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) Gauss-Seidel 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

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

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

$$Au = h^2 f$$

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

$$u = h^2 A^{-1} f$$

remembering the boundary conditions. Since A is tri-diagonal we can use several methods and compare the cost and efficiency. From here we use several methods meant to study the efficiecny and optimization of this method.

Gauss-Seidel 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:

```
function x = gauss_seidel(A, b, N, h, tol, max_iter)
    % Solves Ax = b using Gause Segal's iterative 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(b);                % Number of equations
    x = b;%zeros(N, 1);             % Initial guess (zero vector)
    x_old = x;                     % Store previous iteration
    h2=h^2;
    conv_loc = 1:10000;
```

```
16      conv=conv_loc;
17      cnt=0;
18
19      for k=1:max_iter
20          for i=2:N-1
21  %           x(i) = 1/2*( x_old(i-1) + x_old(i+1) - h2*b(i) ); % Jacobi
22              x(i) = 1/2*( x(i-1) + x_old(i+1) - h2*b(i) );  % Gaus-Segal
23          end
24
25          % Check for convergence
26  %         if norm(x - x_old, 2) < tol
27            conv_loc(k) = norm(x-x_old, Inf);
28            cnt = k;
29
30          if norm(x - x_old, Inf) < tol
31              fprintf('Converged in %d iterations.\n', k);
32              conv = conv_loc;
33              return;
34          end
35          x_old = x;  % Update solution
36      end
37
38      fprintf('Max iterations reached without convergence.\n');
39      conv = conv_loc;
40  end
```

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.

The source code for the Conjugage Gradient Method differs from that of the Conjugate Gradient Method Preconditioned by Cholesky Decomposition by only a few lines of code. The MatLab source code for both will be found in the next section.

Conjugage Gradient Method Preconditioned by Cholesky Decomposition

The primary differences with the Cholesky Decomposition is the use of modified version of the A matrix. This factorized matrix, M, to simplify the M^{-1} process as

$$Au = f \iff M^{-1}Au = M^{-1}Au = M^{-1}f$$

which improves the efficiency of the process. It can be shown that this has the same Condition Number as A. Some initialization is necessary to start with the preconditioning settting (found in lines 8

through 12). Then, when the new resolution is calculated, we utilze the preconditioning to recalculate our adjustment variable β (found in lines 36 through 40).

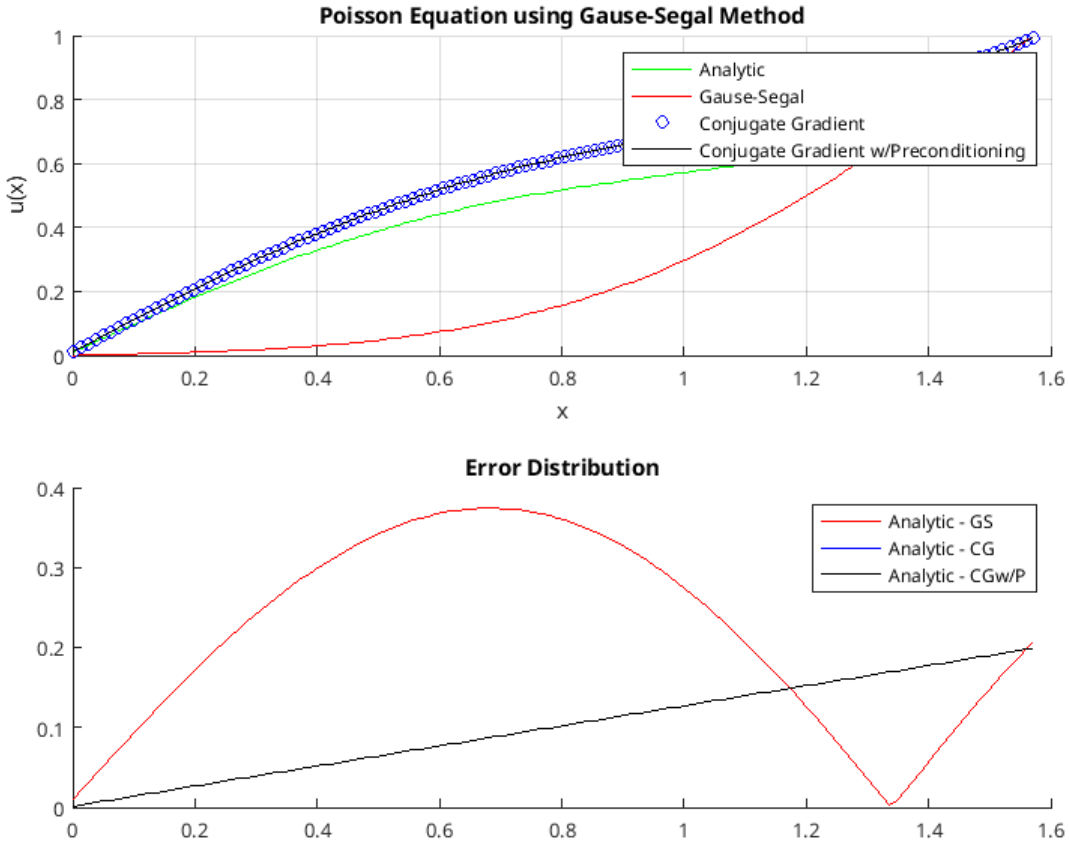
$$r_{k+1} = r_k^T M^{-1} r$$
$$\beta = r_{k+1}/r_k$$
$$p = r_k^T M^{-1} r_+ \beta p$$

This is the source code

```
1  function [u, resvec,k] = conjugate_gradient_method(A, f, N, h, tol,
       max_iter, M)
2      u = zeros(N,1);                    % Initial guess
3      r = f - A*u;                       % Initial residual
4      if M == 0
5          p = r;                         % Initial direction
6          rs_old = r' * r;               % Initial residual norm squared
7      else
8          M = chol(A,'lower');
9          precond = @(r) M' \ (M \ r);
10         z = precond(r);
11         p=z;                           % preconditioned residual
12         rs_old = r' * z;               % Initial residual preconditioned
13     end
14     res0 = norm(r);                    % Initial residual norm
15     resvec = zeros(max_iter,1);    % For storing residuals
16
17     for k = 1:max_iter
18         Ap = A * p;
19         alpha = rs_old / (p' * Ap);
20         u = u + alpha * p;
21         r = r - alpha * Ap;
22         res = norm(r) / res0;
23         resvec(k) = res;
24
25         if res < tol
26             fprintf('CG converged at iteration %d with relative residual
                   %.2e\n', k, res);
27             break;
28         end
29
30         if M == 0
31             rs_new = r' * r;
32             beta = rs_new / rs_old;
33             p = r + beta * p;
34             rs_old = rs_new;
35         else
36             z = precond(r);
37             rs_new = r' * z;
38             beta = rs_new / rs_old;
39             p = z + beta * p;
40             rs_old = rs_new;
41         end
42     end
43
```

```
end
```

From this we generate the following graphs.



We can see from this graph that the Conjugate Gradient Method (the blue circles and no lines) and the Conjugate Gradient Method w/Preconditioning (the black lines between) follow the same path. The resulting error distribution indicates the greater accuracy of the Conjugate Gradient Method over the Gauss-Seidel method as well as greater efficiency.

Analyzing the various solutions.

The following graph shows the rate of convergence of the Conjugate Gradient Method. The Congjugate Gradient Methdo w/Preconditioning used a single iteration before converging. It should be noted that the Gauss-Seidel method maximized its interation count before achieving the tolerance level of 10^{-4}.