%This will be the center of execution for the matlab scripts that we write.

%

%Authors: Zachary Schafer, James DiTucci, Maxim Zaman, Stephanie Lee

%Class: MA373-01 winter quarter 2015-2016

%Professor: Dr. Graves

function [results] = main(x,y, configuration,v)

% x and y are two vectors of the same length

% configuration is a string of parameters deliminated by commans.

% for example to plot degree 2,4 poly and tls call the following:

% main(x,y,'poly\_2,poly\_4,TLS')

% if only one fit is specified, the function returns an array of

% coefficents, if more than one fit is specified it returns an empty set

% v is an optional parameter of values to be evaluated and returned

% by the some method.

% Examples:

% to plot linear, quadratic, and spline of data x and y do:

% main(x,y,'linear,poly\_2,spline')

% to plot exponential enter configuration string 'exp'

% main(x,y,'exp')

if(nargin <2)

error('We require at least and x and y')

end

if(length(x) ~= length(y))

error('x and y must have the same length')

end

if(nargin == 4)

results = [];

v = reshape(v,[],1);

end

%Set up default display values

displayPolyCof = [1,2,10];

colors = {'m','b','r','c'};

%make data expected form

x = reshape(x,[],1);

y = reshape(y,[],1);

xy = sortrows([x y],1);

x = xy(:,1);

y = xy(:,2);

%create an empty array

results=[];

%must have over 3 arguments and the configuration string must not

%be empty

if(nargin >= 3 && ~strcmp(configuration,''))

%Possible configuration options

configcell = strsplit(configuration,',');

%colors = {'m','b'}

strings = {};

for i=1:length(configcell)

configs = char(configcell(i));

%These values will be 1 if the parameter is matched, and 0

%if it isn't.

doesMatchPoly = isequal(findstr(configs,'poly') , 1);

doesMatchLinear = isequal(findstr(configs,'linear') , 1);

doesMatchTLS = isequal(findstr(configs,'tls') , 1);

doesMatchSpline = isequal(findstr(configs,'spline') , 1);

doesMatchExp = isequal(findstr(configs,'exp') , 1);

%If concatenated is 0, that means that we couldn't match

%the string.

concatenated = (doesMatchPoly || doesMatchLinear || doesMatchTLS || doesMatchSpline||doesMatchExp);

%if word parsed is 0 then we did not find a match

wordParsed = isequal(1,concatenated);

%if word parsed is 0 then we did not find a match

if(~wordParsed)

error(['Could not parse configuration string. Arguments ' ...

'are case sensitive!'])

end

if( doesMatchPoly || doesMatchLinear)

if(doesMatchPoly)

n = str2num(configs(5:end));

elseif(doesMatchLinear)

n = 1;

else

error('How did I get here?')

end

[polynf,polycof] = polyreg(x,y,n);

polyyn = arrayfun(polynf,x);

[polyynr, polyynrmse] = functionerror(y,polyyn);

polystring = sprintf('Polynomial degree : %d ',n);

strings{end+1} = polystring;

printerror(polyynr,polyynrmse,polystring)

printcoefficients(polycof);

if(length(configcell)==1 && nargin==3)

results = polycof;

end

if(nargin == 4)

results = [results arrayfun(polynf,v)];

end

plot(x,polyyn,colors{mod(i,length(colors)-1)+1});

hold on

elseif(doesMatchTLS)

[TLSf, TLScof] = TLS(x,y);

TLSy = arrayfun(TLSf,x);

[TLSr2, TLSrmse] = functionerror(y,TLSy);

tlsstring = sprintf('TLS');

strings{end+1} = tlsstring;

printerror(TLSr2,TLSrmse,tlsstring)

printcoefficients(TLScof)

if(length(configcell)==1 && nargin==3)

results = TLScof;

end

if(nargin == 4)

results = [results arrayfun(TLSf,v)];

end

plot(x,TLSy,colors{mod(i,length(colors)-1)+1});

hold on

elseif(doesMatchSpline)

cubicy = cubicSpline(x,y,x);

splinestring = sprintf('Spline');

strings{end+1} = splinestring;

if(nargin == 4)

cubicv = cubicSpline(x,y,v);

results = [results cubicv];

end

plot(x,cubicy,colors{mod(i,length(colors)-1)+1});

hold on

elseif(doesMatchExp)

[expFunc, coeff] = expfit(x,y);

expy = arrayfun(expFunc,x);

expstring = sprintf('Exponential ');

[expr, exprmse] = functionerror(y,expy);

printerror(expr,exprmse,expstring)

fprintf('y = c \* exp(k\*x) -> [c k] \n')

fprintf(mat2str(coeff))

strings{end+1} = expstring;

if(length(configcell)==1&& nargin==3)

results = coeff;

end

if(nargin == 4)

results = [results arrayfun(expFunc,v)];

end

plot(x,expy,colors{mod(i,length(colors)-1)+1});

hold on

else

error('Match not found. Unexpected error')

end

end

strings{end+1} = 'data';

plot(x,y,'rx')

hold on

legend(char(strings),'location','northeastoutside')

%Fit TLS

elseif(nargin == 2 || strcmp(configuration,''))

plot(x,y,'ro')

hold on

legendStrings = {'data'};

itr = 1;

for i=1:length(displayPolyCof)

[polyf,polycof] = polyreg(x,y,displayPolyCof(i));

polyy = arrayfun(polyf,x);

[polyyr2, polyyrmse] = functionerror(y,polyy);

polystring = sprintf(' Polynomial : %d ',displayPolyCof(i));

legendStrings{end+1} = polystring;

%print error

printerror(polyyr2,polyyrmse,polystring)

% run function to print coefficients

printcoefficients(polycof)

plot(x,polyy,colors{mod(i,length(colors)-1)+1})

hold on

itr = i+1;

end

[TLSf, TLScof] = TLS(x,y);

TLSy = arrayfun(TLSf,x);

[TLSr2, TLSrmse] = functionerror(y,TLSy);

plot(x,TLSy,colors{mod(itr,length(colors)-1)+1})

legendStrings{end+1} = 'TLS';

tlsstring = legendStrings{end};

printerror(TLSr2,TLSrmse,tlsstring)

printcoefficients(TLScof)

itr = itr +1;

legendStrings{end+1} = 'Spline';

plotCubicSpline(x,y,1,colors{mod(itr,length(colors)-1)+1});

legend(char(legendStrings));

end

end

function [p,coeff] = polyreg(x,y,n)

% this method should return a polynomial function p which is the best least squares fit to the data [x,y]

%using the basis functions of 1,x,...,x^n

% Reshape our data so that our function works whether we get column or row vectors

x = reshape(x,[],1);

y = reshape(y,[],1);

inner = @(a,b) dot(a,b);

% loop over rows to i

for i=1:n+1

for j=i:n+1

Ax(i,j) = inner(x.^(i-1),x.^(j-1));

Ax(j,i) = Ax(i,j);

end

end

for i=1:n+1

Ay(i,1) = inner(x.^(i-1),y);

end

coeff = Ax\Ay;

%now that we have all the coefficients we must reorder them.

coeff = coeff(n+1:-1:1);

p = @(x) polyval(coeff,x);

%plot(x,y,'ro',x,arrayfun(p,x),'b')

function [TLSf, TLScof ] = TLS( A,B )

%Inputs:

%A = Vector of Independant Variables

%B = Vector of Dependant Variables

% Outputs:

%TLScof = Row Vector [a b] Where a+b\*x is the TLS Apprx

%TLSf = A function handle to the TLS approximation.

%Matricies need to be column vectors

%A=A';

%B=B';

A = reshape(A,[],1);

B = reshape(B,[],1);

% xx will be used later to plot result

xx=min(A)-2:max(A)+2;

M=[ones(size(A)) A B];

[m n]=size(A);

m=max(size(M));

sqm=sqrt(m);

u=ones(size(A));

u(1)=u(1)+sqm;

u=u/sqrt(sqm\*(1+sqm));

QM=M-u\*u'\*M;

MM=QM(2:m,2);

[U,S,Vt]=SVD\_Test([MM,QM(2:m,3)]);

s=Vt(n+1,n+1);

if s==0;

error('TLS Solution DNE')

end

b=-Vt(1:n,n+1)/s;

a=-(QM(1,2)\*b-QM(1,3))/QM(1,1);

% $$$ hold on

% $$$ plot(A,B,'x')

% $$$ plot(xx,a\*xx+b)

% $$$ hold off

TLScof=[b,a];

TLSf = @(x) a + b\*x;

end

function [ U,S,V] = SVD\_Test( A )

%Preform SVD on Matrix AV

%Find U - A\*At -> Eigenvalues and eigen Vectors -> Place in a matrix and

%orthonormalize using Grahm-Schmitt

syms lam; % Will be used for finding eigenvectors

format short;

AAt=A\*transpose(A);

[Evec\_U,Eval\_U]=eig(AAt); %Get eigenvectors and values

n=length(Evec\_U);

U=Evec\_U(:,n:-1:1); %Reorder column do be in decending order, based on eigenvalue

Eval\_U=Eval\_U([n:-1:1],[n:-1:1]); %Keep consistant with above swap

%Find Vt - At\*A -> Eigenvalues and eigen Vectors -> Place in a matrix and

%orthonormalize using Grahm-Schmitt

%Take the transpose

AtA=transpose(A)\*A;

[Evec\_V,Eval\_V]=eig(AtA); %Get eigenvectors and values

n=length(Evec\_V);

V=Evec\_V(:,n:-1:1); %Reorder column do be in decending order, based on eigenvalue

Eval\_V=Eval\_V([n:-1:1],[n:-1:1]); %Keep consistant with above swap

%Vt=transpose(V)

Eval\_V = nonzeros(diag(Eval\_V));

m=length(Eval\_V);

S=zeros(m,n);

N=0;

for N= [1:m];

S(N,N)=sqrt(Eval\_V(N,1));

end

end

function [expf, coeff] = expfit(x,y)

%Find best fit c\*exp(k\*x) for x,y plot.

%Input: Two vectors of the same length x and y

%Output: expf is a function handle to our approximation

%Output: coeff is the coefficients [c,k] for this fit.

%take ln(c\*exp(k\*x)) = ln(c) + k

if(nargin<2)

error('We require two inputs')

end

x = reshape(x,[],1);

y = reshape(y,[],1);

a11 = length(x);

a12 = sum(x);

a21 = a12;

a22 = sum(x.\*x);

lys = log(y);

y1 = sum(lys);

yx = sum(x .\* lys);

M = [a11 a12 ; a21 a22];

rhs = [y1 ;yx];

coeff = M \ rhs;

coeff(1) = exp(coeff(1));

c = coeff(1);

k = coeff(2);

expf = @(x) c\*exp(k\*x);

end

function [z] = cubicSpline(x,y,v)

%return a vector z which is the resulting function evaluated at all

%points v

%If no argument v is provided, the cublic spline is to be plot

%against x and y.

%We will be employing the smart solution which involves solving for

%s, the list of second derivatives, then use that to find our

%solutions.

if(nargin<2)

error('We expect to at least get an x and y input')

end

x = reshape(x,[],1);

y = reshape(y,[],1);

if(nargin==3)

v = reshape(v,[],1);

end

z = [];

n = length(x);

dx = x(2:n) - x(1:n-1);

dy\_x = (y(2:n) - y(1:n-1))./dx;

left = dx(1:end-1);

center = 2\*(dx(1:end-1) + dx(2:end));

right = dx(2:end);

%construct the matrix A which is the trigonal system on the left to solve

A = spdiags([left center right], [-1 0 1], n-2, n-2);

%as per the system.

rhs = 6\*(dy\_x(2:end)-dy\_x(1:end-1));

%solve for these second derivates, s

s = A\rhs;

% Use natural boundary conditions where second derivative

% is zero at the endpoints

s = [ 0; s; 0];

%plug in equations to find s

s0 = y;

s1 = dy\_x - dx.\*(2\*s(1:end-1) + s(2:end))/6;

s2 = s/2;

s3 = (s(2:end)-s(1:end-1))./(6\*dx);

num\_points\_on\_interval = 10;

%if we only get x and y

if(nargin<3)

plot(x,y,'ro')

hold on

for i=1:(n-1)

xx = linspace(x(i),x(i+1),num\_points\_on\_interval);

xi = repmat(x(i),1,num\_points\_on\_interval);

yy = s0(i) + s1(i)\*(xx-xi) + ...

s2(i)\*(xx-xi).^2 + s3(i)\*(xx - xi).^3;

plot(xx,yy,'b')

end

title('Cubic spline plot')

hold off

% if we get x,y,v

elseif(nargin==3)

xincr = -1;

for itr=1:(n-1)

% $$$ fprintf('v indexes for itr: %d \n', itr)

%this assumes x incresaing

if(x(itr+1) > x(itr))

xincr = 1;

xxvIndexes = find(v >= x(itr) & v < x(itr+1));

%assume x decreasing

elseif(x(itr+1) < x(itr))

xincr = 0;

xxvIndexes = find(v < x(itr) & v >= x(itr+1));

end

% $$$ fprintf('v values for itr: %d \n', itr)

xxv = v(xxvIndexes);

% $$$ fprintf('xi values for itr: %d \n', itr)

xvi = repmat(x(itr),1,length(xxv))';

% $$$ fprintf('length xxv: %d length xvi %d \n', length(xxv), length(xvi))

% $$$ fprintf('y values for i: %d \n', itr)

yv = s0(itr) + s1(itr)\*(xxv-xvi) + ...

s2(itr)\*(xxv-xvi).^2 + s3(itr)\*(xxv - xvi).^3;

z(xxvIndexes) = yv;

end

%covers edge case where one of our v might be the last value of

if(xincr == 1)

itr = itr + 1;

xxvIndexes = find(v == x(itr));

z(xxvIndexes) = y(itr);

elseif(xincr == 0)

itr = 1;

xxvIndexes = find(v == x(itr));

z(xxvIndexes) = y(itr);

end

z = reshape(z,[],1);

end

end

function [] = plotCubicSpline(x,y,jump,color)

subx = x(1:jump:length(x));

suby = y(1:jump:length(y));

intx = linspace(min(subx),max(subx),(max(subx)-min(subx)));

cubicy3 = cubicSpline(subx,suby,intx);

plot(intx,cubicy3,color)

% $$$ legendStr = sprintf('jump distance: %d',jump)

% $$$ legappend(legendStr)

function [r2, rmse] = functionerror(y,f)

%Here y is the actual data and f is the model fit.

y = reshape(y,[],1);

f = reshape(f,[],1);

r2 = corr(y,f)^2;

rmse = sqrt(mean((y-f).^2));

end