# *Error in Finite Difference Methods*

We have now cataloged a variety of one-step and multi-step methods. But under what settings is it appropriate to use a given method? We need to characterize the error and stability properties of finite difference methods to answer this question. We now discuss truncation error, notions of zero stability to more fully characterize the convergence of a method, and notions of absolute stability to determine which values of $\Delta t$ are appropriate to use for a given method and IVP.

## 1 *Global & truncation error: one-step and multi-step methods*

Now that we have developed a framework for deriving both one-step and multi-step methods, we turn here to an equally important question: how do we determine the accuracy of the method we have derived. Our ultimate interest is in the difference between the approximate solution we compute and the exact solution at a given time instance. This all-important quantity is called the *global error*.

**Global error**

The *global error* of a finite-difference method at some time instance $t_k$ is defined as $\boldsymbol{e}_k := \boldsymbol{u}(t_k) - \boldsymbol{u}_k$. Often, this error is expressed succinctly using an appropriately defined norm $||\cdot||$ as $||\boldsymbol{e}_k||$.

There are two sources that contribute to this global error. We will first illustrate these different sources on the forward Euler method. For this method, notice that

1. Even if we had the exact solution $\boldsymbol{u}(t)$, applying the forward Euler method to this solution via $\boldsymbol{u}(t_{k+1}) = \boldsymbol{u}(t_k) + \Delta t \boldsymbol{f}(\boldsymbol{u}(t_k), t_k)$ would still lead to an error. We refer to the associated error as the *truncation error*, and define it as follows:

$$\boldsymbol{\tau}_k = \frac{\boldsymbol{u}(t_{k+1}) - \boldsymbol{u}(t_k)}{\Delta t} - \boldsymbol{f}(\boldsymbol{u}(t_k), t_k) \tag{1}$$

In words, the truncation error associated with the forward Euler method is the error in applying the method to advance the *exact solution* by one time increment $\Delta t$.

Note that many other texts define the truncation error by multiplying our truncation error by $\Delta t$. I find that definition unhelpfully confusing, and we will use the convention described in (1).

2. We do not in general have access to the exact solution at time $t_k$, and so we typically inherit a cumulative error from having applied the forward Euler method $k$ time increments from the exact initial condition.

These two sources of error—the truncation error introduced at a single time increment and the cumulative error inherited over several previous time increments—together make up the global error. We illustrated these two contributions to the global error using the forward Euler method, but the same sources exist for any finite-difference method applied to an IVP. We will first focus on the truncation error and subsequently relate this to the global error.

What is the truncation error associated with the trapezoid method?

### Truncation error: one-step methods

To avoid cumbersome algebra, we will work with the forward Euler method (the process is identical for other one-step methods). For the forward Euler method, the truncation error is

$$\begin{aligned}
\boldsymbol{\tau}_k &= \frac{\boldsymbol{u}(t_{k+1}) - \boldsymbol{u}(t_k)}{\Delta t} - \boldsymbol{f}(\boldsymbol{u}(t_k), t_k) \\
&= \frac{\boldsymbol{u}(t_{k+1}) - \boldsymbol{u}(t_k)}{\Delta t} - \dot{\boldsymbol{u}}(t_k) \quad \text{[using the definition of an IVP]} \\
&= \frac{\boldsymbol{u}(t_{k+1}) - \boldsymbol{u}(t_k)}{\Delta t} - \frac{\boldsymbol{u}(t_{k+1}) - \boldsymbol{u}(t_k)}{\Delta t} + O(\Delta t) \\
&\qquad \text{[using a Taylor series expansion of } \boldsymbol{u}(t_k) \text{ about } t_{k+1}\text{]} \\
\implies \quad \boldsymbol{\tau}_k &= O(\Delta t)
\end{aligned} \tag{2}$$

One can use an analogous process to show that Heun's method has a truncation error $\boldsymbol{\tau}_k = O(\Delta t^2)$ and the four-stage Runge-Kutta method has a truncation error $\boldsymbol{\tau}_k = O(\Delta t^4)$.

Test yourself by deriving the scaling of the truncation error for Heun's method.

### Truncation error: multi-step methods

One difference between one-step and multi-step methods is that the truncation error is easier to define generally for multi-step methods, since multi-step methods can be readily cast in the form (**??**). Given this fact, we can write the truncation error as

*Truncation error*: multi-step methods

An $r$-step method defined using (**??**) has a truncation error given by

$$\begin{aligned}
\boldsymbol{\tau}_k = \frac{1}{\Delta t}\Bigg[ &\sum_{j=k-r+1}^{k+1} \alpha_{j-(k-r+1)} \boldsymbol{u}(t_j) - \\
&\Delta t \sum_{j=k-r+1}^{k+1} \beta_{j-(k-r+1)} \boldsymbol{f}\Big(\boldsymbol{u}(t_j), t_j\Big)\Bigg]
\end{aligned} \tag{3}$$

This definition is entirely analogous to our definition for one-step methods: the truncation error remains the error associated with advancing the exact solution one time step using the finite-difference method.

The general definition (**??**) also enables a clean derivation of the

truncation error. Noting that $\boldsymbol{f}\Big(\boldsymbol{u}(t_j), t_j\Big) = \dot{\boldsymbol{u}}(t_j)$, and that

$$\begin{aligned}\boldsymbol{u}(t_j) &= \boldsymbol{u}(t_k) + (j-k)\Delta t\dot{\boldsymbol{u}}(t_j) + \frac{1}{2}(j-k)^2\Delta t^2\ddot{\boldsymbol{u}}(t_j) + \cdots \\ \dot{\boldsymbol{u}}(t_j) &= \dot{\boldsymbol{u}}(t_k) + (j-k)\Delta t\ddot{\boldsymbol{u}}(t_j) + \frac{1}{2}(j-k)^2\Delta t^2\dddot{\boldsymbol{u}}(t_j) + \cdots\end{aligned} \tag{4}$$

the truncation error becomes

$$\begin{aligned}\boldsymbol{\tau}_k = \frac{1}{\Delta t}&\left(\sum_{j=k-r+1}^{k+1}\alpha_{j-(k-r+1)}\right)\boldsymbol{u}(t_k) + \left(\sum_{j=k-r+1}^{k+1}\left[(j-k)\alpha_{j-(k-r+1)} - \beta_{j-(k-r+1)}\right]\right)\dot{\boldsymbol{u}}(t_k) + \\ &\Delta t\left(\sum_{j=k-r+1}^{k+1}\left[\frac{1}{2}(j-k)^2\alpha_{j-(k-r+1)} - (j-k)\beta_{j-(k-r+1)}\right]\right)\ddot{\boldsymbol{u}}(t_k) + \cdots + \\ &\Delta t^{q-1}\left(\sum_{j=k-r+1}^{k+1}\left[\frac{1}{q!}(j-k)^q\alpha_{j-(k-r+1)} - \frac{1}{(q-1)!}(j-k)^{q-1}\beta_{j-(k-r+1)}\right]\right)\frac{d^q\boldsymbol{u}}{dt^q}\Big|_{t_k}\end{aligned} \tag{5}$$

In order for a multi-step method to converge, we must have that $\boldsymbol{\tau}_k \to 0$ as $\Delta t \to 0$. From (5), it is clear that for this condition to be satisfied, we require (at minimum)

$$\sum_{j=k-r+1}^{k+1}\alpha_{j-(k-r+1)} = 0, \quad \sum_{j=k-r+1}^{k+1}\left[(j-k)\alpha_{j-(k-r+1)} - \beta_{j-(k-r+1)}\right] = 0 \tag{6}$$

A multi-step method that satisfies (6) is called *consistent*.

More generally, if the first $p+1$ terms in (5) vanish, the truncation error scales as $O(\Delta t^p)$.

Let us make this more concrete with an example. For the 2-step Adams-Bashforth method,

$$\begin{aligned}&\alpha_0 = 0,\ \alpha_1 = -1, \alpha_2 = 1 \\ &\beta_0 = -\frac{1}{2}, \beta_1 = \frac{3}{2}, \beta_2 = 0\end{aligned} \tag{7}$$

For this set of coefficients, we observe that the first condition in (6) is satisfied: $\alpha_0 + \alpha_1 + \alpha_2 = 0 - 1 + 1 = 0$. For the second condition, noting that $r = 2$, we have

$$\begin{aligned}&\sum_{j=k-2+1}^{k+1}\left[(j-k)\alpha_{j-(k-2+1)} - \beta_{j-(k-2+1)}\right] \\ &= \left[\Big(-\alpha_0 - \beta_0\Big) + \Big(0\alpha_1 - \beta_1\Big) + \Big(\alpha_2 - \beta_2\Big)\right] \\ &= \left[\left(0 + \frac{1}{2}\right) + \left(0(-1) - \frac{3}{2}\right) + \Big(1 - 0\Big)\right] \\ &= 0\end{aligned} \tag{8}$$

Thus, both conditions in (6) are satisfied and the method is consistent.

But we can say more! Considering the $\Delta t$ term in (5), we see that

$$\begin{aligned}
&\sum_{j=k-r+1}^{k+1} \left[\frac{1}{2}(j-k)^2\alpha_{j-(k-r+1)} - (j-k)\beta_{j-(k-r+1)}\right] \\
&\quad = \left[\left(\frac{1}{2}\alpha_0 + \beta_0\right) + \left(\frac{1}{2}(0)\alpha_1 - 0\beta_1\right) + \left(\frac{1}{2}\alpha_2 - \beta_2\right)\right] \\
&\quad = \left[\left(\frac{1}{2}(0) - \frac{1}{2}\right) + \left(\frac{1}{2}(0)(-1) - (0)\frac{3}{2}\right) + \left(\frac{1}{2}(1) - 0\right)\right] \\
&\quad = 0
\end{aligned} \tag{9}$$

If we were to evaluate the $O(\Delta t^2)$ term, we would find that it did not vanish, and the truncation error for this method is therefore $O(\Delta t^2)$.

Recall the error of $O(\Delta t)$ error associated with the forward Euler method. The $O(\Delta t^2)$ error of the 2-step Adams-Bashforth method reveals the reward of its added complexity.

### Relating the truncation error to the global error

A key question is: how does this truncation error relate to the global error $\boldsymbol{e}_k$—the quantity that we really care about? Another way to ask this question is "how does the contribution of error that we inherited from using our numerical method for the $k$ previous time steps affect the truncation error $\boldsymbol{\tau}_k$?"

We do not have time to dwell on the details of this question, but the punchline is that the global error scales at the same rate as the truncation error provided that the numerical method obeys a property referred to as *stability*. We will discuss the concept of stability in the next lecture.

We left the last lecture by stating that the truncation error correctly determined the accuracy of a numerical method, provided that the numerical method was stable. We will make this notion of stability precise in this lecture.

First, why do we need the concept of stability at all? That is, why is the truncation error not sufficient to predict the accuracy of a numerical method. The reason is that the truncation error is derived using a Taylor series, which is only valid for sufficiently small $\Delta t$. To illustrate the dependence of the 'right $\Delta t$' on the finite-difference method being considered, we will consider the model problem

$$\begin{aligned}
\dot{\boldsymbol{u}} &= \boldsymbol{\Lambda}\boldsymbol{u} \\
\boldsymbol{u}(t_0) &= \boldsymbol{u}_0
\end{aligned} \tag{10}$$

where $\boldsymbol{\Lambda}$ is a diagonal matrix

$$\boldsymbol{\Lambda} = \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 & 0 \\ 0 & \lambda_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \ldots & \vdots & \vdots \\ 0 & 0 & \cdots & \lambda_{n-1} & 0 \\ 0 & 0 & \cdots & 0 & \lambda_n \end{bmatrix} \tag{11}$$

with each $\lambda_l \in \mathbb{C}$, $l = 1, \ldots, n$. Using the definition of $e^{\boldsymbol{\Lambda t}}$ from lecture 11, the $j^{th}$ component of the exact solution is given by

$$u_j(t) = e^{\lambda_j (t - t_0)} (u_0)_j \tag{12}$$

where $(u_0)_j$ denotes the $j^{th}$ entry of $u_0$. The real part of $\lambda_j$ gives the growth or decay rate of $u_j(t)$ and the imaginary part gives the frequency of oscillation.

## 2 *Absolute stability for one-step methods*

Let's consider applying the forward Euler method to this model IVP. Doing this gives us an expression to solve for $\boldsymbol{u}_{k+1}$ using known information at time $t_k$:

$$\begin{aligned} \boldsymbol{u}_{k+1} &= \boldsymbol{u}_k + \Delta t \boldsymbol{\Lambda} \boldsymbol{u}_k \\ &= (\boldsymbol{I} + \Delta t \boldsymbol{\Lambda}) \boldsymbol{u}_k \\ &= (\boldsymbol{I} + \Delta t \boldsymbol{\Lambda})(\boldsymbol{I} + \Delta t \boldsymbol{\Lambda}) \boldsymbol{u}_{k-1} \\ &= (\boldsymbol{I} + \Delta t \boldsymbol{\Lambda})^{k+1} \boldsymbol{u}_0 \end{aligned} \tag{18}$$

And since $\boldsymbol{\Lambda}$ is diagonal, the $j^{th}$ entry in $\boldsymbol{u}_{k+1}$ can be expressed simply as

$$(u_{k+1})_j = (1 + \Delta t \lambda_j)^{k+1} (u_0)_j \tag{19}$$

Thus, if $|1 + \Delta t \lambda_j| > 1$, the iterates $(u_{k+1})_j$ will grow without bound as $k$ tends to infinity. That is, the numerical solution will blow up when $\sqrt{(1 + \mathcal{R}(\Delta t \lambda_j))^2 + \mathcal{I}(\Delta t \lambda_j)^2} > 1$, where $\mathcal{R}(\Delta t \lambda_j)$ and $\mathcal{I}(\Delta t \lambda_j)$ denote the real and imaginary parts of $\Delta t \lambda_j$, respectively. Notice that it is the quantity $\Delta t \lambda_j$ that is important, and not $\Delta t$ or $\lambda_j$ separately, that determines the stability of a method. We say that the forward Euler method is *absolutely stable* when $|1 + \Delta t \lambda_j| < 1$.

How does the stability region of the forward Euler method compare with that of other methods we were exposed to earlier? Let us consider the backward Euler method. When applied to the model IVP (10), the backward Euler method becomes

$$\begin{aligned} \boldsymbol{u}_{k+1} &= \boldsymbol{u}_k + \Delta t \boldsymbol{\Lambda} \boldsymbol{u}_{k+1} \\ \boldsymbol{u}_{k+1} &= (\boldsymbol{I} - \Delta t \boldsymbol{\Lambda})^{-1} \boldsymbol{u}_k \\ &= [(\boldsymbol{I} - \Delta t \boldsymbol{\Lambda})^{-1}]^{k+1} \boldsymbol{u}_0 \end{aligned} \tag{20}$$

For the diagonal matrix $\boldsymbol{\Lambda}$, the $j^{th}$ entry in $\boldsymbol{u}_{k+1}$ can be expressed simply as

$$(u_{k+1})_j = \frac{1}{(1 - \Delta t \lambda_j)^{k+1}} (u_0)_j \tag{21}$$

Thus, the backward Euler method is *absolutely stable* when $1/|1 - \Delta t \lambda_j| < 1$.

negative constant would correspond to a negative spring stiffness!)

Note that this IVP can be represented as a first-order system of ODEs by defining $v = \dot{y}$, $\boldsymbol{z} = [y, v]^T$, and $\boldsymbol{z}_0 = [y_0, v_0]^T$. We can therefore recast (13) as

$$\begin{aligned} \dot{\boldsymbol{z}} &= \boldsymbol{A} \boldsymbol{z} \\ \boldsymbol{z}(0) &= \boldsymbol{z}_0 \end{aligned} \tag{14}$$

where

$$\boldsymbol{A} = \begin{bmatrix} 0 & 1 \\ -\omega^2 & 0 \end{bmatrix} \tag{15}$$

You may recognize that $\boldsymbol{A}$ has an eigen-decomposition $\boldsymbol{A} = \boldsymbol{V} \boldsymbol{\Lambda} \boldsymbol{V}^{-1}$ where

$$\begin{aligned} \boldsymbol{V} &= \begin{bmatrix} \frac{1}{2} i & -\frac{1}{2} i \\ \sqrt{\frac{3}{4}} & \sqrt{\frac{3}{4}} \end{bmatrix} \\ \boldsymbol{\Lambda} &= \begin{bmatrix} i\sqrt{\omega} & 0 \\ 0 & -i\sqrt{\omega} \end{bmatrix} \end{aligned} \tag{16}$$

Premultiplying (14) by $\boldsymbol{V}^{-1}$ and defining $\boldsymbol{u} = \boldsymbol{V}^{-1} \boldsymbol{z}$ results in the linear system

$$\begin{aligned} \dot{\boldsymbol{u}} &= \boldsymbol{\Lambda} \boldsymbol{u} \\ \boldsymbol{u}(0) &= \boldsymbol{u}_0 \end{aligned} \tag{17}$$

which is identical to (10). Note that, by virtue of (16), the entries of $\boldsymbol{\Lambda}$ are complex. Thus, even when modeling real physical systems, the associated first-order IVP can involve complex entries.

A picture of this might make things clearer. The forward Euler method is stable when $\Delta t \lambda_j$ is contained within the filled in circle, and unstable otherwise. Notice that, by virtue of (12), the exact solution is stable whenever $\mathcal{R}(\lambda_j) < 0$. This gives us an indication of the frail stability properties of this method: if $\mathcal{R}(\lambda_j) << -2$, then even though the true solution will decay in time, we will have to use a very small $\Delta t$ to stably simulate the system.

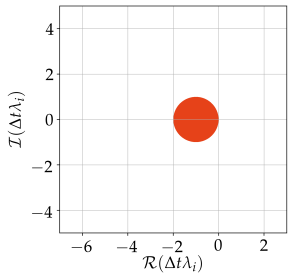
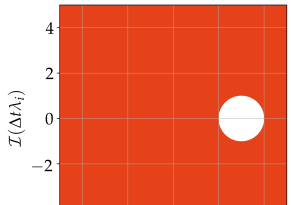



Figure 1: Absolute stability region for the forward Euler method.

A figure is again useful here. Notice the fruits we have reaped from using an implicit Euler method: the method is absolutely stable so long as $\Delta t \lambda_j$ does not lie in the unit circle centered at $\Delta t \mathcal{R}(\lambda_j) = 1$.



We will consider one more example: the RK4 scheme. When applied to the model problem (10), RK4 can be expressed as

$$\begin{aligned}\boldsymbol{u}_{k+1} &= \boldsymbol{u}_k + \frac{1}{6}\Delta t\Bigg\{\boldsymbol{\Lambda}\boldsymbol{u}_k + 2\boldsymbol{\Lambda}\left(\boldsymbol{u}_k + \frac{1}{2}\Delta t\boldsymbol{\Lambda}\boldsymbol{u}_k\right) + \\ &\qquad 2\boldsymbol{\Lambda}\left(\boldsymbol{u}_k + \frac{1}{2}\Delta t\boldsymbol{\Lambda}\left(\boldsymbol{u}_k + \frac{1}{2}\Delta t\boldsymbol{\Lambda}\boldsymbol{u}_k\right)\right) + \\ &\qquad \boldsymbol{\Lambda}\left[\boldsymbol{u}_k + \Delta t\boldsymbol{\Lambda}\left(\boldsymbol{u}_k + \frac{1}{2}\Delta t\boldsymbol{\Lambda}\left(\boldsymbol{u}_k + \frac{1}{2}\Delta t\boldsymbol{\Lambda}\boldsymbol{u}_k\right)\right)\right]\Bigg\} \qquad (22)\\ &= \left\{\boldsymbol{I} + \Delta t\boldsymbol{\Lambda} + \frac{1}{2}\left(\Delta t\boldsymbol{\Lambda}\right)^2 + \frac{1}{6}\left(\Delta t\boldsymbol{\Lambda}\right)^3 + \frac{1}{24}\left(\Delta t\boldsymbol{\Lambda}\right)^4\right\}\boldsymbol{u}_k \\ &= \left\{\boldsymbol{I} + \Delta t\boldsymbol{\Lambda} + \frac{1}{2}\left(\Delta t\boldsymbol{\Lambda}\right)^2 + \frac{1}{6}\left(\Delta t\boldsymbol{\Lambda}\right)^3 + \frac{1}{24}\left(\Delta t\boldsymbol{\Lambda}\right)^4\right\}^{k+1}\boldsymbol{u}_0\end{aligned}$$

Again, the diagonal nature of $\boldsymbol{\Lambda}$ enables a simple representation of the $j^{th}$ entry of $\boldsymbol{u}_{k+1}$:

$$(u_{k+1})_j = \left\{1 + \Delta t\lambda_j + \frac{1}{2}\left(\Delta t\lambda_j\right)^2 + \frac{1}{6}\left(\Delta t\lambda_j\right)^3 + \frac{1}{24}\left(\Delta t\lambda_j\right)^4\right\}^{k+1}(u_0)_j \qquad (23)$$

How are we drawing the figures in the margins? Let's generalize the approach for determining stability regions. For the forward Euler method, backward Euler method, and RK4, we were able to relate $(u_{k+1})_j$ to $(u_0)_j$ using a function of $\Delta t\lambda_j$. It turns out that this is true for one-step methods in general. That is, for one-step methods we can write $(u_{k+1})_j = R(\Delta t\lambda_j)(u_0)_j$.

Since it is the quantity $\Delta t\lambda_j$ that is important, and not $\Delta t$ or $\lambda_j$ separately, it is customary to define $w = \Delta t\lambda_j$. This allows us to rewrite the relation between $(u_{k+1})_j$ and $(u_0)_j$ more succinctly as

$$(u_{k+1})_j = R^{k+1}(w)(u_0)_j \qquad (24)$$

In this notation, a one-step method is stable when $|R(w)| < 1$. This gives us a mechanism for computing the stability region of a one-step method: we need only determine $R(w)$ for the given one-step method and identify the values of $w$ for which $|R(w)| < 1$. Here is a code that plots the stability region for the backward Euler method.

For the RK4 method, the stability region is larger than for the forward Euler method but not as extensive as the backward Euler method.

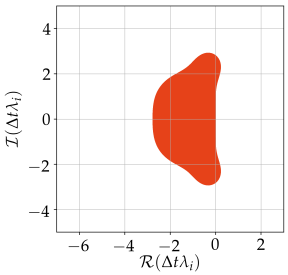



Figure 3: Absolute stability region for the RK4 method.
*E.g.*, $R(\Delta t\lambda_j) = 1 + \Delta t\lambda_j + \frac{1}{2}(\Delta t\lambda_j)^2 + \frac{1}{6}(\Delta t\lambda_j)^3 + \frac{1}{24}(\Delta t\lambda_j)^4$ for RK4.

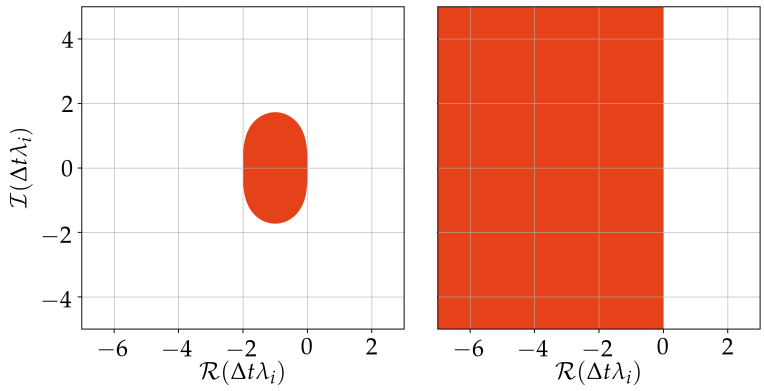
Note that $w \in \mathbb{C}$ since $\lambda_j \in \mathbb{C}$

Listing 1: Code snippet for computing the absolute stability region for the Backward Euler method

```
%define real and imaginary parts of w
wr = -5 : 0.01 : 3;
wi = -4 : 0.01 : 4 ;

%create meshgrid
```

```
[Wr, Wi] = meshgrid( wr, wi );

%define |R(w)| for BE
BE_sc = abs( 1./(1 - (Wr + 1i*Wi)) );

%saturate values for easier coloring
BE_sc( BE_sc < 1 ) = 1;
BE_sc( BE_sc > 1 ) = 2;

%UIUC-themed colormap
cmap = [1 1/2 0; 1 1 1];

%Create a contour plot of the stability region
contourf( Wr, Wi, BE_sc, [1 2])
colormap( cmap ), axis equal
```

For reference, the stability regions of the other methods we explored in earlier lectures are presented in figure 4.




(a) (b)

Figure 4: Stability region for Heun's method a) and the trapezoid method b).

Let us summarize this important result of absolute stability for one step methods.

*Absolute stability criterion*: one-step methods

A one-step method is called *absolutely stable* for values of $\Delta t\lambda_l$ that yield $|R(\Delta t\lambda_l)| < 1$. The method is *unstable* for values of $\Delta t\lambda_l$ that do not satisfy that criteria.

## 3 *Absolute stability for multi-step methods*

As with one-step methods, we must again embrace the fact that all of our computations involve a finite $\Delta t$. In this vein, we characterize the finite values of $\Delta t$ that lead to a stable solution with a given method

using the concept of absolute stability. We will again define this concept with respect to the model problem (10).

Applying a multi-step method to this model problem results in the expression

$$\sum_{j=k-r+1}^{k+1} \alpha_{j-(k-r+1)} \boldsymbol{u}_j = \Delta t \sum_{j=k-r+1}^{k+1} \beta_{j-(k-r+1)} \boldsymbol{\Lambda} \boldsymbol{u}_j \tag{25}$$

which can be rearranged as

$$\sum_{j=k-r+1}^{k+1} \left[ \alpha_{j-(k-r+1)} \boldsymbol{I} - \Delta t \beta_{j-(k-r+1)} \boldsymbol{\Lambda} \right] \boldsymbol{u}_j = 0 \tag{26}$$

As occurred for one-step methods, the diagonal form of $\boldsymbol{\Lambda}$ enables us to write out the equation for each component of $\boldsymbol{u}_j$. In particular, the $l^{th}$ component of $\boldsymbol{u}_j$, $(u_j)_l$, can be solved for via

$$\sum_{j=k-r+1}^{k+1} \left[ \alpha_{j-(k-r+1)} - \Delta t \beta_{j-(k-r+1)} \lambda_l \right] (u_j)_l = 0 \tag{27}$$

How can we solve this equation for $(u_j)_l$? Notice that if we define a variable $\zeta \in \mathbb{R}$ and replace $(u_j)_l$ in (27) with $\zeta^{j+r-1}$, we arrive at the expression

$$\sum_{j=k-r+1}^{k+1} \left[ \alpha_{j-(k-r+1)} - \Delta t \beta_{j-(k-r+1)} \lambda_l \right] \zeta^{j+r-1} = 0 \tag{28}$$

To make our notation simpler, we can divide by $\zeta^k$ and rework our indexing to get

$$\sum_{j=0}^{r} \left[ \alpha_j - \Delta t \beta_j \lambda_l \right] \zeta^j = 0 \tag{29}$$

This polynomial equation in terms of $\zeta$ has $r$ roots. If these roots are distinct, we may denote them as $\zeta_1, \ldots, \zeta_r$. Each of the roots satisfies (28) and, as a result, (27). Notice that any linear combination of the roots also satisfies (28). Thus, we can construct a $(u_j)_l$ that solves (27) by writing $(u_j)_l$ as a linear combination of the roots of (29):

$$(u_j)_l = \sum_{m=1}^{r} c_m \zeta_m^j \tag{30}$$

We have finally arrived at our desired result: we now have a means to establish a meaningful criterion for whether a multi-step method will be stable or not.

Remember that this model IVP is defined as

$$\dot{\boldsymbol{u}} = \boldsymbol{\Lambda} \boldsymbol{u}$$
$$\boldsymbol{u}(t_0) = \boldsymbol{u}_0$$

where $\boldsymbol{\Lambda}$ is the diagonal matrix

$$\boldsymbol{\Lambda} = \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 & 0 \\ 0 & \lambda_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ 0 & 0 & \cdots & \lambda_{n-1} & 0 \\ 0 & 0 & \cdots & 0 & \lambda_n \end{bmatrix}$$

with each $\lambda_l \in \mathbb{C}$, $l = 1, \ldots, n$.

Some extensions must be made in the derivation allowing for non-distinct roots, though the conclusions are unchanged.

The relation (30) demonstrates that if *any* of the roots $\zeta_1, \ldots, \zeta_r$ have modulus greater than one, $(u_j)_l$ will grow without bound as $j \to \infty$. Thus, we will define absolute stability in terms of the values of $\Delta t \lambda_l$ that lead to roots with modulus less than one.

*Absolute stability criterion*: multi-step methods

An $r$-step method is called *absolutely stable* for values of $\Delta t\lambda_l$ that yield solutions $\zeta_1, \ldots, \zeta_r$ to (29) satisfying $|\zeta_1| < 1$, $|\zeta_2| < 1, \ldots, |\zeta_r| < 1$. The method is *unstable* for values of $\Delta t\lambda_l$ that do not satisfy that criteria.

Let us make this definition of absolute stability concrete by considering an example. Recall that the coefficients for the 2-step Adams-Bashforth method are

$$\begin{aligned} \alpha_0 = 0,\ \alpha_1 = -1, \alpha_2 = 1 \\ \beta_0 = -\frac{1}{2}, \beta_1 = \frac{3}{2}, \beta_2 = 0 \end{aligned} \tag{31}$$

The polynomial equation (29) therefore becomes

$$\begin{aligned} & \left[\alpha_0 - (\Delta t\lambda_l)\beta_0\right] + \left[\alpha_1 - (\Delta t\lambda_l)\beta_1\right]\zeta + \left[\alpha_2 - (\Delta t\lambda_l)\beta_2\right]\zeta^2 = 0 \\ \implies \quad & \left[(\Delta t\lambda_l)\frac{1}{2}\right] + \left[-1 - (\Delta t\lambda_l)\frac{3}{2}\right]\zeta + \left[1\right]\zeta^2 = 0 \end{aligned} \tag{32}$$

We may solve this quadratic for $\zeta$ in terms of $\Delta t\lambda_l$:

$$\zeta = \frac{\left[1 + (\Delta t\lambda_l)\frac{3}{2}\right] \pm \sqrt{\left[1 + (\Delta t\lambda_l)\frac{3}{2}\right]^2 - 4\left[(\Delta t\lambda_l)\frac{1}{2}\right]}}{2} \tag{33}$$

We can therefore plot the stability regions for the 2-step Adams-Bashforth method as follows. First, prescribe a range of values of $\Delta t\lambda_l$. For each value of $\Delta t\lambda_l$, compute the two roots of (32) using (33). Next, determine which root has the largest modulus (call it $\zeta_{max}$) and store the modulus, $|\zeta_{max}|$, for the given value of $\Delta t\lambda_l$. The following code uses this procedure to plot the absolute stability region of the 2-step Adams-Bashforth method. The code that implements this procedure is provided in the snippets below.

Listing 2: Code snippet for the absolute stability region for AB2

```
clear all, close all, clc

%Range of values for dt*lambda_l
dtl_r = -1 : 0.0025 : 0;
dtl_i = -0.81 : 0.0025 : 0.81 ;

%make meshgrid of values
[dtlr, dtli] = meshgrid( dtl_r, dtl_i );

%write as complex number
dtl = dtlr + 1i*dtli;
```

Let us make the example more vivid with a figure:

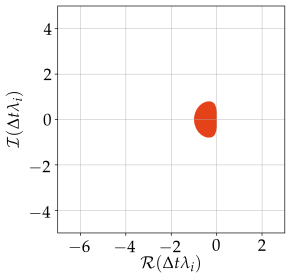



Figure 5: Absolute stability region for the 2-step Adams-Bashforth method.

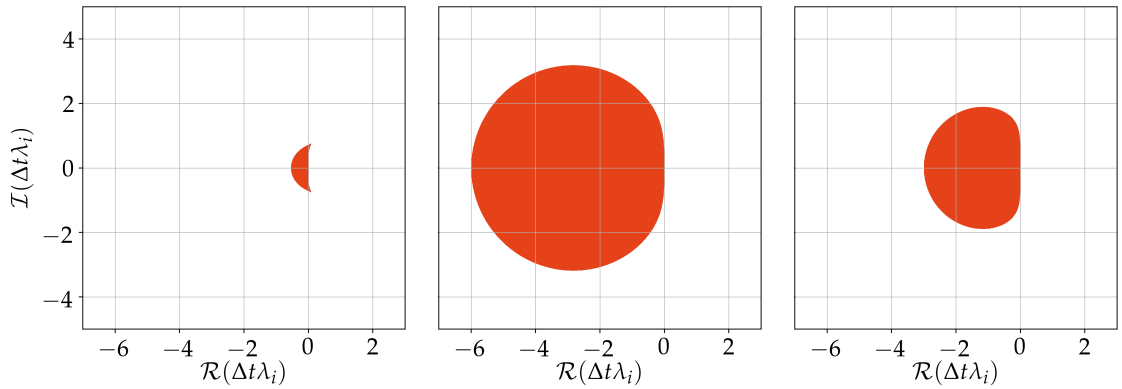
```
%initialize the root matrix
r = zeros( size( dtl ) );

%For each value of dt*lambda_l...
for j = 1 : length( dtl(:,1))
   for jj = 1 : length( dtl(1,:))

   %0^th order coeff
   p0 = 1/2*dtl(j,jj);

   %1^st order coeff
   p1 = -1-3/2*dtl(j,jj);

   %2^nd order coeff
   p2 = 1;

   %Get largest root and store its modulus
   r(j,jj) = max(abs(roots( [p2, p1, p0] )));

   end
end

%saturate for easy coloring
r( abs(r) > 1 ) = 2;
r( abs(r) <= 1 ) = 1;

%UIUC colormap
cmap = [1 1/2 0; 1 1 1];

%Plot stability region
contourf( dtlr, dtli, r, [1 2], 'edgecolor','none')
shading flat
colormap( cmap )
axis equal
```

Note the use of the built-in `roots` command in place of using (33) directly. This handy function makes it easier to generalize the procedure for plotting stability regions to cases where the polynomial function is higher order.

For completeness, we show the stability regions for other Adams methods in figure 6. These figures were created using appropriately adapted versions of the above code.

## 4 *Convergence*

Let us synthesize our results about truncation error and absolute stability to determine when a method applied to an IVP will be accurate.




(a) (b) (c)

Figure 6: Stability region for the 3-step Adams-Bashforth (a), 2-step Adams-Moulton (b), and 3-step Adams-Moulton (c) methods.

**Convergence of numerical methods for IVPs**

A numerical method for a method is *convergent* if and only if it has a truncation error that goes to zero as $\Delta t \rightarrow 0$ and has an absolute stability region that includes $\lambda_l \Delta t = 0$. Moreover, if the method is convergent, the rate of convergence of the method is equal to the order of the truncation error (that is, the global error will decay at the same rate as the truncation error).