CIEG 675 - Matlab For Engineering Analysis (Lab 4)

Abdul Fayeed Abdul Kadir

PROBLEM 1

```
% * Create a movie of timelapse of Blizzard in Newark, DE Jan 2016\

blizzard = dir('/Users/fayeed/Desktop/WINTER 2021/CIEG
675/LAB/4/BlizzardImages/*.jpg');
% Loading the pictures on the said directory; use *.jpg to only load jpg files

for i = 1:length(blizzard)
    cd '/Users/fayeed/Desktop/WINTER 2021/CIEG 675/LAB/4/BlizzardImages'
    % Changing directory because imfinfo and reading the data on each
    % picture requires to be on their own directory
    info(i) = imfinfo(blizzard(i).name);
    % To obtain the info on the picture; using blizzard(i) indexing, cuz
    % each layer of blizzard i has only one filename, not the other way
    % round
    dates{i} = info(i).DateTime;
    % To acquire the actual time of the said picture, and they're strings,
    % so put in cell array
    [removed,keep] = strtok(dates{i});
    [hours{i},spaces] = strtok(keep); % To obtain the clock time
    matlabdates(i) = datenum(dates{i},'yyyy:mm:dd HH:MM:SS');
    % Changing the dates on the info, from string to MATLAB dates, and
    % specify the date format as shown; example of date format is from help
    % datetick
end

matlabdates = matlabdates - matlabdates(1);
% To obtain the times in reference to the first date
time_lapse = round(matlabdates.*24.*60,2); % Convert to minutes, and rounding off to
2 d.p

moviename = 'AbdulKadir_AbdulFayeed.avi';
vid_obj = VideoWriter(moviename); % To write the video
vid_obj.FrameRate = 8; % 8 frames/sec
open(vid_obj);
```

```
figure('name','Blizzard Animation')
set(gcf,'position',[693,203,725,514])

for i = 1:length(blizzard)
    img = imread(blizzard(i).name); % Read the image to uint8 data
    image(img); % Display each image at a time; can use either image or imshow
    % image will show the axis, imshow will not; but since the children
    % size will be changed soon, so the axis won't appear in the entire
    % movie
    im_children = get(gcf,'children');
    % Get the information on the 'children' of the image

    set(im_children,'position',[0 0 1 1]);
    % Changing the size of the picture within the entire figure, so that it
    % can be captured entirely, w/o background in the movies

    text(im_children.XLim(1) + 10,im_children.YLim(1),[num2str(time_lapse(i)),' minutes
elapsed'],...
        'color','y','fontweight','bold','fontsize',20,'horizontalalignment','left',...
        'verticalalignment','top')
    % Putting the timelapse text on top left
    text(im_children.XLim(2)/2,im_children.YLim(2),'Newark, DE Jan. 22,
2016','color','y',...
        'horizontalalignment','center','verticalalignment','bottom',...
        'fontweight','bold','fontsize',20)
    % Date text on middle bottom
    text(im_children.XLim(2) - 10,im_children.YLim(1),hours{i},'color','y',...
        'horizontalalignment','right','verticalalignment','top',...
        'fontweight','bold','fontsize',20)
    % Time text on top right

    drawnow % To capture every image at a time in the video
    currFrame = getframe(gcf); % As if screenshotting the pictures
    writeVideo(vid_obj,currFrame); % Record it essentially
end
cd '/Users/fayeed/Desktop/WINTER 2021/CIEG 675/LAB/4'
close(vid_obj);
```

PROBLEM 2

```
function [zg] = gridinterp(x,y,z,xg,yg,d,alpha)
% function [zg] = gridinterp(x,y,z,xg,yg,d,alpha)
%
% This function does an inverse distance weighting for 2D interpolation, by
% interpolating the scattered data points to a uniform grid.
%
% Inputs:
% x - scattered horizontal locations x; a long vector
% y - scattered vertical locations y; a long vector with the same size as x
% z - measured parameter (e.g. elevation) for the scattered data points;
% a long vector with the same size as x
% xg - uniform grid of horizontal locations x; a long vector
% yg - uniform grid of vertical locations y; a long vector of the same size
% as xg
% d - allowable radial distance from each uniform grid point; Euclidian
% distance
% alpha - order to inversely weight each point at x and y based on its
% distance from xg and yg respectively
%
% Output:
% zg - interpolated data of z on the uniform grid; a long vector of the
% same size as xg

for j = 1:length(xg)
    w = sqrt((xg(j) - x).^2 + (yg(j) - y).^2);
    % Weights; Euclidean distance
    % Comparing each grid point to all scattered points at once
    idx = find(w > d);
    % Finding the points that lie outside d
    w(idx) = NaN; % make it NaN, because we don't want it
    numerator = sum((1./(w.^alpha)).*z,'omitnan'); % omitnan in the calcualtion
    denominator = sum(1./(w.^alpha),'omitnan');
    zg(j) = numerator./denominator;
end
```

```
% * Perform inverse distance weighting; Euclidean Distance
% * Vary dx, d, alpha
load Avalon_survey.mat % Acquire the x,y,z scattered points
% Vary dx values
dx = [1,30];
figure('name','Euclidean Distance: Varying dx')

for i = 1:length(dx)
    s(i) = subplot(2,1,i);
    xx = -10:dx(i):100;
    yy = -180:dx(i):110;
    [X,Y] = meshgrid(xx,yy);
    % X has each row a copy of xx; Y has each column a copy of yy
    % In total, the matrix will be length(y) rows by length(x) columns
    xg=X(:); % Make it long vector, it works by compiling all columns into one long
column
    yg=Y(:);
    d = 10; % radial distance (m)
    alpha = 2; % 1 <= alpha <= 2

    zg = gridinterp(x,y,z,xg,yg,d,alpha);
    ZG = reshape(zg,size(X));
    % Change the shape of the matrix to match with the original X and Y

    surf(X,Y,ZG) % 3D curves
    shading interp
    b(i) = colorbar;

    xtickformat('%.f'); ytickformat('%.f'); ztickformat('%.1f');
    xlabel('Cross-shore Distance (m)','fontweight','bold')
    ylabel('Along-shore Distance (m)','fontweight','bold')
    ylabel(b(i),'Elevation (m)','fontweight','bold','fontname','times');
    % Labelling the colorbar
    title(['\bf\Deltax = ' num2str(dx(i))])
    set(s(i),'fontname','times'); view(s(i),52,20) % Changing the view as I want
    box on % Make the dark box outlines
end
```

```
set(gcf,'position',[788 143 645 603])

% Attempt to make only one colorbar
% s is subplot, b is colorbar
s1_position = get(s(1),'position'); s2_position = get(s(2),'position');
set(b(1),'visible','off'); b2_position = get(b(2),'position');
set(b(2),'position',[b2_position(1:3),0.81]);
set(s(2),'position',[s2_position(1:2),s1_position(3:4)]);
set(s(1),'position',s1_position);

% Changing the decimal places on the colorbar - long method
for i = 1:length(b(2).TickLabels)
    ticklabels = str2num(b(2).TickLabels{i});
    b(2).TickLabels{i} = sprintf('%.1f',ticklabels);
    % sprintf sets how I want the output be written as, and take second
    % input as the thing I want it to be rewritten
end

print -djpeg Problem2_dxVary
```

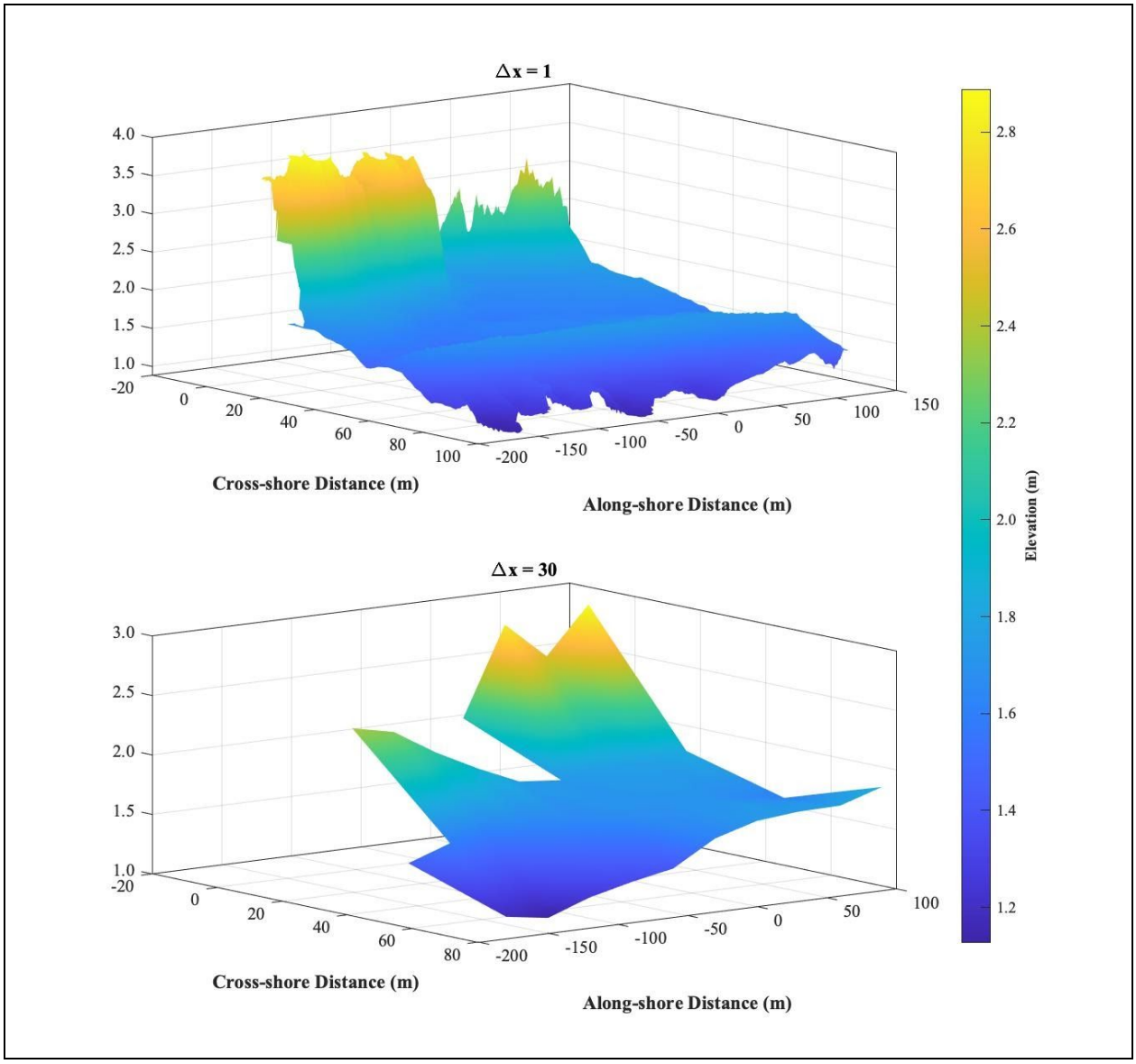

Figure 1: Elevation Plot from Inverse Distance Weighting by Varying Δx

Δx determines the number of grid points (x_g and y_g) to be considered for the interpolation of surface from the scattered data points. The smaller the value of Δx, the higher the number of grid points, which will lead to more Euclidean distance calculation for each grid point with respect to all scattered points. In other words, more scattered points will be considered in the calculation distance, making the curve smoother and having a more accurate representation of the elevations. From **Figure 1**, it could be seen that some of the elevation surfaces are not plotted for Δx = 30, as compared to Δx = 1.

```
%% Vary d
d = [5,100];
figure('name','Euclidean Distance: Varying d')
for i = 1:length(dx)
    s(i) = subplot(2,1,i);
    xx = -10:1:100; yy = -180:1:110;
    [X,Y] = meshgrid(xx,yy);
    xg=X(:); yg=Y(:); alpha = 2;
    zg = gridinterp(x,y,z,xg,yg,d(i),alpha);
    ZG = reshape(zg,size(X));

    surf(X,Y,ZG)
    shading interp
    b(i) = colorbar;

    xtickformat('%.f'); ytickformat('%.f'); ztickformat('%.1f');
    xlabel('Cross-shore Distance (m)','fontweight','bold')
    ylabel('Along-shore Distance (m)','fontweight','bold')
    ylabel(b(i),'Elevation (m)','fontweight','bold','fontname','times');

    title(['\bfd = ' num2str(d(i)) ' m'])
    set(s(i),'fontname','times'); view(s(i),52,20); box on
end
set(gcf,'position',[788 143 645 603])

s1_position = get(s(1),'position'); s2_position = get(s(2),'position');
set(b(1),'visible','off'); b2_position = get(b(2),'position');
set(b(2),'position',[b2_position(1:3),0.81]);
set(s(2),'position',[s2_position(1:2),s1_position(3:4)]);
set(s(1),'position',s1_position);

for i = 1:length(b(2).TickLabels)
    ticklabels = str2num(b(2).TickLabels{i});
    b(2).TickLabels{i} = sprintf('%.1f',ticklabels);
end

print -djpeg Problem2_dVary
```

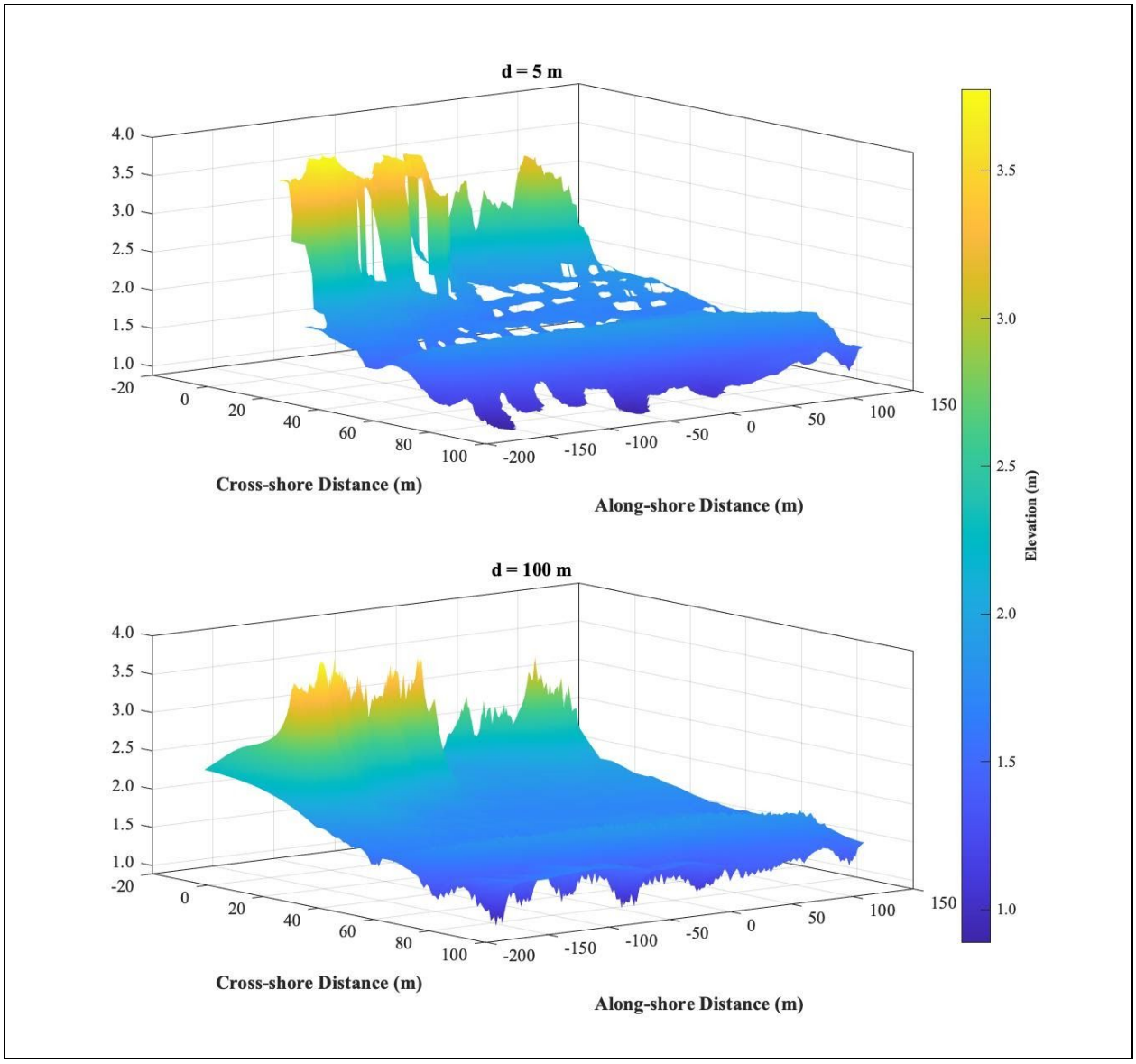

Figure 2: Elevation Plot from Inverse Distance Weighting by Varying d

Based on **Figure 2**, increasing the number of d will increase the number of scattered points to be included into the inverse distance weighting function. This will cause more elevation data points obtained from the function and plotted. It could be seen for d = 5 m that there are some 'holes' on the elevation surface, due to the fact that some scattered points were not included in the surface interpolation.

```
%% Vary alpha
alpha = [1,2];
figure('name','Euclidean Distance: Varying alpha')
for i = 1:length(dx)
    s(i) = subplot(2,1,i);
    xx = -10:1:100; yy = -180:1:110;
    [X,Y] = meshgrid(xx,yy);
    xg=X(:); yg=Y(:); d = 10;
    zg = gridinterp(x,y,z,xg,yg,d,alpha(i));
    ZG = reshape(zg,size(X));

    surf(X,Y,ZG)
    shading interp
    b(i) = colorbar;

    xtickformat('%.f'); ytickformat('%.f'); ztickformat('%.1f');
    xlabel('Cross-shore Distance (m)','fontweight','bold')
    ylabel('Along-shore Distance (m)','fontweight','bold')
    ylabel(b(i),'Elevation (m)','fontweight','bold','fontname','times');

    title(['\bf\alpha = ' num2str(alpha(i))])
    set(s(i),'fontname','times'); view(s(i),52,20); box on
end
set(gcf,'position',[788 143 645 603])

s1_position = get(s(1),'position'); s2_position = get(s(2),'position');
set(b(1),'visible','off'); b2_position = get(b(2),'position');
set(b(2),'position',[b2_position(1:3),0.81]);
set(s(2),'position',[s2_position(1:2),s1_position(3:4)]);
set(s(1),'position',s1_position);

for i = 1:length(b(2).TickLabels)
    ticklabels = str2num(b(2).TickLabels{i});
    b(2).TickLabels{i} = sprintf('%.1f',ticklabels);
end

print -djpeg Problem2_alphaVary
```

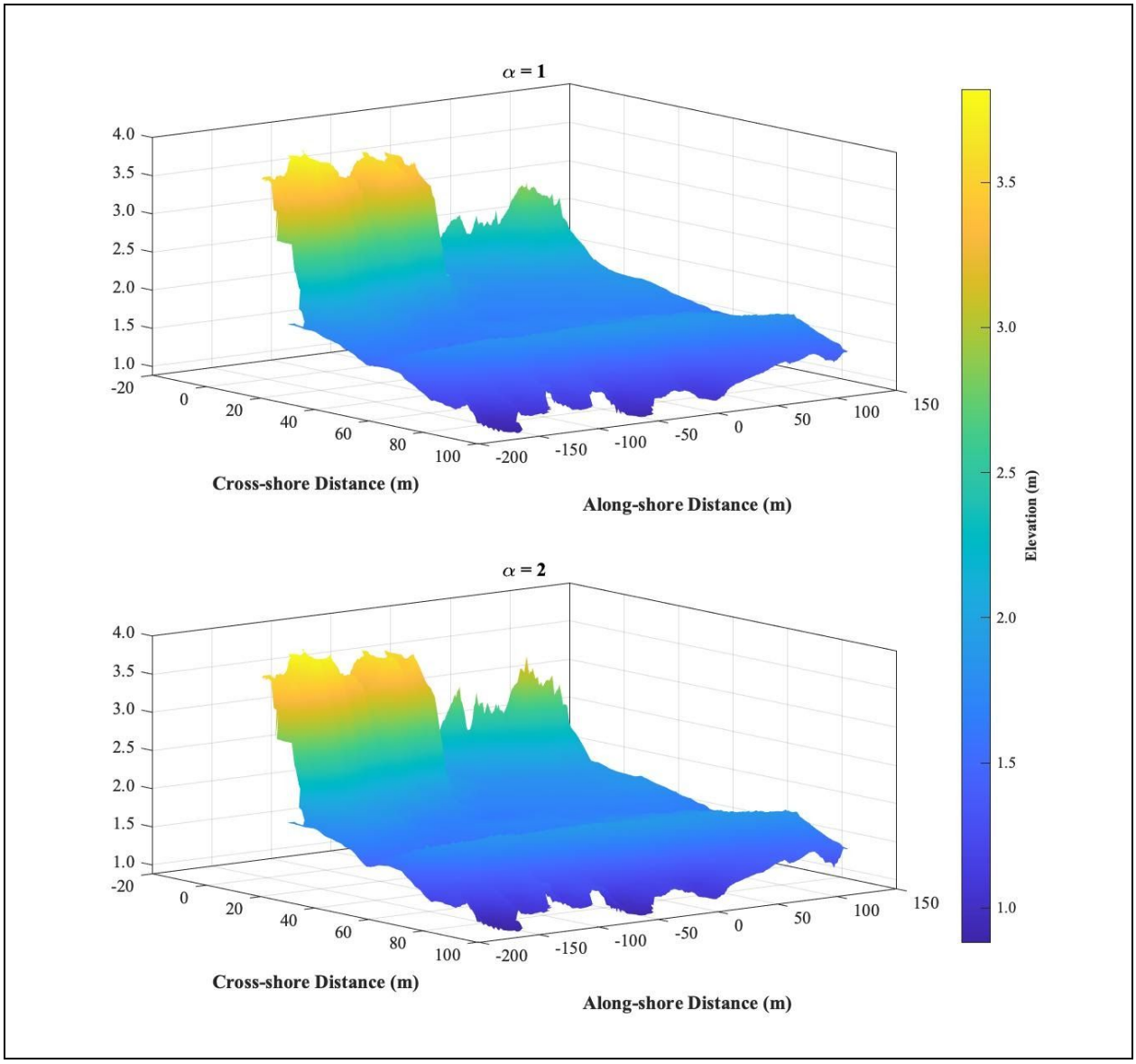

Figure 3: Elevation Plot from Inverse Distance Weighting by Varying α

Higher α has a greater effect on the calculation of Euclidean distance, which will solve for greater value in z. From **Figure 3**, it could be seen that the peaks at the along-shore distance of ~ 100 m and cross-shore distance of ~ 0 m of the higher α have higher elevation than that of smaller α.

PROBLEM 4

```
% * Solve dy/dx = -y(x), y(0) = 1, y = e^(-x) is the real soln.
% * Solve in 2 ways: a) forward difference, b) central difference
% * Each approach, use three different dx
% * Provide table of root-mean-square error (compared to real ans) as a
% function of dx and method
% * Plot the soln
% * OPTIONAL: Use ode23 &/ ode45

% * Forward Difference
% * dy/dx = -y_{t} => y_{t+1} - y_{t} = -dx*y_{t} => y_{t} - y_{t-1} = -dx*y_{t-1}
% * Discretized form: y_t = y_{t-1} - dx*y_{t-1}

dx = [0.01,0.5,1]; % 3 dx values
color = {'k','r','b'}; end_x = 5; % The upper limit of x-axis

figure('name','Numerical Approximation - Forward and Central Difference')
subplot(2,1,1)
for j = 1:length(dx) % Loop over all dx
    x = 0:dx(j):end_x; y_forw(1,j) = 1; % Initial value for each dx
    for i = 2:length(x)
        y_forw(i,j) = y_forw(i-1,j) - dx(j)*y_forw(i-1,j);
        % Apply the discretized form for each dx
        % Since each y_forw has different length, the empty cell will be
        % filled with 0, so have to plot by indexing in the next lines
    end
    plot(x,y_forw(1:1:length(x),j),'linewidth',1.5,'color',color{j})
    % Plot each dx curve; each x has different length due to different dx,
    % so y has to accomodate the length of its respective x, hence why
    % plotting by indexing y_forw(1:1:length(x),j)
    hold on
    y_real = exp(-x);
    rmse_forw(j) = sqrt(sum((y_real' - y_forw(1:1:length(x),j)).^2)./length(x));
    % To find the root mean square error (formula is as shown)
    % y_real has to be transpose ', to match the size of y_forw
end
```

```
x = 0:dx(2):end_x; y_real = exp(-x); % For the real solution markers plotting

plot(x,y_real,'ko','markersize',5,'markerfacecolor','g')
legend(['Approx. (\bf\Deltax = ' num2str(dx(1)) ')'],...
    ['Approx. (\bf\Deltax = ' num2str(dx(2)) ')'],...
    ['Approx. (\bf\Deltax = ' num2str(dx(3)) ')'],'Real Solution')
xlabel('x-value','fontweight','bold'); ylabel('y-value','fontweight','bold');
title('Forward Difference','fontweight','bold')
set(gca,'fontname','times','fontsize',12); xtickformat('%.1f'); ytickformat('%.1f')

table_array = [dx;rmse_forw];
array2table(table_array','variablenames',{'dx','RMSE'})
% Transpose so dx and rmse on columns, not rows
```

Table 1: Root-mean-square Errors for Respective Δx of Forward Difference Method

Δx	RMSE
0.01	1.12×10^{-3}
0.50	6.38×10^{-2}
1.00	1.62×10^{-1}

```
% * Backward Difference
% * dy/dx = -y_{t} => y_{t} - y_{t-1} = -dx*y_{t} => y_{t} + dx*y_{t} = y_{t-1}
% * Discretized form: y_t = y_{t-1} / (1 + dx)

% * Central difference
% * Average of Forward and Backward difference

subplot(2,1,2)
for j = 1:length(dx)
    x = 0:dx(j):end_x;
    y_back(1,j) = 1;
    for i = 2:length(x)
        y_back(i,j) = y_back(i-1,j)/(1 + dx(j));
        % Backward difference
    end
    y_central(:,j) = (y_forw(:,j) + y_back(:,j))./2;
    % Taking the average for central difference
    plot(x,y_central(1:1:length(x),j),'linewidth',1.5,'color',color{j})
    hold on
    y_real = exp(-x);
    rmse_central(j) = sqrt(sum((y_real' - y_central(1:1:length(x),j)).^2)./length(x));
end
x = 0:dx(2):end_x; y_real = exp(-x);

plot(x,y_real,'ko','markersize',5,'markerfacecolor','g')
legend(['Approx. (\bf\Deltax = ' num2str(dx(1)) ')'],...
    ['Approx. (\bf\Deltax = ' num2str(dx(2)) ')'],...
    ['Approx. (\bf\Deltax = ' num2str(dx(3)) ')'],'Real Solution')
xlabel('x-value','fontweight','bold'); ylabel('y-value','fontweight','bold');
title('Central Difference','fontweight','bold')
set(gca,'fontname','times','fontsize',12); xtickformat('%.1f'); ytickformat('%.1f')

set(gcf,'position',[365,150,647,628])
table_array = [dx;rmse_central];
array2table(table_array','variablenames',{'dx','RMSE'})

print -djpeg Problem4_plot
```

Table 2: Root-mean-square Errors for Respective Δx of Central Difference Method

Δx	RMSE
0.01	4.03×10^{-6}
0.50	1.04×10^{-2}
1.00	4.90×10^{-2}

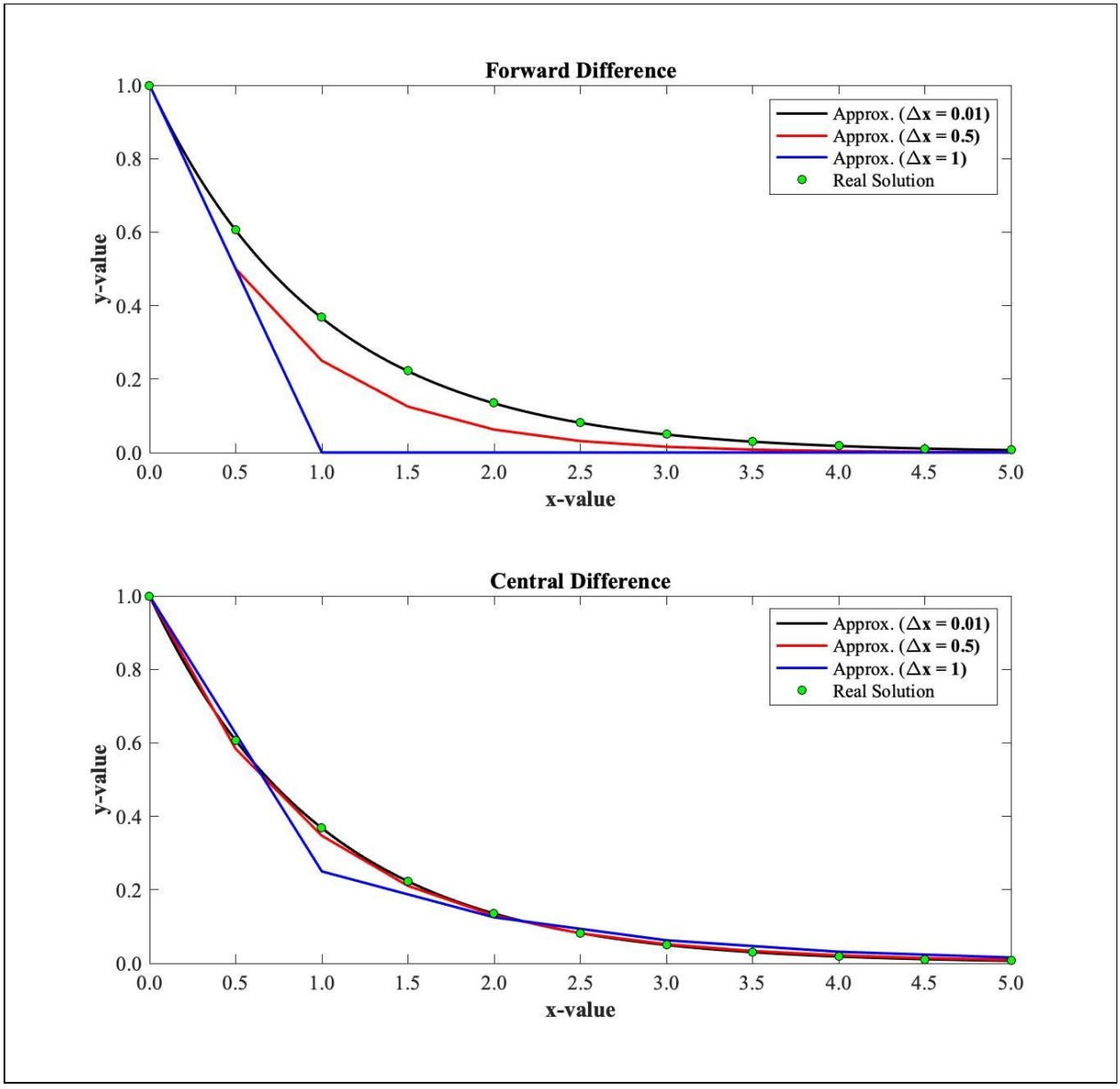

Figure 4: Differential Equation Solutions of Two Different Methods

PROBLEM 5

```
% * edge is like outlining any lines on the picture, but the image should
% be b&w to begin with

im = imread('speed_limit.jpg');
im2 = rgb2gray(im); % To change to gray scale
speed_limit = edge(im2,'canny');
% canny looks better - doesn't matter

im_new = ind2rgb(speed_limit,[1 1 0;0 0 1]);
% Change to color scale, with change of color
% [a;b] - a is color of edges, b is color of background
% This color in terms of single/double RGB Triplet
% [1 1 0] - yellow, [0 0 1] = blue

imwrite(im_new,'New_SpeedLimit.jpeg','jpeg')
% To create new image
```

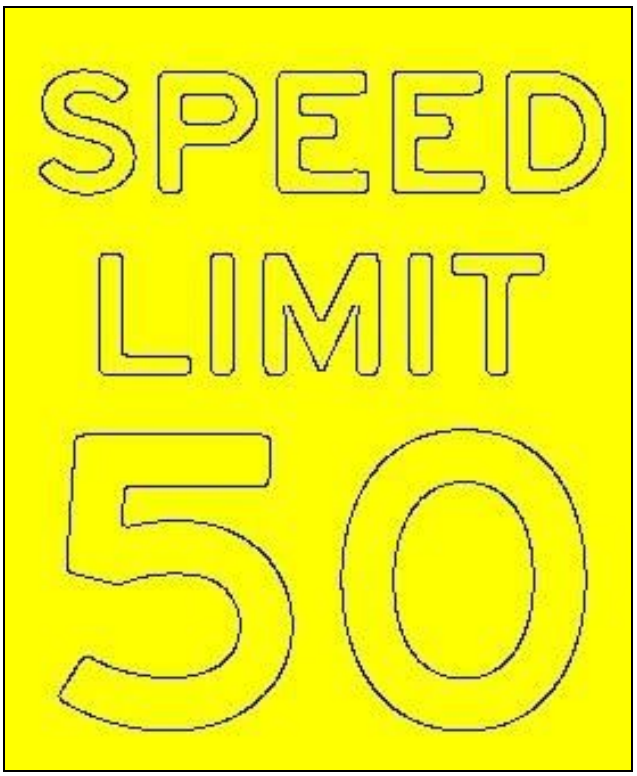

Figure 5: New Speed Limit Image