Unconstrained Optimization II

Asst. Prof. Dr.-Ing. Sudchai Boonto
October 1, 2024

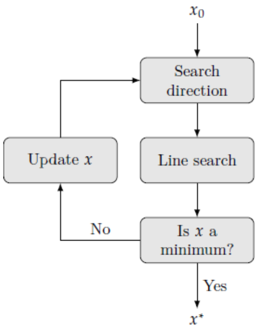
Department of Control System and Instrumentation Engineering
King Mongkut's Unniversity of Technology Thonburi
Thailand

Objective

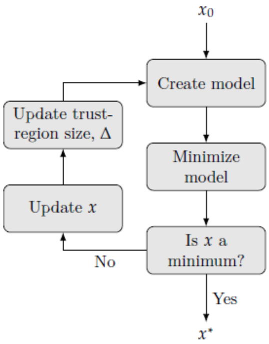
At the end of this chapter you should be able to:

- Describe, implement, and use line-search-based methods.
- Explain the pros and cons of the various search direction methods.
- Understand steepest descent, conjugate gradient, etc.

Two Approaches to Finding and Optimum



Line search approach



Trust-region approach

Basic Concept

Consider a problem

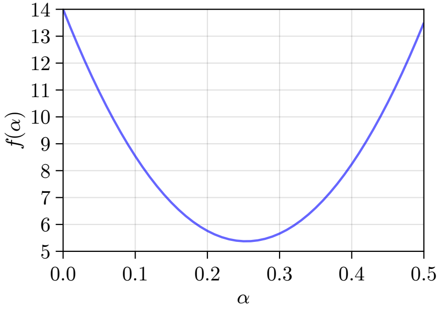
$$\underset{\mathbf{x}}{\text{minimize}} \quad f(\mathbf{x}), \quad \mathbf{x} \in \mathbb{R}^n$$

- Most numerical methods require a starting design or point which we call $\mathbf{x}_0$.
- We then determine the *direction of travel* $\mathbf{d}_0$.
- A *step size* α_0 is then determined based on minimizing f as much as possible and the design point is updated as $\mathbf{x}_1 = \mathbf{x}_0 + \alpha_0 \mathbf{d}_0$.
- The process of where to go and how far to go are repeated from $\mathbf{x}_1$ or $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k$.

Basic Concept: Example

Given $f(x_1, x_2) = x_1^2 + 5x_2^2$, a point $\mathbf{x}_0 = [3\ 1]^T$, $f_0 = f(\mathbf{x}_0) = 14$.

1. construct $f(\alpha)$ along the direction $\mathbf{d} = [-3\ \ -5]^T$ and provide a plot of $f(\alpha)$ versus α, for $\alpha \geq 0$.
   We have $\mathbf{x}(\alpha) = \mathbf{x}_0 + \alpha\mathbf{d} = [3 - 3\alpha,\ 1 - 5\alpha]^T$ and
   $f(\alpha) = (3 - 3\alpha)^2 + 5(1 - 5\alpha)^2$.



Basic Concept : Example

2. Find the slope $df(\alpha)/d\alpha)$ at $\alpha = 0$. Verify that this equal $\nabla f(\mathbf{x}_0)^T \mathbf{d}$.
   We have

$$\left.\frac{df(\alpha)}{d\alpha}\right|_{\alpha=0} = (-6(3 - 3\alpha) - 50(1 - 5\alpha))|_{\alpha=0} = -68$$

$$\nabla f(\mathbf{x}_0)^T \mathbf{d} = \begin{bmatrix} 2(3) & 10(1) \end{bmatrix} \begin{bmatrix} -3 \\ -5 \end{bmatrix} = -68$$

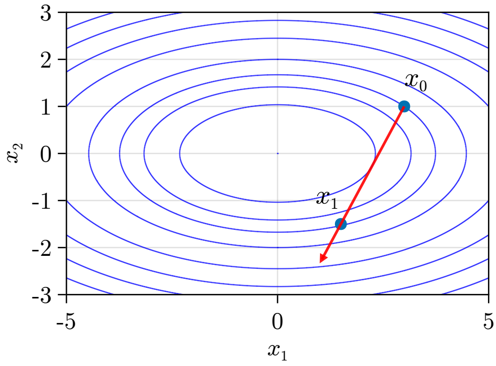
3. Minimize $f(\alpha)$ with respect to α, to obtain step size α_0. Given the corresponding new point $\mathbf{x}_1$ and value of $f_1 = f(\mathbf{x}_1)$
   We have

$$\frac{df(\alpha)}{d\alpha} = -6(3 - 3\alpha) - 50(1 - 5\alpha) = 0 \Longrightarrow 268\alpha = 68 \text{ or } \alpha = 0.2537$$

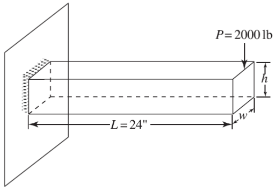
$$\mathbf{x}_1 = \begin{bmatrix} 3 \\ 1 \end{bmatrix} + \alpha_0 \begin{bmatrix} -3 \\ -5 \end{bmatrix} = \begin{bmatrix} 2.2388 \\ -0.2687 \end{bmatrix}, \quad f(\mathbf{x}_1) = 5.3732, \text{ less than } f_0 = 14.$$

Basic Concept : Example

2. Provide a plot showing contours of the function, steepest descent direction x_0 and x_1.



Basic Concept : Example



We want to design the width and height of the rectangular cross-section to increase the bending stress defined by

$$\sigma_0 = \frac{6M}{wh^2}, \text{ where } M \text{ is a moment.}$$

With the initial design $\mathbf{x} = (w, h) = (1, 3)$, we have

$$\sigma_0 = \frac{6(2000 \times 24)}{1(3^2)} = 32,000\text{psi}$$

Using $\mathbf{d} = [-1/\sqrt{5} \;\; -2/\sqrt{5}]^T$ and $\alpha = 0.2$ we have

$$\mathbf{x}_1 = \mathbf{x}_0 + \alpha \mathbf{d} = \begin{bmatrix} 1 \\ 3 \end{bmatrix} + 0.2 \begin{bmatrix} -1/\sqrt{5} \\ -2/\sqrt{5} \end{bmatrix} = \begin{bmatrix} 0.9106 \\ 2.8211 \end{bmatrix}, \qquad \sigma_1 = 71,342 \text{ psi}$$

Line Search : Exact Line Search

Assume we have chosen a descent direction $\mathbf{d}$. We need to choose the step factor α to obtain our next design point. One approach is to use *line search*, which selects the step factor that minimizes the one-dimensional function:

$$\underset{\alpha}{\text{minimize}} \quad f(\mathbf{x} + \alpha\mathbf{d})$$

To inform the search, we can use the derivative of the line search objective, which is simply the directional derivative along $\mathbf{d}$ at $\mathbf{x} + \alpha\mathbf{d}$.

```
function LINE_SEARCH($f, \mathbf{d}$)
    objective = $\alpha \to f(\mathbf{x} + \alpha * \mathbf{d})$
    $a, b$ = brackect_minimum(objective)
    $\alpha$ = minimize(objective, a, b)
    return $\mathbf{x} + \alpha * \mathbf{d}$
end function
```

The exact line search is expensive, if we need to do it every step of the optimization. In Matlab environment, we can use commands `fminbnd` or `fminsearch`.

Line Search : Exact Line Search

- One disadvantage of conducting a line search at each step is the computational cost of optimizing α to a high degree of precision.
- We could quickly find a reasonable value and then move on, selecting $\mathbf{x}_{k+1}$, and then picking a new direction $\mathbf{d}_{k+1}$.
- Some algorithms use a fixed step factor. Large steps will tend to result in fast convergence but risk overshooting the minimum.
- Smaller steps tend to be more stable but can result in slower convergence.
- A fixed step factor α is sometimes referred to as a **learning rate**.
- Another method is to use a **decaying step factor**:

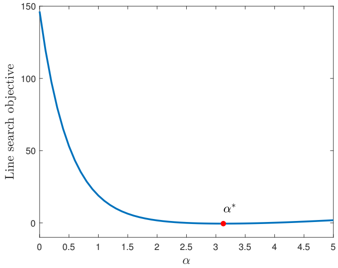
$$\alpha_k = \alpha_1 \gamma_{k-1} \text{ for } \gamma \in (0, 1]$$

The decaying step factors are popular when minimizing noisy objective function, and always used in machine learning applications.

Line Search : Exact Line Search

Consider conducting a line search on $f(x_1, x_2, x_3) = \sin(x_1 x_2) + e^{(x_2+x_3)} - x_3$ from $\mathbf{x} = [1, 2, 3]$ in the direction $\mathbf{d} = [0, -1, -1]$. The corresponding optimization problem is:

$$\begin{aligned} \underset{\alpha}{\text{minimize}} \quad & \sin((1 + 0\alpha)(2 - \alpha)) \\ & + e^{((2-\alpha)+(3-\alpha))} - (3 - \alpha) \end{aligned}$$



which simplifies to:

$$\underset{\alpha}{\text{minimize}} \quad \sin(2 - \alpha) + e^{(5-2\alpha)} + \alpha - 3$$

The minimum is at $\alpha \approx 3.127$ with $\mathbf{x} \approx [1, -1.126, -1.126]$. **Note I:** $\nabla f(\alpha) = -\cos(2 - \alpha) - 2e^{(5-2\alpha)} + 1 = 0$. We can solve for α by using `vpasolve` in `Matlab`.

Note II: We can use Nonlinear search in Matlab or Julia like **fminbnd** from the original problem.

Line Search : Exact Line Search

Nonlinear optimization using `fimnbnd`:

```
f = @(alpha) sin(2-alpha) + exp(5-2*alpha) + alpha - 3
alpha_1 = fminbnd(f, 0, 4, opts)
```

Solve for zero gradient `vpasolve`:

```
syms alpha
f1 = -cos(2-alpha) - 2*exp(5-2*alpha) + 1;
alpha_1 = vpasolve(f1==0,alpha)
```

We should have the same result around 3.127.

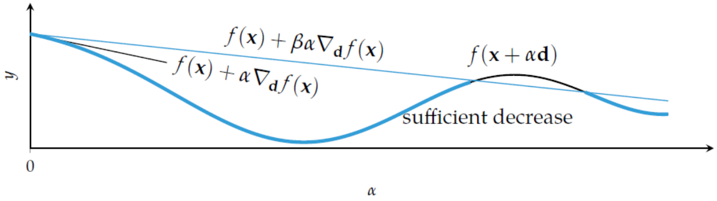
Approximate Line Search

- It is often more computationally efficient to perform more iterations of a descent method than to do an exact line search at each iteration, especially if the function and derivative calculations are expensive.
- Many methods discussed so far can benefit from using **approximate line search** to find a suitable step size with a small number of evaluations.
- Since descent methods must descend, a step size α may be suitable if it causes a decrease in the objective function value. We need the *sufficient decrease* condition. (to protect that the reductions in f values is not to small.)
- The sufficient decrease in the objective function value:

$$f(\mathbf{x}_{k+1}) \leq f(\mathbf{x}_k) + \beta\alpha\nabla_{\mathbf{d}_k} f(\mathbf{x}_k)$$

with $\beta \in [0, 1]$ often set to $\beta = 1 \times 10^{-4}$.

Approximate Line Search



$f(\mathbf{x}) + \beta\alpha\nabla_{\mathbf{d}} f(\mathbf{x})$
$f(\mathbf{x}) + \alpha\nabla_{\mathbf{d}} f(\mathbf{x})$
$f(\mathbf{x} + \alpha\mathbf{d})$
sufficient decrease
y
0
α

- If $\beta = 0$, then any decrease is acceptable. If $\beta = 1$, then the decrease has to be at least as much as what would be predicted by a first-order approximation.
- If $\mathbf{d}$ is a valid descent direction, then there must exist a sufficiently small step size that satisfies the sufficient decrease condition.
- We can start with a large step size and decrease it by a constant reduction factor until the sufficient decrease condition is satisfied.
- The algorithm is known as *backtracking line search* because of how it backtracks along the descent direction.

Approximate Line Search

```
function BACKTRACKING_LINE_SEARCH($(f, \nabla f, x, d, \alpha; p = 0.5, \beta = 1e-4)$)
    while $f(\mathbf{x} + \alpha * \mathbf{d}) > f(\mathbf{x}) + \beta\alpha(\nabla f^T(\mathbf{x})\mathbf{d})$ do
        $\alpha \ = \ p\alpha$
    end while
    return $\alpha$
end function
```

- The first condition is insufficient to guarantee convergence to a local minimum. Very small step sizes will satisfy the first condition but can prematurely converge.
- Backtracking line search avoids premature convergence by accepting the largest satisfactory step size obtained by sequential downscaling and is guaranteed to converge to ta local minimum.

Curvature Condition

The *curvature condition* requires the the directional derivative at the next iterate to be shallower (α is not too close to zero):

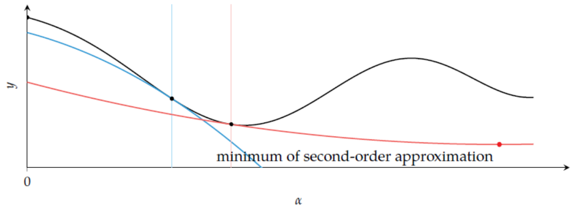
$$\nabla_{\mathbf{d}_k} f(\mathbf{x}_{k+1}) \geq \sigma \nabla_{\mathbf{d}_k} f(\mathbf{x}_k)$$

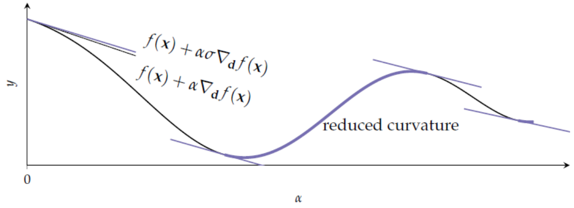
- Where σ controls how shallow the next directional derivative must be.
- It is common to set $\beta < \sigma < 1$ with $\sigma = 0.1$ when approximate linear search is used with the conjugate gradient method and to 0.9 when used with Newton's method.
- The *strong curvature condition*, which is more restrictive criterion in that is also required not to be too positive:

$$|\nabla_{\mathbf{d}_k} f(\mathbf{x}_{k+1})| \leq -\sigma \nabla_{\mathbf{d}_k} f(\mathbf{x}_k)$$

- Both sufficient decrease condition (for α_U) and strong curvature condition are called **strong Wolfe conditions.**(for α_L).

Curvature Condition





Wolfe Condition

Consider approximate line search on $f(x_1, x_2) = x_1^2 + x_1x_2 + x_2^2$ from $\mathbf{x} = [1, 2]$ in the direction $\mathbf{d} = [-1, -1]$, using a maximum step size of 10, a reduction factor of 0.5, a first Wolfe condition parameter $\beta = 1 \times 10^{-4}$ and a second Wolfe condition parameter $\sigma = 0.9$.
The first Wolfe condition is $f(\mathbf{x} + \alpha\mathbf{d}) \leq f(x) + \beta\alpha(\mathbf{g}^T\mathbf{d})$, where $\mathbf{g} = \nabla f(\mathbf{x}) = [4, 5]$.
$\alpha = 10$ we have

$$f\left(\begin{bmatrix}1\\2\end{bmatrix} + 10\begin{bmatrix}-1\\-1\end{bmatrix}\right) \leq 7 + 1 \times 10^{-4}(10)\left(\begin{bmatrix}4 & 5\end{bmatrix}\begin{bmatrix}-1\\-1\end{bmatrix}\right)$$

$217 \leq 6.991$ (It is not satisfied.)

$\alpha = 0.5(10) = 5$ we have

$$f\left(\begin{bmatrix}1\\2\end{bmatrix} + 5\begin{bmatrix}-1\\-1\end{bmatrix}\right) \leq 7 + 1 \times 10^{-4}(5)\begin{bmatrix}4 & 5\end{bmatrix}\begin{bmatrix}-1\\-1\end{bmatrix}$$

$37 \leq 6.996$ (It is not satisfied.)

Wolfe Condition

$\alpha = 0.5(5) = 2.5$, we have

$$f\left(\begin{bmatrix}1\\2\end{bmatrix} + 2.5\begin{bmatrix}-1\\-1\end{bmatrix}\right) \le 7 + 1\times 10^{-4}(2.5)\begin{bmatrix}4 & 5\end{bmatrix}^T\begin{bmatrix}-1\\-1\end{bmatrix}$$

$$3.25 \le 6.998 (\text{ The first Wolfe condition is satisfied.})$$

The candidate design point $\mathbf{x}' = \mathbf{x} + \alpha\mathbf{d} = [-1.5, -0.5]^T$ is checked against the second Wolfe condition:

$$\nabla_{\mathbf{d}} f(\mathbf{x}') \ge \sigma \nabla_{\mathbf{d}} f(\mathbf{x})$$

$$\begin{bmatrix}-3.5 & -2.5\end{bmatrix}\begin{bmatrix}-1\\-1\end{bmatrix} \ge \sigma\begin{bmatrix}4 & 5\end{bmatrix}\begin{bmatrix}-1\\-1\end{bmatrix}$$

$$6 \ge -8.1 (\text{ The second Wolfe condition is satisfied. })$$

Approximate line search terminates with $\mathbf{x} = [-1.5 \ -0.5]^T$.

The Steepest Descent Method

The steepest-descent mehtod (also called *gradient descent*) is a simple and intuitive method for determining the search direction.

Direction Vector:

- Let $\mathbf{x}_k$ be the current point at the kth iteration: $k = 0$ corresponds to the starting point.
- We need to choose a downhill direction $\mathbf{d}$ and then a step size $\alpha > 0$ such that the new point $\mathbf{x}_k + \alpha\mathbf{d}$ is better. We desire $f(\mathbf{x}_k + \alpha\mathbf{d}) < f(\mathbf{x}_k)$.
- To see how $\mathbf{d}$ should be chosen, we use the Taylor's expansion

$$f(\mathbf{x}_k + \alpha\mathbf{d}) = f(\mathbf{x}_k) + \alpha\nabla f(\mathbf{x}_k)^T\mathbf{d} + O(\alpha^2)$$
$$\delta f = \alpha\nabla f(\mathbf{x}_k)^T\mathbf{d} + O(\alpha^2)$$

- For small enough α the term $O(\alpha^2)$ is dominated. Consequently, we have

$$\delta f \approx \alpha\nabla f(\mathbf{x}_k)^T\mathbf{d}$$

The Steepest Descent Method

- For a reduction in f or $\delta f < 0$, we require $\mathbf{d}$ to be a *descent direction* or a direction that satisfies

$$\nabla f(\mathbf{x}_k)^T \mathbf{d} < 0$$

- The steepest descent method is based on choosing $\mathbf{d}$ at the kth iteration, which we will denote as $\mathbf{d}_k$, as

$$\mathbf{d}_k = -\nabla f(\mathbf{x}_k) \quad \Longrightarrow \nabla f(\mathbf{x}_k)^T(-\nabla f(\mathbf{x}_k)) = -\|\nabla f(\mathbf{x}_k)\|^2 < 0$$

- This direction will be referred to as the **steepest descent direction**, and the direction is satisfied the condition.

The Steepest Descent Method : Example

Given $f(\mathbf{x}) = x_1 x_2^2, \qquad \mathbf{x}_0 = \begin{bmatrix} 1 & 2 \end{bmatrix}^T$. The gradient is $\nabla f(\mathbf{x}) = \begin{bmatrix} x_2^2 & 2x_1x_2 \end{bmatrix}^T$.

1. Find the steepest descent direction at $\mathbf{x}_0$

$$\mathbf{d} = -\left.\nabla f(\mathbf{x})\right|_{\mathbf{x}=\mathbf{x}_0} = \begin{bmatrix} -4 & -4 \end{bmatrix}^T$$
$$\mathbf{d}(\text{normalized}) = \begin{bmatrix} -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix}^T$$

2. Is $\mathbf{d} = \begin{bmatrix} -1 & 2 \end{bmatrix}^T$ a direction of descent?

$$\left.\nabla f(\mathbf{x})^T\right|_{x=x_0} \mathbf{d} = \begin{bmatrix} 4 & 4 \end{bmatrix} \begin{bmatrix} -1 \\ 2 \end{bmatrix} = 4 > 0$$

It is not a descent direction.

The Steepest Descent Method

After we have the direction vector $\mathbf{d}_k$ at the point $\mathbf{x}_k$, how far to go along this direction?

- We need to develop a numerical procedure to determine the step size α_k along $\mathbf{d}_k$.
- If we move along $\mathbf{d}_k$ the design variables and the objective function depen only on α as

$$\mathbf{x}(\alpha) = \mathbf{x}_k + \alpha \mathbf{d}_k, \qquad f(\alpha) = f(\mathbf{x}_k + \alpha \mathbf{d}_k)$$
$$\alpha_k = \arg \underset{\alpha}{\text{minimize}}\, f(\mathbf{x}_k + \alpha \mathbf{d}_k)$$

- The optimization above implies that the directional derivative equals zero. We have

$$\nabla f(\mathbf{x}_k + \alpha \mathbf{d}_k)^T \mathbf{d}_k = 0, \quad \mathbf{d}_{k+1} = -\frac{\nabla f(\mathbf{x}_k + \alpha \mathbf{d}_k)}{\|\nabla f(\mathbf{x}_k + \alpha \mathbf{d}_k)\|}$$
$$\mathbf{d}_{k+1}^T \mathbf{d}_k = 0, \quad \mathbf{d}_{k+1} \text{ and } \mathbf{d}_k \text{ are orthogonal.}$$

The Steepest Descent Method

- In the steepest descent method, the direction vector is $-\nabla f(\mathbf{x}_k)$ resulting in the slope at the current point $\alpha = 0$ being

$$\left.\frac{df(\alpha)}{d\alpha}\right|_{\alpha=0} = \nabla f(\mathbf{x}_k)^T(-\nabla f(\mathbf{x}_k)) = -\|\nabla f(\mathbf{x}_k)\|^2 < 0$$

  Implying a move in a downhill direction.

The Steepest Descent Method

- Starting from an initial point, we determine a direction vector and a step size, and obtain a new point as $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k$.
- The question is to know when to stop the iterative process. We have two stop criteria to discuss here.
- First Befor performing line search, the necessary condition for optimality is checked:

$$\|\nabla f(\mathbf{x}_k)\| \leq \varepsilon_G,$$

  where ϵ_G is a tolerance on the gradient and is supplied by the user. If the condition is satisfied the the process is terminated.

The Steepest Descent Method : Stoping Criteria

- **Second:** We check the successive reductions in f as a criterion for stopping.

$$|f(\mathbf{x}_{k+1}) - f(\mathbf{x}_k)| \leq \varepsilon_A + \varepsilon_R |f(\mathbf{x}_k)|$$

  where $\varepsilon_A =$ absolute tolerance on the change in function value and $\varepsilon_R =$ relative tolerance. Only if the condition is satisfied for two consecutive iterations is the descen process stopped.

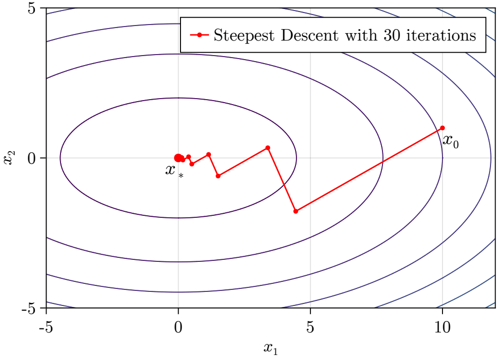
Steepest Descent Algorithm : Algorithm

```
Require: $\mathbf{x}_0, \varepsilon_G, \varepsilon_A, \varepsilon_R$
  $k = 0$
  while true do
      Compute $\nabla f(\mathbf{x}_k)$
      if $\|\nabla f(x_k)\| \leq \varepsilon_G$ then
          Stop
      else if  then
          $\mathbf{d}_k = -\nabla f(\mathbf{x}_k)/\|\nabla f(\mathbf{x}_k)\|$
      end if
      $\alpha_k =$ line_search(f, d),
      $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k,$
      if $|f(\mathbf{x}_{k+1}) - f(\mathbf{x}_k)| \leq \varepsilon_A + \varepsilon_R |f(\mathbf{x}_k)|$ then
          Stop
      else
          $k = k + 1, \mathbf{x}_k = \mathbf{x}_{k+1}$
      end if
  end while
```

Steepest Descent Algorithm: Zig-Zags Property

- The steepest descent method zig-zags its way towards the optimum point. Consider

$$f(x_1, x_2) = x_1^2 + 5x_2^2.$$



Steepest Descent Algorithm: Zig-Zags Property

From the fact that α_k is obtained by minimizing $f(\mathbf{x}_k + \alpha \mathbf{d}_k)$. Thus

$$\begin{aligned}
\frac{\partial f(\mathbf{x}_k + \alpha \mathbf{d}_k)}{\partial \alpha} &= 0 \\
\frac{\partial f(\mathbf{x}_{k+1})}{\partial \alpha} = \frac{\partial f(\mathbf{x}_{k+1})}{\partial \mathbf{x}_{k+1}} \frac{\partial \mathbf{x}_{k+1}}{\partial \alpha} &= \frac{\partial f(\mathbf{x}_{k+1})}{\partial \mathbf{x}_{k+1}} \frac{\partial (\mathbf{x}_k + \alpha \mathbf{d}_k)}{\partial \alpha} = 0 \\
\nabla f(\mathbf{x}_{k+1})^T \mathbf{d}_k &= 0
\end{aligned}$$

Set $\mathbf{d} = -\nabla f_k$, we have

$$-\nabla f(\mathbf{x}_{k+1})^T \nabla f(\mathbf{x}_k) = 0$$

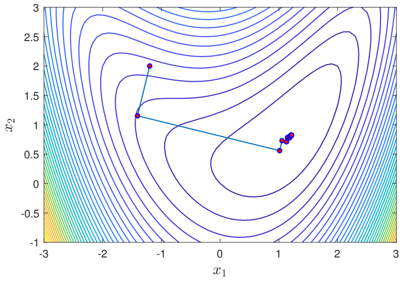
From the last line, it means the $k+1$ direction is perpendicular to the k direction. If you use the approximation line search, this perpendicular property is lost, but the zig-zags are still there.

Steepest Descent Algorithm : The Bean function

Find the minimum of the bean function

$$f(x_1, x_2) = (1 - x_1)^2 + (1 - x_2)^2 + \frac{1}{2}\left(2x_2 - x_1^2\right)^2,$$

using the steepest-descent algorithm with an exact line search, and a convergence tolerance of $\|\nabla f\| \leq 10^{-6}$.



$$\mathbf{x}^* = \begin{bmatrix} 1.2134 \\ 0.8241 \end{bmatrix}, \; f(\mathbf{x}^*) = 0.0919$$

Steepest Descent Algorithm : Convergence Characteristics

- The speed of convergence of the method is related to the spectral condition number of the Hessian matrix. The spectral condition number κ of a symmetric positive definite matrix A is defined as the ratio of the largest to the smallest eigenvalue, or

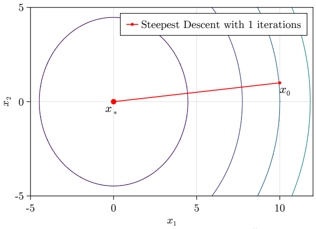
$$\kappa = \frac{\lambda_{\text{max}}}{\lambda_{\text{min}}}$$

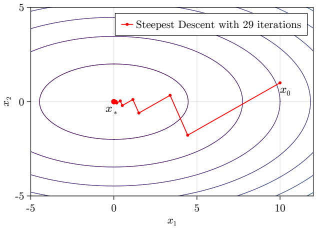
- For well-conditioned Hessian matrices, the condition number is close to unity, contours are more circular, and the method is a its best.
- The higher the condition number, the more ill-conditioned is the Hessian, the contours are more elliptical, more is the amount of zig-zagging near as the optimum is approached, the smaller are the step sizes and thus the poorer is the rate of convergence.

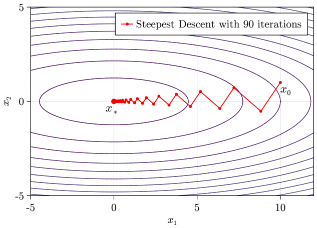
Steepest Descent Algorithm : Convergence Characteristics

Consider a function $f(x_1, x_2) = x_1^2 + \beta x_2^2$, we have

$$\mathbf{H}(f) = \nabla^2 f = \begin{bmatrix} 2 & 0 \\ 0 & 2\beta \end{bmatrix}, \text{ with } \beta = 1, 5, 15$$







The Conjugate Gradient Method

- The Conjugate Gradient Method [Fletcher and Powell 1963] is a dramatic improve over the steepest descent method. The steepest descent perform poorly in narrow valleys.
- It can find the minimum of a quadratic function of n variables in n iterations.
- The conjugate Gradient method is also powerful on general functions.
- Consider the problem of minimizing a quadratic function

$$\underset{\mathbf{x}}{\text{minimize}} \quad q(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} + \mathbf{b}^T\mathbf{x} + c$$

  where $\mathbf{A}$ is a symmetric and positive definite.
- The conjugate directions, or directions that are mutually conjugate with respect to A, as vectors which satisfy

$$\mathbf{d}_i^T\mathbf{A}\mathbf{d}_j = 0, \quad i \neq j, 0 \leq i, j \leq n$$

The Conjugate Gradient Method : The method

- The mutually conjugate vectors are the basis vectors of $\mathbf{A}$. They are generally no orthogonal to one another.
- The algorithm is started with the direction of steepest descent:

$$\mathbf{d}_1 = -\mathbf{g}_1$$

- Use line search to find the next design point For quadratic functions, the step factor α can be computed exactly. The update is then:

$$\mathbf{x}_2 = \mathbf{x}_1 + \alpha_1 \mathbf{d}_1$$

The Conjugate Gradient Method : The method

- Suppose we want ot derive the optimal step factor for a line search on a quadratic function:

$$\underset{\alpha}{\text{minimize}} \quad f(\mathbf{x} + \alpha \mathbf{d})$$

We have

$$\begin{aligned}
\frac{\partial f(\mathbf{x} + \alpha \mathbf{d})}{\partial \alpha} &= \frac{\partial}{\partial \alpha}\left[\frac{1}{2}(\mathbf{x} + \alpha \mathbf{d})^T \mathbf{A}(\mathbf{x} + \alpha \mathbf{d}) + \mathbf{b}^T(\mathbf{x} + \alpha \mathbf{d}) + c\right] \\
&= \mathbf{d}^T \mathbf{A}(\mathbf{x} + \alpha \mathbf{d}) + \mathbf{b}^T \mathbf{d} = \mathbf{d}^T \mathbf{A}(\mathbf{x} + \alpha \mathbf{d}) + \mathbf{d}^T \mathbf{b} \\
&= \mathbf{d}^T(\mathbf{A}\mathbf{x} + \mathbf{b}) + \alpha \mathbf{d}^T \mathbf{A}\mathbf{d}
\end{aligned}$$

Setting $\frac{\partial f(\mathbf{x} + \alpha \mathbf{d})}{\partial \alpha} = 0$ results in:

$$\alpha = -\frac{\mathbf{d}^T(\mathbf{A}\mathbf{x} + \mathbf{b})}{\mathbf{d}^T \mathbf{A}\mathbf{d}}$$

The Conjugate Gradient Method : The method

- Subsequent iterations choose $\mathbf{d}_{k+1}$ based on the current gradient and a contribution from the previous descent direction:

$$\mathbf{d}_k = -\mathbf{g}_k + \beta_k \mathbf{d}_{k-1}$$

  for scalar parameter β. Larger values of β indicate that the previous descent direction contributes more strongly.

- To find the best value for β for a known $\mathbf{A}$, using the fact that $\mathbf{d}_k$ is conjugate to $\mathbf{d}_{k-1}$:

$$\begin{aligned} \mathbf{d}_k^T \mathbf{A} \mathbf{d}_{k-1} = 0 \quad &\Longrightarrow (-\mathbf{g}_k + \beta_k \mathbf{d}_{k-1})^T \mathbf{A} \mathbf{d}_{k-1} = 0 \\ -\mathbf{g}_k^T \mathbf{A} \mathbf{d}_{k-1} + \beta_k \mathbf{d}_{k-1}^T \mathbf{A} \mathbf{d}_{k-1} = 0 \quad &\Longrightarrow \beta_k = \frac{\mathbf{g}_k^T \mathbf{A} \mathbf{d}_{k-1}}{\mathbf{d}_{k-1}^T \mathbf{A} \mathbf{d}_{k-1}} \end{aligned}$$

- The conjugate gradient method can be applied to nonquadratic functions as well.

The Conjugate Gradient Method : The method

We do not know the value of A that best approximates f around $\mathbf{x}_k$. Several choices for β_k tend to work well:

- Fletcher-Reeves:

$$\beta_k = -\frac{\mathbf{g}_k^T \mathbf{g}_k}{\mathbf{g}_{k-1}^T \mathbf{g}_{k-1}}$$

- Polak-Ribière:

$$\beta_k = \frac{\mathbf{g}_k^T \left(\mathbf{g}_k - \mathbf{g}_{k-1}\right)}{\mathbf{g}_{k-1}^T \mathbf{g}_{k-1}}$$

- Convergence for the Polak-Ribière method can be guaranteed if we modify it to allow for automatic resets:

$$\beta \leftarrow \max(\beta, 0)$$

The Conjugate Gradient Method : Example

Consider $f = x_1^2 + 4x_2^2$, $\mathbf{x}_0 = \begin{bmatrix} 1 & 1 \end{bmatrix}^T$. We will perform two iterations of the conjugate gradient algorithm. The first step is the steepest descent iteration. Thus

$$\mathbf{d}_0 = -\nabla f(\mathbf{x}_0) = -\begin{bmatrix} 2 & 8 \end{bmatrix}^T$$

Assuming the direction vectors are not normalized to be unit vectors,

$$f(\alpha) = f(\mathbf{x}_0 + \alpha \mathbf{d}_0) = (1 - 2\alpha)^2 + 4(1 - 8\alpha)^2$$

which yields $\alpha_0 = 0.1308$, $\mathbf{x}_1 = \mathbf{x}_0 + \alpha_0 \mathbf{d}_0 = \begin{bmatrix} 0.7385 & -0.0462 \end{bmatrix}^T$. The next iteration (using Fletcher-Reeves method):

$$\beta_0 = \frac{\|\nabla f(\mathbf{x}_1)\|^2}{\|\nabla f(\mathbf{x}_0)\|^2} = 2.3176/68 = 0.0341$$

The Conjugate Gradient Method : Example

$$\mathbf{d}_1 = -\nabla f^T(\mathbf{x}_1) + \beta_0 \mathbf{d}_0 = \begin{bmatrix} -1.4770 \\ 0.3692 \end{bmatrix} + 0.0341 \begin{bmatrix} -2 \\ -8 \end{bmatrix} = \begin{bmatrix} -1.5451 \\ 0.0966 \end{bmatrix}$$

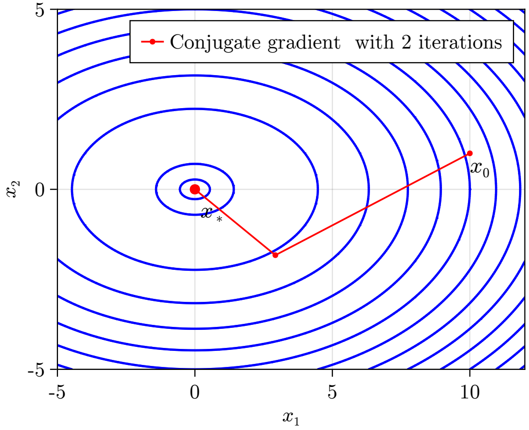
We have

$$f(\alpha) = f(\mathbf{x}_1 + \alpha \mathbf{d}_1) = (0.7385 - 1.5451\alpha)^2 + 4(-0.0462 + 0.0966\alpha)^2$$

which yields

$$\begin{aligned} \alpha_1 &= 0.4780 \\ \mathbf{x}_2 &= \mathbf{x}_1 + \alpha_1 \mathbf{d}_1 = \begin{bmatrix} 0.7385 \\ -0.0462 \end{bmatrix} + 0.4780 \begin{bmatrix} -1.5451 \\ 0.0966 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \end{aligned}$$

The Conjugate Gradient Method : Example



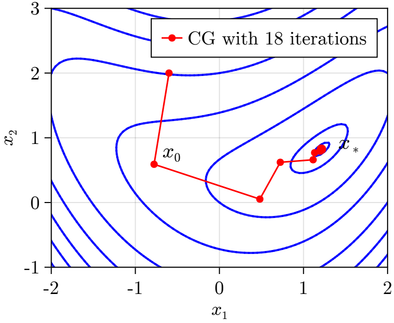
The Conjugate Gradient Method : Algorithm

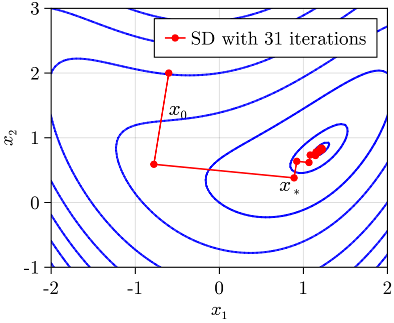
```
Require: $\mathbf{x}_0$, $\varepsilon_G$
  $k = 0$
  while $\|\nabla f_k\| > \varepsilon_G$ do
      if $k = 0$ then
          $\mathbf{d}_k = -\frac{\nabla f_x}{\|\nabla f_k\|}$
      else
          $\beta_k = \frac{\nabla f_x^T \nabla f_k}{\nabla f_{k-1}^T \nabla f_{k-1}}$
          $\mathbf{d}_k = -\frac{\nabla f_k}{\|\nabla f_k\|} + \beta_k \mathbf{d}_{k-1}$
      end if
      $\alpha_k =$ line_search($f, \mathbf{d}_k$)
      $\mathbf{x}_k = \mathbf{x}_{k-1} + \alpha_k \mathbf{d}_k$
      $k = k + 1$
  end while
```

The Conjugate Gradient Method : Example

The minimum of the bean function,

$$f(x_1, x_2) = (1 - x_1)^2 + (1 - x_2)^2 + \frac{1}{2}\left(2x_2 - x_1^2\right)^2$$





Newton's Method

- The function value and gradient can help to determine the direction to travel, but it does not directly help to determine how far to step to reach a local minimum.
- Second-order information allows us to make a quadratic approximation of the objective function and approximate the right step size to reach a local minimum.
- As we have seen with a quadratic fit search, we can analytically obtain the location where a quadratic approximation has a zero gradient. We can use that location as the next iteration to approach a local minimum.
- The quadratic approximation about a point $\mathbf{x}_k$ comes from the second-order Taylor expansion (scalar case):

$$f(x_k + s) = f(x_k) + f'(x_k)s + \frac{1}{2}s^2 f''(x_k)$$

$$\frac{d}{ds} f(x_k + s) = 0 + f'(x_k) + s f''(x_k) = 0$$

$$s = -\frac{f'(x_k)}{f''(x_k)}, \text{ where } s \text{ is the step size}$$

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}$$

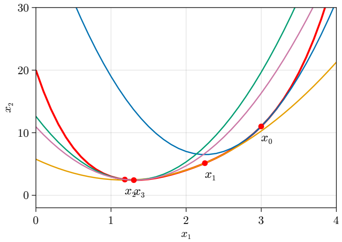
Newton's Method : Example

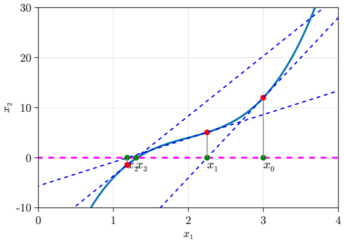
Suppose we want to minimize the following single-variable function:

$$f(x) = (x-2)^4 + 2x^2 - 4x + 4, \quad f'(x) = 4(x-2)^3 + 4x - 4,$$
$$f''(x) = 12(x-2)^2 + 4$$

with $x_0 = 3$, we can form the quadratic using the function value and the first and second derivatives evaluated at the point.

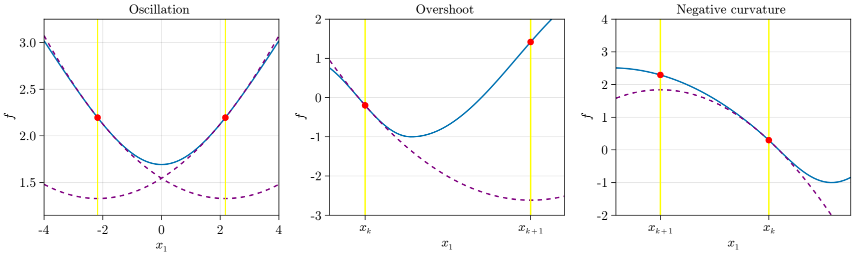
$$x_1 = x_0 - \frac{f'(x_0)}{f\prime\prime(x_0)} = 2.25, \quad x_2 = x_1 - \frac{f'(x_1)}{f\prime\prime(x_1)} = 1.1842, \quad x_3 = 1.3039, \quad x^* = 1.3177$$





Newton' s Method : Disadvantage

- The update rule in Newton's method involves dividing by the second derivative. The update is undefined if the second derivative is zero, which occurs when the quadratic approximation is a horizontal line.
- Instability also ocurs when the second derivative is very close to zero, in which case the next iterate will lie very far from the current design point, far from where the local quadratic approximation is valid.
- Poor local approximations can lead to poor performance with Newton's method.



Newton's Method : Multivariate Optimization

- The multivariate second-order Taylor expansion at x_k is

$$f(\mathbf{x}_k + \mathbf{s}) \approx f(\mathbf{x}_k) + \nabla(f(\mathbf{x}_k)^T \mathbf{s} + \frac{1}{2}\mathbf{s}^T \mathbf{H}_k \mathbf{s}$$
$$\frac{d}{d\mathbf{x}}(\mathbf{x}_k + \mathbf{s}) = \nabla f(\mathbf{x}_k) + \mathbf{H}_k \mathbf{s} = 0$$

We then solve for the next iterate, thereby obtaining Newton's method in multivariate form:

$$\mathbf{s} = -\mathbf{H}_k^{-1} \nabla f(\mathbf{x}_k)$$
$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{H}_k^{-1} \nabla f(\mathbf{x}_k)$$

- If $f(\mathbf{x})$ is quadratic and its Hessian is positive definite, then the update converges to the global minimum in one step. For general functions, Newton's method is often terminated once $\mathbf{x}$ ceases to change by more than a given tolerance.

Newton's Method : Example

With $\mathbf{x}_1 = \begin{bmatrix} 9 & 8 \end{bmatrix}$, we will use Newton's method to minimize Booth's function:

$$f(\mathbf{x}) = (x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2,$$

$$\nabla f(\mathbf{x}) = \begin{bmatrix} 10x_1 + 8x_2 - 34, & 8x_1 + 10x_2 - 38 \end{bmatrix}^T, \quad \mathbf{H}(\mathbf{x}) = \begin{bmatrix} 10 & 8 \\ 8 & 10 \end{bmatrix}$$

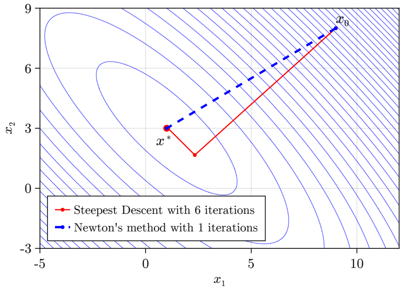
The first iteration of Newton's method yields:

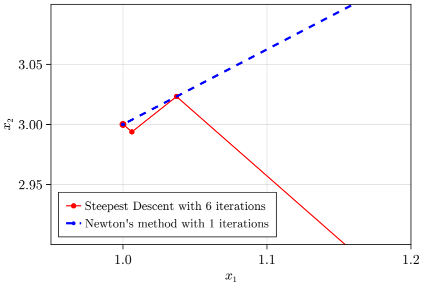
$$\mathbf{x}_2 = \mathbf{x}_1 - \mathbf{H}_1^{-1}\mathbf{g}_1 = \begin{bmatrix} 9 \\ 8 \end{bmatrix} - \begin{bmatrix} 10 & 8 \\ 8 & 10 \end{bmatrix}^{-1} \begin{bmatrix} 10(9) + 8(8) - 34 \\ 8(9) + 10(8) - 38 \end{bmatrix}$$

$$= \begin{bmatrix} 9 \\ 8 \end{bmatrix} - \begin{bmatrix} 10 & 8 \\ 8 & 10 \end{bmatrix}^{-1} \begin{bmatrix} 120 \\ 114 \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \end{bmatrix}$$

The gradient at $\mathbf{x}_2$ is zero, so we have converged after a single iteration. The Hessian is positive definite everywhere, so $\mathbf{x}_2$ is the global minimum.

Newton's Method : Multivariate Optimization





Newton's Method : Algorithm

Let $\mathbf{s} = x_{k+1} - x_k$

```
Require: $\mathbf{x}_0$, $\varepsilon_G$, $\nabla f_k$, $\mathbf{H}_k$
  $k = 0$
  while $\|\nabla f_k\| > \varepsilon_G$ and $k \leq k_{\max}$ do
      $\mathbf{s} = \mathbf{H}(\mathbf{x})^{-1}\nabla f(\mathbf{x})$
      $\mathbf{x} = \mathbf{x} + \mathbf{s}$
      $k = k + 1$
  end while
  return $\mathbf{x}$
```

Quasi-Newton Methods

- Newton's method is efficient because the second-order information results in better search directions. Two main steps in Newton's method
- Need to compute Hessian $\mathbf{H}$
- Solve the system of equations

$$\mathbf{H}\mathbf{s} = -\nabla f(\mathbf{x}) \;\Rightarrow\; \mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{H}^{-1}\nabla f(\mathbf{x})$$

For the Quasi-Newton method

- We can use first-order information (gradients) along each step in the iteration path to build an approximation of Hessian.
- Scalar version

$$f''_{k+1}(\cdot) = \frac{f'_{k+1}(\cdot) - f'_k(\cdot)}{x_{k+1} - x_k} \text{ secant equation}$$

$$\tilde{f}_{k+1}(x_{k+1} + s) \approx f_{k+1}(x_{k+1}) + s f'_{k+1}(x_{k+1}) + \frac{s^2}{2}\left(\frac{f'_{k+1}(x_{k+1}) - f'_k(x_k)}{x_{k+1} - x_k}\right)$$

$$\frac{d}{ds}\tilde{f}_{k+1}(x_{k+1} + s) = \tilde{f}'(x_{k+1} + s) = f'_{k+1}(x_{k+1}) + s\left(\frac{f'_{k+1}(x_{k+1}) - f'_k(x_k)}{x_{k+1} - x_k}\right)$$

Quasi-Newton Methods

- For $s = 0$, we have $\tilde{f}_{k+1}(x_{k+1}) = f'_{k+1}(x_{k+1})$, which mean the slope of the approximation matches the slope of the actual function at x_{k+1} as expect.
- Stepping backward $s = -(x_{k+1} - x_k)$, we have

$$\begin{aligned} \tilde{f}'_{k+1}(x_{k+1} - (x_{k+1} - x_k)) &= \tilde{f}'_{k+1}(x_k) \\ &= f'_{k+1}(x_{k+1}) - (x_{k+1} - x_k)\left(\frac{f'_{k+1}(x_{k+1}) - f'_k(x_k)}{x_{k+1} - x_k}\right) \\ &= f'_k(x_k) \end{aligned}$$

Thus, thenature of this approximation is such that it catches the slope of the actual function at the last two points.

Quasi-Newton Methods: Multivariable case

The quadratic approximation of the objective funcition

$$\tilde{f}(\mathbf{x}_k + \mathbf{p}) = f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)^T \mathbf{p} + \frac{1}{2}\mathbf{p}_k^T \tilde{\mathbf{H}}_k \mathbf{p}_k$$

where $\tilde{H}$ is an approximation of the Hessian. Minimize this quadratic with respect to $\mathbf{p}$, we have

$$\tilde{\mathbf{H}}_k \mathbf{p}_k = -\nabla f(\mathbf{x}_k)$$

We get the $\mathbf{p}_k$ direction and update the point using $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$.
Quasi-Newton methods update the approximate Hessian at every iteration based on the latest information using an update of the form

$$\tilde{\mathbf{H}}_{k+1} = \tilde{\mathbf{H}}_k + \Delta\tilde{\mathbf{H}}_k$$

The approximation of the Hessian must match the slope of the actual function at the last two point.

Quasi-Newton Methods: Multivariable case

Using $\mathbf{x}_k$ in the direction of $\mathbf{p}_k$, we have

$$\begin{aligned} \tilde{f}(\mathbf{x}_{k+1} + \mathbf{p}) &= f(\mathbf{x}_{k+1}) + \nabla f(\mathbf{x}_{k+1})^T \mathbf{p}_k + \frac{1}{2}\mathbf{p}_k^T \tilde{\mathbf{H}}_{k+1}\mathbf{p}_k \\ \nabla \tilde{f}(\mathbf{x}_{k+1} + \mathbf{p}) &= \nabla f(\mathbf{x}_{k+1}) + \tilde{\mathbf{H}}_{k+1}\mathbf{p}_k \end{aligned}$$

If $\mathbf{p} = 0$, we have $\nabla \tilde{f}(\mathbf{x}_{k+1}) = \nabla f(\mathbf{x}_{k+1})$. If $\mathbf{p} = \mathbf{x}_k - \mathbf{x}_{k+1} = -\alpha_k \mathbf{p}_k$

$$\nabla \tilde{f}(\mathbf{x}_{k+1} - \alpha_k \mathbf{p}_k) = \nabla \tilde{f}(\mathbf{x}_k) = \nabla f(\mathbf{x}_{k+1}) - \alpha_k \tilde{\mathbf{H}}_{k+1}\mathbf{p}_k$$

To enforce that the $\nabla \tilde{f}(\mathbf{x}_k) = \nabla f(\mathbf{x}_k)$, we need

$$\nabla f(\mathbf{x}_{k+1}) - \alpha_k \tilde{\mathbf{H}}_{k+1}\mathbf{p}_k = \nabla f(\mathbf{x}_k) \;\Rightarrow\; \alpha_k \tilde{\mathbf{H}}_{k+1}\mathbf{p}_k = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)$$

Simplify the notation, using $\mathbf{s}_k = \mathbf{x}_{k+1} - \mathbf{x}_k = \alpha_k \mathbf{p}_k$, and
$\mathbf{y}_k = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)$. We have

$$\tilde{\mathbf{H}}_{k+1}\mathbf{s}_k = \mathbf{y}_k \text{ secant equation}$$

Quasi-Newton Methods: Multivariable case

We need $\tilde{\mathbf{H}}$ to be positive definite then

$$\tilde{\mathbf{H}}_{k+1}\mathbf{s}_k = \mathbf{y}_k \;\Rightarrow\; \mathbf{s}_k^T\tilde{\mathbf{H}}_{k+1}\mathbf{s}_k = \mathbf{s}_k^T\mathbf{y}_k > 0$$

The latter is called the curvature condition, and it is automatically satisfied if the line search finds a step that satisfies the strong Wolfe conditions.

- The original quasi-Newton update, known ad DFP, was first proposed by Davidon and then refined by Fletcher and slso Powell. The DFP update formula has been superseded by the BFGS formula, which was independently developed by Broyden, Fletcher, Goldfarb, and Shanno.
- The BFGS is currently considered the most effective quasi-Newton update.

Quasi-Newton Methods

- As the secant method approximates f'' in the univariate case, *quasi-Newton* methods approximate the inverse Hessian. Quasi-Newton method updates have the form:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \mathbf{Q}_k \nabla f_k,$$

  where α_k is a scalar step factor and $\mathbf{Q}_k$ approximates the inverse of the Hessian at $\mathbf{x}_k$
- These methods typically set $\mathbf{Q}_0$ to the identity matrix, and they then apply updates to reflect information learned with each iteration. To simplify the equations for the various quasi-Newton methos, we define the following:

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

Quasi-Newton Methods : Davidon-Fletcher-Powell (DFP)

In stead of starting with the update for the Hessian, we use the inverse Hessian $\mathbf{Q}$.

$$\mathbf{Q}_{k+1} = \mathbf{Q}_k + \alpha \mathbf{u}\mathbf{u}^T + \beta \mathbf{v}\mathbf{v}^T$$

$$\tilde{\mathbf{H}}_{k+1}\mathbf{s}_k = \mathbf{y}_k \;\Rightarrow\; \mathbf{Q}_{k+1}\mathbf{y}_k = \mathbf{s}_k$$

Setting $\mathbf{u} = \mathbf{s}_k$ and $\mathbf{v} = \mathbf{Q}_k\mathbf{y}_k$, we have

$$\begin{aligned}
\mathbf{Q}_{k+1} &= \mathbf{Q}_k + \alpha \mathbf{s}_k\mathbf{s}_k^T + \beta \mathbf{Q}_k\mathbf{y}_k\mathbf{y}_k^T\mathbf{Q}_k^T \\
\mathbf{s}_k &= \mathbf{Q}_k\mathbf{y}_k + \alpha \mathbf{s}_k\mathbf{s}_k^T\mathbf{y}_k + \beta \mathbf{Q}_k\mathbf{y}_k\mathbf{y}_k^T\mathbf{Q}\mathbf{y}_k \text{ from secant equation} \\
\mathbf{s}_k - \alpha \mathbf{s}_k\mathbf{s}_k^T\mathbf{y}_k &= \mathbf{Q}_k\mathbf{y}_k + \beta \mathbf{Q}_k\mathbf{y}_k\mathbf{y}_k^T\mathbf{Q}_k\mathbf{y}_k^T \\
\mathbf{s}_k(1 - \alpha \mathbf{s}_k^T\mathbf{y}_k) &= \mathbf{Q}_k\mathbf{y}_k(1 + \beta \mathbf{y}_k^T\mathbf{Q}_k\mathbf{y}_k^T)
\end{aligned}$$

The last equation is correct if both sides are zero or

$$\alpha = \frac{1}{\mathbf{s}_k^T\mathbf{y}_k}, \qquad \beta = \frac{-1}{\mathbf{y}_k^T\mathbf{Q}_k\mathbf{y}_k}$$

Quasi-Newton Methods : Davidon-Fletcher-Powell (DFP)

The *Davidon-Fletcher-Powell* (DFP) method uses:

$$\mathbf{Q}_{k+1} = \mathbf{Q}_k + \frac{\mathbf{s}_k \mathbf{s}_k^T}{\mathbf{s}_k^T \mathbf{y}_k} - \frac{\mathbf{Q}_k \mathbf{y}_k \mathbf{y}_k^T \mathbf{Q}_k}{\mathbf{y}_k^T \mathbf{Q}_k \mathbf{y}_k}$$

The update for $\mathbf{Q}$ in the DFP method havs three properties:

- $\mathbf{Q}$ remains symmetric and positive definite.
- If $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T \mathbf{A}\mathbf{x} + \mathbf{b}^T \mathbf{x} + c$, then $\mathbf{Q} = \mathbf{A}^{-1}$. Thus the DFP has the same convergence properties as the conjugate gradient method.
- For high-dimensional problems, storing and updating $\mathbf{Q}$ can be significant compared to other methods like the conjugate gradient method.
- The DPF algorithm does not guarantee the positiveness of the Hessian $\tilde{\mathbf{H}}$.

Quasi-Newton Methods : Davidon-Fletcher-Powell (DFP)

```
Require: $\mathbf{x}_0$, $\varepsilon_G$, $f, \nabla f(\mathbf{x}_0)$
  $k = 0, \mathbf{Q} = I$
  while $\|\nabla f(\mathbf{x}_k)\| > \varepsilon_G$ && $k \leq k_{\max}$ do
      $\mathbf{g} = \nabla f(\mathbf{x}_k)$
      $\mathbf{x}' =$ line_search($f, \mathbf{x}_k, -\mathbf{Q} * \mathbf{g}$)
      $\mathbf{g}' = \nabla f(\mathbf{x}')$
      $\mathbf{s} = \mathbf{x}' - \mathbf{x}_k$
      $\mathbf{y} = \mathbf{g}' - \mathbf{g}$
      $\mathbf{Q} = \mathbf{Q} - \mathbf{Q}\mathbf{y}_k\mathbf{y}_k^T\mathbf{Q}/\mathbf{y}_k^T\mathbf{Q}\mathbf{y}_k + \mathbf{s}_k\mathbf{s}_k^T/\mathbf{s}_k^T\mathbf{y}_k$
      $k = k + 1$
      $\mathbf{x}_k = \mathbf{x}'$
  end while
  return $\mathbf{x}'$
```

Quasi-Newton Methods : Broyden-Fletcher-Goldfarb-Shanno (BFGS)

In addition to the secant equation, we would like

- $\tilde{\mathbf{H}}_{k+1}$ is symmetric
- $\tilde{\mathbf{H}}_{k+1}$ is close to $\tilde{\mathbf{H}}_k$
- $\tilde{\mathbf{H}}$ is positive definite the $\tilde{\mathbf{H}}_{k+1}$ is positive definite.

Using rank-1 update ($\tilde{\mathbf{H}}_{k+1} = \tilde{\mathbf{H}}_k + \alpha \mathbf{u}\mathbf{u}^T$ is satisfied the secant equation, but is not guaranteed to be positive definite. The BFGS is a rank-2 update

$$\tilde{\mathbf{H}}_{k+1} = \tilde{\mathbf{H}}_k + \alpha \mathbf{u}\mathbf{u}^T + \beta \mathbf{v}\mathbf{v}^T \tilde{\mathbf{H}}_{k+1}\mathbf{s}_k \quad = \mathbf{y}_k \;\Rightarrow\; \tilde{\mathbf{H}}_k\mathbf{s}_k + \alpha \mathbf{u}\mathbf{u}^T\mathbf{s}_k + \beta \mathbf{v}\mathbf{v}^T\mathbf{s}_k = \mathbf{y}_k$$

Setting $\mathbf{u} = \mathbf{y}$ and $\mathbf{v} = \tilde{\mathbf{H}}\mathbf{s}$ yields

$$\begin{aligned} \tilde{\mathbf{H}}_k\mathbf{s}_k + \alpha \mathbf{y}_k\mathbf{y}_k^T\mathbf{s}_k + \beta \tilde{\mathbf{H}}_k\mathbf{s}_k \left(\tilde{\mathbf{H}}_k\mathbf{s}_k\right)^T \mathbf{s}_k &= \mathbf{y}_k \\ \mathbf{y}_k\left(1 - \alpha \mathbf{y}_k^T\mathbf{s}_k\right) &= \tilde{\mathbf{H}}_k\mathbf{s}_k\left(1 + \beta \mathbf{s}_k^T\tilde{\mathbf{H}}_k\mathbf{s}_k\right) \end{aligned}$$

we need $\alpha = \frac{1}{\mathbf{y}_k^T\mathbf{s}_k}$, and $\beta = -\frac{1}{\mathbf{s}_k^T\tilde{\mathbf{H}}_k\mathbf{s}_k}$.

Quasi-Newton Methods : Broyden-Fletcher-Goldfarb-Shanno (BFGS)

We get the BFGS update:

$$\tilde{\mathbf{H}}_{k+1} = \tilde{\mathbf{H}}_k + \frac{\mathbf{y}_k \mathbf{y}_k^T}{\mathbf{y}_k^T \mathbf{s}_k} - \frac{\tilde{\mathbf{H}}_k \mathbf{s}_k \mathbf{s}_k^T \tilde{\mathbf{H}}_k}{\mathbf{s}_k^T \tilde{\mathbf{H}}_k \mathbf{s}_k}$$

It is more efficient to approximate the inverse of the Hessian directly instead. The inverse can be found analytically from the update $\tilde{\mathbf{H}}$ using the Sherman-Morrison-Woodbury formula.

$$\tilde{\mathbf{Q}}_{k+1} = \left(I - \sigma_k \mathbf{s}_k \mathbf{y}_k^T\right) \tilde{\mathbf{Q}}_k \left(I - \sigma_k \mathbf{y}_k \mathbf{s}_k^T\right) + \sigma_k \mathbf{s}_k \mathbf{s}_k^T,$$

where $\sigma_k = \frac{1}{\mathbf{y}_k^T \mathbf{s}_k}$. We have

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha \tilde{\mathbf{Q}}_k \nabla f(\mathbf{x}_k)$$

Broyden-Fletcher-Goldfarb-Shanno (BFGS) proof

Starting with the matrix version of the update equation,

$$\tilde{\mathbf{H}}_{k+1} = \tilde{\mathbf{H}}_k - \tilde{\mathbf{H}}_k \mathbf{U}_k \left(\mathbf{I} + \mathbf{V}_k^T \tilde{\mathbf{H}}_k \mathbf{U}_k\right)^{-1} \mathbf{V}_1 \mathbf{V}_2 \tilde{\mathbf{H}}_k,\ \mathbf{V}_1\mathbf{V}_2 = \mathbf{V}_k^T,\ \mathbf{V}_2 = \mathbf{U}^T,\ \tilde{\mathbf{H}}^{-1} = \mathbf{Q}$$

$$\mathbf{V}_1 = \begin{bmatrix} -\frac{1}{\mathbf{s}_k^T \mathbf{Q}_k \mathbf{s}_k} & 0 \\ 0 & \frac{1}{\mathbf{y}_k^T \mathbf{s}_k} \end{bmatrix},\ \mathbf{V}_2 = \begin{bmatrix} \mathbf{s}_k^T \mathbf{Q}_k \\ \mathbf{y}_k^T \end{bmatrix},\ \mathbf{U}_k = \begin{bmatrix} \mathbf{Q}_k \mathbf{s}_k & \mathbf{y}_k \end{bmatrix},$$

Let $\mathbf{v} = \tilde{\mathbf{H}}_k \mathbf{U} = (\mathbf{V}_2 \tilde{\mathbf{H}}_k)^T = \begin{bmatrix} \mathbf{s}_k & \tilde{\mathbf{H}}_k \mathbf{y}_k \end{bmatrix}$, and

$$(\mathbf{V}_2 \tilde{\mathbf{H}}_k)^T = \begin{bmatrix} \mathbf{s}_k^T \mathbf{Q}_k \tilde{\mathbf{H}}_k \\ \mathbf{y}_k^T \tilde{\mathbf{H}}_k \end{bmatrix} = \begin{bmatrix} \mathbf{s}_k & \tilde{\mathbf{H}}_k \mathbf{y}_k \end{bmatrix},\ \mathbf{M} = \left(\mathbf{I} + \mathbf{V}_k^T \tilde{\mathbf{H}}_k \mathbf{V}_k\right)^{-1} \mathbf{V}_1$$

$$\tilde{\mathbf{H}}_{k+1} = \tilde{\mathbf{H}}_k - \mathbf{V}_k \mathbf{M} \mathbf{V}_k^T$$

$$\mathbf{M} = \left(\mathbf{I} + \mathbf{V}_k^T \tilde{\mathbf{H}}_k^{-1}\right) \mathbf{V}_1 = \left(\mathbf{V}_1^{-1} + \mathbf{V}_2 \tilde{\mathbf{H}}_k \mathbf{V}_2^T\right)^{-1}$$

$$\mathbf{M}^{-1} = \mathbf{V}_1^{-1} + \mathbf{V}_2 \tilde{\mathbf{H}}_k \mathbf{V}_2^T = \begin{bmatrix} \frac{-1}{\mathbf{s}_k^T \mathbf{Q}_k \mathbf{s}_k} & 0 \\ 0 & \frac{1}{\mathbf{y}_k^T \mathbf{s}_k} \end{bmatrix}^{-1} + \begin{bmatrix} \mathbf{s}_k^T \mathbf{Q}_k \\ \mathbf{y}_k^T \end{bmatrix} \tilde{\mathbf{H}}_k \begin{bmatrix} \mathbf{Q}_k \mathbf{s}_k & \mathbf{y}_k \end{bmatrix}$$

$$= \begin{bmatrix} -\mathbf{s}_k^T \mathbf{Q}_k \mathbf{s}_k & 0 \\ 0 & \mathbf{y}_k^T \mathbf{s}_k \end{bmatrix} + \begin{bmatrix} \mathbf{s}_k^T \mathbf{Q}_k \\ \mathbf{y}_k^T \end{bmatrix} \begin{bmatrix} \mathbf{s}_k & \tilde{\mathbf{H}}_k \mathbf{y}_k \end{bmatrix} = \begin{bmatrix} 0 & \mathbf{s}_k^T \mathbf{y}_k \\ \mathbf{y}_k^T \mathbf{s}_k & \mathbf{y}_k^T \mathbf{s}_k + \mathbf{y}_k^T \tilde{\mathbf{H}}_k \mathbf{y}_k \end{bmatrix}$$

Broyden-Fletcher-Goldfarb-Shanno (BFGS) proof

$$\mathbf{M} = \frac{-1}{\mathbf{y}_k^T\mathbf{s}_k\mathbf{s}_k^T\mathbf{y}_k}\begin{bmatrix}\mathbf{y}_k^T\mathbf{s}_k + \mathbf{y}_k^T\tilde{\mathbf{H}}_k\mathbf{y}_k & -\mathbf{s}_k^T\mathbf{y}_k \\ -\mathbf{y}_k^T\mathbf{s}_k & 0\end{bmatrix} = -\rho\begin{bmatrix}1+\rho\mathbf{y}_k^T\tilde{\mathbf{H}}_k\mathbf{y}_k & -1 \\ -1 & 0\end{bmatrix}, \rho = \frac{1}{\mathbf{s}_k^T\mathbf{y}_k}$$

$$\begin{aligned}\tilde{\mathbf{H}}_{k+1} &= \tilde{\mathbf{H}}_k - \mathbf{V}_k^T\mathbf{M}\mathbf{V}_k^T = \tilde{\mathbf{H}}_k + \rho_k\begin{bmatrix}\mathbf{s}_k & \tilde{\mathbf{H}}_k\mathbf{y}_k\end{bmatrix}\begin{bmatrix}1+\rho_k\mathbf{y}_k^T\tilde{\mathbf{H}}_k\mathbf{y}_k & -1 \\ -1 & 0\end{bmatrix}\begin{bmatrix}\mathbf{s}_k^T \\ \mathbf{y}_k^T\tilde{\mathbf{H}}_k\end{bmatrix} \\ &= \tilde{\mathbf{H}}_k + \rho_k\left(\mathbf{s}_k\mathbf{s}_k^T + \rho_k\mathbf{s}_k\mathbf{y}_k^T\tilde{\mathbf{H}}_k\mathbf{y}_k\mathbf{s}_k^T - \tilde{\mathbf{H}}_k\mathbf{y}_k\mathbf{s}_k^T - \mathbf{s}_k\mathbf{y}_k^T\tilde{\mathbf{H}}_k\right) \\ &= \left(\mathbf{I} - \rho_k\mathbf{s}_k\mathbf{y}_k^T\right)\tilde{\mathbf{Q}}_k\left(\mathbf{I} - \rho_k\mathbf{y}_k\mathbf{s}_k^T\right) + \rho_k\mathbf{s}_k\mathbf{s}_k^T,\end{aligned}$$

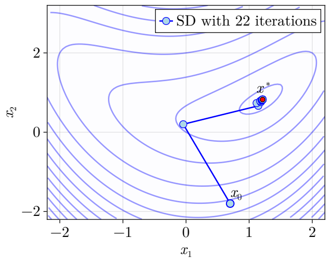
where $\rho_k^2 = \frac{1}{\mathbf{y}_k^T\mathbf{s}_k\mathbf{s}_k^T\mathbf{y}_k}$

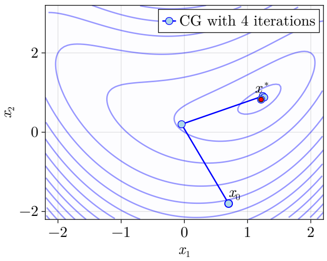
Broyden-Fletcher-Goldfarb-Shanno (BFGS)

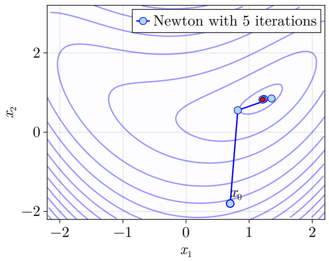
```
Require: $\mathbf{x}_0$, $\varepsilon_G$, $f$, $\nabla f_k$
  $k = 0, \mathbf{Q} = I$
  while $\|\nabla f(\mathbf{x}_k)\| > \varepsilon_G$ && $k \leq k_{\max}$ do
      $\mathbf{g} = \nabla f(\mathbf{x}_k)$
      $\mathbf{x}' = $ line_search($f, \mathbf{x}, -\mathbf{Q} * \mathbf{g}$)
      $\mathbf{g}' = \nabla f(\mathbf{x}')$
      $\mathbf{s} = \mathbf{x}' - \mathbf{x}_k$
      $\mathbf{y} = \mathbf{g}' - \mathbf{g}$
      $\sigma = 1/\mathbf{s}_k^T \mathbf{y}_k$
      $\mathbf{Q} = \left(I - \sigma_k \mathbf{s}_k \mathbf{y}_k^T\right) \tilde{\mathbf{Q}}_k \left(I - \sigma_k \mathbf{y}_k \mathbf{s}_k^T\right) + \sigma_k \mathbf{s}_k \mathbf{s}_k^T$
      $k = k + 1$
      $\mathbf{x}_k = \mathbf{x}'$
  end while
  return $\mathbf{x}'$
```

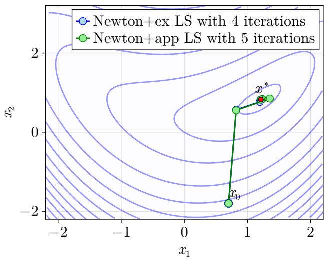
Compare Four Methods

Minimizing of bean function $f(x_1, x_2) = (1 - x_1)^2 + (1 - x_2)^2 + 0.5(2x_2 - x_1^2)^2$



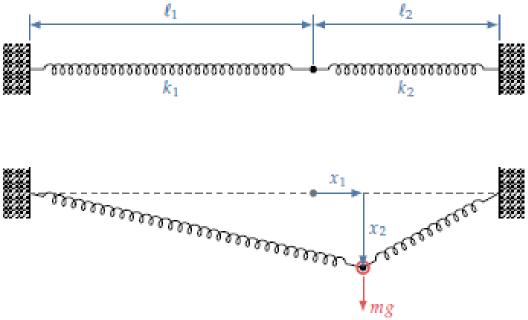






Compare Four Methods

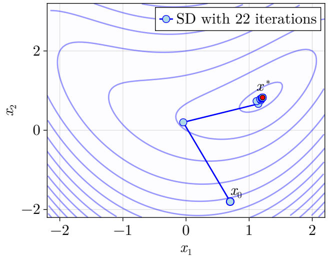
Minimizing the total potential energy for a spring system:

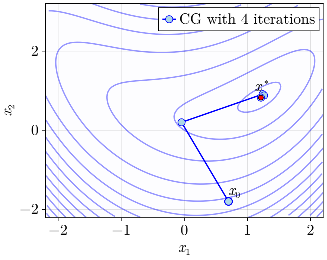


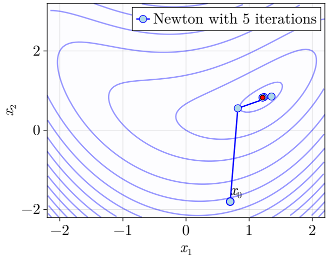
$$\underset{x_1,x_2}{\text{minimize}} \quad \frac{1}{2}k_1\left(\sqrt{(l_1+x_1)^2+x_2^2}-l_1\right)^2+\frac{1}{2}k_2\left(\sqrt{(l_2-x_1)^2+x_2^2}-l_2\right)^2-mgx_2$$

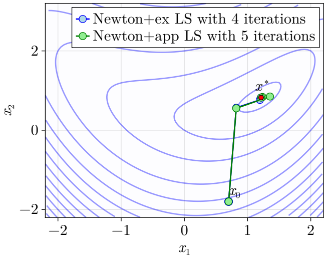
By letting $l_1=12, l_2=8$, $k_1=1$, $k_2=10$, $mg=7$ (with appropriate units).

Compare Four Methods









Limited-Memory Quasi-Newton Methods

- When the problem is large whose Hessian matrices cannot be computed at a reasonable cost.
- Instead of storing fully dense $n \times n$ approximations, we can save only a few vectors of length n that represent the approximations implicitly.
- Here we introduce the limited-Memory BFGS or L-BFGS.
- The BFGS method has the form

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \mathbf{Q}_k \nabla f_k,\ \mathbf{Q}_{k+1} = \mathbf{V}_k^T \mathbf{Q}_k \mathbf{V}_k + \rho_k \mathbf{s}_k \mathbf{s}_k^T,\ \text{where}$$

$$\rho_k = \frac{1}{\mathbf{y}_k^T \mathbf{s}_k},\ \mathbf{V}_k = \mathbf{I} - \rho_k \mathbf{y}_k \mathbf{s}_k^T,\ \text{and } \mathbf{s}_k = \mathbf{x}_{k+1} - \mathbf{x}_k,\ \mathbf{y}_k = \nabla f_{k+1} - \nabla f_k$$

- The inverse Hessian approximation $\mathbf{Q}_k$ will generally be dense, the cost of storing and manipulating it is prohibitive when the number of variables is large.
- To solve this problem, we store a *modified* version of $\mathbf{Q}_k$ implicitly, by storing a certain number m of the vector pairs $\{\mathbf{s}_i, \mathbf{y}_i\}$. The product $\mathbf{Q}_k \nabla f_k$ can be obtained by performing a sequence of inner products and vector summations involving ∇f_k and the pairs $\{\mathbf{s}_i, \mathbf{y}_i\}$.

Limited-Memory Quasi-Newton Methods

Recall and expand the BFGS update:

$$\begin{aligned}
\mathbf{Q}_k &= \mathbf{V}_{k-1}^T \mathbf{Q}_{k-1} \mathbf{V}_{k-1} + \rho_{k-1} \mathbf{s}_{k-1} \mathbf{s}_{k-1}^T \\
&= \mathbf{V}_{k-1}^T \mathbf{V}_{k-2}^T \mathbf{Q}_{k-2} \mathbf{V}_{k-2} \mathbf{V}_{k-1} + \rho_{k-2} \mathbf{V}_{k-2} \mathbf{s}_{k-2} \mathbf{s}_{k-2}^T \mathbf{V}_{k-1} + \rho_{k-1} \mathbf{s}_{k-1} \mathbf{s}_{k-1}^T \\
&= \left(\mathbf{V}_{k-1}^T \mathbf{V}_{k-2}^T \cdots \mathbf{V}_{k-m}^T \right) \mathbf{Q}_{k-m} \left(\mathbf{V}_{k-m} \mathbf{V}_{k-m+1} \cdots \mathbf{V}_{k-1} \right) \\
&\quad + \rho_{k-m} \left(\mathbf{V}_{k-1}^T \cdots \mathbf{V}_{k-m+1}^T \right) \mathbf{s}_{k-m} \mathbf{s}_{k-m}^T \left(\mathbf{V}_{k-m+1} \cdots \mathbf{V}_{k-1} \right) \\
&\quad + \rho_{k-m+1} \left(\mathbf{V}_{k-1}^T \cdots \mathbf{V}_{k-m+2}^T \right) \mathbf{s}_{k-m+1} \mathbf{s}_{k-m+1}^T \left(\mathbf{V}_{k-m+2} \cdots \mathbf{V}_{k-1} \right) \\
&\quad + \cdots \\
&\quad + \rho_{k-2} \mathbf{V}_{k-1}^T \mathbf{s}_{k-2} \mathbf{s}_{k-2}^T \mathbf{V}_{k-1} + \rho_{k-1} \mathbf{s}_{k-1} \mathbf{s}_{k-1}^T .
\end{aligned}$$

In L-BFGS, we replace $\mathbf{Q}_{k-m}$ (a dense $d \times d$ matrix) with some sparse matrix $\mathbf{Q}_k^0$, e.g., a diagonal matrix. Thus, $\mathbf{Q}_k$ can be constructe using the most recent $m \ll d$ pairs $\{\mathbf{s}_i, \mathbf{y}_i\}_{i=k-m}^{k-1}$. That is

Limited-Memory Quasi-Newton Methods

$$
\begin{aligned}
\mathbf{Q}_k = &\left(\mathbf{V}_{k-1}^T\mathbf{V}_{k-2}^T\cdots\mathbf{V}_{k-m}^T\right)\mathbf{Q}_k^0\left(\mathbf{V}_{k-m}\mathbf{V}_{k-m+1}\cdots\mathbf{V}_{k-1}\right)\\
&+\rho_{k-m}\left(\mathbf{V}_{k-1}^T\cdots\mathbf{V}_{k-m+1}^T\right)\mathbf{s}_{k-m}\mathbf{s}_{k-m}^T\left(\mathbf{V}_{k-m+1}\cdots\mathbf{V}_{k-1}\right)\\
&+\rho_{k-m+1}\left(\mathbf{V}_{k-1}^T\cdots\mathbf{V}_{k-m+2}^T\right)\mathbf{s}_{k-m+1}\mathbf{s}_{k-m+1}^T\left(\mathbf{V}_{k-m+2}\cdots\mathbf{V}_{k-1}\right)\\
&+\cdots\\
&+\rho_{k-2}\mathbf{V}_{k-1}^T\mathbf{s}_{k-2}\mathbf{s}_{k-2}^T\mathbf{V}_{k-1}+\rho_{k-1}\mathbf{s}_{k-1}\mathbf{s}_{k-1}^T.
\end{aligned}
$$

- We only need the d-dimensional vector $\mathbf{Q}_k\nabla f(\mathbf{x}_k)$ to update $\mathbf{x}_{k+1}=\mathbf{x}_k-\alpha_k\mathbf{Q}_k\nabla f(\mathbf{x}_k)$.
- We only stor the vectors $\{\mathbf{s}_i,\mathbf{y}_i\}_{i=k-m}^{k-1}$ from which $\mathbf{Q}_k\nabla f(\mathbf{x}_k)$ can be computed using only vector-vector multiplications.
- A popular choice for $\mathbf{Q}_k^0$ is $\mathbf{Q}_k^0=\gamma_k\mathbf{I}$, where $\gamma_k=\frac{\mathbf{s}_{k-1}^T\mathbf{y}_{k-1}}{\mathbf{y}_{k-1}^T\mathbf{y}_{k-1}}$. This choice appears to be quite effective in practice.

Limited-Memory Quasi-Newton Methods

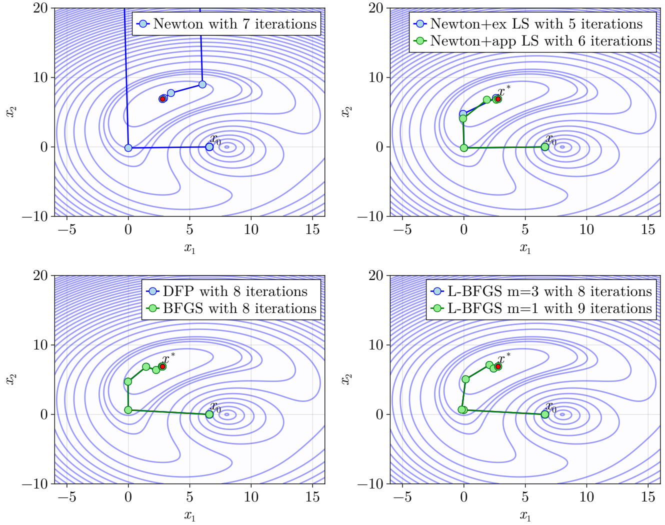
Algorithm 1 L-BFGS two-loop recursion

```
set $\mathbf{q} = \nabla f(\mathbf{x}_k)$ want to compute $\mathbf{Q}_k \nabla f(\mathbf{x})$
for $i = k-1, k-2, \ldots, k = m$ do
    $\alpha_i = \rho_i \mathbf{s}_i^T \mathbf{q}$
    $\mathbf{q} = \mathbf{q} - \alpha_i \mathbf{y}_i$ // RHS = $\mathbf{q} - \rho_i \mathbf{s}_i^T \mathbf{q} \mathbf{y}_i = \underbrace{\left(\mathbf{I} - \rho_i \mathbf{y}_i \mathbf{s}_i^T\right)}_{\mathbf{V}_i} \mathbf{q}$
end for
$\mathbf{r} = \mathbf{Q}_k^0 \mathbf{q}$
for $i = k-m$ to $k-1$ : do
    $\beta = \rho_i \mathbf{y}_i^T \mathbf{r}$
    $\mathbf{r} = \mathbf{r} + \mathbf{s}_i(\alpha_i - \beta)$
     // RHS = $\mathbf{r} + \rho_i \alpha_i - \rho_i \mathbf{y}_i^T \mathbf{r} \mathbf{s}_i = \underbrace{\left(\mathbf{I} - \rho_i \mathbf{s}_i \mathbf{y}_i^T\right) \mathbf{r}}_{\mathbf{V}_i^T} + \rho_i \alpha_i$
end for
return $\mathbf{r}$ // which equals $\mathbf{Q}_k \nabla f(\mathbf{x}_k)$
```

Limited-Memory Quasi-Newton Methods

```
Require: $\mathbf{x}_0$, $m$, $\varepsilon_G$
  $k = 0$
  while $\|\nabla f(\mathbf{x}_k)\| \leq \varepsilon_G$ do
      Choose $\mathbf{Q}_k^0$
      $\mathbf{p}_k = -\mathbf{Q}_k \nabla f(\mathbf{x}_k)$, where $\mathbf{Q}_k \nabla f(\mathbf{x}_k)$ is compute using Algorithm 1
      $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$, where $\alpha_k$ satisfies Wolfe Conditions
      if $k > m$: then
          discard $\{\mathbf{s}_{k-m}, \mathbf{y}_{k-m}\}$ from storage
      end if
      Compute and store $\mathbf{s}_k = \mathbf{x}_{k+1} - \mathbf{x}_k$ and $\mathbf{y}_k = \nabla f(\mathbf{x}_{k+1} - \nabla f(\mathbf{x}_k)$
      $k = k + 1$
  end while
```

Compare Four Methods



Reference

1. Joaquim R. R. A. Martins, and Andrew Ning, ”*Engineering Design Optimization*,” Cambridge University Press, 2021.
2. Jorge Nocedal, and Stephen J. Wright, ”*Numerical Optimization*,” 2nd, Springer, 2026