function result = line\_by\_line\_diffusion(nx, ny, tol, max\_iter)

% Domain size

Lx = 1; Ly = 1;

dx = Lx / (nx-1);

dy = Ly / (ny-1);

% As we have to set the Grid points as

x = linspace(0, Lx, nx);

y = linspace(0, Ly, ny);

phi = zeros(nx, ny);

% Boundary conditions

for j = 1:ny

phi(1,j) = 500 \* exp(-50 \* (1 + y(j)^2)); % left

phi(nx,j) = 100 \* (1 - y(j)) + 500 \* exp(-50 \* y(j)^2); % right

end

for i = 1:nx

phi(i,1) = 100 \* x(i) + 500 \* exp(-50 \* (1 - x(i))^2); % bottom

phi(i,ny) = 500 \* exp(-50 \* ((1 - x(i))^2 + 1)); % top

end

% Source term

S\_phi = zeros(nx, ny);

for j = 2:ny-1

for i = 2:nx-1

S\_phi(i,j) = 50000 \* exp(-50 \* ((1 - x(i))^2 + y(j)^2)) ...

\* (100 \* ((1 - x(i))^2 + y(j)^2) - 2);

end

end

% TDMA coefficients for 3 arrays as

a = -1/dy^2 \* ones(nx,1);

b = 2\*(1/dx^2 + 1/dy^2) \* ones(nx,1);

c = -1/dy^2 \* ones(nx,1);

% Line-by-line iteration (Row Sweep)

residual = [];

iter = 0;

tic;

while iter < max\_iter

iter = iter + 1;

old\_phi = phi;

% Row Sweep method

for j = 2:ny-1

d = phi(:, j+1)/dy^2 + phi(:, j-1)/dy^2 + S\_phi(:,j);

d(1) = d(1) + phi(1, j)/dy^2;

d(nx) = d(nx) + phi(nx, j)/dy^2;

phi(:,j) = tdma\_solver(a, b, c, d);

end

% Compute the residual

res = max(max(abs(phi - old\_phi)));

residual = [residual, res];

% Check for convergence

if res < tol

break;

end

end

cpu\_time = toc;

% Plot the contour of the computed phi field

figure;

contourf(x, y, phi', 50, 'LineColor', 'none');

colorbar;

title(['\phi Field (Line-by-Line), Grid: ' num2str(nx) 'x' num2str(ny)]);

xlabel('x');

ylabel('y');

% Display results

fprintf('Line-by-Line: Grid %dx%d, Iterations: %d, CPU Time: %f seconds, Residual: %e\n', ...

nx, ny, iter, cpu\_time, res);

% Return CPU time, iterations, and residual history for comparison

result.cpu\_time = cpu\_time;

result.iterations = iter;

result.residual = residual;

end

function phi = tdma\_solver(a, b, c, d)

% TDMA solver for tridiagonal systems of equations.

n = length(b);

c\_prime = zeros(n, 1);

d\_prime = zeros(n, 1);

c\_prime(1) = c(1)/b(1);

d\_prime(1) = d(1)/b(1);

for i = 2:n

temp = b(i) - a(i) \* c\_prime(i-1);

c\_prime(i) = c(i) / temp;

d\_prime(i) = (d(i) - a(i) \* d\_prime(i-1)) / temp;

end

phi = zeros(n, 1);

phi(n) = d\_prime(n);

for i = n-1:-1:1

phi(i) = d\_prime(i) - c\_prime(i) \* phi(i+1);

end

end

% Parameters

tolerance = 1e-6;

max\_iterations = 10000;

% Run the function for 41x41, 81x81, and 161x161 grids

grids = [41, 81, 161];

cpu\_times = zeros(size(grids));

total\_points = zeros(size(grids));

iter\_counts = zeros(size(grids));

for k = 1:length(grids)

nx = grids(k);

ny = grids(k);

result = line\_by\_line\_diffusion(nx, ny, tolerance, max\_iterations);

cpu\_times(k) = result.cpu\_time;

total\_points(k) = nx \* ny;

iter\_counts(k) = result.iterations;

end

% Plot CPU run time vs total number of grid points

figure;

plot(total\_points, cpu\_times, '-o');

xlabel('Total Number of Grid Points');

ylabel('CPU Run Time (seconds)');

title('CPU Run Time vs Total Number of Grid Points (Line-by-Line)');

grid on;

% Plot residual vs. iterations for both methods

figure;

plot(1:length(result.residual), log10(result.residual), '-o', 'DisplayName', 'Line-by-Line');

hold on;

xlabel('Iteration');

ylabel('Log10(Residual)');

title('Residual vs Iterations (161x161 grid)');

legend('Location', 'northeast');

grid on;