Solving the heat equation using python

Rafael O. Soto

April 11, 2014
Phys-440
Illinois Institute of Technology, Chicago, IL 60616

Abstract

In this lab I investigated various PDE solving techniques for the heat equation, and used python to study the behavior of heat dissipation.

1. Introduction & Techniques

There are several physical quantities that can be modeled using partial differential equations, one being heat. Heat can be described as a function of position and time, due to this behavior one requires a partial DE over and ordinary DE. Since a partial differential equation depends on more than one variable one must make sure to have enough initial conditions

There are various ways to solve PDEs one being analytically in which one finds a closed form solution. This is usually done by transforming the PDE into several ODEs using separation of variables. As with regular ODEs one can also solve these equations numerically. One such method is based on converting the PDE into a finite difference equation(in this case a central difference), in which small time steps 'push' the equation forward in time and its behavior can be studied. The analytic solution to the wave equation that I'll be studying is eq. 1:

$$T(x,t) = \sum_{n=1,3,..}^{inf} \frac{4T_0}{n\pi} \sin(k_n) e^{-k_n^2 Kt/(C\rho)} \quad (1)$$

And the numerical central difference algorithm I'll be comparing it to will be:

$$T_{i,j+1} = T_{i,j} + \eta[T_{i+1,j} + T_{i-1,j} - 2T_{i,j}], \eta = \frac{K\Delta t}{C\rho\Delta x^2} \quad (2)$$

in these equations C, is the heat capacity, rho is density, and K is thermal conductivity.

2. Analysis and Results

2.1. Solving the heat equation (17.17)

I modified the EqHeat.py code found in A Survey of Computational Physics, in order to solve the heat equation for a system with the following parameters/initial conditions:

$$T(x=0,t) = T(x=L,t) = 0K, T(x,t=0) = 100K \quad (3)$$

$$K = 237W/(mK), C = 900J/(kgK), \rho = 2700kg/m^3 \quad (4)$$

I defined a two dimensional array that indexed 100 space divions for a bar 1 m in length and one for present and past times. I plugged in the initial conditions, and I applied the central difference formula via EqHeat. I did a trial of various timesteps and eventual came up with this Fig.1 for heat dissipation.

2.2. Assessment and Visualization (Answers to 17.18)

1. The solution given by the above method does indeed give a temperature distribution that varies smoothly along the bar and agrees with the boundary conditions like Fig 17.14 in A Survey of Computational Physics. (Fig 1.)

2. The solution also gives a temperature distribution that varies smoothly with time and attains equilibrium at 0 as time goes on.(Fig 1.)

3. I implemented an analytic solution given in eq 1 and set up a system where it would make the same type of graphic as in 17.17, the results are shown in fig 2.

The analytic solution gives a similar graph, however

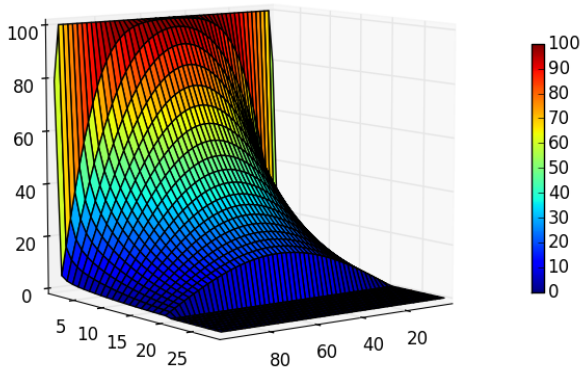

Figure 3: Surface plot with color gradient representing temperature. Like colors are temperature isotherms.

it doesn't decay as fast as the numerical solution. I also had to vary the parameters of the analytical solution a lot in order to make the exponential term decay as fast as the numerical one. Overall they are very similar however the central difference solution is much faster to solve (twice as fast) and it is also more well-behaved.

4. Surface plots are in figure 2.

5. Surface plots are shown in figure 3.

6. The isotherms of constrant temperature are plotted in fgure 3 as colors, and in figure 7 as lines.

7. Stability test, check 17.76 the solution becomes unstable when $\eta > \frac{1}{2}$ The following figure(4) occured when

solving the PDE, using the central-difference method with $\eta = 0.504801$, just as 17. 76 described, the solution becomes very unstable. Any value under this and I get a well-behaved solution like in fig 2.

8.Material Dependence: Repeat the calculation for iron. For this calculation I looked up the constants and used $\kappa = 80, C = 450, and \rho = 7874$ The solution for the calculation is in the following figure(5), I had to adjust the timestep and the x-step in order to get a well behaved solution.

9. Initial Sinusoidal Dist. Compare the analytic solution: $T(x,t) = \sin(\pi * x/L)e^{-\pi^2 \kappa t/(L^2 C\rho}$.
The following figure(6) shows the result of this analytic

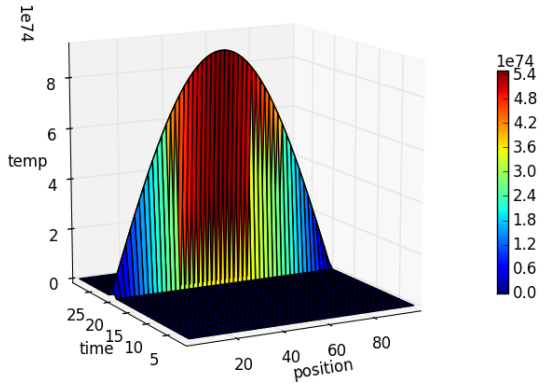

Figure 4: Non well behaved solution for solving numerically with $\eta > .5$

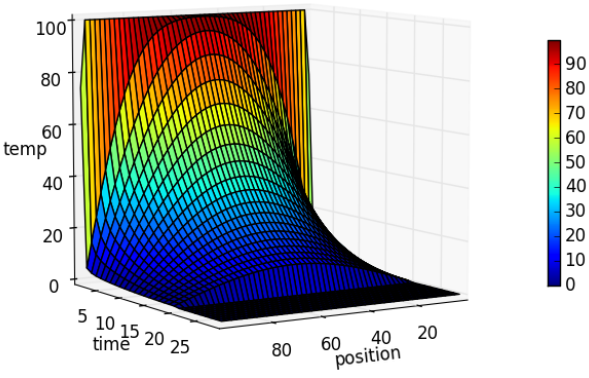

Figure 5: Heat flow through space and time for a 1-D Iron bar of 1 meter.

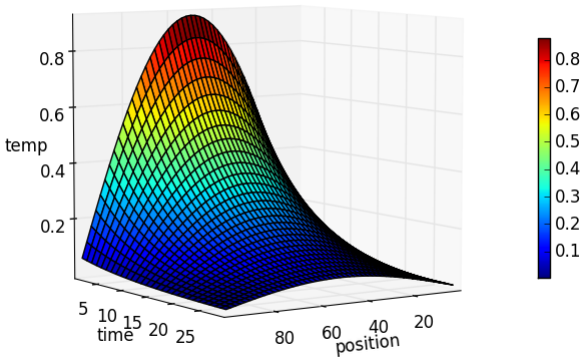

Figure 6: Heat flow through space and time for a 1-D bar of 1 meter using the equation in 9. .

solution, as we can see it very closely resembles the central difference method. This is actually a more well behaved solution than I got using the previous analytic method, which seemed like it would be more accurate at first glance.

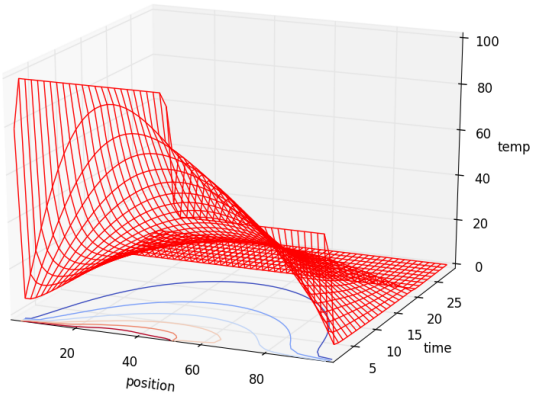

Figure 7: Heat flow through two metal bars in contact one initially at 100 K , the other at 50 K .

10. Two-bars in contact. Find a solution when two idental bars 0.25 are placed in contact along one end. With other ends kept at 0 K. The solution is shown in the following figure(7), with the isotherms plotted under it.

11. The e-mail said to do only the parts you find particulary interesting and important so I decided to omit this one.

3. Conclusions and Discussion

I learned nearly as much about 3D python graphics as I did about partial differential equation modeing in this lab. All the simulations and plotting I did resembled the book's graphics very closely and I think this alone shows that the project was a success. It was very interesting how the numeric method had a specific constant, η, that one had to adjust in order to get a well-behaved solution. As I reflect on this lab and this class in general I wonder how one changes these systems to better mimic real life, would you have to insert chaotic factors to mimic the constant flux in reality? Is this done in current simulations? I guess monte-carlo, and newton's cooling would be good methods to apply for this problem. There are certainly many improvements that could be made to this code to make it more efficient. I neglected making much of the work into methods, even though it would make the code more readable, simply due to time constraints and the varying nature of the questions addressed in this report. The contrasts of the numeric and analytical solutions was quite interesting as well, and I can connected it to other problems that I've seen. For example I've read that investment traders, and (as I've personally experienced) people who model molecular dynamics, both use a wide array of monte carlo simulations. The numeric methods are just so much quicker than analytic calculations and they're highly iterable. All in all it was an enjoyable lab and class in general.

[1] Rubin, H. Landau, Manuel Jose Paez, Cristian C. Bordeianu *A Survey of Computational Physics* 2012: Princeton University Press.

[2] Python Software Foundation *Python v2.7.6 documentation* 1990-2014

[3] The People of Stack Exchange, Google *Internet.*

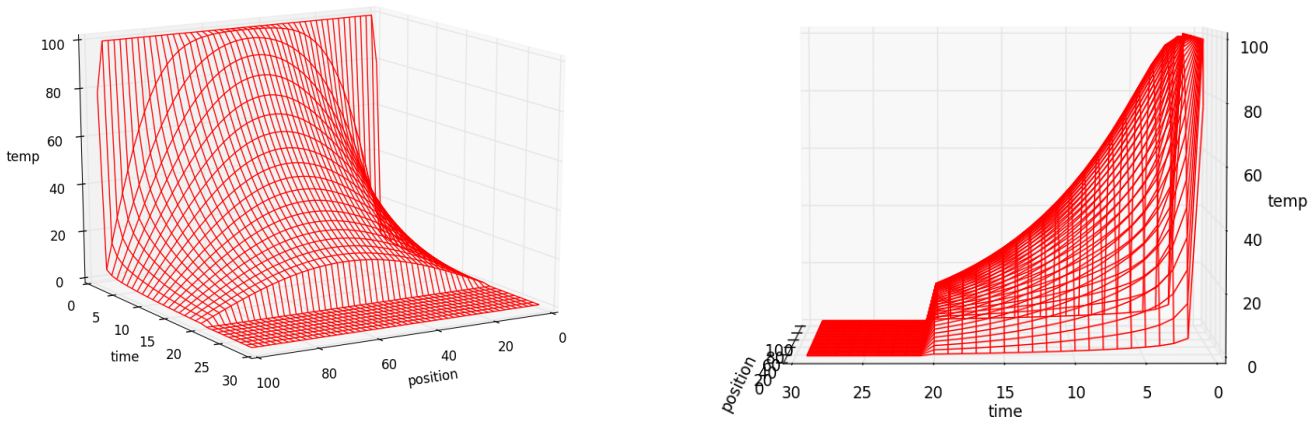

Figure 1: 3D representation of the central difference solution for temperature of a metal bar through space and time.

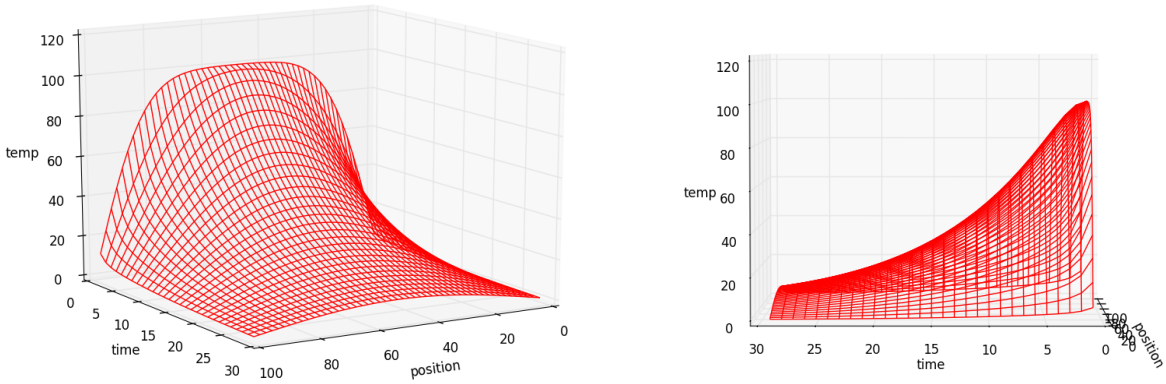

Figure 2: Results from computing the analytical solution of the heat equation for a metal bar.

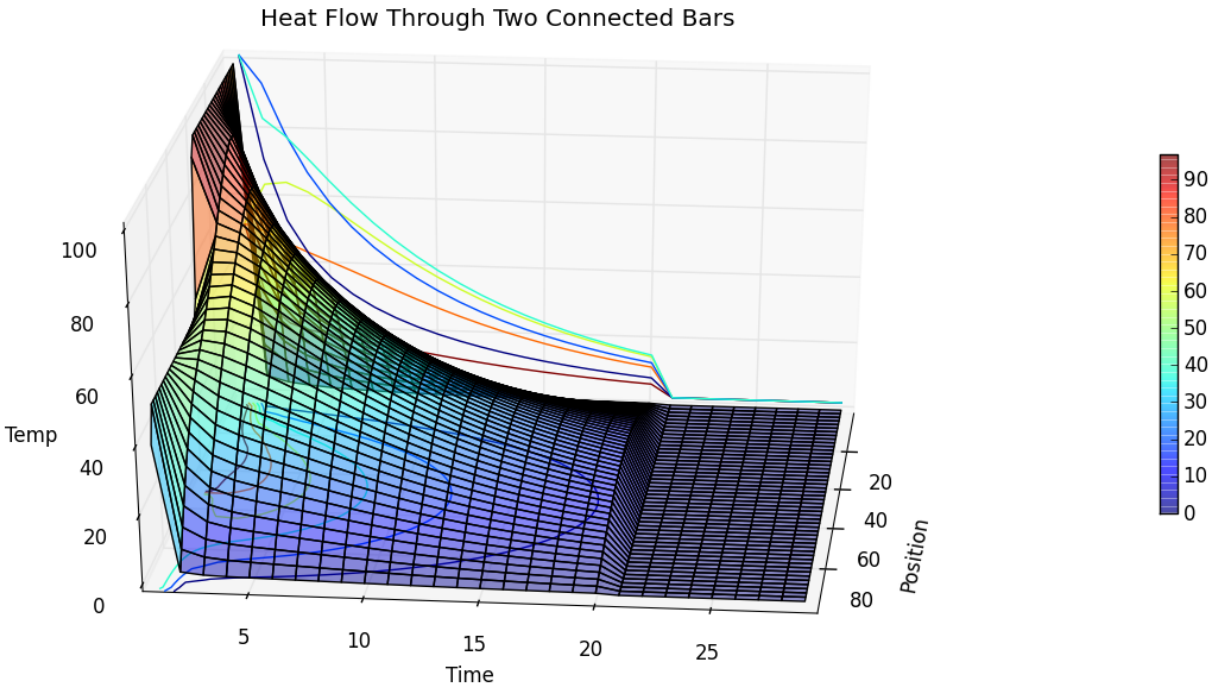

Figure 8: Graph that encompasses every technique learned for 3D plotting in this lab. Also has contour lines.

```
om numpy import *
port matplotlib.pylab as p
om mpl_toolkits.mplot3d import Axes3D

=101; Nt= 10000; Dx=0.03; Dt=2
PPA =230.; SPH=900.;
O= 2700.

 zeros((Nx,2),float); Tpl=zeros((Nx,31),float)

int ("Working, wait for figure")

r ix in range(1,Nx-1):
  T[ix,0]=100;
  T[0,0]=0.0;T[0,1]=0.
  T[Nx - 1,0]=0.; T[Nx - 1,1]=0.0;
  cons=KAPPA/(SPH*RHO)*Dt/(Dx*Dx);
  m=1

r t in range(1, Nt):
  for ix in range(1, Nx -1):
      T[ix,1]=T[ix,0] + cons*(T[ix + 1,0 ]  +T[ix - 1,0] - 2.*T[ix,0])
  if t%500 == 0 or t==1:
      for ix in range (1,Nx -1, 2): Tpl[ix,m]=T[ix,1]
      print(m)
      m=m+1
  for ix in range(1, Nx -1 ): T[ix,0]=T[ix,1]

list(range(1, Nx - 1 ,2))
list(range(1,30))
Y = p.meshgrid(x,y)

f functz(Tpl):
  z=Tpl[X,Y]
  return z

functz(Tpl)
g=p.figure()
=Axes3D(fig)
.plot_wireframe(X,Y,Z, color='r')
.set_xlabel('position')
.set_ylabel('time')
.set_zlabel('temp')
show()
int('finised')
```

```
efine a 2-D array T[101][2] for the temperature as a function of space and time. The
irst index is for the 100 space divisions of the bar, and the second index is for present and
ast times (because you may have to make thousands of time steps, you save memory by
aving only two times).


Analytical Solution
t.figure()
0

f Heat(b,t):
  summ=0.
  cons=KAPPA/(SPH*RHO)
  L=101.
  for n in range(1,10000,2):
      kn=(n*3.14)/L
      summ=summ+((4*T0)/(n*3.14))*np.sin(kn*b)*np.exp(-(kn**2)*cons*t)
  return summ


list(range(1, Nx - 1 ,2))
list(range(1,30))
Y = p.meshgrid(x,y)


l=zeros((Nx,31),float)
 zeros((Nx,2),float)

or j in range(1,30):
   for i in range(1,Nx - 1 ):
       Tpl[i,j]=Heat(i,j)

r t in range(1, 10000000,400000):
  if t==1: T0=100.
  for ix in range (1,Nx -1, 2): Tpl[ix,m]=Heat(ix,t)
  print(m)
  m=m+1
  for ix in range(1, Nx -1 ): T[ix,0]=T[ix,1]


f functz(Tpl):
  z=Tpl[X,Y]
  return z

functz(Tpl)

g=p.figure()
=Axes3D(fig)
.plot_wireframe(X,Y,Z, color='r', cmap=cm.coolwarm)
.set_xlabel('position')
.set_ylabel('time')
.set_zlabel('temp')
show()
int('finised')
```

```
g = plt.figure(figsize=(8,6))

 = fig.add_subplot(1,1,1, projection='3d')

ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap=cm.jet)
et = ax.contour(X, Y, Z, zdir='z', cmap=cm.jet)
et = ax.contour(X, Y, Z, zdir='x', cmap=cm.jet)
et = ax.contour(X, Y, Z, zdir='y', cmap=cm.jet)
.set_xlabel('position')
.set_ylabel('time')
.set_zlabel('temp')
 = fig.colorbar(p, shrink=0.5)


 Stability test.
=101; Nt= 10000; Dx=0.03; Dt=4.8
PPA =230.; SPH=900.;
O= 2700.

 zeros((Nx,2),float); Tpl=zeros((Nx,31),float)

int ("Working, wait for figure")

r ix in range(1,Nx-1):
  T[ix,0]=100;
  T[0,0]=0.0;T[0,1]=0.
  T[Nx - 1,0]=0.; T[Nx - 1,1]=0.0;
  #Changed from .21
  cons=.26
  m=1

r t in range(1, Nt):
  for ix in range(1, Nx -1):
      T[ix,1]=T[ix,0] + cons*(T[ix + 1,0 ]  +T[ix - 1,0] - 2.*T[ix,0])
  if t%500 == 0 or t==1:
      for ix in range (1,Nx -1, 2): Tpl[ix,m]=T[ix,1]
      print(m)
      m=m+1
  for ix in range(1, Nx -1 ): T[ix,0]=T[ix,1]

list(range(1, Nx - 1 ,2))
list(range(1,30))
Y = p.meshgrid(x,y)
```

```
f functz(Tpl):
  z=Tpl[X,Y]
  return z

functz(Tpl)
g=p.figure()
=Axes3D(fig)
.plot_wireframe(X,Y,Z, color='r')
.set_xlabel('position')
.set_ylabel('time')
.set_zlabel('temp')
show()
int('finised')

# Repeat calculation for iron
=101; Nt= 10000; Dx=0.03; Dt=4.8
PPA =80.; SPH=450.;
O= 7874.

 zeros((Nx,2),float); Tpl=zeros((Nx,31),float)

int ("Working, wait for figure")

r ix in range(1,Nx-1):
  T[ix,0]=100;
  T[0,0]=0.0;T[0,1]=0.
  T[Nx - 1,0]=0.; T[Nx - 1,1]=0.0;
  #Changed from .21
  cons=.26
  m=1

r t in range(1, Nt):
  for ix in range(1, Nx -1):
      T[ix,1]=T[ix,0] + cons*(T[ix + 1,0 ]  +T[ix - 1,0] - 2.*T[ix,0])
  if t%500 == 0 or t==1:
      for ix in range (1,Nx -1, 2): Tpl[ix,m]=T[ix,1]
      print(m)
      m=m+1
  for ix in range(1, Nx -1 ): T[ix,0]=T[ix,1]
```

```
list(range(1, Nx - 1 ,2))
list(range(1,30))
Y = p.meshgrid(x,y)

f functz(Tpl):
  z=Tpl[X,Y]
  return z

functz(Tpl)
g=p.figure()
=Axes3D(fig)
.plot_wireframe(X,Y,Z, color='r')
.set_xlabel('position')
.set_ylabel('time')
.set_zlabel('temp')
show()
int('finised')



## Analytical sin solution


Analytical Solution
t.figure()
0

f Heat(b,t):
  summ=0.
  cons=KAPPA/(SPH*RHO)
  L=101.
  summ = np.sin(3.14*b/L)*np.exp(-(3.14**2)*cons*t*(1/(L**2)))
  return summ


list(range(1, Nx - 1 ,2))
list(range(1,30))
Y = p.meshgrid(x,y)


l=zeros((Nx,31),float)
 zeros((Nx,2),float)

or j in range(1,30):
    for i in range(1,Nx - 1 ):
        Tpl[i,j]=Heat(i,j)

r t in range(1, 10000000,400000):
  if t==1: T0=100.
  for ix in range (1,Nx -1, 2): Tpl[ix,m]=Heat(ix,t)
  print(m)
  m=m+1
  for ix in range(1, Nx -1 ): T[ix,0]=T[ix,1]
```