A Comprehensive Report on 3D PDE Solver Codes and Multigrid Methods

Jose Parra

August 8, 2025

Contents

Chapter 1

Introduction

This report presents a detailed discussion of a suite of MATLAB codes developed for solving three-dimensional Partial Differential Equations (PDEs) with a focus on the Poisson equation and similar elliptic problems. All components have been adapted to the 3D case and include:

- A 3D iterative solver (Gauss–Seidel and SOR)
- A 3D multigrid V-cycle implementation (with restriction and prolongation operators)
- A stable time-stepping function for explicit schemes
- Boundary condition enforcement for 3D geometries, with a sample application to a pipe-like domain (one inlet, one outlet)

The report is organized into the following chapters:

1. Background on the 3D Poisson Equation, Discretization, and Multigrid Concepts
2. Core Utility Functions (3D SOR, Gauss–Seidel, and time-step calculation)
3. Residual Computation, 3D Restriction, and Prolongation Operators
4. The 3D Multigrid V-cycle and Poisson Solver Driver
5. 3D Boundary Condition Enforcement (for scalar fields and velocity in a pipe-like geometry)
6. Conclusions and Further Work

Chapter 2

Background: 3D PDE Discretization and Multigrid Concepts

2.1 The 3D Poisson Equation

The 3D Poisson equation is given by

$$\nabla^2 \phi(x, y, z) = f(x, y, z),$$

which, when discretized on a uniform grid with spacing h in all three directions, can be approximated using a 7-point stencil:

$$\frac{\phi_{i+1,j,k} - 2\phi_{i,j,k} + \phi_{i-1,j,k}}{h^2} + \frac{\phi_{i,j+1,k} - 2\phi_{i,j,k} + \phi_{i,j-1,k}}{h^2} + \frac{\phi_{i,j,k+1} - 2\phi_{i,j,k} + \phi_{i,j,k-1}}{h^2} = f_{i,j,k}.$$

This discretization yields a large, sparse linear system $A\phi = f$.

2.2 Iterative Methods in 3D

2.2.1 Gauss–Seidel in 3D

The Gauss–Seidel method updates each interior point using the most recent values available. In 3D, the update becomes

$$\phi_{i,j,k} = \frac{1}{6}\Big[\phi_{i+1,j,k} + \phi_{i-1,j,k} + \phi_{i,j+1,k} + \phi_{i,j-1,k} + \phi_{i,j,k+1} + \phi_{i,j,k-1} - h^2 f_{i,j,k}\Big].$$

2.2.2 Successive Over-Relaxation (SOR) in 3D

The SOR method introduces a relaxation factor ω to accelerate convergence:

$$\phi_{i,j,k}^{\text{new}} = (1-\omega)\,\phi_{i,j,k}^{\text{old}} + \frac{\omega}{6}\Big[\phi_{i+1,j,k} + \phi_{i-1,j,k} + \phi_{i,j+1,k} + \phi_{i,j-1,k} + \phi_{i,j,k+1} + \phi_{i,j,k-1} - h^2 f_{i,j,k}\Big].$$

2.3 Multigrid Techniques in 3D

Multigrid methods are among the fastest solvers for PDEs because they efficiently reduce both high- and low-frequency errors. A typical V-cycle consists of:

1. Pre-smoothing (using Gauss–Seidel or SOR).
2. Computing the residual on the fine grid.
3. Restricting the residual to a coarser grid.
4. Recursively solving (or smoothing) on the coarse grid.
5. Prolonging the coarse correction back to the fine grid.
6. Post-smoothing.

In 3D, restriction typically uses 2x2x2 averaging, and prolongation often uses trilinear interpolation (or simple injection) to fill in the finer grid.

Chapter 3

Core Utility Functions (3D Versions)

3.1 mySOR3D.m

3.1.1 Overview

The function `mySOR3D.m` applies the SOR method to update a 3D grid solution of the Poisson equation.

3.1.2 Mathematical Update (3D)

The SOR update in 3D is given by:

$$\phi_{i,j,k}^{\text{new}} = (1-\omega)\,\phi_{i,j,k}^{\text{old}} + \frac{\omega}{6}\Big[\phi_{i+1,j,k} + \phi_{i-1,j,k} + \phi_{i,j+1,k} + \phi_{i,j-1,k} + \phi_{i,j,k+1} + \phi_{i,j,k-1} - h^2\, f_{i,j,k}\Big].$$

3.1.3 Code Listing

Listing 3.1: mySOR3D.m

```
function phi = mySOR3D(phi, f, h, niter)
    global omega; % relaxation factor
    % Determine the interior dimensions (excluding ghost cells)
    M = size(phi,1)-2;
    N = size(phi,2)-2;
    L = size(phi,3)-2;

    for iter = 1:niter
        for k = 2:L+1
            for j = 2:N+1
                for i = 2:M+1
                    oldVal = phi(i,j,k);
                    phi(i,j,k) = (1-omega)*oldVal + (omega/6)* ( ...
                        phi(i+1,j,k) + phi(i-1,j,k) + ...
                        phi(i,j+1,k) + phi(i,j-1,k) + ...
                        phi(i,j,k+1) + phi(i,j,k-1) - h^2*f(i,j,k) );
                end
            end
```

```
        end
    end
end
```

3.2 myGaussSeidel3D.m

3.2.1 Overview

The 3D Gauss–Seidel method is a special case of SOR with $\omega = 1$.

3.2.2 Code Listing

Listing 3.2: myGaussSeidel3D.m

```
function phi = myGaussSeidel3D(phi, f, h, niter)
    % Determine the interior dimensions
    M = size(phi,1)-2;
    N = size(phi,2)-2;
    L = size(phi,3)-2;

    for iter = 1:niter
        for k = 2:L+1
            for j = 2:N+1
                for i = 2:M+1
                    phi(i,j,k) = ( ...
                        phi(i+1,j,k) + phi(i-1,j,k) + ...
                        phi(i,j+1,k) + phi(i,j-1,k) + ...
                        phi(i,j,k+1) + phi(i,j,k-1) - h^2*f(i,j,k) )/6;
                end
            end
        end
    end
end
```

3.3 calcDt.m

3.3.1 Overview

This function computes a stable time step Δt for an explicit time-stepping scheme in a 3D simulation. The same CFL ideas apply in 3D as in 2D.

3.3.2 Code Listing

Listing 3.3: calcDt.m

```
function [dt, outputFlag] = calcDt(t, outputTime, a)
    global Re Sc h CFL;
```

```
    max_a = max(abs(a(:))); % maximum velocity in the domain
    dt = CFL * (h / max_a);
    outputFlag = 0;
    if (t + dt > outputTime)
        dt = outputTime - t;
        outputFlag = 1;
    end
end
```

Chapter 4

Residual, Restriction, and Prolongation (3D Versions)

4.1 myResidual3D.m

4.1.1 Overview

In 3D, the residual at each interior point is computed using a 7-point stencil:

$$r_{i,j,k} = \frac{\phi_{i+1,j,k} - 2\phi_{i,j,k} + \phi_{i-1,j,k}}{h^2} + \frac{\phi_{i,j+1,k} - 2\phi_{i,j,k} + \phi_{i,j-1,k}}{h^2} + \frac{\phi_{i,j,k+1} - 2\phi_{i,j,k} + \phi_{i,j,k-1}}{h^2} - f_{i,j,k}.$$

4.1.2 Code Listing

Listing 4.1: myResidual3D.m

```
function r = myResidual3D(phi, f, h)
[nx, ny, nz] = size(phi);
M = nx - 2;
N = ny - 2;
L = nz - 2;
r = zeros(size(phi));
for i = 2:M+1
    for j = 2:N+1
        for k = 2:L+1
            lap = (phi(i+1,j,k) - 2*phi(i,j,k) + phi(i-1,j,k))/h^2 + ...
                  (phi(i,j+1,k) - 2*phi(i,j,k) + phi(i,j-1,k))/h^2 + ...
                  (phi(i,j,k+1) - 2*phi(i,j,k) + phi(i,j,k-1))/h^2;
            r(i,j,k) = lap - f(i,j,k);
        end
    end
end
end
```

4.2 myRestrict3D.m

4.2.1 Overview

This function maps a fine-grid 3D residual (or error) r_h to a coarser grid r_{2h} by averaging over each $2 \times 2 \times 2$ block.

4.2.2 Code Listing

Listing 4.2: myRestrict3D.m

```
function r2h = myRestrict3D(rh)
    % Fine grid dimensions (excluding ghost cells)
    M_f = size(rh,1) - 2;
    N_f = size(rh,2) - 2;
    L_f = size(rh,3) - 2;
    % Coarse grid interior size
    M_c = M_f / 2;
    N_c = N_f / 2;
    L_c = L_f / 2;
    % Allocate coarse grid (including ghost cells)
    r2h = zeros(M_c + 2, N_c + 2, L_c + 2);
    for I = 2:M_c+1
        for J = 2:N_c+1
            for K = 2:L_c+1
                % Map coarse index to fine indices:
                iF = 2*I; jF = 2*J; kF = 2*K;
                r2h(I,J,K) = ( rh(iF-1,jF-1,kF-1) + rh(iF,jF-1,kF-1) + ...
                               rh(iF-1,jF,kF-1) + rh(iF,jF,kF-1) + ...
                               rh(iF-1,jF-1,kF) + rh(iF,jF-1,kF) + ...
                               rh(iF-1,jF,kF)   + rh(iF,jF,kF) ) / 8;
            end
        end
    end
end
```

4.3 myProlong3D.m

4.3.1 Overview

This function prolongs a coarse-grid correction e_{2h} to the fine grid e_h by simply replicating each coarse cell's value into its corresponding $2 \times 2 \times 2$ block. More advanced interpolation (trilinear) can be used but here we show simple injection.

4.3.2 Code Listing

Listing 4.3: myProlong3D.m

```
function eh = myProlong3D(e2h)
```

```
    % Coarse grid interior dimensions:
    M_c = size(e2h,1) - 2;
    N_c = size(e2h,2) - 2;
    L_c = size(e2h,3) - 2;
    % Fine grid interior dimensions:
    M_f = 2 * M_c;
    N_f = 2 * N_c;
    L_f = 2 * L_c;
    % Allocate fine grid (including ghost cells)
    eh = zeros(M_f + 2, N_f + 2, L_f + 2);
    for I = 2:M_c+1
        for J = 2:N_c+1
            for K = 2:L_c+1
                % Map coarse cell to fine block indices:
                iF = 2*(I-1);
                jF = 2*(J-1);
                kF = 2*(K-1);
                cval = e2h(I,J,K);
                % Fill the corresponding 2x2x2 block:
                eh(iF,   jF,   kF)   = cval;
                eh(iF+1, jF,   kF)   = cval;
                eh(iF,   jF+1, kF)   = cval;
                eh(iF+1, jF+1, kF)   = cval;
                eh(iF,   jF,   kF+1) = cval;
                eh(iF+1, jF,   kF+1) = cval;
                eh(iF,   jF+1, kF+1) = cval;
                eh(iF+1, jF+1, kF+1) = cval;
            end
        end
    end
end
```

Chapter 5

Multigrid and Poisson Solver (3D Versions)

5.1 `myMultigrid3D.m`

5.1.1 Overview

This function implements a single V-cycle multigrid method in 3D. It performs pre-smoothing, computes the residual (using a 7-point stencil), restricts the residual to a coarser grid, recursively solves on the coarse grid, prolongs the correction back to the fine grid, and applies post-smoothing.

5.1.2 Algorithm

1. **Pre-smooth** on the fine grid using `myGaussSeidel3D`.
2. **Compute residual** r_h using `myResidual3D`.
3. **Restrict** the residual to a coarse grid using `myRestrict3D`.
4. **Recursively solve** for the coarse-grid error e_{2h} using `myMultigrid3D` (with a doubled mesh spacing $2h$).
5. **Prolong** the coarse error back to the fine grid using `myProlong3D`.
6. **Correct** the fine-grid solution.
7. **Post-smooth** the corrected solution.

5.1.3 Code Listing

Listing 5.1: myMultigrid3D.m

```
function phi = myMultigrid3D(phi, f, h)
    % Determine the interior dimensions
    M = size(phi,1) - 2;
```

```
    N = size(phi,2) - 2;
    L = size(phi,3) - 2;

    % Check for coarsest grid condition (e.g., too few interior points)
    if (M <= 2 || N <= 2 || L <= 2)
        phi = myGaussSeidel3D(phi, f, h, 3);
        return;
    end

    % Pre-smoothing: 2 Gauss--Seidel iterations
    phi = myGaussSeidel3D(phi, f, h, 2);

    % Compute residual on the fine grid
    r_h = myResidual3D(phi, f, h);

    % Restrict the residual to the coarse grid
    r_2h = myRestrict3D(r_h);

    % Initialize coarse-grid error
    e_2h = zeros(size(r_2h));

    % Recursive call on the coarse grid (with grid spacing doubled)
    e_2h = myMultigrid3D(e_2h, r_2h, 2*h);

    % Prolong the coarse-grid error back to the fine grid
    e_h = myProlong3D(e_2h);

    % Correct the fine grid solution
    phi = phi + e_h;

    % Post-smoothing: 2 Gauss--Seidel iterations
    phi = myGaussSeidel3D(phi, f, h, 2);
end
```

5.2 myPoisson3D.m

5.2.1 Overview

This function serves as the high-level driver for solving the 3D Poisson equation by repeatedly performing multigrid V-cycles until the residual norm falls below a specified tolerance or a maximum number of iterations is reached.

5.2.2 Code Listing

Listing 5.2: myPoisson3D.m

```
function [phi, Linf, iter] = myPoisson3D(phi, f, h, nIterMax, tol)
    for iter = 1:nIterMax
        phi = myMultigrid3D(phi, f, h);
        r = myResidual3D(phi, f, h);
        Linf = max(abs(r(:)));
```

```
        if Linf < tol
            fprintf('Converged at iteration %d, residual = %g\n', iter, Linf);
            return;
        end
    end
    fprintf('Reached maximum iterations (%d) with residual = %g\n', iter, Linf);
end
```

Chapter 6

Boundary Condition Functions for 3D

6.1 bcGS3D.m

6.1.1 Overview

The function `bcGS3D.m` enforces homogeneous Neumann boundary conditions on all six faces of a 3D cell-centered scalar field. Ghost cells are updated by copying interior values.

6.1.2 Code Listing

Listing 6.1: bcGS3D.m

```
function phi = bcGS3D(phi)
    s = size(phi);
    M = s(1)-2;
    N = s(2)-2;
    L = s(3)-2;

    % Left and Right boundaries (x-direction)
    for j = 1:s(2)
        for k = 1:s(3)
            phi(1,j,k)    = phi(2,j,k);
            phi(M+2,j,k)  = phi(M+1,j,k);
        end
    end

    % Front and Back boundaries (y-direction)
    for i = 1:s(1)
        for k = 1:s(3)
            phi(i,1,k)    = phi(i,2,k);
            phi(i,N+2,k)  = phi(i,N+1,k);
        end
    end

    % Bottom and Top boundaries (z-direction)
    for i = 1:s(1)
        for j = 1:s(2)
            phi(i,j,1)    = phi(i,j,2);
```

```
            phi(i,j,L+2)  = phi(i,j,L+1);
        end
    end
end
```

6.2 bc_u_3D.m

6.2.1 Overview

The function `bc_u_3D.m` enforces boundary conditions on the axial velocity field u for a 3D pipe-like domain. In this example, the domain is cylindrical with a specified radius R and height H. The boundary conditions are:

- **Cylindrical Wall (No-slip):** For points with $r = \sqrt{x^2 + y^2} \geq R$, set $u = 0$.
- **Inlet (Top Plane, z=H):** Set a specified inflow profile (e.g., parabolic) for $r \leq R$.
- **Outlet (Bottom Plane, z=0):** Enforce a zero-gradient condition.

6.2.2 Code Listing

Listing 6.2: $\mathrm{bc}_{u3}D.m$

```
function u = bc_u_3D(u, t)
% bc_u_3D applies boundary conditions for a 3D pipe domain:
% - Inlet at top (z = H) with a parabolic inflow profile.
% - Outlet at bottom (z = 0) with zero gradient.
% - No-slip on the cylindrical wall (r = sqrt(x^2+y^2) = R).
%
% Global variables:
%   x, y, z : 1D coordinate arrays for the mesh.
%   R       : pipe radius.
%   H       : pipe height.
%   Nx, Ny, Nz: grid sizes in x, y, and z.
%
global x y z Nx Ny Nz

% Define pipe radius and height if not already set:
if isempty(R)
    R = 0.5; % Example radius
end
if isempty(H)
    H = z(Nz); % Top plane corresponds to z(Nz)
end

% --- No-slip on cylindrical wall ---
for i = 1:Nx
    for j = 1:Ny
        r_ij = sqrt(x(i)^2 + y(j)^2);
        if r_ij >= R - 1e-3 % near or outside wall
            for k = 1:Nz
```

```
            u(i,j,k) = 0;
        end
    end
end

% --- Inlet at top plane (z = H) ---
kTop = Nz; % top plane index
for i = 1:Nx
    for j = 1:Ny
        r_ij = sqrt(x(i)^2 + y(j)^2);
        if r_ij <= R
            % Parabolic inflow: U(r) = Umax*(1 - (r/R)^2)
            Umax = 1.0; % Maximum inlet speed (adjust as needed)
            u(i,j,kTop) = Umax*(1 - (r_ij/R)^2);
        else
            u(i,j,kTop) = 0;
        end
    end
end

% --- Outlet at bottom plane (z = 0) ---
kBot = 1;
for i = 1:Nx
    for j = 1:Ny
        r_ij = sqrt(x(i)^2 + y(j)^2);
        if r_ij <= R
            % Zero gradient: u at outlet equals u at next interior point.
            u(i,j,kBot) = u(i,j,kBot+1);
        else
            u(i,j,kBot) = 0;
        end
    end
end
end
```

Chapter 7

Conclusions and Further Work

In this report, we have detailed a complete framework for solving three-dimensional PDEs, particularly the Poisson equation, using iterative methods and multigrid acceleration. The codes have been fully adapted to 3D and include:

- **Iterative Solvers:** 3D versions of Gauss–Seidel and SOR.
- **Residual Computation:** Using a 7-point stencil for 3D grids.
- **Multigrid Operations:** Restriction (via 2x2x2 averaging) and prolongation (via simple injection) in 3D.
- **Multigrid V-cycle:** A recursive 3D V-cycle method (`myMultigrid3D.m`).
- **High-level Driver:** `myPoisson3D.m` orchestrates repeated V-cycles until convergence.
- **Time-stepping:** `calcDt.m` ensures stable time steps.
- **Boundary Conditions:** 3D versions of boundary condition functions for both scalar fields (`bcGS3D.m`) and velocity fields in a pipe domain (`bc_u_3D.m`).

This integrated framework provides a robust and efficient approach to solving 3D PDEs on structured grids with ghost cells and is extendable to more complex geometries or PDE systems such as the Navier–Stokes equations.

7.1 Future Directions

Potential improvements and extensions include:

1. Adapting the codes for non-uniform or adaptive meshes.
2. Incorporating higher-order discretization schemes (e.g., 27-point stencils in 3D).
3. Extending the multigrid framework to unstructured grids or using algebraic multigrid.
4. Parallelizing the code using MATLAB's Parallel Computing Toolbox.
5. Integrating with full fluid dynamics solvers for the Navier–Stokes equations.

Appendix A

Appendix: Sample Driver Script for 3D Poisson Solver

Below is a sample driver script demonstrating how to use these functions in a 3D setting:

Listing A.1: driver3D.m

```
% driver3D.m
clear; clc;

% Define grid parameters
Nx = 64; Ny = 64; Nz = 64;
x = linspace(-1,1,Nx); % example: x from -1 to 1
y = linspace(-1,1,Ny); % example: y from -1 to 1
z = linspace(0,1,Nz); % example: z from 0 to 1 (height)
h = x(2)-x(1);

% Define global variables for boundary conditions
global x y z R H Nx Ny Nz
R = 0.5;        % pipe radius
H = z(end);     % pipe height

% Initialize phi and f on a 3D grid with ghost cells
phi = zeros(Nx+2, Ny+2, Nz+2);
f   = ones(Nx+2, Ny+2, Nz+2); % example right-hand side

% Apply initial boundary conditions for phi
phi = bcGS3D(phi);

% Solve the 3D Poisson equation
nIterMax = 100;
tol = 1e-6;
[phi, Linf, iter] = myPoisson3D(phi, f, h, nIterMax, tol);

fprintf('3D Poisson solver converged in %d iterations with residual = %g\n', iter, Linf);

% Apply velocity BC for a 3D pipe (for demonstration)
% Assume u is the axial velocity component
u = zeros(Nx, Ny, Nz);
t = 0; % current time
```

```
u = bc_u_3D(u, t);
```

Appendix B

References

For further reading and a deeper understanding of the topics covered in this report, please consult:

- Briggs, W. L., Henson, V. E., & McCormick, S. F. (2000). *A Multigrid Tutorial* (2nd ed.). SIAM.
- Saad, Y. (2003). *Iterative Methods for Sparse Linear Systems* (2nd ed.). SIAM.
- Thomas, J. W. (1995). *Numerical Partial Differential Equations: Finite Difference Methods*. Springer.
- Hackbusch, W. (1985). *Multi-Grid Methods and Applications*. Springer.