Math 269A Final Project

Building, Analyzing, and Stress-Testing ODE Solvers

Arjun Mittha

December 8, 2025

Contents

1 Introduction **3**

2 Stage 1: First-Order IVPs, Forward Euler, and Error Bounds **3**
- 2.1 Theory . . . 3
- 2.2 Algorithm . . . 4
- 2.3 Experiments . . . 4
- 2.4 Results and Discussion . . . 6

3 Stage 2: Higher-Order *Explicit* One-Step Methods and Validation **7**
- 3.1 Theory . . . 7
- 3.2 Algorithms . . . 7
- 3.3 Experiments . . . 7
- 3.4 Results and Discussion . . . 10

4 Stage 3: Stability and Timestep Selection via the Linear Model Problem **10**
- 4.1 Theory . . . 10
- 4.2 Experiments . . . 12
- 4.3 Results and Discussion . . . 14

5 Stage 4: Systems of ODEs: Time Series and Linear Constant Coefficients **15**
- 5.1 Theory . . . 15
- 5.2 Algorithms . . . 15
- 5.3 Experiments . . . 16
- 5.4 Results and Discussion . . . 20

6 Stage 5: Implicit One-Step Methods and Nonlinear Solvers **20**
- 6.1 Theory . . . 20
- 6.2 Algorithms . . . 21
- 6.3 Experiments . . . 21
- 6.4 Results and Discussion . . . 23

7 Stage 6: Stiffness in Practice: Detect, Quantify, and Choose Methods **24**
7.1 Theory . . . 24
7.2 Algorithms . . . 25
7.3 Experiments . . . 25
7.4 Results and Discussion . . . 27

8 Stage 7: Adaptive Stepsize Control **28**
8.1 Theory . . . 28
8.2 Algorithms . . . 29
8.3 Experiments . . . 29
8.4 Results and Discussion . . . 35

9 Stage 8: Linear Multistep Methods (AB/AM), Zero-Stability (Root Condition), and Starting Values **36**
9.1 Theory . . . 36
9.2 Algorithms . . . 37
9.3 Experiments . . . 37
9.4 Results and Discussion . . . 40

10 Synthesis and Lessons **40**

A Implementation Notes **41**

B Additional Figures **41**

1 Introduction

In order to represent the findings of Math 269A, I have composed this report and an adjacent codebase that numerically analyzes and solves ODEs. The work is organized into eight stages, each covering a unique and key conceptual portion of the course. Stages 1-3 introduce explicit methods, analyze local and global error, and study stability of numerical methods through both theory and problem-dependent factors. Stages 4-6 discuss stiffness and stability of explicit schemes. It also introduces implicit methods as a direct advantage in stiff cases and considers limitations as well. These stages also incorporate sufficient plots and diagrams to visualize key theory with respect to key examples relevant to course material. In stage 7, we explore adaptive step-sizing, primarily through the view of step-doubling and embedded Runge-Kutta pairs. Through valuable examples from previous sections, adaptive RK2 and the embedded RK23 is stress tested. We explore the work-precision tradeoff and efficiencies. For stage 8, we look at linear multistep methods (LMMs). Specifically with respect to AB2 and AM2, we explore the impact of LMMs and the zero stability condition. Through variable starters and a global and local order analysis, we understand the tradeoffs of LMMs and their respective computational costs and starter impact.

Across all stages, theory is paired with numerical experiments that reproduce predicted orders of accuracy, stability and stiffness analysis, and computational work requirements. The result is a complete study demonstrating not only how classical ODE solvers operate, but also how their performance depends on stiffness, smoothness, error tolerances, and overall method design. With these in mind, this report provides a comprehensive overview of Math269A ODE solver material and practical implementation.

2 Stage 1: First-Order IVPs, Forward Euler, and Error Bounds

2.1 Theory

We consider the scalar initial–value problem

$$y'(t) = f(t, y), \qquad y(t_0) = y_0.$$

Forward Euler update. On a uniform grid $t_n = t_0 + nh$ the forward Euler method is obtained by replacing $y'(t_n)$ with the finite difference $(y_{n+1} - y_n)/h$, giving

$$y_{n+1} = y_n + h\, f(t_n, y_n).$$

Local truncation error. The local truncation error (LTE) is the one–step error when the method is applied to the exact solution:

$$\tau_{n+1} := \frac{1}{h}\big(y(t_{n+1}) - y(t_n)\big) - f(t_n, y(t_n)).$$

A Taylor expansion of $y(t)$ about t_n yields

$$y(t_{n+1}) = y(t_n) + hy'(t_n) + \tfrac{h^2}{2}y''(\xi_n),$$

so that

$$\tau_{n+1} = \tfrac{h}{2}y''(\xi_n).$$

If y'' is bounded on the interval by M, then

$$|\tau_{n+1}| \leq \tfrac{M}{2} h = O(h^2).$$

Global error. From above, Forward Euler has local truncation error $O(h^2)$ per step. Over a fixed interval we take about $1/h$ steps, so the accumulated error is

$$O(h^2) \times O(1/h) = O(h).$$

A Lipschitz condition on f ensures that these local errors do not grow faster than exponentially, and over a fixed time interval this growth is bounded by a constant independent of h. Therefore the global error of forward Euler is $O(h)$.

2.2 Algorithm

Forward Euler Method. Goal: Iteratively apply Forward Euler update through the end of IVT interval.
Given $f(t, y)$, initial data (t_0, y_0), final time T, and a step size h:

- Set $t_n = t_0 + nh$ for $n = 0, 1, \ldots, N$.
- Start from y_0.
- For each step:
  1. Evaluate the slope $f(t_n, y_n)$.
  2. Take an explicit step:
  $$y_{n+1} = y_n + h\, f(t_n, y_n).$$

2.3 Experiments

Example A (Primary Convergence Check). The first problem is

$$y'(t) = -2y(t), \qquad y(0) = 1, \qquad t \in [0, 5],$$

with exact solution $y(t) = e^{-2t}$. This example is scalar and non-stiff, with Lipschitz constant $L = 2$ in y.
Below, are the (i) error vs. h in loglog scale and (ii) a small table of error ratios.

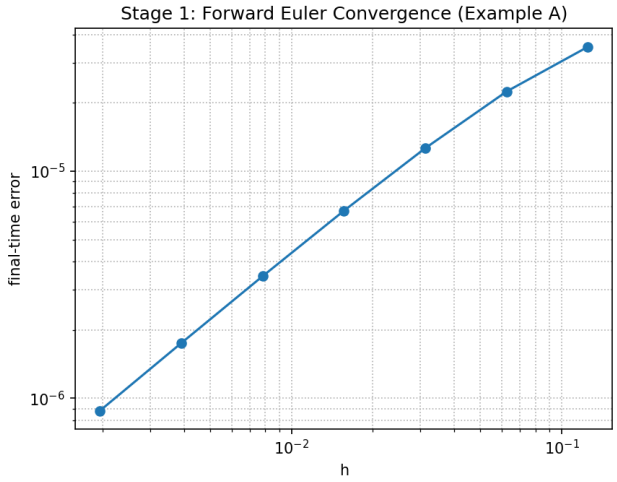

Figure 1: Forward Euler convergence for Example A.

Table 1: Forward Euler convergence for Example A at $t = 5$.

k	$h = 2^{-k}$	final-time error $E(h)$	ratio $E(h)/E(h/2)$
3	0.125000	3.534334e-05	
4	6.250000e-02	2.245724e-05	1.573806
5	3.125000e-02	1.263538e-05	1.777331
6	1.562500e-02	6.696543e-06	1.886851
7	7.812500e-03	3.446420e-06	1.943043
8	3.906250e-03	1.748178e-06	1.971435
9	1.953125e-03	8.803852e-07	1.985697

Example B (Logistic Growth). The second problem is the logistic equation

$$y'(t) = 3y(t)\Big(1 - \frac{y(t)}{2}\Big), \qquad y(0) = 0.2, \qquad t \in [0, 5],$$

On the interval $0 \le y \le 2$ relevant to the solution, the partial $\partial f/\partial y = 3 - 3y$ is bounded in magnitude by 3, so a natural Lipschitz constant is $L = 3$. Below, are the (i) error vs. h in loglog scale and (ii) a small table of error ratios.

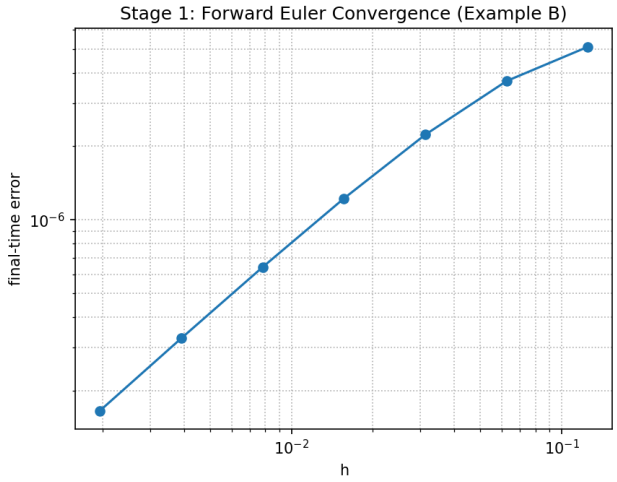

Figure 2: Forward Euler convergence for Example B.

Table 2: Forward Euler convergence for Example B at $t = 5$.

k	$h = 2^{-k}$	final-time error $E(h)$	ratio $E(h)/E(h/2)$
3	0.125000	5.104728e-06	
4	6.250000e-02	3.698204e-06	1.380327
5	3.125000e-02	2.230648e-06	1.657905
6	1.562500e-02	1.223833e-06	1.822674
7	7.812500e-03	6.407475e-07	1.910008
8	3.906250e-03	3.277981e-07	1.954702
9	1.953125e-03	1.657824e-07	1.977279

2.4 Results and Discussion

With respect to both examples, the loglog error plots show the expected first–order behavior: the slope of the line is very close to 1, and the error ratios $E(h)/E(h/2)$ in the tables settle toward 2 as h becomes small. This is directly in line with the theoretical global error bound of $O(h)$, as halving the step size should halve the error. Example A produces slightly smaller absolute errors because its Lipschitz constant is lower. Example B has a larger effective Lipschitz constant, which increases the error constant but does not change the observed order. Ultimately, the numerical results of the two examples match the theory cleanly. Forward Euler method has $O(h^2)$ local truncation error that accumulates over $O(1/h)$ steps to achieve a global $O(h)$ error, and this is visible in both the plots and the tables.

3 Stage 2: Higher-Order *Explicit* One-Step Methods and Validation

3.1 Theory

Heun's method (RK2). Heun uses a predictor step followed by a corrected slope:

$$k_1 = f(t_n, y_n), \qquad k_2 = f(t_n + h,\, y_n + hk_1),$$

$$y_{n+1} = y_n + \frac{h}{2}(k_1 + k_2).$$

This method has LTE $O(h^3)$ and global error $O(h^2)$.

Classical RK4. The standard fourth-order Runge-Kutta method is

$$k_1 = f(t_n, y_n), \qquad k_2 = f(t_n + \tfrac{h}{2},\, y_n + \tfrac{h}{2}k_1),$$

$$k_3 = f(t_n + \tfrac{h}{2},\, y_n + \tfrac{h}{2}k_2), \qquad k_4 = f(t_n + h,\, y_n + hk_3),$$

$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4).$$

Its LTE is $O(h^5)$ and its global error is $O(h^4)$.

Order relationship. Both methods follow the same pattern from stage 1. That is, if a one-step method has LTE $O(h^{p+1})$, then over approximately $1/h$ steps on a fixed interval, the global error fits

$$O(h^{p+1}) \times O(1/h) = O(h^p).$$

This implies that RK2 is second order and RK4 is fourth order. Higher-order methods require more function evaluations per step. Consequently, this stage evaluates work order vs precision.

3.2 Algorithms

See above. These methods are fully explicit since all quantities needed for the update are known at time t_n.

3.3 Experiments

In order to verify the order of Heun's and RK4, we run the same analysis as Stage 1.

Step sizes.

$$h_k = 2^{-k}, \qquad k = 3, \ldots, 9.$$

For each h_k and for each method (Euler, RK2, RK4), we compute the final-time error.

$$E(h_k) = |\, y(T) - y_N^{(k)} \,|.$$

Example A (Primary Convergence Check). Recall the ODE

$$y'(t) = -2y(t), \qquad y(0) = 1,$$

has the exact solution $y(t) = e^{-2t}$, so the error can be computed directly.

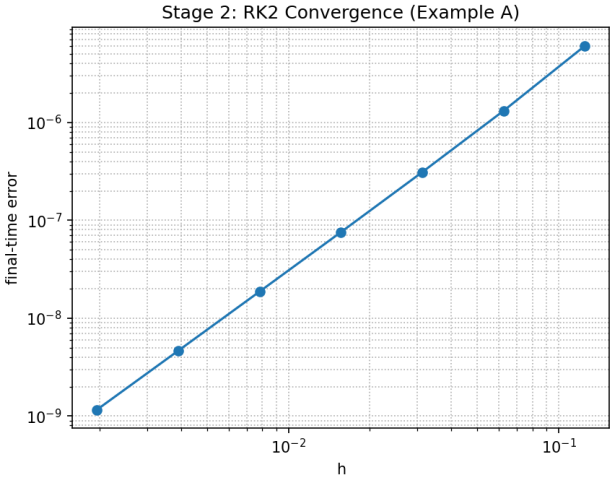

Figure 3: Example A: Heun's (RK2) Convergence

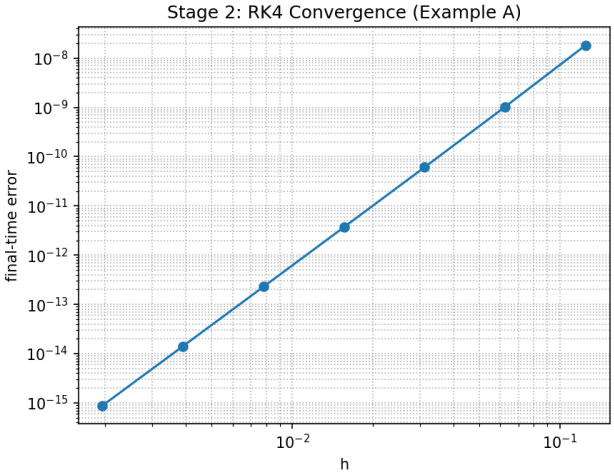

Figure 4: Example A: RK4 Convergence

Example B (Logistic Growth). For

$$y'(t) = 3y(t)\left(1 - \frac{y(t)}{2}\right), \qquad y(0) = 0.2,$$

the following plots compare error with respect to the analytical solution.

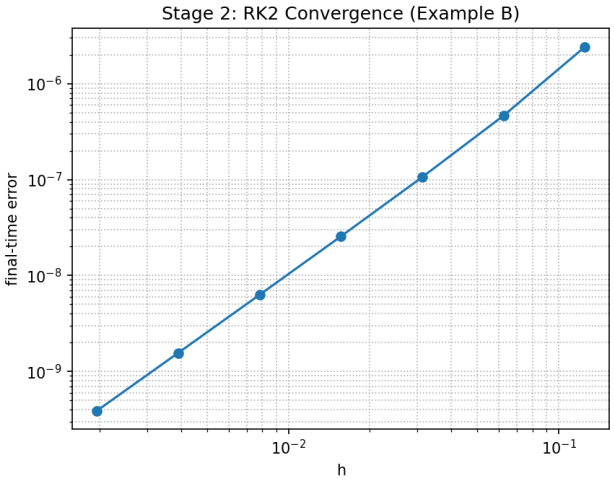

Figure 5: Example B: Heun's (RK2) Convergence

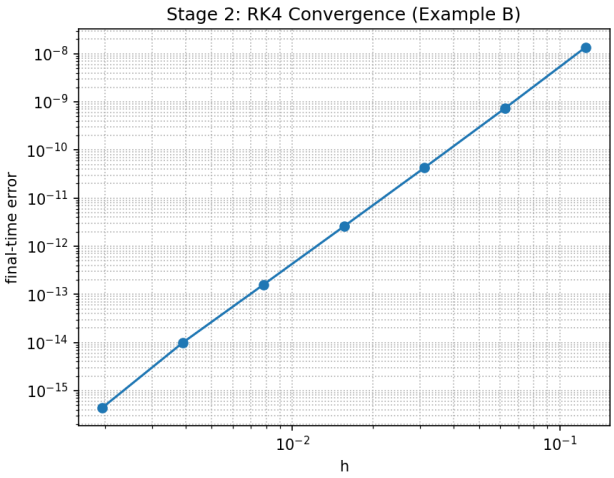

Figure 6: Example B: RK4 Convergence

Work–precision diagram. To compare accuracy versus computational cost, we generate a work–precision diagram with respect to Example A. For each method (Euler, RK2, RK4) and each step size h_k,

$$\text{work}(h_k) = N(h_k) \times m,$$

where $N(h_k)$ is the number of steps and m is the number of RHS computations per step.

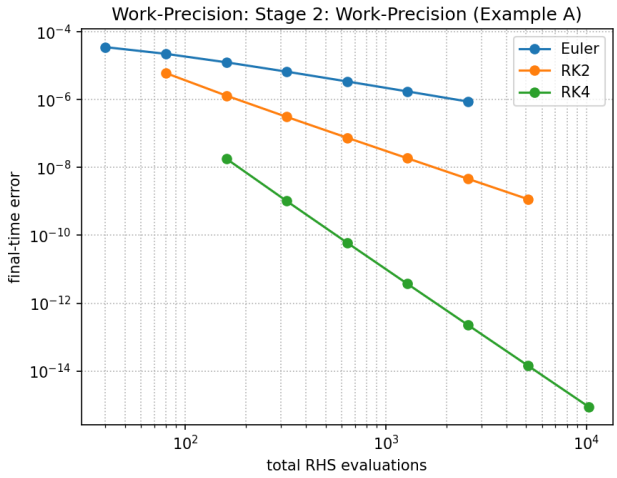

Figure 7: Work-Precision Plot (Example A)

3.4 Results and Discussion

The loglog convergence plots demonstrate the theory accurately. For RK2, the error plot has an approximate slope of 2 and RK4 is even steeper at 4 with extremely small final-time errors for the smallest step sizes. This shows that both methods, on both examples, achieve their theoretical global order.

The work–precision plot reinforces the theory. For a given amount of RHS computations (work), RK4 achieves far smaller errors than RK2 or Euler, and RK2 outperforms Euler. The curves are distinct since lower order methods require more work to reach the same accuracy. Concretely, the plot shows the tradeoff that higher-order methods requiring more work per step, but also achieves greater accuracy.

4 Stage 3: Stability and Timestep Selection via the Linear Model Problem

4.1 Theory

We intend to map absolute stability behavior on the linear test equation

$$y'(t) = \lambda y(t),$$

For a one-step method of the form

$$y_{n+1} = \gamma(h\lambda)\, y_n,$$

the quantity

$$g(z) = \gamma(z), \qquad z = h\lambda,$$

is called the stability function. Theory shows the numerical solution remains bounded for all n $\iff$

$$|g(h\lambda)| < 1.$$

The set of all $z \in \mathbb{C}$ satisfying $|g(z)| < 1$ is the method's absolute stability region.

Forward Euler. For Euler,

$$y_{n+1} = y_n + h\lambda y_n = (1 + h\lambda)y_n,$$

so the stability function is

$$g(z) = 1 + z.$$

The stability condition $|1 + z| < 1$ restricts to the interval

$$-2 < h\lambda < 0$$

when $\lambda \in \mathbb{R}, \lambda < 0$. Therefore Forward Euler is only stable for sufficiently small h.

Heun's method (RK2). For the linear test equation $y' = \lambda y$, Heun's method gives the stability function

$$g(z) = 1 + z + \tfrac{1}{2}z^2, \qquad z = h\lambda.$$

The absolute stability condition $|g(z)| < 1$ is equivalent to

$$-1 < 1 + z + \tfrac{1}{2}z^2 < 1.$$

Hence for $\lambda \in \mathbb{R}, \lambda < 0$ the stability interval of RK2 is

$$-2 < h\lambda < 0,$$

the same as forward Euler, even though Heun has higher order. The stability region in $\mathbb{C}$ is larger than Euler's, but unchanged along $\mathbb{R}^-$.

Classical RK4. For the same test equation RK4 gives

$$y_{n+1} = g(h\lambda)\, y_n,$$

with stability function

$$g(z) = 1 + z + \tfrac{1}{2}z^2 + \tfrac{1}{6}z^3 + \tfrac{1}{24}z^4, \qquad z = h\lambda.$$

This is exactly the fourth-order Taylor polynomial of e^z.

The absolute stability region is again $\{z \in \mathbb{C} : |g(z)| < 1\}$. This is challenging to show a closed form but can be solved numerically, producing a stability region in $\lambda \in \mathbb{R}^-$ of

$$-2.79 \lesssim h\lambda < 0.$$

Local linearization. For a nonlinear ODE $y'(t) = f(t, y)$, the behavior near a state y at time t can be approximated by locally linearizing:

$$f(t, y + \varepsilon) \approx f(t, y) + \frac{\partial f}{\partial y}(t, y)\, \varepsilon.$$

If $\partial f/\partial y < 0$ and large in magnitude, the local problem resembles the stiff linear test problem $y' = \lambda y$ with $\lambda = \frac{\partial f}{\partial y}$. We consequently use the stability condition for the linear model to predict an upper bound on h:

$$h < \frac{z_{\max}}{\left|\min_t \partial f/\partial y(t, y(t))\right|},$$

where $z_{\max}$ is the leftmost point of the method's real stability interval . This shows a simple way to anticipate instability before it happens.

4.2 Experiments

Linear Stability Model: $y' = \lambda y$ For a $\lambda \in \mathbb{R}^-$, both Forward Euler and RK2 are stable exactly when

$$-2 < h\lambda < 0.$$

We tested two values, $\lambda = -1$ and $\lambda = -10$, and for each method chose:

$$h_{\text{stable}} = 0.8 \times \frac{-2}{\lambda}, \qquad h_{\text{unstable}} = 1.2 \times \frac{-2}{\lambda}.$$

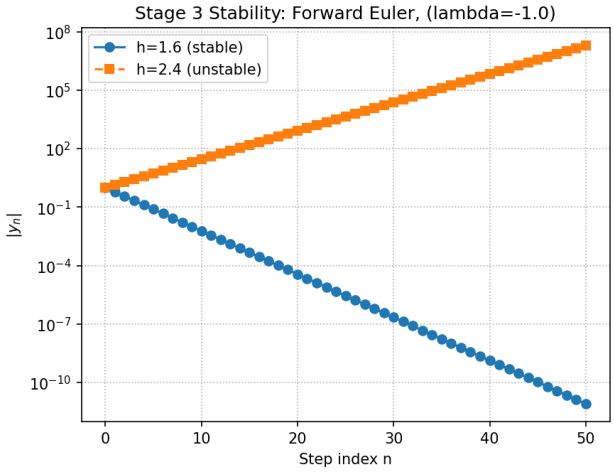

Figure 8: Euler for $\lambda = -1$

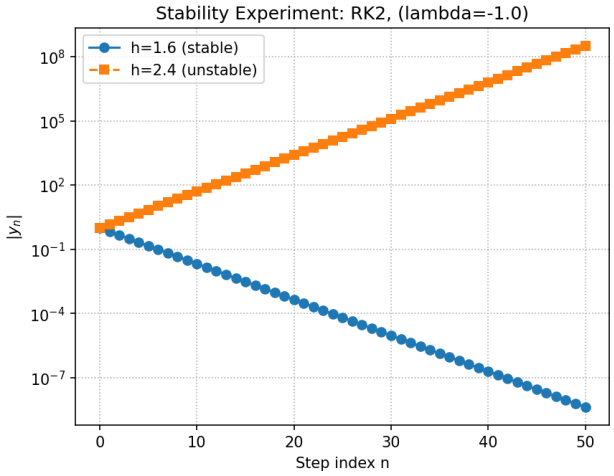

Figure 9: RK2 for $\lambda = -1$

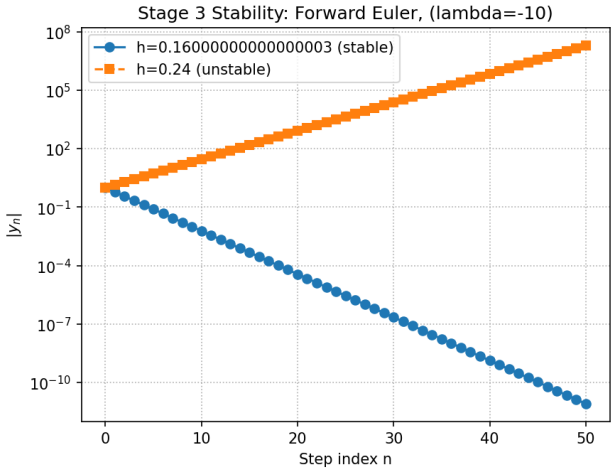

Figure 10: Euler for $\lambda = -10$

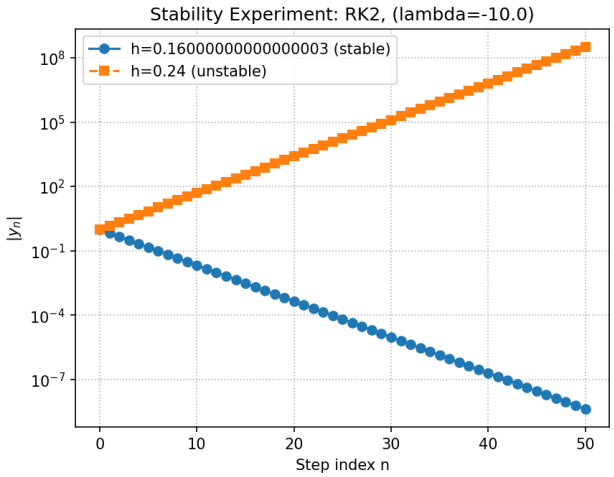

Figure 11: RK2 for $\lambda = -10$

Non-autonomous check with local linearization (Example C). Recall Example C:

$$y'(t) = 4t^2 \cos(y), \qquad y(0) = 0.1, \qquad t \in [0, 4].$$

Notice the effective stiffness of this ODE is not time invariant. We then study the practicality of local linearization and tracked

$$\lambda(t) = \frac{\partial f}{\partial y}(t, y(t)) = -4t^2 \sin(y(t)).$$

A reference solution was computed using RK4 with $h = 0.001$. In order to generate visual instability, we increase

$$h_{\text{stable}} = 0.5h_{\max}, \qquad h_{\text{unstable}} = 2.5h_{\max}.$$

And report

$$z(t) = h\,\lambda(t)$$

for both stepsizes.

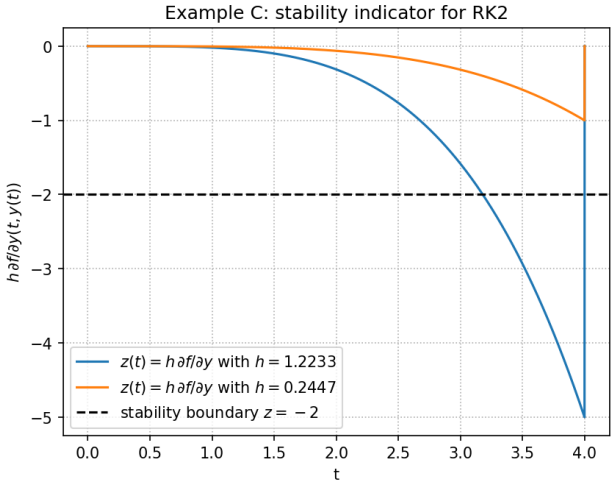

Figure 12: Stability Indicator of RK2 on Example C

We then run a $y(t)$ v. t plot for both Euler and RK2 with the two stepsizes. The Euler plot generated minor distortion which makes sense since the method itself is forgiving. Consequently, I have displayed only RK2 here, yet all plots can be found in the appendix.

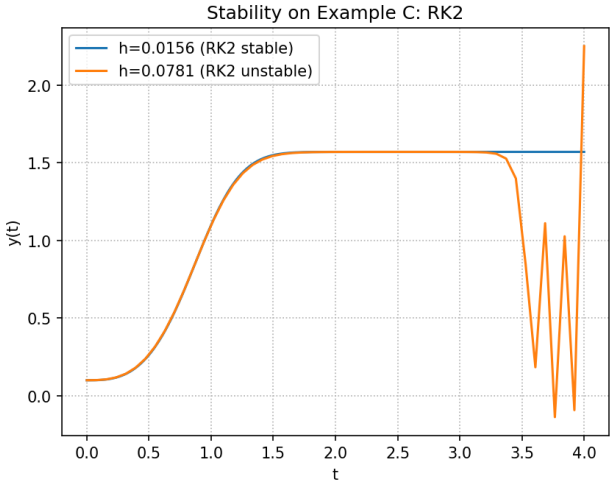

Figure 13: Stability of RK2 for Example C

4.3 Results and Discussion

For the linear test equation $y' = \lambda y$, the numerical behavior matched the predicted stability intervals. When $h\lambda$ lay inside the stability region $(-2, 0) \subset \mathbb{R}$, both Forward Euler and RK2 produced

decay in $|y_n|$. Choosing h so that $h\lambda$ crossed -2 immediately produced growth which verifies the theoretical stability region.

For Example C, the stability indicator $z(t) = h\,\partial f/\partial y(t, y(t))$ validates the observed behavior. With the stable h, $z(t)$ stayed above the boundary $z = -2$ on the entire interval, and both methods remained stable (although we only present RK2, the Euler plots can be found in the Appendix). With the unstable stepsize, $z(t)$ dipped below -2 only near the end of the interval, predicting localized instability. RK2 showed a clear blow-up shortly after this crossing, clearly demonstrating the validity of the indicator and the general theoretics for stability. Overall, the experiments confirm that the stability condition provides a reliable local linearization predictor of where instability first appears. Most specifically, even in a nonlinear, time-dependent example such as Example C, we see the same results.

5 Stage 4: Systems of ODEs: Time Series and Linear Constant Coefficients

5.1 Theory

We extend the scalar analysis to systems

$$y'(t) = Ay(t), \qquad y(0) = y_0, \quad A \in \mathbb{R}^{m\times m}.$$

When A is diagonalizable with eigenvalues and vectors (λ_j, v_j), the exact solution can be written in terms of the matrix exponential,

$$y(t) = e^{tA}y_0 = \sum_j c_j e^{\lambda_j t} v_j,$$

where the coefficients c_j depend on y_0. If $\Re(\lambda_j) < 0$ the corresponding component decays. If $\Re(\lambda_j) > 0$ it grows, and purely imaginary eigenvalues imply undamped oscillations.

A one-step method with stability function $g(z)$ applied to $y' = Ay$ generates the update

$$y_{n+1} = g(hA)\, y_n.$$

If A is diagonalizable, we can write $A = V\Lambda V^{-1}$ with $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_m)$ therefore

$$y_{n+1} = V\, g(h\Lambda)\, V^{-1} y_n.$$

Therefore, the absolute stability condition for the system is

$$|g(h\lambda_j)| < 1 \quad \text{for all eigenvalues } \lambda_j \text{ of } A.$$

5.2 Algorithms

We extend the one–step methods to systems

$$y'(t) = f(t, y(t)), \qquad y(t) \in \mathbb{R}^m,$$

In code, the only change from the scalar case is that y_n and $f(t_n, y_n)$ are now NumPy arrays and each update is applied componentwise.

For each system, we produce

1. Time series, by plotting each component of y_n versus t_n;
2. Phase portraits in 2D, by plotting one component against another.

5.3 Experiments

Time Series and Phase Portrait Experiments (Example D) We test Forward Euler, Heun's method (RK2), and RK4 on the 2D linear system from Example D and E using two step sizes: a small h_{stable} and a larger h_{unstable}. For each method we compute on $t \in [0, 10]$ and record:

1. time series plots of both solution components
2. phase portraits in the (y_1, y_2) plane.

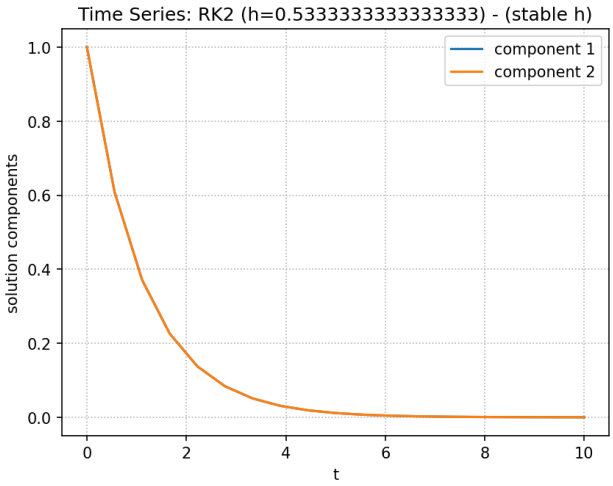

Figure 14: Time Series of RK2 with h_{stable} (Example D)

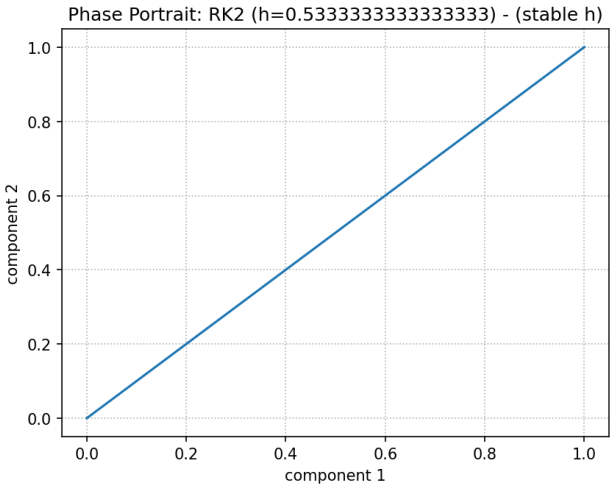

Figure 15: Phase Portrait of RK2 with h_{stable} (Example D)

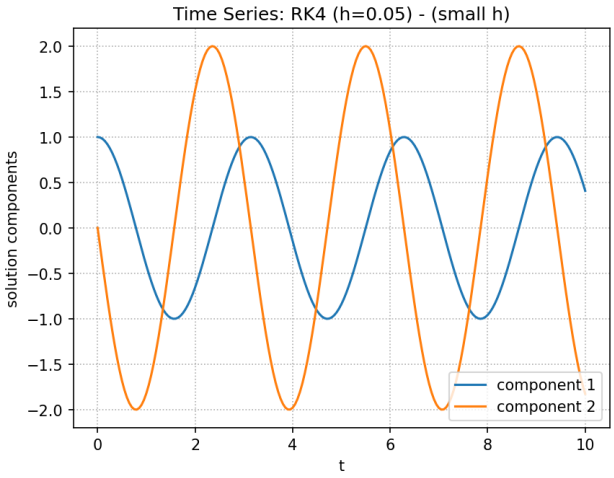

Figure 16: Time Series of RK4 with small h (Example E)

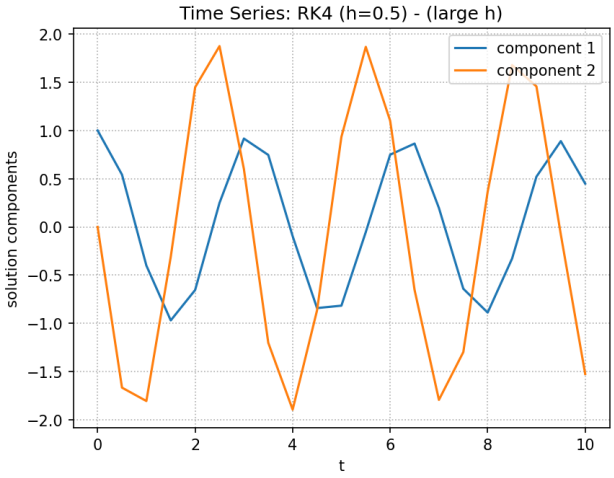

Figure 17: Time Series of RK4 with large h (Example E)

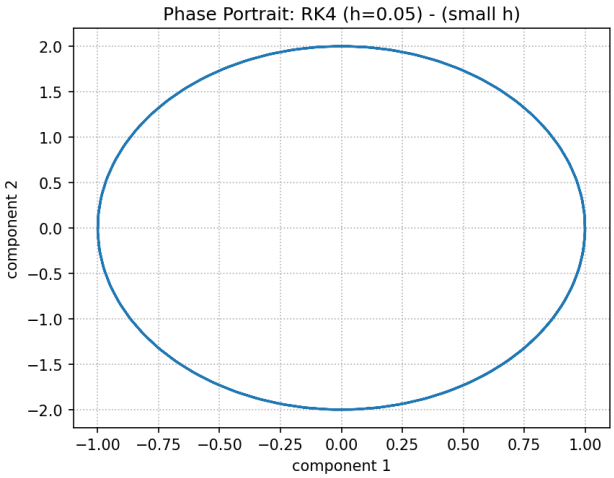

Figure 18: Phase Portrait of RK4 with small h (Example E)

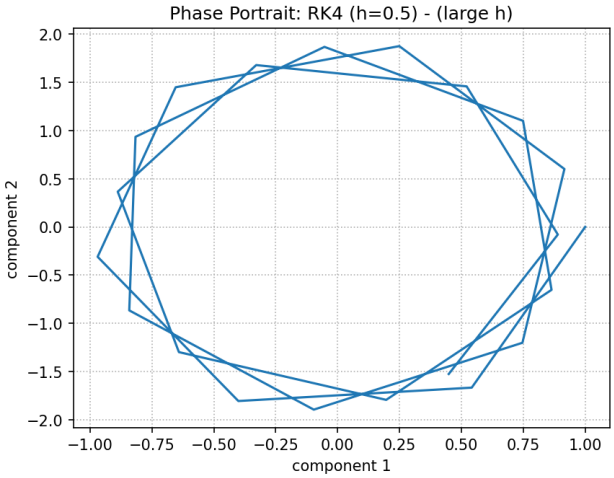

Figure 19: Phase Portrait of RK4 with large h (Example E)

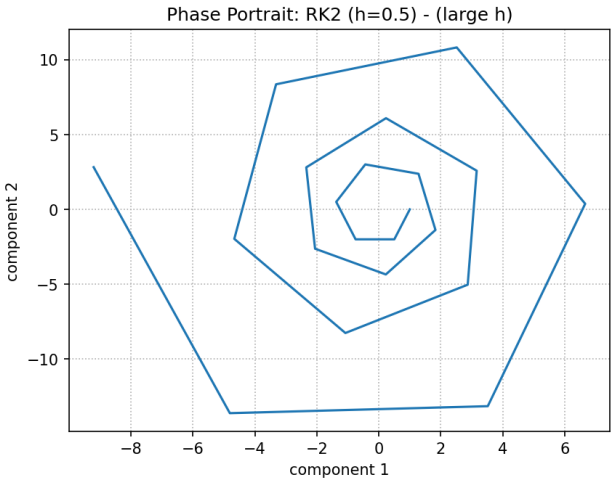

Figure 20: Phase Portrait of RK2 with large h (Example E)

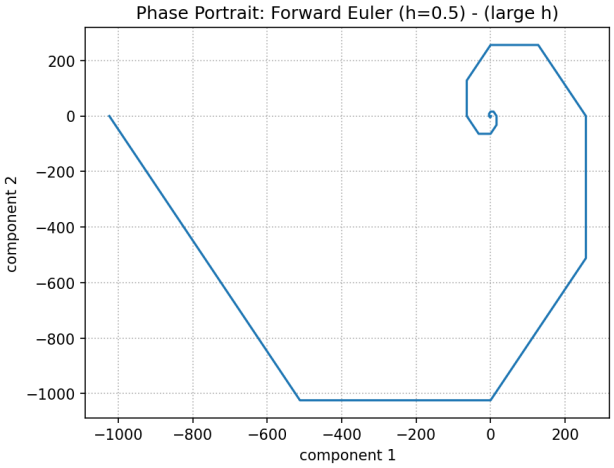

Figure 21: Phase Portrait of Forward Euler with large h (Example E)

For the sake of brevity and due to the sufficient analysis possible with these plots, I have chosen not to include all 20+ required plots in the document. Although considered, I have chosen to leave the remainder entirely off the report. All plots can be found in the github and locally submitted zip file.

For all three methods, the smaller step sizes produce solutions that decay smoothly. The larger step sizes may introduce oscillations, amplification, or phase drift, depending on how far outside of the stability region the step size is.

Stability Predictions vs. Observed Behavior (Eigenvalue Analysis) For a linear system $y' = Ay$, a one-step method with step size h is predicted to be stable when each scaled eigenvalue $h\lambda_j$ lies within the method's stability region.

The 2D system in Example D. Although theory predicts instability once $h > 2/|\lambda_{\min}|$, the experiments show that some unstable h still produce decay. This is because stability in practice depends on how the initial condition projects onto the corresponding eigenvectors. For Example D, the chosen initial condition lies almost entirely in the eigenspace of the lesser $\lambda_2 = -1$, so the unstable significant $\lambda = -3$ is missed. Therefore, Euler and RK2 display stable-looking behavior even though the formal bound is violated.

This distinction between theory and mode-dependent stability behavior is characteristic of multi-component linear systems.

5.4 Results and Discussion

For the linear system in Example D, the numerical results generally follow the eigenvalue-based stability predictions. That is, smaller step sizes produce decay toward equilibrium, while larger step sizes can introduce growth or oscillations. However, the plots also show that violating the theoretical stability bound does not always lead to visible instability. With the initial condition $y_0 = (1, 1)^T$, the solution lies entirely in the eigenspace of $\lambda = -1$, so the more restrictive $\lambda = -3$ is not relevant. Consequently, even step sizes that exceed the theoretical limit $h < 2/|\lambda_{\min}|$ may still produce stable-looking solutions. This demonstrates the idea that stability for systems of ODEs depends not only on eigenvalues but also on how strongly each mode is activated by the initial condition.

Example E demonstrates the differing behavior of the methods on oscillation problems. For small step sizes, all three methods produce quality periodic updates and the higher order RK4 preserves both amplitude and phase most accurately. With larger step sizes, Euler produces an outward spiral, while RK2 produces a noticeable inward spiral. RK4 remains qualitatively accurate even for moderately large h, though some phase drift becomes visible. These observations agree with the theory of the non-zero imaginary component relative to each method's stability region.

6 Stage 5: Implicit One-Step Methods and Nonlinear Solvers

6.1 Theory

Backward Euler and Trapezoidal Methods Backward Euler uses the Forward Euler idea at a future time t_{n+1}:

$$y_{n+1} = y_n + h\, f(t_{n+1}, y_{n+1}).$$

This is an implicit equation for y_{n+1}. Then for

$$G(y_{n+1}) := y_{n+1} - y_n - hf(t_{n+1}, y_{n+1}),$$

we solve $G(y_{n+1}) = 0$ at each step.

The trapezoidal method takes the average slope between t_n and t_{n+1}:

$$y_{n+1} = y_n + \frac{h}{2}\big(f(t_n, y_n) + f(t_{n+1}, y_{n+1})\big),$$

leading to the nonlinear equation

$$G(y_{n+1}) := y_{n+1} - y_n - \frac{h}{2}\big(f(t_n, y_n) + f(t_{n+1}, y_{n+1})\big) = 0.$$

In both cases, G is a scalar nonlinear function, allowing the use of the standard Newton's method for nonlinear solves.

Newton's Method Given a function $G : \mathbb{R} \to \mathbb{R}$, we use the root finding iteration

$$y^{(k+1)} = y^{(k)} - \frac{G(y^{(k)})}{G'(y^{(k)})}.$$

Under the usual assumptions, the sequence converges quadratically.

For the implicit methods above, the derivative in Newton's update is

$$G'(y) = 1 - h\,\partial_y f(t_{n+1}, y) \qquad \text{(backward Euler)},$$

$$G'(y) = 1 - \frac{h}{2}\,\partial_y f(t_{n+1}, y) \qquad \text{(trapezoidal)}.$$

In the experiments, y_n is used as the initial guess.

Local and Global Truncation Error Backward Euler satisfies $\tau_{n+1} = O(h)$ so its global error is $O(h)$. The trapezoidal method has $\tau_{n+1} = O(h^2)$ and global error $O(h^2)$.

For implicit methods, the nonlinear equation is not solved analytically. Instead, Newton iteration pushes $|G(y_{n+1})|$ below a defined tolerance. To preserve the method's theoretical order, this tolerance must be small compared to the LTE. We choose

$$|G(y_{n+1})| < Ch^{p+1},$$

where p is the order of the method. $p = 1$ for Backward Euler and $p = 2$ for trapezoidal. This guarantees the error of Newton is at one order higher than the LTE, so the global accuracy is unaffected.

6.2 Algorithms

With respect to the two implicit methods discussed in this stage, both update steps are listed above and the implementations are trivially different with the exception of the Newton iterations.

6.3 Experiments

For Examples A and B we compute the final-time error for a sequence of step sizes $h = 2^{-k}$ and compare against reference slopes.

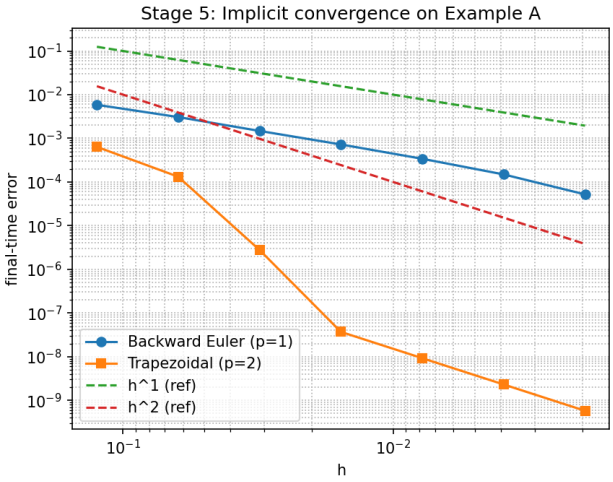

Figure 22: Stage 5: implicit convergence on Example A

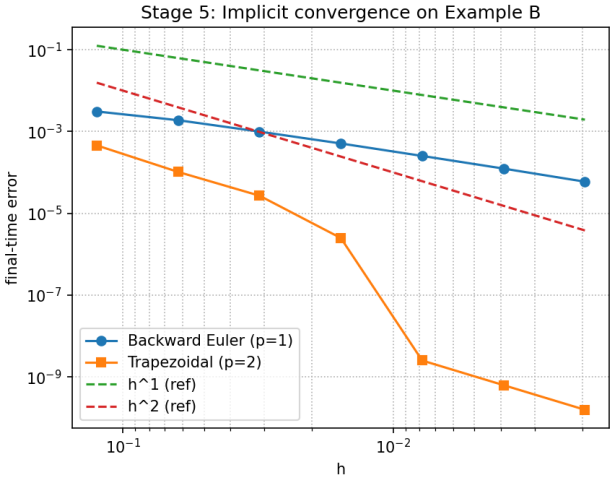

Figure 23: Stage 5: implicit convergence on Example B

Newton Iteration Statistics (Example F) For the nonlinear stiff Example F, for each h, we track the minimum, mean, and maximum number of Newton's method iterations per step.

Table 3: Example F: Newton iteration statistics

h	BE error	BEnewton(min/mean/max)	TR error	TRnewton(min/mean/max)
0.12500	9.61e-07	1/ 1.1/ 2	6.95e-04	2/ 2.0/ 2
0.06250	3.70e-06	1/ 1.1/ 2	1.48e-05	2/ 2.0/ 2
0.03125	1.02e-06	1/ 1.0/ 2	2.15e-07	1/ 1.5/ 2
0.01562	9.05e-07	1/ 1.0/ 2	3.17e-08	1/ 1.1/ 2
0.00781	5.07e-07	1/ 1.0/ 2	1.35e-08	1/ 1.0/ 2

Explicit vs. Implicit Methods on Example F Finally, we compare RK4 (with a stable $h = 2^{-8}$) with Backward Euler and trapezoidal method using a much larger $h = 0.125$. All solutions are compared against the RK4 reference.

Table 4: Example F: explicit vs implicit at a target accuracy

Method (h)	final-time error	steps
RK4 (h=0.00390625)	9.61e-07	512
BE (h=0.125)	3.70e-06	16
Trapezoid (h=0.125)	1.02e-06	16

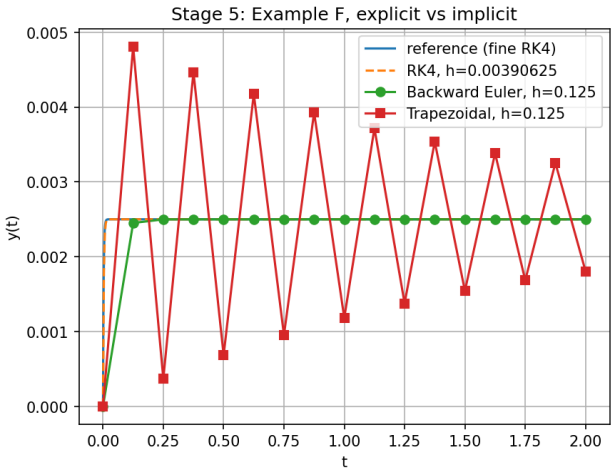

Figure 24: Stage 5: Implicit Methods vs. Explicit RK4 on Example F.

6.4 Results and Discussion

The convergence tests on Examples A and B confirm the theoretical accuracy of the implicit methods. Backward Euler shows first-order behavior and the trapezoidal method second-order behavior, with convergence slopes that match the respective reference h and h^2 lines. The variance in decay at the smallest h values is consistent with the factors of discretization error and Newton tolerance (scaled h^{p+1}). Overall, each scheme achieves their theoretical orders even with inexact Newton solves. This holds true if the residual tolerances decrease with respect to h.

For the nonlinear stiff Example F, the Newton iteration statistics show that backward Euler generally converges in a single Newton iteration per step, while the trapezoidal method requires one to two iterations. The global errors decrease for smaller h. For sufficiently small h the trapezoidal method delivers higher accuracy, reflecting its second-order nature. The explicit vs implicit comparison demonstrates the main qualitative feature of stiff problems. The explicit RK4 requires a very small h to remain stable, but both implicit methods remain stable and reasonably accurate at much larger step sizes. This tradeoff table shows, especially with respect to the tiny number of Newton iterations per step, shows the comparison between the implicit methods and RK4 well.

7 Stage 6: Stiffness in Practice: Detect, Quantify, and Choose Methods

7.1 Theory

Stability-Limited Stepsize h_σ We recall this definition of h_σ from prior stages. As a brief recap, we consider an absolutely stable one step method for a given z if the stability function satisfies $|g(z) < 1|$. Here, $z = h\lambda$ and consequently we can derive an interval of stability for the step size h.

For a nonlinear problem $y' = f(t, y)$, local linearization gives the substitution of f by

$$f(t, y) \approx f(t, y(t)) + \partial_y f(t, y(t))(y - y(t)),$$

such that the y- direction dynamics are controlled by a linear term with eigenvalue

$$\lambda(t) = \partial_y f(t, y(t)).$$

Then for

$$\lambda_{\min} := \min_t \lambda(t) < 0,$$

and the explicit method having a real stability interval $(a, 0)$, then h must satisfy

$$h < \frac{|a|}{|\lambda_{\min}|}$$

in order to remain stable. We use h_σ to refer to this stability-limited step size for the given explicit method and problem.

Accuracy-Limited Stepsize h_τ For a method of order p, the global error shows

$$\|y(T) - y_h(T)\| \approx Ch^p$$

where C is some constant. To meet an error tolerance at the final time, we need

$$Ch^p < \text{tol} \quad \Rightarrow \quad h < (\frac{\text{tol}}{C})^{1/p}.$$

in order to remain stable. We use h_τ to refer to this accuracy-limited step size for a given method and problem. If stability was not a concern, we would be more motivated to use h_τ

Stiffness and Choosing Explicit vs. Implicit Methods For explicit methods, it is natural to choose

$$h < \min\{h_\sigma, h_\tau\}.$$

A problem is effectively stiff for a given explicit method if

$$h_\sigma << h_\tau.$$

In other words, if stability forces the step size to be significantly smaller than what accuracy requires.

Implicit methods such as Backward Euler and the trapezoidal method have much larger stability regions. Consequently, for stiff problems, implicit methods can choose h based mainly on h_τ, taking far fewer steps to achieve the same accuracy.

7.2 Algorithms

In this stage, we consider Example G,

$$y'(t) = -1000\, y(t) + \sin t, \qquad y(0) = 0, \qquad t \in [0, 10],$$

to quantify stiffness and compare explicit and implicit methods.

Estimating the stability step h_σ for explicit methods This is quite simplistic and is done analytically with computer use for rounding. We choose some $h_{stable} < h_\sigma < h_{unstable}$ in order to plot and compare values.

Estimating the accuracy step h_τ and comparing work

1. Fix a target tolerance tol for the final-time error.
2. Compute a a high accuracy reference solution $y_{\text{ref}}(T)$ with RK4 using a very small $h = 2^{-8}$.
3. For each method M (explicit and implicit):
   (a) Run the method on Example G with a sequence of step sizes $h_k = 2^{-k}$ over $[0, 10]$.
   (b) For each run, compute the final-time error.
   (c) Let $h_{\tau,M}$ be the largest h_k such that the final-time error is ¡ tol.
   (d) Keep track of the number of RHS computations for that h_k.

7.3 Experiments

Recall Example G,

$$y'(t) = -1000\, y(t) + \sin t, \qquad y(0) = 0, \qquad t \in [0, 10],$$

which is stiff because the Jacobian satisfies $\partial_y f(t, y) = -1000$. It follows that

$$\lambda_{\min} = \min_{t \in [0,10]} \partial_y f(t, y(t)) = -1000.$$

Stability threshold h_σ for explicit methods The following figure shows the stable Forward Euler method. The numerical solution remains bounded with amplitude $O(10^{-3})$, consistent with the exact steady-state solution. The next figure shows the unstable run. The numerical solution blows up, confirming the predicted stability limit. This effectively shows how the predicted h_σ can be seen on either side. This visual is reproducible against all other explicit methods in this report.

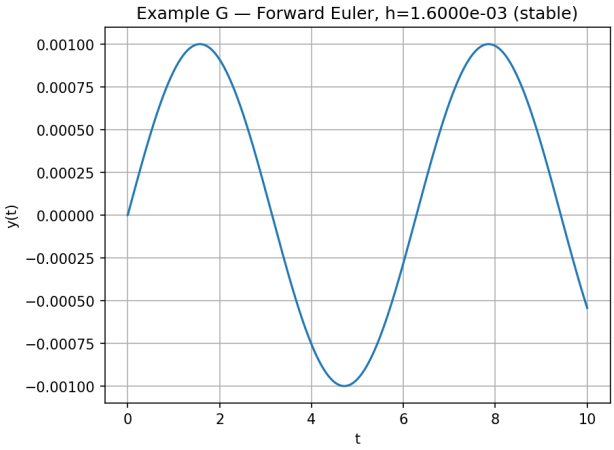

Figure 25: Example G: stable Forward Euler with $h < h_\sigma$

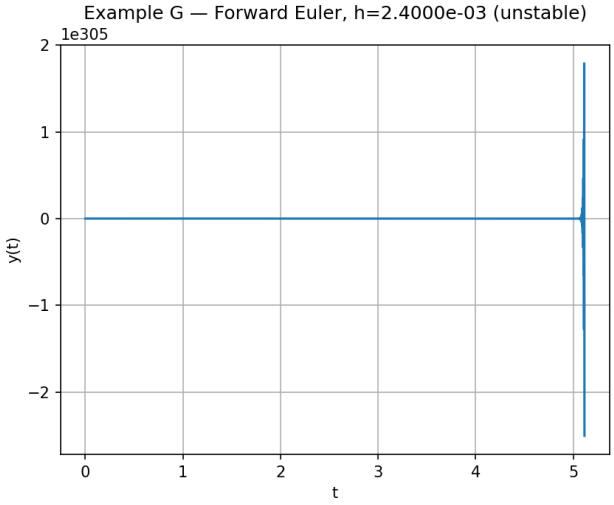

Figure 26: Example G: unstable Forward Euler with $h > h_\sigma$

Accuracy-threshold h_τ For a fixed tolerance tol $= 10^{-4}$, we first compute a high order reference with RK4 ($h = 10^{-8}$. Then, for each method M, we run a sequence of decreasing uniform step sizes h_k and determine the largest one that satisfies the tol and set that to $h_{\tau,M}$.

Work comparisons at fixed accuracy tol We measure work as the total number of right-hand-side evaluations, and the implicit solves use a fixed number of Newton iterations. The following table summarizes the resulting $h_{\tau,M}$ and work. The explicit methods require extremely small time steps to remain stable and this forces the h_τ much lower than the implicit methods.

Method	step size h_τ	RHS evaluations (work)	final-time error
Forward Euler	0.001953125	5120	5.301647202846094e-10
RK2	0.001953125	10240	2.0642213223020373e-08
RK4	0.001953125	20480	2.352644104921045e-10
Backward Euler	0.25	40	5.870984452478571e-08
Trapezoidal	0.25	40	3.8561976364604724e-06

Table 5: Stage 6: work required to reach tol=10^{-4} on Example G.

The next plot displays the same comparison graphically. Explicit methods differ by order but all struggle with the fact that the stability threshold h_σ is far smaller than the accuracy threshold h_τ. On the other hand, the implicit methods perform at $h \gg h_\sigma$ with stability. This shows their value for still problems.

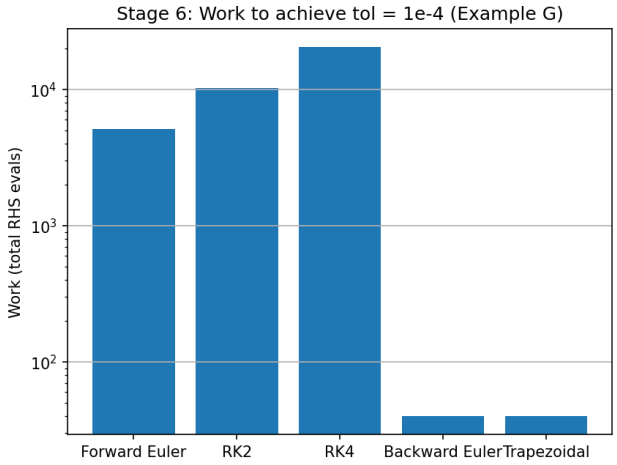

Figure 27: Work to achieve tol = 10^{-4} for Example G.(LOG SCALE)

7.4 Results and Discussion

The stability tests confirm that the explicit methods are severely constrained on Example G. For Forward Euler, the predicted stability threshold $h_\sigma \approx 2 \times 10^{-3}$ acts as an effective and accurate indicator of convergence. When $h < h_\sigma$, the numerical solution remains bounded and has an oscillatory amplitude of the true theoretical steady state solution. On the other hand, $h > h_\sigma$ has a loss of stability and the numerical solution blows up towards the end of the interval. RK2 and RK4 exhibit the same pattern with extremely similar theoretical stability thresholds. These observations show that, with respect to Example G, stability is a far larger bottleneck than accuracy. That is h_σ is a better tool than h_τ.

The accuracy experiments continue with this reasoning. The accuracy-limited step sizes h_τ are even smaller than h_σ for the explicit schemes. In particular, achieving a final-time error near (arbitrarily) tol = 10^{-4} requires step sizes at least an order of magnitude smaller than the stability h_σ. As a consequence, all explicit methods incur very large work counts. Even RK4, a higher order method, cannot compensate for the stiffness-imposed bound on h.

On the flip side, both implicit methods (Backward Euler and Trapezoidal) are able to take

steps several orders of magnitude larger than the explicit schemes while maintaining stability. Their accuracy-limited step sizes lie near 10^{-1} and this forces the amount of RHS computations to be very small. Furthermore, their Newton iterations converge extremely quickly (as seen in Stage 5), so the cost per step remains low. The observed results from the computed h_σ, h_τ, and the work-precision plot and table demonstrate the relevance of stiffness. That is, explicit methods are forced into inefficient regimes by stability concerns while implicit methods achieve the target accuracy with far less computational effort.

8 Stage 7: Adaptive Stepsize Control

8.1 Theory

For this stage, the goal is to adjust the step size for a method adaptively so that the local error stays below a prescribed tolerance while also avoiding unnecessarily small steps in regions where the solution is smooth.

Basic Idea of Step Doubling For a one-step method of order p, say γ, there are two ways to advance the numerical solution:

1. One step of size h:
$$y_{n+1}^{(h)} = \gamma_h(t_n, y_n).$$
2. Two half–steps of size $h/2$:
$$y_{n+1}^{(h/2)} = \gamma_{h/2}\big(t_n + h/2,\, \gamma_{h/2}(t_n, y_n)\big).$$

Both versions converge to the exact solution $y(t_{n+1})$ as $h \to 0$, but with different leading truncation errors. The difference
$$\tau_{n+1} := y_{n+1}^{(h/2)} - y_{n+1}^{(h)}$$
is an $O(h^{p+1})$ quantity that can be used as a placeholder for the local truncation error. We take
$$\text{error}_{n+1} := |\tau_{n+1}|$$
as the local error estimate for the step starting at t_n.

Accept/Reject Criterion and Stepsize Update Given a specified tolerance tol, the step from t_n to t_{n+1} is accepted if
$$\text{error}_{n+1} \le \text{tol},$$
. When this is true, we set $y_{n+1} = y_{n+1}^{(h/2)}$ and proceed since this is the more accurate approximation. Else, the step is rejected, and we retry from t_n with a smaller step size h.

To choose the next step size, we use the standard controller based on the asymptotic scaling $\text{error}_{n+1} \approx Ch^{p+1}$. Solving $Ch^{p+1} \approx \text{tol}$ for h and considered the observed error gives
$$h_{\text{new}} = h \cdot \text{clip}(\eta_{min}, \eta_{max}, s \cdot \left(\frac{\text{tol}}{\text{err}_{n+1}}\right)^{1/(p+1)})$$
where $s \in (0,1)$ is a safety factor, and $\text{clip}(a,b,x) = \min(b, \max(a,x))$ prevents aggressive changes. This update is a practical controller that enforces the global budget model. That is , for $\Sigma E_n \approx \epsilon_g$ we have the individual step target $E_n/h_n \le \epsilon_g/(T - t_o)$.

Global Error Behavior and Limitations If the local error estimator tracks the exact truncation error reliably and the controller maintains $\text{error}_{n+1} < \text{tol}$ at each accepted step, then the global error at a final time scales tol by a order of p where p is the order of the underlying method. In practice, adaptive step sizing provides accuracy at a significantly reduced cost.

However, adaptive step size control based entirely on local error does not factor stability constraints. On stiff problems, an explicit method may still be forced to very small h even when the estimated local error is below the fixed tol. In these cases, the adaptive scheme tends to push h into the stability region discussed in earlier stages. This lends itself to a high computation cost despite an adaptive update.

8.2 Algorithms

We use step doubling with respect to RK2.

Step Doubling Error Estimate See theory section.

Step Acceptance/Rejection and Stepsize Update See theory section.

Adaptive General Method Loop

1. Initialize t_0, y_0, initial stepsize h, and final time T.
2. While $t_n < T$:
   (a) Attempt a step using the step doubling error estimation.
   (b) Verify the acceptance criterion and if pass, update h.
   (c) If the step is accepted, append (t_{n+1}, y_{n+1}) and continue.
3. Stop once $t_n \geq T$; interpolate the final step if needed.

8.3 Experiments

This stage tests adaptive step size control based on step doubling and an embedded RK23 pair. We use a non-stiff scalar Example A and the van der Pol oscillator (Example H) at two values of μ to demonstrate how adaptive step sizing responds to varying stiffness.

Recall Example H,

$$x' = v, \qquad v' = \mu(1 - x^2)v - x,$$

with initial conditions $(x(0), v(0)) = (1, 0)$ on $[0, 40]$. We consider both $\mu = 1$ (mildly nonlinear) and $\mu = 10$ (more stiff).

Adaptive RK2 with step doubling We first consider adaptive RK2 (step doubling) with respect to Example A. We compare it to a fine reference solution, plot accepted step sizes and deliver a histogram of local errors. The specified tolerance is $\text{tol} = 10^{-4}$.

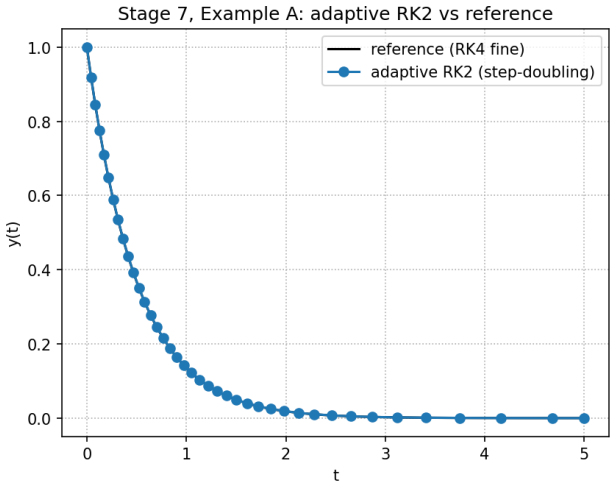

Figure 28: Example A: adaptive RK2 (step doubling) vs. RK4 reference.

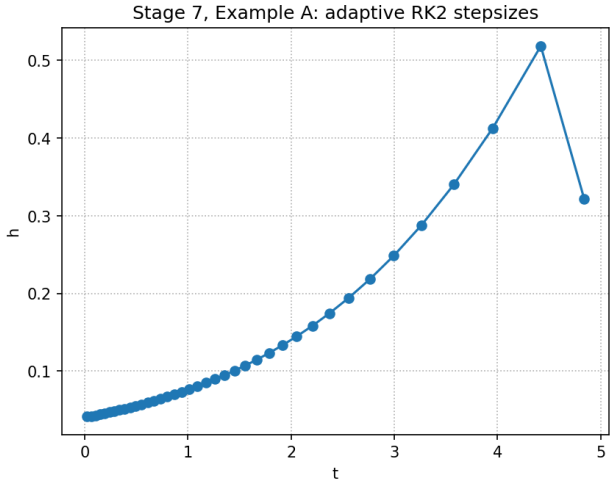

Figure 29: Example A: accepted stepsizes for adaptive RK2.

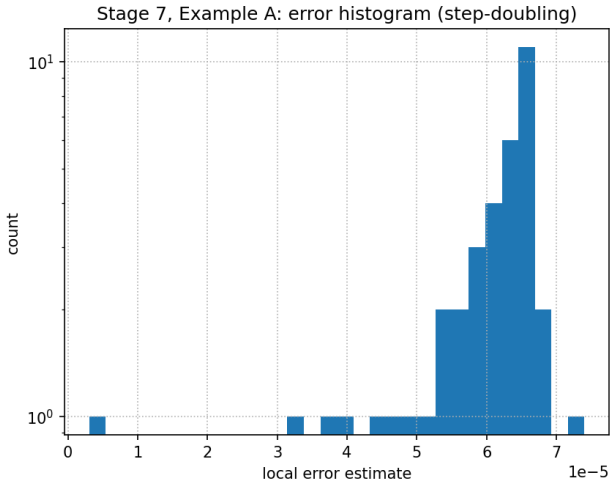

Figure 30: Example A: histogram of local error estimates.

Then, we run the same adaptive RK2 scheme on Example H for $\mu = 1$ and $\mu = 10$ with the same tol and an initial $h = 0.05$. In both cases, we plot the x-component against the fine reference, along with the corresponding step sizes. For $\mu = 10$, there is also a histogram of local error estimates.

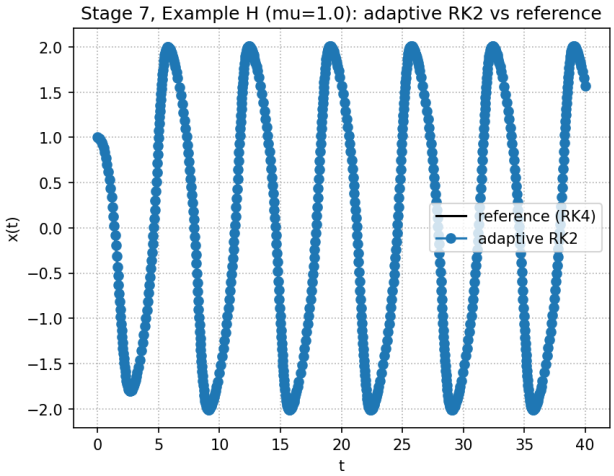

Figure 31: Example H ($\mu = 1$): adaptive RK2 (step doubling) solution v. fine reference.

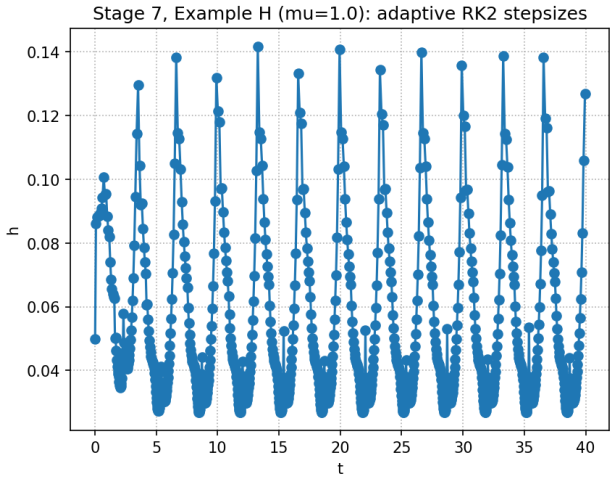

Figure 32: Example H ($\mu = 1$): accepted step sizes for adaptive RK2.

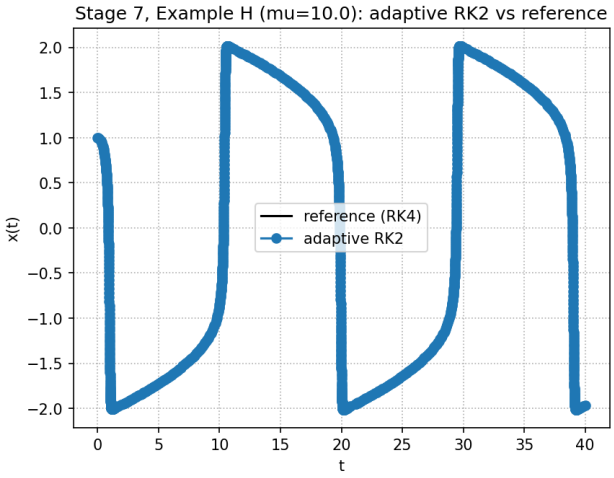

Figure 33: Example H ($\mu = 10$): adaptive RK2 (step doubling) solution v. fine reference.

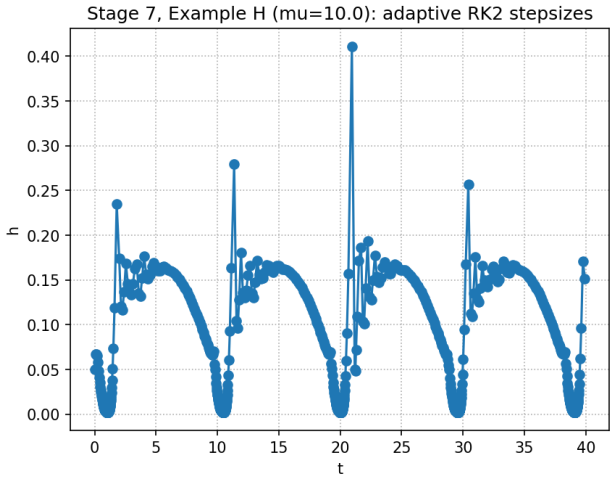

Figure 34: Example H ($\mu = 10$): accepted step sizes for adaptive RK2.

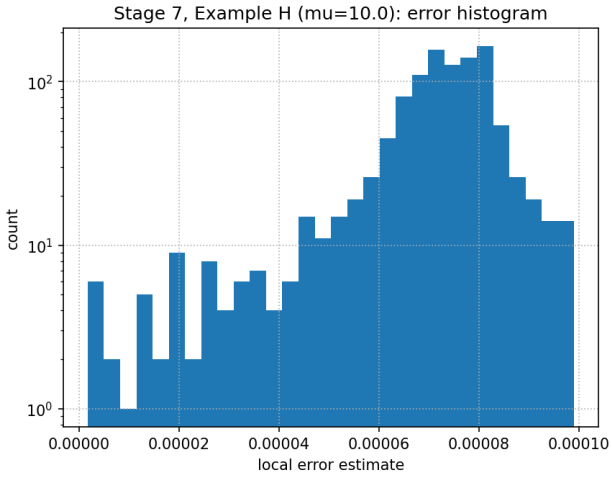

Figure 35: Example H ($\mu = 10$): histogram of local error estimates for adaptive RK2.

Global error and work versus tolerance for Example H We repeat the adaptive RK2 step doubling run on Example H with $\mu = 10$ for a range of tolerances tol $\in \{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$. For each tolerance, compute the final global error relative to the RK4 reference and record the total work.

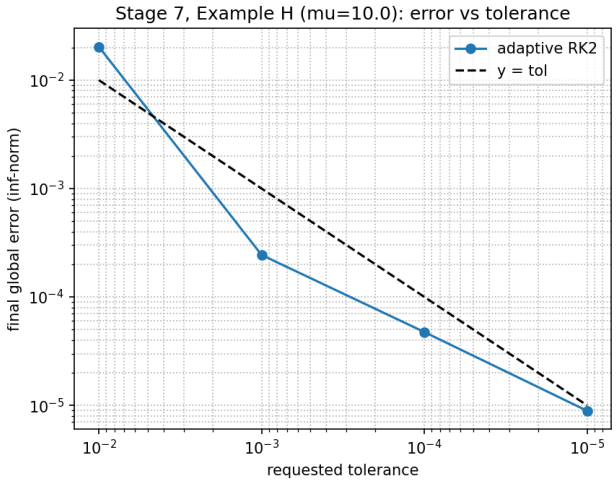

Figure 36: Example H ($\mu = 10$): final global error vs. local tolerance for adaptive RK2 (step doubling).

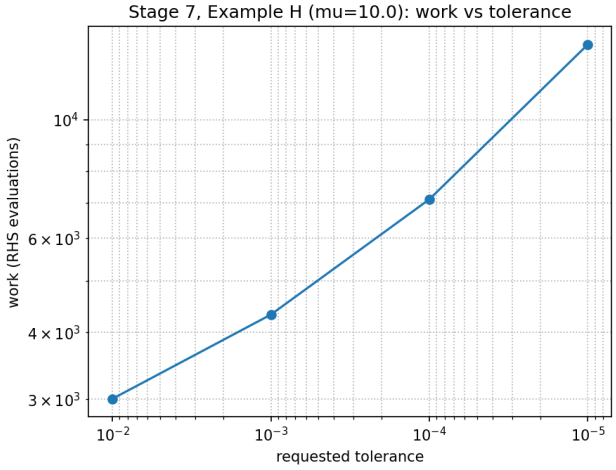

Figure 37: Example H ($\mu = 10$): work vs. local tolerance for adaptive RK2.

Embedded RK23 on Example A instead of step doubling, we use an embedded Runge-Kutta 23 method with respect to Example A. We report the same plots as before.

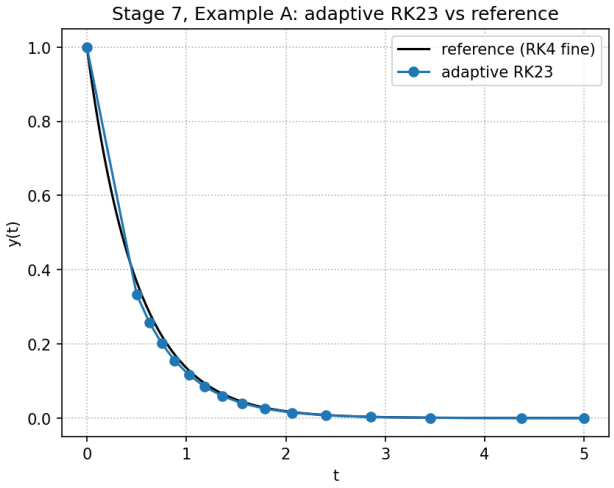

Figure 38: Example A: adaptive RK23 solution compared with fine RK4 reference.

Additional plots for the RK23 stepsize history and error histogram, as well as further diagnostic plots for Example H, are included in the Additional Plots appendix.

8.4 Results and Discussion

The adaptive RK2 step-doubling method performs well on the smooth, nonstiff Example A. With a specified local tolerance of 10^{-4}, the final error is very small using only 246 RHS computations, and the step size history shows growth as the solution decays. The local error histogram is peaked near the specified tol, demonstrating that the controller is regulating the step size effectively and consistently. The embedded RK23 run is more efficient, achieving comparable accuracy with only 72 RHS computations. This shows the idea that when error estimation is built into the scheme, there is a reduction of computation.

For Example H, the adaptive behavior is more clear. At $\mu = 1$, the local oscillations of the van der Pol system lead to a more dramatic range in the accepted step sizes. Consequently, the method requires 5172 RHS computations to maintain a local tol of 10^{-4}. The final error shows the lack of uniform difficulty and the fact that the solution has periods of moderate stiffness. Increasing to $\mu = 10$ produces sharper oscillations and stiffer dynamics. This forces the method to accept more very small steps, raising the total work to 7098 RHS computations. The final error is still quite low and the histogram demonstrates the controller's ability to regulate step size even with more dynamic stiffness.

The tolerance sweep for $\mu = 10$ shows a relationship between the prescribed local tolerance and the final global error. As the tolerance is tightened from 10^{-2} to 10^{-5}, the global error decreases in a similar linear fashion. The corresponding work grows monotonically. This correlation makes sense in the theory of an adaptive method's ability to react to changing stiffness and enforced accuracy. In general, the experiments show that adaptive step size control is quite effective for mild and moderately stiff problems. At the same time, explicit adaptive schemes still require significant computational cost when stiffness forces a small stability region. We also see that embedded RK methods outperform step doubling since they have a smaller local error estimation overhead.

9 Stage 8: Linear Multistep Methods (AB/AM), Zero-Stability (Root Condition), and Starting Values

9.1 Theory

Linear multistep methods (LMMs) advance an approximate solution of the linear test problem using several past approximations. For a fixed step size h , a k-step LMM has the form

$$\sum_{j=0}^{k} \alpha_j y_{n+j} \;=\; h \sum_{j=0}^{k} \beta_j f_{n+j},$$

where $f_{n+j} = f(t_{n+j}, y_{n+j})$ and (α_j, β_j) are fixed coefficients. Explicit methods have $\beta_k = 0$, while implicit methods have $\beta_k \neq 0$. Because the recurrence involves $y_n, \ldots, y_{n+k-1}$, a LMM requires starting values. In practice, this can be determined by a one-step method like RK4.

Consistency and LTE We define the local truncation error

$$\tau_{n+k} := \sum_{j=0}^{k} \alpha_j y(t_{n+j}) \;-\; h \sum_{j=0}^{k} \beta_j y'(t_{n+j}).$$

A LMM is consistent if $\tau_{n+k} = O(h^{p+1})$ for some $p \geq 1$. Equivalently, consistency imposes two conditions

$$\sum_{j=0}^{k} \alpha_j = 0, \qquad \sum_{j=0}^{k} j\alpha_j = \sum_{j=0}^{k} \beta_j.$$

When these hold, the method achieves global order p.

Zero–stability and the root condition The first characteristic polynomial of the LMM is

$$\rho(\zeta) = \sum_{j=0}^{k} \alpha_j \zeta^j.$$

Zero–stability states that all roots of $\rho(\zeta)$ satisfy

$$|\zeta| \leq 1 \quad \text{and any root on the unit circle is simple.}$$

This root condition ensures that errors in starting values do not diverge as the recurrence is iterated. A consistent and zero–stable LMM is convergent of order p.

Two-step Adams methods Adam methods are generated by approximating $f(t, y(t))$ by taking the integral of an interpolating polynomial over previous time levels.

AB2 (explicit). Interpolating f through (t_n, f_n) and (t_{n+1}, f_{n+1}) and integrating over $[t_{n+1}, t_{n+2}]$ yields

$$y_{n+2} = y_{n+1} + h\left(\tfrac{3}{2} f_{n+1} - \tfrac{1}{2} f_n\right),$$

which is explicit and has LTE $O(h^3)$ with global order 2.

AM2 (implicit). Including $f_{n+2} = f(t_{n+2}, y_{n+2})$ gives

$$y_{n+2} = y_{n+1} + \frac{h}{12}\left(5f_{n+2} + 8f_{n+1} - f_n\right),$$

which is implicit and also globally second order. Because f_{n+2} depends on the unknown y_{n+2}, we require a nonlinear (ex. Newton's) solve at every iteration.

The characteristic polynomial of both AM2 and AB2 is

$$\rho(\zeta) = \zeta^2 - \zeta,$$

with roots 0 and 1. This shows that both methods satisfy the root condition and are zero–stable.

Starting values Because two-step methods require y_n and y_{n+1} prior to the run, we compute y_{n+1} from y_0 using a one–step method. Here we choose a fine RK4 reference to compute y_{n+1} since the higher order method provides a higher accuracy and limits error due to starting values.

9.2 Algorithms

For both AB2 and AM2, we require y_1 to start the scheme, and generate it using a fine RK4 reference. See previous stages for RK4 algorithm.

AB2 (explicit two–step method) Given (t_n, y_n) and (t_{n+1}, y_{n+1}):

1. Evaluate $f_n = f(t_n, y_n)$ and $f_{n+1} = f(t_{n+1}, y_{n+1})$.
2. Update
$$y_{n+2} = y_{n+1} + h\left(\tfrac{3}{2}f_{n+1} - \tfrac{1}{2}f_n\right).$$

AM2 (implicit two–step method) Given (t_n, y_n) and (t_{n+1}, y_{n+1}):

1. Form the implicit equation
$$y_{n+2} = y_{n+1} + \frac{h}{12}\left(5f(t_{n+2}, y_{n+2}) + 8f_{n+1} - f_n\right).$$
2. Solve for y_{n+2} using Newton iteration.

9.3 Experiments

We compare a two–step explicit Adams–Bashforth method (AB2) and a two–step implicit Adams–Moulton method (AM2) on the test Example A and consider the zero-stability test, convergence, and the influence of starter values on LMMs.

Zero–stability test on $y' = 0$ We run AB2 and AM2 on the trivial ODE

$$y'(t) = 0, \qquad y(0) = y_0 \neq 0,$$

with a fixed step size $h = 0.1$ over $N = 500$ steps. Because the exact solution is constant, a zero–stable LMM should produce a numerical solution whose magnitude remains uniformly bounded as n grows.

For AM2, we use one AM2 step (with a single Newton iteration) to generate y_1.

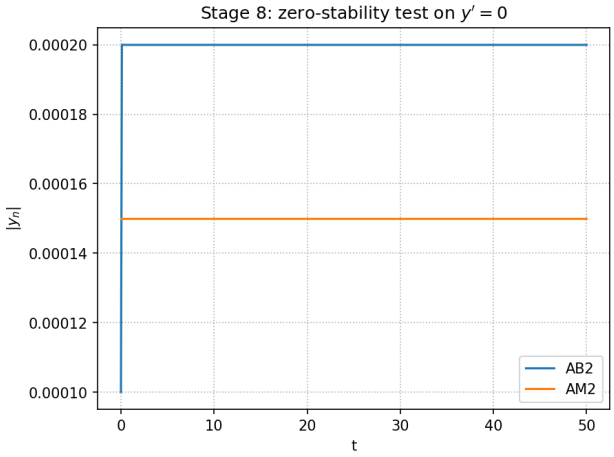

Figure 39: Zero-stability experiment for AB2 and AM2 ($h = 0.1$).

Order test on Example A Recall Example A,

$$y'(t) = -2y(t), \qquad y(0) = 1, \qquad 0 \leq t \leq 5,$$

with exact solution $y(t) = e^{-2t}$. For each

$$h \in \{0.20,\ 0.10,\ 0.0625,\ 0.04\},$$

we compute $N = (T - t_0)/h$ steps and measure the final-time global error.

In order to isolate the order of the multistep methods, we use the same high–order starter, a single step RK4 starter.

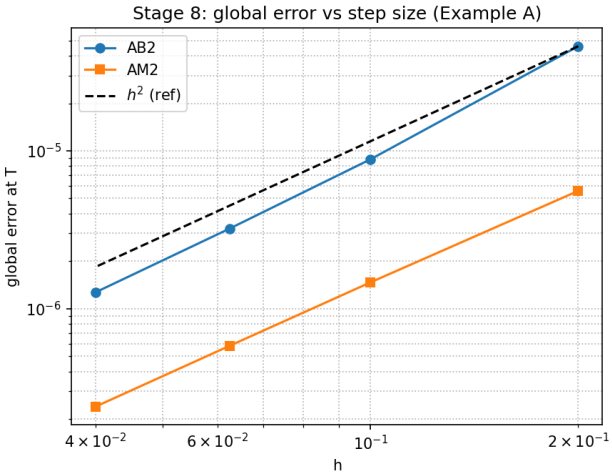

Figure 40: Global error at $T = 5$ vs. h for AB2 and AM2 (Example A).

Starting Value Analysis We use AB2 to explore the impact of low order starters.

- High–order starter: y_1 derived from one RK4 step (order 4).
- Low–order starter: y_1 derived from one Forward Euler step (order 1).

We record the local error pointwise which demonstrates the short–term pollution from a low–quality starter and how zero–stability pushes the LMM back to accuracy.

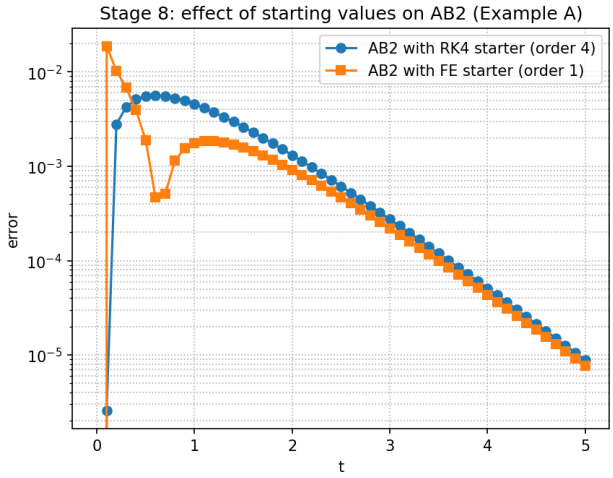

Figure 41: Error of AB2 with RK4 vs. FE starters

We then report the final time errors of both starters. The small difference between the starters demonstrates the power of zero-stability.

Table 6: Starter comparison for AB2 on Example A with $h = 0.1$.

starter	final_error
RK4	8.834×10^{-6}
FE	7.754×10^{-6}

9.4 Results and Discussion

The zero–stability experiment confirms the theory. Both AB2 and AM2 produce bounded horizontal sequences with respect to n despite the pollution induced by inaccurate starters. This is the expected behavior of a zero–stable method. Furthermore, this demonstrates that neither scheme has numerical growth on a problem with constant exact solution. In practice, this test is a strong empirical validation of the stability of both methods.

The convergence experiment on Example A shows that, when supplied with a sufficiently accurate starter, both AB2 and AM2 perform at second-order global accuracy. AM2 exhibits a noticeably smaller error constant than AB2. This is consistent with the fact that AM2 is an implicit method. Alternatively, AB2 is explicit and consequently requires stricter restrictions on h for stiff or moderately stiff systems. Regardless, the overall slopes match the predicted order$p = 2$. This confirms the theoretical error bounds for stable LMMs.

Finally, the starter experiment shows that although the global order is determined by the multistep scheme, a low–order starter can pollute the early portion of the solution. With a forward Euler starter, AB2 has larger transient errors initially and requires many steps before the theoretical h^2 regime dominates. In contrast, the RK4 starter suppresses this error effectively, making the global error order more obvious. This is a limited consideration since the final-time errors of both starters are similar. This confirms the idea that LMMs should be started with a method of higher order in order to minimize initial error.

10 Synthesis and Lessons

Across all eight stages of this project, the question of how numerical ODE solvers succeed or fail depending on problem structure, stability properties, and implementation choices becomes clear. The first few stages show that explicit methods such as forward Euler and RK schemes provide accurate and cost effective solutions when there is an underlying nonstiff problem and a sufficiently small h is chosen. This result is limited since stability-imposed restrictions dominate for problems with rapidly decaying or stiff dynamics. These observations motivate the following section on implicit methods, which maintain stability in more general cases and promote accuracy based step size choice.

Later stages further this idea by introducing adaptive step-sizing and multistep methods. Adaptive step-size control shows how local error estimation can significantly improve efficiency. The continued trade off of explicit schemes is still shown here since stiff problems still force small steps. The stage on linear multistep methods demonstrates their potential efficiency while considering the sensitivity to proper starter choice. This reinforces the idea of zero-stability in ensuring reliable asymptotic behavior. Ultimately, the experiments and theory show that effective ODE solvers depend on a careful combination of method, problem stiffness, error tolerance, and stability structure. This decision making process motivates an understanding of method, parameter, and implementation choice when considering numerical solvers for a wide range of ODEs.

A Implementation Notes

All numerical experiments in this project were implemented in Python using a modular structure. Each time-stepping method was implemented in its own function, with a consistent function declaration. Similarly, each example ODE was encoded into its own function in order to be reproducible across plots, parameters, and stages.

Wherever possible, I vectorized solvers to work with scalar or vector-valued ODEs in place. Implicit schemes rely on Newton's method. Starting values for multistep methods were generated with RK4 unless intentionally degraded to Forward Euler to demonstrate the effects of poor initialization.

All diagnostic figures were produced with `matplotlib` and most of the computational lifting was done with `numpy`. To ensure reproducibility, a single script (`reproduce_all.py`) reproduces every figure in order.

GitHub: https://github.com/amittha1/math269a

B Additional Figures

Stage 2 .

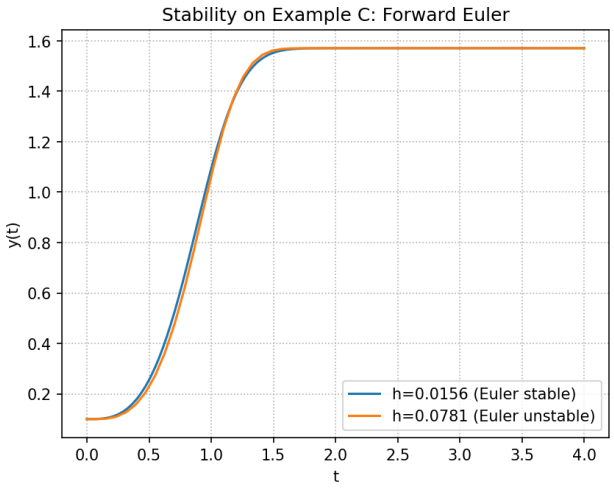

Figure 42: Stability of Euler for Example C

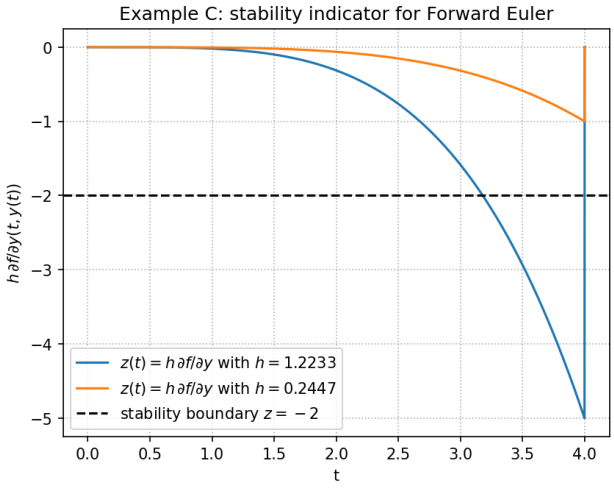

Figure 43: Stability Indicator of Euler for Example C

Stage 4 For the sake of brevity and due to the sufficient analysis possible with these plots, I have chosen not to include all 20+ required plots in the document. Although considered, I have chosen to leave the remainder entirely off the report. All plots can be found in the github and locally submitted zip file.

Stage 7 .

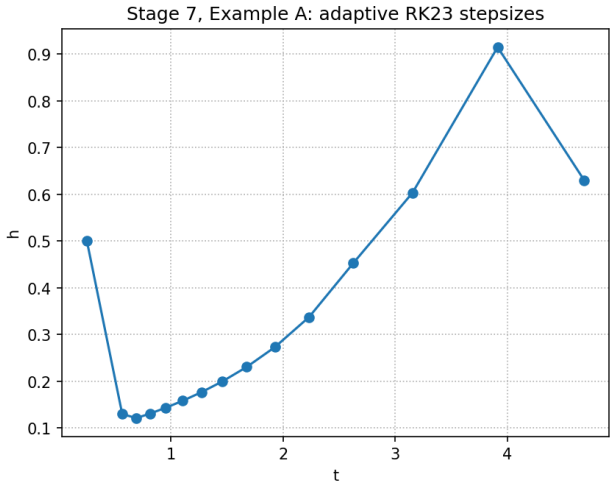

Figure 44: Example A: adaptive embedded RK23 step sizes

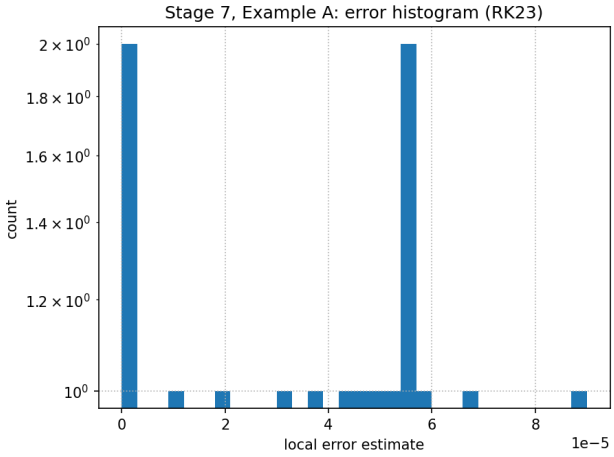

Figure 45: Example A: adaptive embedded RK23 local error histogram

For the sake of brevity, the additional plots for step doubling can be found in the attached files and github.