Python Modeling of Heat Flow

In a Thermal Storage Device

Tim Hider

Cal Poly University

2011

Abstract

A parabolic dish concentrates sunlight into a thermal storage device may comprise an inexpensive, environmentally benign, clean cooking technology appropriate for developing countries. A Scheffler Solar reflector was constructed and a thermal storage device built to eventually be coupled with the Scheffler. We tested the heat flow in the thermal storage device with an electric heater, and wrote Python code solves the heat diffusion in 1D and 2D in order to model heat flow in the thermal storage device.

Introduction to Experiment

For a couple years Dr. Pete Schwartz has been working with the solar concentration community. For the past year, the focus has been on Scheffler solar reflectors, because they concentrate sunlight through the seasons.

The thermal storage device is made to go with a solar concentrator. Dr. Pete Schwartz has been working with a nonprofit in D.R. Congo *WorkingVillages.org* for over a year now. There, the smoke from their fires poses a big health risk, and the distance they have to walk to acquire the wood for their fires is also a hassle. Consequently this place is as ripe as one can be to test out solar concentration techniques.

But there are social barriers that need to be overcome. The biggest factor for what needs to take place for people to really sink their teeth into solar concentration is that it doesn't get in the way of the well established family and community customs around cooking[3,4]. For many families, this time is traditional and past down across generations. That's why the thermal storage device is of particular importance. Not just because it could allow cooking at night, but because it could potentially allow cooking inside over a more or less standard stove top.

The Scheffler solar concentrator, (see Figure 1) was chosen as the concentration device because it does not need to be moved to change the focal point as the season change, but can instead be bent, and allows single axis tracking of the sun by being placed either directly north, or directly south of the target

Fig 1 Scheffler solar Concentrator

In the summer of 2010, work was done to assemble a Scheffler at Cal Poly. There was a bit of excitement for this solar reflector because it was the first Scheffler to be assembled in North America.

The reflector has been assembled in countries such as India for a while with much success. In the entire rooftop of the public kitchen was covered in the reflectors. In Afghanistan, a system of Schefflers was placed on the Jamhuriat school for girls where it feeds over 1000 students[2].

The numerical modeling of the thermal storage device was performed so that we could predict how decent it would be before putting concrete inside of it (theoretically the most ideal stuff to put in there because it is cheap and has decent thermal conduction).

Modeling was first done in Matlab, but eventually the project was built from the ground up again in python. The heat Sdiffusion equation was the focus. Numerical differential equations are tricky due to differences in syntax in various programs, but the process is intuitive if the user is familiar with the basics of discrete calculus theory. Ultimately though, it was clear that the intuitive process was going to be very difficult by traditional iterative methods, so graph theory had to be used to make it easier to actually model a cyclindrical shape. Properties of graph theory are used to discretize through space. The Crank-Nicholson method is used to step through time.

The thermal storage device is composed of sand and pumice because although these materials are not ideal, they are very cheap. The pumice works as a decent insulator, and the sand works as a rough thermal storage device. The reason modeling on the computer helps, is because we can model, for example, cement, before it would be put in to create a permanent installation of a thermal storage device. A good measure for the thickness that this fairly permanent cement block would ideally be can be gotten from a program to give some justification for pouring it. Some of the experimental work on the Scheffler that would be coupled with the thermal storage is shown in fig. 2 below.

fig 2 experimental work on the Scheffler

Formatted: Font color: Blue

Experimental work

The thermal storage was assembled the same time that the Scheffler Concentrator was. The machine was unwieldy. It's basic structure was chicken wire with tar paper separating the layers. A more detailed description is given in another Cal Poly senior Project[3]. First, sand was to be put on the inner cylinder (figure 1). Before the sand could be filled in though, probing thermal couple wires needed to be stuck in at various points in the device. There were about 12 of them and their position was labeled so a good idea could be gotten of the performance throughout the device.

Greg Chavoor working on the thermal storage device

Once the thermal couples were in place, the sand was dumped in, and pumice was put in on the outside and on top to insulate. The system was monitored by an interface with the thermal couple wires (a DAC) shown below, and an old laptop covered in a plastic box to protect from the weather.

The DAC, interface between the computer and the heating probe wires (thermal couples)

The thermal storage system did not flow heat as well as expected. A few crude models were made in excel that seemed to match the data fairly well. But a more accurate model of heat transfer might illuminate more about what the data was showing us. That was the motivation for modeling the system with the actually diffusion equation involved instead of the simpler models used in the excel program.

The model would help here because if an iron or copper rod were inserted into the sand then that would improve the thermal conductivity, but it's not entirely easy to find out how much. Because copper is expensive, and iron is difficult to shape and carry, finding an ideal thickness that would sufficiently improve the conduction of heat would help a lot.

Work on the concentrator

The concentrator took about four weeks to assemble. The parts were shipped in aluminum bars and instructions were obtained from Wolfgang Scheffler who made the original[6]. The instructions were in metrics, which made it difficult to deal with the ridiculous American units that the parts were ordered in. But a week of conversions later and the process of assembling could begin.

Once the stand was assembled, the mirror itself required a bit more improvisation. Strips of expensive reflecting material which needs to be replaced with cheap material were cut, and then fastened via plastic strips to the cylindrical base. Eventually, the machine could reflect strongly enough to burn wood, and if shone on a storage device could theoretically maintain cooking temperature for several hours. The final result is shown below

Jason Rapp with finished concentrator array

The shape of the mirror is the most critical part of the scheffler construction process. If the shape not perfect, decreases in efficiency come about. The current idea is to attempt to create a basic mold perhaps out of a relatively inexpensive foam, and then lay some fiberglass over that original shape to get a nice dish shape. A way to create an initial mold is the most complicated part of this process, and as of this paper is the current focus of Dr. Schwartz's group working on the Scheffler.

Some improvements to the process could be as mentioned the simplification of the base structure. But also a few pieces from the original instructions needed relatively complex machining to bend into shape. However as a first step to show that it could be done by a handful of people, the schefller assembled in the summer of 2010 was a good step.

Introduction to Python Code

The links to the source code are given in the bibliography section of the paper. The code used to model heat flow are written in Python. Python was chosen because it is open source and relatively easy to use, being relatively similar to C. Also, Python has a library for graph theory, which was used to construct the discretized Laplacian. One of the references has a link to a Python tutorial and download site[1].

The usual way to solve a PDE on a computer program is for spatial discretization in both space and time. Graph theory offers to have a convenient way of expressing the Laplacian, which is the main operator in the heat equation PDE used here. Graph theory is less intuitive but it is a lot faster to take the Laplacian of a graph than to deal with a matrix in normal spatial discretization.

The goal of the code is to model the thermal storage device. For one, we can do some cool science on it that will get people excited and hopefully more enthusiastic about the machine. Hopefully it can also provide a guide for optimal size and materials for a thermal storage device, because the bigger ones really are difficult to make.

Theory for Python Code

The heat equation

$$\frac{du}{dt} = D\Delta u \qquad D = k/c\rho \qquad (1)$$

Is used in one two and three dimensions to model heat flow in sand and pumice, where D is the diffusion constant, k is the thermal conductivity, c is the heat capacity, and rho is the density of the medium.

To convert this equation to code, the crank Nicholson method is used. A reference to a the complete structure of Crank Nicholson method[5]. The Crank Nicholson method takes the left side of equation one and turns it into

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} \qquad (2)$$

Where *i* is the position of the temperature u, and *n* is the time index of the temperature u. The right side however, can be changed into

$$\frac{1}{2}Au_{n+1} + \frac{1}{2}Au_n \qquad (3)$$

Where A is some matrix. Putting equations 2 and 3 together gives, after some rearrangement

$$u_{n+1} = \left(1 - \frac{\Delta t}{2}A\right)^{-1}(1 + \frac{\Delta t}{2}A)u_n \qquad (4)$$

This is the equation that is used to step through time in the code below. A full description of the Cranck Nicholson Method is given in the Bibliography.

Implementation of the 1D code

In this section, we present and describe the 1D model. The code is fairly short, since the graph theory methods are nice and elegant.

Python has useful libraries that can be downloaded for free for students. The NetworkX library has graph theory. The numpy library allows for matrix multiplication and is similar to Matlab in many ways. There is an online resource for converting matlab syntax into numpy.

Next, there are a few of the constants that go in front of the Laplacian for the heat equation. The Laplacian is just a double derivative with respect to position, but here in coding it is applied a lot

differently than is taught in traditional pen and paper math classes on PDEs. A detailed mathematical understanding isn't necessary here, because the networkx library has a discrete Laplacian built in. The Laplacian matrix is built here along with the graph G. The graph G is going to be what our actually physical system looks like, or at least contain the information of how the spatial grid is structured. For now, this is just a rectangular grid, or in the 1D case just a straight line.

Next, The rhs variable just contains the information that will be in front of the heat equation. The D value there will eventually be a diagonal matrix to contain information about variable heat conduction throughout the device. In other words, if we want to put a steal rod into the thermal storage device, we can account for heat moving faster through it with that D matrix. The minus sign and the h^2 term are artifacts of the mathematical development of the Laplacian in graph theory.

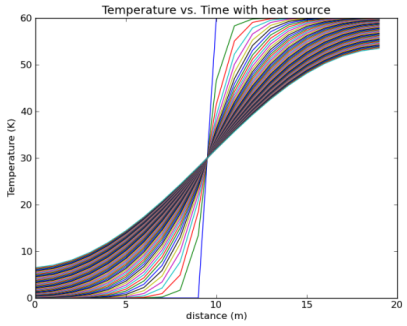
Next, as the comment below says, this part sets the initial temperature in the device. For this experiment, the initial temperature is room temperature (290 Kelvin) and a heat source can be added relatively easily during the iteration. The only trick to the heat source is that in the code it is put in as a temperature change, but this change in temperature to match experiment needs to be calculated from the known power input into the device (the sunlight concentrated on the thermal storage which might be about 800 Watts) and then converted into a temperature change based on the heat capacity of the material inside the thermal storage, which is sand in this case.

There are a couple manipulations that can produce an image of the evolution of the heat in a 1D rod over time, shown in the figures below.

```
"""Solve the 1D diffusion equation using CN and finite differences."""
from time import sleep

import numpy as np
import matplotlib.pyplot as plt
import networkx as nx


# The total number of nodes
nnodes = 10
# The total number of times
ntimes = 500
# The time step
dt = 0.5
# The diffusion constant
D = 0.1
# The spatial mesh size
h = 1.0


G = nx.grid_graph(dim=[nnodes])
L = np.matrix(nx.laplacian(G))

# The rhs of the diffusion equation
rhs = -D*L/h**2
```

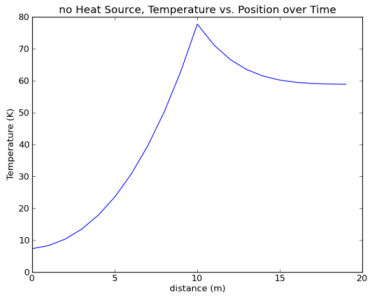
```
# Setting initial temperature
T = 60*np.matrix(np.ones((nnodes,ntimes)))
for i in range(nnodes/2):
    T[i,0] = 0;

# Setup the time propagator. In this case the rhs is time-independent so we
# can do this once.
ident = np.matrix(np.eye(nnodes,nnodes))
pmat = ident+(dt/2.0)*rhs
mmat = ident-(dt/2.0)*rhs
propagator = np.linalg.inv(mmat)*pmat

# Propagate
for i in range(ntimes-1):
        T[nnodes/2,i] = T[nnodes/2, i] + 1
        T[:,i+1] = propagator*T[:,i]

# To plot 1 time
plt.plot(T[:,300])
plt.show()

# To plot all times
#plt.plot(T)
```

The code for 1D modeled heat flow as expected. The figure below shows the time evolution of a 1D rod that starts with one half at 60 degrees Kelvin and the other half at zero degress Kelvin.



Temperature (Kelvin) versus distance (m) with random thermal constants

Additionally, adding a heat source into the system performed well



Temperature (Kelvin) versus
distance (m) with
heat source added at 5 meters

The thermal constants are within an order of magnitude of those that would be in sand and pumice, the main materials in the experimental device.

Implementation of the 2D code

The 2D code was similar to the 1D code. It was especially easy to step up to this, because the the array used to propogate through time is just a 1D vector. That makes it a lot trickier to define specific initial conditions, where the temperature is defined through half the device say, but it makes it much easier and faster to code.

Next, there were a lot of libraries that needed to be imported. Importing the libraries in shortened format such as plt for matplotlib makes it easier to type in before hand, but really it is simplest to use the `import*` because then the library code is just uploaded and you do not have to reference the functions in each step. There are surely subtle speed of program issues with typing it in beforehand, but for the purpose of this code, it helps just so that it is clear which library the functions are coming from. That is to say, by having to type np.matrix(), it is known that that's from the numpy library and it might be tough to get it to interact with something from the matplotlib library.

Below, I set up the diffusion constant. This can vary the thermal conductivity within the device. This part here just makes sure the middle of the array can be varied to have some aluminum thermal conductivity in it, because the device had that rod in it to try and make it conduct better.

To this point the code is very similar to one dimension, except that the vectors for the x and y dimensions are defined here and a few other necessary changes. Conceptually it is relatively straightforward to this point, but it can be difficult to find the right syntax to make the right kind of

array. For instance, the linspace arrays above work really well with a contour plot, where as trying to make an array by means outside of the numpy library could cause some problems.

Adding in the heat source as a matrix makes the code a bit more readable and makes the heat source section easier to locate. Before, energy was just added during the iteration loop. Although this works physically, it can be difficult to keep track of what is happening mathematically.

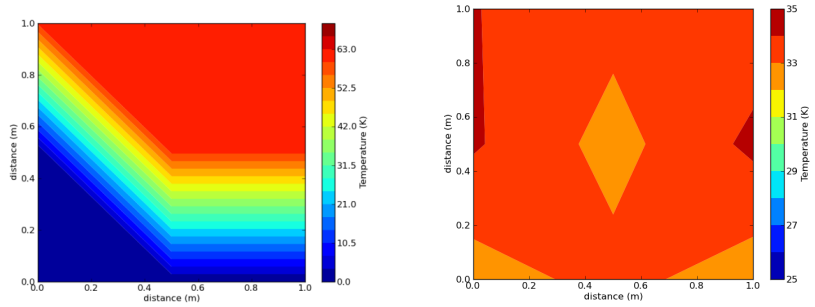
Finding the commands to make sure the contour plot function is fed the type of arrays it needs to be and the labeling syntax can be a bit difficult for someone not used to python since the library sources are a bit scattered across the internet, but there are a lot of good sources to find with a little digging (as of the writing of this paper).

```
"""Solve the 1D diffusion equation using CN and finite differences."""
from time import sleep
import numpy as np
import matplotlib.pyplot as plt
import networkx as nx
from pylab import *
# The total number of nodes
nodx = 3
nody = 3


nnodes = nodx*nody

#this is for the plotting
xmin = 0.0
xmax = 1.0
ymin = 0.0
ymax = 1.0
# Ny = 4
# Nx = 4
# tmin = 0.0
# tmax = 1000.0
# Nt = 3000

# The total number of times
ntimes = 100
# The time step
dt = 0.5
# The diffusion constant
D = np.matrix(np.eye(nnodes,nnodes))
D = .1*D
# this loop trys to set a few of the central D values to be different
#so we can simulate something being stuck in the center of the device
Drod = 2
for i in range(4):
        D[i + nnodes/2, i + nnodes/2] =  1
        D[i + nnodes/2, i + nnodes/2] =  Drod*D[i + nnodes/2, i + nnodes/2]
        # The spatial mesh size

h = 1.0

tmin = 0
tmax = dt*ntimes

x, dx = np.linspace(xmin, xmax, nodx, retstep=True)
y, dy = np.linspace(ymin, ymax, nody, retstep=True)
t, dt = np.linspace(tmin, tmax, ntimes, retstep=True)
G = nx.grid_graph(dim=[nodx,nody])
L = np.matrix(nx.laplacian(G))

#making an expression for the heat source to go into the rhs section
C = np.matrix(np.zeros((nnodes,nnodes)))
C[nnodes/2,nnodes/2] = 0
```

```
# The rhs of the diffusion equation
rhs = -D*L/h**2 + C

# Setting initial temperature
T = 60*np.matrix(np.ones((nnodes,ntimes)))
for i in range(nnodes/2):
    T[i,0] = 0;

# Setup the time propagator. In this case the rhs is time-independent so we
# can do this once.
ident = np.matrix(np.eye(nnodes,nnodes))
pmat = ident+(dt/2.0)*rhs
mmat = ident-(dt/2.0)*rhs
propagator = np.linalg.inv(mmat)*pmat

# Propagate  E is for energy conservation
E = np.zeros(ntimes)
for i in range(ntimes-1):
        E[i] = sum(T[:,i])
        T[:,i+1] = propagator*T[:,i]

# To plot 1 time
print E[2]


#need to convert the big string T into a matrix for plotting and visualization
w = 0

# R = np.matrix(np.zeros((nodx,nody,ntimes)))
# t[:,:,:] = R[:,:,:]

# a 3d array (two stacked 2d arrays)
t = np.zeros((nodx, nody, ntimes))


w = 0
for p in range(ntimes):
        for i in range(nodx):
                for j in range(nody):
                        t[i,j,p] = T[w, p]
                        w = w + 1
        w = 0
        # print w

print t[:,:,1]
#cannot plot numpy arrays! so we have to use a normal u array
# u[:,:] = T[:,:]

V, dV = np.linspace(0, 70, 4, retstep=True)

print V
CS = plt.contourf(x,y,t[:,:,0], V)

plt.ylabel('distance (m)')
plt.xlabel('distance (m)')
# Make a colorbar for the ContourSet returned by the contourf call.
cbar = colorbar(CS)
cbar.ax.set_ylabel('verbosity coefficient')

plt.show()

# To plot all times
#plt.plot(T)
```

The above code produced the following contour plots, which evolve from roughly half at 60 Kelvin and half at 0 Kelvin to balance out around 30 Kelvin.

Time Evolution of 2D system



The time evolution of a 2D system. The system at time zero (left) with half at 0˚ and half at 60˚, and the system 50 seconds later (right)

Once the basic codes were written and seemed to be producing the graphs one would expect with conservation of energy, the specific parameters for the solar concentration device can be thrown in. The heat source was putting about 1000 Watts into the system and it took a eight or so hours to heat from 20˚ C to 70˚ C. Knowing this, the thermal conductivity of sand was put into the system

[1] "The Python Tutorial — Python v2.7.1 documentation." 2 June 2011

Retrieved from http://docs.python.org/tutorial/

[2] Anonymous Author, "Scheffler Community Kitchen - Solar Cooking." 2 June 2011

Retrieved from http://solarcooking.wikia.com/wiki/Scheffler_Community_Kitchen

[3] Jason Rapp, "Construction and Improvement of a Scheffler Reflector and Thermal Storage Device," 1-21 (2010).

[4] Connor Barickman, "Thermal Storage Device Properties," 1-4 (2011).

[5] Gerald W. Recktenwald, "Finite-Difference Approximations to the Heat Equation," (2004).

[6] "Solare Brücke e.V." Web. 2 July 2011

Retrived from www.solare-bruecke.org/

Gist URLs (full code sources)

1D: git://gist.github.com/979154.git

2D: git://gist.github.com/1004879.git