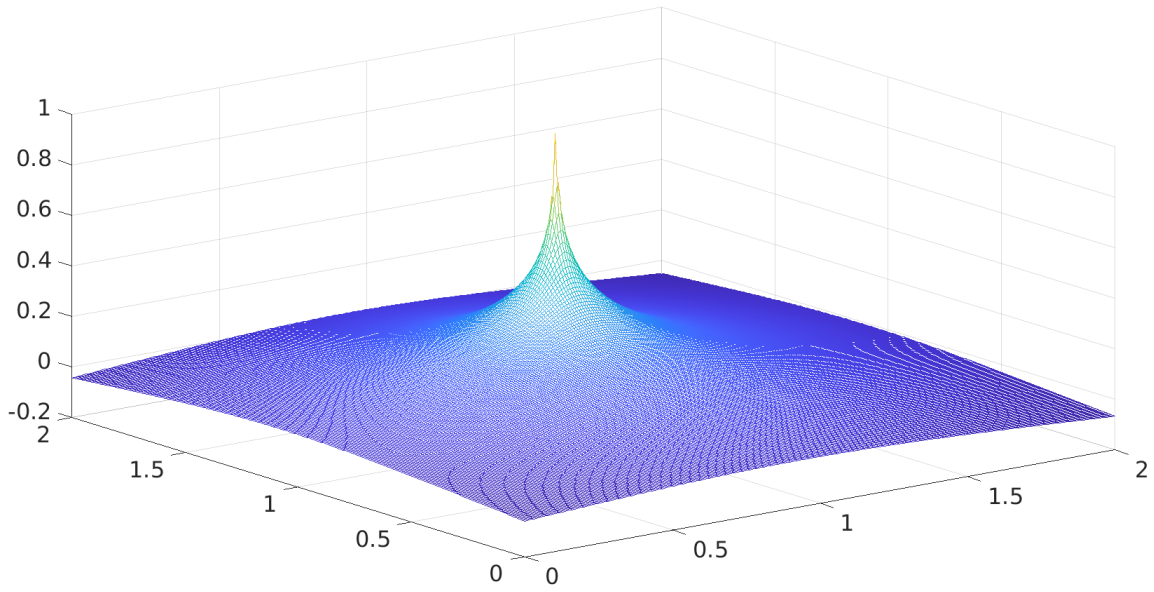
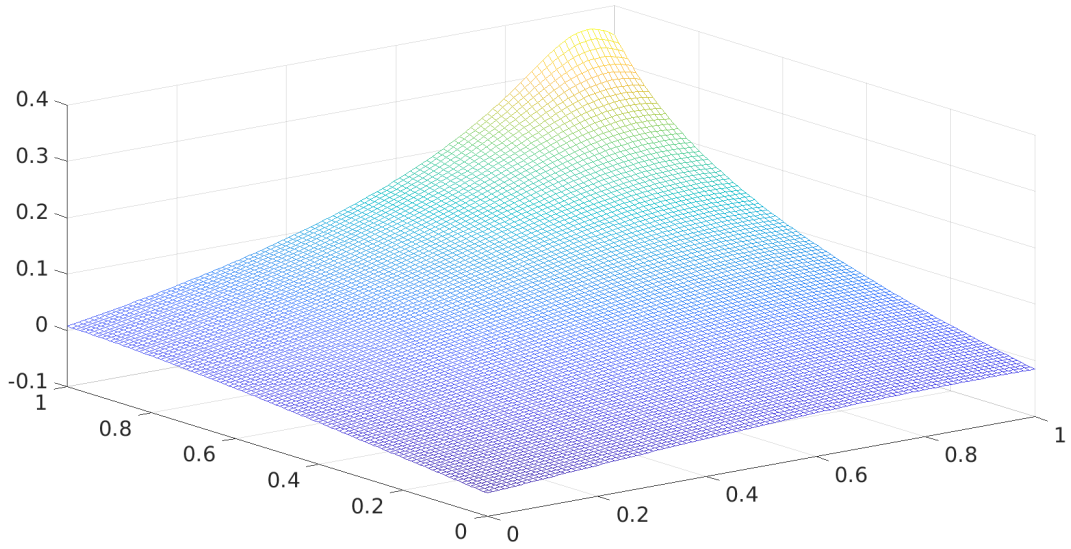
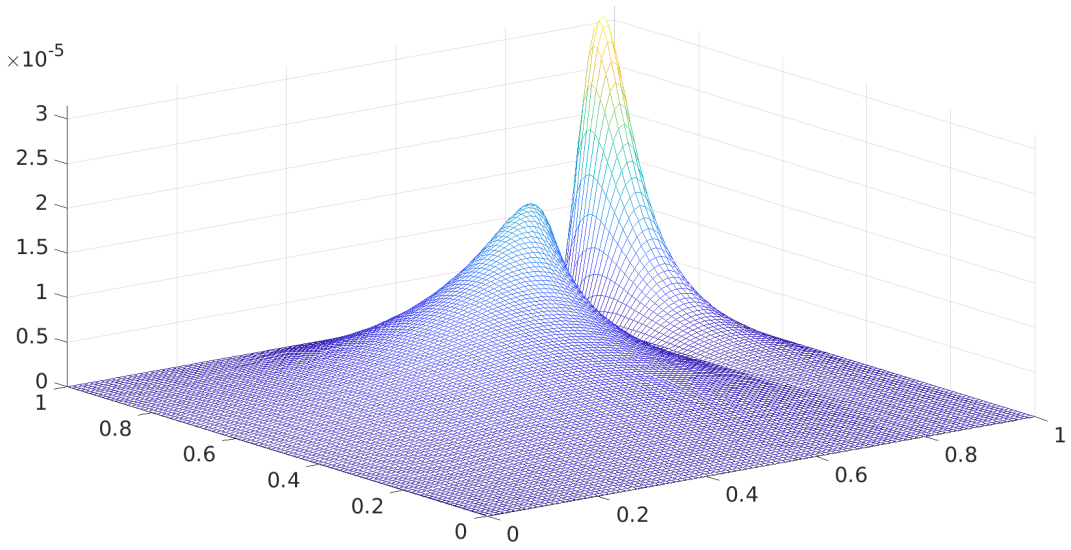


Figure 2: Point charge located at $[0.95, 1.10]$.



(a) Approximation of ψ.



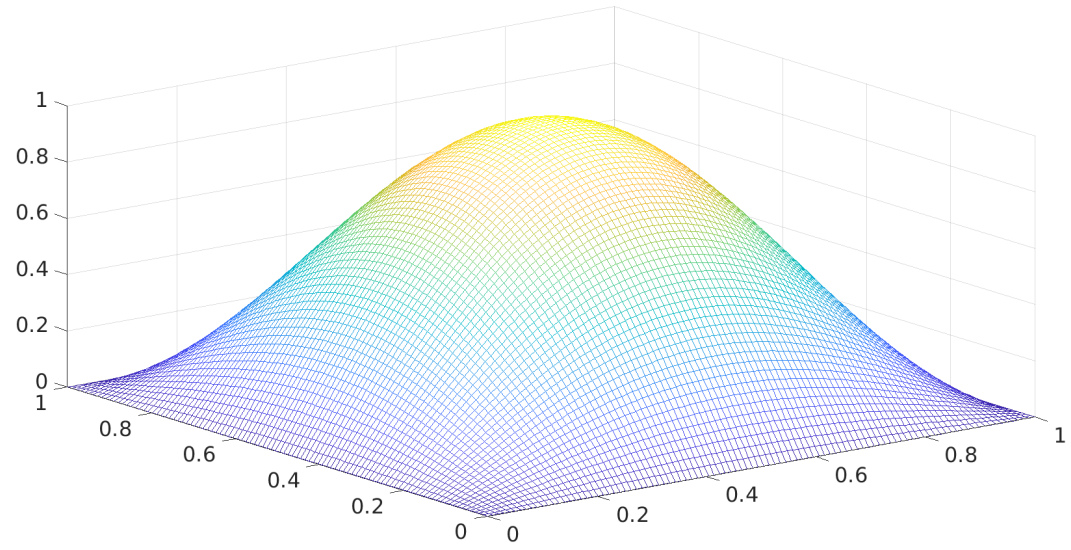
(b) Error in approximation of ψ.

Figure 3: $-\nabla^2\psi = 0 \quad \Omega = [0,1] \times [0,1]$ with $N = 100$ interior points. Approximation produced with the FFT based Fast Poisson Solver.

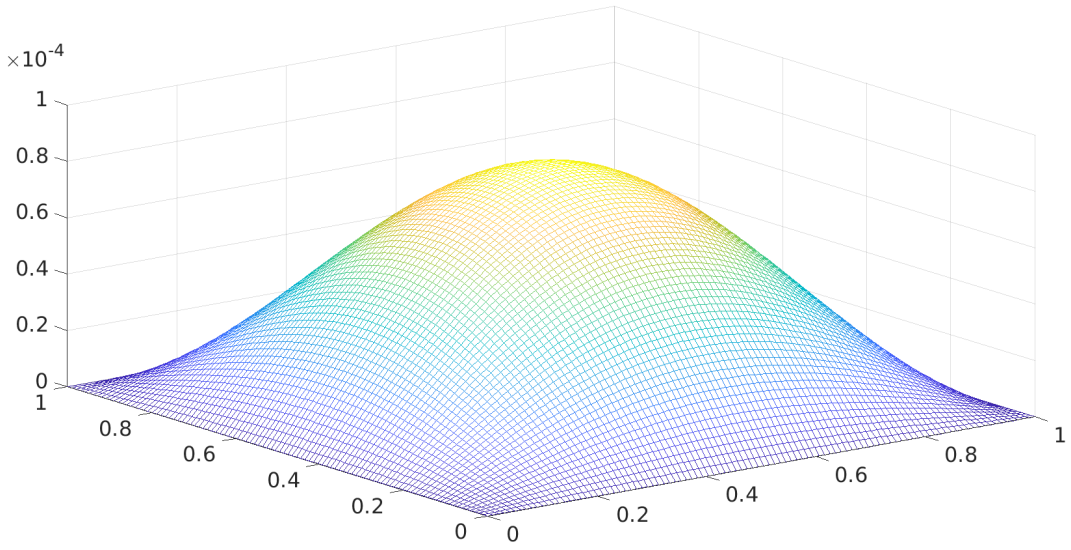
The remaining examples will focus on Poisson's equation. The first of which is a simple example with constant boundary conditions of zero. The following is the PDE we will be solving:

$$-\nabla^2\varphi = 2\pi^2 \sin(\pi\ x) \sin(\pi\ y) \quad \Omega = [0,1] \times [0,1]\ .$$

Where the analytical solution is $\varphi = \sin(\pi\ x)\sin(\pi\ y)$. Figure 4a shows the approximation of the solution with the FFT based method, and figure 4b shows the error in the approximation. $N = 100$ interior points were used to acquire the approximation. The error is not as great for this approximation as the last example. At worst we are only getting 4 digits of accuracy. The error this time is worse near the center of the boundary.



(a) Approximation of φ.



(b) Error in approximation of φ.

Figure 4: $-\nabla^2\varphi = 2\pi^2 \sin(\pi\ x)\sin(\pi\ y)$ inside $\Omega = [0,1]\times[0,1]$ with $N = 100$ interior points. Approximation produced with the FFT based Fast Poisson Solver.

The third example is a more complicated PDE with dynamic boundary conditions that change with x and y. The following is the PDE we will be solving:

$$-\nabla^2\varphi = -e^x + 18\pi^2 sin(6\pi\ y) \quad \text{inside} \quad \Omega = [0,1] \times [0,1]\ .$$

The boundary conditions are the following:

$$\varphi(x,0) = e^x, \quad \varphi(x,1) = e^x, \quad 0 < x < 1,$$

$$\varphi(0,y) = 1 + \frac{1}{2}sin(6\pi\ y), \quad \varphi(1,y) = e + \frac{1}{2}sin(6\pi\ y), \quad 0 < y < 1.$$

The analytical solution is $\varphi = e^x + \frac{1}{2}sin(6\pi\ y)$. Figure 5 shows the approximation of the PDE with the FFT based method. Figure 6 shows various errors with different mesh sizes. What is interesting about this example is that the error is greatest at the peaks and valleys of the sine waves; it makes this ripple effect. Figure 6 also serves as a demonstration of how the truncation error from the finite difference scheme propagates through this method. As we increase the size of the mesh we do not see significant gains in digits of accuracy. With only $N = 50$ interior points we are able to get the same amount of digits as double that—$N = 100$ interior points. Even with $N = 150$ interior points we only gain an extra digit of accuracy.

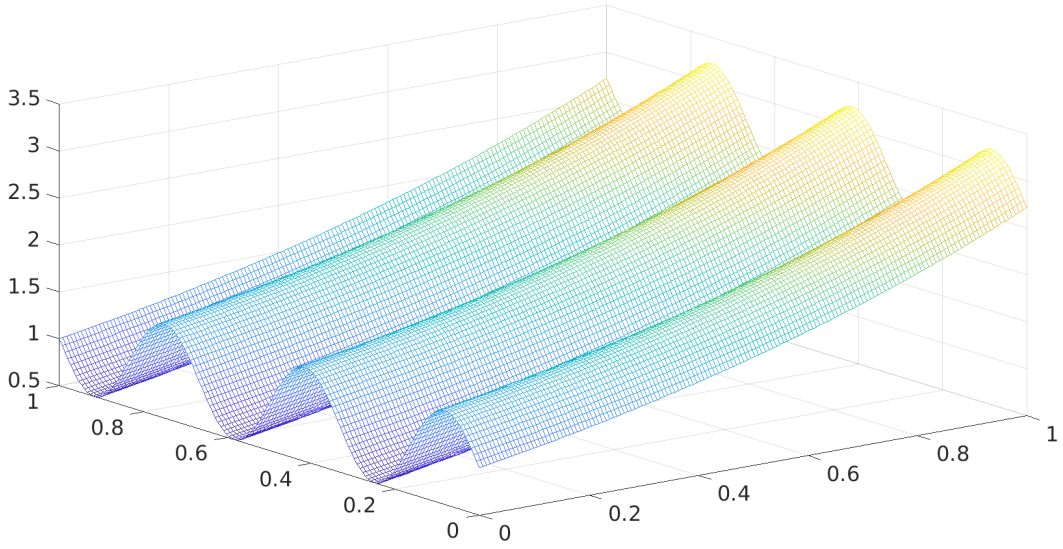
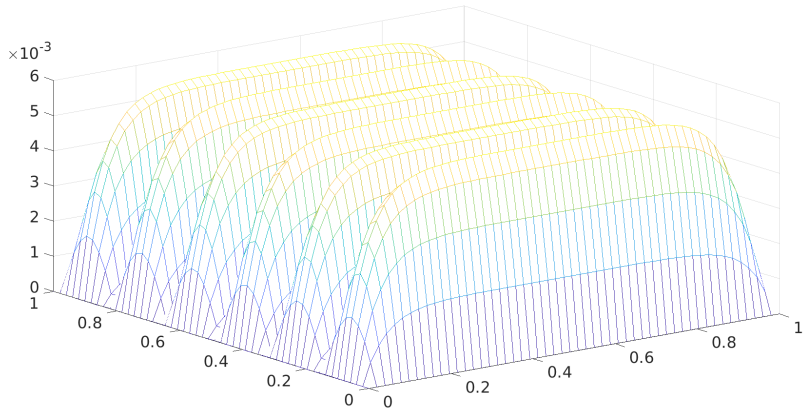
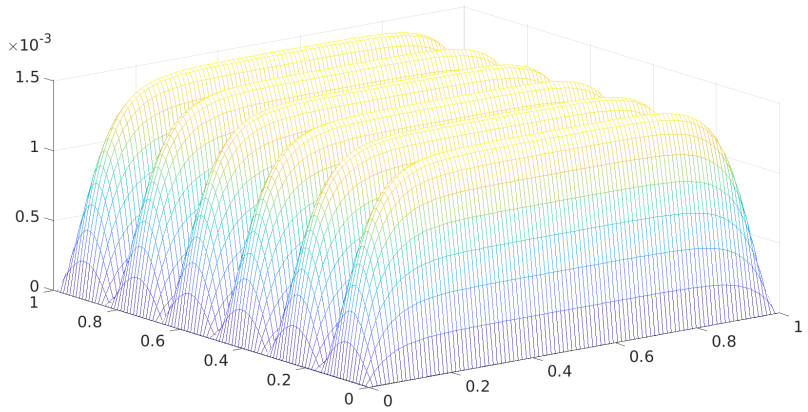


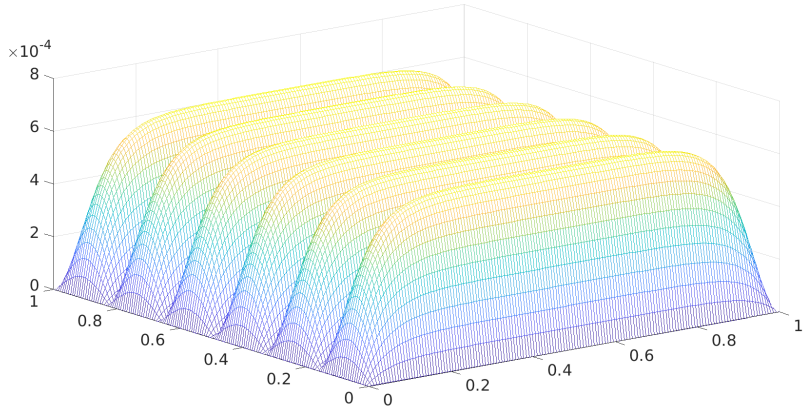
Figure 5: Approximation of $-\nabla^2\varphi = -e^x + 18\pi^2 sin(6\pi\ y)$ inside $\Omega = [0,1] \times [0,1]$ with $N = 150$ interior points. Approximation produced with the FFT based Fast Poisson Solver.



(a) $N = 50$

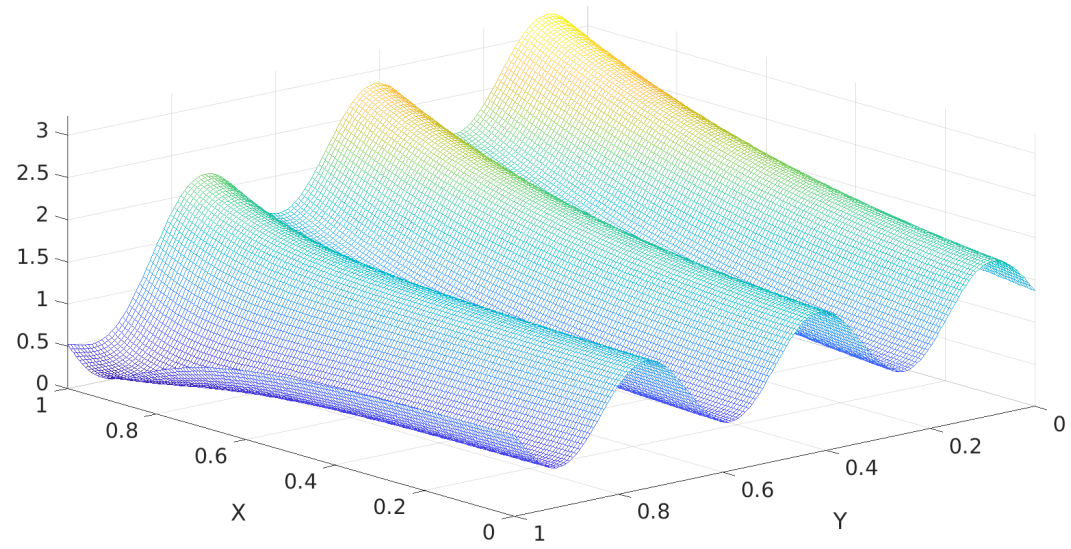


(b) $N = 100$

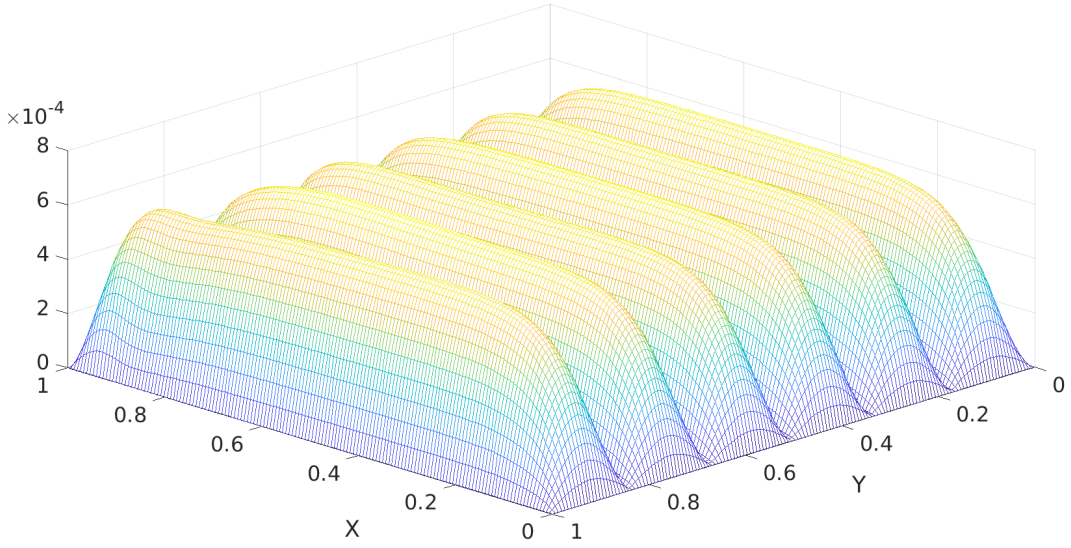


(c) $N = 150$

Figure 6: Error in approximation of $-\nabla^2\varphi = -e^x + 18\pi^2 sin(6\pi\ y)$ inside $\Omega = [0,1] \times [0,1]$ with $N = 50, 100, 150$ interior points. Approximation produced with the FFT based Fast Poisson Solver.



(a) Approximation of φ



(b) Error in approximation of φ.

Figure 7: Error in approximation of $\varphi = e^x + \frac{1}{2}sin(6\pi\ y) + \log((x-0.95)^2 + (y-1.10)^2)$ inside $\Omega = [0,1] \times [0,1]$ with $N = 150$ interior points. Approximation produced with the FFT based Fast Poisson Solver.

The final example we will look at is the same PDE as the previous example but we will now

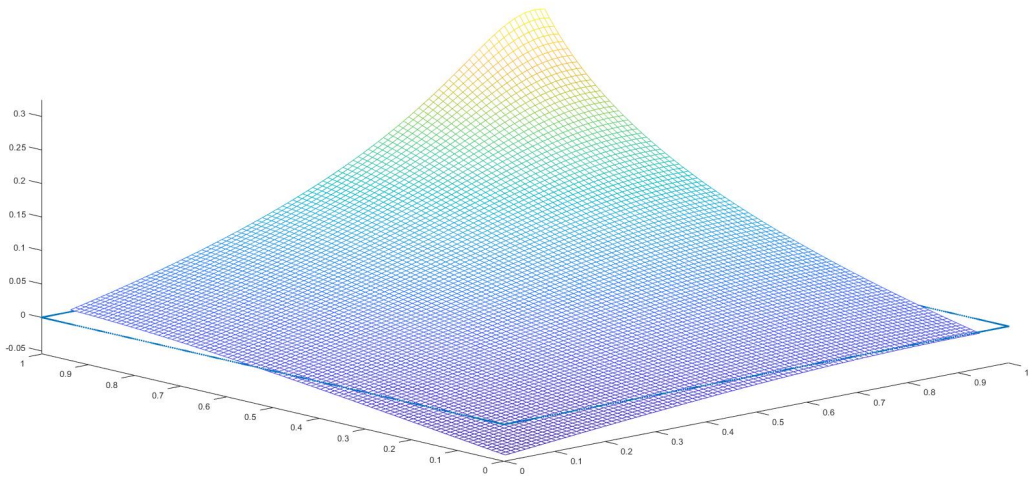
add a point charge at $[0.95, 1.10]$. The full solution to the PDE is as follows:

$$\varphi = e^x + \frac{1}{2} sin(6\pi\ y) + \log((x - 0.95)^2 + (y - 1.10)^2)\ .$$

Again we will solve the PDE on the domain $\Omega = [0, 1] \times [0, 1]$. The FFT method is able to achieve 4 digits of accuracy again with $N = 150$ interior points. We again observe that the error is largest at the peaks and valleys of the sine waves.

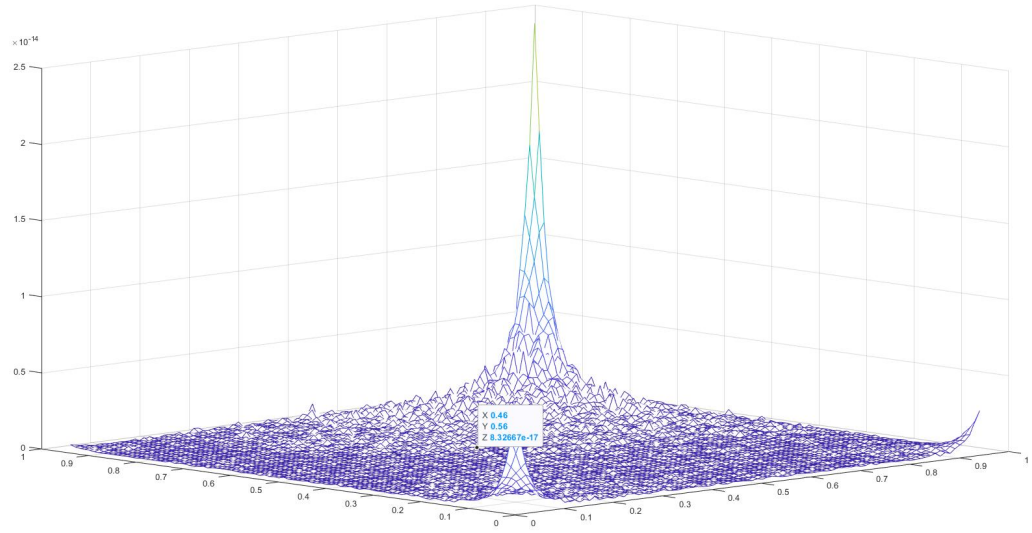
4.2 Integral Equations

Here we will examine three different examples. The first two will compare the accuracy of some problems we already solved with the FFT based Fast Poisson Problem. The first example is the simple point charge located at [.95, 1.10]. Revisiting this with integral equations shows us the power of this method. While FFT was able to achieve error of order 10^{-5}, integral equations achieve an accuracy of order 10^{-17} on average. We can see that the error builds up around the corner where the point is located. We suspect that the discretization can be further refined in order to cut down on error here as well.



(a) Approximation of φ

Figure 8: Error in approximation of $\varphi = \log((x - .95) + (y - 1.15))$ inside $\Omega = [0, 1] \times [0, 1]$ with integral equations formulation



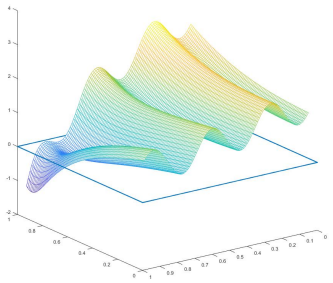
(b) Error in approximation of φ.

Figure 8: Error in approximation of $\varphi = \log((x - .95) + (y - 1.15))$ inside $\Omega = [0, 1] \times [0, 1]$ with integral equations formulation

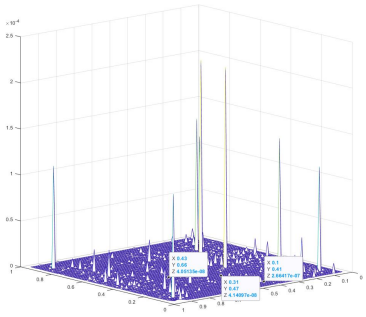
The next example has the analytic solution of:

$$\varphi = e^x + \frac{1}{2} sin(6\pi\ y) + \log((x - 0.95)^2 + (y - 1.10)^2)\ .$$

This is a Poisson problem on the square domain $\Omega = [0, 1] \times [0, 1]$. The error of the integral equations formulation is on average order 10^{-8}. We used a fairly naive method to solve for the particular solution, so we suspect that is why we see such an increase in the order of error when compared to the Laplace problem. There exist certain points which have spikes of error, and we suspect this is also due to the built in MATLAB quadrature. Further exploration of this project would have lead to more sophisticated methods which could take advantage of the fast multipole method.



(a) Approximation of φ



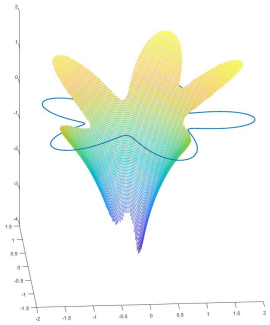
(b) Error in approximation of φ.

Figure 9: Error in approximation of $\varphi = e^x + \frac{1}{2}sin(6\pi\ y) + \log((x-0.95)^2 + (y-1.10)^2)$ inside $\Omega = [0,1] \times [0,1]$ with integral equations formulation and naive quadrature

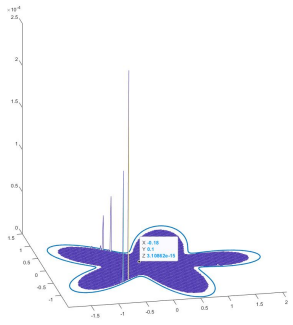
Our final example is another Laplace problem, however this time on a starfish like domain. This problem would be quite difficult to set up with a finite difference method and such solution would be orders of magnitude worse than the order of 10^-15 that integral equations achieve. The analytic solution to this problem is given by:

$$\varphi = \log((x+.3)^2 + (y+.75)^2)$$

This is a source right in between the two 'hands' of the starfish. We suspect that the error spikes at the boundary are cause either by not enough refinement or some other strange boundary effects. The solution on the interior, however, is extremely accurate.



(a) Approximation of φ



(b) Error in approximation of φ.

Figure 10: Error in approximation of $\varphi = \log((x+.3)^2 + (y+.75)^2)$ inside starfish geometry with integral equations formulation

5 Conclusion

Methods involving finite difference schemes are very easy to implement and depending on the domain can be solved quite quickly using FFT. These methods can be used in applications which do not require high accuracy solutions. However, from our exploration, integral equations perform better as a whole. Accuracy can be achieved to machine precision without running into floating point errors due to poorly conditioned matrices. Integral equations can be solved very efficiently despite the dense structure of the discretized system, meaning computational complexity matches the FST algorithm. Integrals equations simply outperform finite difference schemes in 2D.

6 Acknowledgements

Much of the work done for the FFT based Fast Poisson Solver is based off of the lectures for Strang's "Mathematical Methods for Engineers II" course at MIT OpenCourseWare. Most of the research accomplished for this method was due in part to his online teaching resources.

7 Code

Code for the project can be found at this repository.

Works Cited

Askham, T. and A.J. Cerfon. "An adaptive fast multipole accelerated Poisson solver for complex geometries". *Journal of Computational Physics* 344 (2017): 1–22. Web.

"Chapter 10: Integral equation formulations". *Fast Direct Solvers for Elliptic PDEs*. 105–114. eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9781611976045.ch10. Web.

"Chapter 12: Discretization of integral equations". *Fast Direct Solvers for Elliptic PDEs*. 121–135. eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9781611976045.ch12. Web.

Clancy, Richard J. "Numerical Solutions to Poisson's Equation Over Non-Uniform Discretizations with Associated Fast Solvers" (2017). Print.

Houstis, Elias N. and Theodore S. Papatheodorou. "High-order fast elliptic equation solvers". *ACM Transactions on Mathematical Software (TOMS)* 5.4 (1979): 431–441. Print.

Shortley, George H and Royal Weller. "The numerical solution of Laplace's equation". *Journal of Applied Physics* 9.5 (1938): 334–348. Print.

Strang, Gilbert. "18.086 Mathematical Methods for Engineers II, Spring 2006" (2006). Web.