Adaptive Time Stepping

Luke Sellmayer

APPM 4610

April 29, 2024

Why Vary Step Size?

- By making the step size as small as possible, we reduce local truncation error (and therefore global error)
- But, smaller step size means more mesh points, meaning more computational work
- By varying the step size, we can minimize the number of mesh points used while keeping the local truncation in check
- I.e. how can we get the most bang for our buck?

Variable Step Size for One Step Taylor Methods

$$w_0 = \alpha$$
$$w_{i+1} = w_i + h\phi(t_i, w_i, h)$$

Local Truncation Error (LTE)

$$\tau_{i+1}(h) = \frac{y(t_{i+1}) - y(t_i)}{h} - \phi(t_i, y_i, h)$$

- Ideally, we'd have $|\tau_{i+1}(h)| < \epsilon$ at each mesh point
- But, we don't know the exact solution $y(t)$
- How can we approximate $\tau_{i+1}(h)$?
- Solution: use two separate Taylor methods of different orders

Approximating LTE

nth order Taylor method

$$w_0 = \alpha$$
$$w_{i+1} = w_i + h\phi(t_i, w_i, h)$$

$$\tau_{i+1}(h) = O(h^n)$$
$$*$$
$$\approx \frac{1}{h}(y(t_{i+1}) - w_{i+1})$$

$(n+1)$th order Taylor method

$$\tilde{w}_0 = \alpha$$
$$\tilde{w}_{i+1} = \tilde{w}_i + h\tilde{\phi}(t_i, \tilde{w}_i, h)$$

$$\tilde{\tau}_{i+1}(h) = O(h^{n+1})$$
$$*$$
$$\approx \frac{1}{h}(y(t_{i+1}) - \tilde{w}_{i+1})$$

*assuming $y(t_{i+1}) \approx w_{i+1}, y(t_{i+1}) \approx \tilde{w}_{i+1}$

Approximating LTE

nth order Taylor method

$$\tau_{i+1}(h) = \frac{1}{h}(y(t_{i+1}) - w_{i+1})$$

$$= \frac{1}{h}(y(t_{i+1}) - \widetilde{w}_{i+1} + \widetilde{w}_{i+1} - w_{i+1})$$

$$= \tilde{\tau}_{i+1}(h) + \frac{1}{h}(\widetilde{w}_{i+1} - w_{i+1})$$

$$\Rightarrow \tau_{i+1}\ (h) \approx \frac{1}{h}(\widetilde{w}_{i+1} - w_{i+1})$$

$(n+1)$th order Taylor method

$$\tilde{\tau}_{i+1}(h) = \frac{1}{h}(y(t_{i+1}) - \widetilde{w}_{i+1})$$

*since $\tilde{\tau}_{i+1}(h) = O(h^{n+1})$ while $\tau_{i+1}(h) = O(h^n)$

Varying the Step Size

- Given previous step size h_i, construct $h_{i+1} = qh_i$
- Goal: determine q such that:

$$|\tau_{i+1}(h_{i+1})| = |\tau_{i+1}(qh_i)| \leq \epsilon$$

$$\overset{*}{=}$$

$$= |C(qh_i)^n| \leq \epsilon$$

$$= |q^n(Ch_i^n)| \leq \epsilon$$

$$\overset{*}{\approx}$$

$$\approx |q^n \tau_{i+1}(h_i)| \leq \epsilon$$

*assuming $\tau_{i+1}(h) = Ch^n$ since $\tau_{i+1}(h) = O(h^n)$

Varying the Step Size

$$|\tau_{i+1}(h_{i+1})| \approx |q^n \tau_{i+1}(h_i)| \leq \epsilon$$

$$\overset{*}{\approx} \left| q^n \frac{\widetilde{w}_{i+1} - w_{i+1}}{h} \right| \leq \epsilon$$

*from the approximation $\tau_{i+1}(h) \approx \frac{1}{h}(\widetilde{w}_{i+1} - w_{i+1})$

Varying the Step Size

$$|\tau_{i+1}(h_{i+1})| \approx \left| q^n \frac{\widetilde{w}_{i+1} - w_{i+1}}{h} \right| \leq \epsilon$$

So, we choose q such that*

$$q \leq \left(\frac{\epsilon h_i}{|\widetilde{w}_{i+1} - w_{i+1}|} \right)^{\frac{1}{n}}$$

*algebra!

Varying the Step Size

$$q \leq \left(\frac{\epsilon h_i}{|\widetilde{w}_{i+1} - w_{i+1}|} \right)^{\frac{1}{n}}$$

In summary:

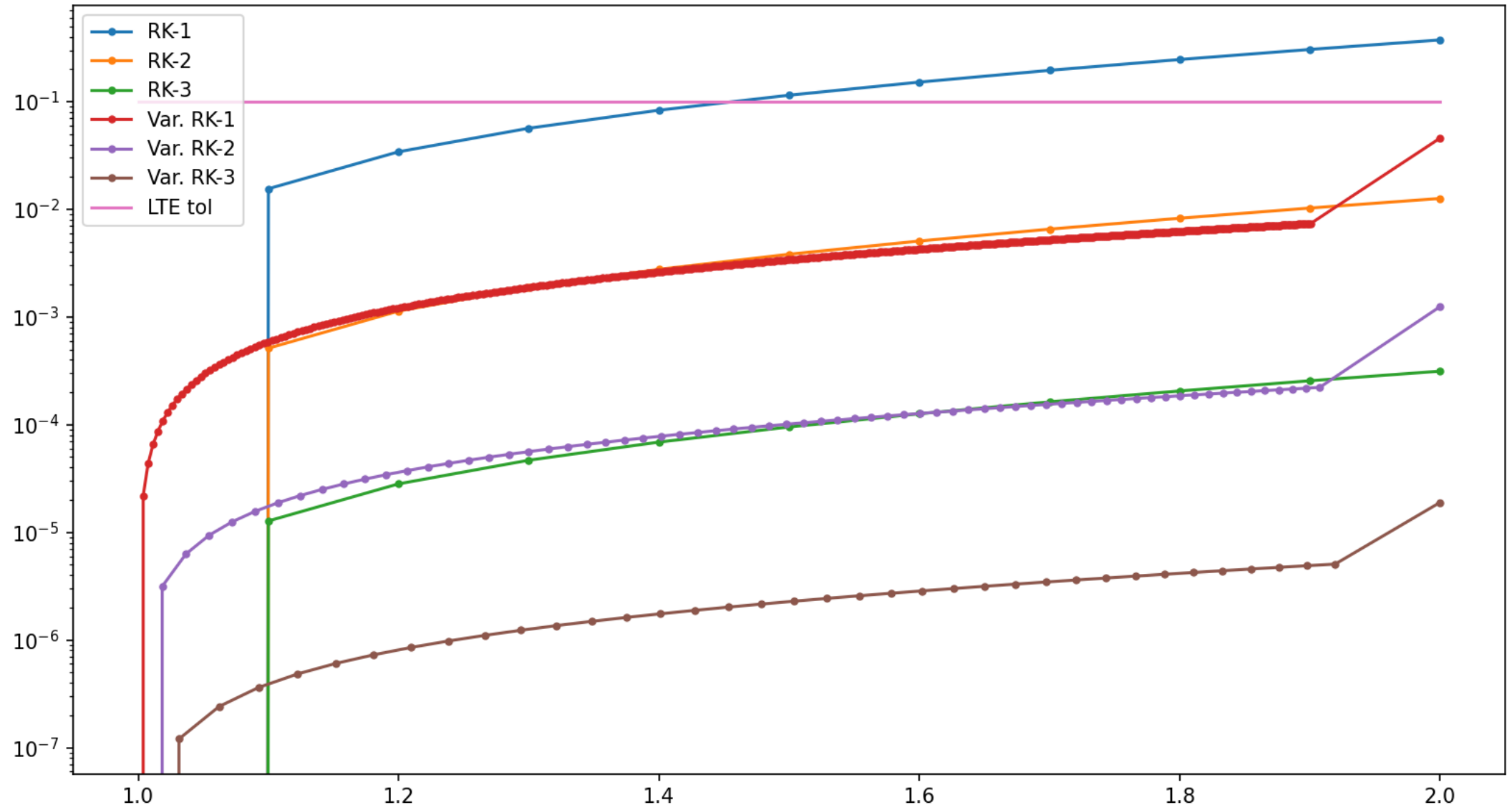
- Use nth and $(n+1)$th order Taylor methods to create w_{i+1} and $\widetilde{w}_{i+1}$ (respectively) using h_i
- Choose a desired tolerance ϵ
- Recalculate w_{i+1} with the nth order Taylor method, this time using $h_{i+1} = qh_i$

Example: Runge-Kutta Methods

$$1 \leq t \leq 2$$
$$y' = t + y$$
$$y(1) = 1$$

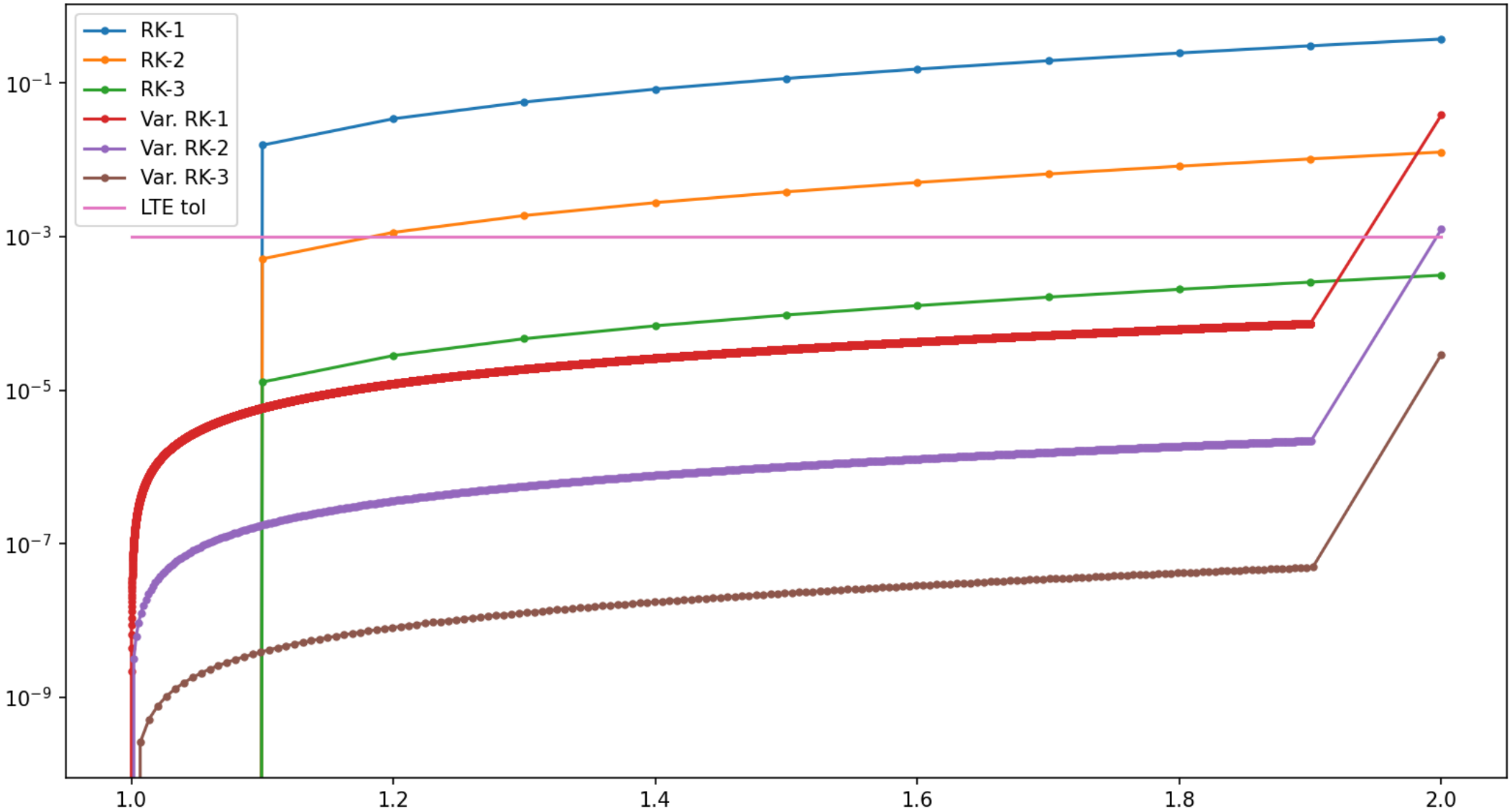
Exact solution: $y = 3e^{t-1} - t - 1$

Example: Runge-Kutta Methods



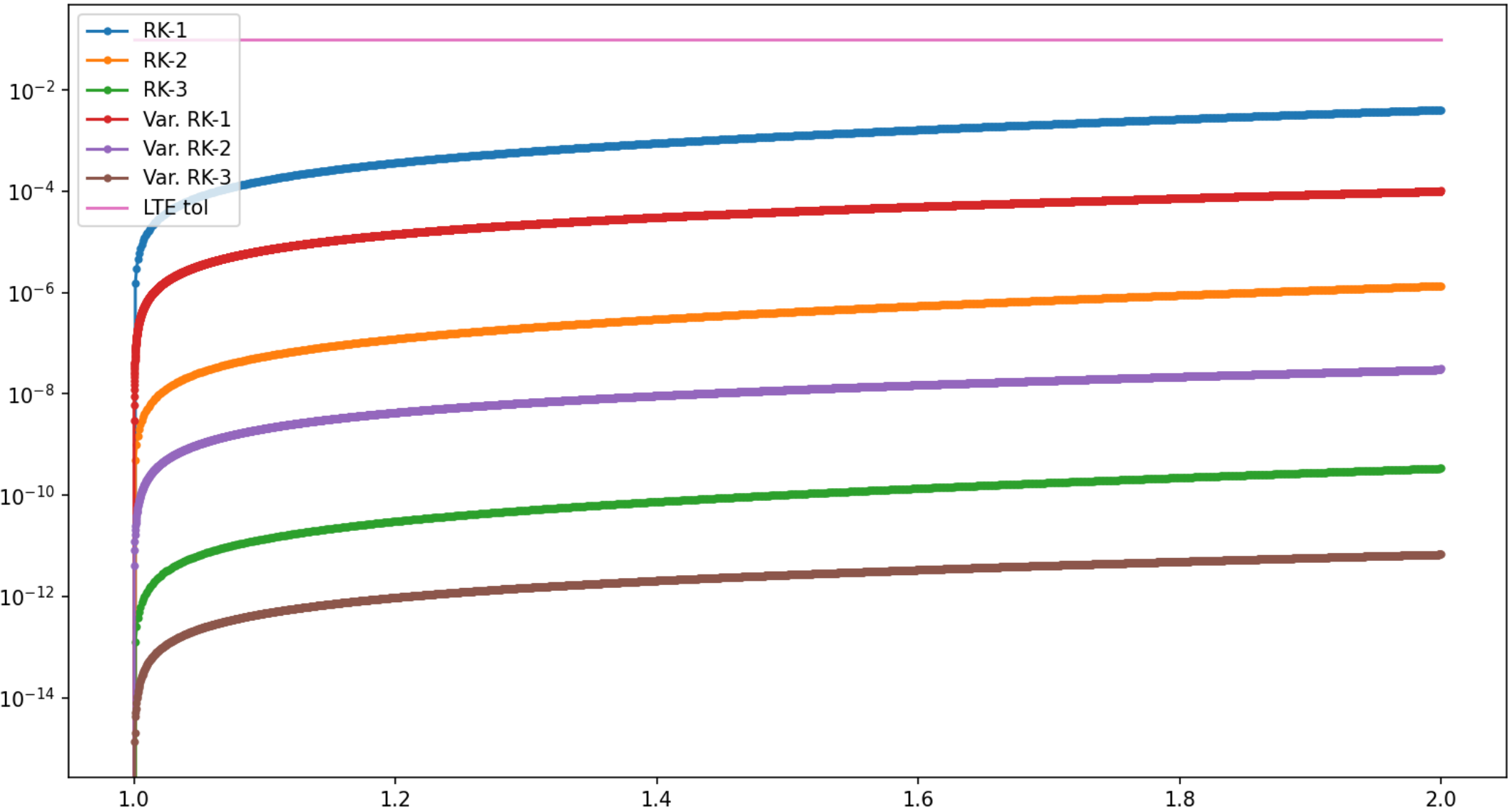
$\epsilon = 0.1$
$h_0 = 0.1$

Example: Runge-Kutta Methods



$\epsilon = 0.001$
$h_0 = 0.1$

Example: Runge-Kutta Methods



$\epsilon = 0.001$
$h_0 = 0.001$

Independent Extension: The DASSL Algorithm

Overview

The **D**ifferential / **A**lgebraic **S**ystem **S**o**L**ver (DASSL) approximates solutions to implicit systems of differential/algebraic equations of the form

$$\vec{F}(t, \vec{y}, \vec{y}') = \vec{0}$$
$$\vec{y}(t_0) = \vec{y}_0$$
$$\vec{y}'(t_0) = \vec{y}'_0$$
$$t_0 \leq t \leq t_f$$

Why?

- May not be able to explicitly solve for $\vec{y}' = f(t, \vec{y})$, or may be impractical to do so

Overview

$$\vec{F}(t, \vec{y}, \vec{y}') = \vec{0}$$
$$\vec{y}(t_0) = \vec{y}_0$$
$$\vec{y}'(t_0) = \vec{y}'_0$$
$$t_0 \leq t \leq t_f$$

What's with the initial conditions?

- DASSL allows input of partial data for $\vec{y}'(t_0)$ if you have it, but it is not needed

Basic Algorithm Outline

1) Choose a step size h_j and order k_j
2) Create an initial guess for $\vec{w}^{(0)}{}_{j+1}$ using $\{t_0, \cdots, t_j\}$ and $\{\vec{w}_0, \cdots, \vec{w}_j\}$
    1) Done by evaluating the *predictor* polynomial, $\vec{\omega}_{j+1}^{P}(t)$, that interpolates the nodes $\left\{\left(t_{j-k_j-1}, \vec{w}_{j-k_j-1}\right), \cdots, \left(t_j, \vec{w}_j\right)\right\}$ at $t = t_{j+1}$
        1) $\vec{w}^{(0)}{}_{j+1} = \vec{\omega}_{j+1}^{P}(t_{j+1})$
3) Improve the initial guess using Newton's method
    1) Run Newton's method on $\vec{F}\left(t_{j+1}, \vec{w}_{j+1}, \alpha\vec{w}_{j+1} + \vec{\beta}\right)$ with initial guess $\vec{w}^{(0)}{}_{j+1}$ to obtain the final approximation
4) Adjust h_j and k_j so that the final approximation $\vec{w}_{j+1}$ satisfies error bounds

Basic Algorithm Outline

Steps 1) and 4): choosing/adjust h_j and k_j

- $\vec{w}_{j+1}$ needs to satisfy both:
  - LTE bounds
  - Interpolating polynomial $\left(\vec{\omega}_{j+1}^{I}(t)\right)$ error bounds
    - $\vec{\omega}_{j+1}^{I}(t) = \vec{w}_{j+1} + (t - t_{j+1})[\vec{w}_{j+1}, \vec{w}_j] + (t - t_{j+1})(t - t_j)[\vec{w}_{j+1}, \vec{w}_j, \vec{w}_{j-1}] + \cdots + (t - t_{j+1})(t - t_j) \cdots \left(t - t_{j-k_j+2}\right)\left[\vec{w}_{j+1}, \vec{w}_j, \cdots \vec{w}_{j-k_j+1}\right]$
  - General adjustments:
    - Increase k_j and h_j for the first few iterations
    - Decrease h_j after an iteration fails
    - Continue to decrease k_j and h_j after multiple iteration failures.
    - After any successful iteration, increase k_j and h_j

Basic Algorithm Outline

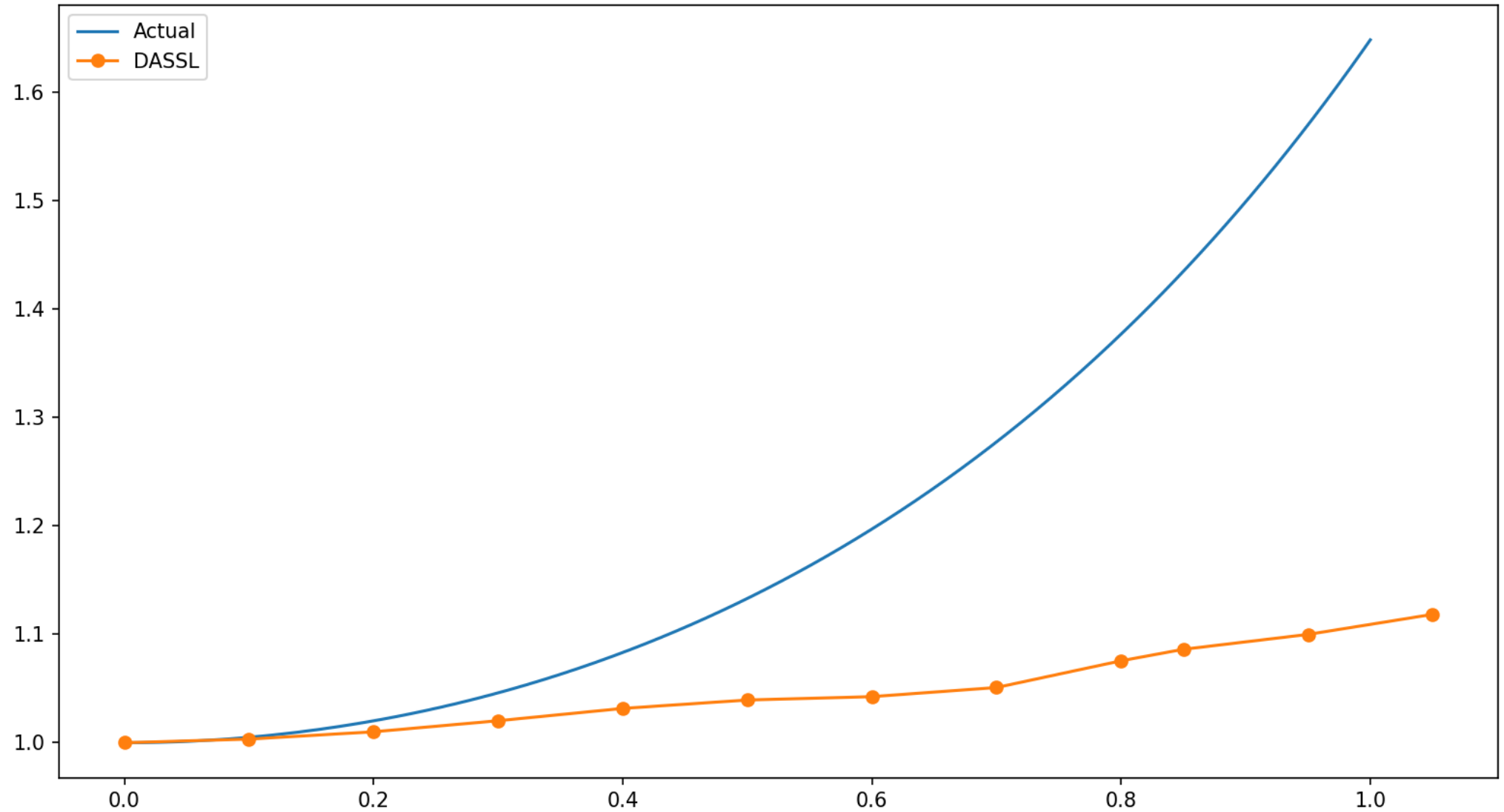
Step 2): creating $\vec{w}^{(0)}{}_{j+1}$

- Also constructed using Newton divided differences

Step 3) Running Newton's method

- Instead of running the method on $\vec{F}\left(t_{j+1}, \vec{w}_{j+1}, \vec{w}'_{j+1}\right)$, we approximate $\vec{w}'_{j+1}$ as $\alpha\vec{w}_{j+1} + \vec{\beta}$
- α and $\vec{\beta}$ chosen according to order k_j, which approximate $\vec{w}'_{j+1}$ using the k_jth order backwards differentiation formula
  - e.g for $k_j = 1$, we get the backward Euler method
    - $\vec{w}_{j+1} - \vec{w}_j = h_j\, \vec{w}'_{j+1}$

Some Disappointing Results :(



What's Going Wrong?

Elements that definitely work

- Newton's method implementation
- Creating the interpolating polynomials
- Creating initial guesses
- Adjusting k_j and h_j based on errors

Elements that probably don't work

- Finding the LTE
- Finding the interpolation error

References

[1] K. E. Brenan, S. L. Campbell, and L. R. Petzold, Numerical solution of initial-value problems in differential-algebraic equations, 14, SIAM, 1996.

[2] Richard L. Burden, J. Douglas Faires, and Annette M. Burden, Numerical analysis, tenth edition, Cengage Learning, 2014.

[3] Fred T. Krogh, Recurrence relations for computing with modified divided differences, Mathematics of Computation 33 (1979), no. 148, 1265–1271.

[4] Linda Petzold, A description of dassl: A differential/algebraic system solver, 01 1982.