Lecture 6: Variable Metric (Secant, Quasi-Newton) Methods, Quadratic Termination, Practicalities, Incremental Methods for NN

Prof. Krishna R. Pattipati
Dept. of Electrical and Computer Engineering
University of Connecticut
Contact: krishna@engr.uconn.edu (860) 486-2890

ECE 6437
Computational Methods for Optimization

Fall 2009
October 6, 2009

Outline of Lecture 6

- Quasi-Newton (QN) Methods
- Motivation from Quadratic Functions
- Square-root Implementation
  - Davidon-Fletcher-Powell (DFP) & Broyden-Fletcher-Goldfarb-Shanno (BFGS) versions
  - Relation to Kalman Filtering with Perfect Observations
- Properties of QN Methods
  - Quadratic Termination
  - Convergence Properties
  - Scaling
- Incremental Methods for Neural Networks

Quasi-Newton Methods - 1

- We want to look at a class of algorithms of the form:

$$\underline{x}_{k+1} = \underline{x}_k - \alpha_k H_k \underline{g}_k, \quad \underline{g}_k = \nabla \underline{f}\left(\underline{x}_k\right)$$

$$H_k \sim \left[\nabla^2 \underline{f}\left(\underline{x}_k\right)\right]^{-1} \quad \Rightarrow \quad \text{Newton is fast, but requires Hessian}$$

$$H_k \sim I \quad \Rightarrow \quad \text{SD}$$

$$\underline{d}_k = -\underline{g}_k + \frac{\left(\underline{g}_k - \underline{g}_{k-1}\right)^{\mathrm{T}} \underline{g}_k}{\left(\underline{g}_k - \underline{g}_{k-1}\right)^{\mathrm{T}} \underline{d}_{k-1}} \underline{d}_{k-1} = -\left[I - \frac{\underline{d}_{k-1}\underline{q}_{k-1}^{\mathrm{T}}}{\underline{q}_{k-1}^{\mathrm{T}}\underline{d}_{k-1}}\right] \underline{g}_k = -H_k \underline{g}_k$$

$$\underline{q}_{k-1} = \underline{g}_k - \underline{g}_{k-1}$$

H_k not symmetric, but can be made symmetric. See Shanno in *M. of OR* (1978)

- Key Question: Can we build up $H_k \ni H_0 \to H_1 \to \ldots \to H_k \to \left[\nabla^2 f\left(\underline{x}_k\right)\right]^{-1}$ without explicitly computing the Hessian?

Essentially an extension of secant approximation to second derivatives:

$$\frac{d^2 f}{dx^2} \cong \frac{\left(df/dx\right)|_{x_1} - \left(df/dx\right)|_{x_2}}{x_1 - x_2}$$ to higher dimensions.

Quasi-Newton Methods - 2

- Sometimes we may want an approximation to Hessian

$$B_0 \to B_1 \to \ldots \to B_k \to \nabla^2 f\left(\underline{x}_k\right)$$

- Let us recall the requirements on H_k and B_k:

1. H_k is PD (or) $H_k > 0$ $\qquad\qquad$ $B_k > 0$
2. H_k symmetric $\Rightarrow$ $H_k = H_k^{\mathrm{T}}$ $\qquad \Longleftrightarrow \qquad$ $B_k = B_k^{\mathrm{T}}$
3. Finite termination on quadratics $\qquad$ quadratic termination
4. $H_k \sim \left[\nabla^2 f\left(\underline{x}_k\right)\right]^{-1}$ $\qquad\qquad$ $B_k \sim \nabla^2 f\left(\underline{x}_k\right)$

- Process: $\underline{x}_k \to \underline{g}_k \overset{\Delta}{\to} H_k \to \underline{d}_k \to \alpha_k \to \underline{x}_{k+1} \to \underline{g}_{k+1} \overset{\Delta}{\to} H_{k+1}$
- Motivation via quadratic functions: $f\left(\underline{x}\right) = \frac{1}{2}\left(\underline{x} - \underline{x}^*\right)^{\mathrm{T}} Q \left(\underline{x} - \underline{x}^*\right)$

$$\underline{g}_k = Q\left(\underline{x} - \underline{x}^*\right), \quad \underline{g}_{k+1} = Q\left(\underline{x}_k - \underline{x}^* + \underline{x}_{k+1} - \underline{x}_k\right) = \underline{g}_k + Q\underline{p}_k$$

where $\underline{p}_k = \underline{x}_{k+1} - \underline{x}_k = \alpha_k \underline{d}_k$

Motivation via Quadratic Functions

- Key Secant Relations:

Look for relations of the form:

$$Q\underline{p}_k = \underline{q}_k = \underline{g}_{k+1} - \underline{g}_k \quad \Rightarrow \quad B_{k+1}\underline{p}_k = \underline{q}_k$$

(or) $$Q^{-1}\underline{q}_k = \underline{p}_k \quad \Rightarrow \quad H_{k+1}\underline{q}_k = \underline{p}_k$$

$$Q\left[\underline{p}_0 \quad \underline{p}_1 \quad \cdots \quad \underline{p}_{n-1}\right] = \left[\underline{q}_0 \quad \underline{q}_1 \quad \cdots \quad \underline{q}_{n-1}\right]$$

$$\Rightarrow \quad Q \sim B = \left[\underline{q}_0 \quad \underline{q}_1 \quad \cdots \quad \underline{q}_{n-1}\right]\left[\underline{p}_0 \quad \underline{p}_1 \quad \cdots \quad \underline{p}_{n-1}\right]^{-1}$$

$$\Rightarrow \quad Q^{-1} \sim H = \left[\underline{p}_0 \quad \underline{p}_1 \quad \cdots \quad \underline{p}_{n-1}\right]\left[\underline{q}_0 \quad \underline{q}_1 \quad \cdots \quad \underline{q}_{n-1}\right]^{-1}$$

- We can approximate Q or Q^{-1} via $\{\underline{p}_i\}$ and $\{\underline{q}_i\}$, but it may not be symmetric.
- Can we do this recursively? $H_k \to H_{k+1}$ (or) $B_k \to B_{k+1}$ starting with H_0 or B_0. Typically $H_0 = B_0 = I$.

Key Ideas of Quasi-Newton Methods - 1

- Problem:

Inverse Hessian Approximation		Hessian Approximation
Given: $\underline{p}_k = \underline{x}_{k+1} - \underline{x}_k$		Given: $\underline{p}_k = \underline{x}_{k+1} - \underline{x}_k$
$\underline{q}_k = \underline{g}_{k+1} - \underline{g}_k$		$\underline{q}_k = \underline{g}_{k+1} - \underline{g}_k$
$H_k > 0 \;\&\; H_k = H_k^{\mathrm{T}}$		$B_k > 0 \;\&\; B_k = B_k^{\mathrm{T}}$
Find: $H_{k+1} = H_{k+1}^{\mathrm{T}} > 0$		Find: $B_{k+1} = B_{k+1}^{\mathrm{T}} > 0$
$\ni \; H_{k+1}\underline{q}_k = \underline{p}_k$	(DUALS) ⟷	$\ni \; B_{k+1}\underline{p}_k = \underline{q}_k$

Secant (or) quasi-Newton conditions

- Alternate Problem:

Given: $\underline{p}_k,\ \underline{q}_k,\ L_k$		Given: $\underline{p}_k,\ \underline{q}_k,\ \hat{L}_k$
Find: $L_{k+1} \ni H_{k+1} = L_{k+1}L_{k+1}^{\mathrm{T}}$	⟷	Find: $\hat{L}_{k+1} \ni B_{k+1} = \hat{L}_{k+1}\hat{L}_{k+1}^{\mathrm{T}}$
$\& \; L_{k+1}L_{k+1}^{\mathrm{T}}\underline{q}_k = \underline{p}_k$		$\& \; \hat{L}_{k+1}\hat{L}_{k+1}^{\mathrm{T}}\underline{p}_k = \underline{q}_k$

Key Ideas of Quasi-Newton Methods - 2

- What are the conditions that guarantee positive definiteness of H_{k+1}?

  Need $\boxed{\underline{p}_k^{\mathrm{T}} \underline{q}_k > 0}$

- Scalar Case: $h_{k+1} = L_{k+1}^2 = \dfrac{p_k}{q_k}$ need $p_k,\ q_k \neq 0$ & $p_k,\ q_k$ same sign.

$$H_{k+1} = H_{k+1}^{\mathrm{T}} > 0 \quad \Leftrightarrow \quad p_k^{\mathrm{T}} \underline{q}_k > 0$$

- Suppose we have L_{k+1} ∋ $L_{k+1} L_{k+1}^{\mathrm{T}} \underline{q}_k = \underline{p}_k$

Let $\underline{v}_k = L_{k+1}^{\mathrm{T}} \underline{q}_k \quad \Rightarrow \quad 0 < \underline{v}_k^{\mathrm{T}} \underline{v}_k = \underline{q}_k^{\mathrm{T}} L_{k+1} L_{k+1}^{\mathrm{T}} \underline{q}_k = \underline{q}_k^{\mathrm{T}} \underline{p}_k$

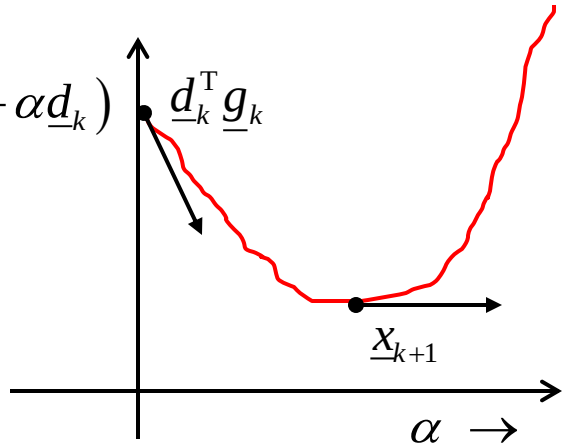
- What does it mean?

$\underline{p}_k^{\mathrm{T}} \underline{q}_k > 0 \quad \Rightarrow \quad \underline{p}_k^{\mathrm{T}} \underline{g}_{k+1} > \underline{p}_k^{\mathrm{T}} \underline{g}_k$

since $\underline{p}_k = \alpha_k \underline{d}_k$ & $\alpha_k > 0 \quad \Rightarrow \quad \underline{d}_k^{\mathrm{T}} \underline{g}_{k+1} > \underline{d}_k^{\mathrm{T}} \underline{g}_k$

$\underline{d}_k^{\mathrm{T}} \underline{g}_{k+1} = 0$ for **exact** line searches

$\underline{d}_k^{\mathrm{T}} \underline{g}_k < 0$ descent direction



Recursive Update Problem - 1

- Suppose $\underline{p}_k^{\mathrm{T}} \underline{q}_k > 0$. How do we construct $H_{k+1}\left(L_{k+1}\right)$ from $H_k\left(L_k\right)$, $\underline{p}_k$, and $\underline{q}_k$?

  We will construct L_{k+1} in two steps:

  1) Construct S_{k+1} ∋ $S_{k+1}S_{k+1}^{\mathrm{T}} = H_{k+1}$. S_{k+1} not lower triangular.

  2) Use QR factorization to construct L_{k+1} ∋ $S_{k+1} = L_{k+1}T_{k+1}^{\mathrm{T}}$

  $L_{k+1} \sim$ Lower triangular

  $T_{k+1} \sim$ orthogonal matrix $\Rightarrow$ $T_{k+1}^{-1} = T_{k+1}^{\mathrm{T}}$

  Note: We do not need to store T_{k+1} since $S_{k+1}S_{k+1}^{\mathrm{T}} = L_{k+1}L_{k+1}^{\mathrm{T}} = H_{k+1}$.

- Quasi-Newton (or secant) condition $H_{k+1}\underline{q}_k = \underline{p}_k$ (or) $S_{k+1}S_{k+1}^{\mathrm{T}}\underline{q}_k = \underline{p}_k$ implies:

$$S_{k+1}^{\mathrm{T}}\underline{q}_k = \underline{u}_k, \quad S_{k+1}\underline{u}_k = \underline{p}_k$$

- So, the problem of finding H_{k+1} ∋ $H_{k+1}\underline{q}_k = \underline{p}_k$ & $\left\|H_{k+1} - H_k\right\|_F$ small is equivalent to:

  1a) Find S_{k+1} ∋ $S_{k+1}\underline{u}_k = \underline{p}_k$ & $\left\|S_{k+1} - L_k\right\|_F$ is small.

  "Least change characterization"

  1b) Once S_{k+1} is known, find $\underline{u}_k$ from $S_{k+1}^{\mathrm{T}}\underline{q}_k = \underline{u}_k$.

$$\| A \|_F = \sqrt{tr(A^T A)} = \sqrt{\sum_{i=1}^{n} \sigma_i^2}$$

$\{\sigma_i\} \sim$ singular values

Recursive Update Problem - 2

Solution of Problem 1a:

$$\min_{S_{k+1}} \frac{1}{2}\left\|S_{k+1}-L_k\right\|_F^2 = \min_{S_{k+1}} \frac{1}{2}\mathrm{tr}\left[\left(S_{k+1}-L_k\right)^{\mathrm{T}}\left(S_{k+1}-L_k\right)\right] \text{ subject to } S_{k+1}\underline{u}_k = \underline{p}_k$$

$$L\left(S_{k+1},\underline{\lambda}\right) = \frac{1}{2}\mathrm{tr}\left[\left(S_{k+1}-L_k\right)^{\mathrm{T}}\left(S_{k+1}-L_k\right)\right] + \underline{\lambda}^{\mathrm{T}}\left[S_{k+1}\underline{u}_k - \underline{p}_k\right]$$

$$\frac{\partial L}{\partial S_{k+1}} = S_{k+1}-L_k+\underline{\lambda}\underline{u}_k^{\mathrm{T}} = 0 \;\Rightarrow\; S_{k+1} = L_k - \underline{\lambda}\underline{u}_k^{\mathrm{T}}$$

$$\frac{\partial L}{\partial \lambda} = S_{k+1}\underline{u}_k - \underline{p}_k = 0 \;\Rightarrow\; L_k\underline{u}_k - \underline{p}_k - \underline{\lambda}\left(\underline{u}_k^{\mathrm{T}}\underline{u}_k\right) = 0 \text{ or } \underline{\lambda} = \frac{\left(L_k\underline{u}_k - \underline{p}_k\right)}{\underline{u}_k^{\mathrm{T}}\underline{u}_k}$$

$$S_{k+1} = L_k + \frac{\left(\underline{p}_k - L_k\underline{u}_k\right)\underline{u}_k^{\mathrm{T}}}{\underline{u}_k^{\mathrm{T}}\underline{u}_k}$$

$$\underline{\lambda}^T S_{k+1}\underline{u}_k = Tr(S_{k+1}\underline{u}_k\underline{\lambda}^T)$$
$$\nabla_A Tr(AB) = B^T$$
$$\Rightarrow \nabla_{S_{k+1}} Tr(S_{k+1}\underline{u}_k\underline{\lambda}^T) = \underline{\lambda}\underline{u}_k^T$$
$$\nabla_{\underline{\lambda}}(\underline{\lambda}^T S_{k+1}\underline{u}_k) = S_{k+1}\underline{u}_k$$

- S_{k+1} is the nearest matrix to L_k while satisfying $S_{k+1}\underline{u}_k=\underline{p}_k$.

Solution of Problem 1b:

$$\text{Need } \underline{u}_k = S_{k+1}^{\mathrm{T}}\underline{q}_k = L_k^{\mathrm{T}}\underline{q}_k + \underline{u}_k\frac{\left(\underline{p}_k - L_k\underline{u}_k\right)^{\mathrm{T}}\underline{q}_k}{\underline{u}_k^{\mathrm{T}}\underline{u}_k}$$

$$\left[1-\frac{\left(\underline{p}_k - L_k\underline{u}_k\right)^{T}\underline{q}_k}{\underline{u}_k^{\mathrm{T}}\underline{u}_k}\right]\underline{u}_k = L_k^{\mathrm{T}}\underline{q}_k \;\Rightarrow\; \underline{u}_k \parallel L_k^{\mathrm{T}}\underline{q}_k \qquad \therefore\; \underline{u}_k = \Theta_k L_k^{\mathrm{T}}\underline{q}_k$$

Recursive Update Problem - 3

$$\left[1-\frac{\underline{p}_k^{\mathrm{T}}\underline{q}_k}{\Theta_k^2\underline{q}_k^{\mathrm{T}}L_kL_k^T\underline{q}_k}+\frac{\underline{q}_k^TL_k^T\underline{u}_k}{\underline{u}_k^T\underline{u}_k}\right]\underline{u}_k=\left[1-\frac{\underline{p}_k^{\mathrm{T}}\underline{q}_k}{\Theta_k^2\underline{q}_k^{\mathrm{T}}H_k\underline{q}_k}+\frac{1}{\Theta_k}\right]\underline{u}_k=L_k^{\mathrm{T}}\underline{q}_k$$

$$\Rightarrow\ \Theta_k-\frac{\underline{p}_k^{\mathrm{T}}\underline{q}_k}{\Theta_k\underline{q}_k^{\mathrm{T}}H_k\underline{q}_k}+1=1 \qquad \Rightarrow \qquad \boxed{\Theta_k^2=\frac{\underline{p}_k^{\mathrm{T}}\underline{q}_k}{\underline{q}_k^{\mathrm{T}}H_k\underline{q}_k}}$$

$$\Rightarrow \qquad \boxed{\underline{u}_k=\pm\sqrt{\frac{\underline{p}_k^{\mathrm{T}}\underline{q}_k}{\underline{q}_k^{\mathrm{T}}H_k\underline{q}_k}}\cdot L_k^{\mathrm{T}}\underline{q}_k}$$

since $\underline{p}_k^{\mathrm{T}}\underline{q}_k>0$ sqrt always exists.

- Take the positive sign since this makes S_{k+1} closer to L_k when $\underline{p}_k\cong H_k\underline{q}_k\Rightarrow\Theta_k=1\ \&\ S_{k+1}\approx L_k$ as it should.
- $\underline{u}_k^{\mathrm{T}}\underline{u}_k=\underline{p}_k^{\mathrm{T}}\underline{q}_k>0$

❑ Before we consider the problem of finding L_{k+1} from S_{k+1}, let us take a step back and review the results obtained so far.

Recursive Update Problem - 4

- Given L_k, $\underline{p}_k$, $\underline{q}_k$ э L_k^{-1} exists, we found an S_{k+1} э $S_{k+1}S_{k+1}^{\mathrm{T}}\underline{q}_k = \underline{p}_k$.

  The result is:

$$S_{k+1} = L_k + \frac{\left(\underline{p}_k - L_k\underline{u}_k\right)\underline{u}_k^{\mathrm{T}}}{\underline{u}_k^{\mathrm{T}}\underline{u}_k}, \qquad \underline{u}_k = \Theta_k L_k^{\mathrm{T}}\underline{q}_k$$

$$\Theta_k = \sqrt{\frac{\underline{p}_k^{\mathrm{T}}\underline{q}_k}{\underline{q}_k^{\mathrm{T}}H_k\underline{q}_k}} > 0; \qquad \mathrm{H}_k = L_k L_k^{\mathrm{T}}$$

- Q: Is S_{k+1} invertible?

  Suppose it is:

$$S_{k+1}^{-1} = L_k^{-1} - \frac{L_k^{-1}\underline{u}_k\left(\underline{p}_k - L_k\underline{u}_k\right)^{\mathrm{T}}L_k^{-1}}{\underline{u}_k^{\mathrm{T}}\underline{u}_k + \underline{u}_k^{\mathrm{T}}L_k^{-1}\left(\underline{p}_k - L_k\underline{u}_k\right)}$$

  The inverse exists if $\underline{u}_k^{\mathrm{T}}\underline{u}_k + \underline{u}_k^{\mathrm{T}}L_k^{-1}\underline{p}_k - \underline{u}_k^{\mathrm{T}}\underline{u}_k = \Theta_k\underline{q}_k^{\mathrm{T}}\underline{p}_k \neq 0.$

  Since $\underline{q}_k^{\mathrm{T}}\underline{p}_k > 0 \,\&\, \Theta_k > 0$, S_{k+1}^{-1} exists.

- So, $H_{k+1} = S_{k+1}S_{k+1}^{\mathrm{T}}$ is PD. We can get a recursive expression for H_k as follows:

$$H_{k+1} = S_{k+1}S_{k+1}^{\mathrm{T}} = \left[L_k + \frac{\left(\underline{p}_k - L_k\underline{u}_k\right)\underline{u}_k^{\mathrm{T}}}{\underline{u}_k^{\mathrm{T}}\underline{u}_k}\right]\left[L_k^{\mathrm{T}} + \frac{\underline{u}_k\left(\underline{p}_k - L_k\underline{u}_k\right)^{\mathrm{T}}}{\underline{u}_k^{\mathrm{T}}\underline{u}_k}\right]$$

Recovering DFP Update

$$H_{k+1} = H_k + \frac{L_k \underline{u}_k \left(\underline{p}_k - L_k \underline{u}_k\right)^{\mathrm{T}}}{\underline{u}_k^{\mathrm{T}} \underline{u}_k} + \frac{\left(\underline{p}_k - L_k \underline{u}_k\right) \underline{u}_k^{\mathrm{T}} L_k^{\mathrm{T}}}{\underline{u}_k^{\mathrm{T}} \underline{u}_k} + \frac{\left(\underline{p}_k - L_k \underline{u}_k\right)\left(\underline{p}_k - L_k \underline{u}_k\right)^{\mathrm{T}}}{\underline{u}_k^{\mathrm{T}} \underline{u}_k}$$

$$= H_k + \frac{\underline{p}_k \underline{p}_k^{\mathrm{T}}}{\underline{p}_k^{\mathrm{T}} \underline{q}_k} - \frac{L_k \underline{u}_k \underline{u}_k^{\mathrm{T}} L_k^{\mathrm{T}}}{\underline{u}_k^{\mathrm{T}} \underline{u}_k}$$

$$\underline{u}_k^T \underline{u}_k = \underline{p}_k^T \underline{q}_k$$

$$\underline{u}_k = \Theta_k L_k^T \underline{q}_k$$

$$H_k = L_k L_k^T$$

$$H_{k+1} = H_k + \frac{\underline{p}_k \underline{p}_k^{\mathrm{T}}}{\underline{p}_k^{\mathrm{T}} \underline{q}_k} - \frac{H_k \underline{q}_k \underline{q}_k^{\mathrm{T}} H_k}{\underline{q}_k^{\mathrm{T}} H_k \underline{q}_k}$$

Davidon-Fletcher-Powelll (DFP) method (or) Inverse Positive Definite Secant Update

- All we need to do to ensure a descent direction ($H_{k+1} > 0$) is

  $\underline{p}_k^{\mathrm{T}} \underline{q}_k > 0 \;\Rightarrow\; \underline{g}_{k+1}^{\mathrm{T}} \underline{d}_k > \underline{g}_k^{\mathrm{T}} \underline{d}_k$

  Line search can be inexact.
- Because of round-off errors, H_{k+1} can still go indefinite. Can rewrite H_{k+1} as:

$$H_{k+1} = \left[I - \frac{H_k \underline{q}_k \underline{q}_k^{\mathrm{T}}}{\underline{q}_k^{\mathrm{T}} H_k \underline{q}_k}\right] H_k \left[I - \frac{H_k \underline{q}_k \underline{q}_k^{\mathrm{T}}}{\underline{q}_k^{\mathrm{T}} H_k \underline{q}_k}\right]^{\mathrm{T}} + \frac{\underline{p}_k \underline{p}_k^{\mathrm{T}}}{\underline{p}_k^{\mathrm{T}} \underline{q}_k} + \frac{H_k \underline{q}_k \underline{q}_k^{\mathrm{T}} H_k}{\underline{q}_k^{\mathrm{T}} H_k \underline{q}_k}$$

Similar to Joseph's form of covariance equation

Recovering BFGS Update

- We can obtain Hessian approximation, B_k via:

$$L_k L_k^{\mathrm{T}} = H_k \;\leftrightarrow\; B_k = \hat{L}_k \hat{L}_k^{\mathrm{T}} \qquad q_k \leftrightarrow \underline{p}_k \qquad \underline{p}_k \leftrightarrow \underline{q}_k \qquad L_k, S_k \leftrightarrow \hat{L}_k, \hat{S}_k$$

$$\hat{S}_{k+1} = \hat{L}_k + \frac{\left(\underline{q}_k - \hat{L}_k \hat{u}_k\right)\hat{u}_k^{\mathrm{T}}}{\hat{u}_k^{\mathrm{T}} \hat{u}_k}, \qquad \hat{u}_k = \sqrt{\frac{p_k^{\mathrm{T}} q_k}{\underline{p}_k^{\mathrm{T}} B_k \underline{p}_k}} \cdot \hat{L}_k^{\mathrm{T}} \underline{p}_k$$

$$B_{k+1} = B_k + \frac{\underline{q}_k \underline{q}_k^{\mathrm{T}}}{\underline{p}_k^{\mathrm{T}} \underline{q}_k} - \frac{B_k \underline{p}_k \underline{p}_k^{\mathrm{T}} B_k^{\mathrm{T}}}{\underline{p}_k^{\mathrm{T}} B_k \underline{p}_k} \quad (*)$$

Broyden-Fletcher-Goldfarb-Shanno (BFGS) formula (or) positive definite secant update

- Found to be the best update in practice. $\underline{d}_k$ from $\hat{L}_k \hat{L}_k^{\mathrm{T}} \underline{d}_k = -\underline{g}_k$
- We arbitrarily modify H_{k+1} update as:

$H_{k+1} = B_{k+1}^{-1}$ of (*) modified

$$= H_k + \frac{\underline{p}_k \underline{p}_k^{\mathrm{T}}}{\underline{p}_k^{\mathrm{T}} \underline{q}_k} - \frac{H_k \underline{q}_k \underline{q}_k^{\mathrm{T}} H_k}{\underline{q}_k^{\mathrm{T}} H_k \underline{q}_k} + \phi\left(\underline{q}_k^{\mathrm{T}} H_k \underline{q}_k\right) \cdot \underline{r}_k \underline{r}_k^{\mathrm{T}}, \qquad \underline{r}_k = \frac{\underline{p}_k}{\underline{p}_k^{\mathrm{T}} \underline{q}_k} - \frac{1}{\underline{q}_k^{\mathrm{T}} H_k \underline{q}_k} \cdot H_k \underline{q}_k$$

$\phi = 1$ actual B_{k+1}^{-1} BFGS inverse PD secant update

$\phi = 0$ DFP

Note that $\underline{r}_k^{\mathrm{T}} \underline{q}_k = 0 \;\Rightarrow\; H_{k+1} \underline{q}_k = \underline{p}_k$ is satisfied.

Analogy with Control & Estimation

- Similarly, we can arbitrarily modify B_{k+1} update as:

$$B_{k+1} = H_{k+1}^{-1} \text{ of DFP modified}$$

$$= B_k + \frac{\underline{q}_k \underline{q}_k^{\mathrm{T}}}{\underline{p}_k^{\mathrm{T}} \underline{q}_k} - \frac{B_k \underline{p}_k \underline{p}_k^{\mathrm{T}} B_k}{\underline{p}_k^{\mathrm{T}} B_k \underline{p}_k} + \phi\left(\underline{p}_k^{\mathrm{T}} B_k \underline{p}_k\right) \underline{w}_k \underline{w}_k^{\mathrm{T}}$$

$$\underline{w}_k = \frac{\underline{q}_k}{\underline{p}_k^{\mathrm{T}} \underline{q}_k} - \frac{B_k \underline{p}_k}{\underline{p}_k^{\mathrm{T}} B_k \underline{p}_k}, \qquad \underline{w}_k^{\mathrm{T}} \underline{p}_k = 0 \Rightarrow B_{k+1} \underline{p}_k = \underline{q}_k \qquad \phi = \begin{cases} 1 \; H_k^{-1} \text{ update of DFP} \\ 0 \text{ BFGS (best)} \end{cases}$$

- Expressions for B_{k+1} and H_{k+1} are similar to those that arise in filtering (with perfect observations) and linear quadratic control problems (with no control weightings).

Filtering

$$\underline{x}_{k+1} = \underline{x}_k + \underline{p}_k w_k; \quad \underline{w}_k = N\left(0, \frac{1}{\underline{p}_k^{\mathrm{T}} \underline{q}_k}\right)$$

$$\underline{y}_k = \underline{q}_k^{\mathrm{T}} \underline{x}_k \quad \text{(NO M/S NOISE)}$$

$$\hat{\underline{x}}_{k+1} = \hat{\underline{x}}_k + K_k\left[\underline{y}_k - \underline{q}_k^{\mathrm{T}} \hat{\underline{x}}_k\right]$$

$$K_k = \frac{\Sigma_k \underline{q}_k}{\underline{q}_k^{\mathrm{T}} \Sigma_k \underline{q}_k}, \quad \Sigma_{k+1} = \Sigma_k + \frac{\underline{p}_k \underline{p}_k^{\mathrm{T}}}{\underline{p}_k^{\mathrm{T}} \underline{q}_k} - \frac{\Sigma_k \underline{q}_k \underline{q}_k^{\mathrm{T}} \Sigma_k}{\underline{q}_k^{\mathrm{T}} \Sigma_k \underline{q}_k}$$

Control

$$\underline{x}_{k+1} = \underline{x}_k + \underline{q}_k u_k$$

$$y_k = \underline{p}_k^{\mathrm{T}} \underline{x}_k$$

$$\min\left\{\sum_{k=0}^{N} \frac{1}{\underline{p}_k^{\mathrm{T}} \underline{q}_k} y_k^2\right\}$$

$$P_{N-k} = P_{N-k+1} + \frac{\underline{p}_{N-k+1} \underline{p}_{N-k+1}^{\mathrm{T}}}{\underline{p}_{N-k+1}^{\mathrm{T}} \underline{q}_{N-k+1}} - \frac{P_{N-k+1} \underline{q}_{N-k+1} \underline{q}_{N-k+1}^{\mathrm{T}} P_{N-k+1}}{\underline{q}_{N-k+1}^{\mathrm{T}} P_{N-k+1} \underline{q}_{N-k+1}}$$

How to Traingularize Matrix *S*?

- Problem 2: How to get L_{k+1} from S_{k+1} ∋ $S_{k+1} = L_{k+1}T_{k+1}^{\mathrm{T}}$ & T_{k+1} orthogonal?
  - ∃ methods for transforming a matrix A into a product of a lower triangular matrix and an orthogonal matrix.
    - Householder, Givens, etc. (ECE 6435). Complexity O(n^3).
  - But, S_{k+1} is a rank one perturbation of a lower Δ matrix, L_k, i.e.,

$$S_{k+1} = L_k + \frac{\left(\underline{p}_k - L_k\underline{u}_k\right)\underline{u}_k^{\mathrm{T}}}{\underline{u}_k^{\mathrm{T}}\underline{u}_k}$$

  - For this problem, we can devise an O(n^2) algorithm based on GIVENS ROTATIONS

- What we have is: $L_k^{\mathrm{T}} = I.L_k^{\mathrm{T}} \Rightarrow T_k = I,\ L_k^{\mathrm{T}} =$ upper Δ.

  We want $S_{k+1}^{\mathrm{T}} = T_{k+1}L_{k+1}^{\mathrm{T}} \rightarrow$ QR decomposition of S_{k+1}^{T}

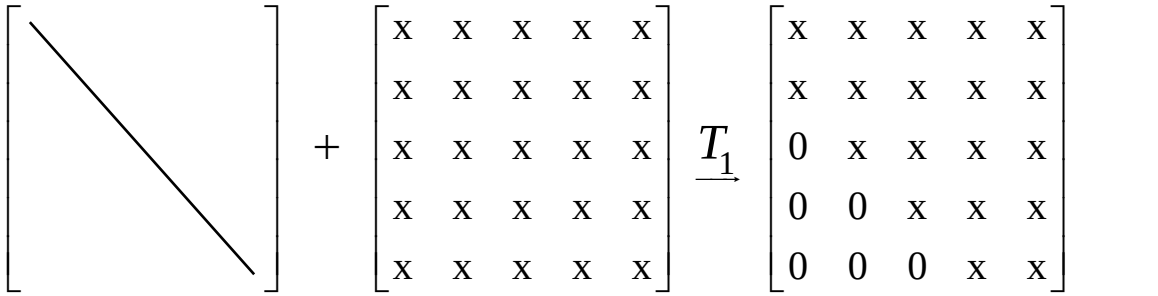
  - Don't actually need to store T_{k+1} since $H_{k+1} = S_{k+1}S_{k+1}^{\mathrm{T}} = L_{k+1}L_{k+1}^{\mathrm{T}}$.
  - The generic problem we want to solve is to find the QR decomposition of

$$S^{\mathrm{T}} = L^{\mathrm{T}} + \underline{u}\underline{y}^{\mathrm{T}} \qquad \underline{y} \sim \frac{\underline{p}_k - L_k\underline{u}_k}{\underline{u}_k^{\mathrm{T}}\underline{u}_k}$$

Triangularization via Givens

- The algorithm involves two steps:

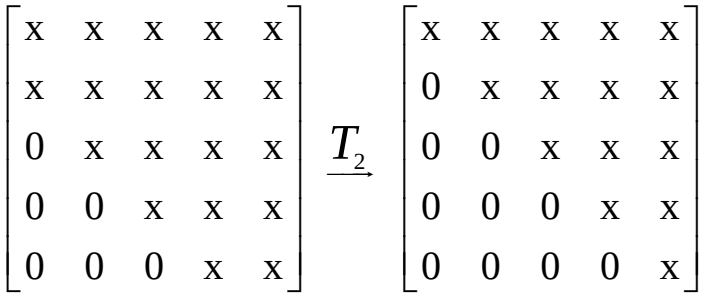
Step 1: Find an orthogonal transform T_1 such that $U_H = T_1^{\mathrm{T}}\left[L^{\mathrm{T}} + \underline{u}\underline{y}^{\mathrm{T}}\right]$.



L^{T} $\quad$ $\underline{u}\underline{y}^{\mathrm{T}}$ $\quad$ U_H upper Hessenberg matrix

Step 2: Find a second orthogonal transformation

T_2 э $\tilde{L}^{\mathrm{T}} = T_2^{\mathrm{T}} U_H = T_2^{\mathrm{T}} T_1^{\mathrm{T}}\left[L^{\mathrm{T}} + \underline{u}\underline{y}^{\mathrm{T}}\right]$.



$T = T_1 T_2$ & $\tilde{L}^{\mathrm{T}} = T^{\mathrm{T}}\left[L^{\mathrm{T}} + \underline{u}\underline{y}^{\mathrm{T}}\right]$

Upper Hessenberg via Givens -1

Step 1:

1.1 Find T_1^{T} ∋ $T_1^{\mathrm{T}}\underline{u} = \begin{bmatrix}\alpha & 0 & \cdots & 0\end{bmatrix}^{\mathrm{T}} = \alpha\underline{e}_1;$ $\qquad \alpha = \pm\sqrt{\underline{u}^{\mathrm{T}}\underline{u}}$

1.2 Apply T_1^{T} to $L^{\mathrm{T}} + \underline{u}\underline{y}^{\mathrm{T}} \Rightarrow$ $\underbrace{T_1^{\mathrm{T}}L^{\mathrm{T}}}_{\text{upper Hessenberg}} + \underbrace{\alpha\underline{e}_1\underline{y}^{\mathrm{T}}}_{\begin{bmatrix}\leftarrow & \alpha\underline{y}^{\mathrm{T}} & \rightarrow\\ \leftarrow & 0 & \rightarrow\\ \vdots & \vdots & \vdots\\ \leftarrow & 0 & \rightarrow\end{bmatrix}} = U_H.$

– The tool to accomplish step 1 is GIVENS rotations.

$$J(i,k) \triangleq J(i,k,\theta) = \begin{array}{c} \\ \\ i \\ \\ k \\ \\ \end{array}\!\!\begin{array}{c} \begin{array}{cccc} i & & k & \end{array} \\ \begin{bmatrix} 1 & & & \\ & \ddots & & \\ & & c & s \\ & & -s & c \\ & & & & \ddots \\ & & & & & 1 \end{bmatrix} \end{array}$$

$$\left.\begin{array}{l} c = \cos\theta \\ s = \sin\theta \end{array}\right\} \Rightarrow \sqrt{c^2 + s^2} = 1$$

$$J^{-1} = J^{\mathrm{T}} = J(i,k,-\theta)$$

$$y = J\underline{x} \Rightarrow y_i = cx_i + sx_k; \qquad y_k = -sx_i + cx_k; \qquad y_j = x_j \quad \forall\, j \neq i,k$$

Upper Hessenberg via Givens -2

- If you want to zero out y_k, select $c = \dfrac{x_i}{\sqrt{x_i^2 + x_k^2}}, \quad s = \dfrac{x_k}{\sqrt{x_i^2 + x_k^2}}$
  - So, to transform $\underline{u} \rightarrow \alpha \underline{e}_1$:
- $\overbrace{J(1,2) \ldots J(n-2,n)^{-1} J(n-1,n)}^{T_1^{\mathrm{T}}} \underline{u} = \alpha \underline{e}_1 = \begin{bmatrix} u_1 & 0 & \cdots & 0 \end{bmatrix}^{\mathrm{T}} = u_1 \underline{e}_1$
- Whatever you do to $\underline{u}$, do the same to L^{T}, i.e.,

  $J(1,2) \ldots J(n-2,n) J(n-1,n) L^{\mathrm{T}} = T_1^{\mathrm{T}} L^{\mathrm{T}} =$ upper Hessenberg
- Actually can do operations directly on L

$\Rightarrow$ lower Hessenberg $\Rightarrow L_{H_1} = L T_1 = L J^{\mathrm{T}}(n-1,n) \ldots J^{\mathrm{T}}(1,2)$

$$\begin{bmatrix} \mathrm{x} \\ \mathrm{x} \\ \mathrm{x} \\ \mathrm{x} \end{bmatrix} \xrightarrow{J(3,4)} \begin{bmatrix} \mathrm{x} \\ \mathrm{x} \\ \otimes \\ 0 \end{bmatrix} \xrightarrow{J(2,3)} \begin{bmatrix} \mathrm{x} \\ \otimes \\ 0 \\ 0 \end{bmatrix} \xrightarrow{J(1,2)} \begin{bmatrix} \otimes \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Step 1 Algorithm via Givens

- Algorithm:

FOR $i = n-1, n-2, \ldots, 1$ DO

$$c = \frac{u_i}{\sqrt{u_i^2 + u_{i+1}^2}}$$

$$s = \frac{u_{i+1}}{\sqrt{u_i^2 + u_{i+1}^2}}$$

$u_i \leftarrow c u_i + s u_{i+1}$

$u_{i+1} \leftarrow 0$

FOR $j = n, n\text{-}1, \ldots, i$ DO

$a = c l_{ji} + s l_{j,i+1}$

$b = -s l_{ji} + c l_{j,i+1}$

$l_{ji} \leftarrow a$

$l_{j,i+1} \leftarrow b$

END

END

only 2 columns of L $(j, i+1)$ are affected.

$L_{H_1} = LT_1$

FOR $i = 1, n$ DO

$l_{i1} \leftarrow l_{i1} + u_1 y_i$

END

$L_{H_1} + \alpha \underline{y}_1 \underline{e}^{\mathrm{T}} = L_H = U_H^{\mathrm{T}}$

Step 2 Algorithm via Givens

$$\begin{bmatrix} x & 0 & 0 & 0 \\ x & x & 0 & 0 \\ x & x & x & 0 \\ x & x & x & x \end{bmatrix} \xrightarrow{J^T(3,4)} \begin{bmatrix} x & 0 & 0 & 0 \\ x & x & 0 & 0 \\ x & x & \Delta & \Delta \\ x & x & \Delta & \Delta \end{bmatrix} \xrightarrow{J^T(2,3)} \begin{bmatrix} x & 0 & 0 & 0 \\ x & \square & \square & 0 \\ x & \square & \square & \Delta \\ x & \square & \square & \Delta \end{bmatrix} \xrightarrow{J^T(2,1)} \begin{bmatrix} \Diamond & \Diamond & 0 & 0 \\ \Diamond & \Diamond & \square & 0 \\ \Diamond & \Diamond & \square & \Delta \\ \Diamond & \Diamond & \square & \Delta \end{bmatrix}$$

- Step 2: Can we use Givens rotations again to transform Hessenberg to desired lower triangular form? YES

FOR $j = n, n-1, \ldots, 2$ DO

$$c = \frac{l_{jj}}{\sqrt{l_{j-1,j}^2 + l_{jj}^2}}$$

$$s = \frac{-l_{j-1,j}}{\sqrt{l_{jj}^2 + l_{j-1,j}^2}}$$

$$l_{jj} \leftarrow \sqrt{l_{jj}^2 + l_{j-1,j}^2}$$

$$l_{j-1,j} \leftarrow 0$$

FOR $i = 1, 2, \ldots, j-1$ DO

$$a \leftarrow c l_{j-1,i} + s l_{ji}$$

$$l_{ji} \leftarrow -s l_{j-1,i} + c l_{ji}$$

$$l_{j-1,i} \leftarrow a$$

} rows j-1 & j are affected

END

END

References

- References:

1. D. Goldfarb, "Factorized variable metric methods for unconstrained minimization," Mathematics of Computation, vol. 30, NO. 136, Oct. 1976, pp. 796-811
2. B. Fletcher and M. J. D. Powell, "On the modifications of LDL^T factorizations," Mathematics of Computation, vol. 28, NO. 128, Oct. 1976, pp. 1067-1087.
3. P. E. Gill, G. H. Golub, W. Murray, and M. A. Saunders, "Methods for modifying matrix factorizations," Mathematics of Computation, vol. 28, NO. 126, April 1974, pp. 505-535.

Quadratic Termination of QN Methods

1. Quadratic Termination: DFP method yields

- Q-conjugate $\underline{p}_i$ ($\Rightarrow$ Q-conjugate $\underline{d}_i$ since $\underline{p}_i = \alpha_i \underline{d}_i$)
- $H_{k+1} Q \underline{p}_i = \underline{p}_i, \quad 0 \le i \le k$
- $H_n = Q^{-1}$
  - In the quadratic case:

$$\underline{q}_k = \underline{g}_{k+1} - \underline{g}_k = Q\left(\underline{x}_{k+1} - \underline{x}_k\right) = Q\underline{p}_k$$

  - Also, quasi-Newton condition yields:

$H_{k+1}\underline{q}_k = \underline{p}_k \;\Rightarrow\; H_{k+1}Q\underline{p}_k \;\Rightarrow\; \underline{p}_k$ is an eigenvector of $H_{k+1}Q$ with a unity eigenvalue

- We prove Q-conjugacy of $\underline{p}_i$ via induction. Assume that $\underline{p}_0, \underline{p}_1, \ldots, \underline{p}_k$ are Q-conjugate. We prove that $\underline{p}_0, \underline{p}_1, \ldots, \underline{p}_{k+1}$ are Q-conjugate. We know that

$$\underline{g}_{k+1} = \underline{g}_{i+1} + Q\left[\underline{p}_{i+1} + \underline{p}_{i+2} + \ldots + \underline{p}_k\right]$$

$$\underline{p}_i^{\mathrm{T}} \underline{g}_{k+1} = \underline{p}_i^{\mathrm{T}} \underline{g}_{i+1} = 0 \;\forall\; 0 \le i \le k \;\because \text{ of } Q\text{-conjugacy}$$

$$\Rightarrow \underline{p}_i^{\mathrm{T}} \underline{g}_{k+1} = 0 \;\Rightarrow\; \underline{p}_i^{\mathrm{T}} Q \underbrace{H_{k+1}\underline{g}_{k+1}}_{\propto \underline{p}_{k+1}} = 0 \;\because H_{k+1} Q \underline{p}_i = \underline{p}_i$$

Since $\underline{p}_{k+1} = -\alpha_k H_{k+1}\underline{g}_{k+1} \;\Rightarrow\; \underline{p}_i^{\mathrm{T}} Q \underline{p}_{k+1} = 0 \quad 0 \le i \le k$

Memoryless Quasi-Newton Method

- Also, for k

$$\underline{q}_{k+1}^{\mathrm{T}} H_{k+1} Q \underline{p}_i = \underline{q}_{k+1}^{\mathrm{T}} \underline{p}_i = \underline{p}_{k+1}^{\mathrm{T}} Q \underline{p}_i = 0 \quad \forall\ 0 \le i \le k$$

$$\Rightarrow\ H_{k+2} Q \underline{p}_i = H_{k+1} Q \underline{p}_i + \frac{\overbrace{\underline{p}_{k+1} \underline{p}_{k+1}^{\mathrm{T}} Q \underline{p}_i}^{=0}}{\underline{p}_{k+1}^{\mathrm{T}} \underline{q}_{k+1}} - \frac{H_{k+1} \underline{q}_{k+1} \overbrace{\left(\underline{q}_{k+1}^{\mathrm{T}} H_{k+1} Q \underline{p}_i \right)}^{=0}}{\underline{q}_{k+1}^{\mathrm{T}} H_{k+1} \underline{q}_{k+1}} = \underline{p}_i \quad \forall\ 0 \le i \le k$$

- Since $\underline{p}_k$'s are Q-orthogonal, DFP is a conjugate gradient method.
- $H_0 = I \ \Rightarrow$ DFP $\Rightarrow$ conjugate gradient method
- $H_k = I \ \ \forall k \ \Rightarrow$
  - BFGS inverse positive definite secant update is $\equiv$ conjugate gradient method.
  - Also known as memoryless Quasi-Newton.
  - Recall:

$$H_{k+1} = H_k + \frac{\underline{p}_k \underline{p}_k^{\mathrm{T}}}{\underline{p}_k^{\mathrm{T}} \underline{q}_k} - \frac{H_k \underline{q}_k \underline{q}_k^{\mathrm{T}} H_k}{\underline{q}_k^{\mathrm{T}} H_k \underline{q}_k} + \underline{q}_k^{\mathrm{T}} H_k \underline{q}_k \underline{r}_k \underline{r}_k^{\mathrm{T}}, \qquad \underline{r}_k = \frac{\underline{p}_k}{\underline{p}_k^{\mathrm{T}} \underline{q}_k} - \frac{H_k \underline{q}_k}{\underline{q}_k^{\mathrm{T}} H_k \underline{q}_k}$$

$$\Rightarrow\ H_{k+1} = I + \frac{\underline{p}_k \underline{p}_k^{\mathrm{T}}}{\underline{p}_k^{\mathrm{T}} \underline{q}_k} - \frac{\underline{q}_k \underline{q}_k^{\mathrm{T}}}{\underline{q}_k^{\mathrm{T}} \underline{q}_k} + \underline{q}_k^{\mathrm{T}} \underline{q}_k \underline{r}_k \underline{r}_k^{\mathrm{T}} = I - \frac{\underline{q}_k \underline{p}_k^{\mathrm{T}} + \underline{p}_k \underline{q}_k^{\mathrm{T}}}{\underline{p}_k^{\mathrm{T}} \underline{q}_k} + \left[1 + \frac{\underline{q}_k^{\mathrm{T}} \underline{q}_k}{\underline{p}_k^{\mathrm{T}} \underline{q}_k} \right] \frac{\underline{p}_k \underline{p}_k^{\mathrm{T}}}{\underline{p}_k^{\mathrm{T}} \underline{q}_k}$$

Memoryless QN & Sorensen-Wolfe CG

$$\underline{d}_{k+1} = -H_{k+1}\underline{g}_{k+1}$$

$$= -\underline{g}_{k+1} + \frac{\underline{q}_k \underline{p}_k^{\mathrm{T}} \underline{g}_{k+1} + \underline{p}_k \underline{q}_k^{\mathrm{T}} \underline{g}_{k+1}}{\underline{p}_k^{\mathrm{T}} \underline{q}_k} + \left[1 + \frac{\underline{q}_k \underline{q}_k^{\mathrm{T}}}{\underline{p}_k^{\mathrm{T}} \underline{q}_k}\right] \underline{p}_k \underline{p}_k^{\mathrm{T}} \underline{g}_{k+1}$$

- For exact line searches, $\underline{p}_k^{\mathrm{T}} \underline{g}_{k+1} = 0$

$$\Rightarrow \quad \boxed{\underline{d}_{k+1} = -\underline{g}_{k+1} + \frac{\underline{q}_k^{\mathrm{T}} \underline{g}_{k+1}}{\underline{d}_k^{\mathrm{T}} \underline{q}_k} \underline{d}_k} \quad \longleftarrow \text{ Sorenson-Wolfe formula}$$

❑ Convergence Properties: Reset $H_n = I$, i.e., every n steps.

- Since approximate $\nabla^2 f(\underline{x}_k)$ near minimum, get superlinear convergence.

$$\Rightarrow \quad \beta = \sup \lim_{k \to \infty} \frac{\left\| \underline{x}_{k+1} - \underline{x}^{'} \right\|}{\left\| \underline{x}_k - \underline{x}^{'} \right\|} = 0$$

- In general, BFGS positive definite secant update is better than DFP.
- Can improve convergence via scaling.

❑ Scaling: Convergence depends on $\lambda_i\left(H_k^{1/2} Q H_k^{1/2}\right) = \lambda_i\left(H_k Q\right)$

- Recall that for quadratic functions $H_{k+1} Q \underline{p}_i = \underline{p}_i \quad \forall\ 0 \le i \le k$

Scaling in Quasi-Newton Methods - 1

- Scaling cont.,
  - At each step of DFP (or Broyden), we move one eigen value at a time to unity.
  - $\kappa(H_k Q)$ may be worse than $\kappa(Q)$.

Idea: Scale H_k by a scalar γ_k so that the eigen values of $H_k Q$ are spread both below and above unity.

- If $\lambda_1 \le \lambda_2 \le \ldots \le \lambda_n$ are eigenvalues of $H_k Q$, we want to multiply H_k by γ_k where $\lambda_1 \le \dfrac{1}{\gamma_k} \le \lambda_n$.

$$\Rightarrow \quad \lambda_i\left[\gamma_k H_k Q\right] \quad \Rightarrow \quad \gamma_k \lambda_1 \le 1 \le \gamma_k \lambda_n$$

Recall:

$$\frac{\underline{q}_k^{\mathrm{T}} H_k \underline{q}_k}{\underline{p}_k^{\mathrm{T}} \underline{q}_k} = \frac{\underline{p}_k^{\mathrm{T}} Q H_k Q \underline{p}_k}{\underline{p}_k^{\mathrm{T}} Q \underline{p}_k} = \frac{\underline{r}_k^{\mathrm{T}} Q^{1/2} H_k Q^{1/2} \underline{r}_k}{\underline{r}_k^{\mathrm{T}} \underline{r}_k} \quad \text{where } \underline{r}_k = Q^{1/2} \underline{p}_k$$

- Since $\lambda_i\left(Q^{1/2} H_k Q^{1/2}\right) = \lambda_i\left(H_k^{1/2} Q H_k^{1/2}\right) = \lambda_i\left(H_k Q\right)$

$$\boxed{\lambda_i(AB) = \lambda_i(BA)}$$

$$\boxed{\lambda_1\left(H_k Q\right) \le \frac{\underline{q}_k^{\mathrm{T}} H_k \underline{q}_k}{\underline{p}_k^{\mathrm{T}} \underline{q}_k} \le \lambda_n\left(H_k Q\right)}$$

So, use $\gamma_k = \dfrac{\underline{p}_k^{\mathrm{T}} \underline{q}_k}{\underline{q}_k^{\mathrm{T}} H_k \underline{q}_k}$.

Scaling in Quasi-Newton Methods - 2

- The scaled quasi-Newton method then works as follows:
  - * $k = 0, \quad H_0 = I$
  - † $\underline{d}_k = -H_k \underline{g}_k$
  - Perform line search to find $\alpha_k = \arg\min_{\alpha} f\left(\underline{x}_k + \alpha \underline{d}_k\right)$
  - If $k = n$ Go back to *

    else $\tilde{H}_k = H_k \dfrac{\underline{p}_k^{\mathrm{T}} \underline{q}_k}{\underline{q}_k^{\mathrm{T}} H_k \underline{q}_k}$

    update H_k and go back to †.
  - For square root implementation $\tilde{L}_k = \sqrt{\dfrac{\underline{p}_k^{\mathrm{T}} \underline{q}_k}{\underline{q}_k^{\mathrm{T}} L_k L_k^{\mathrm{T}} \underline{q}_k}} \cdot L_k = \text{scaled } L_k$
  - For BFGS positive definite secant update, the scaling factor is $\sqrt{\dfrac{\underline{p}_k^{\mathrm{T}} \underline{q}_k}{\underline{p}_k^{\mathrm{T}} B_k \underline{p}_k}}$.

Then $\tilde{B}_k = B_k \cdot \dfrac{\underline{q}_k^{\mathrm{T}} \underline{p}_k}{\underline{p}_k^{\mathrm{T}} B_k \underline{p}_k}$ & $\tilde{\hat{L}}_k = \sqrt{\dfrac{\underline{p}_k^{\mathrm{T}} \underline{q}_k}{\underline{p}_k^{\mathrm{T}} \hat{L}_k \hat{L}_k^{\mathrm{T}} \underline{p}_k}} \cdot \hat{L}_k$

Neural Networks: Least Square Problems

- Least Square Problems

$$\min f(\underline{x}) = \frac{1}{2}\sum_{i=1}^{m} g_i^2(\underline{x})$$

$$\nabla \underline{f}(\underline{x}) = \sum_{i=1}^{m} \nabla \underline{g}_i(\underline{x}) g_i(\underline{x}) = \nabla \underline{\underline{g}}(\underline{x})\, \underline{g}(\underline{x}) = J^{\mathrm{T}}(\underline{x})\, \underline{g}(\underline{x})$$

$$\nabla^2 f(\underline{x}) = \nabla \underline{\underline{g}}(\underline{x}) \nabla \underline{\underline{g}}^{\mathrm{T}}(\underline{x}) + \sum_{i=1}^{m} \nabla^2 g_i(\underline{x}) g_i(\underline{x})$$

- Examples:

1) $$f(\underline{x}) = \frac{1}{2}\sum_{i=1}^{m}\left(b_i - \underline{a}_i^{\mathrm{T}}\underline{x}\right)^2 = \frac{1}{2}\sum_{i=1}^{m} e_i^2$$

$$\nabla \underline{f}(\underline{x}) = -\sum_{i=1}^{m}\left(b_i - \underline{a}_i^{\mathrm{T}}\underline{x}\right)\underline{a}_i = -\sum_{i=1}^{m} e_i \underline{a}_i$$

$$\boxed{\nabla \underline{e}_i = \underline{a}_i}$$

$$\nabla^2 f(\underline{x}) = \sum_{i=1}^{m} \underline{a}_i \underline{a}_i^{\mathrm{T}}$$

Examples of Least Squares Problems - 1

- Examples (continued) :

2) $f(\underline{x}) = \frac{1}{2}\sum_{i=1}^{m}\left(\sum_{j=1}^{n} e^{t_i x_j} - y_i\right)^2 ; \qquad \{t_i, y_i\}_{i=1}^{m}$

$$\nabla \underline{f}(\underline{x}) = \underbrace{\begin{bmatrix} t_1 e^{t_1 x_1} & t_2 e^{t_2 x_1} & t_m e^{t_m x_1} \\ t_1 e^{t_1 x_2} & \vdots & \vdots \\ \vdots & \vdots & \vdots \\ t_1 e^{t_1 x_n} & t_2 e^{t_2 x_n} & t_m e^{t_m x_n} \end{bmatrix}}_{\nabla \underline{\underline{g}}} \begin{bmatrix} \sum_{i=1}^{n} e^{t_1 x_j} - y_1 \\ \sum_{i=1}^{n} e^{t_2 x_j} - y_2 \\ \vdots \\ \sum_{i=1}^{n} e^{t_m x_j} - y_m \end{bmatrix}$$

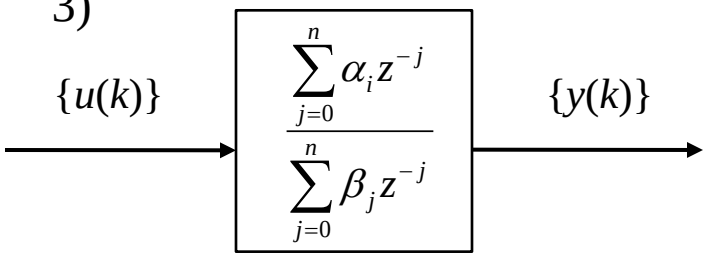
$$\nabla^2 f(\underline{x}) = \nabla \underline{\underline{g}} \nabla \underline{\underline{g}}^{\mathrm{T}} + \sum_{i=1}^{m} g_i(\underline{x}) \nabla^2 g_i(\underline{x})$$

$$\nabla^2 g_i(\underline{x}) = \begin{bmatrix} t_i^2 e^{t_i x_1} & & & \\ & t_i^2 e^{t_i x_2} & & \\ & & \ddots & \\ & & & t_i^2 e^{t_i x_n} \end{bmatrix}$$

Examples of Least Squares Problems - 2

- Examples (continued):

3)

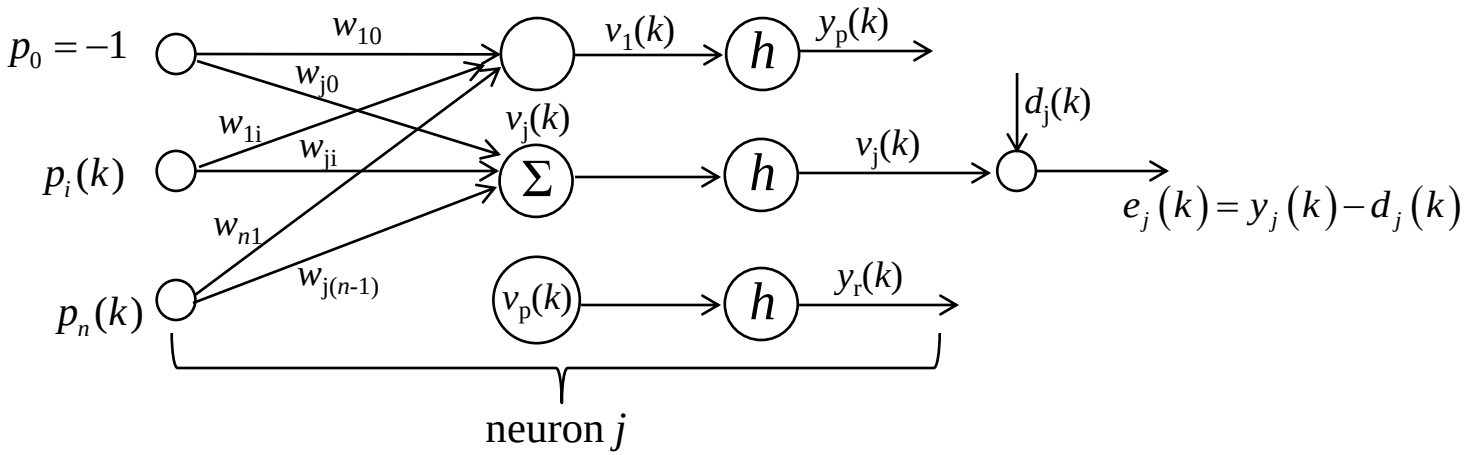


We want to estimate $\{\alpha_i\}$ and $\{\beta_i\}$ from

$\{u(k), y(k)\}_{k=1}^{m}$ $\quad m \gg 2n$

$$f(\underline{\alpha}, \underline{\beta}) = \frac{1}{2}\sum_{k=n}^{m}\left(\sum_{j=0}^{n}\alpha_j u_{k-j} - \sum_{j=0}^{n}\beta_j y_{k-j}\right)^2$$

4) Single layer neural networks: Given pairs of $\{\underline{p}(k), \underline{d}(k)\}_{k=1}^{m}$, find network weight W to minimize $\frac{1}{2m}\sum_{i=1}^{r} e_i^2(k) \rightarrow$ average error.



Examples of Least Squares Problems - 3

Examples (continued):

4) cont... $v_j(k) = \sum_{i=0}^{n} w_{ji} p_i(k) = \underline{w}_j^T \underline{p}(k)$

$$y_j(k) = h\left(v_j(k)\right)$$

$$h\left(v_j\right) = \begin{cases} \dfrac{1}{1+e^{-\sigma v_j}} & \in \quad (0,1) \text{ logistic function} \\ \dfrac{2}{\pi}\tan^{-1}\left(v_j\right) & \in (-1,1) \\ \dfrac{1-e^{-\sigma v_j}}{1+e^{-\sigma v_j}} & \text{bipolar for } \sigma = 1, \ \tanh \text{ for } \sigma = 2 \end{cases}$$

$$f(W) = \frac{1}{2m}\sum_{k=1}^{m}\sum_{j=1}^{r}\left(d_j(k) - h\left(\underline{w}_j^T \underline{p}(k)\right)\right)^2 = \frac{1}{2}\sum_{j=1}^{r} f_j\left(\underline{w}_j\right)$$

- Since the cost function is separable, we can minimize this function for each output separately.

$$f_j\left(\underline{w}_j\right) = \frac{1}{2m}\sum_{k=1}^{m}\left(d_j(k) - h\left(\underline{w}_j^T \underline{p}(k)\right)\right)^2$$

- So, the canonical problem is: $f(\underline{w}) = \frac{1}{2}\sum_{k=1}^{m}\left(d(k) - h\left(\underline{w}_j^T \underline{p}(k)\right)\right)^2$

General Neural Networks

- The Adaline (Adaptive Linear Neuron) Widrow-Hoff (1962)
  LMS algorithm (Least Mean Square)

$$h\left(\underline{w}^{\mathrm{T}}\,\underline{p}\right)=\underline{w}^{\mathrm{T}}\,\underline{p}$$

$$f\left(\underline{w}\right)=\frac{1}{2m}\sum_{k=1}^{m}\underbrace{\left(d\left(k\right)-\underline{w}^{T}\,\underline{p}\left(k\right)\right)^{2}}_{e^{2}\left(k\right)}$$

$$\nabla\underline{f}\left(\underline{w}\right)=-\frac{1}{m}\sum_{k=1}^{m}\left(d\left(k\right)-\underline{w}_{j}^{\mathrm{T}}\,\underline{p}\left(k\right)\right)\underline{P}\left(k\right)=-\frac{1}{m}\sum_{i=1}^{m}e\left(k\right)\underline{p}\left(k\right)$$

$$\nabla^{2}f\left(\underline{w}\right)=\frac{1}{m}\sum_{k=1}^{m}\underline{p}\left(k\right)\underline{p}^{\mathrm{T}}\left(k\right)$$

- General Neural Networks
  - Can approximate any I/O map accurately
  - Adapt to stochastic environments
  - Fault-tolerance & VLSI implementability
  - Theoretically, one hidden layer is sufficient

$$h\left(x\right)=\frac{1}{1+e^{-\sigma x}},\qquad h'\left(x\right)=\sigma h\left(x\right)\left(1-h\left(x\right)\right)$$

Multilayer Perceptrons

$$\nabla \underline{f}(\underline{w}) = -\sum_{k=1}^{m} e(k)\, h'\left(\underline{w}^{\mathrm{T}}\, \underline{p}(k)\right) \underline{p}(k)$$

$$= -\sigma \sum_{k=1}^{m} \left[d(k) - h\left(\underline{w}^{\mathrm{T}}\, \underline{p}(k)\right)\right] h\left(\underline{w}^{\mathrm{T}}\, \underline{p}(k)\right)\left[1 - h\left(\underline{w}^{\mathrm{T}}\, \underline{p}(k)\right)\right] \underline{p}(k)$$

Note that:

$$\frac{\partial f_k}{\partial w_i} = \underbrace{-e(k)\, h'\left(v(k)\right)}_{\delta(k)}\, p_i(k)$$

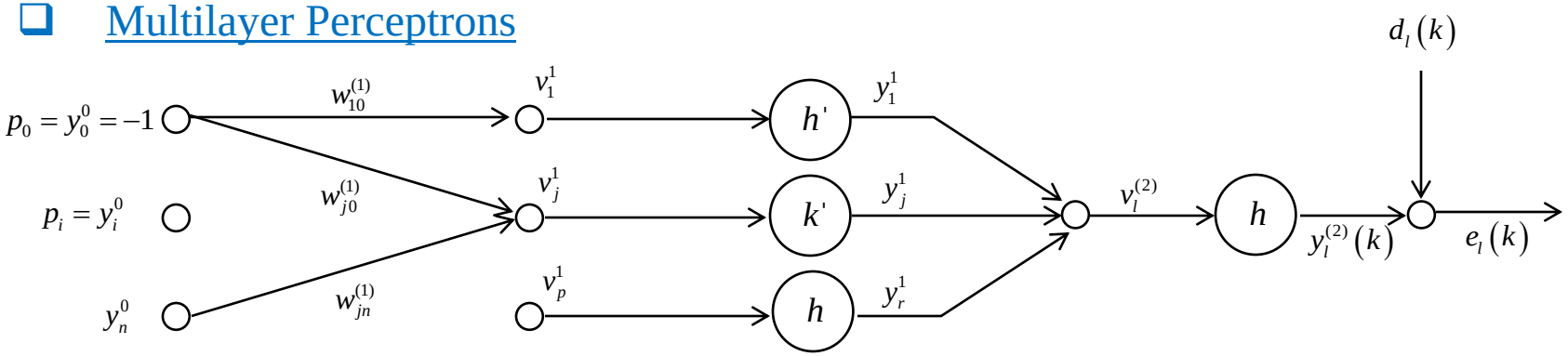
Local computation

local gradient

local gradient of neuron j

- In general, $\dfrac{\partial f}{\partial w_{ji}(k)} = -\delta_j(k)\, p_i(k)$. Second derivatives are a little more complex (see B.D. Ripley, *Pattern recognistion* & *Neural Networks*, 1996 or Bishop, 2006)

❑ Multilayer Perceptrons



Gradient Computation

$$y_j^1 = h^1\left[\left(\underline{w}_j^1\right)^{\mathrm{T}}\,\underline{y}^0\right]; \qquad y_k^2 = h^2\left[\left(\underline{w}_k^2\right)^{\mathrm{T}}\,\underline{y}^1\right] = h^2\left(\underline{w}_k^{\mathrm{T}}\begin{bmatrix} h\left[\left(\underline{w}_1^1\right)^{\mathrm{T}}\,\underline{y}^0\right] \\ \vdots \\ h\left[\left(\underline{w}_n^1\right)^{\mathrm{T}}\,\underline{y}^0\right] \end{bmatrix}\right)$$

Key: Can evaluate gradient recursively backwards

$$\frac{\partial f}{\partial w_{ji}^q(k)} = \delta_j^q(k)\, y_i^{(q-1)}(k); \qquad \delta_j^q(k) = h^1\left(v_j^q(k)\right)\sum_l \delta_l^{q+1}(k)\, w_{lj}^{q+1}(k)$$

end: $\delta_j^L(k) = -e(k)\,h^{L}\left(v^L(k)\right)$ (Last Layer)

- Can use SD, CG, QN, Newton, Modified Newton, etc.
- Gauss-Newton neglects second order terms.

$$\nabla^2 f = \left[\nabla \underline{\underline{g}} \nabla \underline{\underline{g}}^{\mathrm{T}} + \sum_{i=1}^{m} \nabla^2 g_i g_i\right] \approx \nabla \underline{\underline{g}} \nabla \underline{\underline{g}}^{\mathrm{T}}$$

$$\underline{x}_{k+1} = \underline{x}_k - \left[\nabla \underline{\underline{g}}(\underline{x}_k) \nabla \underline{\underline{g}}^{\mathrm{T}}(\underline{x}_k)\right]^{-1} \nabla \underline{\underline{g}}(\underline{x}_k)\, \underline{g}(\underline{x}_k)$$

- If not invertible:

$$\nabla \underline{\underline{g}}(\underline{x}_k) \nabla \underline{\underline{g}}^{\mathrm{T}}(\underline{x}_k) + \Delta_k I \quad \text{Levenberg-Marquardt}$$

Incremental Gradient Algorithm -1

- Some neat implementations:
  - Recursive $k = 1, 2, ..., m$ (Incremental Gradient Methods)

$$f(\underline{x}) = \sum_{l=1}^{m} f_l(\underline{x}) = \sum_{l=1}^{m} g_l^2(\underline{x}) \qquad g_1, g_2, ..., g_m \text{ come in sequentially}$$

$$\underline{x}_{k+1} = \underline{x}_k - \alpha_k \sum_{l=1}^{m} \nabla \underline{g}_l(\underline{x}_k) g_l(\underline{x}_k); \qquad \Psi_0 = \underline{x}_k$$

FOR $l = 1, m$ DO

$$\Psi_l = \Psi_{l-1} - \alpha_k^{(l)} \nabla \underline{g}_l(\Psi_{l-1}) g_l(\Psi_{l-1})$$

END

  - This is precisely what happens in LMS, Recursive Least Squares and Neural Networks
  - Use constant step size $\alpha_k^{(l)} = \alpha_k = \min\left(\gamma, \frac{\gamma_1}{k + \gamma_2}\right) = \eta_k$

- Example 1: LMS algorithm

$$\nabla \underline{f}(\underline{w}) = \sum_{l=1}^{m} \nabla f_l(\underline{w}) = -\sum_{l=1}^{m} e(l)\,\underline{p}(l) \quad \Rightarrow \quad \underline{w}(l+1) = \underline{w}(l) + \eta\rho(l)\,\underline{p}(l)$$

  - LMS = incremental SD

Incremental Gradient Algorithm -2

$$\underline{w}(l+1) = \underline{w}(l) + \eta\left[d(l) - \underline{p}^{\mathrm{T}}(l)\underline{w}(l)\right]\underline{p}(l)$$

$$= \left[I - \eta\,\underline{p}(l)\,\underline{p}^{\mathrm{T}}(l)\right]\underline{w}(l) + \eta d(l)\,\underline{p}(l)$$

- Convergence for $0 < \eta < \dfrac{2}{\lambda_{\max}}$; $\quad \lambda_{\max} = \lambda_{\max}\left[\ \frac{1}{l}\sum \underline{p}(l)\,\underline{p}^{\mathrm{T}}(l)\right] = R$

$$0 < \eta < \frac{2}{\mathrm{tr}(R)} = \frac{2}{\frac{1}{l}\sum p^{\mathrm{T}}(l)\,p(l)}$$

- Example 2: Back Propagation Neural Networks

$$w_{ji}^{q}(l+1) = w_{ji}^{q}(l) + \underbrace{\eta\delta_{j}^{q}(l)\,y_{i}^{(q-1)}(l)}_{\text{delta rule } \Delta w_{ji}^{q}(l)}$$

- Steepest descent with constant step size…very slow.
- Usually employ heavy ball method…adding momentum term to the update.

$$\underline{\Psi}_{l} = \underline{\Psi}_{l-1} + \eta_{k}\nabla\underline{g}_{i}(\underline{x}_{k})\,g_{i}(\underline{x}_{k}) + \beta\left(\underline{\Psi}_{l-1} - \underline{\Psi}_{l-2}\right)$$

NN: $$w_{ji}^{q}(l+1) = w_{ji}^{q}(l) + \underbrace{\eta\delta_{j}^{q}(l)\,y_{i}^{(q-1)}(l)}_{\text{delta rule}} + \beta\left(w_{ji}^{q}(l) - w_{ji}^{q}(l-1)\right)$$

Incremental Gradient Algorithm -3

$$\Delta w_{ji}(l) = \beta \Delta w_{ji}^{q}(l-1) + \eta \delta_j^q(l)\, y_i^{(q-1)}(l)$$

$$= \eta \sum_{k=0}^{l} \beta^{l-k} \delta_j^q(k)\, y_i^{(q-1)}(k) = -\eta \sum_{k=0}^{l} \beta^{l-k} \frac{\partial f}{\partial w_{ji}(k)} \qquad 0 < \beta < 1$$

$\Rightarrow$ same sign for $\dfrac{\partial f}{\partial w_{ji}(k)} \Rightarrow \Delta w_{ji} \uparrow$

$\Rightarrow$ alternate sign for $\dfrac{\partial f}{\partial w_{ji}(k)} \Rightarrow \Delta w_{ji} \downarrow$

- Randomize data blocks $\{l^s\}$
- Use non-symmetric sigmoids.

$$\phi(v) = a \tanh(\sigma v) = \frac{2a}{1+e^{-\sigma v}} - a$$

$$a = 1.716, \qquad \sigma = 2/3$$

- Conjugate gradient algorithms: → FR, SW, PRP

Incremental Gradient Algorithm -4

- Incremental Newton = Incremental Gauss-Newton = RLS

❑ Newton and Linear Least Squares:

$$\underline{x}_{k+1} = \underline{x}_k - \left(\sum_{i=1}^{m} \underline{a}_i \underline{a}_i^{\mathrm{T}} \right)^{-1} \left[A\underline{b} - \sum_{i=1}^{m} \underline{a}_i \underline{a}_i^{\mathrm{T}} \underline{x}_k \right] = \left(AA^{\mathrm{T}} \right)^{-1} A\underline{b}$$

$$\Psi_0 = \underline{x}_k; \qquad P_0 = 10^6 I$$

FOR $i = 1, \ldots, m$ DO

$$\underline{k}_i = \frac{P_{i-1}\underline{a}_i}{1 + \underline{a}_i^{\mathrm{T}} P_{i-1} \underline{a}_i}$$

$$\underline{\Psi}_i = \Psi_{i-1} + \underline{k}_i \left(b_i - \underline{a}_i^{\mathrm{T}} \underline{\Psi}_{i-1} \right)$$

$$P_i = P_{i-1} - \frac{P_{i-1}\underline{a}_i \underline{a}_i^{\mathrm{T}} P_{i-1}}{1 + \underline{a}_i^{\mathrm{T}} P_{i-1} \underline{a}_i}$$

END DO

Sherman – Morrison – Woodbury

$$\left(\sum_{i=1}^{m} \underline{a}_i \underline{a}_i^{\mathrm{T}} \right)^{-1} = P_{i-1}$$

$$P_i^{-1} = P_{i-1}^{-1} + \underline{a}_i \underline{a}_i^{\mathrm{T}}$$

$$P_i = P_{i-1} - \frac{P_{i-1}\underline{a}_i \underline{a}_i^{\mathrm{T}} P_{i-1}}{1 + \underline{a}_i^{\mathrm{T}} P_{i-1} \underline{a}_i}$$

$\Rightarrow$ want to put more emphasis on recent data: $P_{i-1} \rightarrow \frac{P_{i-1}}{\lambda}; \; 0 < \lambda \leq 1$

Incremental Gauss-Newton = EKF -1

- Idea can be extended to nonlinear least squares:

$$\underline{x}_{k+1} = \arg\min_{\underline{x}} \frac{1}{2}\sum_{i=1}^{m}\left\| g_i\left(\underline{x}_k\right) + \nabla \underline{g}_i^{\mathrm{T}}\left(\underline{x}_k\right)\left(\underline{x} - \underline{x}_k\right)\right\|_2^2$$

$$\underline{x}_{k+1} = \underline{x}_k - \left[\sum_{i=1}^{m} \nabla \underline{g}_i\left(\underline{x}_k\right)\nabla \underline{g}_i^{\mathrm{T}}\left(\underline{x}_k\right)\right]^{-1} \sum_{i=1}^{m} g_i\left(\underline{x}_k\right)\nabla \underline{g}_i\left(\underline{x}_k\right)$$

- If we want to implement this recursively, one way to do this is as follows:

$\Psi_0 = \underline{x}_k; \qquad P_0 = 10^6 I$

FOR $i = 1,\ldots,m$ DO

$$\underline{k}_i = \frac{P_{i-1}\nabla \underline{g}_i\left(\underline{x}_k\right)}{1 + \nabla \underline{g}_i^{\mathrm{T}}\left(\underline{x}_k\right) P_{i-1}\nabla \underline{g}_i\left(\underline{x}_k\right)}$$

$$\underline{\Psi}_i = \underline{\Psi}_{i-1} + \underline{k}_i g_i\left(\underline{\Psi}_{i-1}\right) \approx \underline{\Psi}_{i-1} + \underline{k}_i\left[g_i\left(\underline{x}_k\right) + \nabla g_i^{\mathrm{T}}\left(\underline{x}_k\right)\left(\underline{\Psi}_{i-1} - \underline{x}_k\right)\right]$$

$$P_i = P_{i-1} - \frac{P_{i-1}\nabla \underline{g}_i\left(\underline{x}_k\right)\nabla \underline{g}_i^{\mathrm{T}}\left(\underline{x}_k\right)P_{i-1}}{1 + \nabla \underline{g}_i^{\mathrm{T}}\left(\underline{x}_k\right) P_{i-1}\nabla \underline{g}_i\left(\underline{x}_k\right)}$$

END

Incremental Gauss-Newton = EKF -2

$\Rightarrow$ Forgetting factor $P_{i-1} \rightarrow P_{i-1}/\lambda \qquad 0 < \lambda \leq 1$

This is a special case of EKF.

- Application to Neural Networks

$$d(i) = h\left(\underline{w}^{\mathrm{T}}(i)\,\underline{y}(i)\right) + e(i) \quad \text{error}$$

Suppose we have estimate $\underline{\hat{w}}(i)$:

$$d(i) \approx h\left(\underline{\hat{w}}^{\mathrm{T}}(i)\,\underline{y}(i)\right) + \nabla h_{\underline{w}}^{\mathrm{T}}\left(\underline{\hat{w}}\right)\underbrace{\left[\underline{w}(i) - \underline{\hat{w}}(i)\right]}_{\Delta \underline{w}(i)} + e(i)$$

$$\Delta d(i) = d(i) - h\left(\underline{\hat{w}}^{\mathrm{T}}(i)\,\underline{y}(i)\right)$$

$$\Delta d(i) = \underline{q}^{\mathrm{T}}(i)\,\Delta \underline{w}(i) + e(i)$$

$$\underline{q}(i) = \sigma h\left(\underline{\hat{w}}^{\mathrm{T}}(i)\,\underline{y}(i)\right)\left[1 - h\left(\underline{\hat{w}}^{\mathrm{T}}(i)\,\underline{y}(i)\right)\right]\underline{y}(i)$$

Incremental Gauss-Newton = EKF -2

- Application to Neural Networks, continued

$$\Psi_0 = \hat{\underline{w}}(0); \qquad P_0 = 10^6 I$$

FOR $i = 1, \ldots, m$ DO

$$\underline{k}_i = \frac{P_{i-1}\underline{q}(i)}{1 + \underline{q}^{\mathrm{T}}(i) P_{i-1} \underline{q}(i)}$$

$$\hat{w}_i = \hat{w}_{i-1} + \underline{k}_i \left[d(i) - h\left(\hat{\underline{w}}_{i-1} \underline{y}(i)\right) + q_i^{\mathrm{T}} \left(\hat{\underline{w}}_{i-1} - \hat{\underline{w}}_{i-2}\right) \right]$$

$$P_i = P_{i-1} - \frac{P_{i-1}\underline{q}(i)\underline{q}^{\mathrm{T}}(i)P_{i-1}}{1 + \underline{q}^{\mathrm{T}}(i) P_{i-1} \underline{q}(i)}$$

END

Fading memory $P_{i-1} \rightarrow \dfrac{P_{i-1}}{\lambda} \qquad 0 < \lambda \leq 1$

Summary

- Quasi-Newton (QN) Methods
- Motivation from Quadratic Functions
- Square-root Implementation
  - DFP & BFGS versions
  - Relation to Filtering with Perfect Observations
- Properties of QN Methods
  - Quadratic Termination
  - Convergence Properties
  - Scaling
- Incremental Methods for Neural Networks
  - Back propagation, momentum methods
  - LMS and Recursive least squares
  - Gauss-Newton $\Rightarrow$ Extended Kalman Filter