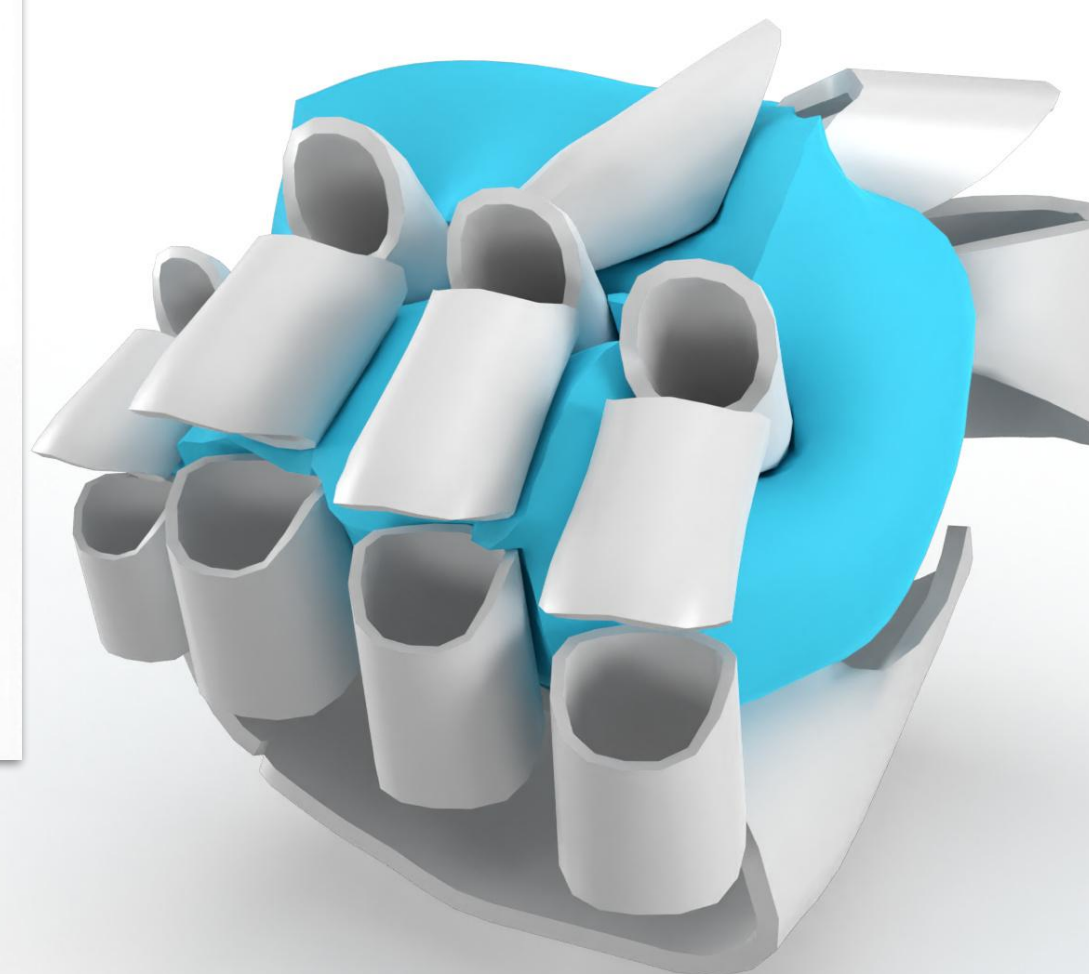
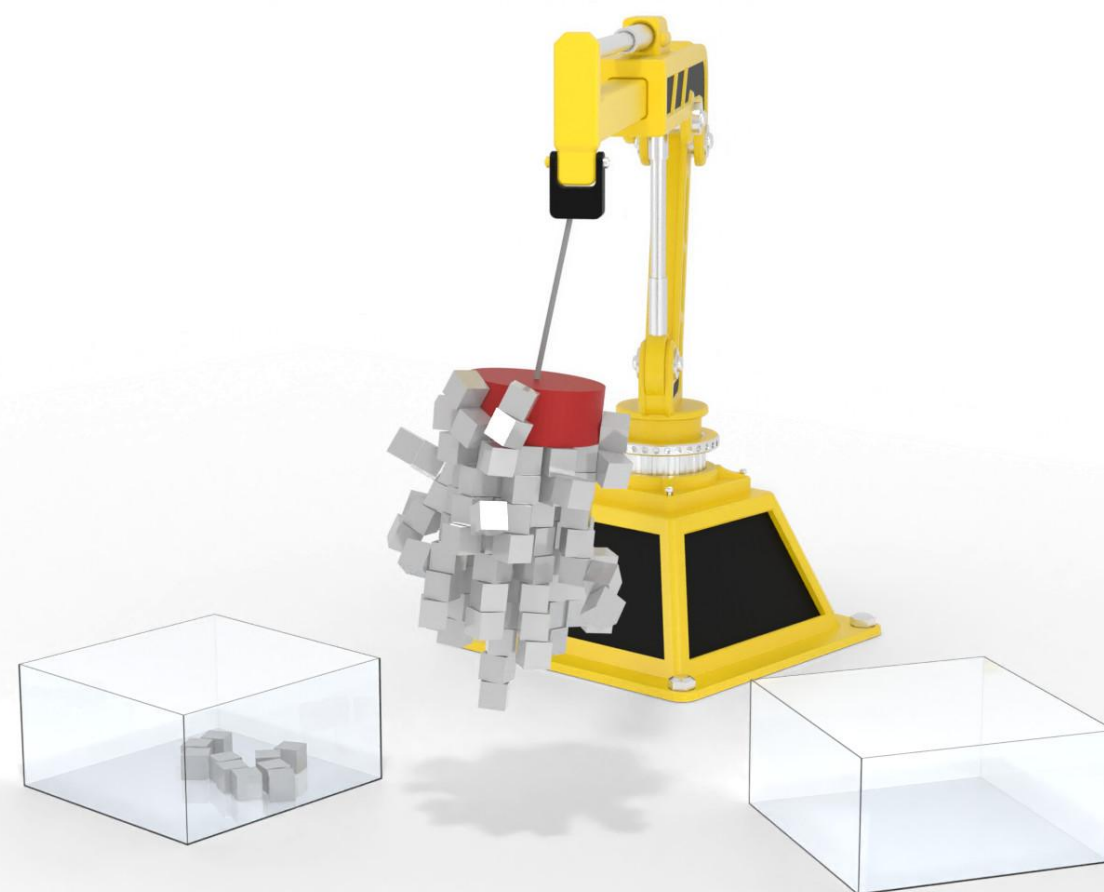


Part I: Energy-Based Simulation Foundations

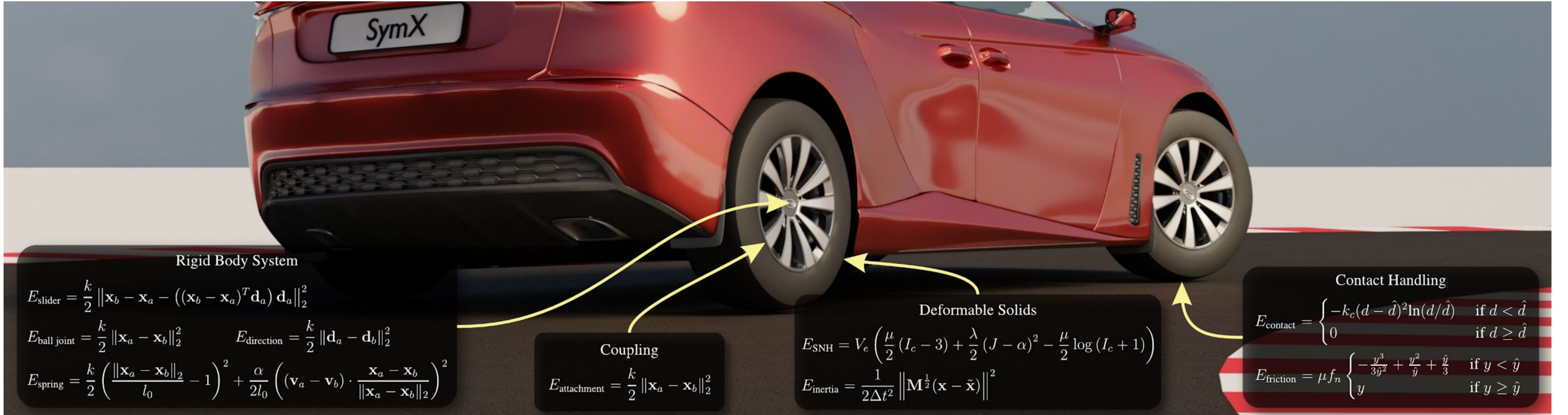


Motivation

Why Energy-Based?

1. Unified formulation in terms of scalar energy potentials
 - Strongly coupled, fully implicit
2. Optimization > Root finding
 - Ensures progress and validity

1. Unified Scalar Formulation



SymX [Fernández-Fernández et al. 2025]

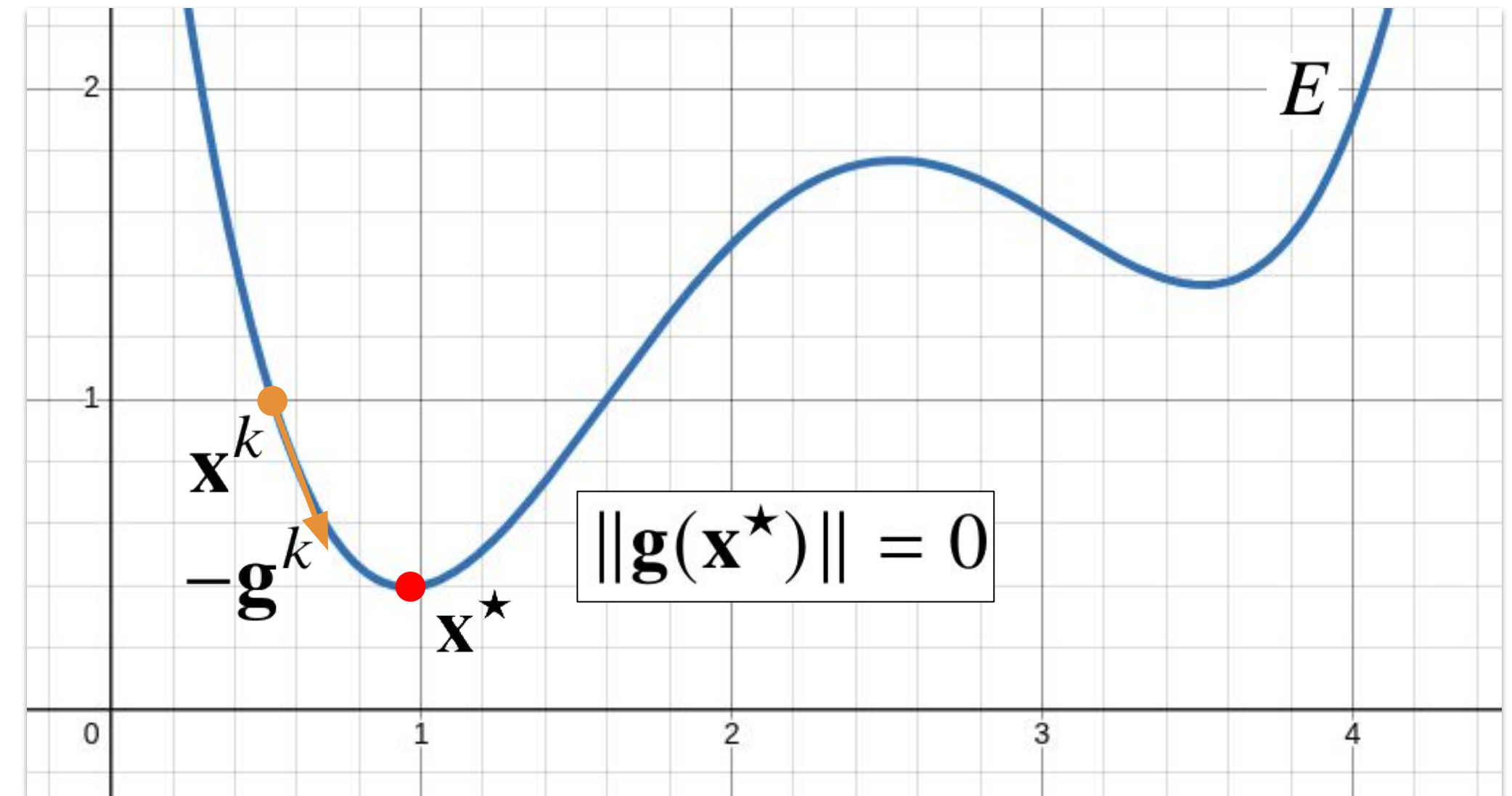
$$E_{\text{Total}} = E_{\text{Inertia}} + E_{\text{slider}} + E_{\text{ball}} + E_{\text{spring}} + E_{\text{dir}} + E_{\text{attach}} + E_{\text{SNH}} + E_{\text{Contact}} + E_{\text{Friction}}$$

2. Robust Optimization

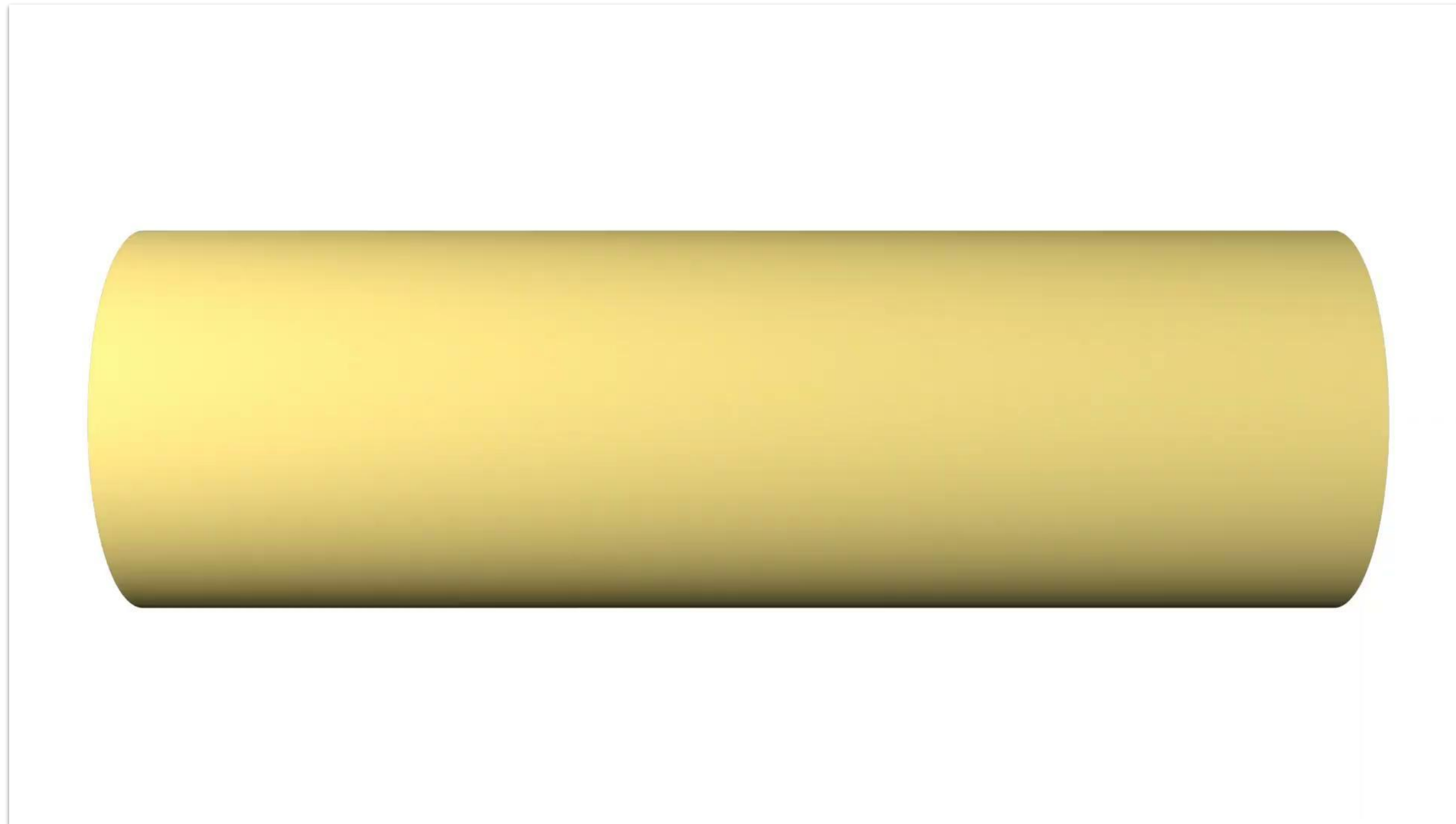
- Balance of forces at $\text{grad}(\mathbf{E}) = \mathbf{0}$
- E is sufficiently continuous and differentiable
- “Slide down to the solution”

Numerical infrastructure:

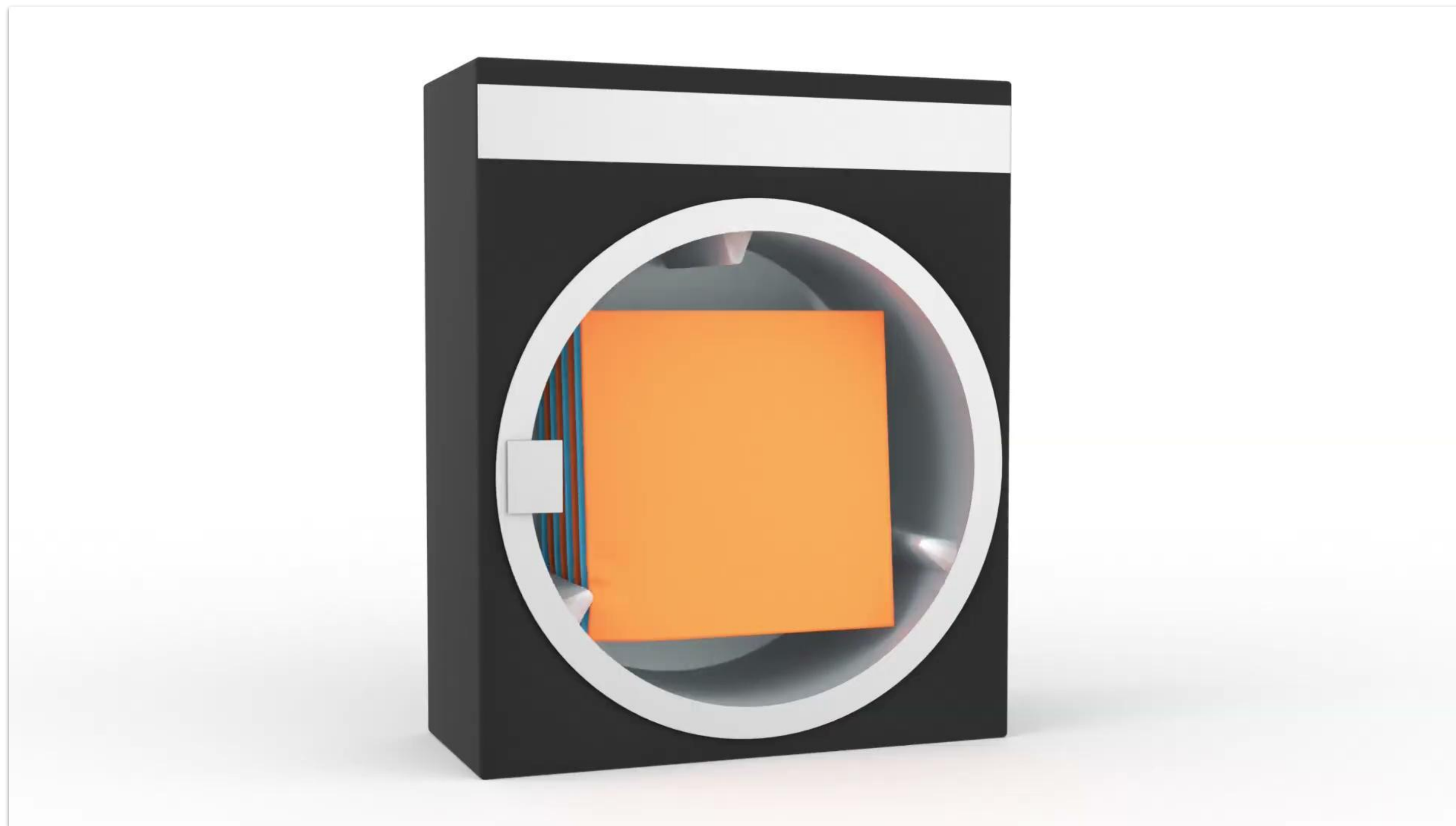
- Ensure descent directions
- Ensure admissibility
- Ensure sufficient descent



Examples



Examples

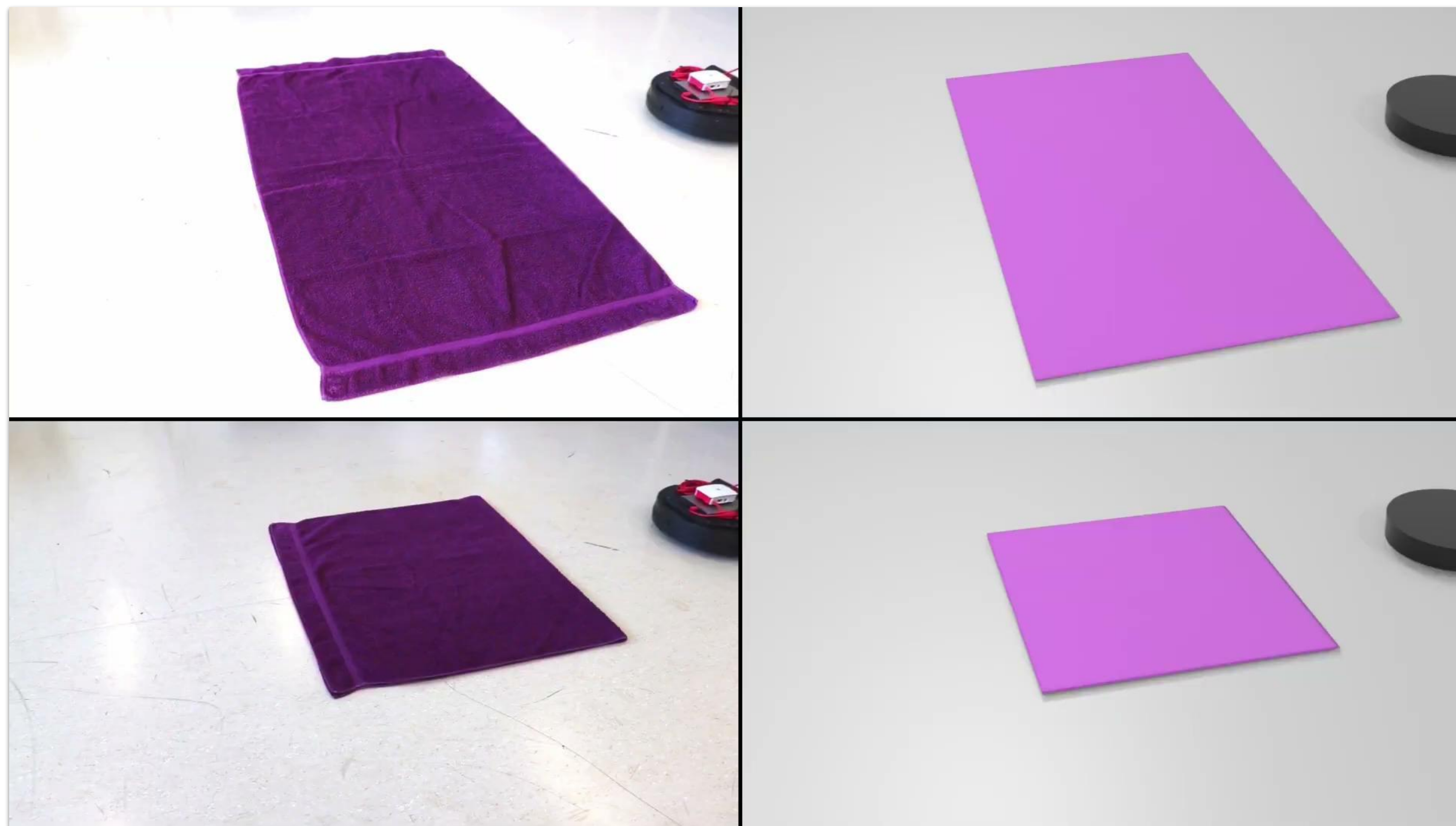


Examples




Strongly Coupled Simulation of Magnetic Rigid Bodies [Westhofen et al. 2024]

Examples



STARK [Fernández-Fernández et al. 2024]

Long History of Optimization in Simulation


 ELSEVIER

**Computer methods
in applied
mechanics and
engineering**

Comput. Methods Appl. Mech. Engrg. 172 (1999) 203–240

Error estimation and adaptive meshing in strongly nonlinear dynamic problems

R. Radovitzky, M. Ortiz*

Graduate Aeronautical Laboratories, California Institute of Technology, Pasadena, CA 91124

Eurographics/ACM SIGGRAPH Symposium on Computer Animation (2006)
 M.-P. Cani, J. O'Brien (Editors)

Geometric, Variational Integrators for Computer Animation

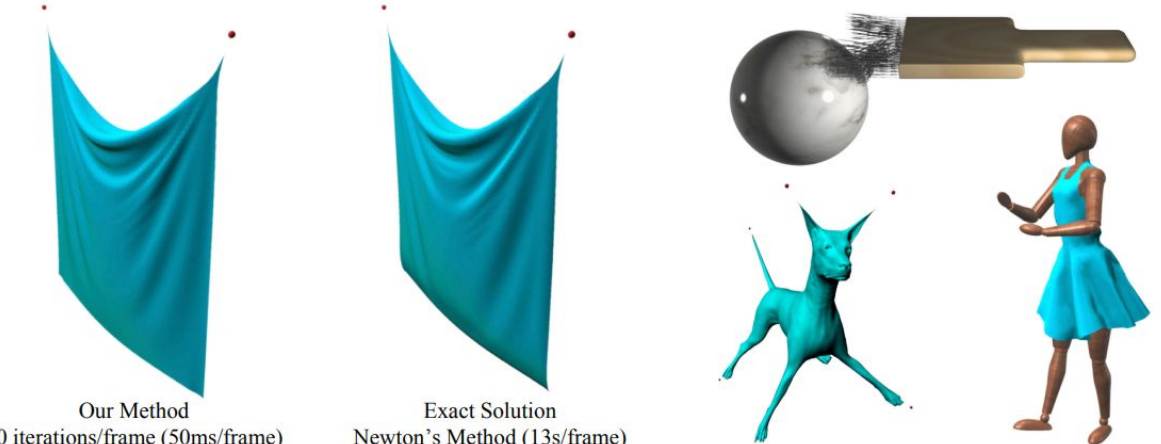
L. Kharevych Weiwei Y. Tong E. Kanso[‡] J. E. Marsden P. Schröder M. Desbrun

Caltech - [‡]USC

Abstract
 We present a general-purpose numerical scheme for time integration of Lagrangian mechanics. This is an important computational tool at the core of most physics-based animation techniques. A particular time integrator is highly desirable for computer animation: it numerically conserves energy and momentum, and is stable for large time steps.

Fast Simulation of Mass-Spring Systems

Tiantian Liu Adam W. Bargteil James F. O'Brien Ladislav Kavan*
 University of Pennsylvania University of Utah University of California, Berkeley University of Pennsylvania



Example-Based Elastic Materials

Sebastian Martin¹ Bernhard Thomaszewski^{1,2} Eitan Grinspun³ Markus Gross^{1,2}
¹ETH Zurich ²Disney Research Zurich ³Columbia University

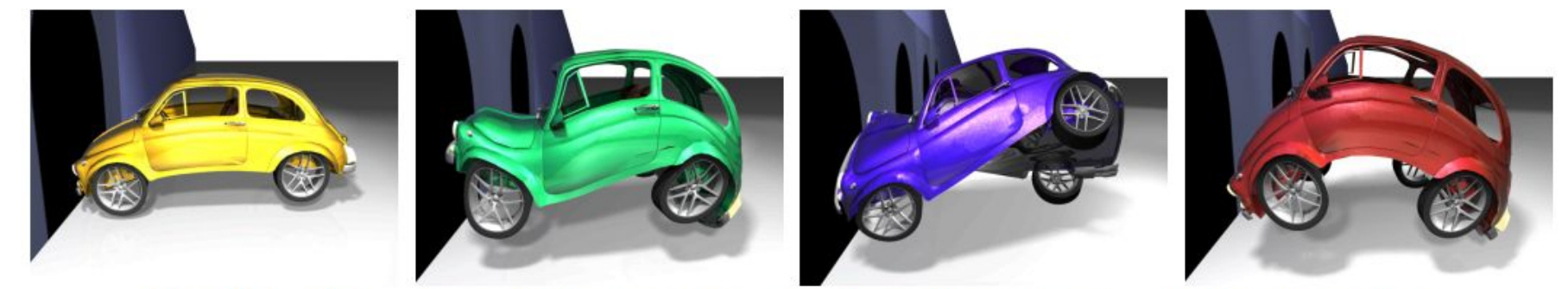


Figure 1: Example-based materials allow the simulation of flexible structures with art-directable deformation behavior.

TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS


Optimization Integrator for Large Time Steps

Theodore F. Gast, Craig Schroeder, Alexey Stomakhin, Chenfanfu Jiang and Joseph M. Teran

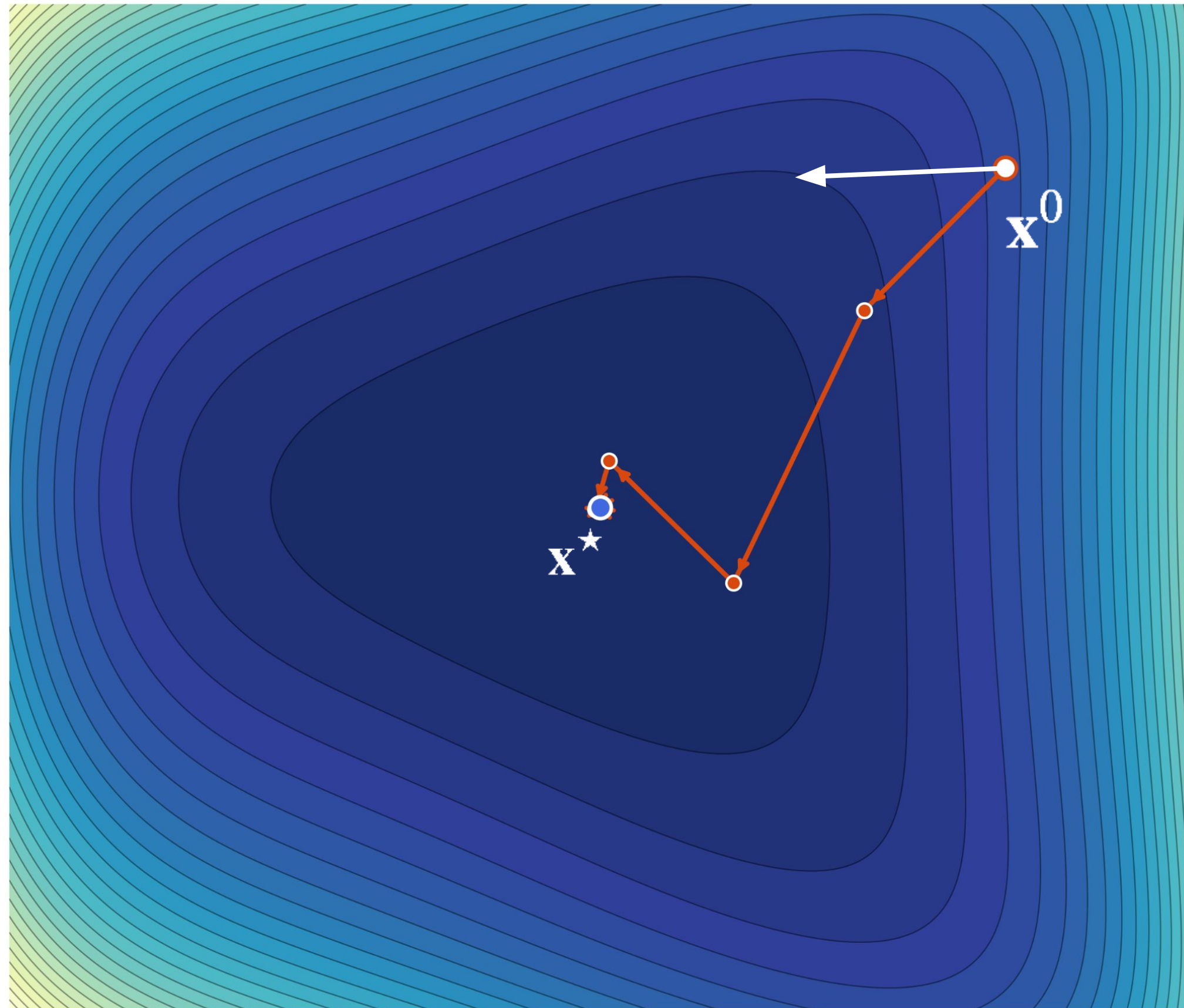
Abstract—Practical time steps in today's state-of-the-art simulators typically rely on Newton's method to solve large systems of nonlinear equations. In practice, this method is often limited by the need to solve large linear systems, which is computationally expensive. We present a new optimization integrator that is designed to be efficient and stable for large time steps. It is based on a novel optimization method that is able to handle large time steps and is robust to non-convex optimization problems. We demonstrate the performance of our method on a variety of simulation problems, including rigid body dynamics, mass-spring systems, and large-deformation dynamics.

Incremental Potential Contact: Intersection- and Inversion-free, Large-Deformation Dynamics

MINCHEN LI, University of Pennsylvania & Adobe Research
 ZACHARY FERGUSON and TESEO SCHNEIDER, New York University
 TIMOTHY LANGLOIS, Adobe Research
 DENIS ZORIN and DANIELE PANOZZO, New York University
 CHENFANFU JIANG, University of Pennsylvania
 DANNY M. KAUFMAN, Adobe Research



Let's Optimize!



Gradient Descent :(

Newton's Method :D

Recap

- First and foremost: modeling
- These need to hold certain properties
- We need information about them (e.g. derivatives)
- We need proper solver infrastructure

We get robust and unified strong coupling

Foundations

Foundations TOC

1. Optimization Time Integration
2. Newton's Method
3. Global Derivatives and Assembly
4. Penalty and Barrier Constraints
5. Line Search
6. Descent Directions and Hessian Definiteness
7. Convergence and Inexactness

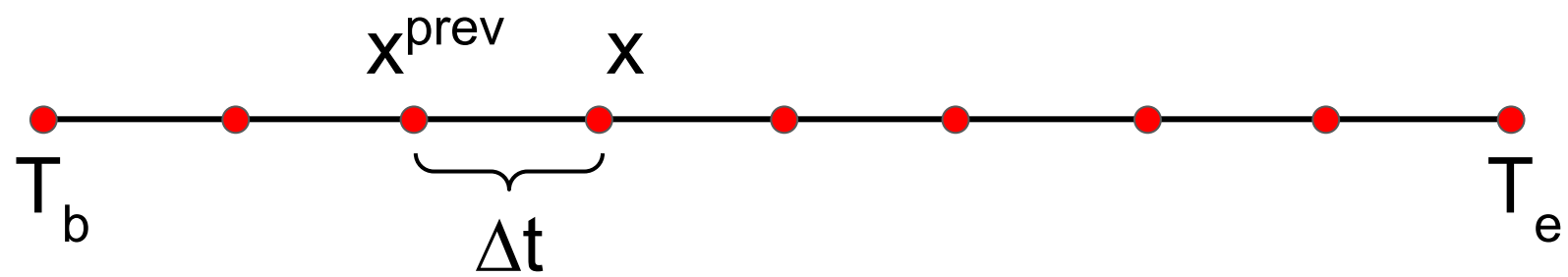


Optimization Time Integration

Equations of Motion:

$$\mathbf{M}\dot{\mathbf{v}} = \sum_i \mathbf{f}_i(\mathbf{x}, \mathbf{v})$$

mass
acc.
position
velocity
forces



$$\dot{\mathbf{v}} \approx \frac{\mathbf{v} - \mathbf{v}^{\text{prev}}}{\Delta t}, \quad \mathbf{v} \approx \frac{\mathbf{x} - \mathbf{x}^{\text{prev}}}{\Delta t} \quad (\text{Backward Euler})$$

Optimization Time Integration

$$\mathbf{M} \frac{\mathbf{x} - \mathbf{x}^{\text{prev}} - \Delta t \mathbf{v}^{\text{prev}}}{\Delta t^2} - \sum_i \mathbf{f}_i(\mathbf{x}) = \mathbf{0} \quad \rightarrow \text{Root finding}$$

Conservative forces: $\mathbf{f}_i(\mathbf{x}) = -\nabla \phi_i(\mathbf{x})$ potential

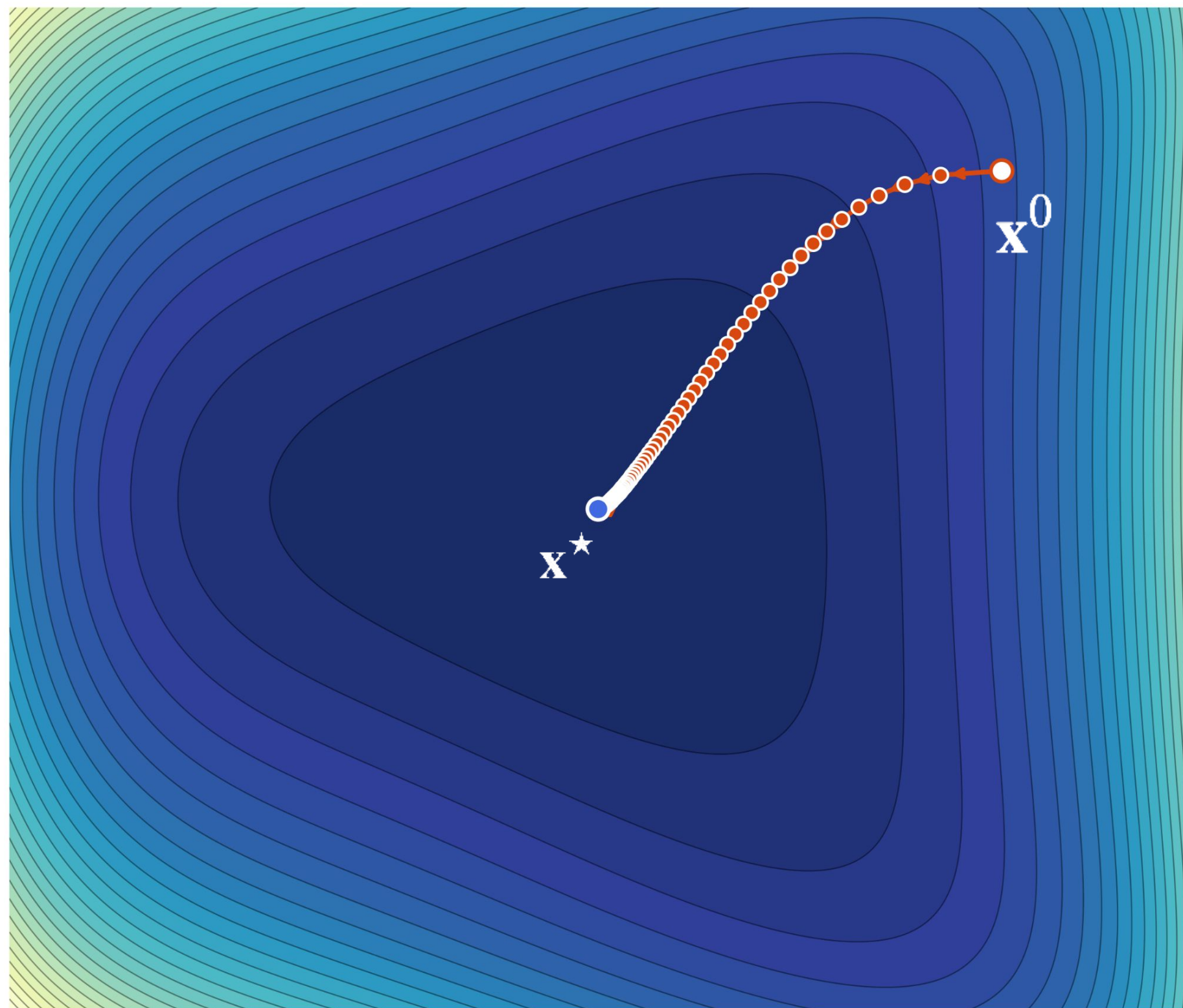
$$\nabla E(\mathbf{x}) = \mathbf{0}$$

$$E(\mathbf{x}) = \frac{1}{2\Delta t^2} (\mathbf{x} - \tilde{\mathbf{x}})^T \mathbf{M} (\mathbf{x} - \tilde{\mathbf{x}}) + \sum_i \phi_i(\mathbf{x})$$

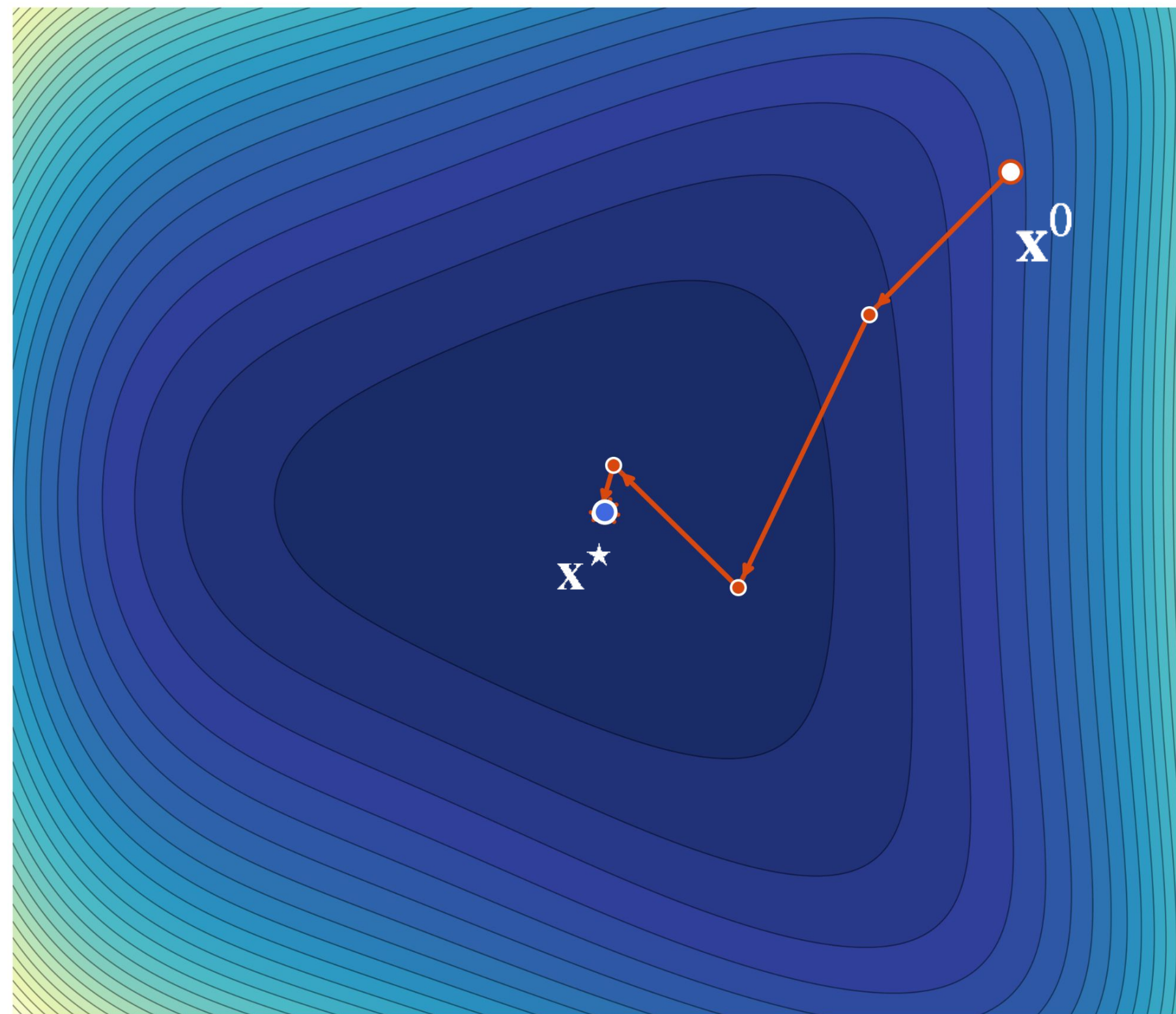
→ Optimization

$$\tilde{\mathbf{x}} = \mathbf{x}^{\text{prev}} + \Delta t \mathbf{v}^{\text{prev}}$$

Newton's Method



Gradient Descent



Newton's Method

But how?

Newton's Method

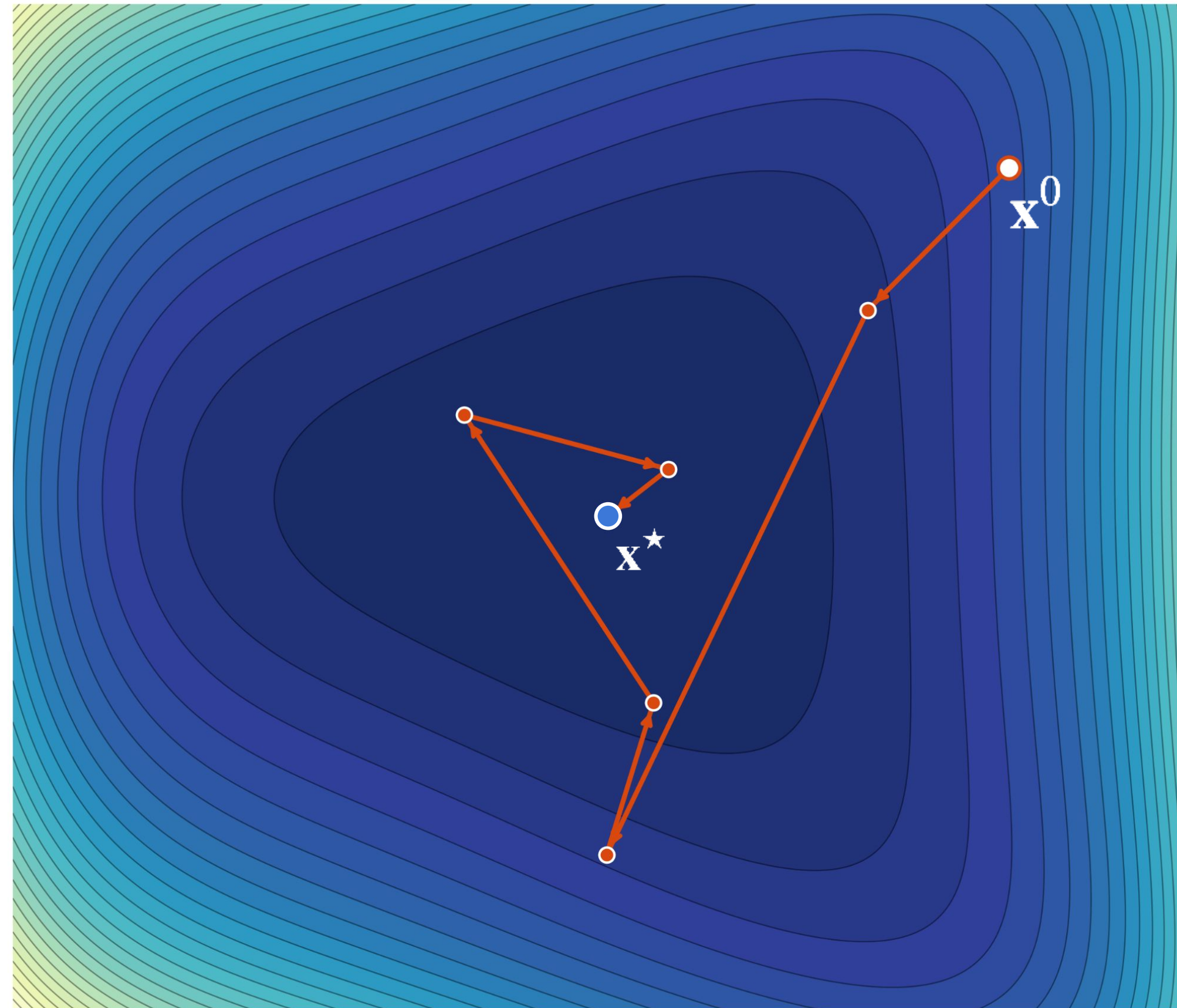
Quadratic model:

$$m_k(\Delta \mathbf{x}) = E(\mathbf{x}) + \nabla E(\mathbf{x})^T \Delta \mathbf{x} + \frac{1}{2} \Delta \mathbf{x}^T \nabla^2 E(\mathbf{x}) \Delta \mathbf{x}$$

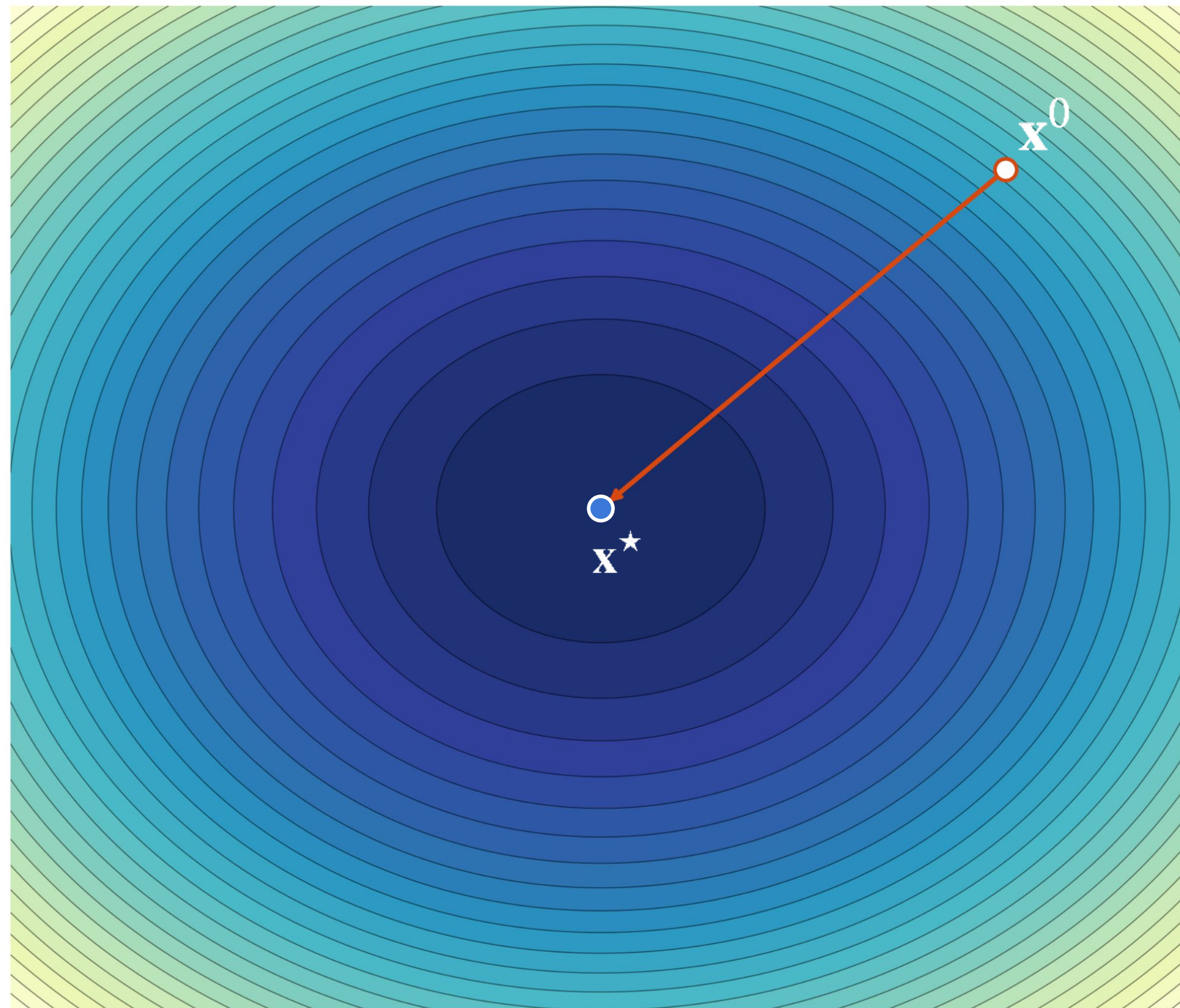
Go to minimizer:

$$\nabla^2 E(\mathbf{x}) \Delta \mathbf{x} = -\nabla E(\mathbf{x})$$

Newton's Method

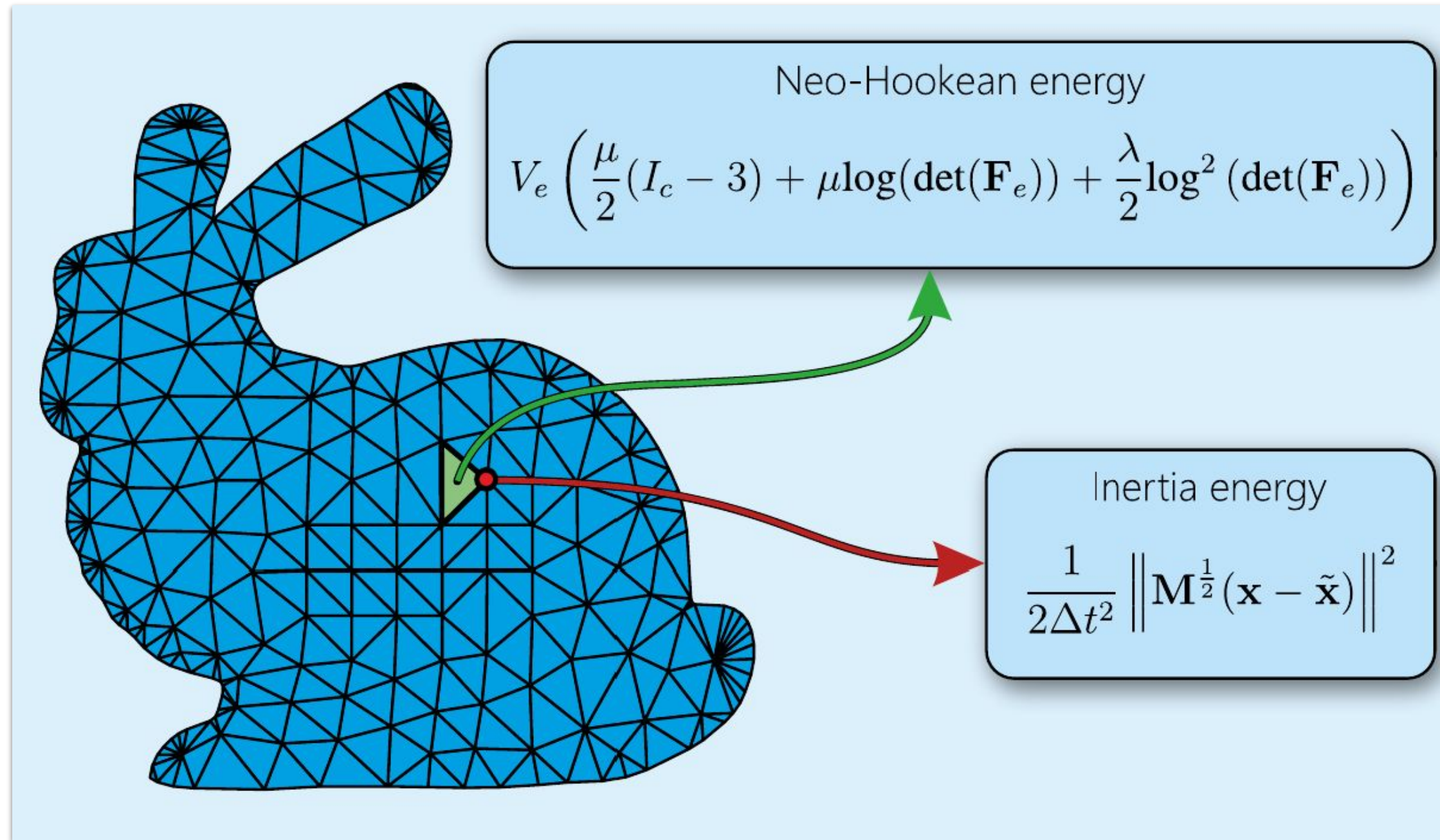


If E is quadratic...



1 iteration!

Global Derivatives and Assembly



SymX [Fernández-Fernández et al. 2025]

\mathcal{P} Set of unique potentials

\mathcal{T} Set of discretization elements

$$\begin{aligned}
 E(\mathbf{x}) &= \sum_{p \in \mathcal{P}} \sum_{e \in \mathcal{T}_p} \phi_p^e \\
 \mathbf{g} = \nabla E(\mathbf{x}) &= \sum_{p \in \mathcal{P}} \sum_{e \in \mathcal{T}_p} (\mathbf{P}^e)^T \nabla \phi_p^e \\
 \mathbf{H} = \nabla^2 E(\mathbf{x}) &= \sum_{p \in \mathcal{P}} \sum_{e \in \mathcal{T}_p} (\mathbf{P}^e)^T \nabla^2 \phi_p^e \mathbf{P}^e
 \end{aligned}$$

Dimensional annotations: 1×1 (orange), $N \times 1$ (orange), $N \times M$ (green), $M \times 1$ (blue), $N \times N$ (orange), $M \times M$ (blue).

Multi Systems

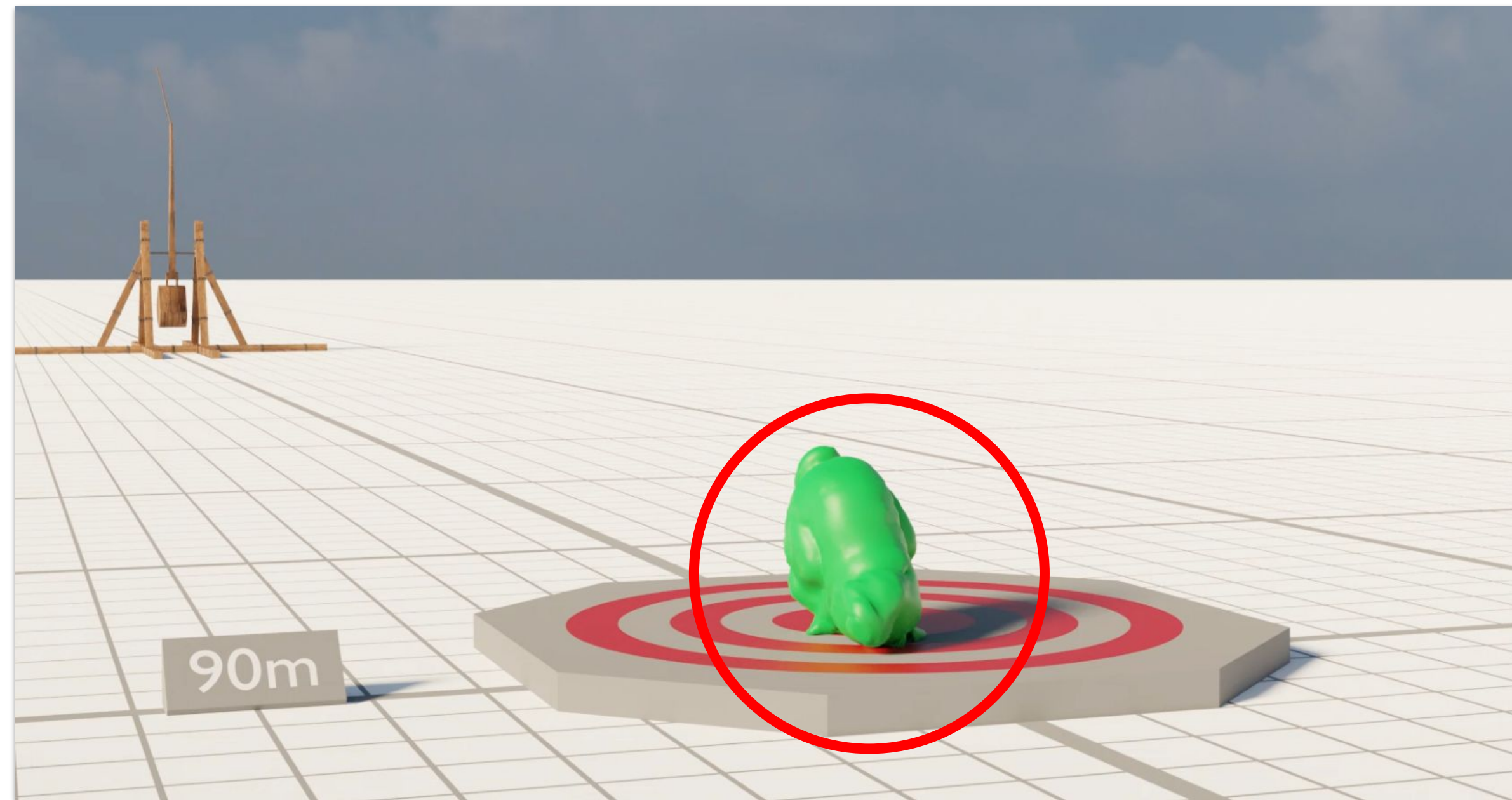
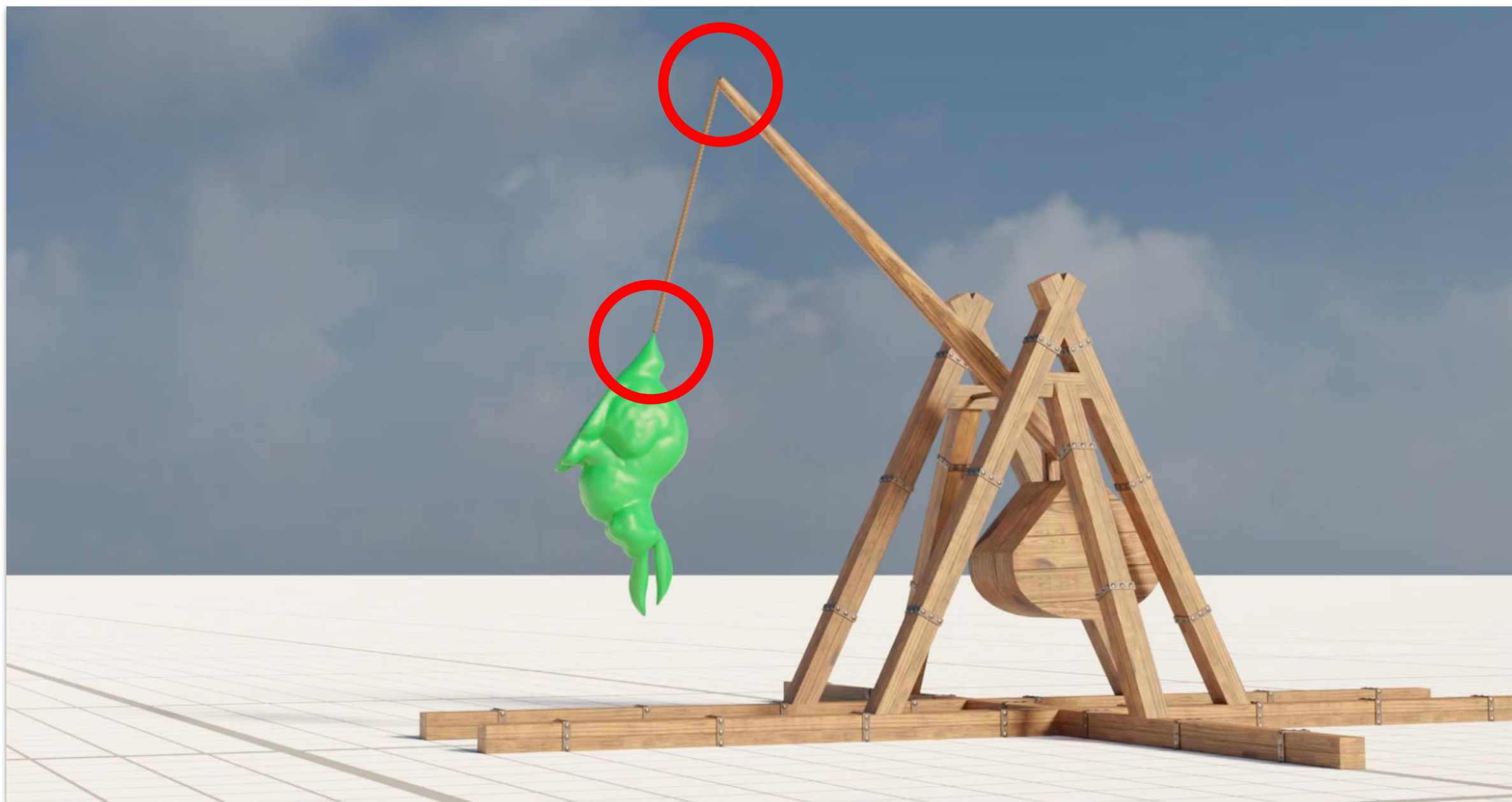
Multiple sets of DoFs:

$$\mathbf{x} = (\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_n)^T$$



STARK [Fernández-Fernández et al. 2024]

Penalty and Barrier Constraints



SymX [Fernández-Fernández et al. 2025]

Penalty and Barrier Constraints

Make it expensive:

$$\phi_p(\mathbf{x}) = \frac{1}{2} k_c C(\mathbf{x})^2$$

stiffness $\gg 0$
constraint violation

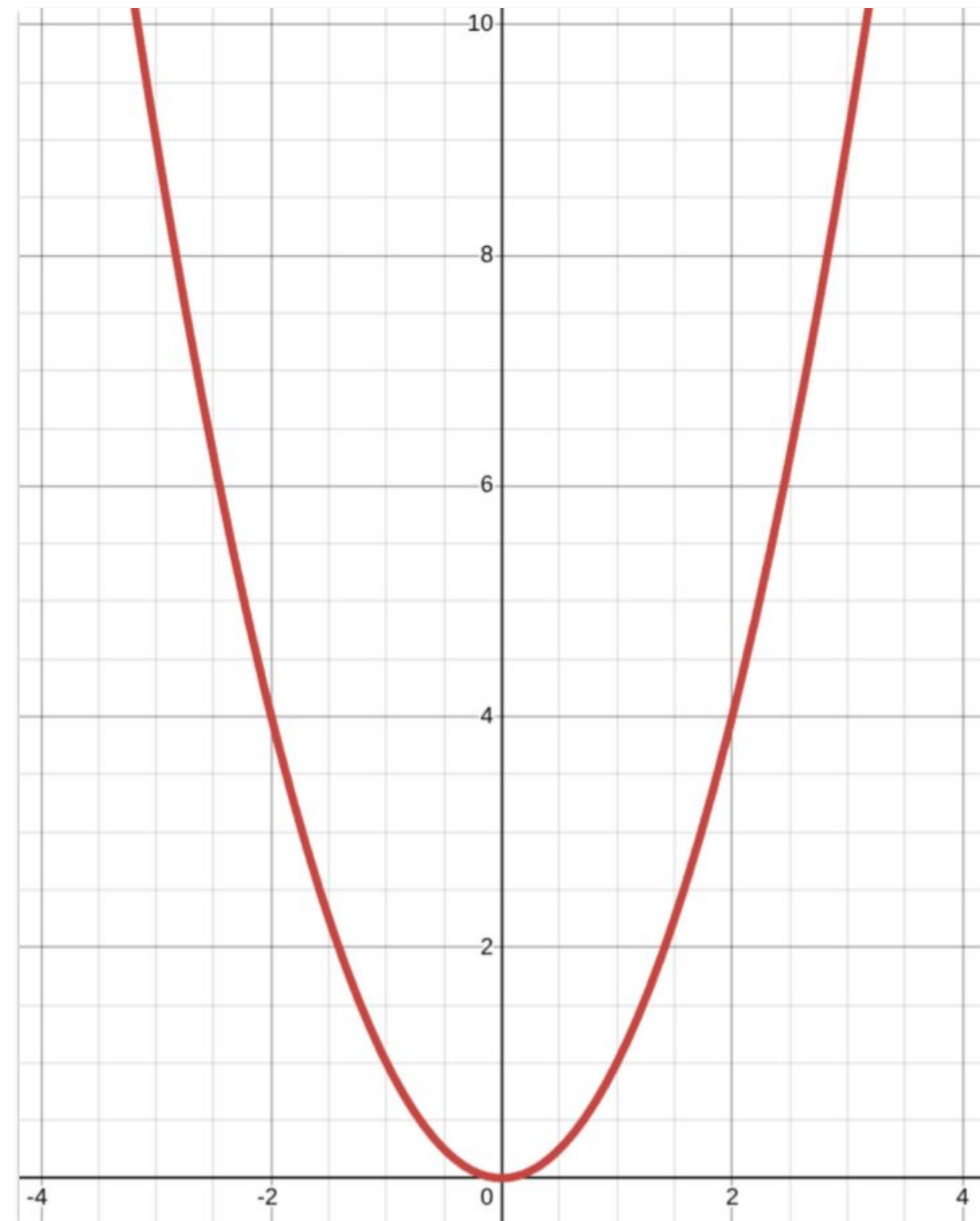
Activation:

$$\phi_{in}(\mathbf{x}) = \begin{cases} 0, & C(\mathbf{x}) \leq 0, \\ \frac{k_c}{3} C(\mathbf{x})^3, & 0 < C(\mathbf{x}) \end{cases}$$

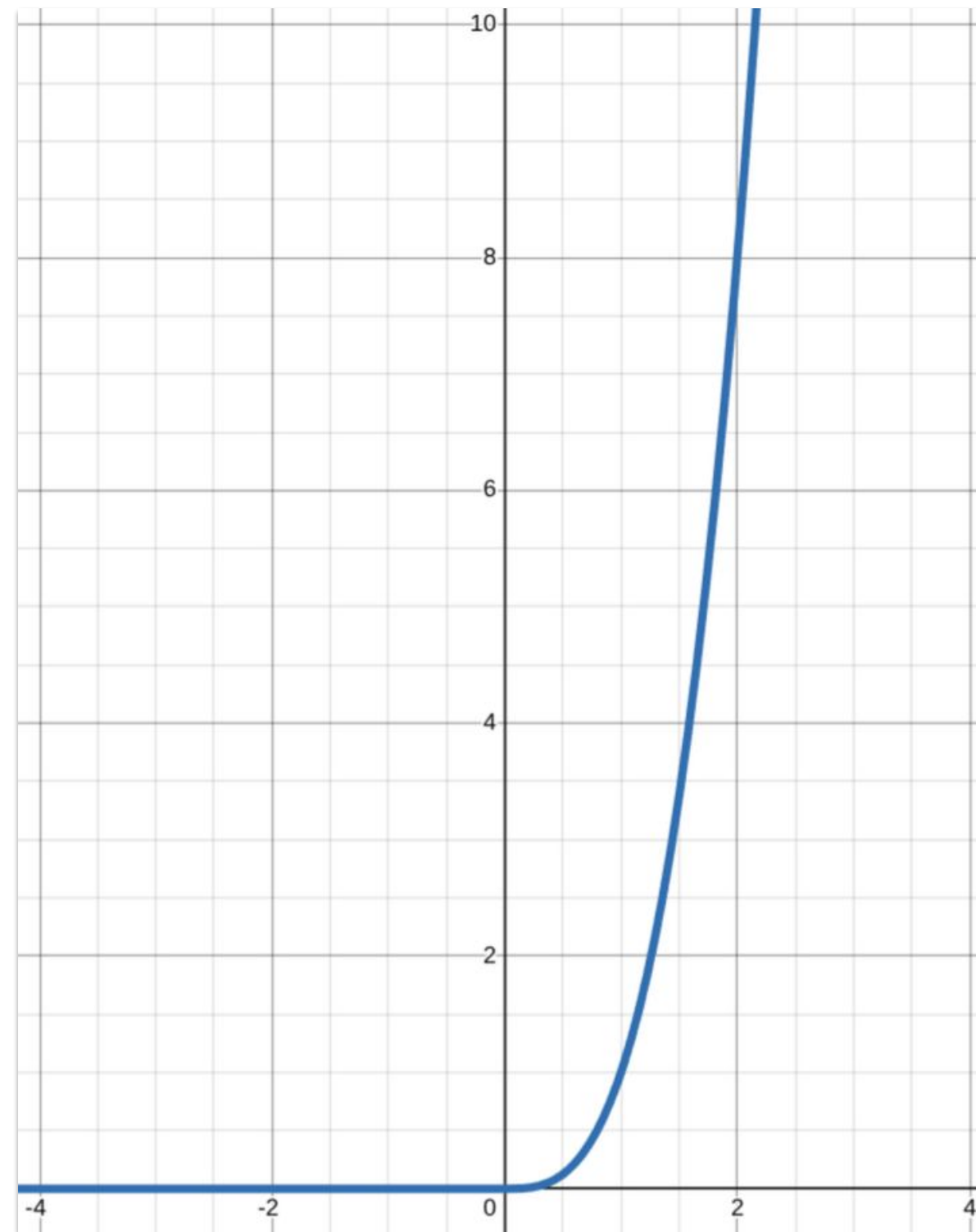
Barrier:

$$\phi_b(\mathbf{x}) = \begin{cases} 0, & C(\mathbf{x}) \leq 0, \\ -k_b C(\mathbf{x})^2 \ln\left(1 - \frac{C(\mathbf{x})}{C_{\text{lim}}}\right), & 0 < C(\mathbf{x}) < C_{\text{lim}}, \\ +\infty, & C(\mathbf{x}) \geq C_{\text{lim}} \end{cases}$$

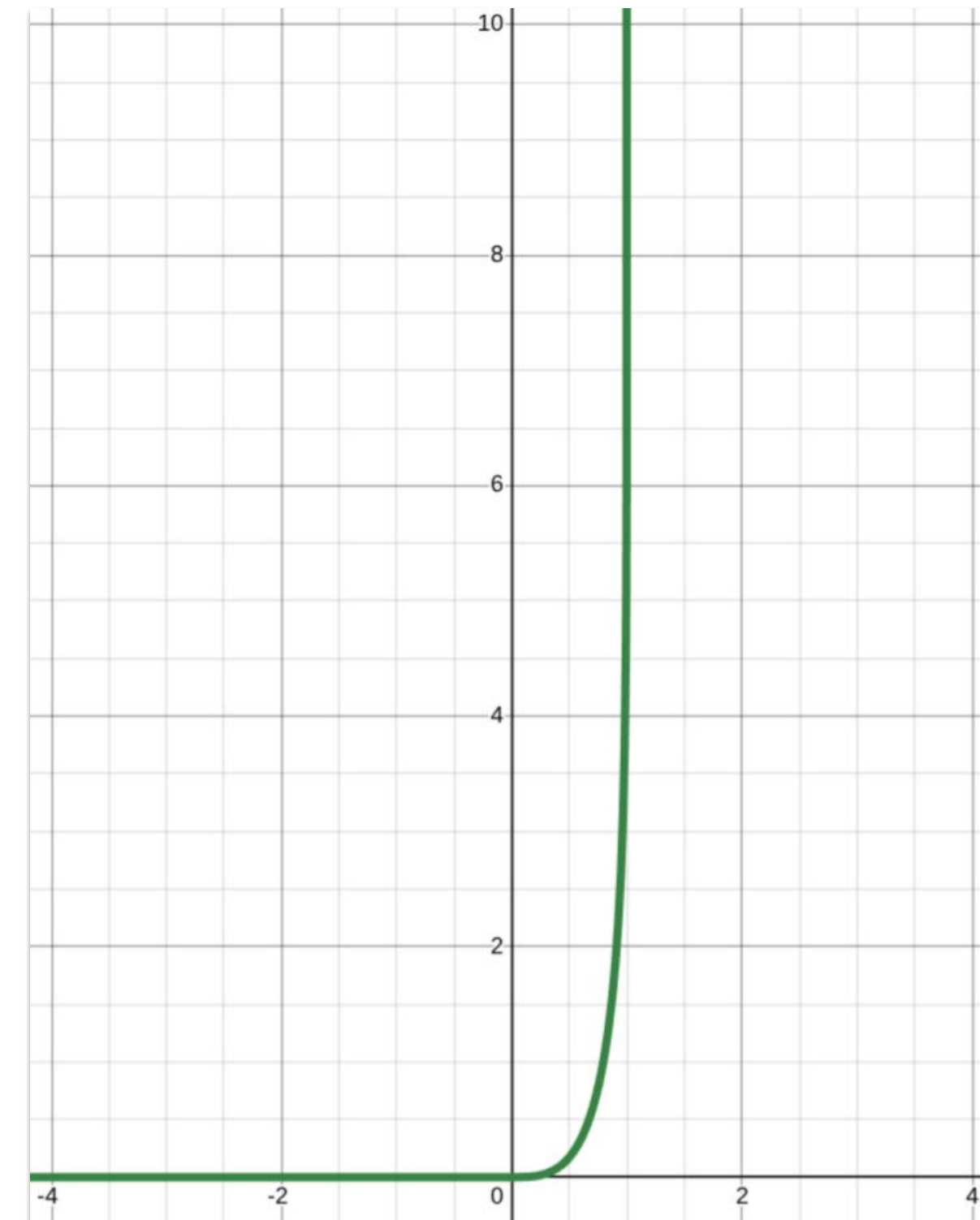
Penalty and Barrier Constraints



Equality Quadratic



Inequality Cubic

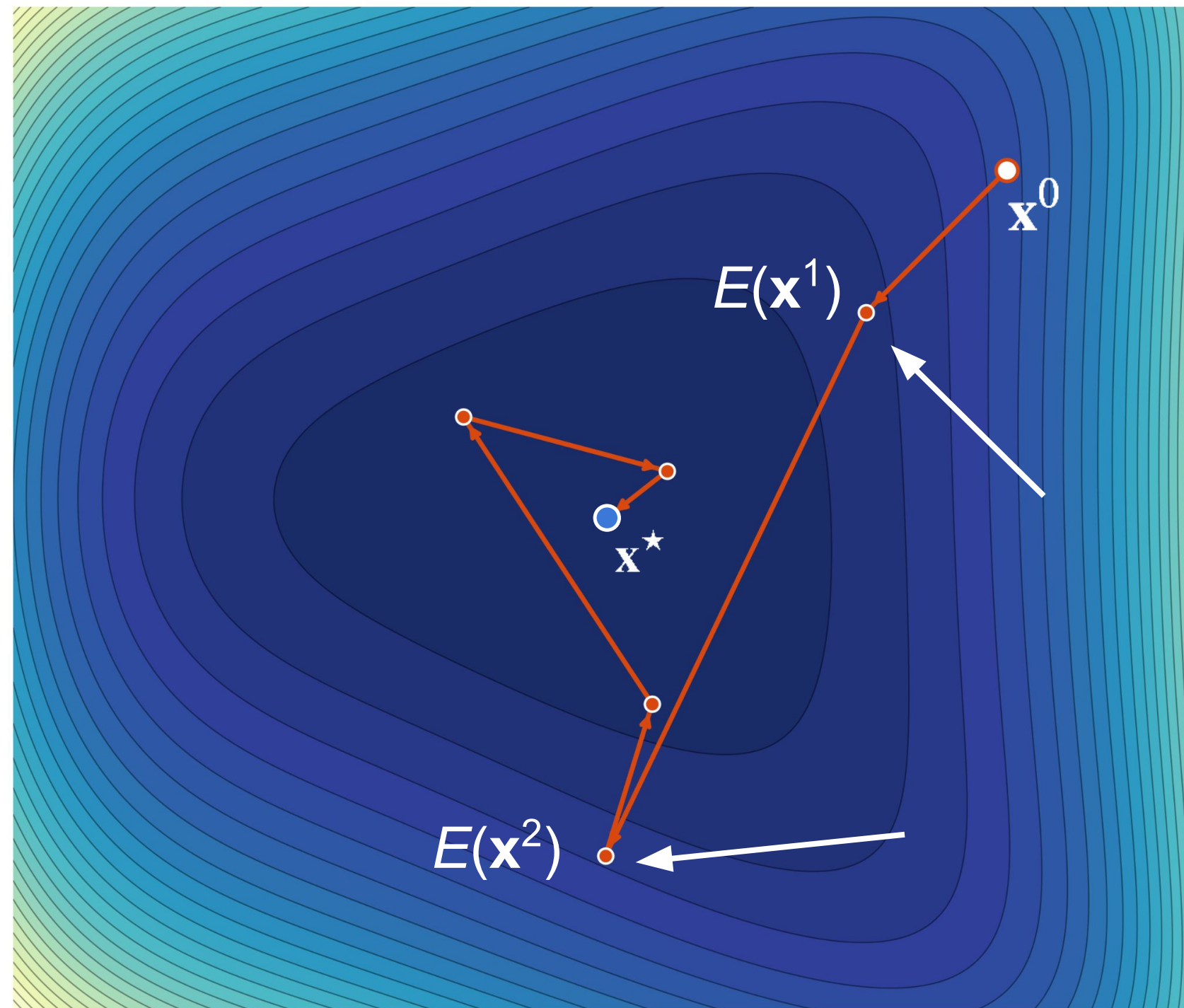


Barrier ($C_{\text{lim}} = 1$)

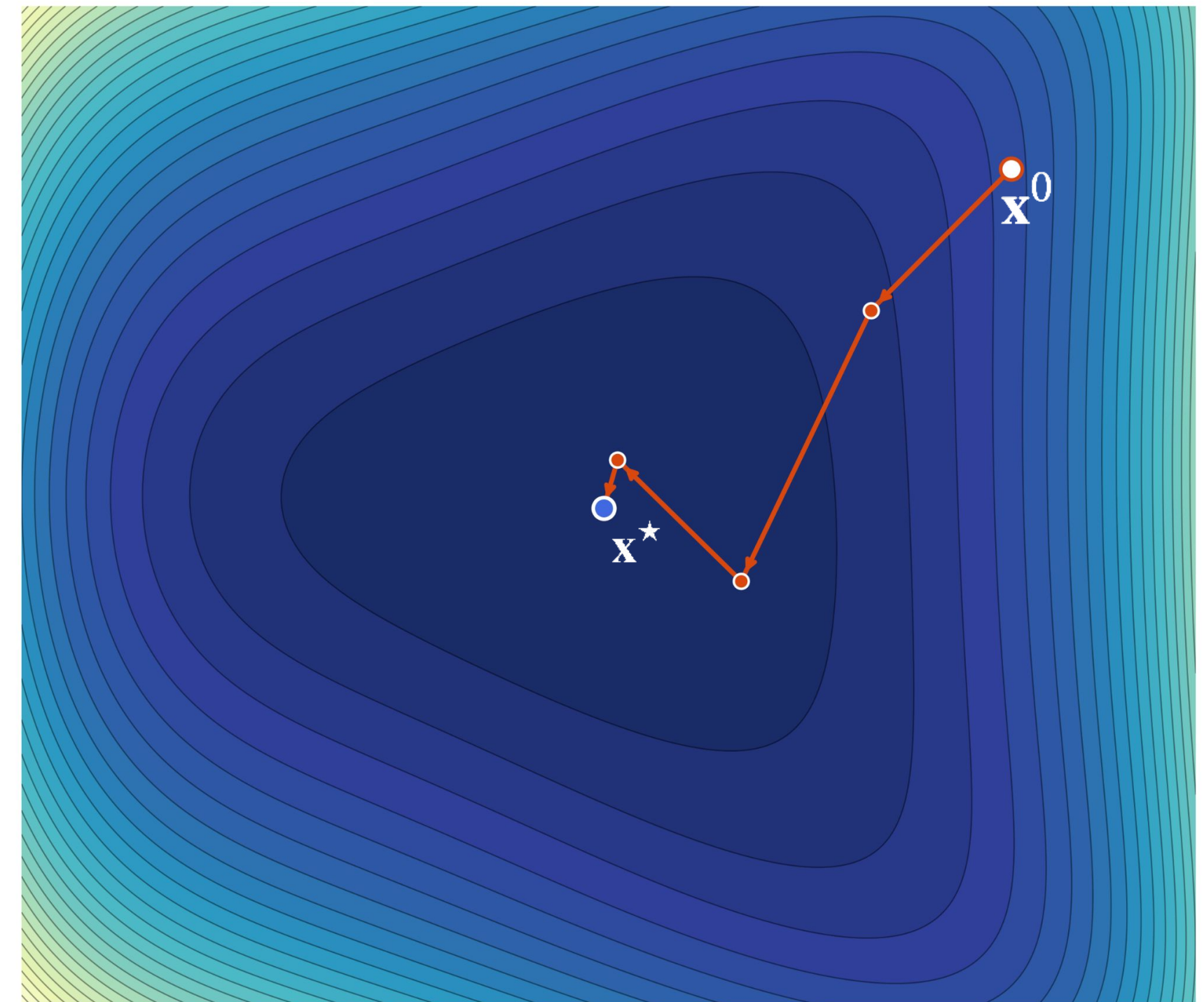
Line Search

Remember?

$$E(\mathbf{x}^2) > E(\mathbf{x}^1)$$

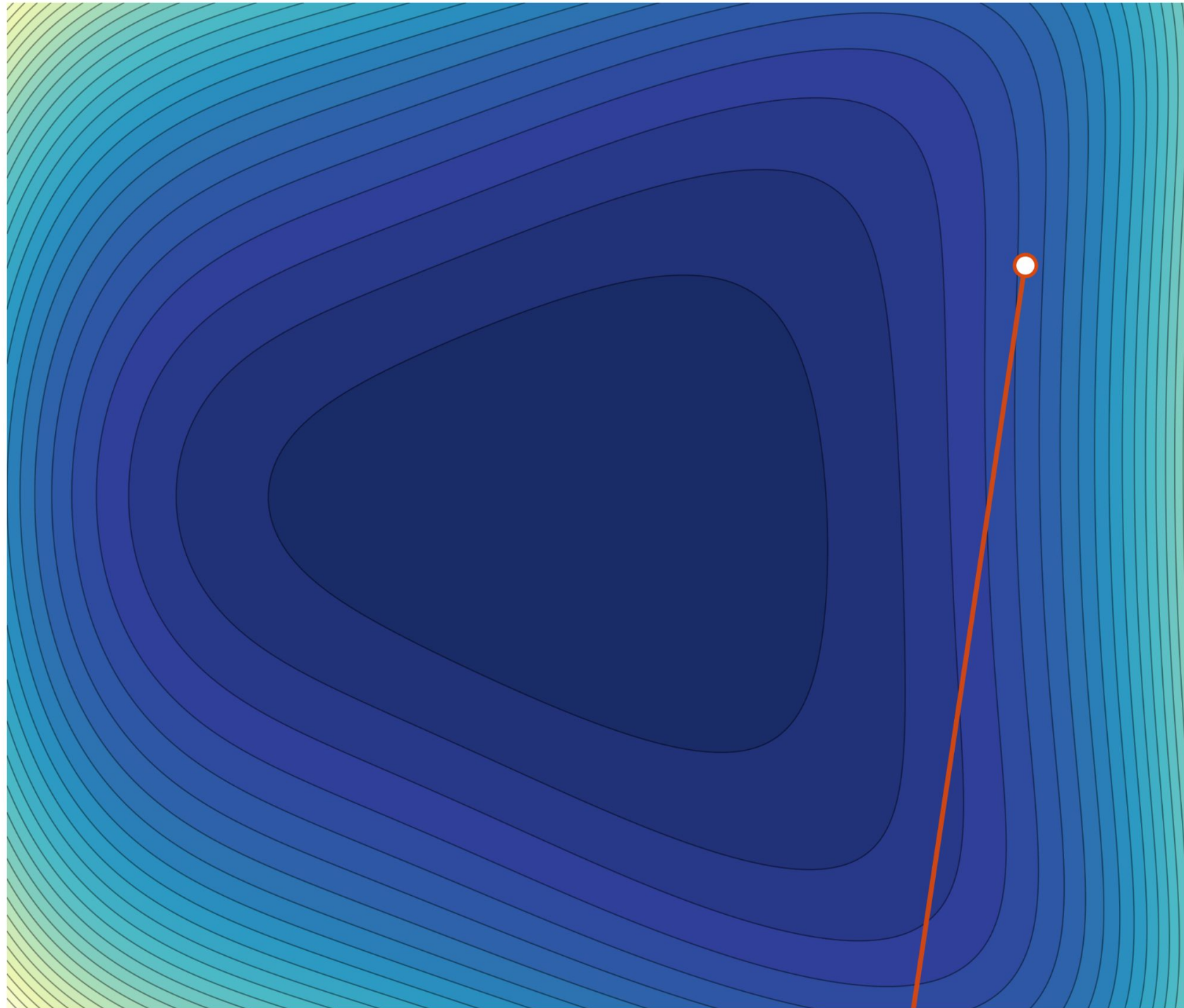


Backtracking!



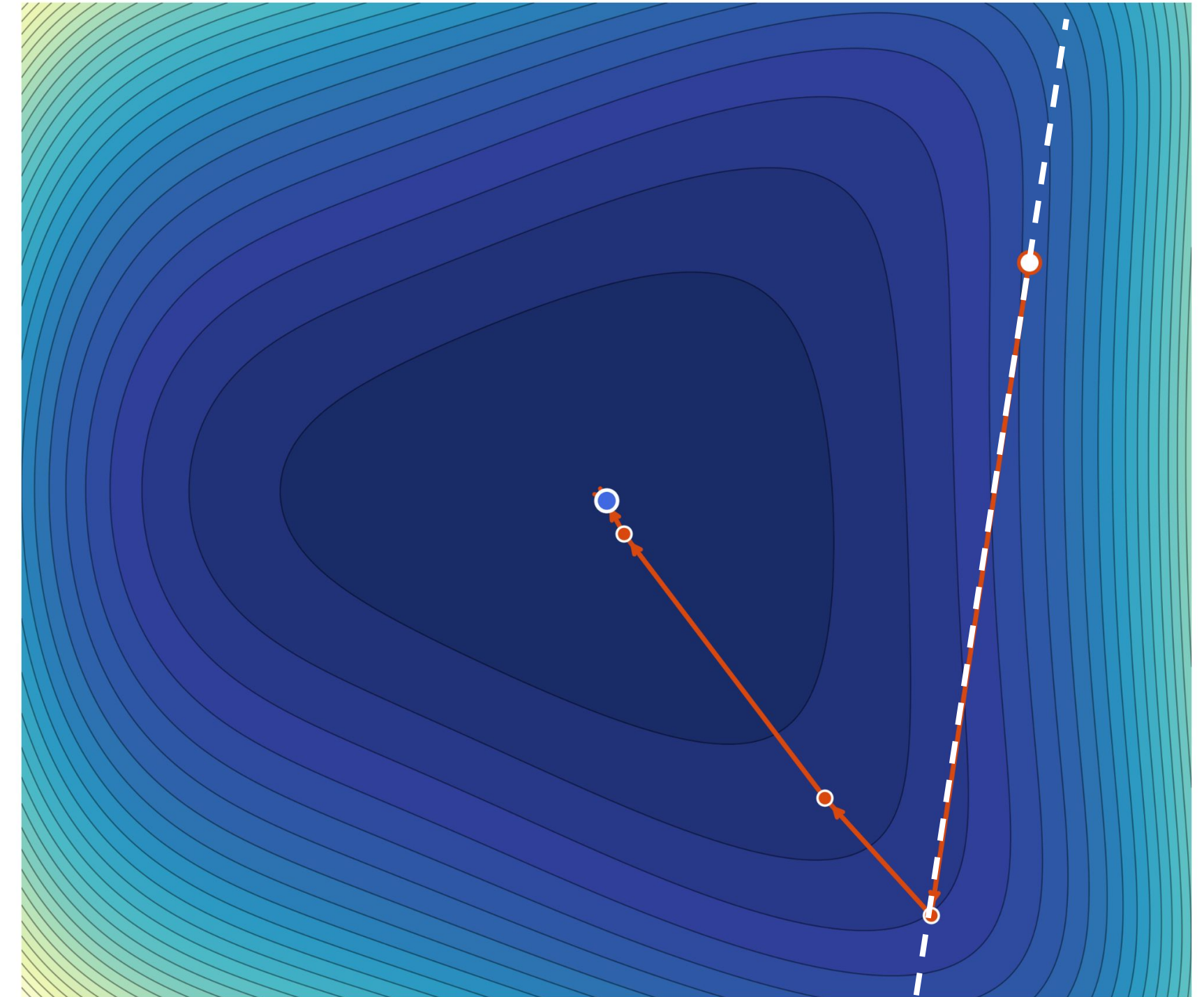
Line Search

Different start

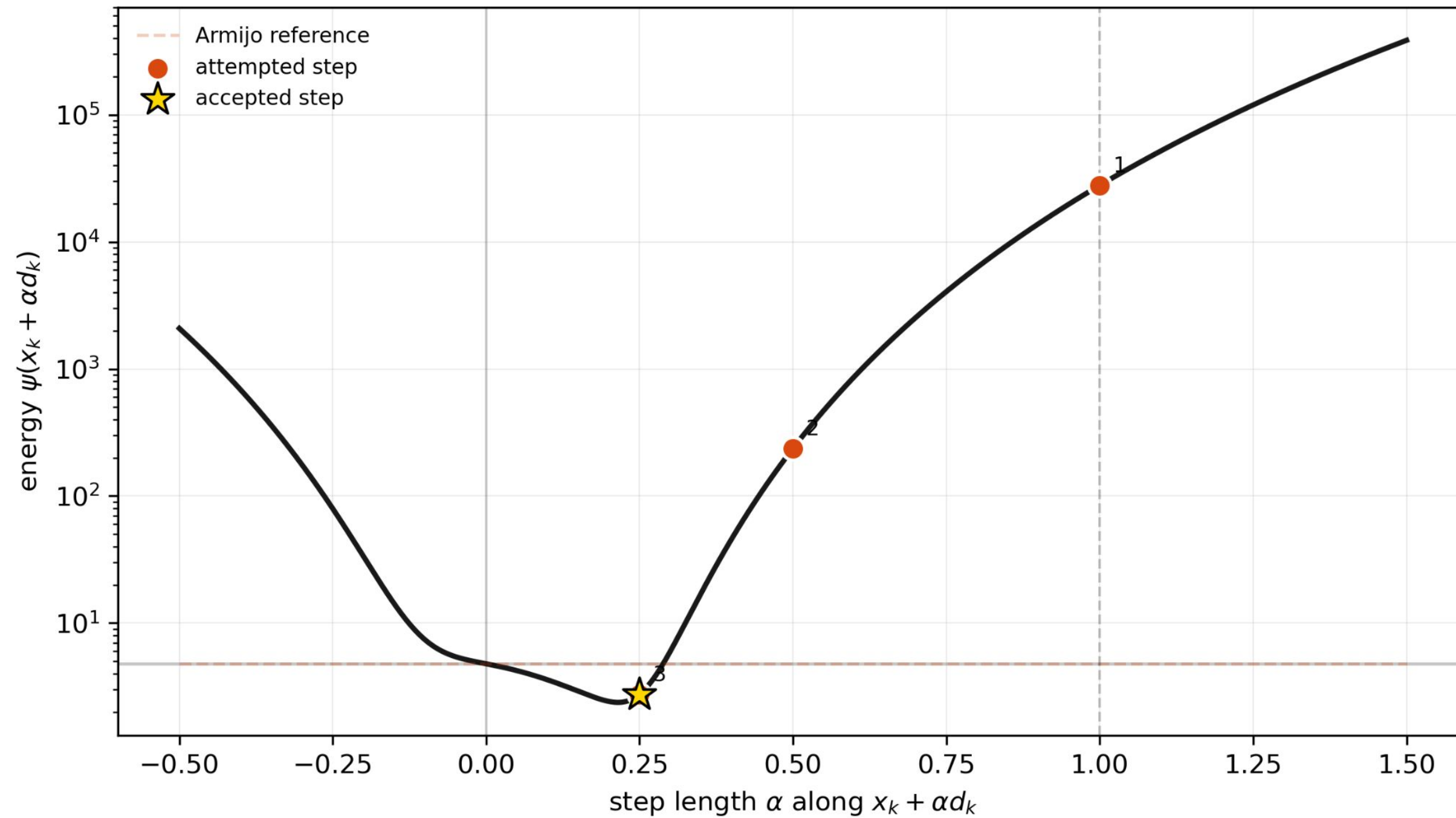


... never to be seen again :(

Backtracking



Line Search



Armijo's Condition

$$E(\mathbf{x} + \alpha \Delta \mathbf{x}) \leq E(\mathbf{x}) + c_1 \underbrace{\alpha \nabla E(\mathbf{x})^T \Delta \mathbf{x}}_{\text{1st order expected decrease}}$$

Tiny fraction

1st order expected decrease

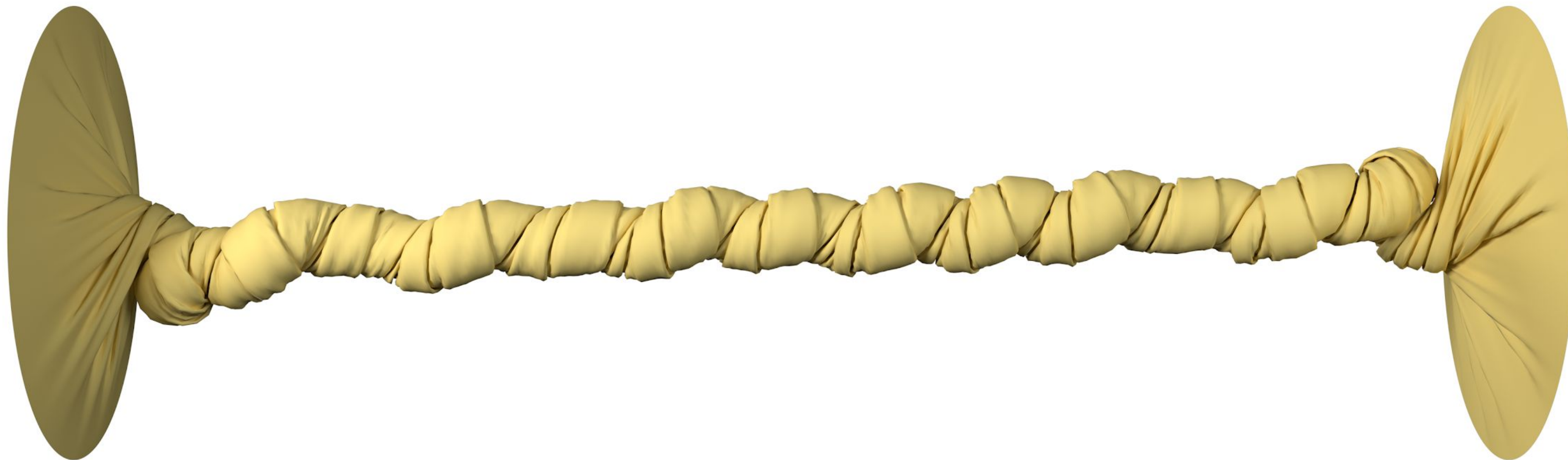
Always Progresses

Line Search

Our line search procedure:

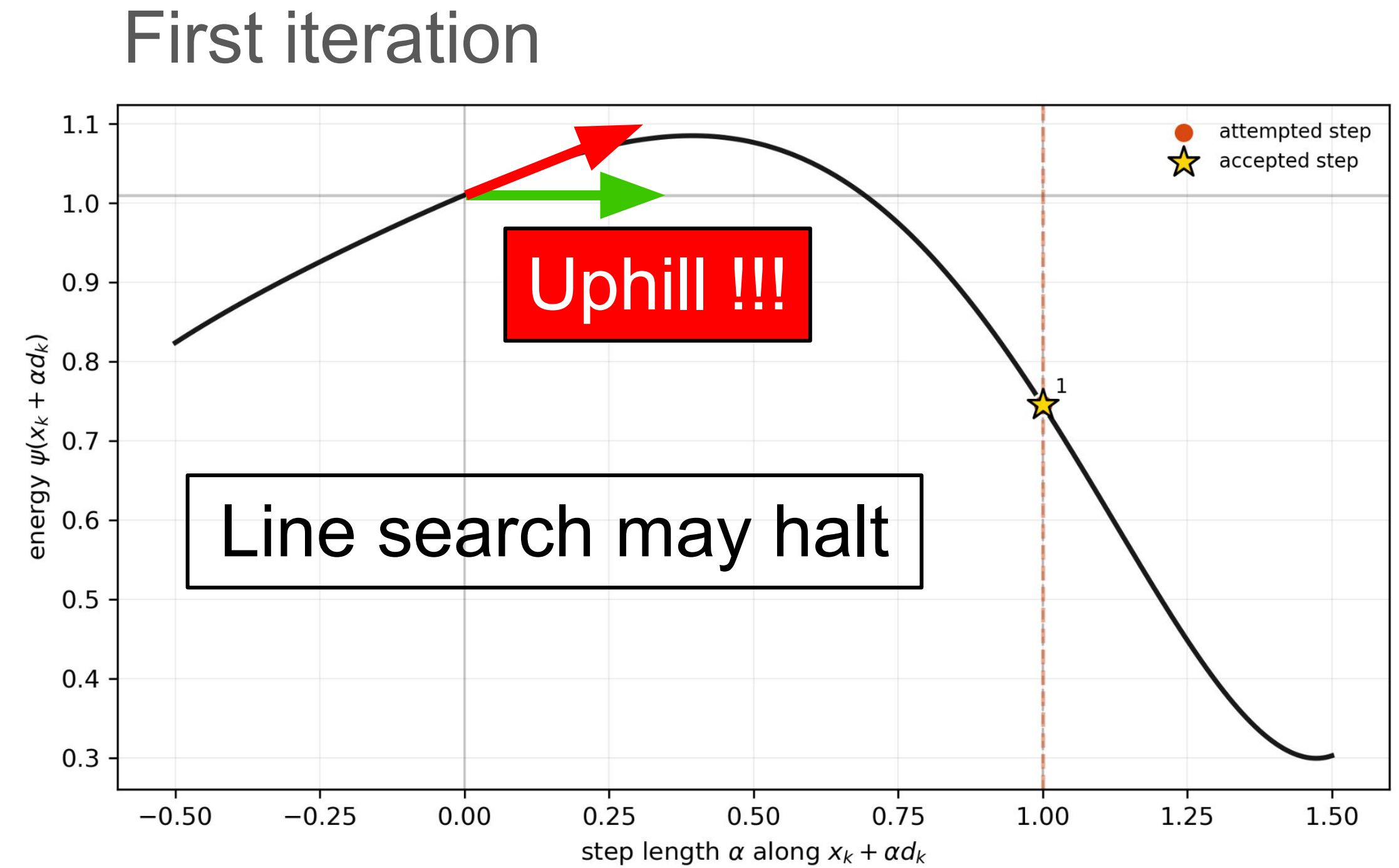
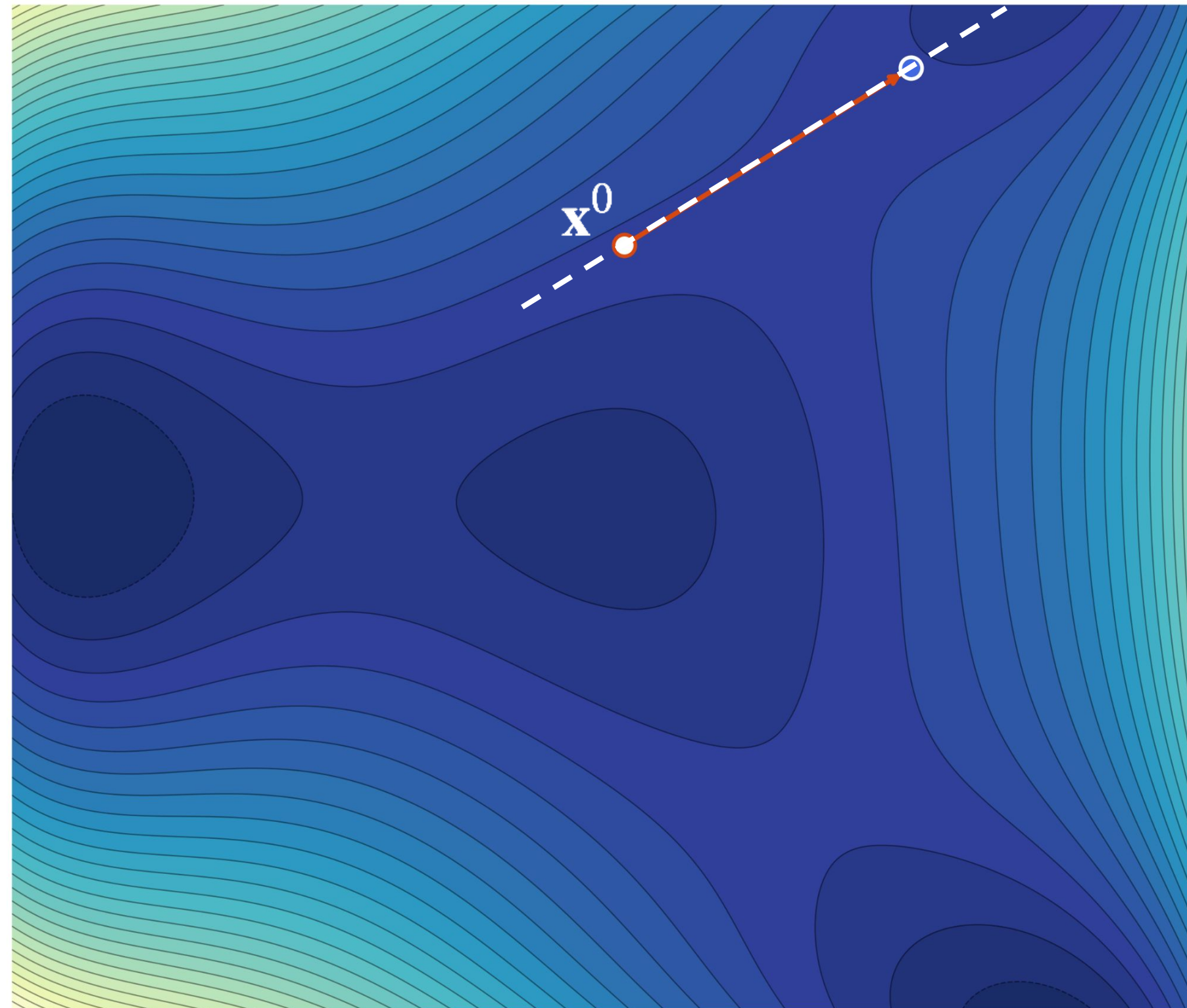
- Step cap
 - Avoids stupidly large steps. Potentially useful for quasistatics.
- Max step
 - Computable max step length in the search direction. E.g. Continuous Collision Detection.
- Admissibility backtracking
 - Backtrack from negative logarithms and sqrts, inverted elements...
- Armijo's condition backtracking
 - Ensures sufficient descend

Line Search

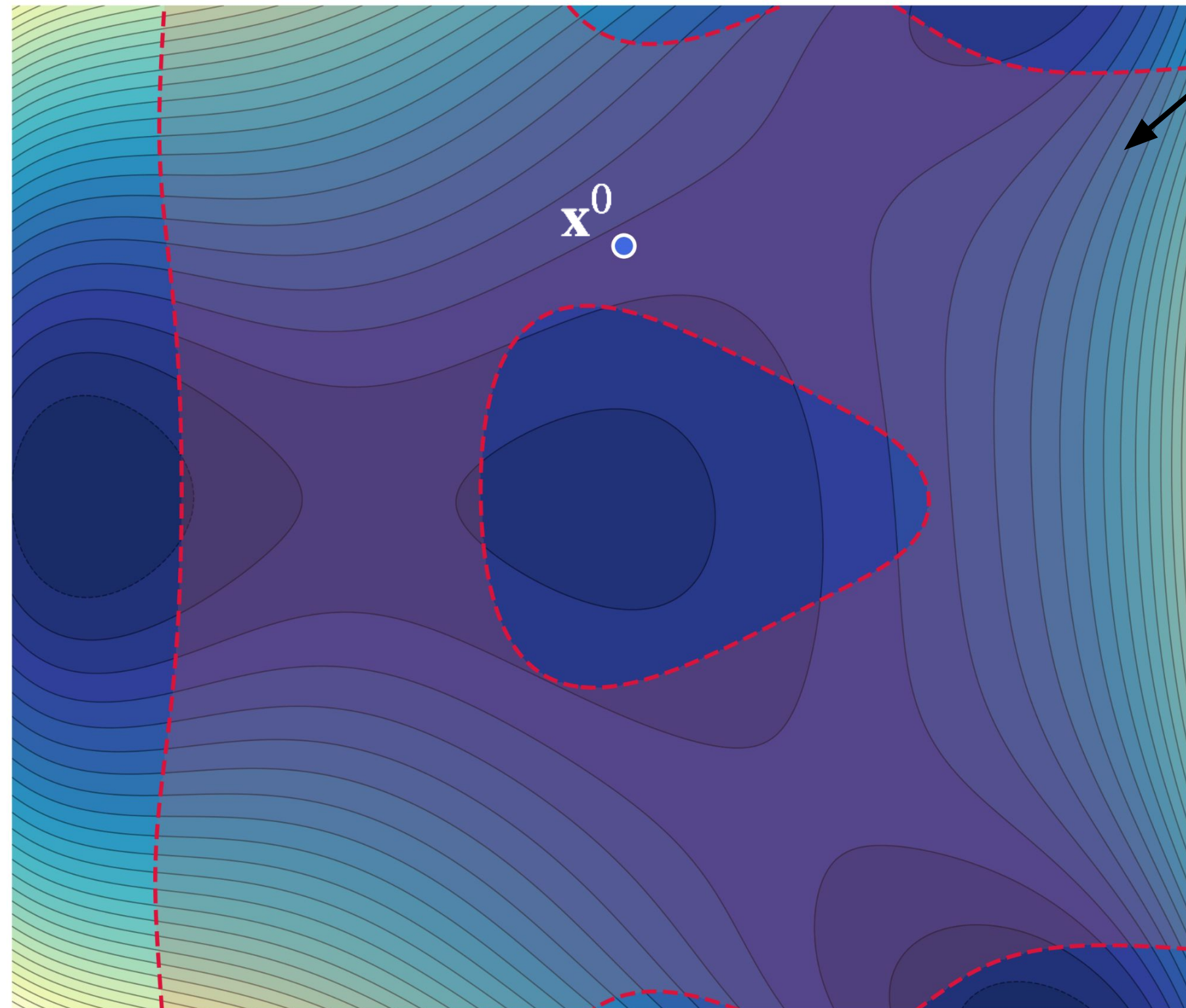


Progressively Projected Newton's Method [Fernández-Fernández et al. 2026]

Descent Directions And Hessian Definiteness



Descent Directions And Hessian Definiteness



Indefiniteness / Non-convex

We need descent directions: $\Delta \mathbf{x}^T \mathbf{g} < 0$

$$\mathbf{H} \Delta \mathbf{x} = -\mathbf{g} \rightarrow \underbrace{-\Delta \mathbf{x}^T \mathbf{H} \Delta \mathbf{x}}_{> 0} = \underbrace{\Delta \mathbf{x}^T \mathbf{g}}_{< 0} < 0$$

$$\mathbf{p}^T \mathbf{H} \mathbf{p} > 0, \quad \forall \mathbf{p} \neq \mathbf{0}$$

\mathbf{H} is Symmetric Positive (Semi)definite

Descent Directions And Hessian Definiteness

Goal: Non-negative eigvals in \mathbf{H}

$$\mathbf{H}(\mathbf{x}) = \sum_{p \in \mathcal{P}} \sum_{e \in \mathcal{T}_p} \mathbf{P}_e^T \mathbf{H}_p^e(\mathbf{x}) \mathbf{P}_e$$

N×N ← $\mathbf{H}(\mathbf{x})$ M×M ← $\mathbf{H}_p^e(\mathbf{x})$

Projected/Filtered Newton

$$\mathbf{H}_p^e = \mathbf{Q} \mathbf{D} \mathbf{Q}^T \longrightarrow \hat{\mathbf{H}}_p^e = \mathbf{Q} \hat{\mathbf{D}} \mathbf{Q}^T$$

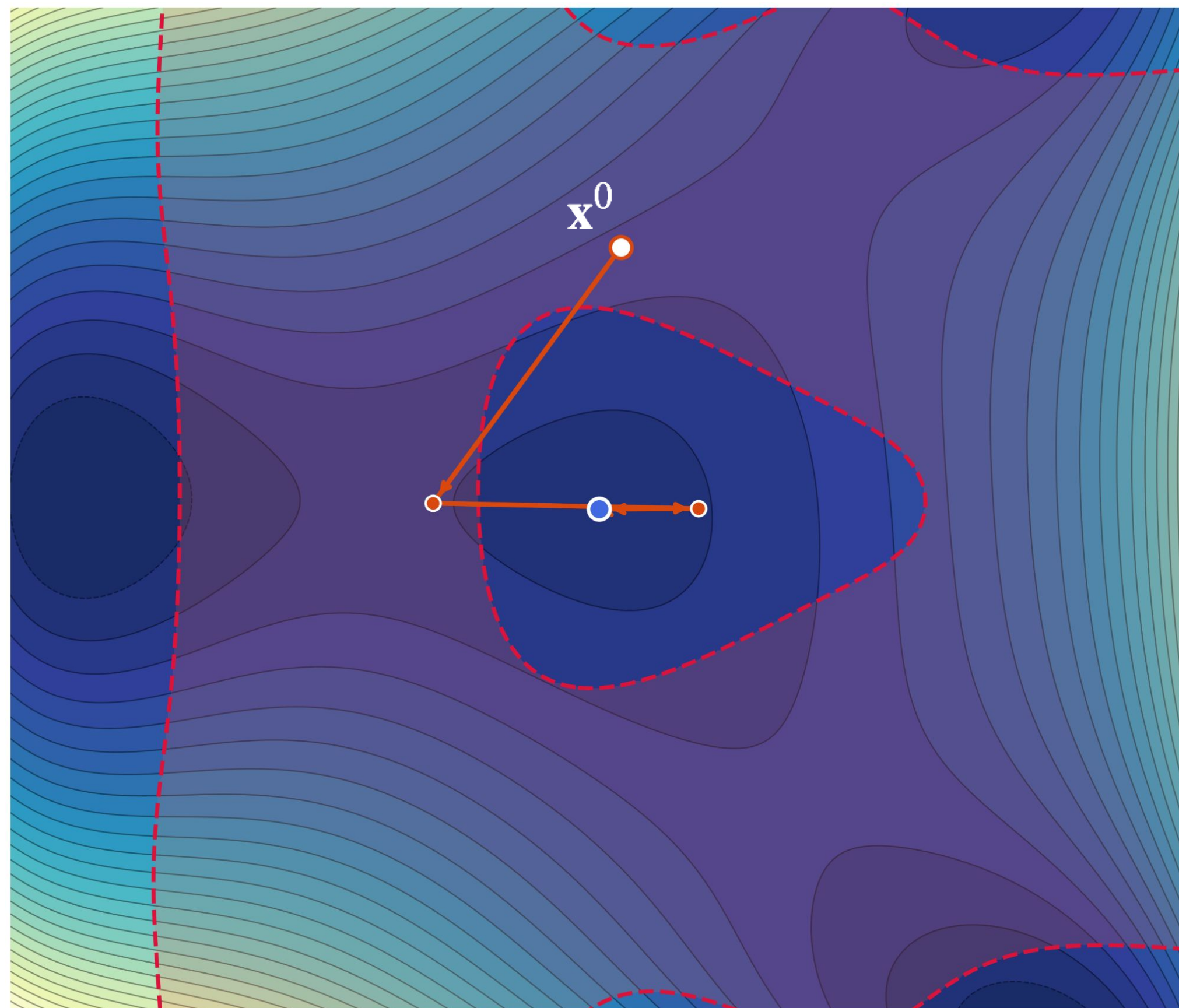
$\hat{D}_{ii}^e = \max(D_{ii}^e, \epsilon)$ Clamping
 $\hat{D}_{ii}^e = \text{abs}(D_{ii}^e)$ Mirroring

Adaptive Projection

- Project-on-Demand (when needed)
- Progressive (when and where needed)

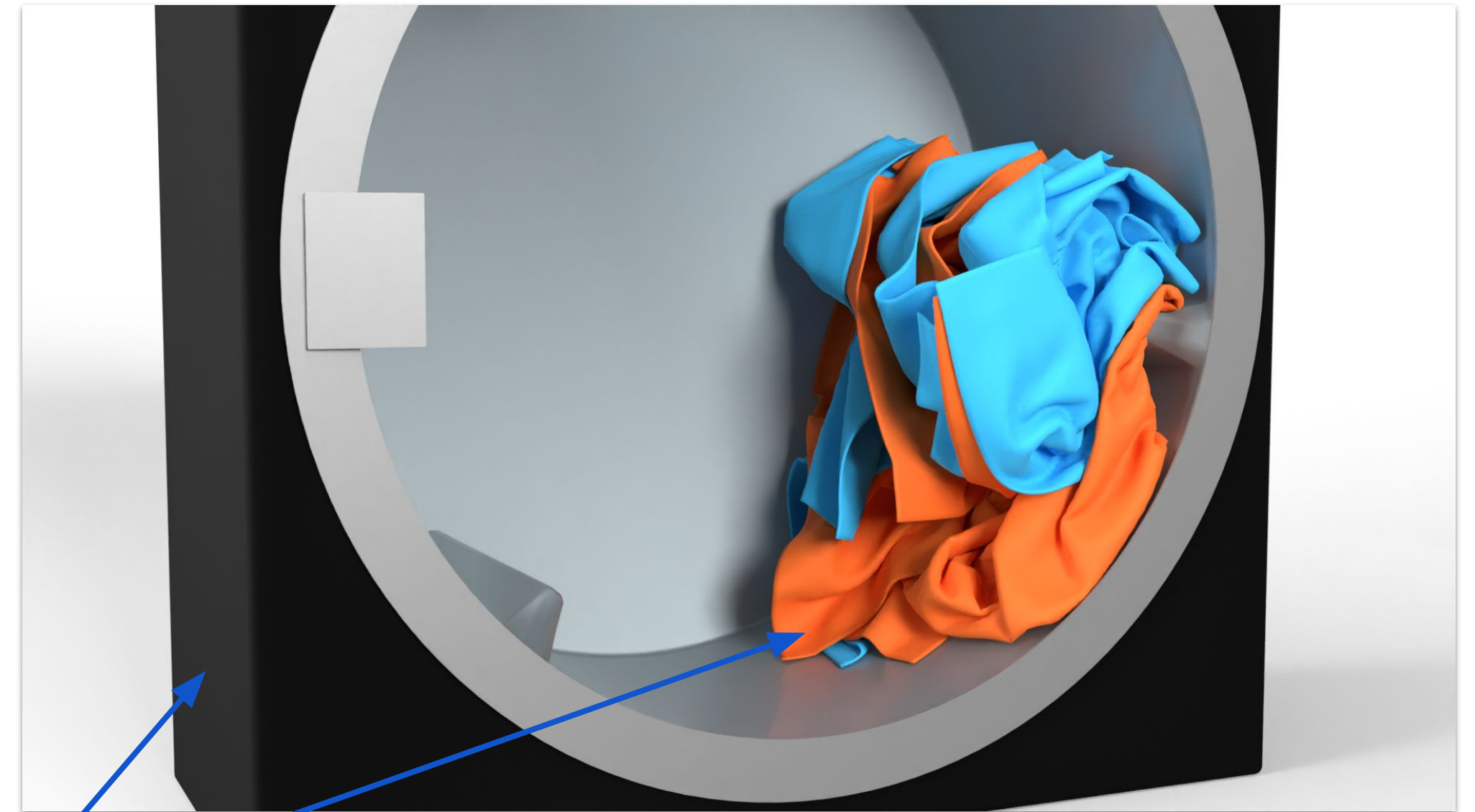
Wednesday · 16:00-17:15 · Kino 5

Descent Directions And Hessian Definiteness



Convergence and Inexactness

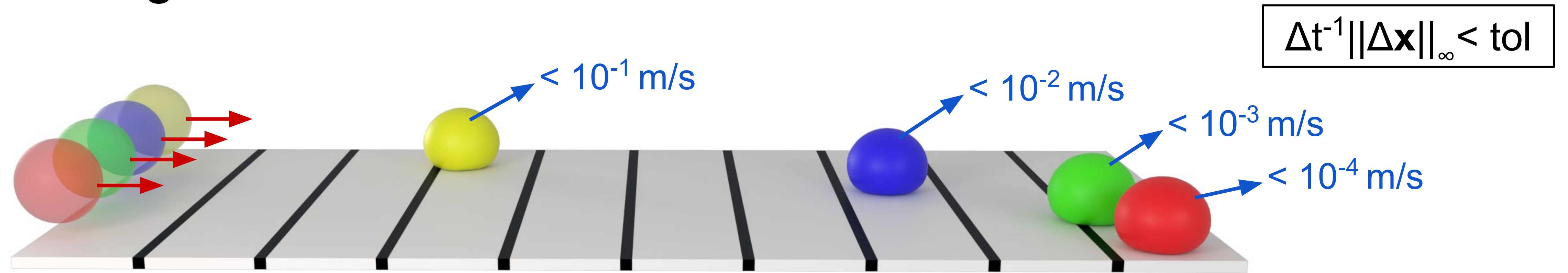
- Newton is rarely driven to very high accuracy
- Line search ensures admissible progress
- Common convergence criteria:
 - Fixed Newton iterations
 - Residual norm $\| \nabla E(\mathbf{x}) \| < \varepsilon_{\text{res}}$ [N]
 - Step size $\| \Delta \mathbf{x} \| < \varepsilon_{\text{step}}$ [m]



SymX [Fernández-Fernández et al. 2025]

Different residual scales!

Convergence and Inexactness

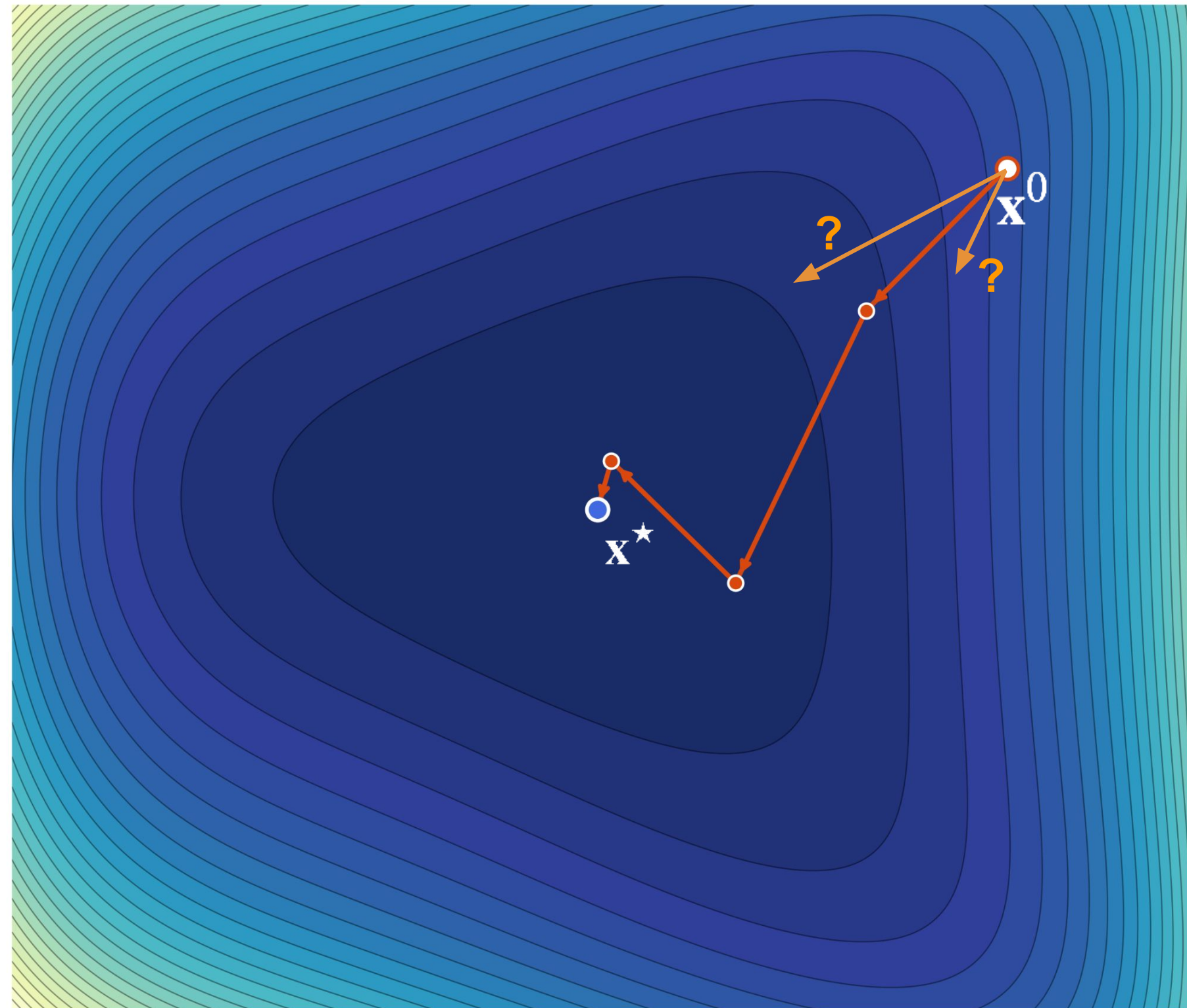


Progressively Projected Newton's Method [Fernández-Fernández et al. 2026]

Consequences of Inexactness:

- Damping
- Overstretching
- Drifting/Sliding

Forcing Sequence



- Iterative linear system solver (PCG)
- Early exits can be very cheap
 - $\varepsilon_{\text{CG}} < 10^{-12}$ vs $\varepsilon_{\text{CG}} < 10^{-3}$
- Far from \mathbf{x}^* we can be inexact
- Very significant performance boost
- Good practical proxy:

$$\|\mathbf{r}_k\| \leq \eta \|\mathbf{r}_0\|, \quad \eta \approx 10^{-4}$$

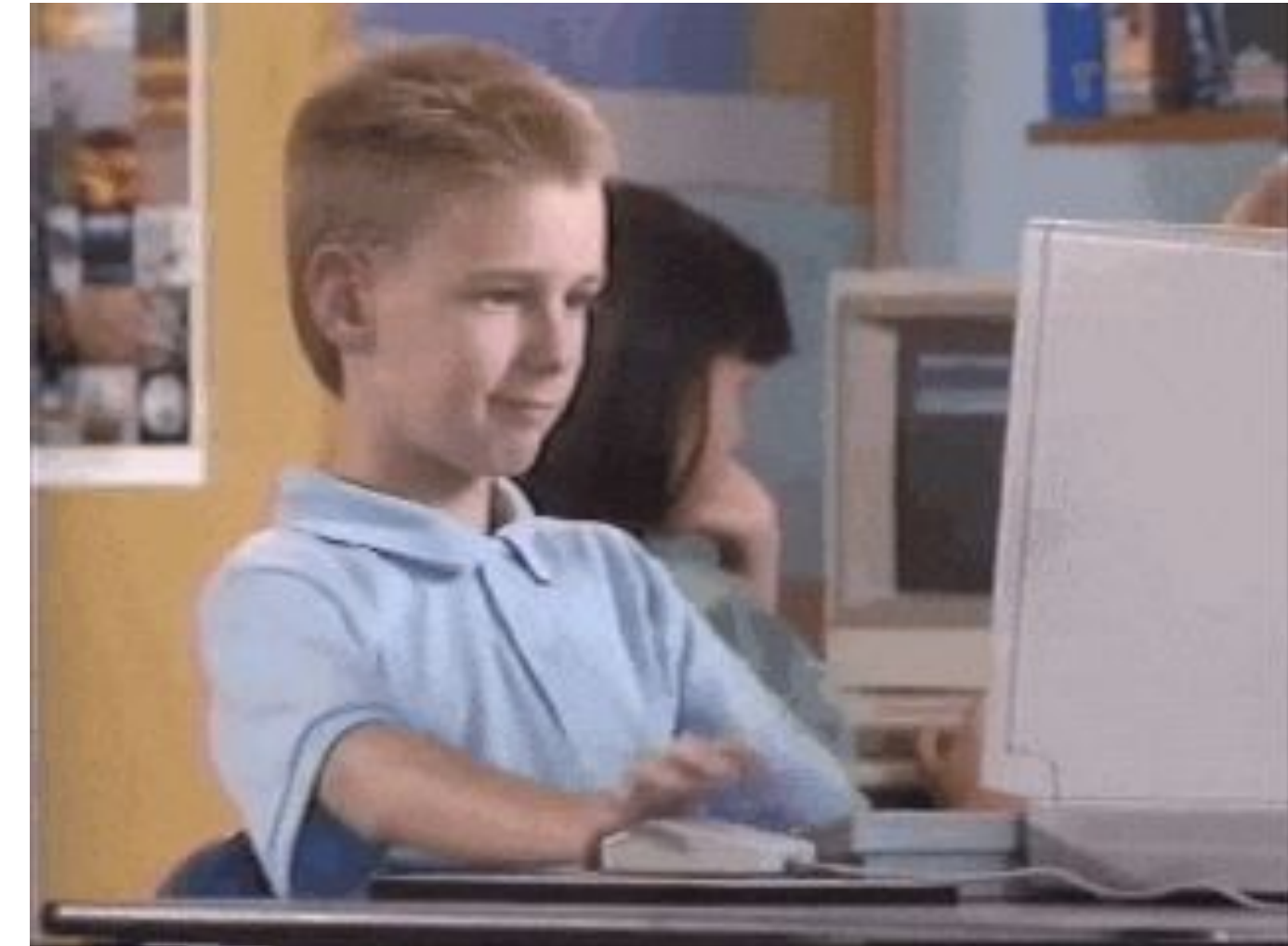
Newton's Procedure

Algorithm 1 Newton's method for optimization of the Backward Euler incremental potential.

- 1: $\mathbf{x} \leftarrow \tilde{\mathbf{x}} = \mathbf{x}^{\text{prev}} + \Delta t \mathbf{v}^{\text{prev}}$
 - 2: **while** not converged **do**
 - 3: assemble gradient $\mathbf{g}(\mathbf{x})$ and Hessian $\mathbf{H}(\mathbf{x})$
 - 4: $\hat{\mathbf{H}} \leftarrow \text{project}(\mathbf{H})$
 - 5: solve $\hat{\mathbf{H}} \Delta \mathbf{x} = -\mathbf{g}$
 - 6: find step size α by line search
 - 7: $\mathbf{x} \leftarrow \mathbf{x} + \alpha \Delta \mathbf{x}$
 - 8: **end while**
 - 9: $\mathbf{v} \leftarrow (\mathbf{x} - \mathbf{x}^{\text{prev}}) / \Delta t$
-

Recap

1. Optimization Time Integration
2. Newton's Method
3. Global Derivatives and Assembly
4. Penalty and Barrier Constraints
5. Line Search
6. Descent Directions and Hessian Definiteness
7. Convergence and Inexactness



Later!

SymX: Symbolic Differentiation for Nonlinear Optimization



README Apache-2.0 license

SymX

{SymX}

Symbolic differentiation. C++ code generation. JIT compilation. Global assembly. Non-linear optimization.
[Docs](#) · [PDF](#) · [ACM Page](#)

SymX is a C++ library for **symbolic differentiation** with automatic **code generation, compilation** and **evaluation**. Write complex mathematical expressions concisely, differentiate them arbitrarily, and let SymX evaluate them on your data structures — including global gradient and Hessian assembly.

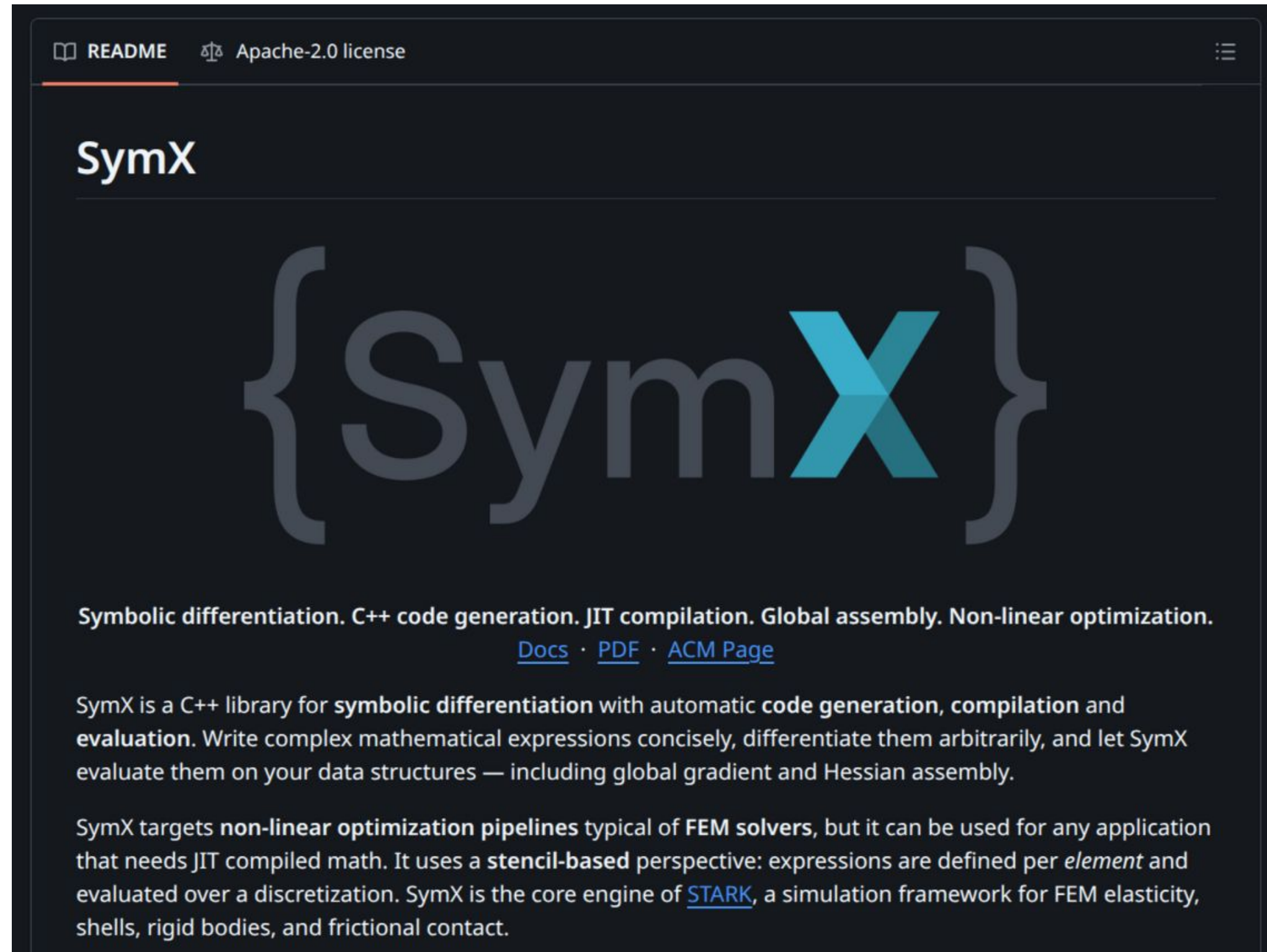
SymX targets **non-linear optimization pipelines** typical of **FEM solvers**, but it can be used for any application that needs JIT compiled math. It uses a **stencil-based** perspective: expressions are defined per *element* and evaluated over a discretization. SymX is the core engine of [STARK](#), a simulation framework for FEM elasticity, shells, rigid bodies, and frictional contact.



Any questions?

SymX Examples

Where to find it



README Apache-2.0 license

SymX



Symbolic differentiation. C++ code generation. JIT compilation. Global assembly. Non-linear optimization.
[Docs](#) · [PDF](#) · [ACM Page](#)

SymX is a C++ library for **symbolic differentiation** with automatic **code generation**, **compilation** and **evaluation**. Write complex mathematical expressions concisely, differentiate them arbitrarily, and let SymX evaluate them on your data structures — including global gradient and Hessian assembly.

SymX targets **non-linear optimization pipelines** typical of **FEM solvers**, but it can be used for any application that needs JIT compiled math. It uses a **stencil-based** perspective: expressions are defined per *element* and evaluated over a discretization. SymX is the core engine of [STARK](#), a simulation framework for FEM elasticity, shells, rigid bodies, and frictional contact.

github.com/InteractiveComputerGraphics/SymX



SymX Documentation



Welcome to the SymX documentation pages. SymX is a C++ library for **symbolic differentiation**, **code generation**, and **evaluation**, designed primarily for non-linear optimization in physics simulation and FEM. Check out the [SymX GitHub repo](#).

In these pages you will find an overview on how SymX works and how to use it through its levels of abstraction. Every explanation comes with example snippets. Note that this is more of a tutorial or a walkthrough than an extensive documentation in a traditional *Doxygen* sense.

Another important goal of this guide is to help you assess whether SymX can be useful to you. There are many different ways to use symbolics, differentiation and just-in-time compilation, and SymX supports a few use cases out-of-the-box. There are three entry points available in SymX:

- Single Expression:** Compose a symbolic expression (possibly differentiated), compile it, set values, run it, read the output.
- Stencil-based Loops:** Compose a symbolic expression (possibly differentiated) to be evaluated in a loop over many instances of a stencil. This is the core of FEM assemblers and iterative predictor/corrector solvers such as Jacobi or Gauss-Seidel.
- Non-linear Optimization:** Define potentials + discretizations + degrees of freedom and let SymX find the global solution using Newton's Method. It will take care of differentiation, compilation and evaluation automatically.

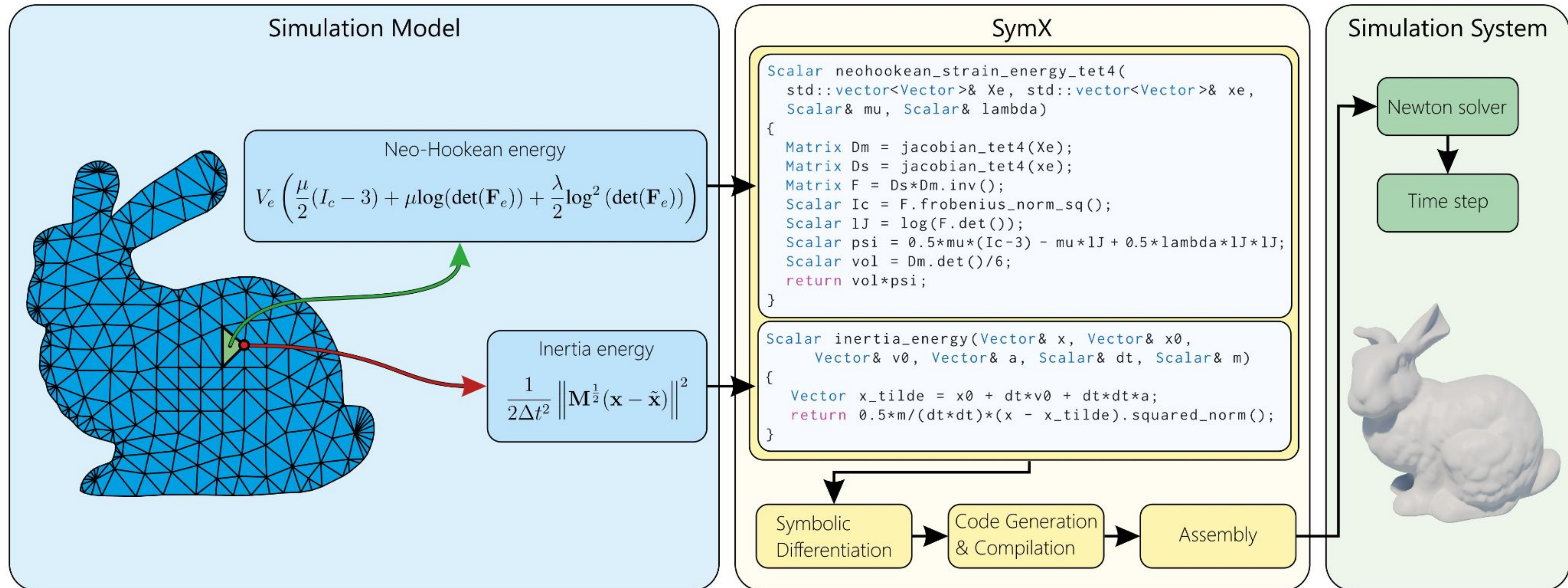
Navigation menu:

- SymX docs
- Search
- GETTING STARTED
 - Hello World
 - Setup
 - Architecture Overview
- CORE CONCEPTS (LAYERS 1, 2)
 - Symbols
 - Symbol Reference API
 - Compilation
- SYMBOL-DATA MAPS (LAYER 3)
 - Symbol-Data Overview
 - Symbol-Data Maps
 - Loops
- SECOND-ORDER SOLVER (LAYER 4)
 - Second-Order Optimization
 - FEM Integration
 - Newton's Method

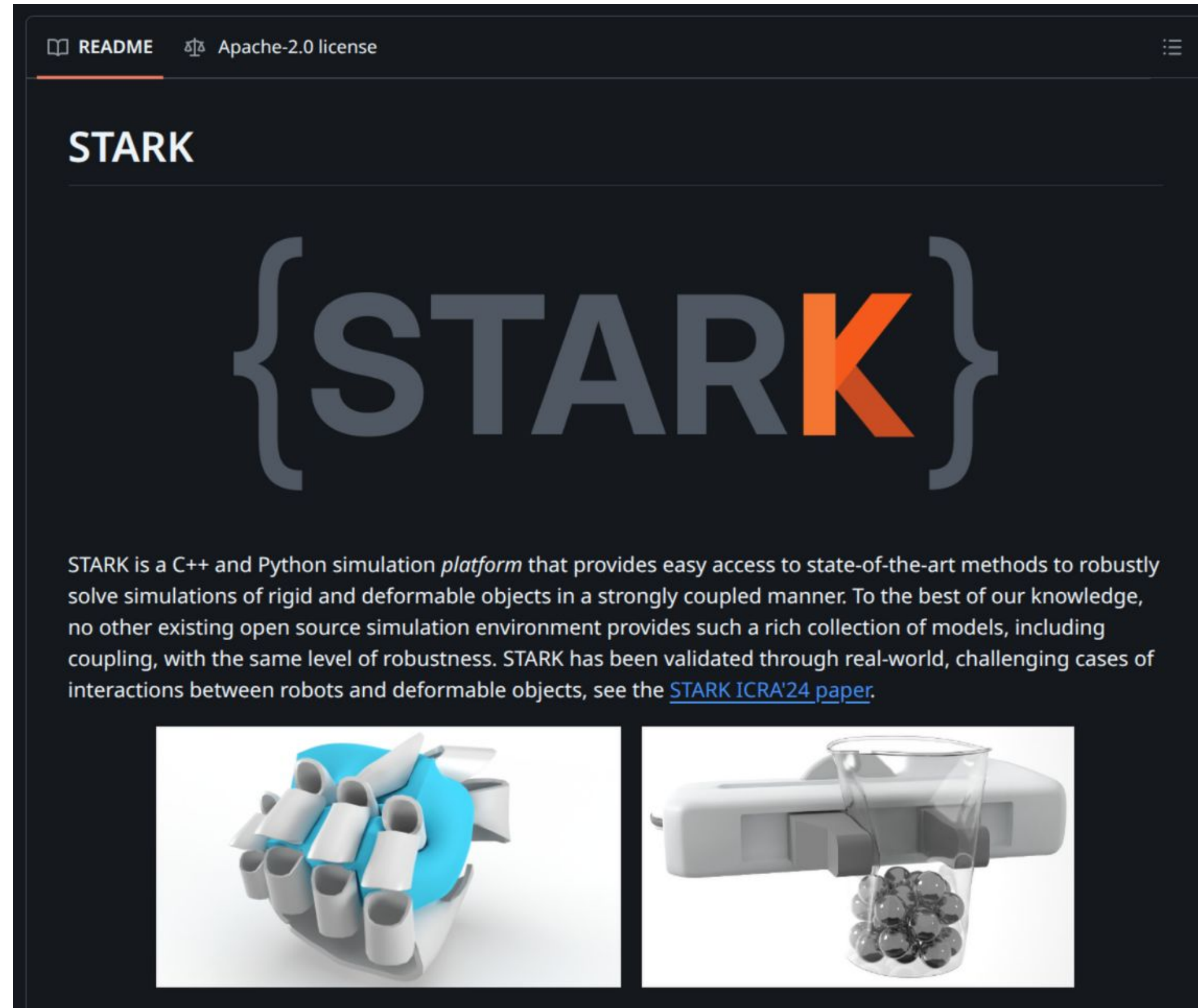
<https://symx.physics-simulation.org/>

What is SymX?

Symbolic Math Engine





STARK



The screenshot shows the GitHub repository page for STARK. At the top, it indicates the license as Apache-2.0. The main heading is "STARK" in a large, stylized font where the "K" is orange and the rest is grey, enclosed in large curly braces. Below this, a paragraph describes STARK as a C++ and Python simulation platform for rigid and deformable objects. At the bottom, there are two 3D simulation images: one showing a blue sphere being held by a white robotic hand, and another showing a white robotic hand holding a clear glass filled with small grey spheres.

STARK

STARK is a C++ and Python simulation *platform* that provides easy access to state-of-the-art methods to robustly solve simulations of rigid and deformable objects in a strongly coupled manner. To the best of our knowledge, no other existing open source simulation environment provides such a rich collection of models, including coupling, with the same level of robustness. STARK has been validated through real-world, challenging cases of interactions between robots and deformable objects, see the [STARK ICRA'24 paper](#).



Simulation Platform

- Model repository
 - Deformables
 - Rigid Bodies
 - Frictional Contact
- Scripting
- Python API

Example: Cloth Simulation

