COL781: Computer Graphics

35. Stiff Systems & Constraints

Müller et al. 2020

Assignment 4 partially posted

Due date?

Forward Euler & instability

For the ODE $\dot{x}(t) = \phi(t, x(t))$, forward Euler:

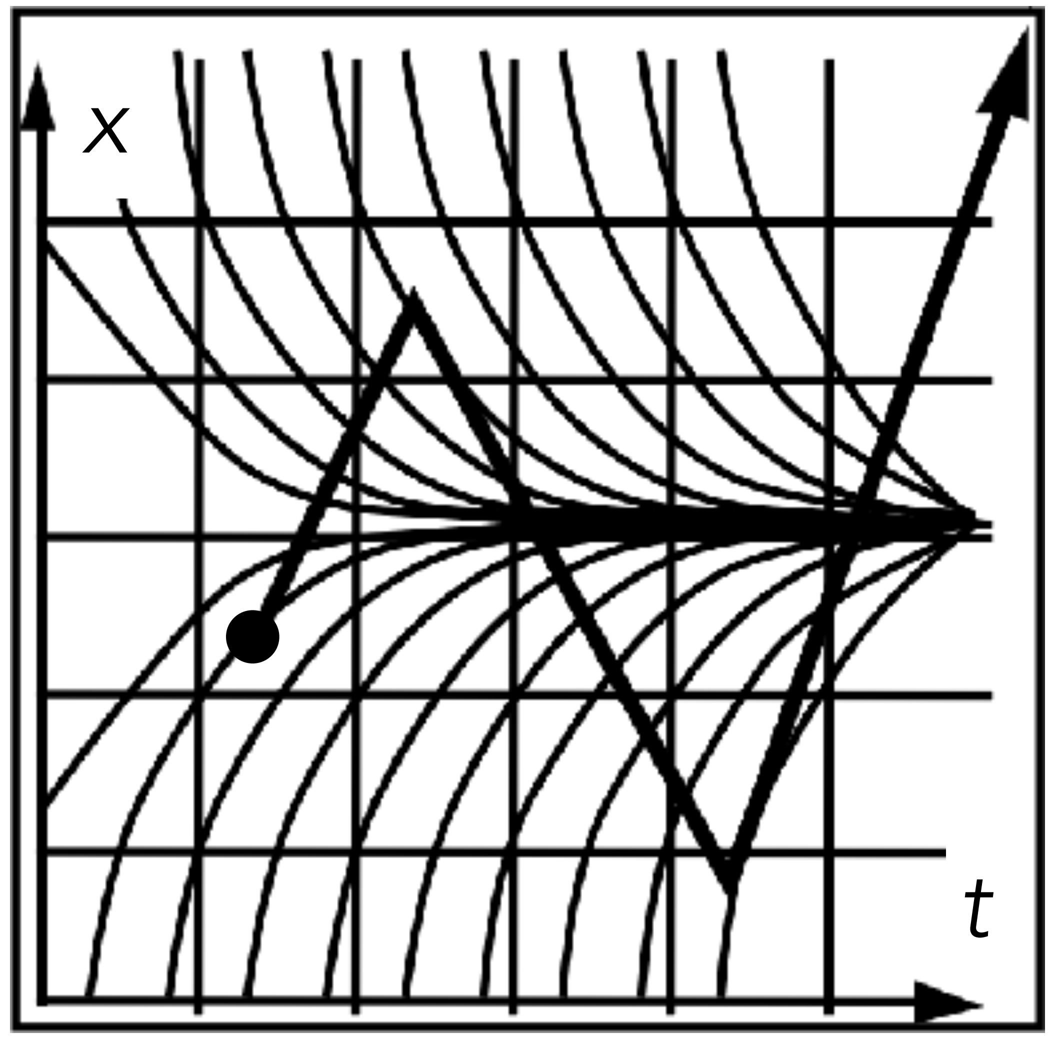
$$x_{n+1} = x_n + \phi(t_n, x_n)\,\Delta t$$

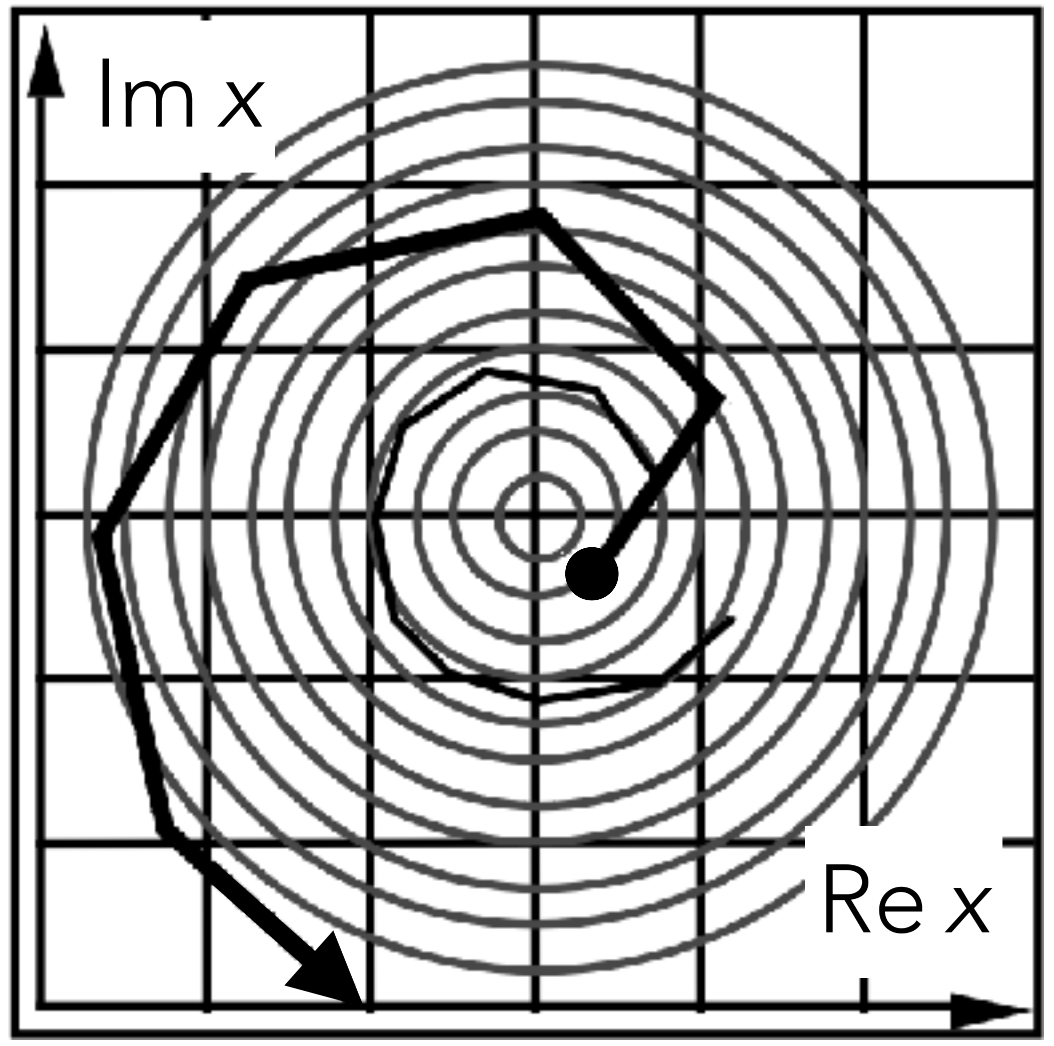
First-order accurate but not always stable.

For $\dot{x} = a\,x$,

- Exact solution is bounded if $\mathrm{Re}(a) \leq 0$
- FE solution is bounded if $|a\,\Delta t + 1| \leq 1$

If $|a|$ is (very) large, Δt needs to be (very) small!





Witkin & Baraff 2001

Example: spring pendulum with rest length ℓ_0, spring constant k_s

- Period of horizontal swing: $T_{\text{slow}} \approx O\left(\sqrt{l_0/g}\right)$
- Period of vertical vibration: $T_{\text{fast}} \approx O\left(\sqrt{m/k_s}\right)$

Take $k_s \to \infty$. Then $T_{\text{fast}} \to 0$, stable $\Delta t \to 0$!

We only care about dynamics on the scale of T_{slow},
but we're forced to take time steps on the scale of $T_{\text{fast}} \ll T_{\text{slow}}$.

In such cases, we say the problem is **stiff**. This happens a lot in graphics…

https://www.youtube.com/watch?v=2R9u-tjhRYA

Backward Euler

In forward Euler, we evaluate the derivative $\phi(t, x)$ at t_n. Let's try t_{n+1}:

$$x_{n+1} = x_n + \phi(t_{n+1}, x_{n+1})\,\Delta t$$

This is an implicit method: unknown x_{n+1} appears on both sides!

- "Look before you leap": Go to the point x_{n+1} where the derivative $\phi(t, x)$ matches the step you just took, $(x_{n+1} - x_n)/\Delta t$
- Can't just plug in values. Solve with e.g. Newton's method ⇒ more expensive!

Still only first-order accurate. So what's the benefit?

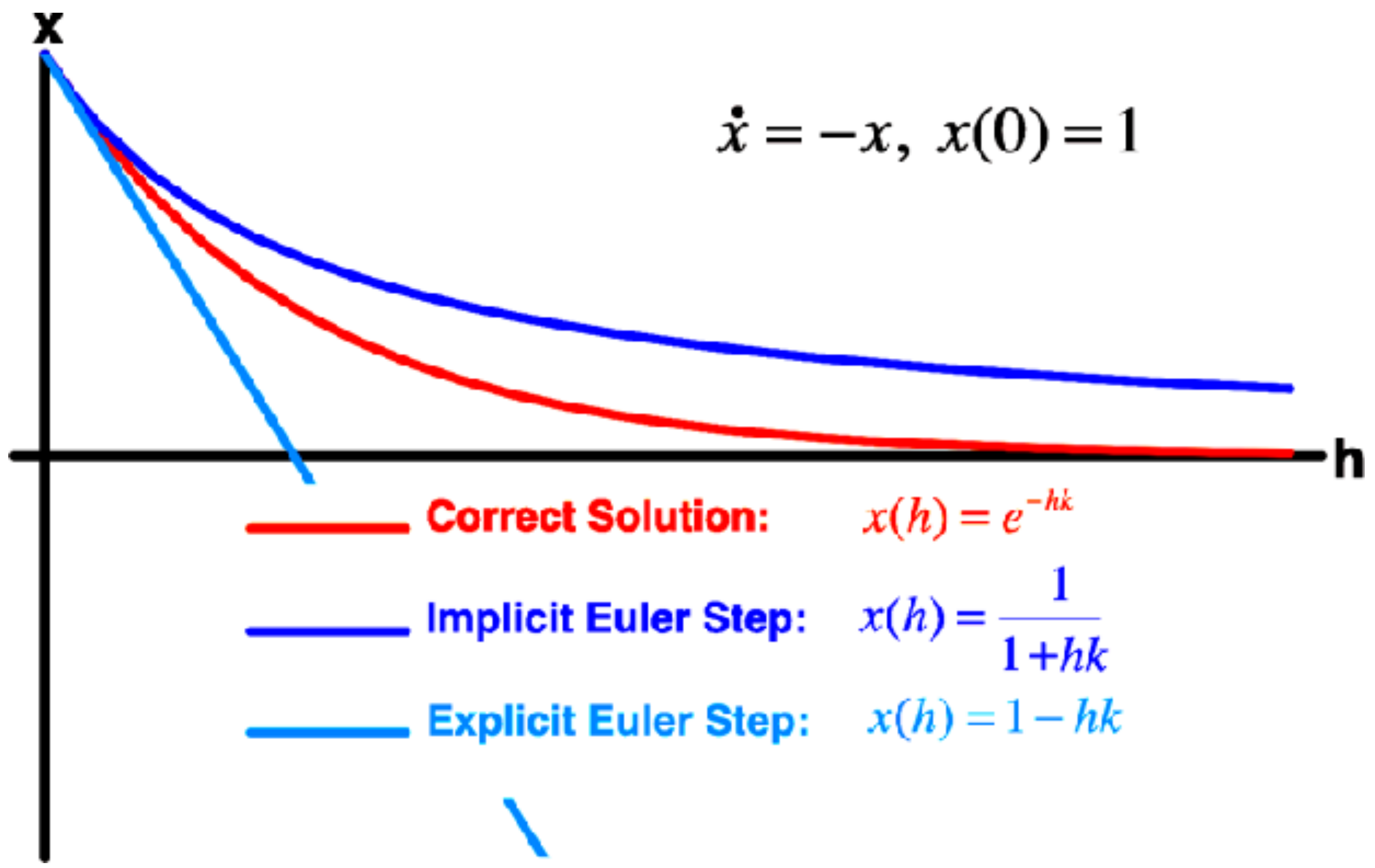
Consider $\dot{x} = a\,x$ again.

- Exact solution: $x(t) = \exp(a\,t)\,x(0)$
- BE solution: $x_{n+1} = x_n + a\,x_{n+1}\,\Delta t$
  $\Rightarrow x_{n+1} = (1 - a\,\Delta t)^{-1}\,x_n$

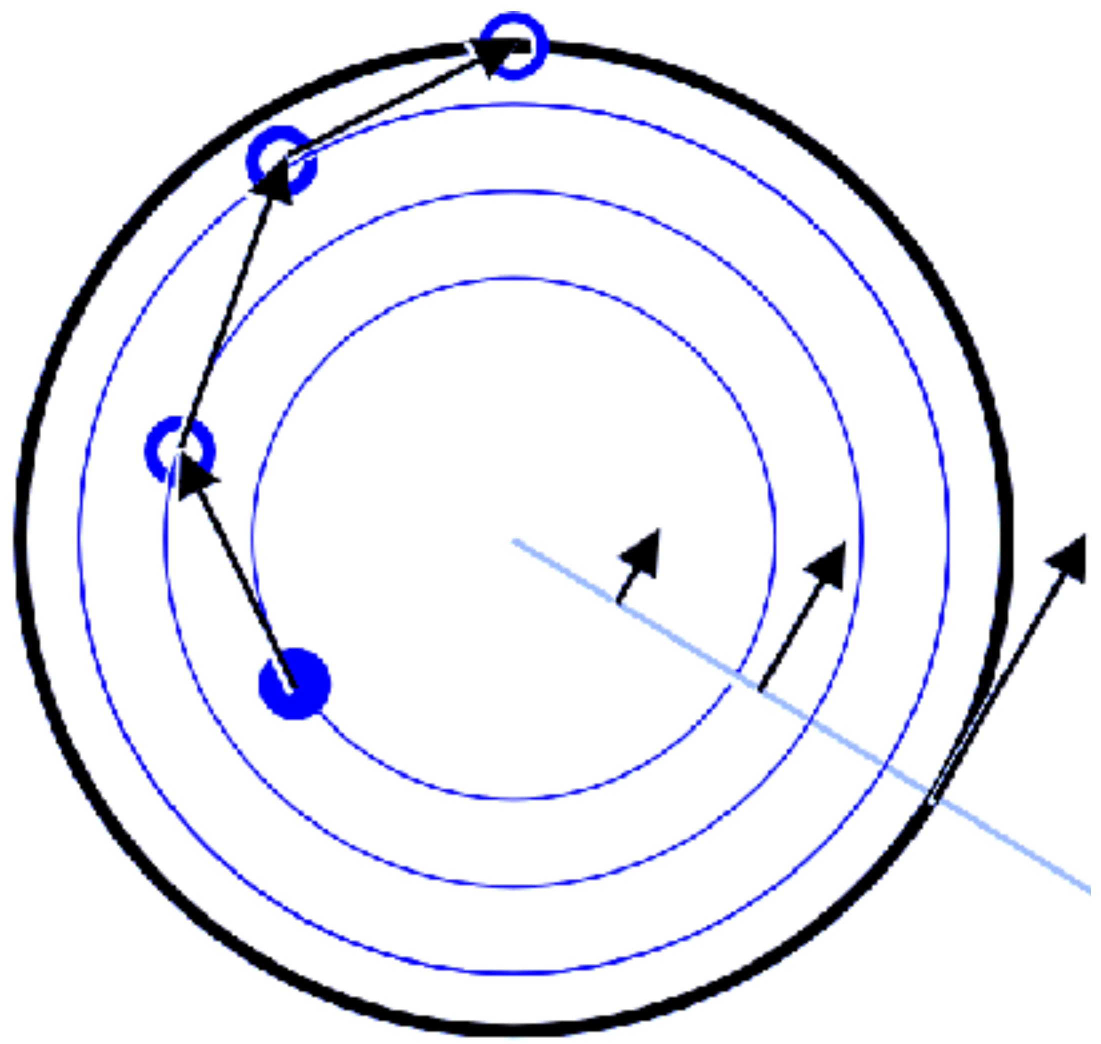
If $a < 0$, BE solution correctly decays for any Δt.

If a is imaginary, BE solution spirals inward: remains stable!

In fact BE is **unconditionally stable** (i.e. stable for any Δt) for all linear ODEs $\dot{\mathbf{x}} = \mathbf{A}\,\mathbf{x}$.



Witkin & Baraff 2001



Witkin & Baraff 2001

How do we apply all this to our 2nd-order ODE, $\ddot{\mathbf{q}} = \mathbf{M}^{-1}\,\mathbf{f}(\mathbf{q}, \dot{\mathbf{q}})$?

Reduce to 1st-order:

$$\begin{aligned} \dot{\mathbf{q}} &= \mathbf{v} \\ \dot{\mathbf{v}} &= \mathbf{M}^{-1}\,\mathbf{f}(\mathbf{q}, \mathbf{v}) \end{aligned}$$

Forward Euler:

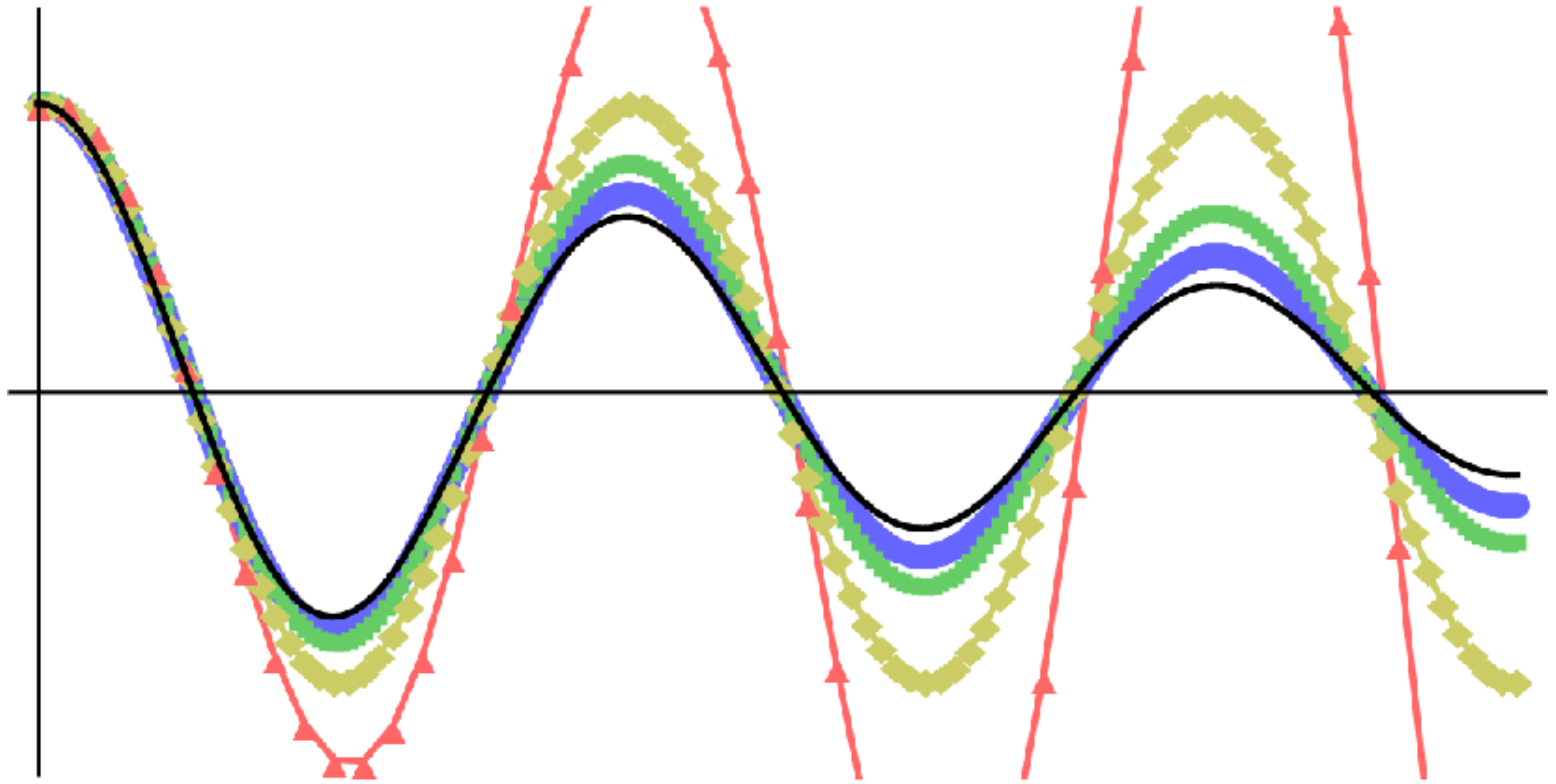
$$\begin{aligned} \mathbf{q}_{n+1} &= \mathbf{q}_n + \mathbf{v}_n\,\Delta t \\ \mathbf{v}_{n+1} &= \mathbf{v}_n + \mathbf{M}^{-1}\,\mathbf{f}(\mathbf{q}_n, \mathbf{v}_n)\,\Delta t \end{aligned}$$

Backward Euler:

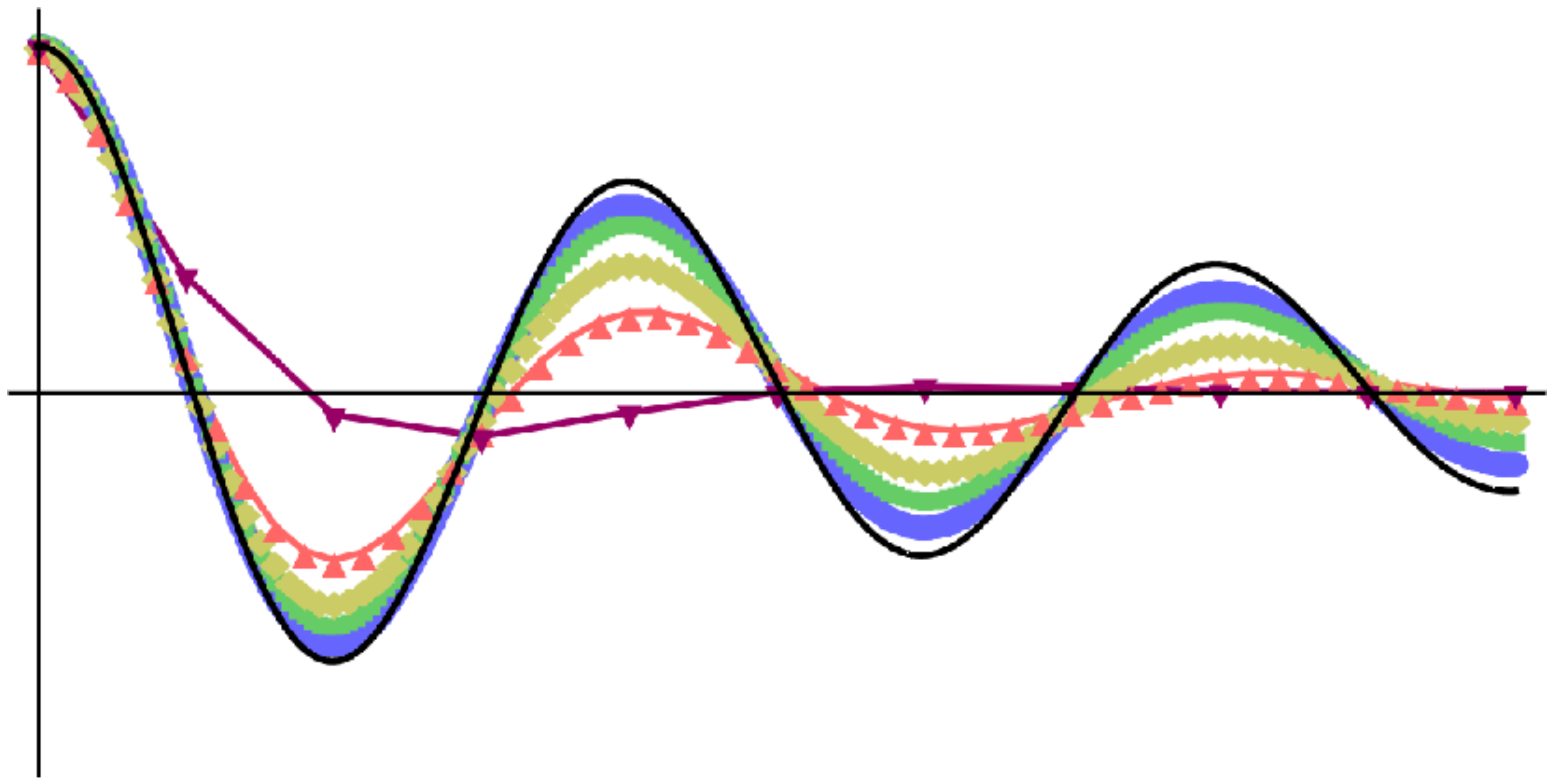
$$\begin{aligned} \mathbf{q}_{n+1} &= \mathbf{q}_n + \mathbf{v}_{n+1}\,\Delta t \\ \mathbf{v}_{n+1} &= \mathbf{v}_n + \mathbf{M}^{-1}\,\mathbf{f}(\mathbf{q}_{n+1}, \mathbf{v}_{n+1})\,\Delta t \end{aligned}$$

Example: Damped harmonic oscillator

$$\ddot{x} = -kx - c\dot{x}$$



Forward Euler solution



Backward Euler solution

Both are inaccurate, but backward Euler has a better failure mode: artificial dissipation

OK, backward Euler gives us a system of equations in the unknown next state $(\mathbf{q}_{n+1}, \mathbf{v}_{n+1})$

$$\mathbf{q}_{n+1} = \mathbf{q}_n + \mathbf{v}_{n+1}\,\Delta t$$
$$\mathbf{v}_{n+1} = \mathbf{v}_n + \mathbf{M}^{-1}\,\mathbf{f}(\mathbf{q}_{n+1}, \mathbf{v}_{n+1})\,\Delta t$$

How do we actually solve this?

Newton’s method

Newton's method

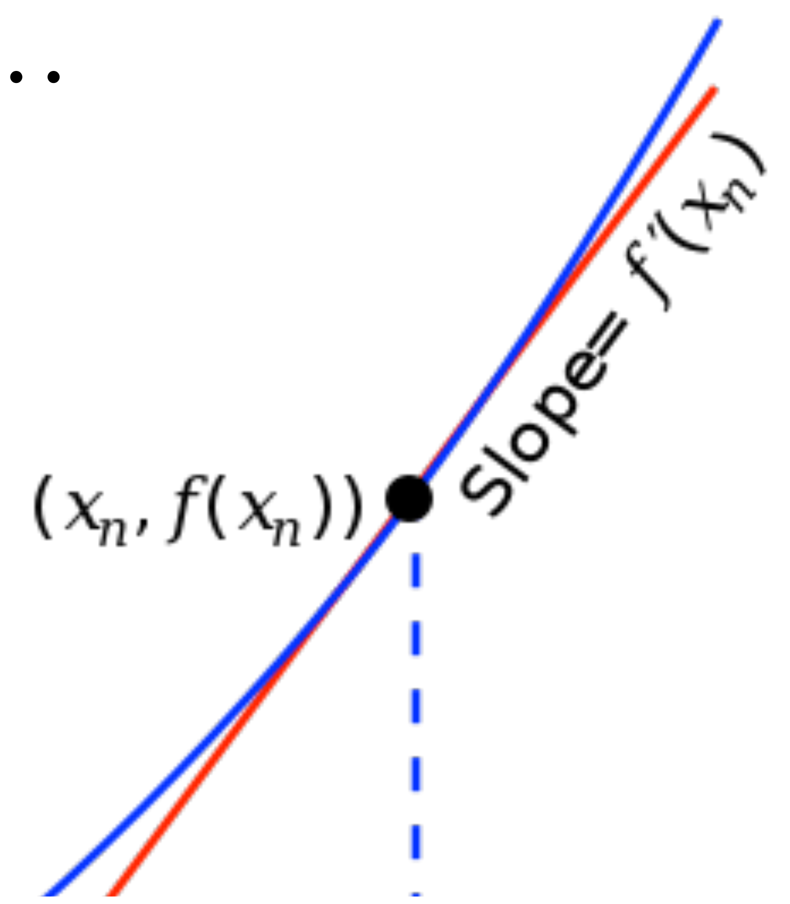
An instance of a very general problem-solving strategy.

Say you have a problem you don't know how to solve exactly:

1. Approximate the problem.
2. Solve the approximation exactly.
3. **Optional:** Use the solution to improve the approximation, and repeat…

In Newton's method, approximation = 1st-order Taylor series

$f(x+\Delta x) \approx f(x) + f'(x)\,\Delta x$



Say you have a nonlinear system of equations you don't know how to solve exactly:
Find x such that $f(x) = 0$.

Start with a **guess**: $\tilde{x}$.

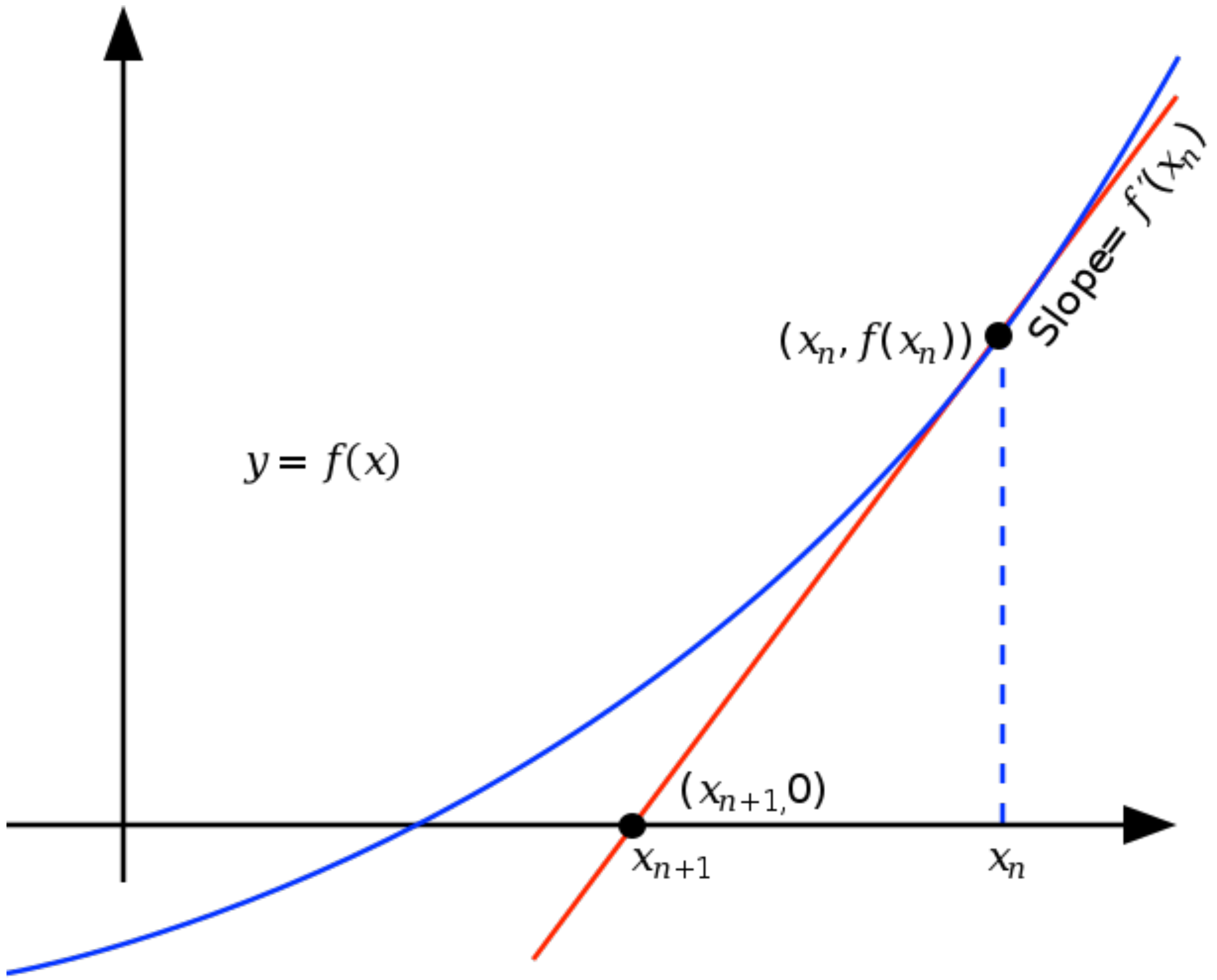
1. **Approximate** the problem near the guess:

$$0 = f(\tilde{x} + \Delta x) \approx f(\tilde{x}) + f'(\tilde{x})\, \Delta x$$

2. Solve the approximation **exactly**:

$$\Delta x = -(f'(\tilde{x}))^{-1} f(\tilde{x})$$

3. **Improve** the guess and repeat: $\tilde{x} \leftarrow \tilde{x} + \Delta x$



$$\mathbf{q}_{n+1} = \mathbf{q}_n + \mathbf{v}_{n+1}\,\Delta t$$
$$\mathbf{v}_{n+1} = \mathbf{v}_n + \mathbf{M}^{-1}\,\mathbf{f}(\mathbf{q}_{n+1}, \mathbf{v}_{n+1})\,\Delta t$$

Pick a guess $(\tilde{\mathbf{q}}, \tilde{\mathbf{v}})$. A natural choice is to start with $\tilde{\mathbf{q}} = \mathbf{q}_n$, $\tilde{\mathbf{v}} = \mathbf{v}_n$.

1. Approximate the problem:

$$(\tilde{\mathbf{q}}+\Delta\mathbf{q}) = \mathbf{q}_n + (\tilde{\mathbf{v}}+\Delta\mathbf{v})\,\Delta t$$

$$(\tilde{\mathbf{v}}+\Delta\mathbf{v}) = \mathbf{v}_n + \mathbf{M}^{-1}\,\mathbf{f}(\tilde{\mathbf{q}}+\Delta\mathbf{q}, \tilde{\mathbf{v}}+\Delta\mathbf{v})\,\Delta t$$

where $\mathbf{f}(\tilde{\mathbf{q}}+\Delta\mathbf{q}, \tilde{\mathbf{v}}+\Delta\mathbf{v}) \approx \mathbf{f}(\tilde{\mathbf{q}}, \tilde{\mathbf{v}}) + \frac{\partial \mathbf{f}}{\partial \mathbf{q}}(\tilde{\mathbf{q}}, \tilde{\mathbf{v}})\,\Delta\mathbf{q} + \frac{\partial \mathbf{f}}{\partial \mathbf{v}}(\tilde{\mathbf{q}}, \tilde{\mathbf{v}})\,\Delta\mathbf{v}$

2. Now the system is linear in $(\Delta\mathbf{q}, \Delta\mathbf{v})$. Plug into any linear solver. (Can simplify a bit first…)

Note: To carry this out, we must able to evaluate the **force Jacobians** $\frac{\partial \mathbf{f}}{\partial \mathbf{q}}$ and $\frac{\partial \mathbf{f}}{\partial \mathbf{v}}$.

What about the thing we did before?

$$\mathbf{q}_{n+1} = \mathbf{q}_n + \mathbf{v}_{n+1}\,\Delta t$$
$$\mathbf{v}_{n+1} = \mathbf{v}_n + \mathbf{M}^{-1}\,\mathbf{f}(\mathbf{q}_n, \mathbf{v}_n)\,\Delta t$$

This is very close to something called **symplectic Euler**:

$$\mathbf{q}_{n+1} = \mathbf{q}_n + \mathbf{v}_{n+1}\,\Delta t$$
$$\mathbf{v}_{n+1} = \mathbf{v}_n + \mathbf{M}^{-1}\,\mathbf{f}(\mathbf{q}_n, \mathbf{v}_{n+1})\,\Delta t$$

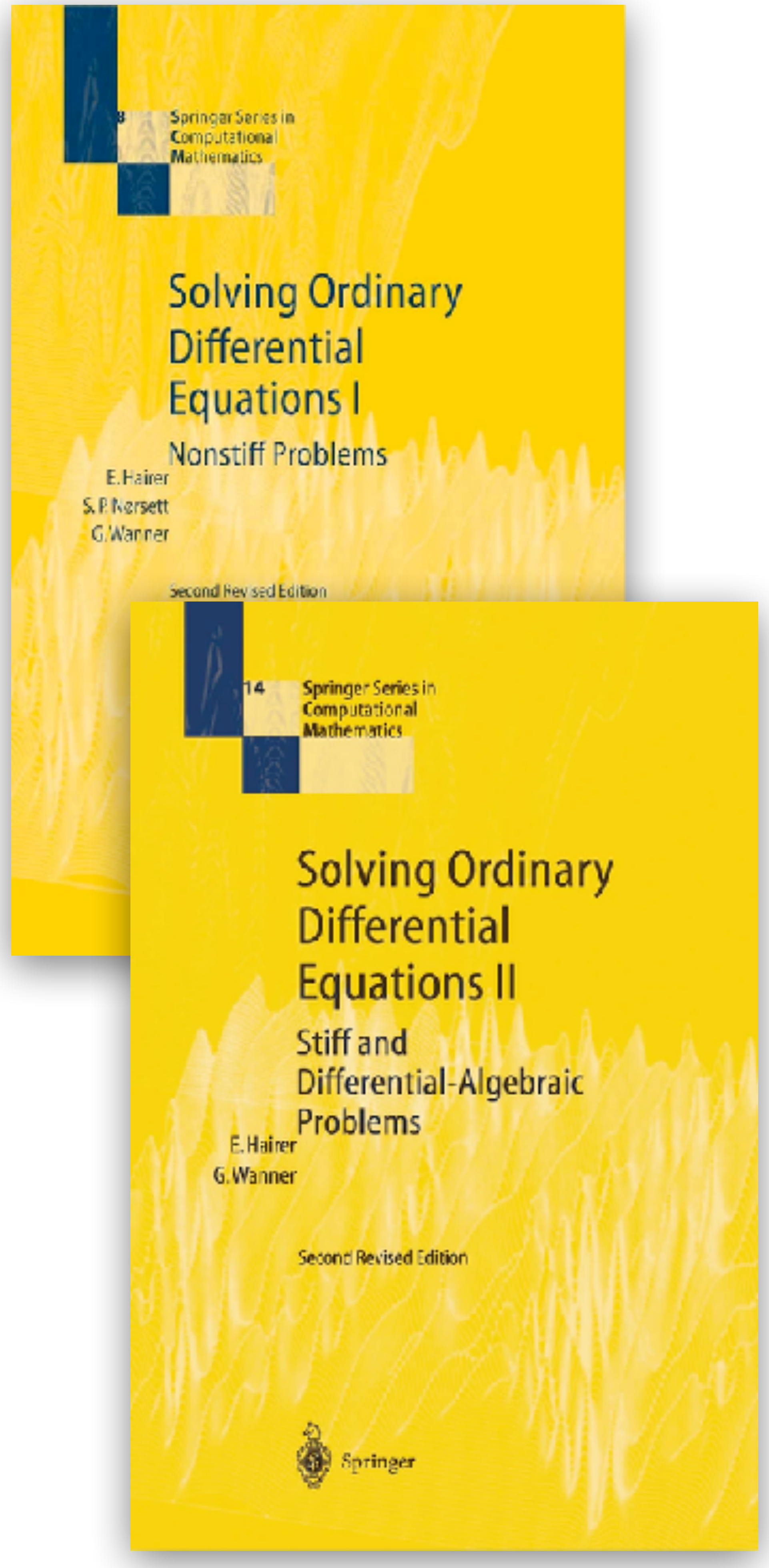
- Equivalent to previous scheme if **f** is independent of **v** (no damping forces)
- Approximately conserves energy (no artificial dissipation)! But only if it's stable
- Still only conditionally stable

Time integration summary

A big topic! Lots of other schemes: trapezoid, Newmark, RK4, BDF2, …

General advice for graphics:

1. Start with symplectic Euler or its variant (easy to implement)
2. If simulation is unstable:
    - Reduce Δt
    - Or: Switch to an implicit method e.g. backward Euler
    - Or… Reformulate the problem!



Constraints

Another general problem-solving strategy:
If a parameter being very large is causing problems, make it infinity instead.

What happens to the spring when $k_s \to \infty$?

$$\mathbf{f}_{ij} = -k_s\, \varepsilon\, \hat{\mathbf{x}}_{ij}$$

No external force is enough to stretch the spring! $\varepsilon = \|\mathbf{x}_{ij}\|/\ell_0 - 1 = 0$. The spring just becomes a distance constraint.

$$\|\mathbf{x}_{ij}\| = \ell_0$$

$$\mathbf{f}_{ij} = \lambda\, \hat{\mathbf{x}}_{ij}$$

Original equations of motion:

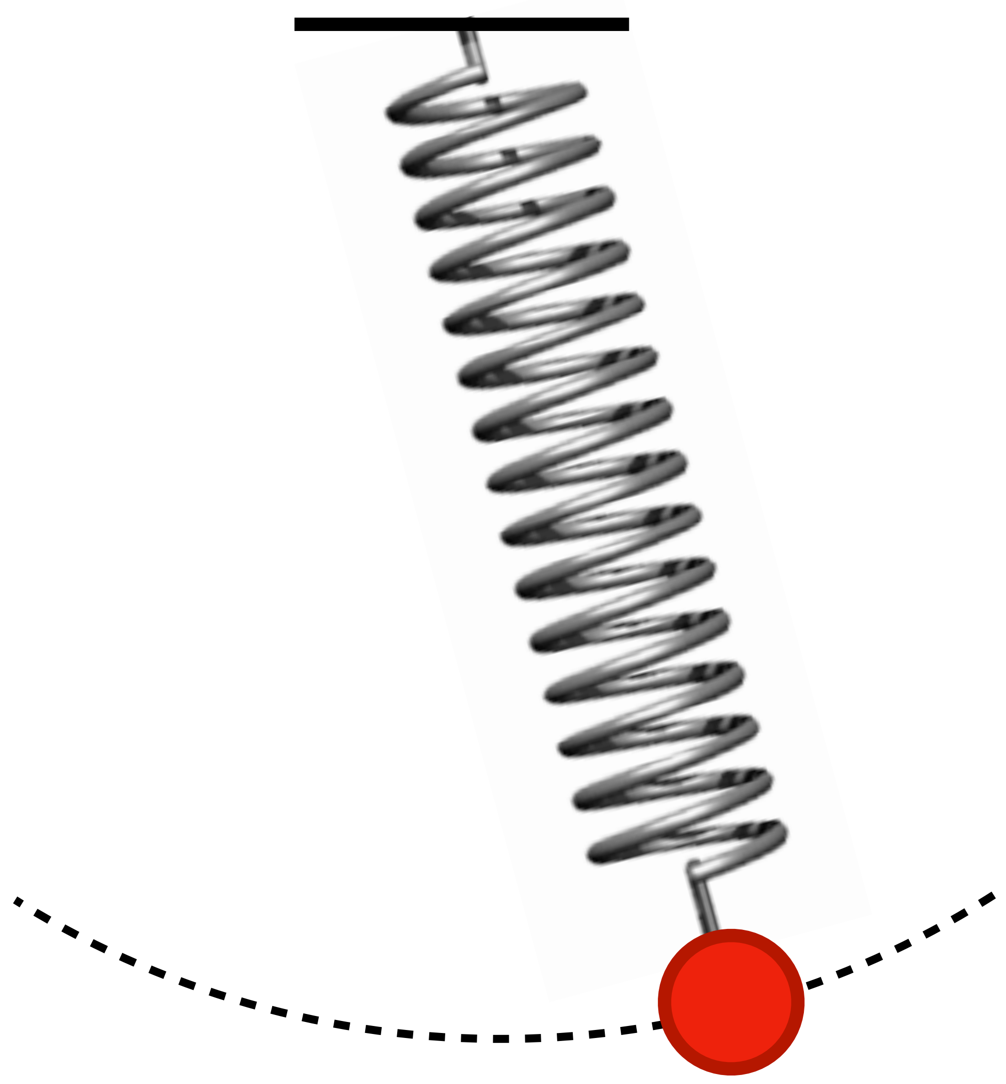
$$\ddot{\mathbf{x}} = \mathbf{g} - m^{-1} k_s \, \varepsilon(\mathbf{x}) \, \hat{\mathbf{x}}$$

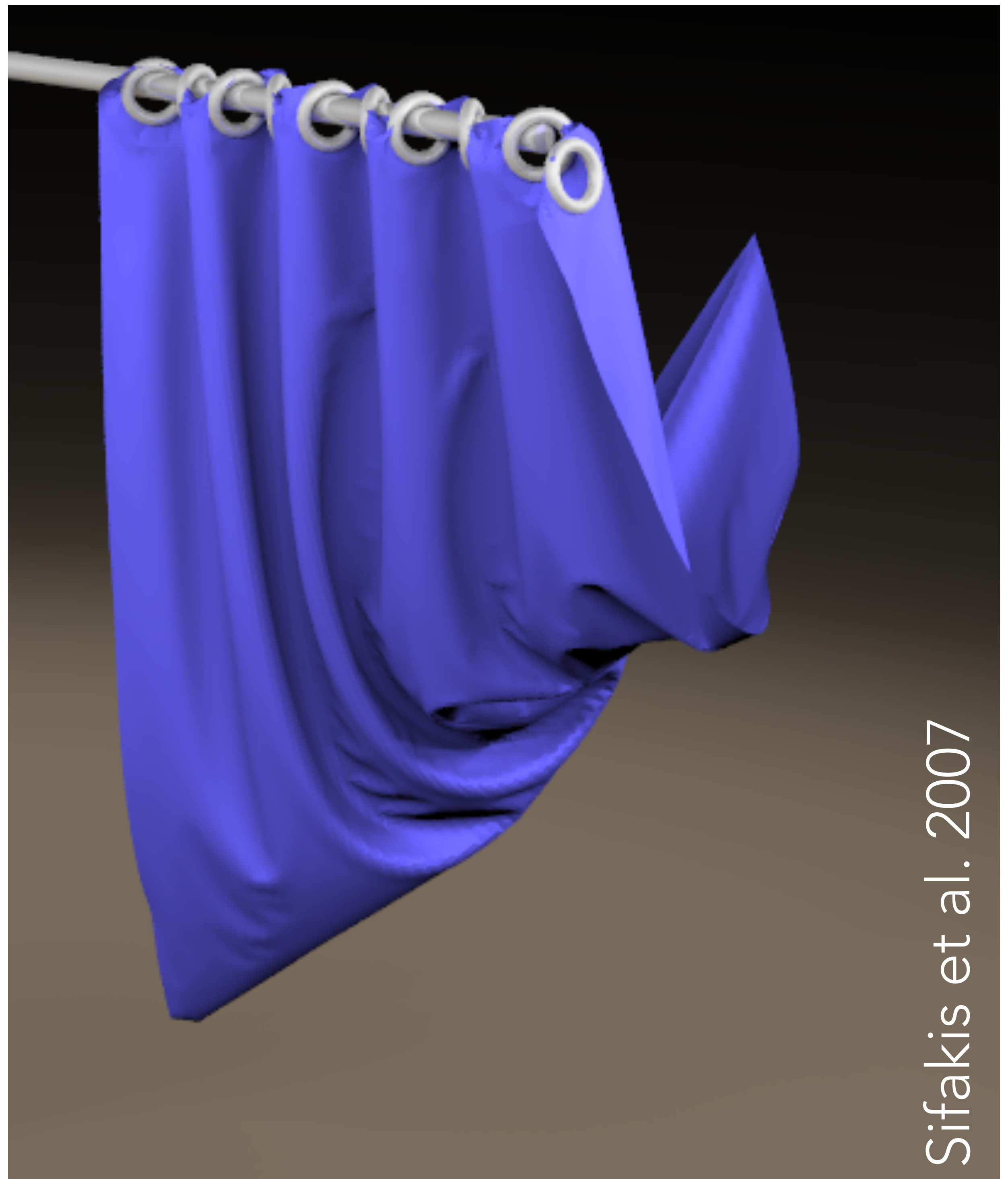
Constrained equations of motion:

$$\ddot{\mathbf{x}} = \mathbf{g} + m^{-1} \lambda \, \hat{\mathbf{x}}$$
$$\|\mathbf{x}_{ij}\| = \ell_0$$

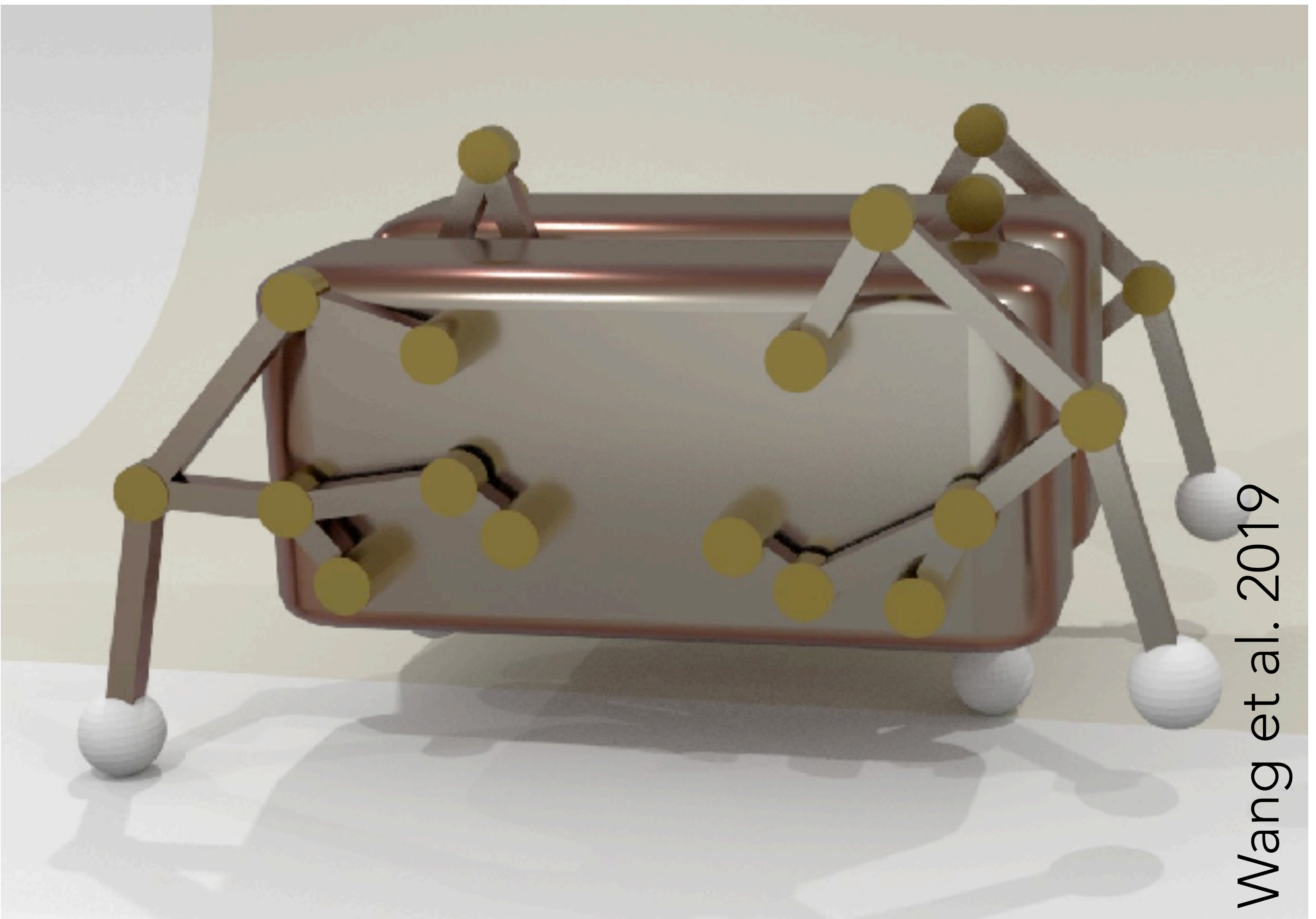
- One new unknown: constraint force magnitude λ.
- One new equation: constraint $\|\mathbf{x}_{ij}\| = \ell_0$.

λ is such that constraint remains satisfied over time…





Sliding on a fixed
line / curve / surface



Joints between
rigid parts

Inextensible cloth

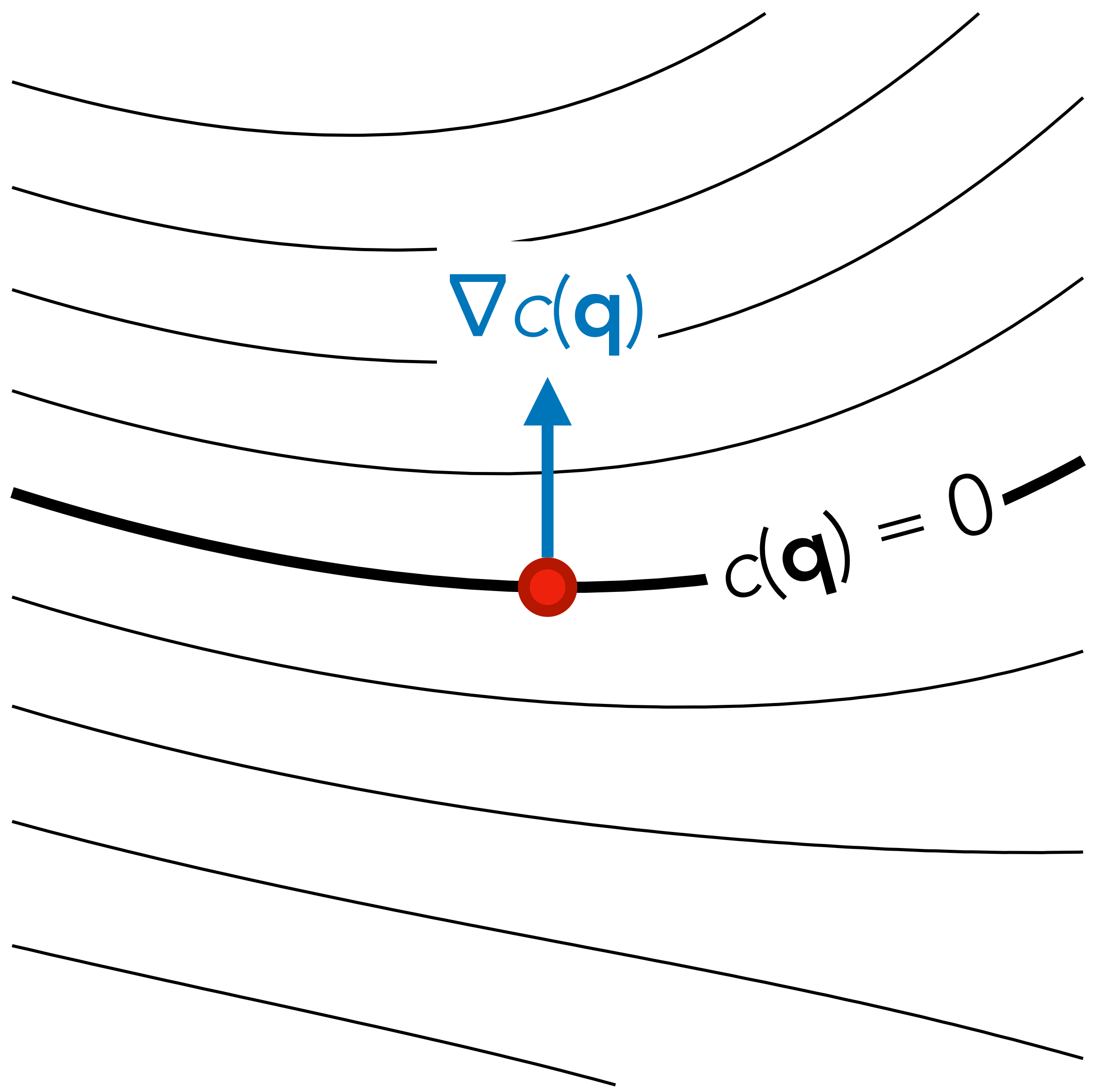
In general, we may have lots of constraints on the system, each of the form

$$c_j(\mathbf{q}) = 0$$

Constraint force:

$$\mathbf{f}_j = \lambda_j \nabla c_j(\mathbf{q})$$

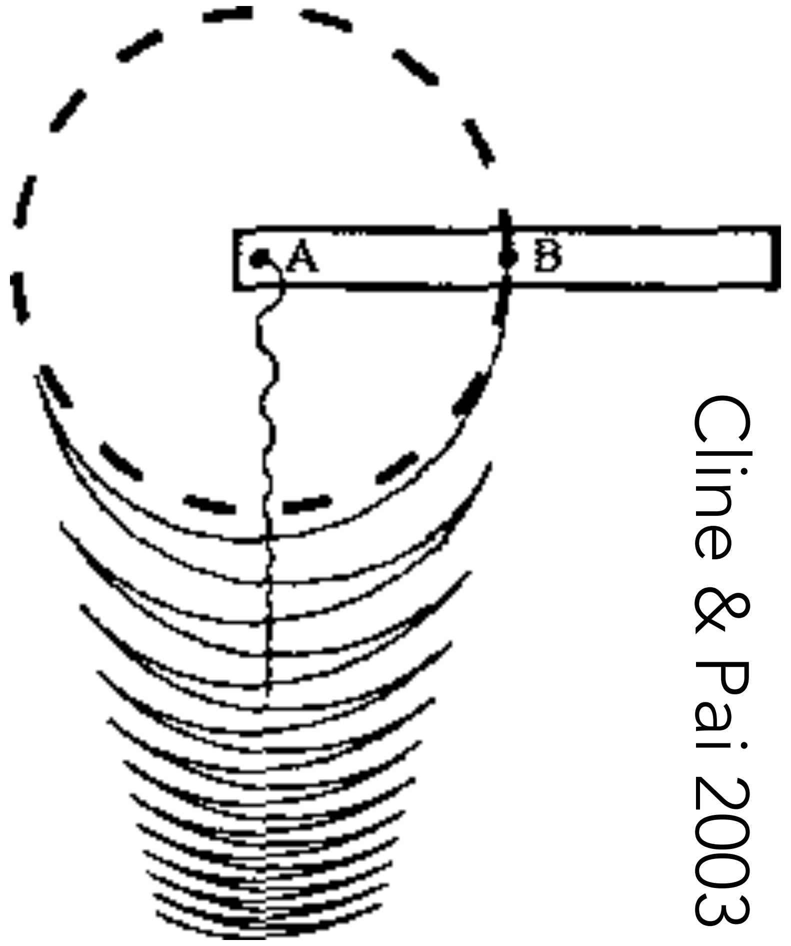
Force is orthogonal to constraint surface
⇒ only resists moving away from constraint, not along constraint



Exercise: verify that the inextensible spring constraint from before is of this form.

$$c_j(\mathbf{q}) = 0$$
$$\mathbf{f}_j = \lambda_j \nabla c_j(\mathbf{q})$$

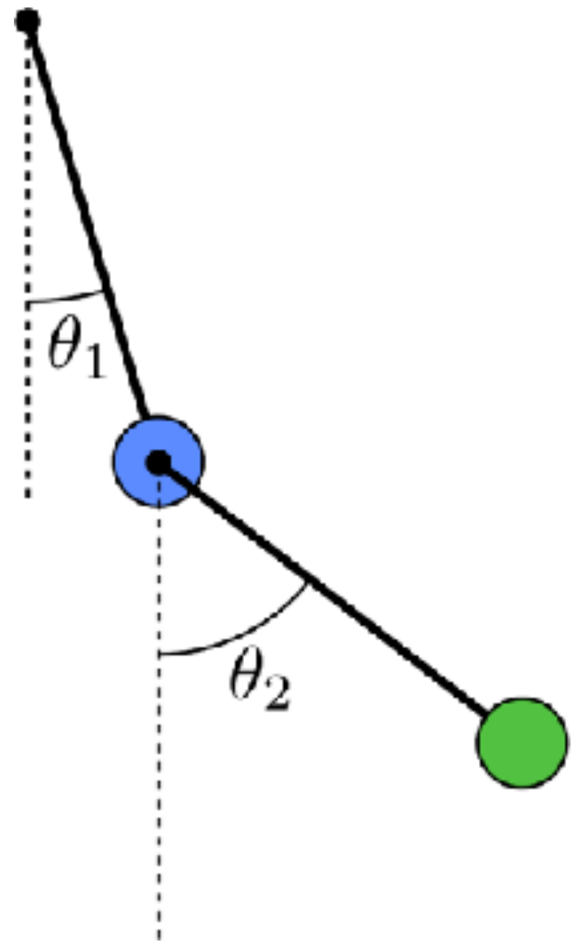
$$\ddot{\mathbf{q}} = \mathbf{M}^{-1}\left(\mathbf{f}(\mathbf{q}, \dot{\mathbf{q}}) + \textstyle\sum \mathbf{f}_j\right)$$



Cline & Pai 2003

How to actually do time stepping of such a system?

- Try to estimate instantaneous λ_j at each t_n $\Rightarrow$ drift
- Replace with **penalty force**: $\lambda_j = -k\, c_j(\mathbf{q})$ $\Rightarrow$ soft constraints
- Choose parameterization that automatically satisfies constraints
  $\Rightarrow$ **reduced coordinates**
- Treat constraint forces implicitly: solve for all λ_j's so that all $c_j(\mathbf{q}_{n+1}) = 0$



Keenan Crane

$$\ddot{\mathbf{q}} = \mathbf{M}^{-1} \left(\mathbf{f}(\mathbf{q}, \dot{\mathbf{q}}) + \sum \lambda_j \nabla c_j(\mathbf{q}) \right)$$

$$c_j(\mathbf{q}) = 0$$

Suppose we treat the external forces explicitly and the constraint forces implicitly.

We can also eliminate $\mathbf{v}_{n+1}$:

$$\mathbf{q}_{n+1} = \mathbf{q}_{\mathrm{pred}} + \sum \mathbf{M}^{-1} \lambda_j \nabla c_j(\mathbf{q}_{n+1}) \Delta t^2$$
$$c_j(\mathbf{q}_{n+1}) = 0$$

where $\mathbf{q}_{\mathrm{pred}} = \mathbf{q}_n + \mathbf{v}_n \Delta t + \mathbf{M}^{-1} \mathbf{f}(\mathbf{q}_n, \mathbf{v}_n) \Delta t^2$.

Solve for $\mathbf{q}_{n+1}$ and $\lambda_1, \lambda_2, \ldots$ simultaneously using Newton's method