Numerical Optimization

Instructor : Sung Chan Jun

Week #9 : October 28 – November 1, 2019

Announcement

- Final Exam (Rescheduled)
    - Date and Time
        - December 6 (Friday), 2019 7:00 PM – 8:30 PM

Course Syllabus (tentative)

1st week	*Sept. 2, 4*	*Introduction of optimization*	
2nd week	*Sept. 9, 11*	*Univariate Optimization*	
3rd week	*Sept. 16, 18*	*Univariate Optimization*	
4th week	*Sept. 23, 25*	*Unconstrained Multivariate Optimization*	
5th week	*Sept. 30, Oct. 2*	*Unconstrained Multivariate Optimization*	
6th week	*Oct. 7, 9*	*Unconstrained Multivariate Optimization*	*National Holiday (Oct. 9)*
7th week	*Oct. 14, 16*	*Unconstrained Multivariate Optimization*	*Midterm (Oct. 16)*
8th week	*Oct. 21, 23*	*Unconstrained Multivariate Optimization*	

Numerical Optimization (2019 Fall)

BioComputing

Course Syllabus (tentative)

9th week	Oct. 28, 30	Constrained Multivariate Optimization	
10th week	Nov. 4, 6	Constrained Multivariate Optimization	
11th week	Nov. 11, 13	Constrained Multivariate Optimization	
12th week	Nov. 18, 20	Global Optimization	
13th week	Nov. 25, 27	Global Optimization	
14th week	Dec. 2, 4	Global Optimization, Wrap-up	Final Exam (Dec. 6)
15th week	Dec. 9	Wrap-up	

Numerical Optimization (2019 Fall)

BioComputing

Recall Last Week

- Quasi-Newton's Method : BFGS update
  - BFGS update (Broyden, Fletcher, Goldfarb, and Shanno)

$$\mathbf{B}_{k+1} = \mathbf{B}_k - \frac{\mathbf{B}_k \mathbf{s}_k \mathbf{s}_k^T \mathbf{B}_k}{\mathbf{s}_k^T \mathbf{B}_k \mathbf{s}_k} + \frac{\mathbf{y}_k \mathbf{y}_k^T}{\mathbf{y}_k^T \mathbf{s}_k}, \text{ assuming } \mathbf{s}_k^T \mathbf{y}_k > 0$$

$$\mathbf{s}_k := \mathbf{x}_{k+1} - \mathbf{x}_k, \ \mathbf{y}_k := \nabla f_{k+1} - \nabla f_k$$

  - Inverse verison of BFGS

$$\mathbf{B}_k \mathbf{p}_k = -\nabla f(\mathbf{x}_k) \Rightarrow \mathbf{p}_k = -\mathbf{B}_k^{-1} \nabla f(\mathbf{x}_k)$$

Assuming $\mathbf{D}_k := \mathbf{B}_k^{-1}$

$$\mathbf{D}_{k+1} = (\mathbf{I} - \rho_k \mathbf{s}_k \mathbf{y}_k^T) \mathbf{D}_k (\mathbf{I} - \rho_k \mathbf{y}_k \mathbf{s}_k^T) + \rho_k \mathbf{s}_k \mathbf{s}_k^T$$

$$\rho_k := 1/(\mathbf{y}_k^T \mathbf{s}_k)$$

(Sherman-Morrison Identity)

- If **A** is nonsingular and **c**, **d** are n x 1 matrices, then

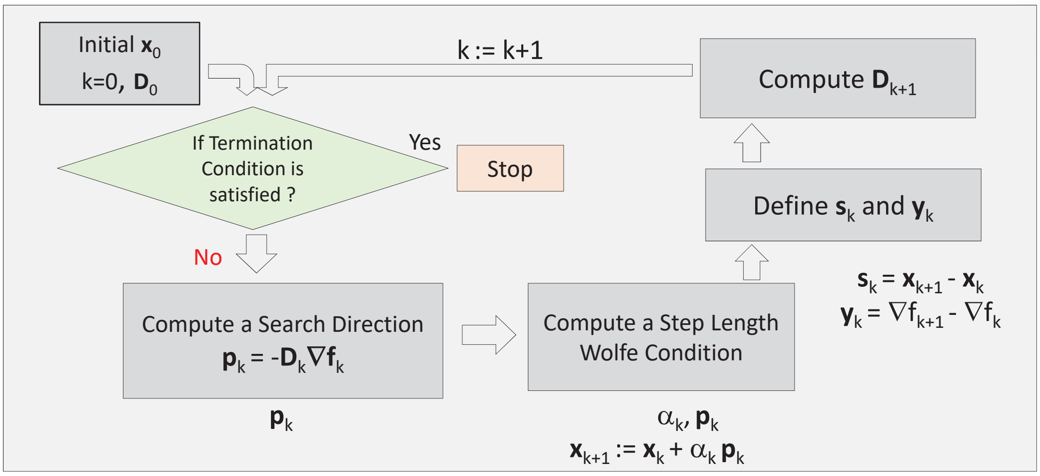
$$(\mathbf{A} + \mathbf{c}\mathbf{d}^T)^{-1} = \mathbf{A}^{-1} - \frac{\mathbf{A}^{-1}\mathbf{c}\mathbf{d}^T\mathbf{A}^{-1}}{1 + \mathbf{d}^T\mathbf{A}^{-1}\mathbf{c}} \quad \text{when} \quad 1 + \mathbf{d}^T\mathbf{A}^{-1}\mathbf{c} \neq 0$$

Recall Last Week

- Quasi-Newton's : BFGS
  - Remarks
    - Wolfe condition yields $\mathbf{s}_k^T \mathbf{y}_k > 0$.
    - If $\mathbf{B}_k$ and $\mathbf{D}_k$ are positive definite, then so are $\mathbf{B}_{k+1}$ and $\mathbf{D}_{k+1}$.
    - Different variants are obtained by different choices of weighting matrix **W**.
    - Bad situations : when $\mathbf{s}_k^T \mathbf{y}_k$ is so tiny
      - Good news : BFGS has effective self-correcting property even if $\mathbf{D}_k$ is a poor approximation.
    - It is known that BFGS is the most effective among them.

Recall Last Week

- Quasi-Newton's : BFGS algorithm



- Quasi-Newton's : Broyden Class

$$\mathbf{B}_{k+1} = \mathbf{B}_k - \frac{\mathbf{B}_k \mathbf{s}_k \mathbf{s}_k^T \mathbf{B}_k}{\mathbf{s}_k^T \mathbf{B}_k \mathbf{s}_k} + \frac{\mathbf{y}_k \mathbf{y}_k^T}{\mathbf{y}_k^T \mathbf{s}_k} + \phi_k (\mathbf{s}_k^T \mathbf{B}_k \mathbf{s}_k) \mathbf{v}_k \mathbf{v}_k^T, \quad \phi_k \text{ is a scalar and } \mathbf{v}_k = \left[\frac{\mathbf{y}_k}{\mathbf{y}_k^T \mathbf{s}_k} - \frac{\mathbf{B}_k \mathbf{s}_k}{\mathbf{s}_k^T \mathbf{B}_k \mathbf{s}_k} \right]$$

  - $\phi_k = 0$ (BFGS) and $\phi_k = 1$ (DFP) , $\mathbf{B}_{k+1} = (1-\phi_k)\, \mathbf{B}^{BFGS}_{k+1} + \phi_k \mathbf{B}^{DFP}_{k+1}$, $\phi_k \in (0,1)$

Recall Last Week

- Derivative Based Methods

Method of Steepest Descent	Newton's Method	Quasi Newton's Method
Direction $\mathbf{p}_k = -\nabla f(\mathbf{x}_k)$ • Global convergence • Slow convergence near minimum	Direction $\mathbf{p}_k = -(\nabla^2 f(\mathbf{x}_k))^{-1} \nabla f(\mathbf{x}_k)$ • Fast convergence (quadratic) • Require expensive Hessian computing every iteration	Direction $\mathbf{p}_k = -\mathbf{B}_k^{-1} \nabla f(\mathbf{x}_k)$ $\mathbf{B}_k \approx (\nabla^2 f(\mathbf{x}_k))$ • Relatively fast convergence close to Newton's • Do not require Hessian computing

Numerical Optimization (2019 Fall)

BioComputing

Recall Last Week

- Conjugate Gradient Method (CG)
  - Iterative method to solve a linear system $\mathbf{Ax} = \mathbf{b}$ for a square symmetric positive definite matrix $\mathbf{A}$.
  - Solving linear system $\Leftrightarrow$ Solving minimization problem

    $\boxed{\mathbf{Ax} = \mathbf{b}}$ $\quad$ $\boxed{\min [\tfrac{1}{2}\mathbf{x}^T\mathbf{Ax} - \mathbf{b}^T\mathbf{x}]}$
  - Conjugacy
    - A set of nonzero vectors $\{\mathbf{p}_0, \mathbf{p}_1, \dots, \mathbf{p}_L\}$ is conjugate with respect to symmetric positive definite matrix $\mathbf{A}$ if $\mathbf{p}_i^T \mathbf{A} \mathbf{p}_j = 0$, for all $i \neq j$.
  - Conjugate direction methods

    For given a set of conjugate directions $\{\mathbf{p}_0, \mathbf{p}_1, \dots, \mathbf{p}_{n-1}\}$ with respect to a symmetric positive definite matrix $\mathbf{A}$ (n x n), the sequence $\{\mathbf{x}_k\}$ by setting $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k\mathbf{p}_k$ converges to the minimum of the quadratic convex function ($f(\mathbf{x}) = \tfrac{1}{2}\mathbf{x}^T\mathbf{Ax} - \mathbf{b}^T\mathbf{x}$) within at most n steps when α_k is given by exact search.

BioComputing

Numerical Optimization (2019 Fall)

Recall Last Week

- Conjugate Gradient Method (CG)
  - Consider convex quadratic function $f(\mathbf{x}) = \tfrac{1}{2}\mathbf{x}^T\mathbf{Ax} - \mathbf{b}^T\mathbf{x}$.
    - Motivation : Present a new conjugate direction ($\mathbf{p}_k$) in terms of residue ($\mathbf{r}_k := \mathbf{Ax}_k - \mathbf{b}$) and the previous conjugate direction ($\mathbf{p}_{k-1}$) as follows:

      $$\mathbf{p}_k = -\mathbf{r}_k + \beta_k\mathbf{p}_{k-1}$$
    - Conjugate gradient method is generating conjugate direction for each iteration, so it is a special case of conjugate direction method.
    - How to generate conjugate directions?
      - determine β_k in order that a new vector $\mathbf{p}_k = -\mathbf{r}_k + \beta_k\mathbf{p}_{k-1}$ is a conjugate with respect to $\mathbf{A}$.
      - So β_k is estimated by $\beta_k = \dfrac{r_k^T A p_{k-1}}{p_{k-1}^T A p_{k-1}}$.

BioComputing

Numerical Optimization (2019 Fall)

Recall Last Week

- Conjugate Gradient Method (CG)
  - Standard CG Algorithm ($f(\mathbf{x}) = ½\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{b}^T\mathbf{x}$)
    - Given $\mathbf{x}_0$, Set k:=0, $\mathbf{r}_0 := \mathbf{A}\mathbf{x}_0 - \mathbf{b}$, $\mathbf{p}_0 := -\mathbf{r}_0$ (initial search direction is $-\nabla f(\mathbf{x}_0)$)
    - While $\mathbf{r}_k \neq \mathbf{0}$

$$\alpha_k := -\mathbf{r}_k^T\mathbf{p}_k \big/ \mathbf{p}_k^T\mathbf{A}\mathbf{p}_k \Rightarrow \mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k\mathbf{p}_k$$

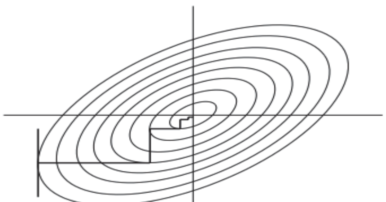
Determine step length (exact line search)

$$\mathbf{r}_{k+1} := \mathbf{A}\mathbf{x}_{k+1} - \mathbf{b} \Rightarrow \beta_{k+1} := \mathbf{r}_{k+1}^T\mathbf{A}\mathbf{p}_k \big/ \mathbf{p}_k^T\mathbf{A}\mathbf{p}_k$$

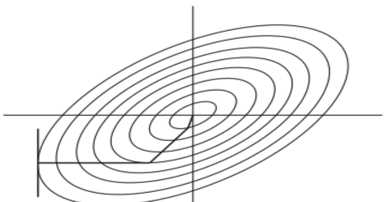
Compute residue

$$\mathbf{p}_{k+1} := -\mathbf{r}_{k+1} + \beta_{k+1}\mathbf{p}_k \Rightarrow k := k+1$$

Search a new direction



Method of Steepest Descent



Conjugate Gradient Method

Multivariate Optimization: Conjugate Gradient Method

- CG properties
  - Search directions are conjugate with respect to matrix **A**.
  - Residues $\mathbf{r}_i$ are mutually orthogonal,

    that is, $\mathbf{r}_k^T\mathbf{r}_i = \mathbf{r}_k \cdot \mathbf{r}_i = 0$ for i = 0, 1, …, k-1.
  - Residue $\mathbf{r}_k$ and search direction $\mathbf{p}_i$ are orthogonal,

    that is, $\mathbf{r}_k^T\mathbf{p}_i = \mathbf{r}_k \cdot \mathbf{p}_i = 0$ for i = 0, 1, …, k-1.
  - Identities

$$\mathbf{r}_{k+1}^T\mathbf{A}\mathbf{p}_k = \mathbf{r}_{k+1}^T\mathbf{r}_{k+1} / \alpha_k$$
$$\mathbf{p}_k^T\mathbf{A}\mathbf{p}_k = \mathbf{r}_k^T\mathbf{r}_k / \alpha_k$$

Multivariate Optimization: Conjugate Gradient Method

- Standard CG Algorithm
  - Given $\mathbf{x}_0$
  - Set $\mathbf{r}_0 := \mathbf{A}\mathbf{x}_0 - \mathbf{b}$, $\mathbf{p}_0 := -\mathbf{r}_0$, $k := 0$
  - While $\mathbf{r}_k \neq 0$

$$\boxed{\begin{array}{ll} \alpha_k := -\mathbf{r}_k^T\mathbf{p}_k / \mathbf{p}_k^T\mathbf{A}\mathbf{p}_k & \Rightarrow \mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k\mathbf{p}_k \\ \mathbf{r}_{k+1} := \mathbf{A}\mathbf{x}_{k+1} - \mathbf{b} & \Rightarrow \beta_{k+1} := \mathbf{r}_{k+1}^T\mathbf{A}\mathbf{p}_k / \mathbf{p}_k^T\mathbf{A}\mathbf{p}_k \\ \mathbf{p}_{k+1} := -\mathbf{r}_{k+1} + \beta_{k+1}\mathbf{p}_k & \Rightarrow k := k+1 \end{array}}$$

- Practical CG Algorithm
  - Given $\mathbf{x}_0$
  - Set $\mathbf{r}_0 := \mathbf{A}\mathbf{x}_0 - \mathbf{b}$, $\mathbf{p}_0 := -\mathbf{r}_0$, $k := 0$
  - While $\mathbf{r}_k \neq 0$

$$\boxed{\begin{array}{ll} \alpha_k := \mathbf{r}_k^T\mathbf{r}_k / \mathbf{p}_k^T\mathbf{A}\mathbf{p}_k & \Rightarrow \mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k\mathbf{p}_k \\ \mathbf{r}_{k+1} := \mathbf{r}_k + \alpha_k\mathbf{A}\mathbf{p}_k & \Rightarrow \beta_{k+1} := \mathbf{r}_{k+1}^T\mathbf{r}_{k+1} / \mathbf{r}_k^T\mathbf{r}_k \\ \mathbf{p}_{k+1} := -\mathbf{r}_{k+1} + \beta_{k+1}\mathbf{p}_k & \Rightarrow k := k+1 \end{array}}$$

$$\begin{aligned} \mathbf{r}_{k+1}^T\mathbf{A}\mathbf{p}_k &= \mathbf{r}_{k+1}^T\mathbf{r}_{k+1} / \alpha_k \\ \mathbf{p}_k^T\mathbf{A}\mathbf{p}_k &= \mathbf{r}_k^T\mathbf{r}_k / \alpha_k \end{aligned}$$

These identities are applied here.

Multivariate Optimization: Conjugate Gradient Method

- Convergence of CG
  - It converges within N-iterations when **A** is a symmetric p·d matrix of size N x N.
- Convergence rate of CG
  - When **A** has eigenvalues $\lambda_1 \le \lambda_2 \le \dots \le \lambda_N$,

$$\boxed{\left\|\mathbf{x}_k - \mathbf{x}^*\right\|_A \le 2\left(\frac{\sqrt{\kappa(\mathbf{A})}-1}{\sqrt{\kappa(\mathbf{A})}+1}\right)^k \left\|\mathbf{x}_0 - \mathbf{x}^*\right\|_A, \quad \kappa(\mathbf{A}) = \lambda_N / \lambda_1}$$

  - CG convergence depends on clustering of eigenvalues of **A**.
    - When κ(**A**) is big enough, i.e. eigenvalues are widely scattered,
      - It converges slowly.
    - When κ(**A**) is around 1, i.e. eigenvalues are well clustered,
      - It converges fast.

Multivariate Optimization:
Conjugate Gradient Method

- How to speed-up CG when CG convergence is slow
  - One idea
    - To use preconditioner 'symmetric positive definte matrix **M**'
    - Transform original problem into new problem

$$\mathbf{A}\,\mathbf{x} = \mathbf{b} \;\Rightarrow\; (\mathbf{M}^{-1}\mathbf{A})\,\mathbf{x} = \mathbf{M}^{-1}\mathbf{b} \quad \text{or}$$

$$\mathbf{A}\,\mathbf{x} = \mathbf{b} \;\Rightarrow\; (\mathbf{M}^{-1}\mathbf{A}\,\mathbf{M}^{-T})\,\mathbf{x}^{\wedge} = \mathbf{M}^{-1}\mathbf{b} \;\text{ and } \mathbf{x}^{\wedge} = \mathbf{M}^{T}\mathbf{x}$$

    - In order for $\kappa(\mathbf{M}^{-1}\mathbf{A})$ or $\kappa(\mathbf{M}^{-1}\mathbf{A}\,\mathbf{M}^{-T})$ to be close to 1, **M** can be chosen properly, then CG can be faster than before.

Numerical Optimization (2019 Fall)

Multivariate Optimization:
Conjugate Gradient Method

- How to choose preconditioner **M**?
  - **M** should be symmetric and positive definite.
  - **M** should be such that $\mathbf{M}^{T}\mathbf{x} = \mathbf{x}^{\wedge}$ can be solved efficiently.
  - **M** should approximate **A** in the sense that $\left\| \mathbf{I} - \mathbf{M}^{-1}\mathbf{A} \right\| << 1$
- Examples
  - For the decomposition $\mathbf{A} = \mathbf{L} + \mathbf{D} + \mathbf{L}^{T}$ (**L**: strictly low triangular, **D**: diagonal) of the symmetric positive definite matrix **A**
    - $\mathbf{M} = \mathbf{D}$: 'Jacobi' preconditioning,
    - $\mathbf{M} = \mathbf{L} + \mathbf{D}$: 'Gauss-Seidel' preconditioning,
    - $\mathbf{M} = (\mathbf{D} + \omega\mathbf{L})/\omega$, $(\omega > 0)$: 'SOR' preconditioning.
  - $\mathbf{M} = \mathbf{H}\,\mathbf{H}^{T}$, where **H** is 'close' to **L**. 'Incomplete Cholesky factorization'

Numerical Optimization (2019 Fall)

Multivariate Optimization: Conjugate Gradient Method

- More Common Preconditioners (Preconditioning)
  - Incomplete LU
  - Algebraic multi-grid (AMG)
  - Inverse based multi-level Incomplete LU

Multivariate Optimization: Conjugate Gradient Method

- Nonlinear CG (Flectcher-Reeves: CG-FR)

  Consider nonlinear function f($\mathbf{x}$)

  - Given $\mathbf{x}_0$, Evaluate $f_0 := f(\mathbf{x}_0)$, $\nabla f_0 := \nabla f(\mathbf{x}_0)$.
  - Set $\mathbf{p}_0 := -\nabla f_0$, $k := 0$
  - While $\nabla f_k \neq 0$

$$\boxed{\begin{aligned}
&\text{compute } \alpha_k \\
&\mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k \mathbf{p}_k. \\
&\text{Evaluate } \nabla f_{k+1} \\
&\beta^{FR}_{k+1} := \nabla f^T_{k+1} \nabla f_{k+1} \big/ \nabla f^T_k \nabla f_k \\
&\mathbf{p}_{k+1} := -\nabla f_{k+1} + \beta^{FR}_{k+1} \mathbf{p}_k \\
&k := k+1
\end{aligned}}$$

Numerical Optimization (2019 Fall)

BioComputing

Multivariate Optimization:
Conjugate Gradient Method

- Comparison : Linear CG and Nonlinear CG

Linear CG	Nonlinear CG (CG-FR)
Given $\mathbf{x}_0$, $f(\mathbf{x}) = 1/2\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{b}^T\mathbf{x}$	Given $\mathbf{x}_0$, Evaluate $f_0 := f(\mathbf{x}_0)$, $\nabla f_0 := \nabla f(\mathbf{x}_0)$.
Set $\mathbf{r}_0 := \mathbf{A}\mathbf{x}_0 - \mathbf{b}$, $\mathbf{p}_0 := -\mathbf{r}_0$, $k := 0$	Set $\mathbf{p}_0 := -\nabla f_0$, $k := 0$
While $\mathbf{r}_k \neq 0$	While $\nabla f_k \neq 0$
$\alpha_k := \mathbf{r}_k^T\mathbf{r}_k / \mathbf{p}_k^T\mathbf{A}\mathbf{p}_k$	compute α_k
$\mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k\mathbf{p}_k$	$\mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k\mathbf{p}_k$.
$\mathbf{r}_{k+1} := \mathbf{r}_k + \alpha_k\mathbf{A}\mathbf{p}_k$	Evaluate ∇f_{k+1}
$\beta_{k+1} := \mathbf{r}_{k+1}^T\mathbf{r}_{k+1} / \mathbf{r}_k^T\mathbf{r}_k$	$\beta_{k+1}^{FR} := \nabla f_{k+1}^T\nabla f_{k+1} / \nabla f_k^T\nabla f_k$
$\mathbf{p}_{k+1} := -\mathbf{r}_{k+1} + \beta_{k+1}\mathbf{p}_k$	$\mathbf{p}_{k+1} := -\nabla f_{k+1} + \beta_{k+1}^{FR}\mathbf{p}_k$
$k := k+1$	$k := k+1$

Note that we observe $\nabla f(\mathbf{x}_k) = \mathbf{A}\mathbf{x}_k - \mathbf{b}$ = residue = $\mathbf{r}_k$

Multivariate Optimization:
Conjugate Gradient Method

- CG-FR
  - If $f(\mathbf{x})$ is a convex function and step length is exact, CG-FR comes to linear CG.
  - For nonlinear $f(\mathbf{x})$, exact search is not easy. So, strong Wolfe condition is recommendable for descent condition.
  - Strong Wolfe condition

$$
\begin{aligned}
& f(\mathbf{x}_k + \alpha_k\mathbf{p}_k) - f(\mathbf{x}_k) \le c_1\alpha_k\nabla f(\mathbf{x}_k)\cdot\mathbf{p}_k \\
& \left|\nabla f(\mathbf{x}_k + \alpha_k\mathbf{p}_k)\cdot\mathbf{p}_k\right| \le -c_2\nabla f(\mathbf{x}_k)\cdot\mathbf{p}_k, \quad 0 < c_1 < c_2 < 1/2
\end{aligned}
$$

Recall : Wolfe condition

$$
\begin{aligned}
& f(\mathbf{x}_k + \alpha_k\mathbf{p}_k) - f(\mathbf{x}_k) \le c_1\alpha_k\nabla f(\mathbf{x}_k)\cdot\mathbf{p}_k \\
& \nabla f(\mathbf{x}_k + \alpha_k\mathbf{p}_k)\cdot\mathbf{p}_k \ge c_2\nabla f(\mathbf{x}_k)\cdot\mathbf{p}_k, \quad 0 < c_1 < c_2 < 1
\end{aligned}
$$

Numerical Optimization (2019 Fall)

Multivariate Optimization: Conjugate Gradient Method

- Some issues on CGs
  - Linear CG
    - It is easy to find exact step length
    - It terminates within finite iterations
  - Nonlinear CG
    - It is common to do inexact search
    - It needs restart strategies
      - After the given number of iteration,

      set β_k to 0 when two gradients are far from orthogonal by such a test:

$$\frac{\left|\nabla f_k^T \nabla f_{k-1}\right|}{\nabla f_k^T \nabla f_k} \geq \nu > 0, \ \nu \text{ is a given number.}$$

Numerical Optimization (2019 Fall)

BioComputing

Multivariate Optimization: Conjugate Gradient Method

- Variants of nonlinear CG
  - Polak-Ribiere (CG-PR) method

$$\beta_{k+1}^{PR} := \nabla f_{k+1}^T (\nabla f_{k+1} - \nabla f_k) \Big/ \nabla f_k^T \nabla f_k$$

  - Hestenes-Stiefel (CG-HS) method

$$\beta_{k+1}^{HS} := \nabla f_{k+1}^T (\nabla f_{k+1} - \nabla f_k) \Big/ (\nabla f_{k+1} - \nabla f_k)^T p_k$$

Numerical Optimization (2019 Fall)

BioComputing

Multivariate Optimization:
Conjugate Gradient Method

When f(**x**) = ½$\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{b}^T\mathbf{x}$ and **A** is non-symmetric

- Bi-conjugate Gradient Methods(BICG)
  - Generate an auxiliary function $g(\tilde{\mathbf{x}}) = 1/2\tilde{\mathbf{x}}^T\mathbf{A}^T\tilde{\mathbf{x}} - \tilde{\mathbf{b}}^T\tilde{\mathbf{x}}$
  - Do the CG procedure together to optimize f(**x**) and g(**x**) at the same time.
  - Given $\mathbf{x}_0$, $\tilde{\mathbf{x}}_0$, $\tilde{\mathbf{b}}$, set $\mathbf{r}_0 := \mathbf{A}\mathbf{x}_0 - \mathbf{b}$ & $\tilde{\mathbf{r}}_0 := \mathbf{A}^T\tilde{\mathbf{x}}_0 - \tilde{\mathbf{b}}$
  - Generate two conjugate sequences:

$$\boxed{\begin{array}{ll} \mathbf{r}_{k+1} := \mathbf{r}_k + \alpha_k\mathbf{A}\mathbf{p}_k & \mathbf{p}_{k+1} = -\mathbf{r}_{k+1} + \beta_{k+1}\mathbf{p}_k \\ \tilde{\mathbf{r}}_{k+1} := \tilde{\mathbf{r}}_k + \alpha_k\mathbf{A}^T\tilde{\mathbf{p}}_k & \tilde{\mathbf{p}}_{k+1} = -\tilde{\mathbf{r}}_{k+1} + \beta_{k+1}\tilde{\mathbf{p}}_k \end{array}}$$

Numerical Optimization (2019 Fall)

BioComputing

Multivariate Optimization:
Conjugate Gradient Method

- Bi-conjugate Gradient Methods(BICG)
  - Choice of α and β : to ensure the orthogonalities:

$$\mathbf{r}_i^T\tilde{\mathbf{r}}_j = \mathbf{p}_i\mathbf{A}\tilde{\mathbf{p}}_j = 0 \text{ if } i \neq j$$

  - So, we get

$$\alpha_k = \frac{\mathbf{r}_k^T\tilde{\mathbf{r}}_k}{\tilde{\mathbf{p}}_k^T\mathbf{A}\mathbf{p}_k}, \quad \beta_{k+1} = \frac{\mathbf{r}_{k+1}^T\tilde{\mathbf{r}}_{k+1}}{\mathbf{r}_k^T\tilde{\mathbf{r}}_k}$$

Numerical Optimization (2019 Fall)

BioComputing

Multivariate Optimization:
Conjugate Gradients Method

When f(**x**) = $\frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{b}^T\mathbf{x}$ and **A** is non-symmetric

- Conjugate Gradients Squares(CGS)
  - Looking at CG, we realize that

$$\boxed{\begin{array}{l}\mathbf{r}_k := \mathbf{r}_{k-1} + \alpha_{k-1}\mathbf{A}\mathbf{p}_{k-1} \Rightarrow \mathbf{r}_k = P_k(\mathbf{A})\mathbf{r}_0 \\ \quad \text{for some polynomial } P_k(\mathbf{A}) \text{ when } \mathbf{p}_0 = -\mathbf{r}_0\end{array}}$$

  - As iteration goes, $\mathbf{r}_k$ will approach 0. It means $P_k(\mathbf{A})$ is a kind of contraction operator.
  - Evidently, when $\mathbf{r}_k = P_k(\mathbf{A})^2\,\mathbf{r}_0$, it is expected to give faster convergence than original CG.

Multivariate Optimization:
Conjugate Gradients Method

When f(**x**) = $\frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{b}^T\mathbf{x}$ and **A** is non-symmetric

- Bi-conjugate Gradients Stabilized(BICGSTAB)
  - In CGS, we realize that

$$\boxed{\mathbf{r}_k = P_k(\mathbf{A})^2\mathbf{r}_0 \Rightarrow \mathbf{r}_k = Q_k(\mathbf{A})P_k(\mathbf{A})\,\mathbf{r}_0}$$

  - Instead of $P_k(\mathbf{A})P_k(\mathbf{A})$, use of $Q_k(\mathbf{A})P_k(\mathbf{A})$ for some polynomial $Q_k(\mathbf{A})$ is possible.
  - When $Q_k(\mathbf{A}) = (1 + \alpha_1\mathbf{A})(1 + \alpha_2\mathbf{A})\ldots(1 + \alpha_k\mathbf{A})$, it is good to have more stabilized convergence than CGS.

Multivariate Optimization:
Conjugate Gradients Method

- **Ax** = **b** Solvers

	Direct A = LU	Iterative y' = Ay
Non-symmetric	**Pivoting LU**	**GMRES, BiCGSTAB, …**
Symmetric positive definite	**Cholesky**	**Conjugate gradient**

More General

↕

More Robust

More Robust ⟷ Less Storage (if sparse)

Ref.: https://nanohub.org/

Multivariate Optimization:
Conjugate Gradient Method

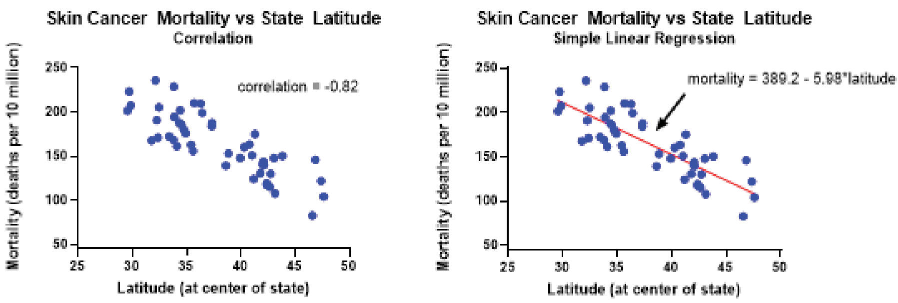
- Homework #5 (**Implementation**)

  Due date : November 6 (Wednesday), 2019 10:30AM

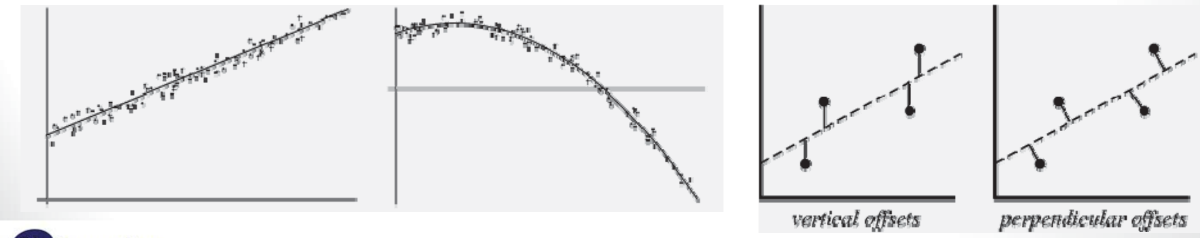
  - Implement linear/nonlinear Conjugate Gradient methods for the following functions:
    - $f(x, y) = (x + 2y - 7)^2 + (2x + y - 5)^2$
    - $f(x, y) = 40(y - x^2)^2 + (1-x)^2$
    - $f(x, y) = (1.5 - x + xy)^2 + (2.25 - x + xy^2)^2 + (2.625 - x + xy^3)^2$
  - Discuss their performances between linear and nonlinear CGs.

Multivariate Optimization:
Least Square Methods

- How to determine a fitted model for the given measurement data



https://www.graphpad.com/



http://mathworld.wolfram.com/LeastSquaresFitting.html

Multivariate Optimization:
Least Square Methods

- Least Square Method is a way to minimize the functional.

functional : a real-valued function

$$f(\mathbf{x}) = \frac{1}{2}\sum_{j=1}^{m}\left[\boxed{\phi(\mathbf{x};t_j)} - y_j\right]^2 = \frac{1}{2}\sum_{j=1}^{m} r_j(\mathbf{x})^2 \qquad r_j(\mathbf{x}) := \varphi(\mathbf{x};\, t_j) - y_j$$

modeling function for the given data y_j

For residue vector component $r_j(\mathbf{x}) : R^n \rightarrow R$

- $\nabla f(\mathbf{x}) = \Sigma^{m}_{j=1}\, r_j(\mathbf{x})\, \nabla r_j(\mathbf{x})$ gradient of $f(\mathbf{x})$
- $\nabla^2 f(\mathbf{x}) = \Sigma^{m}_{j=1}\, \nabla r_j(\mathbf{x})\, \nabla r_j(\mathbf{x})^T + \Sigma^{m}_{j=1}\, r_j(\mathbf{x})\, \nabla^2 r_j(\mathbf{x})$ Hessian of $f(\mathbf{x})$

Jacobian

- Multivariate function $r(\mathbf{x}) = (r_1(\mathbf{x}), r_2(\mathbf{x}), \ldots, r_m(\mathbf{x}))^T : R^n \to R^m$, $\mathbf{x} = (x_1, x_2, \ldots, x_n)$
- Jacobian matrix $\mathbf{J}$ ($m \times n$) of $r(\mathbf{x})$ is defined by

$$J(\mathbf{x}) = \frac{d\mathbf{r}}{d\mathbf{x}} = \frac{\partial(r_1, r_2, \cdots, r_m)}{\partial(x_1, x_2, \cdots, x_n)} := \begin{pmatrix} \frac{\partial r_1(\mathbf{x})}{\partial x_1} & \frac{\partial r_2(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial r_m(\mathbf{x})}{\partial x_1} \\ \frac{\partial r_1(\mathbf{x})}{\partial x_2} & \frac{\partial r_2(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial r_m(\mathbf{x})}{\partial x_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial r_1(\mathbf{x})}{\partial x_n} & \frac{\partial r_2(\mathbf{x})}{\partial x_n} & \cdots & \frac{\partial r_m(\mathbf{x})}{\partial x_n} \end{pmatrix}^T$$

Multivariate Optimization:
Least Square Methods

- Reformulation

$$f(\mathbf{x}) = \frac{1}{2}\sum_{j=1}^{m}\left[\phi(\mathbf{x}; t_j) - y_j\right]^2$$

$r_j(\mathbf{x}) := \varphi(\mathbf{x}; t_j) - y_j$

$$f(\mathbf{x}) = \frac{1}{2}\sum_{j=1}^{m} r_j(\mathbf{x})^2 = \frac{1}{2}\|r(\mathbf{x})\|_2^2 \quad \text{'Euclidean Norm'}$$

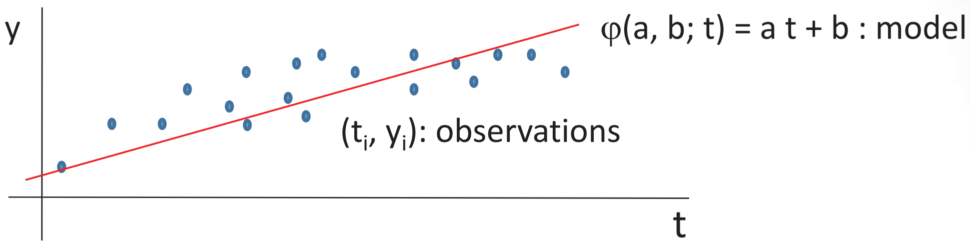
Residue vector $r(\mathbf{x}) = (r_1(\mathbf{x}), r_2(\mathbf{x}), \ldots, r_m(\mathbf{x}))^T : R^n \to R^m$

Let $J(\mathbf{x})$ be Jacobian of $r(\mathbf{x})$. Then

- $\nabla f(\mathbf{x}) = \Sigma_{j=1}^{m} r_j(\mathbf{x}) \nabla r_j(\mathbf{x}) = J(\mathbf{x})^T r(\mathbf{x})$
- $\nabla^2 f(\mathbf{x}) = \Sigma_{j=1}^{m} \nabla r_j(\mathbf{x}) \nabla r_j(\mathbf{x})^T + \Sigma_{j=1}^{m} r_j(\mathbf{x}) \nabla^2 r_j(\mathbf{x})$

  $= J(\mathbf{x})^T J(\mathbf{x}) + \Sigma_{j=1}^{m} r_j(\mathbf{x}) \nabla^2 r_j(\mathbf{x})$

Multivariate Optimization:
Least Square Methods



Step 1. Define cost function $f(a, b) = \Sigma[y_i - \varphi(a, b; t_i)]^2 = \Sigma[y_i - (a\,t_i + b)]^2$

This is how to measure discrepancy between model and observation.

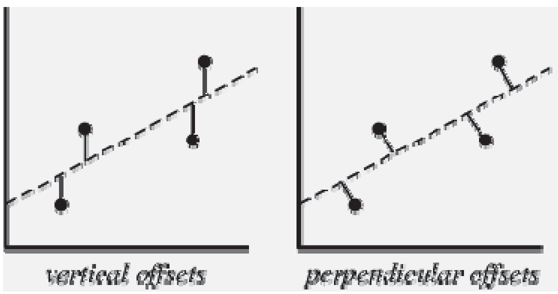
Step 2. Differentiate f(a,b) over undetermined parameters a and b.

Find a & b such that $\partial f/\partial a = \partial f/\partial b = 0$.

Then
$$\left.\begin{aligned} \partial f/\partial a &= -2\Sigma\, t_i[y_i - (a\,t_i + b)] = 0 \\ \partial f/\partial b &= -2\Sigma\, [y_i - (a\,t_i + b)] = 0 \end{aligned}\right\} \begin{pmatrix} \sum_i t_i^2 & \sum_i t_i \\ \sum_i t_i & \sum_i 1 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} \sum_i t_i y_i \\ \sum_i y_i \end{pmatrix}$$

Multivariate Optimization:
Least Square Methods

- Different ways to measure discrepancy between model and observation
  - $\max_{j=1,2,\ldots,m} |\, \varphi(\mathbf{x}; t_j) - y_j \,| \;\rightarrow\; |r(\mathbf{x})|_\infty$
  - $\Sigma^m_{j=1} |\, \varphi(\mathbf{x}; t_j) - y_j \,| \;\rightarrow\; |r(\mathbf{x})|_1$



Multivariate Optimization: Least Square Methods

- Linear least square problems ('Simplest LS')

$$\min_{\mathbf{x}} f(\mathbf{x}), \quad f(\mathbf{x}) := \frac{1}{2}\left\|\mathbf{A}\mathbf{x} - \mathbf{y}\right\|_2^2 = \frac{1}{2}(\mathbf{x}^T\mathbf{A}^T - \mathbf{y}^T)(\mathbf{A}\mathbf{x} - \mathbf{y})$$

  - $\nabla f(\mathbf{x}) = \mathbf{A}^T(\mathbf{A}\,\mathbf{x} - \mathbf{y})$
  - $\nabla^2 f(\mathbf{x}) = \mathbf{A}^T\mathbf{A}$
  - When **A** has full column rank (that is, $\mathbf{A}^T\mathbf{A}$ is invertible), then $\mathbf{A}^T\mathbf{A}$ is positive definite and f(**x**) is a convex quadratic functional. So, seeking **x*** such that $\nabla f(\mathbf{x}) = 0$ yields the global minimizer.

$$\nabla f(\mathbf{x}^*) = 0 \;\leftrightarrow\; \mathbf{A}^T\mathbf{A}\,\mathbf{x}^* = \mathbf{A}^T\mathbf{y} \text{ 'normal equation'}$$

Multivariate Optimization: Least Square Methods

- How to solve normal equation
  - Cholesky factorization of $\mathbf{A}^T\mathbf{A}$
  - QR factorization
  - SVD factorization
  - Iterative methods
    - Conjugate Gradients
    - Generalized Minimal Residue, Bi-conjugate Gradients
    - Conjugate Gradient Squared
    - Bi-conjugate Gradient Stabilized

Multivariate Optimization: Least Square Methods

- Cholesky factorization for **A x** = **b**
  - **A** is decomposed into an upper triangular L and its transpose. (Assume **A** is positive definite symmetric matrix)
    - $\mathbf{A} = \mathbf{L}\,\mathbf{L}^T$
  - Then $\mathbf{L}\,\mathbf{L}^T\,\mathbf{x} = \mathbf{b}$.
    - Solve $\mathbf{L}\,\mathbf{y} = \mathbf{b}$ by back-substitution
    - Then solve $\mathbf{L}^T\,\mathbf{x} = \mathbf{y}$ by back-substitution
    - Back-substitution is simple and efficient.

Multivariate Optimization: Least Square Methods

- QR factorization for **A x** = **b**
  - **A** is decomposed into an orthogonal **Q** and an upper triangular **R**.
    - $\mathbf{A} = \mathbf{Q}\,\mathbf{R}$ (orthogonal matrix $\mathbf{Q}$: $\mathbf{Q}\,\mathbf{Q}^T = \mathbf{I}$)
  - Then $\mathbf{Q}\,\mathbf{R}\,\mathbf{x} = \mathbf{b} \leftrightarrow \mathbf{R}\,\mathbf{x} = \mathbf{Q}^T\,\mathbf{b}$.
    - Solve $\mathbf{R}\,\mathbf{x} = \mathbf{Q}^T\,\mathbf{b}$ by back-substitution

Multivariate Optimization:
Least Square Methods

- SVD factorization for **A x = b**
  - **A** is decomposed into an orthogonal, a diagonal and an orthogonal.
    - $\mathbf{A} = \mathbf{U}\,\mathbf{S}\,\mathbf{V}^T$ (**U**, **V** are orthogonal and **S** is a diagonal)
  - Then $\mathbf{U}\,\mathbf{S}\,\mathbf{V}^T\mathbf{x} = \mathbf{b}$

    $\leftrightarrow \mathbf{S}\,\mathbf{V}^T\mathbf{x} = \mathbf{U}^T\mathbf{b}$

    $\leftrightarrow \mathbf{V}^T\mathbf{x} = \mathbf{S}^{-1}\mathbf{U}^T\mathbf{b} \leftrightarrow \mathbf{x} = \mathbf{V}\,\mathbf{S}^{-1}\mathbf{U}^T\mathbf{b}$
  - When **A** is m x n (m > n), it yields

$$\mathbf{A} = \begin{bmatrix}\mathbf{U}_1 & \mathbf{U}_2\end{bmatrix}\begin{bmatrix}\mathbf{S}_1 \\ \mathbf{0}\end{bmatrix}\mathbf{V}^T = \mathbf{U}_1\,\mathbf{S}_1\,\mathbf{V}^T$$

Multivariate Optimization:
Least Square Methods

- Nonlinear least square problems

Nonlinear model

$$\min_{\mathbf{x}}\ f(\mathbf{x}),\quad f(\mathbf{x}) := \frac{1}{2}\sum_{j=1}^{m}\left[\phi(\mathbf{x};\ t_j) - y_j\right]^2 = \frac{1}{2}\sum_{j=1}^{m} r_j(\mathbf{x})^2$$

- Gauss-Newton method (modified Newton's)

  (Recall : Newton's method : Solve $H(\mathbf{x}_k)\,\mathbf{p} = -\nabla f(\mathbf{x}_k)$)

  - Let $J(\mathbf{x})$ be a Jacobian of $r(\mathbf{x}) = (r_1(\mathbf{x}), r_2(\mathbf{x}), \ldots, r_m(\mathbf{x}))^T$.

$$H(\mathbf{x}_k) = \nabla^2 f(\mathbf{x}_k) = J(\mathbf{x}_k)^T J(\mathbf{x}_k) + \sum_{j=1}^{m} r_j(\mathbf{x}_k)\nabla^2 r_j(\mathbf{x}_k)$$

$$\nabla f(\mathbf{x}_k) = J(\mathbf{x}_k)^T r(\mathbf{x}_k)$$

  - Use the approximation of Hessian

    $\nabla^2 f(\mathbf{x}_k) \approx J(\mathbf{x}_k)^T J(\mathbf{x}_k)$ by dropping off 2nd term having $\nabla^2 r_j$