

Fast Poisson Solver and Integral Equations for 2D Elliptical PDEs

Cole Sturza Spas Angelov

Zoom University

April 28, 2021

What problem are we solving?

We are going to solve Dirichlet boundary value problems for elliptic PDEs in two dimensions.

For a region $\Omega \in \mathbb{R}^2$

Laplace's Equation:

$$-\nabla^2 \psi = 0 \quad \text{or} \quad -\frac{\partial^2 \psi}{\partial x^2} - \frac{\partial^2 \psi}{\partial y^2} = 0 \quad \text{inside } \Omega$$

$$\psi(x, y) = g(x, y) \quad \text{on } \partial\Omega$$

Poisson's Equation:

$$-\nabla^2 \varphi = f(x, y) \quad \text{inside } \Omega$$

$$\varphi(x, y) = g(x, y) \quad \text{on } \partial\Omega$$

Finite Difference Methods in Two Dimensions

The second order centered difference approximations will be used for both partials in the x and y direction¹.

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

Where h is the mesh width.

¹We will be formulating the problem with Poisson's equation, but the same technique can be easily applied to Laplace's equation.

Finite Difference Methods in Two Dimensions

Using our centered difference approximations, Poisson's equation becomes:

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

Adding in our boundary conditions to our points on the edge we get the following:

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

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

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

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

Finite Difference Methods in Two Dimensions

The finite difference scheme is represented by the system:

$$\mathbf{K2D} \mathbf{U} = \mathbf{F} .$$

Where $\mathbf{K2D}$ is an $N^2 \times N^2$ matrix, and \mathbf{U} and \mathbf{F} are both $N^2 \times 1$ vectors.

$$\mathbf{K2D} = \begin{bmatrix} 4 & -1 & 0 & 0 & 0 & \dots & 0 \\ -1 & 4 & -1 & 0 & 0 & \dots & 0 \\ 0 & -1 & 4 & -1 & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & -1 & 4 & -1 & 0 \\ 0 & \dots & \dots & 0 & -1 & 4 & -1 \\ 0 & \dots & \dots & \dots & 0 & -1 & 4 \end{bmatrix}$$

Finite Difference Methods in Two Dimensions

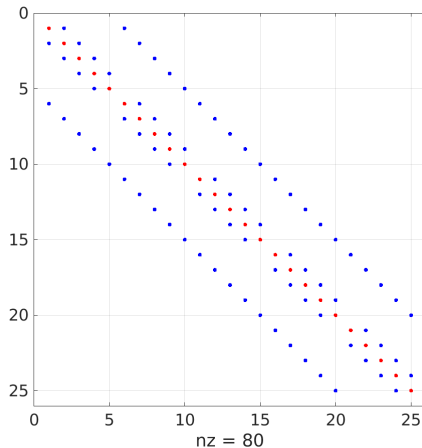


Figure 1: $K2D$, $N = 25$.

The $K2D$ matrix is block tridiagonal with tridiagonal blocks.

Finite Difference Methods in Two Dimensions

So how do we solve $\mathbf{K2D} \mathbf{U} = \mathbf{F}$?

The simplest way would be to invert $\mathbf{K2D}$.

$$\mathbf{U} = \mathbf{K2D}^{-1} \mathbf{F}$$

This would require $\mathcal{O}(n^3)$ operations.

Our matrix is $N^2 \times N^2$, which can be very costly to invert.

Can we do better?

Using the Eigenvalues and Eigenvectors of ***K2D***

Another option is to use the the Eigenvalues and Eigenvectors of ***K2D***.

Recall that a matrix A can be factored using the eigenvector matrix S and the diagonal eigenvalue matrix Λ :

$$K = S\Lambda S^{-1}, \quad K^{-1} = S\Lambda^{-1}S^{-1}.$$

Using this factorization our solution becomes:

$$\mathbf{U} = \mathbf{K2D}^{-1}\mathbf{F} \implies \mathbf{U} = \mathbf{S}\mathbf{\Lambda}^{-1}\mathbf{S}^{-1}\mathbf{F}$$

For this to work efficiently we need to know all the eigenstuff of ***K2D***. Also, we will need to know the inverse of \mathbf{S} .

Using the Eigenvalues and Eigenvectors of $K2D$

All of the eigenvalues and eigenvectors of $K2D$ are known.

The original system is reframed to $(K2D)(U2D) = (F2D)$ since $K2D$ no longer needs to be built.

The eigenvectors y_{kl} of $(K2D)$:

The (i, j) component of y_{kl} is $\sin \frac{ik\pi}{N+1} \sin \frac{jl\pi}{N+1}$

$(K2D)y_{kl} = \lambda_{kl}y_{kl}$:

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

The key distinction now is that Λ is a dense matrix.

Using the Eigenvalues and Eigenvectors of K_2D

The “sine eigenvectors” of S give the Discrete Sine Transform (DST):

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

We can compute the matrix multiplication of S using the DST.

The DST can be computed in $\mathcal{O}(n \log(n))$ with the FFT.

Using the Eigenvalues and Eigenvectors of $\mathbf{K}^2\mathbf{D}$

The eigenvectors of \mathbf{S} are both orthogonal and symmetric.

Using these two properties allows use to use the Fast Sine Transform (FST) for both \mathbf{S} and \mathbf{S}^{-1} ².

We need to apply a normalizing term

$$\sqrt{\frac{2}{N+1}}$$

on each FST call because we are using the FST in place of the inverse FST.

²Recall $Q^\top = Q^{-1}$ for orthogonal matrices.

Zero Padding

We need to zero pad the input to the FFT to get the FST.

Let V be a $n \times n$ matrix we want to perform the FST on, and v_i be a column of V .

We first pad a zero before each vector. Then $n + 1$ zeros after each vector. So we end up with a $2(n + 1) \times n$ matrix.

$$\text{imag}(\text{fft}(\begin{bmatrix} 0 & \dots & \dots & 0 \\ v_1 & \dots & \dots & v_n \\ \mathbf{0} & \dots & \dots & \mathbf{0} \end{bmatrix}))$$

Time Complexity

The overall time complexity for this algorithm is $\mathcal{O}(n \log(n))$.
Where n is the number of interior points.

MATLAB Code

Listing 1: Fast Poisson Solver

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

5 % 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);
10 U = fast_sine_transform(U_prime');
```

Point Charge Example (Laplace's Equation)

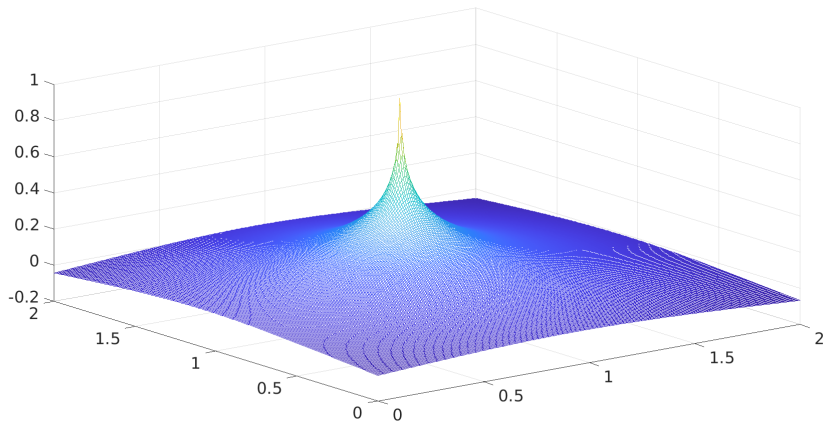
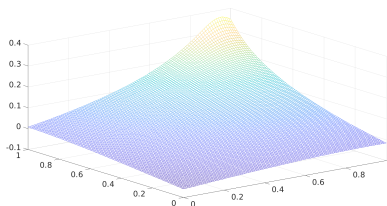
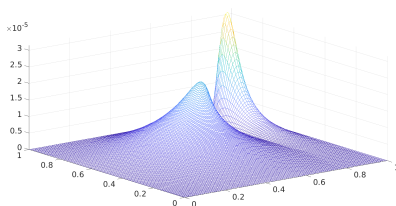


Figure 2: Point charge located at $[0.95, 1.10]$.

Point Charge Example (Laplace's Equation)



(a) Approximation of ψ .



(b) Error in approximation of ψ .

Figure 3: $-\nabla^2 \psi = 0 \quad \Omega = [0, 1] \times [0, 1]$
 $N = 100$

Dynamic Boundary Conditions (Possion's Equation)

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

The analytical solution is $\varphi = e^x + \frac{1}{2} \sin(6\pi y)$

Dynamic Boundary Conditions (Possion's Equation)

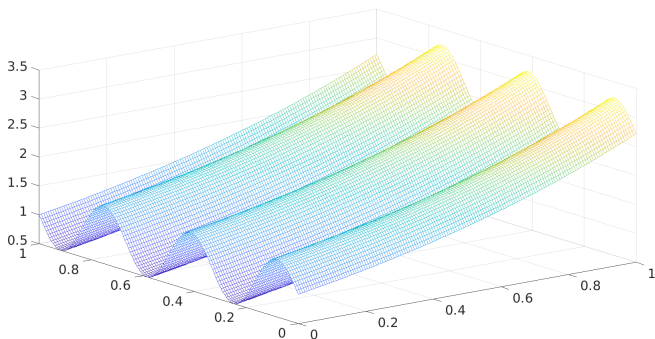
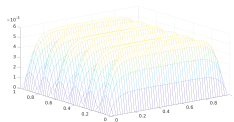
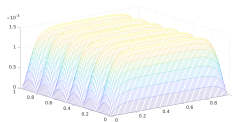


Figure 4: Approximation of φ with $N=150$.

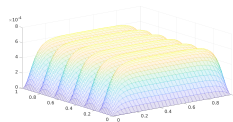
Dynamic Boundary Conditions (Poisson's Equation)



(a) $N = 50$



(b) $N = 100$



(c) $N = 150$

Figure 5: Error in approximation of φ .

Dynamic Boundary Conditions and Point Charge (Poisson's Equation)

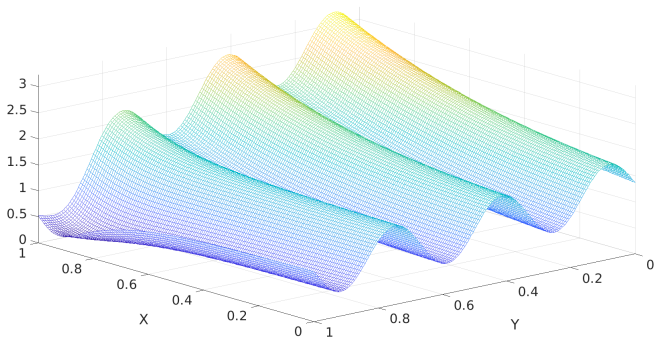


Figure 6: Approximation of

$$\varphi = e^x + \frac{1}{2}\sin(6\pi y) + \log((x - 0.95)^2 + (y - 1.10)^2) \text{ with } N=150.$$

Dynamic Boundary Conditions and Point Charge (Poisson's Equation)

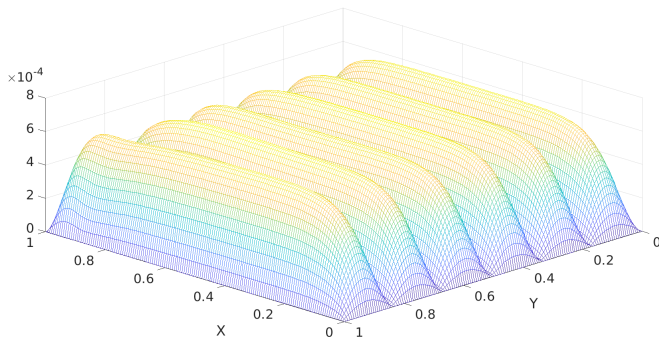


Figure 7: Error in approximation of φ .

Limitations of the FFT based solver

The FFT method will not work on non-uniform grids or for irregular boundaries [4].

There are schemes that can be used for non-uniform grids [6], but they suffer from a lose in order of accuracy.

An alternative, is to use Integral Equations which allow you to solve many types of geometries.

Laplace Problem With Integral Equations

Laplace's Equation:

$$\begin{aligned}\nabla^2 \psi &= 0 \quad \text{inside } \Omega \\ \psi(\mathbf{x}) &= g(\mathbf{x}) \quad \text{for } x \in \partial\Omega\end{aligned}$$

We are going to seek a solution in the following form:

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

Where $\phi(\mathbf{x})$ is a "special" function.

The Fundamental Solution of The Laplace Operator

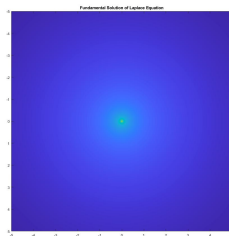
$\phi(\mathbf{x})$ is defined to be the fundamental solution of the Laplace Equation, which solves the following:

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

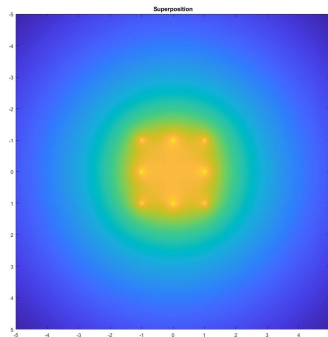
Where $\delta(\mathbf{x})$ is the Dirac-Delta function.
We can solve for $\phi(\mathbf{x})$ analytically:

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

$\phi(\mathbf{x})$ is harmonic away from the origin.



How to Exploit the Harmonic Nature of $\phi(\mathbf{x})$



Superposition of many of these point charges can create complicated looking potentials. If we find some density of these point charges to match our boundary conditions, then we have solved the problem.

What We Gain From Single-Layer Formulation

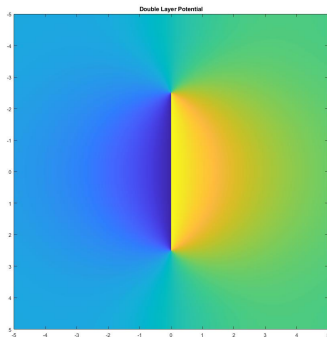
- Reduction by one dimension
- Easily handle interior or exterior problems
- More complex geometries are super easy.
- Condition number scales with $O(h^{-1})$ (However, we can do better!)

Double-Layer Formulation

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

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

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



Double-Layer Formulation Continued

Formulation:

$$\begin{aligned} g(\mathbf{x}) &= \lim_{\mathbf{x}' \rightarrow \mathbf{x}, \mathbf{x}' \in \Omega} \int_{\partial\Omega} d(\mathbf{x}', \mathbf{y}) \sigma(\mathbf{y}) ds(\mathbf{y}) \\ &= -\frac{1}{2} \sigma(\mathbf{x}) + \int_{\partial\Omega} d(\mathbf{x}, \mathbf{y}) \sigma(\mathbf{y}) ds(\mathbf{y}), \quad \mathbf{x} \in \partial\Omega \end{aligned}$$

So:

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

How we find σ :

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

To solve for ψ :

$$\psi = D\sigma$$

Parameterization of the Boundary

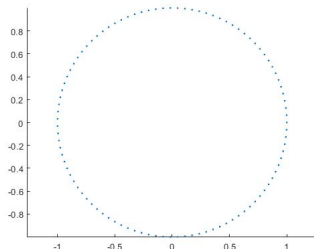
$$[D\sigma](\mathbf{x}) = \int_{\partial\Omega} d(\mathbf{x}, \mathbf{y}) \sigma(\mathbf{y}) ds(\mathbf{y})$$

How do we represent operator D as a matrix?

Let $\mathbf{G}(t)$ be a function that takes parameter t and maps it to the boundary for $t \in [0, T]$

Example:

$$\mathbf{G}(t) = \begin{pmatrix} \cos(t) \\ \sin(t) \end{pmatrix}, \quad t \in [0, 2\pi]$$



Discretization

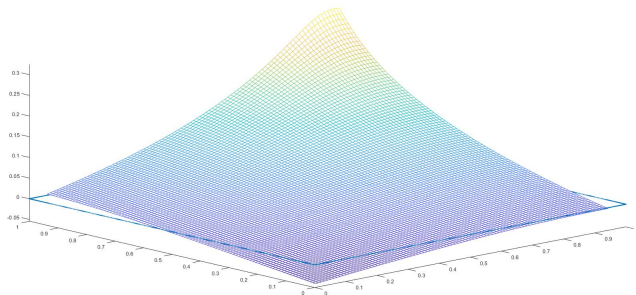
$$[D\sigma](\mathbf{x}) = \int_0^T d(\mathbf{x}, \mathbf{G}(t))\sigma(\mathbf{G}(t))|\mathbf{G}'(t)|dt$$

We can use whatever quadrature we want. Typical choices are trapezoidal rule, or Gaussian panel-based quadrature.

$$[D\sigma](\mathbf{x}) \approx \sum_{i=1}^N d(\mathbf{x}, \mathbf{G}(t_i))\sigma(\mathbf{G}(t_i))w_i$$

Where t_i and w_i will depend on the quadrature used.

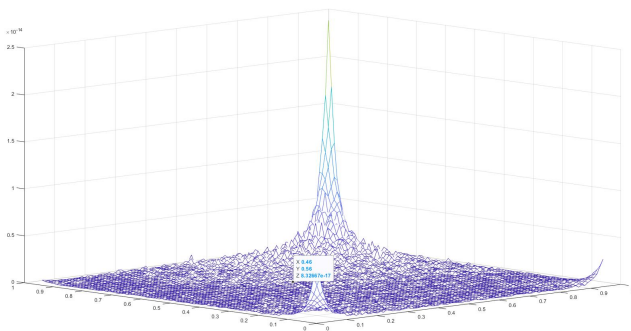
Laplace's Equation Point Charge



(a) Approximation of φ .

Figure 8: $\varphi = -\frac{1}{2\pi} \log(|\mathbf{x} - \mathbf{x}_0|)$ $\Omega = [0, 1] \times [0, 1]$

Laplace's Equation Point Charge Error



(a) Error in approximation of φ .

Figure 9: $\varphi = -\frac{1}{2\pi} \log(|\mathbf{x} - \mathbf{x}_0|)$ $\Omega = [0, 1] \times [0, 1]$

Quick Detour to Poisson Equation

Poisson Equation:

$$\begin{aligned}\nabla^2\psi &= f(\mathbf{x}) \quad \text{inside } \Omega \\ \psi(\mathbf{x}) &= g(\mathbf{x}) \quad \text{for } x \in \partial\Omega\end{aligned}$$

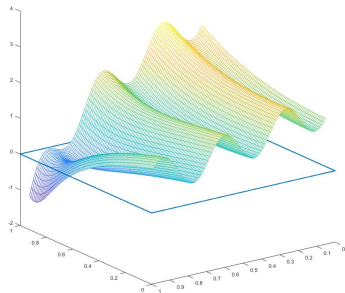
Solution can be formulated as sum homogeneous and particular parts:

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

Particular solution solves the free space Poisson equation, then homogeneous solution can be found by solving Laplace's Equation with altered boundary conditions:

$$\begin{aligned}\nabla^2\psi_h &= 0 \quad \text{inside } \Omega \\ \psi_p(\mathbf{x}) &= g(\mathbf{x}) - \psi_p(\mathbf{x}) \quad \text{for } x \in \partial\Omega\end{aligned}$$

Poisson Equation with Non-Zero Boundary Conditions

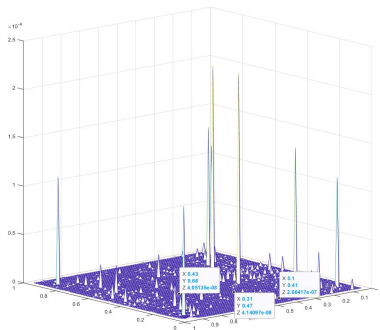


(a) Approximation of φ .

Figure 10:

$$\varphi = e^x - \frac{\sin(6\pi y)}{2} + \log((x - x_0)^2 + (y - y_0)^2) \quad \Omega = [0, 1] \times [0, 1]$$

Poisson Equation with Non-Zero Boundary Conditions Error

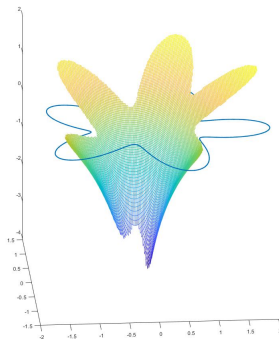


(a) Error in approximation of φ .

Figure 11:

$$\varphi = e^x - \frac{\sin(6\pi y)}{2} + \log((x - x_0)^2 + (y - y_0)^2) \quad \Omega = [0, 1] \times [0, 1]$$

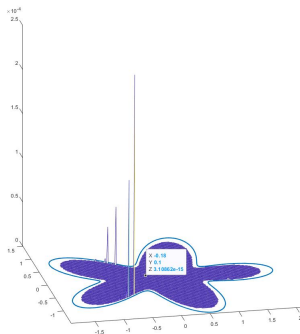
Laplace's Equation on a Starfish



(a) Approximation of φ .

Figure 12: $\varphi = \log((x - x_0)^2 + (y - y_0)^2)$

Laplace's Equation on a Starfish Error



(a) Error in approximation of φ .








Figure 13: $\varphi = \log((x - x_0)^2 + (y - y_0)^2)$

Git Repository Link



<https://github.com/colesturza/APPM4660-Final-Project>

Bibliography

-  *Chapter 10: Integral equation formulations*, pp. 105–114.
-  *Chapter 12: Discretization of integral equations*, pp. 121–135.
-  T. Askham and A. Cerfon, *An adaptive fast multipole accelerated poisson solver for complex geometries*, Journal of Computational Physics, 344 (2017), pp. 1–22.
-  R. J. Clancy, *Numerical solutions to poisson's equation over non-uniform discretizations with associated fast solvers*, (2017).
-  E. N. Houstis and T. S. Papatheodorou, *High-order fast elliptic equation solvers*, ACM Transactions on Mathematical Software (TOMS), 5 (1979), pp. 431–441.
-  G. H. Shortley and R. Weller, *The numerical solution of laplace's equation*, Journal of Applied Physics, 9 (1938), pp. 334–348.
-  G. Strang, *18 086 mathematical methods for engineers ii*

Listing 2: Computing the Eigenstuff of **K2D** Code

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

K = diag(-e(1:N-1), -1) + diag(2*e)...
5   + 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)';
10

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

15 % Create Eigenvalue
lam = 2 - 2*cos(k*pi/(N+1))...
      + 2 - 2*cos(l*pi/(N+1));
```


Listing 3: Fast Sine Transform Code

```
function Y = fast_sine_transform(V)
    [m, n] = size(V);
    % Normalizing term makes eigenvectors
    % orthonormal so we only need one
5   % 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)];
10   V_ext = imag(fft(V_ext));
    Y = normalizing_term.*V_ext(2:m+1, :);
end
```