Physics 562 - Computational Physics

Assignment 1: Fourier's law

Josh Fernandes
Department of Physics & Astronomy
California State University Long Beach

February 3, 2014

Abstract

The goal of this assignment is to continue to use Fortran as well as LaTeX in order to understand to use these languages. The assignment focusses on the parameterization of Fourier's law. Fourier's law is calculated for a range of distances and discrete time values. The results are plotted in order to show that as time increases, the peak max flux is lower and the values are more broadly distributed.

1 Introduction

This is the first assignment for the class. The purpose of the assignment is to familiarize ourselves with Fortran,Make file, and LaTeX notation. Fortran dates back to 1953, but it has been updated nine times since it's release. This class focuses on Fortran95, but there are more recent versions of Fortran with another update coming in 2015. This paper solves Fourier's Law, which gives the heat flux of an object Θ in terms of parameters distance x and time t. The x values range from -10<x<10 and the t values are

$$t = 5$$
$$t = 1$$
$$t = .5$$
$$t = .1$$
$$t = .01$$

The results are plotted on the same graph, so the effect of t on the shape of the curve can be determined.

2 The Fortran95 code

The fortran code solves Fourier's law equation. Θ represents the heat flux of the system. It is centered at some x value x_0 . The function also depends on the conductivity of the substance κ and a discrete time constants denoted by t. The formula is

$$\Theta(x,t) = \frac{Q}{2\sqrt{\pi\kappa t}} e^{-\frac{(x-x_0)^2}{4\kappa t}}. \tag{1}$$

The Makefile in Listing 1 describes the structure of the code. The codes in `OBJS1` are compiled together. The `.o` files were created from the `.f95` file by using the `F95` command with flags `F95FLAGS`. In building the executable code `LDFLAGS` were used, which contains the `LIB` library. These `LDFLAGS` are optimal for Intel Core2 architecture and uses the Apple's `vecLib` library from Xcode. By typing `make` we can produce the executable file.

Listing 1: Typical `Makefile`

```
1
2 OBJS = numtype.o heat.0
3
4 PROG = heat
5
6 F95 = gfortran
7
8 F95FLAGS = -O3 -funroll-loops -fexternal-blas
```

```
9
10 LIBS = -framework vecLib
11
12 LDFLAGS = $(LIBS)
13
14 all: $(PROG)
15
16 $(PROG): $(OBJS)
17     $(F95) $(LDFLAGS) -o $@ $(OBJS)
18
19 clean:
20     rm -f $(PROG) *.{o,mod}
21
22 .SUFFIXES: $(SUFFIXES) .f95
23
24 .f95.o:
25     $(F95) $(F95FLAGS) -c $<
```

The Fortran95 code is shown below. The precision is defined in the Module `NumType`. The constants parameters are defined in Module `setup`. The implicit none statement ensures that all the variables has to be defined. `do while` is used to increase the value of x while also calculating Θ for the five different t values. The function `contains` is a way of defining a function that can be reused within the code. Note that the parameters have to be reassigned with the function `tcoeff`.

Listing 2: Module `NumType`

```
1
2 module NumType
3
4     save
5     integer, parameter :: dp = kind(1.d0)
6     real(dp), parameter :: pi = 4*atan(1._dp)
7     complex(dp), parameter :: iic = (0._dp,1._dp)
8
9 end module NumType
```

Listing 3: `heat.f95`

```
1
2 module setup
```

```
3
4       use NumType
5       implicit none
6       real(dp), parameter :: X0 = 0._dp, Xmin = -10._dp, &
7       Xmax = 10._dp, conductivity = 1._dp, &
8       Q = 1._dp
9       real(dp) :: x, dx, t(5)
10      REAL, DIMENSION(:, :), ALLOCATABLE :: result
11      integer :: i, j, DeAllocateStatus, AllocateStatus, &
12      steps
13
14
15  end module setup
16
17  program heat
18
19      use setup
20      implicit none
21
22      t  = (/ 5.0, 1.0, 0.5, 0.1, 0.01/)
23
24      steps = 500
25      x = Xmin
26      dx = (Xmax-Xmin)/steps
27
28      j = 0._dp
29
30      ALLOCATE ( result(size(t), steps), &
31      STAT = AllocateStatus)
32      IF (AllocateStatus /= 0) STOP  &
33      "*** Not enough memory ***"
34
35      do while(x < Xmax)
36          j = j + 1
37          do i=1,5
38              result(i,j) = tcoeff(x,i)
39          end do
40          print *, x, result(1,j), result(2,j), &
41          result(3,j), result(4,j), result(5,j)
42          x = x + dx
```

```
43      end do
44
45      DEALLOCATE (result, STAT = DeAllocateStatus)
46
47      contains
48          function tcoeff(x,i) result(theta)
49
50              implicit none
51
52              real(dp) :: x, theta
53              integer :: i
54
55              theta = Q/(2*sqrt(pi*conductivity*t(i)))*&
56              exp(-(x-X0)**2/(4*conductivity*t(i)))
57
58          end function tcoeff
59
60  end program heat
```

The code is run by typing `./heat`. The results are printed on the screen, or the data can redirect to the file `indisputableknowledge` by typing `./heat > indisputableknowledge.data`. The `Plot` provides the picture in Fig. 1 and Fig. 2.

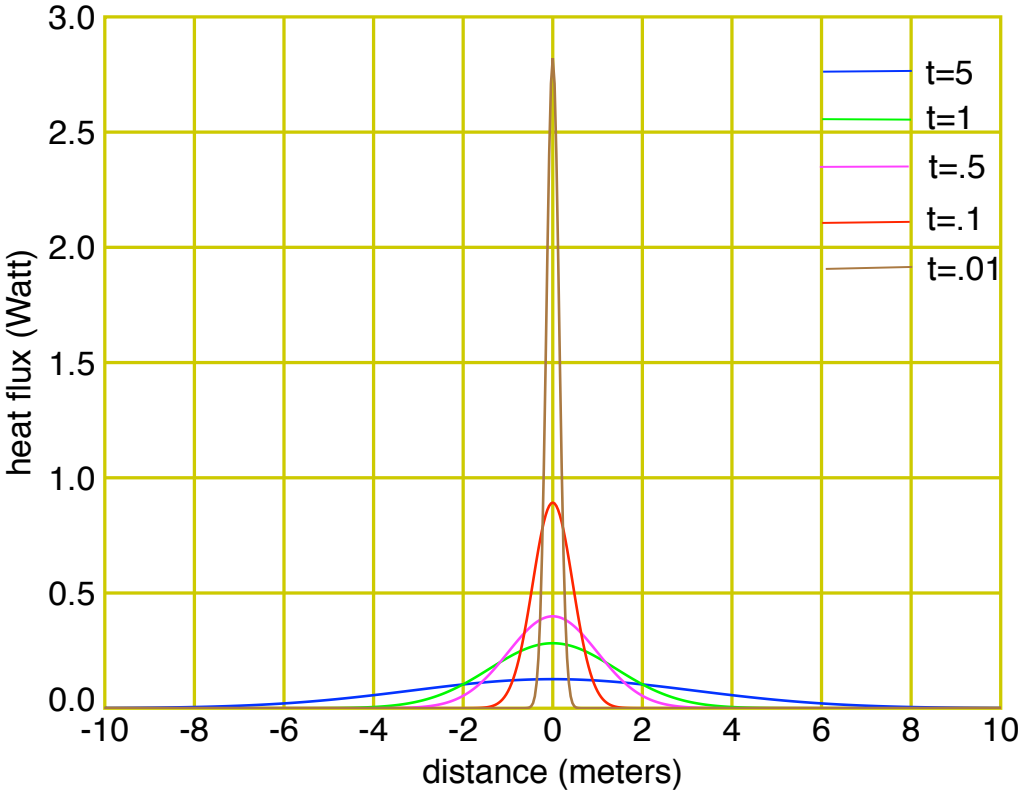

Figure 1: Results of the `heat.f95` code plotted on a linear scale.

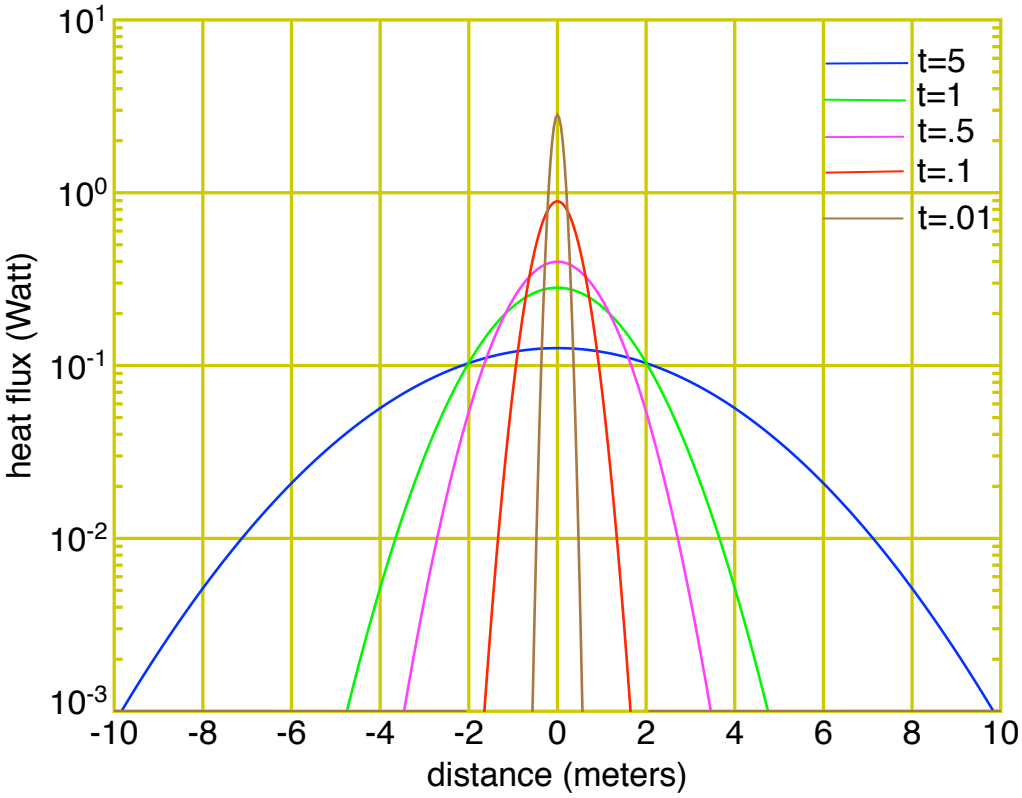

Figure 2: Results of the `heat.f95` code plotted on a log scale.

3 Summary and conclusions

The first Fortran95 code [1] for the class was written. The heat flux as a function of Θ with five different t values were plotted. The graph demonstrates that as t increases the data is more broadly distributed with a smaller maximum heat flux.

References

[1] M. Metcalf, J. Reid and M. Cohen, *Fortran 95/2003 explained*. Oxford University Press, 2004.