This chapter discusses aspects of heat conduction. The equilibrium heat conduction on a rod. In this chapter, Arrays will be discussed.

Arrays provide a mechanism for declaring and accessing several data items with only an index. This simplifies the task of variable naming and data management. If we had to assign a name to each of the temperatures, it will provide a very inflexible program and difficult to maintain. It is desirable to write subprograms that accept arrays as arguments, operate on them, and the change be visible at the point of call.

1.1 Heat Conduction

This chapter discusses aspects of equilibrium heat conduction on a rod. At the molecular level, heat energy is caused by the random motion of molecules. When two objects of different temperatures are in contact, the faster moving atoms of the warmer object vibrate against nearby slower moving atoms of the cooler object. This causes the atoms of the cooler object to vibrate more rapidly and in turn effect other slower moving atoms. The motion of molecules results in the warmer objects losing energy and declining in temperature, and the cooler object gaining energy and increasing in temperature. Heat always transfers from warmer objects to cooler objects. This method of heat transfer is called conduction.

The heat energy of a section of the rod changes in time only due to heat energy flowing across the edges of the section of the rod. The conservation of heat energy states that the rate of change of the heat energy in time is equal to the heat energy flowing across the boundaries per unit time.

Materials that are able to conduct heat are called conductors. Not all materials conduct heat at the same rate. Metals are good conductors because they transfer heat quickly, and air is a poor conductor because it transfers heat slowly. Materials that conduct heat poorly are called insulators, and materials that totally prevent heat flow are called perfect insulators. The coefficient $K$ of heat conduction indicates a material relative rate of conduction compared to silver. Silver has a coefficient of heat conduction of
1. Heat Flow in a Rod

The composition of a material affects its conduction rate. If a copper rod and an iron rod are joined together end to end, and the ends placed in heat sources, the heat will conduct through the copper end more quickly than the iron end because copper has a $K$ value of 92, whereas, iron has a $K$ value of 11.

The heat energy flowing across the section’s boundaries per unit time is the heat flux $\phi(x,t)$. If $\phi(x,t) > 0$ heat is flowing to the right.

Fourier (1768-1830) considered the above properties and summarized them in Fourier’s law of heat conduction.

$$\phi(x,t) = -K \frac{\Delta Temp(x,t)}{\Delta L}$$

Which states that the heat flux ($\phi$) is proportional to the temperature differences per unit length. The proportionality constant is the Thermal conductivity $K$.

1.2 The Model

The physical model is a rod of constant cross sectional area $A$. Without any loss of generality orient the rod in the x direction, from $x = 0$ to $x = L$. The left end of the rod is subjected to a lamp at 472 Kelvin whereas the right end is embedded in ice at 270 Kelvin.

1.2.1 A Section Model

The rod is naturally subdivided in sections, which may be of different material. Conservation of energy implies the heat flowing out of one section is the same as that one flowing into the other section next to it.

$$\phi_{left}A = \phi_{right}A$$

where the heat flux $\phi_{section}$ measures the amount of thermal transfer per unit surface area flowing from this section to the section next it.

We can restate the above equation as:

$$\phi_{left}A - \phi_{right}A = 0.$$ 

Which states that the net change of energy due to heat transfer across the boundary of the sections is zero. Substituting Fourier’s law into the conservation of energy equation yields

$$-K_{left} \frac{\Delta T_{left}}{\Delta L_{left}} A + K_{right} \frac{\Delta T_{right}}{\Delta L_{right}} A = 0$$

which is satisfied for each section boundary.

Replace $\Delta T_{left}(t) = T_{sec}(t) - T_{w}(t)$ and $\Delta T_{right}(t) = T_{e}(t) - T_{sec}(t)$ into the above equation to yield:

$$-K_{left} \frac{T_{sec} - T_{w}}{\Delta L_{left}} + K_{right} \frac{T_{e} - T_{sec}}{\Delta L_{right}} = 0.$$ 

Or equivalently,

$$\frac{K_{left}}{\Delta L_{left}} T_{w} - \left(\frac{K_{left}}{\Delta L_{left}} + \frac{K_{right}}{\Delta L_{right}}\right) T_{sec} + \frac{K_{right}}{\Delta L_{right}} T_{e} = 0.$$
1. Heat Flow in a Rod

We can solve for \( T_c \) to get:

\[
T_c = \frac{K_{left} T_w + K_{right} T_e}{\Delta L_{left} + \Delta L_{right}}.
\]

Consider the homogeneous-uniform problem. The thermal conductivity is given by \( K_o \), and \( \Delta L_{right} = \Delta L_{left} = \Delta L \). Thus,

\[
T_{sec} = \frac{T_w + T_e}{2}.
\]

1.3 Method

The temperature equilibrium equation suggest a method of solution. The equilibrium temperature distribution of the rod can determined by assuming an original distribution and iterating until the temperature is at equilibrium. Use the known values of \( T_e \) and \( T_w \) and use temperature equilibrium equation to determine \( T_{sec} \). The calculations are iterated until an equilibrium heat distribution for the rod is obtained.

Consider a subdivision of the rod into \( N \) equal sections. Then \( \Delta L = \frac{L}{N+1} \). The boundaries of each block, the grid points, are identified by their coordinate \( x_i, i = 1, \cdots, N + 1 \). The heat conduction rates are assigned to each subsection at the half-coordinate location, \( K_i, i = 1, \cdots, N \). The temperatures \( T_i, i = 1, \cdots, N + 1 \) are determined at each boundary point. The boundary conditions specify \( T_1 = 472 \) and \( T_N = 270 \). The unknowns are the temperatures at the grid points: \( T_i, i = 2, \cdots, N \).

Therefore, the Algorithm consist on: Assume a value for \( T_i, i = 2, \cdots, N \) compute a new value as follows:

\[
T_i = \frac{K_{i-1} T_{i-1} + K_i T_{i+1}}{K_{i-1} + K_i}, \quad i = 2, \cdots, N.
\]

In iteration form, we assume we know \( T_i^{(odd)}, i = 1, \cdots \) and calculate \( T_i^{(new)}, i = 1, \cdots, N \) for iterating until convergence to equilibrium. As follows:

Algorithm 1 Equilibrium Heat Iteration:

\begin{itemize}
  \item Loop on \( i \) from \( i = 2 \) to \( N \)
    \[
    T_i^{(new)} = \frac{K_{i-1} T_i^{(odd)} + K_i T_{i+1}^{(odd)}}{K_{i-1} + K_i},
    \]
    \[
    T_i^{(odd)} = T_i^{(new)}
    \]
  \item End Loop on \( i \)
\end{itemize}
Convergence can be tested comparing \( T_i^{(\text{old})} \) with \( T_i^{(\text{new})} \). In other words, we compute the difference in temperatures over each iteration and test whether it is below a predetermined tolerance. Let \( \Delta T_i = T_i^{(\text{new})} - T_i^{(\text{old})} \). Thus we would like to know if \( \text{Maxer} = \max_{2 \leq i \leq N} \Delta T_i \leq \text{Tol} \) where \( \text{Tol} \) is a prescribed tolerance. Thus the above Algorithm can be rewritten as:

\[
\text{Algorithm 2 Equilibrium Heat Iteration:} \\
\text{loop while (Maxer} \geq \text{Tol)}
\]

- \( \text{Maxer} = 0.0 \)

- \( \text{Loop on } i \text{ from } i = 2 \text{ to } N \)

\[
- T_i^{(\text{new})} = \frac{K_{i-1}T_{i-1}^{(\text{old})} + K_iT_{i+1}^{(\text{old})}}{K_{i-1} + K_i},
\]

- \( \text{Maxer} = \max(\text{Maxer}, \|T_i^{(\text{new})} - T_i^{(\text{old})}\|) \)

- \( T_i^{(\text{old})} = T_i^{(\text{new})} \)

- \( \text{End Loop on } i \)

In the next section we discuss concepts in C++ that allow the implementation of Algorithm 2. Array constructs and how they work with counting for loops will be highlighted and void function.

### 1.4 Implementation

A look at the Algorithm 2 reveals that in order to represent the the Temperature and the conductivity in a straight forward manner, it is desirable to to index variables. In the mathematical world variables with indices are known as vectors or arrays. Programming languages have borrowed from the mathematical language to provide the concept of an array which allows the indexing of variables. These provide a way of combining indexed variables into a single name and referred to it by the appropriate index.

Counting for loops in C++ provides the ability to work naturally with arrays. C++ for loops allows the implementation of the loop on \( i \) on Algorithm 2. We still need to be able to represent index variables.

#### 1.4.1 Arrays

Arrays provide a mechanism for declaring several data items with only one name, and accessing each one of them with only an index. This simplifies significantly the task of variable naming and data
management. If we had to assign a name to each of the temperatures, it will provide a very inflexible program and difficult to maintain.

Consider the following program that reads the initial temperature for 5 sections of the rod.

```c
#include <iostream.h>

// This program reads five numbers and then prints them back to the user.

void main()
{
    // program Temper.cpp

    float temp0, temp1, temp2, temp3, temp4;

    cout << "Please enter a temperature" << endl;
    cin >> temp0;
    cout << "Please enter a temperature" << endl;
    cin >> temp1;
    cout << "Please enter a temperature" << endl;
    cin >> temp2;
    cout << "Please enter a temperature" << endl;
    cin >> temp3;
    cout << "Please enter a temperature" << endl;
    cin >> temp4;

    cout << "Here are the numbers you entered:" << endl;
    cout << temp0 << endl;
    cout << temp1 << endl;
    cout << temp2 << endl;
    cout << temp3 << endl;
    cout << temp4 << endl;

} // end program Temper.cpp
```

There are five separate variables used for almost identical purposes in this program. One output statement is used five times, another five virtually identical output statements occur together, as do five virtually identical input statements. The program exhibits the repetition that should be incorporated into a for loop. Unfortunately, this is not possible because the repeated statements use slightly different variables.

An array is an indexed collection of objects. The elements of an array must be of the same type. The subscript that indexes the array is used to access individual elements in the array. Using an array of 5 elements to store the input, instead of five named variables will allow this.

```c
#include <iostream.h>

// This program reads five numbers and then prints them back to the user.

void main()
{
    // program Temper.cpp

    float temp[5];
```
cout << "Please enter a temperature" << endl;
cin >> temp[0];
cout << "Please enter a temperature" << endl;
cin >> temp[1];
cout << "Please enter a temperature" << endl;
cin >> temp[2];
cout << "Please enter a temperature" << endl;
cin >> temp[3];
cout << "Please enter a temperature" << endl;
cin >> temp[4];

cout << "Here are the numbers you entered:" << endl;
cout << temp[0] << endl;
cout << temp[1] << endl;
cout << temp[2] << endl;
cout << temp[3] << endl;
cout << temp[4] << endl;
}

The number 5 in parentheses in the declaration of the array temp signifies that temp is an array of 5 element. Each entry in the array is an storage location and can be treated as an individual variable. float temp[5] declared five variables: temp[0] through temp[4]. In order to access an array element, the index of the desired element is written in parentheses after the array name. Each entry of an array behaves exactly like a variable. Everything known about individual variables applies to elements of single-dimension arrays.

We can now simplify the sample program using for loops.

#include <iostream.h>

// This program reads five numbers and then prints them back to the user.

void main()
{
    // program Temper.cpp

    float temp[5];

    for( int index = 0; index < 5; index++)
    {
        cout << "Please enter a temperature" << endl;
        cin >> temp[index];
    }

    cout << "Here are the numbers you entered:" << endl;
    for( index = 0; index < 5; index++)
    {
        cout << temp[index] << endl;
    }
}

Please note that the program could be changed easily to allow to input 20 temperatures instead of
5. This can be accomplished by changing the array declaration so the array can store 20 items, and changing both for loops so they run from 0 to 19, instead of 4.

**Initialization**

Arrays can be initialized at the time of declaration. The array can be initialized simply by supplying a list of elements at the time of its declaration, one per element. For instance:

```plaintext
float cx[4] = {1.5, 2.5, 3.5, 4.5}
int cindx[5] = {3, 4, 5, 1, 2}
```

The initializer can also figure out the size of an array when omitted by taking it from the number of items of the list.

```plaintext
float cx[] = {0.5, 1.5, 2.5, 3.5, 4.5}  // size is 5
int cindx[] = {3, 4, 5, 0, 1, 2}  //size is 6
```

1.4.2 The Rod

We now are able to write C++ code to implement Algorithm 2.

```c++
int N=20;
float tmpold[N+1], tmpnew[N+1], K[N];
float Maxerr, Tol;
/*
...*/
tmpold[0] = 472.0;
tmpold[20] = 270.0;
do
{
    Maxerr = 0.0;
    for( int ind = 1; ind < N; ind++)
    {
        tmpnew[ind] = (K[ind-1]*tmpold[ind-1] +K[ind]*tmpold[ind+1])/
                       (K[ind-1] + K[ind] );
        Maxerr = max ( Maxerr, abs( tmpnew[ind] - tmpold[ind]));
        tmpold[ind] = tmpnew[ind];
    }
} while (Maxerr >= Tol );
/*
...*/
```

The values for $K$ the heat conductivity, the initial values of temperature, and the tolerance $Tol$ should be received as input.

As we saw before, it is convenient to write individual functions to receive such user input. In fact the calculations for Algorithm 2 should also be done in a subprogram. We discuss how Arrays interact with functions and subprograms in the next section.
Warning

The C++ declaration `float temp[5]` there are a total of five storage locations dedicated to the array. In C++, the lower index is 0 and the higher index is 4(=5-1).

A very common problem encountered by programmers is to attempt to access entries of an array that are not in the range defined for the array. In which case, accessing outside of the array boundaries will most likely corrupt the program data. Its results are completely unreliable. This is a common cause of crashes in programs. Significantly, neither the compiler nor the computer will forewarn the user of such an event.

1.4.3 Subprograms

It is good programming practice to attack problems by subdividing them into simpler parts, solving the parts, and then combining the parts into an overall solution. The concept of void function has been discussed earlier to achieve modularization in C++ programs. Communication from a subprogram can also be achieved by providing it with some references to some of its arguments.

Arrays can be arguments to functions. It is possible to pass a single entry in an array, a section of the array, or the whole array to a subprogram.

Let us consider the implementation of Algorithm 2 as a void function:

```c
void equilib( float tmpold[], float tmpnew[], float K[], Tol, N) {
  // Implementation of Algorithm 2 to obtain the equilibrium temperature.
  // Set side conditions
  tmpold[0] = 472;
  tmpold[N] = 270;

  float Maxerr;
  do {
    Maxerr = 0.0;
    for(int ind = 1; ind < N; ind++) {
      tmpnew[ind] = (K[ind-1]*tmpold[ind-1] + K[ind]*tmpold[ind+1]/
                      (K[ind-1] + K[ind] ));
      Maxerr = max( Maxerr, abs( tmpnew[ind] - tmpold[ind]));
      tmpold[ind] = tmpnew[ind];
    }
  } while (Maxerr >= Tol );
}
```

The first thing to note is that the arrays `tmpold[]` and `tmpnew[]` are not being passed by reference. It is significant in that what is passed by a function is the value of the name of the array. The value of the name of the array is the address of the first element of the array. Therefore, when the called function modifies array elements in its scope, it is actually modifying the array elements in its original memory location.

In the calling program one would call the void function `equilib` by including the statement `equilib(tmp1, tmp2, Kheat, 10**(-3), 20);` in the appropriate place of the program. Before the invocation, the
variable tmp1 has the initial guess of the temperature. After the invocation, the variable tmp1 has the result of Algorithm 2.

1.4.4 Distance Measuring Sensor Lab

1. The description of the Long Distance Measuring Sensor can be found at http://sharp-world.com/ecg. The device name is GP2Y0A02YK. Look at the specification sheet for this device. And measure the coordinates of at least 10 points on Figure 3: Analog Output Object vs Distance of Relative Object corresponding to distances of 15 to 150 cm.

2. Write a program that reads a voltage for the standard input and produces a distance which is to be written to the standard output.
   - Your program should use two arrays to store the distance and voltage coordinates for the Figure.
   - Your program should use linear interpolation to compute the approximate distance.
   - Your program should be save in the cs2503Programs directory in the file named: distanceSensor.cpp.
   - Verify that the results you get are near the curve in Figure 3.

3. Once your program is working correctly modify your program so the input is from a file Voltagetest.dat and the output is to the file ObjectDistance.dat.

4. Copy your files Voltagetest.dat and ObjectDistance.dat into your public_html/cs2503Reports directory.

5. create the report file distanceSensorReport.html which explains the lab and includes relative links to the files Voltagetest.dat and ObjectDistance.dat.

6. Modify your cs2503.html to place a link to distanceSensorReport.html.


1.5 Visualization

At each time interval, the output of the program that solves the Heat in the rod problem is simply the \((x, y)\) pairs where \(x\) is the coordinate of a section and \(y\) is the temperature. This is just lots of numbers. Staring at lots of numbers could get overwhelming when trying to glean any meaning from them. Hence visualization of the data becomes necessary. There are two output procedures in the time_rod_heat program. One outputs the heat coefficients and the other outputs the temperatures.

```cpp
#include <fstream.h>

ostream fout; // output stream fout has not been bounded to a file name!!

/* Subprogram outkof:
   void function with no side effects
   - it changes nothing in the original program!
   Designed for output of pair of data points;
        The Coordinate location of section end points, and
        The Heat Coeffs for each section.
*/
```
void outkof(float x[], float k[], int N)  // Designed to output
{

/* Input Parameters
  x real of dimension(N+1)  // Coordinate location of section ends.
  K real of dimension(N)   // Heat Coeffs for each section.
  N integer               // Number of rod sections.
  End of Input Parameters
*/

// Local variables:
int i;  // i is used for counting
float zero = 0.0;
char fnm[] = "Kvalues.dat";  // 'fnm' is a character string -of length 12.
                           // It will hold the name of the output file.
// End Local variables.

// Bound an output stream to the file named in fnm:
fout.open(fnm);  // output stream fout is bounded to file named
                // in the character string fnm.

// Output is sent to file attached to unit:
fout << x[0] <<", ", zero <<endl;  // Outside of the rod the heat coef 0.
for( i = 0; i < N ; i++ )  // For each section:
{
    fout << x[i] <<", ", k[i]<< endl; // coordinate, coefficient -left end
    fout << x[i+1] <<", ", k[i]<< endl; // coord. and Coeffs. at right end 
}
fout << x[N] <<", ", zero<< endl;  // Outside of the rod the heat coef 0.
// End of Output.

fout.close();  // Close file currently bounded to fout.
cout << "created output file: " << fnm << endl;
}
  // End of subprogram outkof.

There are several things to note in this procedure. The file is made ready to be viewed through matlab.

* ostream fout;  // output stream fout has not been bounded to a file name!!
  /* ...*/
  char fnm[] = "Kvalues.dat";  // 'fnm' is a character string -of length 13.
                              // It will hold the name of the output file.
  /* ...*/
  fout.open(fnm);  // output stream fout is bounded to file named
                   // in the character string fnm.
  /* ...*/
  fout.close();  // Close file currently bounded to fout.

The file name is actually entered through a character variable. This would have permitted the
name to be input from the input line at run time.
The values of the coordinate and the heat coefficient are output as pairs:

```cpp
fout << x[i] << "", " << k[i] << endl; // coordinate, coefficient -left end
fout << x[i+1] << "", " << k[i] << endl; // coord. and Coeffs. at right end
```

One pair for each end of a segment of the rod.

And one pair for both ends of the rod.

```cpp
fout << x[0] << ", " << zero << endl; // Outside of the rod the heat coef 0.
fout << x[N] << ", " << zero << endl; // Outside of the rod the heat coef 0.
```

matlab plot command requires only an x-y pair to display the information.

```cpp
#include <fstream.h>
ostream fout; // output stream fout has not been bounded to a file name!!
/* Subprogram outtmp:
   void function with no side effects
   -it changes nothing in the original program!
   Designed for output of pair of data points;
   The Coordinate location, and
   The Temperature value for each coordinate.
   The current time step number is used to name the output file.
   */
void outtmp( float x[], float temper[], int N, int & timestep)
{
    /* Input Parameters
    temper and X are real of size (N+1) // Temperature and coordinate location.
    N integer // Number of rod sections.
    timestep integer // Counter for the time steps.
    */
    int integ=10;
    // Output will be produced only every integ time steps.
    if ( ( timestep / integ ) * integ == timestep ) // then
    {
        int j = timestep; // Local variable:
        j = 10 + j/integ; // Need a two digit number
        //in order to label output files sequentially.
        int frstdigit, secondigit;
        frstdigit = j/10;
        secondigit = j - frstdigit * 10;
        // Create a unique file name to bound to output stream.
        // "fnm" is a character sting of length 10.
        // It will hold the name of the output file.
        char fnm[] = "temp45.dat"; // locations 45 of the string to be changed.
        fnm[4] = '0' + frstdigit;
        fnm[5] = '0' + secondigit;
        /* First output file name is "temp10.dat",
        next is "temp11.dat", and so on. */
```
Here we tackle the need to output several files from one program.

```cpp
int integ=10;
/*
...
*/
int j = timestep; // Local variable:
j = 10 + j/integ; // Need two digit number to label output files sequentially.
int firstdigit, secondigit;
firstdigit = j/10;
secondigit = j - firstdigit * 10;
```

The int variable `j` represents a two digit sequence of numbers starting from 10. The two digits `firstdigit`, `secondigit` of the int variable `j` are extracted.

The names are created at run time and they include numbers to allow sequencing.

```cpp
// Create a unique file name to bound to output stream.
// 'fnm' is a character sting of length 10.
// It will hold the name of the output file.
char fnm[] = "temp45.dat"; // locations 45 of the string to be changed.
fnm[4] = '0' + firstdigit;
fnm[5] = '0' + secondigit;
/* First output file name is 'temp10.dat',
   next is 'temp11.dat', and so on. */
```

The code allows the creation of a different file name each time the program invokes this void function. Hence `fnm` becomes a character variable that has a unique name labeled sequentially every time the subprogram is invoked. It is used as the name of the output file.

Note the use of single quotes for a single character. "a" is a character string of length 2, a and the end of string character. 'a' is just the character a. C++ allows some arithmetic operations among char variables. Hence, '0' + 5 is the character for 5.
Example 1.1: the file “timeMakeMovie”

1.5.1 Animation

For the creation of MPEG animations the PPM-raw file type is recommended. The image file format can be changed by loading the image into xv and simply saving it with the desired format. Using xv one can generate a sequence of images in hdf, tif, ppm, etc file formats, which were generated with such tools as avs or the scanner.

Assume that there are 15 images in the files example000.pnm, example001.pnm, through example014.pnm. In order to encode an mpeg movie with these files as individual frames, each of the images in the sequence is converted into an file with the ppm-raw format. This is achieved with the xv tool by simply saving images in files with the correct type.

Once this has been accomplished, the individual frames are ready to be encoded into the mpeg animation. This is done through the mpeg_encode command. It requires a parameter file that describes the characteristics of the encoding. Look at the example file timeMakeMovie. The command /local/bin/mpeg_encode timeMakeMovie encodes the sequence of ppm into a temp.mpg animation. The command /local/bin/mpeg_play2 allows viewing of temp.mpg files.

1.6 Alternate Movie Assignment

In this assignment we will practice with the PLOT command of matlab. We will use some of the features of xv to improve the displays of matlab. Since we are unable to use the mpeg1 encoder for this class, we will include the individual frames as inline images in a project report.

1. Change directory into your es2503Programs directory.

2. Save the three files: time_heat_rod.cpp and time_ini_data into your es2503Programs directory.
   The program time_heat_rod.cpp calculates the temperature distribution in a rod as it changes in time. The file time_ini_data provides initial data for the program. Please note each lab section will be given a different set of data.

3. Compile the program:
   % CC -lm time_heat_rod.cpp
   Execute the program with the initial data:
   % a.out < time_ini_data
   It should alert you at the files that it created: Kvalue.dat and a sequence of temp10.dat through temp17.dat or more. We are going to create a plot for each of the temp files with the Kvalues.

4. Start up matlab and use the plot command to plot the pair of values for Kvalue.dat and temp10.dat on the same plot. Please note that the Kvalue and temp10 arrays are not of the same size. Export the plot at temp10.jpg.

5. Use xv to load the plot temp10.jpg, half its size, and save it as temp10.jpg.

6. Use the plot command to plot the pair of values for Kvalue.dat and temp11.dat on the same plot. Use xv to load the plot temp11.jpg, half its size, and save it as temp10.jpg.
7. Repeat these steps for 12 through 17.

8. Now you should have files `temp10.jpg` through `temp17.jpg` in your directory. View them in `xv` to make sure they look right.

9. At this point we would have encode it as an mpeg1 movie. We would include each frame as an in-lined image.


11. In your `public_html/es2503/` directory, create a file `temp_frames.html` which includes all the individual frames `temp10.jpg` through `temp17.jpg` as in-lined images.

12. Change directory to your `public_html/es2503Reports` directory, and create a file named `movie.html` to present your report for this assignment that includes a relative link to your file `temp_frames.html`.

13. Modify your `es2503.html` file to include a link to your report file `movie.html`.

14. Hand in `temp10.ppm` through `temp17.ppm`, `temp_frames.html`, and both the files `movie.html` and your modified `es2503.html` using the `handin` command.

15. Clean up your home directory and any of your public areas to make sure that no files containing your program or portion thereof are accessible by any user.

16. Submit a Journal relating your experiences with this assignment.

### 1.7 Movie Assignment

In this assignment we will practice with the PLOT command of matlab. We will use some of the features of `xv` to “grab” the displays of matlab and to save them as `.ppm` files using the `ppm-raw` format. We will encode the resulting files into an mpeg animation which will be linked into the home pages.

1. Change directory into your `es2503Programs` directory.

2. Save the three files: `time_heat_rod.cpp`, `time_ini_data`, and `time_make_movie` into your `es2503Programs` directory. The program `time_heat_rod.cpp` calculates the temperature distribution in a rod as it changes in time. The file `time_ini_data` provides initial data for the program. Please note each lab section will be given a different set of data.

3. Compile the program:
   
   ```
   % CC -lm time_heat_rod.cpp
   % a.out < time_ini_data
   ```

   Execute the program with the initial data:

   ```
   It should alert you at the files that it created: Kvalue.dat and a sequence of temp10.dat through temp17.dat. We are going to create a plot for each of the temp files with the Kvalues.
   ```

4. Start up matlab and use the plot command to plot the pair of values for `Kvalue.dat` and `temp10.dat` on the same plot. Export the plot at `temp10.jpg`.

5. Use `xv` to load the plot `temp10.jpg`, half its size, and save it as `temp10.ppm` using the `ppm-raw` format.

6. Use the plot command to plot the pair of values for `Kvalue.dat` and `temp11.dat` on the same plot. Use `xv` to load the plot `temp11.jpg`, half its size, and save it as `temp11.ppm`.

7. Repeat these steps for 12 through 17.
8. Now you should have files temp10.ppm through temp17.ppm in your directory. View them in xv to make sure they look right.

9. We are finally ready to encode the movie. Type pwd in a terminal window. The output is your present working directory. Edit the file time_make_movie and replace it into the line that request your present working directory. Make sure that you do not have spurious blanks. Add a blank line at the end of the script.

10. Once the file time_make_movie is saved, issue the command:

    % mpeg_encode time_make_movie
    You should have now a file temp.mpg. You can view it using:
    % mpeg_play temp.mpg

11. Hand in temp10.ppm through temp17.ppm, time_make_movie, and temp.mpg.

12. Copy the file temp.mpg to your public_html/es2503/ directory.

13. Change directory to your public_html/es2503Reports directory, and create a file named movie.html to present your report for this assignment that includes a relative link to your file temp.mpg.

14. Modify your es2503.html file to include a link to your report file movie.html.

15. Hand in both the files movie.html and your modified es2503.html using the handin command.

16. Clean up your home directory and any of your public areas to make sure that no files containing your program or portion thereof are accessible by any user.

17. Submit a Journal relating your experiences with this assignment.