Lab #3 - CIEG 675/010

Chrysostomos Karakasis

Due date: February 1, 2021

```
%% CIEG 675 LAB #4 Due Tuesday February 1, 2021
%% Author: Chrysostomos Karakasis 702529334
close all;
clear all;
% Functions are located in the same folder as this file
```

1 Part (1)

The goal at the end of this problem is to create a .avi movie file that is a time lapse of the beginning of the Blizzard in Newark, DE of January 2016.

```
%% Part (1)
% The goal at the end of this problem is to create a .avimovie file that is
% a time lapse of the beginning of the Blizzard in Newark, DE of January 2016.

% Download the compressed folder from Canvas called "BlizzardImages.zip" and
% extract the images to your computer. You will need to write a script to
% find all of the .JPG files in the directory where you extracted the
% images. Recall the dir command discussed in class.
D = dir('./BlizzardImages/'); % We assume that a folder exists inside the folder that contains this code

% Then, with a for loop, write a code that will create a time lapse of the
% blizzard (using imshow) and write a movie to a .avifile. You should also
% add text somewhere on each image in the time lapse (use a large fontsize)
% that displays the total amount of time that has elapsed, in minutes(you may
% have to refresh yourself on dates in MATLAB and converting dates from
% string format to the number of days relative to 0 AD).In addition, you
% may want to try playing around with the figure window size/position, as
% well as the position of the image inside the figure window, if you want
% to get the movie to look more professional without white/grey borders
% around the image.

% Define the name of the .avi video
vidObj = VideoWriter('Karakasis_Chrysostomos.avi');
% Use a framerate of 8 frames per second (FPS).
FrameRate = 8;
% Open the .avi file
open(vidObj);

for i = 3:length(D) % Skip the two first lines ('.' and '..')
    folder = D(i).folder; % Location of the folder containing the image
    image = D(i).name; % Name of image
    path = [folder '\' image]; % Location of the image
    imshow([D(i).folder '\' image]); % Show the image
    set(gcf,'Position',[446   233   933   700]); % Change the size of the image to minimize any gray ...
        areas

    % maximize the image within a figure window
    im = get(gcf,'children');
    set(im,'units','normalized','position',[0 0 1 1]);

    % Use the 'day2secthat' function to find elapsed time in seconds and
    % divide by 60 to get elapsed time in minutes
    time_elap = day2secthat([D(3).datenum D(i).datenum])/60;

    % Print time elapsed on image
    dim = [0.04 .675 .3 .3]; % Location of the text box in the figure
    str = ['+',num2str(time_elap(2),'%6.2f') ' minutes elapsed']; % The text that will appear on the ...
        figure
    annotation('textbox',dim,'String',str,'FitBoxToText','on','FontSize',16,'color','y','EdgeColor','y');

    pause(0.01) % so we can see the change while making it
    currframe = getframe(gcf); % "grab" the current frame
    writeVideo(vidObj,currframe); % write the current frame
end
close(vidObj); % Close the .avi file
```

```
function [num_sec] = day2secthat(mat_dates_v)
% function [num_sec] = day2secthat(mat_dates_v)
```

```
3
4  % Description: This function receives as an input a vector of MATLAB dates
5  % and returns as an output a vector containing the number of seconds that
6  % have elapsed since the 1st value (date) in the input vector
7
8  mat2str = datestr(mat_dates_v); % Convert the input vector to string format
9  str2vec = datevec(mat2str); % Convert the string format to a vector of components
10 num_sec = []; % Initialize the output vector that will contain the elapsed number of seconds
11 for i = 1:size(str2vec,1) % Iterate through all dates (elements) of the input vector
12     % Calculate the elapsed number of seconds of each date wrt to the
13     % first one and add it to the output vector.
14     num_sec = [num_sec etime(str2vec(i,:),str2vec(1,:))];
15 end
16 end
```

Figure 1: Resulting movie file - time lapse of the beginning of the Blizzard. (must have the .mov file in the same folder as this .pdf in order for the video to play)

2 Part (2)

Write a function (call it *gridinterp*) to do an inverse distance weighting algorithm for 2D interpolation. It is a technique for interpolating to a uniform grid. Your function should have 7 input variables: x,y, z, xg, yg, d and alpha. Your function should have a single output vector, zg.

```
1  %% Part (2)
2  % Write a function (call it gridinterp) to do an inverse distance weighting
3  % algorithm for 2D interpolation. It is a technique for interpolating to a
4  % uniform grid.  Your function should have 7 input variables: x,y, z, xg,
5  % yg, d and alpha. Your function should have a single output vector, zg,
6
7  % Load the "Avalon_survey.mat" file that we will use for testing
8  avalon = load('Avalon_survey.mat');
9  % Save the scatted x data
10 x = avalon.x;
11 % Save the scatted y data
12 y = avalon.y;
13 % Save the scatted z data
14 z = avalon.z;
15
16 % Load default values for the three parameters
17 dx = 4; % Define increment for the desired uniform grid
18 d = 10; % Define the search radius
```

```
alpha = 1; % 1 ≤ a ≤ 2 order to inversely weight each point at x,y

xx=-10:dx:100; % x locations of the desired uniform grid
yy=-180:dx:110; % y locations of the desired uniform grid
[X,Y]=meshgrid(xx,yy); % vectorize the desired uniform grid
xg=X(:); % horizontal grid location in x-axis
yg=Y(:); % horizontal grid location in y-axis
% Basically xx is repeated 'length(yy)' times and yy is repeated
% 'length(xx) times, to create a grid

zg=gridinterp(x,y,z,xg,yg,d,alpha); % Call the function we created to apply the interpolation
ZG=reshape(zg,size(X)); % Reshape zg from a vector to 2D-matrix with appropriate size to use surf plot

fig2 = figure(2); % Create a new figure
clf; % Clear anything that might exist already in the figure
surf(X,Y,ZG); % 3D-plot the interpolated data wrt to the desired uniform grid
colormap(jet); % Change to a RGB spectrum
colorbar % Add a colorbar to have an indication of the values at the surface's points
xlabel('cross-shore distance (m)','fontsize',14,'interpreter','latex'); % Set an appropriate x-axis
ylabel('along-shore distance (m)','fontsize',14,'interpreter','latex'); % Set an appropriate y-axis
zlabel('Elevation (m)','fontsize',14,'interpreter','latex'); % Set an appropriate z-axis
% Set a title to show which values were utilized for (dx,d,alpha)
title({'LAB \# 4 - Part (2) - gridinterp',['dx = ',num2str(dx),', d = ',num2str(d),', $\alpha$ = ...
    ',num2str(alpha)]},'fontsize',14,'interpreter','latex');
set(gcf, 'Position', get(0, 'Screensize'));  % Switch to full-screen to have a better visualization ...
    of the figure
az = 47;
el = 39;
view(az,el); % Change the camera view angle to match the one in the given example
print(fig2,'lab_4_part_2_default','-depsc','-r600'); % Save figure in a colored .eps format using 600 ...
    dpi resolution

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Explore effect of parameter dx, d, alpha
% Defaults: dx=4, d=10, alpha=1
dx_v = [4 10 20];
d_v = [1 10 100];
alpha_v = [1 1.5 2];
for i = 1:3 % Repeat for dx, d and alpha
    for j = 1:3 % Try three different values for each parameter
        if i == 1 % Change only dx paremeter and load default for d and alpha
            param = 'dx'; % Define which parameter is currently explored
            dx = dx_v(j); % Define increment for the desired uniform grid
            d = 10; % (Default value) Define the search radius
            alpha = 1; % (Default value) 1 ≤ a ≤ 2 order to inversely weight each point at x,y
        elseif i == 2 % Change only d paremeter and load default for dx and alpha
            param = 'd'; % Define which parameter is currently explored
            dx = 4; % (Default value) Define increment for the desired uniform grid
            d = d_v(j); % Define the search radius
            alpha = 1; % (Default value) 1 ≤ a ≤ 2 order to inversely weight each point at x,y
        else % Change only alpha paremeter and load default for dx and d
            param = 'alpha'; % Define which parameter is currently explored
            dx = 4; % (Default value) Define increment for the desired uniform grid
            d = 10; % (Default value) Define the search radius
            alpha = alpha_v(j); % 1 ≤ a ≤ 2 order to inversely weight each point at x,y
        end

        xx=-10:dx:100; % x locations of the desired uniform grid
        yy=-180:dx:110; % y locations of the desired uniform grid
        [X,Y]=meshgrid(xx,yy); % vectorize the desired uniform grid
        xg=X(:); % horizontal grid location in x-axis
        yg=Y(:); % horizontal grid location in y-axis
        % Basically xx is repeated 'length(yy)' times and yy is repeated
        % 'length(xx) times, to create a grid

        zg=gridinterp(x,y,z,xg,yg,d,alpha); % Call the function we created to apply the interpolation
        ZG=reshape(zg,size(X)); % Reshape zg from a vector to 2D-matrix with appropriate size to use ...
            surf plot

        fig3(i) = figure(2+i); % Create a new figure (one for each variable)
        hold on;
        sgtitle({['LAB \# 4 - Part (2) - gridinterp','Explore the parameter ...
            ``',param,'"']},'fontsize',14,'interpreter','latex');
        s = subplot(3,1,j);
        surf(X,Y,ZG) % 3D-plot the interpolated data wrt to the desired uniform grid
        title(['dx = ',num2str(dx),', d = ',num2str(d),', $\alpha$ = ...
            ',num2str(alpha)],'fontsize',14,'interpreter','latex');
        az = 47;
        el = 39;
        % Change the camera view angle to match the one in the given example
```

```
        view(s,az,el);

        colormap(jet); % Change to a RGB spectrum
        colorbar; % Add a colorbar to have an indication of the values at the surface's points
        xlabel('cross-shore distance (m)','fontsize',14,'interpreter','latex'); % Set an appropriate ...
            x-axis
        ylabel('along-shore distance (m)','fontsize',14,'interpreter','latex'); % Set an appropriate ...
            y-axis
        zlabel('Elevation (m)','fontsize',14,'interpreter','latex'); % Set an appropriate z-axis
        set(gcf, 'Position', get(0, 'Screensize'));  % Switch to full-screen to have a better ...
            visualization of the figure
    end
    % For each variable save the corresponding figure with a relevant name
    print(fig3(i),['lab_4_part_2_' param 's'],'-depsc','-r600'); % Save figure in a colored .eps ...
        format using 600 dpi resolution
end
```

```
function zg = gridinterp(x, y, z, xg, yg, d, alpha)
% function zg = gridinterp(x, y, z, xg, yg, d, alpha)

% Description:
% This function performs an inverse distance weighting algorithm for 2D
% interpolation. It is a technique for interpolating to a uniform grid. It
% receives as inputs a measured parameter ('z') and the horizontal locations
% ('x') and ('y') at which it is scattered at. Furthermore, it receives the
% horizontal locations ('xg' and 'yd') of the desired uniform grid that the
% user wishes to utilize for the interpolation. Besides that, the function
% receives as an input a parameter ('d'), which specifies the allowable
% radial distance from each uniform grid point that the algorithm should
% search for measured data point over which to interpolate (search radius).
% Finally, the last input of the function is parameter a(alpha), which is
% the order to inversely weight each point at x, y based on its distance
% from xg, yg (i.e., for alpha = 2, the weighting is inverse distance
% squared). The output of the function is an interpolated version of 'z'
% onto the desired uniform grid given by horizontal grid locations xg and
% yg.

% INPUTS:
%x: horizontal location of measured parameter
%y: vertical location of measured parameter
%z: measured parameter (e.g. elevation)
%xg: horizontal location of desired uniform grid
%yg: vertical location of desired uniform grid
%d: allowable radial distance from each uniform grid point(search radius)
%alpha: order to inversely weight each point at x, y based on its distance from xg, yg(i.e., for ...
    alpha= 2, the weighting is inverse distance squared). Hint, xgand ygshould come from ...
    'vectorizing' the uniform grid you develop using the meshgrid function (see below).

% OUTPUTS:
% zg: interpolated version of input variable 'z' onto the desired uniform grid given by xg and yg

zg = []; % Initialization of the output vector
for j = 1 : length(xg) % Iterate for all desired points
    sum_num = 0; % Initialize the sum that corresponds to the numerator of the final function
    sum_den = 0; % Initialize the sum that corresponds to the denominator of the final function
    for i = 1 : length(x)  % Iterate through all initial points
        weight_num = norm([xg(j)-x(i) yg(j)-y(i)]); % Weights - Euclidean distance from the grid ...
            point to the real value
        if weight_num < d % Apply the search radius and determine whether a point should be taken ...
            into account or not
            sum_num = sum_num + z(i)/(weight_num^alpha); % Update the numerator sum in case a point ...
                should be included
            sum_den = sum_den + 1/(weight_num^alpha); % Update the denominator sum in case a point ...
                should be included
        end
    end
    zgj = sum_num/sum_den; % Apply the final formula and calculate another element of the ...
        interpolated output
    zg = [zg; zgj]; % Append the new element to the previous elements of the interpolated output
end

end
```

Discussion about the effect of parameters

As we can see in Fig.3, the parameter dx controls the resolution of the 2D-interpolation, meaning that more points are created on the desired uniform grid. Hence, greater values lead to less points and lower resolution, while smaller values lead to more points and higher resolution.

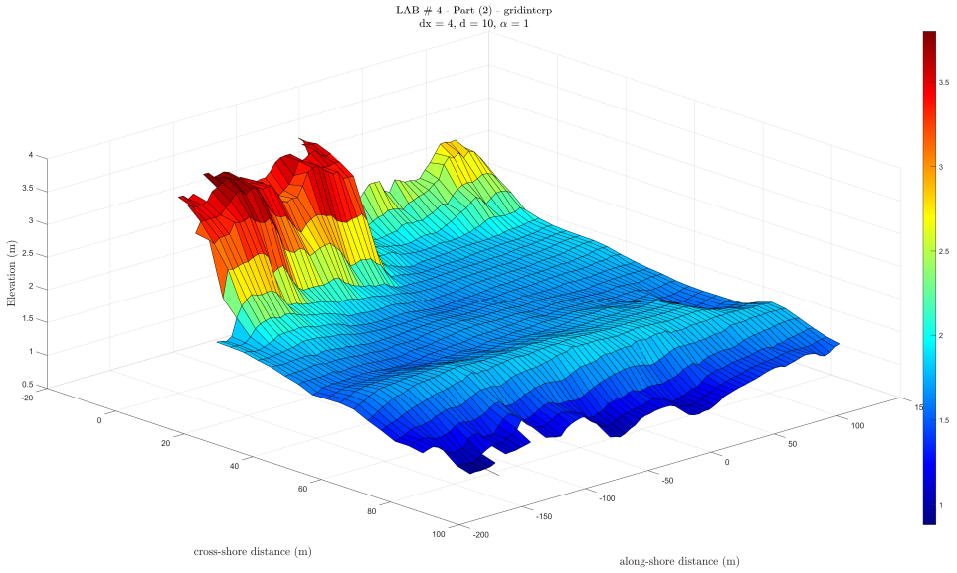

Figure 2: Lab # 4 - Part (2) - Plotting of 2D interpolation using default parameters for dx, d, α.

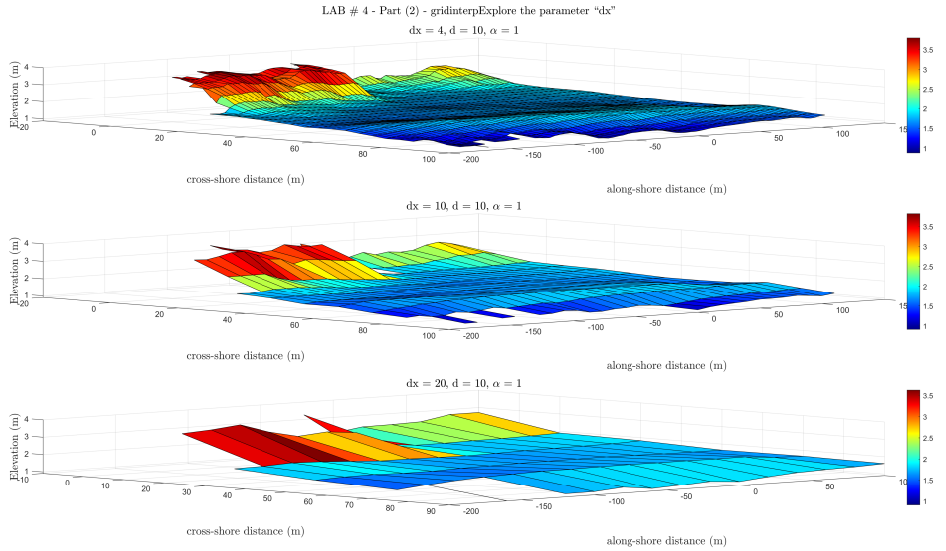

Figure 3: Lab # 4 - Part (2) - Explore the effect of parameter dx on the 2D interpolation.

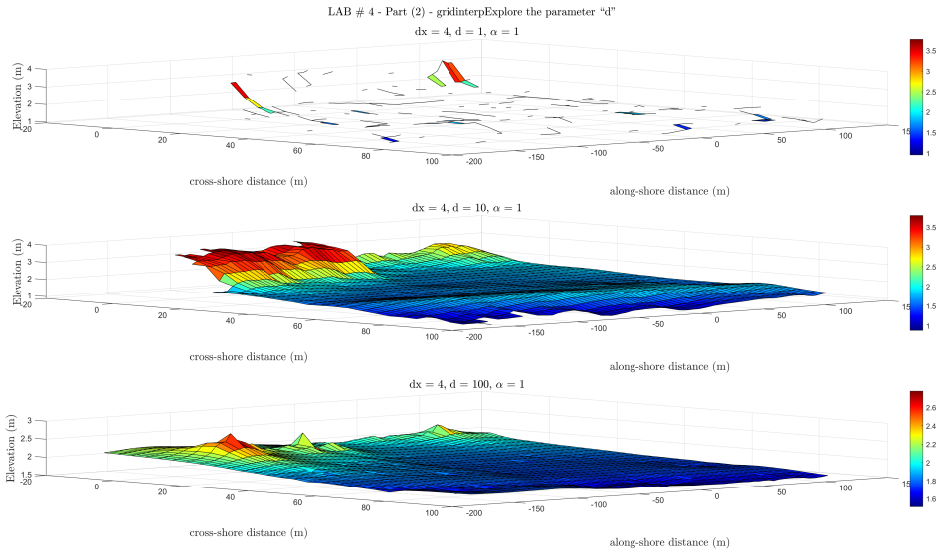

Figure 4: Lab # 4 - Part (2) - Explore the effect of parameter d on the 2D interpolation.

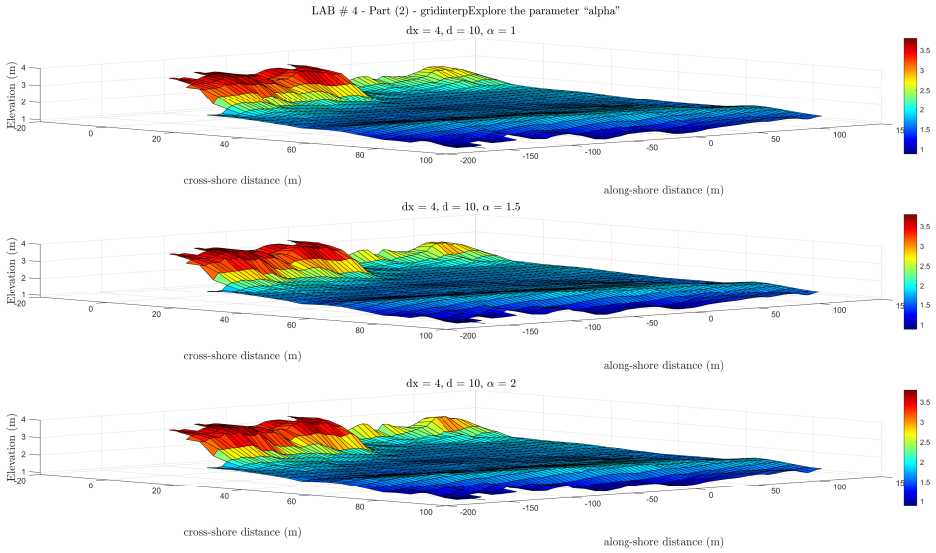

Figure 5: Lab # 4 - Part (2) - Explore the effect of parameter α on the 2D interpolation.

As we can see in Fig.4, the parameter d controls the values of each point at the uniform grid. Again, this parameter is associated with the search radius, meaning how many points should be taken into account for the calculation of the value at each grid point. Hence, really small values mean that each point will only be affected by points residing inside that small neighborhood. This might lead to missing points considering that for small values perhaps no points exist inside that area. For greater values, the neighborhood of affect is increased, meaning that more points should be taken into account and be averaged. Therefore, this would restrain any extreme values and the resulting surface will look more smooth in general.

As we can see in Fig.5, the parameter α controls the weights that affect each point. The range of values in this case is limited, but it can be seen that all three values result in similar figures, while for $\alpha = 2$ slightly increased values are observed. Now, if you consider that the weights are actually the euclidean distance between two points, then we can see that the weights of closer pairs of points (distance less than 1) will be reduced as α increases, and the weights of pairs of points that have a distance greater than 1 will be increased. Therefore, since the values are inversely proportional to the weights, as α increases, the value of a grid point is affected more from closer points than from points that are far away.

3 Part (3) - *OPTIONAL*

COVID-19 can be modeled with the SIR model. The model consists of three coupled differential equations:

$$\text{Susceptible } (S) : \frac{dS}{dt} = -bSI \tag{3.1}$$

$$\text{Infected } (I): \frac{dI}{dt} = bSI - kI \tag{3.2}$$

$$\text{Recovered/Removed } (R): \frac{dR}{dt} = kI, \tag{3.3}$$

where S, I, and R are all functions of time (t) in days. Additionally, S, I, and R are the normalized version of the susceptible, infected and recovered/removed population. That is, they are divided by the total population group, N such that $0 <= S <= 1$ (and the same applies for I and R). Note that $S + I + R = 1$, since a member of the population can either belong in the susceptible, the infected or the recovered/removed group, hence the sum of the member of each group has to represent the whole population.

The value k is known as the average period of infectiousness, $k = 1/k1$; where for example $k1 = 14$ days as the average time to recover.The value b is defined as $1/b1$, where $b1$ is how often a susceptible –infected contact occurs. For example, if $b1 = 4$ then every 4 days there is a contact.

Code the SIR model equations as a function using the easiest differentiating algorithms you can think of (or matlab ODE solvers if you wish). Test your model with some chosen average time to recover($k1$; I used 14) and $b1$ values ranging from 1 to 10. Compare the effect of these changes in $b1$ to how the population responds.

NOTE: Your function must additionally take as input an initial VERY SMALL infected rate. Otherwise no one will be infected and the model makes no sense. Your model should have an initial susceptible rate $S(t = 0) = 1$ and an initial $R(t = 0) = 0$. Of course, initially $S + I + R$ will not be exactly one but the initial I is quite small.

```
%% Part (3)
% OPTIONAL: COVID-19 can be modeled with the SIR model. The model consists
% of three coupled differential equations
for b1 = 1 : 10 % Try 10 different values
    k1 = 14; % Define the average time to recover

    S_0 = 1; % Initial susceptible rate
    I_0 = 0.01; % Initial infected rate (very small)
    R_0 = 0; % Initial recovered rate
    figure_num = 6;
    max_days = 200; % Solve the model for 200 days
    [p1,p2,p3] = covid_sir(S_0,I_0,R_0,k1,b1,max_days,figure_num);
end

% Set a colorbar to indicate the value of "b1" at each curve
% Yellow corresponds to b1=1 and red corresponds to b1=10
h = colorbar;
set(h,'Limits',[0.1 1]); % Keep colors from yellow to red
colormap([ones(10,1) [1:-0.1:0.1]' zeros(10,1)]);
set(h,'ticklabels',[1:1:10]); % Change the tick labels to show the values of parameter b1

% Set a corresponding legend
legend([p1(end) p2(end) p3(end)],'Susceptible (S)','Infected: (I)','Recovered/Removed: ...
    (R)','interpreter','latex');
xlabel('Time (Days)','interpreter','latex'); % x-axis (time in days)
ylabel('Normalized Population','interpreter','latex'); % y-label (normalized population)
ylabel(h, 'Values for parameter ``$b_{1}$"','interpreter','latex','Fontsize',14); % ylabel of ...
    colorbar (values of b1)
ylim([0 1.3]) % Increase the size to avoid overlapping with legend
% Set appropriate title to explain this figure
title({'LAB \# 4 - Part (3) - COVID-19 SIR Model','Explore the parameter ...
    ``$b_{1}$"'},'fontsize',14,'interpreter','latex');
print(figure(figure_num),'lab_4_part_3','-depsc','-r600'); % Save figure in a colored .eps format ...
    using 600 dpi resolution
```

```
function [p1,p2,p3] = covid_sir(S_0,I_0,R_0,k1,b1,max_days,figure_num)
% function covid_sir(S_0,I_0,R_0,k1,b1,max_days,figure_num)

% Description: This function applies a SIR model to simulate the behavior
% of COVID-19

% INPUTS:
% S_0: Initial susceptible rate
% I_0: Initial infected rate (very small)
% R_0: Initial recovered rate
% k1: Average time to recover
% b1: How often a susceptible-infected contact occurs
% max_days: Number of days for which the system should be simulated for
% figure_num: Number of figure to be utilized for plotting

% OUTPUTS:
% p1: variable linked to the first curve plotted on the figure (S)
% p2: variable linked to the first curve plotted on the figure (I)
% p3: variable linked to the first curve plotted on the figure (R)

k = 1/k1; % Define the average period of infectiousness
b = 1/b1; % Frequency of susceptible-infected contacts

syms x(t) [3 1] real; % Define three symbolic variable, each for one the population groups
% Let S = x1(t), I = x2(t) and R = x3(t)
%Eq.(3.1)
eqn1 = diff(x1(t),t) == -b*x1*x2;
%Eq.(3.2)
eqn2 = diff(x2(t),t) == b*x1*x2 - k*x2;
%Eq.(3.3)
eqn3 = diff(x3(t),t) == k*x2;

% Group all equations together
eqns = [eqn1, eqn2, eqn3];
% Group all variables together
vars = [x1(t) x2(t) x3(t)];

% Determine the mass matrix form for the ode45 solver
[M,F] = massMatrixForm(eqns,vars);
f = M\F;

% Set initial condition for the ode45 solver
init_conds = [S_0 I_0 R_0];
```

```
44
45 % Group together all information about the ODE
46 odefun = odeFunction(f,vars);
47 t_total=[]; %In this vector we will store all time variables that the ode45 solver returns
48 sol=[];     %In this vector we will store all state variables that the ode45 solver returns
49
50 i=1;    %Iteration counter for the loop
51 i_th = max_days; %Maximum Number of Iterations (100 days)
52 opts = odeset('RelTol',1e-6,'AbsTol',1e-6); %Absolute and Relative error tolerances for ode45
53 max_step_size = 1; % Time period for which the ode is solved at each step
54 while(1) % Iterate forever until a certain condition is met
55     [t_i,sol_i] = ode45(odefun,[0 max_step_size],init_conds,opts); %Solve the system's equations ...
           using the specified ICs for 1 day with the error tolerances "opts" specified earlier
56     t_total=[t_total;t_i+((i-1)*max_step_size)];    %Adjust the local time vector, so that it ...
           corresponds to the total time elapsed and append it to the total time vector
57     sol=[sol;sol_i];    %Append the local solutions for the state to the total state solutions vector
58     init_conds=[sol(end,1); sol(end,2); sol(end,3)];    %Set the last state of the system for this ...
           iteration, as ICs for the next one
59     i = i + 1; % Increase the iteration counter
60     if i > i_th % Stop simulation when the desired number of days has been reached
61         break
62     end
63 end
64
65 fig7 = figure(figure_num); % Open a new figure
66 hold on; % Keep all plots at the same figure
67 % Plot the time evolution of the susceptible rate
68 p1 = plot(t_total(1:50:end),sol(1:50:end,1),'--','color',[1, 1-b1/10, 0]);
69 % Plot the time evolution of the infected rate
70 p2 = plot(t_total(1:50:end),sol(1:50:end,2),'-','color',[1, 1-b1/10, 0]);
71 % Plot the time evolution of the recovered rate
72 p3 = plot(t_total(1:50:end),sol(1:50:end,3),'-.','color',[1, 1-b1/10, 0]);
73 end
```

Discussion about the effect of parameter b_1
As it can be observed in Fig.8, as the parameter b_1 increases, the response of the system gets slower. Specifically, it takes more time (days) for the infected population to reach a peak, as well as for the system to reach a balance. Basically, this parameter controls how fast the population will get infected.

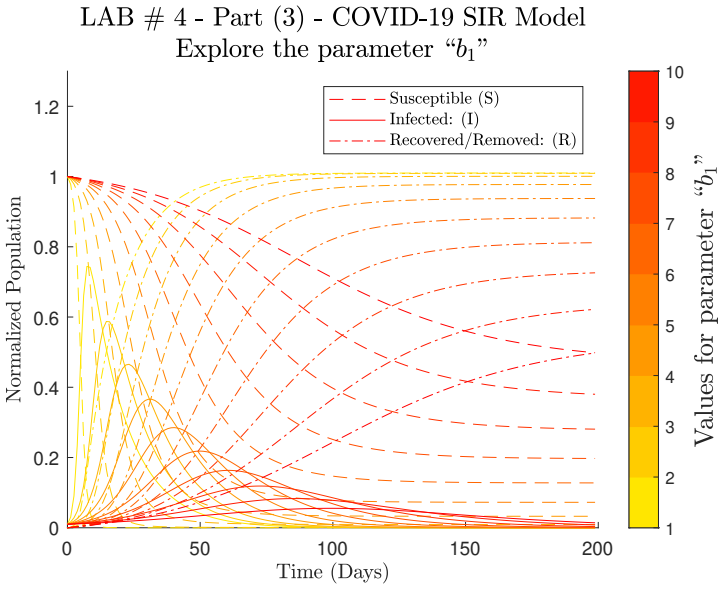

Figure 6: Lab # 4 - Part (3) - COVID-19 SIR model - Explore the effect of parameter b_1.

4 Part (4)

Solve $\frac{dy}{dx} = -y(x)$ subject to initial condition $y(0) = 1$. Note the real answer is e^{-x}. Do the solution in matlab two different ways.
A) Solve using a forward difference.
B) Solve using a central difference.

For each solution approach, use three different dx values and make a table showing the root-mean-square error (compared to the real answer) as a function of dx and numerical scheme. Also, show a plot of your solutions.

OPTIONAL: Solve the same equation by exploring the use of the functions ODE23 and/or ODE45. These are matlab's built in differential equation solvers (there are others) for differential equations over a certain range in independent variable and with a known boundary condition. Make a plot of your solutions from 1a (A and B) with the solution from one of these ODE solvers.

```
%% Part (4)
% Solve dy/dx = -y(x) subject to initial condition y(0)=1. Note the real
% answer is exp(-x). Do the solution in matlab two different ways.
% A) Solve using a forward difference.
% B) Solve using a central difference.
% For each solution approach, use three different dx values and make a
% table showing the root-mean-square error (compared to the real answer) as
% a function of dx and numerical scheme. Also, show a plot of your
% solutions.

% OPTIONAL: Solve the same equation by exploring the use of the functions
% ODE23 and/or ODE45.  These are matlab's built in differential equation
% solvers (there are others) for differential equations over a certain
% range in independent variable and with a known boundary condition.  Make
% a plot of your solutions from 1a (A and B) with the solution from one of
% these ODE solvers.
clear all; % Delete any previous data

% Part (4.a) - Forward Difference
% df(x)/dx = {f(x+Dx)-f(x)}/Dx => y(x+Dx) = y_dot(x)*Dx + y(x)

% Part (4.b) - Central Difference
% df(x)/dx = (1/2)*{f(x+Dx)-f(x-Dx)}/Dx

% In this case you wish to solve this equation y_dot(x) = -y(x)
dx_v = [0.5 0.1 0.001]; % Vector containing the three values for dx

fig7 = figure(7); % Open a new figure
clf; % Delete everything in this figure in case it was open before execution

for i = 1:3 % Repeat the same process for all three different dx values
    clear x y_f y_c; % Clear every time the variables that will be utilized later
    dx = dx_v(i); % Load the desired dx value for this iteration (Step size)
    endx = 4; % Where you want to stop calculations
    y_f(1) = 1;  % the initial value in the y vector corresponding to x=0 (forward case)
    y_c(1) = 1;  % the initial value in the y vector corresponding to x=0 (central case)

    x(1) = 0; % initial x location
    ct = 1;  % initialize counter

    while x(ct)<endx % Keep solving the ODE until reaching the endx threshold
        ct = ct+1; % increment counter
        x(ct) = x(ct-1)+dx;   % increment x (keep it to simulate real solution later)

        % Forward Difference: df(x)/dx = {f(x+Dx)-f(x)}/Dx => y(x+Dx) = y_dot(x)*Dx + y(x)
        y_f(ct) = y_f(ct-1)-dx*y_f(ct-1);  % find the new y for the new x using the forward difference

        % Central Difference: df(x)/dx = (1/2)*{f(x+Dx)-f(x-Dx)}/Dx =>
        % y_dot(x)*Dx*2 = y(x+Dx)-y(x-Dx)=> y(x+Dx) = y(x-Dx) + y_dot(x)*Dx*2
        y_c(ct) = y_c(ct-1)+2*dx*(-y_c(ct-1)); % find the new y for the new x using the central ...
            difference
    end
    yreal = exp(-x);  % this is the real solution found from integration

    subplot(3,1,i); % Plot the results for all three cases
    hold on; % Plot everything in the same subplot
    plot(x,yreal,'k'); % Plot the real solution
    plot(x,y_f,'ro'); % Plot the approximate solution for the forward case
    plot(x,y_c,'bd'); % Plot the approximate solution for the central case
    legend('real soln','approximation - forward','approximation - central','interpreter','latex'); % ...
        Set appropriate legend
    title({['$dx=$',num2str(dx)],['Forward RMS: ',num2str(rms(yreal-y_f)),' - Central RMS: ...
        ',num2str(rms(yreal-y_c))]},'interpreter','latex');
    sgtitle({'LAB \# 4 - Part (4) - Numerical Methods','Explore the parameter ...
        ``$dx$"'},'fontsize',14,'interpreter','latex');
    xlabel('x');
    ylabel('y');
end

% Time for the optional part

% ODE Solver
syms y(t) [1 1] real; % Define one symbolic variable
% Plug in the desired ODE
```

```
eqn1 = diff(y1(t),t) == -y1; % Define the equation we wish to solve

eqns = [eqn1]; % (Optinal step, mostly useful when we wish to solve numerous coupled equations like ...
    in Part (3))
vars = [y1(t)]; % (Optinal step, mostly useful when we wish to define numerous coupled variables like ...
    in Part (3))
[M,F] = massMatrixForm(eqns,vars); % Retrieve the mass matrix form (might be an overkill but I am ...
    used to this methodology)
f = M\F; % Retrieve the system that we wish to solve in a compact form

% Set the desired initial conditions for the ODE
init_conds = [1];

odefun = odeFunction(f,vars); % Plug in everything using the odefun function to utilize afterwards ...
    the ODE45 solver
t_total=[]; % In this vector we will store all time variables that the ode45 solver returns
sol=[];     %In this vector we will store all state variables that the ode45 solver returns

i=1;        % Iteration counter for the outer loop
i_th = 100; % Maximum Number of Iterations
opts = odeset('RelTol',1e-6,'AbsTol',1e-6); %Absolute and Relative error tolerances for ode45
max_step_size = dx_v(3); % Utilize the best combination from above
while(1) % Keep solving the ODE until reaching the desired number of time steps
    [t_i,sol_i] = ode45(odefun,[0 max_step_size],init_conds,opts); %Solve the ODE using the specified ...
        ICs for dx time steps with the error tolerances "opts" specified earlier
    t_total=[t_total;t_i+((i-1)*max_step_size)]; % Adjust the local time vector, so that it ...
        corresponds to the total time elapsed and append it to the total time vector
    sol=[sol;sol_i]; % Append the local solutions for the state to the total state solutions vector
    init_conds=[sol(end,1)]; % Set the last state of the system for this iteration, as ICs for the ...
        next one
    i = i + 1; % Increase the iteration counter
    if t_total(end) > endx % Check whether the system has reached the desired number of time steps
        break % Break in case the desired number of time steps has been reached
    end
end

yreal_ode = exp(-t_total);  % this is the real solution found from integration (for this time step)

fig8 = figure(8); % Open a new figure
clf; % Clear everything in this figure
hold on; % Plot everything in the same figure
plot(x,yreal,'k'); % Plot the real solution
plot(x,y_f,'ro'); % Plot the approximate solution using the forward difference (was more accurate ...
    than central)
plot(t_total,sol,'bd'); % Plot the ode45 solution
legend('real soln','approximation - forward','ode45','interpreter','latex'); % Set appropriate legend
title({'Compare Forward Difference to ODE45',['Forward RMS: ',num2str(rms(yreal-y_f)),' - ODE45 RMS: ...
    ',num2str(rms(yreal_ode-sol))]},'interpreter','latex');
xlabel('x');
ylabel('y');

set(fig7,'units','normalized','Position',[0.2641    0.3657    0.3812    0.5389]); % Increase the ...
    figure size a bit.
print(fig7,'lab_4_part_4_dxs','-depsc','-r600'); % Save figure in a colored .eps format using 600 dpi ...
    resolution
print(fig8,'lab_4_part_4_for_vs_ode45','-depsc','-r600'); % Save figure in a colored .eps format ...
    using 600 dpi resolution
```

```
function [num_sec] = day2secthat(mat_dates_v)
% function [num_sec] = day2secthat(mat_dates_v)

% Description: This function receives as an input a vector of MATLAB dates
% and returns as an output a vector containing the number of seconds that
% have elapsed since the 1st value (date) in the input vector

mat2str = datestr(mat_dates_v); % Convert the input vector to string format
str2vec = datevec(mat2str); % Convert the string format to a vector of components
num_sec = []; % Initialize the output vector that will contain the elapsed number of seconds
for i = 1:size(str2vec,1) % Iterate through all dates (elements) of the input vector
    % Calculate the elapsed number of seconds of each date wrt to the
    % first one and add it to the output vector.
    num_sec = [num_sec etime(str2vec(i,:),str2vec(1,:))];
end
end
```

	Forward Difference	Central Difference
dx = 0.5	0.070426	0.45425
dx = 0.1	0.012658	0.16003
dx = 0.001	0.00012415	0.14433

Table 1: RMS errors of numerical methods wrt to the real solution of the differential equation.

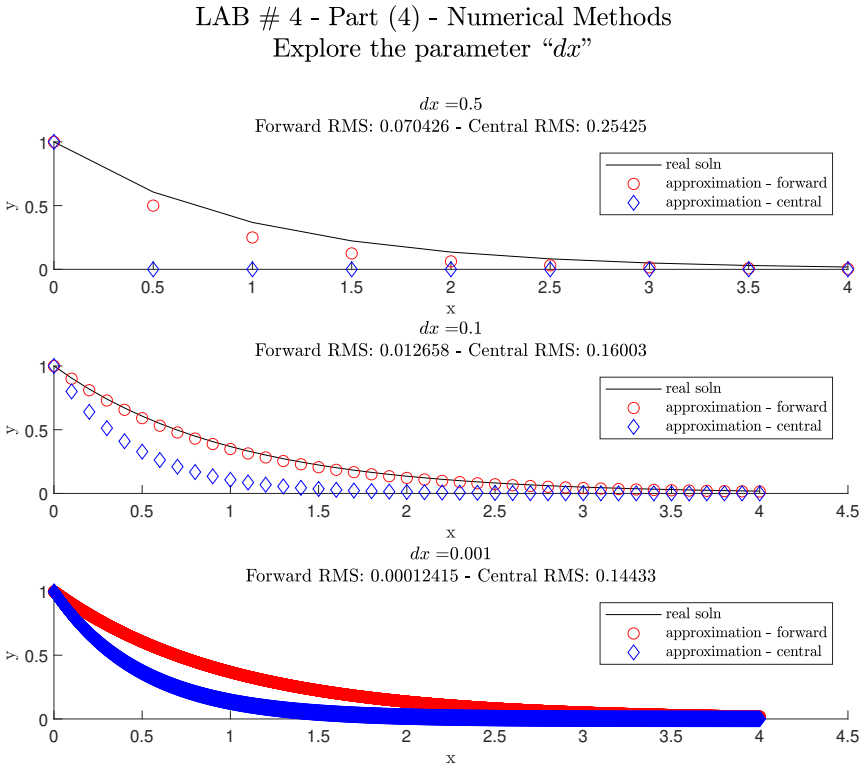

Figure 7: Lab # 4 - Part (4) - Comparison between forward and central approximations.

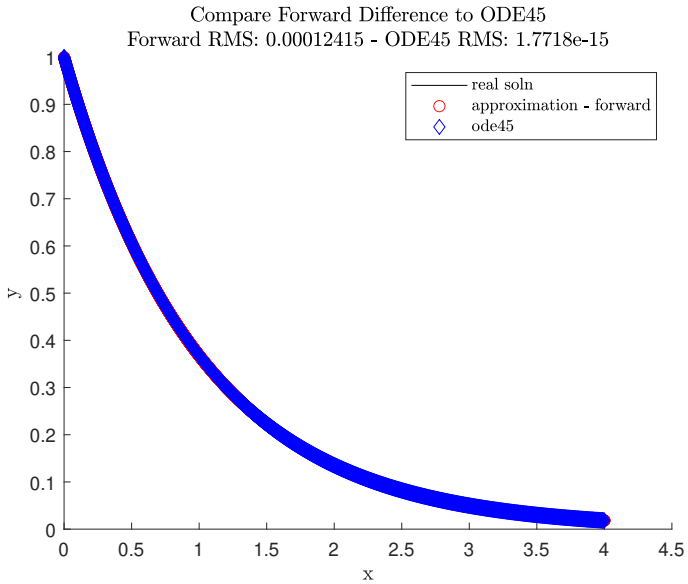

Figure 8: Lab # 4 - Part (4) - Comparison between forward and ode45 approximations.

5 Part (5)

Load the image *speed_ limit.JPG* from canvas and explore the matlab function called 'edge' to perform edge detection on the simple image. Obtain the output from edge and change the background image color to yellow and the letter and number edges to blue. There are multiple ways to do this and you will need to figure out how to go from the original RGB image to grayscale for analysis and then back to RGB for output.

```
%% Part (5)
% Load the image speed_limit.JPG from canvas and explore the matlab
% function called 'edge' to perform edge detection on the simple image.
% Obtain the output from edge and change the background image color to
% yellow and the letter and number edges to blue. There are multiple ways
% to do this and you will need to figure out how to go from the original
% RGB image to grayscale for analysis and then back to RGB for output.

% Load the image which should located in the same folder as this code
im = imread('speed_limit.JPG');

% Convert image from RGB to grayscale
I = rgb2gray(im);

% Detect edges using the Sobel method
BW1 = edge(I,'sobel');

% After the edge detection, matrix BW1 will contain for each element of the
% image either a 0 or a 1, in order to indicate whether that that
% element-pixel corresponds to an edge or not (background)

% Turn all zeros to yellow( = [1 1 0] in RGB) - background elements
[back_row,back_col] = find(BW1 == 0);
% Turn all ones to blue(=[0 0 1] in RGB) - letter/numbers edges
[let_num_row,let_num_col] = find(BW1 == 1);

% In order to change the colors of the image in the RGB format, we have to
% specify for each element the corresponding RGB combination that
% corresponds to the desired color. For instance, im(1,1,1) corresponds to
% the R(red) percentage of the first "pixel", im(1,1,2) corresponds to the
% G(green) percentage of the first "pixel" and finally im(1,1,3)
% corresponds to the B(blue) percentage of the first "pixel". Hence, the
% above indices correspond to the location of every pixel that we will to
% change its color.

for i = 1:length(back_row) % Go through all background elements-pixels
    im(back_row(i),back_col(i),1) = 255; % Change the R(red) part to 1(255)
    im(back_row(i),back_col(i),2) = 255; % Change the G(green) part to 1(255)
    im(back_row(i),back_col(i),3) = 0; % Change the B(blue) part to 0(0)
    % Hence, now all background pixels have the color [1 1 0] which corresponds
    % to yellow
end
for i = 1:length(let_num_row) % Go through all edge elements-pixels
    im(let_num_row(i),let_num_col(i),1) = 0; % Change the R(red) part to 0(0)
    im(let_num_row(i),let_num_col(i),2) = 0; % Change the G(green) part to 0(0)
    im(let_num_row(i),let_num_col(i),3) = 255; % Change the B(blue) part to 255(1)
    % Hence, now all edge pixels have the color [0 0 1] which corresponds
    % to blue
end

figure(9); % Open a new figure
imshow(im); % Show the modified image
imwrite(im,'speed_limit_mod.JPG'); % Save the modified image
```

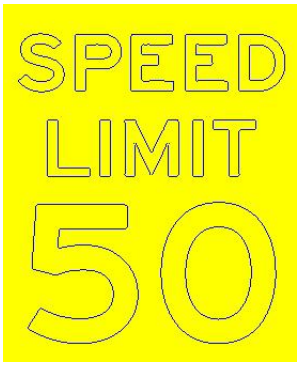

Figure 9: Lab # 3 - Part (5) - Modified image with yellow background and blue edges.