GAMES103: Intro to Physics-Based Animation

Linear Finite Element Method II

Huamin Wang

Dec 2021

Last Time…

A FEM/FVM Framework

$$\mathbf{D}_\mathrm{m} = [\mathbf{X}_{10} \quad \mathbf{X}_{20} \quad \mathbf{X}_{30}]$$

Deformation gradient →

$$\mathbf{F} = [\mathbf{x}_{10} \quad \mathbf{x}_{20} \quad \mathbf{x}_{30}]\mathbf{D}_\mathrm{m}^{-1}$$

Principal stretches $\boldsymbol{\Lambda}$ →

$$[\mathbf{U} \quad \boldsymbol{\Lambda} \quad \mathbf{V}^\mathrm{T}] = \mathrm{svd}(\mathbf{F})$$

Second PK Stress →

$$\mathbf{P} = \mathbf{U}\mathrm{diag}\left(\frac{\partial W}{\partial \lambda_0}, \frac{\partial W}{\partial \lambda_1}, \frac{\partial W}{\partial \lambda_2}\right)\mathbf{V}^\mathrm{T}$$

Forces →

$$[\mathbf{f}_1 \quad \mathbf{f}_2 \quad \mathbf{f}_3] = -\frac{1}{6\det(\mathbf{D}_\mathrm{m}^{-1})}\mathbf{P}\mathbf{D}_\mathrm{m}^{-\mathrm{T}}$$

$$\mathbf{f}_0 = -\mathbf{f}_1 - \mathbf{f}_2 - \mathbf{f}_3$$

Topics for the Day

- Hessian of Elastic Energy
- Implicit Integration
- Nonlinear optimization

Hessian of Elastic Energy

Recall that…

Forces:

$$[\mathbf{f}_1 \quad \mathbf{f}_2 \quad \mathbf{f}_3] = -V\mathbf{U}\mathrm{diag}\left(\frac{\partial W}{\partial \lambda_0}, \frac{\partial W}{\partial \lambda_1}, \frac{\partial W}{\partial \lambda_2}\right)\mathbf{V}^{\mathrm{T}}\mathbf{D}_{\mathrm{m}}^{-\mathrm{T}}$$

$$[\mathbf{f}_0 \quad \mathbf{f}_1 \quad \mathbf{f}_2 \quad \mathbf{f}_3\,] = -V\mathbf{U}\mathrm{diag}\left(\frac{\partial W}{\partial \lambda_0}, \frac{\partial W}{\partial \lambda_1}, \frac{\partial W}{\partial \lambda_2}\right)\mathbf{V}^{\mathrm{T}}[\mathbf{d}_0 \quad \mathbf{d}_1 \quad \mathbf{d}_2 \quad \mathbf{d}_3\,]$$

$$\mathbf{f}_i = -V\mathbf{U}\mathrm{diag}\left(\frac{\partial W}{\partial \lambda_0}, \frac{\partial W}{\partial \lambda_1}, \frac{\partial W}{\partial \lambda_2}\right)\mathbf{V}^{\mathrm{T}}\mathbf{d}_i$$

Energy Hessian:

$$\frac{\partial \mathbf{f}_i}{\partial \mathbf{x}_j} = \sum_{k,l} \frac{\partial \mathbf{f}_i}{\partial \mathbf{F}_{kl}} \frac{\partial \mathbf{F}_{kl}}{\partial \mathbf{x}_j} = \sum_{k,l} \frac{\partial \left(\mathbf{U}\mathrm{diag}\left(\frac{\partial W}{\partial \lambda_0}, \frac{\partial W}{\partial \lambda_1}, \frac{\partial W}{\partial \lambda_2}\right)\mathbf{V}^{\mathrm{T}}\right)}{\partial \mathbf{F}_{kl}} -V\mathbf{d}_i \frac{\partial \mathbf{F}_{kl}}{\partial \mathbf{x}_j}$$

constant

Energy Hessian

$$\frac{\partial\left(\mathbf{U}\mathrm{diag}\left(\frac{\partial W}{\partial\lambda_0},\frac{\partial W}{\partial\lambda_1},\frac{\partial W}{\partial\lambda_2}\right)\mathbf{V}^{\mathrm{T}}\right)}{\partial\mathbf{F}_{kl}}$$

$$=\frac{\partial\mathbf{U}}{\partial\mathbf{F}_{kl}}\mathrm{diag}\left(\frac{\partial W}{\partial\lambda_0},\frac{\partial W}{\partial\lambda_1},\frac{\partial W}{\partial\lambda_2}\right)\mathbf{V}^{\mathrm{T}}+\mathbf{U}\frac{\partial\left(\mathrm{diag}\left(\frac{\partial W}{\partial\lambda_0},\frac{\partial W}{\partial\lambda_1},\frac{\partial W}{\partial\lambda_2}\right)\right)}{\partial\mathbf{F}_{kl}}\mathbf{V}^{\mathrm{T}}+\mathbf{U}\mathrm{diag}\left(\frac{\partial W}{\partial\lambda_0},\frac{\partial W}{\partial\lambda_1},\frac{\partial W}{\partial\lambda_2}\right)\frac{\partial\mathbf{V}^{\mathrm{T}}}{\partial\mathbf{F}_{kl}}$$

$$=\frac{\partial\mathbf{U}}{\partial\mathbf{F}_{kl}}\mathrm{diag}\left(\frac{\partial W}{\partial\lambda_0},\frac{\partial W}{\partial\lambda_1},\frac{\partial W}{\partial\lambda_2}\right)\mathbf{V}^{\mathrm{T}}+\mathbf{U}\left(\sum_{d=0}^{2}\frac{\partial\left(\mathrm{diag}\left(\frac{\partial W}{\partial\lambda_0},\frac{\partial W}{\partial\lambda_1},\frac{\partial W}{\partial\lambda_2}\right)\right)}{\partial\lambda_d}\frac{\partial\lambda_d}{\partial\mathbf{F}_{kl}}\right)\mathbf{V}^{\mathrm{T}}+\mathbf{U}\mathrm{diag}\left(\frac{\partial W}{\partial\lambda_0},\frac{\partial W}{\partial\lambda_1},\frac{\partial W}{\partial\lambda_2}\right)\frac{\partial\mathbf{V}^{\mathrm{T}}}{\partial\mathbf{F}_{kl}}$$

known

Given by hyperelastic model

known

What we don't know: $\frac{\partial\mathbf{U}}{\partial\mathbf{F}_{kl}}, \frac{\partial\lambda_d}{\partial\mathbf{F}_{kl}}, \frac{\partial\mathbf{V}^{\mathrm{T}}}{\partial\mathbf{F}_{kl}}$.

SVD Derivative

Since $\mathbf{F} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{V}^{\mathrm{T}}$ is singular value decomposition ($\boldsymbol{\Lambda} = \mathrm{diag}(\lambda_0, \lambda_1, \lambda_2)$), we can do:

$$\mathbf{U}^{\mathrm{T}}\frac{\partial \mathbf{F}}{\partial \mathbf{F}_{kl}}\mathbf{V} = \mathbf{U}^{\mathrm{T}}\left(\frac{\partial \mathbf{U}}{\partial \mathbf{F}_{kl}}\boldsymbol{\Lambda}\mathbf{V}^{\mathrm{T}} + \mathbf{U}\frac{\partial \boldsymbol{\Lambda}}{\partial \mathbf{F}_{kl}}\mathbf{V}^{\mathrm{T}} + \mathbf{U}\boldsymbol{\Lambda}\frac{\partial \mathbf{V}^{\mathrm{T}}}{\partial \mathbf{F}_{kl}}\right)\mathbf{V}$$

$$\mathbf{U}^{\mathrm{T}}\frac{\partial \mathbf{F}}{\partial \mathbf{F}_{kl}}\mathbf{V} = \left(\mathbf{U}^{\mathrm{T}}\frac{\partial \mathbf{U}}{\partial \mathbf{F}_{kl}}\right)\boldsymbol{\Lambda} + \frac{\partial \boldsymbol{\Lambda}}{\partial \mathbf{F}_{kl}} + \boldsymbol{\Lambda}\left(\frac{\partial \mathbf{V}^{\mathrm{T}}}{\partial \mathbf{F}_{kl}}\mathbf{V}\right)$$

$$\mathbf{U}^{\mathrm{T}}\frac{\partial \mathbf{F}}{\partial \mathbf{F}_{kl}}\mathbf{V} = \mathbf{A}\boldsymbol{\Lambda} + \frac{\partial \boldsymbol{\Lambda}}{\partial \mathbf{F}_{kl}} + \boldsymbol{\Lambda}\mathbf{B}$$

Skew-Symmetric Matrix

Matrix $\mathbf{A}$ is skew-symmetric (or anti-symmetric), if $\mathbf{A} = -\mathbf{A}^{\mathrm{T}}$:

$$\mathbf{A} = \begin{bmatrix} 0 & a & b \\ -a & 0 & c \\ -b & -c & 0 \end{bmatrix}$$

If $\mathbf{D}$ is diagonal, then:

$$\mathbf{AD} = \begin{bmatrix} 0 & a & b \\ -a & 0 & c \\ -b & -c & 0 \end{bmatrix} \begin{bmatrix} d & 0 & 0 \\ 0 & e & 0 \\ 0 & 0 & f \end{bmatrix} = \begin{bmatrix} 0 & ? & ? \\ ? & 0 & ? \\ ? & ? & 0 \end{bmatrix} \qquad \mathbf{DA} = \begin{bmatrix} d & 0 & 0 \\ 0 & e & 0 \\ 0 & 0 & f \end{bmatrix} \begin{bmatrix} 0 & a & b \\ -a & 0 & c \\ -b & -c & 0 \end{bmatrix} = \begin{bmatrix} 0 & ? & ? \\ ? & 0 & ? \\ ? & ? & 0 \end{bmatrix}$$

When $\mathbf{U}$ is orthogonal, we have:

$$\mathbf{0} = \frac{\partial(\mathbf{U}^{\mathrm{T}}\mathbf{U})}{\partial \mathbf{F}_{kl}} = \mathbf{U}^{\mathrm{T}} \frac{\partial \mathbf{U}}{\partial \mathbf{F}_{kl}} + \frac{\partial \mathbf{U}^{\mathrm{T}}}{\partial \mathbf{F}_{kl}} \mathbf{U} = \mathbf{U}^{\mathrm{T}} \frac{\partial \mathbf{U}}{\partial \mathbf{F}_{kl}} + \left(\mathbf{U}^{\mathrm{T}} \frac{\partial \mathbf{U}}{\partial \mathbf{F}_{kl}} \right)^{\mathrm{T}}$$

Therefore, $\mathbf{A} = \mathbf{U}^{\mathrm{T}} \frac{\partial \mathbf{U}}{\partial \mathbf{F}_{kl}}$ is skew-symmetric. So is $\mathbf{B} = \frac{\partial \mathbf{V}^{\mathrm{T}}}{\partial \mathbf{F}_{kl}} \mathbf{V}$.

SVD Derivative (cont.)

Since $\mathbf{F} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{V}^{\mathrm{T}}$ is singular value decomposition ($\boldsymbol{\Lambda} = \mathrm{diag}(\lambda_0, \lambda_1, \lambda_2)$), we can do:

$$\mathbf{U}^{\mathrm{T}}\frac{\partial \mathbf{F}}{\partial \mathbf{F}_{kl}}\mathbf{V} = \mathbf{A}\boldsymbol{\Lambda} + \frac{\partial \boldsymbol{\Lambda}}{\partial \mathbf{F}_{kl}} + \boldsymbol{\Lambda}\mathbf{B}$$

After expansion, we get:

$$\mathbf{U}^{\mathrm{T}}\frac{\partial \mathbf{F}}{\partial \mathbf{F}_{kl}}\mathbf{V} = \begin{bmatrix} 0 & a_0 & a_1 \\ -a_0 & 0 & a_2 \\ -a_1 & -a_2 & 0 \end{bmatrix}\begin{bmatrix} \lambda_0 & & \\ & \lambda_1 & \\ & & \lambda_2 \end{bmatrix} + \begin{bmatrix} \frac{\partial \lambda_0}{\partial \mathbf{F}_{kl}} & & \\ & \frac{\partial \lambda_1}{\partial \mathbf{F}_{kl}} & \\ & & \frac{\partial \lambda_2}{\partial \mathbf{F}_{kl}} \end{bmatrix} + \begin{bmatrix} \lambda_0 & & \\ & \lambda_1 & \\ & & \lambda_2 \end{bmatrix}\begin{bmatrix} 0 & b_0 & b_1 \\ -b_0 & 0 & b_2 \\ -b_1 & -b_2 & 0 \end{bmatrix}$$

SVD Derivative (cont.)

Since $\mathbf{F} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{V}^{\mathrm{T}}$ is singular value decomposition ($\boldsymbol{\Lambda} = \mathrm{diag}(\lambda_0, \lambda_1, \lambda_2)$), we can do:

$$\mathbf{U}^{\mathrm{T}}\frac{\partial \mathbf{F}}{\partial \mathbf{F}_{kl}}\mathbf{V} = \mathbf{A}\boldsymbol{\Lambda} + \frac{\partial \boldsymbol{\Lambda}}{\partial \mathbf{F}_{kl}} + \boldsymbol{\Lambda}\mathbf{B}$$

After expansion, we get:

$$\mathbf{U}^{\mathrm{T}}\frac{\partial \mathbf{F}}{\partial \mathbf{F}_{kl}}\mathbf{V} = \begin{bmatrix} \partial\lambda_0/\partial\mathbf{F}_{kl} & \lambda_1 a_0 + \lambda_0 b_0 & \lambda_2 a_1 + \lambda_0 b_1 \\ -\lambda_0 a_0 - \lambda_1 b_0 & \partial\lambda_1/\partial\mathbf{F}_{kl} & \lambda_2 a_2 + \lambda_1 b_2 \\ -\lambda_0 a_1 - \lambda_2 b_1 & -\lambda_1 a_0 a_2 - \lambda_2 b_2 & \partial\lambda_2/\partial\mathbf{F}_{kl} \end{bmatrix}$$

$$\begin{bmatrix} m_{00} & m_{01} & m_{02} \\ m_{10} & m_{11} & m_{12} \\ m_{20} & m_{21} & m_{22} \end{bmatrix} = \begin{bmatrix} \partial\lambda_0/\partial\mathbf{F}_{kl} & \lambda_1 a_0 + \lambda_0 b_0 & \lambda_2 a_1 + \lambda_0 b_1 \\ -\lambda_0 a_0 - \lambda_1 b_0 & \partial\lambda_1/\partial\mathbf{F}_{kl} & \lambda_2 a_2 + \lambda_1 b_2 \\ -\lambda_0 a_1 - \lambda_2 b_1 & -\lambda_1 a_0 a_2 - \lambda_2 b_2 & \partial\lambda_2/\partial\mathbf{F}_{kl} \end{bmatrix}$$

Eventually, we get: $\mathbf{A} = \mathbf{U}^{\mathrm{T}}\frac{\partial \mathbf{U}}{\partial \mathbf{F}_{kl}}$, $\mathbf{B} = \frac{\partial \mathbf{V}^{\mathrm{T}}}{\partial \mathbf{F}_{kl}}\mathbf{V}$ and $\partial\lambda_0/\partial\mathbf{F}_{kl}$, $\partial\lambda_1/\partial\mathbf{F}_{kl}$, $\partial\lambda_2/\partial\mathbf{F}_{kl}$.

A Quick Summary

- Step 1: By SVD derivatives, we get: $\frac{\partial \mathbf{U}}{\partial \mathbf{F}_{kl}}, \frac{\partial \lambda_d}{\partial \mathbf{F}_{kl}}, \frac{\partial \mathbf{V}^{\mathrm{T}}}{\partial \mathbf{F}_{kl}}$.

- Step 2: we then compute $\frac{\partial\left(\mathbf{U}\mathrm{diag}\left(\frac{\partial W}{\partial \lambda_0},\frac{\partial W}{\partial \lambda_1},\frac{\partial W}{\partial \lambda_2}\right)\mathbf{V}^{\mathrm{T}}\right)}{\partial \mathbf{F}_{kl}}$:

$$= \frac{\partial \mathbf{U}}{\partial \mathbf{F}_{kl}} \mathrm{diag}\left(\frac{\partial W}{\partial \lambda_0},\frac{\partial W}{\partial \lambda_1},\frac{\partial W}{\partial \lambda_2}\right)\mathbf{V}^{\mathrm{T}} + \mathbf{U}\left(\sum_{d=0}^{2} \frac{\partial\left(\mathrm{diag}\left(\frac{\partial W}{\partial \lambda_0},\frac{\partial W}{\partial \lambda_1},\frac{\partial W}{\partial \lambda_2}\right)\right)}{\partial \lambda_d} \frac{\partial \lambda_d}{\partial \mathbf{F}_{kl}}\right)\mathbf{V}^{\mathrm{T}} + \mathbf{U}\mathrm{diag}\left(\frac{\partial W}{\partial \lambda_0},\frac{\partial W}{\partial \lambda_1},\frac{\partial W}{\partial \lambda_2}\right)\frac{\partial \mathbf{V}^{\mathrm{T}}}{\partial \mathbf{F}_{kl}}$$

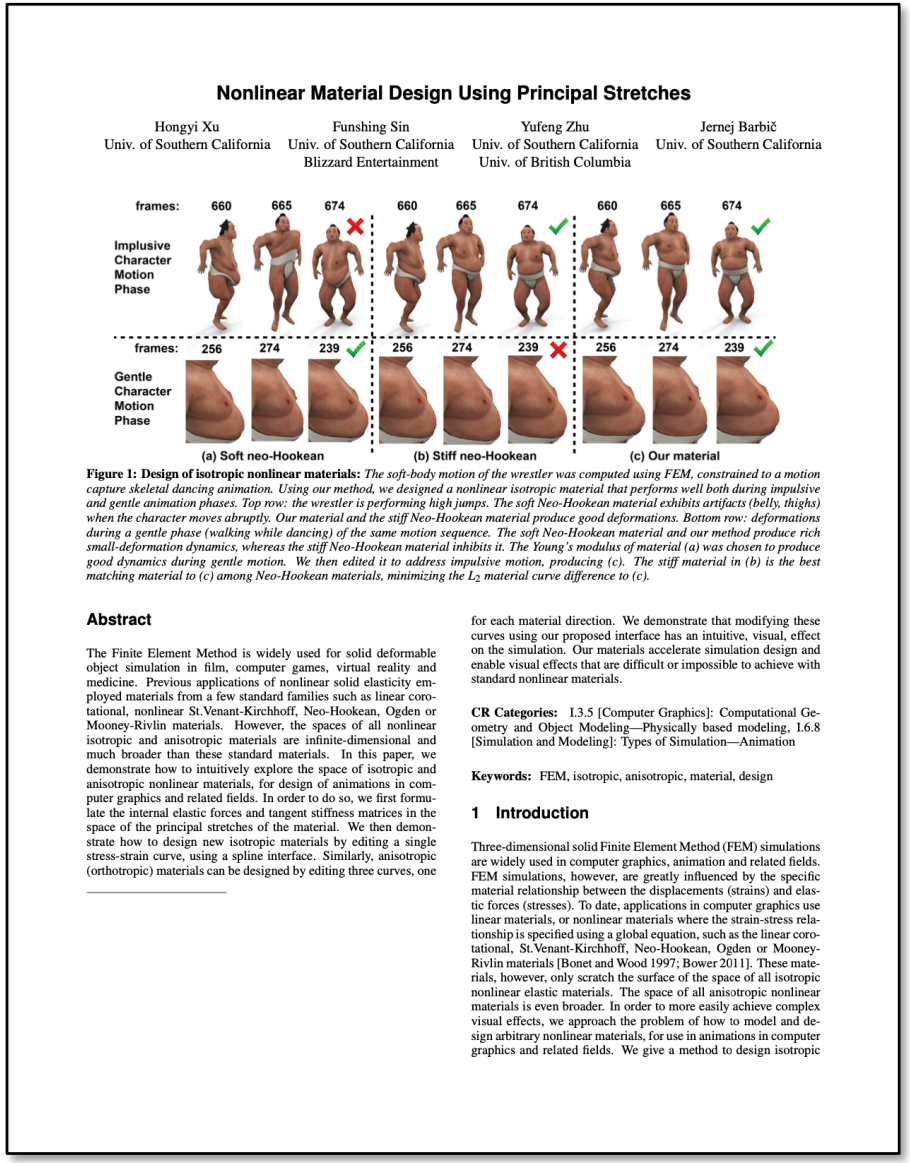
- Step 3: Finally, we reach our goal in Hessian matrix:

$$\frac{\partial \mathbf{f}_i}{\partial \mathbf{x}_j} = \sum_{k,l} \frac{\partial \mathbf{f}_i}{\partial \mathbf{F}_{kl}} \frac{\partial \mathbf{F}_{kl}}{\partial \mathbf{x}_j} = \sum_{k,l} \frac{\partial\left(\mathbf{U}\mathrm{diag}\left(\frac{\partial W}{\partial \lambda_0},\frac{\partial W}{\partial \lambda_1},\frac{\partial W}{\partial \lambda_2}\right)\mathbf{V}^{\mathrm{T}}\right)}{\partial \mathbf{F}_{kl}} - V\mathbf{d}_i \frac{\partial \mathbf{F}_{kl}}{\partial \mathbf{x}_j}$$

After-Class Reading

Xu et al. 2015. *Nonlinear Material Design Using Principal Stretches*. TOG (SIGGRAPH).

Definitely read this paper if you decide to implement it.



Nonlinear Material Design Using Principal Stretches

Hongyi Xu, Univ. of Southern California
Funshing Sin, Univ. of Southern California, Blizzard Entertainment
Yufeng Zhu, Univ. of Southern California, Univ. of British Columbia
Jernej Barbič, Univ. of Southern California

Figure 1: Design of isotropic nonlinear materials: *The soft-body motion of the wrestler was computed using FEM, constrained to a motion capture skeletal dancing animation. Using our method, we designed a nonlinear isotropic material that performs well both during impulsive and gentle animation phases. Top row: the wrestler is performing high jumps. The soft Neo-Hookean material exhibits artifacts (belly, thighs) when the character moves abruptly. Our material and the stiff Neo-Hookean material produce good deformations. Bottom row: deformations during a gentle phase (walking while dancing) of the same motion sequence. The soft Neo-Hookean material and our method produce rich small-deformation dynamics, whereas the stiff Neo-Hookean material inhibits it. The Young's modulus of material (a) was chosen to produce good dynamics during gentle motion. We then edited it to address impulsive motion, producing (c). The stiff material in (b) is the best matching material to (c) among Neo-Hookean materials, minimizing the L_2 material curve difference to (c).*

Abstract

The Finite Element Method is widely used for solid deformable object simulation in film, computer games, virtual reality and medicine. Previous applications of nonlinear solid elasticity employed materials from a few standard families such as linear corotational, nonlinear St.Venant-Kirchhoff, Neo-Hookean, Ogden or Mooney-Rivlin materials. However, the spaces of all nonlinear isotropic and anisotropic materials are infinite-dimensional and much broader than these standard materials. In this paper, we demonstrate how to intuitively explore the space of isotropic and anisotropic nonlinear materials, for design of animations in computer graphics and related fields. In order to do so, we first formulate the internal elastic forces and tangent stiffness matrices in the space of the principal stretches of the material. We then demonstrate how to design new isotropic materials by editing a single stress-strain curve, using a spline interface. Similarly, anisotropic (orthotropic) materials can be designed by editing three curves, one for each material direction. We demonstrate that modifying these curves using our proposed interface has an intuitive, visual, effect on the simulation. Our materials accelerate simulation design and enable visual effects that are difficult or impossible to achieve with standard nonlinear materials.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Physically based modeling, I.6.8 [Simulation and Modeling]: Types of Simulation—Animation

Keywords: FEM, isotropic, anisotropic, material, design

1 Introduction

Three-dimensional solid Finite Element Method (FEM) simulations are widely used in computer graphics, animation and related fields. FEM simulations, however, are greatly influenced by the specific material relationship between the displacements (strains) and elastic forces (stresses). To date, applications in computer graphics use linear materials, or nonlinear materials where the strain-stress relationship is specified using a global equation, such as the linear corotational, St.Venant-Kirchhoff, Neo-Hookean, Ogden or Mooney-Rivlin materials [Bonet and Wood 1997; Bower 2011]. These materials, however, only scratch the surface of the space of all isotropic nonlinear elastic materials. The space of all anisotropic nonlinear materials is even broader. In order to more easily achieve complex visual effects, we approach the problem of how to model and design arbitrary nonlinear materials, for use in animations in computer graphics and related fields. We give a method to design isotropic

Implicit Integration

Implicit Integration

Recall that we need implicit integration to avoid numerical instability (Class 5, page 13):

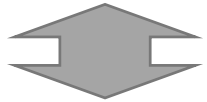
$$\begin{cases} \mathbf{v}^{[1]} = \mathbf{v}^{[0]} + \Delta t \mathbf{M}^{-1}\mathbf{f}^{[1]} \\ \mathbf{x}^{[1]} = \mathbf{x}^{[0]} + \Delta t \mathbf{v}^{[1]} \end{cases}$$

or

$$\begin{cases} \mathbf{x}^{[1]} = \mathbf{x}^{[0]} + \Delta t \mathbf{v}^{[0]} + \Delta t^2 \mathbf{M}^{-1}\mathbf{f}^{[1]} \\ \mathbf{v}^{[1]} = (\mathbf{x}^{[1]} - \mathbf{x}^{[0]})/\Delta t \end{cases}$$

We also said that:

$$\mathbf{x}^{[1]} = \mathbf{x}^{[0]} + \Delta t \mathbf{v}^{[0]} + \Delta t^2 \mathbf{M}^{-1}\mathbf{f}(\mathbf{x}^{[1]})$$



$$\mathbf{x}^{[1]} = \operatorname{argmin} F(\mathbf{x}) \quad \text{for} \quad F(\mathbf{x}) = \frac{1}{2\Delta t^2}\left\|\mathbf{x} - \mathbf{x}^{[0]} - \Delta t \mathbf{v}^{[0]}\right\|_{\mathbf{M}}^2 + E(\mathbf{x})$$

Newton-Raphson Method

The Newton-Raphson method, commonly known as Newton's method, solves the optimization problem: $x^{[1]} = \mathbf{argmin}\, F(x)$. ($F(x)$ is Lipschitz continuous.)

Given a current $x^{(k)}$, we approximate our goal by:

$$0 = F'(x) \approx F'\left(x^{(k)}\right) + F''\left(x^{(k)}\right)\left(x - x^{(k)}\right)$$

Newton's Method

Initialize $x^{(0)}$

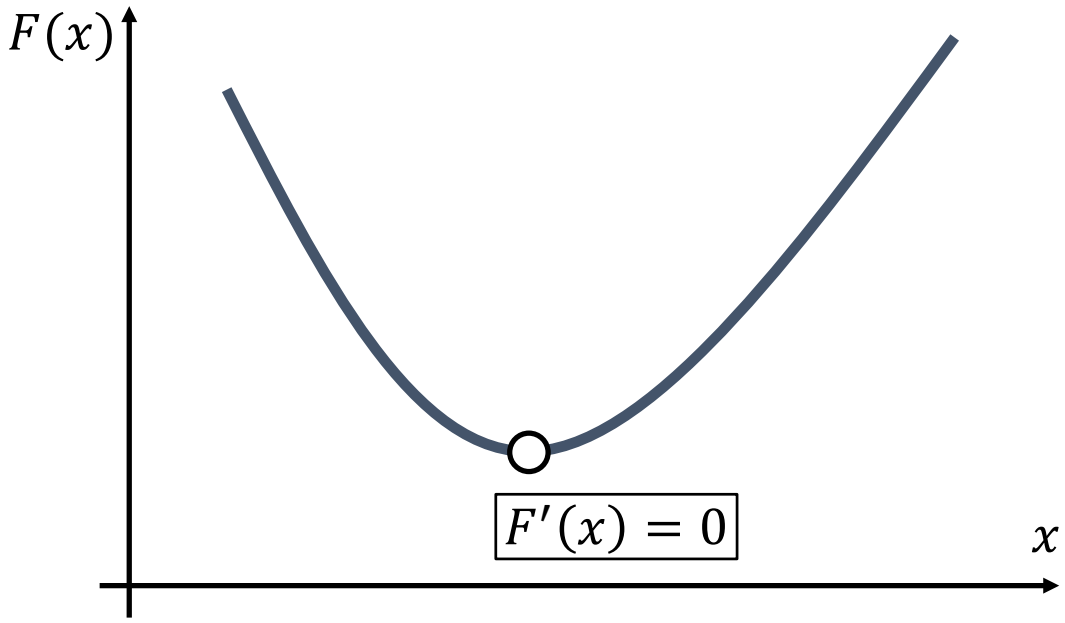
For $k = 0 \ldots K$

$$\Delta x \leftarrow -\left(F''\left(x^{(k)}\right)\right)^{-1} F'\left(x^{(k)}\right)$$

$$x^{(k+1)} \leftarrow x^{(k)} + \Delta x$$

If $|\Delta x|$ is small then break

$$x^{[1]} \leftarrow x^{(k+1)}$$



Newton-Raphson Method

Now we can apply Newton's method to: $\mathbf{x}^{[1]} = \mathbf{argmin}\, F(\mathbf{x})$.

Given a current $x^{(k)}$, we approximate our goal by:

$$\mathbf{0} = \nabla F(\mathbf{x}) \approx \nabla F\left(\mathbf{x}^{(k)}\right) + \frac{\partial F^2\left(\mathbf{x}^{(k)}\right)}{\partial \mathbf{x}^2}\left(\mathbf{x} - \mathbf{x}^{(k)}\right)$$

Newton's Method

Initialize $x^{(0)}$

For $k = 0 \ldots K$

$$\Delta\mathbf{x} \leftarrow -\left(\frac{\partial F^2\left(\mathbf{x}^{(k)}\right)}{\partial \mathbf{x}^2}\right)^{-1} \nabla F\left(\mathbf{x}^{(k)}\right)$$

$$\mathbf{x}^{(k+1)} \leftarrow \mathbf{x}^{(k)} + \Delta\mathbf{x}$$

If $\|\Delta\mathbf{x}\|$ is small then break

$$\mathbf{x}^{[1]} \leftarrow \mathbf{x}^{(k+1)}$$

Simulation by Newton's Method

Specifically to simulation, we have: $$F(\mathbf{x}) = \frac{1}{2\Delta t^2}\left\|\mathbf{x} - \mathbf{x}^{[0]} - \Delta t\mathbf{v}^{[0]}\right\|_{\mathbf{M}}^2 + E(\mathbf{x})$$

$$\nabla F\left(\mathbf{x}^{(k)}\right) = \frac{1}{\Delta t^2}\mathbf{M}\left(\mathbf{x}^{(k)} - \mathbf{x}^{[0]} - \Delta t\mathbf{v}^{[0]}\right) - \mathbf{f}\left(\mathbf{x}^{(k)}\right)$$

$$\frac{\partial^2 F\left(\mathbf{x}^{(k)}\right)}{\partial \mathbf{x}^2} = \frac{1}{\Delta t^2}\mathbf{M} + \mathbf{H}\left(\mathbf{x}^{(k)}\right)$$

Initialize $\mathbf{x}^{(0)}$, often as $\mathbf{x}^{[0]}$ or $\mathbf{x}^{[0]} + \Delta t\mathbf{v}^{[0]}$

For $k = 0 \ldots K$

Solve $$\left(\frac{1}{\Delta t^2}\mathbf{M} + \mathbf{H}\left(\mathbf{x}^{(k)}\right)\right)\Delta\mathbf{x} = -\frac{1}{\Delta t^2}\mathbf{M}\left(\mathbf{x}^{(k)} - \mathbf{x}^{[0]} - \Delta t\mathbf{v}^{[0]}\right) + \mathbf{f}\left(\mathbf{x}^{(k)}\right)$$

$\mathbf{x}^{(k+1)} \leftarrow \mathbf{x}^{(k)} + \Delta\mathbf{x}$

If $\|\Delta\mathbf{x}\|$ is small then break

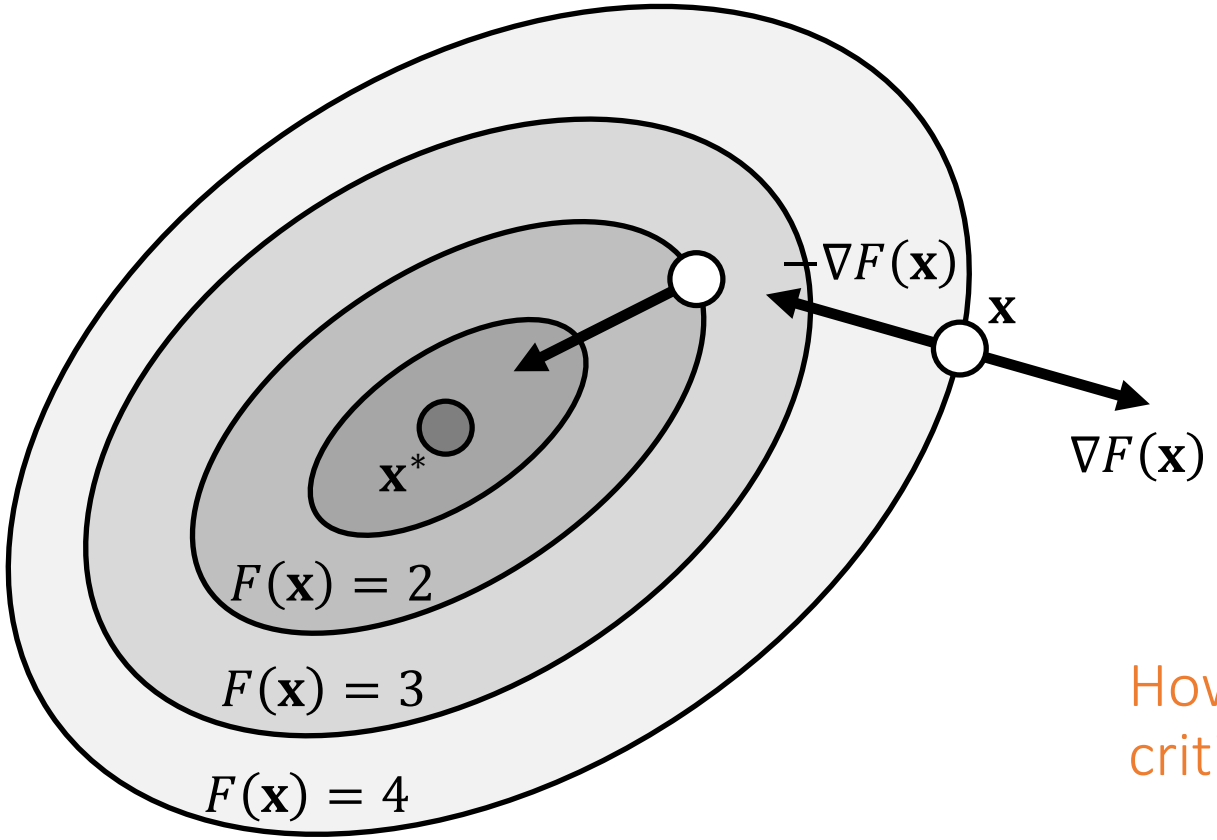
$\mathbf{x}^{[1]} \leftarrow \mathbf{x}^{(k+1)}$

$\mathbf{v}^{[1]} \leftarrow \left(\mathbf{x}^{[1]} - \mathbf{x}^{[0]}\right)/\Delta t$

Nonlinear Optimization

Gradient Descent

Another way to solve $\mathbf{x}^* = \mathbf{argmin}\, F(\mathbf{x})$ is the gradient descent method.



Gradient Descent

Initialize $\mathbf{x}^{(0)}$

For $k = 0 \ldots K$

$$\mathbf{x}^{(k+1)} \leftarrow \mathbf{x}^{(k)} - \alpha^{(k)} \nabla F\left(\mathbf{x}^{(k)}\right)$$

$\mathbf{x}^* \leftarrow \mathbf{x}^{(k+1)}$

Step size

How to find the optimal step size becomes a critical question.

Line Search Methods

There two popular line search methods for step size adjustment.

Exact Line Search

Tries to solve:

$$\alpha = \operatorname{argmin} F\left(\mathbf{x}^{(k)} - \alpha \nabla F\left(\mathbf{x}^{(k)}\right)\right)$$

Fast convergence

Large Overhead

Complicated

Backtracking Line Search

Initialize α

For $l = 0 \ldots \infty$

Wolfe Conditions

If $F\left(\mathbf{x}^{(k)} - \alpha \nabla F\left(\mathbf{x}^{(k)}\right)\right) < F\left(\mathbf{x}^{(k)}\right) + \cdots$

then break

$\alpha \leftarrow \beta\alpha$

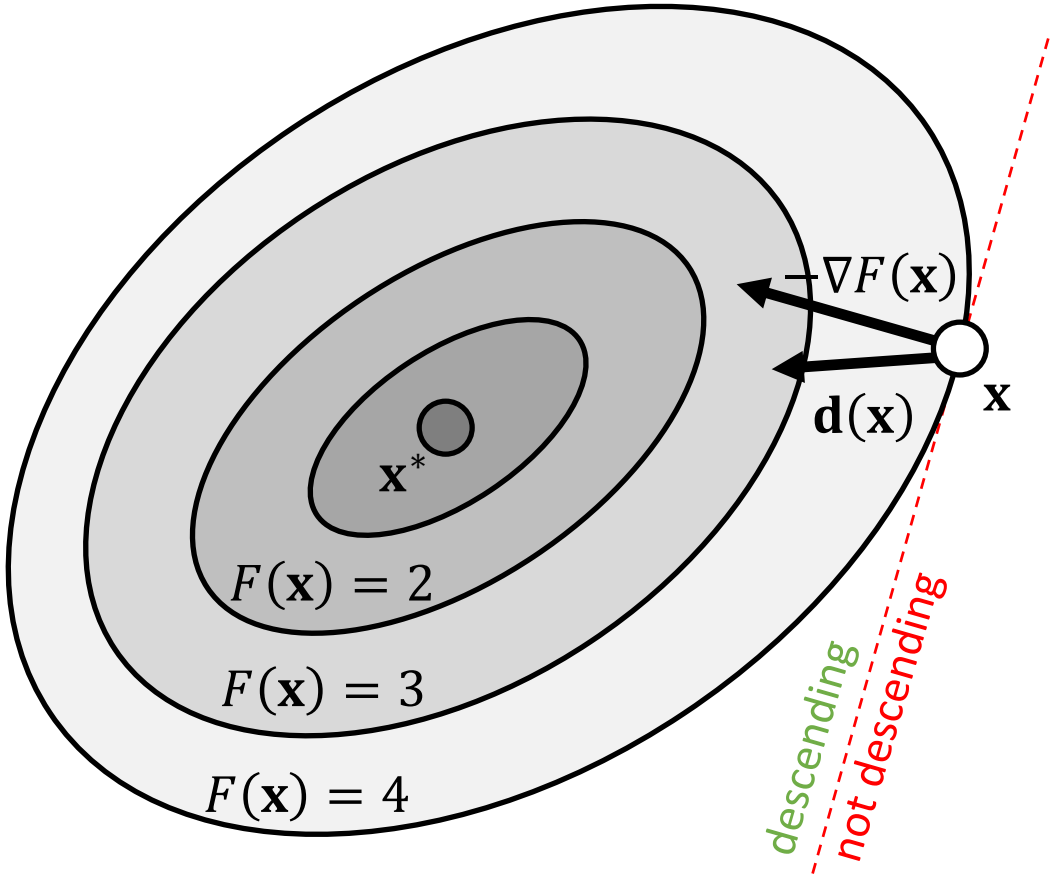
simple

Low overhead

Descent Directions

The direction $\mathbf{d}(\mathbf{x})$ is descending, if a sufficiently small step size α exists for:

$$F(\mathbf{x}) > F\big(\mathbf{x} + \alpha \mathbf{d}(\mathbf{x})\big)$$



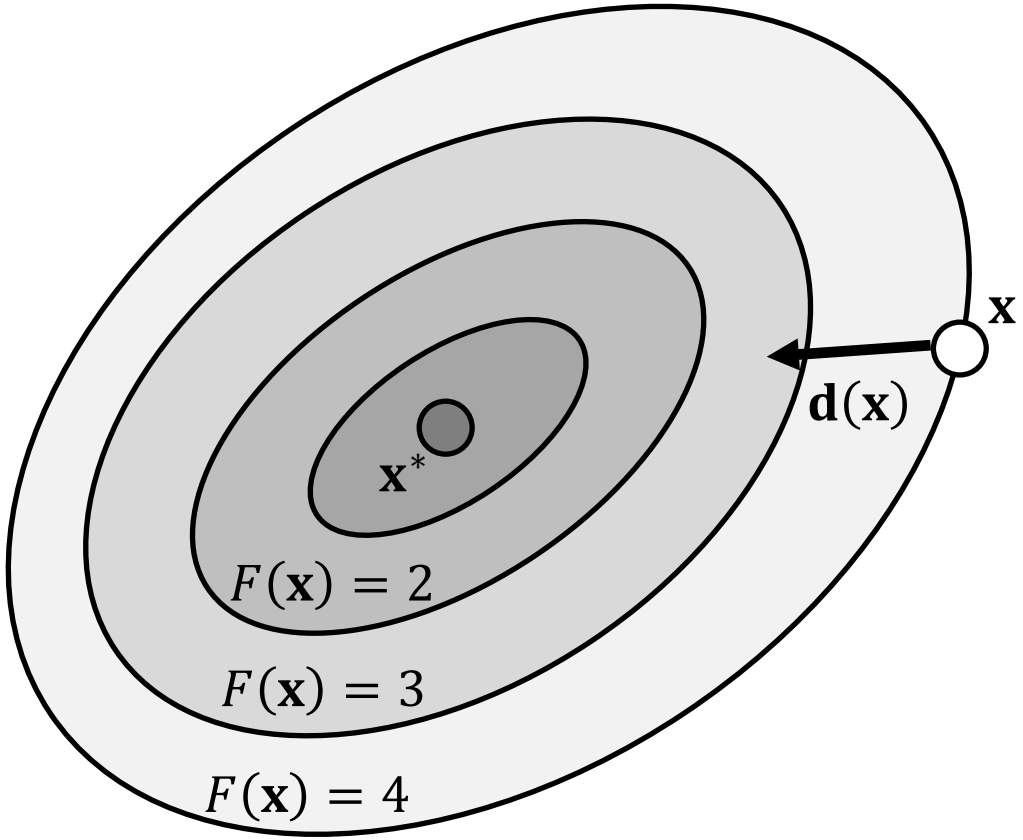
In other words,

$$-\nabla F(\mathbf{x}) \cdot \mathbf{d}(\mathbf{x}) > 0$$

Descent Methods

With line search, we can use any search direction as long as it's descending:

$$F\left(\mathbf{x}^{(0)}\right) > F\left(\mathbf{x}^{(1)}\right) > F\left(\mathbf{x}^{(2)}\right) > F\left(\mathbf{x}^{(3)}\right) > \cdots$$



Descent Methods

Initialize $\mathbf{x}^{(0)}$

For $k = 0 \ldots K$

$$\mathbf{x}^{(k+1)} \leftarrow \mathbf{x}^{(k)} + \alpha^{(k)}\mathbf{d}\left(\mathbf{x}^{(k)}\right)$$

$$\mathbf{x}^* \leftarrow \mathbf{x}^{(k+1)}$$

Descent Methods

- Gradient descent is a descent method, since:

$$\mathbf{d}(\mathbf{x}) = -\nabla F(\mathbf{x}) \quad \Longrightarrow \quad -\nabla F(\mathbf{x}) \cdot \left(-\nabla F(\mathbf{x})\right) > 0$$

- Newton's method is also a descent method, if the Hessian is always positive definite:

$$\mathbf{d}(\mathbf{x}) = -\left(\frac{\partial^2 F(\mathbf{x})}{\partial \mathbf{x}^2}\right)^{-1} \nabla F(\mathbf{x}) \quad \Longrightarrow \quad -\nabla F(\mathbf{x}) \cdot \left(-\left(\frac{\partial^2 F(\mathbf{x})}{\partial \mathbf{x}^2}\right)^{-1} \nabla F(\mathbf{x})\right) > 0$$

- Any method using a positive definite matrix **P** to modify the gradient yields a descent method:

$$\mathbf{d}(\mathbf{x}) = -\mathbf{P}^{-1}\nabla F(\mathbf{x}) \quad \Longrightarrow \quad -\nabla F(\mathbf{x}) \cdot \left(-\mathbf{P}^{-1}\nabla F(\mathbf{x})\right) > 0$$

Descent Methods

A unified descent framework

Descent Methods

Initialize $x^{(0)}$

For $k = 0 \dots K$

$$\mathbf{x}^{(k+1)} \leftarrow \mathbf{x}^{(k)} - \alpha^{(k)} \left(\mathbf{P}^{(k)}\right)^{-1} \nabla F\left(\mathbf{x}^{(k)}\right)$$

$$\mathbf{x}^{*} \leftarrow \mathbf{x}^{(k+1)}$$

Gradient Descent	[Wang et al. 2015]	[Fratarcangeli et al. 2016]	Projective Dynamics	Newton's Method
$\mathbf{P}^{(k)} = \mathbf{I}$	$\mathbf{P}^{(k)} = \text{diag}\left(\frac{\partial^2 F(\mathbf{x})}{\partial \mathbf{x}^2}\right)$	$\mathbf{P}^{(k)} = \text{lower}\left(\frac{\partial^2 F(\mathbf{x})}{\partial \mathbf{x}^2}\right)$	$\mathbf{P}^{(k)} = \mathbf{Const}$	$\mathbf{P}^{(k)} = \frac{\partial^2 F(\mathbf{x})}{\partial \mathbf{x}^2}$

Different $\mathbf{P}^{(k)}$

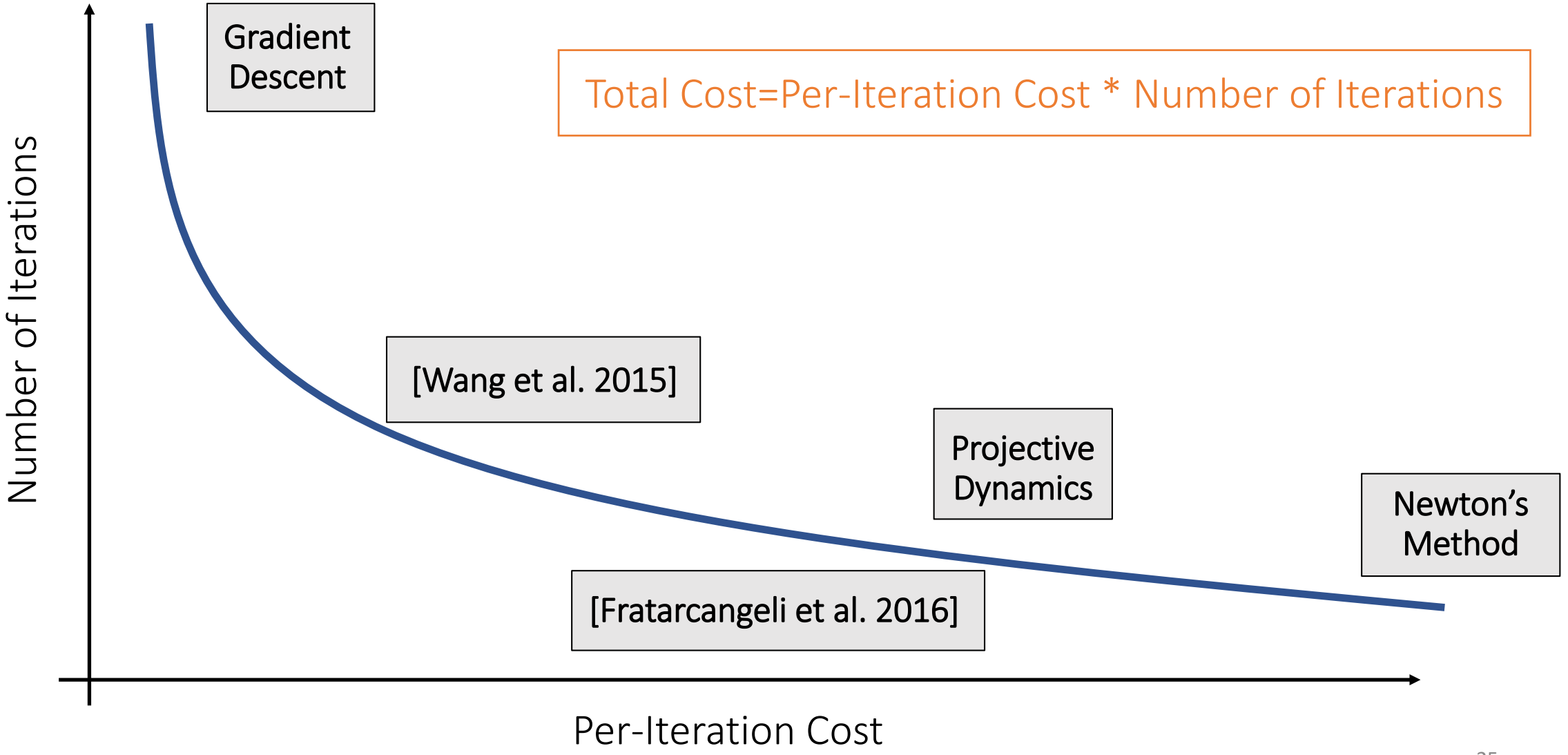
Cheap iterations

Many iterations

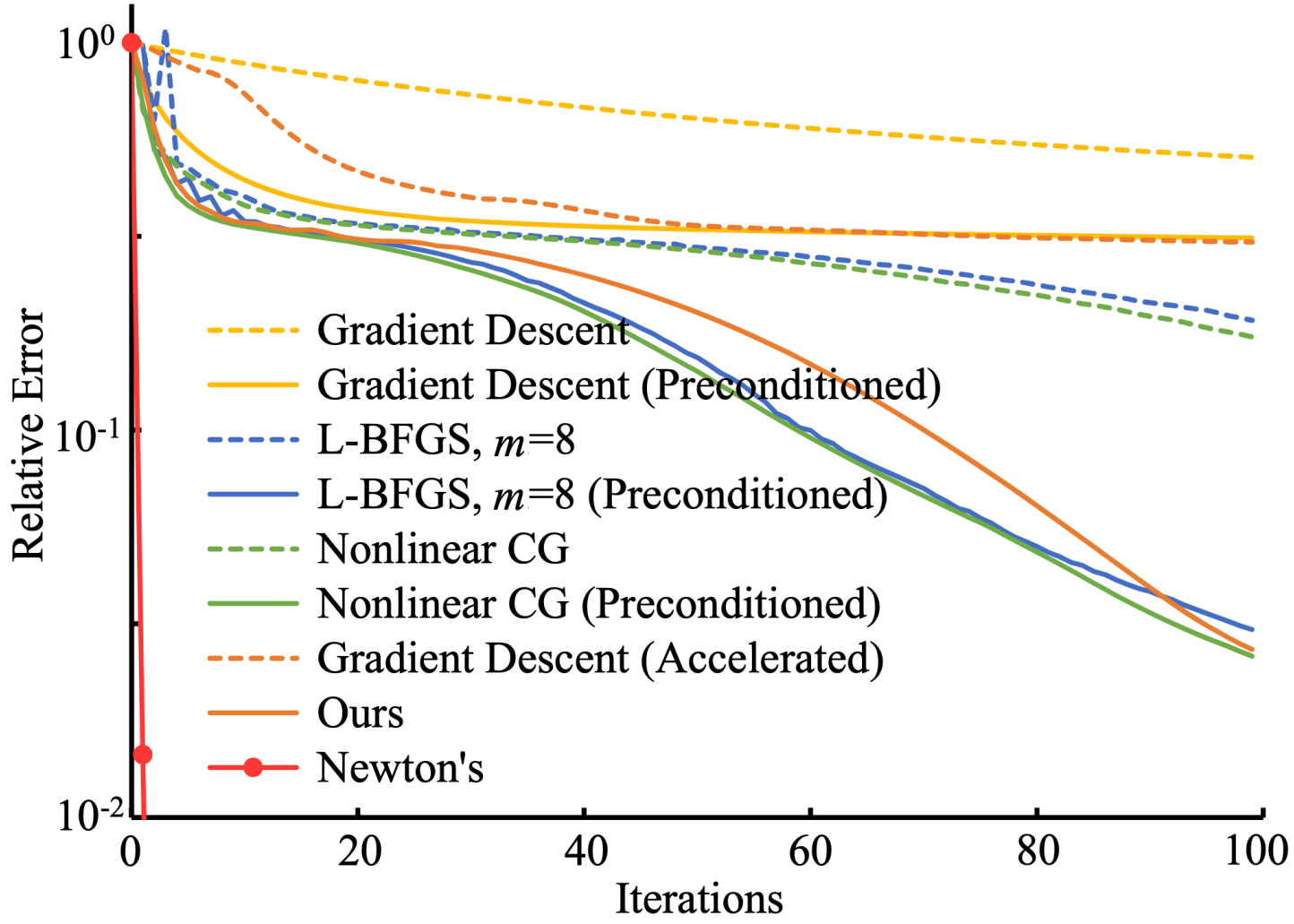
Few iterations

Costly iterations

Descent Methods



Comparisons



Wang. 2016. *Descent Methods for Elastic Body Simulation on the GPU.* TOG (SIGGRAPH Asia).

After-Class Reading

Wang. 2016. *Descent Methods for Elastic Body Simulation on the GPU.* TOG (SIGGRAPH Asia).

Descent Methods for Elastic Body Simulation on the GPU

Huamin Wang* The Ohio State University
Yin Yang* The University of New Mexico

(a) A resting dragon (b) A stretched dragon (c) A spiral dragon

Figure 1: *The dragon example. This model contains 16K vertices and 58K tetrahedra. Our elastic body simulator animates this example on the GPU at 30.5FPS, under the Mooney-Rivlin model. Thanks to a series of techniques we developed in this paper, the simulator can robustly handle very large time steps (such as $h = 1/30s$) and deformations.*

Abstract

We show that many existing elastic body simulation approaches can be interpreted as descent methods, under a nonlinear optimization framework derived from implicit time integration. The key question is how to find an effective descent direction with a low computational cost. Based on this concept, we propose a new gradient descent method using Jacobi preconditioning and Chebyshev acceleration. The convergence rate of this method is comparable to that of L-BFGS or nonlinear conjugate gradient. But unlike other methods, it requires no dot product operation, making it suitable for GPU implementation. To further improve its convergence and performance, we develop a series of step length adjustment, initialization, and invertible model conversion techniques, all of which are compatible with GPU acceleration. Our experiment shows that the resulting simulator is simple, fast, scalable, memory-efficient, and robust against very large time steps and deformations. It can correctly simulate the deformation behaviors of many elastic materials, as long as their energy functions are second-order differentiable and their Hessian matrices can be quickly evaluated. For additional speedups, the method can also serve as a complement to other techniques, such as multi-grid.

Keywords: Nonlinear optimization, Jacobi preconditioning, the Chebyshev method, GPU acceleration, hyperelasticity.

Concepts: •**Computing methodologies** → **Physical simulation;**

*e-mail: whmin@cse.ohio-state.edu; yangy@unm.edu

SA '16 Technical Papers,, December 05-08, 2016, , Macao
ISBN: 978-1-4503-4514-9/16/12
DOI: http://dx.doi.org/10.1145/2980179.2980236

1 Introduction

Solid materials often exhibit complex elastic behaviors in the real world. While we have seen a variety of models being developed to describe these behaviors over the past few decades, our ability to simulate them computationally is rather limited. Early simulation techniques often use explicit time integration, which is known for its numerical instability problem. A typical solution is to use implicit time integration instead. Given the nonlinear force-displacement relationship of an elastic material, we can formulate implicit time integration into a nonlinear system. Baraff and Witkin [1998] proposed to linearize this system at the current shape and solve the resulting linear system at each time step. Their method is equivalent to running one iteration of Newton's method. Alternatively, we can linearize the system at the rest shape and solve a linear system with a constant matrix. While this method is fast thanks to matrix pre-factorization, the result becomes unrealistic under large deformation. To address this issue, Müller and Gross [2004] factored out the rotational component from the displacement and ended up with solving a new linear system at each time step again. Even if we accept formulating elastic simulation into a linear system, we still face the challenge of solving a large and sparse system. Unfortunately, most linear solvers are not fully compatible with parallelization and they cannot be easily accelerated by the GPU.

In recent years, graphics researchers studied the use of geometric constraints and developed a number of constraint-based simulation techniques, such as strain limiting [Provot 1996; Thomaszewski et al. 2009; Wang et al. 2010], position-based dynamics [Müller et al. 2007; Müller 2008; Kim et al. 2012], and shape matching [Müller et al. 2005; Rivers and James 2007]. While these techniques are easy to implement and compatible with GPU acceleration, they offer little control on their underlying elastic models. To solve this problem, Liu and collaborators [2013] and Bouaziz and colleagues [2014] described geometric constraints as elastic energies in a quadratic form. This implies that their technique, known as *projective dynamics*, is not suitable for simulating generic elastic models. The recent work by Tournier and colleagues [2015] proposed to incorporate both elastic forces and compliant constraints into a single linear system. This technique is designed

A Summary For the Day

- We can calculate the Hessian of the FEM elastic energy based on SVD derivatives.
- The goal of doing this is for implicit time integration.
- Fundamentally, the goal is to solve a nonlinear optimization.
  - Gradient Descent, Newton's method, and others can all be considered as descent methods.
  - The key question is the matrix for calculating the search direction.
  - We need both the per-iteration cost and the number of iterations to be small.