Computational Laboratory #1

Due: Class time, September 5, 2023

Problem 1: Difference Equation Analysis. Decay functions pervade biophysics.

$$\frac{dy}{dt} = -ky \qquad (1)$$

where the parameter k is a decay constant. If you were to apply Euler's method to the above, the expression realized on the computer would represent a difference equation like the below:

$$w_{i+1} = w_i - k\Delta t * w_i \qquad (2)$$

where w is the estimate of the value of y on the computer. A different approach could produce an approach that would look like:

$$w_{i+1} = w_i + \frac{\Delta t}{2}(-2k * w_i + k^2\Delta t * w_i) \qquad (3)$$

(a) In the notes I provided a quick refresher on solving continuous ODEs. Solve for the analytic solution of (1). For this problem assume *y(t=0)*=1.0.

$$\frac{dy}{dt} = -ky$$
$$\text{Characteristic equation is } \ldots s + k = 0$$
$$s = -k$$
$$y(t) = Ae^{-kt}$$
$$at\ t = 0, y = 1.0, so\ A = 1.0$$
$$y(t) = e^{-kt}$$

(b) Now solve for the closed form solution of the *difference equations* associated with (2), and (3) in terms of iteration # *N*. An easy check is to compare the output of your solution to the sequence that unfolds from (2) and (3) (to start (2) and (3) off, you need the first value which in this case would be w_1=1).

$$w_{i+1} = w_i - k\Delta t * w_i$$
$$w_{i+1} = (1 - k\Delta t)w_i$$
$$\text{Characteristic equation is } \ldots \lambda - (1 - k\Delta t) = 0$$
$$\lambda = (1 - k\Delta t)$$
$$w_N = A(1 - k\Delta t)^N$$
$$at\ N = 0, w_N = 1.0$$
$$w_N = (1 - k\Delta t)^N$$

Similarly,

$$w_{i+1} = w_i + \frac{\Delta t}{2}(-2k * w_i + k^2\Delta t * w_i)$$
$$w_{i+1} = w_i\left(1 - \Delta tk + \frac{\Delta t^2 k^2}{2}\right)$$
$$\text{Characteristic equation is } \ldots \lambda - \left(1 - \Delta tk + \frac{\Delta t^2 k^2}{2}\right) = 0$$

$$\lambda = \left(1 - \Delta t k + \frac{\Delta t^2 k^2}{2}\right)$$

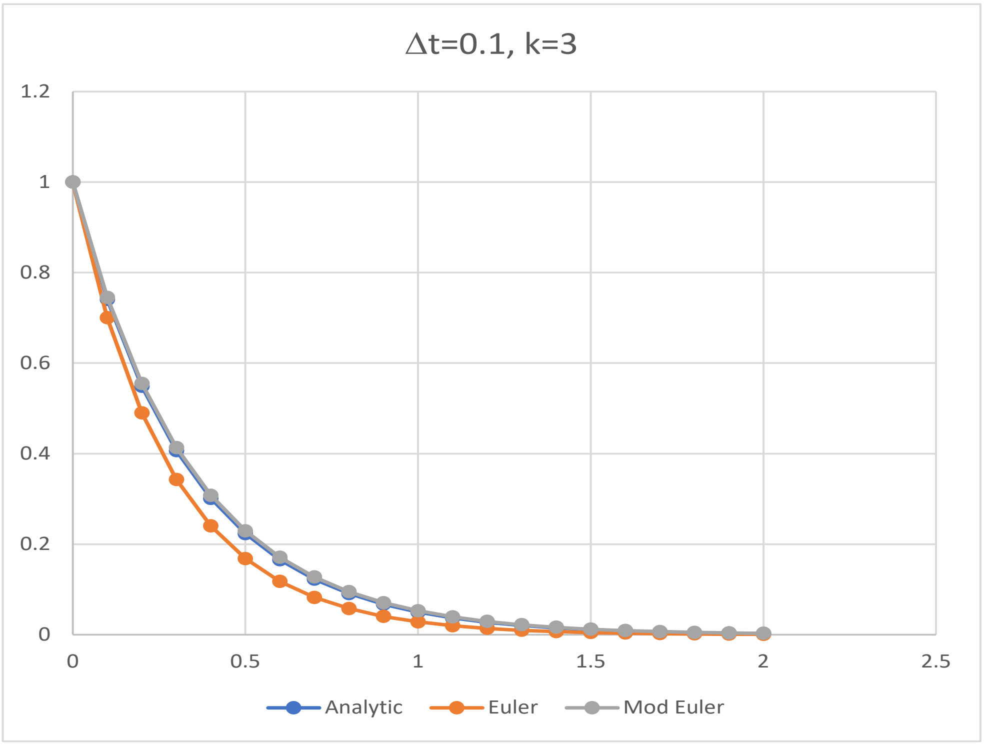
$$w_N = A\left(1 - \Delta t k + \frac{\Delta t^2 k^2}{2}\right)^N$$

$$at\ N = 0, w_N = 1.0$$

$$w_N = \left(1 - \Delta t k + \frac{\Delta t^2 k^2}{2}\right)^N$$

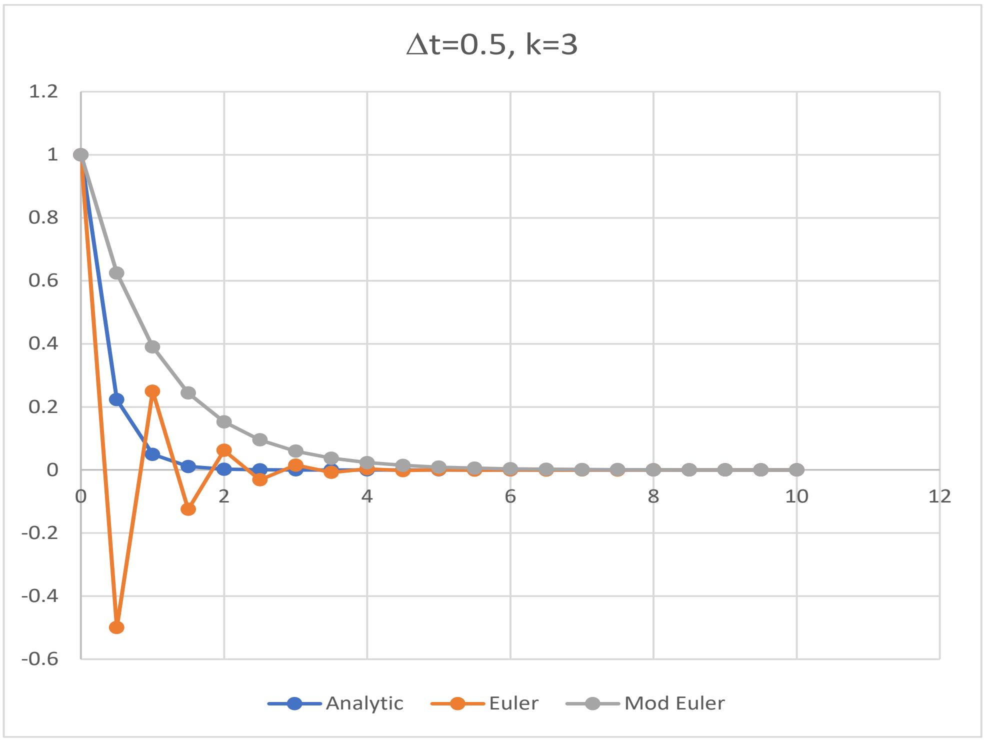
(c) Using your closed form solutions from (a), and (b), make an overlay of the 3 solutions for each of the conditions below. In completing this problem, you will generate 3 plots with 3 curves overlaid on each plot.

- First use *Δt=0.1,* and *k=3* and plot over the domain $0 \leq t \leq 2$ for (1), or $0 \leq N \leq 20$ for (2), and (3)



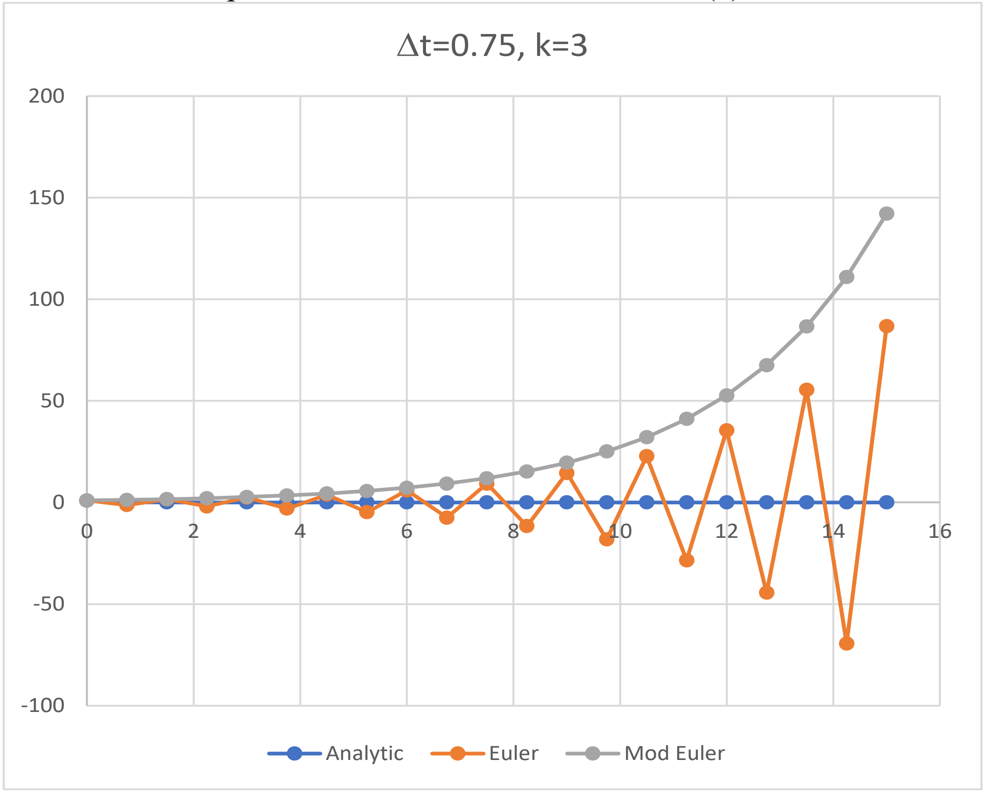
Nice match with Mod Euler outperforming Euler.

- Next, $\Delta t=0.5$, and $k=3$ and plot over the domain $0 \leq t \leq 10$ for (1), or $0 \leq N \leq 20$ for (2), and (3)



Mod Euler is a better predictor but clearly a lot of error. Euler creates artifactual oscillatory behavior.

- Next, $\Delta t=0.75$, and $k=3$ and plot over the domain $0 \leq t \leq 15$ for (1), or $0 \leq N \leq 20$ for (2), and (3)



Both Euler and Mod Euler are unstable.

(d) Now look back on how error propagates and can amplify with respect to difference equations. Can you explain your findings in part (c)?
So first… recall that the error acts the same way as the sequence ….

$$\varepsilon_{i+1} = (1 - k\Delta t)\varepsilon_i \rightarrow \text{amplification factor} = 1 - k\Delta t$$

$$\varepsilon_{i+1} = \varepsilon_i \left(1 - \Delta t k + \frac{\Delta t^2 k^2}{2}\right) \rightarrow \text{amplification factor} = 1 - \Delta t k + \frac{\Delta t^2 k^2}{2}$$

What we note is that with k=3, and Δt = 0.1, the amplification factors equal 0.7, and 0.745 respectively. This is less than one so will not go unstable, hence the results in (c, plot 1).

What we note is that with k=3, and Δt = 0.5, the amplification factors equal -0.5, and 0.625 respectively. While these are both less than one so they are stable, we see that the amplification factor for Euler is -0.5. This indicates that with each step the error will flip sign, hence the oscillatory behavior is explained.

What we note is that with k=3, and Δt = 0.75, the amplification factors equal -1.25, and 1.28 respectively. These are both greater than 1 so they will go unstable. We see with the negative factor associated with Euler, it grows but the solution is oscillatory. With Mod Euler, it grows, but does not oscillate which reflects the positive value.

(e) Lastly,

- Solve equation (2) with the decay rate equal to *k=3,* and *Δt=0.2* from $0 \le t \le 2$. Now solve the equation again with *Δt=0.1*. Compare your solution to the analytic solution of (a), and calculate the ℓ_2 norm of the error for the vector of error.
- Solve equation (3) with the decay rate equal to *k=3,* and *Δt=0.2* from $0 \le t \le 2$. Now solve the equation again with *Δt=0.1*. Compare your solution to the analytic solution of (a) at each time step. This will produce a vector of errors. Calculate the ℓ_2 norm of the error vector for each condition.
- Once calculated fill in this table:

	Error (*Δt=0.2*)	Error (*Δt=0.1*)
Method 1 (eq 2)	0.2428	0.1511
Method 2 (eq 3)	0.0637	0.0174

What you should see is that halfing the step size reduces the error by approximately 4 whereas for Mod Euler, where is approximately half for Euler.

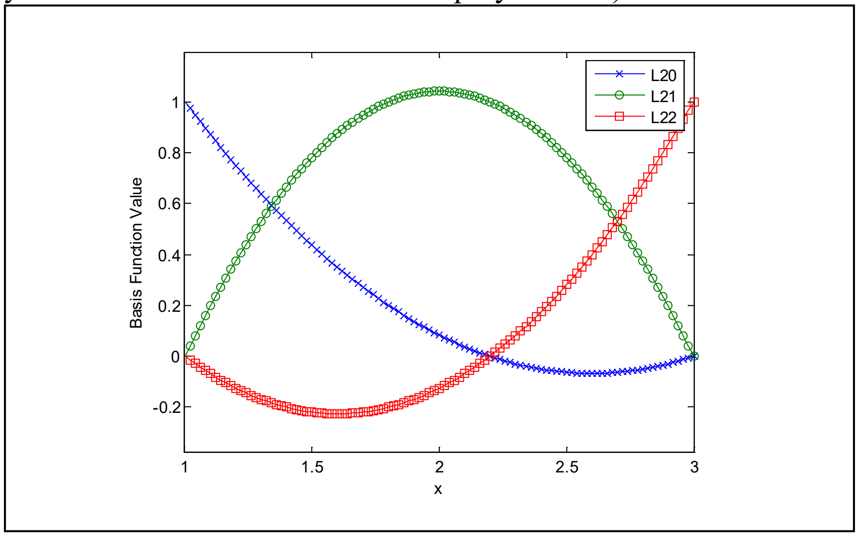
Problem 2: Lagrange Polynomial. Lagrange polynomials are extremely useful. They take a set of discrete function values and very elegantly allow for meaningful local interpolation among values to arbitrary order.

a. On your remediation videos, you were given the general form of a Lagrange Polynomial.

$$\mathbf{L}_{N,k}(\mathbf{x}) = \prod_{\substack{i=0 \\ i \neq k}}^{N} \frac{\mathbf{x} - \mathbf{x}_i}{\mathbf{x}_k - \mathbf{x}_i}$$

For this assignment, you are given three points x0, x1, x2. You are to generate the 3 quadratic basis functions for $\mathbf{L}_{2,0}, \mathbf{L}_{2,1}, \mathbf{L}_{2,2}$ centered about x0, x1, x2, respectively, by modifying the m-file provided. To demonstrate your results, the m-file has been designed to evaluate your functions at 100 points in the interval between [x0, x2]. Evaluate each basis function at each of these points and store the evaluation values for each Lagrange polynomial. Plot them versus the interval. Use the legend command to delineate each basis function. For this task, assume: x0=1.0, x1=2.2, and x2=3.0

In the window below, show the plots of the values of $\mathbf{L}_{2,0}, \mathbf{L}_{2,1}, \mathbf{L}_{2,2}$ based on the x0, x1, x2 values provided (note you should have 3 curves for the 3 polynomials):



b. Now assume that f(x0)=3, f(x1)=6, f(x2)=4. Report the following values in the box:

For X = 1.1
L20(X)=0.870833
L21(X)=0.197917
L22(X)=-0.068750
F(X)= 3.525000

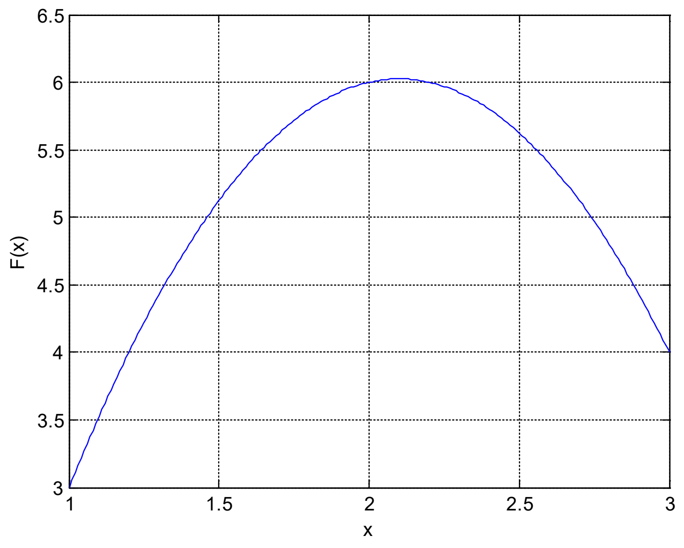
For X = 2.0
L20(X)=0.083333
L21(X)=1.041667
L22(X)= -0.125000
F(X)=6.0

For X = 2.2
L20(X)= 0.0
L21(X)= 1.0
L22(X)= 0.0
F(X)= 6.0

For X = 1.75
L20(X)= 0.234375
L21(X)= 0.976563
L22(X)= -0.210938
F(X)= 5.718750

For X = 2.5
L20(X)= -0.062500
L21(X)= 0.781250
L22(X)= 0.281250
F(X)= 5.625

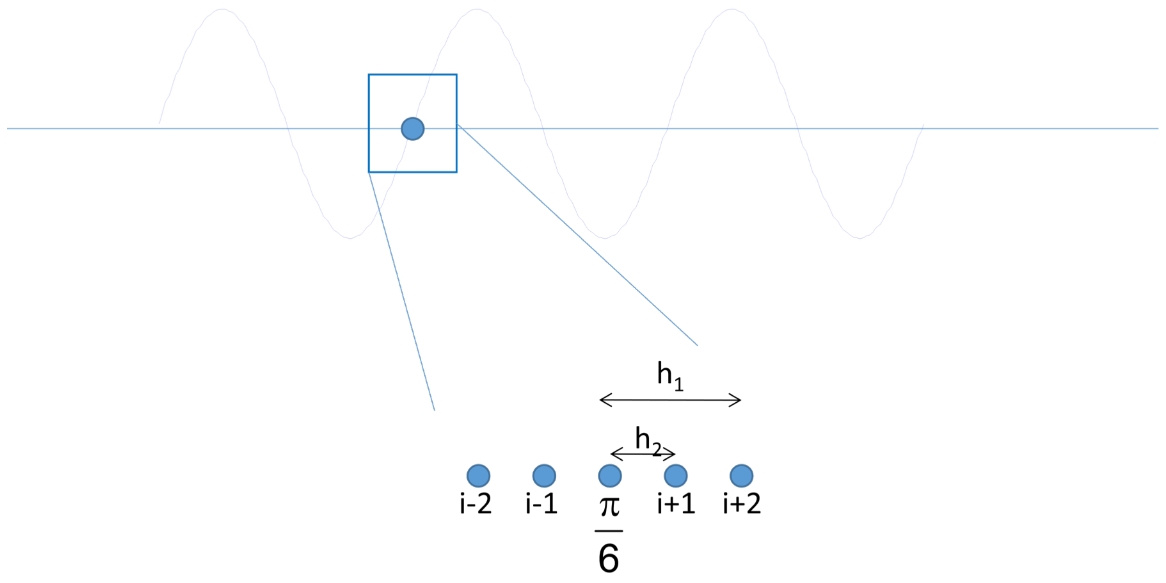
c. With respect to the functional values in part (b) above, interpolate a value for every value for $1 \leq x \leq 3$. Put the graphed function below to see how the Lagrange polynomial interpolated the data on the range.



d. Place your MATLAB code in the box below as to how you completed this problem.

```
function [xr,L20,L21,L22]=lagrange_quadratic_complete(x,yvals,xi)
% Function assumes x0 < x1 < x2
% Inputs:
%         x      :    the three x values [x0 x1 x2]
%         yvals  :    the specific values of your function at each x0,x1,x2
%         xi     :    the specific value for x that you want to interpolate
%                     your function too
x0=x(1);
x1=x(2);
x2=x(3);

dx=(x2-x0)/100;
xr=[x0:dx:x2];
h01=x1-x0;
h02=x2-x0;
h12=x2-x1;

nn=length(xr);

% In this loop, I just want you to evaluate the 3 Lagrange quadratic
% polynomials, L20, L21, L22 for the range of xvalues between x0 and and
% x2.  The values have been stored in 'xr' above.
for i=1:nn
    L20(i)=(xr(i)-x1)*(xr(i)-x2)/((-h01)*(-h02));
    L21(i)=(xr(i)-x0)*(xr(i)-x2)/((h01)*(-h12));
    L22(i)=(xr(i)-x0)*(xr(i)-x1)/((h02)*(h12));
end

% Now let's evaluate a sample interpolated value using the quadratic
% Lagrange basis.  Now you will not only evaluate the 3 basis functions at
% a specific value 'x', but you will calculate the interpolated value of
% your function at that specific value of 'x'.
L20v=(xi-x1)*(xi-x2)/((-h01)*(-h02));
L21v=(xi-x0)*(xi-x2)/((h01)*(-h12));
L22v=(xi-x0)*(xi-x1)/((h02)*(h12));

yv=L20v*yvals(1)+L21v*yvals(2)+L22v*yvals(3);

fprintf('Your Values for the Basis functions at the Xi\n');
fprintf('L20, L21, L22 = %f, %f, %f, respectively\n',L20v, L21v, L22v);
fprintf('f(Xi) %f\n',yv);
```

Problem 3: Numerical Differentiation. In class, we talked about how to generate a numerical derivative and the power of sampling with respect to the calculation of that derivative. To demonstrate this, we are going to do a small experiment. The task will be to evaluate numerically the value of the derivative for the *sin* function evaluated at π/6. We have also designated three separate sampling intervals h_1, h_2, and h_3 which are π/20, π/40, and π/80 respectively (note h_3 is not shown in graphic below but you need to do). Fortunately, we know the analytic answer to the derivative of a *sin* function to see how accurate our estimates are.



Create a MATLAB file to construct the derivative using the 3 expressions below with the 3 different sampling intervals h_1, h_2, and h_3.

Center Difference Expression: $\frac{du}{dt} = \frac{u_{i-2} - 8u_{i-1} + 8u_{i+1} - u_{i+2}}{12h}$

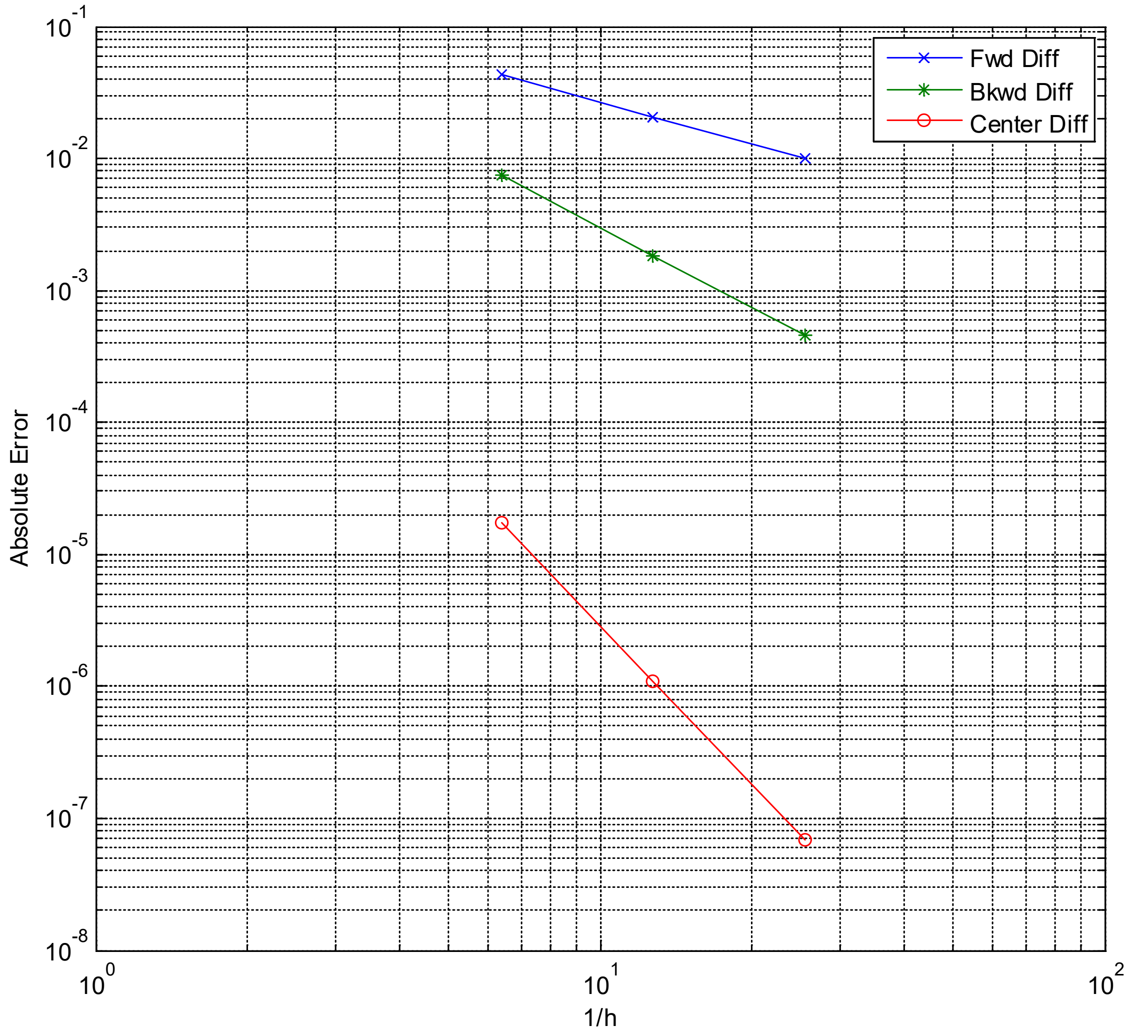
Backward Difference Expression: $\frac{du}{dt} = \frac{u_{i-2} - 4u_{i-1} + 3u_i}{2h}$

Forward Difference Expression: $\frac{du}{dt} = \frac{u_{i+1} - u_i}{h}$

1. After you have constructed the derivatives, be sure to compare your numerical answers to the analytic cosine function evaluated at π/6 to determine error. Once complete, fill in the below table. For error, report the absolute error:

Expr.	cos(π/6)	h_1 val.	h_2 val.	h_3 val.	Error for h_1	Error for h_2	Error for h_3
Cntr	0.866025403784	0.866007880594	0.866024306169	0.866025335145717	1.75231895e-05	1.09761485e-06	6.8638721706371e-08
Back	0.866025403784	0.873569309894	0.867862752074	0.866477904830691	0.007543906109	0.001837348290	4.5250104625227e-04
Fwd	0.866025403784	0.823279179126	0.845510468697	0.855986618817722	0.042746224658	0.020514935086	0.010038784966716

2. Now make a log-log plot of the error on the y-axis, and the reciprocal of each respective step size, i.e. on the x-axis, plot the values of (1/h) and on the y-axis plot the absolute error for each type of difference. Please provide this graph as an overlay for each line produced. Looking at the plot, what trends do you notice and how does that correlate with your understanding of the difference expressions given? How does the shape of the plot correlate with your understanding of the error associated with the FD approximation?



The error ratio (ratio of error for h1 to error for h2) is approximately 16 in the center difference, 4 is backward difference, and 2 is forward difference. Look at the number of points needed to calculate du/dt, and the orders of the error. Does this match the calculated error ratios considering you reduced the h value by a factor of 2? The backwards method uses one more point than the forward method, error cut by a factor of 4 (error controlled by an h^2 term). For the center difference one more point is used, (symmetry- error controlled by h^4 term)>> the error is cut by a factor of 16. So the FD expression is an order (h), the BD is order (h^2) and the CD is order (h^4). On a loglog plot of error versus 1/h, we should see a constant linear negative slope that reflects the order of the truncation error term – here we see, the slope of the FD is -1, the BD is -2, and the CD is -4.

3. Provide the M-file you used to generate your table in Part 1.

```
theta = pi/6;
h1 = pi/20;
h2 = pi/40;
h3 = pi/80;
%calculations for h1
uim1 = sin(theta-h1);
uim2 = sin(theta-2*h1);
uip1 = sin(theta+h1);
uip2 = sin(theta+2*h1);
ui = sin(theta);

CDh1 = (uim2-8*uim1+8*uip1-uip2)/(12*h1);
BDh1 = (uim2-4*uim1+3*ui)/(2*h1);
FDh1 = (uip1-ui)/h1;

%calculations for h2
uhim1 = sin(theta-h2);
uhim2 = sin(theta-2*h2);
uhip1 = sin(theta+h2);
uhip2 = sin(theta+2*h2);
uhi = sin(theta);

CDh2 = (uhim2-8*uhim1+8*uhip1-uhip2)/(12*h2);
BDh2 = (uhim2-4*uhim1+3*uhi)/(2*h2);
FDh2 = (uhip1-uhi)/h2;

%calculations for h3
uhim1 = sin(theta-h3);
uhim2 = sin(theta-2*h3);
uhip1 = sin(theta+h3);
uhip2 = sin(theta+2*h3);
uhi = sin(theta);

CDh3 = (uhim2-8*uhim1+8*uhip1-uhip2)/(12*h3);
BDh3 = (uhim2-4*uhim1+3*uhi)/(2*h3);
FDh3 = (uhip1-uhi)/h3;

Hinv=[1/h1;1/h2;1/h3];
ErrorFD=abs([cos(pi/6)-FDh1; cos(pi/6)-FDh2; cos(pi/6)-FDh3]);
ErrorBD=abs([cos(pi/6)-BDh1; cos(pi/6)-BDh2; cos(pi/6)-BDh3]);
ErrorCD=abs([cos(pi/6)-CDh1; cos(pi/6)-CDh2; cos(pi/6)-CDh3]);

loglog(Hinv,ErrorFD,'-x',Hinv,ErrorBD, '-*',Hinv,ErrorCD,'-o');
xlabel('1/h')
ylabel('Absolute Error')
legend('Fwd Diff', 'Bkwd Diff', 'Center Diff')
```