Initial value problems: introduction and one-step methods

Now that we have developed numerical methods for approximating functions and integrals, we will turn our attention to the more challenging task of numerically solving ordinary differential equations.

In turn, our study of numerical methods for ordinary differential equations will prepare us to solve partial differential equations, which we will tackle in the final portion of this class.

The first class of problems that we will study in this category are called *initial value problems* (IVPs). These equations are distinct from the other dominant category of ordinary differential equations, *boundary value problems* (BVPs). The key distinction between IVPs and BVPs is this:

> Whereas BVPs describe the *static (equilibrium)* response of a system to external forcing, IVPs govern the *dynamical time evolution* of a system to excitation.

1 *Review of some IVP fundamentals*

This philosophical distinction is borne out through a difference in the governing equations, which may be expressed as

$$\begin{aligned} \dot{\boldsymbol{u}} &= \boldsymbol{f}(\boldsymbol{u}, t) \\ \boldsymbol{u}(t_0) &= \boldsymbol{u}_0 \end{aligned} \tag{1}$$

Note that we are assuming the state variable $\boldsymbol{u}$ is a vector. The scalar case is a specific example with the dimension of the vector equal to one. Thus, the methods we develop here apply equally to vector or scalar systems.

Whereas a prime was sometimes used to denote a spatial derivative, the overdot here indicates a time derivative.

This mathematical description is intuitive based on the difference between BVPs and IVPs highlighted above: whereas BVPs require knowledge of the physical boundaries of the system, IVPs only require information of the state at some initial time instance and evolve according to the dynamics encoded in $\boldsymbol{f}$.

At first glance, the definition of IVPs provided in (1) may appear to omit Newton's second law, which represents perhaps the most important class of IVPs in engineering! When applied to systems of constant mass, Newton's second law is a second order system that reads

$$\begin{aligned} \boldsymbol{M}\ddot{\boldsymbol{u}} &= \boldsymbol{F}(t) \\ \boldsymbol{u}(t_0) &= \boldsymbol{u}_0 \\ \dot{\boldsymbol{u}}(t_0) &= \boldsymbol{v}_0 \end{aligned} \tag{2}$$

where $\boldsymbol{M}$ is a diagonal matrix containing the masses of the system and $\boldsymbol{F}(t)$ describes the forces that drive the acceleration of the masses. In fact, the definition (1) extends to these key systems as well. To see

this, note that we can recast (2) as

$$\begin{bmatrix} \dot{\boldsymbol{u}} \\ \ddot{\boldsymbol{u}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{I} \\ \boldsymbol{0} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{u} \\ \dot{\boldsymbol{u}} \end{bmatrix} + \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{M}^{-1}\boldsymbol{F}(t) \end{bmatrix} \qquad \begin{bmatrix} \boldsymbol{u}(t_0) \\ \dot{\boldsymbol{u}}(t_0) \end{bmatrix} = \begin{bmatrix} \boldsymbol{u}_0 \\ \boldsymbol{v}_0 \end{bmatrix} \tag{3}$$

(assuming $\boldsymbol{M}$ is invertible). Defining $\boldsymbol{z} := [\boldsymbol{u}, \dot{\boldsymbol{u}}]^T$ and $\boldsymbol{z}_0 := [\boldsymbol{u}_0, \boldsymbol{v}_0]^T$ allows us to rewrite (3) as

$$\begin{aligned} \dot{\boldsymbol{z}} &= \boldsymbol{f}(\boldsymbol{z}, t) \\ \boldsymbol{z}(t_0) &= \boldsymbol{z}_0 \end{aligned} \tag{4}$$

which therefore falls under the definition (1). In fact, nearly all higher-order initial value problems can be recast as a first-order IVP of the form (1), so assuming this definition is not very restrictive.

An important subset of IVPs occur when $\boldsymbol{f}(\boldsymbol{u}, t)$ is a linear time-invariant function acting on $\boldsymbol{u}$. That is, when (1) simplifies to

$$\begin{aligned} \dot{\boldsymbol{u}} &= \boldsymbol{A}\boldsymbol{u} \\ \boldsymbol{u}(t_0) &= \boldsymbol{u}_0 \end{aligned} \tag{5}$$

for some constant matrix $\boldsymbol{A}$.

The solution to this linear IVP is

$$\boldsymbol{u}(t) = e^{\boldsymbol{A}(t-t_0)}\boldsymbol{u}_0 \tag{6}$$

Having recalled these critical facts about the properties of IVPs and their analytical solutions, we can now sink our teeth into the problem of computing numerical solutions to the IVP system (1). The overwhelming majority of methods in the literature for achieving this end take the form of finite difference methods; *i.e.*, methods based on locally interpolating the IVP. In this spirit, this chapter will be devoted to finite-difference solutions to the IVP (1).

You may recall that the matrix exponential $e^{\boldsymbol{A}(t-t_0)}$ is defined analogously to its scalar counterpart in terms of an infinite sum: $e^{\boldsymbol{A}(t-t_0)} = \boldsymbol{I} + \boldsymbol{A}(t-t_0) + \frac{1}{2}\boldsymbol{A}^2(t-t_0)^2 + \frac{1}{6}\boldsymbol{A}^3(t-t_0)^3 + \cdots$

Why do local interpolation methods predominate over global interpolation (spectral collocation) and least-squares-error-minimizing (spectral) methods when numerically solving IVPs? The reason is because, whereas BVPs are based on information at all boundaries and therefore have solutions that depend on global information, IVPs only involve an initial condition and evolve based on local derivative information. These are perfect candidates for locally-focused approaches!

2 *Finite-difference methods for IVPs*

Finite-difference methods are implemented on IVPs by advancing the dynamical system (1) in discrete time increments denoted by Δt.

We assume Δt to be unchanging, though many important methods consider a Δt that changes in time based on the dynamics.

Problem statement: finite-difference methods for IVPs

Given: $\boldsymbol{u}_k \approx \boldsymbol{u}(t_k) = \boldsymbol{u}(t_0 + k\Delta t)$
Compute: $\boldsymbol{u}_{k+1} \approx \boldsymbol{u}(t_{k+1}) = \boldsymbol{u}(t_k + \Delta t)$.

Note that we distinguish between $\boldsymbol{u}(t_k)$, the true solution to the IVP (1) at time $t_k = t_0 + k\Delta t$, and $\boldsymbol{u}_k$, the approximate solution at t_k obtained through our finite difference method.

The key idea is that we have an approximation for the state $\boldsymbol{u}$ at time $t_k = t_0 + k\Delta t$—which was obtained by advancing the state k

increments from the known initial condition $\boldsymbol{u}(t_0)$—and we want to compute an approximation of the state at the next time increment.

How do we compute this approximate solution at the next time increment with a finite difference method? We will use local interpolation of the differential equation. It is instructive to consider an example of how this might be done. Observe that we can integrate the IVP (1) from t_k to t_{k+1}:

$$\int_{t_k}^{t_{k+1}} \dot{\boldsymbol{u}}\, dt = \int_{t_k}^{t_{k+1}} \boldsymbol{f}(\boldsymbol{u}, t)\, dt \tag{7}$$

The left-hand side can be computed exactly as $\boldsymbol{u}(t_{k+1}) - \boldsymbol{u}(t_k)$, and the right-hand side can be approximated be interpolating $\boldsymbol{f}(\boldsymbol{u}, t)$ locally near the interval $t \in [t_k, t_{k+1}]$. Devising this local interpolation of $\boldsymbol{f}$ provides one possible framework for obtaining our approximate numerical solution, $\boldsymbol{u}_{k+1}$.

There are two general classes of finite difference strategies for IVPs. The first category is the set of *one-step* methods, which use only information in the interval $t \in [t_k, t_{k+1}]$ to compute the approximate solution at t_{k+1}. The second category is the collection of *multi-step* methods, which utilize information over a broader interval $t \in [t_{k-j}, t_{k+1}]$ ($j \geq 1$) to advance to the solution to t_{k+1}. We will consider the analysis and implementation of both one-step and multi-step methods. We will focus on one-step methods today, and turn to multi-step methods in the next lecture.

Take a moment to appreciate the beauty of this: because of the framework we develop for interpolating functions, we will be able to solve differential equations!

Equation (7) is not the only means by which a finite difference method may be obtained. Indeed, an alternative approach would be to interpolate $\dot{\boldsymbol{u}} = \boldsymbol{f}(\boldsymbol{u}, t)$ directly by expressing $\boldsymbol{u}$ as a linear combination of basis functions in terms of unknown coefficients and requiring that its time derivative match $\boldsymbol{f}$ at specific time instances. Thus, (7) is meant as an instructive example, rather than a comprehensive representation of all finite difference approaches for IVPs.

3 *One-step methods*

Perhaps the simplest finite-difference method we could derive comes from representing $\boldsymbol{f}$ in (7) as a constant over the interval $t \in [t_k, t_{k+1}]$. Some natural choices for the value of this constant are $\boldsymbol{f}(\boldsymbol{u}(t_k), t_k)$ and $\boldsymbol{f}(\boldsymbol{u}(t_{k+1}), t_{k+1})$, respectively. These choices lead to the forward and backward Euler methods, respectively.

Forward Euler method:

$$\boldsymbol{u}_{k+1} = \boldsymbol{u}_k + \Delta t \boldsymbol{f}(\boldsymbol{u}_k, t_k) \tag{8}$$

Backward Euler method:

$$\boldsymbol{u}_{k+1} = \boldsymbol{u}_k + \Delta t \boldsymbol{f}(\boldsymbol{u}_{k+1}, t_{k+1}) \tag{9}$$

Notice that the difference between these Euler methods is more than superficial. To advance the approximate solution with the forward Euler method, one need only evaluate the right-hand side of (8). This is not possible with the backward Euler method, as the

How did we arrive at these specific expressions for the forward and backward Euler methods? Let's consider the forward Euler method first (the process is nearly identical for the backward Euler method). First, we used the fact that the left-hand side of (7) can be exactly written as $\boldsymbol{u}(t_{k+1}) - \boldsymbol{u}(t_k)$ and recognized that since we do not have the exact solution $\boldsymbol{u}(t)$, this must be approximated $\boldsymbol{u}_{k+1} - \boldsymbol{u}_k$. Then, we evaluated the right-hand side by representing $\boldsymbol{f}$ as a constant chosen as $\boldsymbol{f}(\boldsymbol{u}(t_k), t_k)$. The integral thus evaluates to $\Delta t \boldsymbol{f}(\boldsymbol{u}(t_k), t_k)$. Again, we do not have access to the true solution $\boldsymbol{u}(t)$, so we write this as $\Delta t \boldsymbol{f}(\boldsymbol{u}_k, t_k)$. Rearranging so that $\boldsymbol{u}_{k+1}$ is the only term on the left-hand side results in equation (8).

unknown $\boldsymbol{u}_{k+1}$ is in the right-hand side of (9); that is, advancing the approximate solution with the backward Euler method requires the solution of a nonlinear algebraic system of equations! Why would we subject ourselves to that additional complexity? We will find that the backward Euler method has better *stability* properties—for time steps that are too large, the forward Euler method is more prone to provide exceedingly large estimates to $\boldsymbol{u}_{k+1}$.

Methods such as backward Euler that involve the unknown $\boldsymbol{u}_{k+1}$ in the right-hand side are called *implicit*. Otherwise, the method is called *explicit*.

We could alternatively represent $\boldsymbol{f}$ as a line over $t \in [t_k, t_{k+1}]$. The result of this is the trapezoid method, which can be expressed as

The name 'trapezoid method' is not a coincidence—this is exactly the same approach that leads to the trapezoid quadrature rule!

TRAPEZOID METHOD:

$$\boldsymbol{u}_{k+1} = \boldsymbol{u}_k + \frac{1}{2}\Delta t\Big(\boldsymbol{f}(\boldsymbol{u}_k, t_k) + \boldsymbol{f}(\boldsymbol{u}_{k+1}, t_{k+1})\Big) \tag{10}$$

Note that not only does the trapezoid method involve multiple evaluations of $\boldsymbol{f}$, but it is also implicit, as it involves the unknown $\boldsymbol{u}_{k+1}$ in the right-hand side. Why would we wish to add complexity to our lives and use the trapezoid method as compared with one of the Euler methods? We will demonstrate in a subsequent lecture that the trapezoid method is more *accurate* than the Euler methods: for a given Δt, the trapezoid rule will in general provide a better approximate solution to the IVP (1) than either of the Euler methods. As with the concept of stability, we will make this notion of accuracy more precise later.

Methods that involve multiple evaluations of $\boldsymbol{f}$ in the interval $t \in [t_k, t_{k+1}]$ are called *multi-stage* methods. To see why, notice that we can express the trapezoid method, which involves two evaluations of $\boldsymbol{f}$, as a two-stage process.

TRAPEZOID METHOD, TAKE 2:

$$\begin{aligned} \boldsymbol{u}_k^* &= \boldsymbol{u}_k + \frac{1}{2}\Delta t \boldsymbol{f}(\boldsymbol{u}_k, t_k) \\ \boldsymbol{u}_{k+1} &= \boldsymbol{u}_k^* + \frac{1}{2}\Delta t \boldsymbol{f}(\boldsymbol{u}_{k+1}, t_{k+1}) \end{aligned} \tag{11}$$

Thus, the trapezoid method is a multi-stage method.

In this text, we will restrict our attention of multi-stage methods to a special class of methods called *Runge-Kutta* methods. There is a preponderance of Runge-Kutta methods in existence that involve various numbers of stages and a wide range of stability and accuracy properties. The trapezoid method is a multi-stage method that may be classified as a Runge-Kutta method. One simple way to arrive at a different multi-stage Runge-Kutta method is to tweak the trapezoid method so that it is no longer implicit. We can do this by recognizing that the $\boldsymbol{f}(\boldsymbol{u}_{k+1}, t_{k+1})$ term (the implicit term) can be approximated

Technically, we can view either of the Euler methods as a one-stage Runge-Kutta method.

with a Taylor series as $f(u_k + \Delta t f(u_k, t_k), t_{k+1})$. Using this approximation leads to an explicit multi-stage method called Heun's method.

HEUN'S METHOD (TWO-STAGE RUNGE-KUTTA METHOD):

$$u_{k+1} = u_k + \frac{1}{2}\Delta t\left[f(u_k, t_k) + f\Big(u_k + \Delta t f(u_k, t_k), t_{k+1}\Big)\right] \tag{12}$$

Other Runge-Kutta methods can be arrived at by interpolating f in expression (7) using higher-order polynomials that use information not only at t_k and t_{k+1}, but at intermediate time instances as well. One example of this is the following commonly used four-stage Runge-Kutta method.

FOUR-STAGE RUNGE-KUTTA METHOD (RK4):

$$u_{k+1} = u_k + \frac{1}{6}\Delta t\Big(y_1 + 2y_2 + 2y_3 + y_4\Big) \tag{13}$$

where

$$\begin{aligned} y_1 &= f(u_k, t_k) \\ y_2 &= f\left(u_k + \frac{1}{2}\Delta t y_1, t_k + \frac{1}{2}\Delta t\right) \\ y_3 &= f\left(u_k + \frac{1}{2}\Delta t y_2, t_k + \frac{1}{2}\Delta t\right) \\ y_4 &= f\left(u_k + \Delta t y_3, t_k + \Delta t\right) \end{aligned} \tag{14}$$

The details of how this method is derived involve some subtlety, but we can remove much of the mystery by observing that if f were not a function of u, then we would arrive at the expression (13) by representing $f(t)$ as a quadratic function in equally spaced increments over the interval $t \in [t_k, t_{k+1}]$.

We have discussed some one-step methods for advancing the IVP system (1). In the next lecture, we will introduce a different class of methods for solving IVPs: multi-step methods.

Notice one crucial oversight in our discussion so far: we still have no mechanism for deciding which method to use for which problem. This is because we do not yet understand the accuracy properties of numerical methods for IVPs. We will address this crucial issue next week, after we have finished introducing one-step and multi-step methods.