# Comparing Stability, Accuracy, and Computational Efficiency of Three-Step Adams Bashforth Predictor-Corrector Methods Using the Duffing Equation

Emmanuel Lyngberg, Teerapong Poltue

April 26, 2024

## I. Problem Statement

Applying numerical methods for integration of typical Initial Value Problems (IVP) is a balance between maximizing performance and efficiency against numerical stability. There is no general purpose choice for solving IVP as these decisions are very much dependent on the characteristics of the ODE (or system of ODEs) of interest. Three very popular options are the explicit, linear, multi-step Adams-Bashforth (AB) methods, their implicit counter-part the Adams-Moulton (AM) methods, and the implicit, linear, multi-step Backward Differentiation Formulas (BDF).

$$AB-3: y_{n+3} = y_{n+2} + h(\frac{23}{12}f(t_{n+2}, y_{n+2}) - \frac{16}{12}f(t_{n+1}, y_{n+1}) + \frac{5}{12}f(t_n, y_n)) \tag{1}$$

$$AM-3: y_{n+3} = y_{n+2} + h(\frac{9}{24}f(t_{n+3}, y_{n+3}) + \frac{19}{24}f(t_{n+2}, y_{n+2}) - \frac{5}{24}f(t_{n+1}, y_{n+1}) + \frac{1}{24}f(t_n, y_n)) \tag{2}$$

$$BDF-3: y_{n+3} - \frac{18}{11}y_{n+2} + \frac{9}{11}y_{n+1} - \frac{2}{11}y_n = \frac{6}{11}hf(t_{n+3}, y_{n+3}) \tag{3}$$

Notice that the AB and AM method are very similar in construction, with the main difference being the implicit $f(t_{n+3}, y_{n+3})$ evaluation in the AM-3 formula. This contrasts with the BDF formula where there is only one function evaluation. When comparing these three methods, their stability is one aspect that can be directly compared.

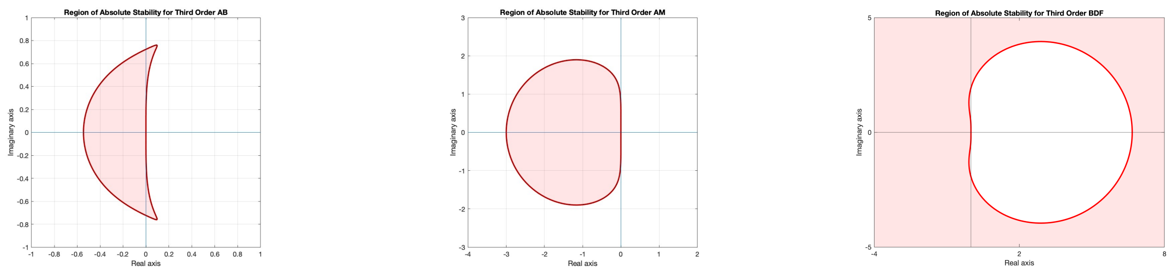

Figure 1: The Absolute Stability regions for AB-3 (left), AM-3 (center), and BDF-3 (right) where the region of stability is shaded in red, in the complex plane.

Visualizing their absolute stability in the complex plane, we notice that BDF-3 has a much larger area of absolute stability when compared with AB-3 and AM-3, but it is not A-stable. Unsurprisingly, the AM-3 method does have a larger region of stability compared with AB-3, but not as dramatically as BDF-3 given that they are very close in form. This comparison gives rise to some questions regarding their stability and performance when coupled in predictor-corrector methods. Thus, we consider the AB-AM and AB-BDF predictor corrector methods of the third order and compare their stability and performance against the Duffing equation to better understand how the characteristics of the stability between AM and BDF change when coupled with AB.

In comparing the computational efficiency and overall accuracy of these two methods, it is important to choose a test equation that can provide sufficient irregularities and parameter tuning to elucidate any glaring differences between the method's performance. For this exploration, the Duffing equation is a great example of a non-linear second-order differential equation that can exhibit chaotic behavior, whilst still allowing for 2-D visualization.

$$x'' + \delta x' + \alpha x + \beta x^3 = \gamma F(t) \tag{4}$$

Note that the Duffing equation can take many forms based on particular use cases, but for this exploration the forcing term will be with respect to time $t$, and more specifically it will be $F(t) = cos(t)$ to provide oscillatory behavior. Notice that if $\beta = 0$, the ODE simplifies to a forced-mass spring system. This simplified version has been applied to many phenomena, namely simple harmonic motion, damped motion, pendulum tuned mass damper, LRC circuit, etc., thanks to the coefficients $\alpha, \beta, \delta, \gamma$ that govern the system. For instance, in the mechanical oscillation system $\delta$ is the viscous damping coefficient, $\alpha$ and $\beta$ are stiffness coefficients, and $\gamma$ governs the strength of the oscillating force applied to the system as a function of time. This equation exhibits high dynamics including chaos, bifurcations, and multiple stable solutions, making this a liable model equation for studying the performance of different numerical methods. Note that in this particular exploration we set $\alpha = 0$, as it is a parameter that does not vary chaos and stiffness as much as other parameters, and thus we omit it to reduce the number of parameters involved in comparing the two predictor-corrector methods.

# II. Computation

We now present the computational analysis required to derive the stability of the two predictor-corrector methods. Beginning with 0-stability, we let all function evaluations equal to zero, and replace each $y_{n+k}$ with $\xi^k$ and determine the roots of the characteristic equation we achieve from this. For AB-AM this is simply

$$y_{n+3} = y_{n+2} \rightarrow \xi^3 = \xi^2 \rightarrow \xi^2(\xi - 1) = 0 \rightarrow \xi = 0, \xi = 1$$

For AB-BDF, we have

$$\begin{aligned} y_{n+3} - \frac{8}{11}y_{n+2} + \frac{9}{11}y_{n+1} - \frac{2}{11} &= 0 \\ \xi^3 - \frac{8}{11}\xi^2 + \frac{9}{11}\xi - \frac{2}{11} &= 0 \\ \xi = 1,\ \ \xi = 7/22 \pm (i\sqrt{39})/22 & \end{aligned}$$

Thus, it is clear that in both cases these methods are 0-stable and in fact strongly stable.

We now consider the absolute stability for each method. In this instance we consider the function evaluations, and use the test equation $f(y) = y' = \lambda y$ to create the stability region in the complex plane. For the AB-AM method, we plug in the function evaluation of AB into the implicit term of AM, achieving the equations below.

$$y_{n+3} = y_{n+2} + \frac{h}{24}(9\lambda(y_{n+2} + \frac{h}{12}(23f(t_{n+2}, y_{n+2}) - 16f(t_{n+1}, y_{n+1}) + 5f(t_n, y_n))) + 19f(t_{n+2}, y_{n+2}) - 5f(t_{n+1}, y_{n+1}) + f(t_n, y_n))$$

$$y_{n+3} = y_{n+2} + \frac{h}{24}(9\lambda(y_{n+2} + \frac{h}{12}(23\lambda y_{n+2} - 16\lambda y_{n+1} + 5\lambda y_n)) + 19\lambda y_{n+2} - 5\lambda y_{n+1} + \lambda y_n$$

In order to visualize this in the complex plane we now let $z = \lambda h$ and manipulate the equation such that it becomes a quadratic one with respect to $z$. We then let $\xi = e^{i\theta}$ and iterate $\theta$ across 0 to $2\pi$, to capture the entire plane. This allows us to reduce this equation involving two unknowns into an equation involving just one where we can use a root finding solver of choice to then solve the equation for the roots of $z$ for that particular value of $\xi$. The iteration across $\theta$ for many points allows us to then construct the boundary of the stability region. The manipulated equation becomes the following quadratic equation with respect to $z$:

$$0 = (\frac{23}{32}\xi^2 - \frac{\xi}{2} + \frac{5}{32})z^2 - (\frac{28}{24}\xi^2 - \frac{5}{24}\xi + \frac{1}{24})z + \xi^2 - \xi^3$$

Repeating this process for AB-BDF, we plug in AB for the implicit function evaluation in BDF. We then let $z = \lambda h$ and iterate across $\theta$ for $\xi$ to get a quadratic equation with respect to $z$ that we can solve for many iterations. These equations are given below:

$$y_{n+3} = \frac{6h}{11}\lambda(y_{n+2} + \frac{h}{12}(23f(t_{n+2}, y_{n+2}) - 16f(t_{n+1}, y_{n+1}) + 5f(t_n, y_n)) + \frac{18}{11}y_{n+2} - \frac{9}{11}y_{n+1} + \frac{2}{11}y_n$$

$$0 = (-\frac{23}{22}\xi^2 + \frac{8}{11}\xi - \frac{5}{22})z^2 - (\frac{6}{11}\xi^2)z + (\xi^3 - \frac{18}{11}\xi^2 + \frac{9}{11}\xi - \frac{2}{11})$$

Implementing the iterative process outlined previously in MATLAB for both AB-AM and AB-BDF, we obtain the plots given in Figure 2. Note that the stability region for AB-AM is actually larger than that of AB-BDF, we just rescale the axes to fit the stability regions nicely within the confines of the plots. Interestingly, the stability regions for each appear to be hybrids of their respective sub-methods when considering the original shapes and "tail" features exhibited in the left plot of Figure 1 for AB.

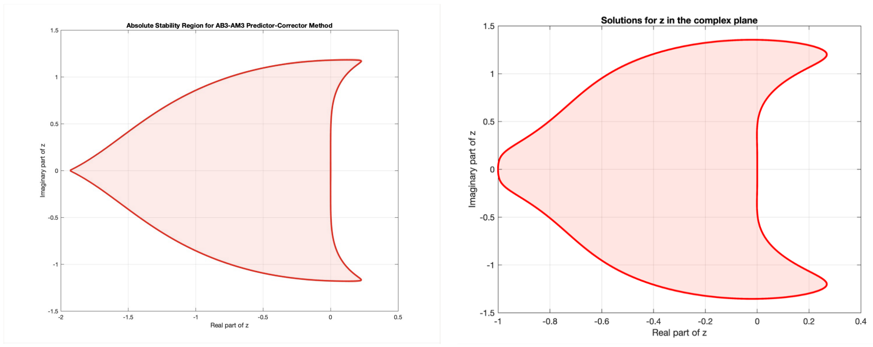

Figure 2: The Absolute Stability regions for AB-AM (left) and AB-BDF (right) in the complex plane, highlighted in red.

## III. Results

To evaluate the performance of the proposed predictor-corrector methods, AB-BDF and AB-AM, a parametric study is compared to the benchmark, ODE45, under five different cases, as shown in Table 1, with the step size of 0.1, 0.01, and 0.001 using the Duffing equation as a model equation due to its stiff behavior and chaotic. We also provide the visual results of the fives cases applied to both predictor-corrector methods. Note that we choose cases one and two to compare chaos that is clear in case one but is not as apparent in case two since the non-linear parameter $\beta$ is reduced. This was chosen with the hopes of ellucidating any performance differences from chaos. In cases three and four there is clear damping added to the system to ellucidate performance differences from stiffness. In case five we have a unique instance where there is regular periodic activity that is bisected by the third period having a chaotic moment (focus on the black ODE45 curve).

For a quantitative comparison across each case, root mean square error criteria, given by

$$RMSE = \sqrt{\sum_{i=1}^{N} \frac{(y_{pred-cor} - y_{ODE45})}{N}}$$

is utilized in discussing its performance.

Beginning the discussion and interpretation of our results, Figure 4 shows that with smaller mesh size, (step size decreased going left to right for each bar with respect to each case) higher accuracy was captured. Especially when the model equation is stiffer (note the greater damping coefficient $\delta$ in case 3), both predictor-corrector approaches could capture more precise solution. The important take-away here is that even in cases with odd chaos or stiffness, as is clear from the visuals of Figure 3, both methods are able to improve their performance with smaller step sizes across the board. This is promising, as there are always going to be concerns about stability given that these predictor-corrector methods are explicit, with specific stability regions as given in Figure 2. However, there is no obvious trend regarding AB-BDF outperforming AB-AM or vice versa. Therefore, it is worth further study on the computational cost required for each scheme to reach the same tolerance level. In this particular exploration we choose below 50 mm. According to Figure 5, as expected every predictor-corrector scheme shows bigger mesh size require for the same tolerance compares to those implicit schemes (AM and BDF). Since the predictor-corrector schemes are composed of a three-step explicit scheme for the prediction step and thus remove the implicit nature in the evaluation step, the predictor-corrector requires a larger mesh size, most likely because of the reduced stability compared to BDF and AM on their own, as expected. Between the two predictor-corrector methods we only see marginal difference in step size required, with AB-BDF slightly outperforming AB-AM. Interestingly, both BDF and AM have truncation error that scales $O(h^3)$, but the AB-AM method generally requires bigger mesh size in the test cases given in this study.

These results culminate in two major findings. The first is that despite BDF having a large stability region on its own, the coupled AB-BDF did not have greater stability compared to AB-AM as is clear from Figure 2, despite AM having a much smaller stability region compared to BDF. This also translated to the accuracy and computational efficiency results, as we can see from Figures 4 and 5 that even when we attempted to exhaust different variations of chaos and stiffness, there is no clear difference in performance between the AB-AM and AB-BDF methods. The second major result is a derivative of the first, which is that we realize that the use of particular predictor-corrector methods has to be carefully considered within the context of the problem being solved. With smaller time steps, the predictor-corrector methods outperform their individual explicit and implicit sub-methods particularly in more stiff or chaotic instances because of the added complexity in the method. However, this comes at a computational cost where there are a much greater number of function evaluations at each step, which might be undesireable in particular circumstances (once again reaffirming the second lesson learned). Despite our best efforts to choose parameter combinations for the Duffing Equation that would hopefully ellucidate differences between AB-AM and AB-BDF, we were unable to draw any conclusions regarding relative performance from these results. If repeated, the use of a more complex, possibly higher order, ODE compared to the Duffing equation might prove more fruitful in magnifying any stability and performance differences between these two predictor-corrector methods.

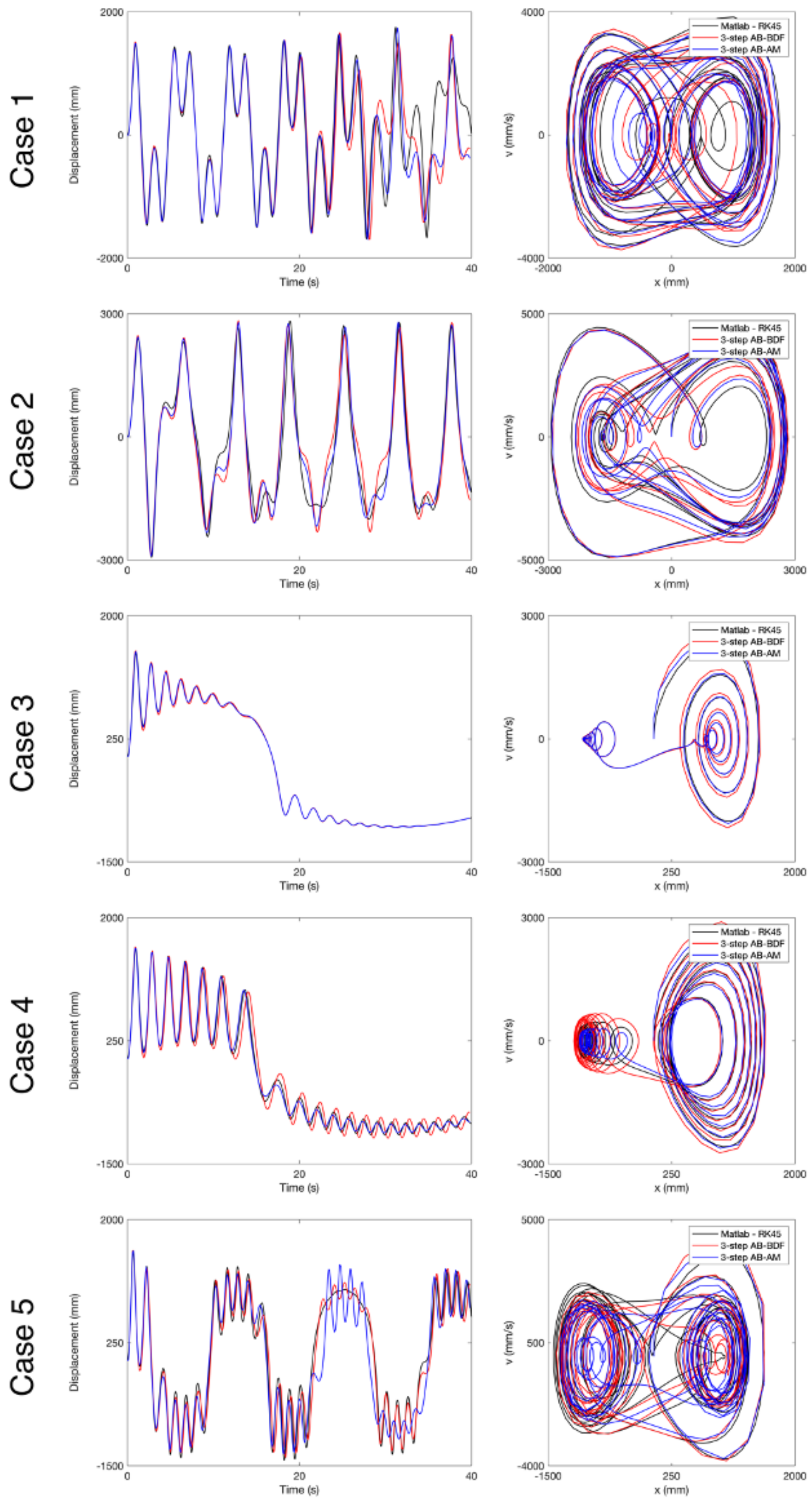

Figure 3: The five plots of AB-AM, AB-BDF, and ODE45 for each combination of parameters within the duffing equation.

| **Parameters** | **Case 1** | **Case 2** | **Case 3** | **Case 4** | **Case 5** |
|---|---|---|---|---|---|
| $\boldsymbol{\alpha}$ | 0 | 0 | 0 | 0 | 0 |
| $\boldsymbol{\beta}$ | 5 | 1 | 5 | 5 | 10 |
| $\boldsymbol{\gamma}$ | 5 | 5 | 5 | 5 | 10 |
| $\boldsymbol{\delta}$ | 0.1 | 0.1 | 0.5 | 0.1 | 0.1 |
| $\boldsymbol{\omega}$ | 1 | 1 | 1 | 0.1 | 0.1 |

Table 1: A table consisting of the fives cases explored, with all of the parameter combinations given for the Duffing Equation. Note that certain values are highlighted in red to show the key differences between cases when visually comparing in Figure 3.

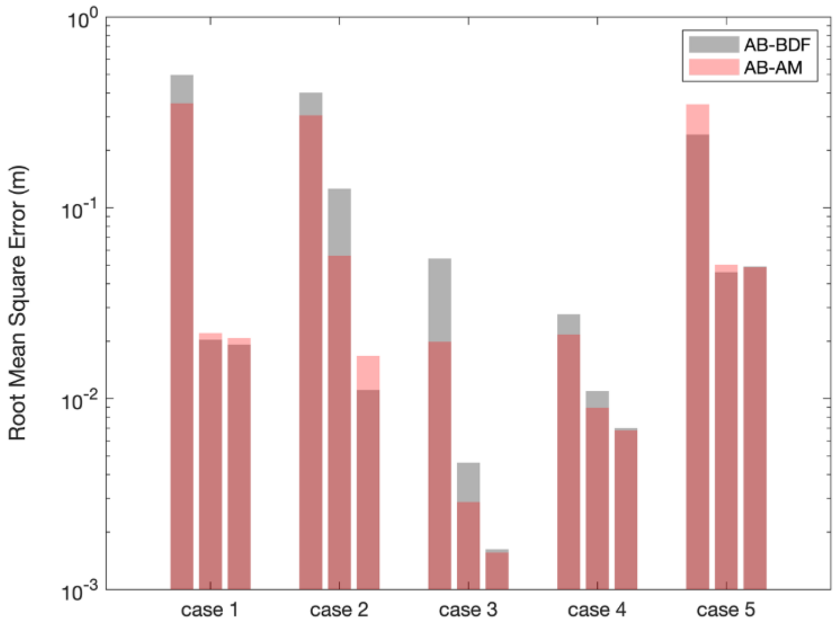

Figure 4: The Root Mean Square Error (RMSE) of each parameter combination case, where each bar represents step size of 0.1, 0.01, and 0.001 from left to right.

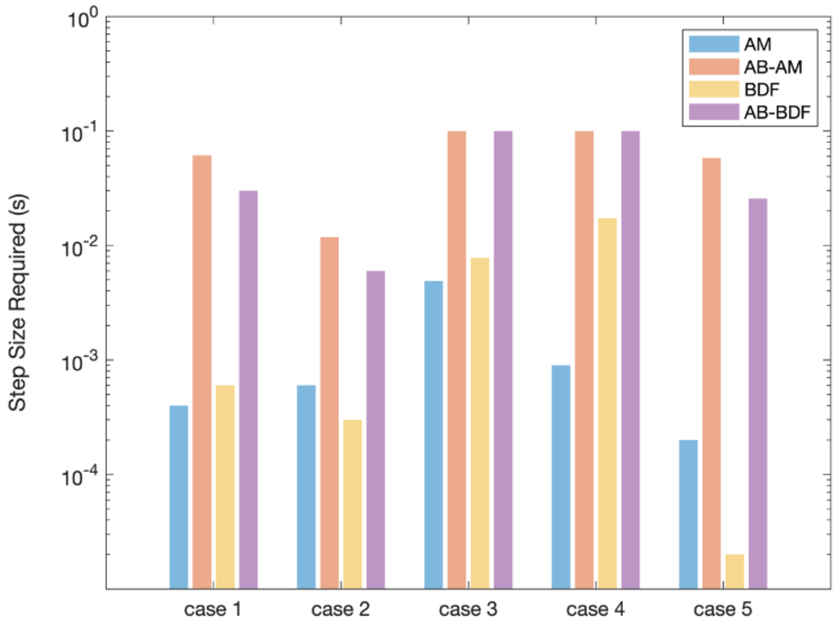

Figure 5: Step size required to achieve tolerance of 50 mm. of each numerical schemes compares to benchmark, ODE45.

## IV. Team Member Participation

The problem statement was drafted by Teerapong and finalized by Emmanuel. Coding requirements were split evenly among the two group members. Teerapong was responsible for implementing AB-BDF whilst Emmanuel was responsible for doing the same with AM. Additionally, Teerapong created the figures for the results section while Emmanuel created the stability plots. Emmanuel wrote the computations section and both worked on the results section. Given his extensive background using LaTeX, Emmanuel created and organized the final report document.

## References

[1] M. L. Abell and J. P. Braselton, "Applications of Higher Order Differential Equations," in Introductory Differential Equations, Elsevier, 2018, pp. 229–275. doi: 10.1016/B978-0-12-814948-5.00005-7.

[2] C. Anyigor and J. Afiukwa, "Application of matlab ordinary differential equation function solver (ode45) in modelling and simulation of batch reaction kinetics," American Journal of Scientific and Industrial Research, vol. 4, no. 3, pp. 285–287, Jun. 2013, doi: 10.5251/ajsir.2013.4.3.285.287.

[3] C. F. Curtiss and J. O. Hirschfelder, "Integration of Stiff Equations," Proceedings of the National Academy of Sciences, vol. 38, no. 3, pp. 235–243, Mar. 1952, doi: 10.1073/pnas.38.3.235.

[4] Ghrist, Michelle L., et al. "Stability Ordinates of Adams Predictor-Corrector Methods." BIT Numerical Mathematics, vol. 55, no. 3, 17 Dec. 2014, pp.733–750, www.colorado.edu/amath/sites/default/files/attached-files/2014_gfr_ab_pc_stability_bit_submitted.pdf, https://doi.org/10.1007/s10543-014-0528-7. Accessed 16 Apr. 2024.

[5] "ODEs: Chaos." Introduction to Differential Equations, 2023, web.uvic.ca/ tbazett/diffyqs.html.

## Appendix (Code)

```
% This code is for plotting stability regions
% Define the range of theta values
theta_values = linspace(0, 2*pi, 1000);

% Initialize array to store solutions for z
z_solutions = [];

% Loop over each value of theta
for theta = theta_values
    % Define e^(i*theta), e^(2i*theta), and e^(3i*theta)
    e_i_theta = exp(1i*theta);
    e_2i_theta = exp(2*1i*theta);
    e_3i_theta = exp(3*1i*theta);

    % Define the coefficients of the polynomial equation in z
    coeff = [23*e_2i_theta/22-8*e_i_theta/11+5/22, 6*e_2i_theta/11,
                -e_3i_theta+18*e_2i_theta/11-9*e_i_theta/11+2/11];

    % Find the roots of the polynomial equation
    roots_all = roots(coeff);

    % Append the roots to the solutions array
    z_solutions = [z_solutions; roots_all];
end

% Plot the values of z in the complex plane
figure;
plot(real(z_solutions(:)), imag(z_solutions(:)), '.', 'Color','red');
xlabel('Real part of z');
ylabel('Imaginary part of z');
title('Solutions for z in the complex plane');
hold on;
% h=fill(real(z_solutions(:)), imag(z_solutions(:)), 'r');
% set(h,'facealpha',.1)
hold off;

% This code is for running the predictor-corrector method AB-AM
function [t, x, v] = BDF(fn, interval, initial_condition, dt)
    % Expected user input
    % fn: input ODE function - dy/dt = fn(y, t)
    % interval: solving intercal [a, b]
    % y0: initial condition for at time 0 - y(0)
    % y1: initial condition for at time 0+h - y(1)
    % dt: step size in numerical solution

    N = round((interval(2) - interval(1)) / dt);

    % initialize arrays
    t = zeros(N+1, 1);
    x = zeros(N+1, 1);
    v = zeros(N+1, 1);
    % initial condition
    t(1) = interval(1);
    x(1) = initial_condition(1);
    v(1) = initial_condition(2);

    % numerical step
    for i = 1:N-2
        % using FE as the initial guess for BDF
        t(i+1) = t(i) + dt;
        f0 = feval(fn, t(i), [x(i), v(i)]);
        x(i+1) = x(i) + dt*f0(1);
        v(i+1) = v(i) + dt*f0(2);

        % 1-step BDF
        f1 = feval(fn, t(i+1), [x(i+1), v(i+1)]);
        x(i+1) = x(i) + dt * f1(1);
        v(i+1) = v(i) + dt * f1(2);

        % 2-step BDF
        f2 = feval(fn, t(i+2), [x(i+2), v(i+2)]);
        x(i+2) = (4/3)*x(i+1) - (1/3)*x(i) + (2/3)*dt*f2(1);
        v(i+2) = (4/3)*v(i+1) - (1/3)*v(i) + (2/3)*dt*f2(2);

        % 3-step BDF
        f3 = feval(fn, t(i+3), [x(i+3), v(i+3)]);
        x(i+3) = (18/11)*x(i+2) - (9/11)*x(i+1) + (2/11)*x(i) + (6/11)*dt*f3(1);
        v(i+3) = (18/11)*v(i+2) - (9/11)*v(i+1) + (2/11)*v(i) + (6/11)*dt*f3(2);

    end
end

%%

function [t, x, v] = predcor_ABAM(fn, interval, initial_condition, dt)

    N = round((interval(2) - interval(1)) / dt);

    % initialize arrays
    t = linspace(interval(1), interval(2), N+1);
    x = zeros(N+1, 1);
    v = zeros(N+1, 1);



    % initial conditions
    x(1) = initial_condition(1);
    v(1) = initial_condition(2);

    % using 1-step AB to get second position value to use in 3-step method
    t(2) = t(1) + dt;
    f0 = feval(fn, t(1), [x(1), v(1)]);
    x(2) = x(1) + dt*f0(1);
    v(2) = v(1) + dt*f0(2);
    % using 2-step AB to get third position value to use in 3-step method
    f1 = feval(fn, t(2), [x(2),v(2)]);
    x(3) = x(2) + dt*(3/2)*f1(1) - dt*(1/2)*dt*f0(1);
    v(3) = v(2) + dt*(3/2)*f1(2) - dt*(1/2)*dt*f0(2);

    % numerical step
    for i = 1:N-2
        % predictor step (AB-3)
        f_0 = feval(fn, t(i), [x(i),v(i)]);
        f_1 = feval(fn, t(i+1), [x(i+1),v(i+1)]);
        f_2 = feval(fn, t(i+2), [x(i+2), v(i+2)]);

        x_3 = x(i+2) + dt*(23/12)*f_2(1) - dt*(16/12)*f_1(1) + dt*(5/12)*f_0(1);
        v_3 = v(i+2) + dt*(23/12)*f_2(2) - dt*(16/12)*f_1(2) + dt*(5/12)*f_0(2);

        % now we use x_3 and v_3 for AM-3 to calculate f_3
        f_3 = feval(fn, t(i+3), [x_3,v_3]);

        % corrector step (AM-3)
        x(i+3) = x(i+2) + (dt/24)*(9*f_3(1) + 19*f_2(1) - 5*f_1(1) + f_0(1));
        v(i+3) = v(i+2) + (dt/24)*(9*f_3(2) + 19*f_2(2) - 5*f_1(2) + f_0(2));
    end

end

% This code is for calculating convergence times as well as the predictor-corrector method for AB-BDF
alpha =  [0 0 0 0 0];
beta =   [5 1 5 0.5 10];
gamma =  [5 5 5 0.5 10];
delta =  [0.1 0.1 0.5 0.1 0.1];
omega =  [1 1 1 0.1 0.1];

for i = 5

eval_case = i;
interval = [0, 40];
initial_condition = [0, 0]; % x_ddot(0), x_dot(0)
tolerance_lim = 50e-3;
dt = 0.1;

tol = 1e3;
counter = 1;
while tol >= tolerance_lim

% Matlab RK(4,5)
fn = @(t,x)[x(2); -delta(eval_case)*x(2) - alpha(eval_case)*x(1) - beta(eval_case)*x(1)^3 + gamma(eval_c
[t, x] = ode45(fn, interval(1):dt:interval(2), initial_condition); %(1):dt:interval(2)

% AB-BDF
[t_BDF, x_BDF, ~] = ABBDF(fn, interval, initial_condition, dt);


% AB-AM
[t_AM, x_AM, ~] = ABAM(fn, interval, initial_condition, dt);
```

```
tol = mean(abs(x(:,1)-x_BDF(1:size(x(:,1)))));
diff(counter) = tol;
dt = dt-0.0001;
counter = counter + 1;

end


% plotting
figure(1);
x0 = 10;
y0 = 10;
width = 1000;
height = 350;
set(gcf, 'position', [x0 y0 width height]);

fig1 = figure(1);
h = subplot(1, 5, 1:3);
pos = get(h(1),'Position');
set(h(1),'Position',[pos(1) pos(2) pos(3)-0.05 pos(4)]);
hold on
plot(t, x(:, 1),'k-', 'linewidth', 1)
plot(t_BDF(1:end-2), x_BDF(1:end-2, 1), 'r-', 'linewidth', 1)
plot(t_AM(1:end-2), x_AM(1:end-2, 1), 'b-', 'linewidth', 1)
xlabel('Time (s)')
ylabel({'Displacement (mm)',''})
y = get(gca, 'ylim');
xticks(linspace(interval(1), interval(2), 3));
yticks(linspace(y(1), y(2), 3));
ytickformat('%.0f');
set(gca, 'fontsize', 12);
box on

subplot(1, 5, 4:5);
hold on
plot(x(:, 1), x(:, 2), 'k-', 'linewidth', 1)
plot(x_BDF(1:end-1), v_BDF(1:end-1), 'r-', 'linewidth', 1)
plot(x_AM(1:end-1), v_AM(1:end-1), 'b-', 'linewidth', 1)
xlabel('x (mm)')
ylabel({'v (mm/s)'})
x = get(gca, 'xlim');
y = get(gca, 'ylim');
xticks(linspace(x(1), x(2), 3));
yticks(linspace(y(1), y(2), 3));
xtickformat('%.0f');
set(gca, 'fontsize', 12);
legend('Matlab - RK45', '3-step AB-BDF', '3-step AB-AM')
box on

end

function [t, x, v] = ABBDF(fn, interval, initial_condition, dt)
    % Expected user input
    % fn: input ODE function - dy/dt = fn(y, t)
    % interval: solving intercal [a, b]
    % y0: initial condition for at time 0 - y(0)
    % y1: initial condition for at time 0+h - y(1)
    % dt: step size in numerical solution

    N = round((interval(2) - interval(1)) / dt);

    % initialize arrays
    t = linspace(interval(1), interval(2), N+1);
    x = zeros(N+1, 1);
    v = zeros(N+1, 1);

    % initial condition
    t(1) = interval(1);
    x(1) = initial_condition(1);
    v(1) = initial_condition(2);

    % using 1-step AB to get second position value to use in 3-step method
    t(2) = t(1) + dt;
    f0 = feval(fn, t(1), [x(1), v(1)]);
    x(2) = x(1) + dt*f0(1);
    v(2) = v(1) + dt*f0(2);
    % using 2-step AB to get third position value to use in 3-step method
    f1 = feval(fn, t(2), [x(2),v(2)]);
    x(3) = x(2) + dt*(3/2)*f1(1) - dt*(1/2)*dt*f0(1);
    v(3) = v(2) + dt*(3/2)*f1(2) - dt*(1/2)*dt*f0(2);

    % numerical step
    for i = 1:N-2
        % % 1-step AB (predictor)
        % t(i+1) = t(i) + dt;
        % f0 = feval(fn, t(i), [x(i), v(i)]);
        % x(i+1) = x(i) + dt*f0(1);
        % v(i+1) = v(i) + dt*f0(2);
        %
        % % 1-step BDF (corrector)
        % f1 = feval(fn, t(i+1), [x(i+1), v(i+1)]);
        % x(i+1) = x(i) + dt * f1(1);
        % v(i+1) = v(i) + dt * f1(2);
        %
        % % 2-step AB (predictor)
        % f2 = feval(fn, t(i), [x(i), v(i)]);
        % f3 = feval(fn, t(i+1), [x(i+1), v(i+1)]);
        % x(i+2) = x(i+1) + dt*(3/2*f3(1) - 1/2*f2(1));
        % v(i+2) = v(i+1) + dt*(3/2*f3(2) - 1/2*f2(2));
        %
        % % 2-step BDF (corrector)
        % f4 = feval(fn, t(i+2), [x(i+2), v(i+2)]);
        % x(i+2) = (4/3)*x(i+1) - (1/3)*x(i) + (2/3)*dt*f4(1);
        % v(i+2) = (4/3)*v(i+1) - (1/3)*v(i) + (2/3)*dt*f4(2);

        % 3-step AB (predictor)
        f5 = feval(fn, t(i), [x(i), v(i)]);
        f6 = feval(fn, t(i+1), [x(i+1), v(i+1)]);
```

```
        f7 = feval(fn, t(i+2), [x(i+2), v(i+2)]);

        x(i+3) = x(i+2) + dt*(23/12*f7(1) - 16/12*f6(1) + 5/12*f5(1));
        v(i+3) = v(i+2) + dt*(23/12*f7(2) - 16/12*f6(2) + 5/12*f5(2));

        % 3-step BDF (corrector)
        f8 = feval(fn, t(i+3), [x(i+3), v(i+3)]);
        x(i+3) = (18/11)*x(i+2) - (9/11)*x(i+1) + (2/11)*x(i) + (6/11)*dt*f8(1);
        v(i+3) = (18/11)*v(i+2) - (9/11)*v(i+1) + (2/11)*v(i) + (6/11)*dt*f8(2);

    end
end

function [t, x, v] = ABAM(fn, interval, initial_condition, dt)

    N = round((interval(2) - interval(1)) / dt);

    % initialize arrays
    t = linspace(interval(1), interval(2), N+1);
    x = zeros(N+1, 1);
    v = zeros(N+1, 1);

    % initial conditions
    x(1) = initial_condition(1);
    v(1) = initial_condition(2);

    % using 1-step AB to get second position value to use in 3-step method
    t(2) = t(1) + dt;
    f0 = feval(fn, t(1), [x(1), v(1)]);
    x(2) = x(1) + dt*f0(1);
    v(2) = v(1) + dt*f0(2);
    % using 2-step AB to get third position value to use in 3-step method
    f1 = feval(fn, t(2), [x(2),v(2)]);
    x(3) = x(2) + dt*(3/2)*f1(1) - dt*(1/2)*dt*f0(1);
    v(3) = v(2) + dt*(3/2)*f1(2) - dt*(1/2)*dt*f0(2);

    % numerical step
    for i = 1:N-2
        % predictor step (AB-3)
        f_0 = feval(fn, t(i), [x(i),v(i)]);
        f_1 = feval(fn, t(i+1), [x(i+1),v(i+1)]);
        f_2 = feval(fn, t(i+2), [x(i+2), v(i+2)]);

        x_3 = x(i+2) + dt*(23/12)*f_2(1) - dt*(16/12)*f_1(1) + dt*(5/12)*f_0(1);
        v_3 = v(i+2) + dt*(23/12)*f_2(2) - dt*(16/12)*f_1(2) + dt*(5/12)*f_0(2);

        % now we use x_3 and v_3 for AM-3 to calculate f_3
        f_3 = feval(fn, t(i+3), [x_3,v_3]);

        % corrector step (AM-3)
        x(i+3) = x(i+2) + (dt/24)*(9*f_3(1) + 19*f_2(1) - 5*f_1(1) + f_0(1));
        v(i+3) = v(i+2) + (dt/24)*(9*f_3(2) + 19*f_2(2) - 5*f_1(2) + f_0(2));
    end
end
```

```
function [t, x, v] = AM(fn, interval, initial_condition, dt)

    N = round((interval(2) - interval(1)) / dt);

    % initialize arrays
    t = linspace(interval(1), interval(2), N+1);
    x = zeros(N+1, 1);
    v = zeros(N+1, 1);

    % initial conditions
    x(1) = initial_condition(1);
    v(1) = initial_condition(2);

    % numerical step
    for i = 1:N-2
        % predictor step (AB-3)
        t(i+1) = t(i) + dt;
        f0 = feval(fn, t(i), [x(i), v(i)]);
        x(i+1) = x(i) + dt*f0(1);
        v(i+1) = v(i) + dt*f0(2);

        % 1
        f_1 = feval(fn, t(i), [x(i),v(i)]);
        f_2 = feval(fn, t(i+1), [x(i+1),v(i+1)]);
        x(i+2) = x(i+1) + (dt/2)*(f_2(1) + f_1(1));
        v(i+2) = v(i+1) + (dt/2)*(f_2(2) + f_1(2));

        % 2
        f_3 = feval(fn, t(i), [x(i),v(i)]);
        f_4 = feval(fn, t(i+1), [x(i+1),v(i+1)]);
        f_5 = feval(fn, t(i+2), [x(i+2), v(i+2)]);
        x(i+2) = x(i+1) + (dt)*(5/12*f_5(1) + 8/12*f_4(1) - 1/12*f_3(1));
        v(i+2) = v(i+1) + (dt)*(5/12*f_5(2) + 8/12*f_4(2) - 1/12*f_3(2));

        % 3
        f_6 = feval(fn, t(i), [x(i),v(i)]);
        f_7 = feval(fn, t(i+1), [x(i+1),v(i+1)]);
        f_8 = feval(fn, t(i+2), [x(i+2), v(i+2)]);
        f_9 = feval(fn, t(i+3), [x(i+3), v(i+3)]);
        x(i+3) = x(i+2) + (dt/24)*(9*f_9(1) + 19*f_8(1) - 5*f_7(1) + f_6(1));
        v(i+3) = v(i+2) + (dt/24)*(9*f_9(2) + 19*f_8(2) - 5*f_7(2) + f_6(2));
    end
end


function [t, x, v] = BDF(fn, interval, initial_condition, dt)
    % Expected user input
    % fn: input ODE function - dy/dt = fn(y, t)
    % interval: solving intercal [a, b]
    % y0: initial condition for at time 0 - y(0)
    % y1: initial condition for at time 0+h - y(1)
    % dt: step size in numerical solution
    N = round((interval(2) - interval(1)) / dt);

    % initialize arrays
    t = zeros(N+1, 1);
    x = zeros(N+1, 1);
    v = zeros(N+1, 1);

    % initial condition
    t(1) = interval(1);
    x(1) = initial_condition(1);
    v(1) = initial_condition(2);

    % numerical step
    for i = 1:N-2
        % using FE as the initial guess for BDF
        t(i+1) = t(i) + dt;
        f0 = feval(fn, t(i), [x(i), v(i)]);
        x(i+1) = x(i) + dt*f0(1);
        v(i+1) = v(i) + dt*f0(2);

        % 1-step BDF
        f1 = feval(fn, t(i+1), [x(i+1), v(i+1)]);
        x(i+1) = x(i) + dt * f1(1);
        v(i+1) = v(i) + dt * f1(2);

        % 2-step BDF
        f2 = feval(fn, t(i+2), [x(i+2), v(i+2)]);
        x(i+2) = (4/3)*x(i+1) - (1/3)*x(i) + (2/3)*dt*f2(1);
        v(i+2) = (4/3)*v(i+1) - (1/3)*v(i) + (2/3)*dt*f2(2);

        % 3-step BDF
        f3 = feval(fn, t(i+3), [x(i+3), v(i+3)]);
        x(i+3) = (18/11)*x(i+2) - (9/11)*x(i+1) + (2/11)*x(i) + (6/11)*dt*f3(1);
        v(i+3) = (18/11)*v(i+2) - (9/11)*v(i+1) + (2/11)*v(i) + (6/11)*dt*f3(2);

    end
end
```