AME50541: Finite Element Methods
Numerical Solution of Ordinary Differential Equations

1 Ordinary differential equations

An ordinary differential equation (ODE) is an equation of one (scalar) or multiple (system) functions of a single *independent* variable and their derivatives. ODEs differ from partial differential equations (PDEs) in that ODEs depend on a single independent variable while PDEs depend on multiple. A scalar ODE takes the general form

$$F\left(t, y, \frac{dy}{dt}, \ldots, \frac{d^m y}{dt^m}\right) = 0,$$

where $t \in [0, T]$ is the independent variable, y is the unknown function, and F is the linear or nonlinear function defining the ODE. The highest derivative of the unknown function y appearing in the equation, m in the case, is called the *order* of the ODE. An ODE of order m must be augmented with m boundary conditions. Since we are most interest in studying ODEs that model time-varying phenomena, we will only consider *initial value problems*, i.e., problems where all boundary conditions are prescribed at $t = 0$, for example,

$$y(0) = \bar{y}, \quad \frac{dy}{dt} = \bar{y}', \quad \ldots \quad , \frac{d^{m-1} y}{dt^{m-1}} = \bar{y}^{(m-1)}, \tag{1}$$

where $\bar{y}, \bar{y}', \ldots, \bar{y}^{(m-1)} \in \mathbb{R}$ are known constants. If F can be written as a linear combination of the unknown function y and its derivatives, the ODE is linear; otherwise, it is nonlinear. That is, the ODE is linear if F can be written as

$$F\left(t, y, \frac{dy}{dt}, \ldots, \frac{d^m y}{dt^m}\right) = c(t) + a_0(t) y(t) + \sum_{i=1}^{k} a_k(t) \frac{d^k y}{dt^k}(t),$$

where $a_k(t)$ for $k = 0, \ldots, m$ and $c(t)$ are known functions of the independent variable.

Example: Linear, scalar model problem
The simplest ODE is the linear, first-order differential equation

$$\frac{dy}{dt} + \lambda y = 0 \quad t \in (0, T), \qquad y(0) = \beta, \tag{2}$$

where $\lambda, \beta \in \mathbb{C}$ and $\Re(\lambda) \geqslant 0$. This ODE is commonly used to analyze the stability and accuracy of numerical schemes for solving ODEs. It can easily be verified that the exact solution of this ODE is $y(t) = \beta e^{-\lambda t}$. The requirement that $\Re(\lambda) \geqslant 0$ guarantees the solution of the ODE does not diverge to infinity as $t \to \infty$.

Example: Oscillations of a pendulum
The governing equation for the (time-dependent) angle $\theta(t)$ that a pendulum of length ℓ makes with the vertical axis given it is released from an angle of $\theta(0) = \theta_0$ with a velocity of $\frac{d\theta}{dt}(0) = \dot{\theta}_0$ is a nonlinear, second-order ODE

$$\frac{d^2\theta}{dt^2} + \frac{g}{\ell} \sin\theta = 0 \quad t \in (0, T), \qquad \theta(0) = \theta_0, \quad \frac{d\theta}{dt}(0) = \dot{\theta}_0.$$

Example: Damped harmonic oscillator
The governing equation for a damped harmonic oscillator of mass m with damping ratio c, spring constant k, and force vector F is a second-order, linear ODE

$$m\frac{d^2 x}{dt^2} + c\frac{dx}{dt} + kx = F, \tag{3}$$

where $x(t)$ is the displacement of the mass.

A *system* of N ordinary differential equations is a system of N equations in N unknown functions of a single independent variable and their derivatives. A system of ODEs of order m takes the general form

$$\boldsymbol{F}\left(t, \boldsymbol{y}, \frac{d\boldsymbol{y}}{dt}, \ldots, \frac{d^m\boldsymbol{y}}{dt^m}\right) = 0, \tag{4}$$

where $t \in [0,T]$ is the independent variable, $\boldsymbol{y}(t) = (y_1(t), \ldots, y_N(t))^T$ are the N unknown functions, and $\boldsymbol{F}(t, \boldsymbol{y}, \frac{d\boldsymbol{y}}{dt}, \ldots, \frac{d^m\boldsymbol{y}}{dt^m})$ is the linear or nonlinear vector-valued function with N entries defining the system of ODEs. A system of ODEs reduces to a scalar ODE in the special case where $N = 1$.

Example: Mass-spring network

Consider a mass-spring network consisting of N masses $m_1, \ldots, m_N$ and M springs with stiffnesses $k_1, \ldots, k_M$, connected in some configuration. Let the time-dependent displacement of each mass be denoted $x_i(t)$ for $i = 1, \ldots, N$. The governing equations for the mass-spring system is the second-order system of N linear ODEs

$$\boldsymbol{M}\ddot{\boldsymbol{x}} + \boldsymbol{K}\boldsymbol{x} = \boldsymbol{F}(t) \quad t \in (0,T), \quad \boldsymbol{x}(0) = \boldsymbol{x}_0, \quad \frac{d\boldsymbol{x}}{dt}(0) = \dot{\boldsymbol{x}}_0$$

where $\boldsymbol{M} \in \mathbb{R}^{N\times N}$ is the diagonal matrix of masses $M_{ii} = m_i$ for $i = 1, \ldots, N$, $\boldsymbol{K} \in \mathbb{R}^{N\times N}$ is the stiffness matrix that encodes the connectivity of the mass-spring system and stiffness of each spring, $\boldsymbol{F}(t) \in \mathbb{R}^N$ is the external forcing applied to each mass, and $\boldsymbol{x}_0, \dot{\boldsymbol{x}}_0 \in \mathbb{R}^N$ are the displacement and velocity of each mass at $t = 0$.

Example: Semi-discretization of a partial differential equation

Any time-dependent partial differential equation of order $m = 1$ with respect to the time vairable that has been discretized solely in space (not in time), e.g., with the finite element method, reduces to a system of ordinary differential equations of the form

$$\boldsymbol{M}\frac{d\boldsymbol{u}}{dt} + \boldsymbol{r}(\boldsymbol{u}) = 0 \quad t \in (0,T), \quad \boldsymbol{u}(0) = \boldsymbol{u}_0,$$

where $\boldsymbol{u}(t)$ are the time-dependent coefficients of the finite element basis fucntions, $\boldsymbol{M}$ is the mass matrix, $\boldsymbol{r}(\boldsymbol{u})$ is the discretization of the spatial operator (or linear problems, it reduces to $\boldsymbol{r}(\boldsymbol{u}) = \boldsymbol{K}\boldsymbol{u} + \boldsymbol{F}$, where $\boldsymbol{K}$ is the stiffness matrix and $\boldsymbol{F}$ is the force vector), and $\boldsymbol{u}_0$ is the initial condition for the solution coefficients.

A system of N ODEs of degree m (4) can be re-cast into a system of mN first-order ODEs by introducing the following $m-1$ auxiliary functions

$$\boldsymbol{v}_k(t) := \frac{d^k\boldsymbol{y}}{dt^k}(t),$$

for $k = 1, \ldots, m-1$ and $t \in [0,T]$. The auxiliary variables are the time derivatives of the solution $\boldsymbol{y}(t)$, i.e., $\boldsymbol{v}_1$ is the velocity of $\boldsymbol{y}$ and $\boldsymbol{v}_2$ is the acceleration. The auxiliary variables are related to one another by $\boldsymbol{v}_k = \frac{d\boldsymbol{v}_{k-1}}{dt}$ for $k = 2, \ldots, m-1$. From the definition of these auxiliary variables, the governing equations can be re-written as

$$\boldsymbol{F}(t, \boldsymbol{y}, \boldsymbol{v}_1, \ldots, \boldsymbol{v}_{m-1}, \frac{d\boldsymbol{v}_{m-1}}{dt}) = 0, \quad \boldsymbol{v}_1 = \frac{d\boldsymbol{y}}{dt}, \quad \boldsymbol{v}_2 = \frac{d\boldsymbol{v}_1}{dt}, \quad \ldots, \quad \boldsymbol{v}_{m-1} = \frac{d\boldsymbol{v}_{m-2}}{dt}, \tag{5}$$

which is a system of mN equations in mN unknown functions, i.e., m unknown functions $(\boldsymbol{y}, \boldsymbol{v}_1, \ldots, \boldsymbol{v}_{m-1})$ each of size n. This can be written in compact notation as a single system of first-order ODEs

$$\boldsymbol{G}\left(t, \boldsymbol{w}, \frac{d\boldsymbol{w}}{dt}\right) = 0,$$

where $\boldsymbol{w}(t) = (\boldsymbol{y}(t), \boldsymbol{v}_1(t), \ldots, \boldsymbol{v}_{m-1}(t))^T$ and

$$\boldsymbol{G}\left(t, \boldsymbol{w}, \frac{d\boldsymbol{w}}{dt}\right) = \begin{bmatrix} \boldsymbol{F}\left(t, \boldsymbol{y}, \boldsymbol{v}_1, \ldots, \boldsymbol{v}_{m-1}, \frac{d\boldsymbol{v}_{m-1}}{dt}\right) \\ \boldsymbol{v}_1 - \frac{d\boldsymbol{y}}{dt} \\ \boldsymbol{v}_2 - \frac{d\boldsymbol{v}_1}{dt} \\ \vdots \\ \boldsymbol{v}_{m-1} - \frac{d\boldsymbol{v}_{m-2}}{dt} \end{bmatrix}.$$

This transformation makes the design of numerical schemes for solving ODEs of any order convenient since we only need to consider first-order ODEs; the scheme would apply to higher order ODEs after they have been converted to their first-order form.

Example: Damped harmonic oscillator, first-order form

To reduce the harmonic oscillator to a first-order system of ODEs, introduce the velocity $v = \frac{dx}{dt}$ and apply the transformation in (5). The second-order ODE in (3) becomes the following system of two first-order ODEs

$$m\frac{dv}{dt} + cv + kx - F = 0, \qquad \frac{dx}{dt} - v = 0.$$

This can be re-written compactly as $\boldsymbol{M}\frac{d\boldsymbol{w}}{dt} + \boldsymbol{K}\boldsymbol{w} + \boldsymbol{F} = 0$, where

$$\boldsymbol{w} = \begin{bmatrix} v \\ x \end{bmatrix}, \quad \boldsymbol{M} = \begin{bmatrix} m & 0 \\ 0 & 1 \end{bmatrix}, \quad \boldsymbol{K} = \begin{bmatrix} c & k \\ -1 & 0 \end{bmatrix}, \quad \boldsymbol{F} = \begin{bmatrix} -F \\ 0 \end{bmatrix}.$$

2 Numerical solution of ordinary differential equations

In this section, we develop numerical schemes for solving ordinary differential equations and assess their stability and accuracy. We begin by classifying various types of ODE solvers, which will setup notation and terminology for the remainder of the section. Subsequently, our development of numerical methods for solving ODEs begins by considering first-order scalar ODEs since this provides a straightforward starting point and will serve as the foundation for solving systems of ODEs. Then we repeat the construction for systems of first-order ODEs and conclude with a brief discussion of higher order scalar and systems of ODEs.

Throughout this section, we will consider differential equations defined in terms of an independent variable t that lies in the interval $[0, T]$ for some $T \in \mathbb{R}_+$. While this independent variable can represent any physical variable, we understand it to be *time* since we are primarily interested in initial value problems. Partition the time interval into N_t segments (t_n, t_{n+1}) for $n = 0, \ldots, N_t$, where $0 = t_0 < t_1 < \cdots < t_{N_t-1} < t_{N_t} = T$ are the discrete time instances used to partition the time domain and we define $\Delta t_n = t_n - t_{n-1}$ for $n = 1, \ldots, N_t$ for convenience (Figure 1). In many cases the time segments are the same size, i.e., $t_1, \ldots, t_{N_t}$ are equally spaced, in which cases all time steps are equal $\Delta t_n = \Delta t$.

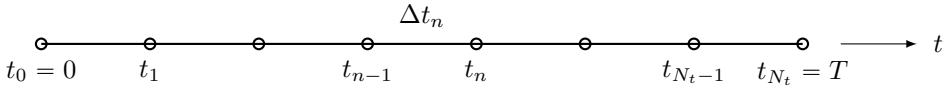

Figure 1: Schematic of discretization of time domain.

2.1 Classification of ODE solvers

Consider the general form of a system of ODEs in (4) defined on the time domain $[0, T]$, where $\boldsymbol{y}(t)$ is the unknown vector-valued function. Define the numerical solution at each time step as $\boldsymbol{y}_n$, which is an

approximation of the ODE solution at time t_n, i.e., $\boldsymbol{y}_n \approx \boldsymbol{y}(t_n)$. An *$M$-step method* is a method computes the solution at the current time step $\boldsymbol{y}_n$ by considering the solution at the M previous time steps, i.e., $\boldsymbol{y}_{n-M}, \dots, \boldsymbol{y}_{n-1}$. If $M = 1$, the method is called a single step method, otherwise it is a multistep method. An *K-stage method* computes the solution at the current time step $\boldsymbol{y}_n$ as a combination of K intermediate stages that depend on previous time steps. If $K = 1$, the method is called a single stage method, otherwise it is a multistage method. ODE solvers with any number of stages and steps can exist; however, the most popular ODE solvers are single stage, multistep methods (finite difference methods) and multistage, single step methods (Runge-Kutta methods).

In addition to classification based on the number of stages and steps used to advance the solution, ODE solvers can be classified as *explicit* or *implicit*. An ODE solver is *explicit* if each stage can be written explicitly in terms of other stages and previous time steps without requiring the solution of a linear or nonlinear system. These methods are usually very simple and efficient since an explicit function advances the solution; however, they suffer from stability issues that can place strict requirments on size of the time step Δt, which would require a large number of time steps to reach the final time T. An ODE solver is *implicit* if any stage is only defined implicitly in terms of other stages and previous time steps and requires the solution of a linear or nonlinear system to compute it. Implicit methods are more complicated than explicit method and more expensive per time step because they require the solution of a system of equations; however, they usually have better stability properties than explicit schemes, allowing for larger time steps.

2.2 Scalar, first-order ordinary differential equations

Consider an explicit form of a scalar, first-order ODE

$$F(t, y, \dot{y}) = \dot{y} + f(t, y) = 0,$$

whereby the dependence of ODE on highest derivative of the unknown function $y(t)$ is made explicit and we have used the dot notation to indicate a time derivative, i.e., $\dot{y} = \frac{dy}{dt}$. There are several main classes of methods for solving the above ODE include finite difference methods, Runge-Kutta methods, exponential integrators, and spectral deferred corrections. This document focuses on finite difference methods.

2.2.1 Finite difference methods

Finite difference methods enforce the ODE on the nodes of the time discretization introduced in Figure 1

$$F(t_n, y_n, \dot{y}_n) = \dot{y}_n + f(t_n, y_n) = 0, \quad n = 0, \dots, N_t, \tag{6}$$

where y_n and $\dot{y}_n$ are our numerical approximations of $y(t_n)$ and $\dot{y}(t_n)$, respectively, and define the time derivative approximation $\dot{y}_n$ using a finite difference stencil. Since we are only enforcing the ODE at a discrete set of nodes, instead of pointwise or in a weighted-residual sense, finite difference methods are *collocation* methods.

The simplest finite difference method approximates the time derivative using a first-order, forward difference stencil

$$\dot{y}_n = \frac{y_{n+1} - y_n}{\Delta t_{n+1}}, \quad n = 0, \dots, N_t - 1.$$

Substituting this expression into the general form of the finite difference method leads to the update formula

$$y_{n+1} = y_n - \Delta t_{n+1} f(t_n, y_n), \qquad n = 0, \dots, N_t - 1. \tag{7}$$

Since the solution at time t_{n+1} only depends on the solution at time t_n, this is a single step method; it is also a single stage method if we define $k_1 = -\Delta t_{n+1} f(t_n, y_n)$ as an intermediate stage. Furthermore, since the solution at time t_{n+1} is given as an explicit update to the solution at time t_n, this is an explicit method.

Another simple finite difference method approximates the time derivative using a first-order, backward difference stencil

$$\dot{y}_n = \frac{y_n - y_{n-1}}{\Delta t_n}, \quad n = 1, \dots, N_t.$$

Substituting this expression into the general form of the finite difference method leads to the equation

$$\frac{y_n - y_{n-1}}{\Delta t_n} + f(t_n, y_n) = 0, \quad n = 1, \ldots, N_t,$$

or equivalently

$$\frac{y_{n+1} - y_n}{\Delta t_{n+1}} + f(t_{n+1}, y_{n+1}) = 0, \quad n = 0, \ldots, N_t - 1.$$

Since the solution at time t_{n+1} only depends on the solution at time t_n, this is a single step method. Furthermore, since the solution at time t_{n+1} is only defined implicitly, it is an implicit method.

Example: Linear model problem

Recall the linear model problem from (2), which fits into the general form of our first-order ODE with $f(t, y) = \lambda y$. The forward Euler step is

$$y_{n+1} = y_n - \Delta t_{n+1} \lambda y_n = (1 - \Delta t_{n+1} \lambda) y_n \tag{8}$$

and the backward Euler step is

$$y_{n+1} = (1 + \Delta t_{n+1} \lambda)^{-1} y_n \tag{9}$$

for $n = 0, \ldots, N_t - 1$.

2.2.2 Accuracy

The *local trucation error* e_{n+1} of a numerical scheme is defined as the error between the exact and computed solution at time t_{n+1}, assuming no error at time t_n, i.e.,

$$e_{n+1} = |y(t_{n+1}) - y_{n+1}| \quad \text{given} \quad y_n = y(t_n). \tag{10}$$

The local truncation error quantifies the error introduced by the numerical scheme in a single time step. We are primarily interested in the behavior of the local truncation error with respect to the size of the time step rather than the exact value and are satisfied to only determine it up to a constant. The local truncation error is usually derived by expanding the exact solution in a Taylor series, using the assumption that $y_n = y(t_n)$, and introducing the numerical scheme into the Taylor expansion.

In contrast to the local truncation error, the *global error* E_{n+1} of a numerical scheme is the error between the exact and computed solution at a given time step

$$E_{n+1} = |y(t_{n+1}) - y_{n+1}| \tag{11}$$

and incorporates the accumulation of error over steps $1, \ldots, n$. It is generally difficult to compute the global error since the exact solution is not known; however, if we assume the error accumulation is additive, i.e., the global error at t_n is a sum of the local truncation error at t_n and the global error at t_{n-1}

$$E_n = E_{n-1} + e_n, \quad n = 1, \ldots, N_t$$

the global error can be determined from the truncation error. Furthermore, we trivially have the $E_0 = 0$ since $y_0 = y(t_0)$, i.e., the numerical solution is exact at the initial condition. Apply the additive accumulation of error formula recursively to obtain the error at the final time step solely in terms of the local truncation errors: $E_{N_t} = \sum_{n=1}^{N_t} e_n$. If the local trucation errors are $\mathcal{O}(\Delta t^{k+1})$, then $E_{N_t} = N_t \cdot \mathcal{O}(\Delta t^{k+1})$, where we have assumed a uniform time step of size Δt. The number of (uniform) time steps is related to the time step size as $N_t = T/\Delta t$ so the global error becomes $E_{N_t} = \mathcal{O}(\Delta t^k)$. Therefore, under the assumption of additive accumulation of error, a scheme with local trunction error $\mathcal{O}(\Delta t^{k+1})$ has global error $\mathcal{O}(\Delta t^k)$. Accordingly, a numerical scheme with local truncation error of $\mathcal{O}(\Delta t^{k+1})$ is called a scheme of order k.

Example: Accuracy of the Forward Euler Method
Consider the following expansion of solution of the ODE at time t_{n+1} using a Taylor series with remainder

$$y(t_{n+1}) = y(t_n + \Delta t) = y(t_n) + \Delta t \frac{dy}{dt}(t_n) + \frac{\Delta t^2}{2}\frac{d^2y}{dt^2}(\bar{t})$$

for some $\bar{t} \in (t_n, t_{n+1})$. Assuming the second derivative of y is bounded over the time domain, the last term can be replaced with $\mathcal{O}(\Delta t^2)$. From the definition of the ODE in (6), we have

$$y(t_{n+1}) = y(t_n) - \Delta t f(t_n, y(t_n)) + \mathcal{O}(\Delta t^2).$$

Using the condition that the solution is exact at step n, i.e., $y_n = y(t_n)$ leads to

$$y(t_{n+1}) = y_n - \Delta t f(t_n, y_n) + \mathcal{O}(\Delta t^2) = y_{n+1} + \mathcal{O}(\Delta t^2),$$

where the second equality used the forward Euler update formula in (7). Therefore, the local truncation error is $e_{n+1} = |y(t_{n+1}) - y_{n+1}| = \mathcal{O}(\Delta t^2)$ and the method is first-order globally. It can also be shown that the backward Euler method is first-order accurate; however, the proof is more involved.

2.2.3 Stability

To assess the stability of ODE solvers, we restrict our attention to the linear model problem in (2) and write the numerical discretization as an update formula of the form

$$y_n = C(\lambda \Delta t) y_{n-1},$$

where $C(\lambda \Delta t)$ is a discretization-dependent function; for simplicity, we assume a uniform time step Δt. Applying this relationship recursively, we obtain

$$y_n = C(\lambda \Delta t)^n y_0,$$

which is only buonded as $n \to \infty$ if the modulus of $C(\lambda \Delta t)$ is less than unity, i.e., $|C(\lambda \Delta t)| < 1$ or equivalently $\Re(C(\lambda \Delta t))^2 + \Im(C(\lambda \Delta t))^2 < 1$.

Example: Stability analysis of forward Euler method
From (8) we have the forward Euler method applied to the linear model problem leads to the update formula

$$y_{n+1} = (1 - \lambda \Delta t) y_n,$$

which implies $C(\lambda \Delta t) = 1 - \lambda \Delta t$. Therefore, the forward Euler method applied to the linear model problem is stable only if $|1 - \lambda \Delta t| < 1$ or equivalently $(\Re(\lambda \Delta t) - 1)^2 + \Im(\Delta t \lambda)^2 < 1$. In the special case where λ is a real number, the assumption that $\Re(\lambda) > 0$ (boundedness of exact solution) implies that $\lambda > 0$ and the stability condition reduces to $\Delta t < 2/\lambda$. Therefore the forward Euler method is only *conditionally stable*, i.e., it is only stable for certain choices of the time step size. Figure 2 (left) shows the stability region in the complex plane for the forward Euler method; the shaded region are the values of $\lambda \Delta t$ corresponding to a stable discretization.

Example: Stability analysis of backward Euler method
From (9) we have the backward Euler method applied to the linear model problem leads to the update formula

$$y_{n+1} = (1 + \lambda \Delta t)^{-1} y_n,$$

which implies $C(\lambda \Delta t) = (1 + \lambda \Delta t)^{-1}$. Therefore, the backward Euler method applied to the linear model problem is stable only if $|1 + \lambda \Delta t| > 1$ or equivalently $(\Re(\lambda \Delta t) + 1)^2 + \Im(\Delta t \lambda)^2 > 1$. In the special case where λ is a real number, the assumption that $\Re(\lambda) > 0$ (boundedness of exact solution) implies that $\lambda > 0$ and the stability condition reduces to $\Delta t > 0$. Since backward Euler is stable for all time steps, it is called *unconditionally stable*. Figure 2 (right) shows the stability region in the complex plane for the

backward Euler method; the shaded region are the values of $\lambda\Delta t$ corresponding to a stable discretization.

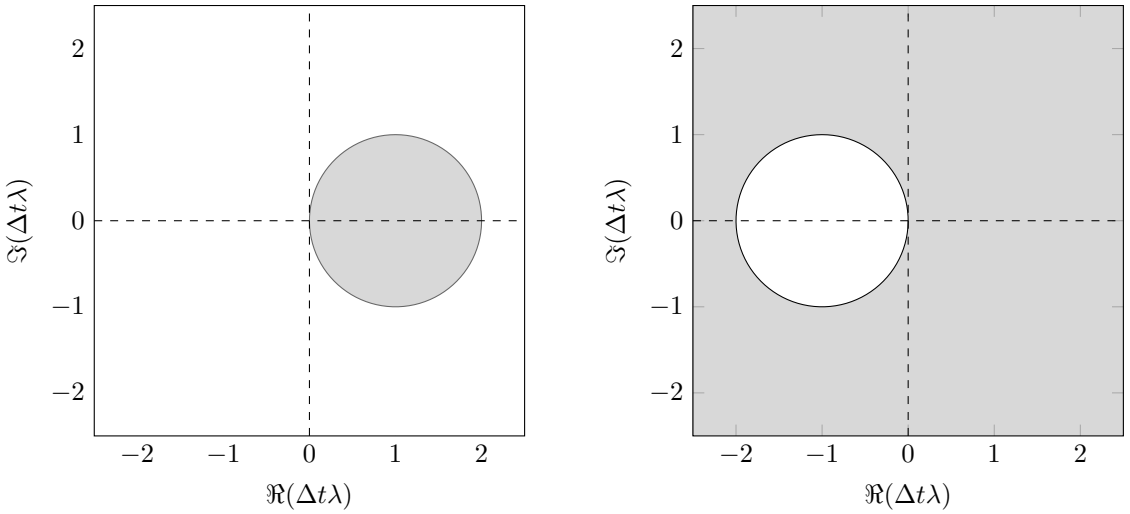

Figure 2: Stability region in the complex plane of $\lambda\Delta t$ corresponding to the forward Euler (*left*) and backward Euler (*right*) methods. The shaded region indicates values of $\lambda\Delta t$ corresponding to a stable discretization.

2.3 System of first-order ordinary differential equations

In this section we extend the development of numerical methods for scalar first-order differential equations to systems of first-order ODEs. Consider the explicit form of system of first-order ODEs

$$\boldsymbol{F}(t, \boldsymbol{y}, \dot{\boldsymbol{y}}) = \dot{\boldsymbol{y}} + \boldsymbol{f}(t, \boldsymbol{y}) = 0,$$

where the dependence on the highest derivative of the unknown functions $\boldsymbol{y}(t)$ is made explicit and we used the dot notation to abbreviate differentiation, i.e., $\dot{\boldsymbol{y}} = \frac{d\boldsymbol{y}}{dt}$. Any scalar ODE solvers can be extended to solvers for systems of ODEs; in this document, we focus on finite difference methods.

2.3.1 Finite difference methods

Identical to the scalar case, finite difference methods for systems of ODEs enforce the governing equation on the nodes of the time discretization introduced in Figure 1

$$\boldsymbol{F}(t_n, \boldsymbol{y}_n, \dot{\boldsymbol{y}}_n) = \dot{\boldsymbol{y}}_n + \boldsymbol{f}(t_n, \boldsymbol{y}_n) = 0, \quad n = 0, \ldots, N_t,$$

instead of pointwise throughout the time domain, i.e., a collocation approach. In this setting, $\boldsymbol{y}_n$ and $\dot{\boldsymbol{y}}_n$ are the numerical approximations of $\boldsymbol{y}(t_n)$ and $\dot{\boldsymbol{y}}(t_n)$, respectively, and we will define the velocity approximation $\dot{\boldsymbol{y}}_n$ using a finite difference stencil.

The forward Euler method is derived by approximating the time derivative using a first-order, forward difference stencil

$$\dot{\boldsymbol{y}}_n = \frac{\boldsymbol{y}_{n+1} - \boldsymbol{y}_n}{\Delta t_{n+1}}, \quad n = 0, \ldots, N_t - 1,$$

which leads to the expected forward Euler update formula

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n - \Delta t_{n+1} \boldsymbol{f}(t_n, \boldsymbol{y}_n), \quad n = 0, \ldots, N_t - 1.$$

Similarly, the backward Euler method is derived by approximating the time derivative using a first-order, backward difference stencil

$$\dot{\boldsymbol{y}}_n = \frac{\boldsymbol{y}_n - \boldsymbol{y}_{n-1}}{\Delta t_n}, \quad n = 1, \ldots, N_t,$$

which lead to the implicit backward Euler system

$$\frac{\boldsymbol{y}_{n+1} - \boldsymbol{y}_n}{\Delta t_{n+1}} + \boldsymbol{f}(t_{n+1}, \boldsymbol{y}_{n+1}) = 0, \quad n = 0, \dots, N_t - 1.$$

Example: Linear system of ODEs
Consider the linear system of N ODEs

$$\dot{\boldsymbol{y}} + \boldsymbol{A}\boldsymbol{y} = 0 \quad t \in (0, T), \qquad \boldsymbol{y}(0) = \boldsymbol{y}_0, \tag{12}$$

where $\boldsymbol{y}$ is the unknown vector-valued function with N entries, i.e., $\boldsymbol{y}(t) \in \mathbb{R}^N$ for any $t \in (0, T)$, $\boldsymbol{A} \in \mathbb{R}^{N \times N}$ is a fixed matrix defining the system of ODEs, $\boldsymbol{y}_0 \in \mathbb{R}^N$ is the initial condition, and we assume $\boldsymbol{A}$ is diagonalizable and all its eigenvalues have a positive real part (required for the solution to be bounded at $t \to \infty$). This simple system of ODEs will be used in subsequent sections to study the stability of numerical methods for solving systems of ODEs. The forward Euler method applied to this linear system of ODEs leads to the update formula

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n - \Delta t_{n+1} \boldsymbol{A}\boldsymbol{y}_n = (\boldsymbol{I} - \Delta t_{n+1}\boldsymbol{A})\boldsymbol{y}_n$$

for $n = 0, \dots, N_t - 1$ and where $\boldsymbol{I}_N$ is the $N \times N$ identity matrix. Similarly, the backward Euler method leads to the implicit system

$$\frac{\boldsymbol{y}_{n+1} - \boldsymbol{y}_n}{\Delta t_{n+1}} + \boldsymbol{A}\boldsymbol{y}_{n+1} = 0, \tag{13}$$

which can be re-written as the following update formula

$$\boldsymbol{y}_{n+1} = (\boldsymbol{I}_n + \Delta t_{n+1}\boldsymbol{A})^{-1}\boldsymbol{y}_n. \tag{14}$$

2.3.2 Accuracy

For systems of ODEs, the local truncation error is defined as

$$e_{n+1} = \|\boldsymbol{y}(t_{n+1}) - \boldsymbol{y}_{n+1}\| \quad \text{given} \quad \boldsymbol{y}_n = \boldsymbol{y}(t_n),$$

where $\|\cdot\|$ is any vector norm. Similar to the scalar case, a numerical scheme is kth order if the local truncation error is $\mathcal{O}(\Delta t^{k+1})$. Using the same method described in Section 2.2.2, it can be shown that both the forward and backward Euler methods are first-order schemes.

2.3.3 Stability

To assess the stability of solvers for systems of ODEs, we restrict our attention to the linear model problem in (12) and write the numerical discretization as an update formula of the form

$$\boldsymbol{y}_n = \boldsymbol{C}(\Delta t \boldsymbol{A})\boldsymbol{y}_{n-1},$$

where $\boldsymbol{C}(\Delta t \boldsymbol{A})$ is a discretization-dependent, $N \times N$ matrix-valued function; for simplicity, we assume a uniform time step Δt. Applying this relationship recursively, we obtain

$$\boldsymbol{y}_n = \boldsymbol{C}(\Delta t \boldsymbol{A})^n \boldsymbol{y}_0. \tag{15}$$

Denote the eigenvalue decomposition of $\boldsymbol{C}(\Delta t \boldsymbol{A})$ as $\boldsymbol{C} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^{-1}$, where the columns of $\boldsymbol{V} \in \mathbb{R}^{N \times N}$ contain the eigenvectors of $\boldsymbol{C}(\Delta t \boldsymbol{A})$ and $\boldsymbol{\Lambda} \in \mathbb{C}^{N \times N}$ is a diagonal matrix whose diagonal entries are the eigenvalues of $\boldsymbol{C}(\Delta t \boldsymbol{A})$. Since the $\boldsymbol{C}(\Delta t \boldsymbol{A})$ matrix is not necessarily symmetic, the eigenvalues $\{\lambda_i\}_{i=1}^N$ will be complex numbers, in general. Using the eigenvalue decomposition of $\boldsymbol{C}(\Delta t \boldsymbol{A})$ in (15) leads to

$$\boldsymbol{y}_n = \boldsymbol{V}\boldsymbol{\Lambda}^n\boldsymbol{V}^{-1}\boldsymbol{y}_0.$$

Multiply both sides of the equation by $\boldsymbol{V}^{-1}$ and define a new variable $\boldsymbol{z}_n = \boldsymbol{V}^{-1}\boldsymbol{y}_n$ to obtain the system

$$\boldsymbol{z}_n = \boldsymbol{\Lambda}^n \boldsymbol{z}_0.$$

This system is completely decoupled because the matrix of eigenvalues $\boldsymbol{\Lambda}$ is diagonal. Therefore, the stability condition reduces to $|\lambda_i| < 1$, i.e., all eigenvalues of the $\boldsymbol{C}(\Delta t \boldsymbol{A})$ matrix must have modulus less than one.

Example: Stability analysis of the Forward Euler method for system of ODEs
From (13), the update matrix for the forward Euler method applied to the linear model problem in (12) is

$$\boldsymbol{C}(\Delta t \boldsymbol{A}) = \boldsymbol{I} - \Delta t \boldsymbol{A}.$$

The eigenvalues of $\boldsymbol{C}(\Delta t \boldsymbol{A})$ are $\lambda_i = 1 - \Delta t \omega_i$ for $i = 1, \dots, N$, where $\omega_i \in \mathbb{C}$ is the ith eigenvalue of $\boldsymbol{A}$. The stability condition reduces to $|1 - \Delta t \omega_i| < 1$ or equivalently $(\Re(\Delta t \omega_i) - 1)^2 + \Im(\Delta t \omega_i)^2 < 1$. The stability diagram in the complex plane of each $\Delta t \omega_i$ term is identical to the forward Euler stability diagram for the scalar model problem in Figure 2.

Example: Stability analysis of the Backward Euler method for system of ODEs
From (14), the update matrix for the backward Euler method applied to the linear model problem in (12) is

$$\boldsymbol{C}(\Delta t \boldsymbol{A}) = (\boldsymbol{I} + \Delta t \boldsymbol{A})^{-1}.$$

The eigenvalues of $\boldsymbol{C}(\Delta t \boldsymbol{A})$ are $\lambda_i = (1 + \Delta t \omega_i)^{-1}$ for $i = 1, \dots, N$, where $\omega_i \in \mathbb{C}$ is the ith eigenvalue of $\boldsymbol{A}$. The stability condition reduces to $|1 + \Delta t \omega_i|^{-1} < 1$, or equivalently $(\Re(\Delta t \omega_i) + 1)^2 + \Im(\Delta t \omega_i)^2 > 1$. The stability diagram in the complex plane of each $\Delta t \omega_i$ term is identical to the backward Euler stability diagram for the scalar model problem in Figure 2.

2.4 Higher order ordinary differential equations

Scalar or systems of ordinary differential equations of order $m > 1$ can be solved by converting the ODEs to first-order form and applying the methods discussed in Section 2.2-2.3. There are also a class of ODE solvers that work with the ODE directly in its m-order form such as the Newmark schemes for second-order ODEs, which are very common in structural dynamics applications.