Zachary Keener
SDS 322
Intro to Scientific Computing

Modeling Heat in 2-D Space through Discretization of the Heat Equation Using the Forward in Time, Central in Space (FTCS) Method

Abstract: Using the Forward in Time, Central in Space (FTCS) method, we can discretize the partial differential heat equation in 2 dimensions. By doing this, we can model the heat equation, a continuous function, on a computer, a machine only capable of handling discrete values. This modeling allows us to view distribution of heat through a system, controlling variables such as the thermal diffusivity (α) and constant or changing temperature values (referred to as *hold* in this paper). Heat maps can then be created from the data output by the program to visually represent heat in the system. Given these attributes, this process provides a powerful tool for representing isolated systems in our physical world virtually and with little effort in changing variables.

Forward in Time, Central in Space (FTCS): Used almost solely for solving the heat equation and other parabolic partial equations, the FTCS method is a finite difference method, a first-order method in time, and is explicit in time. In other words, it uses the current state of the system to calculate the later state of the system, using difference equations in which finite differences approximate the derivatives to solve differential equations, which, as mentioned, are by nature continuous. This all can be seen by comparing the following equations:

$$\frac{\delta u}{\delta t} - \alpha\left(\frac{\delta u^2}{\delta^2 x} + \frac{\delta u^2}{\delta^2 y}\right) = 0$$

Figure 1: 2-D Heat Equation

```
u(x,y,t) = u(x,y,t-1) +
a*(u(x-1,y-1,t-1)+u(x,y-1,t-1)+
                    u(x+1,y-1,t-1)+u(x-1,y,t-1)+
                    u(x+1,y,t-1)+u(x-1,y+1,t-1)+
                    u(x,y+1,t-1)+u(x+1,y+1,t-1)-
                    8*u(x,y,t-1))
```

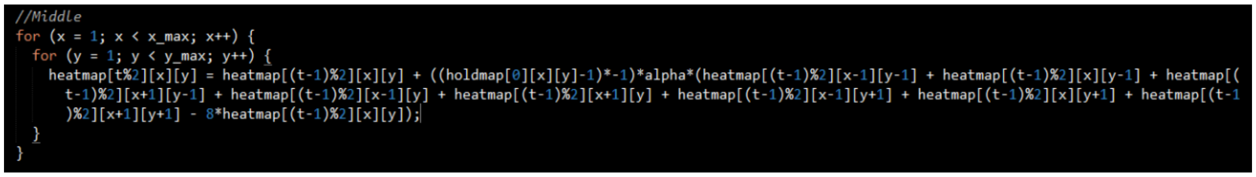
Figure 2: Discretized 2-D Heat Equation Using FTCS

The two equations above represent essentially the same thing: heat in two dimensions. The first is the partial differential equation, and the second is the discretized version of the first, using the FTCS method. Figure 2 takes finite differences approximate to the derivatives (seen in figure 1) to calculate the later of the system; u(x,y,t) is the later state we want to calculate, and u(x,y,t-1) is the current state we will use to calculate the later state. As we move forward in time, we increase the values of *t* to match the discretized location of the later state, using values at *t-1*, the current state, to calculate it.

Input file: Format for the input is as follows:

```
size_x  size_y  alpha  num_timesteps
      x_pos  y_pos  temp  hold
      x_pos  y_pos  temp  hold
                 . . .
```

In practice, the program uses a *double buffer* to switch between these two states, as shown in the equation below:



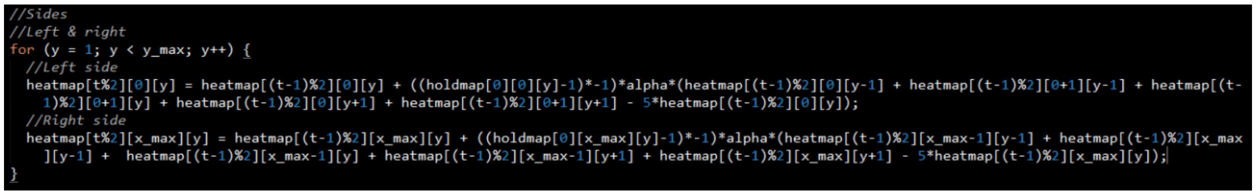
```
//Middle
for (x = 1; x < x_max; x++) {
  for (y = 1; y < y_max; y++) {
    heatmap[t%2][x][y] = heatmap[(t-1)%2][x][y] + ((holdmap[0][x][y]-1)*-1)*alpha*(heatmap[(t-1)%2][x-1][y-1] + heatmap[(t-1)%2][x][y-1] + heatmap[(
      t-1)%2][x+1][y-1] + heatmap[(t-1)%2][x-1][y] + heatmap[(t-1)%2][x+1][y] + heatmap[(t-1)%2][x-1][y+1] + heatmap[(t-1)%2][x][y+1] + heatmap[(t-1
      )%2][x+1][y+1] - 8*heatmap[(t-1)%2][x][y]);
  }
}
```

Figure 3: Practical application of the discretized equation

Zachary Keener
SDS 322
Intro to Scientific Computing

By taking t%2, we can determine the state of the system, dependent upon the current time step, *t*. t begins at 0 and continues the number of steps specified in the input file. As it increases, it allows for the detection of both current and future states, where t%2 and (t-1)%2 will always be different, and (t-1)%2 will always represent the current state, ie the state used to calculate the later state, t%2.

Stencils: In implementing the discretized equation, 3 stencil shapes are required to calculate the temperature values in individual location on the 2-D grid: 4-point stencils for the 4 corners, 6-point stencils for the 4 sides, and a 9-point stencil for the middle locations. The "point" of a stencil represents the number of surrounding locations holding temperature values that may or may not influence the temperature of the location under focus. For example, compare Figure 3 to the following figure:



```
//Sides
//Left & right
for (y = 1; y < y_max; y++) {
  //Left side
  heatmap[t%2][0][y] = heatmap[(t-1)%2][0][y] + ((holdmap[0][0][y]-1)*-1)*alpha*(heatmap[(t-1)%2][0][y-1] + heatmap[(t-1)%2][0+1][y-1] + heatmap[(t-
    1)%2][0+1][y] + heatmap[(t-1)%2][0][y+1] + heatmap[(t-1)%2][0+1][y+1] - 5*heatmap[(t-1)%2][0][y]);
  //Right side
  heatmap[t%2][x_max][y] = heatmap[(t-1)%2][x_max][y] + ((holdmap[0][x_max][y]-1)*-1)*alpha*(heatmap[(t-1)%2][x_max-1][y-1] + heatmap[(t-1)%2][x_max
    ][y-1] +  heatmap[(t-1)%2][x_max-1][y] + heatmap[(t-1)%2][x_max-1][y+1] + heatmap[(t-1)%2][x_max][y+1] - 5*heatmap[(t-1)%2][x_max][y]);
}
```

Figure 4: Calculating right and left sides of the grid, using the 6-point stencil

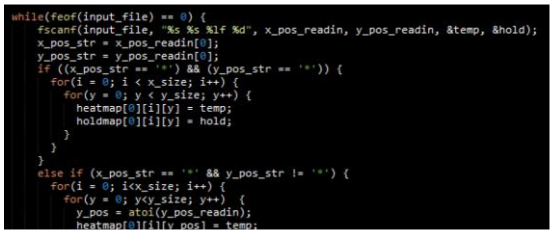
Hold value: Hold values are specified in every line following the first of the input file. They represent whether or not the specified location(s) are influenced by surrounding temperatures, ie if temperature(s) for the specified location(s) will change; 0 indicated a state is open to change, 1 indicated it is not. In practice, this is done by storing the hold value in a separate array, holdmap, from the heat array, and calculated to be either 1 or 0 using (holdmap[0][..][..]-1)*-1). Thus, if hold = 0 at the current location, it will equal -1*-1 = 1. As seen in figures 3 and 4, this adds the calculated new state when hold = 0, and does not when hold = 1.

Structure of code: Structure of the code is characterized by 6 general functions: getXSize, getYSize, getAlpha, getNumTimesteps, readValues, and updateMap. Two header files for C code are used for the I/O and update functions. They are described in detail below:

getXSize, getYSize: store x and y dimension read from the input file into integer variables x_size and y_size, respectively.
getAlpha: stores alpha value, or thermal diffusivity, read from the input file into the double variable alpha.

getNumTimesteps: stores the number of specified time steps, as recorded in the input file, into integer num_timesteps.

readValues: reads in all values following the first line of the input file; feof is used to determine when the end of the file is reached and no more values can be read. readValues detects whether or not a line includes the wildcard sign, *, and deals with it appropriately. In order to do this, the * character is first read into a character

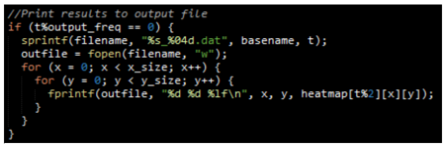


```
while(feof(input_file) == 0) {
    fscanf(input_file, "%s %s %lf %d", x_pos_readin, y_pos_readin, &temp, &hold);
    x_pos_str = x_pos_readin[0];
    y_pos_str = y_pos_readin[0];
    if ((x_pos_str == '*') && (y_pos_str == '*')) {
      for(i = 0; i < x_size; i++) {
        for(y = 0; y < y_size; y++) {
          heatmap[0][i][y] = temp;
          holdmap[0][i][y] = hold;
        }
      }
    }
    else if (x_pos_str == '*' && y_pos_str != '*') {
      for(i = 0; i<x_size; i++) {
        for(y = 0; y<y_size; y++)  {
          y_pos = atoi(y_pos_readin);
          heatmap[0][i][y_pos] = temp;
```

Figure 5: Example of how the readValues function works

array of size 10 for its respected x or y values, then stored in another character value of size 1. By doing this, the char[10] can also store number values up to 10 digits long, and then stored in an integer value, should the value not contain the * wildcard character.

Outputting files: File names are determined using the output_freq variable, whose value is provided as the third command line argument. Output values include x and y coordinates and their corresponding temperature. Method shown in Figure 6.



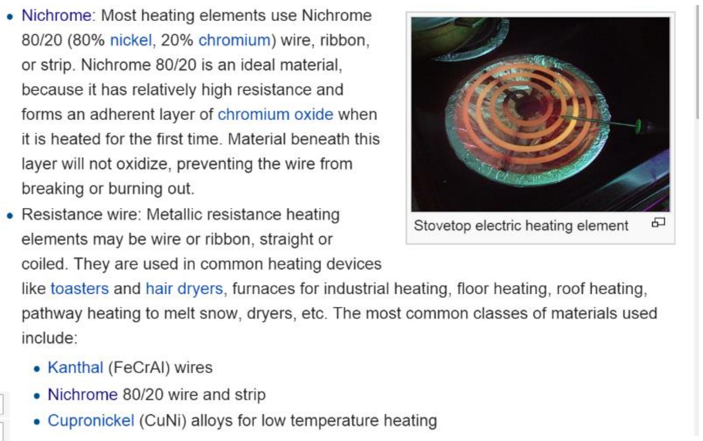
```
//Print results to output file
if (t%output_freq == 0) {
  sprintf(filename, "%s_%04d.dat", basename, t);
  outfile = fopen(filename, "w");
  for (x = 0; x < x_size; x++) {
    for (y = 0; y < y_size; y++) {
      fprintf(outfile, "%d %d %1f\n", x, y, heatmap[t%2][x][y]);
    }
  }
}
```

Figure 6: How file names are determined and data written to output files

Real Application: Simulating the Heating of a Nichrome Electric Stove Top Coil

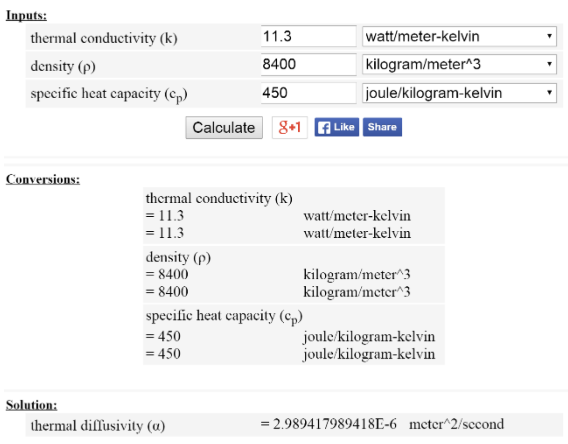
Nichrome is substance commonly used in electric stove top coils. It has a density of 8400 kg/m^3, specific heat of 450 J/(kg*K), and thermal conductivity of 11.3 W/(m*K), giving it a thermal diffusivity of 2.989E-6 m^2/s. This is rounded to 3E-6 when stored in a double type value, alpha. More information can be found in the surrounding figures.

Figure 7: Taken from https://en.wikipedia.org/wiki/Nichrome



- Nichrome: Most heating elements use Nichrome 80/20 (80% nickel, 20% chromium) wire, ribbon, or strip. Nichrome 80/20 is an ideal material, because it has relatively high resistance and forms an adherent layer of chromium oxide when it is heated for the first time. Material beneath this layer will not oxidize, preventing the wire from breaking or burning out.
- Resistance wire: Metallic resistance heating elements may be wire or ribbon, straight or coiled. They are used in common heating devices like toasters and hair dryers, furnaces for industrial heating, floor heating, roof heating, pathway heating to melt snow, dryers, etc. The most common classes of materials used include:
  - Kanthal (FeCrAl) wires
  - Nichrome 80/20 wire and strip
  - Cupronickel (CuNi) alloys for low temperature heating

Stovetop electric heating element



Inputs:

thermal conductivity (k)	11.3	watt/meter-kelvin
density (ρ)	8400	kilogram/meter^3
specific heat capacity (c_p)	450	joule/kilogram-kelvin

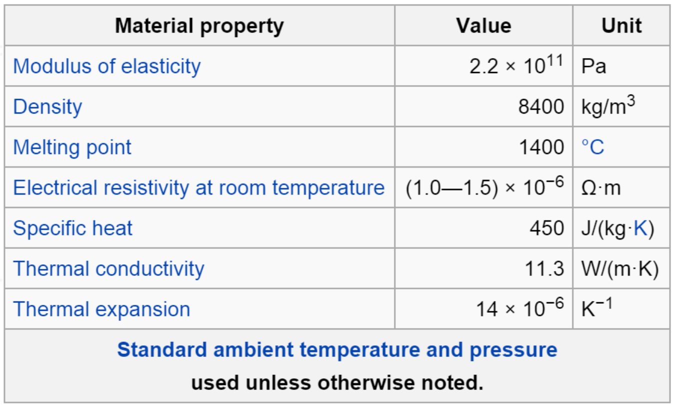
Conversions:

thermal conductivity (k)	
= 11.3	watt/meter-kelvin
= 11.3	watt/meter-kelvin
density (ρ)	
= 8400	kilogram/meter^3
= 8400	kilogram/meter^3
specific heat capacity (c_p)	
= 450	joule/kilogram-kelvin
= 450	joule/kilogram-kelvin

Solution:

thermal diffusivity (α) = 2.989417989418E-6 meter^2/second

Figure 9: Calculation of thermal diffusivity for nichrome; calculator found at http://www.ajdesigner.com/phpthermaldiffusivity/thermal_diffusivity_equation.php



Material property	Value	Unit
Modulus of elasticity	2.2×10^{11}	Pa
Density	8400	kg/m^3
Melting point	1400	°C
Electrical resistivity at room temperature	$(1.0—1.5) \times 10^{-6}$	Ω·m
Specific heat	450	J/(kg·K)
Thermal conductivity	11.3	W/(m·K)
Thermal expansion	14×10^{-6}	K^{-1}

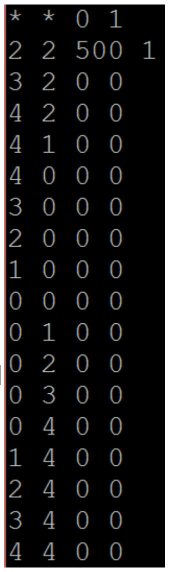
Standard ambient temperature and pressure used unless otherwise noted.

Figure 8: Properties of nichrome, from https://en.wikipedia.org/wiki/Nichrome

Zachary Keener
SDS 322
Intro to Scientific Computing

Input data: Input data for the simulation is as follows:

x_size: 5
y_size: 5
alpha (thermal diffusivity): 3.E-6
num_timesteps: 1,000,000

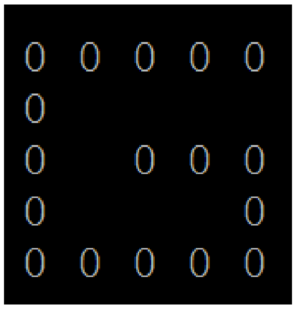


```
* * 0 1
2 2 500 1
3 2 0 0
4 2 0 0
4 1 0 0
4 0 0 0
3 0 0 0
2 0 0 0
1 0 0 0
0 0 0 0
0 1 0 0
0 2 0 0
0 3 0 0
0 4 0 0
1 4 0 0
2 4 0 0
3 4 0 0
4 4 0 0
```

Figure 10: Input values, designed to model a stove top coil

The values are written to represent the spiraling out of heat in a system. With all values in the 5x5 grid but those specified in Figure 10 set to 1, the system has a controlled path on which the hear can disperse. The center is held at a constant temperature values of 500, as would the coil be heated to a constant temperature that spreads throughout the rest of the coil.

The basic design is shown below in Figure 11, where spaces in the 5x5 grid whose hold values equal 0 are specified, and those whose hold values are 1 are left black.



```
0 0 0 0 0
0
0   0 0 0
0       0
0 0 0 0 0
```

Figure 11: Spiral design of coil model

Invoking the executable: To start the program, we can enter the following into terminal, assuming all files are in the current directory:

1. make all

2. For C
   ./p_C coilinput.dat 1000 coiloutput

   For Fortran
   ./p_F coilinput.dat 1000 coiloutput

3. make figures

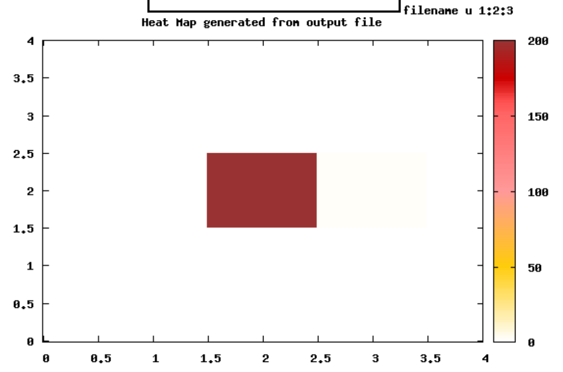
Description of steps: 1. Compiles all necessary files and create C and Fortran executables; 2. Runs the appropriate executable, where coilinput.dat is the input file, 1000 represents the frequency at which the files are created, and coiloutput is the basename of the output files; 3. Creates heat maps for the data create from running the executables, stored in the ./figures directory

Zachary Keener
SDS 322
Intro to Scientific Computing

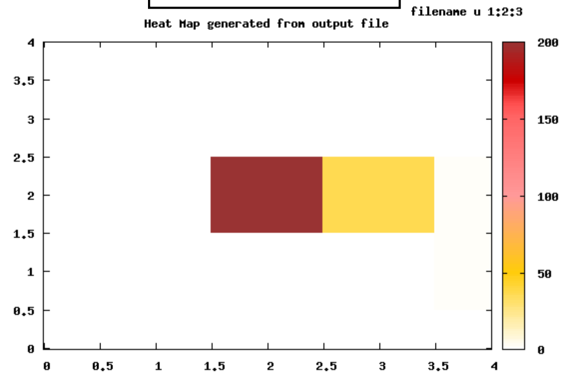
Analyzing the data:

Time steps 1000, 35000, 110000, 250000, 500000, and 1000000 are provided below:

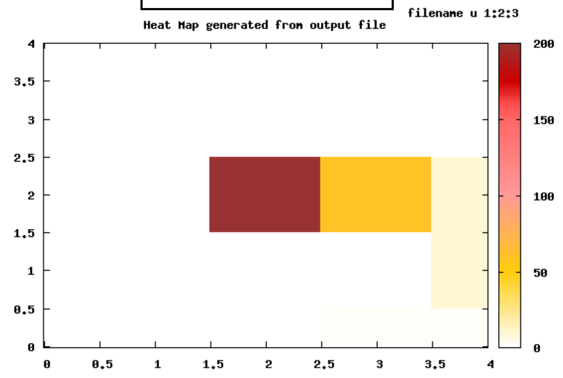
Time step = 1000



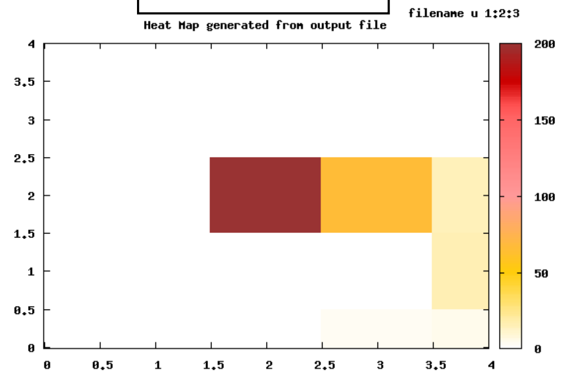
Time step = 35,000



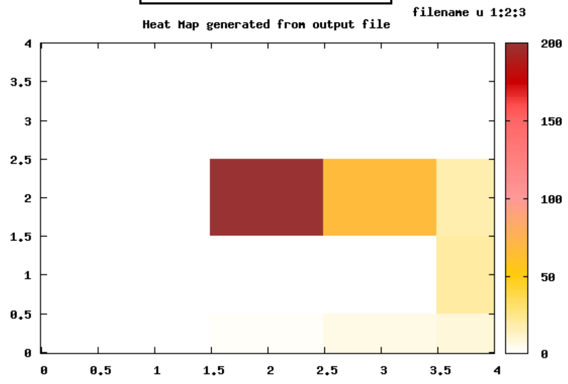
Time step = 110,000



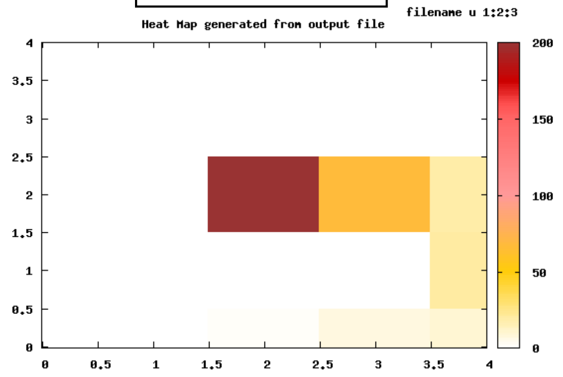
Time step = 250,000



Time step = 500,000



Time step = 1,000,000



Zachary Keener
SDS 322
Intro to Scientific Computing
The heat disperses throughout the "coil" as predicted, however, even at 1,000,000 time steps, only a small fraction of the coil has been reached by the heat. This suggests a limitation to the model that might be improved upon with a larger matrix and a wider coil path (representing larger surface area).

Note: Although not the time steps requested in the project guidelines, these time step measurements were chosen because they are more appropriate for this specific model.

Conclusion: The program is accurate to a degree, but limitations were reached in testing the heat dispersion over a nichrome coil model. Larger paths for the dispersion of heat may result more rapid heat dispersion over a greater distance, however more testing is needed to determine what this looks like and appropriate matrix sizes.