GES 554 PDE MEMO

Subject: Heat Diffusion with Rectified Sine Initial Condition

TO: GES 554.001
GES 554.996

Date: 12 Feb 2014

CC:

Memo: GES554-Project-2

From: Charles O'Neill

REF:

Ext: 8-5161

Summary:

This project investigates the heat diffusion of a 1D domain in time. An initial rectified sine input diffuses to zero with the maximum temperature monotonically decreasing and moving towards the domain's center. The maximum temperature at x=0.6 is 0.207 at time 0.041. The fastest change in temperature occurs at x=0.5 at time 0. Zero flux occurs on an asymptotic curve between 0.25 and 0.5. The total energy transferred out of the system is 0.318 Cv.

Discussion:

This memo discusses heat diffusion for a domain with a rectified sine wave initial condition.

$$u_t = u_{xx}$$
$$u(0,\mathrm{t}) = 0$$
$$\mathrm{u}(1,\mathrm{t}) = 0$$
$$\mathrm{u}(\mathrm{x},0) = \begin{matrix} \sin(2\pi x) & when\ 0 < x < \frac{1}{2} \\ 0 & otherwise \end{matrix}$$

The project requirement require the solution and discussion of the heat diffusion problem. The first section solves the heat diffusion equation. Next, four questions are answered: 1) maximum temperature at x=0.6, 2) location and magnitude of the fastest change in temperature, 3) locations where the heat flux is zero, and 4) an evaluation of the total change in energy. The author used the Anaconda distribution of Python 2.7.

General Solution

The governing equation for 1D diffusion with a heat transfer coefficient of unity is

$$u_t = u_{xx}$$

Separating variables and solving gives the temporal and spatial solutions

$$u(x,t) = X(x)T(t)$$
$$X(x) = a_n \sin(\mathrm{n}\,\pi x)$$
$$\mathrm{T(t)} = e^{-n^2\pi^2 t}$$

Enforcing the initial condition requires a sine expansion of a rectified sine. From theory, computing the coefficients reduces to an integral.

$$a_n = 2\int_0^1 u(x,0)sin(n\pi x)dx$$
$$= 2\int_0^{0.5} \sin(2\pi x)\, sin(n\pi x)dx + 2\int_{0.5}^1 0dx$$

Computing these coefficients gives

$$a_2 = 0.5$$
$$a_n = -4\frac{\sin(0.5\pi m)}{\pi m^2 - 4\pi}$$

The solution is plotted in Figure 1.

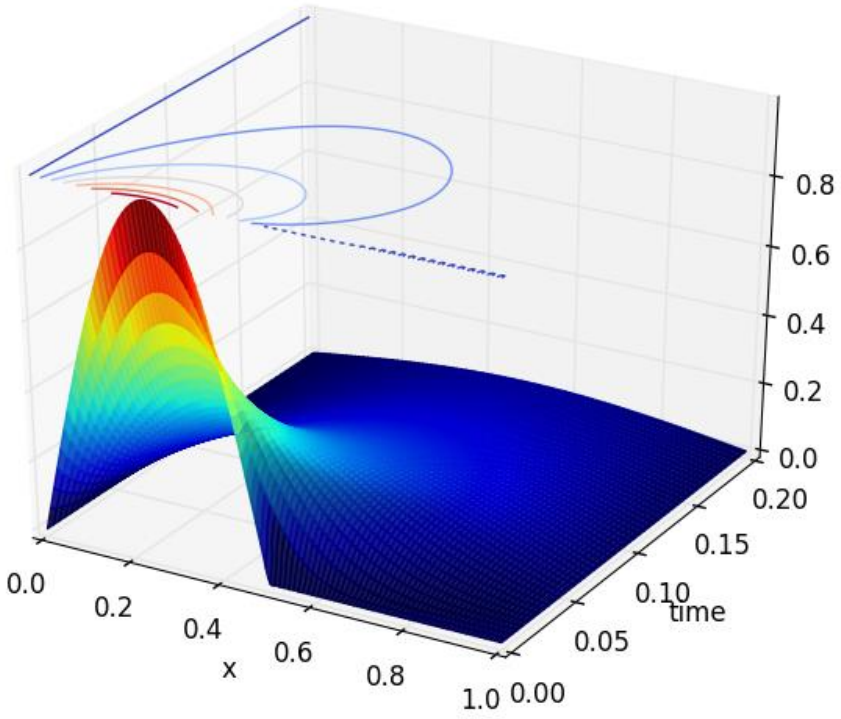

Figure 1 Temperature in Space and Time

Visually, the initial condition convergence as approximated by the sine series is illustrated in Figure 2. Sine series truncation was determined by selecting an acceptable error of 1 part in 1 million (10^{-6}). This error tolerance is governing by the subsequent tasks involving

fluxes and large temporal derivatives. In particular, the interface at x=0.5 needed to be highly resolved.

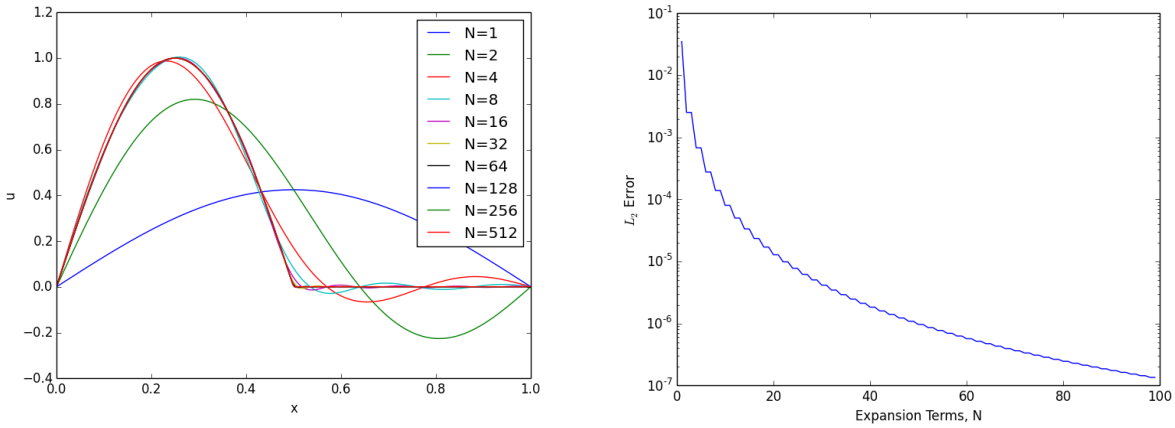

Figure 2 Initial Condition and L_2 Convergence of Sine Series

A Python code to generate the series solution and figures is attached in the Appendix.

Maximum Temperature and Time at x = 0.6

This section determines the maximum temperature at x=0.6 and the corresponding time. The code also uses a search routine to numerically determine that the maximum temperature seen at x=0.6 is 0.207 at time 0.041. Figure 3 shows the response in time.

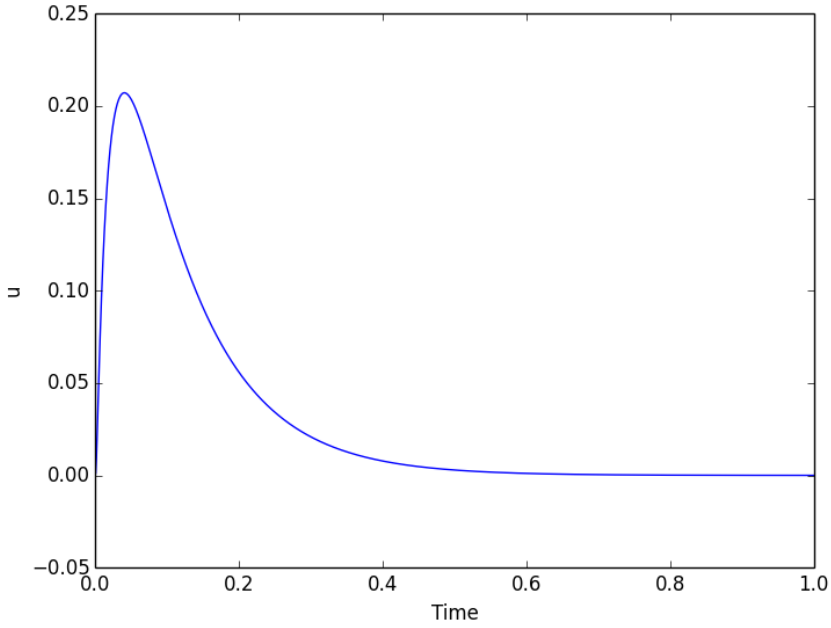

Figure 3 Temperature in Time at x=0.6

Moving the critical location strongly impacts the maximum temperature, since the spatial gradient of this maximum temperature is large.

Fastest Temperature Change

This section determines the fastest temperature change in the domain. Searching for the maximum temporal derivative magnitude is identical to searching for the largest spatial Laplacian magnitude. By inspection, the largest 2nd derivative occurs exactly at the initial condition at the rectified-sine to zero interface (x=0.5). As we do not have further information regarding the actual physical configuration, we must assume that the initial conditions are as written. (Perhaps this is modeling the behavior after a heated and a non-heated bar are forced into contact.) It should be noted that if the bar is heated as one piece, the sharp C_0 initial condition at x=0.5 is physically impossible. In this case, the fastest temperature change would still likely occur at x=0.5.

Zero Flux

This section determines locations where the flux is zero. Visually, we expect zero flux at the crests and valleys of the solution, where the derivative is zero. Figure 4 plots the solution as filled contours. The zero flux curves are plotted in black. Notice that there are two curves with zero flux.

The first is the crest of the decaying and diffusing rectified sine. The crest at t=0 begins at x=0.25. As time progresses, the zero-flux curve asymptotically approaches x=0.5.

The second zero-flux curve occurs at the zero boundary condition between x=0.5 and x=1. Notice that aliasing is present in the 2nd case.

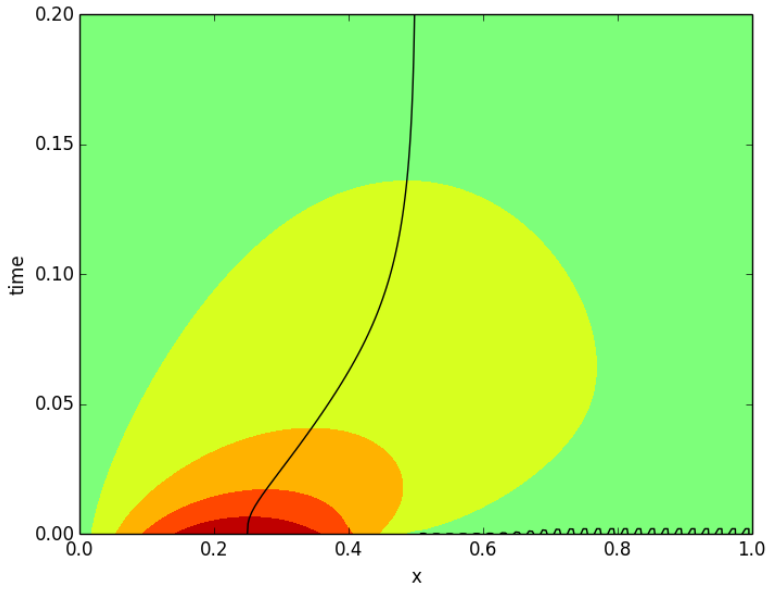

Figure 4 Zero Flux Curves

Energy Transported

This section discusses the energy transported from the system between the initial conditions and the steady state temperature distribution. It should be noted that the steady state temperature distribution is zero everywhere. The problem statement requested two independent methods for determining the total energy lost.

The 1st method comes from thermodynamics and compares the initial energy and the final energy. From thermodynamics, the specific energy in a solid substance is

$$\Delta e = C_v \Delta T$$

Integrating over the domain determines the total energy

$$\Delta E = \int_0^1 C_v \Delta T$$

The initial energy in the rectified sine wave is

$$\begin{aligned} \Delta E &= C_v \int_0^{0.5} (sin(2\pi x) dx + C_v \int_{0.5}^1 0 dx - C_v \int_0^1 0 dx \\ &= \frac{C_v}{\pi} \end{aligned}$$

The 2nd method comes from heat transfer and integrates the boundary fluxes over time. Remembering that the boundary normal depend on location, we can derive the following expression for the total energy exiting the system.

$$\Delta E = C_v \int_0^\infty \left(q(0,\mathrm{t}) \cdot \hat{n}_0 + q(1,\mathrm{t}) \cdot \hat{n}_1 \right) dt$$

This integral is performed in the Python code with a simple trapezoidal scheme. The result is

$$\begin{aligned} \Delta E &= 0.31829 C_v \\ &\approx \frac{C_v}{\pi} \end{aligned}$$

These approaches are fundamentally identical when considering Green's theorem.

Further Information

If further information is needed regarding this project's analysis, please contact Charles O'Neill at 8-5161 or croneill@eng.ua.edu.

Appendix A:

Heat Diffusion Code

```
import pylab
from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt
from matplotlib import cm

# Discretization
N = 100
nX = 200
nT = 100

# Physical Constants
alpha = 1

# T(t)
def T(L,c,t):
return exp(-L*L * c * t)

# A_n for sine series
def A(m):
if(m==2):
return 0.5
else:
return -4*sin(0.5*pi*m)/(pi*m**2 - 4*pi)

# X(x)
def X(m,x):
return sin(m*pi*x)
def dX(m,x):
return m*pi*cos(m*pi*x)
def ddX(m,x):
return -(m*pi)**2*sin(m*pi*x)


# u = X(x)T(t)
def U(alpha,x,t,N):
Ys = 0
for n in range(1,N+1):
Ys = Ys + T(n*pi,alpha,t)*A(n)*X(n,x)
return Ys

# du/dx = dX(x)/dx T(t)
def dU(alpha,x,t,N):
Ys = 0
for n in range(1,N+1):
Ys = Ys + T(n*pi,alpha,t)*A(n)*dX(n,x)
return Ys

# du/dt = ddX(x)/ddx T(t)
def ddU(alpha,x,t,N):
Ys = 0
for n in range(1,N+1):
Ys = Ys + T(n*pi,alpha,t)*A(n)*ddX(n,x)
return Ys

# Create an x-t grid (appears as a matrix)
```

```
[Xm,Tm] = np.mgrid[0:1:nX*1j, 0:0.2:nT*1j]

# Solve U at each x-t point
Um = U(alpha,Xm,Tm,N)

def plotXTUsurf():
# Plot the x-t-u surface and a contour above the surface
fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot_surface(Xm, Tm, Um, rstride=2, cstride=2, alpha=0.9, facecolors=cm.jet(Um))
cset = ax.contour(Xm, Tm, Um, zdir='z', offset=1, cmap=cm.coolwarm)
plt.show()
xlabel("x")
ylabel("time")

# Plot the temperature response at x=0.6
def plotTempX0p6():
t = linspace(0,1,1000)
Ut = U(alpha,0.6,t,50)
fig2 = plt.figure()
ax2 = fig2.gca()
ax2.plot(t, Ut)
xlabel("Time")
ylabel("u")
print "Maximum Value at x =0.6 is " + str(max(Ut))
print "Located at " + str(t[argmax(Ut)])
# L squared error versus N
def L2(N):
Error = 0
for n in range(1,N+1):
Error += -1.0/2.0*A(n)**2
Error += 0.25
return Error

def PlotL2Error():
n = zeros(N)
L2E = zeros(N)
for i in range(1,N):
n[i] = i+1
L2E[i] = L2(i+1)
fig3 = plt.figure()
ax3 = fig3.gca()
ax3.semilogy(L2E)
xlabel("Expansion Terms, N")
ylabel("$L_2$ Error")

# Coefficients
def plotCoeffs():
ACoeffs = zeros(N)
for i in range(1,N):
ACoeffs[i]=A(i)
fig4 = plt.figure()
ax4 = fig4.gca()
ax4.semilogy(ACoeffs,'.')
xlabel("n")
ylabel("$A_n$")

def FiniteDiffCheck(t,x,h,N):
dt = ( U(alpha,x,t+h,N) - U(alpha,x,t-h,N)) /(2*h)
dxx = ( U(alpha,x+h,t,N) - 2.0* U(alpha,x,t,N) + U(alpha,x-h,t,N)) /(h**2)
return dt-alpha*dxx
print(FiniteDiffCheck(0.25,0.25,0.001,100))
```

```
# Calculate the energy the hard way
# Create array of du/dx on both boundaries
def TotalEnergy():
TimeEffectivelyZero = 4
t = linspace(0,TimeEffectivelyZero,5000)
dUtLeft = dU(alpha,0,t,50)
dUtRight = dU(alpha,1.0,t,50)
fig5 = plt.figure()
ax5 = fig5.gca()
ax5.plot(t, dUtLeft)
ax5.plot(t, dUtRight)
xlabel("Time")
ylabel("u")

# Create a rough trapezoidal rule integration routine
def TrapIntegrate(array,a,b):
n = size(array)
I = 0
for i in range(0,n-1):
I = I + 0.5*(array[i]+array[i+1])
return I*(b-a)/(n-1)
#Energy via flux
print("Energy Transport via Fluxes " + str(TrapIntegrate(dUtLeft-dUtRight,0,TimeEffectivelyZero)))

# Check Initial Condition
def CheckIC():
x = linspace(0,1,300)
fig6 = plt.figure()
ax6 = fig6.gca()
for i in [1,2,4,8,16,32,64,128,256,512]:
U0 = U(alpha,x,0,i)
ax6.plot(x, U0, label="N="+str(i))
xlabel("x")
ylabel("u")
legend(loc='upper right')

# Plot the x-t-u surface and a contour above the surface
def FastestChange():
dT = ddU(alpha,Xm,Tm,N)
fig7 = plt.figure()
ax7 = fig7.gca(projection='3d')
ax7.plot_surface(Xm, Tm, dT, rstride=2, cstride=2, alpha=0.9, facecolors=cm.jet(Um))
cset = ax7.contour(Xm, Tm, dT, zdir='z', offset=1, cmap=cm.coolwarm)
plt.show()
xlabel("x")
ylabel("time")


# Plot the x-t-u surface and a contour above the surface
def ZeroFlux():
dT = dU(alpha,Xm,Tm,N)
fig7 = plt.figure()
ax7 = fig7.gca()
V = linspace(-1,1,10)
cset = ax7.contourf(Xm, Tm, Um,V, cmap=cm.jet)
V2 = [0]
cset = ax7.contour(Xm, Tm, dT, V2, cmap=cm.binary_r)
plt.show()
xlabel("x")
ylabel("time")
```