Chapter 1
Introduction to Numerical Methods for ODEs

In this chapter we will introduce the numerical solution to an ordinary differential equation (ODE). While some differential equations, like many of those you saw in 18.03, have analytical solutions, there are many interesting ODEs that do not have analytical solutions. For those problems without analytical solutions we will use numerical methods to approximate the solution.

1 Self-Assessment

Before reading this chapter, you may wish to review...

- solutions of first-order ODEs [18.03 Lecture 3: Video]
- Taylor series expansions [18.01 Lecture 38: Video]
- Matlab plotting basics [Matlab plotting documentation]

After reading this chapter you should be able to...

- describe the general form for a first-order scalar ordinary differential equation
- represent a continuous solution as a set of discrete points
- adequately communicate using the shorthand notation
- write Taylor series expansions for u^{n+1}, u^{n-1}, and u^{n+2} about u^n
- describe the forward Euler method to a friend

2 First-order ODEs

Many interesting physical phenomena can be modeled by a first-order ODE of the form

$$u_t = f(u(t)), \qquad u(0) = u_0, \;\; 0 < t < T, \tag{1}$$

where $u(t)$ is the time-dependent state of the system, $u_t = du/dt$ is the time-derivative of the state, $f(u(t))$ is the forcing function depending on the state, u_0 is the initial condition, and $0 < t < T$ indicates that we want to solve the problem forward in time until time $t = T$. We will begin by considering scalar problems where $u(t) \in \mathbb{R}$ (a real number), but in Chapter 6 we will extend to the vector-valued case where $\mathbf{u}(t) \in \mathbb{R}^d$ is a d-dimensional state vector.

Let's consider an example.

Example 1. Consider transient heat transfer associated to convective heating or cooling (Unified Propulsion 18.3). The first law states that the heat into the object is given by the product $\rho V c T_t$ where ρ is the density, V is the volume, and c is the specific heat. Let h be the heat transfer coefficient and A be the surface area of the body, then the time evolution of temperature is given by

$$Ah(T - T_\infty) = -\rho V c T_t.$$

Let $u(t)$ be the non-dimensional temperature difference $(T - T_\infty)/(T_i - T_\infty)$ where $T(t)$ is the time-dependent temperature in the body (assumed constant throughout, i.e., small Biot number), T_∞ is the surrounding ambient temperature, and T_i is the initial temperature of the body at time $t = 0$. Then the governing equation is

$$u_t = -\lambda u$$

where $\lambda = \dfrac{hA}{\rho V c}$. The initial condition is $u(0) = 1$.

Exercise 1. Write down the analytical solution to the initial value problem of Example 1 and plot it in Matlab for the values $h = 2$, $A = 4$, $\rho = 10$, $V = 9$, and $c = 1$. What is the value of $u(2)$?

3 Discretization

When you plot the solution in Matlab, you are likely creating two vectors: one corresponding to points in time $\mathbf{t} = [t_0, t_1, \ldots, t_N]$ and another corresponding to the solution $\mathbf{u}(t) = [u(t_0), u(t_1), \ldots, u(t_N)]$ at those points in time. This process of representing a continuous function by a finite set of numbers is referred to as *discretization*. The main idea is illustrated in Figure 1. Instead of representing the function continuously, we represent it as a finite set of ordered pairs $(t_n, u(t_n))$.

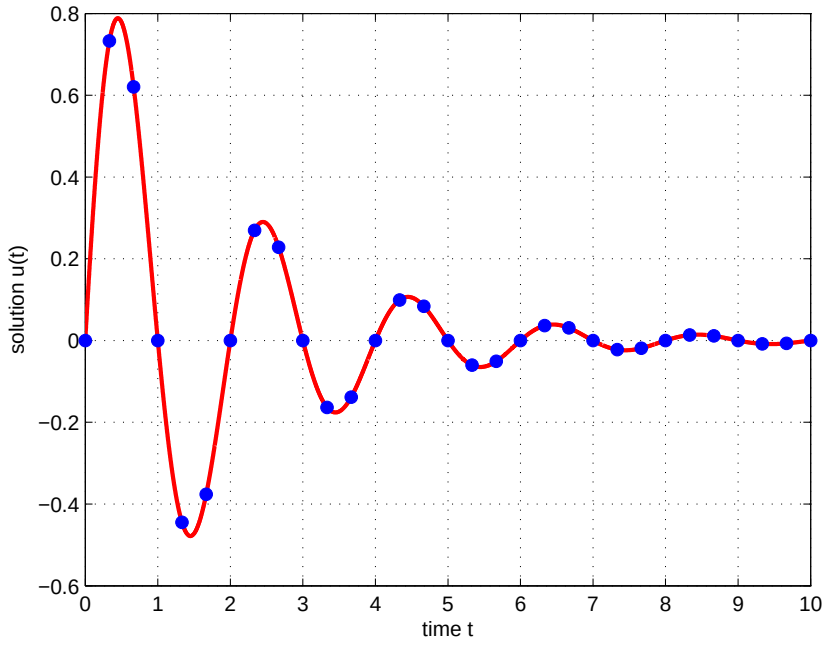

Fig. 1 Numerical solutions are represented as a finite set of ordered pairs (blue dots) representing the discretization of a continuous function.

When we solve mathematical problems on a computer, it will always be necessary to discretize them. For the initial value problems of the form (1), we will begin with the initial condition u_0 at time $t = 0$ and solve forward in time. First we select a time step $\Delta t > 0$ representing the length of the interval between any two adjacent time points t_n and t_{n+1}. Although it is not necessary to choose a constant Δt for the entire simulation, this is the approach we will take for this course. (You should be aware that state of the art numerical simulation codes adaptively select the time step, e.g., based on an estimate of the error.) Our numerical solution will then involve computing an approximation to the solution $u(t_n)$ using information up to (and sometimes including, see implicit methods in Chapter 9) time step n.

4 Notation

Let's establish some convenient shorthand notation. In what follows we will place time indices as superscripts. For example, the discrete time point at $t = 0$ will be indicated by t^0. Analogously, one time step later, we will have $t^1 = t^0 + \Delta t$, then $t^2 = t^1 + \Delta t = t^0 + 2\Delta t$, and so on. Therefore, if $t^0 = 0$, at time step n, we have $t = t^n$. Similarly, we will refer to the solution at time step n as $u^n = u(t^n)$. It is understood that these superscripts indicate time step and are **not** exponents. Since our numerical solution will rarely be exact, we will introduce v^n as the numerical approximation to the solution u^n at time t^n. Thus, our numerical solution will be a set of ordered pairs (t^n, v^n) that (we hope) closely approximate the analytical solution $u(t)$.

It will also be necessary for us to refer to time derivatives of the solution. For example, we indicate the first time derivative of the solution at time step n as $u_t^n = u_t(t^n)$. For higher derivatives we add more subscripts; for example,

$$u_{tt}^n = \left.\frac{d^2u}{dt^2}\right|_{t^n}, \tag{2}$$

that is, the second time derivative evaluated at t^n. The notation is summarized in Table 4.

Table 1 Shorthand notation.

Symbol	Meaning
n	time step index
Δt	time step interval
t^n	time at step n
u^n	analytical solution at $t = t^n$
v^n	numerical approximation at $t = t^n$
u_t^n	first time derivative of u at $t = t^n$
u_{tt}^n	second time derivative of u at $t = t^n$

Thought Experiment Suppose we wish to solve the general initial value problem

$$u_t = f(u,t), \qquad u^0 = 0. \tag{3}$$

Assume that $\Delta t > 0$ is given. Devise a method to approximate u^1; what information can get you from u^0 to u^1 and how might you use it?

5 Forward Euler

We will conclude this chapter by introducing our first numerical method for the solution of ODEs. The *forward Euler* method approximates the solution at time step $n+1$ using the approximation at the previous time step v^n and the forcing function $f(v^n, t^n)$ at time step n. Suppose we have solved numerically the ODE up through time step n. That is, we have chosen Δt, we were given $v^0 = u^0$, and we subsequently obtained approximations $v^1, v^2, \ldots, v^n$. It is now our task to obtain an approximation v^{n+1} for the solution u^{n+1} at time step $n+1$.

Consider the Taylor series for u^{n+1}:

$$u^{n+1} = u^n + \Delta t u_t^n + \frac{1}{2}\Delta t^2 u_{tt}^n + \mathscr{O}(\Delta t^3). \tag{4}$$

This "Big-O" notation is shorthand for the remaining terms of the Taylor series expansion. It indicates that in the limit $\Delta t \to 0$, the dominant term is of the form $c\Delta t^3$ for some constant c.

Let's take the first two terms in the series and ignore the remaining terms. Then we have the approximation

$$u^{n+1} \approx u^n + \Delta t u_t^n. \tag{5}$$

Recall that $u_t^n = u_t(t^n)$ and since the governing equation (1) is $u_t = f(u,t)$, this means that $u_t^n = f(u^n, t^n)$. Substituting into (5) we find

$$u^{n+1} \approx u^n + \Delta t f(u^n, t^n). \tag{6}$$

In practice, we do not know u^n (except for when $n = 0$) but instead we have an approximation v^n of it by the time we reach time step n. We have arrived at the forward Euler method:

$$v^{n+1} = v^n + \Delta t f(v^n, t^n), \qquad n \geq 0 \tag{7}$$

with initial condition $v^0 = u^0$. We begin with the given initial condition and iteratively solve for the subsequent numerical solution at each time step, first obtaining v^1 using v^0 and $f(v^0, 0)$, then calculating v^2 based on v^1 and $f(v^1, t^1)$, and so on.

Video demonstration of the forward Euler method

Example 2. Consider the initial value problem

$$u_t = -2u, \qquad u^0 = 1. \tag{8}$$

Although this problem has an analytical solution, we demonstrate here the numerical solution using the forward Euler method. Grab the MATLAB script

```
N = 10;                          % 10 forward Euler steps
dt = 0.1;                        % time step size
u(1) = 1;                        % initial condition
for n=1:N                        % loop over forward Euler steps
    dudt = -2 * u(n)             % compute time derivative
    u(n+1) = u(n) + dt * dudt    % forward Euler step
end
```

and run it to see for yourself how the forward Euler method works.

Exercise 2. Consider the equation and initial condition in Example 2. Using $\Delta t = 0.01$, calculate by hand v^2 using the forward Euler method. Which solution below most accurately reflects your answer?

(a) $v^2 = 0.9604$
(b) $v^2 = 0.0099$
(c) $v^2 = 0.9800$
(d) $v^2 = 1.0098$

In the next chapter we will discuss *convergence* of numerical schemes. The following exercise serves as a preview for that discussion.

Exercise 3. In this exercise you will modify the code provided in Example 2 to observe the behavior of the error in our numerical solution as you change the time step Δt. Implement the forward Euler method for the problem in Example 2 (or modify the provided code), and compute the error between the numerical solution and the exact solution at time $t = 1$ for time steps $\Delta t = 0.005$, $\Delta t = 0.01$, and $\Delta t = 0.02$. Do you notice any pattern in the results?

(a) errors approximately double when the timestep is doubled
(b) errors decrease as timestep decreases
(c) (a) and (b)
(d) none of the above