# Computational methods in applied inverse problems

Uri Ascher

Department of Computer Science
University of British Columbia

October 2017

## Four lectures

1. Calibration and simulation of deformable objects
2. Data manipulation and completion
3. Estimating the trace of a large implicit matrix and applications
4. Numerical analysis in visual computing: not too little, not too much

Please see synopsis at
http://mtm.ufsc.br/∼aleitao/public/impa-pt2017/index.html

## Collaborators

1. Soft object calibration and simulation, 2015
    - Bin Wang
    - Longhua Wu
    - KangKang Yin
    - Libin Liu
    - Hui Huang
2. Ongoing
    - Edwin Chen
    - Dinesh Pai
    - Danny Kaufman & David Levin
    - Bin & Hui

## Outline

- Motivation
- Deformation capture and modeling of soft objects
- Large-step time integration of elastodynamics equations
- Optimization

## Motivation: the problem

- Motion simulation of structures containing flexible soft objects is ubiquitous in current computer graphics and robotics research.
- Many industrial applications, including touching, facial expression, cloth motion, etc.
- Such high quality simulations can be rather costly for articulated systems or bodies, and the assembly of internal forces through an often nonlinear stiffness matrix is expensive.
- Furthermore, the model typically requires calibration, e.g., specifying Young modulus and damping properties, which are expressed as (distributed) parameters in the elastodynamics differential equations governing the motion.

## Motivation: the problem

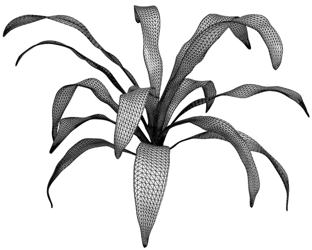
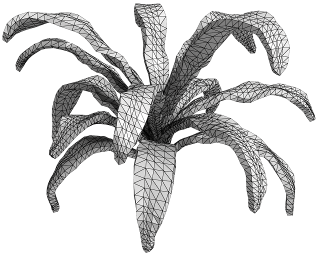
- Motion simulation of structures containing flexible soft objects is ubiquitous in current computer graphics and robotics research.
- Many industrial applications, including touching, facial expression, cloth motion, etc.
- Such high quality simulations can be rather costly for articulated systems or bodies, and the assembly of internal forces through an often nonlinear stiffness matrix is expensive.
- Furthermore, the model typically requires calibration, e.g., specifying Young modulus and damping properties, which are expressed as (distributed) parameters in the elastodynamics differential equations governing the motion.

## Motivation: the problem

- Motion simulation of structures containing flexible soft objects is ubiquitous in current computer graphics and robotics research.
- Many industrial applications, including touching, facial expression, cloth motion, etc.
- Such high quality simulations can be rather costly for articulated systems or bodies, and the assembly of internal forces through an often nonlinear stiffness matrix is expensive.
- Furthermore, the model typically requires calibration, e.g., specifying Young modulus and damping properties, which are expressed as (distributed) parameters in the elastodynamics differential equations governing the motion.

## Motivation: how to handle this challenge

1. For a given calibration, semi-discretize elastodynamics equations in variational form using (co-rotated) FEM on a coarse moving tetrahedral mesh.
2. To obtain parameters (i.e., calibrate model), acquire position data in controlled environment and solve inverse problem.
3. Use physics-based simulation: in many applications, require result to look good, rather than be accurate to within **tol**. In particular:
    - It's the motion simulation results, rather than accuracy of parameters, that is eventually observed.
    - Can often use semi-implicit methods with large time steps to dampen invisible high oscillations.
    - (Some aspects of this remind me of geometric integration methods.)

Fine surface mesh Coarser volumetric mesh

[Wang, Wu, Yin, A., Liu & Huang '15]

[Muller et al., 2002; Kim & Pollard, 2011]

Motion capture

[Wang, Wu, Yin, A., Liu & Huang '15]

## Outline

- Motivation
- Deformation capture and modeling of soft objects
- Large-step time integration of elastodynamics equations
- Optimization

## Challenges from deformation capturing and modeling work

- Discretizing ODE system in presence of high frequencies
- Highly non-convex, highly nonlinear optimization
- Somewhat dubious statistics
- Heterogeneous material: homogenization
- More realistic damping model
- Better measurements (for Poisson ratio), better equipment
- etc.

## Outline

- Motivation
- Deformation capture and modeling of soft objects
- Large-step time integration of elastodynamics equations
- Optimization

## Outline

- Soft material models
- Large-step methods
    - Backward Euler (BE) and semi-implicit (SI)
    - Generalized $\alpha$ method
    - When we don't want to conserve total energy: decoupling

## Simulating soft object motion: Elastodynamics

- Large deformation in 3D leads to nasty partial differential systems [Ciarlet '93]
- Convenient to semi-discretize without even forming PDEs, e.g., using mass-spring system, or at the variational level using FEM and possibly co-rotated FEM.
- Obtain ODE system

$$\begin{aligned} M\ddot{\mathbf{q}}(t) &= \mathbf{f}_{\text{tot}}(\mathbf{q},\mathbf{v}) \\ &\equiv \mathbf{f}_{\text{els}}(\mathbf{q}) + \mathbf{f}_{\text{dmp}}(\mathbf{q},\mathbf{v}) + \mathbf{f}_{\text{ext}}. \end{aligned}$$

[Note change of notation from $x$ to $\mathbf{q}$: sorry]

## An elastodynamics ODE system

- Equations of motion ($\mathbf{v} = \dot{\mathbf{q}}$ are node velocities)

$$M\ddot{\mathbf{q}}(t) = \mathbf{f}_{\mathrm{els}}(\mathbf{q}) + \mathbf{f}_{\mathrm{dmp}}(\mathbf{q}, \mathbf{v}) + \mathbf{f}_{\mathrm{ext}},$$

  with the elastic and damping forces

$$\mathbf{f}_{\mathrm{els}}(\mathbf{q}) = -\frac{\partial}{\partial \mathbf{q}} W(\mathbf{q}(t)), \quad \mathbf{f}_{\mathrm{dmp}}(\mathbf{q}, \mathbf{v}) = D\mathbf{v}(t),$$

  where $W(\mathbf{q}(t))$ is the elastic potential of the corresponding model.
- This elastic potential can be
  - quadratic in a linear elasticity model;
  - quartic in a linear FEM model with StVK material;
  - nonlinear for co-rotated FEM for linear material;
  - nonlinear for neo-Hookean material and artificially designed material [Xu et al. '15]

## A 1st order ODE system

- Rewrite at some $t = t_n$ as

$$\begin{aligned} \dot{\mathbf{u}}(t) \equiv \begin{pmatrix} \dot{\mathbf{q}}(t) \\ \dot{\mathbf{v}}(t) \end{pmatrix} &= \begin{pmatrix} \mathbf{v}(t) \\ M^{-1}\mathbf{f}_{\mathrm{tot}}(\mathbf{q},\mathbf{v}) \end{pmatrix} \\ &= \begin{pmatrix} 0 & I \\ -M^{-1}K & M^{-1}D \end{pmatrix} \begin{pmatrix} \mathbf{q}(t) \\ \mathbf{v}(t) \end{pmatrix} + \begin{pmatrix} 0 \\ \mathbf{g}(\mathbf{u}(t)) \end{pmatrix}, \\ \mathbf{g}(\mathbf{u}(t)) &= \mathbf{f}_{\mathrm{tot}} + M^{-1}K\mathbf{q}(t), \end{aligned}$$

  where $K = \frac{\partial}{\partial \mathbf{q}} \mathbf{f}_{\mathrm{els}}(\mathbf{q})$ is the tangent stiffness matrix at $\mathbf{q} = \mathbf{q}(t_n)$.
- Note that the (symmetric) *stiffness matrix is not always positive definite! The elastic energy need not be always convex.*
- Further, often there is highly oscillatory stiffness, even though the observed motion is damped and does not vibrate rapidly. This happens when
  - the scale of the simulation is large, and/or
  - the material stiffens under large deformation.

## Large steps methods

- Want to use a time step size $\tau$ commensurate with the damped motion.
- Can't use explicit Runge-Kutta (RK) discretization.
- Moreover, implicit RK requires solution of nonlinear system at each step: can be nasty if the step size $\tau$ is large.
- Can use a semi-implicit (SI) method, i.e., backward Euler (BE) with only one Newton iteration at each time step starting from $\mathbf{u}_n$. [Baraff & Witkin '98; Ascher '08].
  This and generalized $\alpha$ are the most popular methods in use to date.
- However, heavy step-size dependent damping is introduced: not easy for an artist to work with; and affects different materials differently.

## Large steps methods

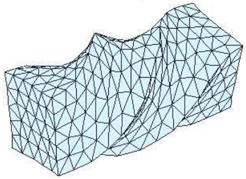
- Want to use a time step size $\tau$ commensurate with the damped motion.
- Can't use explicit Runge-Kutta (RK) discretization.
- Moreover, implicit RK requires solution of nonlinear system at each step: can be nasty if the step size $\tau$ is large.
- Can use a semi-implicit (SI) method, i.e., backward Euler (BE) with only one Newton iteration at each time step starting from $\mathbf{u}_n$. [Baraff & Witkin '98; Ascher '08].
  This and generalized $\alpha$ are the most popular methods in use to date.
- However, heavy step-size dependent damping is introduced: not easy for an artist to work with; and affects different materials differently.

## Large steps methods

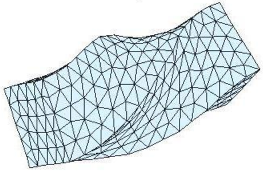
- Want to use a time step size $\tau$ commensurate with the damped motion.
- Can't use explicit Runge-Kutta (RK) discretization.
- Moreover, implicit RK requires solution of nonlinear system at each step: can be nasty if the step size $\tau$ is large.
- Can use a semi-implicit (SI) method, i.e., backward Euler (BE) with only one Newton iteration at each time step starting from $\mathbf{u}_n$. [Baraff & Witkin '98; Ascher '08].
  This and generalized $\alpha$ are the most popular methods in use to date.
- However, heavy step-size dependent damping is introduced: not easy for an artist to work with; and affects different materials differently.

## Twisted bar: neo-Hookean material

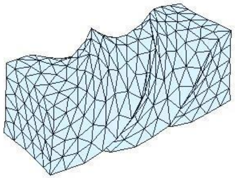
ERE step size 1/60s



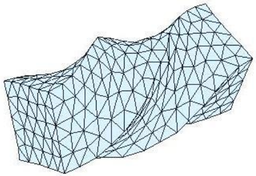
SI step size 1/60s



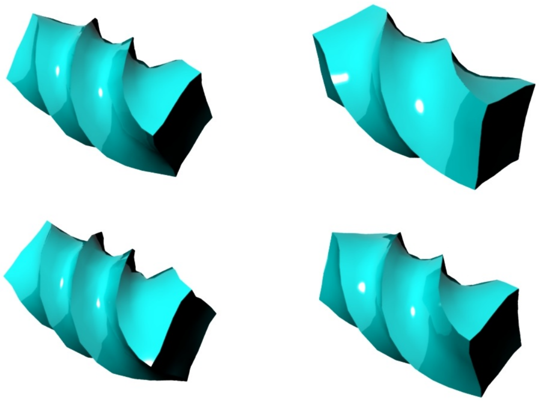
IM step size 1/60s



BE step size 1/60s



## Twisted bar dressed



## The simplest case

- To see what happens with SI and BE, consider the scalar constant-coefficient ODE

$$\ddot{q} + d\dot{q} + \omega^2 q = 0,$$

where $d \geq 0$ is a damping parameter, and $\omega > d/2$ is a real-valued frequency.

- Setting $q(t) = \exp(\lambda t)$, obtain $\lambda^2 + d\lambda + \omega^2 = 0$, so

$$\lambda = \frac{1}{2}[-d \pm \sqrt{d^2 - 4\omega^2}].$$

- In particular, for the undamped case $d = 0$, the modes are $\exp(\lambda t)$ with $\lambda = \pm \imath\omega$ , i.e., these are oscillatory, undamped modes.
- Furthermore, for $d \geq 0$, the modes damping is

$$|\exp(\lambda t_n)| = \exp(Re(\lambda) t_n) = \exp\left(-\frac{d}{2} t_n\right).$$

## SI/BE DAMPING

- Rewrite as a 1st order system and employ BE≡SI at $t_n = n\tau$:

$$\begin{pmatrix} q_n \\ v_n \end{pmatrix} = \frac{1}{1+\tau d+\tau^2\omega^2} \begin{pmatrix} 1+\tau d & \tau \\ -\tau\omega^2 & 1 \end{pmatrix} \begin{pmatrix} q_{n-1} \\ v_{n-1} \end{pmatrix}.$$

- Next, apply this to the *undamped* case: set $d = 0$ and write the method as

$$\begin{pmatrix} q_n \\ v_n \end{pmatrix} = T \begin{pmatrix} q_{n-1} \\ v_{n-1} \end{pmatrix}.$$

- The eigenvalues of the propagator $T$ are $\mu = \frac{1}{1 \pm \imath\tau\omega}$, so the spectral radius is

$$\rho(T) = \max|\mu| = 1/\sqrt{1+(\tau\omega)^2}.$$

- Obviously, (i) $\rho < 1$ for any $\tau > 0$, (ii) $\rho$ decreases as $\tau\omega$ increases, and (iii) $\rho \to 0$ as $\tau\omega \to \infty$ – all of which are BE trademarks.

## SI/BE artificial damping

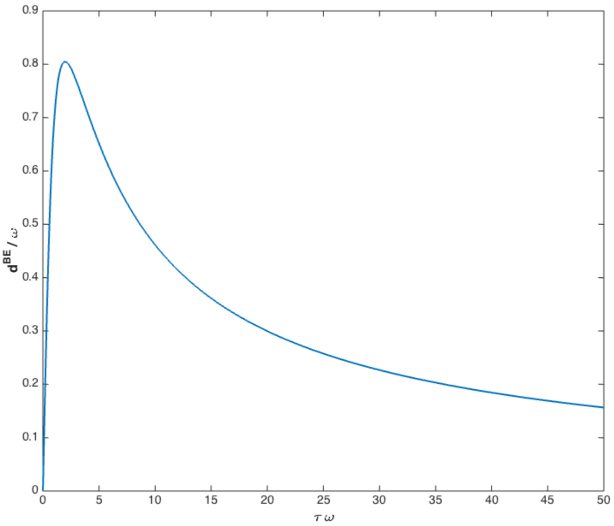
- But $\rho < 1$ for $d = 0$ implies artificial damping!
- To see how much artificial damping, equate $\exp(-\frac{d^{BE}}{2} n\tau) = \rho^n = (1 + (\tau\omega)^2)^{-n/2}$. Taking ln and cleaning, obtain

$$d^{BE} = \frac{1}{\tau} \ln(1 + (\tau\omega)^2). \quad \text{or} \quad \frac{d^{BE}}{\omega} = \frac{1}{\tau\omega} \ln(1 + (\tau\omega)^2).$$

- So, applying BE/SI to the highly oscillatory, undamped ODE $\ddot{q} + \omega^2 q = 0$, the solution is best related instead to the exact solution (mode) of the *ghost ODE*

$$\ddot{q} + d^{BE}\dot{q} + \omega^2 q = 0.$$

## BE artificial damping curve

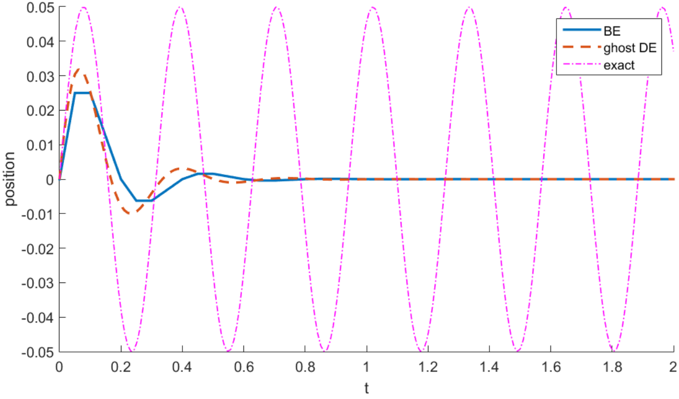


## SI/BE artificial damping: Case 1

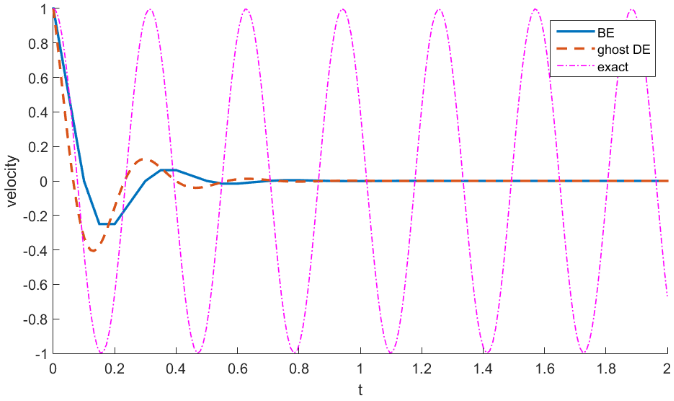
If $\tau = 1/\omega$ then $d^{BE} = \omega \ln(2)$. Applying BE/SI to the highly oscillatory, undamped ODE $\ddot{q} + \omega^2 q = 0$, the solution is best related instead to the exact solution (mode) of the ghost ODE

$$\ddot{q} + \omega \ln(2) \dot{q} + \omega^2 q = 0.$$

## EXAMPLE: $\omega = 20,\ \tau = 1/\omega$



## EXAMPLE: $\omega = 20,\ \tau = 1/\omega$

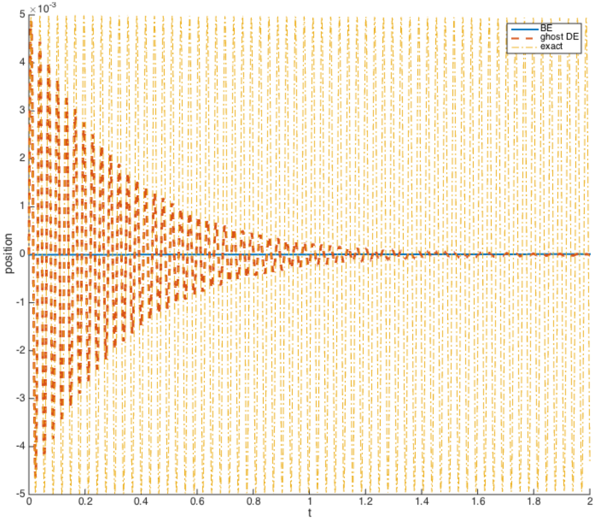


## SI/BE artificial damping: Case 2

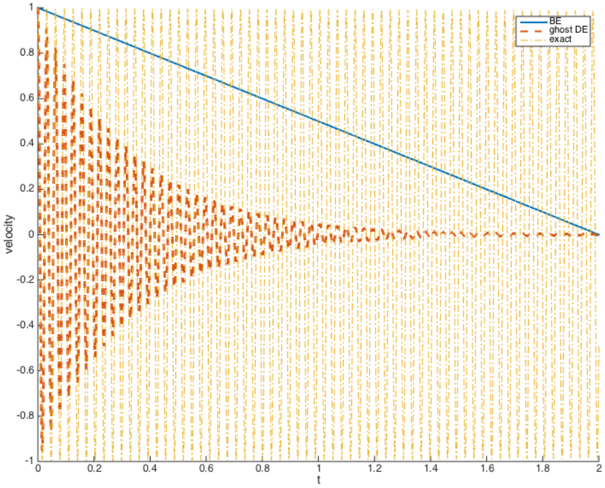
Really large step: if $\tau = 1$ then $d^{BE} = \ln(1+\omega^2) \approx 2\ln(\omega)$. Applying BE/SI to the highly oscillatory, undamped ODE $\ddot{q} + \omega^2 q = 0$, the solution is best related instead to the exact solution (mode) of the ghost ODE

$$\ddot{q} + \ln(1+\omega^2)\dot{q} + \omega^2 q = 0.$$

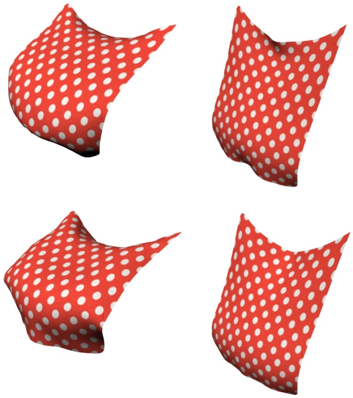
## EXAMPLE: $\omega = 200$, $\tau = 1$



## EXAMPLE: $\omega = 200,\ \tau = 1$

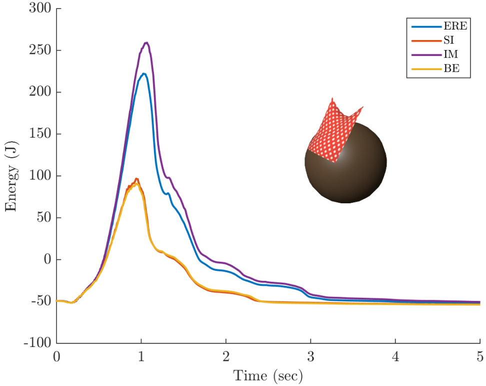


## Cloth after colliding with a sphere



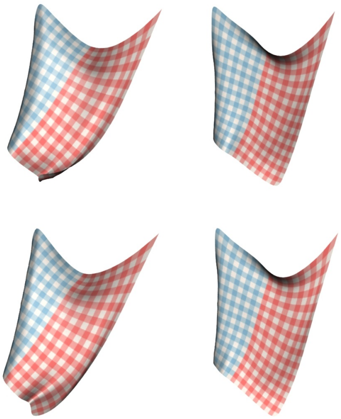
Top – left: ERE, right: SI
Bottom – left: IM, right: BE

## Example cont.



Energy profile of each method in the simulation with cloth collision.

## Cloth with mixed stiffness after colliding with a sphere



Top – left: ERE, right: SI
Bottom – left: IM, right: BE

## Generalized $\alpha$ method

- Mechanical engineers often use the Generalized $\alpha$ method [Chung & Hullbert, '93, Kobis & Arnold, '16] rather than backward Euler.
- It is a one-step Newmark-type method (discretize $\dot{\mathbf{v}} = \mathbf{a},\ M\mathbf{a} = \mathbf{f}(\mathbf{q}, \mathbf{v})$, rather than $\dot{\mathbf{v}} = M^{-1}\mathbf{f}(\mathbf{q}, \mathbf{v})$).
- It has a parameter $r$ that can be tuned to select anywhere between BE-like strong damping of high frequencies and no damping at all.
- For any choice of $0 \le r \le 1$ the method is second order in $\tau$.
- The size of nonlinear system to solve at each step is minimal.

## Generalized $\alpha$ method detail

- Rewrite ODE system $M\ddot{\mathbf{q}} = \mathbf{f}(t, \mathbf{q}, \dot{\mathbf{q}})$ as a simple semi-explicit DAE:

$$\dot{\mathbf{q}} = \mathbf{v}, \quad \dot{\mathbf{v}} = \mathbf{a}, \quad \mathbf{0} = M\mathbf{a} - \mathbf{f}(t, \mathbf{q}, \mathbf{v}).$$

- Set $\alpha = \alpha_m - \alpha_f$. Step from $t_{n-1}$ to $t_n$:

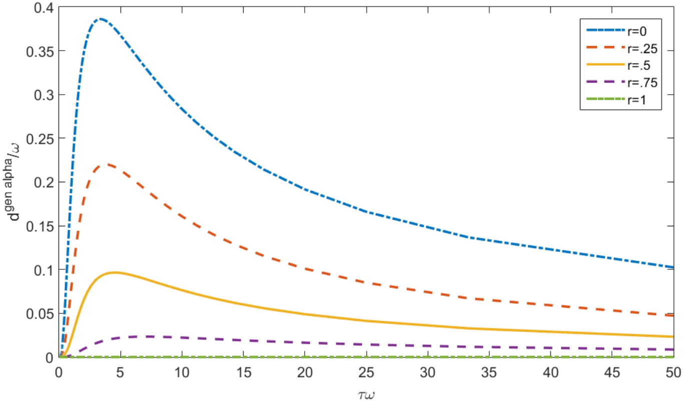
$$\begin{aligned}
\mathbf{q}_n &= \mathbf{q}_{n-1} + h\mathbf{v}_{n-1} + \frac{h^2}{2}\left((1-2\beta)\mathbf{a}_{n-1+\alpha} + 2\beta\mathbf{a}_{n+\alpha}\right), \\
\mathbf{v}_n &= \mathbf{v}_{n-1} + h\left((1-\gamma)\mathbf{a}_{n-1+\alpha} + \gamma\mathbf{a}_{n+\alpha}\right), \\
& (1-\alpha_m)M\mathbf{a}_{n+\alpha} + \alpha_m M\mathbf{a}_{n-1+\alpha} \\
&= (1-\alpha_f)\mathbf{f}(t_n, \mathbf{q}_n, \mathbf{v}_n) + \alpha_f\mathbf{f}(t_{n-1}, \mathbf{q}_{n-1}, \mathbf{v}_{n-1}).
\end{aligned}$$

- The coefficients are

$$\alpha_m = \frac{2r-1}{1+r}, \quad \alpha_f = \frac{r}{1+r}, \quad \beta = \frac{(1-\alpha)^2}{4}, \quad \gamma = \frac{1}{2} - \alpha.$$

Therefore, $-1 \le \alpha = \frac{r-1}{r+1} \le 0$.

## Generalized $\alpha$ artificial damping curve



Generalized $\alpha$ (GA) curves $d^{GA}/\omega$ as a function of $\tau\omega$, for $r = 0 : .25 : 1$

## Splitting fast and slow components

- The problem with generalized $\alpha$ with significant damping is similar to that with BE: all modes are damped, the fast modes not being decoupled from slow ones.
- Example [Ascher & Reich, '99]: stiff-spring pendulum.

## Stiff-spring pendulum with gravity

1. Stiff-spring pendulum Hamiltonian:

$$H(r,\phi,p_r,p_\phi) = \frac{1}{2}\left[p_r^2 + r^{-2}p_\phi^2 + \omega^2(r-r_0)^2\right] + gr\sin(\phi).$$

2. Corresponding ODE system:

$$\begin{aligned}
\dot{r} &= p_r \\
\dot{p}_r &= -\omega^2(r-r_0) + r^{-3}p_\phi^2 - g\sin(\phi) \\
\dot{\phi} &= r^{-2}p_\phi \\
\dot{p}_\phi &= -gr\cos(\phi)
\end{aligned}$$

## Stiff-spring pendulum

$$
\begin{aligned}
\dot{r} &= p_r \\
\dot{p}_r &= -\omega^2(r - r_0) + r^{-3}p_\phi^2 - g\sin(\phi) \\
\dot{\phi} &= r^{-2}p_\phi \\
\dot{p}_\phi &= -gr\cos(\phi)
\end{aligned}
$$

- In polar coordinates (radius $r$, angle $\phi$) the radius of the spring oscillates, faster and with smaller amplitude the larger the frequency $\omega$, while the angle varies slowly.
- But in Cartesian coordinates $\mathbf{q} = (r\cos\phi, -r\sin\phi)$ contains both fast and slow components.
- Visually, high oscillations are not seen, so want them damped away.
- Can do this easily in $(r, \phi)$ coordinate system but not in $\mathbf{q}$.

## IMEX of sorts

1. Visually, high oscillations are not seen, so want them damped away.
2. Thus, although the system is Hamiltonian, we *don't want to conserve the total energy*! Only the slow energy matters for visual computing.
3. For **large** $\tau\omega$, can discretize DEs for $r, p_r$ implicitly (by SI, BE, gen. $\alpha$...) and eqns for $\phi, p_\phi$ explicitly (e.g., by RK4).
4. Only the slow energy is approximated well.
5. This works very well. But unfortunately, *it's hard to generalize* :-(

## IMEX OF SORTS

1. Visually, high oscillations are not seen, so want them damped away.
2. Thus, although the system is Hamiltonian, we *don't want to conserve the total energy*! Only the slow energy matters for visual computing.
3. For **large** $\tau\omega$, can discretize DEs for $r$, $p_r$ implicitly (by SI, BE, gen. $\alpha$...) and eqns for $\phi$, $p_\phi$ explicitly (e.g., by RK4).
4. Only the slow energy is approximated well.
5. This works very well. But unfortunately, *it's hard to generalize* :-(

## Outline

- Motivation
- Deformation capture and modeling of soft objects
- Large-step time integration of elastodynamics equations
- Optimization

## Inverse problem for deformable object

- Fortunately, the (hopefully few, but nasty) parameters $\mathbf{u}$, including Young modulus and damping, are only weakly coupled to the (many, but easy-going) parameters $\mathbf{X}$.
- So, apply a splitting scheme, alternately solving for $\mathbf{X}$ and $\mathbf{u}$.
- Find $\mathbf{X}$ by requiring equilibrium (a specialized treatment: see paper).
- Optimize with respect to $\mathbf{u}$ – **how?**
  Let us concentrate on this latter question.

## Inverse problem for deformable object

- Fortunately, the (hopefully few, but nasty) parameters **u**, including Young modulus and damping, are only weakly coupled to the (many, but easy-going) parameters **X**.
- So, apply a splitting scheme, alternately solving for **X** and **u**.
- Find **X** by requiring equilibrium (a specialized treatment: see paper).
- Optimize with respect to **u** – **how?**
  Let us concentrate on this latter question.

## Inverse problem for deformable object

- Fortunately, the (hopefully few, but nasty) parameters **u**, including Young modulus and damping, are only weakly coupled to the (many, but easy-going) parameters **X**.
- So, apply a splitting scheme, alternately solving for **X** and **u**.
- Find **X** by requiring equilibrium (a specialized treatment: see paper).
- Optimize with respect to **u** – **how?**
  Let us concentrate on this latter question.

## Outline

- Case study: soft body deformation and calibration
- Unconstrained optimization
- Gradient descent methods
- Derivative-free (search) methods
- Nonconvex problems

## Optimization problems

**Optimization problem**

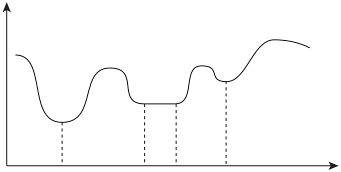
$$\text{minimize}_{\mathbf{x}} \quad f(\mathbf{x})$$

- $\mathbf{x} = (x_1, \ldots, x_n)$: optimization variables (unknowns)
- $f : \Re^n \to \Re$: objective function
- In general, there may be constraints: both equality and inequality. But we shall assume no constraints.

**Optimal solution** $\mathbf{x}^* = (x_1^*, x_2^*, \ldots, x_n^*)$ has the *smallest* value of $f$.

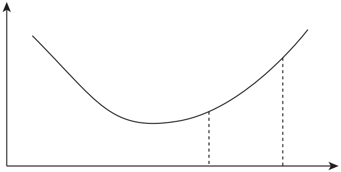
## Illustrative Examples

$\text{minimize}_{\mathbf{x}} \quad f(\mathbf{x})$



$\text{minimize}_{\mathbf{x}} \quad f(\mathbf{x})$

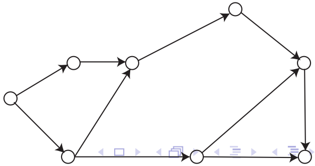
$s.t. \quad \ell \leq \mathbf{x} \leq \mathbf{u}$



$\text{minimize}_{\mathbf{x}}$ cost of flow

st network capacity

flow conservation



## Flavours in optimization problems

- Continuous vs. discrete optimization
- Constrained vs. unconstrained optimization
- Global vs. local optimization
- Stochastic vs. deterministic optimization

Our domain is in blue: continuous, (mainly) unconstrained and constrained, (mainly) local, (mainly) deterministic, often but not always convex (where local=global).

## Solving optimization problems

**General optimization problems**

- Algorithms are iterative: starting with an initial guess $\mathbf{x}_0$, obtain sequentially iterates $\mathbf{x}_k$ that hopefully converge to $\mathbf{x}^*$.
- Can be difficult to solve. Most algorithms involve some compromise.
- Often not clear which methods work best (may depend on our needs and the problem).

**Ideally** recognize the type/class of problem – some are relatively easier

- linear least-squares $f(\mathbf{x}) = \|A\mathbf{x} - \mathbf{b}\|^2 = \sum_{i=1}^{m}(\sum_{j=1}^{n} a_{ij}x_j - b_i)^2$
- convex problems

**Structure** is often hidden (e.g., the weak coupling allowing splitting in our *case study*).

## Convex Optimization

The objective function is convex if

$$f(\alpha \mathbf{x} + \beta \mathbf{y}) \leq \alpha f(\mathbf{x}) + \beta f(\mathbf{y})$$

for all $\alpha + \beta = 1$ and $\alpha, \beta \geq 0$.

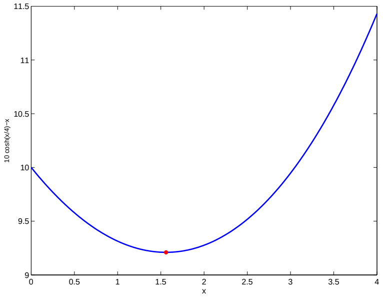
- Convexity generalizes linearity
- Includes many important problem classes (in particular, linear least-squares problems).
- Important feature: a local minimizer $\mathbf{x}^*$ is also a global minimizer.

## Minimizing a function in one variable

- Many issues simplify when $\mathbf{x} = x$ is scalar, $n = 1$.
- **Example**: find $x = x^*$ that minimizes

$$f(x) = 10\cosh(x/4) - x.$$

- From the figure below, this function has no zeros but does appear to have one minimizer around $x = 1.6$.



## Conditions for optimum and algorithm

- Necessary condition for an optimum:
  Suppose $f \in C^2$ and denote $g(x) = f'(x)$ (the "scalar gradient"). Then a zero of $g$ is a critical point of $f$, i.e., where
  $$f'(x^*) = 0.$$
  To be a minimizer or a maximizer, it is necessary for $x^*$ to be a critical point.
- Sufficient condition for an optimum:
  Denote $h(x) = g'(x) = f''(x)$ (the "scalar Hessian"). A critical point $x^*$ is a minimizer if also $h(x^*) > 0$.
- Hence, an algorithm for finding a minimizer is obtained by finding the roots of $g(x)$, then checking for each such root $x^*$ if also $h(x^*) > 0$.
- Note: rather than finding all roots of $f' = g$ first and checking for minimum condition later, can do things more carefully and wisely, e.g. by sticking to steps that decrease $f(x)$.

## Conditions for optimum and algorithm

- Necessary condition for an optimum:
  Suppose $f \in C^2$ and denote $g(x) = f'(x)$ (the "scalar gradient"). Then a zero of $g$ is a critical point of $f$, i.e., where
  $$f'(x^*) = 0.$$
  To be a minimizer or a maximizer, it is necessary for $x^*$ to be a critical point.
- Sufficient condition for an optimum:
  Denote $h(x) = g'(x) = f''(x)$ (the "scalar Hessian"). A critical point $x^*$ is a minimizer if also $h(x^*) > 0$.
- Hence, an algorithm for finding a minimizer is obtained by finding the roots of $g(x)$, then checking for each such root $x^*$ if also $h(x^*) > 0$.
- Note: rather than finding all roots of $f' = g$ first and checking for minimum condition later, can do things more carefully and wisely, e.g. by sticking to steps that decrease $f(x)$.

## Conditions for optimum and algorithm

- Necessary condition for an optimum:
  Suppose $f \in C^2$ and denote $g(x) = f'(x)$ (the "scalar gradient").
  Then a zero of $g$ is a critical point of $f$, i.e., where

  $$f'(x^*) = 0.$$

  To be a minimizer or a maximizer, it is necessary for $x^*$ to be a critical point.
- Sufficient condition for an optimum:
  Denote $h(x) = g'(x) = f''(x)$ (the "scalar Hessian"). A critical point $x^*$ is a minimizer if also $h(x^*) > 0$.
- Hence, an algorithm for finding a minimizer is obtained by finding the roots of $g(x)$, then checking for each such root $x^*$ if also $h(x^*) > 0$.
- Note: rather than finding all roots of $f' = g$ first and checking for minimum condition later, can do things more carefully and wisely, e.g. by sticking to steps that decrease $f(x)$.

## Conditions for optimum and algorithm

- Necessary condition for an optimum:
  Suppose $f \in C^2$ and denote $g(x) = f'(x)$ (the "scalar gradient"). Then a zero of $g$ is a critical point of $f$, i.e., where

  $$f'(x^*) = 0.$$

  To be a minimizer or a maximizer, it is necessary for $x^*$ to be a critical point.
- Sufficient condition for an optimum:
  Denote $h(x) = g'(x) = f''(x)$ (the "scalar Hessian"). A critical point $x^*$ is a minimizer if also $h(x^*) > 0$.
- Hence, an algorithm for finding a minimizer is obtained by finding the roots of $g(x)$, then checking for each such root $x^*$ if also $h(x^*) > 0$.
- Note: rather than finding all roots of $f' = g$ first and checking for minimum condition later, can do things more carefully and wisely, e.g. by sticking to steps that decrease $f(x)$.

## Example

To find a minimizer for

$$f(x) = 10\cosh(x/4) - x,$$

1. Calculate gradient

$$g(x) = f'(x) = 2.5\sinh(x/4) - 1$$

2. Find root of $g(x) = 0$ using any standard method, obtaining

$$x^* \approx 1.56014.$$

3. Second derivative

$$h(x) = 2.5/4\cosh(x/4) > 0 \quad \text{for all } x,$$

so $x^*$ is a minimizer.

## Multivariate Taylor expansion

- Consider the problem

$$\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x})$$

  Assume that $f(\mathbf{x})$ is smooth enough.
- Define gradient vector $\mathbf{g}(\mathbf{x}) = \nabla f(\mathbf{x})$ and Hessian matrix $H(\mathbf{x}) = \nabla^2 f(\mathbf{x})$ by

$$\nabla f(\mathbf{x}) = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix}, \quad \nabla^2 f(\mathbf{x}) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{pmatrix}.$$

- Taylor expansion near a point $\mathbf{x} \in \mathbb{R}^n$:

$$f(\mathbf{x}+\mathbf{p}) = f(\mathbf{x}) + \nabla f(\mathbf{x})^T \mathbf{p} + \frac{1}{2}\mathbf{p}^T \nabla^2 f(\mathbf{x})\mathbf{p} + \mathcal{O}(\|\mathbf{p}\|^3).$$

## Example

- Given the function

$$f(x_1, x_2) = x_1^4 - 2x_1^3x_2^2 + 4x_1x_2^3,$$

- the gradient at a point $\mathbf{x}$ is

$$\mathbf{g}(\mathbf{x}) = \nabla f(\mathbf{x}) = \begin{pmatrix} 4x_1^3 - 6x_1^2x_2^2 + 4x_2^3 \\ -4x_1^3x_2 + 12x_1x_2^2 \end{pmatrix};$$

- the Hessian matrix is

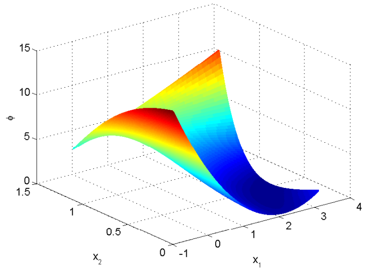
$$H(\mathbf{x}) = \nabla^2 f(\mathbf{x}) = \begin{pmatrix} 12x_1^2 - 12x_1x_2^2 & -12x_1^2x_2 + 12x_2^2 \\ -12x_1^2x_2 + 12x_2^2 & -4x_1^3 + 24x_1x_2 \end{pmatrix}.$$

## Critical points

- Taylor expansion near a suspected minimum point $\mathbf{x}^*$: in any direction $\mathbf{p}$ with magnitude $\|\mathbf{p}\|$ small enough, must have $f(\mathbf{x}^* + \mathbf{p}) \geq f(\mathbf{x}^*)$.
- Hence

$$\nabla f(\mathbf{x}^*) = \mathbf{0}.$$

  This defines a critical point.
- Similar condition also for a maximum or a saddle point.



## Conditions for unconstrained minimum

$$\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}).$$

Assume that $f(\mathbf{x})$ is smooth enough.

- A necessary condition for having a local minimum at $\mathbf{x}^*$ is that $\mathbf{x}^*$ be a critical point
$$\mathbf{g}(\mathbf{x}^*) = \nabla f(\mathbf{x}^*) = \mathbf{0},$$
*and* that the symmetric Hessian matrix $H(\mathbf{x}^*) = \nabla^2 f(\mathbf{x}^*)$ be positive semi-definite.
- A sufficient condition is that also $H(\mathbf{x}^*)$ be positive definite.
- **NB**: a symmetric matrix $H$ is **positive definite** if for any vector $\mathbf{y} \neq \mathbf{0}$, we have $\mathbf{y}^T H\mathbf{y} > 0$.

## Outline

- Case study: soft body deformation and calibration
- Unconstrained optimization
- Gradient descent methods
- Derivative-free (search) methods
- Nonconvex problems

## Descent direction

- At point $\mathbf{x}$ the vector $\mathbf{p}$ is a descent direction if

$$\mathbf{p}^T\mathbf{g}(\mathbf{x}) = \sum_{i=1}^{n} p_i \frac{\partial f}{\partial x_i} < 0.$$

- A small step in a descent direction gives reduction in the objective function:

$$f(\mathbf{x} + \alpha\mathbf{p}) < f(\mathbf{x})$$

for scalar step size $0 < \alpha \ll 1$.

- Therefore, we can construct an iterative method that keeps reducing $f$ until convergence by using descent directions and controlled step sizes.
- Starting from an initial guess $\mathbf{x}_0$, generate iterates $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k, \mathbf{x}_{k+1}, \ldots$ so that $f(\mathbf{x}_{k+1}) < f(\mathbf{x}_k)$.

## Descent direction

- At point $\mathbf{x}$ the vector $\mathbf{p}$ is a descent direction if
$$\mathbf{p}^T \mathbf{g}(\mathbf{x}) = \sum_{i=1}^{n} p_i \frac{\partial f}{\partial x_i} < 0.$$
- A small step in a descent direction gives reduction in the objective function:
$$f(\mathbf{x} + \alpha \mathbf{p}) < f(\mathbf{x})$$
 for scalar step size $0 < \alpha \ll 1$.
- Therefore, we can construct an iterative method that keeps reducing $f$ until convergence by using descent directions and controlled step sizes.
- Starting from an initial guess $\mathbf{x}_0$, generate iterates $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k, \mathbf{x}_{k+1}, \ldots$ so that $f(\mathbf{x}_{k+1}) < f(\mathbf{x}_k)$.

## Descent direction

- At point $\mathbf{x}$ the vector $\mathbf{p}$ is a descent direction if

$$\mathbf{p}^T \mathbf{g}(\mathbf{x}) = \sum_{i=1}^{n} p_i \frac{\partial f}{\partial x_i} < 0.$$

- A small step in a descent direction gives reduction in the objective function:

$$f(\mathbf{x} + \alpha \mathbf{p}) < f(\mathbf{x})$$

  for scalar step size $0 < \alpha \ll 1$.

- Therefore, we can construct an iterative method that keeps reducing $f$ until convergence by using descent directions and controlled step sizes.
- Starting from an initial guess $\mathbf{x}_0$, generate iterates $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k, \mathbf{x}_{k+1}, \ldots$ so that $f(\mathbf{x}_{k+1}) < f(\mathbf{x}_k)$.

## Gradient-based methods

Consider

$$\begin{aligned}\mathbf{x}_{k+1} &= \mathbf{x}_k + \alpha_k \mathbf{p}_k, \quad \text{where} \\ \mathbf{p}_k &= -B_k^{-1}\mathbf{g}(\mathbf{x}_k).\end{aligned}$$

If $B_k$ is symmetric positive definite then $\mathbf{p}_k$ is a descent direction. Note $\alpha_k > 0$.

1. Gradient descent: $B_k = I$. How to choose step size $\alpha_k$?
2. Newton: $B_k = \nabla^2 f(\mathbf{x}_k)$. Set $\alpha_k = 1$ or damped Newton: search for $\alpha_k \leq 1$ guaranteeing descent.
3. Secant, or quasi-Newton (e.g., BFGS).
4. Gauss-Newton for nonlinear least squares data fitting.
5. Stochastic gradient descent : the most popular method in machine learning.

## Gradient-based methods

Consider

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathbf{x}_k + \alpha_k \mathbf{p}_k, \quad \text{where} \\ \mathbf{p}_k &= -B_k^{-1}\mathbf{g}(\mathbf{x}_k). \end{aligned}$$

If $B_k$ is symmetric positive definite then $\mathbf{p}_k$ is a descent direction. Note $\alpha_k > 0$.

1. Gradient descent: $B_k = I$. How to choose step size $\alpha_k$?
2. Newton: $B_k = \nabla^2 f(\mathbf{x}_k)$. Set $\alpha_k = 1$ or damped Newton: search for $\alpha_k \leq 1$ guaranteeing descent.
3. Secant, or quasi-Newton (e.g., BFGS).
4. Gauss-Newton for nonlinear least squares data fitting.
5. Stochastic gradient descent : the most popular method in machine learning.

## Gradient-based methods

Consider

$$\begin{aligned}\mathbf{x}_{k+1} &= \mathbf{x}_k + \alpha_k \mathbf{p}_k, \quad \text{where}\\ \mathbf{p}_k &= -B_k^{-1}\mathbf{g}(\mathbf{x}_k).\end{aligned}$$

If $B_k$ is symmetric positive definite then $\mathbf{p}_k$ is a descent direction. Note $\alpha_k > 0$.

1. Gradient descent: $B_k = I$. How to choose step size $\alpha_k$?
2. Newton: $B_k = \nabla^2 f(\mathbf{x}_k)$. Set $\alpha_k = 1$ or damped Newton: search for $\alpha_k \leq 1$ guaranteeing descent.
3. Secant, or quasi-Newton (e.g., BFGS).
4. Gauss-Newton for nonlinear least squares data fitting.
5. Stochastic gradient descent : the most popular method in machine learning.

## Gradient-based methods

Consider

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathbf{x}_k + \alpha_k \mathbf{p}_k, \quad \text{where} \\ \mathbf{p}_k &= -B_k^{-1}\mathbf{g}(\mathbf{x}_k). \end{aligned}$$

If $B_k$ is symmetric positive definite then $\mathbf{p}_k$ is a descent direction. Note $\alpha_k > 0$.

1. Gradient descent: $B_k = I$. How to choose step size $\alpha_k$?
2. Newton: $B_k = \nabla^2 f(\mathbf{x}_k)$. Set $\alpha_k = 1$ or damped Newton: search for $\alpha_k \leq 1$ guaranteeing descent.
3. Secant, or quasi-Newton (e.g., BFGS).
4. Gauss-Newton for nonlinear least squares data fitting.
5. Stochastic gradient descent : the most popular method in machine learning.

## Gradient-based methods

Consider

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathbf{x}_k + \alpha_k \mathbf{p}_k, \quad \text{where} \\ \mathbf{p}_k &= -B_k^{-1}\mathbf{g}(\mathbf{x}_k). \end{aligned}$$

If $B_k$ is symmetric positive definite then $\mathbf{p}_k$ is a descent direction. Note $\alpha_k > 0$.

1. Gradient descent: $B_k = I$. How to choose step size $\alpha_k$?
2. Newton: $B_k = \nabla^2 f(\mathbf{x}_k)$. Set $\alpha_k = 1$ or damped Newton: search for $\alpha_k \leq 1$ guaranteeing descent.
3. Secant, or quasi-Newton (e.g., BFGS).
4. Gauss-Newton for nonlinear least squares data fitting.
5. Stochastic gradient descent : the most popular method in machine learning.

## Gradient-based methods

Consider

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathbf{x}_k + \alpha_k \mathbf{p}_k, \quad \text{where} \\ \mathbf{p}_k &= -B_k^{-1}\mathbf{g}(\mathbf{x}_k). \end{aligned}$$

If $B_k$ is symmetric positive definite then $\mathbf{p}_k$ is a descent direction. Note $\alpha_k > 0$.

1. Gradient descent: $B_k = I$. How to choose step size $\alpha_k$?
2. Newton: $B_k = \nabla^2 f(\mathbf{x}_k)$. Set $\alpha_k = 1$ or damped Newton: search for $\alpha_k \leq 1$ guaranteeing descent.
3. Secant, or quasi-Newton (e.g., BFGS).
4. Gauss-Newton for nonlinear least squares data fitting.
5. Stochastic gradient descent : the most popular method in machine learning.

## Nonlinear least squares

$$\min_{\mathbf{x}} f(\mathbf{x}) = \frac{1}{2}\|\phi(\mathbf{x}) - \mathbf{b}\|^2,$$

where $\mathbf{b}$ is data ($m$ values) and $\phi$ a nonlinear function of $n$ arguments $\mathbf{x}$.

Jacobian

$$A(\mathbf{x}) = \begin{pmatrix} \frac{\partial \phi_1}{\partial x_1} & \frac{\partial \phi_1}{\partial x_2} & \cdots & \frac{\partial \phi_1}{\partial x_n} \\ \frac{\partial \phi_2}{\partial x_1} & \frac{\partial \phi_2}{\partial x_2} & \cdots & \frac{\partial \phi_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \phi_{m-1}}{\partial x_1} & \frac{\partial \phi_{m-1}}{\partial x_2} & \cdots & \frac{\partial \phi_{m-1}}{\partial x_n} \\ \frac{\partial \phi_m}{\partial x_1} & \frac{\partial \phi_m}{\partial x_2} & \cdots & \frac{\partial \phi_m}{\partial x_n} \end{pmatrix}$$

Then

$$\mathbf{g}(\mathbf{x}) = \nabla f(\mathbf{x}) = A(\mathbf{x})^T(\phi(\mathbf{x}) - \mathbf{b}).$$

## Nonlinear least squares

$$\min_{\mathbf{x}} f(\mathbf{x}) = \frac{1}{2}\|\phi(\mathbf{x}) - \mathbf{b}\|^2,$$

where $\mathbf{b}$ is data ($m$ values) and $\phi$ a nonlinear function of $n$ arguments $\mathbf{x}$.
Jacobian

$$A(\mathbf{x}) = \begin{pmatrix} \frac{\partial \phi_1}{\partial x_1} & \frac{\partial \phi_1}{\partial x_2} & \cdots & \frac{\partial \phi_1}{\partial x_n} \\ \frac{\partial \phi_2}{\partial x_1} & \frac{\partial \phi_2}{\partial x_2} & \cdots & \frac{\partial \phi_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \phi_{m-1}}{\partial x_1} & \frac{\partial \phi_{m-1}}{\partial x_2} & \cdots & \frac{\partial \phi_{m-1}}{\partial x_n} \\ \frac{\partial \phi_m}{\partial x_1} & \frac{\partial \phi_m}{\partial x_2} & \cdots & \frac{\partial \phi_m}{\partial x_n} \end{pmatrix}$$

Then

$$\mathbf{g}(\mathbf{x}) = \nabla f(\mathbf{x}) = A(\mathbf{x})^T(\phi(\mathbf{x}) - \mathbf{b}).$$

## Gauss-Newton

- Define $B_k = A(\mathbf{x}_k)^T A(\mathbf{x}_k)$.
- So the iteration is defined by

$$\begin{aligned} \left[A(\mathbf{x}_k)^T A(\mathbf{x}_k)\right] \mathbf{p}_k &= -\nabla f(\mathbf{x}_k) = A(\mathbf{x}_k)^T(\mathbf{b} - \phi(\mathbf{x}_k)) \\ \mathbf{x}_{k+1} &= \mathbf{x}_k + \mathbf{p}_k . \end{aligned}$$

- These are normal equations for

$$\min_{\mathbf{p}} \frac{1}{2}\|A(\mathbf{x}_k)\mathbf{p} - (\mathbf{b} - \phi(\mathbf{x}_k))\|^2,$$

so at each iteration solve a linear least squares problem.

## (Weak) line search

- Suppose that $\mathbf{p}_k$ is a descent direction at $\mathbf{x}_k$. Then for $\alpha_k > 0$ small enough, $f(\mathbf{x}_k + \alpha_k \mathbf{p}_k) < f(\mathbf{x}_k)$.
- Here is a simple algorithm (Armijo) for determining step size $\alpha_k$, given descent direction $\mathbf{p}_k$:
  Starting from $\alpha = \alpha_{max}$, repeat until sufficient decrease in $f$ is obtained,
  $$\alpha \leftarrow \alpha/2.$$
- The result is $\alpha_k = \alpha$, and set $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$ .
- For Gauss-Newton, set $\alpha_{max} = 1$.
- For gradient descent, $B_k = I$, there is no obvious default value $\alpha_{max}$.

## (Weak) line search

- Suppose that $\mathbf{p}_k$ is a descent direction at $\mathbf{x}_k$. Then for $\alpha_k > 0$ small enough, $f(\mathbf{x}_k + \alpha_k \mathbf{p}_k) < f(\mathbf{x}_k)$.
- Here is a simple algorithm (Armijo) for determining step size $\alpha_k$, given descent direction $\mathbf{p}_k$:
  Starting from $\alpha = \alpha_{max}$, repeat until sufficient decrease in $f$ is obtained,
  $$\alpha \leftarrow \alpha/2.$$

- The result is $\alpha_k = \alpha$, and set $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$ .
- For Gauss-Newton, set $\alpha_{max} = 1$.
- For gradient descent, $B_k = I$, there is no obvious default value $\alpha_{max}$.

## (Weak) line search

- Suppose that $\mathbf{p}_k$ is a descent direction at $\mathbf{x}_k$. Then for $\alpha_k > 0$ small enough, $f(\mathbf{x}_k + \alpha_k \mathbf{p}_k) < f(\mathbf{x}_k)$.
- Here is a simple algorithm (Armijo) for determining step size $\alpha_k$, given descent direction $\mathbf{p}_k$:
  Starting from $\alpha = \alpha_{max}$, repeat until sufficient decrease in $f$ is obtained,
  $$\alpha \leftarrow \alpha/2.$$
- The result is $\alpha_k = \alpha$, and set $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$ .
- For Gauss-Newton, set $\alpha_{max} = 1$.
- For gradient descent, $B_k = I$, there is no obvious default value $\alpha_{max}$.

## (Weak) line search

- Suppose that $\mathbf{p}_k$ is a descent direction at $\mathbf{x}_k$. Then for $\alpha_k > 0$ small enough, $f(\mathbf{x}_k + \alpha_k \mathbf{p}_k) < f(\mathbf{x}_k)$.
- Here is a simple algorithm (Armijo) for determining step size $\alpha_k$, given descent direction $\mathbf{p}_k$:
  Starting from $\alpha = \alpha_{max}$, repeat until sufficient decrease in $f$ is obtained,

$$\alpha \leftarrow \alpha/2.$$

- The result is $\alpha_k = \alpha$, and set $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$ .
- For Gauss-Newton, set $\alpha_{max} = 1$.
- For gradient descent, $B_k = I$, there is no obvious default value $\alpha_{max}$.

## Gradient descent vs Gauss-Newton

- Gauss-Newton is often faster than gradient descent, significantly so when the condition number is large $\|A^TA\|\|(A^TA)^{-1}\| \gg 1$.
- The cost of carrying out the Gauss-Newton step can be significantly higher than that of gradient descent.
- However, line search if needed can be more costly for gradient descent.
- For very large problems, gradient descent may be favored also because of fast storage considerations. Such is the case for many problems in machine learning (ML), where fortunately the condition number is typically not very large either.

## Gradient descent vs Gauss-Newton

- Gauss-Newton is often faster than gradient descent, significantly so when the condition number is large $\|A^TA\|\|(A^TA)^{-1}\| \gg 1$.
- The cost of carrying out the Gauss-Newton step can be significantly higher than that of gradient descent.
- However, line search if needed can be more costly for gradient descent.
- For very large problems, gradient descent may be favored also because of fast storage considerations. Such is the case for many problems in machine learning (ML), where fortunately the condition number is typically not very large either.

## Gradient descent vs Gauss-Newton

- Gauss-Newton is often faster than gradient descent, significantly so when the condition number is large $\|A^TA\|\|(A^TA)^{-1}\| \gg 1$.
- The cost of carrying out the Gauss-Newton step can be significantly higher than that of gradient descent.
- However, line search if needed can be more costly for gradient descent.
- For very large problems, gradient descent may be favored also because of fast storage considerations. Such is the case for many problems in machine learning (ML), where fortunately the condition number is typically not very large either.

## Gradient descent vs Gauss-Newton

- Gauss-Newton is often faster than gradient descent, significantly so when the condition number is large $\|A^TA\|\|(A^TA)^{-1}\| \gg 1$.
- The cost of carrying out the Gauss-Newton step can be significantly higher than that of gradient descent.
- However, line search if needed can be more costly for gradient descent.
- For very large problems, gradient descent may be favored also because of fast storage considerations. Such is the case for many problems in machine learning (ML), where fortunately the condition number is typically not very large either.

## Calculating the gradient

- Suppose that the gradient $\mathbf{g}(\mathbf{x}) = \nabla f(\mathbf{x})$ exists, but is hard to derive analytically (for instance, our case study!).
- It is possible to consider methods that automatically attempt to approximate the gradient from values of $f$.

1. Numerical differencing: for some $0 < h \ll 1$,

$$\frac{\partial f}{\partial x_j}(\mathbf{y}) \approx \frac{f(y_1, \ldots, y_{j-1}, y_j + h, y_{j+1}, \ldots, y_n) - f(\mathbf{y})}{h}, \quad j = 1, 2, \ldots, n.$$

   - Conceptually simple, and can be very effective for mid-range $n$ values.
   - However, the cost is $n+1$ function evaluations per gradient, which can be prohibitive for large $n$.
   - Moreover, noise is amplified by a factor of $\mathcal{O}(h^{-1})$: **so avoid using this directly on noisy functions** $f$.
2. More special-purpose possibility: automatic differentiation.

In MATLAB: `fminunc` has the option of not requiring derivatives.

## Calculating the gradient

- Suppose that the gradient $\mathbf{g}(\mathbf{x}) = \nabla f(\mathbf{x})$ exists, but is hard to derive analytically (for instance, our case study!).
- It is possible to consider methods that automatically attempt to approximate the gradient from values of $f$.
  1. Numerical differencing: for some $0 < h \ll 1$,

$$\frac{\partial f}{\partial x_j}(\mathbf{y}) \approx \frac{f(y_1, \ldots, y_{j-1}, y_j + h, y_{j+1}, \ldots, y_n) - f(\mathbf{y})}{h}, \quad j = 1, 2, \ldots, n.$$

     - Conceptually simple, and can be very effective for mid-range $n$ values.
     - However, the cost is $n+1$ function evaluations per gradient, which can be prohibitive for large $n$.
     - Moreover, noise is amplified by a factor of $\mathcal{O}(h^{-1})$: **so avoid using this directly on noisy functions $f$.**
  2. More special-purpose possibility: automatic differentiation.

In MATLAB: `fminunc` has the option of not requiring derivatives.

## Outline

- Case study: soft body deformation and calibration
- Unconstrained optimization
- Gradient descent methods
- Derivative-free (search) methods
- Nonconvex problems

## Derivative-free methods

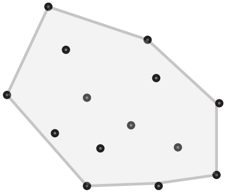
1. Questions:
   - What if the function $f(\mathbf{x})$ is continuous but not differentiable? Or we don't have the gradient?
   - Can't we just sample $f$ at some locations methodically and descend in this way towards a local minimum?
   - Could this avoid poor conditioning and skip local minima?
2. Answers:
   - Yes, indeed, such search methods exist! However, note that in general:
   - Convergence can be very slow unless the dimension $n$ is small. (In fact, these methods are good only for small problems.)
   - Existing theory for such methods is not as complete as before.
   - In the ML context stochastic gradient methods are more popular.

## Derivative-free methods

1. Questions:
   - What if the function $f(\mathbf{x})$ is continuous but not differentiable? Or we don't have the gradient?
   - Can't we just sample $f$ at some locations methodically and descend in this way towards a local minimum?
   - Could this avoid poor conditioning and skip local minima?
2. Answers:
   - Yes, indeed, such search methods exist! However, note that in general:
   - Convergence can be very slow unless the dimension $n$ is small. (In fact, these methods are good only for small problems.)
   - Existing theory for such methods is not as complete as before.
   - In the ML context stochastic gradient methods are more popular.

## The Nelder-Mead algorithm: overview

The convex hull of a set of points is a simplex.

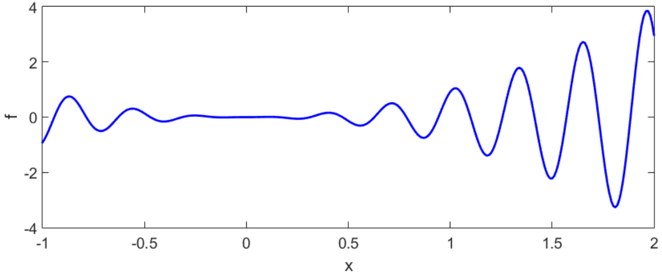


The Nelder-Mead algorithm (1965) keeps track of $n+1$ points (vectors) in $\Re^n$. At each iteration, we seek to replace the point with worst value of $f$ with another point that has a better $f$-value, obtained by reflecting, expanding or contracting the simplex along the line between the worst vertex and the centroid of the other vertices. If no better point is found in this way then shrink the simplex by moving all points toward the best current one.

## Outline

- Case study: soft body deformation and calibration
- Unconstrained optimization
- Gradient descent methods
- Derivative-free (search) methods
- Nonconvex problems

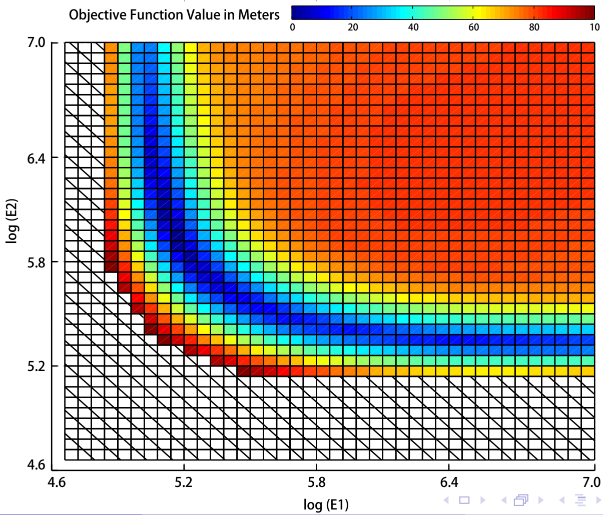
## What if the problem is non-convex?



- All our gradient-based algorithms are local: starting at point $\mathbf{x}_0$ in some locally convex basin, they are likely to converge to the corresponding local minimum.
- In such a situation, gradient-based methods may no longer be always superior.
- Ad hoc approach: choose many initial guesses randomly and solve for each in parallel.

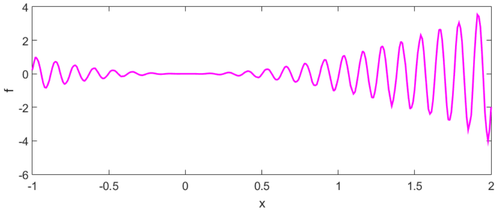
## Does this happen in practice?

In our *case study* (deformable body calibration) the optimal Young moduli can be in a narrow valley.

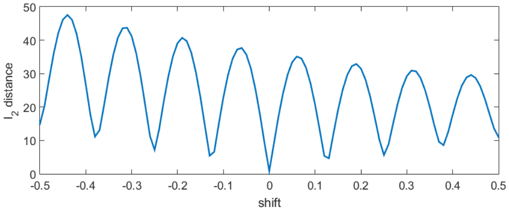


## Another example

Data $\hat{q}_t$ synthesized from function with 0 shift with added 5% noise:



$\ell_2$-distance of data from same function $q_t$ with shift:



## Deformable body calibration

- Indeed in [Wang et al. '15] we used a derivative-free method.
- Therefore we were constrained to consider objects that are not far from homogeneous (because of problem size).
- For simple objects can change tack completely, and consider direct inversion: inverse harmonics [Mandelstahm & Taylor, 1997; Chen, Levin, Matusik & Kaufman, 2017].
- If we are able to modify the objective function responsibly (e.g., by relaxed optimal mass transport (OMT)) to enlarge the convex region basins, then a switch to gradient-based methods will naturally follow [Froese, Engquist, Haber]. (Work in progress.)
- Lots of recent work on "convexification" – exploit special structure in given problem.

## Deformable body calibration

- Indeed in [Wang et al. '15] we used a derivative-free method.
- Therefore we were constrained to consider objects that are not far from homogeneous (because of problem size).
- For simple objects can change tack completely, and consider direct inversion: inverse harmonics [Mandelstahm & Taylor, 1997; Chen, Levin, Matusik & Kaufman, 2017].
- If we are able to modify the objective function responsibly (e.g., by relaxed optimal mass transport (OMT)) to enlarge the convex region basins, then a switch to gradient-based methods will naturally follow [Froese, Engquist, Haber]. (Work in progress.)
- Lots of recent work on "convexification" – exploit special structure in given problem.

## Deformable body calibration

- Indeed in [Wang et al. '15] we used a derivative-free method.
- Therefore we were constrained to consider objects that are not far from homogeneous (because of problem size).
- For simple objects can change tack completely, and consider direct inversion: inverse harmonics [Mandelstahm & Taylor, 1997; Chen, Levin, Matusik & Kaufman, 2017].
- If we are able to modify the objective function responsibly (e.g., by relaxed optimal mass transport (OMT)) to enlarge the convex region basins, then a switch to gradient-based methods will naturally follow [Froese, Engquist, Haber]. (Work in progress.)
- Lots of recent work on "convexification" – exploit special structure in given problem.

## Conclusions

- Soft object calibration and simulation pose interesting, challenging computational problems
- Physics-based algorithms can occasionally produce impressive visual results at affordable cost, allowing interactive response rates.
- Care must be taken, though, to avoid extending conclusion beyond reason.
- For the SI/BE and generalized $\alpha$ methods, our simple 1D analysis generally agrees with observed physics-based simulations.