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.

Solution: Analytic solution to (1) has the form $y(t) = Ce^{-kt}$. From the initial condition:

$$y(0) = Ce^{-k.0} = 1.0$$

We get, $C = 1.0$

Thus, the analytic solution is $\boldsymbol{y(t) = e^{-kt}}$.

(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).

Here we seek solution of form $w = C.\lambda^N$ where *N* is number of iterations.

First, write (2) in characteristic equation:

$$\lambda = 1 - k\Delta t$$

Use initial condition, N=0:

$$w = C.(1 - k\Delta t)^0 = 1$$

Then,

$$C = 1$$

Thus, we get the closed form solution for (2) as:

$$\boldsymbol{w = (1 - k\Delta t)^N.}$$

Similarly, write (3) in characteristic equation:

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

Use initial condition:

$$w = C.\left(1 + \frac{\Delta t}{2}(-2k + k^2\Delta t)\right)^0 = 1$$

Then,

$$C = 1$$

Thus, we get the closed form solution for (3) as:

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

(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 $\Delta t=0.1$, and $k=3$ and plot over the domain $0 \le t \le 2$ for (1), or $0 \le N \le 20$ for (2), and (3)
- Next, $\Delta t=0.5$, and $k=3$ and plot over the domain $0 \le t \le 10$ for (1), or $0 \le N \le 20$ for (2), and (3)
- Next, $\Delta t=0.75$, and $k=3$ and plot over the domain $0 \le t \le 15$ for (1), or $0 \le N \le 20$ for (2), and (3)

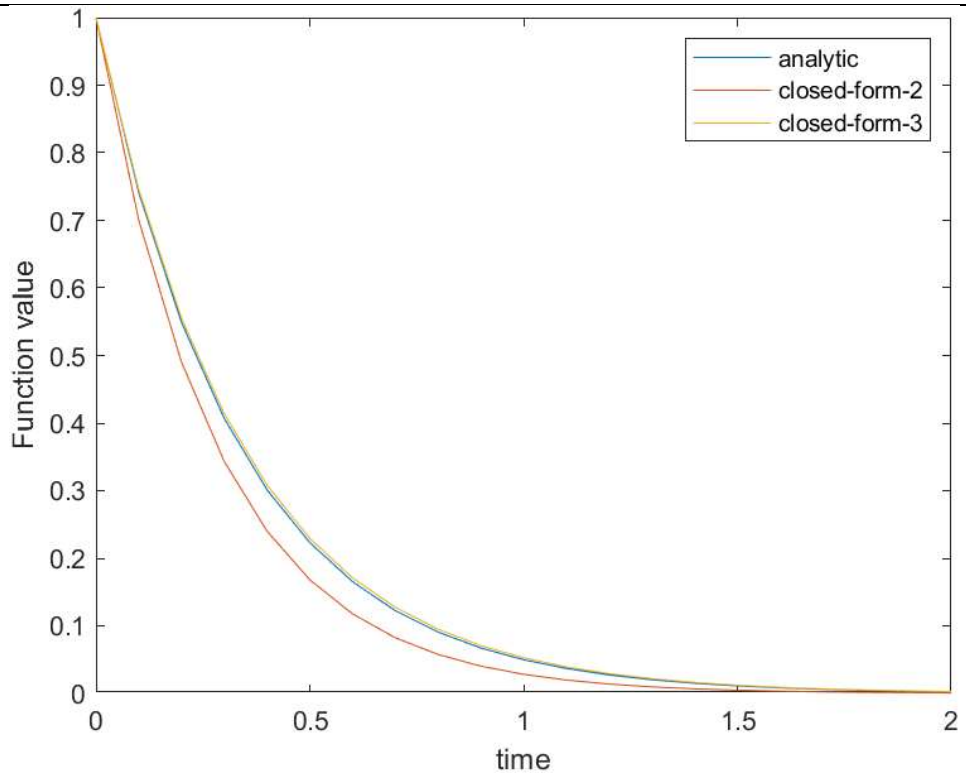


Figure 1. $\Delta t=0.1$, and $k=3$ and plot over the domain $0\leq t\leq 2$ for (1), or $0\leq N\leq 20$ for (2), and (3)

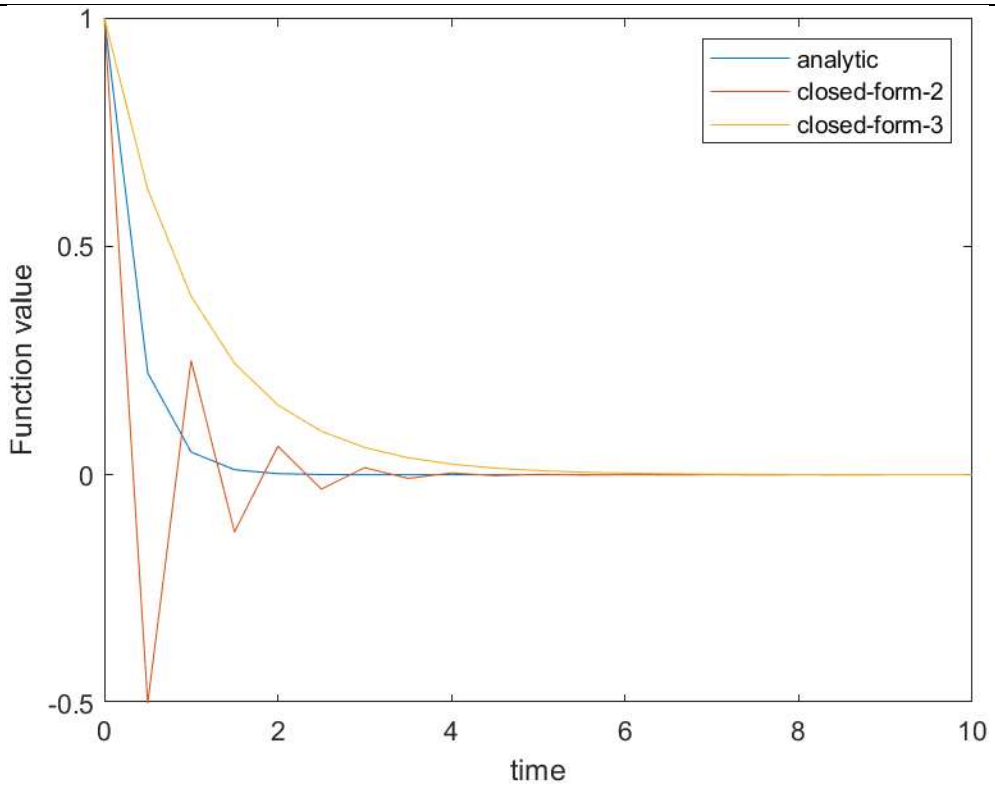


Figure 2. $\Delta t=0.5$, and $k=3$ and plot over the domain $0\leq t\leq 10$ for (1), or $0\leq N\leq 20$ for (2), and (3)

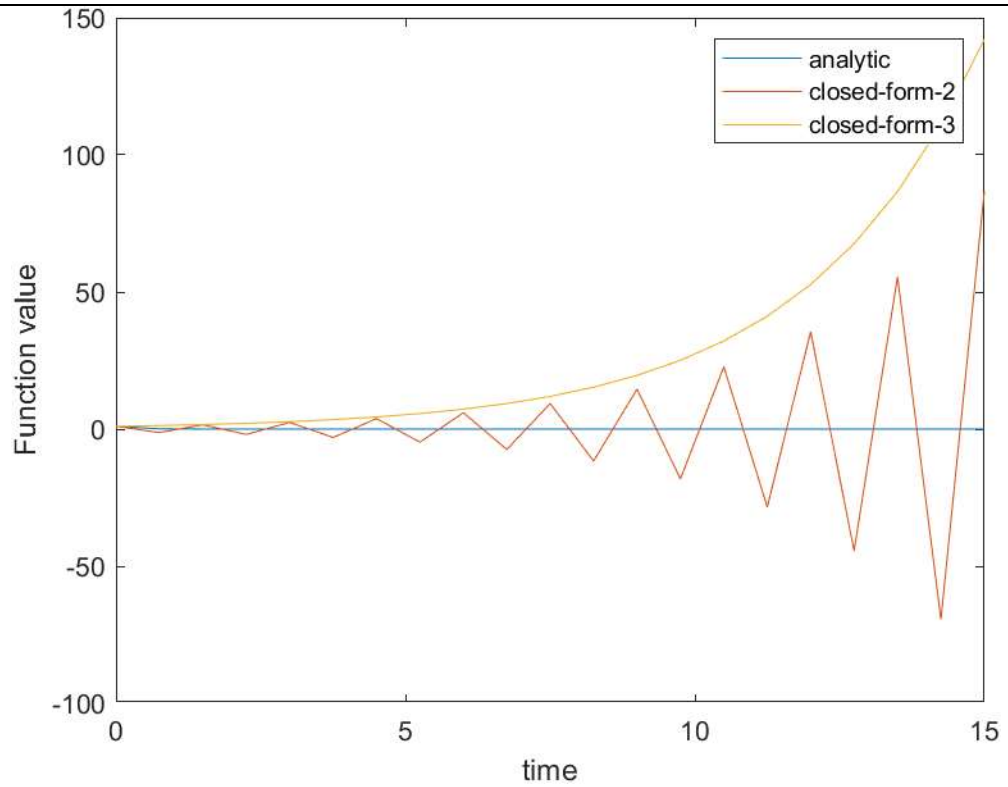


Figure 3. Δt=0.75, and k=3 and plot over the domain $0 \leq t \leq 15$ *for (1), or* $0 \leq N \leq 20$ *for (2), and (3)*

(d) Now look back on **how error propagates** and **can amplify with respect to difference equations**. Can you explain your findings in part (c)?

Here we get 'Round off' error:

We can represent (2) and (3) as $w = \alpha^N$ where $\begin{cases} \alpha = 1 - k\Delta t & \text{for (2)} \\ \alpha = 1 + \frac{\Delta t}{2}(-2k + k^2\Delta t) & \text{for (3)} \end{cases}$

$w = \alpha^N$ exact

$\hat{w} = \hat{\alpha}^N$ computer generated

$$\hat{w} - w = \hat{\alpha}^N - \alpha^N$$

$$\begin{aligned} \varepsilon_N &= (\alpha + \delta)^N - \alpha^N \\ &= \cancel{\alpha^N} + {}^N_1C\alpha^{N-1}\delta + C^N_2\alpha^{N-2}\delta^2 + \dots + \delta^N - \cancel{\alpha^N} \\ &= {}^N_1C\alpha^{N-1}\delta + C^N_2\alpha^{n-2}\delta^2 + \dots + \delta^N \\ &\simeq O(\alpha^{N-1}) \end{aligned}$$

- For $\Delta t = 0.1$, $k = 3$

(2): $|\alpha| = |1 - 3(0.1)| = \left|\frac{7}{10}\right| < 1$ → error $\varepsilon_n = O(\alpha^{N-1})$ dissipates

(3): $|\alpha| = \left|\frac{149}{200}\right| < 1$ → error $\varepsilon = O(\alpha^{N-1})$ dissipates

- For $\Delta t = 0.5$, $k = 3$

(2): $|\alpha| = |1 - 3(0.5)| = \left|\left(\frac{-1}{2}\right)\right| < 1$ → error dissipates

(3): $|\alpha| = \left|\frac{5}{8}\right| < 1$ → error dissipates

- For $\Delta t = 0.75$, $k = 3$

  (2): $|\alpha| = |1 - 3(0.75)| = \left|-\frac{5}{4}\right| > 1$ → error accumulates

  (3): $|\alpha| = \left|\frac{41}{32}\right| > 1$ → error accumulates

  Thus we see a diverged graph!

(e) Lastly,

- Solve equation (2) with the decay rate equal to $k=3$, and $\Delta t=0.2$ from $0 \leq t \leq 2$. Now solve the equation again with $\Delta t=0.1$. 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 $\Delta t=0.2$ from $0 \leq t \leq 2$. Now solve the equation again with $\Delta t=0.1$. 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 ($\Delta t=0.2$)	Error ($\Delta t=0.1$)
Method 1 (eq 2)	0.242787025884	0.151114500992
Method 2 (eq 3)	0.063674456963	0.017364751711

```
plot_equations_with_error_norm(0.2, 2, 3, 10);
plot_equations_with_error_norm(0.1, 2, 3, 20);

function [] = plot_equations_with_error_norm(delta_t, t_max, k, num_iter)
N = 0 : 1 : num_iter;
t = 0 : delta_t : t_max;

analytic_sol = exp(-k*t);
close_form_2 = (1-k*delta_t).^N;
close_form_3 = ( 1 + (delta_t / 2) * (-2*k + k^2 * delta_t) ).^N;

error_close_2 = norm(close_form_2 - analytic_sol);
error_close_3 = norm(close_form_3 - analytic_sol);
fprintf('Error_close_2: %.12f \n', error_close_2);
fprintf('Error_close_3: %.12f \n', error_close_3);
```

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.

$$L_{N,k}(x) = \prod_{\substack{i=0 \\ i \neq k}}^{N} \frac{x - x_i}{x_k - x_i}$$

For this assignment, you are given **three points x0, x1, x2**. You are to generate the **3 quadratic basis functions for $L_{2,0}, L_{2,1}, L_{2,2}$** 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 $L_{2,0}, L_{2,1}, L_{2,2}$ based on the x0, x1, x2 values provided (note you should have **3 curves for the 3 polynomials**):

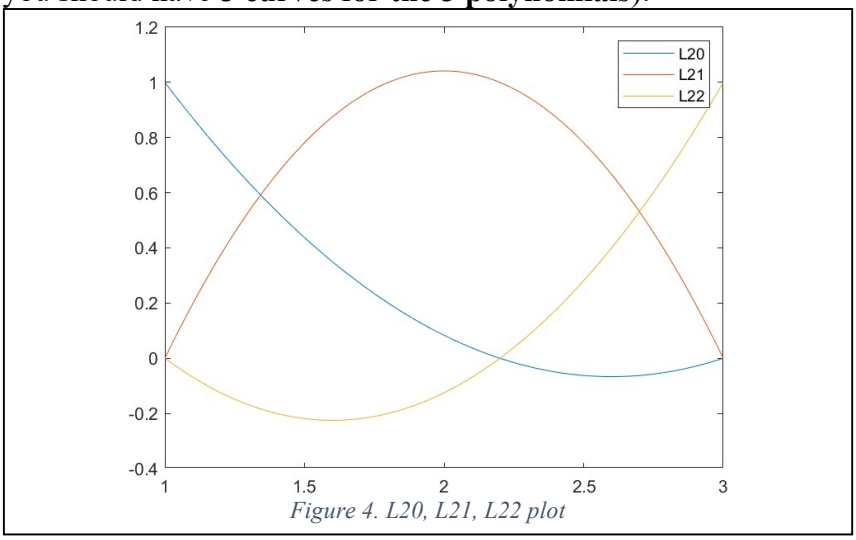


Figure 4. L20, L21, L22 plot

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 = 1.75
L20(X)= 0.234375
L21(X)= 0.976562
L22(X)= -0.210938
F(X)= 5.718750

For X = 2.0
L20(X)=0.083333
L21(X)=1.041667
L22(X)=-0.125000
F(X)= 6.000000

For X = 2.5
L20(X)= -0.062500
L21(X)= 0.781250
L22(X)= 0.281250
F(X)= 5.625000

For X = 2.2
L20(X)= -0.000000
L21(X)= 1.000000
L22(X)= 0.000000
F(X)= 6.000000

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.

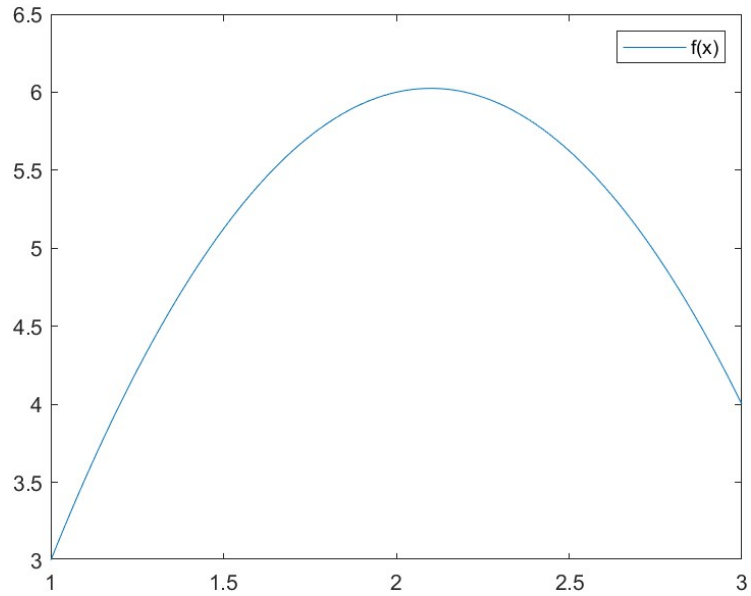


Figure 5. f(x) interpolated.

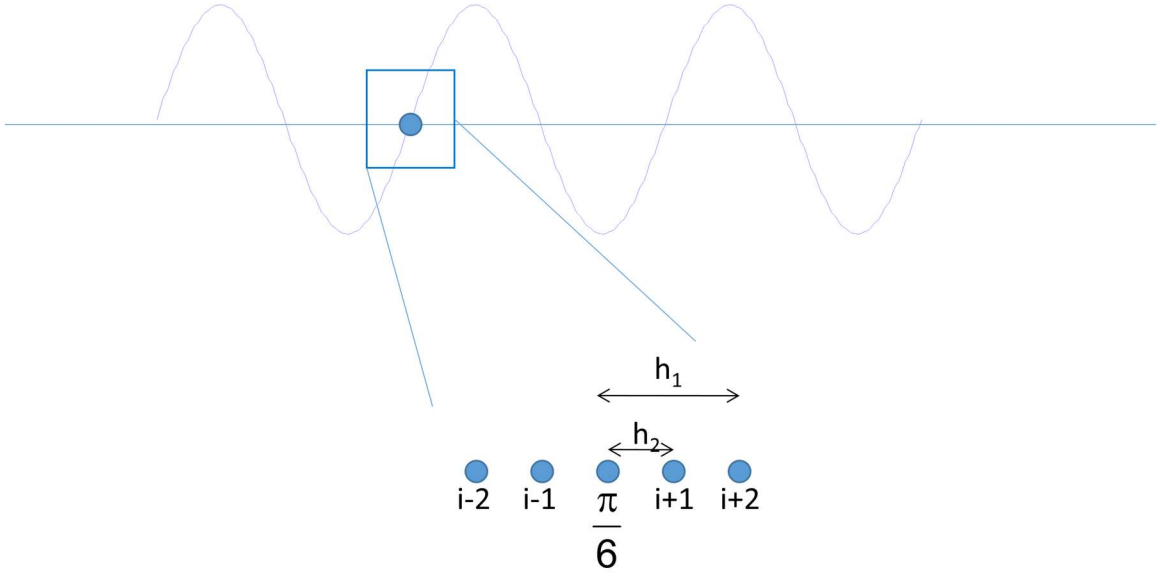
d. Place your MATLAB code in the box below as to how you completed this problem.

```
% -- plot_lagrange_quadratic.m


x = [1.0, 2.2, 3.0];
yvals = [3.0,6.0,4.0];
xi = 1.1;
lagrange_quadratic(x, yvals,xi);

xi = 1.75;
lagrange_quadratic(x, yvals,xi);

xi = 2.0;
lagrange_quadratic(x, yvals,xi);

xi = 2.2;
lagrange_quadratic(x, yvals,xi);

xi = 2.5;
lagrange_quadratic(x, yvals,xi);
```

```
% -- lagrange_quadratic.m

function [xr,L20,L21,L22]=lagrange_quadratic(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
% ---------------------- Part a ------------------------
figure(1)
plot(xr, L20)
hold on
plot(xr,L21)
hold on
plot(xr,L22)
hold off
legend('L20','L21','L22')

% ---------------------- Part b ------------------------
% Now let's evaluate a sample interpolated value using the quadratic
% Lagrange basis.  Now you will evaluate the 3 basis functions at
% a specific value x, your 'xi' input above.  You will calculate the
% interpolated value of your function at that specific value.
L20v=(xi-x1)*(xi-x2)/((-h01)*(-h02));
L21v=(xi-x0)*(xi-x2)/((h01)*(-h12));
L22v=(xi-x0)*(xi-x1)/((h02)*(h12));

% Here is where you will calculate the interpolated value of your function
% at the specific value of x, 'xi'
yv = yvals(1)*L20v + yvals(2)*L21v + yvals(3)*L22v;
```

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

% ---------------------- Part c ------------------------

for i=1:nn
    y_interpolated(i) = yvals(1)*L20(i) + yvals(2)*L21(i) + yvals(3)*L22(i);
end
figure(2)
plot(xr, y_interpolated)
legend('f(x)')
end
```

Problem 3: Numerical Differentiation. In class, we talked about how to generate a numerical derivative and the power of sampling (h) 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	0.866007880595	0.866024306170	0.866025335146	0.000017523189	0.000001097614	0.000000068638
Back	0.866025403784	0.873569309894	0.867862752075	0.866477904831	0.007543906110	0.001837348291	0.000452501047
Fwd	0.866025403784	0.823279179126	0.845510468697	0.855986618818	0.042746224658	0.020514935087	0.010038784966

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). This should produce 3 lines (one with error based Cntr, one on Back, and one on Fwd). Please provide this as one graph with the overlay of 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 and the nature of truncation error?** How does the shape of the plot correlate with your understanding of the error associated with the FD approximation?

- Looking at the plot, what trends do you notice and how does that correlate with your understanding of the difference expressions given and the nature of truncation error**?**
  - **Since Truncation Error has $O(h^n)$ behavior where *n* is the Order of Accuracy**, we see a negative slope in the plot of Error against (1/h). Higher ***n*** results in steeper slope. The downward trend in the loglog plot shows with each h value half-ed from $\pi/20$, $\pi/40$, to $\pi/80$, we ultimately have the error decrease linearly.
- How does the shape of the plot correlate with your understanding of the error associated with the FD approximation?
  - **Higher Order of Accuracy → 1) Lower error value & 2) Steeper slope:**
    - We can see that Center Difference (4^{th} order accuracy) has the lowest error values and steepest slope. Even First Central Difference exhibits one higher order of accuracy for the same number of function evaluations compared to Back/Forward Difference.
    - Backward difference (2^{nd} order accuracy) has second lowest error values and moderately-steep slope.
    - First Order Forward difference has the highest error value and least steep slope.

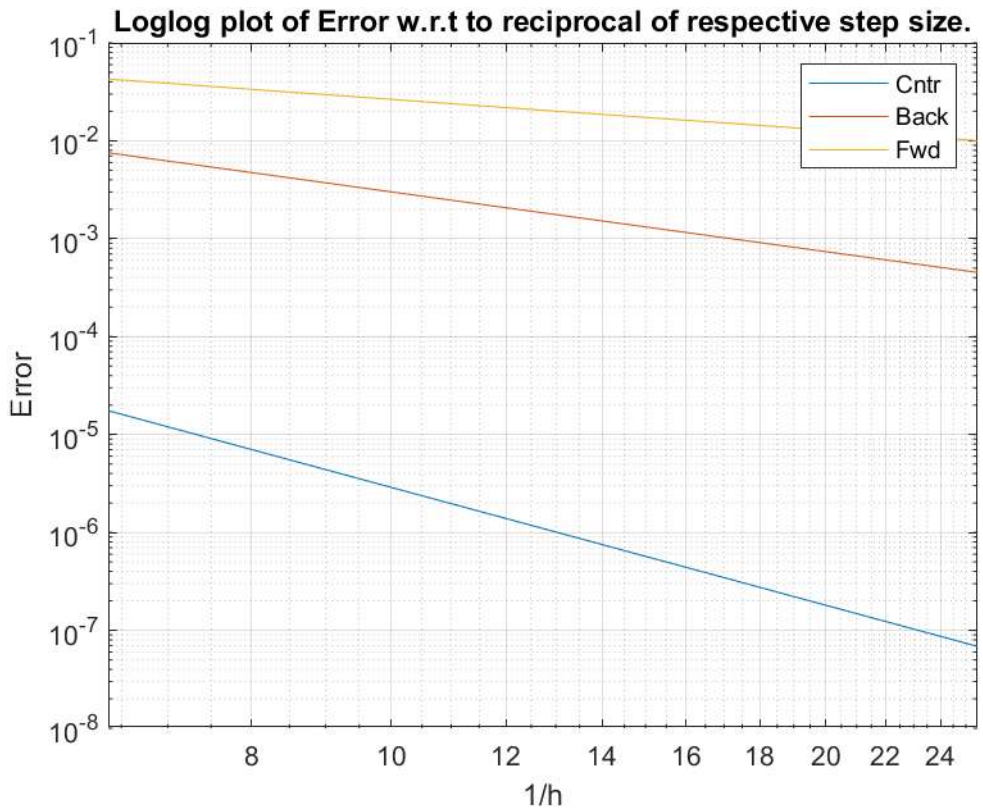


Figure 6. Center, Backward, Forward Error w.r.t (1/h)

Mfile:

```
format long
center = pi/6;
center_val = sin(pi/6);
analytical_deriv = 0.866025403784;
h1 = pi/20;
h2 = pi/40;
h3 = pi/80;

center_err = zeros(1,3);
backward_err = zeros(1,3);
forward_err = zeros(1,3);
[center_err(1), backward_err(1), forward_err(1)] = get_u_from_h(center, h1,
analytical_deriv);
[center_err(2), backward_err(2), forward_err(2)] = get_u_from_h(center, h2,
analytical_deriv);
[center_err(3), backward_err(3), forward_err(3)] = get_u_from_h(center, h3,
analytical_deriv);

h = 1./[h1,h2,h3];
figure(1)
loglog(h, center_err,...
       h, backward_err,...
       h, forward_err)
legend('Cntr', 'Back', 'Fwd')
grid on
ylabel('Error')
xlabel('1/h')
title('Loglog plot of Error w.r.t to reciprocal of respective step size.')

function [center_error, backward_error, forward_error] =  get_u_from_h(center, h,
analytical_deriv)

u = sin(center-2*h : h : center+2*h);

center_diff = (u(1) - 8*u(2) + 8*u(4) - u(5)) / (12*h);
backward_diff = (u(1) - 4*u(2) + 3*u(3)) / (2*h);
forward_diff = (u(4) - u(3))/h;

center_error = abs(center_diff - analytical_deriv);
backward_error = abs(backward_diff - analytical_deriv);
forward_error = abs(forward_diff - analytical_deriv);
fprintf('---------Values for derivatives at %.12f with %.12f--------\n', sin(center),
h);
fprintf('Center diff   : %.12f, Error %.12f \n', center_diff, center_error);
fprintf('Backward diff : %.12f, Error %.12f \n', backward_diff, backward_error);
fprintf('Forward diff  : %.12f, Error %.12f \n', forward_diff, forward_error);


end
```