HIGH PERFORMANCE SCIENTIFIC COMPUTING - LAB 2

Tyler Renken, Nick Rovito

September 6, 2024

Simulation Description

We are given a solid unit cube to compute the temperature distribution within. We define the computational domain as Ω and boundaries as $\partial\Omega = \Gamma$. The governing equations are the conservation of energy equations given as:

$$\frac{DT}{Dt} = \alpha \nabla^2 T + q \tag{1}$$

where T is the temperature profile; alpha is the thermal diffusivity; and q is the volumetric heat generation. In this problem, we consider steady-state thermal conduction with constant material properties and no heat generation. These assumptions reduce the governing equations to:

$$0 = \alpha \nabla^2 T \tag{2}$$

We consider Dirichlet boundary conditions on each of cube's faces, with boundary conditions taking the form:

$$\begin{aligned} &T(\underline{\boldsymbol{x}} = 100) \forall \underline{\boldsymbol{x}} \in \Gamma_1 \\ &T(\underline{\boldsymbol{x}} = 100) \forall \underline{\boldsymbol{x}} \in \Gamma_2 \\ &T(\underline{\boldsymbol{x}} = 100) \forall \underline{\boldsymbol{x}} \in \Gamma_3 \\ &T(\underline{\boldsymbol{x}} = 100) \forall \underline{\boldsymbol{x}} \in \Gamma_4 \\ &T(\underline{\boldsymbol{x}} = 10) \forall \underline{\boldsymbol{x}} \in \Gamma_5 \\ &T(\underline{\boldsymbol{x}} = 0) \forall \underline{\boldsymbol{x}} \in \Gamma_6 \end{aligned} \tag{3}$$

where Γ_1 is the face defined by the $x = 0$ plane; Γ_2 is the face defined by the $x = 1$ plane; Γ_3 is the face defined by the $y = 0$ plane; Γ_4 is the face defined by the $y = 1$ plane; Γ_5 is the face defined by the $z = 0$ plane; and Γ_6 is the face defined by the $z = 1$ plane.

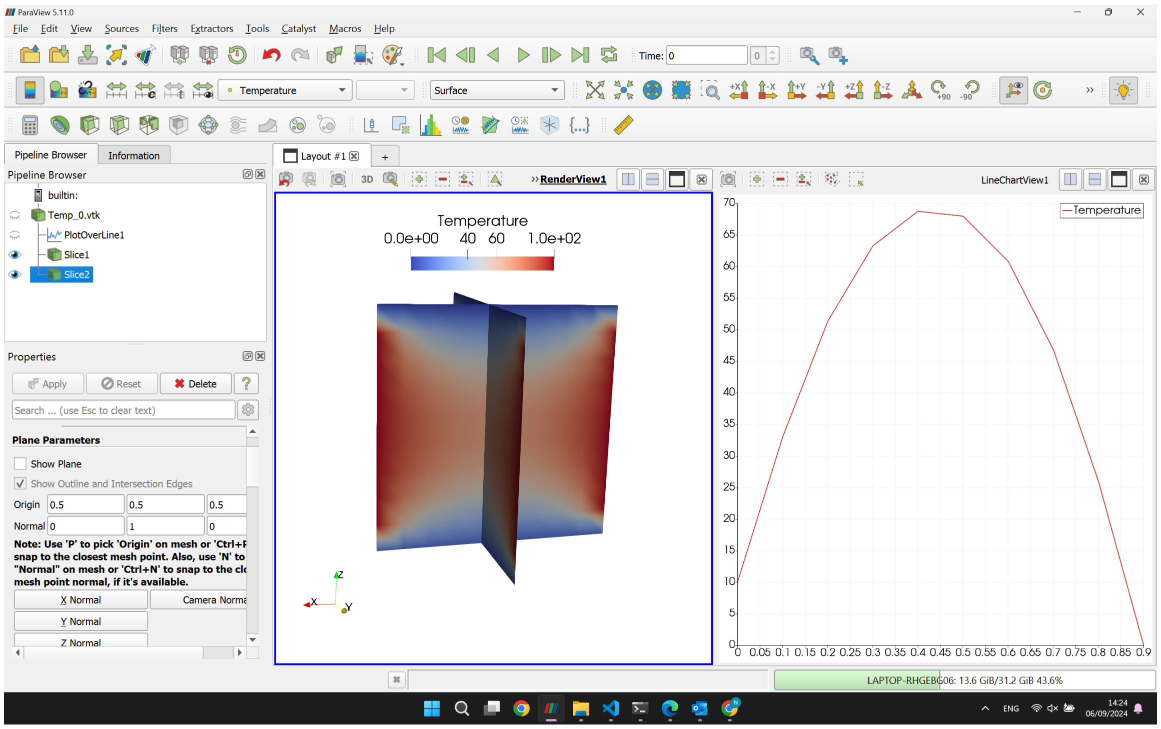
We use a structured mesh, with 10 equally spaced elements in the x, y, and z directions for a total of 1000 elements in the domain.

We use the finite difference method to iteratively solve the energy conservation equations (Eq. 2). The numerica solver is an iterative jacobi solver, a black box solver at this point in the semester.

Results Discussion

The solver converged after 185 iterations. Figure 1 shows the temperature distribution in the interior of the cube (left), defined by x-normal and y-normal plane clips taken at the geometric center of the cube (0.5, 0.5, 0.5). Figure 1 additionally shows the temperature profile at the center of the cube along the z axis (Right). The line we plotted against is defined by the points (0.5, 0.5, 0) and (0.5, 0.5, 1).

We expect a non-centered parabolic temperature profile in a steady sate, constant thermal property, cubic domain. These results agree well with known heat transfer solutions. We expect that as the mesh element diameter approaches zero, the computational profile will become smoother, and the plotted results will become more parabolic.



Plane Parameters

Show Plane

✓ Show Outline and Intersection Edges

Origin 0.5 0.5 0.5

Normal 0 1 0

Note: Use 'P' to pick 'Origin' on mesh or 'Ctrl+P' snap to the closest mesh point. Also, use 'N' to "Normal" on mesh or 'Ctrl+N' to snap to the cl mesh point normal, if it's available.

Figure 1: **Left:** Paraview screenshot of temperature profile in the unit cube. **Right**: Plotted profile on the line defined by points (0.5, 0.5, 0) and (0.5, 0.5, 1).

Self Evaluation

The general program workflow was quickly designed and implemented. We encountered no issues in the implementation of the remaining boundary conditions and interior nodes.

Appendix A: Code

```
//  ================================================================================
//  ||                                                                            ||
//  ||              fd                                                            ||
//  ||              -------------------------------------------                   ||
//  ||              F I N I T E   D I F F E R E N C E                             ||
//  ||                                                                            ||
//  ||              D E M O N S T R A T I O N   C O D E                           ||
//  ||              -------------------------------------------                   ||
//  ||                                                                            ||
//  ||       Developed by: Scott R. Runnels, Ph.D.                                ||
//  ||                     University of Colorado Boulder                         ||
//  ||                                                                            ||
//  ||                For: CU Boulder CSCI 4576/5576 and associated labs          ||
//  ||                                                                            ||
//  ||           Copyright 2024 Scott Runnels                                     ||
//  ||                                                                            ||
//  ||                     Not for distribution or use outside of the             ||
//  ||                     this course.                                           ||
//  ||                                                                            ||
//  ================================================================================

#include "fd.h"

//  ==
//  ||
//  ||      C L A S S:   L A P L A C I A N O N G R I D
//  ||
//  ==

class LaplacianOnGrid
{

public:

  double x0, x1, y0, y1,z0, z1;
  VD x,y,z;
  int ncell_x    , ncell_y   , ncell_z, nField;
  double dx, dy,dz;
  VDD  A;
  VD   phi ;
  VD   b ;

  //  ==
  //  ||
  //  ||  Constructor:  Initialize values
  //  ||
  //  ==

  LaplacianOnGrid(double _x0   , double _x1,  double _y0, double _y1 , double _z0, double _z1, int
    _ncell_x , int _ncell_y, int _ncell_z)
  {

    x0 = _x0;    x1 = _x1;
    y0 = _y0;    y1 = _y1;
    z0 = _z0;    z1 = _z1;

    ncell_x = _ncell_x;
    ncell_y = _ncell_y;
    ncell_z = _ncell_z;
    nField  = ncell_x*ncell_y*ncell_z;
    dx      = (x1-x0)/ncell_x;
    dy      = (y1-y0)/ncell_y;
    dz      = (z1-z0)/ncell_z;

    phi.resize(nField+1);
    b.resize(nField+1);
    A.resize(nField+1);  rLOOP A[r].resize(nField+1);
```

```c

}

//  ==
//  ||
//  ||  Matrix-Free Laplacian
//  ||
//  ==

void FormLS(double *bcs){
  rLOOP cLOOP A[r][c] = 0.;
  rLOOP       b[r]    = 0.;

  // --------------------------------------------------
  // For Lab Only
  // --------------------------------------------------
  //
  // The following line has been inserted so that
  // the code runs even though you have not
  // begun to finish it yet.  Once you have completed
  // the code, this line can be removed.  It places
  // a 1 on the diagonal, just so there are no rows
  // in the matrix that are all zeros.
  //
  rLOOP A[r][r] = 1.;
  //
  // --------------------------------------------------

  // As we're analyzing rectangular prism shaped elements, calculations for the
  // cross section area of each side of an element are below
  double dx2 = dx*dx;
  double dy2 = dy*dy;
  double dz2 = dz*dz;

  // Populate Boundary Conditions

  kLOOP jLOOP { int r = pid(       1,       j,       k )  ; A[r][r] = 1.;  b[r] = bcs[1]; }
  kLOOP jLOOP { int r = pid( ncell_x,       j,       k )  ; A[r][r] = 1.;  b[r] = bcs[2]; }
  iLOOP kLOOP { int r = pid(       i,       1,       k )  ; A[r][r] = 1.;  b[r] = bcs[3]; }
  iLOOP kLOOP { int r = pid(       i,       ncell_y, k )  ; A[r][r] = 1.;  b[r] = bcs[4]; }
  iLOOP jLOOP { int r = pid(       i,       j,       1 )  ; A[r][r] = 1.;  b[r] = bcs[5]; }
  iLOOP jLOOP { int r = pid(       i,       j, ncell_z )  ; A[r][r] = 1.;  b[r] = bcs[6]; }


  // In Lab: Complete the application of boundary conditions for the south-north sides (j = 1 and j =
  ncell_y)
  //           and for the bottom-top sides (k = 1 and k = ncell_z)

  for ( int i = 2 ; i <= ncell_x - 1 ; ++i ){
    for ( int j = 2 ; j <= ncell_y - 1 ; ++j ){
      for ( int k = 2 ; k <= ncell_z - 1 ; ++k ){
        int p = pid(i,j,k); // get the element number
        A[p][ p                    ] = -2./dx2 - 2./dy2 - 2./dz2;
        A[p][ pid( i-1, j   ,  k   ) ] =  1./dx2 ; // divide by the cross sectional area of the
        A[p][ pid( i+1, j   ,  k   ) ] =  1./dx2 ; // appropriate face
        A[p][ pid( i, j+1   ,  k   ) ] =  1./dy2 ;
        A[p][ pid( i, j-1   ,  k   ) ] =  1./dy2 ;
        A[p][ pid( i, j   ,  k+1   ) ] =  1./dz2 ;
        A[p][ pid( i, j   ,  k-1   ) ] =  1./dz2 ;

        // In Lab : Complete the formation of row p of the matrix, for
        //          interior cell at location i,j,k.

      }
    }
  }
}

```

```
  //  ==
  //  ||
  //  ||  Utility routines
  //  ||
  //  ==

  int pid(int i,int j,int k) { return i + (j-1)*ncell_x + (k-1)*(ncell_x*ncell_y); }  // Given i-j, return
     point ID.  Here i-j is the physical grid.

  #include "plotter.h"
  #include "linear_solver.h"

};



//  ==
//  ||
//  ||
//  ||  Main Program
//  ||
//  ||
//  ==

int main(int argc, char *argv[])
{


   int nPEx, nPEy, nCellx, nCelly, nCellz;

   cout << "\n";
   cout << "---------------------------------------------\n";
   cout << "\n";
   cout << " F I N I T E   D I F F E R E N C E         \n";
   cout << " D E M O   C O D E                           \n";
   cout << "\n";
   cout << "---------------------------------------------\n";
   cout << "\n";

   // Default domain size: 1 x 1 x 1 cube

   double lenx = 1.;
   double leny = 1.;
   double lenz = 1.;

   // Default BCs

   double bcs[7];
   bcs[1] = 1.;         bcs[2] = -1.;
   bcs[2] = 1.;         bcs[3] = -1.;
   bcs[4] = 1.;         bcs[4] = -1.;

   // Parse command-line options

   for (int count =  0 ; count < argc; ++count)
     {
       if ( !strcmp(argv[count],"-nCellx") ) nCellx = atoi(argv[count+1]);
       if ( !strcmp(argv[count],"-nCelly") ) nCelly = atoi(argv[count+1]);
       if ( !strcmp(argv[count],"-nCellz") ) nCellz = atoi(argv[count+1]);
       if ( !strcmp(argv[count],"-lenx"  ) ) lenx   = atoi(argv[count+1]);
       if ( !strcmp(argv[count],"-leny"  ) ) leny   = atoi(argv[count+1]);
       if ( !strcmp(argv[count],"-lenz"  ) ) lenz   = atoi(argv[count+1]);
       if ( !strcmp(argv[count],"-bce"   ) ) bcs[1] = atoi(argv[count+1]);
       if ( !strcmp(argv[count],"-bcw"   ) ) bcs[2] = atoi(argv[count+1]);
       if ( !strcmp(argv[count],"-bcs"   ) ) bcs[3] = atoi(argv[count+1]);
       if ( !strcmp(argv[count],"-bcn"   ) ) bcs[4] = atoi(argv[count+1]);
       if ( !strcmp(argv[count],"-bcb"   ) ) bcs[5] = atoi(argv[count+1]);
```

```
        if ( !strcmp(argv[count],"-bct"   ) ) bcs[6] = atoi(argv[count+1]);
      }

   // Set up LaplaciaOnGrid object

   double x0, x1;
   double y0, y1;
   double z0, z1;

   x0 =  0.;   x1 = x0 + lenx;
   y0 =  0.;   y1 = y0 + leny;
   z0 =  0.;   z1 = z0 + lenz;

   LaplacianOnGrid F(x0,x1,y0,y1,z0,z1,nCellx,nCelly,nCellz);

   // Form the linear system

   F.FormLS(bcs);

   // Solve the linear system

   F.SolveLinearSystem(500, F.b , F.phi);

   // Plot the results

   F.plot("Temp",F.phi);

   return 0;

}
```

Appendix B: Math Primer Solutions

1. Find the solution to the following system of equations

$$x_1 + 3x_2 + x_3 = 6$$
$$x_2 - x_3 = -3$$
$$-x_1 - 3x_2 = 12$$

$$x_2 = x_3 - 3$$
$$x_1 + 3(x_3 - 3) + x_3 = 6$$
$$x_1 + 4x_3 = 15$$
$$x_1 = 15 - 4x_3$$
$$-(15 - 4x_3) - 3(x_3 - 3) = 12$$
$$4x_3 - 15 - 3x_3 + 9 = 12$$
$$x_3 = 18$$
$$x_2 = 18 - 3$$
$$x_2 = 15$$
$$x_1 = 15 - 4(18)$$
$$x_1 = -57$$
$$x_1 = -57, x_2 = 15, x_3 = 18$$

2. Find the solution to the following system of equations

$$x_1 - 2x_3 = -1$$
$$-2x_1 + x_2 + 6x_3 = 7$$
$$3x_1 - 2x_2 - 5x_3 = -3$$

$$x_1 = 2x_3 - 1$$
$$-2(2x_3 - 1) + x_2 + 6x_3 = 7$$
$$-4x_3 + 2 + x_2 + 6x_3 = 7$$
$$x_2 + 2x_3 = 5$$
$$x_2 = 5 - 2x_3$$
$$3(2x_3 - 1) - 2(5 - 2x_3) - 5x_3 = -3$$
$$6x_3 - 3 - 10 + 4x_3 - 5x_3 = -3$$
$$5x_3 = 10$$
$$x_3 = 2$$
$$x_2 = 5 - 2(2)$$
$$x_2 = 1$$
$$x_1 = 2(2) - 1$$
$$x_1 = 3$$
$$x_1 = 3, x_2 = 1, x_3 = 2$$

3. Write the system in Problem 1 in matrix-vector notation

$$\begin{bmatrix} 1 & 3 & 1 \\ 0 & 1 & -1 \\ -1 & -3 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 6 \\ -3 \\ 12 \end{bmatrix} \tag{4}$$

4. Write the system in Problem 2 in matrix-vector notation

$$\begin{bmatrix} 1 & 0 & -2 \\ -2 & 1 & 6 \\ 3 & -2 & -5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -1 \\ 7 \\ -3 \end{bmatrix} \tag{5}$$

5. Write out the result of the following matrix vector product

$$\begin{bmatrix} 1 & 0 & -2 \\ 0 & 1 & 1 \\ 1 & 3 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ 2 \end{bmatrix}$$

$$\begin{bmatrix} 1(1) + 0(-1) - 2(2) \\ 0(1) + 1(-1) + 1(2) \\ 1(1) + 3(-1) + 1(2) \end{bmatrix}$$

$$\begin{bmatrix} -3 \\ 1 \\ 0 \end{bmatrix}$$

6. Write out the result of the following matrix vector product

$$\begin{bmatrix} 3 & -2 & 2 \\ 1 & 4 & -2 \\ 2 & -5 & 0 \end{bmatrix} \begin{bmatrix} 2 \\ 4 \\ -1 \end{bmatrix}$$

$$\begin{bmatrix} 3(2) - 2(4) + 2(-1) \\ 1(2) + 4(4) - 2(-1) \\ 2(2) - 5(4) + 0(-1) \end{bmatrix}$$

$$\begin{bmatrix} -4 \\ 20 \\ -16 \end{bmatrix}$$