Unconstrained Optimization Overview

Notation

In what follows, we will work with vectors in $\mathbb{R}^n$.

The vectors will be (usually) denoted by $x \in \mathbb{R}^n$.

We often consider sequences of vectors, $x_0, x_1, \ldots, x_k, \ldots$.
The index k will usually indicate that x_k is the k-the vector in a sequence.

When we talk (relatively rarely) about components of vectors, we use i as an index, i.e., x_i will be the i-th component of $x \in \mathbb{R}^n$.

We denote by $||x||$ the Euclidean norm of x.

We denote by $||x||_\infty$ the $\mathcal{L}^\infty$ norm giving the maximum of absolute values of components of x.

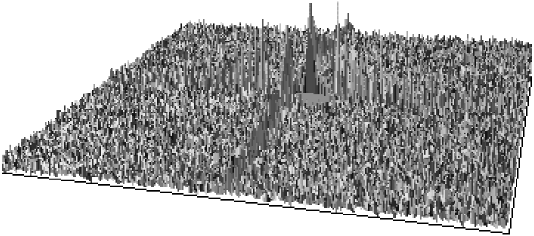
We ocasionally use the matrix morn $||A||$, consistent with the Euclidean norm, defined by

$$||A|| = \sup_{||x||=1} ||Ax|| = \sqrt{\lambda_1}$$

Here λ_1 is the largest eigenvalue of $A^\top A$.

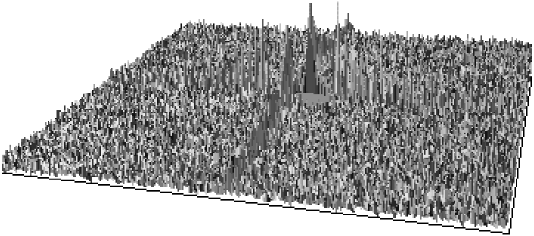
How to Recognize (Local) Minimum

How do we verify that $x^* \in \mathbb{R}^n$ is a minimizer of f?



How to Recognize (Local) Minimum

How do we verify that $x^* \in \mathbb{R}^n$ is a minimizer of f?



Technically, we should examine *all* points in the immediate vicinity if one has a smaller value (impractical).

Assuming the smoothness of f, we may benefit from the "stable" behavior of f around x^*.

Derivatives and Gradients

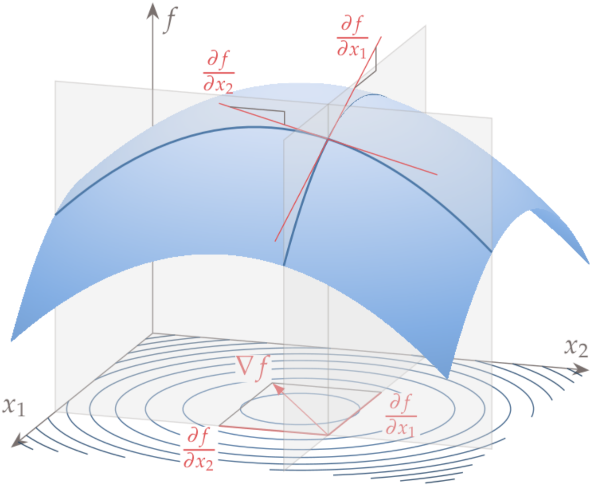
The gradient of $f : \mathbb{R}^n \to \mathbb{R}$, denoted by $\nabla f(x)$, is a column vector of first-order partial derivatives of the function concerning each variable:

$$\nabla f(x) = \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n}\right]^\top,$$

Where each partial derivative is defined as the following limit:

$$\frac{\partial f}{\partial x_i} = \lim_{\varepsilon \to 0} \frac{f\left(x_1, \ldots, x_i + \varepsilon, \ldots, x_n\right) - f\left(x_1, \ldots, x_i, \ldots, x_n\right)}{\varepsilon}$$

Gradient



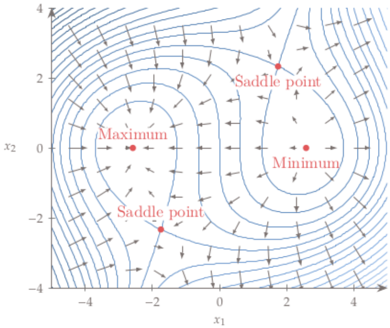
The gradient is a vector pointing in the direction of the most significant function increase from the current point.

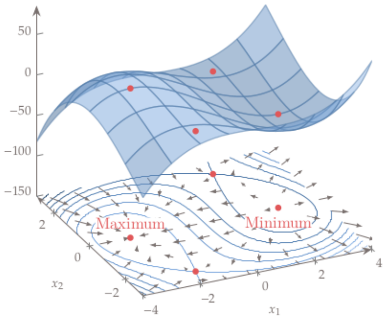
Gradient

Consider the following function of two variables:

$$f\left(x_1, x_2\right) = x_1^3 + 2x_1x_2^2 - x_2^3 - 20x_1.$$

$$\nabla f\left(x_1, x_2\right) = \left[\begin{array}{c} 3x_1^2 + 2x_2^2 - 20 \\ 4x_1x_2 - 3x_2^2 \end{array}\right]$$



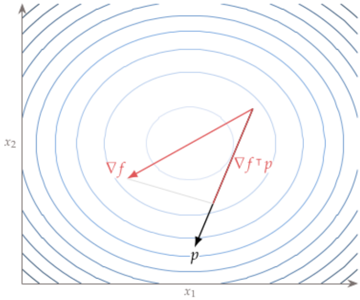


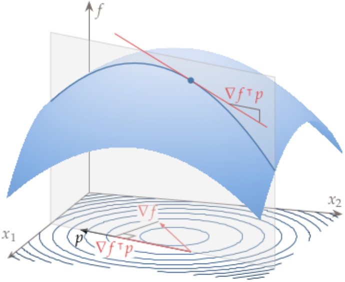
Directional Derivatives vs Gradient

The rate of change in a direction p is quantified by a directional derivative, defined as

$$\nabla_p f(x) = \lim_{\varepsilon \to 0} \frac{f(x + \varepsilon p) - f(x)}{\varepsilon}.$$

We can find this derivative by projecting the gradient onto the desired direction p using the dot product $\nabla_p f(x) = (\nabla f(x))^\top p$





(Here, we assume continuous partial derivatives.)

Geometry of Gradient

Consider the geometric interpretation of the dot product:

$$\nabla_p f(x) = (\nabla f(x))^\top p = ||\nabla f|| \, ||p|| \cos\theta$$

Here θ is the angle between ∇f and p.

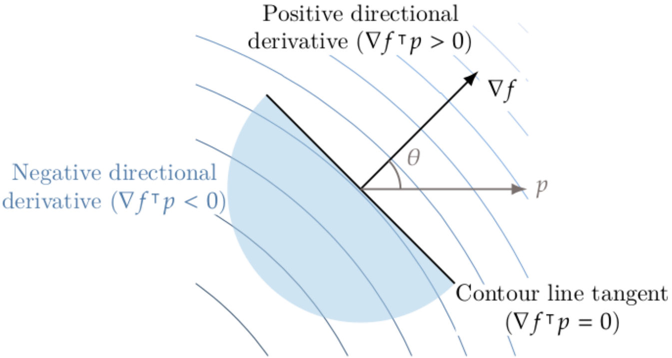
Geometry of Gradient

Consider the geometric interpretation of the dot product:

$$\nabla_p f(x) = (\nabla f(x))^\top p = ||\nabla f||\,||p||\cos\theta$$

Here θ is the angle between ∇f and p.

The directional derivative is maximized by $\theta = 0$, i.e. when ∇f and p point in the same direction.



Hessian

Taking derivative twice, possibly w.r.t. different variables, gives the *Hessian* of f

$$\nabla^2 f(x) = H(x) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}.$$

Note that the Hessian is a function which takes $x \in \mathbb{R}^n$ and gives a $n \times n$-matrix of second derivatives of f.

Hessian

Taking derivative twice, possibly w.r.t. different variables, gives the *Hessian* of f

$$\nabla^2 f(x) = H(x) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}.$$

Note that the Hessian is a function which takes $x \in \mathbb{R}^n$ and gives a $n \times n$-matrix of second derivatives of f.

We have

$$H_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j}.$$

If f has continuous second partial derivatives, then H is symmetric, i.e., $H_{ij} = H_{ji}$.

Geometry of Hessian

Let x be fixed and let $g(t) = f(x + tp)$ and let $h_i(t) = \frac{\partial f}{\partial x_i}(x + tp)$ for $t \in \mathbb{R}$.

What exactly are $g'(0)$ and $g''(0)$?

Geometry of Hessian

Let x be fixed and let $g(t) = f(x + tp)$ and let $h_i(t) = \frac{\partial f}{\partial x_i}(x + tp)$ for $t \in \mathbb{R}$.

What exactly are $g'(0)$ and $g''(0)$?

$$g'(t) = f(x + tp)' = [\nabla f(x + tp)]^\top p = \sum_{i=1}^{n} h_i(t) p_i$$

Geometry of Hessian

Let x be fixed and let $g(t) = f(x+tp)$ and let $h_i(t) = \frac{\partial f}{\partial x_i}(x+tp)$ for $t \in \mathbb{R}$.

What exactly are $g'(0)$ and $g''(0)$?

$$g'(t) = f(x+tp)' = [\nabla f(x+tp)]^\top p = \sum_{i=1}^{n} h_i(t) p_i$$

$$\begin{aligned} h_i'(t) &= \left[\nabla \frac{\partial f}{\partial x_i}(x+tp)\right]^\top p = \sum_{j=1}^{n} \left(\frac{\partial f}{\partial x_i \partial x_j}(x+tp)\right) p_j \\ &= [H(x+tp)p]_i \end{aligned}$$

Geometry of Hessian

Let x be fixed and let $g(t) = f(x + tp)$ and let $h_i(t) = \frac{\partial f}{\partial x_i}(x + tp)$ for $t \in \mathbb{R}$.

What exactly are $g'(0)$ and $g''(0)$?

$$g'(t) = f(x + tp)' = [\nabla f(x + tp)]^\top p = \sum_{i=1}^{n} h_i(t) p_i$$

$$h_i'(t) = \left[\nabla \frac{\partial f}{\partial x_i}(x + tp)\right]^\top p = \sum_{j=1}^{n} \left(\frac{\partial f}{\partial x_i \partial x_j}(x + tp)\right) p_j$$

$$= [H(x + tp)p]_i$$

$$g''(t) = \sum_{i=1}^{n} h_i'(t) p_i = \sum_{i=1}^{n} [H(x + tp)p]_i p_i = p^\top H(x + tp) p$$

Geometry of Hessian

Let x be fixed and let $g(t) = f(x + tp)$ and let $h_i(t) = \frac{\partial f}{\partial x_i}(x + tp)$ for $t \in \mathbb{R}$.

What exactly are $g'(0)$ and $g''(0)$?

$$g'(t) = f(x + tp)' = [\nabla f(x + tp)]^\top p = \sum_{i=1}^{n} h_i(t) p_i$$

$$\begin{aligned} h_i'(t) &= \left[\nabla \frac{\partial f}{\partial x_i}(x + tp)\right]^\top p = \sum_{j=1}^{n} \left(\frac{\partial f}{\partial x_i \partial x_j}(x + tp)\right) p_j \\ &= [H(x + tp)p]_i \end{aligned}$$

$$g''(t) = \sum_{i=1}^{n} h_i'(t) p_i = \sum_{i=1}^{n} [H(x + tp)p]_i p_i = p^\top H(x + tp)p$$

Thus,

$$g''(0) = p^\top H(x)p.$$

Principal Curvature Directions

Fix x and consider $H = H(x)$. Consider unit eigenvectors $\hat{v}_k$ of H:

$$H\hat{v}_k = \kappa_k \hat{v}_k$$

For symmetric H, the unit eigenvectors form an orthonormal basis,

Principal Curvature Directions

Fix x and consider $H = H(x)$. Consider unit eigenvectors $\hat{v}_k$ of H:

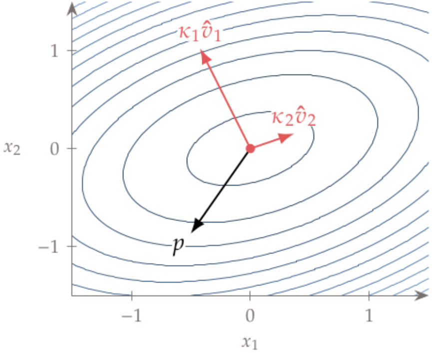
$$H\hat{v}_k = \kappa_k \hat{v}_k$$

For symmetric H, the unit eigenvectors form an orthonormal basis, and there is a rotation matrix R such that

$$H = RDR^{-1} = RDR^{\top}$$

Here D is diagonal with $\kappa_1, \ldots, \kappa_n$ on the diagonal.

If $\kappa_1 \geq \cdots \geq \kappa_n$, the direction of $\hat{v}_1$ is the maximum curvature direction of f at x.



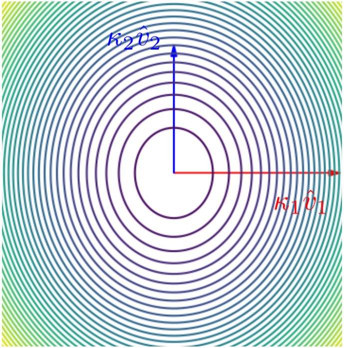
Consider $f(x) = x^\top Hx$ where

$$H = \begin{pmatrix} 4/3 & 0 \\ 0 & 1 \end{pmatrix}$$

The eigenvalues are

$$\kappa_1 = 4/3 \quad \kappa_2 = 1$$

Their corresponding eigenvectors are $(1, 0)^\top$ and $(0, 1)^\top$.



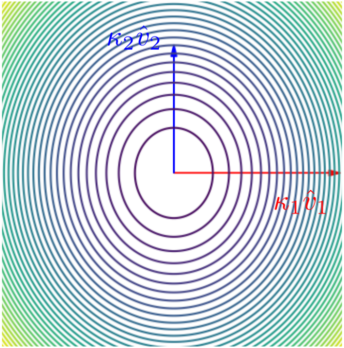
Consider $f(x) = x^\top H x$ where

$$H = \begin{pmatrix} 4/3 & 0 \\ 0 & 1 \end{pmatrix}$$

The eigenvalues are

$$\kappa_1 = 4/3 \quad \kappa_2 = 1$$

Their corresponding eigenvectors are $(1, 0)^\top$ and $(0, 1)^\top$.



Note that

$$f(x) = \kappa_1 x_1^2 + \kappa_2 x_2^2$$

Considering a direction vector p we get

$$g(t) = f(0 + tp) = t^2 \left(\kappa_1 p_1^2 + \kappa_2 p_2^2\right)$$

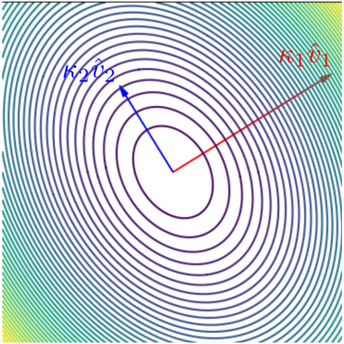
which is a parabola with $g'' = 2\left(\kappa_1 p_1^2 + \kappa_2 p_2^2\right)$.

Consider $f(x) = x^\top H x$ where

$$H = \begin{pmatrix} 4/3 & 1/3 \\ 1/3 & 3/3 \end{pmatrix}$$

Consider $f(x) = x^\top H x$ where

$$H = \begin{pmatrix} 4/3 & 1/3 \\ 1/3 & 3/3 \end{pmatrix}$$



The eigenvalues are

$$\kappa_1 = \frac{1}{6}(7+\sqrt{5}) \quad \kappa_2 = \frac{1}{6}(7-\sqrt{5})$$

Their corresponding eigenvectors are

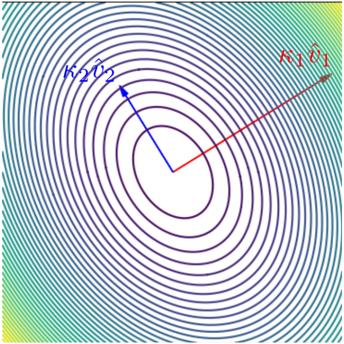
$$\hat{v}_1 = \left(\frac{1}{2}(1+\sqrt{5}), 1\right) \quad \hat{v}_2 = \left(\frac{1}{2}(1-\sqrt{5}), 1\right)$$

Consider $f(x) = x^\top H x$ where

$$H = \begin{pmatrix} 4/3 & 1/3 \\ 1/3 & 3/3 \end{pmatrix}$$

The eigenvalues are

$$\kappa_1 = \frac{1}{6}(7+\sqrt{5}) \quad \kappa_2 = \frac{1}{6}(7-\sqrt{5})$$



Their corresponding eigenvectors are

$$\hat{v}_1 = \left(\frac{1}{2}(1+\sqrt{5}), 1\right) \quad \hat{v}_2 = \left(\frac{1}{2}(1-\sqrt{5}), 1\right)$$

Note that

$$H = (\hat{v}_1 \ \hat{v}_2) \begin{pmatrix} \kappa_1 & 0 \\ 0 & \kappa_2 \end{pmatrix} (\hat{v}_1 \ \hat{v}_2)^\top$$

Here $(\hat{v}_1 \ \hat{v}_2)$ is a 2×2 matrix whose columns are $\hat{v}_1, \hat{v}_2$.

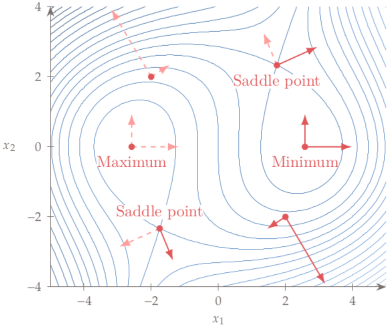
Hessian Visualization Example

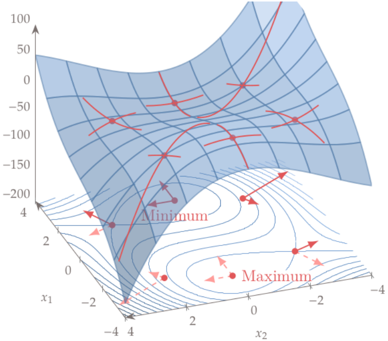
Consider

$$f(x_1, x_2) = x_1^3 + 2x_1x_2^2 - x_2^3 - 20x_1.$$

And it's Hessian.

$$H(x_1, x_2) = \begin{bmatrix} 6x_1 & 4x_2 \\ 4x_2 & 4x_1 - 6x_2 \end{bmatrix}.$$





Taylor's Theorem

Theorem 1 (Taylor)

Suppose that $f : \mathbb{R}^n \to \mathbb{R}$ is twice continuously differentiable and that $p \in \mathbb{R}^n$. Then, we have

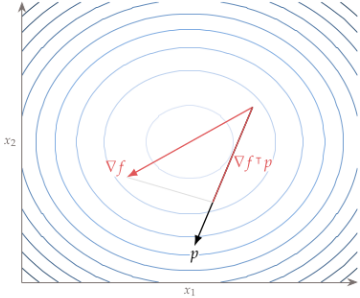
$$f(x+p) = f(x) + \nabla f(x)^T p + \frac{1}{2} p^T H(x) p + o(||p||^2).$$

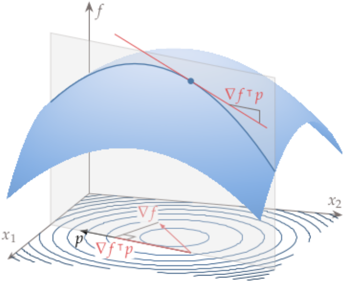
Here $H = \nabla^2 f$ is the Hessian of f.

First-Order Necessary Conditions

Theorem 2

If x^* is a local minimizer and f is continuously differentiable in an open neighborhood of x^*, then $\nabla f(x^*) = 0$.





Second-Order Conditions

Note that $\nabla f(x^*) = 0$ does not tell us whether x^* is a minimizer, maximizer, or a saddle point.

Second-Order Conditions

Note that $\nabla f(x^*) = 0$ does not tell us whether x^* is a minimizer, maximizer, or a saddle point.

However, knowing the curvature in all directions from x^* might tell us what x^* is, right?

Second-Order Conditions

Note that $\nabla f(x^*) = 0$ does not tell us whether x^* is a minimizer, maximizer, or a saddle point.

However, knowing the curvature in all directions from x^* might tell us what x^* is, right?

Second-Order Conditions

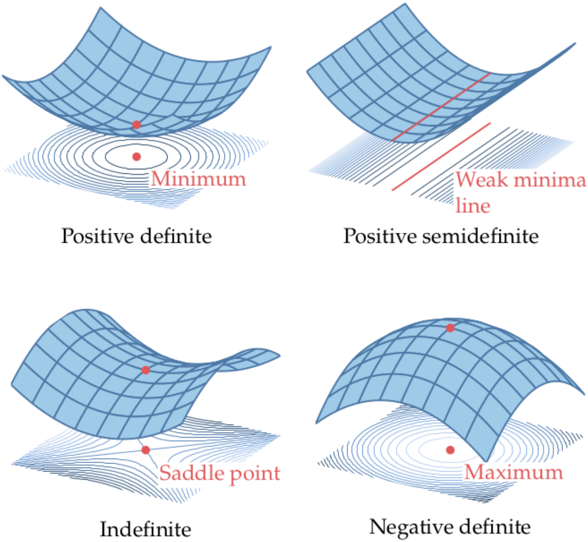
Note that $\nabla f(x^*) = 0$ does not tell us whether x^* is a minimizer, maximizer, or a saddle point.

However, knowing the curvature in all directions from x^* might tell us what x^* is, right?

All comes down to the *definiteness* of $H := H(x^*)$.

- H is positive definite if $p^\top H p > 0$ for all p
  iff all eigenvalues of H are positive
- H is positive semi-definite if $p^\top H p \geq 0$ for all p
  iff all eigenvalues of H are nonnegative
- H is negative semi-definite if $p^\top H p \leq 0$ for all p
  iff all eigenvalues of H are nonpositive
- H is negative definite if $p^\top H p < 0$ for all p
  iff all eigenvalues of H are negative
- H is indefinite if it is not definite in the above sense
  iff H has at least one positive and one negative eigenvalue.

Definiteness



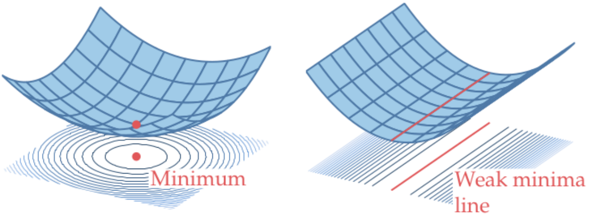
Second-Order Necessary Condition

Theorem 3 (Second-Order Necessary Conditions)

If x^* is a local minimizer of f and $\nabla^2 f$ is continuous in a neighborhood of x^*, then $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is positive semidefinite.

Theorem 4 (Second-Order Sufficient Conditions)

Suppose that $\nabla^2 f$ is continuous in a neighborhood of x^* and that $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is positive definite. Then x^* is a strict local minimizer of f.



Positive definite   Positive semidefinite

Example

Consider the following function of two variables:

$$f(x_1, x_2) = 0.5x_1^4 + 2x_1^3 + 1.5x_1^2 + x_2^2 - 2x_1x_2.$$

Example

Consider the following function of two variables:

$$f(x_1, x_2) = 0.5x_1^4 + 2x_1^3 + 1.5x_1^2 + x_2^2 - 2x_1x_2.$$

Consider the gradient equal to zero:

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 2x_1^3 + 6x_1^2 + 3x_1 - 2x_2 \\ 2x_2 - 2x_1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Example

Consider the following function of two variables:

$$f(x_1, x_2) = 0.5x_1^4 + 2x_1^3 + 1.5x_1^2 + x_2^2 - 2x_1x_2.$$

Consider the gradient equal to zero:

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 2x_1^3 + 6x_1^2 + 3x_1 - 2x_2 \\ 2x_2 - 2x_1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

From the second equation, we have that $x_2 = x_1$. Substituting this into the first equation yields

$$x_1 \left(2x_1^2 + 6x_1 + 1\right) = 0.$$

Example

Consider the following function of two variables:

$$f(x_1, x_2) = 0.5x_1^4 + 2x_1^3 + 1.5x_1^2 + x_2^2 - 2x_1x_2.$$

Consider the gradient equal to zero:

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 2x_1^3 + 6x_1^2 + 3x_1 - 2x_2 \\ 2x_2 - 2x_1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

From the second equation, we have that $x_2 = x_1$. Substituting this into the first equation yields

$$x_1 \left(2x_1^2 + 6x_1 + 1\right) = 0.$$

The solution of this equation yields three points:

$$x_A = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad x_B = \begin{bmatrix} -\frac{3}{2} - \frac{\sqrt{7}}{2} \\ -\frac{3}{2} - \frac{\sqrt{7}}{2} \end{bmatrix}, \quad x_C = \begin{bmatrix} \frac{\sqrt{7}}{2} - \frac{3}{2} \\ \frac{\sqrt{7}}{2} - \frac{3}{2} \end{bmatrix}.$$

Example

Consider the following function of two variables:

$$f(x_1, x_2) = 0.5x_1^4 + 2x_1^3 + 1.5x_1^2 + x_2^2 - 2x_1x_2.$$

Example

Consider the following function of two variables:

$$f(x_1, x_2) = 0.5x_1^4 + 2x_1^3 + 1.5x_1^2 + x_2^2 - 2x_1x_2.$$

To classify x_A, x_B, x_C, we need to compute the Hessian matrix:

$$H(x_1, x_2) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} \end{bmatrix} = \begin{bmatrix} 6x_1^2 + 12x_1 + 3 & -2 \\ -2 & 2 \end{bmatrix}.$$

Example

Consider the following function of two variables:

$$f(x_1, x_2) = 0.5x_1^4 + 2x_1^3 + 1.5x_1^2 + x_2^2 - 2x_1x_2.$$

To classify x_A, x_B, x_C, we need to compute the Hessian matrix:

$$H(x_1, x_2) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} \end{bmatrix} = \begin{bmatrix} 6x_1^2 + 12x_1 + 3 & -2 \\ -2 & 2 \end{bmatrix}.$$

The Hessian, at the first point, is

$$H(x_A) = \begin{bmatrix} 3 & -2 \\ -2 & 2 \end{bmatrix},$$

whose eigenvalues are $\kappa_1 \approx 0.438$ and $\kappa_2 \approx 4.561$. Because both eigenvalues are positive, this point is a local minimum.

Example

Consider the following function of two variables:

$$f(x_1, x_2) = 0.5x_1^4 + 2x_1^3 + 1.5x_1^2 + x_2^2 - 2x_1x_2.$$

To classify x_A, x_B, x_C, we need to compute the Hessian matrix:

$$H(x_1, x_2) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} \end{bmatrix} = \begin{bmatrix} 6x_1^2 + 12x_1 + 3 & -2 \\ -2 & 2 \end{bmatrix}.$$

For the second point,

$$H(x_B) = \begin{bmatrix} 3(3+\sqrt{7}) & -2 \\ -2 & 2 \end{bmatrix}.$$

The eigenvalues are $\kappa_1 \approx 1.737$ and $\kappa_2 \approx 17.200$, so this point is another local minimum.

Example

Consider the following function of two variables:

$$f(x_1, x_2) = 0.5x_1^4 + 2x_1^3 + 1.5x_1^2 + x_2^2 - 2x_1x_2.$$

To classify x_A, x_B, x_C, we need to compute the Hessian matrix:

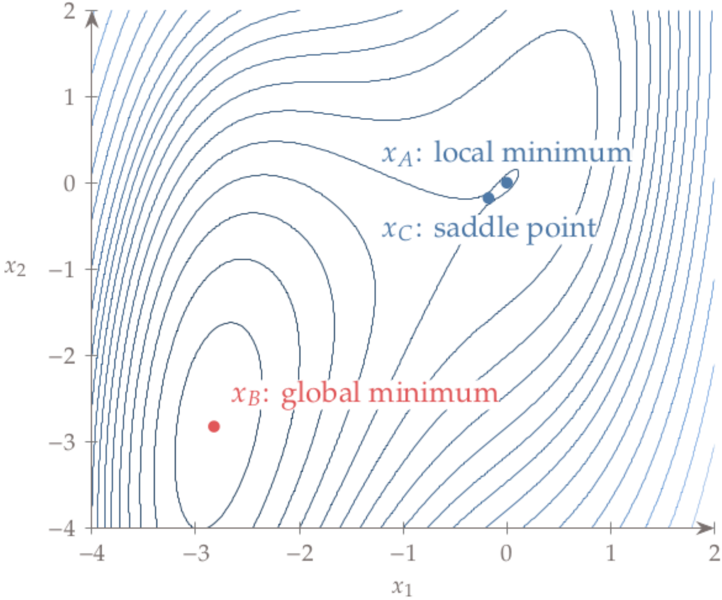
$$H(x_1, x_2) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} \end{bmatrix} = \begin{bmatrix} 6x_1^2 + 12x_1 + 3 & -2 \\ -2 & 2 \end{bmatrix}.$$

For the third point,

$$H(x_C) = \begin{bmatrix} 9 - 3\sqrt{7} & -2 \\ -2 & 2 \end{bmatrix}.$$

The eigenvalues for this Hessian are $\kappa_1 \approx -0.523$ and $\kappa_2 \approx 3.586$, so this point is a saddle point.

Example



2
1
0
x_2
−1
−2
−3
−4
−4
−3
−2
−1
0
1
2
x_1
x_A: local minimum
x_C: saddle point
x_B: global minimum

Proofs of Some Theorems

Optional

Taylor's Theorem

To prove the theorems characterizing minima/maxima, we need the following form of Taylor's theorem:

Theorem 5 (Taylor)

Suppose that $f : \mathbb{R}^n \to \mathbb{R}$ is continuously differentiable and that $p \in \mathbb{R}^n$. Then we have that.

$$f(x + p) = f(x) + \nabla f(x + tp)^T p,$$

for some $t \in (0, 1)$. Moreover, if f is twice continuously differentiable, we have that

$$f(x + p) = f(x) + \nabla f(x)^T p + \frac{1}{2} p^T \nabla^2 f(x + tp) p,$$

for some $t \in (0, 1)$.

Proof of Theorem 2 (Optional)

We prove that if x^* is a local minimizer and f is continuously differentiable in an open neighborhood of x^*, then $\nabla f(x^*) = 0$.

Suppose for contradiction that $\nabla f(x^*) \neq 0$. Define the vector $p = -\nabla f(x^*)$ and note that $p^T \nabla f(x^*) = -\|\nabla f(x^*)\|^2 < 0$. Because ∇f is continuous near x^*, there is a scalar $T > 0$ such that

$$p^T \nabla f(x^* + tp) < 0, \quad \text{for all } t \in [0, T]$$

For any $\bar{t} \in (0, T]$, we have by Taylor's theorem that

$$f(x^* + \bar{t}p) = f(x^*) + \bar{t}p^T \nabla f(x^* + tp), \quad \text{for some } t \in (0, \bar{t}).$$

Therefore, $f(x^* + \bar{t}p) < f(x^*)$ for all $\bar{t} \in (0, T]$. We have found a direction leading away from x^* along which f decreases, so x^* is not a local minimizer, and we have a contradiction. $\square$

Proof of Theorem 3 (Optional)

We prove that if x^* is a local minimizer of f and $\nabla^2 f$ is continuous in an open neighborhood of x^*, then $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is positive semidefinite.

We know that $\nabla f(x^*) = 0$. For contradiction, assume that $\nabla^2 f(x^*)$ is not positive semidefinite.

Then we can choose a vector p such that $p^T \nabla^2 f(x^*)\, p < 0$.

As $\nabla^2 f$ is continuous near x^*, $p^T \nabla^2 f(x^* + tp)\, p < 0$ for all $t \in [0, T]$ where $T > 0$.

By Taylor we have for all $\bar{t} \in (0, T]$ and some $t \in (0, \bar{t})$

$$f(x^* + \bar{t}p) = f(x^*) + \bar{t}p^T \nabla f(x^*) + \frac{1}{2}\bar{t}^2 p^T \nabla^2 f(x^* + tp)\, p < f(x^*).$$

Thus, x^* is not a local minimizer. $\square$

Proof of Theorem 4 (Optional)

We prove the following: Suppose that $\nabla^2 f$ is continuous in an open neighborhood of x^* and that $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is positive definite. Then x^* is a strict local minimizer of f.

Because the Hessian is continuous and positive definite at x^*, we can choose a radius $r > 0$ so that $\nabla^2 f(x)$ remains positive definite for all x in the open ball $\mathcal{D} = \{z \mid \|z - x^*\| < r\}$. Taking any nonzero vector p with $\|p\| < r$, we have $x^* + p \in \mathcal{D}$ and so

$$
\begin{aligned}
f(x^* + p) &= f(x^*) + p^T \nabla f(x^*) + \frac{1}{2} p^T \nabla^2 f(z) p \\
&= f(x^*) + \frac{1}{2} p^T \nabla^2 f(z) p,
\end{aligned}
$$

where $z = x^* + tp$ for some $t \in (0, 1)$. Since $z \in \mathcal{D}$, we have $p^T \nabla^2 f(z) p > 0$, and therefore $f(x^* + p) > f(x^*)$, giving the result. $\square$

Unconstrained Optimization Algorithms

Search Algorithms

We consider algorithms that

- Start with an initial guess x_0
- Generate a sequence of points $x_0, x_1, \ldots$
- Stop when no progress can be made or when a minimizer seems approximated with sufficient accuracy.

To compute x_{k+1} the algorithms use the information about f at the previous iterates $x_0, x_1, \ldots, x_k$.

Search Algorithms

We consider algorithms that

- Start with an initial guess x_0
- Generate a sequence of points $x_0, x_1, \ldots$
- Stop when no progress can be made or when a minimizer seems approximated with sufficient accuracy.

To compute x_{k+1} the algorithms use the information about f at the previous iterates $x_0, x_1, \ldots, x_k$.

The *monotone* algorithms satisfy $f(x_{k+1}) < f(x_k)$.

Search Algorithms

We consider algorithms that

- Start with an initial guess x_0
- Generate a sequence of points $x_0, x_1, \ldots$
- Stop when no progress can be made or when a minimizer seems approximated with sufficient accuracy.

To compute x_{k+1} the algorithms use the information about f at the previous iterates $x_0, x_1, \ldots, x_k$.

The *monotone* algorithms satisfy $f(x_{k+1}) < f(x_k)$.

There are two overall strategies:

- Line search
- Trust region

Line Search Overview

To compute x_{k+1}, a line search algorithm chooses

- *direction* p_k
- *step size* α_k

and computes

$$x_{k+1} = x_k + \alpha_k p_k$$

Line Search Overview

To compute x_{k+1}, a line search algorithm chooses

- *direction* p_k
- *step size* α_k

and computes

$$x_{k+1} = x_k + \alpha_k p_k$$

The vector p_k should be a *descent* direction, i.e., a direction in which f decreases locally.

Line Search Overview

To compute x_{k+1}, a line search algorithm chooses

- *direction* p_k
- *step size* α_k

and computes

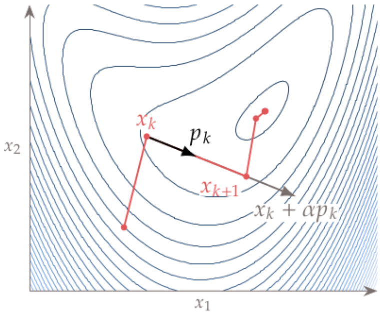
$$x_{k+1} = x_k + \alpha_k p_k$$

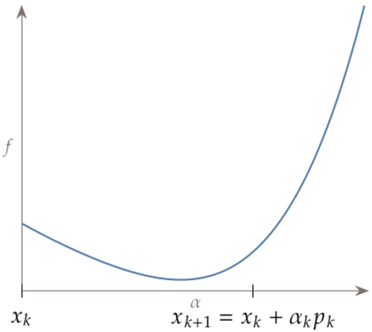
The vector p_k should be a *descent* direction, i.e., a direction in which f decreases locally.

α_k is selected to approximately solve

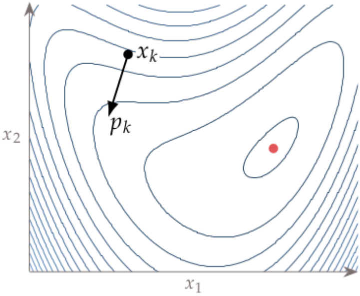
$$\min_{\alpha>0} f(x_k + \alpha p_k)$$

However, typically, an exact solution is expensive and unnecessary. Instead, line search algorithms inspect a limited number of trial step lengths and find one that decreases f appropriately (see later).





A descent direction does not have to be followed to the minimum.



Trust Region

To compute x_{k+1}, a trust region algorithm chooses

- *model function* m_k whose behavior near x_k is similar to f
- a *trust region* $R \subseteq \mathbb{R}^n$ around x_k. Usually R is the ball defined by $||x - x_k|| \leq \Delta$ where $\Delta > 0$ is *trust region radius*.

and finds x_{k+1} solving

$$\min_{x \in R} m_k(x)$$

Trust Region

To compute x_{k+1}, a trust region algorithm chooses

- *model function* m_k whose behavior near x_k is similar to f
- a *trust region* $R \subseteq \mathbb{R}^n$ around x_k. Usually R is the ball defined by $||x - x_k|| \leq \Delta$ where $\Delta > 0$ is *trust region radius*.

and finds x_{k+1} solving

$$\min_{x \in R} m_k(x)$$

If the solution does not sufficiently decrease f, we shrink the trust region and re-solve.

Trust Region

To compute x_{k+1}, a trust region algorithm chooses

- *model function* m_k whose behavior near x_k is similar to f
- a *trust region* $R \subseteq \mathbb{R}^n$ around x_k. Usually R is the ball defined by $||x - x_k|| \leq \Delta$ where $\Delta > 0$ is *trust region radius*.

and finds x_{k+1} solving

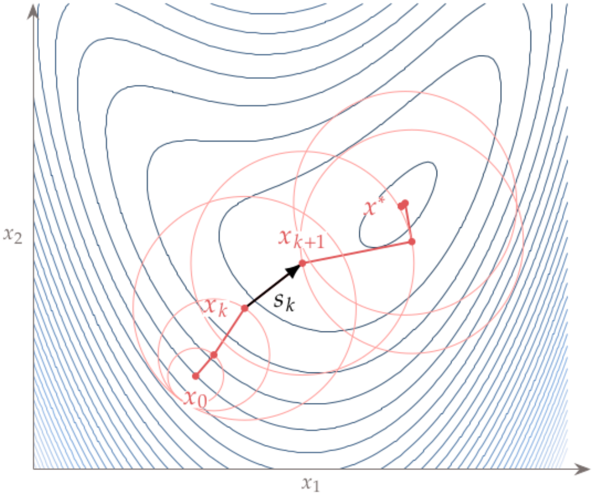
$$\min_{x \in R} m_k(x)$$

If the solution does not sufficiently decrease f, we shrink the trust region and re-solve.

The model m_k is usually derived from the Taylor's theorem.

$$m_k\left(x_k + p\right) = f_k + p^T \nabla f_k + \frac{1}{2} p^T B_k p$$

Where B_k approximates the Hessian of f at x_k.


x_2
x_1
x^*
x_{k+1}
s_k
x_k
x_0

Line Search Methods

Line Search

For setting the step size, we consider

- Armijo condition and backtracking algorithm
- strong Wolfe conditions and bracketing & zooming

Line Search

For setting the step size, we consider

- Armijo condition and backtracking algorithm
- strong Wolfe conditions and bracketing & zooming

For setting the direction, we consider

- Gradient descent
- Newton's method
- quasi-Newton methods (BFGS)
- (Conjugate gradients)

We start with the step size.

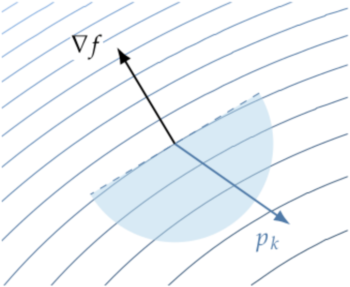
Step Size

Assume

$$x_{k+1} = x_k + \alpha_k p_k$$

Where p_k is a descent direction

$$p_k^\top \nabla f_k < 0$$



Step Size

Assume

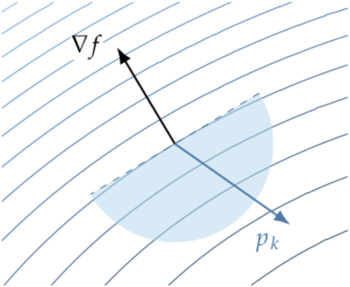
$$x_{k+1} = x_k + \alpha_k p_k$$

Where p_k is a descent direction

$$p_k^\top \nabla f_k < 0$$

Define

$$\phi(\alpha) = f(x_k + \alpha p_k)$$



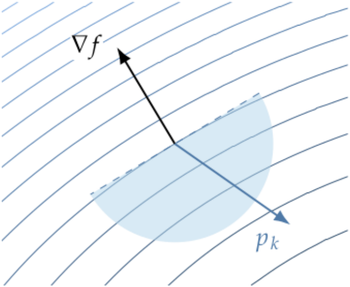
Step Size

Assume

$$x_{k+1} = x_k + \alpha_k p_k$$

Where p_k is a descent direction

$$p_k^\top \nabla f_k < 0$$



Define

$$\phi(\alpha) = f(x_k + \alpha p_k)$$

We know that

$$\phi'(\alpha) = \nabla f(x_k + \alpha p_k)^\top p_k \quad \text{which means} \quad \phi'(0) = \nabla f_k^\top p_k$$

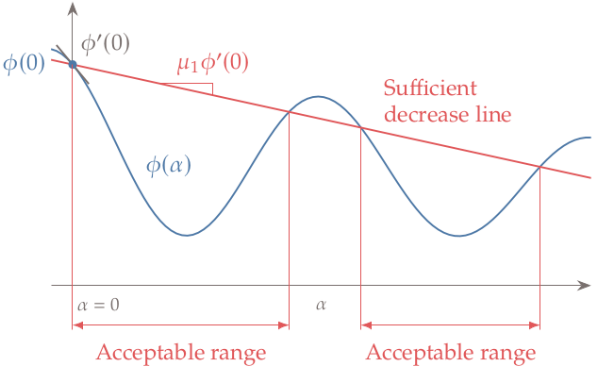
Note that $\phi'(0)$ must be negative as p_k is a descent direction.

Armijo Condition

The *sufficient decrease condition* (aka *Armijo condition*)

$$\phi(\alpha) \leq \phi(0) + \alpha \left(\mu_1 \phi'(0)\right)$$

where μ_1 is a constant such that $0 < \mu_1 \leq 1$



In practice, μ_1 is several orders smaller than 1, typically $\mu_1 = 10^{-4}$.

Backtracking Line Search Algorithm

Algorithm 1 Backtracking Line Search

Input: $\alpha_{\text{init}} > 0$, $0 < \mu_1 < 1$, $0 < \rho < 1$

Output: α^* satisfying sufficient decrease condition

1: $\alpha \leftarrow \alpha_{\text{init}}$
2: **while** $\phi(\alpha) > \phi(0) + \alpha\mu_1\phi'(0)$ **do**
3: $\quad \alpha \leftarrow \rho\alpha$
4: **end while**

The parameter ρ is typically set to 0.5. It can also be a variable set by a more sophisticated method (interpolation).

The α_{init} depends on the method for setting the descent direction p_k. For Newton and quasi-Newton, it is 1.0, but for other methods, it might be different.

Issues with Backtracking

There are two scenarios where the method does not perform well:

Issues with Backtracking

There are two scenarios where the method does not perform well:

- The guess for the initial step is far too large, and the step sizes that satisfy sufficient decrease are smaller than the starting step by several orders of magnitude. Depending on the value of ρ, this scenario requires many backtracking evaluations.

Issues with Backtracking

There are two scenarios where the method does not perform well:

- The guess for the initial step is far too large, and the step sizes that satisfy sufficient decrease are smaller than the starting step by several orders of magnitude. Depending on the value of ρ, this scenario requires many backtracking evaluations.
- The guess for the initial step immediately satisfies sufficient decrease. However, the function's slope is still highly negative, and we could have decreased the function value by much more if we had taken a more significant step. In this case, our guess for the initial step is far too small.

Issues with Backtracking

There are two scenarios where the method does not perform well:

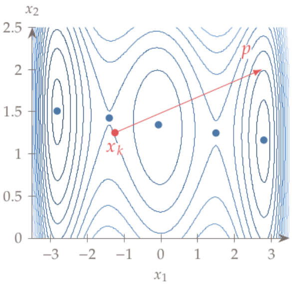
- The guess for the initial step is far too large, and the step sizes that satisfy sufficient decrease are smaller than the starting step by several orders of magnitude. Depending on the value of ρ, this scenario requires many backtracking evaluations.
- The guess for the initial step immediately satisfies sufficient decrease. However, the function's slope is still highly negative, and we could have decreased the function value by much more if we had taken a more significant step. In this case, our guess for the initial step is far too small.

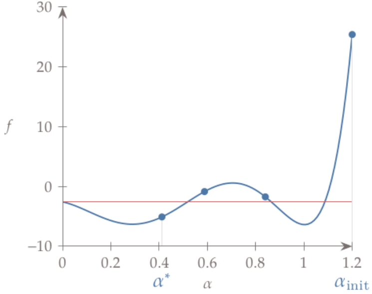
Even if our original step size is not too far from an acceptable one, the basic backtracking algorithm ignores any information we have about the function values and gradients. It blindly takes a reduced step based on a preselected ratio ρ.

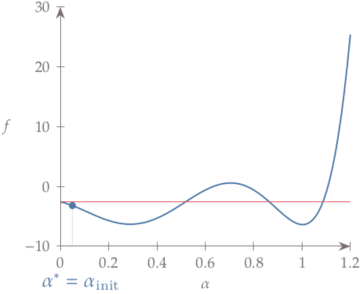
Backtracking Example

$$f(x_1, x_2) = 0.1x_1^6 - 1.5x_1^4 + 5x_1^2 + 0.1x_2^4 + 3x_2^2 - 9x_2 + 0.5x_1x_2$$

$\mu_1 = 10^{-4}$ and $\rho = 0.7$.







Sufficient Curvature Condition

We want to prevent too short of steps and to "motivate" the search to move closer to the minimum.

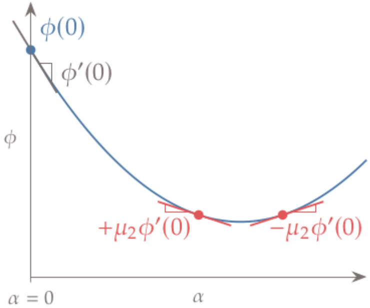
Sufficient Curvature Condition

We want to prevent too short of steps and to "motivate" the search to move closer to the minimum.

We introduce the *sufficient curvature condition*

$$\left|\phi'(\alpha)\right| \leq \mu_2 \left|\phi'(0)\right|$$

where $\mu_1 < \mu_2 < 1$ is a constant.



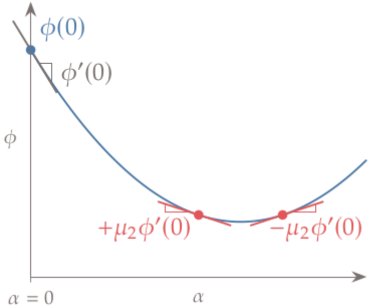
Sufficient Curvature Condition

We want to prevent too short of steps and to "motivate" the search to move closer to the minimum.

We introduce the *sufficient curvature condition*

$$\left|\phi'(\alpha)\right| \leq \mu_2 \left|\phi'(0)\right|$$

where $\mu_1 < \mu_2 < 1$ is a constant.



Typical values of μ_2 range from 0.1 to 0.9, depending on the direction setting method.

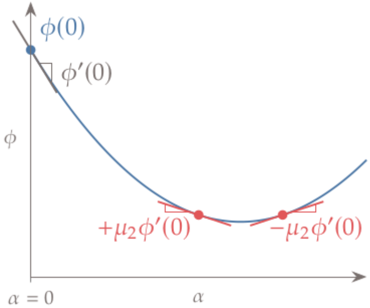
Sufficient Curvature Condition

We want to prevent too short of steps and to "motivate" the search to move closer to the minimum.

We introduce the *sufficient curvature condition*

$$\left|\phi'(\alpha)\right| \leq \mu_2 \left|\phi'(0)\right|$$

where $\mu_1 < \mu_2 < 1$ is a constant.



Typical values of μ_2 range from 0.1 to 0.9, depending on the direction setting method.

As μ_2 tends to 0, the condition enforces $\phi'(\alpha) = 0$, which would yield an exact line search.

Strong Wolfe Conditions

Putting together Armijo and sufficient curvature conditions, we obtain *strong Wolfe conditions*

Strong Wolfe Conditions

Putting together Armijo and sufficient curvature conditions, we obtain *strong Wolfe conditions*

- *Sufficient decrease condition*

$$\phi(\alpha) \leq \phi(0) + \mu_1 \alpha \phi'(0)$$

Strong Wolfe Conditions

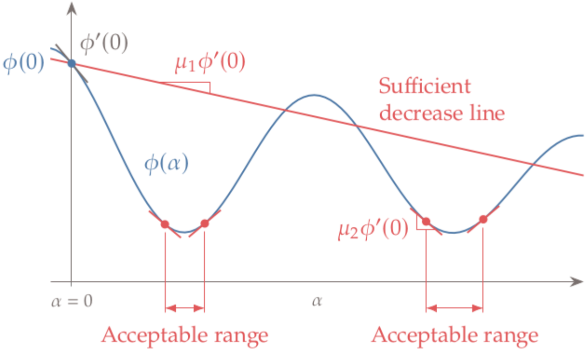
Putting together Armijo and sufficient curvature conditions, we obtain *strong Wolfe conditions*

- *Sufficient decrease condition*

$$\phi(\alpha) \le \phi(0) + \mu_1 \alpha \phi'(0)$$

- *Sufficient curvature condition*

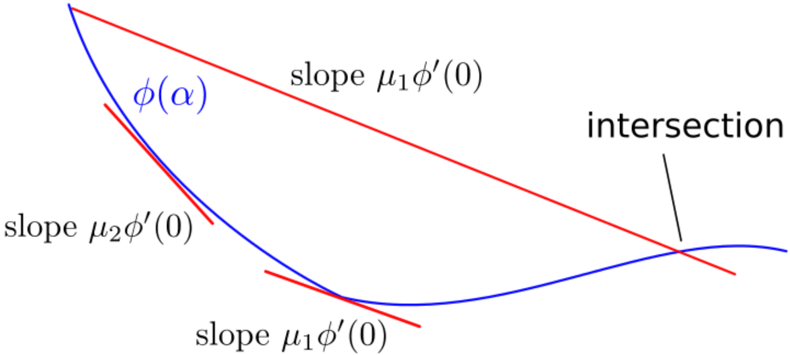
$$\left|\phi'(\alpha)\right| \le \mu_2 \left|\phi'(0)\right|$$



Satisfiability of Strong Wolfe Conditions

Theorem 6

Suppose $f : \mathbb{R}^n \to \mathbb{R}$ is continuously differentiable. Let p_k be a descent direction at x_k, and assume that f is bounded below along the ray $\{x_k + \alpha p_k \mid \alpha > 0\}$. Then, if $0 < \mu_1 < \mu_2 < 1$, step length intervals exist that satisfy the strong Wolfe conditions.



Convergence of Line Search

Denote by θ_k the angle between p_k and $-\nabla f_k$, i.e., satisfying

$$\cos\theta_k = \frac{-\nabla f_k^T p_k}{\|\nabla f_k\| \, \|p_k\|}$$

Convergence of Line Search

Denote by θ_k the angle between p_k and $-\nabla f_k$, i.e., satisfying

$$\cos\theta_k = \frac{-\nabla f_k^T p_k}{\|\nabla f_k\| \, \|p_k\|}$$

Recall that f is L-smooth for some $L > 0$ if

$$\|\nabla f(x) - \nabla f(\tilde{x})\| \leq L\|x - \tilde{x}\|, \qquad \text{for all } x, \tilde{x} \in \mathbb{R}^n$$

Convergence of Line Search

Denote by θ_k the angle between p_k and $-\nabla f_k$, i.e., satisfying

$$\cos\theta_k = \frac{-\nabla f_k^T p_k}{\|\nabla f_k\| \, \|p_k\|}$$

Recall that f is L-smooth for some $L > 0$ if

$$\|\nabla f(x) - \nabla f(\tilde{x})\| \leq L\|x - \tilde{x}\|, \qquad \text{for all } x, \tilde{x} \in \mathbb{R}^n$$

Theorem 7 (Zoutendijk)

Consider $x_{k+1} = x_k + \alpha_k p_k$, where p_k is a descent direction *and α_k satisfies the* strong Wolfe conditions*. Suppose that f is bounded below, continuously differentiable, and* L-smooth*. Then*

$$\sum_{k \geq 0} \cos^2\theta_k \, \|\nabla f_k\|^2 < \infty.$$

Line Search Algorithm

How can we find a step size that satisfies *strong* Wolfe conditions?

Line Search Algorithm

How can we find a step size that satisfies *strong* Wolfe conditions?

Use a bracketing and zoom algorithm, which proceeds in the following two phases:

Line Search Algorithm

How can we find a step size that satisfies *strong* Wolfe conditions?

Use a bracketing and zoom algorithm, which proceeds in the following two phases:

1. The bracketing phase finds an interval within which we are certain to find a point that satisfies the strong Wolfe conditions.

Line Search Algorithm

How can we find a step size that satisfies *strong* Wolfe conditions?

Use a bracketing and zoom algorithm, which proceeds in the following two phases:

1. The bracketing phase finds an interval within which we are certain to find a point that satisfies the strong Wolfe conditions.
2. The zooming phase finds a point that satisfies the strong Wolfe conditions within the interval provided by the bracketing phase.

Algorithm 2 Bracketing

Input: $\alpha_1 > 0$ and $\alpha_{\max}$

Set $\alpha_0 \leftarrow 0$
$i \leftarrow 1$
repeat
  Evaluate $\phi(\alpha_i)$
  if $\phi(\alpha_i) > \phi(0) + \alpha_i \mu_1 \phi'(0)$ or $[\phi(\alpha_i) \geq \phi(\alpha_{i-1})$ and $i > 1]$ **then**
    $\alpha^* \leftarrow$ **zoom**(α_{i-1}, α_i) and stop
  end if
  Evaluate $\phi'(\alpha_i)$
  if $|\phi'(\alpha_i)| \leq \mu_2 |\phi'(0)|$ **then**
    set $\alpha^* \leftarrow \alpha_i$ and stop
  else if $\phi'(\alpha_i) \geq 0$ **then**
    set $\alpha^* \leftarrow$ **zoom**(α_i, α_{i-1}) and stop
  end if
  Choose $\alpha_{i+1} \in (\alpha_i, \alpha_{\max})$
  $i \leftarrow i + 1$
until a condition is met

Explanation of Bracketing

Note that the sequence of trial steps α_i is monotonically increasing.

Explanation of Bracketing

Note that the sequence of trial steps α_i is monotonically increasing.

Note that **zoom** is called when one of the following conditions is satisfied:

- α_i violates the sufficient decrease condition (lines 5 and 6)
- $\phi(\alpha_i) \geq \phi(\alpha_{i-1})$ (also lines 5 and 6)
- $\phi'(\alpha_i) \geq 0$ (lines 11 and 12)

The last step increases the α_i. May use, e.g., a constant multiple.

Zoom

The following algorithm keeps two step lengths: α_{lo} and α_{hi}

Zoom

The following algorithm keeps two step lengths: α_{lo} and α_{hi}

The following invariants are being preserved:

- The interval bounded by α_{lo} and α_{hi} always contains one or more intervals satisfying the strong Wolfe conditions.

  Note that we *do not* assume $\alpha_{\text{lo}} \leq \alpha_{\text{hi}}$

Zoom

The following algorithm keeps two step lengths: α_{lo} and α_{hi}

The following invariants are being preserved:

- The interval bounded by α_{lo} and α_{hi} always contains one or more intervals satisfying the strong Wolfe conditions.

  Note that we *do not* assume $\alpha_{\mathsf{lo}} \leq \alpha_{\mathsf{hi}}$

- α_{lo} is, among all step lengths generated so far and satisfying the sufficient decrease condition, the one giving the smallest value of ϕ,

Zoom

The following algorithm keeps two step lengths: α_{lo} and α_{hi}

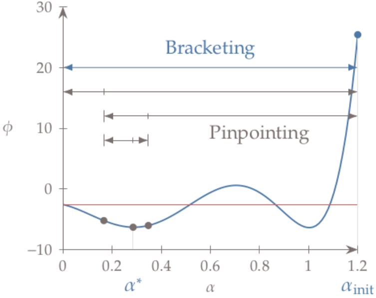
The following invariants are being preserved:

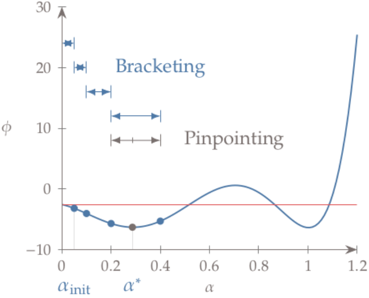
- The interval bounded by α_{lo} and α_{hi} always contains one or more intervals satisfying the strong Wolfe conditions.
  Note that we *do not* assume $\alpha_{\mathsf{lo}} \leq \alpha_{\mathsf{hi}}$
- α_{lo} is, among all step lengths generated so far and satisfying the sufficient decrease condition, the one giving the smallest value of ϕ,
- α_{hi} is chosen so that $\phi'(\alpha_{\mathsf{lo}})(\alpha_{\mathsf{hi}} - \alpha_{\mathsf{lo}}) < 0$.
  That is, ϕ always slopes down from α_{lo} to α_{hi}.

function ZOOM($\alpha_{\text{lo}}, \alpha_{\text{hi}}$)
    repeat
        Set α between α_{lo} and α_{hi} using interpolation
        (bisection, quadratic, etc.)
        Evaluate $\phi(\alpha)$
        if $\phi(\alpha) > \phi(0) + \alpha\mu_1\phi'(0)$ or $\phi(\alpha) \geq \phi(\alpha_{\text{lo}})$ **then**
            $\alpha_{\text{hi}} \leftarrow \alpha$
        else
            Evaluate $\phi'(\alpha)$
            if $|\phi'(\alpha)| \leq \mu_2|\phi'(0)|$ **then**
                Set $\alpha^* \leftarrow \alpha$ and stop
            end if
            if $\phi'(\alpha)(\alpha_{\text{hi}} - \alpha_{\text{lo}}) \geq 0$ **then**
                $\alpha_{\text{hi}} \leftarrow \alpha_{\text{lo}}$
            end if
            $\alpha_{\text{lo}} \leftarrow \alpha$
        end if
    until a condition is met
end function

Bracketing & Zooming Example

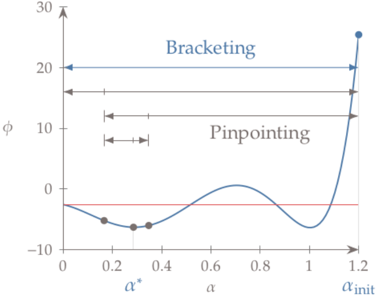
We use quadratic interpolation; the bracketing chooses $\alpha_{i+1} = 2\alpha_i$, and the sufficient curvature factor is $\mu_2 = 0.9$.





Bracketing & Zooming Example

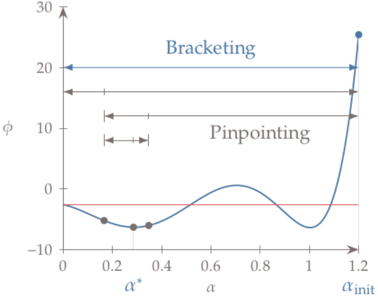
We use quadratic interpolation; the bracketing chooses $\alpha_{i+1} = 2\alpha_i$, and the sufficient curvature factor is $\mu_2 = 0.9$.



Bracketing is achieved in the first iteration by using a significant initial step of $\alpha_{\text{init}} = 1.2$ (left). Then, zooming finds an improved point through interpolation.

Bracketing & Zooming Example

We use quadratic interpolation; the bracketing chooses $\alpha_{i+1} = 2\alpha_i$, and the sufficient curvature factor is $\mu_2 = 0.9$.



Bracketing is achieved in the first iteration by using a significant initial step of $\alpha_{\text{init}} = 1.2$ (left). Then, zooming finds an improved point through interpolation.

The small initial step of $\alpha_{\text{init}} = 0.05$ (right) does not satisfy the strong Wolfe conditions, and the bracketing phase moves forward toward a flatter part of the function.

Comments on Line Search

- The interpolation of the zoom phase that determines α should be safeguarded to ensure that the new step length is not too close to the endpoints of the interval.

Comments on Line Search

- The interpolation of the zoom phase that determines α should be safeguarded to ensure that the new step length is not too close to the endpoints of the interval.
- Practical line search algorithms also use the interpolating polynomials' properties to make educated guesses of where the next step length should lie.

Comments on Line Search

- The interpolation of the zoom phase that determines α should be safeguarded to ensure that the new step length is not too close to the endpoints of the interval.
- Practical line search algorithms also use the interpolating polynomials' properties to make educated guesses of where the next step length should lie.
- A problem that can arise in the implementation is that as the optimization algorithm approaches the solution, two consecutive function values $f(x_k)$ and $f(x_{k-1})$ may be indistinguishable in finite-precision arithmetic.

Comments on Line Search

- The interpolation of the zoom phase that determines α should be safeguarded to ensure that the new step length is not too close to the endpoints of the interval.
- Practical line search algorithms also use the interpolating polynomials' properties to make educated guesses of where the next step length should lie.
- A problem that can arise in the implementation is that as the optimization algorithm approaches the solution, two consecutive function values $f(x_k)$ and $f(x_{k-1})$ may be indistinguishable in finite-precision arithmetic.
- Some procedures also stop if the relative change in x is close to machine accuracy or some user-specified threshold.

Comments on Line Search

- The interpolation of the zoom phase that determines α should be safeguarded to ensure that the new step length is not too close to the endpoints of the interval.
- Practical line search algorithms also use the interpolating polynomials' properties to make educated guesses of where the next step length should lie.
- A problem that can arise in the implementation is that as the optimization algorithm approaches the solution, two consecutive function values $f(x_k)$ and $f(x_{k-1})$ may be indistinguishable in finite-precision arithmetic.
- Some procedures also stop if the relative change in x is close to machine accuracy or some user-specified threshold.
- The presented algorithm is implemented in `https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.line_search.html`

Unconstrained Optimization Algorithms

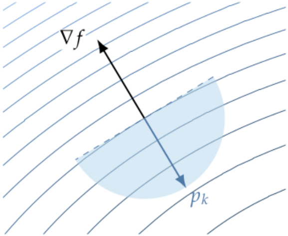
Descent Direction

First-Order Methods

Gradient Descent

Consider the *gradient descent* (aka *gradient descent*) method where

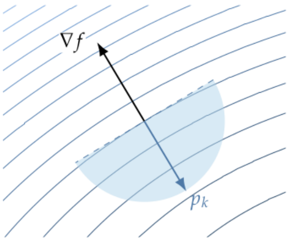
$$x_{k+1} = x_k + \alpha_k p_k \qquad p_k = -\nabla f(x_k)$$



Gradient Descent

Consider the *gradient descent* (aka *gradient descent*) method where

$$x_{k+1} = x_k + \alpha_k p_k \qquad p_k = -\nabla f(x_k)$$



Unfortunately, the gradient does not possess much information about the step size.

So usually, a normalized gradient is used to obtain the direction, and then a line search is performed:

$$x_{k+1} = x_k + \alpha_k p_k \qquad p_k = -\frac{\nabla f(x_k)}{||\nabla f(x_k)||}$$

The line search is *exact* if α_k minimizes $f(x_k + \alpha_k p_k)$. Not practical, we usually find α_k satisfying the strong Wolfe conditions.

Gradient Descent Algorithm with Line Search

Algorithm 3 Gradient Descent with Line Search

Input: x_0 starting point, $\varepsilon > 0$

Output: x^* approximation to a stationary point

```
k ← 0
while ‖∇f‖∞ > ε do
    p_k ← −∇f(x_k)/‖∇f(x_k)‖
    Set α_init for line search
    α_k ← linesearch(p_k, α_init)
    x_{k+1} ← x_k + α_k p_k
    k ← k + 1
end while
```

Gradient Descent Algorithm with Line Search

Algorithm 4 Gradient Descent with Line Search

Input: x_0 starting point, $\varepsilon > 0$

Output: x^* approximation to a stationary point

$k \leftarrow 0$
while $\|\nabla f\|_\infty > \varepsilon$ **do**
  $p_k \leftarrow -\frac{\nabla f(x_k)}{\|\nabla f(x_k)\|}$
  Set α_{init} for line search
  $\alpha_k \leftarrow \text{linesearch}(p_k, \alpha_{\text{init}})$
  $x_{k+1} \leftarrow x_k + \alpha_k p_k$
  $k \leftarrow k + 1$
end while

Here α_{init} can be estimated from the previous step size α_{k-1} by demanding similar decrease in the objective:

$$\alpha_{\text{init}} p_k^\top \nabla f_k \approx \alpha_{k-1} p_{k-1}^\top \nabla f_{k-1} \quad \Rightarrow \quad \alpha_{\text{init}} = \alpha_{k-1} \frac{p_{k-1}^\top \nabla f_{k-1}}{p_k^\top \nabla f_k}$$

Gradient Descent Algorithm with Line Search

Algorithm 5 Gradient Descent with Line Search

Input: x_0 starting point, $\varepsilon > 0$

Output: x^* approximation to a stationary point

```
k ← 0
while ‖∇f‖_∞ > ε do
    p_k ← −∇f(x_k) / ‖∇f(x_k)‖
    Set α_init for line search
    α_k ← linesearch(p_k, α_init)
    x_{k+1} ← x_k + α_k p_k
    k ← k + 1
end while
```

Gradient Descent Algorithm with Line Search

Algorithm 6 Gradient Descent with Line Search

Input: x_0 starting point, $\varepsilon > 0$

Output: x^* approximation to a stationary point

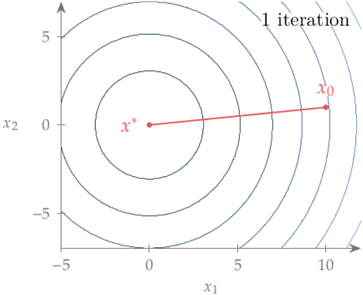
1: $k \leftarrow 0$
2: **while** $\|\nabla f\|_\infty > \varepsilon$ **do**
3: $\quad p_k \leftarrow -\frac{\nabla f(x_k)}{\|\nabla f(x_k)\|}$
4: $\quad$ Set α_{init} for line search
5: $\quad \alpha_k \leftarrow \text{linesearch}(p_k, \alpha_{\text{init}})$
6: $\quad x_{k+1} \leftarrow x_k + \alpha_k p_k$
7: $\quad k \leftarrow k + 1$
8: **end while**

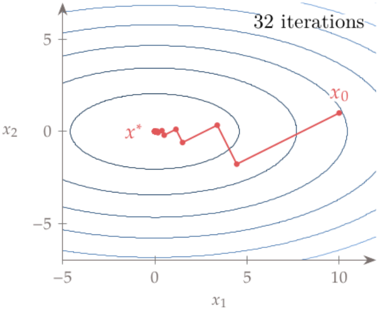
Here α_{init} can be estimated from the previous step size α_{k-1} by demanding similar decrease in the objective:

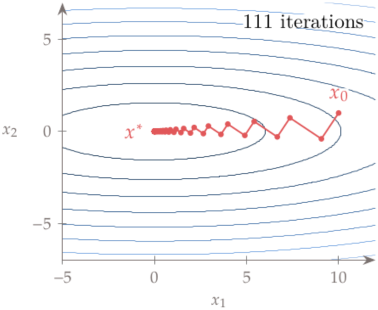
$$\alpha_{init} p_k^\top \nabla f_k^\top \approx \alpha_{k-1} p_{k-1}^\top \nabla f_{k-1}^\top \quad \Rightarrow \quad \alpha_{init} = \alpha_{k-1} \frac{\alpha_{k-1} p_{k-1}^\top \nabla f_{k-1}^\top}{\nabla p_k^\top f_k^\top}$$

$$f(x_1, x_2) = x_1^2 + \beta x_2^2$$

Consider $\beta = 1, 5, 15$ and *exact* line search



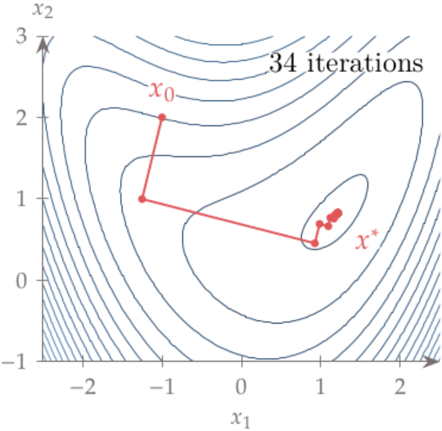




Note that p_{k+1} and p_k are always orthogonal.

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

Stopping: $\|\nabla f\|_\infty \leq 10^{-6}$.



The gradient descent can be prolonged.

Global Convergence with Line Search

Recall the Zoutendijk's theorem.

Denote by θ_k the angle between p_k and $-\nabla f_k$, i.e., satisfying

$$\cos\theta_k = \frac{-\nabla f_k^T p_k}{\|\nabla f_k\| \, \|p_k\|}$$

Recall that f is L-smooth on a set $\mathcal{N}$ for some $L > 0$ if

$$\|\nabla f(x) - \nabla f(\tilde{x})\| \leq L\|x - \tilde{x}\|, \qquad \text{for all } x, \tilde{x} \in \mathcal{N}$$

Theorem 8 (Zoutendijk)

Consider $x_{k+1} = x_k + \alpha_k p_k$, where p_k is a descent direction and α_k satisfies the strong Wolfe conditions. Suppose that f is bounded below in $\mathbb{R}^n$ and that f is continuously differentiable in an open set $\mathcal{N}$ containing the level set $\{x : f(x) \leq f(x_0)\}$. Assume also that f is L-smooth on $\mathcal{N}$. Then

$$\sum_{k\geq 0} \cos^2\theta_k \, \|\nabla f_k\|^2 < \infty.$$

Global Convergence of Gradient Descent

Assume that each α_k satisfies strong Wolfe conditions.

Global Convergence of Gradient Descent

Assume that each α_k satisfies strong Wolfe conditions.

Note that the angle θ_k between $p_k = -\nabla f_k$ and the negative gradient $-\nabla f_k$ equals 0. Hence, $\cos\theta_k = 1$.

Global Convergence of Gradient Descent

Assume that each α_k satisfies strong Wolfe conditions.

Note that the angle θ_k between $p_k = -\nabla f_k$ and the negative gradient $-\nabla f_k$ equals 0. Hence, $\cos\theta_k = 1$.

Thus, under the assumptions of Zoutendijk's theorem, we obtain

$$\sum_{k\geq 0} \cos^2\theta_k \, \|\nabla f_k\|^2 = \sum_{k\geq 0} \|\nabla f_k\|^2 < \infty$$

which implies that $\lim_{k\to\infty} ||\nabla f_k|| = 0$.

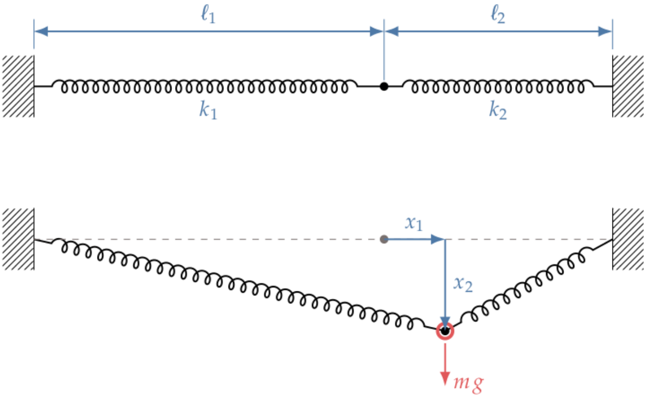
Local Linear Convergence of Gradient Descent

Theorem 9

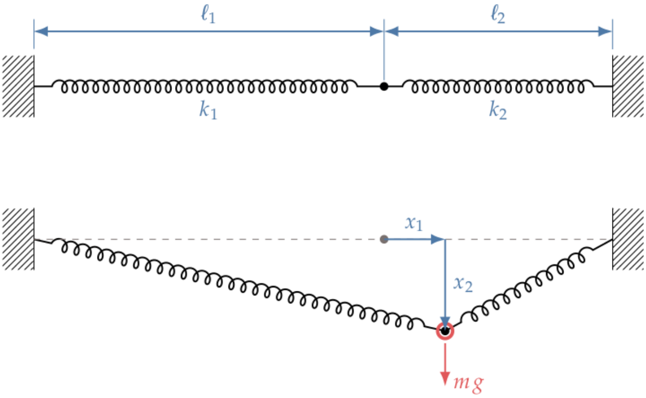
Suppose that $f : \mathbb{R}^n \to \mathbb{R}$ is twice continuously differentiable, that the line search is exact, and that the descent converges to x^* where $\nabla f(x^*) = 0$ and the Hessian matrix $\nabla^2 f(x^*)$ is positive definite. Then

$$f(x_{k+1}) - f(x^*) \leq \left(\frac{\lambda_n - \lambda_1}{\lambda_n + \lambda_1}\right)^2 [f(x_k) - f(x^*)],$$

where $\lambda_1 \leq \cdots \leq \lambda_n$ are the eigenvalues of $\nabla^2 f(x^*)$.



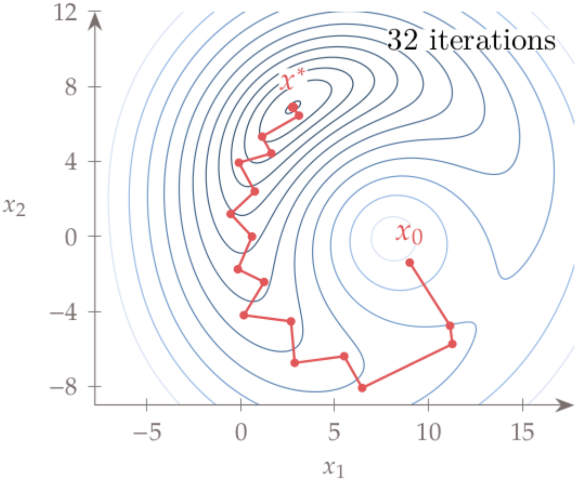
ℓ_1
ℓ_2
k_1
k_2
x_1
x_2
mg



$$f(x_1, x_2) = \frac{1}{2} k_1 \left(\sqrt{(\ell_1 + x_1)^2 + x_2^2} - \ell_1 \right)^2 + \frac{1}{2} k_2 \left(\sqrt{(\ell_2 - x_1)^2 + x_2^2} - \ell_2 \right)^2 - mgx_2$$

Here $\ell_1 = 12, \ell_2 = 8, k_1 = 1, k_2 = 10, mg = 7$

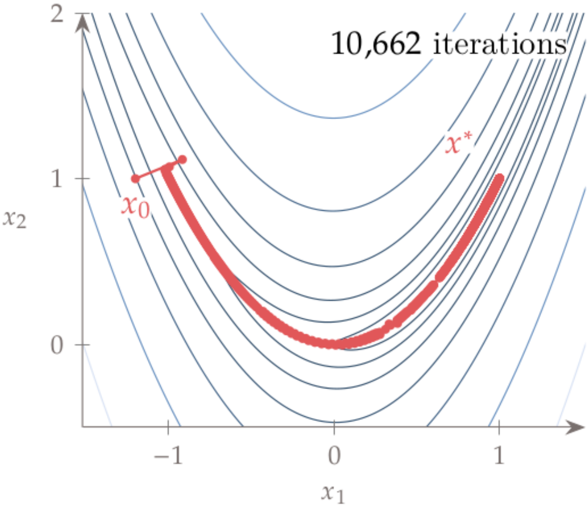
Two Spring Problem - Gradient Descent



Gradient descent, line search, stop. cond. $||\nabla f||_\infty \leq 10^{-6}$.

Rosenbrock Function - Gradient Descent

Rosenbrock: $f\left(x_1, x_2\right)=\left(1-x_1\right)^2+100\left(x_2-x_1^2\right)^2$



Gradient descent, line search, stop. cond. $\|\nabla f\|_\infty \leq 10^{-6}$.

Comments on Gradient Descent

- The method needs evaluation of ∇f at each x_k. If f is not differentiable at x_k, subgradients can be considered (out of the scope of this course).

Comments on Gradient Descent

- The method needs evaluation of ∇f at each x_k. If f is not differentiable at x_k, subgradients can be considered (out of the scope of this course).
- Slow, zig-zagging, provides insufficient information for line search initialization.

Comments on Gradient Descent

- The method needs evaluation of ∇f at each x_k. If f is not differentiable at x_k, subgradients can be considered (out of the scope of this course).
- Slow, zig-zagging, provides insufficient information for line search initialization.
- Susceptible to scaling of variables (see the paraboloid example).

Comments on Gradient Descent

- The method needs evaluation of ∇f at each x_k. If f is not differentiable at x_k, subgradients can be considered (out of the scope of this course).
- Slow, zig-zagging, provides insufficient information for line search initialization.
- Susceptible to scaling of variables (see the paraboloid example).
- THE basis for algorithms training neural networks - a huge amount of specific adjustments are developed for working with huge numbers of variables in neural networks (trillions of weights).

Unconstrained Optimization Algorithms

Descent Direction

Second-Order Methods

Newton's Method

Consider an objective $f : \mathbb{R}^n \to \mathbb{R}$.

Assume that f is twice differentiable.

Newton's Method

Consider an objective $f : \mathbb{R}^n \to \mathbb{R}$.

Assume that f is twice differentiable.

Then, by the Taylor's theorem,

$$f\left(x_k + s\right) \approx f_k + \nabla f_k^\top s + \frac{1}{2} s^\top H_k s$$

where we denote the Hessian $\nabla^2 f(x_k)$ by H_k.

Newton's Method

Consider an objective $f : \mathbb{R}^n \to \mathbb{R}$.

Assume that f is twice differentiable.

Then, by the Taylor's theorem,

$$f\left(x_k + s\right) \approx f_k + \nabla f_k^\top s + \frac{1}{2} s^\top H_k s$$

where we denote the Hessian $\nabla^2 f(x_k)$ by H_k.

Define

$$q(s) = f_k + \nabla f_k^\top s + \frac{1}{2} s^\top H_k s$$

and minimize q w.r.t. s by setting $\nabla q(s) = 0$.

Newton's Method

Consider an objective $f : \mathbb{R}^n \to \mathbb{R}$.

Assume that f is twice differentiable.

Then, by the Taylor's theorem,

$$f(x_k + s) \approx f_k + \nabla f_k^\top s + \frac{1}{2} s^\top H_k s$$

where we denote the Hessian $\nabla^2 f(x_k)$ by H_k.

Define

$$q(s) = f_k + \nabla f_k^\top s + \frac{1}{2} s^\top H_k s$$

and minimize q w.r.t. s by setting $\nabla q(s) = 0$. We obtain:

$$H_k s = -\nabla f_k$$

Denote by s_k the solution, and set $x_{k+1} = x_k + s_k$.

Newton's Method

Algorithm 7 Newton's Method

Input: x_0 starting point, $\varepsilon > 0$

Output: x^* approximation to a stationary point

$k \leftarrow 0$

while $\|\nabla f_k\|_\infty > \varepsilon$ **do**

    $p_k \leftarrow -H_k^{-1}\nabla f(x_k)$

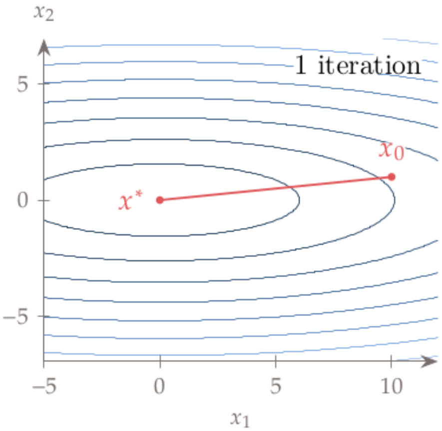
    $x_{k+1} \leftarrow x_k + p_k$

    $k \leftarrow k + 1$

end while

Newton's Method - Example

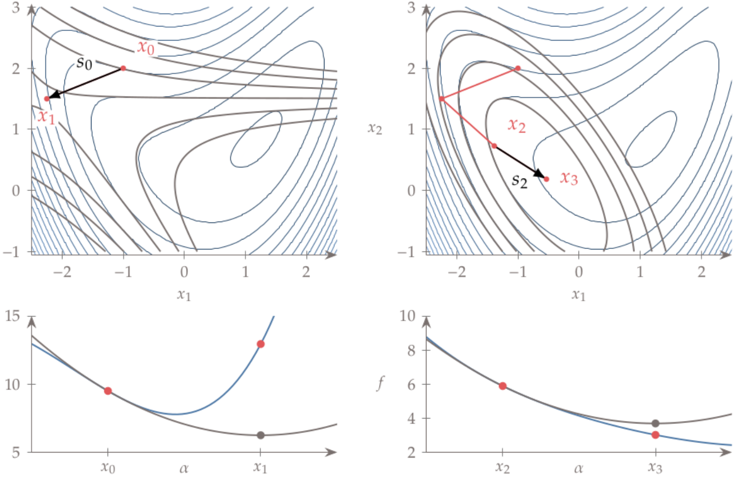
Newton's method finds the minimum of a quadratic function in a single step.



Note that the Newton's method is scale-invariant!

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

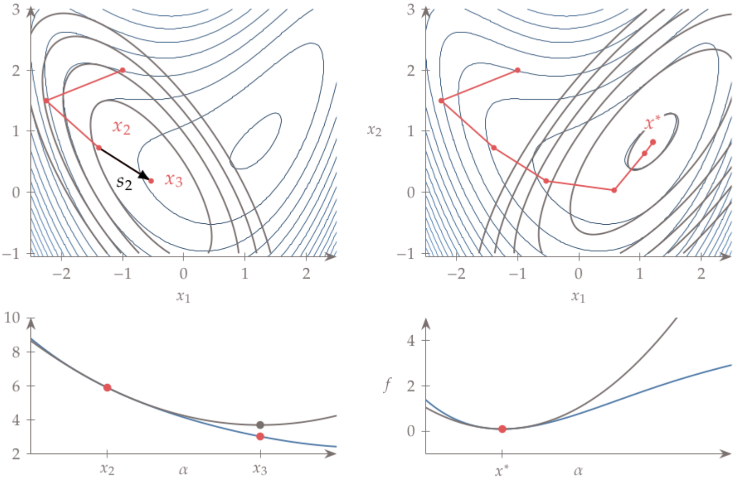
Stopping: $||\nabla f||_\infty \leq 10^{-6}$.



$k = 0$ $\qquad$ $k = 2$

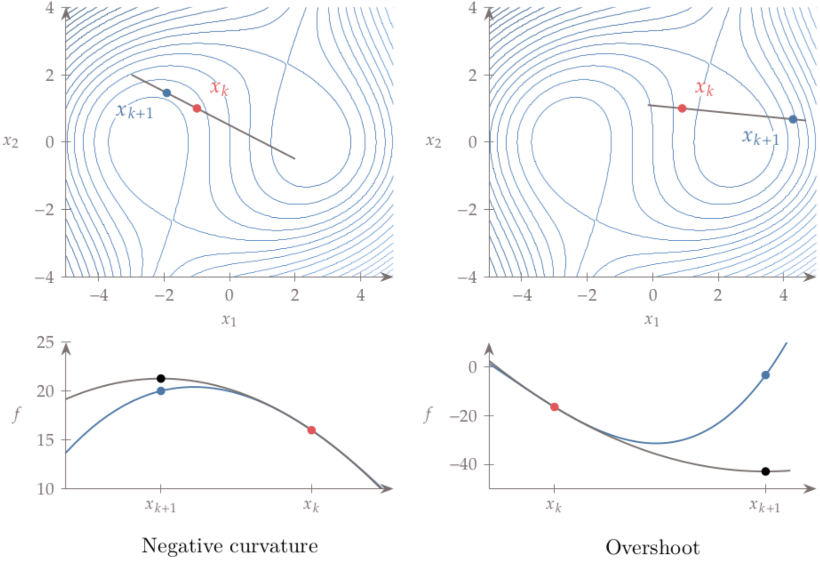
$$f(x_1, x_2) = (1 - x_1)^2 + (1 - x_2)^2 + \frac{1}{2}\left(2x_2 - x_1^2\right)^2$$

Stopping: $||\nabla f||_\infty \leq 10^{-6}$.



$k = 2$ $k = 8$

Convergence Issues



Also, the computation of the Hessian is costly.

Local Quadratic Convergence of Newton's Method

Theorem 10

Assume f is defined and twice differentiable and assume that ∇f is L-smooth on $\mathcal{N}$.
Let x_* be a minimizer of $f(x)$ in $\mathcal{N}$ and assume that $\nabla^2 f(x_*)$ is positive definite.
If $\|x_0 - x_*\|$ is sufficiently small, then $\{x_k\}$ converges quadratically to x_*.

Local Quadratic Convergence of Newton's Method

Theorem 10

Assume f is defined and twice differentiable and assume that ∇f is L-smooth on $\mathcal{N}$.
Let x_* be a minimizer of $f(x)$ in $\mathcal{N}$ and assume that $\nabla^2 f(x_*)$ is positive definite.
If $\|x_0 - x_*\|$ is sufficiently small, then $\{x_k\}$ converges quadratically to x_*.

Note that the theorem implicitly assumes that $\nabla^2 f(x_k)$ is nonsingular for every k.

Local Quadratic Convergence of Newton's Method

Theorem 10

Assume f is defined and twice differentiable and assume that ∇f is L-smooth on $\mathcal{N}$.
Let x_* be a minimizer of $f(x)$ in $\mathcal{N}$ and assume that $\nabla^2 f(x_*)$ is positive definite.
If $\|x_0 - x_*\|$ is sufficiently small, then $\{x_k\}$ converges quadratically to x_*.

Note that the theorem implicitly assumes that $\nabla^2 f(x_k)$ is nonsingular for every k.

As the theorem is concerned only with x_k approaching x^*, the continuity of $\nabla^2 f(x_k)$ and positive definiteness of $\nabla^2 f(x^*)$ imply that $\nabla^2 f(x_k)$ is positive definite for all sufficiently large k.

Local Quadratic Convergence of Newton's Method

Theorem 10

Assume f is defined and twice differentiable and assume that ∇f is L-smooth on $\mathcal{N}$.
Let x_* be a minimizer of $f(x)$ in $\mathcal{N}$ and assume that $\nabla^2 f(x_*)$ is positive definite.
If $\|x_0 - x_*\|$ is sufficiently small, then $\{x_k\}$ converges quadratically to x_*.

Note that the theorem implicitly assumes that $\nabla^2 f(x_k)$ is nonsingular for every k.

As the theorem is concerned only with x_k approaching x^*, the continuity of $\nabla^2 f(x_k)$ and positive definiteness of $\nabla^2 f(x^*)$ imply that $\nabla^2 f(x_k)$ is positive definite for all sufficiently large k.

However, what happens if we start far away from a minimizer?

Newton's Method with Line Search

Algorithm 8 Newton's Method with Line Search

Input: x_0 starting point, $\varepsilon > 0$

Output: x^* approximation to a stationary point

```
k ← 0
α_init ← 1
while ‖∇f_k‖_∞ > ε do
    p_k ← −H_k^{−1} ∇f(x_k)
    α_k ← linesearch(p_k, α_init)
    x_{k+1} ← x_k + p_k
    k ← k + 1
end while
```

$k \leftarrow 0$

$\alpha_{\text{init}} \leftarrow 1$

while $\|\nabla f_k\|_\infty > \varepsilon$ **do**

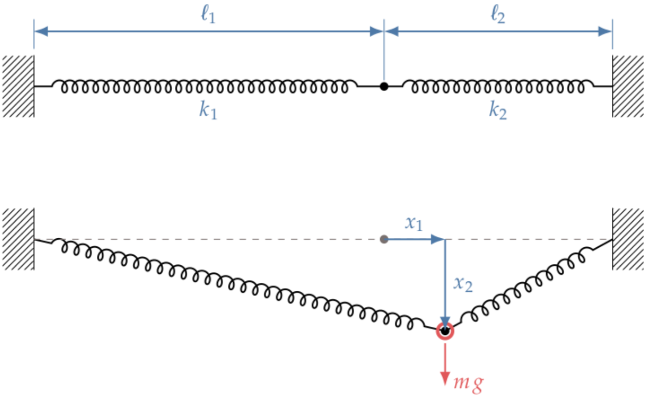
$\quad p_k \leftarrow -H_k^{-1}\nabla f(x_k)$

$\quad \alpha_k \leftarrow \text{linesearch}(p_k, \alpha_{\text{init}})$

$\quad x_{k+1} \leftarrow x_k + p_k$

$\quad k \leftarrow k + 1$

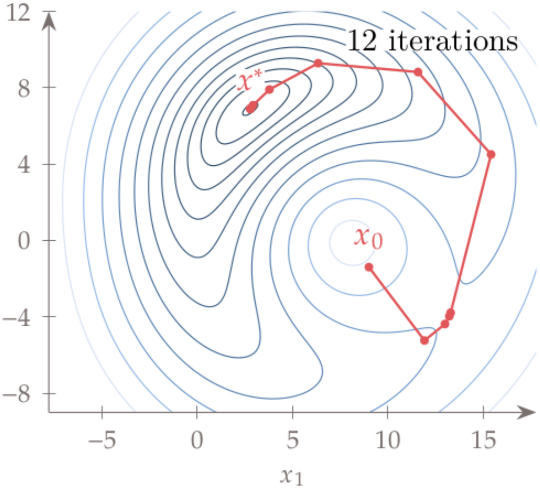
end while



$$
\begin{aligned}
f(x_1, x_2) = &\frac{1}{2}k_1\left(\sqrt{(\ell_1 + x_1)^2 + x_2^2} - \ell_1\right)^2 \\
&+ \frac{1}{2}k_2\left(\sqrt{(\ell_2 - x_1)^2 + x_2^2} - \ell_2\right)^2 - mgx_2
\end{aligned}
$$

Here $\ell_1 = 12, \ell_2 = 8, k_1 = 1, k_2 = 10, mg = 7$

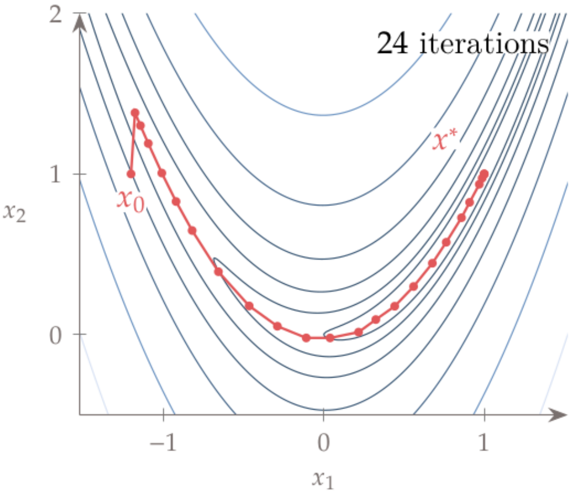
Two Spring Problem - Newton's Method



Gradient descent, line search, stop. cond. $||\nabla f||_\infty \leq 10^{-6}$.
Compare this with 32 iterations of gradient descent.

Rosenbrock Function - Newton's Method

Rosenbrock: $f(x_1, x_2) = (1 - x_1)^2 + 100\left(x_2 - x_1^2\right)^2$



Gradient descent, line search, stop. cond. $||\nabla f||_\infty \leq 10^{-6}$.
Compare this with 10,662 iterations of gradient descent.

Global Convergence of Line Search

Denote by θ_k the angle between p_k and $-\nabla f_k$, i.e., satisfying

$$\cos\theta_k = \frac{-\nabla f_k^T p_k}{\|\nabla f_k\| \, \|p_k\|}$$

Recall that f is L-smooth for some $L > 0$ if

$$\|\nabla f(x) - \nabla f(\tilde{x})\| \leq L\|x - \tilde{x}\|, \qquad \text{for all } x, \tilde{x} \in \mathbb{R}^n$$

Theorem 11 (Zoutendijk)

Consider $x_{k+1} = x_k + \alpha_k p_k$, where p_k is a descent direction *and α_k satisfies the* strong Wolfe conditions*. Suppose that f is bounded below, continuously differentiable, and* L-smooth*. Then*

$$\sum_{k\geq 0} \cos^2\theta_k \, \|\nabla f_k\|^2 < \infty.$$

Global Convergence of Newton's Method

Assume that all α_k satisfy strong Wolfe conditions.

Global Convergence of Newton's Method

Assume that all α_k satisfy strong Wolfe conditions.

Assume that the Hessians H_k are positive definite with a uniformly bounded condition number:

$$||H_k|| \left|\left|H_k^{-1}\right|\right| \leq M \quad \text{for all } k$$

Global Convergence of Newton's Method

Assume that all α_k satisfy strong Wolfe conditions.

Assume that the Hessians H_k are positive definite with a uniformly bounded condition number:

$$||H_k|| \, \left|\left|H_k^{-1}\right|\right| \leq M \quad \text{for all } k$$

Then θ_k between $p_k = -H_k^{-1}\nabla f_k$ and $-\nabla f_k$ and satisfies

$$\cos\theta_k \geq 1/M$$

Global Convergence of Newton's Method

Assume that all α_k satisfy strong Wolfe conditions.

Assume that the Hessians H_k are positive definite with a uniformly bounded condition number:

$$||H_k|| \left|\left|H_k^{-1}\right|\right| \leq M \quad \text{for all } k$$

Then θ_k between $p_k = -H_k^{-1}\nabla f_k$ and $-\nabla f_k$ and satisfies

$$\cos\theta_k \geq 1/M$$

Thus, under the assumptions of Zoutendijk's theorem, we obtain

$$\frac{1}{M^2}\sum_{k\geq 0}\|\nabla f_k\|^2 \leq \sum_{k\geq 0}\cos^2\theta_k\,\|\nabla f_k\|^2 < \infty$$

which implies that $\lim_{k\to\infty}||\nabla f_k|| = 0$.

Global Convergence of Newton's Method

Assume that all α_k satisfy strong Wolfe conditions.

Assume that the Hessians H_k are positive definite with a uniformly bounded condition number:

$$||H_k|| \, \left|\left|H_k^{-1}\right|\right| \leq M \quad \text{for all } k$$

Then θ_k between $p_k = -H_k^{-1}\nabla f_k$ and $-\nabla f_k$ and satisfies

$$\cos\theta_k \geq 1/M$$

Thus, under the assumptions of Zoutendijk's theorem, we obtain

$$\frac{1}{M^2}\sum_{k\geq 0} \|\nabla f_k\|^2 \leq \sum_{k\geq 0} \cos^2\theta_k \, \|\nabla f_k\|^2 < \infty$$

which implies that $\lim_{k\to\infty} ||\nabla f_k|| = 0$.

What if H_k is not positive definite or is (nearly) singular?

Eigenvalue Modification

Consider $H_k = \nabla^2 f(x_k)$ and consider its diagonal form:

$$H_k = QDQ^T$$

Where D contains the eigenvalues of H_k on the diagonal, i.e., $D = \text{diag}(\lambda_1, \ldots, \lambda_n)$ and Q is an orthogonal matrix.

Eigenvalue Modification

Consider $H_k = \nabla^2 f(x_k)$ and consider its diagonal form:

$$H_k = QDQ^T$$

Where D contains the eigenvalues of H_k on the diagonal, i.e., $D = \text{diag}(\lambda_1, \dots, \lambda_n)$ and Q is an orthogonal matrix.

Observe that

- H_k is not positive definite iff $\lambda_i \leq 0$ for some i
- $||H_k||$ grows with $\max\{\lambda_1, \dots, \lambda_n\}$ going to infinity.
- $\left|\left|H_k^{-1}\right|\right|$ grows with $\min\{\lambda_1, \dots, \lambda_n\}$ going to 0
  (i.e., the matrix becomes close to a singular matrix)

We want to prevent all three cases, i.e., make sure that for some reasonably large $\delta > 0$ we have $\lambda_i \geq \delta$ but not too large.

Eigenvalue Modification

Consider $H_k = \nabla^2 f(x_k)$ and consider its diagonal form:

$$H_k = QDQ^T$$

Where D contains the eigenvalues of H_k on the diagonal, i.e., $D = \text{diag}(\lambda_1, \ldots, \lambda_n)$ and Q is an orthogonal matrix.

Observe that

- H_k is not positive definite iff $\lambda_i \leq 0$ for some i
- $||H_k||$ grows with $\max\{\lambda_1, \ldots, \lambda_n\}$ going to infinity.
- $\left|\left|H_k^{-1}\right|\right|$ grows with $\min\{\lambda_1, \ldots, \lambda_n\}$ going to 0
  (i.e., the matrix becomes close to a singular matrix)

We want to prevent all three cases, i.e., make sure that for some reasonably large $\delta > 0$ we have $\lambda_i \geq \delta$ but not too large.

Two questions are in order:

- What is a reasonably large δ?
- How to modify H_k so the minimum is large enough?

Sufficiently Large Eigenvalues

Consider an example:

$$\nabla f(x_k) = (1, -3, 2) \qquad \text{and} \qquad \nabla^2 f(x_k) = \mathrm{diag}(10, 3, -1)$$

Sufficiently Large Eigenvalues

Consider an example:

$$\nabla f(x_k) = (1, -3, 2) \qquad \text{and} \qquad \nabla^2 f(x_k) = \mathrm{diag}(10, 3, -1)$$

Now, the diagonalization is trivial:

$$\nabla^2 f(x_k) = Q \,\mathrm{diag}(10, 3, -1)\, Q^\top \quad Q = I \text{ is the identity matrix}$$

Sufficiently Large Eigenvalues

Consider an example:

$$\nabla f(x_k) = (1, -3, 2) \qquad \text{and} \qquad \nabla^2 f(x_k) = \mathrm{diag}(10, 3, -1)$$

Now, the diagonalization is trivial:

$$\nabla^2 f(x_k) = Q\,\mathrm{diag}(10, 3, -1)\, Q^\top \quad Q = I \text{ is the identity matrix}$$

What if we consider a minimum modification replacing the negative eigenvalue with a small number, say $\delta = 10^{-8}$?

Sufficiently Large Eigenvalues

Consider an example:

$$\nabla f(x_k) = (1, -3, 2) \qquad \text{and} \qquad \nabla^2 f(x_k) = \operatorname{diag}(10, 3, -1)$$

Now, the diagonalization is trivial:

$$\nabla^2 f(x_k) = Q \operatorname{diag}(10, 3, -1)\, Q^\top \quad Q = I \text{ is the identity matrix}$$

What if we consider a minimum modification replacing the negative eigenvalue with a small number, say $\delta = 10^{-8}$? Obtain

$$B_k = Q \operatorname{diag}(10, 3, 10^{-8})\, Q^\top = \operatorname{diag}(10, 3, 10^{-8})$$

Sufficiently Large Eigenvalues

Consider an example:

$$\nabla f(x_k) = (1, -3, 2) \qquad \text{and} \qquad \nabla^2 f(x_k) = \mathrm{diag}(10, 3, -1)$$

Now, the diagonalization is trivial:

$$\nabla^2 f(x_k) = Q \,\mathrm{diag}(10, 3, -1)\, Q^\top \quad Q = I \text{ is the identity matrix}$$

What if we consider a minimum modification replacing the negative eigenvalue with a small number, say $\delta = 10^{-8}$? Obtain

$$B_k = Q \ \mathrm{diag}(10, 3, 10^{-8})\ Q^\top = \mathrm{diag}(10, 3, 10^{-8})$$

If used in Newton's method, we obtain the following direction:

$$p_k = -B_k^{-1} \nabla f(x_k) = (1/10, -1, -(2 \cdot 10^8))$$

Thus, a very long vector almost parallel to the third dimension.

Sufficiently Large Eigenvalues

Consider an example:

$$\nabla f(x_k) = (1, -3, 2) \qquad \text{and} \qquad \nabla^2 f(x_k) = \mathrm{diag}(10, 3, -1)$$

Now, the diagonalization is trivial:

$$\nabla^2 f(x_k) = Q\,\mathrm{diag}(10, 3, -1)\,Q^\top \quad Q = I \text{ is the identity matrix}$$

What if we consider a minimum modification replacing the negative eigenvalue with a small number, say $\delta = 10^{-8}$? Obtain

$$B_k = Q\ \mathrm{diag}(10, 3, 10^{-8})\ Q^\top = \mathrm{diag}(10, 3, 10^{-8})$$

If used in Newton's method, we obtain the following direction:

$$p_k = -B_k^{-1} \nabla f(x_k) = (1/10, -1, -(2 \cdot 10^8))$$

Thus, a very long vector almost parallel to the third dimension.

Even though f decreases along p_k, it is far from the minimum of the quadratic approximation of f.

Note that the original Newton's direction is
$-\mathrm{diag}(1/10, 1/3, -1)(1, -3, 2)^\top = (-1/10, 1, 2)$ which is completely different.

Modifying the Eigenvalues

Other strategies for eigenvalue modification can be devised.

Modifying the Eigenvalues

Other strategies for eigenvalue modification can be devised.

The criteria are rather loose. The resulting matrix B_k should be

- positive definite,
- of bounded norm (for all k),
- not too close to being singular.

  (i.e., the eigenvalues should be sufficiently large)

Modifying the Eigenvalues

Other strategies for eigenvalue modification can be devised.

The criteria are rather loose. The resulting matrix B_k should be

- positive definite,
- of bounded norm (for all k),
- not too close to being singular.

  (i.e., the eigenvalues should be sufficiently large)

Strategies for eigenvalue modification include flipping negative eigenvalues to positive values, substituting negative eigenvalues with small positive ones, etc.

Modifying the Eigenvalues

Other strategies for eigenvalue modification can be devised.

The criteria are rather loose. The resulting matrix B_k should be

- positive definite,
- of bounded norm (for all k),
- not too close to being singular.

  (i.e., the eigenvalues should be sufficiently large)

Strategies for eigenvalue modification include flipping negative eigenvalues to positive values, substituting negative eigenvalues with small positive ones, etc.

There is no consensus on the best method for the modification.

Modifying the Eigenvalues

Other strategies for eigenvalue modification can be devised.

The criteria are rather loose. The resulting matrix B_k should be

- positive definite,
- of bounded norm (for all k),
- not too close to being singular.

  (i.e., the eigenvalues should be sufficiently large)

Strategies for eigenvalue modification include flipping negative eigenvalues to positive values, substituting negative eigenvalues with small positive ones, etc.

There is no consensus on the best method for the modification.

The implementation is based on computing $B_k = H_k + \Delta H_k$ for an appropriate modification matrix ΔH_k.

What is ΔH_k in our example?

Modifying the Eigenvalues

Other strategies for eigenvalue modification can be devised.

The criteria are rather loose. The resulting matrix B_k should be

- positive definite,
- of bounded norm (for all k),
- not too close to being singular.

  (i.e., the eigenvalues should be sufficiently large)

Strategies for eigenvalue modification include flipping negative eigenvalues to positive values, substituting negative eigenvalues with small positive ones, etc.

There is no consensus on the best method for the modification.

The implementation is based on computing $B_k = H_k + \Delta H_k$ for an appropriate modification matrix ΔH_k.

What is ΔH_k in our example?

Various methods for computing ΔH_k have been devised in literature. Typically, it is based on some computationally cheaper decomposition than spectral decomposition (e.g., Cholesky).

Modified Newton's Method

Algorithm 9 Newton's Method with Line Search

Input: x_0 starting point, $\varepsilon > 0$

Output: x^* approximation to a stationary point

```
 1: k ← 0
 2: while ‖∇f_k‖_∞ > ε do
 3:     H_k ← ∇²f(x_k)
 4:     if H_k is not sufficiently positive definite then
 5:         H_k ← H_k + ΔH_k so that H_k is sufficiently pos. definite
 6:     end if
 7:     Solve H_k p_k = −∇f(x_k) for p_k
 8:     Set x_{k+1} = x_k + α_k p_k, here α_k sat. the Wolfe cond.
 9:     k ← k + 1
10: end while
```

Comments on Newton's Method

- Newton's method is scale invariant.

Comments on Newton's Method

- Newton's method is scale invariant.
- Quadratic convergence in a close vicinity of a strict minimizer.

Comments on Newton's Method

- Newton's method is scale invariant.
- Quadratic convergence in a close vicinity of a strict minimizer.
- Without modification, it may converge to an arbitrary stationary point (maximum, saddle point).

Comments on Newton's Method

- Newton's method is scale invariant.
- Quadratic convergence in a close vicinity of a strict minimizer.
- Without modification, it may converge to an arbitrary stationary point (maximum, saddle point).
- Computationally expensive:
  - $\mathcal{O}(n^2)$ second derivatives in the Hessian, each may be hard to compute.
    Automated derivation methods help but still need store $\mathcal{O}(n^2)$ results.

Comments on Newton's Method

- Newton's method is scale invariant.
- Quadratic convergence in a close vicinity of a strict minimizer.
- Without modification, it may converge to an arbitrary stationary point (maximum, saddle point).
- Computationally expensive:
  - $\mathcal{O}(n^2)$ second derivatives in the Hessian, each may be hard to compute.
    Automated derivation methods help but still need store $\mathcal{O}(n^2)$ results.
  - $\mathcal{O}(n^3)$ arithmetic operations to solve the linear system for the direction p_k.
    May be mitigated by more efficient methods in case of sparse Hessians.

Comments on Newton's Method

- Newton's method is scale invariant.
- Quadratic convergence in a close vicinity of a strict minimizer.
- Without modification, it may converge to an arbitrary stationary point (maximum, saddle point).
- Computationally expensive:
  - $\mathcal{O}(n^2)$ second derivatives in the Hessian, each may be hard to compute.
    Automated derivation methods help but still need store $\mathcal{O}(n^2)$ results.
  - $\mathcal{O}(n^3)$ arithmetic operations to solve the linear system for the direction p_k.
    May be mitigated by more efficient methods in case of sparse Hessians.

In a sense, Newton's method is an impractical "ideal" with which other methods are compared.

The efficiency issues (and the necessity of second-order derivatives) will be mitigated by using quasi-Newton methods.

Quasi-Newton Methods

Quasi-Newton Methods

Recall that Newton's method step p_k in $x_{k+1} = x_k + p_k$ comes from minimization of

$$q(p) = f_k + \nabla f_k^\top p + \frac{1}{2} p^\top H_k p$$

w.r.t. p by setting $\nabla q(p) = 0$ and solving

$$H_k p = -\nabla f_k$$

So Newton's method needs the second derivative (Hessian), which is computationally hard to obtain.

Quasi-Newton Methods

Recall that Newton's method step p_k in $x_{k+1} = x_k + p_k$ comes from minimization of

$$q(p) = f_k + \nabla f_k^\top p + \frac{1}{2} p^\top H_k p$$

w.r.t. p by setting $\nabla q(p) = 0$ and solving

$$H_k p = -\nabla f_k$$

So Newton's method needs the second derivative (Hessian), which is computationally hard to obtain.

Gradient descent needs only the first derivatives but converges slowly.

Quasi-Newton Methods

Recall that Newton's method step p_k in $x_{k+1} = x_k + p_k$ comes from minimization of

$$q(p) = f_k + \nabla f_k^\top p + \frac{1}{2} p^\top H_k p$$

w.r.t. p by setting $\nabla q(p) = 0$ and solving

$$H_k p = -\nabla f_k$$

So Newton's method needs the second derivative (Hessian), which is computationally hard to obtain.

Gradient descent needs only the first derivatives but converges slowly.

Can we find a compromise?

Quasi-Newton Methods

Recall that Newton's method step p_k in $x_{k+1} = x_k + p_k$ comes from minimization of

$$q(p) = f_k + \nabla f_k^\top p + \frac{1}{2} p^\top H_k p$$

w.r.t. p by setting $\nabla q(p) = 0$ and solving

$$H_k p = -\nabla f_k$$

So Newton's method needs the second derivative (Hessian), which is computationally hard to obtain.

Gradient descent needs only the first derivatives but converges slowly.

Can we find a compromise?

Quasi-Newton methods use first derivatives to approximate the Hessian H_k in Newton's method with a matrix $\tilde{H}_k$.

Quasi-Newton Methods

Suppose we have just obtained the new point x_{k+1} after a line search starting from x_k in the direction p_k.

Quasi-Newton Methods

Suppose we have just obtained the new point x_{k+1} after a line search starting from x_k in the direction p_k.

Consider the Hessian $H_{k+1} = \nabla^2 f(x_{k+1})$ and its approximation denoted by $\tilde{H}_{k+1}$.

Quasi-Newton Methods

Suppose we have just obtained the new point x_{k+1} after a line search starting from x_k in the direction p_k.

Consider the Hessian $H_{k+1} = \nabla^2 f(x_{k+1})$ and its approximation denoted by $\tilde{H}_{k+1}$.

We aim to use $\tilde{H}_{k+1}$ in the next step, that is, in the equation $\tilde{H}_{k+1} p = -\nabla f_{k+1}$ yielding p_{k+1}.

Quasi-Newton Methods

Suppose we have just obtained the new point x_{k+1} after a line search starting from x_k in the direction p_k.

Consider the Hessian $H_{k+1} = \nabla^2 f(x_{k+1})$ and its approximation denoted by $\tilde{H}_{k+1}$.

We aim to use $\tilde{H}_{k+1}$ in the next step, that is, in the equation $\tilde{H}_{k+1} p = -\nabla f_{k+1}$ yielding p_{k+1}.

What conditions should $\tilde{H}_{k+1}$ satisfy so that it functions as the "true" Hessian H_{k+1}?

Quasi-Newton Methods

Suppose we have just obtained the new point x_{k+1} after a line search starting from x_k in the direction p_k.

Consider the Hessian $H_{k+1} = \nabla^2 f(x_{k+1})$ and its approximation denoted by $\tilde{H}_{k+1}$.

We aim to use $\tilde{H}_{k+1}$ in the next step, that is, in the equation $\tilde{H}_{k+1} p = -\nabla f_{k+1}$ yielding p_{k+1}.

What conditions should $\tilde{H}_{k+1}$ satisfy so that it functions as the "true" Hessian H_{k+1}?

First, it should be *symmetric positive definite*.

To always yield decrease direction.

Quasi-Newton Methods

Suppose we have just obtained the new point x_{k+1} after a line search starting from x_k in the direction p_k.

Consider the Hessian $H_{k+1} = \nabla^2 f(x_{k+1})$ and its approximation denoted by $\tilde{H}_{k+1}$.

We aim to use $\tilde{H}_{k+1}$ in the next step, that is, in the equation $\tilde{H}_{k+1} p = -\nabla f_{k+1}$ yielding p_{k+1}.

What conditions should $\tilde{H}_{k+1}$ satisfy so that it functions as the "true" Hessian H_{k+1}?

First, it should be *symmetric positive definite*.

To always yield decrease direction.

Second, extrapolating from the single variable secant method, we demand

$$\tilde{H}_{k+1}(x_{k+1} - x_k) = \nabla f_{k+1} - \nabla f_k$$

This is the *secant condition*.

Secant Condition

Consider the secant condition:

$$\tilde{H}_{k+1}(x_{k+1} - x_k) = \nabla f_{k+1} - \nabla f_k$$

The notation is usually simplified by

$$s_k = x_{k+1} - x_k \qquad y_k = \nabla f_{k+1} - \nabla f_k$$

So that the secant condition becomes

$$\tilde{H}_{k+1} s_k = y_k$$

Secant Condition

Consider the secant condition:

$$\tilde{H}_{k+1}(x_{k+1} - x_k) = \nabla f_{k+1} - \nabla f_k$$

The notation is usually simplified by

$$s_k = x_{k+1} - x_k \qquad y_k = \nabla f_{k+1} - \nabla f_k$$

So that the secant condition becomes

$$\tilde{H}_{k+1} s_k = y_k$$

Does it have a symmetric positive definite solution?

Curvature Condition

Consider the secant condition:

$$\tilde{H}_{k+1} s_k = y_k$$

Curvature Condition

Consider the secant condition:

$$\tilde{H}_{k+1} s_k = y_k$$

The following is true:

- The secant condition has a symmetric positive definite solution iff the following condition is satisfied:

$$s_k^\top y_k > 0$$

Curvature Condition

Consider the secant condition:

$$\tilde{H}_{k+1} s_k = y_k$$

The following is true:

- The secant condition has a symmetric positive definite solution iff the following condition is satisfied:

$$s_k^\top y_k > 0$$

- The condition $s_k^\top y_k > 0$ is satisfied if the line search satisfies the strong Wolfe conditions.

Curvature Condition

Consider the secant condition:

$$\tilde{H}_{k+1} s_k = y_k$$

The following is true:

- The secant condition has a symmetric positive definite solution iff the following condition is satisfied:

  $$s_k^\top y_k > 0$$

- The condition $s_k^\top y_k > 0$ is satisfied if the line search satisfies the strong Wolfe conditions.

As a corollary, we obtain the following:

Theorem 12

Assume that we use line search satisfying strong Wolfe conditions. Then in every step, the secant condition

$$\tilde{H}_{k+1} s_k = y_k$$

has a symmetric positive definite solution $\tilde{H}_{k+1}$.

Now, we can obtain an approximate Hessian $\tilde{H}_{k+1}$ by solving the secant condition $\tilde{H}_{k+1}s_k = y_k$.

Now, we can obtain an approximate Hessian $\tilde{H}_{k+1}$ by solving the secant condition $\tilde{H}_{k+1}s_k = y_k$.

Note that even if we demand symmetric positive definite solutions to the secant condition, there are infinitely many.

Indeed, there are $n(n+1)/2$ degrees of freedom in a symmetric matrix, and the secant conditions represent only n conditions.

Now, we can obtain an approximate Hessian $\tilde{H}_{k+1}$ by solving the secant condition $\tilde{H}_{k+1}s_k = y_k$.

Note that even if we demand symmetric positive definite solutions to the secant condition, there are infinitely many.

Indeed, there are $n(n+1)/2$ degrees of freedom in a symmetric matrix, and the secant conditions represent only n conditions.

Moreover, we want to obtain $\tilde{H}_{k+1}$ from $\tilde{H}_k$ by

$$\tilde{H}_{k+1} = \tilde{H}_k + \text{something}$$

To have a nice iterative algorithm.

Now, we can obtain an approximate Hessian $\tilde{H}_{k+1}$ by solving the secant condition $\tilde{H}_{k+1}s_k = y_k$.

Note that even if we demand symmetric positive definite solutions to the secant condition, there are infinitely many.

Indeed, there are $n(n+1)/2$ degrees of freedom in a symmetric matrix, and the secant conditions represent only n conditions.

Moreover, we want to obtain $\tilde{H}_{k+1}$ from $\tilde{H}_k$ by

$$\tilde{H}_{k+1} = \tilde{H}_k + \text{something}$$

To have a nice iterative algorithm.

We also want $\tilde{H}_{k+1}$ to be symmetric positive definite.

Now, we can obtain an approximate Hessian $\tilde{H}_{k+1}$ by solving the secant condition $\tilde{H}_{k+1}s_k = y_k$.

Note that even if we demand symmetric positive definite solutions to the secant condition, there are infinitely many.

Indeed, there are $n(n+1)/2$ degrees of freedom in a symmetric matrix, and the secant conditions represent only n conditions.

Moreover, we want to obtain $\tilde{H}_{k+1}$ from $\tilde{H}_k$ by

$$\tilde{H}_{k+1} = \tilde{H}_k + \text{something}$$

To have a nice iterative algorithm.

We also want $\tilde{H}_{k+1}$ to be symmetric positive definite.

We strive to choose $\tilde{H}_{k+1}$ "close" to $\tilde{H}_k$.

Symmetric Rank One Update (SR1)

Note that the information about the solution is present in s_k and y_k, so it is natural to compose the solution using these vectors.

Symmetric Rank One Update (SR1)

Note that the information about the solution is present in s_k and y_k, so it is natural to compose the solution using these vectors.

Consider $u = \left(y_k - \tilde{H}_k s_k\right)$

$$\tilde{H}_{k+1} = \tilde{H}_k + \frac{uu^\top}{u^\top s_k}$$

Symmetric Rank One Update (SR1)

Note that the information about the solution is present in s_k and y_k, so it is natural to compose the solution using these vectors.

Consider $u = \left(y_k - \tilde{H}_k s_k\right)$

$$\tilde{H}_{k+1} = \tilde{H}_k + \frac{uu^\top}{u^\top s_k}$$

Now, the secant condition is satisfied:

$$\tilde{H}_{k+1} s_k = \tilde{H}_k s_k + \frac{uu^\top s_k}{u^\top s_k} = \tilde{H}_k s_k + u = \tilde{H}_k s_k + \left(y_k - \tilde{H}_k s_k\right) = y_k$$

By the way, the matrix $\frac{uu^\top}{u^\top s_k}$ is of rank one and is a unique symmetric rank one matrix which makes $\tilde{H}_{k+1}$ satisfy the secant condition.

Symmetric Rank One Update (SR1)

Note that the information about the solution is present in s_k and y_k, so it is natural to compose the solution using these vectors.

Consider $u = \left(y_k - \tilde{H}_k s_k\right)$

$$\tilde{H}_{k+1} = \tilde{H}_k + \frac{uu^\top}{u^\top s_k}$$

Now, the secant condition is satisfied:

$$\tilde{H}_{k+1} s_k = \tilde{H}_k s_k + \frac{uu^\top s_k}{u^\top s_k} = \tilde{H}_k s_k + u = \tilde{H}_k s_k + \left(y_k - \tilde{H}_k s_k\right) = y_k$$

By the way, the matrix $\frac{uu^\top}{u^\top s_k}$ is of rank one and is a unique symmetric rank one matrix which makes $\tilde{H}_{k+1}$ satisfy the secant condition.

To obtain a quasi-Newton method, it suffices to initialize $\tilde{H}_0$, typically to the identity I, and use $\tilde{H}_k$ instead of the Hessian $H_k = \nabla^2 f_k$ in Newton's method.

Symmetric Rank One Update

Algorithm 10 SR1

$k \leftarrow 0$
$\alpha_{\text{init}} \leftarrow 1$
$\tilde{H}_0 \leftarrow I$
while $\|\nabla f_k\|_\infty > \tau$ **do**
  Solve for p_k in $\tilde{H}_k p_k = -\nabla f_k$
  $\alpha \leftarrow \text{linesearch}(p_k, \alpha_{\text{init}})$
  $x_{k+1} \leftarrow x_k + \alpha p_k$
  $s \leftarrow x_{k+1} - x_k$
  $y \leftarrow \nabla f_{k+1} - \nabla f_k$
  $u \leftarrow y - \tilde{H}_k s$
  $\tilde{H}_{k+1} \leftarrow \tilde{H}_k + \frac{uu^\top}{u^\top s}$
  $k \leftarrow k + 1$
end while

Note that the denominator $u^\top s_k$ can be 0, in which case the update is impossible. The usual strategy is to skip the update and set $\tilde{H}_{k+1} = \tilde{H}_k$.

Example

We will look at a three-dimensional quadratic problem $f(x) = \frac{1}{2}x^\top Qx - c^\top x$ with

$$Q = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 4 \end{pmatrix} \quad \text{and} \quad c = \begin{pmatrix} -8 \\ -9 \\ -8 \end{pmatrix},$$

whose solution is $x_* = (-4, -3, -2)^\top$. Use the exact line search.

The initial guesses are $\tilde{H}_0 = I$ and $x_0 = (0, 0, 0)^\top$.

Example

We will look at a three-dimensional quadratic problem $f(x) = \frac{1}{2}x^\top Qx - c^\top x$ with

$$Q = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 4 \end{pmatrix} \quad \text{and} \quad c = \begin{pmatrix} -8 \\ -9 \\ -8 \end{pmatrix},$$

whose solution is $x_* = (-4, -3, -2)^\top$. Use the exact line search.

The initial guesses are $\tilde{H}_0 = I$ and $x_0 = (0, 0, 0)^\top$.

At the initial point, $\|\nabla f(x_0)\|_\infty = \|-c\|_\infty = 9$, so this point is not optimal.

Example

We will look at a three-dimensional quadratic problem $f(x) = \frac{1}{2}x^\top Qx - c^\top x$ with

$$Q = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 4 \end{pmatrix} \quad \text{and} \quad c = \begin{pmatrix} -8 \\ -9 \\ -8 \end{pmatrix},$$

whose solution is $x_* = (-4, -3, -2)^\top$. Use the exact line search.

The initial guesses are $\tilde{H}_0 = I$ and $x_0 = (0, 0, 0)^\top$.

At the initial point, $\|\nabla f(x_0)\|_\infty = \|-c\|_\infty = 9$, so this point is not optimal.The first search direction is

$$p_0 = \begin{pmatrix} -8 \\ -9 \\ -8 \end{pmatrix}.$$

The exact line search gives $\alpha_0 = 0.3333$.

Example

The new estimate of the solution, the update vectors, and the new Hessian approximation are:

$$x_1 = \begin{pmatrix} -2.66 \\ -3.00 \\ -2.66 \end{pmatrix}, \nabla f_1 = \begin{pmatrix} 2.66 \\ 0 \\ -2.66 \end{pmatrix}, s_0 = \begin{pmatrix} -2.66 \\ -3.00 \\ -2.66 \end{pmatrix}, y_0 = \begin{pmatrix} -5.33 \\ -9.00 \\ -10.66 \end{pmatrix},$$

Example

The new estimate of the solution, the update vectors, and the new Hessian approximation are:

$$x_1 = \begin{pmatrix} -2.66 \\ -3.00 \\ -2.66 \end{pmatrix}, \nabla f_1 = \begin{pmatrix} 2.66 \\ 0 \\ -2.66 \end{pmatrix}, s_0 = \begin{pmatrix} -2.66 \\ -3.00 \\ -2.66 \end{pmatrix}, y_0 = \begin{pmatrix} -5.33 \\ -9.00 \\ -10.66 \end{pmatrix},$$

and

$$\tilde{H}_1 = I + \frac{(y_0 - Is_0)(y_0 - Is_0)^\top}{(y_0 - Is_0)^\top s_0} = \begin{pmatrix} 1.1531 & 0.3445 & 0.4593 \\ 0.3445 & 1.7751 & 1.0335 \\ 0.4593 & 1.0335 & 2.3780 \end{pmatrix}.$$

Example

The new estimate of the solution, the update vectors, and the new Hessian approximation are:

$$x_1 = \begin{pmatrix} -2.66 \\ -3.00 \\ -2.66 \end{pmatrix}, \nabla f_1 = \begin{pmatrix} 2.66 \\ 0 \\ -2.66 \end{pmatrix}, s_0 = \begin{pmatrix} -2.66 \\ -3.00 \\ -2.66 \end{pmatrix}, y_0 = \begin{pmatrix} -5.33 \\ -9.00 \\ -10.66 \end{pmatrix},$$

and

$$\tilde{H}_1 = I + \frac{(y_0 - Is_0)(y_0 - Is_0)^\top}{(y_0 - Is_0)^\top s_0} = \begin{pmatrix} 1.1531 & 0.3445 & 0.4593 \\ 0.3445 & 1.7751 & 1.0335 \\ 0.4593 & 1.0335 & 2.3780 \end{pmatrix}.$$

At this new point $\|\nabla f(x_1)\|_\infty = 2.66$ so we keep going, obtaining the search direction

$$p_1 = \begin{pmatrix} -2.9137 \\ -0.5557 \\ 1.9257 \end{pmatrix},$$

and the step length $\alpha_1 = 0.3942$.

Example

This gives the new estimates:

$$x_2 = \begin{pmatrix} -3.81 \\ -3.21 \\ -1.90 \end{pmatrix}, \quad \nabla f_2 = \begin{pmatrix} 0.36 \\ -0.65 \\ 0.36 \end{pmatrix}, \quad s_1 = \begin{pmatrix} -1.14 \\ -0.21 \\ 0.75 \end{pmatrix}, \quad y_1 = \begin{pmatrix} -2.29 \\ -0.65 \\ 3.03 \end{pmatrix}$$

and

$$\tilde{H}_2 = \begin{pmatrix} 1.6568 & 0.6102 & -0.3432 \\ 0.6102 & 1.9153 & 0.6102 \\ -0.3432 & 0.6102 & 3.6568 \end{pmatrix}.$$

At the point x_2, $\|\nabla f(x_2)\|_\infty = 0.65$ so we keep going, with

$$p_2 = \begin{pmatrix} -0.4851 \\ 0.5749 \\ -0.2426 \end{pmatrix},$$

and $\alpha = 0.3810$.

Example

This gives

$$x_3 = \begin{pmatrix} -4 \\ -3 \\ -2 \end{pmatrix}, \quad \nabla f_3 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \quad s_2 = \begin{pmatrix} -0.18 \\ 0.21 \\ -0.09 \end{pmatrix}, \quad y_2 = \begin{pmatrix} -0.36 \\ 0.65 \\ -0.36 \end{pmatrix},$$

and $\tilde{H}_3 = Q$. Now $\|\nabla f(x_3)\| = 0$, so we stop.

Properties of SR1

Does symmetric rank one update satisfy our demands?

We want every $\tilde{H}_k$ to be a symmetric positive definite solution to the secant condition.

Properties of SR1

Does symmetric rank one update satisfy our demands?

We want every $\tilde{H}_k$ to be a symmetric positive definite solution to the secant condition.

Unfortunately, though $\tilde{H}_k$ is a symmetric positive definite, the updated matrix $\tilde{H}_{k+1}$ does not have to be a positive definite.

Properties of SR1

Does symmetric rank one update satisfy our demands?

We want every $\tilde{H}_k$ to be a symmetric positive definite solution to the secant condition.

Unfortunately, though $\tilde{H}_k$ is a symmetric positive definite, the updated matrix $\tilde{H}_{k+1}$ does not have to be a positive definite.

Still, the symmetric rank one approximation is used in practice, especially in trust region methods.

Properties of SR1

Does symmetric rank one update satisfy our demands?

We want every $\tilde{H}_k$ to be a symmetric positive definite solution to the secant condition.

Unfortunately, though $\tilde{H}_k$ is a symmetric positive definite, the updated matrix $\tilde{H}_{k+1}$ does not have to be a positive definite.

Still, the symmetric rank one approximation is used in practice, especially in trust region methods.

However, for line search, let us try a bit "richer" solution to the secant condition.

Symmetric Rank Two Update

Consider

$$\tilde{H}_{k+1} = \tilde{H}_k - \frac{\left(\tilde{H}_k s_k\right)\left(\tilde{H}_k s_k\right)^\top}{s_k^\top \tilde{H}_k s_k} + \frac{y_k y_k^\top}{y_k^\top s_k}$$

Once again, verifying $\tilde{H}_{k+1} s_k = y_k$ is not difficult.

Lemma 1

Assume that $\tilde{H}_k$ is symmetric positive definite.
Then $\tilde{H}_{k+1}$ is symmetric positive definite iff $y_k^\top s_k > 0$.

We know that line search satisfying the strong Wolfe conditions preserves $y_k^\top s_k > 0$.

Thus, starting with a symmetric positive definite $\tilde{H}_0$ (e.g., a scalar multiple of I), every $\tilde{H}_k$ is symmetric positive definite and satisfies the secant condition.

BFGS

Algorithm 11 BFGS v1

$k \leftarrow 0$
$\alpha_{\text{init}} \leftarrow 1$
$\tilde{H}_0 \leftarrow I$
while $\|\nabla f_k\|_\infty > \tau$ **do**
  Solve for p_k in $\tilde{H}_k p_k = -\nabla f_k$
  $\alpha \leftarrow \text{linesearch}(p_k, \alpha_{\text{init}})$
  $x_{k+1} \leftarrow x_k + \alpha p_k$
  $s \leftarrow x_{k+1} - x_k$
  $y \leftarrow \nabla f_{k+1} - \nabla f_k$
  $\tilde{H}_{k+1} \leftarrow \tilde{H}_k - \frac{(\tilde{H}_k s)(\tilde{H}_k s)^\top}{s^\top \tilde{H}_k s} + \frac{yy^\top}{y^\top s}$
  $k \leftarrow k + 1$
end while

Note that we still have to solve a linear system for p_k.

Example

Consider the quadratic problem $f(x) = \frac{1}{2}x^\top Qx - c^\top x$ with

$$Q = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 4 \end{pmatrix} \quad \text{and} \quad c = \begin{bmatrix} -8 \\ -9 \\ -8 \end{bmatrix},$$

whose solution is $x_* = (-4, -3, -2)^\top$. Use the exact line search.

Choose $\tilde{H}_0 = I$ and $x_0 = (0, 0, 0)^T$.

At iteration 0, $\|\nabla f(x_0)\|_\infty = 9$, so this point is not optimal.

The search direction is

$$p_0 = \begin{pmatrix} -8 \\ -9 \\ -8 \end{pmatrix}$$

and $\alpha_0 = 0.3333$.

Example

The new estimate of the solution and the new Hessian approximation are

$$x_1 = \begin{pmatrix} -2.6667 \\ -3.0000 \\ -2.6667 \end{pmatrix} \quad \text{and} \quad \tilde{H}_1 = \begin{pmatrix} 1.1021 & 0.3445 & 0.5104 \\ 0.3445 & 1.7751 & 1.0335 \\ 0.5104 & 1.0335 & 2.3270 \end{pmatrix}.$$

At iteration 1, $\|\nabla f(x_1)\|_\infty = 2.6667$, so we continue. The next search direction is

$$p_1 = \begin{pmatrix} -3.2111 \\ -0.6124 \\ 2.1223 \end{pmatrix}$$

and $\alpha_1 = 0.3577$.

Example

This gives the estimates.

$$x_2 = \begin{pmatrix} -3.8152 \\ -3.2191 \\ -1.9076 \end{pmatrix} \quad \text{and} \quad \tilde{H}_2 = \begin{pmatrix} 1.6393 & 0.6412 & -0.3607 \\ 0.6412 & 1.8600 & 0.6412 \\ -0.3607 & 0.6412 & 3.6393 \end{pmatrix}.$$

At iteration 2, $\|\nabla f(x_2)\|_\infty = 0.6572$, so we continue, computing

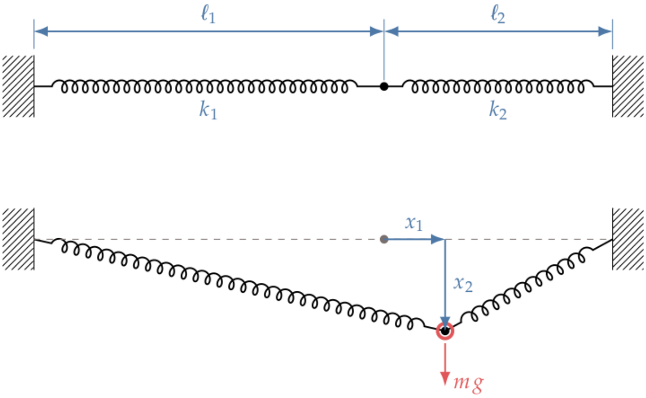
$$p_2 = \begin{pmatrix} -0.5289 \\ 0.6268 \\ -0.2644 \end{pmatrix}$$

and $\alpha_2 = 0.3495$. This gives

$$x_3 = \begin{pmatrix} -4 \\ -3 \\ -2 \end{pmatrix} \quad \text{and} \quad \tilde{H}_3 = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 4 \end{pmatrix}.$$

Now $\|\nabla f(x_3)\|_\infty = 0$, so we stop.

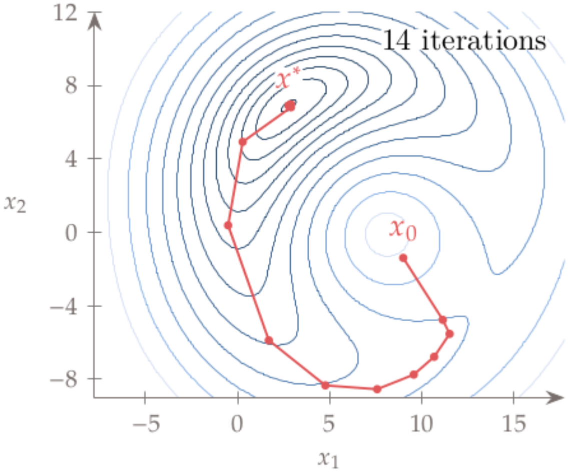
Notice that we got the same x_1, x_2, x_3 as for SR1. This follows from using the exact line search and the quadratic problem. It does not hold in general.



$$f(x_1, x_2) = \frac{1}{2}k_1\left(\sqrt{(\ell_1 + x_1)^2 + x_2^2} - \ell_1\right)^2 + \frac{1}{2}k_2\left(\sqrt{(\ell_2 - x_1)^2 + x_2^2} - \ell_2\right)^2 - mgx_2$$

Here $\ell_1 = 12, \ell_2 = 8, k_1 = 1, k_2 = 10, mg = 7$

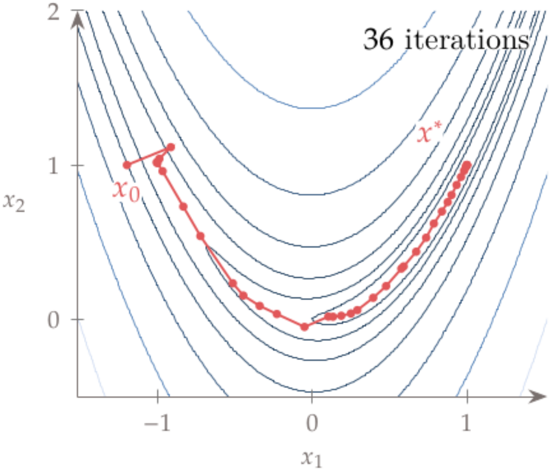
Two Spring Problem - BFGS



Gradient descent, line search, stop. cond. $||\nabla f||_\infty \leq 10^{-6}$.
Compare this with 32 iterations of gradient descent and 12 iterations of Newton's method.

Rosenbrock Function - BFGS

Rosenbrock: $f(x_1, x_2) = (1 - x_1)^2 + 100\left(x_2 - x_1^2\right)^2$



Gradient descent, line search, stop. cond. $||\nabla f||_\infty \leq 10^{-6}$.
Compare with 10,662 iterations of gradient descent and 24 iterations of Newton's method.

Sherman–Morrison–Woodbury Formula

Problem: SR1 and BFGS solve $\tilde{H}_k p = -\nabla f_k$ repeatedly. What if we could iteratively update H_k^{-1}?

Sherman–Morrison–Woodbury Formula

Problem: SR1 and BFGS solve $\tilde{H}_k p = -\nabla f_k$ repeatedly. What if we could iteratively update H_k^{-1}?

The equation would be solved by $p_k = -H_k^{-1} \nabla f_k$.

Sherman–Morrison–Woodbury Formula

Problem: SR1 and BFGS solve $\tilde{H}_k p = -\nabla f_k$ repeatedly. What if we could iteratively update H_k^{-1}?
The equation would be solved by $p_k = -H_k^{-1} \nabla f_k$.

Ideally, we would like to compute $\tilde{H}_k^{-1}$ iteratively along the optimization, i.e.,

$$\tilde{H}_{k+1}^{-1} = \tilde{H}_k^{-1} + \text{something}$$

Sherman–Morrison–Woodbury Formula

Problem: SR1 and BFGS solve $\tilde{H}_k p = -\nabla f_k$ repeatedly. What if we could iteratively update H_k^{-1}?
The equation would be solved by $p_k = -H_k^{-1} \nabla f_k$.

Ideally, we would like to compute $\tilde{H}_k^{-1}$ iteratively along the optimization, i.e.,

$$\tilde{H}_{k+1}^{-1} = \tilde{H}_k^{-1} + \text{something}$$

To get such a "something" we use the following Sherman–Morrison–Woodbury (SMW) formula:

$$\left(A + UV^T\right)^{-1} = A^{-1} - A^{-1} U \left(I + V^T A^{-1} U\right)^{-1} V^T A^{-1}$$

Here A is a $(n \times n)$-matrix, U, V are $(n \times m)$-matrices with $m \leq n$.

Rank 1 – Iterative Inverse Hessian Approximation

Applying SMW to the rank one update

$$\tilde{H}_{k+1} = \tilde{H}_k + \frac{\left(y_k - \tilde{H}_k s_k\right)\left(y_k - \tilde{H}_k s_k\right)^\top}{\left(y_k - \tilde{H}_k s_k\right)^\top s_k}$$

Rank 1 – Iterative Inverse Hessian Approximation

Applying SMW to the rank one update

$$\tilde{H}_{k+1} = \tilde{H}_k + \frac{\left(y_k - \tilde{H}_k s_k\right)\left(y_k - \tilde{H}_k s_k\right)^\top}{\left(y_k - \tilde{H}_k s_k\right)^\top s_k}$$

yields

$$\tilde{H}_{k+1}^{-1} = \tilde{H}_k^{-1} + \frac{\left(s_k - \tilde{H}_k^{-1} y_k\right)\left(s_k - \tilde{H}_k^{-1} y_k\right)^\top}{\left(s_k - \tilde{H}_k^{-1} y_k\right)^\top y_k}$$

Yes, only y and s swapped places.

Rank 1 – Iterative Inverse Hessian Approximation

Applying SMW to the rank one update

$$\tilde{H}_{k+1} = \tilde{H}_k + \frac{\left(y_k - \tilde{H}_k s_k\right)\left(y_k - \tilde{H}_k s_k\right)^\top}{\left(y_k - \tilde{H}_k s_k\right)^\top s_k}$$

yields

$$\tilde{H}_{k+1}^{-1} = \tilde{H}_k^{-1} + \frac{\left(s_k - \tilde{H}_k^{-1} y_k\right)\left(s_k - \tilde{H}_k^{-1} y_k\right)^\top}{\left(s_k - \tilde{H}_k^{-1} y_k\right)^\top y_k}$$

Yes, only y and s swapped places.

This allows us to avoid solving $\tilde{H}_k p_k = -\nabla f_k$ for p_k in every iteration.

Rank One Update V2

Algorithm 12 Rank 1 update v1

$k \leftarrow 0$
$\alpha_{\text{init}} \leftarrow 1$
$\tilde{H}_0 \leftarrow I$
while $\|\nabla f_k\|_\infty > \tau$ **do**
  $p_k \leftarrow -\tilde{H}_k^{-1} \nabla f_k$
  $\alpha \leftarrow \text{linesearch}(p_k, \alpha_{\text{init}})$
  $x_{k+1} \leftarrow x_k + \alpha p_k$
  $s \leftarrow x_k - x_{k-1}$
  $y \leftarrow \nabla f_k - \nabla f_{k-1}$
  $\tilde{H}_{k+1}^{-1} \leftarrow \tilde{H}_k^{-1} + \frac{\left(s - \tilde{H}_k^{-1} y\right)\left(s - \tilde{H}_k^{-1} y\right)^\top}{\left(s - \tilde{H}_k^{-1} y\right)^\top y}$
  $k \leftarrow k + 1$
end while

BFGS

Applying SMW to the BFGS Hessian update

$$\tilde{H}_{k+1} = \tilde{H}_k - \frac{\left(\tilde{H}_k s_k\right)\left(\tilde{H}_k s_k\right)^\top}{s_k^\top \tilde{H}_k s_k} + \frac{y_k y_k^\top}{y_k^\top s_k}$$

BFGS

Applying SMW to the BFGS Hessian update

$$\tilde{H}_{k+1} = \tilde{H}_k - \frac{\left(\tilde{H}_k s_k\right)\left(\tilde{H}_k s_k\right)^\top}{s_k^\top \tilde{H}_k s_k} + \frac{y_k y_k^\top}{y_k^\top s_k}$$

yields

$$\tilde{H}_{k+1}^{-1} = \left(I - \frac{s_k y_k^\top}{s_k^\top y_k}\right) \tilde{H}_k^{-1} \left(I - \frac{y_k s_k^\top}{s_k^\top y_k}\right) + \frac{s_k s_k^\top}{s_k^\top y_k}$$

We avoid solving the linear system for p_k.

BFGS V2

Algorithm 13 BFGS v2

$k \leftarrow 0$
$\alpha_{\text{init}} \leftarrow 1$
$\tilde{H}_0 \leftarrow I$
while $\|\nabla f_k\|_\infty > \tau$ **do**
    $p_k \leftarrow -\tilde{H}_k^{-1} \nabla f_k$
    $\alpha \leftarrow \text{linesearch}(p_k, \alpha_{\text{init}})$
    $x_{k+1} \leftarrow x_k + \alpha p_k$
    $k \leftarrow k + 1$
    $s \leftarrow x_k - x_{k-1}$
    $y \leftarrow \nabla f_k - \nabla f_{k-1}$
    $\tilde{H}_{k+1}^{-1} \leftarrow \left(I - \frac{sy^\top}{s^\top y}\right) \tilde{H}_k^{-1} \left(I - \frac{ys^\top}{s^\top y}\right) + \frac{ss^\top}{s^\top y}$
end while

Limited Memory BFGS Idea

Let us denote by $s_0, \ldots, s_k$ and $y_0, \ldots, y_k$ the values of the variables s and y, resp., during the iterations $1, \ldots, k$ of BFGS.

Limited Memory BFGS Idea

Let us denote by $s_0, \ldots, s_k$ and $y_0, \ldots, y_k$ the values of the variables s and y, resp., during the iterations $1, \ldots, k$ of BFGS.

Observe that $\tilde{H}_k$ is determined completely by H_0 and the two sequences $s_0, \ldots, s_k$ and $y_0, \ldots, y_k$.

Limited Memory BFGS Idea

Let us denote by $s_0, \ldots, s_k$ and $y_0, \ldots, y_k$ the values of the variables s and y, resp., during the iterations $1, \ldots, k$ of BFGS.

Observe that $\tilde{H}_k$ is determined completely by H_0 and the two sequences $s_0, \ldots, s_k$ and $y_0, \ldots, y_k$.

So, the matrix $\tilde{H}_k$ does not have to be stored if the algorithm remembers the values $s_0, \ldots, s_k$ and $y_0, \ldots, y_k$.

Note that this would be more space efficient for $k < n$.

Limited Memory BFGS Idea

Let us denote by $s_0, \ldots, s_k$ and $y_0, \ldots, y_k$ the values of the variables s and y, resp., during the iterations $1, \ldots, k$ of BFGS.

Observe that $\tilde{H}_k$ is determined completely by H_0 and the two sequences $s_0, \ldots, s_k$ and $y_0, \ldots, y_k$.

So, the matrix $\tilde{H}_k$ does not have to be stored if the algorithm remembers the values $s_0, \ldots, s_k$ and $y_0, \ldots, y_k$.

Note that this would be more space efficient for $k < n$.

However, we may go further and observe that typically only a few, say m, past values of s and y are sufficient for a good approximation of $\tilde{H}_k$ when we set $\tilde{H}_{k-m-1} = I$.

Limited Memory BFGS Idea

Let us denote by $s_0, \ldots, s_k$ and $y_0, \ldots, y_k$ the values of the variables s and y, resp., during the iterations $1, \ldots, k$ of BFGS.

Observe that $\tilde{H}_k$ is determined completely by H_0 and the two sequences $s_0, \ldots, s_k$ and $y_0, \ldots, y_k$.

So, the matrix $\tilde{H}_k$ does not have to be stored if the algorithm remembers the values $s_0, \ldots, s_k$ and $y_0, \ldots, y_k$.

Note that this would be more space efficient for $k < n$.

However, we may go further and observe that typically only a few, say m, past values of s and y are sufficient for a good approximation of $\tilde{H}_k$ when we set $\tilde{H}_{k-m-1} = I$.

This is the basic idea behind limited-memory BFGS which stores only the running window $s_{k-m}, \ldots, s_k$ and $y_{k-m}, \ldots, y_k$ and computes $\tilde{H}_k$ using these values as if initialized by $\tilde{H}_{k-m-1} = I$.

Limited Memory BFGS Idea

Let us denote by $s_0, \ldots, s_k$ and $y_0, \ldots, y_k$ the values of the variables s and y, resp., during the iterations $1, \ldots, k$ of BFGS.

Observe that $\tilde{H}_k$ is determined completely by H_0 and the two sequences $s_0, \ldots, s_k$ and $y_0, \ldots, y_k$.

So, the matrix $\tilde{H}_k$ does not have to be stored if the algorithm remembers the values $s_0, \ldots, s_k$ and $y_0, \ldots, y_k$.

Note that this would be more space efficient for $k < n$.

However, we may go further and observe that typically only a few, say m, past values of s and y are sufficient for a good approximation of $\tilde{H}_k$ when we set $\tilde{H}_{k-m-1} = I$.

This is the basic idea behind limited-memory BFGS which stores only the running window $s_{k-m}, \ldots, s_k$ and $y_{k-m}, \ldots, y_k$ and computes $\tilde{H}_k$ using these values as if initialized by $\tilde{H}_{k-m-1} = I$.

The space complexity becomes nm, which is beneficial when n is large.

Another View on BFGS (Optional)

We search for $\tilde{H}_{k+1}^{-1}$ where $\tilde{H}_{k+1}$ satisfies $\tilde{H}_{k+1} s_k = y_k$. Search for a solution $\tilde{V}$ for $\tilde{V} y_k = s_k$.

The idea is to use $\tilde{V}$ close to $\tilde{H}_k^{-1}$ (in some sense):

$$\min_{\tilde{H}} \left\| \tilde{V} - \tilde{H}_k^{-1} \right\|$$

$$\text{subject to} \quad \tilde{V} = \tilde{V}^\top, \quad \tilde{V} y_k = s_k$$

Here the norm is *weighted Frobenius norm*:

$$\|A\| \equiv \left\| W^{1/2} A W^{1/2} \right\|_F ,$$

where $\| \cdot \|_F$ is defined by $\|C\|_F^2 = \sum_{i=1}^n \sum_{j=1}^n c_{ij}^2$. The weight W can be chosen as any matrix satisfying the relation $W y_k = s_k$.

BFGS is obtained with $W = \bar{G}_k^{-1}$ where $\bar{G}_k$ is the average Hessian defined by $\bar{G}_k = \left[\int_0^1 \nabla^2 f\left(x_k + \tau \alpha_k p_k\right) d\tau \right]$

Solving this gives precisely the BFGS formula for $\tilde{H}_{k+1}^{-1}$.

Global Convergence of Line Search

Denote by θ_k the angle between p_k and $-\nabla f_k$, i.e., satisfying

$$\cos\theta_k = \frac{-\nabla f_k^T p_k}{\|\nabla f_k\| \, \|p_k\|}$$

Recall that f is L-smooth for some $L > 0$ if

$$\|\nabla f(x) - \nabla f(\tilde{x})\| \leq L\|x - \tilde{x}\|, \qquad \text{for all } x, \tilde{x} \in \mathbb{R}^n$$

Theorem 13 (Zoutendijk)

Consider $x_{k+1} = x_k + \alpha_k p_k$, where p_k is a descent direction *and α_k satisfies the* strong Wolfe conditions*. Suppose that f is bounded below, continuously differentiable, and* L-smooth*. Then*

$$\sum_{k\geq 0} \cos^2\theta_k \, \|\nabla f_k\|^2 < \infty.$$

Global Convergence of Quasi-Newton's Method

Assume that all α_k satisfy strong Wolfe conditions.

Assume that the approximations to the Hessians $\tilde{H}_k$ are positive definite with a uniformly bounded condition number:

$$\left|\left|\tilde{H}_k\right|\right| \left|\left|\tilde{H}_k^{-1}\right|\right| \leq M \quad \text{for all } k$$

Global Convergence of Quasi-Newton's Method

Assume that all α_k satisfy strong Wolfe conditions.

Assume that the approximations to the Hessians $\tilde{H}_k$ are positive definite with a uniformly bounded condition number:

$$\left|\left|\tilde{H}_k\right|\right| \left|\left|\tilde{H}_k^{-1}\right|\right| \leq M \quad \text{for all } k$$

Then θ_k between $p_k = -\tilde{H}_k^{-1}\nabla f_k$ and $-\nabla f_k$ and satisfies

$$\cos\theta_k \geq 1/M$$

Global Convergence of Quasi-Newton's Method

Assume that all α_k satisfy strong Wolfe conditions.

Assume that the approximations to the Hessians $\tilde{H}_k$ are positive definite with a uniformly bounded condition number:

$$\left|\left|\tilde{H}_k\right|\right| \left|\left|\tilde{H}_k^{-1}\right|\right| \leq M \quad \text{for all } k$$

Then θ_k between $p_k = -\tilde{H}_k^{-1}\nabla f_k$ and $-\nabla f_k$ and satisfies

$$\cos\theta_k \geq 1/M$$

Thus, under the assumptions of Zoutendijk's theorem, we obtain

$$\frac{1}{M^2}\sum_{k\geq 0}\|\nabla f_k\|^2 \leq \sum_{k\geq 0}\cos^2\theta_k \|\nabla f_k\|^2 < \infty$$

which implies that $\lim_{k\to\infty} ||\nabla f_k|| = 0$.

Behavior of BFGS

- It may happen that $\tilde{H}_k$ becomes a poor approximation of the Hessian H_k. If, e.g., $y_k^\top$ is tiny, then $\tilde{H}_{k+1}$ will be huge.

  However, it has been proven experimentally that if $\tilde{H}_k$ wrongly estimates the curvature of f and this estimate slows down the iteration, then the approximation will tend to correct the bad Hessian approximations.

  The above self-correction works only if an appropriate line search is performed (strong Wolfe conditions).

Behavior of BFGS

- It may happen that $\tilde{H}_k$ becomes a poor approximation of the Hessian H_k. If, e.g., $y_k^\top$ is tiny, then $\tilde{H}_{k+1}$ will be huge.

  However, it has been proven experimentally that if $\tilde{H}_k$ wrongly estimates the curvature of f and this estimate slows down the iteration, then the approximation will tend to correct the bad Hessian approximations.

  The above self-correction works only if an appropriate line search is performed (strong Wolfe conditions).

- There are more sophisticated ways of setting the initial Hessian approximation H_0.

  See Numerical Optimization, Nocedal & Wright, page 201.

Quasi-Newton Methods - Comments

- Each iteration is performed for $\mathcal{O}(n^2)$ operations as opposed to $\mathcal{O}(n^3)$ for methods involving solutions of linear systems.

Quasi-Newton Methods - Comments

- Each iteration is performed for $\mathcal{O}(n^2)$ operations as opposed to $\mathcal{O}(n^3)$ for methods involving solutions of linear systems.
- There is even a memory-limited variant (L-BFGS) that uses only information from past m steps, and its single iteration complexity is $\mathcal{O}(mn)$.

Quasi-Newton Methods - Comments

- Each iteration is performed for $\mathcal{O}(n^2)$ operations as opposed to $\mathcal{O}(n^3)$ for methods involving solutions of linear systems.
- There is even a memory-limited variant (L-BFGS) that uses only information from past m steps, and its single iteration complexity is $\mathcal{O}(mn)$.
- Compared with Newton's method, no second derivatives are computed.

Quasi-Newton Methods - Comments

- Each iteration is performed for $\mathcal{O}(n^2)$ operations as opposed to $\mathcal{O}(n^3)$ for methods involving solutions of linear systems.
- There is even a memory-limited variant (L-BFGS) that uses only information from past m steps, and its single iteration complexity is $\mathcal{O}(mn)$.
- Compared with Newton's method, no second derivatives are computed.
- Local superlinear convergence can be proved under specific conditions.

  Compare with local quadratic convergence of Newton's method and linear convergence of gradient descent.

Limited-Memory BFGS

Limited-Memory BFGS (L-BFGS)

When the number of design variables is extensive, working with the whole Hessian inverse approximation matrix might not be practical.

This motivates limited-memory quasi-Newton methods,

In addition, these methods also improve the computational efficiency of medium-sized problems (hundreds or thousands of design variables) with minimal sacrifice in accuracy.

L-BFGS

Recall that we compute iteratively the approximation to the inverse Hessian by

$$H_{k+1}^{-1} = \left(I - \frac{s_k y_k^\top}{s_k^\top y_k}\right) H_k^{-1} \left(I - \frac{y_k s_k^\top}{s_k^\top y_k}\right) + \frac{s_k s_k^\top}{s_k^\top y_k}$$

However, eventually, we are interested in

$$p_k = H_k^{-1} \nabla f$$

Note that given the sequences $s_1, \ldots, s_k$ and $y_1, \ldots, y_k$ and H_0^{-1} we can recursively compute H_{k+1}^{-1} for every k.

What if we limit the sequences in memory to just m last elements:

$$s_{k-m+1}, s_{k-m+2}, \ldots, s_k \qquad y_{k-m+1}, y_{k-m+2}, \ldots, y_k$$

In practice, m between 5 and 20 is usually sufficient. We also initialize the recurrence with the last iterate:

L-BFGS

Let us rewrite the BFGS update formula as follows:

$$\tilde{H}_{k+1}^{-1} = V_k^T \tilde{H}_k^{-1} V_k + \rho_k s_k s_k^\top$$

where

$$\rho_k = s_k^\top y_k \qquad \text{and} \qquad V_k = I - \rho_k s_k y_k^\top$$

$$s_k = x_{k+1} - x_k \qquad \text{and} \qquad y_k = \nabla f_{k+1} - \nabla f_k$$

By substitution, we obtain

$$\begin{aligned}
\tilde{H}_k^{-1} = & \left(V_{k-1}^T \cdots V_{k-m}^T\right) \tilde{H}_k^0 \left(V_{k-m} \cdots V_{k-1}\right) \\
& + \rho_{k-m} \left(V_{k-1}^T \cdots V_{k-m+1}^T\right) s_{k-m} s_{k-m}^T \left(V_{k-m+1} \cdots V_{k-1}\right) \\
& + \rho_{k-m+1} \left(V_{k-1}^T \cdots V_{k-m+2}^T\right) s_{k-m+1} s_{k-m+1}^T \left(V_{k-m+2} \cdots V_k\right. \\
& + \cdots \\
& + \rho_{k-1} s_{k-1} s_{k-1}^T
\end{aligned}$$

L-BFGS Algorithm

Algorithm 14 L-BFGS two-loop recursion

Input: : $s_{k-1}, \ldots, s_{k-m}$ and $y_{k-1}, \ldots, y_{k-m}$

Output: : p_k the search direction $-\tilde{H}_k^{-1} \nabla f_k$

$q \leftarrow \nabla f_k$
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
$r \leftarrow H_k^0 q$
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
stop with result $\tilde{H}_k^{-1} \nabla f_k = r$

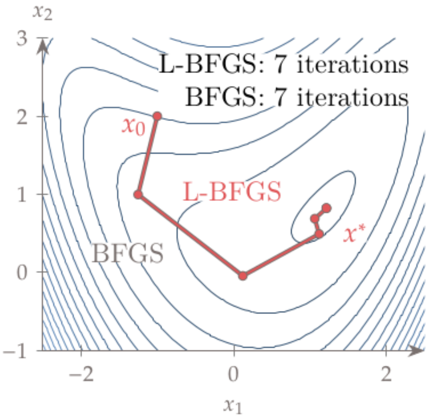
L-BFGS Algorithm

Algorithm 15 L-BFGS

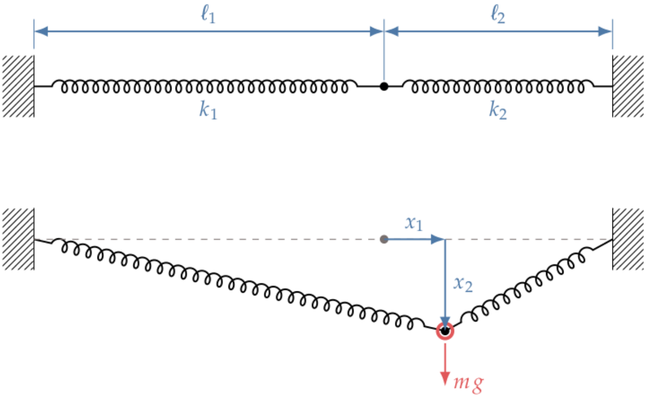
Choose starting point x_0, integer $m > 0$
$k \leftarrow 0$
repeat
  Choose H_k^0 e.g. $\frac{s_{k-1}^\top y_{k-1}}{y_{k-1}^\top y_{k-1}}$
  Compute $p_k \leftarrow -H_k \nabla f_k$ using the previous algorithm
  Compute $x_{k+1} \leftarrow x_k + \alpha_k p_k$, where α_k is chosen to satisfy the strong Wolfe conditions
  if $k > m$ **then**
    Discard the vector pair $\{s_{k-m}, y_{k-m}\}$ from storage
  end if
  Compute and save $s_k \leftarrow x_{k+1} - x_k$, $y_k \leftarrow \nabla f_{k+1} - \nabla f_k$
  $k \leftarrow k + 1$
until convergence

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

Stopping: $||\nabla f||_\infty \leq 10^{-6}$.

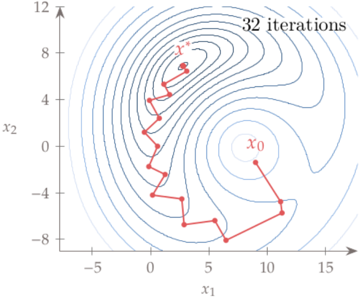


In L-BFGS, the memory length m was 5. The results are similar.

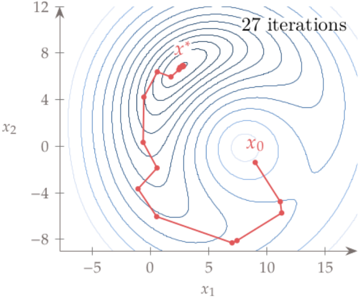


$$f(x_1, x_2) = \frac{1}{2}k_1\left(\sqrt{(\ell_1 + x_1)^2 + x_2^2} - \ell_1\right)^2 + \frac{1}{2}k_2\left(\sqrt{(\ell_2 - x_1)^2 + x_2^2} - \ell_2\right)^2 - mgx_2$$

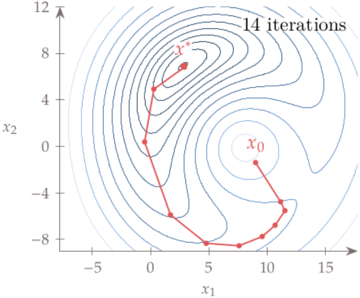
Here $\ell_1 = 12, \ell_2 = 8, k_1 = 1, k_2 = 10, mg = 7$



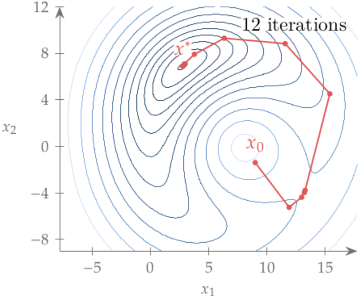
Steepest descent



Conjugate gradient

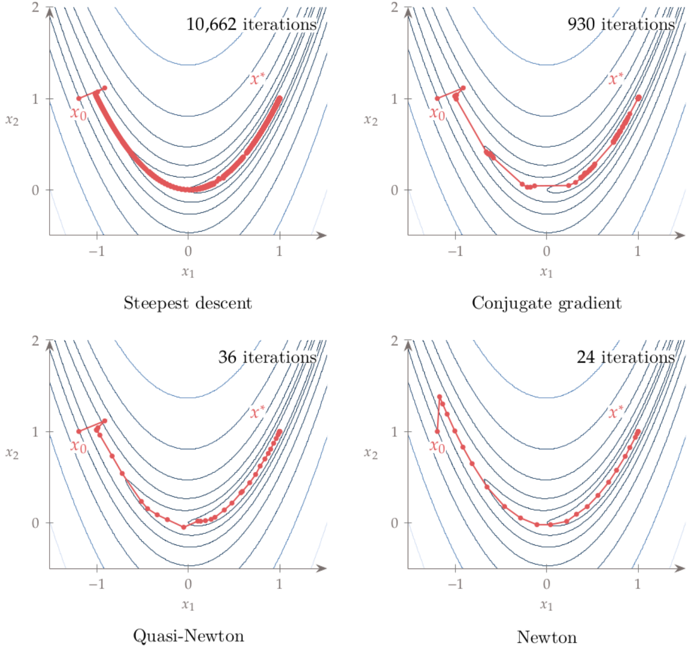


Quasi-Newton



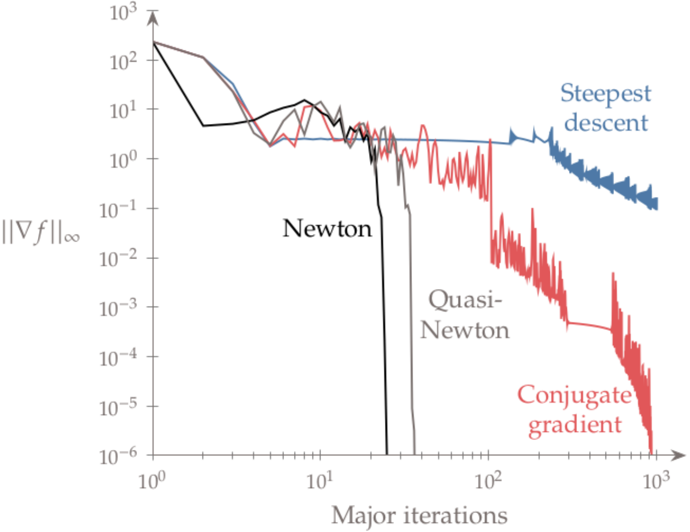
Newton

Rosenbrock: $f(x_1, x_2) = (1 - x_1)^2 + 100\left(x_2 - x_1^2\right)^2$



Rosenbrock:

$$f(x_1, x_2) = (1 - x_1)^2 + 100\left(x_2 - x_1^2\right)^2$$



Computational Complexity

Algorithm	Computational Complexity
Steepest Descent	$O(n)$ per iteration
Newton's Method	$O(n^3)$ to compute Hessian and solve system
BFGS	$O(n^2)$ to update Hessian approximation

Table: Summary of the computational complexity for each optimization algorithm.

- Steepest Descent: Simple but often slow, requiring many iterations.
- Newton's Method: Fast convergence but expensive per iteration.
- BFGS: Quasi-Newton, no Hessian needed, good speed and iteration count balance.