# **AMATH 585** Assignment 5

Tyler Chen

### Problem 1

Write a code to solve Poisson's equation on the unit square with Dirichlet boundary conditions:

$$u_{xx} + u_{yy} = f(x, y), \quad 0 < x, y < 1$$

$$u(x, 0) = u(x, 1) = u(0, y) = u(1, y) = 1.$$

Take $f(x, y) = x^2 + y^2$, and demonstrate numerically that your code achieves second order accuracy. [Note: If you do not know an analytic solution to a problem, one way to check the code is to solve the problem on a fine grid and pretend that the result is the exact solution, then solve on coarser grids and compare your answers to the fine grid solution. However, you must be sure to compare solution values corresponding to the same points in the domain.]

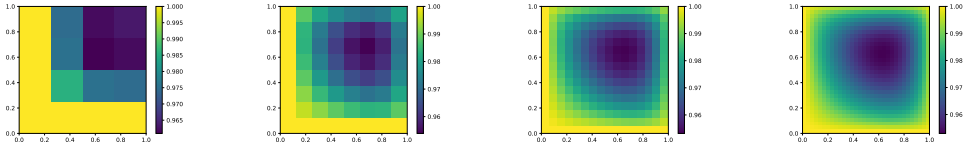
### Solution

We implement our five point stencil for solving the Poisson equation in Python.

```
def five_point_poisson(m,img_q=False):
    # define RHS function
    def f(x,y): return x**2 + y**2

    # define boundary conditions
    bd = np.ones((m+2,m+2))
    bd[1:m+1,1:m+1] = np.zeros((m,m)) #must be zero on interior for below

    x = np.linspace(0,1,m+2)
    h = 1/(m+1)

    # construct A
    T = -4*sparse.eye(m,m,k=0) + sparse.eye(m,m,k=1) + sparse.eye(m,m,k=-1)
    I = sparse.eye(m)
    II = sparse.eye(m,k=1)+sparse.eye(m,k=-1)
    A = sparse.kron(I,T)+sparse.kron(II,I)
    A /= h**2

    # build RHS
    F = f(x[1:m+1,None],x[None,1:m+1])

    # subtract stencil evaluated at known boundary points
    F -= (bd[0:m,1:m+1] + bd[2:m+2,1:m+1] + bd[1:m+1,0:m] + bd[1:m+1,2:m+2])
        /h**2

    # evaluate f along grid, account for known boundary points, and flatten
    F_ = np.reshape(F,(-1,))

    # solve system
    U_ = sparse.linalg.spsolve(A.tocsr(),F_)

    # construct 2D solution
    U = bd
    U[1:m+1,1:m+1] = U_.reshape(m,m)

    # plot output
    if(img_q):
        plt.figure()
```

```
        plt.pcolormesh(x,x,U.T) # transpose because x are rows
        plt.axis('image')
        plt.colorbar()
        plt.savefig('img/1/'+str(m)+'.pdf')

    return U
```

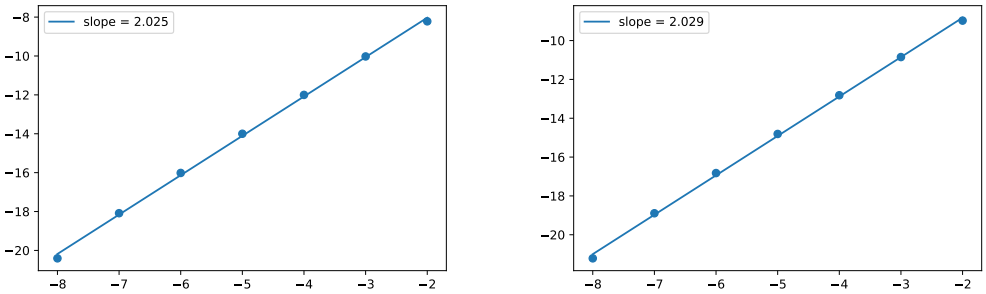
This gives solutions shown in Figure 1.



(a) $m = 3$ (b) $m = 7$ (c) $m = 15$ (d) $m = 31$

Figure 1: Solutions using 5 point Laplacian with $m$ interior points in each direction

We now analyze the results by comparing the results on a coarse mesh to those on a finer mesh. By plotting the error vs the mesh width on a log-log plot, the slope will give the order of convergence. This is done in Figure 2. As expected the convergence is order $h^2$.




(a) $\log h$ vs. $\log \|\text{error}\|_\infty$ (b) $\log h$ vs. $\log \|\text{error}\|_{L2}$

Figure 2: Error vs fine mesh for 5-point Laplacian

### Problem 2

Now use the 9-point formula with the correction term described in Sec. 3.5 to solve the same problem as in the previous exercise. Again take $f(x, y) = x^2 + y^2$, and numerically test the order of accuracy of your code by solving on a fine grid, pretending that is the exact solution, and comparing coarser grid approximations to the corresponding values of the fine grid solution. Show what order of accuracy you achieve and try to explain why.

### Solution

We have 9-point formula given as,

$$\nabla_9^2 u_{ij} = \frac{1}{6h^2}\left[4u_{i-1,j} + 4u_{i+1,j} + 4u_{i,j-1} + 4u_{i,j+1} + u_{i-1,j-1} + u_{i-1,j+1} + u_{i+1,j-1} + u_{i+1,j+1} - 20u_{ij}\right]$$

Given that $u$ is sufficiently smooth we have,

$$\frac{1}{12}h^2(u_{xxxx} + 2u_{xxyy} + u_{yyyy}) + \mathcal{O}(h^4)$$

We note that,

$$u_{xxxx} + 2u_{xxyy} + u_{yyyy} = \nabla^2(\nabla^2 u) = \nabla^2 f$$

Therefore we have fourth order accurate finite difference method given by,

$$\nabla_9^2 u_{ij} = f(x_i, y_j) + \frac{h^2}{12}\nabla^2 f(x_i, y_j)$$

In this problem we have,

$$\nabla^2 f = f_{xx} + f_{yy} = 2 + 2 = 4$$

The matrix equation corresponding to this finite difference method is of the form,

$$Au = f$$

where

$$A = \begin{bmatrix} T & S & & \\ S & \ddots & \ddots & \\ & \ddots & \ddots & S \\ & & S & T \end{bmatrix}, \quad T = \begin{bmatrix} -20 & 4 & & \\ 4 & \ddots & \ddots & \\ & \ddots & \ddots & 4 \\ & & 4 & -20 \end{bmatrix}, \quad S = \begin{bmatrix} 4 & 1 & & \\ 1 & \ddots & \ddots & \\ & \ddots & \ddots & 1 \\ & & 1 & 4 \end{bmatrix}$$

and, given $f = x^2 + y^2$ so that $\nabla^2 f = 4$,

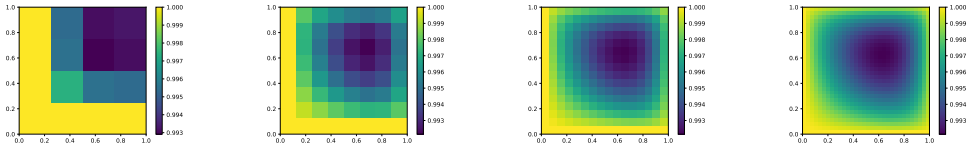
$$f_{ij} = x_i^2 + x_j^2 - \partial u_{ij}/h^2 + 4, \qquad \partial u_{ij} = 6|\{i, j\} \cap \{1, m\}| - \max(0, |\{i, j\} \cap \{1, m\}| - 1)$$

We implement our nine point stencil for solving the Poisson equation in Python.

```
def nine_point_poisson(m,img_q=False):
    # define RHS function
    def f(x,y): return x**2 + y**2
```

```
    def d2f(x,y): return 4

    # define boundary conditions
    bd = np.ones((m+2,m+2))
    bd[1:m+1,1:m+1] = np.zeros((m,m)) #must be zero on interior for below

    x = np.linspace(0,1,m+2)
    h = 1/(m+1)

    # construct A
    T = -20*sparse.eye(m,m,k=0) + 4*sparse.eye(m,m,k=1) + 4*sparse.eye(m,m,k
        =-1)
    S = 4*sparse.eye(m,m,k=0) + sparse.eye(m,m,k=1) + sparse.eye(m,m,k=-1)
    I = sparse.eye(m)
    II = sparse.eye(m,k=1)+sparse.eye(m,k=-1)
    A = sparse.kron(I,T)+sparse.kron(II,S)
    A /= h**2

    # build RHS
    F = f(x[1:m+1,None],x[None,1:m+1])

    # subtract stencil evaluated at known boundary points
    F -= 4*(bd[0:m,1:m+1] + bd[2:m+2,1:m+1] + bd[1:m+1,0:m] + bd[1:m+1,2:m
        +2])/h**2
    F -= (bd[0:m,0:m] + bd[0:m,2:m+2] + bd[2:m+2,0:m] + bd[2:m+2,2:m+2])/h
        **2

    # add dominant error term
    F += (h**2/12)*d2f(x[1:m+1,None],x[None,1:m+1])

    # evaluate f along grid, account for known boundary points and error
        term, and flatten
    F_ = np.reshape(F,(-1,))

    # solve system
    U_ = sparse.linalg.spsolve(A.tocsr(),F_)

    # construct 2D solution
    U = bd
    U[1:m+1,1:m+1] = U_.reshape(m,m)

    # plot output
    if(img_q):
        plt.figure()
        plt.pcolormesh(x,x,U.T)
        plt.axis('image')
        plt.colorbar()
        plt.savefig('img/2/'+str(m)+'.pdf')

    return U
```

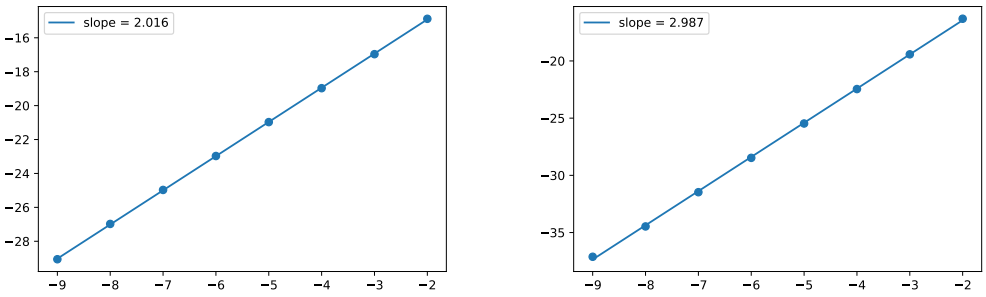
This gives solutions shown in Figure 3.



(a) $m = 3$ (b) $m = 7$ (c) $m = 15$ (d) $m = 31$

Figure 3: Solutions using 9 point Laplacian with $m$ interior points in each direction

As before the errors are plotted on a log-log plot. We use a grid as fine as $1023 \times 1023$ interior points which corresponds to solving a system with over a million equations!




(a) $\log h$ vs. $\log \|\text{error}\|_\infty$ (b) $\log h$ vs. $\log \|\text{error}\|_{L2}$

Figure 4: Error vs fine mesh for 9-point Laplacian

The condition for using this error correction to get order $h^4$ convergence was the smoothness of $u$. In particular, we require $u_{xxxx}, u_{xxyy}, u_{yyyy}$ to all exist.

However, our boundary conditions correspond to a level curve of $u$ in the shape of a square. In the corners we have $u_x = u_y = 0$ so that $\nabla^2 u = u_{xx} + u_{yy} = 0$. However, $\nabla^2 u = f(x, y) = x^2 + y^2$ is nonzero at $(0, 1), (1, 0)$, and $(1, 1)$. This means there is some discontinuity in $u_{xx} + u_{yy}$ as we approach these corners.

As seen in our numerical results, this is enough to change the error term in our analysis and prevent $h^4$ convergence.

**Problem 3**

We have discussed using finite element methods to solve elliptic PDE's such as

$$\triangle u = f \quad \text{in } \Omega, \quad u = 0 \quad \text{on } \partial\Omega,$$

with *homogeneous* Dirichlet boundary conditions. How could you modify the procedure to solve the *inhomogeneous* Dirichlet problem:

$$\triangle u = f \quad \text{in } \Omega, \quad u = g \quad \text{on } \partial\Omega,$$

where $g$ is some given function? Derive the equations that you would need to solve to compute, say, a continuous piecewise bilinear approximation for this problem when $\Omega$ is the unit square $(0,1) \times (0,1)$.

**Solution**

We seek a solution to $\mathcal{L}u = f$ in $\Omega = (0,1) \times (0,1)$ given some boundary condition $u = g$ on $\partial\Omega$.

With homogeneous Dirichlet boundary conditions of zero we have,

$$S = \{\varphi : \varphi \text{ is a continuous bilinear function on given grid with } \varphi(\omega) = 0, \forall \omega \in \partial\Omega\}$$

Define an inner product $\langle \cdot, \cdot \rangle : S \times S \to \mathbb{R}_+$ by,

$$\langle f, g \rangle = \iint_\Omega f \cdot g \; dA$$

Note that when we write $\langle \mathcal{L}\hat{u}, \hat{v} \rangle$ we mean the value of the integral after applying Green's theorem as $\mathcal{L}\hat{u}$ may not be well defined if $\hat{u}$ is not sufficiently smooth. In particular, with $\mathcal{L}u = \nabla^2 u = u_x x + u_y y$ we have,

$$\langle \mathcal{L}\hat{u}, \hat{v} \rangle = \iint_\Omega (\hat{u}_{xx} + \hat{u}_{yy})\hat{v} dA = -\iint_\Omega (\hat{u}_x \hat{v}_x + \hat{u}_y \hat{v}_y) dA + \int_{\partial\Omega} \hat{u}_n \hat{v} d\gamma$$

where $\hat{u}_n$ is the derivative of $\hat{u}$ in the outward normal direction along the boundary. Note further that if $\hat{v} = 0$ (as is the case for entries of $A$ and $F$ below) on $\partial\Omega$ that the second term is zero.

We extend the current set of basis functions for $S$ to span the set of all piecewise bilinear functions on the given grid, denoted $S'$.

Let $\{x_0, x_1, \ldots, x_{n_x}, x_{n_x+1}\} \times \{y_0, y_1, \ldots, y_{n_y}, y_{n_y+1}\}$ define the mesh for $S$. A basis $B$ for $S$ is $B = \{\varphi_k\}_{k=1}^{n_x n_y}$, where $\varphi_{i+(j-1)n_x}(x,y)$ is centered at $(x_i, y_j)$ and is defined as,

$$\varphi_{i+(j-1)n_x}(x,y) = \begin{cases} (x - x_{i-1})(y - y_{j-1})/(x_i - x_{i-1})(y_j - y_{j-1}) & \text{in } [x_{i-1}, x_i] \times [y_{j-1}, y_j] \\ (x - x_{i+1})(y - y_{j-1})/(x_i - x_{i+1})(y_j - y_{j-1}) & \text{in } [x_i, x_{i+1}] \times [y_{j-1}, y_j] \\ (x - x_{i-1})(y - y_{j+1})/(x_i - x_{i-1})(y_j - y_{j+1}) & \text{in } [x_{i-1}, x_i] \times [y_j, y_{j+1}] \\ (x - x_{i+1})(y - y_{j+1})/(x_i - x_{i+1})(y_j - y_{j+1}) & \text{in } [x_i, x_{i+1}] \times [y_j, y_{j+1}] \\ 0 & \text{otherwise} \end{cases}$$

We now extend $B$ to a basis $B'$ for $S'$. Indeed, take the above definition, except, since $x_{-1}, x_{n_x+2}, y_{-1}, y_{n_y+2}$ are not defined, ignore the definition in regions which would be outside $[0,1] \times [0,1]$. Let all such functions be called $B'$.

Clearly $|B'| = (n_x + 2)(n_y + 2)$ which is the dimension of $S'$ as functions in $S'$ have this many free parameters. Moreover, they elements of $B'$ are linearly independent since they do not overlap at the centers. This proves $B'$ is a basis for $S'$.

Keep the numbering for the basis for $S$ and append the new vectors with some new ordering. For convenience define $M = n_x n_y$ and $N = (n_x + 2)(n_y + 2)$. That is, $M = |B|$ and $N = |B'|$.

With the new basis our approximate solution $\hat{u}$ to the differential equation $\mathcal{L}u = f$ is of the form,

$$\hat{u} = \sum_{k=1}^{N} c_k \varphi_k$$

Given boundary condition $u = g$ on the boundary we have $c_k = g(x_i, y_j)$ for $k = M+1, \dots N$, where $\varphi_k$ is centered at $(x_i, y_j)$.

This leaves $M$ unknown coefficients. Thus, for any $k = 1, ..., M$, write,

$$\langle f, \varphi_k \rangle = \langle \mathcal{L}\hat{u}, \varphi_k \rangle = \sum_{n=1}^{N} c_n \langle \mathcal{L}\varphi_n, \varphi_k \rangle = \sum_{n=1}^{M} c_n \langle \mathcal{L}\varphi_n, \varphi_k \rangle + \sum_{n=M+1}^{N} c_n \langle \mathcal{L}\varphi_n, \varphi_k \rangle$$

Since we know the value of $c_n$ for all $n \in \{M+1, \dots, N\}$ we can subtract the second sum from $\langle f, \varphi_k \rangle$. This means the matrix $A$ will be the same as previously. However, the right hand side $F$ will be changed.

Specifically, we are solving $AC = F$ where, for $k, n \in \{1, 2, \dots, n_x n_y\}$,

$$A_{k,n} = \langle \mathcal{L}\varphi_n, \varphi_k \rangle, \qquad C_n = c_n, \qquad F_k = \langle f, \varphi_k \rangle - \sum_{n=M+1}^{N} c_n \langle \mathcal{L}\varphi_n, \varphi_k \rangle$$

We make a few remarks. First, $\langle \varphi_n, \varphi_m \rangle = 0$ if $\varphi_n$ and $\varphi_m$ are not centered in adjacent grid points (diagonals included). This means the sums above can be simplified quite a bit by not evaluating needless things. Moreover, $\langle \mathcal{L}\varphi_n, \varphi_k \rangle$ may have a nice closed form expression depending on $\mathcal{L}$. In particular, if $\mathcal{L}$ does not depend on $x$ or $y$, then only the relative positions of the centers of $\varphi_n$ and $\varphi_k$ are important.

**Problem 4**

The key to efficiency in the `chebfun` package, which you used in a previous homework exercise, is the ability to rapidly translate between the values of a function at the Chebyshev points, $\cos(\pi j/n)$, $j = 0, \ldots, n$, and the coefficients $a_0, \ldots, a_n$, in a Chebyshev expansion of the function's $n$th-degree polynomial interpolant: $p(x) = \sum_{j=0}^{n} a_j T_j(x)$, where $T_j(x) = \cos(j \arccos(x))$ is the $j$th degree Chebyshev polynomial. Knowing the coefficients $a_0, \ldots, a_n$, one can evaluate $p$ at the Chebyshev points by evaluating the sums

$$p(\cos(k\pi/n)) = \sum_{j=0}^{n} a_j \cos(jk\pi/n), \quad k = 0, \ldots, n. \tag{1}$$

These sums are much like the real part of the sums in the FFT,

$$F_k = \sum_{j=0}^{n-1} e^{2\pi ijk/n} f_j, \quad k = 0, \ldots, n-1,$$

but the argument of the cosine differs by a factor of 2 from the values that would make them equal. Explain how the FFT or a closely related procedure could be used to evaluate the sums in (1). To go in the other direction, and efficiently determine the coefficients $a_0, \ldots, a_n$ from the function values $f(\cos(k\pi/n))$, what method would you use?

**Solution**

First observe that for any integer $k$ and $n$ that,

$$\cos\left(\frac{\pi(n-j)k}{n}\right) = \cos(\pi k)\cos\left(-\frac{\pi jk}{n}\right) = \cos(\pi k)\cos\left(\frac{\pi jk}{n}\right) = \cos\left(\frac{\pi(n+j)k}{n}\right)$$
$$\sin\left(\frac{\pi(n-j)k}{n}\right) = \cos(\pi k)\sin\left(-\frac{\pi jk}{n}\right) = -\cos(\pi k)\sin\left(\frac{\pi jk}{n}\right) = -\sin\left(\frac{\pi(n+j)k}{n}\right)$$

With this in mind, for $j = 1, 2, \ldots, n$ define,

$$a_{n+j} = a_{n-j}$$

Then for any integer $k = 1, 2, \ldots, n$ the DFT gives,

$$A_k = \sum_{j=0}^{2n-1} a_j e^{\pi ijk/n} = \sum_{j=0}^{2n-1} a_j \cos\left(\frac{\pi jk}{n}\right) + i a_j \sin\left(\frac{\pi jk}{n}\right)$$

Therefore,

$$a_0 + a_n e^{ik\pi} + A_k = a_{2n} + a_n e^{\pi nk/n} + \sum_{j=0}^{n} a_j e^{\pi ijk/n} + \sum_{j=n+1}^{2n-1} a_j e^{\pi ijk/n} = \sum_{j=0}^{n} a_j e^{\pi ijk/n} + \sum_{j=n}^{2n} a_j e^{\pi ijk/n}$$

By the relationships between cosine and sine shown above the imaginary parts of these sums

cancel, and the real parts are the same. That is,

$$a_0 + a_n e^{ik\pi} + A_k = 2\,\mathrm{Re}\left[\sum_{j=0}^{n} a_j e^{\pi i jk/n}\right] = 2\sum_{j=0}^{n} a_j \cos\left(\frac{\pi jk}{n}\right)$$

Therefore,

$$p(\cos(k\pi/n)) = \sum_{j=0}^{n} a_j \cos\left(\frac{\pi jk}{n}\right) = \frac{a_0 + (-1)^k a_n + A_k}{2}$$

We can use the DFT to compute $A_k$ in $n \log n$ time so we can compute all of the $p(\cos(k\pi/n))$ in $n \log n$ time. This is better than the $n^2$ time it would take to compute each of the $n$ sums of $n$ terms directly.

Given $p(\cos(k\pi/n))$ for $k = 0, \ldots, n$ and assumign we know $a_0$ and $a_n$ we can compute $A_k$ for $k = 0, \ldots, n$. We then would apply a similar process to above, but with the inverse DFT.