Numerical ODE Methods with Variable Step Size

Luke Sellmayer

Applied Mathematics, University of Colorado Boulder

May 7th, 2024

Abstract: By varying the step sizes used in numerical methods that approximate solutions to ODEs and systems of ODEs, local truncation error can be bounded while minimizing the amount of computational work performed. It is shown that variable step size Runge-Kutta methods of order n can reach the same levels of accuracy compared to their constant step size order $n+1$ counterparts. The DASSL algorithm, which varies both step size and the order of the backwards differentiation formulas that it uses, is shown to be effective at solving systems of ODEs. The methods that DASSL uses to choose step size and order are discussed and implemented.

1. Introduction

When using finite difference methods that produce approximations $\{w_0, w_1, ..., w_N\}$ to the solution $y(t)$ of the initial-value problem

$$
\begin{aligned}
y'(t) &= f(t, y) \\
y(a) &= \alpha \\
a \leq t &\leq b,
\end{aligned}
\tag{1}
$$

at mesh points $\{t_0, t_1, ..., t_N\}$ where $w_i \approx y(t_i)$, the ideal method would be able to use a minimum number of mesh points N such that for a given tolerance $\epsilon > 0$, the global error at each mesh point would not be greater than ϵ, $|y(t_i) - w_i| \leq \epsilon$ [2]. In other words, the ideal method would give the most accurate approximation to the solution at all mesh points while minimizing the computational cost required, as each mesh point used requires additional function evaluations. However, these two goals directly contradict each other when using a fixed step size h; smaller values of h lead to lower global error but a higher number of mesh points, while larger values of h lead to a smaller number of mesh points but larger global error [2]. The solution, then, is to vary the step size at each mesh point, $h = h_i$, such that global error is minimized while using the fewest mesh points possible.

Generally, one cannot determine the global error for a given finite difference method. Luckily, there is a close connection between a method's global error and its local truncation error [2], which is much more easily predicted. By varying the step sizes to bound the local truncation error, the global error will be kept in check.

2. One-Step nth-Order Taylor Methods

A one-step difference method of the form

$$
\begin{aligned}
&w_0 = \alpha \\
&w_{i+1} = w_i + h\phi(t_i, w_i, h) \text{ for } i = 0, 1, ..., N-1
\end{aligned}
\tag{2}
$$

approximating the solution to (1) has a local truncation error given by

$$
\tau_{i+1}(h) = \frac{y(t_{i+1}) - (y(t_i) + h\phi(t_i, y(t_i), h))}{h} = \frac{y(t_{i+1}) - y(t_i)}{h} - \phi(t_i, y_i, h) \text{ for } i = 0, 1, ..., N-1. \tag{3}
$$

The goal is to bound the local truncation error at all mesh points, $|\tau_{i+1}(h)| \leq \epsilon$ for some ϵ, but the local truncation error cannot be determined without knowing the exact solution to (1). Instead, it will have to be approximated, which can be done by using two Taylor methods of differing orders [2].

Suppose there are two difference methods for approximating the solution to (1). The first is an nth order Taylor method

$$
\begin{aligned}
&w_0 = \alpha \\
&w_{i+1} = w_i + h\phi(t_i, w_i, h) \text{ for } i = 0, 1, ..., N-1.
\end{aligned}
\tag{4}
$$

with local truncation error $\tau_{i+1}(h) = O(h^n)$, which must be approximated. The second is an $(n+1)$th order Taylor method

$$
\begin{aligned}
&\widetilde{w}_0 = \alpha \\
&\widetilde{w}_{i+1} = \widetilde{w}_i + h\widetilde{\phi}(t_i, \widetilde{w}_i, h) \text{ for } i = 0, 1, ..., N-1.
\end{aligned}
\tag{5}
$$

with local truncation error $\widetilde{\tau}_{i+1}(h) = O(h^{n+1})$.

Assuming that $y(t_i) \approx w_i$ and $y(t_i) \approx \widetilde{w}_i$, this implies that

$$
\begin{aligned}
\tau_{i+1}(h) = \frac{y(t_{i+1}) - (y(t_i) + h\phi(t_i, y(t_i), h))}{h} &\approx \frac{y(t_{i+1}) - (w_i + h\phi(t_i, w_i, h))}{h} \\
&\approx \frac{1}{h}\left(y(t_{i+1}) - w_{i+1}\right)
\end{aligned}
$$

$$
\begin{aligned}
\widetilde{\tau}_{i+1}(h) = \frac{y(t_{i+1}) - (y(t_i) + h\phi(t_i, y(t_i), h))}{h} &\approx \frac{y(t_{i+1}) - (\widetilde{w}_i + h\phi(t_i, \widetilde{w}_i, h))}{h} \\
&\approx \frac{1}{h}\left(y(t_{i+1}) - \widetilde{w}_{i+1}\right).
\end{aligned}
$$

Further manipulation of $\tau_{i+1}(h)$ reveals that

$$
\begin{aligned}
\tau_{i+1}(h) &\approx \frac{1}{h}\left(y(t_{i+1}) - w_{i+1}\right) \\
&= \frac{1}{h}\left(y(t_{i+1}) - w_{i+1} + \widetilde{w}_{i+1} - \widetilde{w}_{i+1}\right) \\
&= \frac{1}{h}\left(y(t_{i+1}) - \widetilde{w}_{i+1} + \widetilde{w}_{i+1} - w_{i+1}\right) \\
&= \widetilde{\tau}_{i+1}(h) + \frac{1}{h}\left(\widetilde{w}_{i+1} - w_{i+1}\right)
\end{aligned}
$$

Using the fact that $\widetilde{\tau}_{i+1}(h)$ is only $O(h^{n+1})$ compared to τ_{i+1} which is $O(h^n)$ [2], the approximation is

$$\tau_{i+1}(h) \approx \frac{1}{h}\left(\widetilde{w}_{i+1} - w_{i+1}\right). \tag{6}$$

This approximation of $\tau_{i+1}(h)$ can now be used to determine how to vary the step size. The new step size h_{i+1} will be constructed as some multiple q of the previous step size h_i: $h_{i+1} = qh_i$. For some given tolerance ϵ, q must be found such that $|\tau_{i+1}(h_{i+1})| = |\tau_{i+1}(qh_i)| \leq \epsilon$. Assuming that $\tau_{i+1}(h) \approx Ch^n$ for some constant C since $\tau_{i+1}(h)$ is $O(h^n)$ [2]:

$$\begin{aligned}
\tau_{i+1}(h_{i+1}) &= \tau_{i+1}(qh_i) \\
&\approx C(qh_i)^n \\
&= q^n(Ch^h) \\
&\approx q^n\tau_{i+1}(h) \\
&\approx q^n\frac{\widetilde{w}_{i+1} - w_{i+1}}{h}
\end{aligned}$$

Therefore, to ensure that $|\tau_{i+1}(h_{i+1})| \leq \epsilon$, q must be chosen such that [2]

$$q \leq \left(\frac{\epsilon h}{\widetilde{w}_{i+1} - w_{i+1}h}\right)^{\frac{1}{n}}. \tag{7}$$

Computing both $\widetilde{w}_{i+1}$ and w_{i+1} requires twice as many function evaluations compared to using the method without a variable step size [2]. In order to make these extra evaluations worth while, the typical procedure is to first calculate w_{i+1} using the previous step size h_i, and then estimate its local truncation error $\tau_{i+1}(h_i)$. If $|\tau_{i+1}(h_i)| > \epsilon$, the calculation of w_{i+1} is recomputed with the updated step size $h_{i+1} = qh_i$ to ensure $|\tau_{i+1}(h_{i+1})| \leq \epsilon$ before moving to the next step. If instead $|\tau_{i+1}(h_i)| \leq \epsilon$, the value of w_{i+1} is accepted, and instead force an update to the step size for the next iteration.

Figure 1 shows the error plots when using both standard Runge-Kutta methods and variable step size Runge-Kutta methods of orders 1, 2, and 3 to solve

$$\begin{gathered}
y' = t + y \ \text{ for } 1 \leq t \leq 2 \\
y(1) = 1
\end{gathered} \tag{8}$$

with exact solution

$$y(t) = 3e^{t-1} - t - 1. \tag{9}$$

Using a variable method of order n leads to the same error as a nonvariable method of order $n+1$. For higher order methods, fewer mesh points are needed.

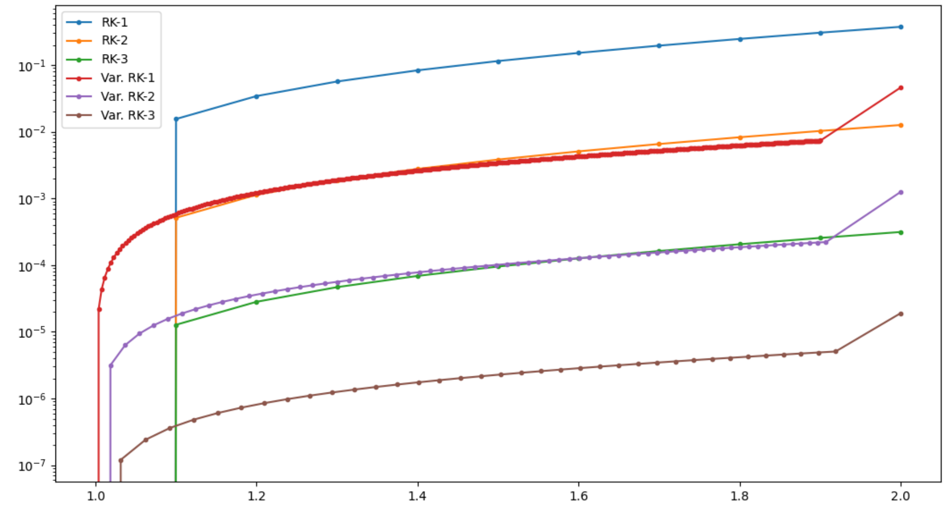

Figure 1. Errors of Fixed and Variable Step Size Runge-Kutta Methods

3. The DASSL Algorithm

The **D**ifferential **A**lgebraic **S**ystem **S**o**L**ver, or DASSL, is an algorithm created at Sandia National Laboratories developed to compute numerical solutions to ODE systems of the form

$$\begin{aligned} \vec{F}(t,\vec{y},\vec{y'}) &= \vec{0} \\ \vec{y}(t_0) &= \vec{y_0} \\ \vec{y'}(t_0) &= \vec{y'_0} \end{aligned} \tag{10}$$

where $\vec{F}$, $\vec{y}$, and $\vec{y'}$ are N dimensional vectors. DASSL was developed in order to solve ODE systems where it is impractical or impossible to solve for $\vec{y'} = f(t,\vec{y})$ [5].

3.1. **Algorithm Outline.** The basic idea of DASSL is to start by approximating $y'(t)$ at some mesh point $t = t_j$ using a variable order **B**ackwards **D**ifferentiation **F**ormula (BDF) [1], such as the order 1 (11) or order 2 (12) BDF's

$$\vec{y'}(t_{j+1}) \approx \vec{w'}_{j+1} = \frac{\vec{w}_{j+1} - \vec{w}_j}{h_j}, \tag{11}$$

$$\vec{y'}(t_{j+1}) \approx \vec{w'}_{j+1} = \frac{\frac{3}{2}\vec{w}_{j+1} - 2\vec{w}_j + \frac{1}{2}\vec{w}_{j-1}}{h_j}. \tag{12}$$

Here, $h_j = t_j - t_{j-1}$ is the step size and $\vec{w}_j$ is the approximation of the solution $\vec{y}(t)$ at $t = t_j$ [5]. Using approximations at previous time steps, DASSL creates an initial guess for the next time step, $\vec{w}^{(0)}_{j+1}$, and then runs Newton's method with this initial guess to solve

$$\vec{F}(t_{j+1}, \vec{w}_{j+1}, \vec{w'}_{j+1}) = 0 \tag{13}$$

for $\vec{w}_{j+1}$ [1]. At each time step, DASSL analyzes previous solution behavior to choose a step size h_j and order k_j, which are used in determining $\vec{w}_{j+1}^{(0)}$ and the order of the BDF used when approximating $\vec{y'}(t_{j+1})$ [5].

3.2. **Choosing the Initial Guess.** In order to use Newton's method, one must have initial guesses for $\vec{w}_{j+1}$ and $\vec{w'}_{j+1}$. DASSL creates these guesses by first choosing an order k_j to use for the BDF (how DASSL chooses k_j will be described later) in order to construct the *predictor polynomial*, $\vec{\omega}_{j+1}^P(t)$. This polynomial interpolates the solution at the previous $k+1$ approximations [1]. In other words,

$$\vec{\omega}_{j+1}^P(t_{j-i}) = \vec{w}_{j-i} \text{ for } i = 0, 1, ..., k. \tag{14}$$

The predictor polynomial is constructed using Newton divided differences as described by Krogh [4],

$$\vec{\omega}_{j+1}^P = \vec{w}_j + (t-t_j)[\vec{w}_j, \vec{w}_{j-1}] + (t-t_j)(t-t_{j-1})[\vec{w}_j, \vec{w}_{j-1}, \vec{w}_{j-2}] + \cdots + (t-t_j)\cdots(t-t_{j-k+1})[\vec{w}_j, \cdots, \vec{w}_{j-k}], \tag{15}$$

where

$$\begin{gathered} [\vec{w}_j] = \vec{w}_j, \\ [\vec{w}_j, \vec{w}_{j-1}, \cdots, \vec{w}_{j-k}] = \frac{[\vec{w}_j, \vec{w}_{j-1}, \cdots, \vec{w}_{j-k+1}] - [\vec{w}_{j-1}, \vec{w}_{j-2}, \cdots, \vec{w}_{j-k}]}{t_j - t_{j-k}}. \end{gathered} \tag{16}$$

The initial guesses for $\vec{w}_{j+1}$ and $\vec{w'}_{j+1}$ are $\vec{w}_{j+1}^{(0)}$ and $\vec{w'}_{j+1}^{(0)}$, which are created using this polynomial by evaluating it and its derivative at $t = t_{j+1}$ [1]:

$$\begin{gathered} \vec{w}_{j+1}^{(0)} = \vec{\omega}_{j+1}^P(t_{j+1}), \\ \vec{w'}_{j+1}^{(0)} = \omega'^P_{j+1}(t_{j+1}). \end{gathered} \tag{17}$$

3.3. **Implementation of BDF Formulas.** The BDF formulas in (11) and (12) assume that the step size h is constant between approximations. Since DASSL varies the step size at each time step ($h = h_j$), these must be adjusted to compensate for this. DASSL chooses to use a *fixed leading coefficient* implementation for these formulas as described by Jackson and Sacks-Davis [3]. Rather than solving (13) using the initial derivative approximation $\vec{w'}_{j+1}^{(0)}$, a modified equation, given by

$$\vec{F}(t_{j+1}, \vec{w}_{j+1}, \alpha\vec{w}_{j+1} + \beta) = 0, \tag{18}$$

is solved instead. α and β are chosen to account for the variable step size when using a BDF, and are given by

$$\alpha = \frac{-\alpha_s}{h_j}, \tag{19}$$

$$\beta = \vec{w'}_{j+1}^{(0)} - \alpha\vec{w'}_{j+1}^{(0)}, \tag{20}$$

where $\alpha_s = -\sum_{m=1}^{k} \frac{1}{m}$.

3.4. **Newton's Method.** Once an initial guess $\vec{w}_{j+1}^{(0)}$ has been created, approximation $\vec{w}_{j+1}$ can be found by solving (18) using Newton's method. DASSL uses a modified Newton iteration, given by

$$\vec{w}_{j+1}^{(i+1)} = \vec{w}_{j+1}^{(i)} - cG^{-1}\vec{F}(t_j, \vec{w}_{j+1}^{(i)}, \alpha\vec{w}_{j+1}^{(i)} + \vec{\beta}), \tag{21}$$

where

$$G = \alpha\frac{\partial\vec{F}}{\partial\vec{w}} + \frac{\partial\vec{F}}{\partial\vec{w}'} \tag{22}$$

which is approximated using finite differences [1] (c will be defined momentarily).

DASSL uses this modified Newton iteration for a few key reasons. Computing and inverting G at every time step is expensive for large systems. Many times, it is possible to reuse the matrix and α value from the previous time step, denoted as $\hat{G}$ and $\hat{\alpha}$, if G and $\hat{G}$ close enough. When this is done, c is chosen to speed up the rate of the convergence, and is given by

$$c = \frac{2\hat{\alpha}}{\alpha + \hat{\alpha}}. \tag{23}$$

By default, DASSL will always attempt to use the most recent $\hat{G}$ and $\hat{\alpha}$ when running the modified Newton iteration. DASSL will then estimate the rate of convergence of (21) after 2 iterations to determine if G and/or α need to be recomputed. If the desired rate of convergence is still not satisfied or if more than 4 iterations occur, DASSL will reset by trying again with a new step size and/or order [1].

3.5. **Adapting Step Size and Order.** DASSL updates either the step size, order, or both at each approximation to account for two types of error [1]:

(1) Local truncation error.
(2) Interpolating polynomial error.

The local truncation error can be estimated by [1]

$$\tau_{j+1} \approx \left|\frac{h_j}{t_{j+1} - t_{j-k}} + \alpha_s - \left(\frac{h_j}{t_{j+1} - t_j} + \frac{h_j}{t_{j+1} - t_{j-1}} + \cdots + \frac{h_j}{t_{j+1} - t_{j+1-k}}\right)\right| \cdot ||\vec{w}_{j+1} - \vec{w}_{j+1}^{(0)}||. \tag{24}$$

The *interpolating polynomial* is created by DASSL at the end of every successful approximation and is given by

$$\vec{\omega}_{j+1}^{I}(t) = \vec{w}_{j+1} + (t - t_{j+1})[\vec{w}_{j+1}, \vec{w}_j] + \cdots + (t - t_{j+1})(t - t_j)\cdots(t - t_{j-k+2})[\vec{w}_{j+1}\cdots\vec{w}_{j-k+1}]. \tag{25}$$

It is used for interpolating the solution between $t = t_j$ and $t = t_{j+1}$. Its error is estimated by [1]

$$E_{\omega^I} = \frac{h_j}{t_{j+1} - t_{j-k}} \cdot ||\vec{w}_{j+1} - \vec{w}_{j+1}^{(0)}||. \tag{26}$$

So, after Newton's method has successfully converged, DASSL requires that the total error satisfies [1]

$$E = \max(\tau_{j+1}, E_{\omega^I}) \leq 1. \tag{27}$$

The reasoning behind the error bound of 1 is not directly explained in the paper, but is likely related to the fact that DASSL uses a weighted root mean square norm, with the weights depending on the value of $\vec{w}_{j+1}$ at the beginning of a step as well as the absolute and relative tolerances inputted by the user [1].

Before choosing a step size, either for retrying a step or for advancing to the next step, DASSL first decides the order of the BDF to use. The reason for varying the order is due to the fact that some ODE systems are unstable for higher order BDF's [1]. DASSL can detect this behavior and correct it by simply using a lower order and decreasing the step size as needed.

Detecting behavior like this is accomplished by estimating what the error *would have been if other orders were used* to create the last approximation *with a constant step size* [1]. Estimating the error for some order k requires $k+1$ previous approximations since the predictor polynomial (14) used to create the initial guess $\vec{w}_{j+1}^{(0)}$ uses $k+1$ nodes. Depending on the number of previous approximations, DASSL will attempt to create error estimates E_{k_j-2}, E_{k_j-1}, E_{k_j}, and E_{k_j+1} for orders k_j-2, k_j-1, k_j, and k_j+1. If these estimates decrease as the order increases, then the order will be increased on the next step; if the estimates decrease as order increases, the order will be lowered on the next step; if the estimates neither increase nor decrease, the order will remain the same [1].

Once the new order κ has been decided, the next step size can be chosen. DASSL's strategy is to estimate what the error *would have been* if the last $\kappa+1$ steps were taken at some new step size. Letting the new step size $h_{j+1} = rh_j$, an initial estimate for r is given by

$$r = (2 \cdot E_\kappa)^{-\frac{1}{\kappa+1}}, \tag{28}$$

where E_κ is the error estimate for the new order κ that was selected for the next step. Let n_f denote the number of failed attempts at producing the approximation $\vec{w}_{j+1}$ from previous iterations of step size and order. Based on the values of E_κ, r, n_f, and j, DASSL adjusts the step size and order as follows [1]:

- If $n_f \geq 3$, it is likely that error estimates can no longer be trusted. r is set to $1/4$ and κ is set to 1.
- If $j = 0$, $n_f = 1$, and $E_\kappa > 1$, there are not enough previous approximations to increase the order. r is set to $1/4$ and κ is set to the previous order k_j.
- If $j = 1$, $n_f = 0$, and $E_\kappa < 1$, there are enough approximations to increase the order to 2. r is set to 1 and κ is set to 2.
- If $j < k+1$ and $n_f = 0$, then this is the first attempt at creating $\vec{w}_{j+1}$. There are not enough steps to create any of the error estimates used to determine step size, so r is calculated using (28) and κ is set to $k_j + 1$.

- If $j < k_j + 1$ and $n_f \neq 0$, then this is not the first attempt at creating $\vec{w}_{j+1}$. There are not enough steps to create any of the error estimates used to determine step size, so r is set to $1/4$ and κ is set to $k_j - 1$.
- If $j \geq k_j + 1$, there are enough steps to create error estimates. κ is set to $k_j + 1$, $k_j - 1$, or k_j depending on whether the error estimates increase or decrease with increasing order. r is calculated using (28), and then normalized as follows:
  - If r can be at least doubled, i.e. $r \geq 2$, r is doubled. Set r to 2.
  - If $r < 1$, r is set to value between $1/4$ and $9/10$. The higher n_f is, the lower the value r is set to.
  - If r does not meet any of the above criteria, it is set to 1. Step size stays the same.

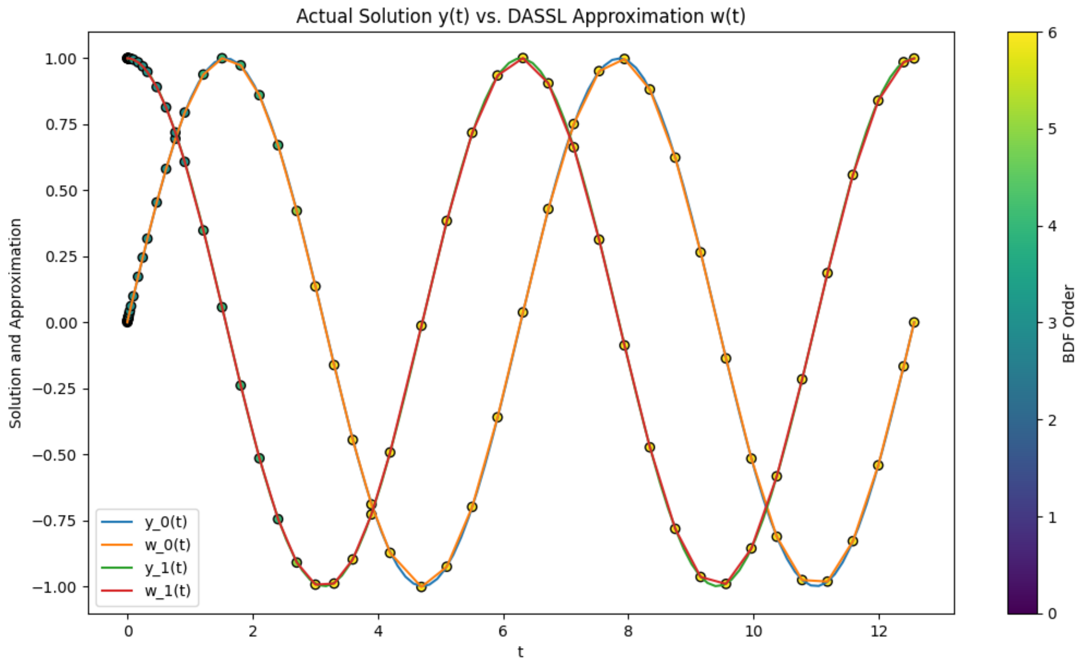

FIGURE 2.

3.6. **Numerical Examples.** The following ODE system

$$\begin{aligned} y_0'(t) &= y_1(t) \\ y_1'(t) &= -y_0(t) \\ y_0(0) &= 0 \\ y_1(0) &= 1 \end{aligned} \tag{29}$$

has exact solution

$$\begin{aligned} y_0(t) &= sin(t) \\ y_1(t) &= cos(t). \end{aligned} \tag{30}$$

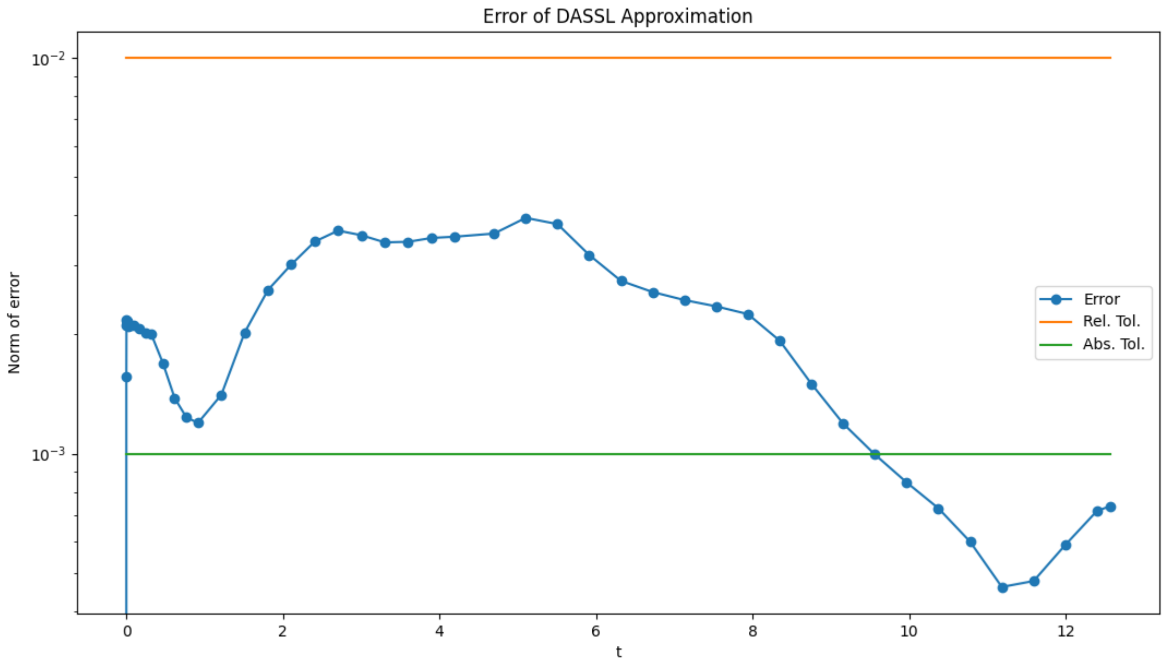

FIGURE 3.

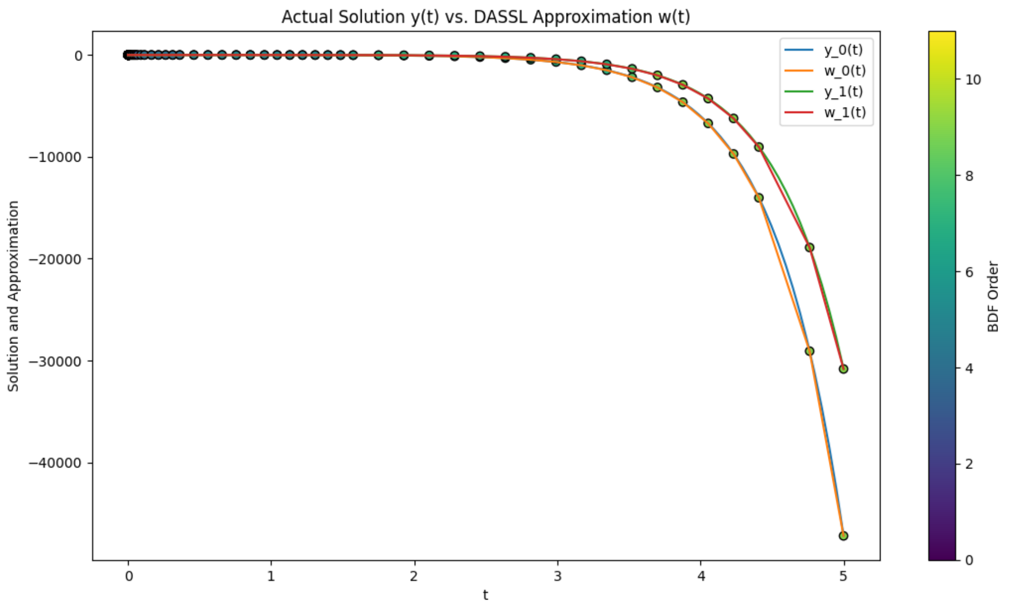

FIGURE 4.

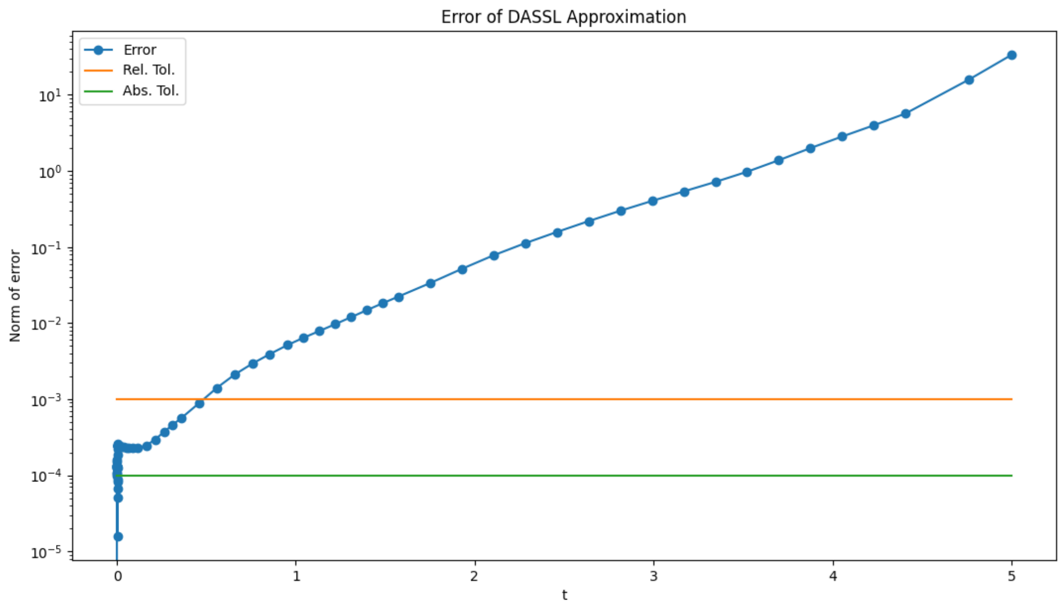

FIGURE 5.

Figure 2 shows the results of running DASSL [1] on this system on the interval $[0, 4\pi]$ with a relative tolerance of 10^{-2}, absolute tolerance of 10^{-3}, an initial step size of 0.1, and a maximum step size of 0.5. Initial guesses for $\vec{y}'(0)$ were $y_0'(0) = 1$ and $y_1'(0) = 1$. DASSL increases the BDF order all the way to 6 near $t = 3$, where it remains for the rest of the interval; this system is not prone to instability when using higher order BDF's. The higher order allows for maximizing the step size between peaks and valleys, with slight decreases on the peaks and valleys themselves. Figure 3 shows the error of the approximation, which is well within the desired relative tolerance.

On the other hand, the below ODE system

$$\begin{aligned} y_0'(t) &= 4y_0(t) - 3y_1(t) + t \\ y_1'(t) &= 2y_1(t) - y_1(t) + e^t \\ y_0(0) &= 0 \\ y_1(0) &= 0 \end{aligned} \tag{31}$$

has exact solution

$$\begin{aligned} y_0(t) &= 3e^t t + \tfrac{t}{2} + e^t - \tfrac{9e^{2t}}{4} + \tfrac{5}{4} \\ y_1(t) &= 3e^t t + t - \tfrac{3e^{2t}}{2} + \tfrac{3}{2}. \end{aligned} \tag{32}$$

In Figure 4, DASSL was run on the interval $[0, 5]$, relative and absolute tolerances of 10^{-3} and 10^{-4} respectively, an initial step size of 0.1, and a maximum step size of 1. Initial guesses for $\vec{y}'(0)$ were again $y_0'(0) = 1$ and

[1]`https://github.com/luse9638/APPM_4610_Final_Project`

$y_1'(0) = 1$. The maximum order was also increased, but here DASSL does not maximize the order unlike the previous example, as that would lead to instability for the ODE system. Most of the error is not bound by the desired relative tolerance as seen in Figure 5. However, at $t = 5$, the norm of the error is only a factor of 10^1, which is quite good relative to value of the actual solutions $y_0(5) \approx -47181$ and $y_1(t) \approx -30807$.

4. Conclusion

Variable step size methods have immense utility when solving ODEs. Compared to constant step size methods, they combine the benefits of low truncation error found with using smaller step sizes with the benefits of less computational work that come with using larger step sizes. Besides their application in Runge-Kutta methods for first order ODEs, they also have use in solving systems of ODE as demonstrated through the DASSL algorithm.

However, variable step size methods are not perfect. They can be quite tricky to implement in code, requiring lots of bookkeeping on past approximations and past step sizes. The implementation of DASSL used in this paper was particularly difficult to write; there is likely a better way to adjust order, step size, and calculate error estimates compared to the current implementation. Additionally, DASSL is limited to solving only first order ODE systems; a possible future exploration would be to extend DASSL to work for higher order systems.

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] K. R. Jackson and R. Sacks-Davis, *An alternative implementation of variable step-size multistep formulas for stiff odes*, ACM Transactions on Mathematical Software **6** (1980), no. 3, 295–318.

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

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