Problem Set 1 Matlab Tutorial
10.34 Fall 2005, KJ Beers

Ben Wang, Mark Styczynski

September 16, 2005
Modified September 23, 2005 to reflect points and common errors

These assignments are designed to familiarize or refresh you with basic MATLAB commands and operations. There are many ways to go accomplish these tasks. Some are more involved and others less so. Thus the problem statements are quite open-ended in nature and many solutions can be expected and accepted as correct. The goals remain the same though: navigating through MATLAB's wide variety of functions and operations, and should be emphasized.

Furthermore we are looking to emphasize the coding procedure that Prof. Beers emphasize in class. Obviously many of these problems do not lend themselves to this method, as many of these problems do not require much code at all, but it is a good habit to start practicing for when your code becomes more complex.

Problem 1: Vectors (2 points)

The problem asks you to plot the function:

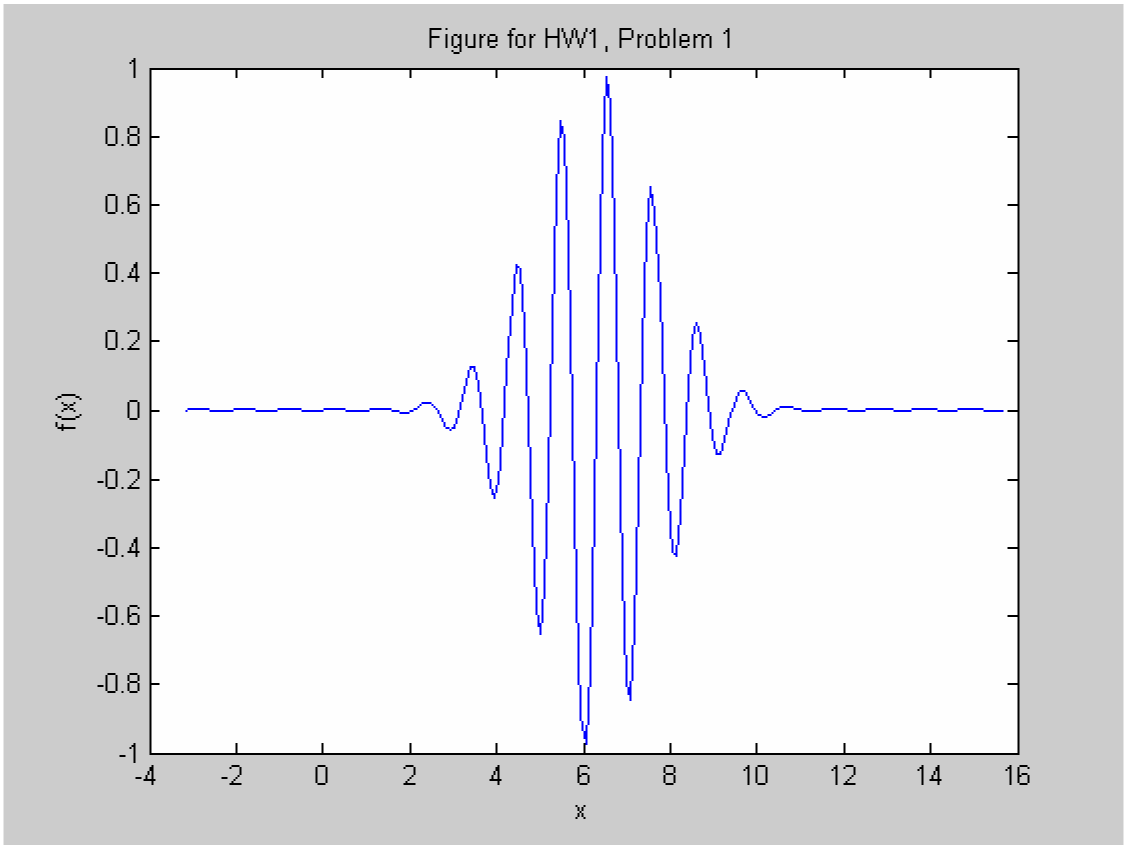
$$f(x) = \exp\left(\frac{(x-2\pi)^2}{4}\right)\sin(6x)$$

and add some labels and a title. This can be accomplished in the following code:

```
% benwang_HW1_P1.m
% Ben Wang
% HW#1 Problem #1
% due 9/14/05 9 am

% This purpose of these assignments are to review basic matlab operations,
% syntax and implementation.  Design concepts involving Steve O'Connel's
% {Code Complete} and the PDL method

% Problem #1 requires you to plot out the equation f(x) =
% exp(-(x-2pi)^2/4)*sin(6x) over a the domain [-pi, 5*pi]
% How we will go about solving this will be to specify a vector containing
% the values of x.  We then create a equally sized vector to contain the
% values of f(x) after solving for the value.  Finally we will plot the two
% vectors against each other and add labels, etc.

% Sys: data structure with information about the problem
%   Sys.x_lo: will be the low end: -pi
%   Sys.x_hi: will be the high end: 5*pi
% Grid: data structure that defines information about the spatial grid:
%   Grid.N: this defines the number of points, and thus the length of each
%       vector

% x: the vector containing points from -pi to 5 pi
% f_x: the resulting values of f(x) at each point along x

% ======= main routine benwang_HW1_prob1.m

function iflag_main = benwang_HW1_P1();

iflag_main = 0;

%PDL> clear graphs, screen etc. general initialization

clear all; close all; clc;

%PDL> specify lower and upper bounds of x

Sys.x_lo = -pi;
Sys.x_hi = 5*pi;

%PDL> specify how many points (resolution) in grid

Grid.N = 200;

%PDL> create a vector x with specified bounds and grid resolution

x = linspace(Sys.x_lo,Sys.x_hi,Grid.N);

%PDL> initialize a vector f_x with the same size as x

f_x = zeros(0, Grid.N);

%PDL> create a for loop to calculate values for f_x for each value of x,
%using i as an index.  First initialize i = 0.  Then for i< Grid.N, do a
%calculation.

for i = 1:Grid.N
    f_x(i) = exp(-1/4*(x(i)-2*pi)^2)*sin(6*x(i));
end

%PDL> now we invoke some plotting commands to display f_x graphically

figure;
plot(x,f_x);

%PDL> now we add axis labels and a title
xlabel('x');
ylabel('f(x)');
title('Figure for HW1, Problem 1');
```

```
iflag_main = 1;

return;
```

Your plot should look something like this:



Problem 2: Log2 (1.5 points)

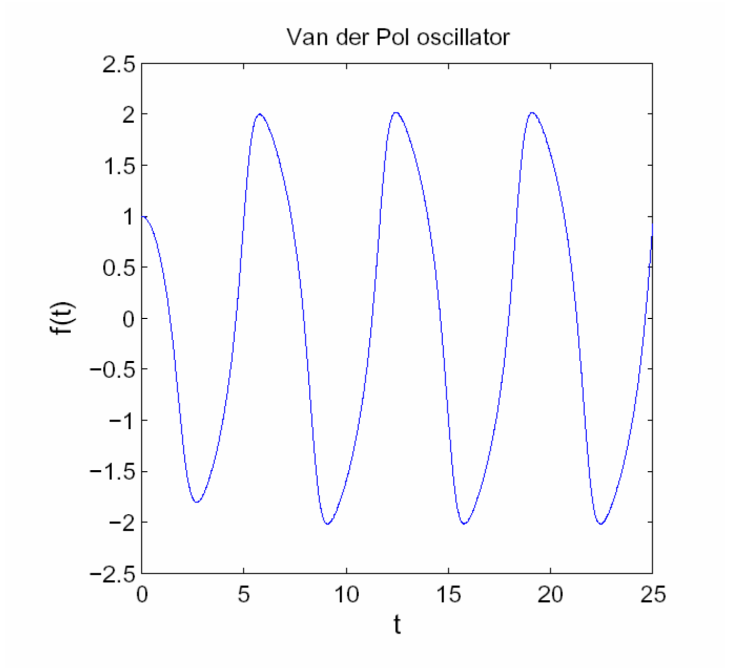
We are asked to make use of the log2(x) function in MATLAB to write a simple routine to check if an inputted integer is a power of 2. This can be accomplished with the following code:

```
% benwang_HW1_P2.m
% Ben Wang
% TA for 10.34
% HW#1 Problem #2
% due 9/14/05 9 am

% Problem #2 requires you to write a simple matlab code that takes a
% numerical user input and determines whether or not if it is a power of 2.
% It should return 1 if the input number is a power of 2 and 0 if it is
% not.
% ======= main routine benwang_HW1_prob2.m

function iflag_main = benwang_HW1_P2();

%PDL> clear graphs, screen etc. general initialization

clear all; close all; clc;

%PDL> Take user input and store it's value in the variable x

x = input('Please enter a whole number and I will check to see if it is a power of 2: ');

%PDL> Take the value of log2(x) and check to see if the result is an
%integer.  We mod the value of log2(x) and check to see if the remainder is
%greater than 0.  We will store this value in the variable y

y = mod(log2(x),1);

%PDL> Take the result and convert to a text based response using a simple
%if logic test combined with some concatentation of strings.

if (y==0)
   phrase = ['Yes ', int2str(x), ' is a power of 2'];
   disp(phrase);
else
   phrase = ['No ', int2str(x), ' is not a power of 2'];
   disp(phrase);
end

return;
```

Problem 3: Van der Pol (0.5 points)

For this problem, you are simply asked to run some code that Prof. Beers had written to solve a second order differential equation using the command **ode45**, which you will make ample use of later in the course. He shows an example of converting a second order ODE into a two coupled first order ODEs which is a very useful technique and it will be worthwhile to come back to this in a few lectures.

Ultimately you should get a plot that looks something like:



Problem 4: String Search (2 points)

This problem asks you to write a function that can search for matched strings using the find() function. There are many ways to implement this problem. In fact, a solution using the findstr function will be considered correct. The following is just the code for one method of doing this:

```
% benwang_HW1_P4.m
% Ben Wang
% TA for 10.34
% HW#1 Problem #4
% due 9/14/05 9 am

% This purpose of these assignments are to review basic matlab operations,
% syntax and implementation.  Design concepts involving Steve O'Connel's
% {Code Complete} and the PDL method

% Problem #4 requires you to write some code to do some DNA sequence
% matching.  This utilizes the function find, taking in some input and a
% search criteria.  The function should then return the integer locations
% of the matched sequences.

% ======= main routine benwang_HW1_prob4.m

function iflag_main = benwang_HW1_P4();

%PDL> clear graphs, screen etc. general initialization

clear all;close all;clc;
```

```
%PDL> Specify DNA sequences, and search sequences

x = 'ACGTAAAACGTAGG';
xs = 'ACGT';

x2 = 'AAAATGATATGAAAAATGATGATGAAAAA';
xs2 = 'AAAA';

%PDL> call a subroutine search_sequence(x,xs) which will return the values
%of the integer locations of where matches occur and store these values in
%an array called index_master.  Call this function twice for the two
%different DNA sequences and match criteria.

index_match = search_sequence(x,xs);
index_match2 = search_sequence(x2,xs2);

index_master = find(index_match)
index_master2 = find(index_match2)

return;

% ========= subroutine search_sequence(sequence, search_criteria)

function match = search_sequence(seq,search)

%PDL> determine how long passed DNA sequence is and store in variable
%y

y = length(seq);

%PDL> determine how long the passed search criteris is and store in
%variable z

z = length(search);

%PDL> initialize an array called match that can store a value whenever a
%match is made.  The total length of this row will be =  y - z + 1

match = zeros(y-z+1,1);

%PDL> iteratively step through from left to right to check if there is a
%match with the sequence.  If there is a match, then record the value as 1.
%We will also create a temporary array, temp, to store the result of each
%sequential analysis

for i=1:length(match)
    temp = zeros(length(search),1);
    for j=1:length(search)
        if (eq(search(j),seq(i+j-1)))
            temp(j) = 1;
        else
            j = length(search);
        end
        end
        if (sum(temp) == length(search))
            match(i) = 1;
        else
        end
    end

    return;
```

If you want to get real sexy and elegant, rather than brute force it, there are multiple ways you can do it. Here Mark has written two ways to do it (one is commented out, but both work) in four lines:

```
    function locVector = search_sequence(haystack,needle)

    % intermediate = [];
    % for i=1:length(haystack) - length(needle) + 1,
    %     intermediate(i) = mean(haystack(i:i+length(needle)-1)==needle);
    % end
    %
    % locVector = find(intermediate == 1);

    locVector = [];
    for i=1:length(haystack) - length(needle) + 1,
        if length(find(haystack(i:i + length(needle) - 1)==needle)) == length(needle),
            locVector(length(locVector) + 1) = i;
        end
    end
```

Problem 5: What in the world are tacks? (4 points)

This problem is designed to work with some of the functions that MATLAB involves switching between polar and Cartesian coordinates and coding up some simple logic. These are used in the context of vector analysis (to be differentiated from the terminology of vector in linear algebra).

We are asked to write a program that involves helping a sailor figure out their tacks when faced against the wind. If they are outside a particular no sail zone, they can sail directly to their destination of choice. If they are within the no sail zone, they have to travel towards their destination, close-hauled, which is staying just outside of +/- 40 degrees of the direction along the wind.

Basically the problem asks for any solution, not an optimized one. If you can come up with any feasible solution that returns realistic direction of tacks, number of tacks, the total distance traveled on this tacked path, and the percentage increase in distance traveled over a direct straight path, you will receive full credit.

This can be accomplished in some code that looks like:

```
% benwang_HW1_P5.m
% Ben Wang
% TA for 10.34
% HW#1 Problem #5
% due 9/14/05 9 am

% Problem #5 involves writing a program that takes user inputs regarding
% wind speed, sailing bearing, and distance to the final location and
% determines how to sail to the destination.  Remember this is not an
% optimization problem.  There are multiple ways of solving this problem
% and all we are looking for is

% User: data structure that stores user inputed data
%   User.w stores the angle of the wind direction
%   User.d stores the distance required to sail (knots)
%   User.T stores the max time desired to tack (minutes)
%   User.B stores the bearing to the destination (degrees)

% Parameters: data structure that includes other problem related information
%   Parameters.speed stores the value of boat speed
%   Parameters.nosail stores the angle value, to the right and left, that
%   encompasses the no-sail zone

% ======= main routine benwang_HW1_prob5.m

function iflag_main = benwang_HW1_P5();

%PDL> clear graphs, screen etc. general initialization

clear all; close all; clc;

%PDL> Get user inputs for wind, distance, bearing, and time.  We will store
%these values in radians, but ask the user in degrees.  We will treat north
%at 90 degrees

User.w = input('Enter the direction of the wind (in degrees, 0 = E, 90 = N, etc.): ');
User.d = input('Enter the distance to the destination (in knots): ');
User.T = input('Enter the time you want to spend on any one tack (hours): ');
User.B = input('Enter the bearing to the destination (in degrees, 0 = E, 90 = N, etc.): ');

%PDL> Initialize other parameters for the problem

Parameters.speed = 6;                          % knots/hr
Parameters.nosail = 40;                        % degrees
Parameters.tackdist = User.T*Parameters.speed;  % distance traveled on any one tack

%PDL> We are now going to adjust our axis such that the wind is always
%coming from 180 degrees.  So if the wind comes at us from 45 degrees, or
%specifically from the NE, we will create an angle adjustment of 45 degrees
%and shift our bearing by that much.

User.B = (abs(mod(User.B - User.w,360)))*pi/180;        % adjust to radians
```

```
%PDL> Now our axis is aligned such that our no sail region is +/-
%Parameters.nosail around 180 degrees.  We assume our destination is at the
%origin.

loc_final = [0 0];

%PDL> We will define the our initial location in a Cartesian axis, despite
%having information in polar coordinates
%This is accomplished by a MATLAB function, pol2cart.  We will store the coordinates in
a row vector
%loc_init.  The dimension this Cartesian axis will be in knots.

[loc_init(1), loc_init(2)] = pol2cart(User.B, User.d); % destination

%PDL> Create a plot to see our trajectories

figure;
hold on;

%PDL> Call a loop that unless we are close to our destination to within
%Parameters.tolerance to keep making a step that brings us closer to our
%destination.  Call a subroutine to figure out if we are within the no-sail
%zone and if we are how many steps, etc. to get out.

numsteps = 0;       % initialize number of steps
totaldist = 0;      % initialize total distance traveled

while (norm(loc_final-loc_init) > 0)

   % call a function to make a step if we need to tack
   [newbearing,ministep,travel] = sail_away(Parameters,User,loc_init,loc_final);

   % increment the number of steps
   numsteps = numsteps + 1;

   % plot each step of the path from initial location to destination
   plot([loc_init(1) loc_init(1) + ministep(1)], [loc_init(2) loc_init(2) + ministep(2)], '-o');

   % update the current location
   loc_init = loc_init + ministep;

   % check to see if we have arrived at the destination to exit while
   % loops
   if newbearing==ministep
       loc_init = loc_final;
   end

   % update totaldistance
   totaldist = totaldist + travel;
end

axis square;
xlabel('WE (nautical miles)');
```

```
ylabel('SN (nautical miles)');

numsteps
totaldist
percentincrease = 100*(totaldist/User.d-1)

return

%========= subroutine sail_away(Parameters,User,init,final)

% This function takes the two data structures of the program, the current
% location of the boat, and the final destination and determines how to
% make a single "tack."  This function will also determine whether or not
% we are in the no-sail zone.  If we are in the no-sail zone, it will make
% a few direct steps

% For ease of use, we will use the same names for the data structures
% Parameters and User

% init will store current location of the boat and final will store the
% destination

function [minibearing, del_bearing, distance] = sail_away(Parameters, User, init, final)

distance = 0;                   % initialize distance traveled on current step
minibearing = final - init;     % this calculates the most recent bearing to destination
[theta, r] = cart2pol(minibearing(1), minibearing(2));

if theta <=0
   theta = -round(theta*180/pi);
else
   theta = round(theta*180/pi);
end
if ((theta >= (180 + Parameters.nosail))|(theta <= (180 - Parameters.nosail)))
   % you are outside of the no-sail zone, sail directly there
   disp('you are outside the no-sail zone.  You have a direct path');
   del_bearing = minibearing;
   distance = (del_bearing(1)^2+del_bearing(2)^2)^0.5;
else
   % you are inside the no-sail zone
   disp('you need to tack');
   del_bearing = [0,0];
   % the following if statement just makes you tack left or right
   % sometimes based on some arbitrary logic of if theta >= 180 or not.
   % It does not really accomplish what you would do in real life, but for
   % our problem it is an adequate definition
   if (theta >= 180);
      disp('Tack Left');
      left_angle = (180 + Parameters.nosail)*pi/180;
      [del_bearing(1), del_bearing(2)] = pol2cart(left_angle, Parameters.tackdist);
      distance = (del_bearing(1)^2+del_bearing(2)^2)^0.5;
      minibearing = minibearing - del_bearing;
   else
      disp('Tack Right');
```

```
            right_angle = (180 - Parameters.nosail)*pi/180;
            [del_bearing(1), del_bearing(2)] = pol2cart(right_angle, Parameters.tackdist);
            distance = (del_bearing(1)^2+del_bearing(2)^2)^0.5;
            minibearing = minibearing - del_bearing;
        end
    end

    return;
```

Note that this is just **one** way of doing this and there are many, many more. You are not asked to plot anything, just return output such as:

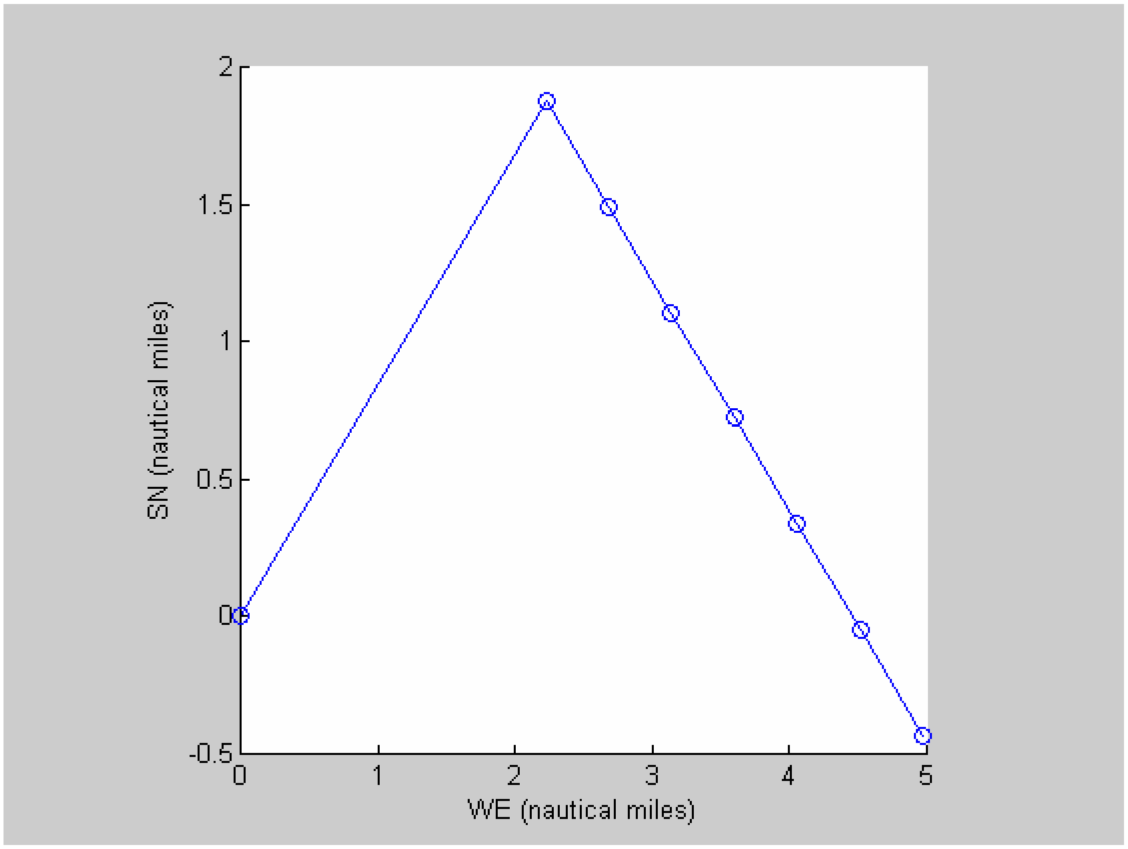
```
Enter the direction of the wind (in degrees, 0 = E, 90 = N, etc.): 40
Enter the distance to the destination (in knots): 5
Enter the time you want to spend on any one tack (hours): 0.1
Enter the bearing to the destination (in degrees, 0 = E, 90 = N, etc.): 35
you need to tack
Tack Right
you need to tack
Tack Right
you need to tack
Tack Right
you need to tack
Tack Right
you need to tack
Tack Right
you need to tack
Tack Right
you are outside the no-sail zone.  You have a direct path

numsteps =

     7


totaldist =

    6.5104


percentincrease =

   30.2083
```

You are not required to plot a graph or anything, but it is helpful to look at your trajectory. Note the destination is located at the origin (0,0).

This was the tough problem on the problem set and a typical 'functional' program where you would have to implement some logic, control data, and utilize different matlab functions.

The major errors that occurred in this assignment were:

- account for the symmetry between 0 and 360 degrees. This can be fixed with a modulo operator. 1 point was deducted if this was not accounted for
- If the output was incorrect or incomplete, 1 point was deducted.
- If there were minor coding errors resulting in a small bug that needed to be fixed between 0 and 1 point was deducted.
- If there were major coding errors resulting in partial functionality 2 points were deducted.
- If the major operational portions of the code were missing, then 4 points were deducted.


SN (nautical miles)
WE (nautical miles)