Numerical Optimization

Unit 5: Large-Scale Unconstrained Optimization

Kai-Ting Weng, David Liao, Che-Rung Lee

Department of Computer Science
National Tsing Hua University

October 18, 2022

What's new for large-scale problems?

Recall what we have learned.

- Gradient descent method: basic ingredient for optimization.
- Newton's method: providing faster convergence.
- Modified Newton: avoid the non-positive definite Hessian.
- Line Search and Trust Region: methods that guarantee convergence.
- Quasi-Newton Methods: avoid the computation of Hessian matrices.
- CG: avoid the computation of solving linear systems.

But for large-scale problems, $O(n^2)$ is still too expensive, for both computation and storage. Can we do better?

- Inexact Newton (Newton CG)
- Limited memory quasi-Newton.
- Sparse quasi-Newton.

1. Inexact Newton methods

- Recall that the basic Newton search direction $\vec{p}_k$ is obtained by solving the symmetric $n \times n$ linear system

$$\nabla^2 f_k \vec{p}_k = -\nabla f_k \tag{1}$$

- But for modified Newton or quasi-Newton methods, we learned that $\vec{p}_k$ need not be the exact solution of Newton's direction.
- Inexact Newton methods are approaches using the CG with modifications to handle negative curvature in the Hessian $\nabla^2 f_k$.
- A direction of negative curvature $\vec{p}$ is one that satisfies $\vec{p}^T \nabla^2 f(x_k) \vec{p} < 0$.

Local convergence of inexact Newton methods

- Most rules for terminating the iterative solver for (1) are based on the residual

$$\vec{r}_k = \nabla^2 f_k \vec{p}_k + \nabla f_k \tag{2}$$

where $\vec{p}_k$ is the inexact Newton step. Usually, we terminate the CG iterations when

$$\|\vec{r}_k\| \leq \eta_k \|\nabla f_k\| \tag{3}$$

where the sequence η_k (with $0 < \eta_k < 1$ for all k) is called the **forcing sequence**.

- How the rate of convergence of inexact Newton methods based on (3) is affected by the choice of the forcing sequence.
- For example, one can set

$$\eta_k = \min(0.5, \sqrt{\|\nabla f_k\|}) \tag{4}$$

Convergence of inexact Newton method

Theorem (7.1)

Suppose that ∇f exists and is continuous near a minimizer $\vec{x}^*$, and $\nabla^2 f(\vec{x}^*)$ is positive definite. Consider $\vec{x}_{k+1} = \vec{x}_k + \vec{p}_k$ where $\vec{p}_k$ satisfies (3), and assume that $\eta_k \leq \eta$ for some constant $\eta \in [0, 1)$. If $\vec{x}_0$ is sufficiently near $\vec{x}^*$, the sequence $\{\vec{x}_k\}$ converges to $\vec{x}^*$ and satisfies

$$\|\nabla^2 f(\vec{x}^*)(\vec{x}_{k+1} - \vec{x}^*)\| \leq \hat{\eta}\|\nabla^2 f(\vec{x}^*)(\vec{x}_k - \vec{x}^*)\|$$

for some constant $\hat{\eta}$ with $\eta < \hat{\eta} < 1$.

Theorem (7.2)

Suppose the conditions of Theorem 7.1 hold, and $\{\vec{x}_k\}$ generated by the inexact Newton method converge to $\vec{x}^*$. The rate of convergence is superlinear if $\eta_k \to 0$. In addition, if $\nabla^2 f$ is Lipschitz continuous for $\vec{x}$ near $\vec{x}^*$ and $\eta_k = O(\|\nabla f_k\|)$, the convergence is quadratic.

Line search Newton-CG method

- In the **line search Newton–CG method**, also known as the **truncated Newton method**, we compute the search direction by applying the CG method to the Newton equations (1) and attempt to satisfy a termination test of the form (3).
- However, the CG method is designed to solve positive definite systems, and the Hessian $\nabla^2 f_k$ may have negative eigenvalues when $\vec{x}_k$ is not close to a solution.
- Therefore, we terminate the CG iteration as soon as a direction of negative curvature is generated.
  - can produce a search direction $\vec{p}_k$ that is a descent direction.
  - can guarantee that the fast convergence rate of the pure Newton method is preserved, provided that the step length $\alpha_k = 1$ is used whenever it satisfies the acceptance criteria

Line search Newton-CG method

The differences between the line search Newton-CG and CG

1. The specific starting point $\vec{x}_0 = 0$ is used.
2. The use of a tolerance ϵ_k allows the CG iterations to terminate at an inexact solution.
3. The negative curvature test $\vec{p}_j^T B_k \vec{p}_j \leq 0$ ensures that $\vec{p}_k$ is a descent direction for f at $\vec{x}_k$

- Newton-CG is well suited for large problems, but when the Hessian $\nabla^2 f_k$ is nearly singular, the line search Newton–CG direction can be long and of poor quality.
- The **trust-region Newton–CG method** is preferable for nearly singular Hessian.

Trust-region Newton-CG method

- The trust-region Newton-CG method solves the constrained optimization sub-problem.

$$\min_{\vec{p}} m_k(\vec{p}) = f_k + \vec{g}_k^T \vec{p} + \frac{1}{2} \vec{p}^{\,T} H_k \vec{p} \;\text{ s.t. }\; \|\vec{p}\| \leq \Delta_k.$$

- The choice of the tolerance ϵ_k of residual at each iteration is important in keeping the overall cost low.
- Near a well-behaved solution $\vec{x}^*$, the trust-region bound becomes inactive, and the method reduces to the inexact Newton method analyzed in Theorems 7.1 and 7.2.
- Rapid convergence can be obtained in these circumstances by choosing ϵ_k in a similar to (4).

Termination of Trust-region Newton-CG method

Theorem (7.3)

The sequence of vectors $\vec{z}_j$ generated by the trust-region inexact Newton-CG satisfies

$$0 = \|\vec{z}_0\|_2 < \cdots < \|\vec{z}_j\|_2 < \|\vec{z}_{j+1}\|_2 < \ldots < \|\vec{p}_k\|_2 \leq \Delta_k$$

The differences between Trust-region Newton-CG and CG

1. Terminates when $\vec{p}_j^T B_k \vec{p}_j \leq 0$, which means it encounters a direction of negative curvature in $\nabla^2 f_k$.
2. Terminates when $\|\vec{z}_{j+1}\| \geq \Delta_k$, which means it violates the trust-region bound $\|\vec{p}\| < \Delta$. (based on Theorem 7.3)
3. Terminates when $||\vec{r}_{j+1}|| \leq \epsilon_k$, which means it satisfies a convergence tolerance defined by a parameter ϵ_k

2. Limited memory quasi-Newton

- Even though quasi-Newton methods need not compute the Hessian matrix, they need to store the Hessian matrices for updates. For large-scale problems, the storage is a critical problem.
- Instead of storing fully dense $n \times n$ matrices, limited-memory quasi-Newton methods save only m vectors of length n, $m \ll n$, that represent the approximations implicitly.
  - The storage and computation can be reduced from $O(n^2)$ to $O(mn)$.
- We will discuss L-BFGS (limited memory BFGS) mainly, but the same techniques, such as delayed update, can be applied to similar algorithms.

Idea of L-BFGS

- The BFGS method updates the inverse Hessian H_{k+1} using the formula,

$$H_{k+1} = (I - \rho_k \vec{s}_k \vec{y}_k^T) H_k (I - \rho_k \vec{y}_k \vec{s}_k^T) + \rho_k \vec{s}_k \vec{s}_k^T \text{ where } \rho_k = \frac{1}{\vec{y}_k^T \vec{s}_k}$$

- Let $V_k = I - \rho_k \vec{y}_k \vec{s}_k^T$. Above formula can be written as

$$H_{k+1} = V_k^T H_k V_k + \rho_k \vec{s}_k \vec{s}_k^T. \tag{5}$$

- Equation (5) is a recursive form, so we can expand it.

$$H_{k+1} = V_k^T (V_{k-1}^T H_{k-1} V_{k-1} + \rho_{k-1} \vec{s}_{k-1} \vec{s}_{k-1}^T) V_k + \rho_k \vec{s}_k \vec{s}_k^T \tag{6}$$

- If we only expand it m times, then we have the m step L-BFGS algorithm.

Computation of $H_k \nabla f$

- We only need Hessian for computing searching direction: $\vec{p} = -H_k \nabla f$, where H_k is the inverse Hessian.

Two loop algorithm for computing $H_k \nabla f$

$q \leftarrow \nabla f$
for $i = k-1, k-2, \ldots, k-m$ **do**
    $\alpha_i \leftarrow \rho_i s_i^T q$
    $q \leftarrow q - \alpha_i y_i$
end for
$r \leftarrow H_k q$
for $i = k-m, k-m+1, \ldots, k-1$ **do**
    $\beta \leftarrow \rho_i y_i^T r$
    $r \leftarrow r + s_i(\alpha_i - \beta)$
end for

Example

1: Let $m = 2$, and $\nabla f_k = \vec{q}_k$. From (6), we have

$$H_k\vec{q}_k = V_k^T(V_{k-1}^T H_{k-1} V_{k-1} + \rho_{k-1}\vec{s}_{k-1}\vec{s}_{k-1}^T)V_k\vec{q}_k + \rho_k\vec{s}_k\vec{s}_k^T\vec{q}_k$$

3: Let $\alpha_k = \rho_k\vec{s}_k^T\vec{q}_k$. We have $V_k\vec{q}_k = (I - \rho_k\vec{y}_k\vec{s}_k^T)\vec{q}_k = \vec{q}_k - \alpha_k\vec{y}_k$.

$$H_k\vec{q}_k = V_k^T(V_{k-1}^T H_{k-1} V_{k-1} + \rho_{k-1}\vec{s}_{k-1}\vec{s}_{k-1}^T)(\vec{q}_k - \alpha_k\vec{y}_k) + \alpha_k\vec{s}_k$$

4: Let $\vec{q}_{k-1} = \vec{q}_k - \alpha_k\vec{y}_k$.

$$H_k\vec{q}_k = V_k^T(V_{k-1}^T H_{k-1} V_{k-1} + \rho_{k-1}\vec{s}_{k-1}\vec{s}_{k-1}^T)\vec{q}_{k-1} + \alpha_k\vec{s}_k$$

3: Let $\alpha_{k-1} = \rho_{k-1}\vec{s}_{k-1}^T\vec{q}_{k-1}$.

$$\begin{aligned} H_k\vec{q}_k =& V_k^T(V_{k-1}^T H_{k-1} V_{k-1}\vec{q}_{k-1} + \alpha_{k-1}\vec{s}_{k-1}) + \alpha_k\vec{s}_k \\ =& V_k^T(V_{k-1}^T H_{k-1}(\vec{q}_{k-1} - \alpha_{k-1}\vec{y}_{k-1}) + \alpha_{k-1}\vec{s}_{k-1}) + \alpha_k\vec{s}_k \end{aligned}$$

4: Let $\vec{q}_{k-2} = \vec{q}_{k-1} - \alpha_{k-1}\vec{y}_{k-1}$.

$$H_k\vec{q}_k = V_k^T(V_{k-1}^T H_{k-1}\vec{q}_{k-2} + \alpha_{k-1}\vec{s}_{k-1}) + \alpha_k\vec{s}_k$$

Example–continue

6: Let $\vec{r}_{k-2} = H_{k-1}\vec{q}_{k-2}$.

$$H_k\vec{q}_k = V_k^T(V_{k-1}^T\vec{r}_{k-2} + \alpha_{k-1}\vec{s}_{k-1}) + \alpha_k\vec{s}_k$$

8: Let $\beta_{k-1} = \rho_{k-1}\vec{y}_{k-1}^T\vec{r}_{k-2}$. We have

$$V_{k-1}^T\vec{r}_{k-2} = (I - \rho_{k-1}\vec{s}_{k-1}\vec{y}_{k-1}^T)\vec{r}_{k-2} = \vec{r}_{k-2} - \beta_{k-1}\vec{s}_{k-1}$$

$$H_k\vec{q}_k = V_k^T(\vec{r}_{k-2} + (\alpha_{k-1} - \beta_{k-1})\vec{s}_{k-1}) + \alpha_k\vec{s}_k$$

9: Let $\vec{r}_{k-1} = \vec{r}_{k-2} + (\alpha_{k-1} - \beta_{k-1})\vec{s}_{k-1}$. We have

$$H_k\vec{q}_k = V_k^T\vec{r}_{k-1} + \alpha_k\vec{s}_k$$

8: Let $\beta_k = \rho_k\vec{y}_k^T\vec{r}_{k-1}$. $V_k^T\vec{r}_{k-1} = (I - \rho_k\vec{s}_k\vec{y}_k^T)\vec{r}_{k-1} = \vec{r}_{k-1} - \beta_k\vec{s}_k$.

$$H_k\vec{q}_k = \vec{r}_{k-1} + (\alpha_k - \beta_k)\vec{s}_k$$

9: Return $\vec{r}_k = \vec{r}_{k-1} + (\alpha_k - \beta_k)\vec{s}_k$.

L-BFGS

- If we choose $H_k = I$ or some diagonal matrix, such as $H_k = \frac{\vec{s}_{k-1}^T \vec{y}_{k-1}}{\vec{y}_{k-1}^T \vec{y}_{k-1}} I$, the two loop algorithm can be done in $O(mn)$ time.

L-BFGS

Choose starting $\vec{x}_0$, and $m > 0$
for $k = 0, 1, \ldots$ until convergence **do**
    Choose H_k.
    Compute $\vec{p}_k = -H_k \nabla f$ using two-loop algorithm.
    Compute a proper step length α_k and $\vec{x}_{k+1} \leftarrow \vec{x}_k + \alpha_k \vec{p}_k$
    if $k > m$ **then**
        Discard $\{\vec{s}_{k-m}, \vec{y}_{k-m}\}$
    end if
    Compute and save $\vec{s}_k = \vec{x}_{k+1} - \vec{x}_k$ and $\vec{y}_k = \nabla f_{k+1} - \nabla f_k$.
end for

Compact representation of Hessian inverse

- The Hessian inverse of BFGS can be expanded as

$$\begin{aligned} H_{k+1} =& (V_k^T \cdots V_{k-m+1}^T) H_{k-m} (V_{k-m+1} \cdots V_k) \\ &+ \rho_{k-m+1} (V_{k-1}^T \cdots V_{k-m+2}^T) \vec{s}_{k-m+1} \vec{s}_{k-m+1}^T (V_{k-m+2} \cdots V_{k-1}) \\ &+ \rho_{k-m+2} (V_{k-1}^T \cdots V_{k-m+3}^T) \vec{s}_{k-m+2} \vec{s}_{k-m+2}^T (V_{k-m+3} \cdots V_{k-1}) \\ &+ \cdots + \rho_k \vec{s}_k \vec{s}_k^T \end{aligned} \tag{7}$$

Theorem (1. Compact representation of H_k)

$$H_k = H_0 + \begin{bmatrix} S_k & H_0 Y_k \end{bmatrix} \begin{bmatrix} R_k^{-T} (D_k + Y_k^T H_0 Y_k) R_k^{-1} & -R_k^{-T} \\ -R_k^{-1} & 0 \end{bmatrix} \begin{bmatrix} S_k^T \\ Y_k^T H_0 \end{bmatrix}$$

- $S_k = [\vec{s}_0, \ldots, \vec{s}_{k-1}]$ *and* $Y_k = [\vec{y}_0, \ldots, \vec{y}_{k-1}]$.
- $R_k(i,j) = \begin{cases} \vec{s}_{i-1}^T \vec{y}_{j-1} & \text{for } i \le j \\ 0 & \text{otherwise.} \end{cases}$
- D_k *is a diagonal matrix,* $D_k = \mathrm{diag}[\vec{s}_0^T \vec{y}_0, \ldots, \vec{s}_{k-1}^T \vec{y}_{k-1}]$

Lemma: Compact form of $V_0 \cdots V_{k-1}$

Theorem

Let $V_k = I - \vec{y}_k \vec{s}_k^T / \vec{s}_k^T \vec{y}_k$. Then $V_0 \cdots V_{k-1} = I - Y_k R_k^{-1} S_k$.

For $k = 1$, $V_0 = I - \vec{y}_0 \vec{s}_0^T / \vec{s}_0^T \vec{y}_0 = I - \vec{y}_0 [\vec{s}_0^T \vec{y}_0]^{-1} \vec{s}_0^T$.

Assume it is hold for k. For $k+1$, $R_{k+1} = \begin{bmatrix} R_k & S_k^T \vec{y}_k \\ 0 & \vec{s}_k^T \vec{y}_k \end{bmatrix}$ whose inverse is

$R_{k+1}^{-1} = \begin{bmatrix} R_k^{-1} & -\rho_k R_k^{-1} S_k^T \vec{y}_k \\ 0 & \rho_k \end{bmatrix}$, where $\rho_k = 1/\vec{s}_k^T \vec{y}_k$.

$$
\begin{aligned}
I - Y_{k+1} R_{k+1}^{-1} S_{k+1} =& I - \begin{bmatrix} Y_k & \vec{y}_k \end{bmatrix} \begin{bmatrix} R_k^{-1} & -\rho_k R_k^{-1} S_k^T \vec{y}_k \\ 0 & \rho_k \end{bmatrix} \begin{bmatrix} S_k^T \\ \vec{s}_k^T \end{bmatrix} \\
=& I - Y_k R_k^{-1} S_k^T + \rho_k Y_k R_k^{-1} S_k^T \vec{y}_k \vec{s}_k^T - \rho_k \vec{y}_k \vec{s}_k^T \\
=& (I - Y_k R_k^{-1} S_k^T)(I - \rho_k \vec{y}_k \vec{s}_k^T)
\end{aligned}
$$

By induction,

$$(V_0 \cdots V_{k-1}) V_k = (I - Y_k R_k^{-1} S_k^T)(I - \rho_k \vec{y}_k \vec{s}_k^T) = I - Y_{k+1} R_{k+1}^{-1} S_{k+1}$$

Proof of theorem 1

We have $H_{k+1} = (I - \rho_k \vec{s}_k \vec{y}_k^T) H_k (I - \rho_k \vec{y}_k \vec{s}_k^T) + \rho_k \vec{s}_k \vec{s}_k^T$ where $\rho_k = \frac{1}{\vec{y}_k^T \vec{s}_k}$.
Let's write it as $H_k = M_k + N_k$, where

$$M_0 = H_0, \qquad M_{k+1} = V_k^T M_k V_k = (I - S_k R_k^{-T} Y_k^T) H_0 (I - Y_k R_k^{-1} S_k^T)$$
$$N_1 = \rho_0 \vec{s}_0 \vec{s}_0^T, \quad N_{k+1} = V_k^T N_k V_k^T + \rho_k \vec{s}_k \vec{s}_k^T.$$

We can show that $N_k = S_k R_k^{-1} D_k R_k^{-1} S_k^T$ by induction.
For $k = 1$, that is true. Assume it holds for k. For $k+1$,

$$N_{k+1} = V_k^T S_k R_k^{-1} D_k R_k^{-1} S_k^T V_k + \rho_k \vec{s}_k \vec{s}_k^T$$
$$R_k^{-1} S_k^T V_k = R_k^{-1} S_k^T (I - \rho_k \vec{y}_k \vec{s}_k) = \begin{bmatrix} R_k^{-1} & -\rho_k R_k^{-1} S_k^T \vec{y}_k \end{bmatrix} \begin{bmatrix} S_k^T \\ \vec{s}_k^T \end{bmatrix}$$
$$= [I \ O] \begin{bmatrix} R_k^{-1} & -\rho_k R_k^{-1} S_k^T \vec{y}_k \\ 0 & \rho_k \end{bmatrix} S_{k+1}^T = [I \ O] R_{k+1}^{-1} S_{k+1}^T$$

Proof of theorem 1–continue

It can be shown that $\vec{s}_k = S_{k+1} R_{k+1}^{-T} \vec{e}_{k+1} / \rho_k$, so that

$$
\begin{aligned}
N_{k+1} =& V_k^T S_k R_k^{-1} D_k R_k^{-1} S_k^T V_k + \rho_k \vec{s}_k \vec{s}_k^T \\
=& S_{k+1} R_{k+1}^{-T} \begin{bmatrix} I & O \end{bmatrix} D_k \begin{bmatrix} I \\ O \end{bmatrix} R_{k+1}^{-1} S_{k+1}^T + \frac{S_{k+1} R_{k+1}^{-T} \vec{e}_{k+1} \vec{e}_{k+1}^T R_{k+1}^{-1} S_{k+1}^T}{\rho_k} \\
=& S_{k+1} R_{k+1}^{-T} \begin{bmatrix} D_k & O \\ O & 1/\rho_k \end{bmatrix} R_{k+1}^{-1} S_{k+1}^T = S_{k+1} R_{k+1}^{-T} D_{k+1} R_{k+1}^{-1} S_{k+1}^T
\end{aligned}
$$

Put them together

$$
\begin{aligned}
H_k =& (I - S_k R_k^{-T} Y_k^T) H_0 (I - Y_k R_k^{-1} S_k^T) + S_k R_k^{-T} D_k R_k^{-1} S_k^T \\
=& H_0 + \begin{bmatrix} S_k & H_0 Y_k \end{bmatrix} \begin{bmatrix} R_k^{-T} (D_k + Y_k^T H_0 Y_k) R_k^{-1} & -R_k^{-T} \\ R_k^{-1} & O \end{bmatrix} \begin{bmatrix} S_k^T \\ Y_k^T H_0 \end{bmatrix}
\end{aligned}
$$

Compact representation of Hessian

The Hessian matrix of BFGS (not its inverse) is

$$B_{k+1} = B_k - \frac{B_k \vec{s}_k \vec{s}_k^T B_k}{\vec{s}_k^T B_k \vec{s}_k} + \frac{\vec{y}_k \vec{y}_k^T}{\vec{y}_k^T \vec{s}_k}$$

Theorem (2. Compact representation of B_k)

$$B_k = B_0 - \begin{bmatrix} B_0 S_k & Y_k \end{bmatrix} \begin{bmatrix} S_k^T B_0 S_k & L_k \\ L_k^T & -D_k \end{bmatrix}^{-1} \begin{bmatrix} S_k^T B_0 \\ Y_k^T \end{bmatrix}$$

- $S_k = [\vec{s}_0, \ldots, \vec{s}_{k-1}]$ *and* $Y_k = [\vec{y}_0, \ldots, \vec{y}_{k-1}]$.
- L_k *is an upper triangular matrix, where* $(L_k)_{i,j} = \vec{s}_{i-1}^T \vec{y}_{j-1}$ *for* $i > j$ *and* $(L_k)_{i,j} = 0$ *otherwise.*
- D_k *is a diagonal matrix,* $D_k = \mathrm{diag}[\vec{s}_0^T \vec{y}_0, \ldots, \vec{s}_{k-1}^T \vec{y}_{k-1}]$.

Proof of theorem 2

- From theorem 1, $H_k = H_0 + U_k C_k U_k^T$ where $U_k = \begin{bmatrix} S_k & H_0 Y_k \end{bmatrix}$ and $C_k = \begin{bmatrix} R_k^{-T}(D_k + Y_k^T H_0 Y_k) R_k^{-1} & -R_k^{-T} \\ R_k^{-1} & O \end{bmatrix}$.
- The inverse of C_k is $C_k^{-1} = \begin{bmatrix} O & -R_k \\ -R_k^T & -(D_k + Y_k^T H_0 Y_k) \end{bmatrix}$.
- Using the Sherman-Morrison-Woodbury formula

$$\begin{aligned} B_k =& H_k^{-1} = B_0 - B_0 U_k (I + C_k U_k^T B_0 U_k)^{-1} C_k U_k^T B_0 \\ =& B_0 - B_0 U_k (C_k^{-1} + U_k^T B_0 U_k)^{-1} U_k^T B_0 \end{aligned}$$

- $U_k^T B_0 U_k = \begin{bmatrix} S_k^T \\ Y_k^T H_0 \end{bmatrix} B_0 \begin{bmatrix} S_k & H_0 Y_k \end{bmatrix} = \begin{bmatrix} S_k^T B_0 S_k & S_k^T Y_k \\ Y_k^T S_k & Y_k^T H_0 Y_k \end{bmatrix}$
- Let $L_k = S_k^T Y_k - R_k$. Then $C_k^{-1} + U_k^T B_0 U_k = \begin{bmatrix} S_k^T B_0 S_k & L_k \\ L_k^T & -D_k \end{bmatrix}$.
- Also since $B_0 = H_0^{-1}$, $B_0 U_k = [B_0 S_k \;\; Y_k]$. The result will be obtained by assembling every piece together.

Compact representation of L-BFGS

- Let $B_k = \delta_k I$ and $\delta_k = \frac{\vec{y}_{k-1}^T \vec{y}_{k-1}}{\vec{s}_{k-1}^T \vec{y}_{k-1}}$. The matrix form of L-BFGS is

$$B_k = \delta_k I - \begin{bmatrix} \delta_k S_k & Y_k \end{bmatrix} \begin{bmatrix} \delta_k S_k^T S_k & L_k \\ L_k^T & -D_k \end{bmatrix}^{-1} \begin{bmatrix} \delta_k S_k^T \\ Y_k^T \end{bmatrix}$$

  where
  - $S_k = [\vec{s}_{k-m}, \ldots, \vec{s}_{k-1}]$ and $Y_k = [\vec{y}_{k-m}, \ldots, \vec{y}_{k-1}]$.
  - L_k is an upper triangular matrix, where $(L_k)_{i,j} = \vec{s}_{k-m-1+i}^T \vec{y}_{k-m-1+j}$ for $i > j$ and $(L_k)_{i,j} = 0$ otherwise.
  - D_k is a diagonal matrix, $D_k = \mathrm{diag}[\vec{s}_{k-m}^T \vec{y}_{k-m}, \ldots, \vec{s}_{k-1}^T \vec{y}_{k-1}]$.
- The computation of B_k takes $O(mn + m^3)$ time. When $m \ll n$, it is $O(mn)$ time.

Memoryless BFGS

- If we set $m = 1$ and let $H_k = I$, the inverse Hessian H_{k+1} becomes

$$H_{k+1} = \left(I - \frac{\vec{s}_k \vec{y}_k^T}{\vec{y}_k^T \vec{s}_k} \right) \left(I - \frac{\vec{y}_k \vec{s}_k^T}{\vec{y}_k^T \vec{s}_k} \right) + \frac{\vec{s}_k \vec{s}_k^T}{\vec{y}_k^T \vec{s}_k}.$$

- The search direction is $\vec{p}_{k+1} = -H_{j+1} \nabla f_{k+1}$.
- If $\nabla f_{k+1}^T \vec{s}_k = 0$, the search direction becomes

$$\vec{p}_{k+1} = -\nabla f_{k+1} + \frac{\nabla f_{k+1}^T \vec{y}_k}{\vec{y}_k^T \vec{s}_k} \vec{s}_k,$$

which is the same as the Hestenes-Stiefel CG algorithm.

3. Sparse quasi-Newton updates

- Quasi-Newton method is used as an alternative when the Hessian matrix is too expensive to compute.
- For a large scale problem, if the true Hessian is a sparse matrix, then we hope the approximation matrix B_k used in quasi-Newton method has the same sparse pattern.
- Moreover, we hope that B_{k+1} is close to B_k and still satisfies the secant condition.

Updates of the sparse approximation matrix B

$$B_{k+1} = \arg\min_B \|B - B_k\|_F^2 = \arg\min_B \sum [B_{ij} - (B_k)_{ij}]^2$$

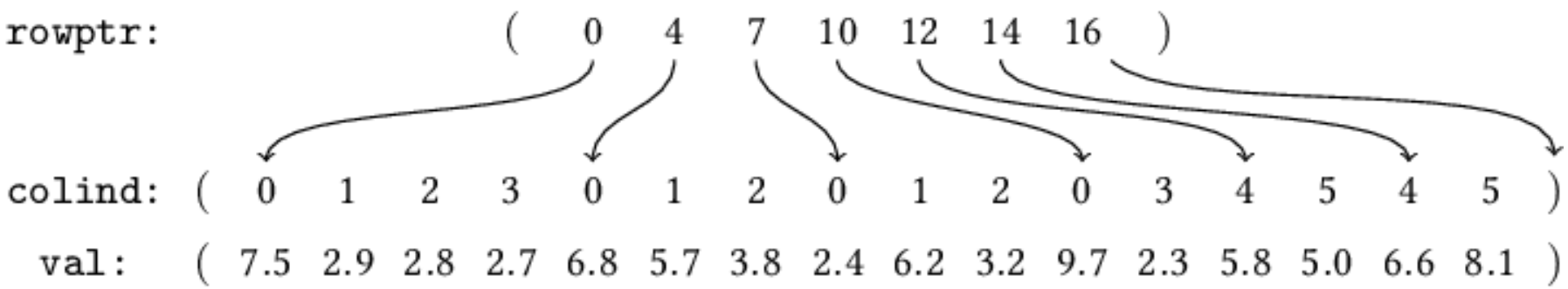
subject to

- $B_{k+1}\vec{s}_{k+1} = \vec{y}_{k+1}$ and $B_{k+1} = B_{k+1}^T$
- $(B_{k+1})_{ij} = 0$ for $(i,j) \notin \Omega$, where $\Omega = \{(i,j) | [\nabla^2 f(\vec{x})]_{ij} \neq 0\}$

What is a sparse matrix?

- A matrix is called **sparse** if it has many zeros and its computation and/or storage can take advantage of such property.
- Instead of storing the content in a two dimensional array, sparse matrices usually only store non-zero elements using some special data structures, such as CRS (Compressed Row Storage).

$$A = \begin{pmatrix} 7.5 & 2.9 & 2.8 & 2.7 & 0 & 0 \\ 6.8 & 5.7 & 3.8 & 0 & 0 & 0 \\ 2.4 & 6.2 & 3.2 & 0 & 0 & 0 \\ 9.7 & 0 & 0 & 2.3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 5.8 & 5.0 \\ 0 & 0 & 0 & 0 & 6.6 & 8.1 \end{pmatrix}$$



Example

Let $f(\vec{x}) = x_1^2 + 2x_2^2 + 0.5x_3^2 + 1.5x_4^2 - x_1x_3$, $H = \begin{bmatrix} 2 & 0 & -1 & 0 \\ 0 & 4 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 3 \end{bmatrix}$

- Suppose $\vec{x}_0 = (1, 1, 1, 1)$ and $B_0 = I$.
- $\vec{p}_0 = -B_0^{-1}\nabla f(\vec{x}_0) = \begin{bmatrix} -1 \\ -4 \\ 0 \\ -3 \end{bmatrix}$ $\left(\nabla f(\vec{x}) = \begin{bmatrix} 2x_1 - x_3 \\ 4x_2 \\ x_3 - x_1 \\ 3x_4 \end{bmatrix} \right)$
- Base on the line search method and set $\bar{\alpha} = 1$, $\rho = 0.5$, $c = 0.5$, then we will get $\alpha_0 = 0.25$.
- $\vec{x}_1 = \vec{x}_0 + \alpha_0\vec{p}_0 = \begin{bmatrix} 0.75 \\ 0 \\ 1 \\ 0.25 \end{bmatrix}$, $\vec{s}_1 = \vec{x}_1 - \vec{x}_0 = \begin{bmatrix} -0.25 \\ -1 \\ 0 \\ -0.75 \end{bmatrix}$

Example: continue

- $\vec{y}_1 = \nabla f(\vec{x}_1) - \nabla f(\vec{x}_0) = \begin{bmatrix} -0.5 \\ -4 \\ 0.25 \\ -2.25 \end{bmatrix}$
- Since B_1 must be symmetric and satisfy the same sparse pattern as Hessian of $f(\vec{x})$, $B_1 = \begin{bmatrix} a & 0 & e & 0 \\ 0 & b & 0 & 0 \\ e & 0 & c & 0 \\ 0 & 0 & 0 & d \end{bmatrix}$.
- To satisfy the secant condition $B_1\vec{s}_1 = \vec{y}_1$:

$$\Rightarrow \begin{bmatrix} a & 0 & e & 0 \\ 0 & b & 0 & 0 \\ e & 0 & c & 0 \\ 0 & 0 & 0 & d \end{bmatrix} \begin{bmatrix} -0.25 \\ -1 \\ 0 \\ -0.75 \end{bmatrix} = \begin{bmatrix} -0.5 \\ -4 \\ 0.25 \\ -2.25 \end{bmatrix} \Rightarrow \begin{cases} a = 2 \\ b = 4 \\ d = 3 \\ e = -1 \end{cases}$$

Example: continue

- Since we wish to minimize $\|B_1 - B_0\|_F^2$, c should be 1.

$$\Rightarrow B_1 = \begin{bmatrix} 2 & 0 & -1 & 0 \\ 0 & 4 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 3 \end{bmatrix} \text{ and } B_1^{-1} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1/4 & 0 & 0 \\ 1 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1/3 \end{bmatrix}$$

- $\vec{p}_1 = -B_1^{-1}\nabla f(\vec{x}_1) = \begin{bmatrix} -0.75 \\ 0 \\ -1 \\ -0.25 \end{bmatrix}$, and we pick $\alpha_1 = 1$ with the same algorithm.
- $\vec{x}_2 = \vec{x}_1 + \alpha_1 \vec{p}_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$, which is the optimal solution of $f(\vec{x})$.

Problems of sparse quasi-Newton updates

- However, there is no simple way like SR1 or BFGS to solve the updates of sparse approximation matrix B_{k+1}.
- Also, the practical performance of this kind of updates is not good in large-scale problems since it produces inadequate model and poor Hessian approximations.
- Another way to update B is to solve

$$B_{k+1} = \min_{B} \|BS_{k+1} - Y_{k+1}\|_F^2$$

$$\text{subject to } B_{k+1} = B_{k+1}^T, \text{ and } (B_{k+1})_{ij} = 0 \text{ for } (i,j) \notin \Omega$$

where

$$S_k = [\vec{s}_{k-m}, \ldots, \vec{s}_{k-1}], Y_k = [\vec{y}_{k-m}, \ldots, \vec{y}_{k-1}]$$

- Even though the new update method frequently outperforms the original one, it is still hard to compute and may produce singular or poor Hessian approximations.