1. Mathematical Preliminaries

The concept of **ordinary differential equations** (ODEs) comes from a more general concept — **differential equations**. This field studies the problem of how variables change and has a long and successful history [Edwards, 1994; Dunham, 2005]. Today, differential equations have become the essential language of modern science and engineering, and their applications are wide-ranging. In this section, we introduce the basic concepts of differential equations. In particular, we focus on ODEs and demonstrate a few applications of these mathematical tools in deep learning.

1.1 Notation of ODEs

We begin by introducing a simple problem of a moving car, which we will use as an example throughout this section to motivate several key concepts. Suppose we observe a car moving along a straight path. Let $t \in \mathbb{R}$ represent time, which serves as our independent variable. We denote the position of the car at any given time t as $x(t)$. Here $x(t)$ is the dependent variable, whose value depends on the independent variable t. In some cases, we may omit the argument to simplify the notation (i.e., use x instead of $x(t)$).

We know that the velocity of the car is the rate of change of its position with respect to time. In calculus, this quantity is denoted as the derivative $\frac{dx(t)}{dt}$. Here $dx(t)$ and dt are called **differentials**, which represent infinitesimal changes in the dependent and independent variables, respectively. The derivative $\frac{dx(t)}{dt}$ describes the instantaneous rate of change of the position $x(t)$ with respect to time t. So this concept is defined at a point in time, not over a finite interval. Mathematically, $\frac{dx(t)}{dt}$ can be written as the limit of the difference quotient

$$\frac{dx(t)}{dt} = \lim_{\Delta t \to 0} \frac{x(t + \Delta t) - x(t)}{\Delta t}. \tag{1}$$

One can find more details on the formal definition of derivatives in calculus textbooks [Apostol, 1991; Spivak, 2006].

If the velocity of the car is governed by a rule depending on its current position and the time (for example, due to varying road friction or wind resistance), we can express this relationship using a function $f(\cdot)$

$$\frac{dx(t)}{dt} = f(x(t), t). \tag{2}$$

This equation is formally known as an ODE. The term *ordinary* implies that the dependent variable $x(t)$ is a function of a single independent variable t. This stands in contrast to **partial differential equations** (PDEs), where the unknown function depends on multiple independent variables[1].

1. An example of partial differential equations is the **heat equation**, which describes how the temperature u changes over time t and spatial location s. It is typically written as

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial s^2}, \tag{3}$$

where α is a positive coefficient. In such cases, the equation involves partial derivatives, denoted by the symbol ∂. $\frac{\partial^2 u}{\partial s^2}$ is the second spatial derivative of u in the s direction.

Alternatively, Eq. (2) can also be written in differential form

$$dx(t) = f(x(t), t)dt. \tag{4}$$

This notation suggests that the infinitesimal change in position is the product of the velocity and the time increment. We will show in subsequent sections that such a form is particularly useful for developing numerical integration methods and modeling stochastic processes.

Clearly, the complexity of the ODE given by Eqs. (2) and (4) is determined by that of the function $f(\cdot)$. The simplest case is that $f(\cdot)$ is a constant. For example, if the car travels at a constant velocity v, then the function takes the form $f(x(t), t) = v$. In this scenario, Eq. (2) reduces to

$$\frac{dx(t)}{dt} = v. \tag{5}$$

It is easy to figure out that this ODE corresponds to the familiar linear equation of motion

$$x(t) \quad = \quad vt + x(0), \tag{6}$$

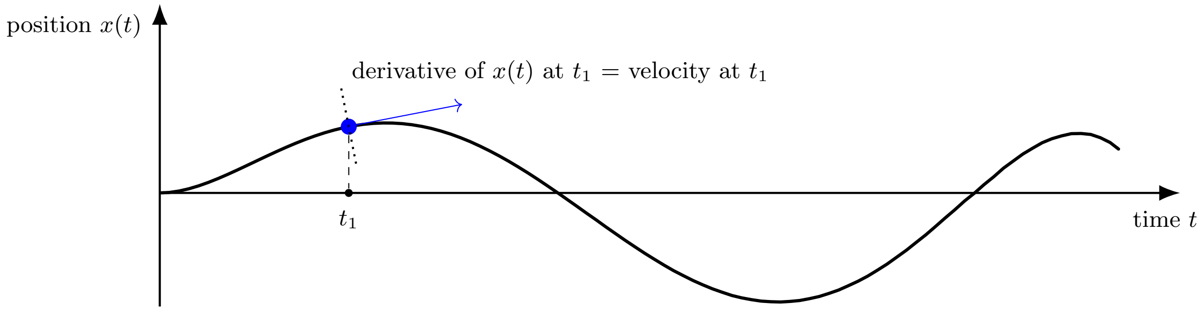
where $x(0)$ represents the position of the car at time $t = 0$. This simple example demonstrates a fundamental property of differential equations: the general solution typically involves an arbitrary constant (here, $x(0)$). Geometrically, this means the ODE describes a family of parallel curves in the (x, t) plane, each corresponding to a different starting point.

The situation becomes more interesting when $f(\cdot)$ depends explicitly on the position $x(t)$. For instance, suppose the velocity decreases as the car moves away from the origin due to some control rule. If the function $f(\cdot)$ depends linearly on $x(t)$ (e.g., $f(x(t), t) = -kx(t)$ for some constant k), the equation is classified as a **linear ODE**. Such equations are of particular interest because they often admit closed-form analytical solutions [Hartman, 2002].

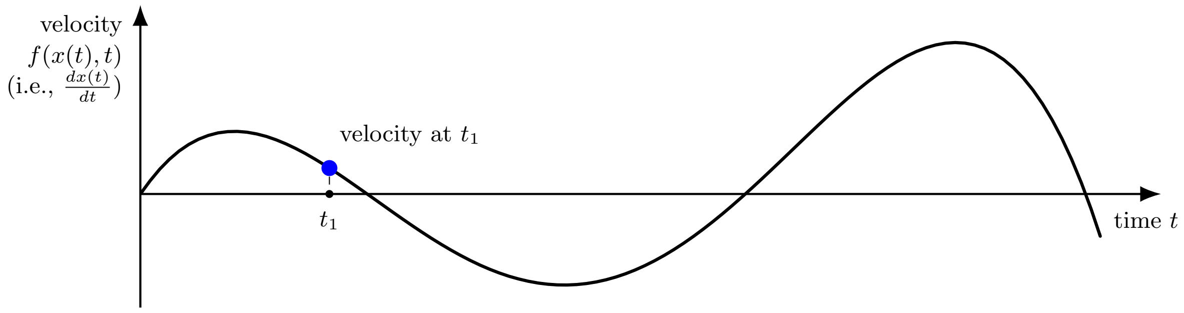
However, in many real-world applications, $f(\cdot)$ is nonlinear. If our car were moving in an environment where the governing relationship involved terms like $x(t)^2$ or $\sin(x(t))$, we would be dealing with a **nonlinear ODE**. Unlike their linear counterparts, nonlinear equations rarely possess simple analytical solutions. Consequently, exact analytical derivation is often impossible, and we must resort to numerical approximation techniques. As most problems discussed in the subsequent chapters involve nonlinear ODEs, exploring approximate solutions will be the focus of the following discussion.

Note that, although $f(\cdot)$ takes two arguments, $x(t)$ and t, the term $x(t)$ itself is a function of t. Thus, along any specific trajectory, the velocity can be seen as a composite function of t. In this case, we can plot $f(\cdot)$ as a function of t, where at any point the value corresponds to the derivative $\frac{dx(t)}{dt}$. See Figure 1 for a simple example.

In practice, however, the specific dependence of $f(\cdot)$ on t is often unknown because the trajectory $x(t)$ itself is unknown. Furthermore, additional complexity arises from the explicit time-dependence of $f(\cdot)$. Unlike autonomous functions that possess a static structure, here the functional form itself evolves over time (to emphasize this, we can adopt the notation $f_t(\cdot)$ to represent $f(\cdot, t)$). This means that even if the car returns to the exact same position at a later instant, it



(a) Position $x(t)$ against time t



(b) Velocity $f(x(t), t)$ (i.e., $\frac{dx(t)}{dt}$) against time t

Figure 1: Plots of the dependent variable ($x(t)$) and its derivative ($\frac{dx(t)}{dt}$) of an ODE. While the ODE requires $x(t)$ as an input to $f(\cdot)$, $x(t)$ itself is driven by time. Therefore, if the trajectory is fixed, the velocity $f(\cdot)$ implicitly becomes a function of t alone. So we can plot the velocity $f(\cdot)$ directly against t.

may have a different velocity. Since the "rules of the road" are not constant, the behavior of the car cannot be predicted solely by its current state $x(t)$, and one must also account for the specific moment in time.

More fundamentally, such an ODE defines a **dynamical system** [Hirsch et al., 2013]. In this framework, the variable $x(t)$ represents the **state** of the system, which describes its configuration at any specific moment. The function $f(\cdot)$ characterizes the **dynamics** of the system (often referred to as the **dynamic rule**), which indicates how the state changes at any given instant. From this perspective, the distinction regarding time-dependence becomes a classification of the rules themselves: if the dynamic rule is static and depends solely on the state (i.e., $\frac{dx(t)}{dt} = f(x(t))$), the system is **autonomous**. If the rule itself changes over time (i.e., $\frac{dx(t)}{dt} = f(x(t), t)$), the system is **non-autonomous**. Non-autonomous systems are powerful tools for describing complex processes, though they are inherently more complicated than autonomous systems. For example, in Section 2, we will show that a stack of Transformer layers can be modeled as a non-autonomous system.

Notation	Description
t	The independent variable, typically representing time.
$x(t)$	The dependent variable, representing the state of the system at time t.
$dx(t)/dt$	The derivative of the state, representing the instantaneous rate of change (e.g., velocity).
$f(x(t), t)$	The function (or dynamic rule) governing the system's evolution. It describes the dynamics of the system.
$\frac{dx(t)}{dt} = f(x(t), t)$	A non-autonomous ODE, where the dynamics explicitly depend on time.
$\frac{dx(t)}{dt} = f(x(t))$	An autonomous ODE, where the dynamics depend solely on the state (time-invariant rules).
$dx(t) = f(x(t), t)dt$	The differential form of the ODE, often used in numerical integration and stochastic calculus.
$\frac{d\mathbf{x}(t)}{dt} = f(\mathbf{x}(t), t)$	A vector-valued ODE, describing a system with a multi-dimensional state vector $\mathbf{x}(t) \in \mathbb{R}^d$ (where $f(\cdot)$ defines a vector field).

Table 1: Basic concepts in ODEs.

So far, our discussion has focused on scalar variables and functions, that is, the state $x(t)$ is a scalar variable and the function $f(\cdot)$ is a scalar function. One natural extension is to define ODEs on vectors. Let $\mathbf{x}(t) \in \mathbb{R}^d$ denote a vector state containing d distinct components (e.g., the neuron activations in a neural network layer). Correspondingly, the function $f(\cdot)$ is defined as a vector-valued mapping: $\mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$. Then, the ODE becomes

$$\frac{d\mathbf{x}(t)}{dt} = f(\mathbf{x}(t), t). \tag{7}$$

Here, the derivative $\frac{d\mathbf{x}(t)}{dt}$ is applied component-wise to the vector $\mathbf{x}(t)$. This shift to higher dimensions introduces a richer geometric interpretation. While the scalar function in our previous examples simply represents the slope of a curve, the vector function $f(\cdot)$ defines a **vector field** over the state space. At every point $\mathbf{x}(t)$, $f(\mathbf{x}(t), t)$ assigns a vector that specifies both the magnitude and direction of the system's instantaneous velocity.

Although we extend the state to a d-dimensional space, the fundamental concepts and properties of ODEs remain valid. In this section, we will continue to use scalar notation for simplicity. In subsequent sections, however, we will primarily employ vector-valued ODEs to model more complex problems. We summarize the key concepts introduced thus far in Table 1.

1.2 Solving ODEs

We have shown that ODEs describe the derivative of the state $x(t)$. One might ask: why do we define the system using derivatives instead of describing the state $x(t)$ directly? Indeed, if the full trajectory of $x(t)$ were known, the ODE would be redundant. However, in many applications, we do not have analytical expressions for $x(t)$. Instead, we are only provided with the dynamics $f(\cdot)$ and the initial condition $x(0)$. This is analogous to driving a car: while it is easy to know our current speed at any moment, it is much harder to determine the current position relative to a starting point. In the language of ODEs, we are given the "speedometer" $f(\cdot)$ and the starting point $x(0)$, and our goal is to reconstruct the "map" of the journey $x(t)$.

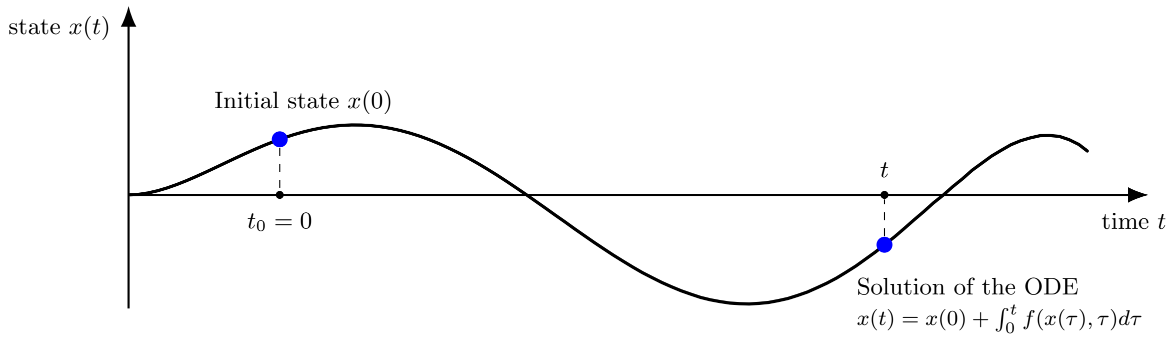
In this case, we need to solve ODEs. This is often referred to as the **initial value problem** (IVP) where an ODE and a specific constraint (e.g., the initial state $x(0)$) are given, and we recover the state $x(t)$ at any time t from its derivative. By the fundamental theorem of calculus, the state at time t is the sum of the initial state and the accumulation of all instantaneous changes over the interval $[0, t]$. Thus we can write the solution to the ODE by integration

$$\begin{aligned} x(t) &= x(0) + \int_0^t \frac{dx(\tau)}{d\tau} d\tau \\ &= x(0) + \int_0^t f(x(\tau), \tau) d\tau. \end{aligned} \tag{8}$$

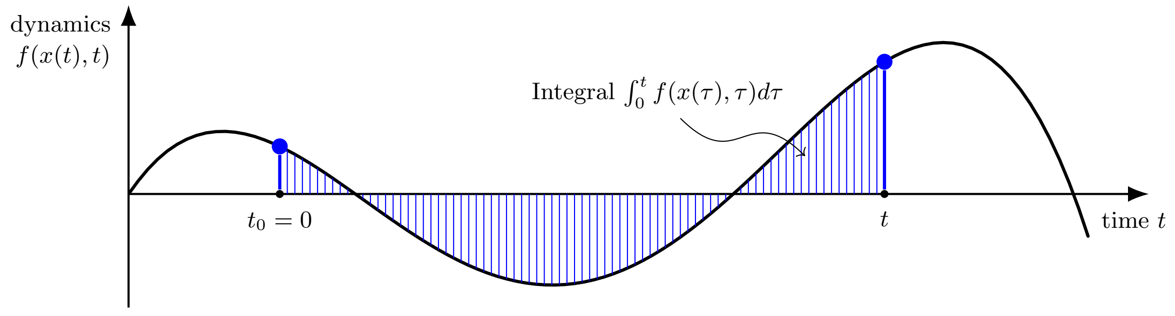
As shown in Figure 2, this formula has a very intuitive geometric interpretation. The integral term $\int_0^t f(x(\tau), \tau) d\tau$ represents the signed area under the curve of the function $f(x(\tau), \tau)$, from $\tau = 0$ to $\tau = t$. This area quantifies the total accumulated change in the state $x(t)$ over the interval $[0, t]$. Therefore, the equation simply states that the state at time t is found by adding this total change (the area) to the initial state $x(0)$.

However, this integral formulation presents a computational challenge of ODE solving: while the equation appears simple, it cannot be directly computed because the integrand $f(x(\tau), \tau)$ depends on the unknown state trajectory $x(\tau)$ itself. Moreover, even if the trajectory were implicitly defined, evaluating the integral analytically is often difficult due to the complexity and nonlinearity of the dynamics $f(\cdot)$ (e.g., when modeling with deep neural networks). Instead, we must resort to numerical methods to approximate the integration. The core idea behind these methods is **discretization**: instead of tracking the continuous evolution of $x(t)$ at every infinitesimal moment, we approximate the trajectory at a sequence of discrete time steps $\{t_0, t_1, \ldots, t_N\}$. Since evaluating the function $f(\cdot)$ at a finite number of points is computationally inexpensive, discretization provides a very efficient way to obtain approximate solutions.

Discretization can be interpreted from the perspective of Riemann integration. We can approximate the continuous integral in Eq. (8) by partitioning the integration interval into small sub-intervals and summing the areas of rectangles defined by the derivative function. Figure 3 shows an illustration. To formalize this, let us define a temporal grid on the interval $[0, t]$ consisting of $N + 1$ points $t_0 < t_1 < \cdots < t_N$, where $t_0 = 0$ is the start time and $t_N = t$ is the end time.



(a) State $x(t)$ against time t



(b) Dynamics $f(x(t), t)$ against time t

Figure 2: Illustration of solving the ODE $\frac{dx(t)}{dt} = f(x(t), t)$ by integration. The blue shaded area represents the integral $\int_0^t f(x(\tau), \tau) d\tau$. The solution to the ODE at time t is given by the sum of the initial state $x(0)$ and the integral: $x(t) = x(0) + \int_0^t f(x(\tau), \tau) d\tau$.

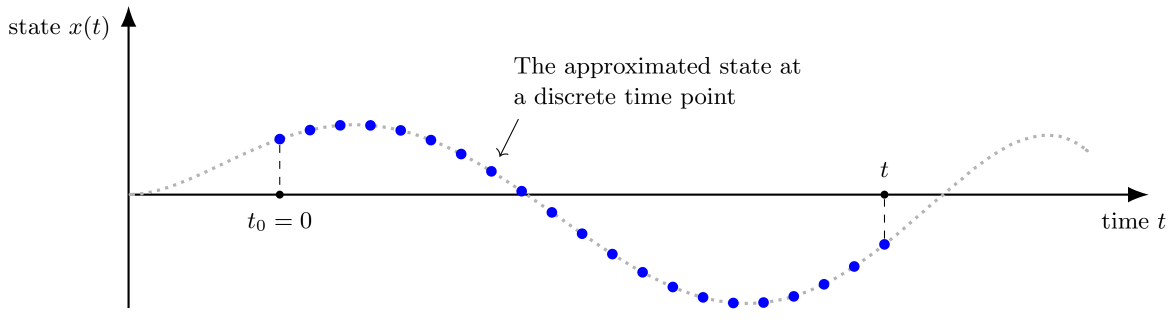
For simplicity, we often assume the time intervals are constant, i.e.,

$$\Delta t \quad = \quad t_{k+1} - t_k, \tag{9}$$

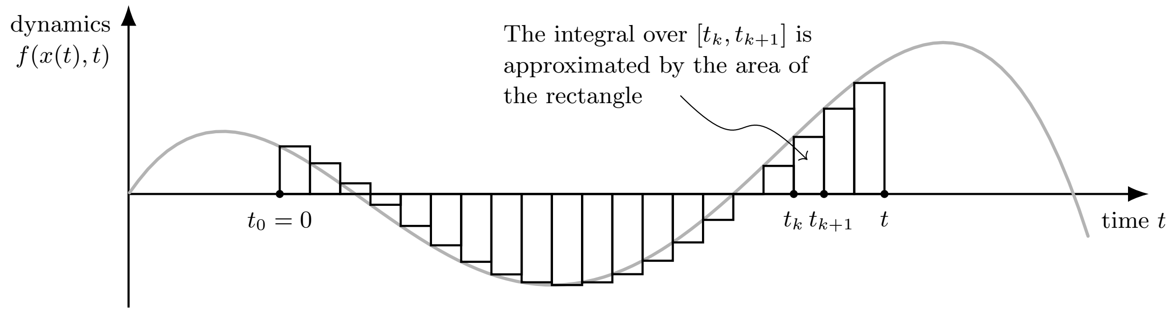
where Δt is referred to as the **step size**.

Our objective is to compute a sequence of values $\{\bar{x}_0, \bar{x}_1, \ldots, \bar{x}_N\}$, where each $\bar{x}_k$ serves as an approximation to the true state $x(t_k)$. The transition from the current state $\bar{x}_k$ to the next state $\bar{x}_{k+1}$ requires approximating the integral of the dynamics over the small interval $[t_k, t_{k+1}]$. Mathematically, we can express the discretized model using the following recurrent form

$$\bar{x}_{k+1} = \bar{x}_k + \text{Approx}\left(\int_{t_k}^{t_{k+1}} f(x(\tau), \tau) d\tau\right), \tag{10}$$



(a) State $x(t)$ against time t



(b) Dynamics $f(x(t), t)$ against time t

Figure 3: Illustration of discretizing an ODE. The function $f(\cdot)$ is evaluated at a number of discrete time points. At each time point, an approximation of the state is obtained, denoted by blue circles. The integration of $f(\cdot)$, which determines the state evolution, is approximated over each sub-interval by the area of a rectangle, given by $\Delta t \cdot \Psi(\bar{x}_k, t_k, \Delta t, f)$, where Δt is the width and $\Psi(\bar{x}_k, t_k, \Delta t, f)$ is the height.

where the function Approx$(\cdot)$ approximates the local integral. In discretized ODE models, this function is generally defined using the local geometry of $f(\cdot)$, like this

$$\text{Approx}\left(\int_{t_k}^{t_{k+1}} f(x(\tau), \tau) d\tau\right) \quad = \quad \Delta t \cdot \Psi(\bar{x}_k, t_k, \Delta t, f). \tag{11}$$

Then, Eq. (10) can be rewritten as

$$\bar{x}_{k+1} = \bar{x}_k + \Delta t \cdot \Psi(\bar{x}_k, t_k, \Delta t, f). \tag{12}$$

Here, $\Psi(\cdot)$ is called the **increment function** or **step function**. It estimates the integral over the interval using the values of $f(\cdot)$ computed at the available discrete points. Different numerical methods correspond to different choices of $\Psi(\cdot)$. For example, choosing $\Psi(\cdot)$ based on the slope at the left endpoint (t_k) corresponds to the classic **Euler method** (analogous to a left Riemann sum). In this case, the increment function is simply the dynamics function itself: $\Psi(\bar{x}_k, t_k, \Delta t, f) = f(\bar{x}_k, t_k)$. Substituting this back into Eq. (12) yields the update rule of the Euler method

$$\bar{x}_{k+1} = \bar{x}_k + \Delta t \cdot f(\bar{x}_k, t_k). \tag{13}$$

As only one function evaluation is required per step, the Euler method is computationally efficient. However, it often suffers from poor accuracy, particularly when the step size Δt is large or when the underlying dynamics $f(\cdot)$ vary rapidly. To address these limitations, numerical solvers generally follow two main strategies: higher-order single-step methods and multi-step methods.

- **Higher-order Single-step Methods**. In numerical analysis, the order of a method refers to the rate at which the approximation error decreases as the step size Δt approaches zero. A method is of order p if the local error scales with $O((\Delta t)^{p+1})$. While the classic Euler method is first-order, higher-order methods improve accuracy by evaluating the dynamics $f(\cdot)$ at multiple intermediate points within the interval $[t_k, t_{k+1}]$ to better estimate the average slope. For instance, the fourth-order **Runge-Kutta method** (RK4) computes a weighted average of four different slopes (one at the start, two at the midpoint, and one at the end). It can achieve high precision without requiring an extremely small step size.

- **Multi-step Methods**. Unlike single-step methods (like Euler and RK4) which discard previous information and compute the next state $\bar{x}_{k+1}$ relying solely on the current state $\bar{x}_k$, multi-step methods exploit the history of the trajectory. By fitting a polynomial to the past derivative values, methods such as the Adams-Bashforth family can predict the integral over the next interval. The primary advantage of multi-step methods is computational efficiency: they can achieve high-order accuracy by reusing previously computed evaluations of $f(\cdot)$, rather than performing multiple new expensive function evaluations per step. However, they are not self-starting and require a single-step method to initialize the first few steps of the trajectory.

There are also other ways to improve ODE solvers. For instance, in **adaptive step-size methods**, the assumption of a constant step size Δt is relaxed. Instead, the solver dynamically adjusts Δt at every step. Here we do not go into details of these ODE solvers, and will use them as standard tools in the following discussion. Interested readers can refer to books on numerical analysis for further details [Stoer et al., 1980; Burden et al., 2015].

Before proceeding, it is necessary to clarify the notation used in this work, as conventions vary across the literature. Typically, the function notation $x(t)$ denotes the continuous-time state variable. In contrast, the subscript notation x_t is frequently used in machine learning and stochastic calculus to represent the state at a specific time snapshot t. In the context of numerical solvers, x_k (or x_{t_k}) often refers to the computed value at the k-th integration step corresponding to time t_k.

In our previous discussion, we distinguished the numerical approximation $\bar{x}_k$ from the true analytical solution $x(t_k)$. However, to avoid cluttered notation, and to make our mathematical representation more consistent with existing literature, we will use x_k (without a bar) to represent the estimated value at step k in numerical methods. More generally, we can use x_k to denote the state at some discrete time step. Although this introduces a slight abuse of notation, the meaning will be clear from the context. For instance, the Euler method can be written as

$$x_{k+1} = x_k + \Delta t \cdot f(x_k, t_k) \tag{14}$$

Likewise, occasionally we may use $x(t)$ instead of $\bar{x}(t)$ to denote the estimated value of the state at time t, where the distinction from the true solution is not critical.

To summarize, given an ODE governed by the dynamics f and an initial state $x(0)$, we can view the ODE solver as a function $\phi(\cdot)$. For any $t \geq 0$, this function returns an approximation of the state, denoted as

$$x(t) \quad = \quad \phi(f, x(0), t), \tag{15}$$

while satisfying the initial condition $x(0) = \phi(f, x(0), 0)$.

1.3 Two Examples

We now illustrate the application of ODEs through two examples. The first example concerns modeling parameter updates in training, and the second example focuses on **continuous-time Markov chains**.

1.3.1 Training with Gradient Descent

The most popular method for training neural networks is gradient descent. The idea is that we first define a loss function $L(\theta)$ that quantifies the discrepancy between the model's predictions and the ground truth, where $\theta \in \mathbb{R}^d$ represents the vector of model parameters. To minimize this loss, we iteratively adjust the parameters in the direction of the steepest descent, which is given by the negative gradient $-\frac{dL(\theta)}{d\theta}$. At each iteration k, the update rule is given by

$$\theta_{k+1} = \theta_k - \eta \cdot \frac{dL(\theta_k)}{d\theta_k}, \tag{16}$$

where $\eta > 0$ is the learning rate.

Recall from Eq. (14) that the Euler method for solving ODEs is given by $x_{k+1} = x_k + \Delta t \cdot f(x_k, t_k)$. By comparing the two equations, we observe a direct correspondence between them:

$$\begin{aligned} \text{State } x_k \quad &\Leftrightarrow \quad \text{Parameters } \theta_k \\ \text{Step Size } \Delta t \quad &\Leftrightarrow \quad \text{Learning Rate } \eta \\ \text{Dynamics } f(x_k, t_k) \quad &\Leftrightarrow \quad \text{Negative Gradient } -dL(\theta_k)/d\theta_k \end{aligned}$$

Considering the similarity between these equations, we can view gradient descent as the explicit Euler discretization of a continuous-time ODE. If we take the limit as the learning rate $\eta \to 0$

and correspondingly the number of steps $k \to \infty$, the sequence of discrete parameter updates converges to the solution of the following ODE

$$\frac{d\theta_t}{dt} = -\frac{dL(\theta_t)}{d\theta_t}. \tag{17}$$

This ODE is formally known as **gradient flow**. Imagine a massless particle moving on a complex surface (e.g., the loss landscape). Since the particle has no mass, its instantaneous velocity is determined entirely by the slope of the terrain. The steeper the slope, the faster it moves, guided strictly by the direction of the negative gradient. This continuous descent process is exactly what Eq. (17) describes.

By connecting gradient descent with ODEs, we can use the well-established theory of ODEs to better understand the behavior of neural network optimization. For instance, the stability of the ODE solution provides insights into why training might diverge if the learning rate is too large (a phenomenon known as stiffness in numerical analysis). Furthermore, this perspective can extend naturally to more advanced optimizers. Consider stochastic gradient descent with momentum [Qian, 1999], which is designed to accelerate training by accumulating past gradients. The update rule is typically formulated as a system of two coupled equations:

$$v_{k+1} = \mu v_k - \eta \cdot \frac{dL(\theta_k)}{d\theta_k} \tag{18}$$

$$\theta_{k+1} = \theta_k + v_{k+1}, \tag{19}$$

where v_k represents the momentum variable at step k, and $\mu \in [0, 1)$ is the momentum coefficient that controls how much of the past accumulation is retained.

If we analyze the continuous-time limit of these discrete updates (by letting the step size $\eta \to 0$), we no longer obtain a first-order ODE. Instead, the system converges to a second-order ODE [Su et al., 2016]

$$\frac{d^2\theta_t}{dt^2} + \lambda \frac{d\theta_t}{dt} = -\frac{dL(\theta_t)}{d\theta_t}, \tag{20}$$

where λ is a coefficient related to the damping friction. This equation is physically analogous to Newton's second law of motion and is formally known as the **heavy ball ODE**. The interested reader can refer to Appendix A for a more detailed derivation of the heavy ball ODE.

1.3.2 Continuous-time Markov Chains

A continuous-time Markov chain (CTMC) describes a stochastic process that moves among a finite number of discrete states, where the transitions occur continuously in time [Norris, 1998]. While CTMCs are often introduced via jump processes, they can be modeled as a deterministic linear dynamical system governing the evolution of probability distributions. This perspective connects probability theory to ODEs.

Consider a system with N discrete states, denoted by the set $\mathcal{S} = \{1, 2, \ldots, N\}$. Instead of tracking a single random trajectory $x(t)$, we focus on the evolution of the probability distribution over all states. Let $\mathbf{p}(t) = \begin{bmatrix} p_1(t) & p_2(t) & \cdots & p_N(t) \end{bmatrix}^\top \in \mathbb{R}^N$ be a column vector where the i-th component represents the probability of being in state i at time t:

$$p_i(t) \quad = \quad \Pr(x(t) = i), \tag{21}$$

subject to

$$\sum_{i=1}^{N} p_i(t) \quad = \quad 1. \tag{22}$$

To describe the evolution of the system, we need to specify how this probability distribution changes with respect to time. Intuitively, we can view this as a flow of probability mass between states. The dynamics of this flow are defined by a transition rate matrix $\mathbf{Q} \in \mathbb{R}^{N\times N}$, often called the infinitesimal generator. The elements of $\mathbf{Q}$ quantify the rate of probability mass moving between states:

- Off-diagonal elements (q_{ij} for $i \neq j$): represent the instantaneous rate of transition from state i to state j. These must be non-negative ($q_{ij} \geq 0$).

- Diagonal elements (q_{ii}): defined as $q_{ii} = -\sum_{j\neq i} q_{ij}$. This ensures that each row sums to zero, reflecting the conservation of total probability (mass flowing out must equal the total rate of departure).

Given the infinitesimal generator, the evolution of the probability vector $\mathbf{p}(t)$ can be defined by a system of linear ODEs known as the **Kolmogorov forward equation** (or the **master equation** in physics):

$$\frac{d\mathbf{p}(t)}{dt} \quad = \quad \mathbf{Q}^\top \mathbf{p}(t). \tag{23}$$

This ODE represents a global balance of probability currents. The matrix product $\mathbf{Q}^\top \mathbf{p}(t)$ aggregates the probability flow for each state. Specifically, for a state j, the rate of change $\frac{dp_j}{dt}$ is determined by the difference between the total probability mass entering j from all other states (inflow) and the mass departing from j (outflow). Thus, Eq. (23) simply states that the rate of change of the probability distribution depends linearly on the current distribution and the transition rates.

If $\mathbf{Q}$ is constant, the CTMC is time-homogeneous. In this case, Eq. (23) is a standard linear ODE system, and its analytical solution is given by the matrix exponential

$$\mathbf{p}(t) = e^{\mathbf{Q}^\top t}\mathbf{p}(0), \tag{24}$$

where $\mathbf{p}(0)$ is the initial distribution, and $e^{\mathbf{Q}^\top t}$ is the propagator that maps the initial distribution to the distribution at time t. In Section 5, we show that CTMCs can serve as the foundation for discrete diffusion models.

Appendix A. Deriving the Heavy Ball ODE

Recall that stochastic gradient descent with momentum defines the following update rule

$$v_{k+1} = \mu v_k - \eta \cdot \frac{dL(\theta_k)}{d\theta_k} \tag{25}$$

$$\theta_{k+1} = \theta_k + v_{k+1}, \tag{26}$$

where $\mu \in [0, 1)$ is the momentum coefficient and $\eta > 0$ is the learning rate.

To derive the continuous-time limit, we first eliminate the momentum (or velocity) variable v_k. From Eq. (26), we have $v_{k+1} = \theta_{k+1} - \theta_k$, which implies $v_k = \theta_k - \theta_{k-1}$. Substituting these expressions into Eq. (25) yields a single recurrence relation for the parameters

$$\theta_{k+1} - \theta_k = \mu(\theta_k - \theta_{k-1}) - \eta \cdot \frac{dL(\theta_k)}{d\theta_k}. \tag{27}$$

Rearranging the terms to identify finite difference approximations, we obtain

$$\underbrace{(\theta_{k+1} - 2\theta_k + \theta_{k-1})}_{\text{Acceleration Term}} + \underbrace{(1-\mu)(\theta_k - \theta_{k-1})}_{\text{Velocity Term}} = -\eta \cdot \frac{dL(\theta_k)}{d\theta_k}. \tag{28}$$

To obtain a meaningful limit, we introduce a temporal step size Δt such that $t = k \cdot \Delta t$. Following the standard scaling analysis [Polyak, 1964; Su et al., 2016], we assume the learning rate scales quadratically with the step size ($\eta = (\Delta t)^2$) and the momentum coefficient scales as $\mu = 1 - \lambda(\Delta t)^2$, where $\lambda > 0$ acts as a damping factor. Substituting these scalings into the equation and dividing by $(\Delta t)^2$, we get

$$\frac{\theta_{k+1} - 2\theta_k + \theta_{k-1}}{(\Delta t)^2} + \lambda \cdot \frac{\theta_k - \theta_{k-1}}{\Delta t} = -\frac{dL(\theta_k)}{d\theta_k}. \tag{29}$$

Taking the limit as $\Delta t \to 0$, the finite difference quotients converge to the second and first derivatives with respect to time, respectively. Finally, the sequence of updates converges to the heavy ball ODE

$$\frac{d^2\theta_t}{dt^2} + \lambda \cdot \frac{d\theta_t}{dt} = -\frac{dL(\theta_t)}{d\theta_t}. \tag{30}$$

References

Tom M Apostol. Calculus, Volume 1. John Wiley & Sons, 1991.

Richard L. Burden, J. Douglas Faires, and Annette M. Burden. Numerical Analysis. Cengage Learning, 2015.

William Dunham. The calculus gallery: Masterpieces from Newton to Lebesgue. Princeton University Press, 2005.

CH Jr Edwards. The historical development of the calculus. Springer Science & Business Media, 1994.

Philip Hartman. Ordinary differential equations. SIAM, 2002.

Morris W Hirsch, Stephen Smale, and Robert L Devaney. Differential equations, dynamical systems, and an introduction to chaos. Academic press, 2013.

James R Norris. Markov chains. Cambridge university press, 1998.

Boris T Polyak. Some methods of speeding up the convergence of iteration methods. Ussr computational mathematics and mathematical physics, 4(5):1–17, 1964.

Ning Qian. On the momentum term in gradient descent learning algorithms. Neural networks, 12(1):145–151, 1999.

Michael Spivak. Calculus. Cambridge University Press, 2006.

Josef Stoer, Roland Bulirsch, R Bartels, Walter Gautschi, and Christoph Witzgall. Introduction to numerical analysis. Springer, 1980.

Weijie Su, Stephen Boyd, and Emmanuel J Candes. A differential equation for modeling nesterov's accelerated gradient method: Theory and insights. Journal of Machine Learning Research, 17 (153):1–43, 2016.