COL865: Special Topics in Computer Applications

Physics-Based Animation

6 — Equation solving and optimization

Today's agenda

Solving linear systems

- Gauss-Seidel, Jacobi, conjugate gradient, direct methods

Solving nonlinear systems

- Application to backward Euler

Numerical optimization

- Gradient descent, Newton's method, application to quasistatics and dynamics

Motivation

To do implicit time integration, need to solve a system of equations for the unknown $\mathbf{y}^1$

- Backward Euler: $\mathbf{y}^1 = \mathbf{y}^0 + \boldsymbol{\varphi}(t^1, \mathbf{y}^1)\,\Delta t$
- Implicit midpoint: $\mathbf{y}^1 = \mathbf{y}^0 + \boldsymbol{\varphi}(\frac{1}{2}(t^0+t^1), \frac{1}{2}(\mathbf{y}^0+\mathbf{y}^1))\,\Delta t$
- Trapezoidal method: $\mathbf{y}^1 = \mathbf{y}^0 + \frac{1}{2}(\boldsymbol{\varphi}(t^0,\mathbf{y}^0) + \boldsymbol{\varphi}(t^1,\mathbf{y}^1))\,\Delta t$

In general, solve a system of (possibly nonlinear) equations $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ for $\mathbf{x}$

Let's consider the linear case first

Solving linear systems

Linear systems

$$\mathbf{Ax} = \mathbf{b} \quad \Rightarrow \quad \mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$$

Don’t actually compute $\mathbf{A}^{-1}$ and multiply by **b**!
Call linear solver on **A** and **b**

- Faster, less memory, more accurate

What linear solver to use? Depends on properties of **A**

In graphics, **A** is usually:

- quite large
- very sparse
- usually symmetric positive definite

Linear solvers

Common choices:

- ***Gauss-Seidel*** and ***Jacobi*** iterations
- Preconditioned ***conjugate gradient*** method
- ***Direct solvers*** using sparse factorization

Gauss-Seidel and Jacobi iterations

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

Solve ith row for variable x_i, keeping others fixed

$$a_{i1}\,x_1 + a_{i2}\,x_2 + \cdots + a_{ii}\,x_i + \cdots + a_{in}\,x_n = b_i$$

$$\Rightarrow x_i = a_{ii}^{-1}\,(b_i - a_{i1}\,x_1 - a_{i2}\,x_2 - \cdots\ \cdots - a_{in}\,x_n)$$

- sequentially for one i at a time: ***Gauss-Seidel***
- in parallel for all i independently: ***Jacobi***

Straightforward extension to ***blockwise*** GS/Jacobi

Gauss-Seidel and Jacobi iterations

Convergence:

- GS and Jacobi converge for diagonally dominant matrices, GS also converges for s.p.d. matrices
- Convergence is slow, but iterations are cheap
- GS typically converges faster, because x_i update can use new values of $x_1, x_2, \ldots, x_{i-1}$

Parallelizability:

- Jacobi is trivially parallelizable
- GS can be parallelized for sparse matrices by careful choice of ordering (graph colouring)

Conjugate gradient method

$$\mathbf{Ax} = \mathbf{b}$$

Idea: Solve the problem in the Krylov subspaces: span {**b**}, span {**b**, **Ab**}, span {**b**, **Ab**, $\mathbf{A}^2\mathbf{b}$}, …

- Assume $\mathbf{x} = \alpha_0\,\mathbf{b}$, compute best solution for α_0
- Assume $\mathbf{x} = \alpha_0\,\mathbf{b} + \alpha_1\,\mathbf{Ab}$, compute best α_0, α_1
- Assume $\mathbf{x} = \alpha_0\,\mathbf{b} + \alpha_1\,\mathbf{Ab} + \alpha_2\,\mathbf{A}^2\mathbf{b}$, compute best α_0, α_1, α_2
- …

Magic: We can do this easily, and cheaply, and without keeping track of **b**, **Ab**, $\mathbf{A}^2\mathbf{b}$, … !

$\mathbf{r}_0 := \mathbf{b} - \mathbf{A}\mathbf{x}_0$

$\mathbf{p}_0 := \mathbf{r}_0$

$k := 0$

repeat

$$\alpha_k := \frac{\mathbf{r}_k^\mathsf{T}\mathbf{r}_k}{\mathbf{p}_k^\mathsf{T}\mathbf{A}\mathbf{p}_k}$$

$\mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k\mathbf{p}_k$

$\mathbf{r}_{k+1} := \mathbf{r}_k - \alpha_k\mathbf{A}\mathbf{p}_k$

if r_{k+1} is sufficiently small, then exit loop

$$\beta_k := \frac{\mathbf{r}_{k+1}^\mathsf{T}\mathbf{r}_{k+1}}{\mathbf{r}_k^\mathsf{T}\mathbf{r}_k}$$

$\mathbf{p}_{k+1} := \mathbf{r}_{k+1} + \beta_k\mathbf{p}_k$

$k := k + 1$

end repeat

The result is $\mathbf{x}_{k+1}$

Conjugate gradient

How do we get from there to here? See Shewchuk, “An Introduction to the Conjugate Gradient Method Without the Agonizing Pain”, 1994

Algorithm uses **A** only to compute matrix-vector products

- Don’t even need **A** as a matrix, just a function `multByA` : $\mathbb{R}^n \to \mathbb{R}^n$

Conjugate gradient method

Usually a good default choice

Convergence:

- CG converges for s.p.d. matrices, much faster than GS/Jacobi
- Converges even faster with a ***preconditioner***: a matrix **M** which approximates $\mathbf{A}^{-1}$ [Shewchuk, Ch. 12]

Direct solvers

1. Compute factorization of **A**, e.g. Cholesky: $\mathbf{A} = \mathbf{L}\mathbf{L}^T$
2. Obtain ***exact*** solution of $\mathbf{A}\mathbf{x} = \mathbf{b}$ quickly via backsubstitution

Issues:

- Computing factorization is still expensive! Only useful if **A** is the same for every time step
- Factors of sparse matrix can be dense! Use a sparse matrix library, e.g. PARDISO, Eigen

Solving nonlinear systems

Nonlinear Gauss-Seidel and Jacobi

$$\mathbf{f}(\mathbf{x}) = \mathbf{0}$$

$$\begin{aligned} f_1(x_1, x_2, \ldots, x_n) &= 0, \\ f_2(x_1, x_2, \ldots, x_n) &= 0, \\ &\vdots \\ f_n(x_1, x_2, \ldots, x_n) &= 0 \end{aligned}$$

Same principle as GS/Jacobi: solve one equation for one variable, holding others fixed

- Makes sense if *i*th equation primarily involves *i*th variable
- Hard to prove convergence guarantees, but usually works

Newton's method

$$\mathbf{f}(\mathbf{x}) = \mathbf{0}$$

1. Pick initial guess $\mathbf{x}_0$
2. Linearize $\mathbf{f}$ about $\mathbf{x}_0$ via first-order Taylor series:

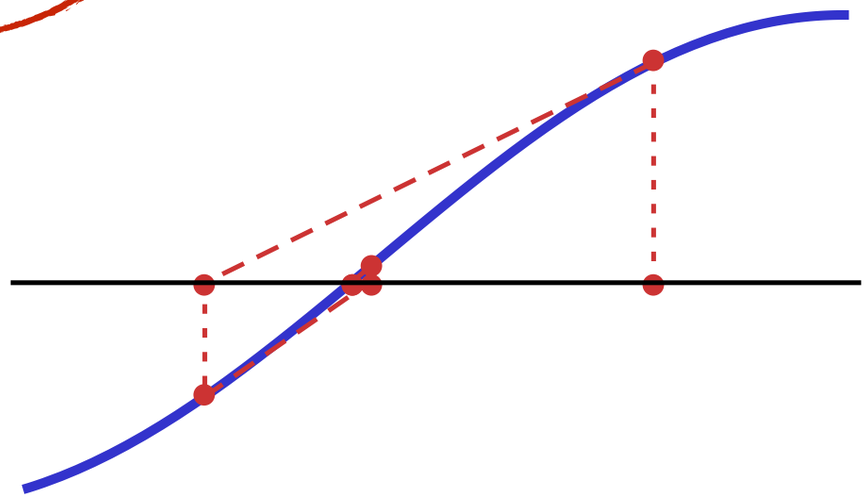
$$\mathbf{f}(\mathbf{x}_0+\Delta\mathbf{x}) \approx \mathbf{f}(\mathbf{x}_0) + \mathbf{J}(\mathbf{x}_0)\,\Delta\mathbf{x}$$

3. Set linearization = $\mathbf{0}$, solve for $\Delta\mathbf{x}$:

$$\Delta\mathbf{x} = -\mathbf{J}(\mathbf{x}_0)^{-1}\,\mathbf{f}(\mathbf{x}_0)$$

Compute using linear solvers discussed earlier

4. Update guess $\mathbf{x}_1 = \mathbf{x}_0 + \Delta\mathbf{x}$, repeat



Newton's method

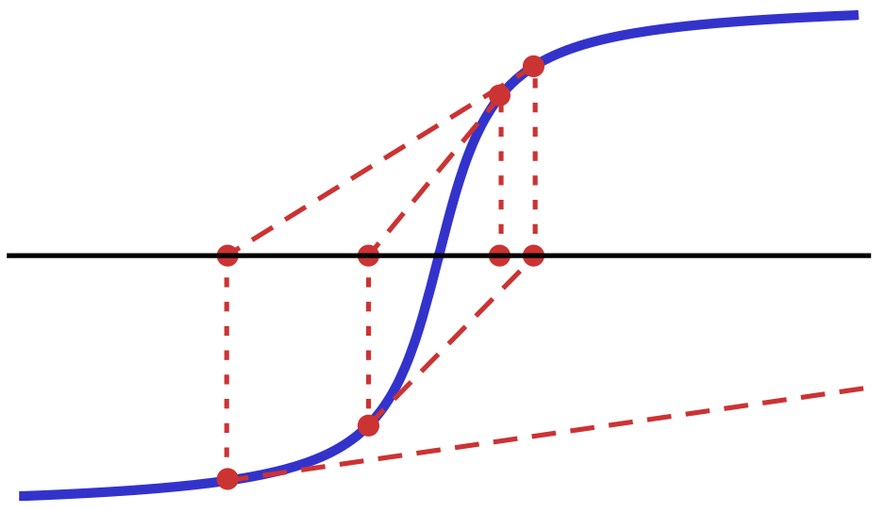
Convergence:

- Error decreases ***quadratically*** once close to solution:

$$\|\mathbf{x}_{k+1} - \mathbf{x}^*\| = O(\|\mathbf{x}_k - \mathbf{x}^*\|^2)$$

as long as $\mathbf{J}(\mathbf{x}^*)$ is nonsingular

- Error may not decrease when not close to solution!



Newton's method for backward Euler

Solve $\mathbf{y}^1 = \mathbf{y}^0 + \boldsymbol{\varphi}(t^1, \mathbf{y}^1)\,\Delta t$ for $\mathbf{y}^1$

1. Pick initial guess $\tilde{\mathbf{y}}$, e.g. $\tilde{\mathbf{y}} = \mathbf{y}^0$
2. Linearize equation about $\mathbf{y}^1 = \tilde{\mathbf{y}} + \Delta\mathbf{y}$, solve, repeat

$$(\tilde{\mathbf{y}} + \Delta\mathbf{y}) = \mathbf{y}^0 + (\boldsymbol{\varphi}(t^1, \tilde{\mathbf{y}}) + \mathbf{J}(t^1, \tilde{\mathbf{y}})\,\Delta\mathbf{y})\,\Delta t$$

$$\Rightarrow (\mathbf{I} - \mathbf{J}(t^1, \tilde{\mathbf{y}})\,\Delta t)\,\Delta\mathbf{y} = \mathbf{y}^0 + \boldsymbol{\varphi}(t^1, \tilde{\mathbf{y}})\,\Delta t - \tilde{\mathbf{y}}$$

In practice, one iteration is often enough

Newton's method for backward Euler

For dynamics equations, this can be reduced a bit.

$$\mathbf{x}^1 = \mathbf{x}^0 + \mathbf{v}^1 \, \Delta t$$
$$\mathbf{v}^1 = \mathbf{v}^0 + \mathbf{M}^{-1} \, \mathbf{f}(\mathbf{x}^1, \mathbf{v}^1) \, \Delta t$$

Consider initial guess $(\mathbf{x}^0, \mathbf{v}^0)$, so $\Delta\mathbf{x} = \mathbf{x}^1 - \mathbf{x}^0$, $\Delta\mathbf{v} = \mathbf{v}^1 - \mathbf{v}^0$

$$\mathbf{f}^0 = \mathbf{f}(\mathbf{x}^0, \mathbf{v}^0)$$
$$\mathbf{f}^1 = \mathbf{f}(\mathbf{x}^1, \mathbf{v}^1) \approx \mathbf{f}^0 + \mathbf{J}_\mathbf{x} \, \Delta\mathbf{x} + \mathbf{J}_\mathbf{v} \, \Delta\mathbf{v}$$

where $\mathbf{J}_\mathbf{x} = d\mathbf{f}/d\mathbf{x}$, $\mathbf{J}_\mathbf{v} = d\mathbf{f}/d\mathbf{v}$ are the ***Jacobians*** of the forces with respect to positions and velocities.

Simplify…

$$(\mathbf{M} - \mathbf{J}_\mathbf{x} \, \Delta t^2 - \mathbf{J}_\mathbf{v} \, \Delta t) \, \Delta\mathbf{v} = (\mathbf{f}^0 + \mathbf{J}_\mathbf{x} \, \mathbf{v}^0 \, \Delta t) \, \Delta t$$
$$\Delta\mathbf{x} = (\mathbf{v}^0 + \Delta\mathbf{v}) \, \Delta t$$

Newton's method for backward Euler

$$(\mathbf{M} - \mathbf{J_x}\,\Delta t^2 - \mathbf{J_v}\,\Delta t)\,\Delta\mathbf{v} = (\mathbf{f}^0 + \mathbf{J_x}\,\mathbf{v}^0\,\Delta t)\,\Delta t$$

Backward Euler via Newton's method:

1. Compute current force Jacobians $\mathbf{J_x}$ and $\mathbf{J_v}$ and form LHS matrix $(\mathbf{M} - \mathbf{J_x}\,\Delta t^2 - \mathbf{J_v}\,\Delta t)$, ***or*** implement a function that computes $(\mathbf{M} - \mathbf{J_x}\,\Delta t^2 - \mathbf{J_v}\,\Delta t)\,\Delta\mathbf{v}$ given $\Delta\mathbf{v}$
2. Compute RHS vector, do linear solve to get velocity update $\Delta\mathbf{v}$
3. Update $\mathbf{v}^1 = \mathbf{v}^0 + \Delta\mathbf{v}$, $\mathbf{x}^1 = \mathbf{x}^0 + (\mathbf{v}^0 + \Delta\mathbf{v})\,\Delta t$
4. Repeat if needed, linearizing $\mathbf{f}$ about new guess $(\mathbf{x}^1, \mathbf{v}^1)$

What happens if $\mathbf{J_x}$, $\mathbf{J_v}$ are inaccurate? e.g. What if $\mathbf{J_x} = \mathbf{0} = \mathbf{J_v}$?

Force Jacobians

In physics, $\mathbf{J_x}$ and $\mathbf{J_v}$ are almost always symmetric, often negative semidefinite

- ***Symmetry:*** If $\mathbf{f}(\mathbf{x}) = -\nabla U(\mathbf{x})$, then $\mathbf{J_x} = -\mathrm{Hess}(U)$
- ***Negative semidefiniteness:*** Moving particle in one direction usually creates restoring force in opposite direction

If so, LHS matrix ($\mathbf{M} - \mathbf{J_x}\,\Delta t^2 - \mathbf{J_v}\,\Delta t$) is s.p.d.
⇒ always has solution, can use conjugate gradient

If not negative semidefinite, can replace with semidefinite approximation

Example: spring Jacobians

$$\mathbf{f}_{ij} = -k_s \left(\|\mathbf{x}_i - \mathbf{x}_j\| - \ell_0 \right) \hat{\mathbf{d}}_{ij} - k_d \left((\mathbf{v}_i - \mathbf{v}_j) \cdot \hat{\mathbf{d}}_{ij} \right) \hat{\mathbf{d}}_{ij}$$

$$\mathbf{f}_{ji} = -\mathbf{f}_{ij}$$

where $\hat{\mathbf{d}}_{ij} = (\mathbf{x}_i - \mathbf{x}_j)/\|\mathbf{x}_i - \mathbf{x}_j\|$

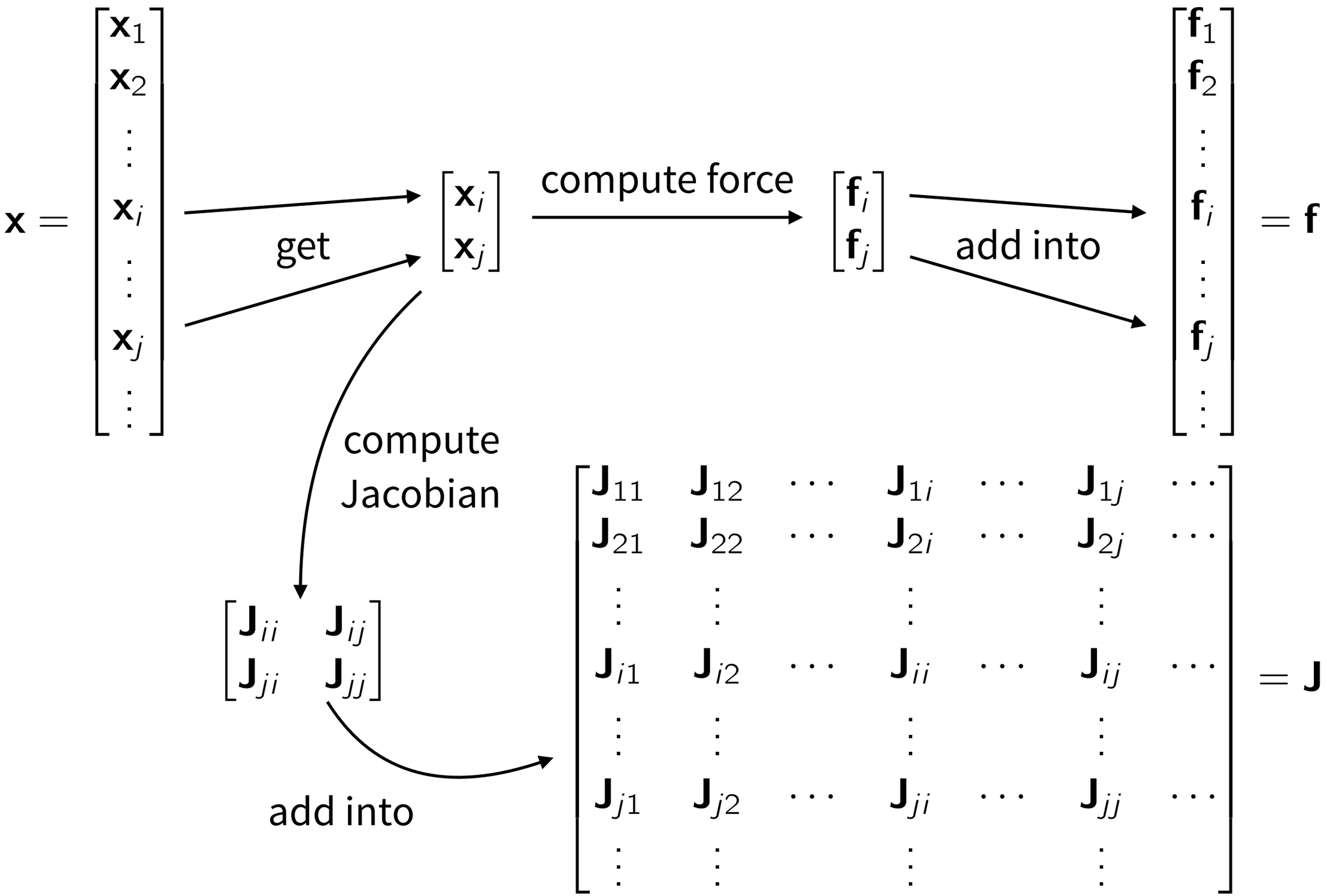
$$\frac{\partial \mathbf{f}_{ij}}{\partial \mathbf{x}_i} = -k_s \left(\left(1 - \frac{\ell_0}{\|\mathbf{x}_i - \mathbf{x}_j\|} \right) (\mathbf{I} - \hat{\mathbf{d}}_{ij} \hat{\mathbf{d}}_{ij}^T) + \hat{\mathbf{d}}_{ij} \hat{\mathbf{d}}_{ij}^T \right) + k_d(\ldots)$$

$$\frac{\partial \mathbf{f}_{ij}}{\partial \mathbf{v}_i} = -k_d \hat{\mathbf{d}}_{ij} \hat{\mathbf{d}}_{ij}^T$$

Set to zero if negative, for definiteness

safe to ignore

Example: matrix assembly for springs



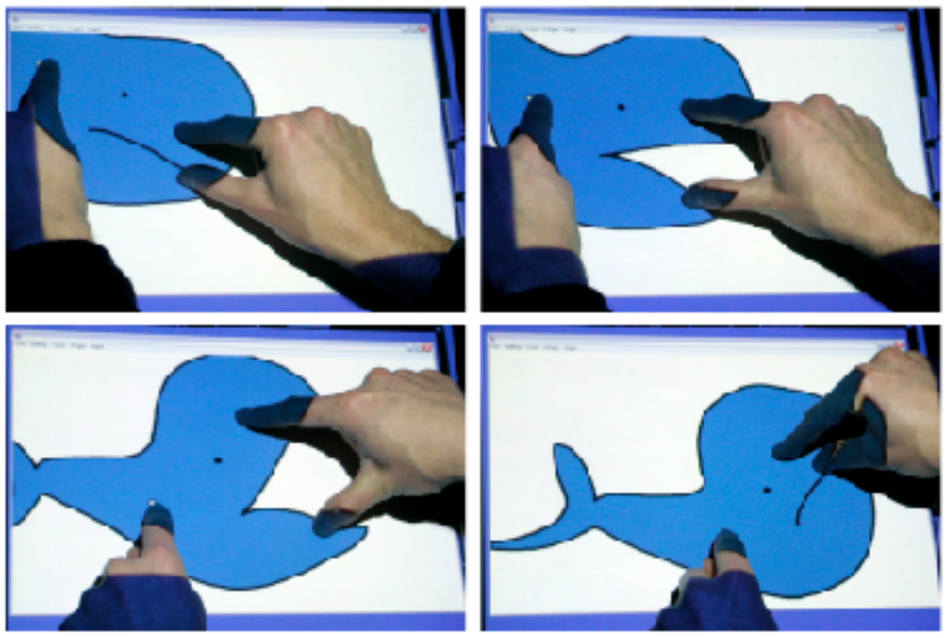
Numerical optimization

Optimization

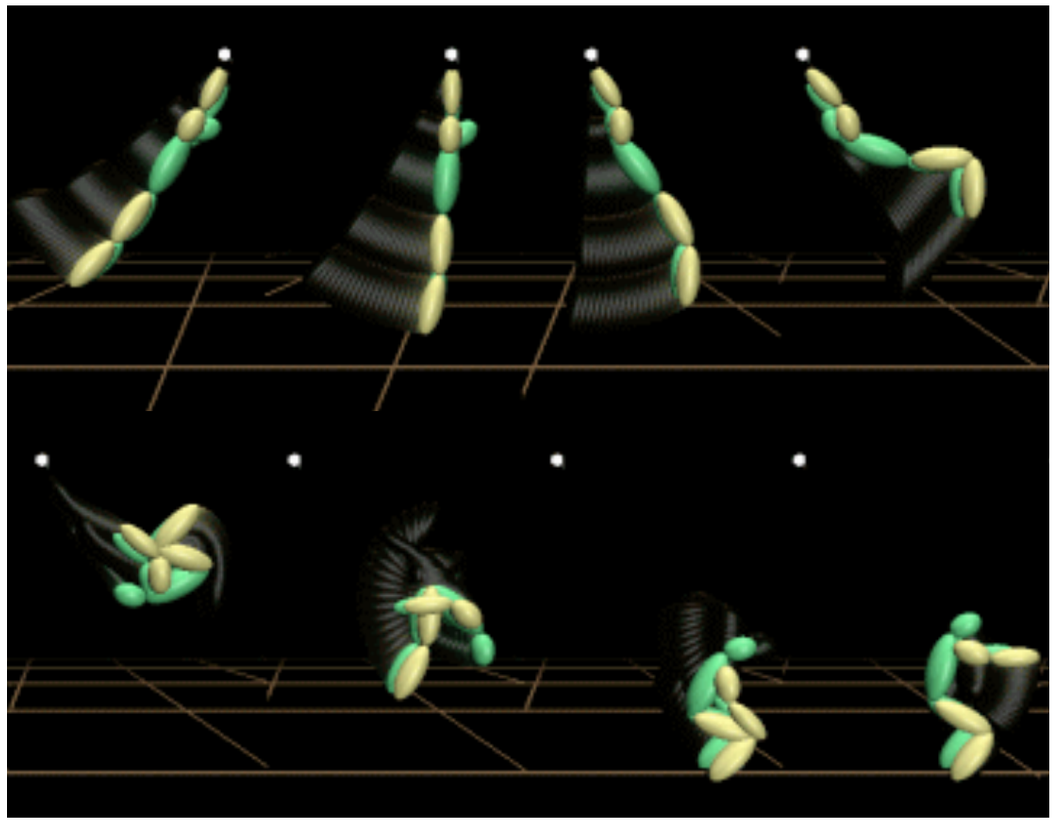
Equation solving: Given $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n$, find $\mathbf{x}$ which satisfies $\mathbf{f}(\mathbf{x}) = \mathbf{0}$

Optimization: Given $f : \mathbb{R}^n \to \mathbb{R}$, find $\mathbf{x}$ which minimizes $f(\mathbf{x})$

Applications in graphics:



[Igarashi et al. 2005]



[Fang and Pollard 2003]

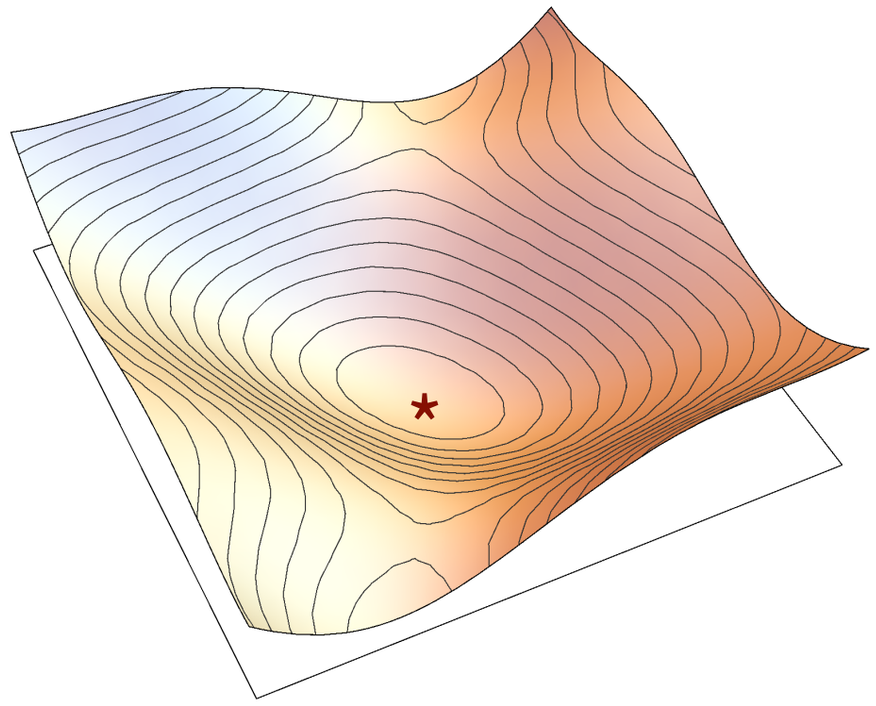
Optimization: basic theory

Global minimum:

$$f(\mathbf{x}^*) \leq f(\mathbf{x}) \text{ for } \textit{all } \mathbf{x}$$

Local minimum:

$$f(\mathbf{x}^*) \leq f(\mathbf{x}) \text{ for all } \mathbf{x} \textit{ near } \mathbf{x}^*$$



If f is differentiable, then $\nabla f(\mathbf{x}^*) = \mathbf{0}$ at local minimum $\mathbf{x}^*$

If f is twice differentiable, $\nabla f(\mathbf{x}) = \mathbf{0}$, and $\mathbf{H}(\mathbf{x})$ is positive definite, then $\mathbf{x}$ is a local minimum

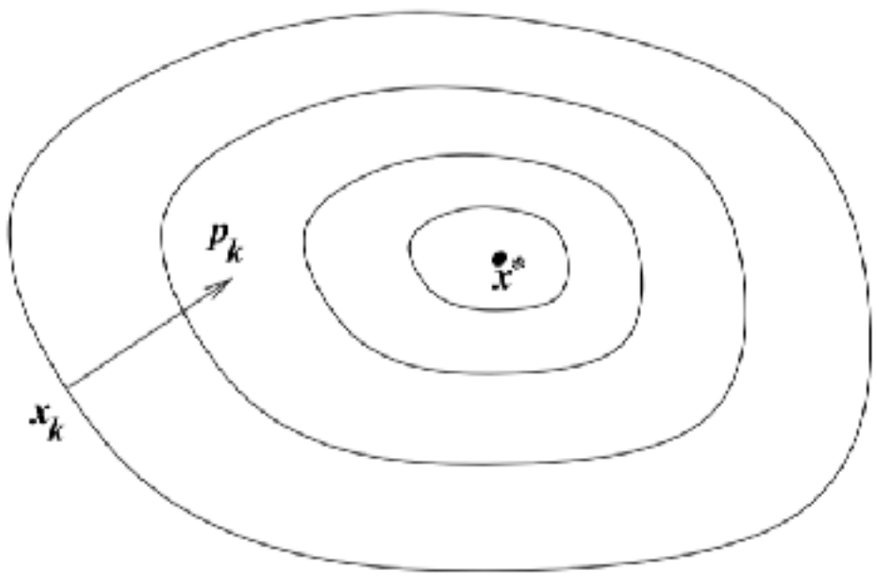
Descent methods

Basic approach:

- Choose some initial guess $\mathbf{x}_0$
- Choose descent direction $\mathbf{p}$ which reduces f:

$$\mathbf{p} \cdot \nabla f(\mathbf{x}_0) < 0$$

- Update guess $\mathbf{x}_1 = \mathbf{x}_0 + \alpha\, \mathbf{p}$ for some step length α



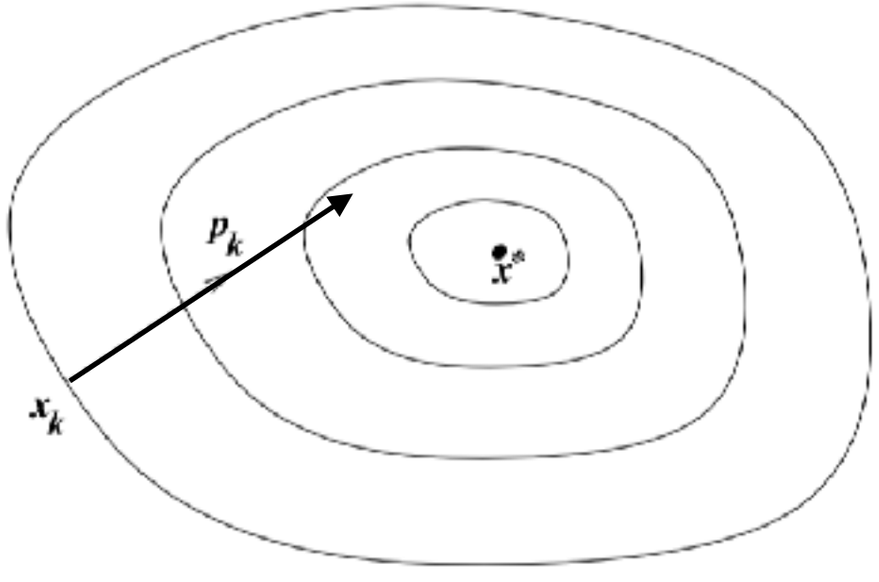
[Nocedal and Wright]

Two issues:

1. How to choose descent direction $\mathbf{p}$?
2. How to choose step length α?

Gradient descent

Easy choice: $\mathbf{p} = -\nabla f(\mathbf{x}_0)$

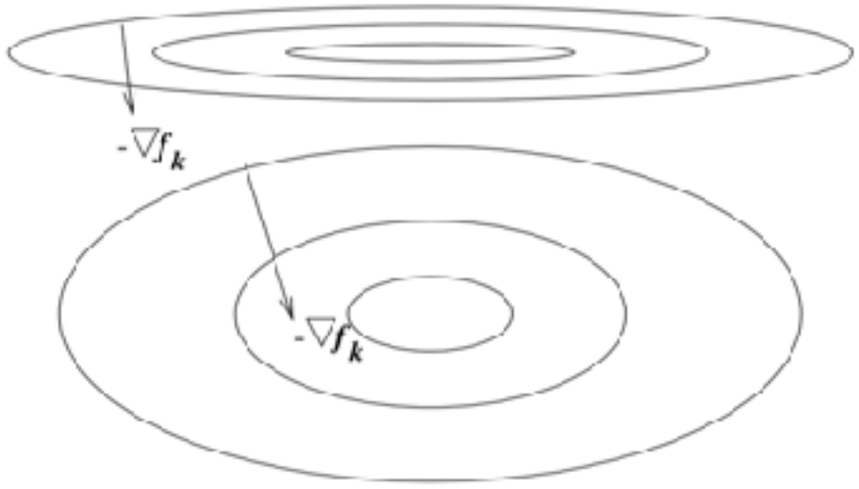


- Fixed α is suboptimal:
  too small → slow progress,
  too large → overshooting

- ***Exact line search***: find α which minimizes $f(\mathbf{x}_0 + \alpha\, \mathbf{p})$.

- ***Inexact line search***: find α which decreases f "enough"
  [Nocedal and Wright Ch. 3.1]

Gradient descent converges slowly
for problems that are "poorly scaled"



Newton's method for optimization

Build a quadratic approximation of f around $\mathbf{x}_0$

$$f(\mathbf{x}_0+\Delta\mathbf{x}) \approx f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0) \cdot \Delta\mathbf{x} + \tfrac{1}{2}\, \Delta\mathbf{x}^T\, \mathbf{H}(\mathbf{x}_0)\, \Delta\mathbf{x}$$

Minimized at $\Delta\mathbf{x} = -\mathbf{H}(\mathbf{x}_0)^{-1}\, \nabla f(\mathbf{x}_0)$. Set $\mathbf{p} = \Delta\mathbf{x}$, do line search

- $\Delta\mathbf{x}$ is only guaranteed to be a descent direction if $\mathbf{H}$ is s.p.d.!
  If not, revert to gradient descent

Computing $\mathbf{H}$, solving linear system is expensive.

- ***Quasi-Newton methods*** (e.g. L-BFGS): Iteratively build approximation of $\mathbf{H}^{-1}$ using previous values of ∇f.

Application: Quasistatics

Consider a deformable object subject to externally imposed constraints. Find the resting configuration

Equilibrium **x** minimizes potential energy $U(\mathbf{x})$

[Smith et al. 2018]

Application: Optimization-based integration

Backward Euler:

$$\mathbf{x}^1 = \mathbf{x}^0 + \mathbf{v}^1 \, \Delta t$$
$$\mathbf{v}^1 = \mathbf{v}^0 + \mathbf{M}^{-1} \mathbf{f}(\mathbf{x}^1, \mathbf{v}^1) \, \Delta t$$

$$\Rightarrow \mathbf{M}(\mathbf{x}^1 - \underbrace{\mathbf{x}^0 - \mathbf{v}^0 \, \Delta t}_{\tilde{\mathbf{x}}}) = \mathbf{f}(\mathbf{x}^1, \mathbf{v}^1) \, \Delta t^2$$

[Martin et al. 2013]

If $\mathbf{f}(\mathbf{x}) = -U(\mathbf{x})$, then this is equivalent to

$$\min \tfrac{1}{2} (\mathbf{x}^1 - \tilde{\mathbf{x}})^\mathsf{T} \mathbf{M} (\mathbf{x}^1 - \tilde{\mathbf{x}}) + U(\mathbf{x}^1)$$

Next class: Paper discussions

1. Selle et al., “A Mass Spring Model for Hair Simulation”, 2008

2. Liu et al., “Fast Simulation of Mass-Spring Systems”, 2013

Lead: me

Your job:

- Read both papers before Thursday’s class
- Come prepared with questions, comments, ideas (at least one)