02610
Optimization and Data Fitting

Week 11: Conjugate Gradient Methods & Large-Scale Unconstrained Optimization

Yiqiu Dong

DTU Compute
Technical University of Denmark

Unconstrained quadratic problems

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T A\boldsymbol{x} - \boldsymbol{b}^T\boldsymbol{x}$$

f'(x) = Ax - b
Optimizer: Ax = b

where A is an $n \times n$ symmetric positive definite matrix.

- It is equivalent to solve the linear system of equations: $A\boldsymbol{x} = \boldsymbol{b}$.
- The **residual** $\boldsymbol{r} = \boldsymbol{b} - A\boldsymbol{x}$ is the negative gradient: $\boldsymbol{r} = -\nabla f(\boldsymbol{x})$.

1: A = c*I (constant * identity matrix)

- **Steepest descent method:** 1 or ∞ iterations.
- **Newton's method:** 1 iteration. Newton uses quadratic function to approximate object function. f(x) here is already quadratic, so this is exact, and therefore only 1 iteration
- **Coordinate search method:** n or ∞ iterations. n iterations for n-dimensional problem, and A is diagonal
- **Conjugate gradient method:** n iterations.

Conjugate gradient (CG) method

- It was proposed by Hestenes and Stiefel in the 1950s.
- It is the most widely used iterative method for solving $A\boldsymbol{x} = \boldsymbol{b}$ with $A \succ 0$
- It was extended to solve nonlinear unconstrained minimization problems in 1960s.

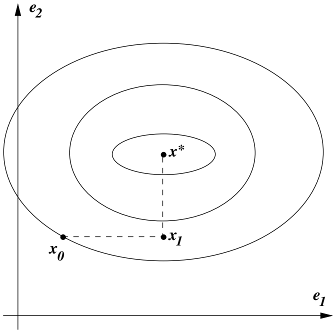
Main advantages of CG:

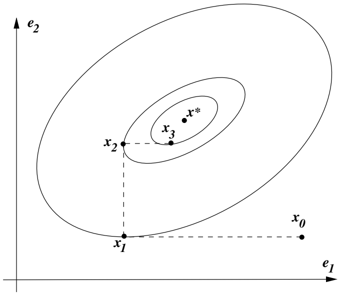
- It takes at most n iterations to the solution (theoretically).
- It does not alter A.
- At each iteration, it only need one computation of the matrix-vector product ($O(n^2)$) and a few vector product and sum ($O(n)$).
- For storage, it only need store a few vectors.
- CG is only used for solving large-scale problems.
- CG is proved with linear convergence rate, but generally much faster than the steepest descent method.

Conjugate directions

Idea:

- If A is diagonal, then the coordinate search method can find the minimizer of $f(\boldsymbol{x})$ in n iterations.





- If A is **NOT** diagonal, we can diagonalize A, that is, accordingly transform the coordinate directions.

Conjugate directions

Suppose that a $n \times n$ matrix $S = [\boldsymbol{p}_0, \boldsymbol{p}_1, \cdots, \boldsymbol{p}_{n-1}]$ diagonalizes A, i.e., $S^T A S$ is diagonal. Then, we have

$$\boldsymbol{p}_i^T A \boldsymbol{p}_j = 0, \qquad \text{for all } i \neq j,$$

and we call $\{\boldsymbol{p}_0, \cdots, \boldsymbol{p}_{n-1}\}$ to be **conjugate** with respect to spd. A.

- $\{\boldsymbol{p}_0, \cdots, \boldsymbol{p}_{n-1}\}$ are conjugate, if and only if they are orthogonal for the inner product $\langle \boldsymbol{u}, \boldsymbol{v} \rangle_A = \boldsymbol{u}^T A \boldsymbol{v}$. = 0
- If $\boldsymbol{p}_i \neq \boldsymbol{0}$ for all i, they are also linearly independent.

If $\{\boldsymbol{p}_0, \cdots, \boldsymbol{p}_{n-1}\}$ are **conjugate** (*conjugate directions*) and $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$, then the exact line search has a closed-form and gives

$$\alpha_k = \frac{\boldsymbol{r}_k^T \boldsymbol{p}_k}{\boldsymbol{p}_k^T A \boldsymbol{p}_k}.$$

Conjugate direction methods

initial guess for conjugate direction:
- eigenvalues
- choose linearly independent basis

They are very expensive, so NOT USED

Instead use Conjugate gradient directions (slide8)

Algorithm

```
Given $\boldsymbol{x}_0$ and a set of conjugate directions $\{\boldsymbol{p}_0, \cdots, \boldsymbol{p}_{n-1}\}$.
loop
    Compute $\alpha_k = \frac{\boldsymbol{r}_k^T \boldsymbol{p}_k}{\boldsymbol{p}_k^T A \boldsymbol{p}_k}$;
    Update $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$;
end loop
```

use exact line search to find alpha

Theorem

For any $\boldsymbol{x}_0 \in \mathbb{R}^n$ the sequence $\{\boldsymbol{x}_n\}$ generated by the above conjugate direction method converges to the solution $\boldsymbol{x}^*$ of the linear system $A\boldsymbol{x} = \boldsymbol{b}$ in at most n iterations.

Conjugate direction methods

Expanding subspace minimization

Let $\boldsymbol{x}_0 \in \mathbb{R}^n$ be any starting point and the sequence $\{\boldsymbol{x}_k\}$ be generated by the conjugate direction method shown in the previous page. Then,

- $\boldsymbol{r}_k^T \boldsymbol{p}_i = 0$, for $i = 0, 1, \cdots, k-1$;
- $\boldsymbol{x}_k$ is the minimizer of $f(\boldsymbol{x})$ over the set $\{\boldsymbol{x} | \boldsymbol{x} = \boldsymbol{x}_0 + \mathrm{span}\{\boldsymbol{p}_0, \cdots, \boldsymbol{p}_{k-1}\}\}$.

- The current residual $\boldsymbol{r}_k$ is orthogonal to all previous search directions.
- The conjugate direction method minimizes $f(\boldsymbol{x})$ along one conjugate direction at one iteration.

Conjugate gradient directions

- $\boldsymbol{p}_k$ is generated by using only the previous vector $\boldsymbol{p}_{k-1}$.
- $\boldsymbol{p}_k$ is automatically conjugate to $\{\boldsymbol{p}_0, \cdots, \boldsymbol{p}_{k-1}\}$.

r0 = negative gradient of f

Recursion for $\boldsymbol{p}_k$: We start with $\boldsymbol{p}_0 = \boldsymbol{r}_0$ and choose $\boldsymbol{p}_k$ to be a linear combination of the residual $\boldsymbol{r}_k$ and the previous direction $\boldsymbol{p}_{k-1}$:

$$\boldsymbol{p}_k = \boldsymbol{r}_k + \beta_k \boldsymbol{p}_{k-1}.$$

Since $\boldsymbol{p}_k$ is conjugate to $\boldsymbol{p}_{k-1}$ w.r.t. A, then we have

$$\beta_k = -\frac{\boldsymbol{p}_{k-1}^T A \boldsymbol{r}_k}{\boldsymbol{p}_{k-1}^T A \boldsymbol{p}_{k-1}}.$$

Conjugate gradient method (preliminary version)

Algorithm

```
Given $\boldsymbol{x}_0$;
Set $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$, $\boldsymbol{p}_0 = \boldsymbol{r}_0$;
**loop**
    Compute $\alpha_k = \frac{\boldsymbol{r}_k^T \boldsymbol{p}_k}{\boldsymbol{p}_k^T A \boldsymbol{p}_k}$;
    Update $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$;
    Compute $\boldsymbol{r}_{k+1} = \boldsymbol{b} - A\boldsymbol{x}_{k+1}$;
    Compute $\beta_{k+1} = -\frac{\boldsymbol{p}_k^T A \boldsymbol{r}_{k+1}}{\boldsymbol{p}_k^T A \boldsymbol{p}_k}$;
    Compute $\boldsymbol{p}_{k+1} = \boldsymbol{r}_{k+1} + \beta_{k+1} \boldsymbol{p}_k$;
    Check for convergence;
**end loop**
Output $\boldsymbol{x}_{k+1}$.
```

Properties of CG method

Theorem

Suppose that the kth iterate of the CG method is not the solution $\boldsymbol{x}^*$. Then,

1. $\boldsymbol{r}_k^T \boldsymbol{r}_i = 0, \qquad \text{for } i = 0, 1, \cdots, k-1,$
2. $\text{span}\{\boldsymbol{r}_0, \boldsymbol{r}_1, \cdots, \boldsymbol{r}_k\} = \text{span}\{\boldsymbol{p}_0, \boldsymbol{p}_1, \cdots, \boldsymbol{p}_k\} = \text{span}\{\boldsymbol{r}_0, A\boldsymbol{r}_0, \cdots, A^k\boldsymbol{r}_0\},$
3. $\boldsymbol{p}_k^T A \boldsymbol{p}_i = 0$, for $i = 0, 1, \cdots, k-1$.

Therefore, the sequence $\{\boldsymbol{x}_k\}$ converges to $\boldsymbol{x}^*$ in at most n steps.

- The proof of this theorem relies on the fact that $\boldsymbol{p}_0 = \boldsymbol{r}_0$ (the steepest descent direction).
- The result (1) shows that the residuals/gradients at all iterates are orthogonal to each other.
- The result (3) shows that $\{\boldsymbol{p}_0, \cdots, \boldsymbol{p}_k\}$ are conjugate directions.
- The result (2) shows that the search directions and the residuals from CG method generate the Krylov subspaces.

Krylov subspaces

Definition: A sequence of subspaces generated by a matrix A and a vector $\boldsymbol{b}$:

$$\mathcal{K}_k(A, \boldsymbol{b}) = \mathrm{span}\{\boldsymbol{b}, A\boldsymbol{b}, \cdots, A^{k-1}\boldsymbol{b}\} \quad \text{for } k \geq 1.$$

Properties:

- $\mathcal{K}_k(A, \boldsymbol{r}_0) = \mathrm{span}\{\boldsymbol{r}_0, A\boldsymbol{r}_0, \cdots, A^{k-1}\boldsymbol{r}_0\}$.
- The Krylov subspaces are nested: $\mathcal{K}_1 \subseteq \mathcal{K}_2 \subseteq \mathcal{K}_3 \subseteq \cdots$
- The dimensions of the Krylov subspaces increase by at most one: $\dim\mathcal{K}_{k+1} - \dim\mathcal{K}_k$ is zero or one.
- If $\mathcal{K}_{k+1} = \mathcal{K}_k$, then $\mathcal{K}_i = \mathcal{K}_k$ for all $i \geq k$:

$$\begin{aligned} & A^k\boldsymbol{b} \in \mathrm{span}\{\boldsymbol{b}, A\boldsymbol{b}, \cdots, A^{k-1}\boldsymbol{b}\} \\ \Longrightarrow \quad & A^i\boldsymbol{b} \in \mathrm{span}\{\boldsymbol{b}, A\boldsymbol{b}, \cdots, A^{k-1}\boldsymbol{b}\} \quad \text{for } i > k. \end{aligned}$$

Simplified CG method

- Using $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$ and $\boldsymbol{r}_{k+1} = \boldsymbol{b} - A\boldsymbol{x}_{k+1}$, we obtain

$$\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k A\boldsymbol{p}_k. \tag{1}$$

- Using $\boldsymbol{p}_k = \boldsymbol{r}_k + \beta_k \boldsymbol{p}_{k-1}$ and $\boldsymbol{r}_k^T \boldsymbol{p}_{k-1} = 0$, we obtain $\boldsymbol{r}_k^T \boldsymbol{p}_k = \boldsymbol{r}_k^T \boldsymbol{r}_k$, then

$$\alpha_k = \frac{\boldsymbol{r}_k^T \boldsymbol{p}_k}{\boldsymbol{p}_k^T A\boldsymbol{p}_k} = \frac{\|\boldsymbol{r}_k\|_2^2}{\boldsymbol{p}_k^T A\boldsymbol{p}_k}. \tag{2}$$

- Using (1), (2) and $\boldsymbol{r}_{k+1}^T \boldsymbol{r}_k = 0$, we obtain

$$\beta_{k+1} = -\frac{\boldsymbol{p}_k^T A\boldsymbol{r}_{k+1}}{\boldsymbol{p}_k^T A\boldsymbol{p}_k} = \frac{\|\boldsymbol{r}_{k+1}\|_2^2}{\|\boldsymbol{r}_k\|_2^2}. \tag{3}$$

Conjugate gradient method

Algorithm

```
Given $\boldsymbol{x}_0$;
Set $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$, $\boldsymbol{p}_0 = \boldsymbol{r}_0$;
loop
    Compute $\alpha_k = \frac{\|\boldsymbol{r}_k\|_2^2}{\boldsymbol{p}_k^T A \boldsymbol{p}_k}$;
    Update $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$;
    Compute $\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k A\boldsymbol{p}_k$;
    Compute $\beta_{k+1} = \frac{\|\boldsymbol{r}_{k+1}\|_2^2}{\|\boldsymbol{r}_k\|_2^2}$;
    Compute $\boldsymbol{p}_{k+1} = \boldsymbol{r}_{k+1} + \beta_{k+1}\boldsymbol{p}_k$;
    Check for convergence;
end loop
Output $\boldsymbol{x}_{k+1}$.
```

Main computation per iteration is matrix-vector product $A\boldsymbol{p}_k$.

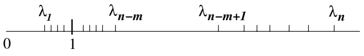
Rate of convergence

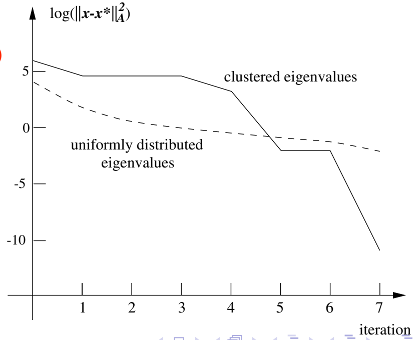
- If A has only r distinct eigenvalues, then the CG method will terminate at the solution in at most r iterations.
- If A has eigenvalues $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$, we have that

$$\|\boldsymbol{x}_{k+1} - \boldsymbol{x}^*\|_A^2 \leq \left(\frac{\lambda_{n-k} - \lambda_1}{\lambda_{n-k} + \lambda_1} \right)^2 \|\boldsymbol{x}_0 - \boldsymbol{x}^*\|_A^2.$$

Example: We apply the CG method to solve $A\boldsymbol{x} = \boldsymbol{b}$.

if frac close to 1: sub-linear convergence (slower)
if frac close to 0: super-linear convergence (faster)





Preconditioning

- **Idea:** Make change of variables $\hat{\boldsymbol{x}} = C\boldsymbol{x}$ with C nonsingular, and apply CG to

  -T = inverse and then tranpose

  $$C^{-T}AC^{-1}\hat{\boldsymbol{x}} = C^{-T}\boldsymbol{b}.$$
- The spectrum of the new matrix $C^{-T}AC^{-1}$ should be clustered, then PCG converges fast.
- We need consider the trade-off between enhanced convergence and cost of extra computation.
- The matrix $M = C^TC$ is called the preconditioner.
- Python implementation:
  `scipy.optimize.minimize(method='CG')`
- Matlab implementation: `pcg`

Example:

- diagonal $C = \mathrm{diag}(\sqrt{a_{11}}, \sqrt{a_{22}}, \cdots, \sqrt{a_{nn}})$
- incomplete or approximate Cholesky factorization of A
- Good preconditioners are often application-dependent.

Nonlinear conjugate gradient method

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} f(\boldsymbol{x}), \qquad f \text{ is convex and differentiable.}$$

Nonlinear CG methods

- Extend linear CG method to nonquadratic functions.
- Limited global convergence theory.

Modifications needed to extend linear CG method

- Replace $\boldsymbol{r}_k = \boldsymbol{b} - A\boldsymbol{x}_k$ with $-\nabla f(\boldsymbol{x}_k)$.
- Determine the step length α by line search.

Fletcher-Reeves method

Algorithm

```
Given $\boldsymbol{x}_0$;
Compute $f_0 = f(\boldsymbol{x}_0)$ and $\nabla f_0 = \nabla f(\boldsymbol{x}_0)$;
Set $\boldsymbol{p}_0 = -\nabla f_0$;
loop
    Compute $\alpha_k$ by line search method;
    Update $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$;
    Evaluate $\nabla f_{k+1}$;
    Compute $\beta_{k+1}^{FR} = \frac{\|\nabla f_{k+1}\|_2^2}{\|\nabla f_k\|_2^2}$;
    Compute $\boldsymbol{p}_{k+1} = -\nabla f_{k+1} + \beta_{k+1}^{FR} \boldsymbol{p}_k$;
    Check for convergence;
end loop
Output $\boldsymbol{x}_{k+1}$.
```

Some observations

Interpretation

- First iteration is a steepest descent step.
- General update is a steepest descent step with momentum term

$$\boldsymbol{x}_{x+1} = \boldsymbol{x}_k - \alpha_k \nabla f_k + \frac{\alpha_k \beta_k}{\alpha_{k-1}}(\boldsymbol{x}_k - \boldsymbol{x}_{k-1}).$$

- It is common to restart the algorithm every n iterations by taking a steepest descent step to periodically refresh the algorithm.

Line search

- With exact line search, it reduces to linear CG for quadratic f.
- Exact line search in computation of α_k implies that α_k is a local minimizer along $\boldsymbol{p}_k$, i.e., $\nabla f_{k+1}^T \boldsymbol{p}_k = 0$. Therefore, $\boldsymbol{p}_{k+1}$ is a descent direction at $\boldsymbol{x}_{k+1}$:

$$\nabla f_{k+1}^T \boldsymbol{p}_{k+1} = -\|\nabla f_{k+1}\|^2 + \beta_{k+1}^{FR} \nabla f_{k+1}^T \boldsymbol{p}_k = -\|\nabla f_{k+1}\|^2 < 0.$$

- For inexact line search, if α_k satisfies the strong Wolfe conditions, then $\boldsymbol{p}_{k+1}$ is descent.

Variations

Polak-Ribière method: Compute β_{k+1} from

$$\beta_{k+1} = \frac{\nabla f_{k+1}^T (\nabla f_{k+1} - \nabla f_k)}{\|\nabla f_k\|_2^2}.$$

Hestenes-Stiefel method: Compute β_{k+1} from

$$\beta_{k+1} = \frac{\nabla f_{k+1}^T (\nabla f_{k+1} - \nabla f_k)}{(\nabla f_{k+1} - \nabla f_k)^T \boldsymbol{p}_k}.$$

- All these formulas are equivalent for quadratic f and exact line search.
- With restarts and the strong Wolfe conditions, all three methods have global convergence.
- Without restarts, FR has global convergence with the strong Wolfe conditions, but PR not.
- In practice, PR is more robust and efficient than FR.

Large-scale unconstrained optimization

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} f(\boldsymbol{x}), \qquad f \in \mathcal{C}^2(\mathbb{R}^n)$$

- Large-scale problems (today): $10^3 \sim 10^6$ variables.
- When solving large-scale problems, we have to take the storage and computational costs of the optimization algorithm into account.
- In large problems, the following can have a prohibitive cost:
  - computing the Hessian or multiplying it
  - factorizing the Hessian (solving for the Newton step)
  - storing a dense approximate Hessian like in quasi-Newton methods
- Linear/nonlinear conjugate gradient methods can be applied directly to large-scale problems without modification, but not fast.

Inexact Newton methods

Ideas: Use some inexpensive iterative algorithm to *very approximately* solve either

$$\nabla^2 f_k \boldsymbol{p}_k = -\nabla f_k \qquad \text{(line search)}$$

or

$$\begin{aligned} &\min_{\boldsymbol{p}\in\mathbb{R}^n} m_k(\boldsymbol{p}) = f_k + \nabla f_k^T \boldsymbol{p} + \frac{1}{2}\boldsymbol{p}^T \nabla^2 f_k \boldsymbol{p}, \\ &\text{s. t. } \|\boldsymbol{p}\|_2 \le \Delta_k, \qquad \text{(trust region)} \end{aligned}$$

without ruining global and fast local convergence of exact LS/TR Newton methods.

Stopping criterion for iterative solver:

$$\|\boldsymbol{r}_k\|_2^2 = \|\nabla^2 f_k \boldsymbol{p}_k + \nabla f_k\|_2^2 \le \eta_k \|\nabla f_k\|_2^2,$$

where the sequence $\{\eta_k\}$ with $0 \le \eta_k \le 1$ for all k is called the **forcing sequence**.

Local convergence

Convergence theorem

Inexact Newton with unit steps:

- $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{p}_k$
- $\|\boldsymbol{r}_k\|_2^2 \le \eta_k \|\nabla f_k\|_2^2$
- $0 < \eta_k \le \eta < 1$

Then, if the starting point $\boldsymbol{x}_0$ is sufficiently near $\boldsymbol{x}^*$,

- the sequence $\{\boldsymbol{x}_k\}$ converges to $\boldsymbol{x}^*$,
- and
$$\|\nabla^2 f(\boldsymbol{x}^*)(\boldsymbol{x}_{k+1} - \boldsymbol{x}^*)\|_2 \le \hat{\eta}\|\nabla^2 f(\boldsymbol{x}^*)(\boldsymbol{x}_k - \boldsymbol{x}^*)\|_2$$
for some constant $\hat{\eta}$ with $\eta < \hat{\eta} < 1$ (linear convergence).

Local convergence

Convergence rate

Inexact Newton with unit steps:

- $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{p}_k$
- $\|\boldsymbol{r}_k\|_2^2 \leq \eta_k \|\nabla f_k\|_2^2$
- $0 < \eta_k \leq \eta < 1$

Then,

- if $\eta_k \to 0$, the sequence $\{\boldsymbol{x}_k\}$ converges to $\boldsymbol{x}^*$ superlinearly;
- if $\nabla^2 f(\boldsymbol{x})$ is Lipschitz continuous for $\boldsymbol{x}$ near $\boldsymbol{x}^*$ and $\eta_k = O(\|\nabla f_k\|_2)$, then the convergence is quadratic.

Example:

- $\eta_k = \min(0.5, \sqrt{\|\nabla f_k\|_2})$ would yield superlinear convergence;
- $\eta_k = \min(0.5, \|\nabla f_k\|_2)$ would yield quadratic convergence.

Line search Newton-CG method

Algorithm

```
Given $\boldsymbol{x}_0$;
loop
    Define the forcing sequence $\eta_k = \min(0.5, \sqrt{\|\nabla f_k\|_2})$
    Use CG to solve $\nabla^2 f_k \boldsymbol{p}_k = -\nabla f_k$ approximately with accuracy $\eta_k$
    Compute $\alpha_k$ by line search method;
    Update $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$;
end loop
```

Remark: $\nabla^2 f_k$ is not necessarily positive definite, but the CG method is designed to solve positive definite systems. So we need to modify CG loop:

- If $\boldsymbol{d}_j^T \nabla^2 f_k \boldsymbol{d}_j \leq 0$, where $\boldsymbol{d}_j$ is current conjugate direction,
  - if $j = 0$, then we stop CG and return steepest descent direction: $\boldsymbol{p}_k = \boldsymbol{d}_0$;
  - otherwise, we stop CG and return the current iterate in CG: $\boldsymbol{p}_k = \boldsymbol{z}_j$.

Line search Newton-CG method

- Inner CG loop always produces a descent direction for f.
- When the Hessian $\nabla^2 f_k$ is nearly singular, the line search Newton-CG direction can take long and of poor quality.
- It does not require explicit knowledge of the Hessian, and it requires only the Hessian-vector products. Finite differencing and automatic differentiation techniques can be used.
- Preconditioning can be introduced to speed up CG.
- Python implementation:
  `scipy.optimize.minimize(method='Newton-CG')`

Limited-memory quasi-Newton methods

Idea: They save only a few vectors that represent the approximation of the Hessian implicitly.

- Useful for solving large problems with costly or nonsparse Hessian.
- Linear convergence but fast rate.

Limited-memory BFGS (L-BFGS):

- It uses curvature information from only the most recent m iterations to construct the Hessian approximation.
- Modest values of m ($\sim 3 - 20$) work fine in practice, but the best m depends on the problem.
- Slow convergence in ill-conditioned problems.

L-BFGS update

Review: BFGS inverse Hessian update:

$$H_{k+1} = V_k^T H_k V_k + \rho_k \boldsymbol{s}_k \boldsymbol{s}_k^T$$

where $V_k = I - \rho_k \boldsymbol{s}_k \boldsymbol{y}_k^T$, $\rho_k = 1/(\boldsymbol{y}_k^T \boldsymbol{s}_k)$, $\boldsymbol{s}_k = \boldsymbol{x}_{k+1} - \boldsymbol{x}_k$ and $\boldsymbol{y}_k = \nabla f_{k+1} - \nabla f_k$.

- Since H_k is generally dense, the cost of storing and manipulating it is prohibitive when n is large.
- We store a modified version of H_{k+1} implicitly, by storing $m \ll n$ of the vector pairs $\{\boldsymbol{s}_k, \boldsymbol{y}_k\}$.
- The product $H_{k+1}\nabla f_{k+1}$ can be obtained by performing a sequence of inner products and vector summations.
- After the new iterate is computed, we replace the oldest pair with the new pair.

L-BFGS update

Update algorithm: Compute $H_k \nabla f_k$

```
Given $H_k^0$; Set $\boldsymbol{q} = \nabla f_k$;
for $i = k-1, k-2, \cdots, k-m$ do
    $\alpha_i = \rho_i \boldsymbol{s}_i^T \boldsymbol{q}$;
    $\boldsymbol{q} = \boldsymbol{q} - \alpha_i \boldsymbol{y}_i$;
end for
$\boldsymbol{r} = H_k^0 \boldsymbol{q}$;
for $i = k-m, k-m+1, \cdots, k-1$ do
    $\beta = \rho_i \boldsymbol{y}_i^T \boldsymbol{r}$;
    $\boldsymbol{r} = \boldsymbol{r} + \boldsymbol{s}_i(\alpha_i - \beta)$;
end for
Output $\boldsymbol{r}$.
```

- It recursively expands the update with m pairs $\{\boldsymbol{s}_k, \boldsymbol{y}_k\}$.
- H_k^0 is allowed to vary from iteration to iteration.
- It requires $4mn$ multiplications and calculation of $H_k^0 \boldsymbol{q}$.

L-BFGS method

```
Given $\boldsymbol{x}_0$ and $m$;
loop
    Choose $H_k^0$;
    Compute $\boldsymbol{p}_k = -H_k \nabla f_k$ by update algorithm;
    Compute $\alpha_k$ by line search method;
    Update $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$;
    if $k > m$ then
        Discard $\{\boldsymbol{s}_{k-m}, \boldsymbol{y}_{k-m}\}$ from storage;
    end if
    Store $\boldsymbol{s}_k = \boldsymbol{x}_{k+1} - \boldsymbol{x}_k$ and $\boldsymbol{y}_k = \nabla f_{k+1} - \nabla f_k$;
end loop
```

- A good choice for H_k^0 in practice: $H_k^0 = \gamma_k I$ with $\gamma_k = (\boldsymbol{s}_{k-1}^T \boldsymbol{y}_{k-1})/(\boldsymbol{y}_{k-1}^T \boldsymbol{y}_{k-1})$.
- The line search based on the (strong) Wolfe conditions makes BFGS stable.
- The first $m - 1$ iterates are the same as in BFGS.

Relationship with CG methods

- Limited-memory methods historically evolved as improvements of nonlinear CG methods.
- The Hestenes-Stiefel form of nonlinear CG method:

$$\boldsymbol{p}_{k+1} = -\nabla f_{k+1} + \frac{\nabla f_{k+1}^T \boldsymbol{y}_k}{\boldsymbol{y}_k^T \boldsymbol{p}_k} \boldsymbol{p}_k = -\hat{H}_{k+1} \nabla f_{k+1} \quad \text{with } \hat{H}_{k+1} = I - \frac{\boldsymbol{s}_k \boldsymbol{y}_k^T}{\boldsymbol{y}_k^T \boldsymbol{s}_k},$$

  which resembles quasi-Newton iterates, but $\hat{H}_{k+1}$ is neither symmetric nor positive definite.
- A symmetric positive definite modification, which also satisfies the secant equation, is

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

  which is exactly the L-BGFS method with $m = 1$ and $H_k^0 = I$ (*memoryless BFGS*).

Final evaluation

Final evaluation
in DTUinside
from 18. Nov. to 29. Nov.