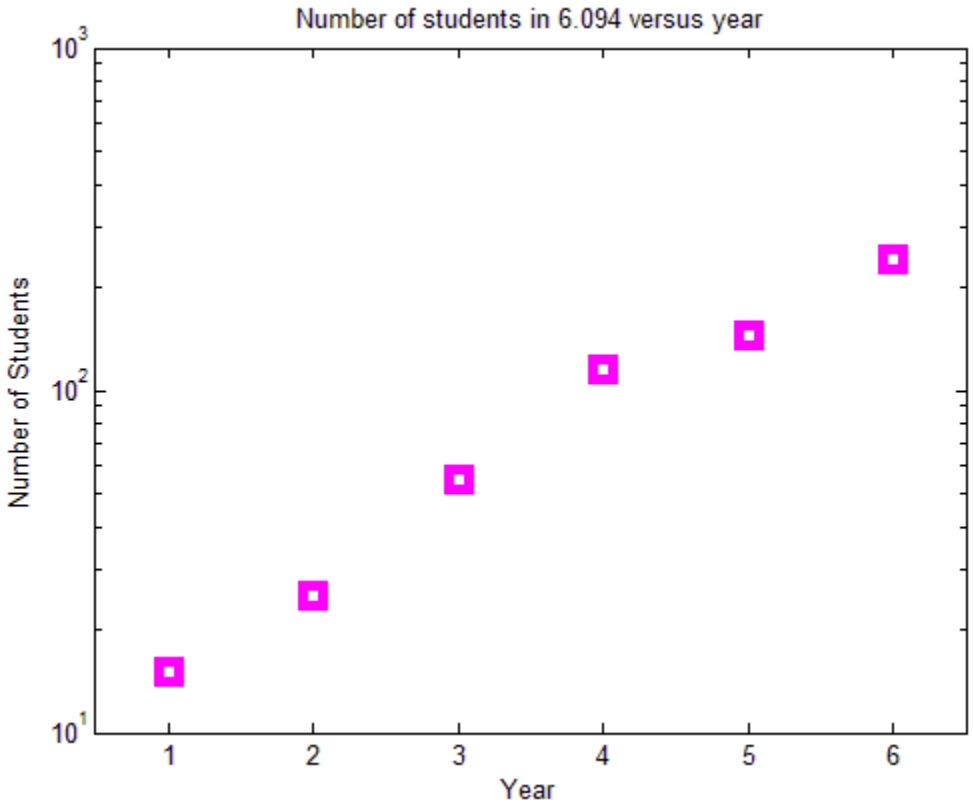
Homework 2 Solutions

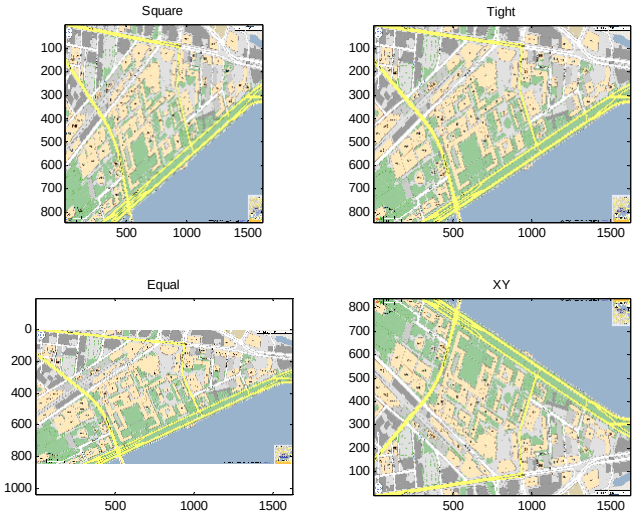
1. The script that generates the desired figure is pasted below:

```
% plotClassSize
% plots the number of students in 6.094 over 5 years on a semilog plot

% open a figure and plot the data
figure
semilogy([1 2 3 4 5 6],[15 25 55 115 144
242],'ms','markersize',10,'linewidth',4);
% change x limits and label it
xlim([0.5 6.5]);
xlabel('Year');
ylabel('Number of Students')
title('Number of students in 6.094 versus year');
```



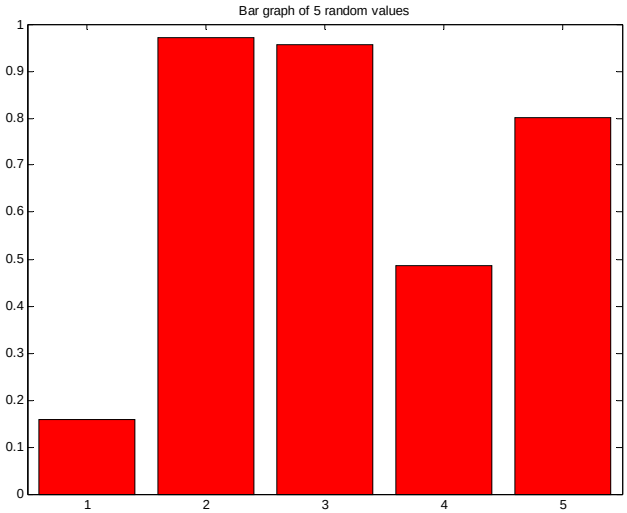
2. The script that generates the desired figure is pasted below:

```
% plotAxisModes
% displays an image in a 2x2 subplot and assigns a different axis mode to
% each panel

% load the data
load mitMap

% make the figure and display each image, changing the axis mode and giving
% it the appropriate title
figure
subplot(2,2,1)
image(mit)
colormap(cMap)
axis square
title('Square')

subplot(2,2,2)
image(mit); % there is only one colormap per figure, so no need to set again
axis tight
title('Tight');

subplot(2,2,3)
image(mit)
axis equal
title('Equal');

subplot(2,2,4)
image(mit)
axis xy
title('XY')
```



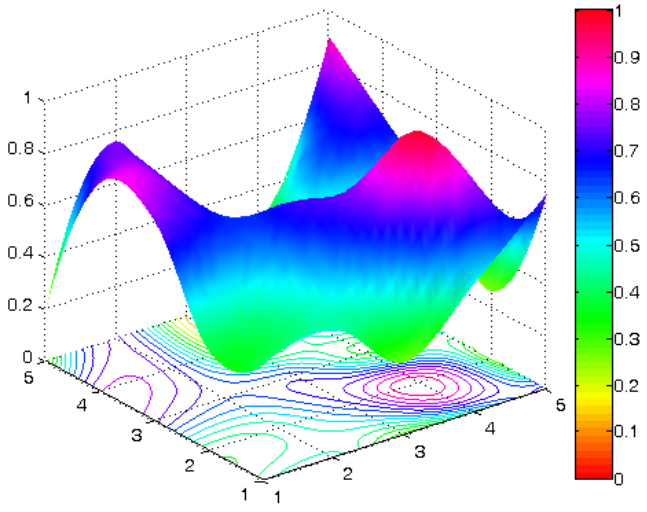
3. The script that generates the desired figure is pasted below:

```
% randomBars
% plots 5 random values as red bars

% open a new figure and plot the bars
figure
bar(rand(5,1),'r');
title('Bar graph of 5 random values');
```



4. The plot that generates the desired figure is pasted below:

```
% randomSurface
% plots a random surface plot

% make the random values and the corresponding X and Y grids
Z0=rand(5);
[X0,Y0]=meshgrid(1:5,1:5);

% make the new X and Y grids using a finer spacing
[X1,Y1]=meshgrid(1:.1:5,1:.1:5);

% interpolate
Z1=interp2(X0,Y0,Z0,X1,Y1,'cubic');

% draw the surface, specify hsv colormap and interpolated shading
figure
surf(X1,Y1,Z1)
colormap(hsv)
shading interp

% plot the contour plot on the same figure
hold on
contour(X1,Y1,Z1,15);

% add colorbar and set caxis
colorbar
caxis([0 1]);
```



To get the smooth 'interpolated' colors to show up when the figure is copied and pasted into a document, use the 'bitmap', and 'force white background' options in 'copy options'.

5. The `findNearest.m` function is pasted below:

```
function ind=findNearest(x,desiredVal)

% calculate the difference from each value in x and the desired value
differences=abs(x-desiredVal);
minDiff=min(differences(:));
% find the index (or indices) where the difference of x and desiredVal is
% at a minimum
ind=find(differences==minDiff);
```

To test the function, we did the following

```
>> x=1:5
x =
     1     2     3     4     5
>> findNearest(x,4.3)
ans =
     4
>> findNearest(x,-5)
ans =
     1
>> findNearest(x,2.5)
ans =
     2     3
>> y=x'*x
y =
     1     2     3     4     5
     2     4     6     8    10
     3     6     9    12    15
     4     8    12    16    20
     5    10    15    20    25
>> findNearest(y,9.1)
ans =
    13
>> y(13)
ans =
     9
>> findNearest(y,6)
ans =
     8
    12
```

6. The code for the function `loopTest.m` is pasted below:

```
% loopTest(N)
%
% loops through the values 1 to N and for each number, displays whether the
% number is divisible by 2, 3, both 2 and 3, or neither

function loopTest(N)

% loop through the numbers
for n=1:N
    if ~rem(n,2) % if divisible by 2
        if ~rem(n,3) % if also divisible by 3
            disp([num2str(n) ' is divisible by 2 and 3'])
        else % divisible by 2 but not 3
            disp([num2str(n) ' is divisible by 2'])
        end
    elseif ~rem(n,3) % if divisible by 3 but not 2
        disp([num2str(n) ' is divisible by 3']);
    else % not divisible by either
        disp([num2str(n) ' is NOT divisible by 2 or 3']);
    end
end
```

The sample output for N=10 is:

```
>> loopTest(10)
1 is NOT divisible by 2 or 3
2 is divisible by 2
3 is divisible by 3
4 is divisible by 2
5 is NOT divisible by 2 or 3
6 is divisible by 2 and 3
7 is NOT divisible by 2 or 3
8 is divisible by 2
9 is divisible by 3
10 is divisible by 2
```

7. and 10. The `rectFilt.m` code below contains all the functionality required for problem 7, as well as the optional problem 10. By reading through the if/else statements, you can figure out which parts are required for question 7 only.

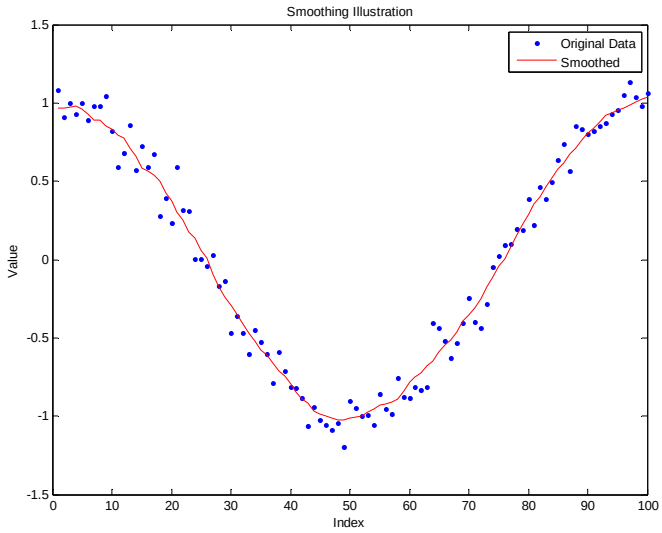
```
% y= rectFilt(x,width)
%
% filters data with a rectangular filter (moving average), and fixes edge
% effects
%
% inputs
% x - a vector of data
% width - the width of the desired rectangle in samples, must be an ODD
integer.
%
% outputs
% y - the filtered version of x
%
% Y= rectFilt(X,width)
% X can be an Nx2 matrix, where the first column are the x values and the
% second column are the y values (for nonuniformly sampled data). in this
% case, width needs to be in the units of the x values and does not need to
% be an integer or odd. Y will also be Nx2 where the first column is again
the
% x values and the second column is the smoothed y values


function x= rectFilt(x,width)

% don't do anything if x is a scalar
if length(x)>1
    % if x is a vector, just filter it with a rectangle
    if size(x,1)==1 || size(x,2)==1

        % if not odd, then display warning
        if rem(width+1,2)
            % width must be odd, so make it odd
            side=floor(width/2);
            width=(side*2)+1;
            disp(['Width should be odd, using ' num2str(width) ' instead']);
        else
            side=floor(width/2); % calculate side for later
        end

        % convolve the mask with the signal
        mask=1/width*ones(1,width);
        x=conv(mask,x);
        % get rid of extra values
        if nargin==2
            x=x(ceil(width/2):end-floor(width/2));
            % fix edge effects
            leftEdge=(side+1)/width:1/width:1;
            rightEdge=1:-1/width:(side+1)/width;
        end
```

```
        % fix the orientation of these if necessary
        if size(x,1)>size(x,2)
            leftEdge=leftEdge';
            rightEdge=rightEdge';
        end
        x(1:length(leftEdge))=x(1:length(leftEdge))./leftEdge;
        x(end-length(rightEdge)+1:end)=x(end-
length(rightEdge)+1:end)./rightEdge;
    else
        % it's not uniformly sampled, so pull out the x and y values and
        % smooth accordingly. sortrows first to get all the x values in
        % increasing order
        x=sortrows(x);
        y=x(:,2);
        x=x(:,1);

        temp=zeros(size(y));
        for n=1:length(temp)
            % find the points that are within the width from the current
            % point
            start=find(x>x(n)-width/2,1);
            stop=find(x<x(n)+width/2,1,'last');
            temp(n)=mean(y(start:stop));
        end
        % assign temp to x since x is what's returned
        x=[x temp];
    end
end
```

The script below runs the `rectFilt.m` program on the data in noisyData.mat and plots the results.

```
% testRectFilt
% loads the noisy data and filters it using rectFilt, and then plots the
% data and smooth curve

% load the data
load noisyData

% smooth it
smoothed=rectFilt(x,11);

% plot the data and smooth line
figure
plot(x,'.');
hold on
plot(smoothed,'r')
legend('Original Data','Smoothed')
xlabel('Index')
ylabel('Value')
title('Smoothing Illustration')
```

The figure of noisyData and the smoothed data generated by the script `testRectFilt.m` is below:

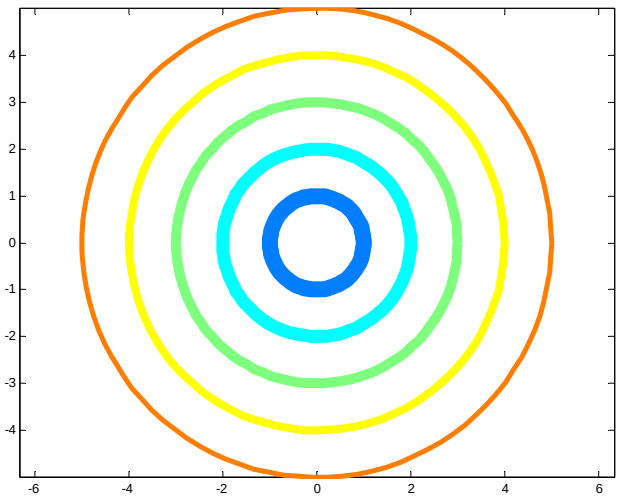


8. The `getCircle.m` code is pasted below:

```
% [x,y]=getCircle(center,r)
%
% returns the x and y coordinates of a circle with its center at center
% (2 element vector) and radius r

function [x,y]=getCircle(center,r)

% make the time vector and evaluate at sin and cos to get a circle
t=linspace(0,2*pi,100);
x=center(1)+r*cos(t);
y=center(2)+r*sin(t);
```

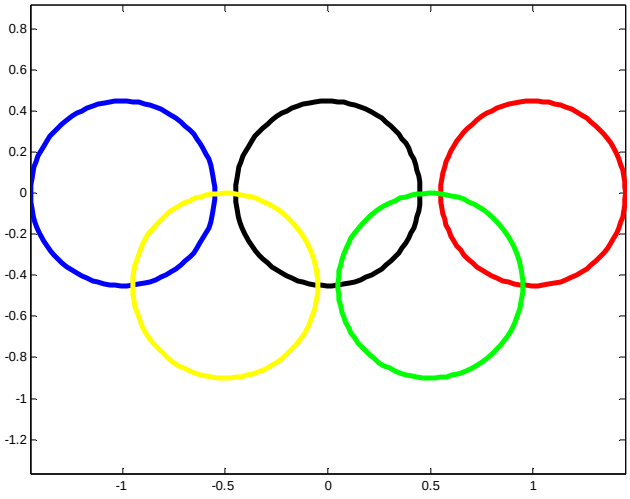
The script concentric.m is pasted below:

```
% concentric
% plots 5 concentric circles with 5 different colors

% make the 5 colors using a jet colormap
colors=jet(5);

figure

for n=1:5
    [x,y]=getCircle([0 0],n);
    plot(x,y,'linewidth',14-2*n,'color',colors(n,:));
    hold on;
end

axis equal
```



The `olympic.m` script is pasted below:

```
% olympic
% plots the olympic logo

r=0.45;

figure
[x1,y1]=getCircle([-1 0],r);
plot(x1,y1,'linewidth',4,'color','b');
hold on
[x2,y2]=getCircle([0 0],r);
plot(x2,y2,'linewidth',4,'color','k');
[x3,y3]=getCircle([1 0],r);
plot(x3,y3,'linewidth',4,'color','r');
[x4,y4]=getCircle([-.5 -r],r);
plot(x4,y4,'linewidth',4,'color','y');
[x5,y5]=getCircle([.5 -r],r);
plot(x5,y5,'linewidth',4,'color','g');
axis equal
```



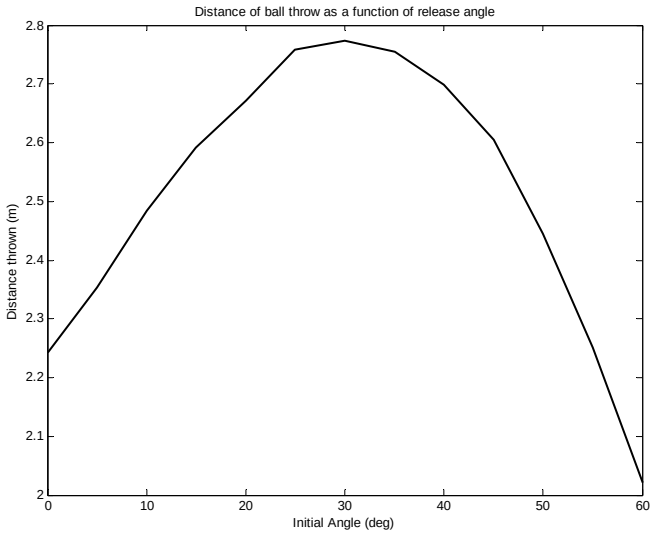
9. The `throwBall.m` function is pasted below

```
% distance=throwBall(v,theta)
%
% this is a function that throws a ball at a specified angle
% and with a specified initial velocity and calculates where the ball
% lands.
%
% v is the initial velocity in m/s
% theta is the angle in DEGREES (assume throwing ball to the right)
%
% distance is the distance in meters at which the ball hits the ground

function distance=throwBall(v,theta)

% define the constants
h=1.5; %meters
g=9.8; %gravitational acceleration in m/s^2


% make a time vector
t=linspace(0,10,1000);

% calculate the x and y positions as a function of time
x=v*cos(theta/180*pi)*t;
y=h+v*sin(theta/180*pi)*t-1/2*g*t.^2;

% find when it hits the ground
ind=find(y<0,1,'first');
distance=x(ind);

% if the ball doesn't hit the ground, display the warning
if isempty(distance)
   disp('The ball does not hit the ground in 10 seconds');
   distance=nan;
end
```

The `testBall.m` script is pasted below

```
% testBall
% test the throwBall function and plot distance as a function of angle

theta=0:5:60;
for n=1:length(theta)
    % calcualte the distance for each angle
    dist(n)=throwBall(4,theta(n));
end

% plot it
figure
plot(theta,dist,'k','linewidth',1.5);
xlabel('Initial Angle (deg)');
ylabel('Distance thrown (m)');
title('Distance of ball throw as a function of release angle');
```

The figure below is generated by `testBall.m`

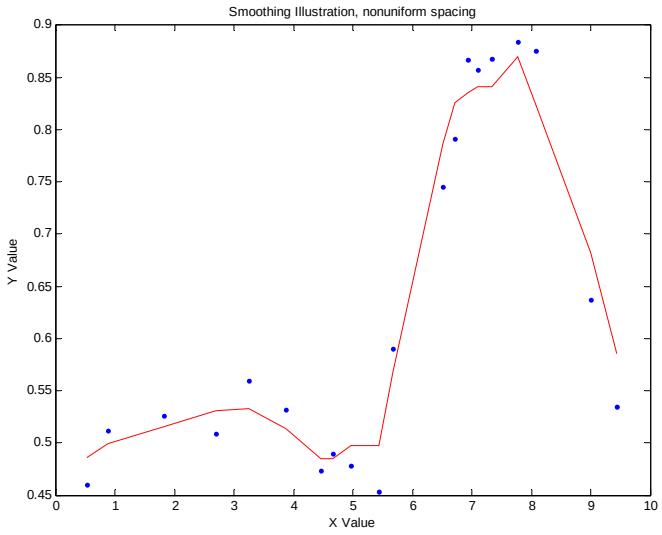


10. See the solution to problem 7.

The script that tests this function is pasted below:

```
% testRectFilt2
% runs the rectFilt program on nonuniformly sampled data and plots the
% result

% load the data
load optionalData

% smooth it
smoothed=rectFilt(x,2);

% plot the original and the smoothed data
figure
plot(x(:,1),x(:,2),'.');
hold on
plot(smoothed(:,1),smoothed(:,2),'r')
xlabel('X Value')
ylabel('Y Value')
title('Smoothing Illustration, nonuniform spacing');
```

The figure below was generated by the `testRectFilt2.m` script.



11. The `tradeStock.m` function is pasted below

```
% endValue=tradeStock(initialInvestment,price,buy,sell)
%
% calculates the total value of an investment in a particular stock
% initialInvestment - dollar amount of initial cash
% price - a vector of prices as a function of time
% buy - a vector of indices at which the stock should be bought
% sell - a vector of indices at which the stock should be sold
%
% endValue - the value of the initial investment at the end of the
% specified buys and sells

function endValue=tradeStock(initialInvestment,price,buy,sell)

% this is the cost of each transaction
transactionCost=12.95;

% initially, all your money is cash, and you have 0 shares
cash=initialInvestment;
shares=0;
lastBuyPrice=0; % remember the price at which you last bought it

% sort all the transaction days. put a 1 next to all the buys and a -1 next
% to all the sells
transactionDays=[[buy(:) ones(length(buy),1)];[sell(:) -
1*ones(length(sell),1)]];
transactionDays=sortrows(transactionDays);

% now all the transaction days are sorted so we can go through them
for n=1:size(transactionDays,1)
    % see what day it is
    day=transactionDays(n,1);
    % get if it's a buy or sell day
    kindOfTransaction=transactionDays(n,2);
    % get the current price
    currentPrice=price(day);

    % if it's a buy day, then buy as much as you can, accounting for the
    % transaction cost
    if kindOfTransaction==1
        numToBuy=floor((cash-transactionCost)/currentPrice);
        if numToBuy>0 % only buy if you can buy some
            % buy them
            cash=cash-numToBuy*currentPrice-transactionCost;
            shares=shares+numToBuy;

            % remember the price you paid
            lastBuyPrice=currentPrice;
        end
    else % it's a sell day
        if shares>0 % only sell if you have shares to sell
            % profit is the money you get by selling minus the money you
```

```
            % paid to get the shares
            profit=(currentPrice*shares-transactionCost)-(lastBuyPrice*shares
+transactionCost);
            if profit>0 %only sell if it makes sense
                % exchange the shares for cash at today's price, paying the
                % transaction cost
                cash=cash+currentPrice*shares-transactionCost;
                shares=0;
            end
        end
    end
end

% calcuate the end value
endValue=shares*price(end)+cash;
```

The following script runs the `tradeStock.m` function with a few initial investments and the Google stock data.

```
% testTradeStock
% runs the tradeStock program with the google data for a few initial
% investments

% load the google data
load googlePrices

% run the program with a few initial investments
endValue=tradeStock(100,price,lows,peaks);
disp(['With an initial investment of $100, the end value is '...
    num2str(endValue)]);

endValue=tradeStock(100000,price,lows,peaks);
disp(['With an initial investment of $100,000, the end value is '...
    num2str(endValue)]);

endValue=tradeStock(2000,price,lows,peaks);
disp(['With an initial investment of $2,000, the end value is '...
    num2str(endValue)]);
```

The above script generates the following output:

```
>> testTradeStock
With an initial investment of $100, the end value is 100
With an initial investment of $100,000, the end value is 61231407.15
With an initial investment of $2,000, the end value is 1017523.52
```