Lec 18: Ordinary Differential Equations

15-369/769: Numerical Methods (Foundations, ML & Visual Computing)

Instructor: Minchen Li

Table of Content

- Motivation
- Theory of ODEs
- Time-Stepping Schemes
- Multivalue Methods

Table of Content

- Motivation
- Theory of ODEs
- Time-Stepping Schemes
- Multivalue Methods

Motivation

From Interpolating $f(x)$ to Finding Functions

- We now design unknown functions f subject to additional criteria rather than merely filling in missing values.
- Common design goals:
  - Find $f \approx f_0$ while enforcing smoothness / continuity / bounds.
  - Simulate a process $f(t)$ evolving in time.
  - Find f matching f_0 in value but sharing properties with a different function g_0.
- Ordinary Differential Equations (ODEs): find $\mathbf{f} : \mathbb{R}^+ \to \mathbb{R}^n$ given $\mathbf{f}(0)$ and an equation relating $\mathbf{f}$ and its derivatives.
- Initial Value Problem (IVP): predict $\mathbf{f}(t)$ for $t > 0$ from initial data.

Motivation

Where ODEs Arise (I) — Simulation

- **Newton's Second Law**
  - For one particle in 1D: $m\,x''(t) = f(t, x, x')$, where f is the applied force.
  - For n particles in 3D, stack positions into $\mathbf{x}(t) \in \mathbb{R}^{3n}$ and masses into M:

$$M\mathbf{x}''(t) = \mathbf{f}\big(t, \mathbf{x}(t), \mathbf{x}'(t)\big) \quad \Rightarrow \quad \mathbf{x}''(t) = M^{-1}\mathbf{f}\big(t, \mathbf{x}(t), \mathbf{x}'(t)\big).$$

  - Given initial positions and velocities, an ODE integrator advances the simulation in time.
- **Molecular dynamics:** nonlinear force fields create fast and slow time scales.
- **Crowd / agent simulation:** velocities depend on neighbors and environment; the integrator controls numerical stability and artifacts.

Motivation

Where ODEs Arise (II) — Optimization

- **Gradient descent as an ODE**
  - Discrete update with step $h > 0$:
    $$\mathbf{x}_{i+1} = \mathbf{x}_i - h\nabla E(\mathbf{x}_i).$$
  - Rewrite as a difference quotient:
    $$\frac{\mathbf{x}_{i+1} - \mathbf{x}_i}{h} = -\nabla E(\mathbf{x}_i).$$
  - Taking $h \to 0$ suggests the *gradient flow*
    $$\mathbf{x}'(t) = -\nabla E(\mathbf{x}(t)).$$
- Example: solving $A\mathbf{x} = \mathbf{b}$ ($A \succ 0$) by minimizing $E(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{b}^T\mathbf{x}$ gives ODE $\mathbf{x}'(t) = \mathbf{b} - A\mathbf{x}(t)$.

Motivation

Where ODEs Arise (III) — Neural ODEs

- **Generative modeling:** evolve a simple random $\mathbf{x}(0) \sim \mathcal{N}(0, I)$ via
$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{f}_\theta\big(t, \mathbf{x}(t)\big),$$
where $\mathbf{f}_\theta$ is a neural vector field, then output $\mathbf{x}(T)$.
- **Learning:** adjust θ so that generated samples match the data distribution.
- **Practical use:** standard ODE integrators execute the learned dynamics at test time.

Table of Content

- Motivation
- Theory of ODEs
- Time-Stepping Schemes
- Multivalue Methods

Theory of ODEs

Canonical IVP

- **Problem form**

$$\text{Find } \mathbf{f}(t) : \mathbb{R}^+ \to \mathbb{R}^n \text{ s.t. } \mathbf{F}\left[t, \mathbf{f}(t), \mathbf{f}'(t), \ldots, \mathbf{f}^{(k)}(t)\right] = \mathbf{0},$$

$$\text{given } \mathbf{f}(0), \mathbf{f}'(0), \ldots, \mathbf{f}^{(k-1)}(0).$$

- Examples (scalar $y = f(t)$):
  - $y' = 1 + \cos t$ (integrable in t; also solvable by quadrature).
  - $y' = ay$ (linear, time-invariant).
  - $y' = ay + e^t$ (time-dependent forcing).
  - $y'' + 3y' - y = t$ (second order, linear).
  - $y'' \sin y = e^t y'$ (nonlinear).

Theory of ODEs

Explicit ODEs

- **Definition (Explicit ODE).**

$$\mathbf{f}^{(k)}(t) = \mathbf{F}\left[t, \mathbf{f}(t), \mathbf{f}'(t), \ldots, \mathbf{f}^{(k-1)}(t)\right].$$

- Many implicit forms can be reduced to explicit by root-finding; but this approach can fail in the presence of multiple roots.
- Key viewpoint: any explicit k-th order ODE can be written as a first-order system.
- For a scalar second-order ODE $y'' = F[t, y, y']$:
  - Let $z := \dfrac{dy}{dt}$.
  - Then

$$\frac{d}{dt}\begin{pmatrix} y \\ z \end{pmatrix} = \begin{pmatrix} z \\ F[t, y, z] \end{pmatrix}.$$

Theory of ODEs

Reducing Higher Order to First Order

- For the general k-th order explicit ODE for $\mathbf{f} : \mathbb{R}^+ \to \mathbb{R}^n$:
  - Define $\mathbf{f}_0 = \mathbf{f}$, $\mathbf{f}_1 = \mathbf{f}'$, …, $\mathbf{f}_{k-1} = \mathbf{f}^{(k-1)}$.
  - First-order system in dimension nk:

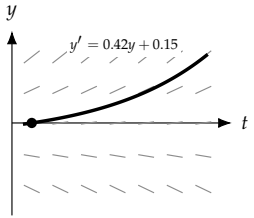
$$\frac{d}{dt}\begin{pmatrix} \mathbf{f}_0 \\ \mathbf{f}_1 \\ \mathbf{f}_2 \\ \vdots \\ \mathbf{f}_{k-1} \end{pmatrix} = \begin{pmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \\ \mathbf{f}_3 \\ \vdots \\ \mathbf{F}[t, \mathbf{f}_0, \mathbf{f}_1, \ldots, \mathbf{f}_{k-1}] \end{pmatrix}.$$

- Trade-off: fewer derivatives but higher system dimension.
- From here on, we focus mainly on the first-order form $\mathbf{y}' = \mathbf{F}(t, \mathbf{y})$.

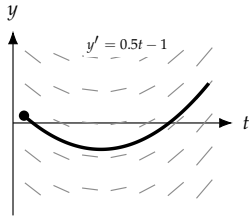
Theory of ODEs

How to Visualize an ODE

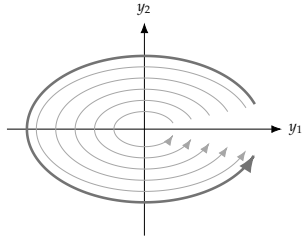
- For a scalar first-order ODE $y' = F(t, y)$, a **slope field** draws a short line segment with slope $F(t, y)$ at many points in the (t, y) plane.
- A solution curve is one path that follows those local slopes from its initial condition.
- For autonomous systems $\mathbf{y}' = \mathbf{F}(\mathbf{y})$ in two or three dimensions, plot trajectories directly in **state space**; each point is the current state $\mathbf{y}$.



Time-independent



Time-dependent

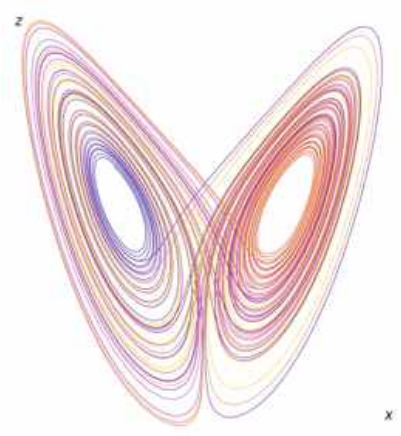


$\mathbf{y} = (y_1, y_2)$: 2D first-order state

Theory of ODEs

State-Space Example: Lorenz System

- A three-variable ODE defines a velocity vector at every state (x, y, z).
- Starting from one initial condition traces a single trajectory through 3D state space.
- Nearby initial conditions separate rapidly, even though trajectories remain in the familiar bounded "butterfly" region.
- This is continuous-system sensitivity, not a numerical-instability artifact.



Numerical Lorenz-attractor rendering; standard parameters $\sigma = 10$, $\rho = 28$, $\beta = 8/3$.

Theory of ODEs

Existence & Uniqueness: Incompatible Initial Condition

- Before discretizing, the continuous IVP itself must be well posed.
- Consider the implicit first-order ODE

$$\boxed{t\,y' = 2y}, \qquad y(0) = y_0.$$

- Evaluate the differential equation at the initial point:

$$0 \cdot y'(0) = 2y(0) \quad \Rightarrow \quad \boxed{y(0) = 0}.$$

- Therefore, if $y_0 \neq 0$, **no differentiable solution exists**: the initial condition directly conflicts with the differential equation.
- For $t \neq 0$, the solutions are $y(t) = Ct^2$, which are consistent only with $y(0) = 0$.

Theory of ODEs

Existence & Uniqueness: Nonuniqueness at a Degenerate Point

- Use the same ODE $t\,y' = 2y$, now with the compatible initial condition $y(0) = 0$.
- For every constant C,

$$\boxed{y(t) = Ct^2}$$

  is a smooth solution, including at $t = 0$:

$$t\,y'(t) = t(2Ct) = 2Ct^2 = 2y(t).$$

- Thus $y(0) = 0$ does **not** determine C: infinitely many solutions satisfy the same initial condition.
- Why can uniqueness fail? At $t = 0$, the coefficient of y' vanishes, so the equation no longer determines y' and cannot be rewritten locally as $y' = F(t, y)$.

Theory of ODEs

Existence and Uniqueness: What the Assumptions Buy Us

Global autonomous version. If $\mathbf{F} : \mathbb{R}^n \to \mathbb{R}^n$ is **globally Lipschitz**,

$$\|\mathbf{F}(\mathbf{y}) - \mathbf{F}(\mathbf{x})\|_2 \le L\|\mathbf{y} - \mathbf{x}\|_2 \qquad \text{for all } \mathbf{x}, \mathbf{y},$$

then for every initial value $\mathbf{f}(0) = \mathbf{f}_0$, the IVP $\mathbf{f}'(t) = \mathbf{F}(\mathbf{f}(t))$ has one unique solution for all $t \ge 0$.

Locally Lipschitz: unique, but may blow up

$$y' = y^2, \qquad y(0) = 1 \quad \Rightarrow \quad \boxed{y(t) = \frac{1}{1-t}}.$$

$F(y) = y^2$ is smooth, so the solution is locally unique, but $y(t) \to \infty$ as $t \to 1^-$. Thus local Lipschitz does not guarantee existence for all time.

Continuous, not Lipschitz: may be nonunique

$$y' = \sqrt{|y|}, \qquad y(0) = 0.$$

Besides $y(t) = 0$, for any $c \ge 0$,

$$y(t) = \begin{cases} 0, & t \le c, \\ \dfrac{(t-c)^2}{4}, & t > c, \end{cases}$$

is also a solution.

Takeaway: global Lipschitz control rules out both failure modes.

Theory of ODEs

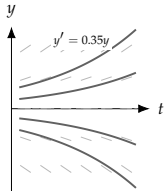
Model Equation: Shift to the Essential Dynamics

- Near a state, a scalar ODE often looks locally like
$$y' = ay + b.$$
- For $a \neq 0$, shift the equilibrium by
$$\tilde{y} := y + \frac{b}{a}.$$
Then
$$\tilde{y}' = a\tilde{y}, \qquad \tilde{y}(t) = Ce^{at}.$$
- Returning to the original variable,
$$\boxed{y(t) = Ce^{at} - \frac{b}{a}}.$$
- The constant forcing changes the equilibrium location; a controls growth, decay, and stiffness.

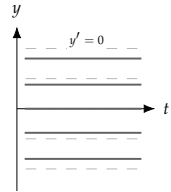
Theory of ODEs

Qualitative Behavior (Scalar)

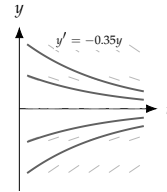
- $a > 0$: solutions grow $\rightarrow$ instability; nearby ICs diverge. *(IC: initial conditions)*
- $a = 0$: constant solutions; distances preserved.
- $a < 0$: solutions decay to 0; nearby ICs converge $\rightarrow$ stability.



$a > 0$: solutions move away from 0



$a = 0$: every solution is constant

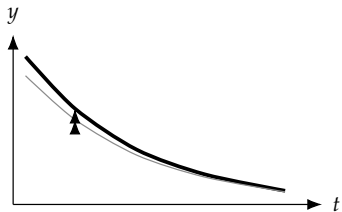


$a < 0$: solutions decay toward 0

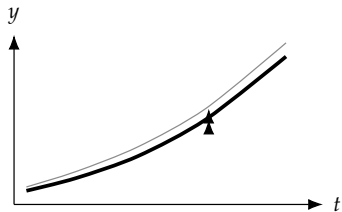
Theory of ODEs

Stability (Scalar)

- Terminology:
  - **Stable** if perturbations do not grow with time.
  - **Unstable** otherwise.



Stable ($a < 0$)



Unstable ($a > 0$)

Theory of ODEs

Multivariate Extension: Eigenvalues Control Local Behavior

- For $\mathbf{y}' = A\mathbf{y}$ with diagonalizable A,
$$\mathbf{y}(t) = \sum_k c_k e^{\lambda_k t}\mathbf{v}_k.$$
- All modes decay when $\operatorname{Re}\lambda_k < 0$; for diagonalizable A, non-growing behavior allows $\operatorname{Re}\lambda_k \le 0$.
- For a nonlinear autonomous system $\mathbf{y}' = \mathbf{F}(\mathbf{y})$, linearize around an **equilibrium** $\mathbf{y}^\star$ satisfying $\mathbf{F}(\mathbf{y}^\star) = \mathbf{0}$.
- With perturbation $\delta\mathbf{y} := \mathbf{y} - \mathbf{y}^\star$,
$$\delta\mathbf{y}' \approx J_{\mathbf{F}}(\mathbf{y}^\star)\,\delta\mathbf{y}.$$
- Hence the eigenvalues of the Jacobian $J_{\mathbf{F}}(\mathbf{y}^\star)$ play the role of the scalar parameter a.

Table of Content

- Motivation
- Theory of ODEs
- Time-Stepping Schemes
- Multivalue Methods

Time-Stepping Schemes

Discrete Viewpoint & The Accuracy–Stability Tension

- Goal: given $\mathbf{y}(t)$ at $t = t_k$, predict $\mathbf{y}(t_{k+1})$ at $t_{k+1} = t_k + h$.
- Generate a sequence $\{\mathbf{y}_k\}$ via a *time-stepping scheme* (integrator).
- Two central qualities:
  - **Accuracy:** truncation error size per step / globally.
  - **Stability:** does the discrete solution blow up on problems where the ODE decays?
- Often a compromise: very accurate methods may be unstable for large h, while very stable methods may be less accurate per cost.

Time-Stepping Schemes

Forward Euler

- Start from the forward difference:

$$\mathbf{F}[\mathbf{y}_k] = \mathbf{y}'(t_k) = \frac{\mathbf{y}_{k+1} - \mathbf{y}_k}{h} + O(h).$$

- Solve for the next iterate:

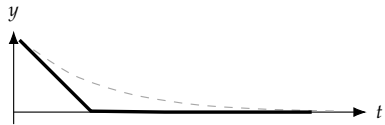
$$\boxed{\mathbf{y}_{k+1} = \mathbf{y}_k + h\mathbf{F}[\mathbf{y}_k] + O(h^2)} \qquad \Rightarrow \qquad \mathbf{y}_{k+1} \approx \mathbf{y}_k + h\mathbf{F}[\mathbf{y}_k].$$

- Error notions:
  - Local truncation error: $O(h^2)$ (one step).
  - After $O(1/h)$ steps to reach fixed t, the global error is $O(h)$.
  - Hence Forward Euler is **first-order accurate**.
- Explicit method: next state given by a formula; inexpensive per step.

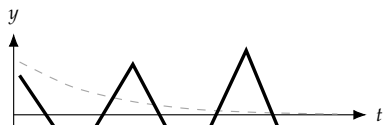
Time-Stepping Schemes

Stability of Forward Euler

- For the scalar test equation $y' = ay$, Forward Euler gives
$$y_{k+1} = R(z)y_k, \qquad R(z) = 1 + z, \qquad z := ha.$$
- Numerical stability requires $|R(z)| \leq 1$.
- For real $a < 0$,
$$0 < h \leq \frac{2}{|a|}.$$
- The key point is that stability depends on the **product** ha, not on a alone.
- Same stable ODE, $a = -1$:
  - $h = 1$: $R = 0$ — stable decay.
  - $h = 2.5$: $R = -1.5$ — alternating growth and numerical blow-up.



$a = -1,\ h = 1$: stable



$a = -1,\ h = 2.5$: unstable

Time-Stepping Schemes

Stiff Problems

- Stiff IVPs: stable continuous dynamics but *explicit* methods (e.g., Forward Euler) require prohibitively small h for stability.
- Typical sources: rapidly decaying modes (large negative eigenvalues), fast transients, multiple time scales.
- Remedy preview:
  - Implicit schemes (e.g., backward Euler) have much larger stability regions and often no step-size restriction for $a < 0$.
- We trade per-step cost (solve a system) for robustness.

Time-Stepping Schemes

Backward Euler

- Apply backward difference at t_{k+1}:

$$\mathbf{F}[\mathbf{y}_{k+1}] = \mathbf{y}'(t+h) = \frac{\mathbf{y}_{k+1} - \mathbf{y}_k}{h} + O(h).$$

- Rearranging gives an implicit update:

$$\boxed{\mathbf{y}_{k+1} = \mathbf{y}_k + h\mathbf{F}[\mathbf{y}_{k+1}]}.$$

- Requires solving for $\mathbf{y}_{k+1}$, possibly nonlinear $\Rightarrow$ **implicit method**.
- More expensive than Forward Euler but often more stable.

Time-Stepping Schemes

Backward Euler: Linear System Case

- For $\mathbf{y}' = A\mathbf{y}$,

$$\mathbf{y}_{k+1} = \mathbf{y}_k + hA\mathbf{y}_{k+1} \quad \Rightarrow \quad \boxed{(I_n - hA)\mathbf{y}_{k+1} = \mathbf{y}_k}.$$

- This is a **linear solve**; do not explicitly form $(I_n - hA)^{-1}$.
- If A and h are fixed, factorize $I_n - hA$ once and reuse the factorization at every timestep.
- For nonlinear $\mathbf{F}$, the same implicit idea instead requires a nonlinear solve (e.g., Newton's method).

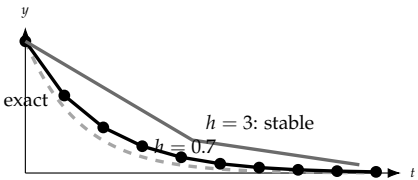
Time-Stepping Schemes

Stability of Backward Euler

- For the scalar test equation $y' = ay$,

$$y_{k+1} = \frac{1}{1 - ha}\, y_k, \qquad R(z) = \frac{1}{1 - z}.$$

- For every real $a < 0$ and every $h > 0$, $|R(ha)| < 1$.
- More generally, Backward Euler is **A-stable**: every mode with $\mathrm{Re}(ha) \leq 0$ lies in its stability region.
- It is also strongly damping for very stiff modes, which is useful for dissipative problems but can overdamp oscillatory dynamics.



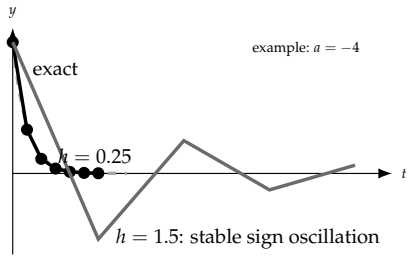
Time-Stepping Schemes

Trapezoidal Method

- If $\mathbf{F}[\mathbf{y}_{k+1/2}]$ at midpoint $t + h/2$ is available,
$$\mathbf{y}_{k+1} = \mathbf{y}_k + h\mathbf{F}[\mathbf{y}_{k+1/2}] + O(h^3).$$
- From Taylor expansion,
$$\frac{\mathbf{F}[\mathbf{y}_{k+1}] + \mathbf{F}[\mathbf{y}_k]}{2} = \mathbf{F}[\mathbf{y}_{k+1/2}] + O(h^2).$$
- Substitute to obtain:
$$\boxed{\mathbf{y}_{k+1} = \mathbf{y}_k + \frac{h}{2}\left(\mathbf{F}[\mathbf{y}_k] + \mathbf{F}[\mathbf{y}_{k+1}]\right)}.$$
- Second-order accurate; implicit since $\mathbf{y}_{k+1}$ appears on both sides.

Time-Stepping Schemes
Stability of Trapezoidal Method

- For the scalar test equation $y' = ay$,
$$R(z) = \frac{1 + z/2}{1 - z/2}, \qquad z = ha.$$
- For real $a < 0$, $|R(ha)| < 1$ for every $h > 0$; the method is also **A-stable**.
- Unlike Backward Euler, $R(z) \to -1$ as $z \to -\infty$.
- Therefore very stiff modes may alternate in sign instead of being rapidly damped: stable, but potentially oscillatory for large h.



Time-Stepping Schemes

Integral Formulation

- Fundamental theorem of calculus:

$$\mathbf{y}_{k+1} = \mathbf{y}_k + \int_{t_k}^{t_k+h} \mathbf{F}[\mathbf{y}(t)]\, dt.$$

- Approximate integral using quadrature $\Rightarrow$ Runge–Kutta (RK) family.
- Using trapezoidal rule:

$$\mathbf{y}_{k+1} = \mathbf{y}_k + \frac{h}{2}\left(\mathbf{F}[\mathbf{y}_k] + \mathbf{F}[\mathbf{y}_{k+1}]\right) + O(h^3).$$

- To make explicit, approximate $\mathbf{F}[\mathbf{y}_{k+1}]$ via Forward Euler:

$$\mathbf{F}[\mathbf{y}_{k+1}] \approx \mathbf{F}[\mathbf{y}_k + h\mathbf{F}[\mathbf{y}_k]].$$

- **Heun's method:**

$$\boxed{\mathbf{y}_{k+1} = \mathbf{y}_k + \frac{h}{2}\left(\mathbf{F}[\mathbf{y}_k] + \mathbf{F}[\mathbf{y}_k + h\mathbf{F}[\mathbf{y}_k]]\right).}$$

Time-Stepping Schemes

Stability of Heun's Method

- Applied to the scalar test equation $y' = ay$,

$$y_{k+1} = \left(1 + ha + \frac{1}{2}h^2a^2\right) y_k, \qquad R(z) = 1 + z + \frac{1}{2}z^2.$$

- Along the negative real axis, stability requires

$$\boxed{-2 \leq z = ha \leq 0}.$$

- Thus for real $a < 0$, Heun has the same step-size stability limit as Forward Euler,

$$h \leq \frac{2}{|a|},$$

but improves accuracy from first to second order.

Time-Stepping Schemes

Classical 4th-Order Runge–Kutta (RK4)

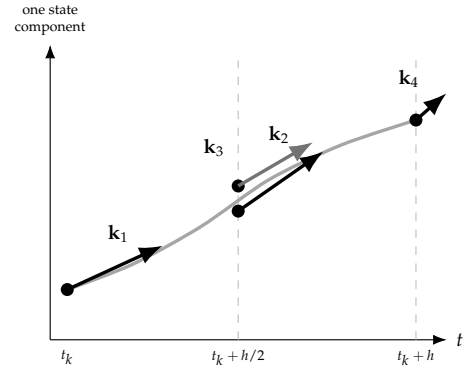
- RK4 probes four slope vectors within one timestep:

$$\begin{aligned}\mathbf{k}_1 &= \mathbf{F}[\mathbf{y}_k],\\ \mathbf{k}_2 &= \mathbf{F}[\mathbf{y}_k + \tfrac{h}{2}\mathbf{k}_1],\\ \mathbf{k}_3 &= \mathbf{F}[\mathbf{y}_k + \tfrac{h}{2}\mathbf{k}_2],\\ \mathbf{k}_4 &= \mathbf{F}[\mathbf{y}_k + h\mathbf{k}_3].\end{aligned}$$

- Combine them as a weighted average:

$$\boxed{\mathbf{y}_{k+1} = \mathbf{y}_k + \frac{h}{6}(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4)}.$$

- Fourth-order accurate; four $\mathbf{F}$ evaluations per step.

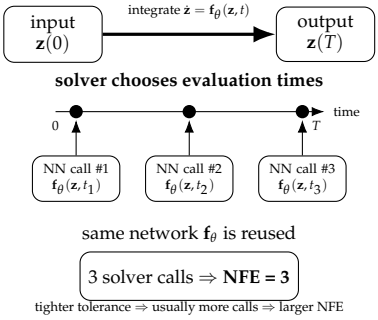


sample slope vectors at start, midpoint, midpoint, end

Time-Stepping Schemes

ML Application: Neural ODEs

- A residual block resembles one Forward Euler step: $\mathbf{z}_{k+1} = \mathbf{z}_k + h\,\mathbf{f}_\theta(\mathbf{z}_k, t_k)$.
- Neural ODEs make depth continuous: $\boxed{\dfrac{d\mathbf{z}}{dt} = \mathbf{f}_\theta(\mathbf{z}, t)}$. Here $\mathbf{f}_\theta$ is a neural network giving the instantaneous change of the feature state.
- A numerical ODE solver repeatedly calls the *same* network to move from $\mathbf{z}(0)$ to $\mathbf{z}(T)$.
- **NFE = number of function evaluations**: the number of solver calls to $\mathbf{f}_\theta$.
- Solver choice / tolerance trades **accuracy** against **NFE and runtime**.



Reference: Chen et al., "Neural Ordinary Differential Equations," NeurIPS 2018.

Time-Stepping Schemes

Exponential Integrators: Semilinear Split

- For $\mathbf{y}' = A\mathbf{y}$, the exact one-step map is $\mathbf{y}_{k+1} = e^{hA}\mathbf{y}_k$. For nonlinear dynamics, choose a **fixed linear part** over the step and write

$$\mathbf{y}' = A\mathbf{y} + \mathbf{G}(\mathbf{y}), \qquad \mathbf{G}(\mathbf{y}) := \mathbf{F}(\mathbf{y}) - A\mathbf{y}.$$

- A may be a known stiff linear operator, or a Jacobian frozen at the current state.
- Variation of constants gives

$$\boxed{\mathbf{y}_{k+1} = e^{hA}\mathbf{y}_k + \int_0^h e^{(h-\tau)A}\mathbf{G}(\mathbf{y}(t_k+\tau))\,d\tau}.$$

- Exponential integrators approximate only the remaining integral while treating the chosen linear dynamics exactly.

Time-Stepping Schemes

Exponential Euler

- Freeze the nonlinear term over one step: $\mathbf{G}(\mathbf{y}(t_k + \tau)) \approx \mathbf{G}(\mathbf{y}_k)$.
- Then

$$\mathbf{y}_{k+1} = e^{hA}\mathbf{y}_k + h\,\varphi_1(hA)\mathbf{G}(\mathbf{y}_k),$$

where

$$\boxed{\varphi_1(z) = \frac{e^z - 1}{z}}, \qquad \varphi_1(0) = 1.$$

- This form remains well-defined even when A is singular; it avoids writing $A^{-1}(e^{hA} - I)$.
- In large systems, implementations usually compute the **action** $e^{hA}\mathbf{v}$ or $\varphi_1(hA)\mathbf{v}$ rather than forming a dense matrix exponential.

Time-Stepping Schemes

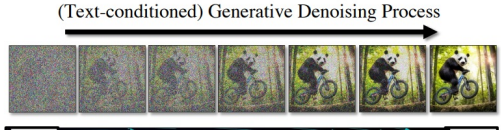
ML Application: Diffusion / Flow Matching as ODE Solving

- Flow matching learns a velocity field

$$\boxed{\frac{d\mathbf{x}}{dt} = \mathbf{v}_\theta(\mathbf{x}, t)}, \qquad \mathbf{x}(0) \sim \mathcal{N}(\mathbf{0}, I),$$

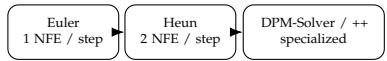
  then integrates from noise to a data sample $\mathbf{x}(1)$.
- Diffusion sampling can likewise be written as a deterministic **probability-flow ODE**.
- Same trained model, different inference solver: Euler, Heun, or specialized methods such as DPM-Solver / ++.
- Each neural-network call is expensive, so solver choice trades

  NFE / runtime / image quality.



Actual text-conditioned diffusion sampling: random noise → progressively denoised image.

Source: Balaji et al., eDiff-I (NVIDIA, 2022).



higher-order / specialized integration can reduce steps for a target quality

References: Balaji et al., eDiff-I; Lipman et al., Flow Matching; Lu et al., DPM-Solver (NeurIPS 2022); Hugging Face Diffusers FlowMatch Euler/Heun schedulers.

Table of Content

- Motivation
- Theory of ODEs
- Time-Stepping Schemes
- Multivalue Methods

Multivalue Methods

Second-Order Dynamics: Keep the Natural Structure

- For mechanics and other second-order systems, it is often useful to keep position, velocity, and acceleration explicitly rather than treating everything as an arbitrary first-order state.
- For $\mathbf{y}''(t) = \mathbf{F}[t, \mathbf{y}, \mathbf{y}']$, define:

$$\mathbf{v}(t) = \mathbf{y}'(t) \quad \text{(velocity)}, \qquad \mathbf{a}(t) = \mathbf{y}''(t) = \mathbf{F}[t, \mathbf{y}, \mathbf{v}] \quad \text{(acceleration)}.$$

- Equivalently,

$$\begin{cases} \mathbf{y}' = \mathbf{v}, \\ \mathbf{v}' = \mathbf{a} = \mathbf{F}[t, \mathbf{y}, \mathbf{v}]. \end{cases}$$

- This structure motivates Newmark and leapfrog / Verlet integrators.

Multivalue Methods

Derivation of Newmark Integrators

- Define position, velocity, and acceleration: $\mathbf{y}_k$, $\mathbf{v}_k$, and $\mathbf{a}_k$.
- Goal: advance to $t_{k+1} = t_k + h$.
- By the Fundamental Theorem of Calculus:

$$\mathbf{v}_{k+1} = \mathbf{v}_k + \int_{t_k}^{t_{k+1}} \mathbf{a}(t)\, dt.$$

- Similarly, position update:

$$\begin{aligned}\mathbf{y}_{k+1} &= \mathbf{y}_k + \int_{t_k}^{t_{k+1}} \mathbf{v}(t)\, dt \\ &= \mathbf{y}_k + h\mathbf{v}_k + \int_{t_k}^{t_{k+1}} (t_{k+1} - t)\mathbf{a}(t)\, dt.\end{aligned}$$

Multivalue Methods

Approximation of Acceleration

- Fix $\tau \in [t_k, t_{k+1}]$ and expand $\mathbf{a}_k, \mathbf{a}_{k+1}$ via Taylor:

$$\mathbf{a}_k = \mathbf{a}(\tau) + \mathbf{a}'(\tau)(t_k - \tau) + O(h^2), \qquad \mathbf{a}_{k+1} = \mathbf{a}(\tau) + \mathbf{a}'(\tau)(t_{k+1} - \tau) + O(h^2).$$

- Combine (with $\gamma \in [0, 1]$ being a parameter):

$$\mathbf{a}(\tau) = (1 - \gamma)\mathbf{a}_k + \gamma \mathbf{a}_{k+1} + \mathbf{a}'(\tau)(\tau - h\gamma - t_k) + O(h^2).$$

- Integrate to obtain:

$$\mathbf{v}_{k+1} = \mathbf{v}_k + h[(1 - \gamma)\mathbf{a}_k + \gamma \mathbf{a}_{k+1}] + O(h^p),$$

where $p = 3$ if $\gamma = \frac{1}{2}$, otherwise $p = 2$.

Multivalue Methods

General Newmark Form

- Newmark advances position and velocity using endpoint accelerations:

$$\begin{cases} \mathbf{y}_{k+1} = \mathbf{y}_k + h\mathbf{v}_k + h^2\left[(\frac{1}{2}-\beta)\mathbf{a}_k + \beta\mathbf{a}_{k+1}\right], \\ \mathbf{v}_{k+1} = \mathbf{v}_k + h\left[(1-\gamma)\mathbf{a}_k + \gamma\mathbf{a}_{k+1}\right]. \end{cases}$$

- The dynamics close the system at both endpoints:

$$\mathbf{a}_k = \mathbf{F}(t_k, \mathbf{y}_k, \mathbf{v}_k), \qquad \boxed{\mathbf{a}_{k+1} = \mathbf{F}(t_{k+1}, \mathbf{y}_{k+1}, \mathbf{v}_{k+1})}.$$

- If $\beta > 0$, $\mathbf{a}_{k+1}$ generally makes the method implicit and requires a solve.
- The method is second-order accurate when $\gamma = \frac{1}{2}$; stability and numerical damping depend on (β, γ).

Multivalue Methods

Useful Newmark Parameter Choices

- $\beta = 0$, $\gamma = \frac{1}{2}$: **central difference**
  - explicit for acceleration depending only on position/time;
  - second-order but conditionally stable.
- $\beta = \frac{1}{4}$, $\gamma = \frac{1}{2}$: **average-acceleration Newmark**
  - implicit and second-order;
  - unconditionally stable for linear undamped structural dynamics.
- $\beta = \frac{1}{6}$, $\gamma = \frac{1}{2}$: **linear-acceleration Newmark**
  - implicit and second-order, but only conditionally stable.

Multivalue Methods

Staggered Grid and Leapfrog Scheme

- Use centered difference about $t_{k+1/2} = t_k + h/2$:

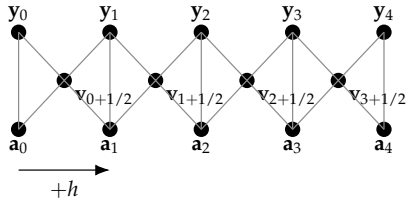
$$\mathbf{y}_{k+1} = \mathbf{y}_k + h\mathbf{v}_{k+1/2}.$$

- Velocity update:

$$\mathbf{v}_{k+3/2} = \mathbf{v}_{k+1/2} + h\mathbf{a}_{k+1}.$$

- Explicit leapfrog integrator:

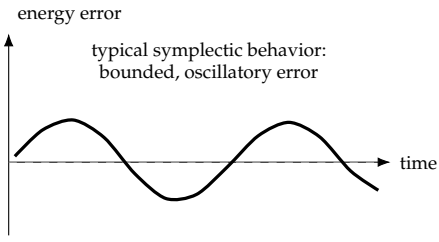
$$\begin{cases} \mathbf{y}_{k+1} = \mathbf{y}_k + h\mathbf{v}_{k+1/2}, \\ \mathbf{a}_{k+1} = \mathbf{F}[t_{k+1}, \mathbf{y}_{k+1}], \\ \mathbf{v}_{k+3/2} = \mathbf{v}_{k+1/2} + h\mathbf{a}_{k+1}. \end{cases}$$



Multivalue Methods

Why Leapfrog / Verlet Is Popular in Mechanics

- For standard conservative, velocity-independent forces, leapfrog is **time reversible** and **symplectic**.
- Symplectic does *not* mean exact energy conservation. Instead, for many Hamiltonian systems the energy error stays bounded and oscillatory over long times.
- Momentum is conserved when the discrete forces respect the corresponding translational/rotational symmetry.
- If acceleration depends on velocity, the simple explicit staggered update no longer applies directly and may become implicit.



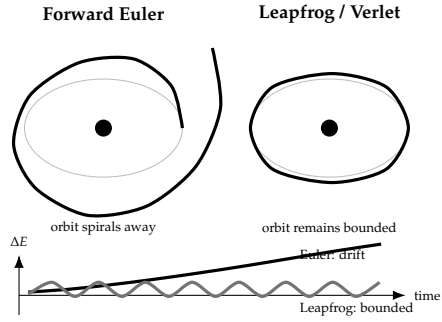
Multivalue Methods

Simulation Application: Orbital Dynamics

- A two-body orbit obeys

$$\boxed{\mathbf{x}'' = -\mu \frac{\mathbf{x}}{\|\mathbf{x}\|^3}}, \qquad E = \frac{1}{2}\|\mathbf{v}\|^2 - \frac{\mu}{\|\mathbf{x}\|}.$$

  The true Kepler problem conserves E.
- **Forward Euler:** local errors accumulate into secular energy drift, so the numerical orbit can spiral.
- **Leapfrog / Verlet:** symplectic; E is not exact, but its error is typically bounded and oscillatory.
- For long-horizon orbital dynamics, preserving geometry can matter more than minimizing one-step error alone.



Reference: Keplerian symplectic-integration energy-error examples; visualization redrawn for lecture use.

Time-Stepping Schemes

Comparison of ODE Integrators

Integrator	*Order*	*Type*	*Stability character*	*Good fit*
Forward Euler	1	Explicit	Conditional	Nonstiff; very cheap
Backward Euler	1	Implicit	A-/L-stable	Stiff / dissipative
Trapezoidal	2	Implicit	A-stable	Stiff; low numerical damping
Heun / RK2	2	Explicit	Conditional	Smooth nonstiff dynamics
RK4	4	Explicit	Conditional	Smooth nonstiff; high accuracy
Exponential Euler	1	Exp. action	Exact linear part	Semilinear / stiff systems
Newmark ($\beta = 1/4, \gamma = 1/2$)	2	Implicit	Unconditional[†]	Structural dynamics
Leapfrog / Verlet	2	Explicit	Conditional	Conservative mechanics

[†]For the standard linear undamped structural model.

Time-Stepping Schemes

Choosing an ODE Integrator

- **Smooth, nonstiff dynamics:** RK4 or an adaptive RK method.
- **Stiff / dissipative dynamics:** Backward Euler, another implicit method, or an exponential method when the linear part is exploitable.
- **Conservative mechanics:** leapfrog / Verlet or another symplectic integrator.
- **Second-order structural dynamics:** Newmark family.
- Collisions and discrete events need **event detection / root finding**; adaptive timesteps help localize them accurately.

