ODE Lecture Notes
Amath 584

David George
University of Washington

Fall 2007

1 Introduction

In these notes we will look at one type of numerical method (finite difference methods) for one problem–a first order initial value problem of the form

$$y'(t) = f(y,t), \tag{1a}$$
$$y(t_0) = y_0, \tag{1b}$$

where in general $y \in \mathbb{C}^m$ is a vector, and $f(y,t) \in \mathbb{C}^m$ is a vector valued, possibly nonlinear, function[1]. The goal will be to find a numerical approximation to the solution $y(t)$ over some fixed interval $t = [t_0, T]$, as a discrete set of values $Y_i \in \mathbb{C}^m$, $i = 0, \ldots, N$ where Y_i is a pointwise approximation to $y(t_i)$. This means that we have broken the domain $t = [t_0, T]$ into a set of points $t_0, t_1, \ldots, t_N$, with $t_N = T$, and we find an approximation to the continuous solution at those discrete points (see Figure 1).

Note that the system (1) is more general that it might first appear, since you can turn a system of ODEs with higher-order derivatives into a first-order system by adding components to the solution vector $y(t)$. For instance, consider the following example of a set of two second order ODEs

$$x''(t) = h(x, z, x', z'), \tag{2a}$$
$$z''(t) = g(x, z, x', z'), \tag{2b}$$

where x and z are scalar functions, with initial conditions $x(0) = a$, $z(0) = b$, $x'(0) = c$ and $z'(0) = c$. By letting

$$y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} x \\ z \\ x' \\ z' \end{bmatrix}, \tag{3}$$

we can derive a first order system the governs the vector $y(t)$:

$$y' = f(y), \tag{4}$$

[1] We will assume that f satisfies properties such that our solution is unique and multiply differentiable, etc. etc. but not concern ourselves with such issues here.

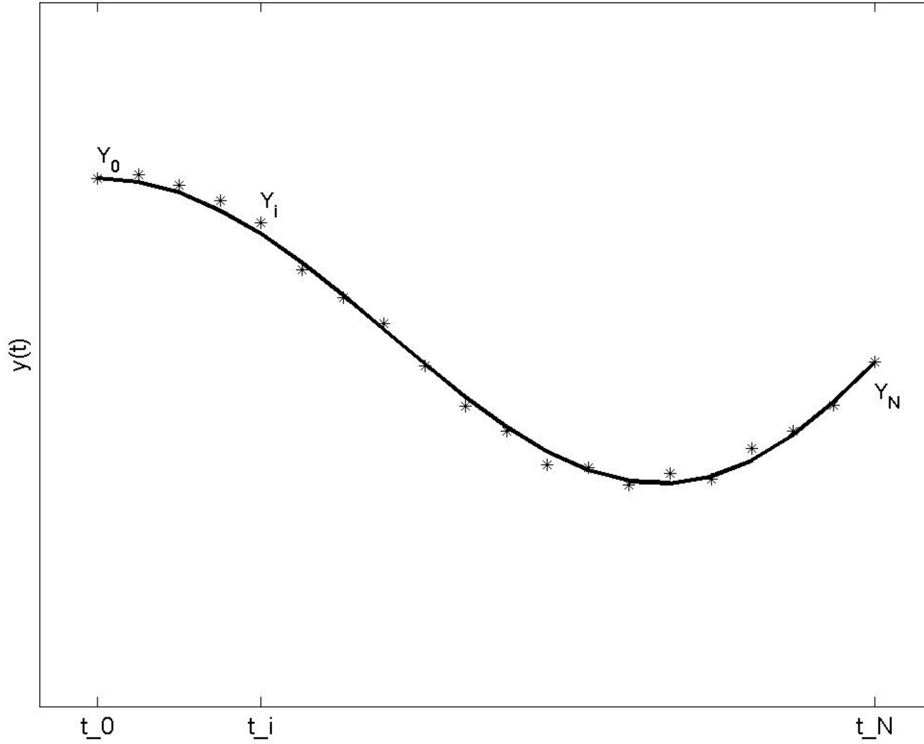

Figure 1: Numerical approximation for $y(t)$ is the set of values Y_i, $i = 0, \ldots, N$, where Y_i is the discrete approximation to $y(t_i)$, for $i = 0, \ldots, N$.

where

$$f(y) = \begin{bmatrix} x' \\ z' \\ h(x, z, x', z') \\ g(x, z, x', z') \end{bmatrix} = \begin{bmatrix} y_2 \\ y_3 \\ h(y) \\ g(y) \end{bmatrix}, \tag{5}$$

with initial conditions

$$y(0) = \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}. \tag{6}$$

So by being able to solve (1), we really can solve any higher-order nonlinear system. Additionally, we can eliminate the direct dependence of $f(y, t)$ on t, by letting t be a component of the solution vector $y \in \mathbb{C}^m$. That is, we add another component to y, which is t. The governing ODE for that last component is just $(y_{m+1})' = t' = 1$. This is not necessary to use the numerical methods we'll

discuss, and is in fact rarely done numerically. However, it will save a lot of typing by letting us focus on the system:

$$y' = f(y), \tag{7a}$$
$$y(t_0) = y_0, \tag{7b}$$

where the direct dependence of f on t has been removed. (Of course, in reality the system still depends on time, but that dependence occurs through the influence of the last component of y.)

2 Problem Stability

Just as we were interested in the condition of a problem in linear algebra (which is not a property of the numerical method), we will be interested in "problem stability" in ODEs. Problem stability is not a property of the numerical method, but a property of the ODEs themselves. The accuracy of our numerical method will depend on problem stability, just as in many linear algebra problems our relative error was related to the condition of a matrix.

Problem stability indicates how a perturbation to the initial problem (7) changes the solution $y(t)$. That is, suppose that (7) is changed by adding a small perturbation ϵ to the initial data:

$$y'(t) = f(y), \tag{8a}$$
$$y(t_0) = y_0 + \epsilon. \tag{8b}$$

If we let $\tilde{y}(t)$ be the solution to the perturbed initial value problem (8), and let $\bar{y}(t)$ be the solution to the original problem (7), then if

$$||\tilde{y}(t) - \bar{y}(t)|| \tag{9}$$

grows in time, the problem is said to be unstable, and if it shrinks as time proceeds the problem is stable. (The norm can be any norm due to norm equivalence.) Note that "problem" refers not only to the governing ODE but also the initial condition. Some ODEs can be stable for some initial condition and unstable for others. A real, scalar problem is stable at a given state $\bar{y}(t)$, if

$$f'(\bar{y}) < 0, \tag{10}$$

at that state. Similar expression can be derived if y is more general. If $y \in \mathbb{C}$, then only the real part of $f'(\bar{y})$ need be considered (assuming f is analytical), and needs to be less than 0 for stability. If $y \in \mathbb{C}^m$, and if the Jacobian matrix $f'(\bar{y})$ is diagonalizable, the real part of all eigenvalues should be considered, and need to be less than zero for problem stability. (For problems where the Jacobian is defective, stability can get much more complicated, and we will totally ignore such cases—except maybe on the exam (just kidding)). The reason why this implies stability is explained heuristically below.

If we let $z(t) = \tilde{y}(t) - \bar{y}(t)$, then $z(t)$ obeys the governing IVP:

$$z'(t) = f(\bar{y}) - f(\tilde{y}), \tag{11a}$$
$$z(t_0) = \epsilon. \tag{11b}$$

If the norm of the solution to (11) grows in time, the original problem (7) is unstable, if it damps to zero in time the original problem is stable, and if it remains constant the original problem is sometimes said to be "neutrally stable."

In order to analyze (11) for nonlinear functions, we will do a form of linear stability analysis. The idea is as follows. We know $\bar{y}(t_0)$, it is just the initial condition of the original problem (7). We want to consider arbitrary (yet "small") values for ϵ. We then do a Taylor expansion of the right hand side of the ODE in (11), about the state $\bar{y}$ which gives

$$f(\bar{y}) - f(\tilde{y}) = f'(\bar{y})(\bar{y} - \tilde{y}) + O(z^2) = f'(\bar{y})z + O(z^2). \tag{12}$$

(Again, in the case of systems where y is a vector, the term f' refers to the Jacobian matrix in the Taylor expansion of f. Now, if ϵ is small enough we can ignore $O(z^2)$ initially, which implies that $z(t)$ is initially governed by something close to

$$z'(t) = f'(\bar{y})z, \tag{13a}$$
$$z(t_0) = \epsilon. \tag{13b}$$

This is a linear system for z resembling

$$z'(t) = \lambda z, \tag{14a}$$
$$z(t_0) = z_0 = \epsilon, \tag{14b}$$

the solution to which is of course $z(t) = z_0 e^{\lambda(t-t_0)}$. If $\Re(\lambda) < 0$, then $z(t)$ decays to zero, meaning that $\tilde{y}$ approaches $\bar{y}$, and $O(z^2)$ terms in (12) can continue to be neglected. Now, $f'(\bar{y})$ is really a function of time in general, since $\bar{y}$ does not need to be stationary. So problem stability can change as the solution evolves if the sign of the real part of $f'(\bar{y})$ changes. (For problems where $\bar{y}$ is constant, such as if it is a critical point of f, stability doesn't change as the solution evolves.)

If $y \in \mathbb{C}^m$, we might write (13) as

$$z'(t) = Az, \tag{15a}$$
$$z(t_0) = z_0 = \epsilon, \tag{15b}$$

where $A \in \mathcal{C}^{m\times m}$ is the m by m Jacobian $f'(\bar{y})$. The solution to (15) is

$$z(t) = e^{At} z_0, \tag{16}$$

where $e^{At} \in \mathcal{C}^{m\times m}$ is the matrix exponential, defined by the series

$$e^{At} = I + At + \frac{A^2t^2}{2} + \cdots + \frac{A^k t^k}{k!} + \cdots . \tag{17}$$

If A is diagonalizable

$$A^k = (R\Lambda R^{-1})^k = (R\Lambda R^{-1})(R\Lambda R^{-1})\cdots(R\Lambda R^{-1}) = R\Lambda^k R^{-1}, \tag{18}$$

since all of the $R^{-1}R$ terms are the identity. We can therefore write the series for the matrix exponential as

$$e^{At} = R(I + \Lambda t + \frac{\Lambda^2 t^2}{2} + \cdots + \frac{\Lambda^k t^k}{k!} + \cdots)R^{-1}, \tag{19a}$$
$$\Rightarrow R^{-1}e^{At}R = I + \Lambda t + \frac{\Lambda^2 t^2}{2} + \cdots + \frac{\Lambda^k t^k}{k!} + \cdots = e^{\Lambda t}. \tag{19b}$$

The significance of this, is that $e^{\Lambda t}$ is diagonal, since all the terms in its series are diagonal (the products of diagonal matrices are still diagonal.) That is

$$e^{\Lambda t} = \begin{bmatrix} e^{\lambda^1} & & & \\ & e^{\lambda^2} & & \\ & & \ddots & \\ & & & e^{\lambda^m} \end{bmatrix}, \tag{20}$$

where λ^j is the jth eigenvalue of A. So we have

$$z(t) = e^{At} z_0 \Rightarrow R^{-1} z(t) = R^{-1} e^{At} z_0 \Rightarrow w(t) = R^{-1} e^{At} R w_0, \tag{21a}$$

$$\Rightarrow w(t) = e^{\Lambda t} w_0, \tag{21b}$$

where $w_0 = R^{-1} z_0$. In other words, by changing to the basis of eigenvectors, we see that each component w^j of the solution obeys the scalar equation

$$w^j(t) = e^{\lambda^j t} w_0^j. \tag{22}$$

Therefore, if the real part of all of the eigenvalues of $f'(\bar{y})$ have negative real parts, each component of $w(t)$ decays, so $z(t)$ decays, and $\tilde{y}$ approaches $\bar{y}$. When looking at a general nonlinear system then, it is the eigenvalues of the Jacobian at any state $\bar{y}$ that determine problem stability.

When doing linear stability analysis, terms such as "small" when referring to ϵ, depend on the relative size of first and higher order terms in the Taylor expansion (12), which depends on the specific problem at hand since it depends on the size of $f''(\bar{y})$ in relation to $f'(\bar{y})$ etc. etc. Don't be too concerned about whether this analysis is always valid. It depends on the problem. For some problems linear stability analysis is sufficient and others it may not be. (A good example is when the real part of all eigenvalues is zero. Stability then depends on higher order terms in the Taylor expansion.) For our purposes, we are content to examine how our numerical methods will behave on stable systems. This section therefore, is really just an explanation of why later we will focus on the test problem

$$u'(t) = \lambda u, \tag{23a}$$

$$u(t_0) = u_0. \tag{23b}$$

If we consider how a numerical method treats (23), it gives us an indication of how it will treat more general problems where the true solution is perturbed from $\bar{y}(t)$ to $\tilde{y}(t)$ at any time.

3 Finite Difference Methods

Finite difference methods provide formulas to generate a numerical sequence

$$U_0, U_1, U_2, \ldots, U_N, \tag{24}$$

as an approximation to the solution $u(t)$ of the ODE:

$$u'(t) = f(u), \tag{25a}$$

$$u(t_0). \tag{25b}$$

Each U_n is an approximation to $u(t_n)$, and $[t_0, t_N] = [t_0, T]$, the time interval we want the solution over. There are various ways to derive a given finite difference method for a set of ODEs. Since (25) is a first order system, one way to derive a given method is to use quadrature—formulas for approximating to integrals. That is, note that for any Δt

$$u(t+\Delta t) - u(t) = \int_t^{t+\Delta t} u'(\tau)\,d\tau = \int_t^{t+\Delta t} f(u(\tau))\,d\tau = \int_t^{t+\Delta t} F(\tau)\,d\tau. \tag{26}$$

We can derive various finite difference methods by approximating in various ways the last integral in (26). For instance, if we look at two points t_{n+1} and t_n, we have

$$u(t_{n+1}) - u(t_n) = \int_{t_n}^{t_{n+1}} f(u(t))\,dt \approx \Delta t_n f(u(t_n)), \tag{27}$$

where $\Delta t_n = (t_{n+1} - t_n)$. Of course, the smaller Δt_n is, the closer the quadrature approximations are to the integrals. Unless otherwise stated, assume that each Δt_n is the same, so we will just call it Δt. However this is not strictly necessary in any of the methods we will look at. This quadrature approximation gives rise to a numerical method, known as Forward Euler:

$$U_{n+1} = U_n + \Delta t f(U_n), \tag{28a}$$

$$U_0 = u(t_0). \tag{28b}$$

Note that we just start with U_0 and then build the sequence $U_1, U_2, \ldots, U_N$. Since finding U_{n+1} only requires evaluating $f(U_n)$, the method is known as *explicit*. Another quadrature approximation gives rise to Backward Euler:

$$U_{n+1} = U_n + \Delta t f(U_{n+1}), \tag{29a}$$

$$U_0 = u(t_0). \tag{29b}$$

Note that now we need to solve (29a) for U_{n+1} at each time-step. Unless $f(u)$ is linear, this may require a root solving iteration. The solution U_{n+1} is the root of the equation

$$G(U_{n+1}) \equiv U_{n+1} - U_n - \Delta t f(U_{n+1}). \tag{30}$$

If we use Newton iteration, this requires finding $G'(q)$, which we can only do if we can find $f'(q)$. For systems, this requires finding the Jacobian of $G(q)$ analytically (on paper). Be sure and realize that U_n is a constant when finding the Jacobian. If the function f is too complicated to be able to do this, some other form of root finding would be needed (such as secant iteration). For Newton iteration, you would iterate by solving

$$G'(q^{(k)})\delta q = -F(q^{(k)}), \tag{31a}$$

$$q^{(k+1)} = q^{(k)} + \delta q, \tag{31b}$$

until some tolerance is reached so that $q^{(k)}$ is close enough to a root. The test might be $||G(q^{(k)}|| < \epsilon$, for some small epsilon, such as $\epsilon_{\text{machine}}$. Then set $U_{n+1} = q^{(k)}$. Note that you need some $q^{(0)}$ to start the iteration. A good choice is $q^{(0)} = U_n$, since we presume that U_n will typically not be too far from the root U_{n+1}, especially if Δt is small.

Schemes that require solving implicit functions at each time-step (due to the $f(U_{n+1})$) such as Backward Euler, are called *implicit*. Implicit schemes therefore come with an added cost, however, as we will see when we talk about stability, they convey some benefit, and are in fact imperative for some ODEs.

Another implicit scheme is the Trapezoid Method:

$$U_{n+1} = U_n + \frac{\Delta t}{2}\left(f(U_n) + f(U_{n+1})\right), \tag{32a}$$

$$U_0 = u(t_0). \tag{32b}$$

The Trapezoid Method (32) would also need Newton iteration (31).

The methods we have discussed so far are all *1-step* methods, since U_{n+1} only depends directly on U_n. More generally, we might define *Linear Multistep Methods*, of the form

$$\sum_{j=0}^{r} \alpha_j U_{n+j} = \sum_{j=0}^{r} \beta_j f(U_{n+j}), \tag{33}$$

where α_j and β_j are constants that define this general r-step method. Even the 1-step methods we've looked at above are Linear Multistep Methods, with $r = 1$. Note that if $\beta_r = 0$ the method is explicit, otherwise it is implicit. An example of an explicit 2-step method (r=2), is the Midpoint Method:

$$U_{n+1} = U_{n-1} + 2\Delta t f(U_n). \tag{34}$$

You might notice that we need to find U_1 some other way, before we can start using (34). Typically some other 1-step method would be used to find U_1, and then then (34) is used for $U_2, U_3, \ldots, U_N$. More generally, an r-step method must find the first $U_0, U_1, \ldots, U_{r-1}$ starting values using other methods. However, since only a small number of steps are used to generate these values, it doesn't change the properties of the method overall.

3.1 Global Error

The global error of our numerical method is simply the distance between the numerical solution and the actual solution in some norm. It is defined at each point t_n, $n = 0, 1, \ldots, N$. That is

$$e_n = ||U_n - u(t_n)||. \tag{35}$$

3.2 Truncation Error

Ultimately we are interested in the global error, however, it can be difficult to ascertain the global error directly, since of course we don't know the true solution generally. The truncation error is another type of measure of the methods accuracy. Note that our numerical solution sequence $\{U_n\}_{n=0}^{N}$ is the solution to the difference equation

$$\sum_{j=0}^{r} \alpha_j U_{n+j} - \sum_{j=0}^{r} \beta_j f(U_{n+j}) = 0. \tag{36}$$

However, if we plug the true solution into (36) as

$$\sum_{j=0}^{r} \alpha_j u(t_{n+j}) - \sum_{j=0}^{r} \beta_j f(u(t_{n+j})), \tag{37}$$

it will not be equal to zero. That is, if we turn the true solution into a discrete set of points $u(t_0), u(t_1), \ldots, u(t_N)$ by evaluating the true solution at the points $t_0, t_1, \ldots, t_N$, it won't satisfy the same difference equation (36) that the numerical solution satisfies. The truncation error is a measure of how far the difference equation is from the true ODE.

This is most easily explained by example. Consider Forward Euler (28). To find the truncation error, we arrange (28) so that it resembles the actual ODE, not (26). That is, we rearrange Forward Euler as

$$\frac{U_{n+1} - U_n}{\Delta t} - f(U_n) = 0. \tag{38}$$

The numerical solution is the solution to this difference equation. Now, the truncation error is defined by

$$\frac{u(t_{n+1}) - u(t_n)}{\Delta t} - f(u(t_n)) = \tau_n. \tag{39}$$

The truncation error tells us how good or bad our difference equation approximates the actual ODE. Now, we need to find a more informative value for τ_n. The trick is to use Taylor expansions of $u(t)$, about the point t_n. Note that

$$\frac{u(t_{n+1}) - u(t_n)}{\Delta t} = u'(t_n) + \frac{u''(t_n)\Delta t}{2} + \frac{u'''(t_n)\Delta t^2}{3!} + O(\Delta t^3) \tag{40}$$

Because of the presence of the Δt term on the right hand side of (45), the left hand side is known as a “first-order” approximation to $u'(t_n)$. Now, we know from the ODE that $f(u(t_n)) = u'(t_n)$. So plugging that and (45) into (39) gives

$$\tau_n = \frac{u''(t_n)\Delta t}{2} + \frac{u'''(t_n)\Delta t^2}{3!} + O(\Delta t^3). \tag{41}$$

In fact, we often write that

$$\tau_n = \frac{u''(t_n)\Delta t}{2}, \tag{42}$$

since that indicates the lowest order error. Since this involves Δt, it is known as a “first-order” method. If the lowest order term was a constant times Δt^2, we would say that it is a “second-order” method, and so on. Therefore, “higher-order” methods are more accurate, since the truncation error is smaller as Δt gets small.

As one more example, we'll look at the Trapezoid Method. We write the method as

$$\frac{U_{n+1} - U_n}{\Delta t} - \frac{1}{2}\left(f(U_n) + f(U_{n+1})\right) = 0. \tag{43}$$

The truncation error is defined by

$$\frac{u(t_{n+1}) - u(t_n)}{\Delta t} - \frac{1}{2}\left(f(u(t_n)) + f(u(t_{n+1})\right) = \tau_n. \tag{44}$$

We can again use (45) for the left hand side of (44), and again use the fact that $f(u(t_n)) = u'(t_n)$. So it remains to find $f(u(t_{n+1}))$. The trick is to note that $f(u(t_{n+1})) = u'(t_{n+1})$, and then do a Taylor expansion of $u'(t_{n+1})$ about t_n. This gives

$$f(u(t_{n+1})) = u'(t_{n+1}) = u'(t_n) + u''(t_n)\Delta t + \frac{u'''(t_n)\Delta t^2}{2} + O(\Delta t^3). \tag{45}$$

Now when we plug these expressions into (44), we get

$$\tau_n = \frac{-u''(t_n)\Delta t^2}{12} + O(\Delta t^3). \tag{46}$$

So we see that the Trapezoid Method is "second-order."

3.3 One-step Error

The truncation error is related to the one-step error. The one-step error is actually easier to understand, but is not as useful theoretically as the truncation error. The one-step error can always be found by the truncation error, and some authors refer to the one-step error as the truncation error. The one-step error $\mathcal{L}_n$ is the error that is made in one-step of the numerical method. That is, if we consider $U_n, U_{n-1}, \ldots, U_0$ to be exact, the one-step error is the difference $U_{n+1} - u(t_{n+1})$. The norm of this is not the global error, because in reality, $U_n, U_{n-1}, \ldots, U_0$ are not exact. As an example, for Forward Euler, the one step error is

$$\mathcal{L}_n = u(t_{n+1}) - u(t_n) - \Delta t f(u(t_n)). \tag{47}$$

In this case, the one-step error is just Δt times the truncation error. More generally, the one step error is $C\Delta t\tau_n$, for some constant C for the method. A naive analysis might suggest that the global error at t_N, $e_N = ||U_N - u(t_N)||$ is the sum of all of the one-step errors, on the order of $O(\mathcal{L}_n N) = O(\mathcal{L}_n(T/\Delta t)) = O(\tau^n)$. However, this is only true if the one-step errors do not accumulate adversely. This involves the numerical stability of the method.

3.4 Consistency

Consistency of a numerical method merely means that the truncation error goes to zero as $\Delta t \to 0$. So a method is said to be consistent if

$$\tau_n = O(\Delta t^k), k \geq 1. \tag{48}$$

Alternatively, the one-step error must satisfy $\mathcal{L}_n = O(\Delta t^2)$ or better. (The one-step error would always be $O(\Delta t)$ or better, since step size is only Δt.)

3.5 Convergence

Convergence of a method means that as $\Delta t \to 0$, $U_N \to u(t_N)$. More precisely, if we let $t_0 = 0$, we say that a method on a problem (25) is *convergent*, if

$$\lim_{\Delta t \to 0, N\Delta t = T} ||U_N - u(t_N)|| = 0. \tag{49}$$

We say that a *method is convergent*, if (49) is satisfied for all T, and all problems (25) (where $f(u)$ guarantees a unique differentiable solution for all finite T.)[2]

4 Numerical Stability

Numerical stability addresses the question of how numerical errors accumulate. There are different types of numerical stability. We will only look at one type in any detail known as absolute stability. However, here are the main types of stability as a summary.

4.1 Types of Numerical Stability for ODEs

4.1.1 Zero Stability

Zero stability addresses whether convergence is achieved. Therefore, it is only concerned with the stability of the method in the limit as $\Delta t \to 0$. In fact, we can say that for all Linear Multistep Methods (33)

$$\text{Consistency} + \text{Zero Stability} \Leftrightarrow \text{Convergence}. \tag{50}$$

Furthermore, the global error will typically be of the same order as the truncation error. We always hope to use a convergent method, so we always expect zero stability. (For nonlinear PDEs, this might even be a lofty goal!). Determining zero stability of a method will be explained when we look at a more general type of stability known as absolute stability.

4.1.2 Absolute Stability

Absolute stability is a more general type of stability theory used to investigate how error accumulates even for finite Δt, not necessarily in the limit $\Delta t \to 0$. This is often much more practical and useful, since in practice we always use some finite Δt in applications. Absolute stability is more relevant, particularly in the field of scientific computation, where scientists simulate "real world" problems. Unfortunately, in practice, often it is erroneously assumed that convergence is all that matters. As we will see, a zero-stable convergent method can be unstable for all finite Δt. (You saw this in your homework—Euler's method is a convergent method, but is unstable for the pendulum problem.) We will look at absolute stability in detail in Section 5. Absolute stability theory is usually confined to stable problems. For unstable problems, absolute stability of a numerical method will not tell you much about global error.

[2]From ODE theory, $f(u)$ must be Lipschitz continuous for such a result.

4.1.3 Relative Stability

One might ask, as evidenced in class, "what if I'm dealing with an unstable problem." Relative stability tells you how the relative error in the solution grows or decays, even if the solution is unstable. Additionally, one might hope that even for a stable problem, the error should decay *faster* than the true deviation $\bar{y} - \tilde{y}$, which itself decays for a stable problem. Relative stability can be a complicated subject. If you are interested, you might do a Google search on "order stars and relative stability." We obviously won't address this fascinating topic. (No, "fascinating" wasn't meant as sarcasm.)

5 Absolute Stability

As mentioned, absolute stability tells how error decays for a finite time-step Δt. It is usually only informative for stable, or neutrally stable problems. We consider the scalar "test problem"

$$u'(t) = \lambda u. \tag{51}$$

Remember that we are thinking of this test problem as the governing equation for the difference between the true solution and a perturbed solution. Since we are considering linear methods, the numerical solution will treat this deviation similarly to the way that it treats (51). Conveniently then, we will isolate ourselves to (51). The idea is best introduced by example. To adopt standard notation in the literature, let's denote $\Delta t = k$. For Forward Euler, the numerical solution satisfies the difference equation

$$U_{n+1} = U_n + kf(U_n) = U_n + k\lambda U_n. \tag{52}$$

Solving for U_{n+1} gives

$$U_{n+1} + (1 + k\lambda)U_n. \tag{53}$$

Iterating, this implies that the numerical solution at any t_n is

$$U_{n+1} = (1 + k\lambda)^n U_0. \tag{54}$$

Now, we look at values of $k\lambda$ such that the solution either grows or decays. We allow λ to be complex. Now, if $|1+k\lambda| > 1$, the modulus of $U_n \in \mathbb{C}$ grows as n increases. It decays if $|1+k\lambda| < 1$, and remains constant if $|1 + k\lambda| = 1$. We can graph these regions of the complex plane $z = \lambda k$. The stable region $|1 + k\lambda| < 1$ is separated from the unstable region $|1 + k\lambda| > 1$, by the unit circle $|1 + k\lambda| = 1$, centered at $k\lambda = -1$, and intersecting the origin. This unit circle is known as the stability region for Forward Euler. For a given λ, this means that k must be chosen such that $k\lambda$ is in the stability region, if we expect the solution to (51) to decay. This is somewhat nonintuitive—for very stable problems, where $\Re(\lambda) << 0$, we would need a very small step size k to put λk in the stable region guaranteeing a decaying numerical solution. Otherwise the numerical solution would grow in magnitude, even for a true solution that is rapidly decaying to zero! Forward Euler is therefore usually not a good choice for problems with eigenvalues large in magnitude, with negative real parts.

Backward Euler has a very different stability region. Looking at the test problem again, we have

$$U_{n+1} = U_n + kf(U_{n+1}) = U_n + k\lambda U_{n+1}. \tag{55}$$

Solving for U_{n+1} gives

$$U_{n+1} + \frac{1}{(1-k\lambda)}U_n. \tag{56}$$

Iterating, this implies that the numerical solution at any t_n is

$$U_{n+1} = \left(\frac{1}{(1-k\lambda)}\right)^n U_0. \tag{57}$$

This means that $|\frac{1}{(1-k\lambda)}| = 1$ separates the stable from the unstable region. The stable region occurs where $|1-k\lambda| > 1$. The boundary is the unit circle in the complex plane $z = k\lambda$, centered at $k\lambda = 1$. The stable region is therefore the exterior of this unit circle. Therefore the entire left half plane and more is the stable region for Backward Euler. Therefore, for a stable problem, we can take whatever step-size we wish with Backward Euler, and the numerical solution will still decay to zero. Therefore our step size is not limited by stability, but only by accuracy concerns due to the truncation error. This is true more generally of implicit methods—they tend to have stability regions that include the entire left half plane. That is the justification for the added computational cost of needing to do Newton iteration. (Interestingly, since Backward Euler has a stability region the contains much of the right half plane, even for unstable problems where $\Re(\lambda) > 0$, a large step size with Backward Euler will produce a decaying numerical solution!)

It is easy to show that the Trapezoid Method has a stability region that is exactly the left half plane, with the boundary being the imaginary axis. It is possible, but somewhat more difficult to show that the Midpoint method has a stability region that is only the interval $(-i, i)$ on the imaginary axis. The explicit Midpoint Method is therefore the cheapest choice for problems that are neutrally stable, with $\Re(\lambda) = 0$, *i.e.* imaginary eigenvalues. Of course, k must be less in magnitude than $1/|\lambda|$ for such problems, so that $k\lambda$ is in the stability region, keeping the solution from growing in magnitude. (Note that the solution will not decay in such cases, because we are always on the boundary of the stability region. The same is true for the Trapezoid Method, since the imaginary axis is the boundary. Note also that for neutrally stable problems with imaginary eigenvalues, Forward Euler will always produce a growing numerical solution with finite k, and Backward Euler will always produce a decaying numerical solution with finite k, since λk is on the imaginary axis—the exterior and interior of the stability regions of those methods respectively. However, both methods will be convergent as $k \to 0$).

5.0.4 Relation to Zero Stability

For Linear Multistep Methods, they are zero stable if the origin of the complex plane is on the boundary of their absolute stability regions. All of the methods discussed are therefore zero stable.