Part 3.1: Discretization of the Poisson Equation**

Consider the two-dimensional Poisson equation:

$$-\Delta u(x,y) = f(x,y), \qquad (x,y) \in \Omega = (0,1)^2 \tag{9}$$

with homogeneous Dirichlet boundary conditions:

$$u(x,y) = 0 \qquad forall(x,y) \in \partial\Omega \tag{10}$$

To discretize this PDE, we use a regular grid with spacing

$$h = 1/(N+1) \tag{11}$$

which results in `N × N` internal points. We approximate the Laplacian operator using the 5-point finite difference stencil:

$$-\Delta u(x_{ij}) \approx -1/h^2[u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{i,j}] \tag{12}$$

This leads to the following discrete equation for each interior point:

$$-u_{i+1,j} - u_{i-1,j} - u_{i,j+1} - u_{i,j-1} + 4u_{i,j} = h^2 * f(x_i, y_j) \tag{13}$$

Express abbove's partial differential equation with specified boundary conditions as a linear system of the form:

$$Ay = b \tag{14}$$

Where:

- y is the vector of unknowns, obtained by flattening the 2D grid $u_{i,j}$ in a row-major order : $u_{i,j} \mapsto yk$ with $k = (i-1)N + j$, for $1 \le i,\ j \le N$, as it allows for straightforward flattening of the 2D grid into a 1D vector and aligns naturally with Kronecker product constructions of the matrix A.
- `A` is a symmetric positive-definite sparse matrix of size `N² × N²`, formed by combining 1D Laplacian matrices using Kronecker products:

$$A = kron(I,T) + kron(T,I) \tag{15}$$

Here, `T` is a tridiagonal matrix representing the 1D Laplace operator with Dirichlet boundary conditions: tridiag $(1,-2,1) \in R^{N\times N}$and `I` is the $N \times N$dentity matrix.

- `b` is the right-hand-side vector constructed by evaluating `f(x, y)` at each interior grid point and scaling by `h²`:

$$b_k = h^2 f(x_i, y_j) \tag{16}$$

correspongding to the same ordering as in vector y.

Thus, we have successfully transformed the continuous Poisson problem with homogeneous Dirichlet boundary conditions into a discrete linear system $Ay = b$, suitable for numerical solution via the CG method.

Part 3.2: Serial Conjugate Gradient Solver**

In this part, we solve the linear system derived in Part 3.1, which results from the 5-point finite difference discretization of the 2D Poisson equation on the unit square $\Omega = (0,1)^2$, with homogeneous Dirichlet boundary conditions.

The discretized system has the form: $Au = b$,

where A is a symmetric positive-definite sparse matrix of size $N^2 \times N^2$, and b is a vector obtained by evaluating the source function:

$f(x,y) = 2\pi 2 sin(\pi x) sin(\pi y)$on an $N \times N$ internal grid, scaled by h^2.

1） Implementation

I implemented the Conjugate Gradient (CG) algorithm in Matlabl. The function was designed to stop when the residual norm fell below the specified tolerance 10^{-8} or the maximum number of iterations was reached.

I tested the solver on various grid sizes N, doubling the resolution up to N=256. The Laplacian matrix was constructed using Kronecker products of tridiagonal matrices, in line with the structure described in Part 3.1.

A random initial guess was used to avoid any “lucky” alignment with the true solution, and timing was recorded for each test case.

I tried to avoid recalculating expensive terms like matrix-vector products and norms. For example, I reused `r'*r` and computed `A*p` just once per iteration. Also, instead of assembling the matrix manually, I used Kronecker products to construct the 2D Laplacian efficiently.

2） Numerical Results

The following table summarizes the performance of the CG solver across different grid sizes:

N	Iterations	Time (s)
8	28	0.0055
16	56	0.0097
32	110	0.0033
64	206	0.0061
128	412	0.1829
256	792	0.8452

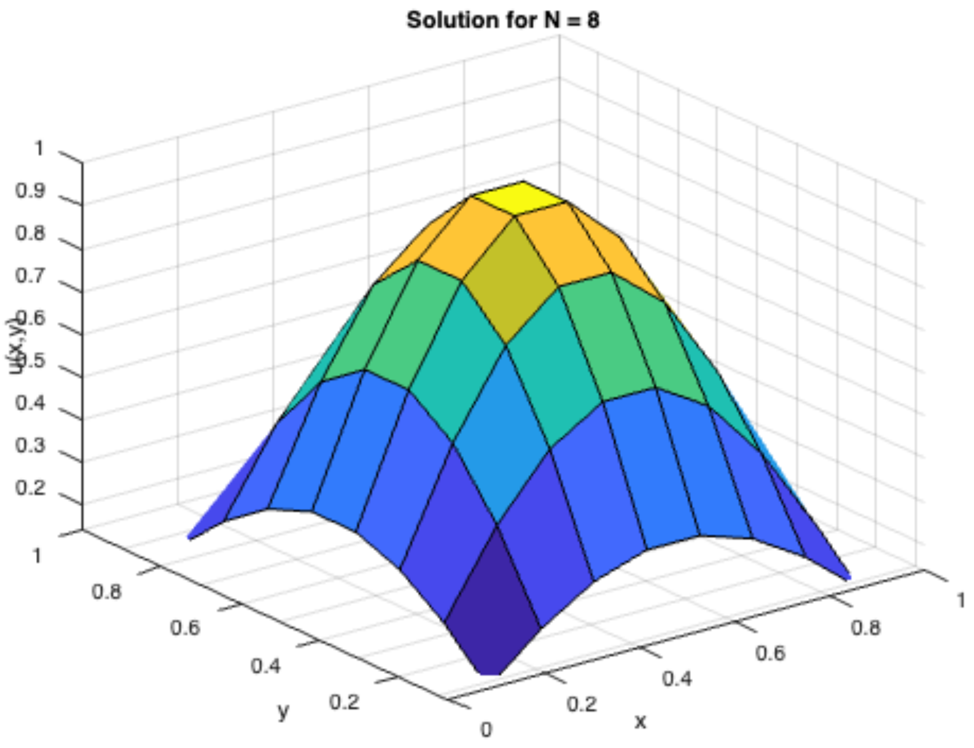
From the table, we observe that:

- As expected, the number of iterations roughly doubles as N is doubled.

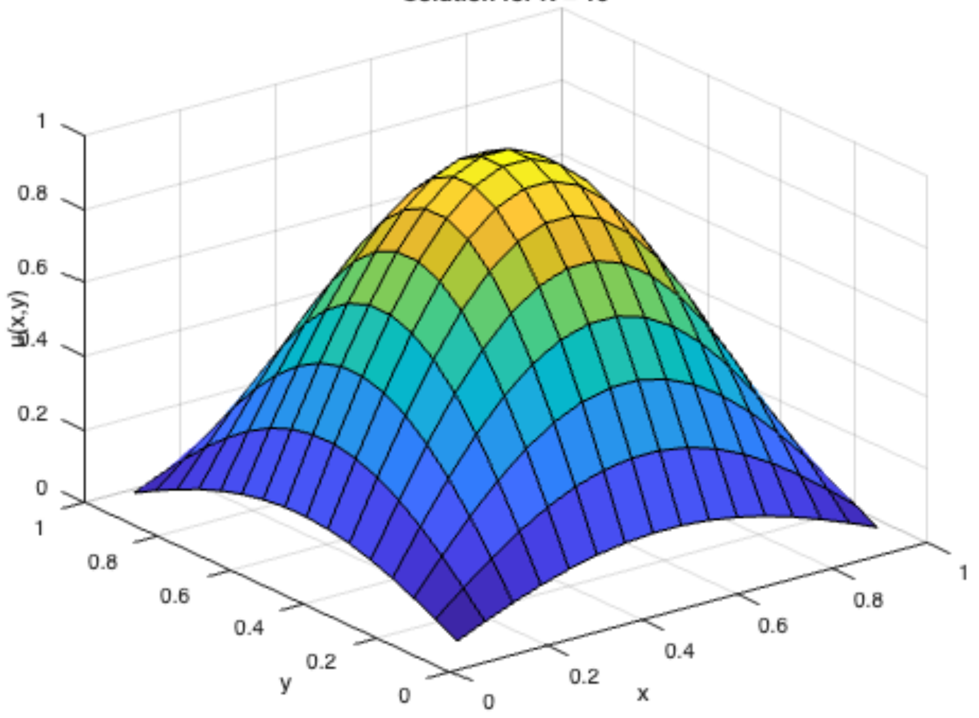
- The runtime increases significantly for large N, primarily due to the increase in the number of unknowns (N^2) and the cost of matrix-vector products.
- The convergence rate is slower for finer grids, which is consistent with the fact that the condition number of A Aincreases with N.

3) Approximate Solutions

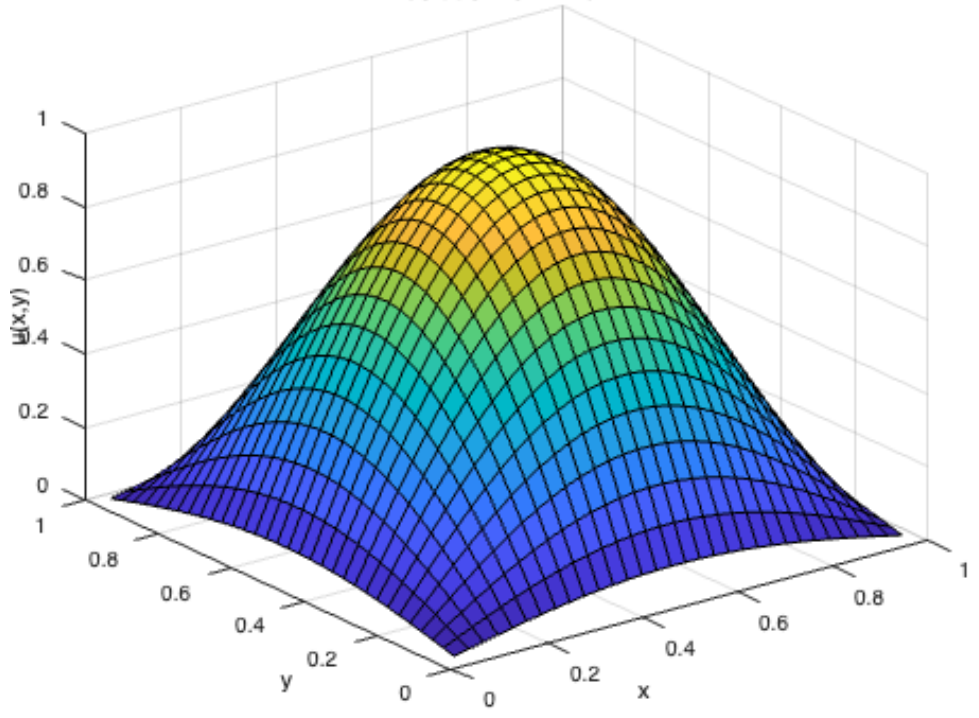
The computed numerical solutions $u(x, y)$ for increasing N show a clear convergence pattern toward the analytical solution $u(x, y) = sin(\pi x)sin(\pi y)$. Below are the surface plots of the solutions for selected grid sizes:

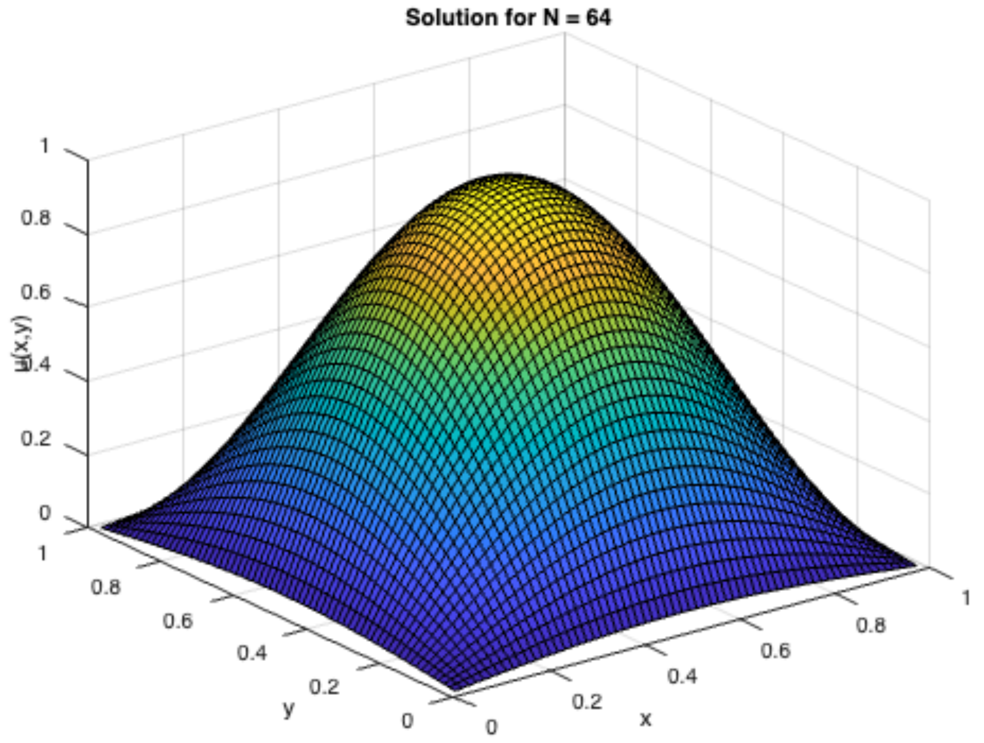


Solution for N = 16

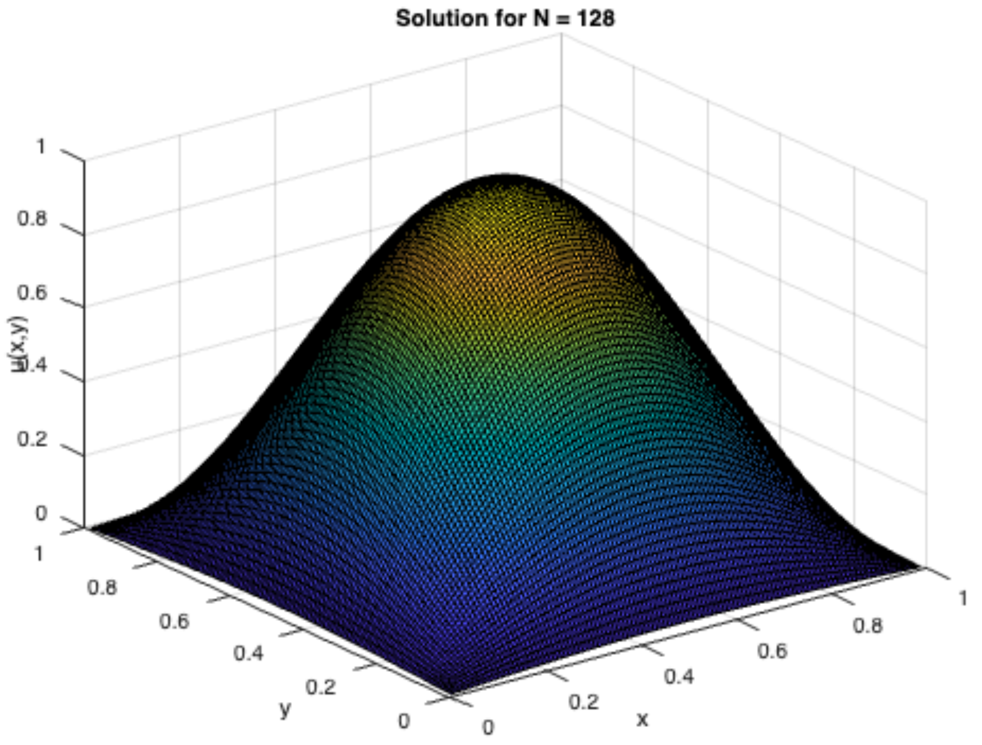


Solution for N = 32

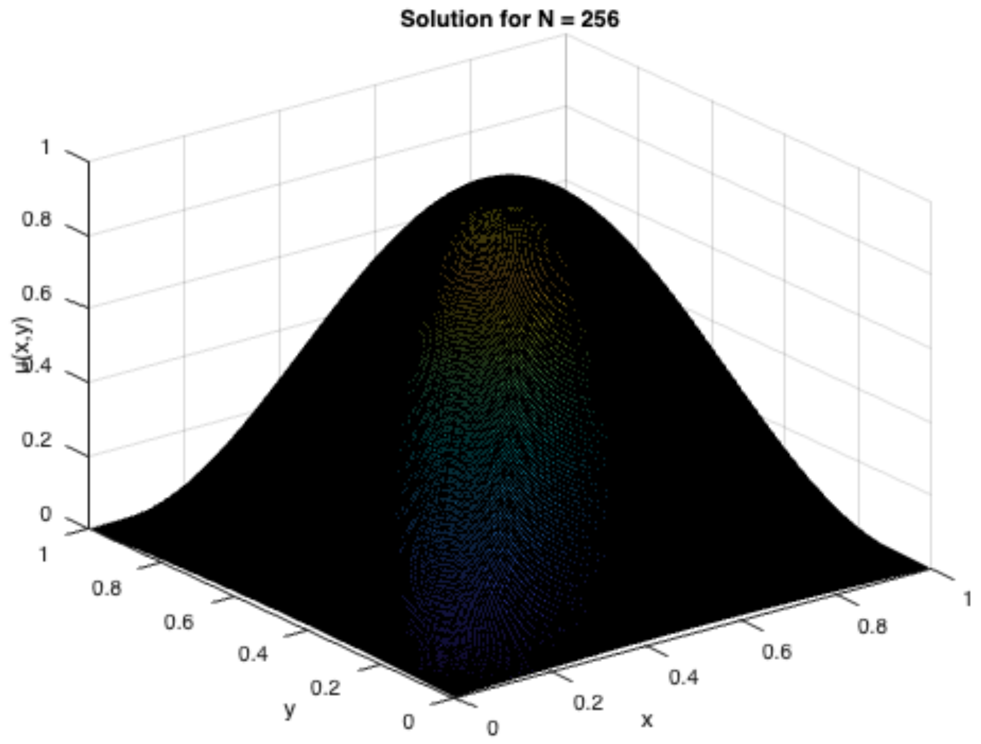




Solution for N = 64
u(x,y)
1
0.8
0.6
0.4
0.2
0
y
x



Solution for N = 128
u(x,y)
1
0.8
0.6
0.4
0.2
0
y
x



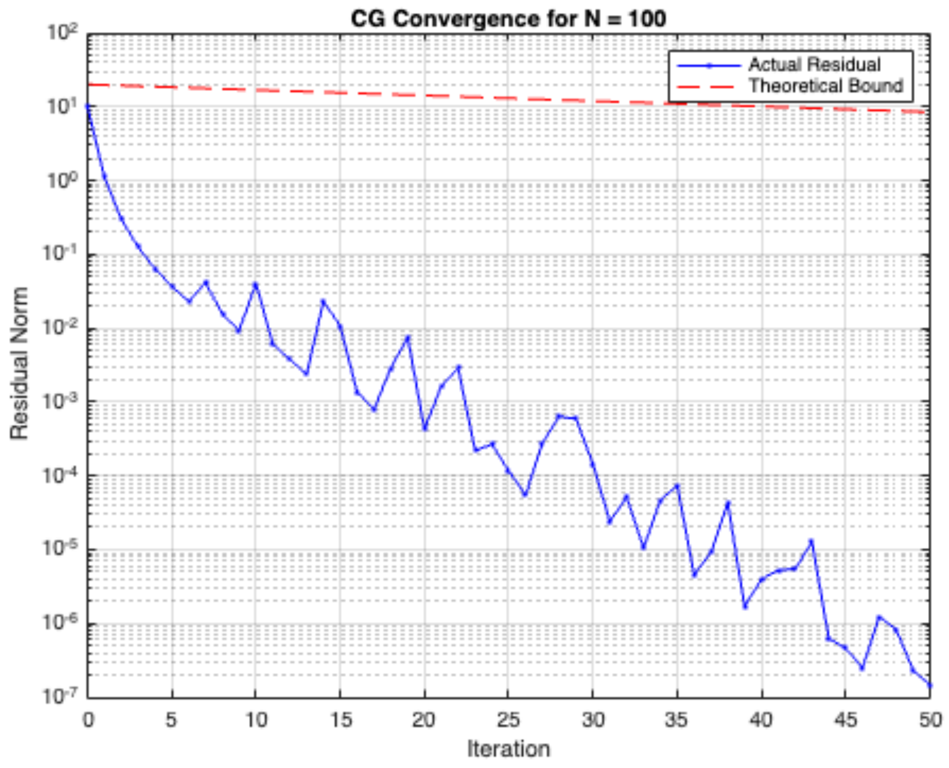
These figures demonstrate that the numerical method accurately captures the expected sinusoidal shape, and the solution becomes smoother and more refined as the resolution increases.

4） Conclusion

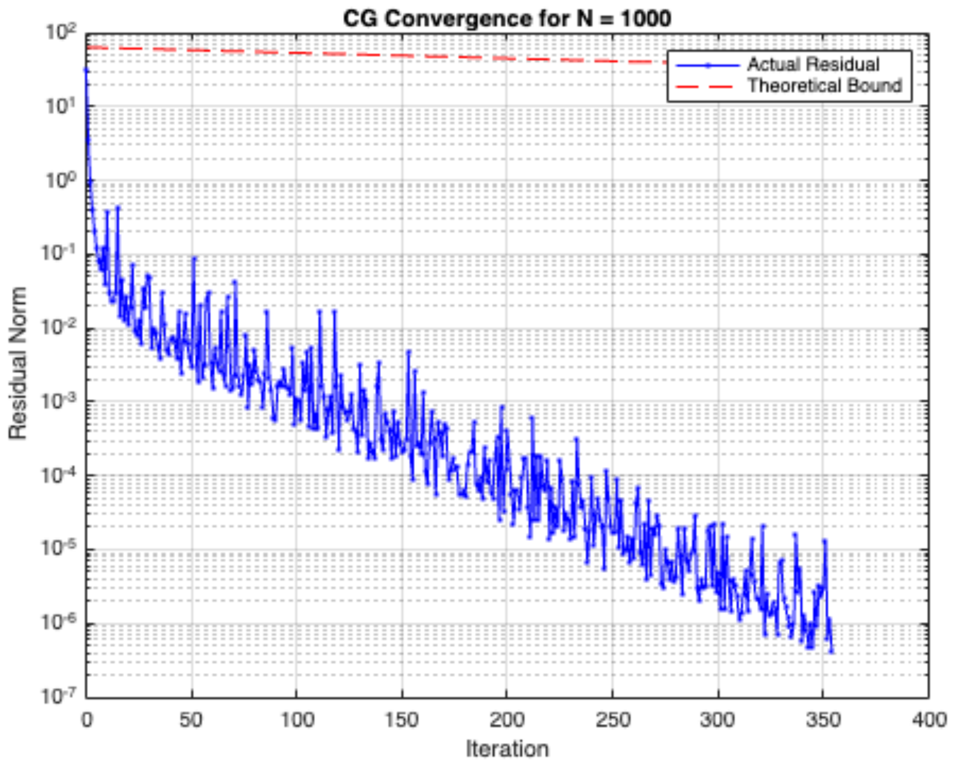
The CG solver performs robustly for solving the sparse linear systems resulting from the discretized Poisson problem. As the system size grows, the number of iterations increases, but the method remains stable and convergent. This confirms that CG is an effective method for solving large SPD systems derived from PDE discretizations.

Part 3.3: Convergence Analysis of CG

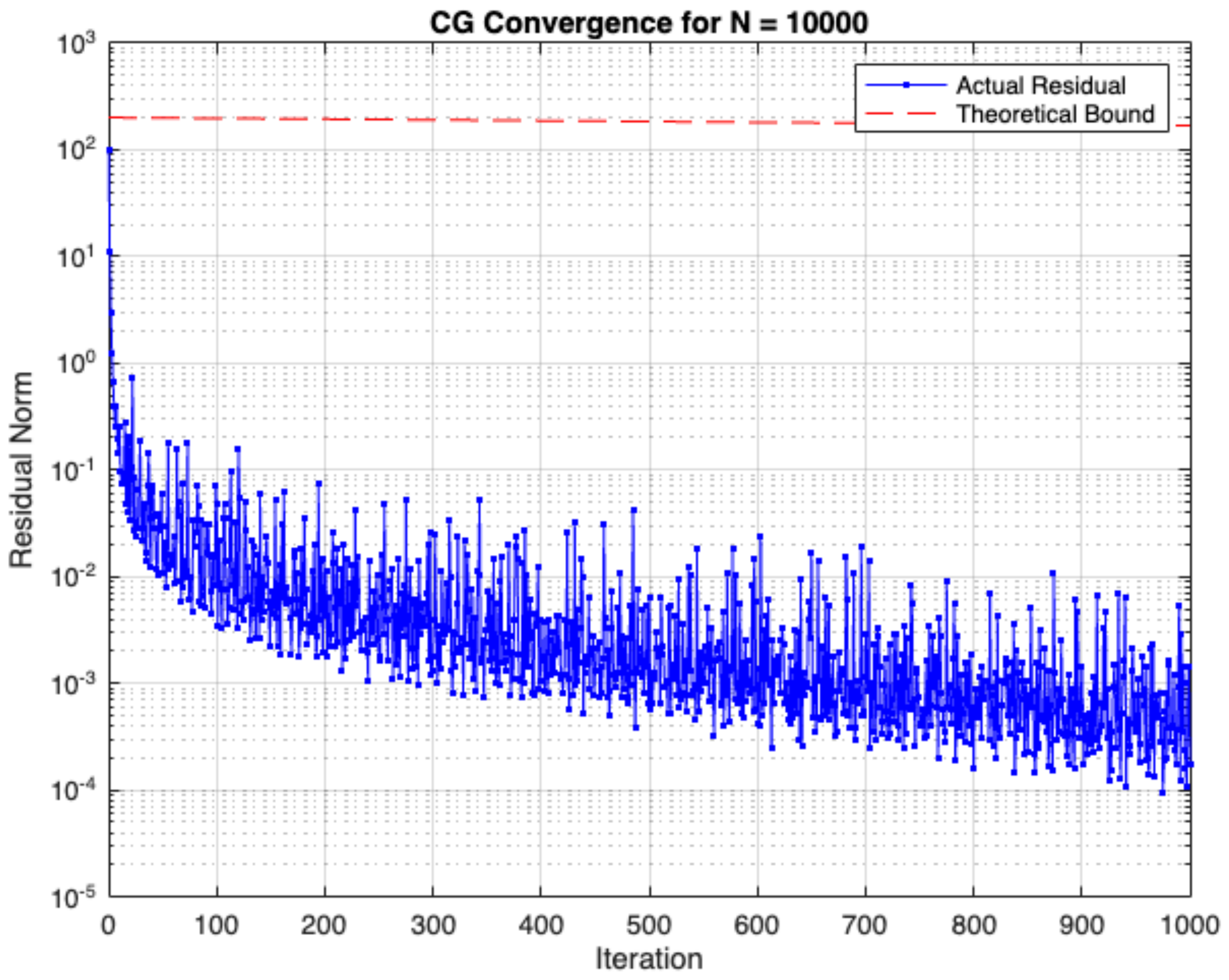
Reruls:



CG Convergence for N = 100
Actual Residual
Theoretical Bound
Residual Norm
Iteration



CG Convergence for N = 1000
Actual Residual
Theoretical Bound
Residual Norm
Iteration



In this part, I applied the Conjugate Gradient (CG) method to solve the dense SPD system $Ax = 1$, where the matrix $A \in R^{N \times N}$has entries $A_{ij} = \frac{N-|i-j|}{N}$. This Toeplitz matrix is symmetric and positive definite, so CG is appropriate.

I tested for N=100, 1000, and 10000. For each case, I recorded the residual norm at each iteration and compared it with the theoretical bound based on the condition number $\kappa(A)$. My implementation avoided redundant computations by reusing vector dot products and computing the matrix-vector product Ap only once per iteration.

As expected, the number of iterations increased with N, due to the growing condition number. However, the **actual residual decay was significantly faster than the theoretical upper bound**, especially for large N. This illustrates the fact that the convergence of CG in practice is often much better than the worst-case estimate.

I did not run the case N=100000 due to memory and computational constraints—storing a dense 100,000×100,000 matrix requires around 80 GB, which exceeded available resources.

CODE :Matlab

`Assignment3.mlx`

Q3.2:

```
lear;clc
```

```matlab
% Conjugate Gradient function
function [x, iter, res_hist] = my_cg(A, b, tol, max_iter)
    x = randn(size(b));
    r = b - A * x;
    p = r;
    res_old = r' * r;
    res_hist = sqrt(res_old);
    iter = 0;

    while sqrt(res_old) > tol && iter < max_iter
        Ap = A * p;
        alpha = res_old / (p' * Ap);
        x = x + alpha * p;
        r = r - alpha * Ap;
        res_new = r' * r;
        beta = res_new / res_old;
        p = r + beta * p;
        res_old = res_new;
        res_hist(end+1) = sqrt(res_new);
        iter = iter + 1;
    end
end
    Ns = [8, 16, 32, 64, 128,256];
    tol = 1e-8;
    max_iter = 10000;
    results = [];


    % Loop over different grid sizes
    for i = 1:length(Ns)
        N = Ns(i);
        h = 1 / (N + 1);
        x = linspace(h, 1 - h, N);
        [X, Y] = meshgrid(x, x);

        % Right-hand side function as specified in assignment
         f = @(x, y) 2*pi^2 * sin(pi*x) .* sin(pi*y);

        b = h^2 * f(X, Y);
        b = reshape(b, [], 1);

        % Build 2D Laplacian matrix using Kronecker product
        e = ones(N, 1);
        T = spdiags([-e 2*e -e], -1:1, N, N);
        I = speye(N);
        A = kron(I, T) + kron(T, I);

        % Start timer
        tic;
        [x_cg, iters, res_hist] = my_cg(A, b, tol, max_iter);
        elapsed = toc;

        results = [results; N, iters, elapsed];
        % Plot solution
        if N <= 256
            figure;
            surf(X, Y, reshape(x_cg, N, N));
            title(['Solution for N = ', num2str(N)]);
            xlabel('x');ylabel('y');zlabel('u(x,y)');
        end
    end

    % Display summary
    disp(array2table(results, 'VariableNames', {'N', 'Iterations', 'Time_sec'}));
```

Q3.3:

```
Ns = [100, 1000, 10000];
tol = sqrt(eps('double'));
max_iter = 1000;

for i = 1:length(Ns)
    N = Ns(i);
    A = toeplitz((N - (0:N-1)) / N);
    b = ones(N,1);
    x = zeros(N,1);
    r = b - A*x; p = r;
    r0_norm = norm(r);
    res_hist = r0_norm;

    for k = 1:max_iter
        Ap = A * p;
        alpha = (r' * r) / (p' * Ap);
        x = x + alpha * p;
        r_new = r - alpha * Ap;
        res_norm = norm(r_new);
        res_hist(end+1) = res_norm;
        if res_norm <= max(tol * r0_norm, 0)
            break;
        end
        beta = (r_new' * r_new) / (r' * r);
        p = r_new + beta * p;
        r = r_new;
    end

    eigvals = eig(A);
    kappa = max(eigvals) / min(eigvals);
    bound = 2 * ((sqrt(kappa)-1)/(sqrt(kappa)+1)).^(0:length(res_hist)-1) * r0_norm;

    figure;
    semilogy(0:length(res_hist)-1, res_hist, 'b.-', ...
             0:length(bound)-1, bound, 'r--');
    legend('Actual Residual', 'Theoretical Bound');
    title(['CG Convergence for N = ', num2str(N)]);
    xlabel('Iteration'); ylabel('Residual Norm');
    grid on;
```

```
end
```