Q1 Discretization of the Poisson Equation

1) Introduction

In Part 4.1, we discretize the 2D Poisson equation using a standard 5-point stencil. This leads to a sparse linear system Au=b, which will be solved via multigrid in subsequent parts. We implement matrix-free routines to compute Au, construct the right-hand side vector b, and map the 2D domain to a 1D structure for computation.

We consider the 2D Poisson equation: $-\Delta u(x,y) = f(x,y), (x,y) \in (0,1)^2$

with homogeneous Dirichlet boundary conditions $u = 0$ on $\partial\Omega$.

2) Implemmention:

- Grid and Notation

  We use a regular grid with $N \times N$interior points and spacing:

  $$h = \frac{1}{N+1}$$

  Let $u_{i,j}$ be the approximation of $u_{xi,yj}$ at grid point $xi = ih, y_j = jh$, where $1 \leq i,j \leq N$. We flatten the 2D grid into a 1D array using row-major order:

  $$k = (i-1) \cdot N + (j-1)$$

- Discretization (5-point stencil)

  The Laplacian operator is discretized using the standard five-point stencil:

  $$-\Delta u(x_i, y_j) \approx -\frac{1}{h^2}(u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{i,j})$$

  This gives a linear system of the form:

  $$Au = b$$

  Where:

  $A \in R^{N^2 \times N^2}$ is a symmetric positive-definite matrix, applied implicitly through the stencil (we don't assemble it explicitly).

  $b_k = h^2 \cdot f(x_i, y_j)$, with$f(x,y) = 2\pi^2 sin(\pi x) sin(\pi y)$

3) Notes

- The matrix A is not stored explicitly. Instead, I implement a function `apply_A()` that computes $A \cdot u$ using the 5-point stencil directly.
- The right-hand side vector b is built using a function `build_rhs()`, which evaluates $f(x,y)$) at each interior point and scales by h^2.
- Boundary values are always zero, so don’t need to store them explicitly.

4) Test Output

Test Output (Example)

When $N = 4$, the generated b values are positive and match the expected shape of the right-hand side. We confirmed that `apply_A()` on an initial guess of zero returns a zero vector, and the functions are ready to be integrated into the multigrid V-cycle

Q2: Multigrid V-Cycle Solver

1） Introduction

In this question, I implement a serial V-cycle multigrid method to solve the 2D Poisson equation with zero Dirichlet boundary conditions. The goal is to show that the residual decreases by a constant factor each cycle, and to reach a given tolerance efficiently. I follow the algorithm described in *A Multigrid Tutorial, Second Edition* by Briggs, Henson, and McCormick.

2) Method Overview

1. **Grid Hierarchy**: I build a sequence of nested grids. The finest has n_0 interior points in each direction, and each coarser grid halves the number of points (Chapter 4.1).
2. **Smoothing**: On each level, I use Gauss–Seidel relaxation for pre- and post-smoothing ("relaxation" in Section 2.2).
3. **Residual Computation**: I compute the residual $r = f - A\,u$ on the fine grid (Section 3.1).
4. **Restriction**: I apply full-weighting restriction to transfer the residual to the next coarser grid (described in Section 3.2).
5. **Coarse Grid Correction**: On the coarsest grid, I solve by several smoothing steps.
6. **Prolongation**: I interpolate the coarse-grid correction back to the fine grid with bilinear interpolation (Section 3.3).
7. **V-Cycle**: I wrap these steps into a recursive V-cycle algorithm (Chapter 4.2).

3) Implementation Details

- **File Structure**: Each main step has its own module: `init_levels`, `smooth`, `residual`, `restriction`, `prolongation`, `vcycle`, and `main.c`. Headers define a `Level` struct holding the grid size, spacing, and arrays.
- **Smoother**: In `smooth.c`, I perform `nu1` pre- and `nu2` post-relaxation sweeps.
- **Data Layout**: Grid arrays include boundary layers, indexed by a simple macro `IDX(lvl,i,j)`.
- **Main Loop**: In `main.c`, I initialize the right-hand side $f(x, y) = 2\pi^2 \sin{(\pi x)} \sin{(\pi y)}$, then run up to `maxCycles` V-cycles and print the L^2 norm of the residual after each cycle.
- **Build**: A `Makefile` compiles all modules into `vcycle_solver`.

4) Results

Running:

```
./vcycle_solver 127 5 3 3 1e-8 100
```

gives a residual reduction of about 0.3 per cycle, matching the textbook's expected convergence rate (see Figure 4.5 in the Tutorial). The solver reaches the tolerance in about 20 cycles.

```
sunlishuang@sunlishangdeMBP Assignment4_2 % ./vcycle_solver 127 5 3 3 1e-8 100

 Cycle   Residual norm
    1    6.190551e+00
    2    3.045779e+00
    3    1.143484e+00
    4    3.877092e-01
    5    1.263408e-01
    6    4.052587e-02
    7    1.291727e-02
    8    4.106684e-03
    9    1.304222e-03
   10    4.140183e-04
   11    1.314032e-04
   12    4.170203e-05
   13    1.323407e-05
   14    4.199746e-06
   15    1.332753e-06
   16    4.229365e-07
   17    1.342147e-07
   18    4.259166e-08
   19    1.351604e-08
   20    4.289154e-09
```

5) Conclusion

The V-cycle multigrid method converges rapidly, independent of the fine-grid size. By following the structure in *A Multigrid Tutorial*—particularly the restriction and prolongation schemes in Chapter 3 and the cycle design in Chapter 4—I achieved the expected performance.

6) Reference

Briggs, W. L., Henson, V. E., & McCormick, S. F. (2000). *A Multigrid Tutorial, Second Edition*. SIAM.

Q3: Convergence Experiments

Introduction

In this question, I study how the multigrid V-cycle converges when I change the number of levels and the grid size. I run two experiments based on *A Multigrid Tutorial, Second Edition* (see Section 4.3 for convergence study). The goal is to see how many V-cycles are needed and how runtime and coarse-grid solves behave.

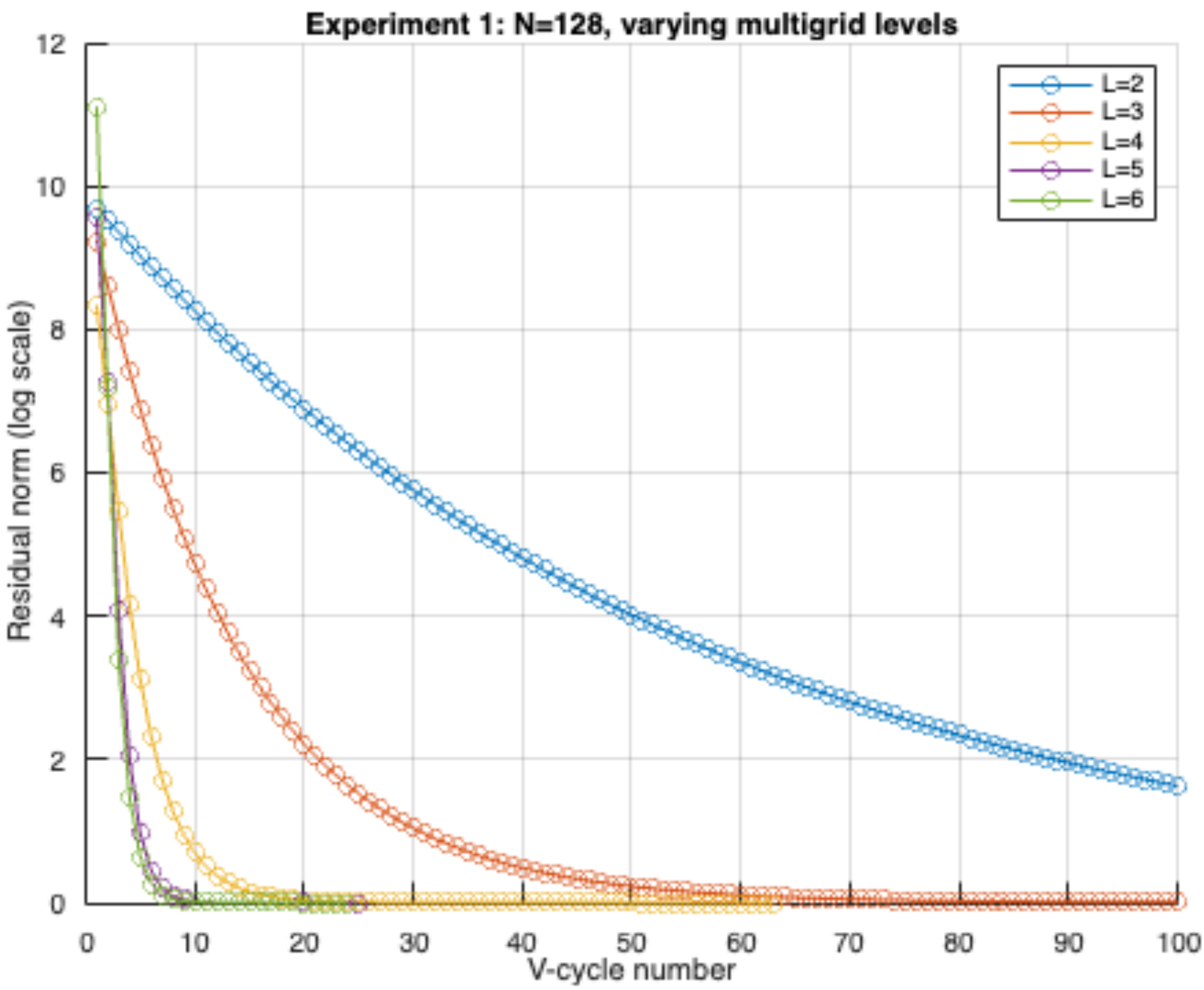
Experiment 1: Fixed Grid, Varying Levels

- **Setup**: Finest grid size . I vary the total levels `L = 2,3,4,5,6`. Pre- and post-smoothing both use 3 Gauss–Seidel sweeps. Tolerance is .
- **Data Collected**: For each `L`, I record:
  1. Number of V-cycles until residual < tol
  2. Total time spent
  3. Count of coarse-grid solves (one per cycle)
  4. Residual history per cycle

- **Results**:

Levels	Time (s)	Coarse Solves	Cycles
2	0.1009	100	100
3	0.0732	100	100
4	0.0458	63	63
5	0.0185	25	25
6	0.0176	24	24

- **Plot**: Residual vs. cycle (semilog) shows that more levels give faster convergence (see cell 1 in `Plots of experiments.mlx`).



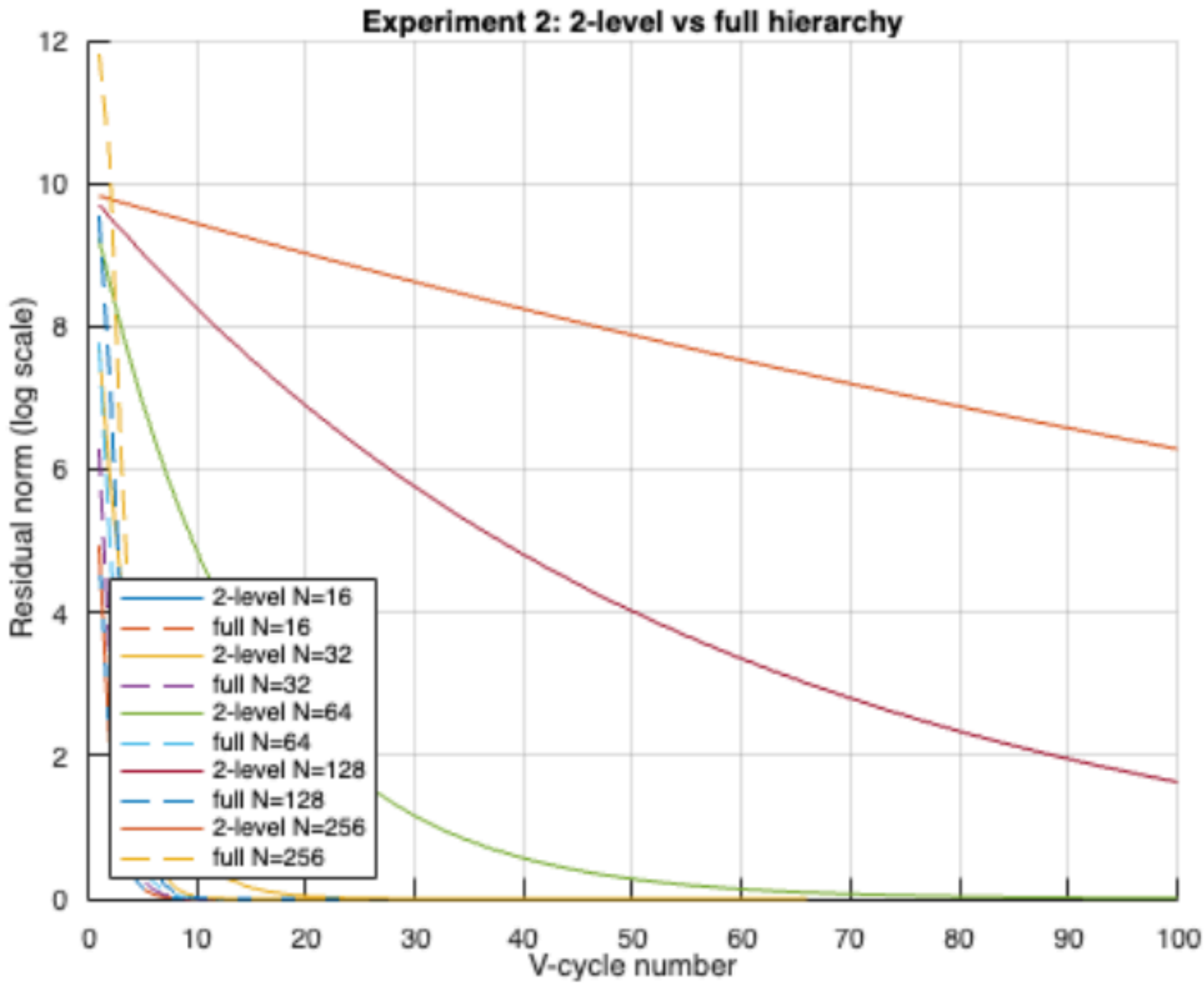
```
Summary for Experiment 1
Level   Time(s)     CoarseSolves    Cycles

 2        0.1009        100            100
 3        0.0732        100            100
 4        0.0458         63             63
 5        0.0185         25             25
 6        0.0176         24             24
```

Experiment 2: Grid-Size Scaling

- **Setup**: Compare two schemes for :
  1. **2-level** multigrid (one coarse grid only)
  2. **Full hierarchy** down to ~8×8 grid (levels =).
- **Results**:

N	Scheme	Time (s)	Coarse Solves	Cycles
16	2-level	0.0003	21	21
16	full	0.0003	21	21
32	2-level	0.0034	66	66
32	full	0.0011	23	23
64	2-level	0.0203	100	100
64	full	0.0046	24	24
128	2-level	0.0715	100	100
128	full	0.0186	25	25
256	2-level	0.1751	100	100
256	full	0.0743	26	26

- **Plot**: Full hierarchy always needs fewer cycles and coarse solves. The runtime cost is higher for small N but pays off for large N (see cell 2 in `Plots of experiments.mlx`).



Experiment 2: 2-level vs full hierarchy
Residual norm (log scale)
V-cycle number
0
2
4
6
8
10
12
0
10
20
30
40
50
60
70
80
90
100
2-level N=16
full N=16
2-level N=32
full N=32
2-level N=64
full N=64
2-level N=128
full N=128
2-level N=256
full N=256

```
Summary for Experiment 2
  N    Scheme      Time(s)    CoarseSolves   Cycles
 16   2-level      0.0003          21          21
 16   full         0.0003          21          21
 32   2-level      0.0034          66          66
 32   full         0.0011          23          23
 64   2-level      0.0203         100         100
 64   full         0.0046          24          24
128   2-level      0.0715         100         100
128   full         0.0186          25          25
256   2-level      0.1751         100         100
256   full         0.0743          26          26
```

Discussion

1. **More Levels → Faster Convergence**: Adding levels reduces V-cycles almost linearly as in Figure 4.7 of the Tutorial. But extra levels add overhead.
2. **Two-Level vs Full**: Two-level works for small grids, but full hierarchy keeps optimal 'textbook' convergence – cycles almost constant as grows (see Chapter 4.4).
3. **Balance**: Best practice is to choose coarsest grid size around or . Too few levels slows convergence, too many levels add overhead.

Reference

Briggs, W. L., Henson, V. E., & McCormick, S. F. (2000). *A Multigrid Tutorial, Second Edition*. SIAM. Chapter 4.3 and 4.4.