**Final Examination Solutions**
**May 6, 2014**

## 1. Multivariate Nonlinear Optimization

Download the data located at
http://utkstair.org/clausius/docs/mse506/data/mse506_xm02_p01.txt
The first column corresponds to wavelength. The second column corresponds to signal intensity.

Perform a multivariate nonlinear optimization in order to fit this data to a series of Gaussian curves.

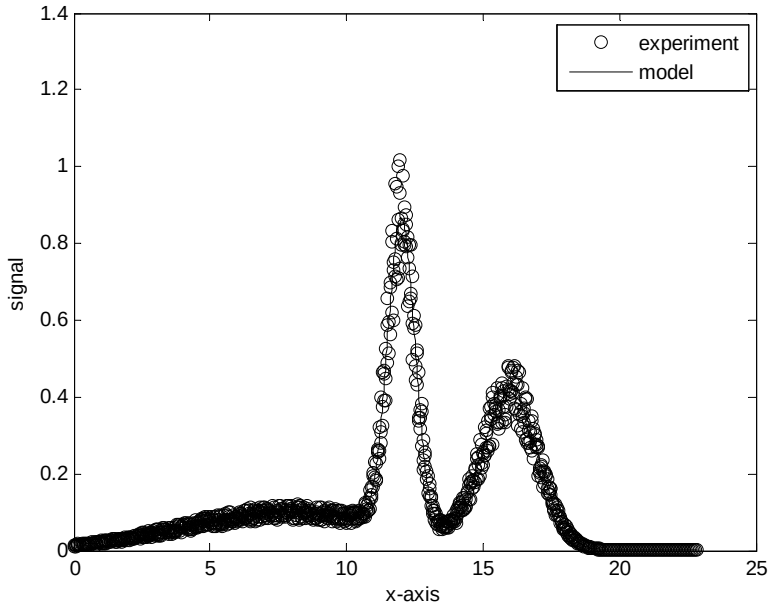
$$f(x;\mu,\sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

Determine the minimum number of Gaussian curves necessary and the mean and standard deviation of each.

Solution:

I decided to use the amoeba method to solve this problem.

I first plotted the data in order to determine how many Gaussians were necessary.




It appears to me that three curves are necessary. From this plot, I chose initial guesses of 6, 11 and 15 for the three means and 3, 3 and 2 for the standard deviations. (These are not particularly good initial guesses.)

I used the following script to perform the optimization and plot the result.

```
clear all;
close all;
%
%  input data
%
global datamat
datamat = [0    0.011506086
0.02    0.012146601
0.04    0.015849062
...
22.78   0.000109832
22.8    0.000105579
22.82   0.000100679];
%
%  initial guess
%
mu1 = 6;
sig1 = 3;
mu2 = 11;
sig2 = 3;
mu3 = 15;
sig3 = 2;
xo = [mu1,sig1,mu2,sig2,mu3,sig3];
%
%  call amoeba
%
[f,x] = amoeba_xm02p01(xo,1.0e-8,1.0e-8);
%
%  plot
%
n = 3;
%
%  read in mean and standard deviation
%
for i = 1:1:n
    mu(i) = x(2*i-1);
    sig(i) = x(2*i);
end
ndata = max(size(datamat));
xvec(1:ndata) = datamat(1:ndata,1);
yexp(1:ndata) = datamat(1:ndata,2);
fac = sqrt(2.0*pi);
for i = 1:1:ndata
    ymod(i) = 0.0;
    for j = 1:1:n
       ymod(i) = ymod(i) + 1.0/(sig(j)*fac)*exp(-((xvec(i)-mu(j))^2)/(2.0*sig(j)^2));
    end
end
%
%  plot
%
figure(1);
plot(xvec,yexp,'ko');
hold on;
plot(xvec,ymod,'k-');
hold off;
legend('experiment','model');
xlabel('x-axis');
ylabel('signal');
```

The objective function used by amoeba.m contains the same content as this script, namely

```
function fobj = funkeval(x)
n = 3;
%
%  read in mean and standard deviation
%
for i = 1:1:n
    mu(i) = x(2*i-1);
    sig(i) = x(2*i);
end
%
global datamat

ndata = max(size(datamat));
xvec(1:ndata) = datamat(1:ndata,1);
yexp(1:ndata) = datamat(1:ndata,2);
fac = sqrt(2.0*pi);
for i = 1:1:ndata
    ymod(i) = 0.0;
    for j = 1:1:n
       ymod(i) = ymod(i) + 1.0/(sig(j)*fac)*exp(-((xvec(i)-mu(j))^2)/(2.0*sig(j)^2));
    end
end
fobj = 0.0;
for i = 1:1:ndata
    fobj = fobj + (yexp(i) - ymod(i))^2;
end
fobj = sqrt(fobj/ndata);
```

At the command line prompt, I typed:

```
>> xm02p01m
```

This generated the following output.

```
i =   1  6.0000000e+00  3.0000000e+00  1.1000000e+01  3.0000000e+00  1.5000000e+01  2.0000000e+00 f =  1.4659212e-01
i =   2  6.6000000e+00  3.0000000e+00  1.1000000e+01  3.0000000e+00  1.5000000e+01  2.0000000e+00 f =  1.4605742e-01
i =   3  6.0000000e+00  3.3000000e+00  1.1000000e+01  3.0000000e+00  1.5000000e+01  2.0000000e+00 f =  1.4498941e-01
i =   4  6.0000000e+00  3.0000000e+00  1.2100000e+01  3.0000000e+00  1.5000000e+01  2.0000000e+00 f =  1.4251167e-01
i =   5  6.0000000e+00  3.0000000e+00  1.1000000e+01  3.3000000e+00  1.5000000e+01  2.0000000e+00 f =  1.4790593e-01
i =   6  6.0000000e+00  3.0000000e+00  1.1000000e+01  3.0000000e+00  1.6500000e+01  2.0000000e+00 f =  1.5249627e-01
i =   7  6.0000000e+00  3.0000000e+00  1.1000000e+01  3.0000000e+00  1.5000000e+01  2.2000000e+00 f =  1.4598268e-01
   1  6.0000000e+00  3.0000000e+00  1.2100000e+01  1.4251167e-01  5.4985740e-02  6.7690401e-02
   2  6.0000000e+00  3.0000000e+00  1.2100000e+01  1.4251167e-01  5.4985740e-02  3.7148286e-02
   3  6.4000000e+00  3.2000000e+00  1.1733333e+01  1.3909108e-01  1.0001374e-01  5.2512959e-02
   4  6.4000000e+00  3.2000000e+00  1.1733333e+01  1.3909108e-01  9.7297567e-02  4.8861146e-02
...
1104  8.0733740e+00  4.0070509e+00  1.6006770e+01  2.4339112e-02  6.8303891e-08  9.5363380e-14
1105  8.0733739e+00  4.0070515e+00  1.6006770e+01  2.4339112e-02  9.2493218e-08  3.8915101e-14
1106  8.0733742e+00  4.0070508e+00  1.6006770e+01  2.4339112e-02  2.7847165e-08  4.1908571e-14
1107  8.0733745e+00  4.0070511e+00  1.6006770e+01  2.4339112e-02  9.3561784e-09  3.1645247e-14
```

The method converged in 1107 iterations. The values of the parameters are given by

```
>> x

x =   8.0734   4.0071  16.0068   1.0018  12.0067   0.4954
```

Thus $\mu_1 = 8.07$, $\sigma_1 = 4.01$, $\mu_2 = 16.01$, $\sigma_2 = 1.00$, $\mu_3 = 12.01$, $\sigma_3 = 0.50$

## 2. Single Non-Linear Parabolic PDE

The one-dimensional heat equation can describe heat transfer in a material with both heat conduction and radiative heat loss.

$$\frac{\partial T}{\partial \mathrm{t}} = \frac{k}{\rho C_p}\frac{d^2T}{dz^2} - \frac{\varepsilon\sigma S}{\rho C_p}\left(T^4 - T_s^4\right)$$

where the following variables [with units] are given as

temperature in the material $T$ [K]
surrounding temperature $T_s = 300$ [K]
axial position along material $z$ [m]
thermal conductivity $k = 401$ [J/K/m/s] (for Cu)
mass density $\rho =$ 8960 [kg/m$^3$] (for Cu)
heat capacity $C_p = 384.6$ [J/kg/K] (for Cu)
Stefan–Boltzmann constant $\sigma = 5.670373x10^{-8}$ [J/s/m$^2$/K$^4$]
gray body permittivity $\varepsilon = 0.15$ (for dull Cu)
surface area to volume ratio $S = 200$ [m$^{-1}$] (for a cylindrical rod of diameter 0.05 m)

A cylindrical Cu rod of diameter 0.05 m and length 0.3 m is initially at $T(z, t=0) = 1000$ K. One end of the rod is maintained at $T(z=0, t) = 1000$ K. The other end of the rod is insulated, $\left.\frac{dT}{dz}\right|_{z=0.3} = 0$ K/m.

(a) Plot the transient behavior.
(b) Find the approximate steady-state temperature in the material at z=0.3 m.

**Solution:**

This is a single non-linear parabolic PDE with one spatial dimension and a Dirichlet boundary condition at $z$=0 and a Neumann boundary condition at $z$=0.3. To solve this problem, I will use the code `parapde_1_anyBC.m`.

I modified the input functions in `parapde_1_anyBC.m` as follows.

I assigned the appropriate type of boundary conditions.

```
BC(1) = 'D';
BC(2) = 'N';
```

I set the final time to 1000 seconds and chose dt to be 0.1 seconds, so I had 10,000 temporal intervals.

```
% discretize time
```

```
to = 0;
tf = 1.0e+3;
dt = 1.0e-1;
```

The rod spans from 0 to 0.1 meter. I set dx to be 0.005 m, so I had 60 spatial intervals.

```
% discretize space
xo = 0;
xf = 0.3;
dx = 5.0e-3;
```

I defined the PDE in the following function.

```
%
%  function defining PDE
%
function k = pdefunk(x,t,y,dydx,d2ydx2);
%
Temp = y;
% rho = density [kg/m^3]
rho = 8960.0;
% Cp = heat capacity [J/kg/K]
Cp = 384.6;
% k = thermal conductivity [W/m/K]
k = 401.0;
%  alpha = thermal diffusivity
alpha = k/rho/Cp;
% length of rod [m]
L = 0.3;
% diameter in [m]
radius = 0.025;
diameter = 2.0*radius;
% surface Area in [m^2]
Area = pi*diameter*L;
% Volume in [m^3]
Volume = pi/4*diameter^2*L;
% surface area to volume ratio
S = Area/Volume;
%  Temperature of the surroundings [K]
Tsurround = 300.0;
% Stefan-Boltzmann constant [J/s/m^2/K^4]
sigma = 5.670373e-8;
% gray body permittivity [dimensionless]
eps = 0.15;
fac = eps*sigma*S/(rho*Cp);
k = alpha*d2ydx2 - fac*(Temp^4 - Tsurround^4);
```

I defined the IC and BCs in the functions below.

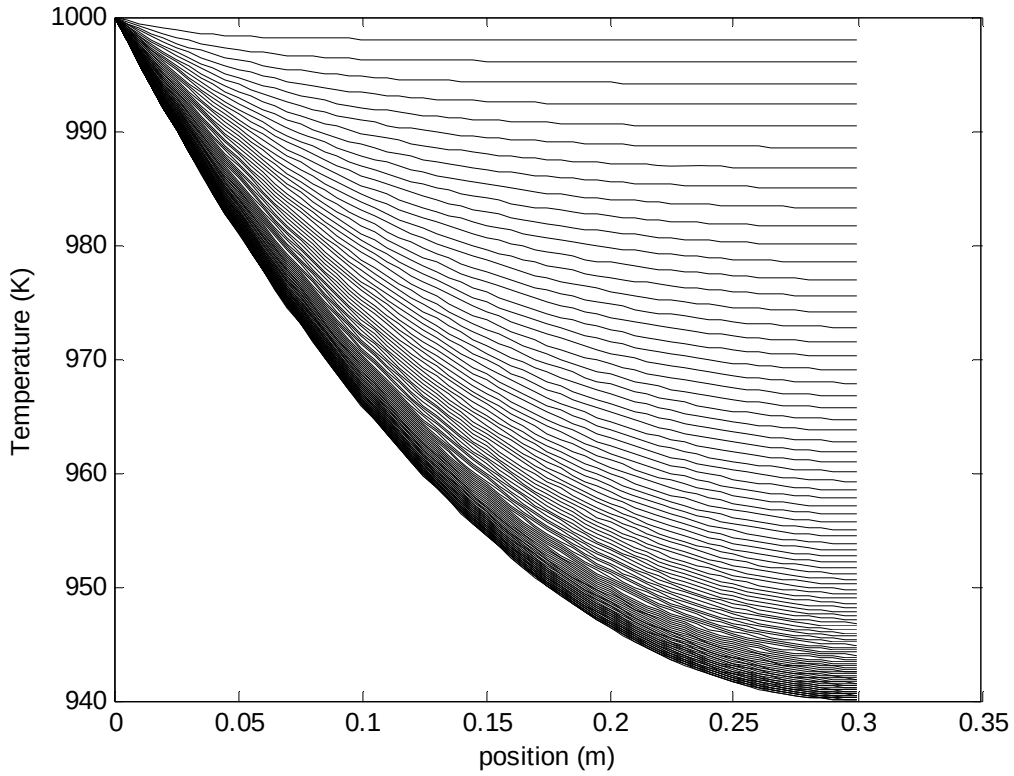
```
%
%  function defining initial condition
%

function ic = icfunk(x);
ic = 1000;

%
```

```
%  functions defining LHS boundary condition
%

function f = aBCo(t);
f = 1;

function f = bBCo(t);
f = 0;

function f = cBCo(t);
f = -1000;

%
%  functions defining RHS boundary condition
%

function f = aBCf(t);
f = 0;

function f = bBCf(t);
f = 1;

function f = cBCf(t);
f = 0;
```

At the command line prompt, I typed

```
[xvec,tvec,Tmat] = parapde_1_anyBC;
```

This command generated the following plot.




To find the last value at x = 0.3 m, I confirmed that I knew the correct spatial and temporal indices.

```
>> xvec(62)
ans = 0.3000

>> tvec(10001)
ans =   1000

>> Tmat(62,10001)
ans =  940.0365
```

Therefore the temperature at the end at 1000 seconds is 940.04 K.

I don't know that this is steady state. I can run the simulation longer. If I change nothing but the final time to 5000 seconds, then I generate the data point

```
>> [xvec,tvec,Tmat] = parapde_1_anyBC_xm02p02;
>> Tmat(62,50001)
ans = 938.7146
```

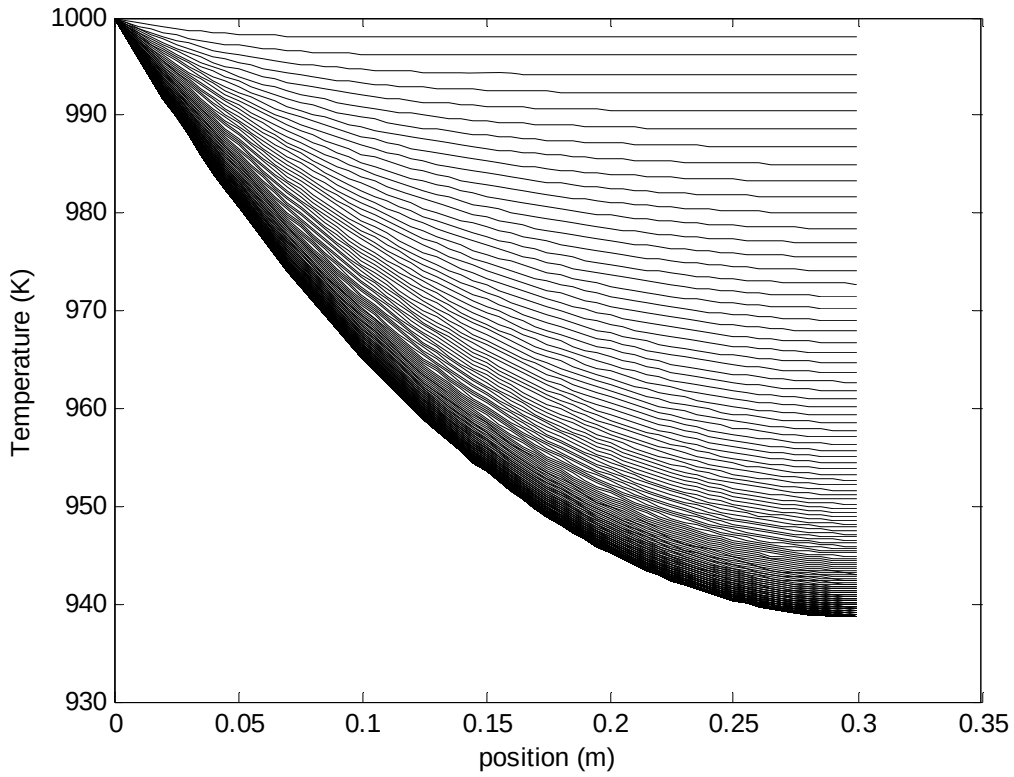
Therefore the temperature at the end at 5000 seconds is 938.71 K.

The two answers agree to two digits, so we are pretty close to the steady state solution, but we can run for a longer time. If I change nothing but the final time to 10000 seconds, then I generate the data point

```
>> [xvec,tvec,Tmat] = parapde_1_anyBC_xm02p02;
>> Tmat(62,100001)
ans =  938.7146
```

Therefore the temperature at the end at 10,000 seconds is 938.71 K. This is the steady state temperature.

Below is the corresponding plot.



## 3. ODE Boundary Value Problem

Consider the following boundary value problem

$$0 = D\frac{d^2C_A}{dx^2} - kC_A$$

with the boundary conditions

$$C_A(x=0) = C_{A_o} = 1.0 \text{ mol/m}^3$$
$$C_A(x=2) = C_{A_f} = 0.1 \text{ mol/m}^3$$

where

$$D = 1.0 \cdot 10^{-5} \text{ m}^2/\text{s}$$
$$k = 1.0 \cdot 10^{-5} \text{ 1/s}$$

(a) Convert this single second-order ODE, to a system of two first-order ODEs.
(b) Plot the solution.
(c) What is the concentration gradient at $x = 0$?

**Solution:**
(a) Convert this single second-order ODE, to a system of two first-order ODEs.

Follow the three step process. First, define new variables.

$$y_1 = C_A \qquad\qquad y_2 = \frac{dC_A}{dx}$$

Second write the ODEs in the new variables.

$$\frac{dy_1}{dx} = y_2$$

$$\frac{dy_2}{dx} = \frac{k}{D}y_1$$

Third, write the conditions in terms of the new variables.

$$y_1(x=0) = C_{A_o} = 1.0$$
$$y_1(x=2) = C_{A_f} = 0.1$$

To solve this BVP, I will use both the code for Newton-Raphson method with Numerical Derivatives for 1 equation (`nrnd1.m`) and the classical 4th-order Runge-Kutta method for N equations (`rk4n.m`).

I modified the input function in `nrnd1.m` as follows:

```
function f = funkeval(x)
xo = 0;
yo_1 = 1;
yo_2 = x;
xf = 2.0;
yf = 0.1;
n = 1000;
[x,y]=rk4n(n,xo,xf,[yo_1,yo_2]);
yf_calc = y(n+1,1);
f = yf_calc-yf;
```

I entered the ODEs in the input file for `rk4n.m` as follows

```
function dydx = funkeval(x,y);
D = 1.0e-5;
k = 1.0e-5;
dydx(1) = y(2);
dydx(2) = k/D*y(1);
```

At the command line prompt, I typed

```
>> [x0,err] = nrnd1(0.5)
```

where 0.5 was my initial guess for the initial slope. This command generated the following output.

```
icount = 1 xold = 5.000000e-01 f = 5.475626e+00 df = 3.626860e+00 xnew = -1.009743e+00  err = 1.000000e+02
icount = 2 xold = -1.009743e+00 f = 2.506051e-12 df = 3.626860e+00 xnew = -1.009743e+00  err = 6.843356e-13

x0 =   -1.0097

err =   6.8434e-13
```

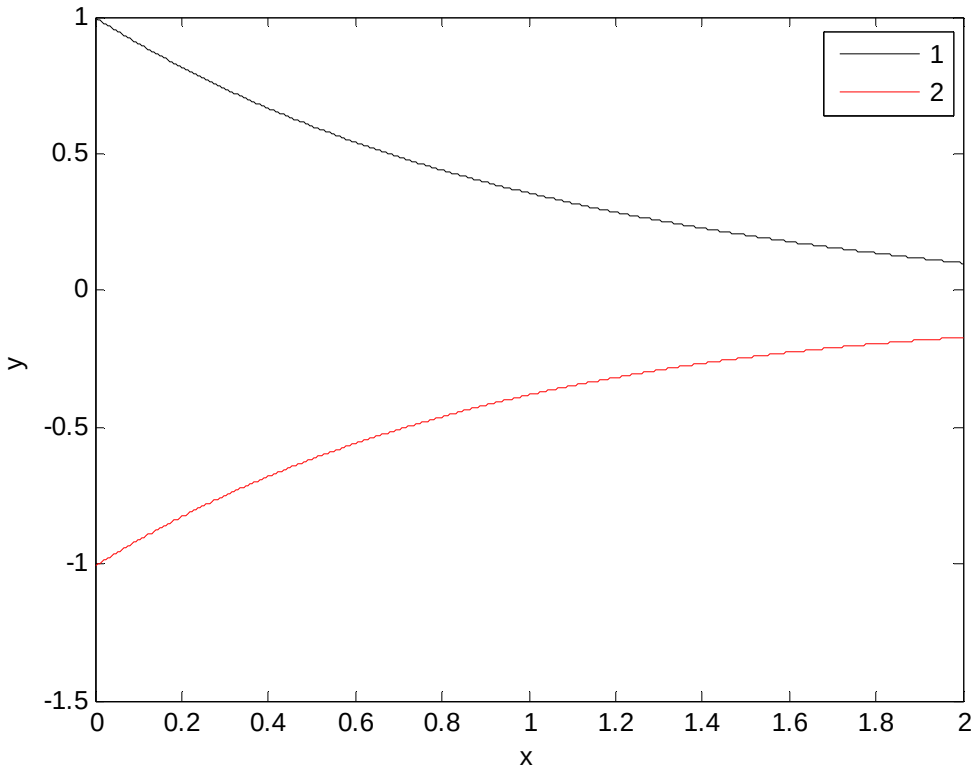
The initial slope is -1.0097.

Not surprisingly this converged in one iteration (with a second iteration required for confirmation) because the ODEs are linear in the unknown, *y*.

The converged solution can be viewed by typing the following command,

```
>> [x,y]=rk4n(1000,0,2,[1,-1.0097]);
```

The resulting figure is provided below.




We can confirm that this is the solution to the boundary condition by checking the value of *y* at the final value of *x*.

```
>> y(1001,1)

ans =    0.1002
```