# Math 5301 – Numerical Analysis– Spring 2025
w/Professor Du

Paul Carmody
Midterm – April 14, 2025

---

**Assignment:** Consider 1D Poisson Equation $-\Delta x = \sin(\pi x)$, over the region $(0, \pi/2)$ with boundary conditions $u(0) = 0$ and $u(\pi/2) = 1$. Using central difference scheme and a mesh of 128, obtain a linear system of $Au = f$ for the problem, then the solve the system using the following methods until a relative residual of $10^{-4}$ is reached. For all methods below, plot the analytical solution, the numerical solution and the error distribution. Use zero vectors as your initial guess.

(a) Gause-Segal Method. Plot the rate of convergence and compare it with analysis.

(b) Conjugate Gradient Method. Compare the rate of convergence with that obtained in (a). Do not use the CG solver provided by MatLab.

(c) Conjugage Gradient Method Preconditioned by Cholesky Decomposition. Compare the rate of convergence with that obtained by (b).

---

## Analytical and several Numerical Solutions to Poisson's Equation

**Introduction:**

We will begin by describing the equation and providing an analytical solution. Then solve this equation using Jacobi's Method and the Steepest Descent Method. Comparisons will be made as to accurancy and rate of convergence.

**Poison's Equation and an Analytical Solution:**

As described in the assignment we will focus our attenion on this Poisson equation and initial conditions.

$$-\Delta x = \sin(\pi x)$$
$$u(0) = 0 \text{ and } u(\pi/2) = 1.$$

When we solve this problem analtyically we get

$$\begin{aligned}
u'(x) &= \int -\sin(\pi x)dx \\
&= \frac{1}{\pi}\cos(\pi x) + C \\
u(x) &= \int \left(\frac{1}{\pi}\cos(\pi x) + C\right)dx \\
&= \frac{1}{\pi^2}sin(\pi x) + Cx + D \\
u(0) = 0 &= \frac{1}{\pi^2}sin(\pi 0) + Cx + D \\
D &= 0 \\
u(\pi/2) = 1 &= \frac{1}{\pi^2}sin(\pi^2/2) + C(\pi/2) \\
C &= \frac{2\left(1 - \frac{1}{\pi^2}\right)sin(\pi^2/2)}{\pi^2} = \frac{2\left(1 - \frac{1}{9.869604064}\right)0.086022097}{9.869604064} = 1.998233797 \approx 2 \\
u(x) &\approx \frac{1}{\pi^2}sin(\pi x) + 2x
\end{aligned}$$

### Centered Difference Scheme

We will attempt to approximate the curve of the solution at particular points $u_i$ by calculating a slope at a point by using the preceeding point $u_{i-1}$ and succeeding point $u_{i+1}$.

$$\frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} \approx -\sin(\pi x)$$

where $h = (\pi/2)/129 = \pi/258$ (we use $N+1$ as we start with the left boundary 0). This can be reduced to

$$-u_{i+1} + 2u_i - u_{i-1} = h^2 \sin(\pi x).$$

This expands to a linear function over the a matrix $A$ and vector $u = \{u_i\}$ reflecting the left hand side and the value to our function on the right with $f = \{f_i\}, f_i = \sin(\pi x_i)$ or

$$\begin{aligned}
Au &= h^2 f \\
\begin{pmatrix} 2 & -1 & 0 & 0 & \cdots & 0 \\ -1 & 2 & -1 & 0 & \cdots & 0 \\ 0 & -1 & 2 & -1 & \cdots & 0 \\ 0 & 0 & -1 & 2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 2 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ \vdots \\ u_i \end{pmatrix} &= h^2 \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ \vdots \\ f_i \end{pmatrix} \\
u &= h^2 A^{-1} f
\end{aligned}$$

remembering the boundary conditions. Since $A$ is tri-diagonal we can use several methods and compare the cost and efficiency. The MatLab code for the Central Difference Method is

```
clc; clear; close all;

figure; hold on;

function [x, dsc] = centered_difference_scheme(mesh)

```

```
    h = 1/mesh;
    x = 0:h:pi/2;
    N = length(x) - 2;

    % Construct finite difference matrix A
    A = (1/h^2) * (diag(-2*ones(N,1)) + diag(ones(N-1,1),1) + diag(ones(N
        -1,1),-1));
    b = sin(pi*x(2:end-1));

    % Solve the linear system A*u = b
    %u = A \ reshape(b, [], 1);
    u = A \ b.';

    % Include boundary values u(0) = 0, u(pi/2) = 1)
    dsc = [0; u; 1];

end

[x, u_full]= centered_difference_scheme(128);

%calculate the analytic solution
%analytic = (-1/pi^2)*sin(pi*x);
analytic = 1/pi^2*sin(pi*x)+2/pi*(1-1/pi^2)*x;
%analytic = (-1/pi^2)*sin(pi*x)+2*x;

% Plot the solutions
plot(x, u_full, '-b', 'DisplayName', sprintf('h = %.3f', 1/128))
plot(x, analytic, '-r', 'DisplayName', 'analytic')
xlabel('x');
ylabel('u(x)');
title('Poisons Equation on [0,pi/2]: Solution using 128');
legend;
```

**Gause-Segal Method.**

Since $A$ is a tri-diagonal matrix it can be divided into three separate matrices that add up. Let $D$ be zero everywhere except the diagonal where it will hold the values of $A_{ii}$ (namely all 2s). By applying this iteratively we get

$$u_i^{[k+1]} = \frac{1}{2}(u_{i-1}^{[k]} + u_{i+1}^{[k]} - h^2 f_i)$$

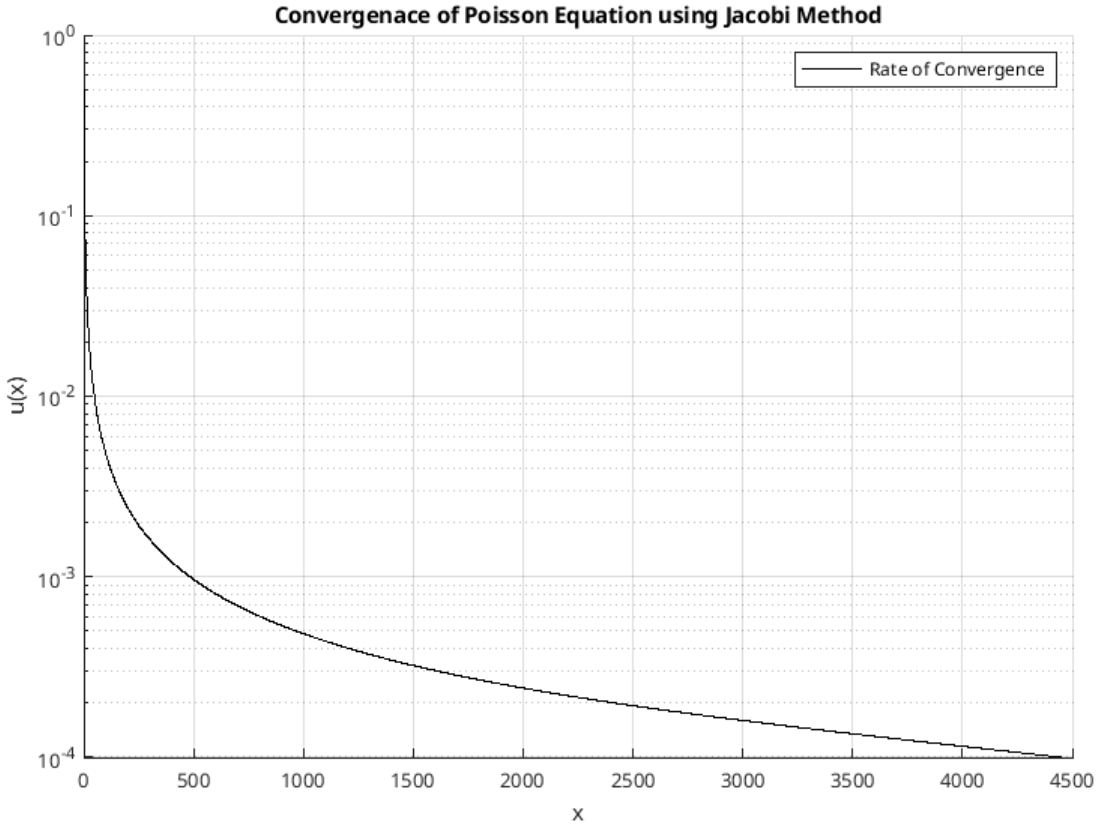
Here is the MatLab code used to generate the graphs that follow:

```
clc; clear; close all;

figure; hold on;

function [conv, cnt] = jacobi_method(A, b, h, tol, max_iter)
    % Solves Ax = b using Jacobi's iterative method
    % Inputs:
    %   A        - Coefficient matrix (NxN)
    %   b        - Right-hand side vector (Nx1)
    %   tol      - Convergence tolerance
    %   max_iter - Maximum number of iterations
    % Output:
```

```
13      %   x - Solution vector (Nx1)
14
15      N = length(b);                  % Number of equations
16      x = b;%zeros(N, 1);               % Initial guess (zero vector)
17      x_old = x;                      % Store previous iteration
18      h2=h^2;
19      conv_loc = 1:10000;
20      conv=conv_loc;
21      cnt=0;
22
23      for k=1:max_iter
24          for i=2:N-1
25              x(i) = 1/2*( x_old(i-1) + x_old(i+1) - h2*b(i) );
26          end
27
28          % Check for convergence
29  %         if norm(x - x_old, 2) < tol
30            conv_loc(k) = norm(x-x_old, Inf);
31            cnt = k;
32
33          if norm(x - x_old, Inf) < tol
34              fprintf('Converged in %d iterations.\n', k);
35              conv = conv_loc;
36              return;
37          end
38          x_old = x;  % Update solution
39      end
40
41      fprintf('Max iterations reached without convergence.\n');
42      conv = conv_loc;
43  end
44  % Define problem parameters
45  N = 128;                    % Number of internal grid points
46  h = (pi/2) / (N+1);         % Grid spacing
47  x = linspace(h, pi/2-h, N)';  % Grid points
48  f = h^2* sin(pi * x);     % Right-hand side vector
49
50  % Construct tridiagonal matrix A
51  A = 2 * eye(N) - diag(ones(N-1,1),1) - diag(ones(N-1,1),-1);
52
53  % Modify last element of f to include boundary condition u(pi/2) = 1
54  f(1) = 0; f(N) = 1;
55
56  % Solve using Jacobi method
57  tol = 1e-4;  % Convergence tolerance
58  max_iter = 10000;  % Maximum iterations
59
60  [u, cnt] = jacobi_method(A, f, h, tol, max_iter);
61
62  output = u(1:cnt);
63  grid on;
64  yscale log;
65  horizontal = 1:length(output);%linspace(1, cnt, 1);
```

```
plot(horizontal, output, '-k', 'DisplayName', 'Rate of Convergence');
xlabel('x'); ylabel('u(x)');
title('Convergenace of Poisson Equation using Jacobi Method');
legend;
grid on;
```

From this we generate the following graphs.




### Conjugate Gradient Method

This technique for approximating our solution to the Poisson equation is to make each iteration in the direction of greatest change. That is,

$$\nabla\phi(u_{k-1}) = Au_{k-1} - f \equiv -r_{k-1}$$

where $\phi : \mathbb{R}^m \to \mathbb{R}$ of the form

$$\phi(u) = \frac{1}{2}u^T Au - u^T f$$

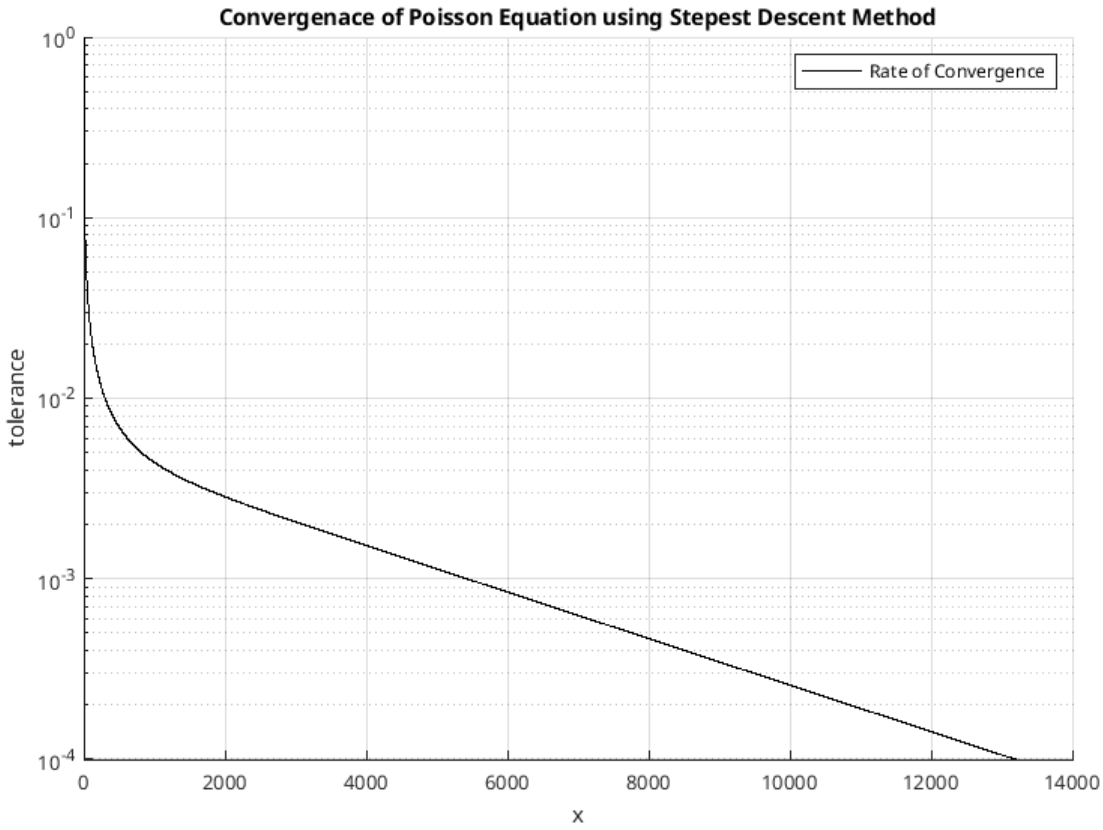
which is a quadratic function in $u$ and can be mapped with local extrema either as a top, a bowl or a saddle point, all based on the eigenvalues of $A$ (negative, positive, or neither, respectively). Thus, when $A$ is SPD we can expect $r_k$ to progress ever closer towards the extremum.

This is the source code

```
clc; clear; close all;

figure; hold on;

function [conv, cnt] = steepest_descent(A, f, tol, max_iter)
    % Solves Ax = b using steepest descent method
    % Inputs:
    %   A         - Coefficient matrix (NxN)
    %   b         - Right-hand side vector (Nx1)
```

```
%   tol      - Convergence tolerance
%   max_iter - Maximum number of iterations
% Output:
%   x - Solution vector (Nx1)

N = length(f);                % Number of equations
u = zeros(N, 1);              % Initial guess (zero vector)
u_old = u;
conv = 1:10000;

for k=1:max_iter
    r_old = f - A*u_old;
    conv(k) = norm(r_old);
    cnt = k;

    if norm(r_old) < tol
        fprintf('Converged in %d iterations.\n', k);
        return;
    end
    alpha_old = (r_old.' * r_old)/(r_old.' * A * r_old);
    u = u_old +  alpha_old*r_old;
    u_old = u;
end

fprintf('Max iterations reached without convergence.\n');
end
% Define problem parameters
N = 128;                      % Number of internal grid points
h = (pi/2) / (N+1);           % Grid spacing
x = linspace(h, pi/2-h, N)';  % Grid points
f = h^2* sin(pi * x);     % Right-hand side vector

% Construct tridiagonal matrix A
A = 2 * eye(N) - diag(ones(N-1,1),1) - diag(ones(N-1,1),-1);

% Modify last element of f to include boundary condition u(pi/2) = 1
f(1) = 0; f(N) = 1;

% Solve using steepest descent method
tol = 1e-4;  % Convergence tolerance
max_iter = 100000;  % Maximum iterations

[conv, size] = steepest_descent(A, f, tol, max_iter);

output = conv(1:size);
grid on;
yscale log;
horizontal = 1:length(output);%linspace(1, cnt, 1);
plot(horizontal, output, '-k', 'DisplayName', 'Rate of Convergence');
xlabel('x'); ylabel('tolerance');
title('Convergenace of Poisson Equation using Stepest Descent Method');
legend;
```

From this we generate the following graphs.




### Conjugage Gradient Method Preconditioned by Cholesky Decomposition

This technique for approximating our solution to the Poisson equation is to make each iteration in the direction of greatest change. That is,

$$\nabla\phi(u_{k-1}) = Au_{k-1} - f \equiv -r_{k-1}$$

where $\phi : \mathbb{R}^m \to \mathbb{R}$ of the form

$$\phi(u) = \frac{1}{2}u^T Au - u^T f$$

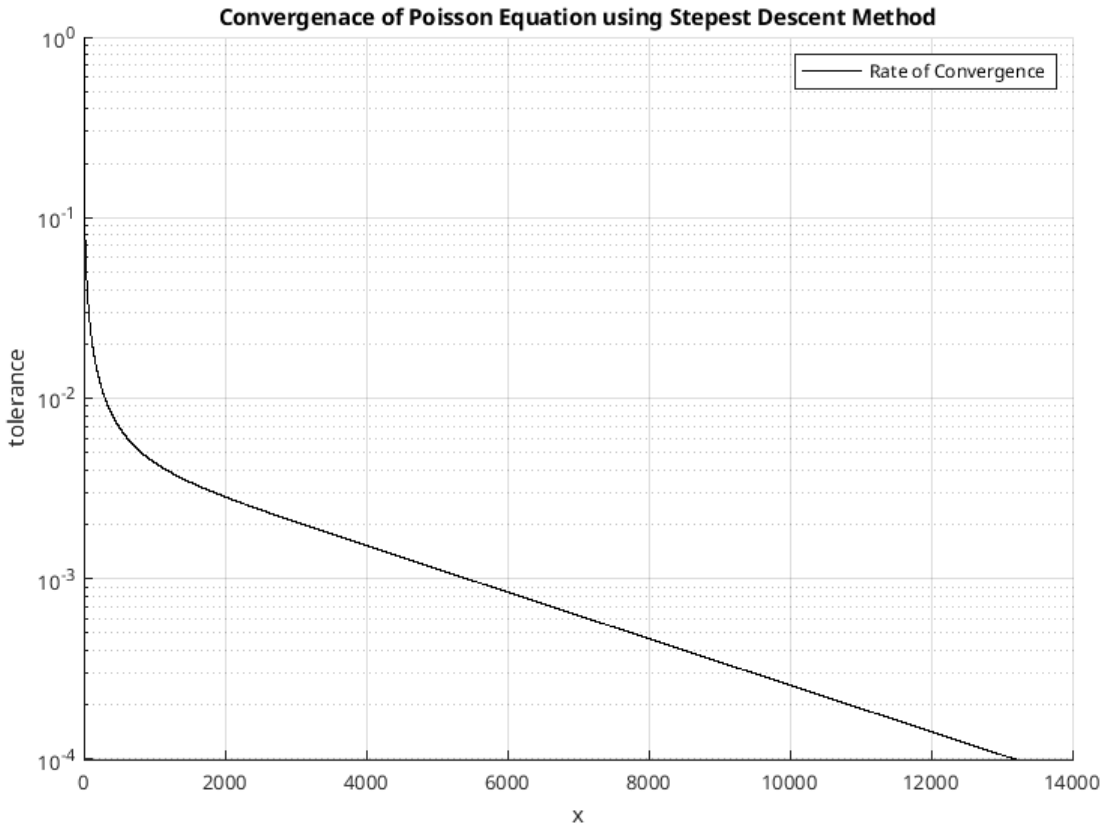
which is a quadratic function in $u$ and can be mapped with local extrema either as a top, a bowl or a saddle point, all based on the eigenvalues of $A$ (negative, positive, or neither, respectively). Thus, when $A$ is SPD we can expect $r_k$ to progress ever closer towards the extremum.

This is the source code

```
clc; clear; close all;

figure; hold on;

function [conv, cnt] = steepest_descent(A, f, tol, max_iter)
    % Solves Ax = b using steepest descent method
    % Inputs:
    %   A        - Coefficient matrix (NxN)
    %   b        - Right-hand side vector (Nx1)
    %   tol      - Convergence tolerance
    %   max_iter - Maximum number of iterations
    % Output:
    %   x - Solution vector (Nx1)

    N = length(f);                % Number of equations
```

```
    u = zeros(N, 1);              % Initial guess (zero vector)
    u_old = u;
    conv = 1:10000;

    for k=1:max_iter
        r_old = f - A*u_old;
        conv(k) = norm(r_old);
        cnt = k;

        if norm(r_old) < tol
            fprintf('Converged in %d iterations.\n', k);
            return;
        end
        alpha_old = (r_old.' * r_old)/(r_old.' * A * r_old);
        u = u_old +  alpha_old*r_old;
        u_old = u;
    end

    fprintf('Max iterations reached without convergence.\n');
end
% Define problem parameters
N = 128;                      % Number of internal grid points
h = (pi/2) / (N+1);           % Grid spacing
x = linspace(h, pi/2-h, N)';  % Grid points
f = h^2* sin(pi * x);     % Right-hand side vector

% Construct tridiagonal matrix A
A = 2 * eye(N) - diag(ones(N-1,1),1) - diag(ones(N-1,1),-1);

% Modify last element of f to include boundary condition u(pi/2) = 1
f(1) = 0; f(N) = 1;

% Solve using steepest descent method
tol = 1e-4;  % Convergence tolerance
max_iter = 100000;  % Maximum iterations

[conv, size] = steepest_descent(A, f, tol, max_iter);

output = conv(1:size);
grid on;
yscale log;
horizontal = 1:length(output);%linspace(1, cnt, 1);
plot(horizontal, output, '-k', 'DisplayName', 'Rate of Convergence');
xlabel('x'); ylabel('tolerance');
title('Convergenace of Poisson Equation using Stepest Descent Method');
legend;
```

From this we generate the following graphs.




**Analyzing the various solutions**.

It should be noted that the Jacobi method took 4,452 iterations before being completed and the Steepest Descent Method took 13,454.

The two plots use a logorithmic scale. A standard scale didn't emphasize the decay rate effectively as the initial iterations showed the great change, so much so that the following improved accuracy couldn't be seen. The logarithmic scale more effictively shows the improvement. Each iteration provides very little but necessary improvement.