Appendix: Functions

Table of Contents

QUESTION 1

Functions used in question 1

```
% Q1.1 FC

function rtn = fc(c)
x=c(1) ;
y=c(2) ;
rtn = (x-2).^2 + 2*(y-3).^2 ;

% Q1.1 DFC

function rtn = dfc(c)
x=c(1) ;
y=c(2) ;
rtn = [2.*(x-2)  4.* (y-3)] ;

% Q1.1 GRADIDESC

function [soln,path,path2,path3x,path3y,path3z] = graddesc(fc, dfc,
 i,e, t)
% This function finds the minimum of a function(fc) using gradient
 descent
% method and returns it together with the gradient descent path
 followed.
%===Define=imputs=====
% fc -- function fed as a function
% dfc-- gradient fed as a function
% i -- initial guess
% e -- step size
% t -- tolerance
%===Define=outputs======
%soln contains the x and y coordinates of the minimum
% path is a cell array to store path in a m by 2 cell array where each
 row is a separate data point (not used in the end)
% path2 ia sn vector array to store path as a series of values one
 after the other (not used in the end)
% path3x is an vector array to store the x-coordinate components of
 path as a series of values
% path3y is a vector array to store the y-coordinate components of
 path as a series of values
```

```
% path3z is a vector array to store the z-coordinate components of
 path as a series of values


%===initate=variables====
gi = feval(dfc,i) ; %cals value of function dfc(i)
path={i,feval(fc,i)}; %creates a cell to store path in a m by 2 cell
 array where each row is a separate data point (not used in the end)
path2=[i,feval(fc,i)]; %creates an array to store path as a series of
 values one after the other (not used in the end)
path3x=[i(1)]; %creates an array to store the x-coordinate components
 of path as a series of values
path3y=[i(2)]; %creates an array to store the y-coordinate components
 of path as a series of values
path3z=feval(fc,i); %creates an array to store the z-coordinate
 components of path as a series of values
c=0; %initialise counter to know how many loops have been performed
 and consequently know expected size of path

%===perform gradient descent=====

while(norm(gi)>t)  % crude termination condition idea is that gradient
 eventually approaches zero and that is the point very near the
 minimum of the function
  i = i - e .* feval(dfc,i) ;
 gi = feval(dfc,i) ;
 c=c+1;
 path{c+1,1}=i;
 path{c+1,2}=feval(fc,i);
 path2=[path2,i,feval(fc,i)];
 path3x=[path3x,i(1)];
 path3y=[path3y,i(2)];
 path3z=[path3z,feval(fc,i)];
end
soln = i;

% Q1.2 Least Square error

function [solution,path]=myLSE(A,b,guess,step,tol)

% function takes inputs defined below and computes Least Square error
% solution by gradient descent. Solution is that combination of X's
% which minimises sum of error squared, where error is defined as:
 e=Ax-b
%
%------INPUTS-----
%A is coefficinets matrix of size m*n
%b is the corresponding solutions vector of size m*1
%guess is a vector of initial guess of values of x (size m by 1)
%step is the step size of the gradient descent
%tol is the tolerance and dictates when function will stop iterating
%i.e. what diff. betweeen predcicted and observed values are we ok
 with?
```

```
%------OUTPUTS-----
%Function outputs a column vextor of size m containing the LSE
 solution
%Optionaly a path metrix can also be outputed which contains all the
%iterations of teh algorithm, i.e. the path of the gradient descent

g = 2*A'*A*guess - 2*A'*b;
fc = (A*guess - b)'*(A*guess -b);
temp = [guess', fc];
while(norm(g)>tol)  % crude termination condition
   guess = guess - step.* g;
   g = 2*A'*A*guess - 2*A'*b;
   fc = (A*guess - b)'*(A*guess -b);
   temp = [temp; guess', fc];
end
path = temp;
solution=[path(end,1:end-1)];
```

QUESTION 2

Functions used in question 2

```
% Q2.1 MYpolynom

function [sol] = MYpolynom(x,k)
%function fits polynomial of order k to vector of x and outputs a
 vector of
%all instances of x^k. Corrspodns to: [1, x, x^2, x^3,...,X^k]

%sanity check:
if k<1
    disp('polynomial degree needs to be larger than one');
else
%function
    sol=[];
    for i=1:k
        sol=[sol, x.^(i-1)];
    end
end


% Q2.2 G of X

function [sol] = Gofx(x,mean,sd)
%Gofx(x,sd) takes as imputs a matrix.vector or value of X and the
 standard
%deviation and outputs the function G of x

sol = ((sin(2*x*pi)).^2) + MYrandom(mean,sd,size(x,1),size(x,2));
end

% Q2.2 random gaussian number generator
```

```
function [N] = MYrandom(mean,sd,rowsize,colsize)

%%function takes as inputs: MYrandom(mean,sd,rowsize,colsize)
% mean and standard deviation(sd) of the gaussian distribution to be
% sampled from at random and returns matrix of specified row and
 column
% size

%   method:
    % Z=(X-mean)/standard deviation
    % thus: X=(Z*sd)+mean and Z is given by the randn function
    % randn samples from gaussian distribution
temp=randn(rowsize,colsize);
N=(sd*temp)+mean;
end

% Q2.3 MySIN2

function [sol] = MySIN(x,k)
% function fits curve basis given by: sin(1*pi*x), . . . , sin(k*pi*x)
% input is row vector of x values and degree of basis (k)
% output is a matrix of all basis orders up to and including k, were
% columns are the consecutive orders of k and rows are the x data
 points.

sol=zeros(size(x,1),k);

for i=1:k
    temp=sin(i*pi*x);
    sol(:,i) = temp;
```

Published with MATLAB® R2016b