# (Quasi-)Newton methods

## 1 Introduction

### 1.1 Newton method

Newton method is a method to find the zeros of a differentiable non-linear function $g$, x such that $g(x) = 0$, where $g : \mathbb{R}^n \to \mathbb{R}^n$. Given a starting point $x_0$, Newton method consists in iterating:

$$x_{k+1} = x_k - g'(x_k)^{-1} g(x_k)$$

where $g'(x)$ is the derivative of $g$ at point $x$. Applying this method to the optimization problem:

$$\min_{x \in \mathbb{R}^n} f(x)$$

consists in setting $g(x) = \nabla f(x)$, i.e. looking for stationary points. The iterations read:

$$x_{k+1} = x_k - \nabla^2 f(x_k)^{-1} \nabla f(x_k) \ .$$

Newton method is particularly interesting as its convergence is quadratic locally around $x^*$, i.e.:

$$\|x_{k+1} - x^*\| \leq \gamma \|x_k - x^*\|^2, \gamma > 0 \ .$$

However the convergence is guaranteed only if $x_0$ is sufficiently close to $x^*$. The method may diverge if the initial point is too far from $x^*$ or if the Hessian is not positive definite. In order to address this issue of local convergence, Newton method can be combined with a line search method in the direction $d_k = -\nabla^2 f(x_k)^{-1} \nabla f(x_k)$ .

proof. We now prove the quadratic local convergence of Newton method.
TODO

### 1.2 Variable metric methods

The idea behind variable metric methods consists in using iterations of the form

$$\begin{cases} d_k = -B_k g_k \ , \\ x_{k+1} = x_k + \rho_k d_k \ , \end{cases}$$

where $g_k = \nabla f(x_k)$ and $B_k$ is a positive definite matrix. If $B_k = I_n$, it corresponds to gradient descent. Fixing $B_k = B$ leads to the following remark.

*Remark.* When minimizing

$$\min_{x \in \mathbb{R}^n} f(x)$$

one can set $x = Cy$ with $C$ invertible (change of variable). Let us denote $\tilde{f}(y) = f(Cy)$. This leads to:

$$\nabla \tilde{f}(y) = C^T f(Cy) \ .$$

Gradient descent applied to $\tilde{f}(y)$ reads:

$$y_{k+1} = y_k - \rho_k C^T \nabla f(C y_k)$$

which is equivalent to

$$x_{k+1} = x_k - \rho_k C C^T \nabla f(x_k) \ .$$

This amounts to using $B = CC^T$. In the case where $f$ is a quadratic form, it is straightforward to observe that this will improve convergence.

**Theorem 1.** *Let $f(x)$ a positive definite quadratic form and $B$ a positive definite matrix. Le preconditioned gradient algorithm:*

$$\begin{cases} x_0 = \textit{fixed}, \\ x_{k+1} = x_k - \rho_k B g_k, \ \ \rho_k \ \textit{optimal} \end{cases}$$

*has a linear convergence:*

$$\|x_{k+1} - x^*\| \leq \gamma \|x_k - x^*\|$$

*where:*

$$\gamma = \frac{\chi(BA) - 1}{\chi(BA) + 1} < 1 \ .$$

*Remark.* The lower the conditioning of BA, the faster is the algorithm. One cannot set $B = A^{-1}$ as it would implied having already solved the problem, but this however suggests to use $B$ so that is approximate $A^{-1}$. This is the idea behind quasi-Newton methods.

# 2 Quasi-Newton methods

## 2.1 Quasi-Newton relation

A quasi-Newton method reads

$$\begin{cases} d_k = -B_k g_k \ , \\ x_{k+1} = x_k + \rho_k d_k \ , \end{cases}$$

or

$$\begin{cases} d_k = -H_k^{-1} g_k \ , \\ x_{k+1} = x_k + \rho_k d_k \ , \end{cases}$$

where $B_k$ (resp. $H_k$) is a matrix which aims to approximate the inverse of the Hessian (resp. the Hessian) of $f$ at $x_k$. The question is how to achieve this? One can start with $B_0 = I_n$, but then how to update $B_k$ at every iteration? The idea is the following: by applying a Taylor expansion on the gradient, we know that at point $x_k$, the gradient and the Hessian are such that:

$$g_{k+1} = g_k + \nabla^2 f(x_k)(x_{k+1} - x_k) + \epsilon(x_{k+1} - x_k) \ .$$

If one assumes that the approximation is good enough one has:

$$g_{k+1} - g_k \approx \nabla^2 f(x_k)(x_{k+1} - x_k) \ ,$$

which leads to the quasi-Newton relation.

**Definition 1.** Two matrices $B_{k+1}$ and $H_{k+1}$ verify the quasi-Newton relation if:

$$H_{k+1}(x_{k+1} - x_k) = \nabla f(x_{k+1}) - \nabla f(x_k)$$

or

$$x_{k+1} - x_k = B_{k+1}(\nabla f(x_{k+1}) - \nabla f(x_k))$$

The follow up question is how to update $B_k$ while making sure that it says positive definite.

## 2.2 Update formula of Hessian

The update strategy at iteration

$$\begin{cases} d_k = -B_k g_k \ , \\ x_{k+1} = x_k + \rho_k d_k \ , \end{cases}$$

is to correct $B_k$ with a symmetric matrix $\Delta_k$:

$$B_{k+1} = B_k + \Delta_k$$

such that quasi-Newton relation holds:

$$x_{k+1} - x_k = B_{k+1}(g_{k+1} - g_k)$$

with $B_{k+1}$ positive definite, assuming $B_k$ is positive definite. For simplicity we will write $B_{k+1} > 0$. We will now see different formulas to define $\Delta_k$, under some assumptions that its rank is 1 or 2. We will speak about rank 1 or rank 2 corrections.

## 2.3 Broyden formula

Broyden formula is a rank 1 correction. Let us write

$$B_{k+1} = B_k + vv^T \ .$$

The vector $v \in \mathbb{R}^n$ should verify the quasi-Newton relation:

$$B_{k+1} y_k = s_k \, ,$$

where $y_k = g_{k+1} - g_k$ and $s_k = x_{k+1} - x_k$. It follows that:

$$B_k y_k + vv^T y_k = s_k \ ,$$

which leads by application of the dot product with $y_k$ to:

$$(y_k^T v)^2 = (s_k - B_k y_k)^T y_k \ ,$$

Using the equality

$$vv^T = \frac{vv^T y_k (vv^T y_k)^T}{(v^T y_k)^2},$$

one can write after replacing $vv^T y_k$ by $s_k - B_k y_k$, and $(v^T y_k)^2$ by $y_k^T (s_k - B_k y_k)$, the correction formula

$$B_{k+1} = B_k + \frac{(s_k - B_k y_k)(s_k - B_k y_k)^T}{(s_k - B_k y_k)^T y_k} \ ,$$

also known as Broyden formula.

**Theorem 2.** *Let $f$ a quadratic form positive definite. Let us consider the method that, starting for $x_0$, iterates:*

$$x_{k+1} = x_k + s_k \ ,$$

*where the vectors $s_k$ are linearly independent. Then the sequence of matrices starting by $B_0$ and defined as:*

$$B_{k+1} = B_k + \frac{(s_k - B_k y_k)(s_k - B_k y_k)^T}{(s_k - B_k y_k)^T y_k} \ ,$$

*where $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$, converges in less than n iterations towards $A^{-1}$, the inverse of the Hessian of f.*

PROOF. Since we consider here a quadratic function, the Hessian is constant and equal to A. This leads to:

$$y_i = \nabla f(x_{i+1}) - \nabla f(x_i), \forall i.$$

We have seen that $B_{k+1}$ is constructed such that:

$$B_{k+1} y_k = s_k \ .$$

Let us show that:

$$B_{k+1} y_i = s_i \, , i = 0, \ldots, k-1 \ .$$

By recurrence, let us assume that it is true for $B_k$:

$$B_k y_i = s_i \, , i = 0, \ldots, k-2 \ .$$

Let $i \leq k-2$. One has

$$B_{k+1} y_i = B_k y_i + \frac{(s_k - B_k y_k)(s_k^T y_i - B_k y_k^T y_i)}{(s_k - B_k y_k)^T y_k} \ . \tag{1}$$

By hypothesis, one has $B_k y_i = s_i$ which implies that $y_k^T B_k y_i = y_k^T s_i$, but since $A s_j = y_j$ for all j, it leads to:

$$y_k^T s_i = s_k^T A s_i = s_k^T y_i \ ,$$

The numerator in (1) is therefore 0, which leads to: $B_{k+1}y_i = B_k y_i = s_i$. One therefore has:

$$B_{k+1}y_i = s_i, \forall i = 0, \dots, k \ .$$

After n iterations, one has

$$B_n y_i = s_i, \forall i = 0, \dots, n-1 \ .$$

but since $y_i = As_i$, this last equation is equivalent to:

$$B_n A s_i = s_i, \forall i = 0, \dots, n-1 \ .$$

As the $s_i$ form a basis of $\mathbb{R}^n$ this implies that $B_n A = I_n$ or $B_n = A^{-1}$.

The issue with Broyden's formula is that there is no guarantee that the matrices $B_k$ are positive definite, even if the function $f$ is quadratic and $B_0 = I_n$. It is nevertheless interesting to have $B_n = A^{-1}$.

## 2.4 Davidon, Fletcher and Powell formula

The formula from Davidon, Fletcher and Powell is a rank 2 correction. It reads:

$$B_{k+1} = B_k + \frac{s_k s_k^T}{s_k^T y_k} - \frac{B_k y_k y_k^T B_k}{y_k^T B_k y_k} \ . \tag{2}$$

The following theorem states that under certain conditions, the formula guarantees to have $B_k$ positive definite.

**Theorem 3.** *Let us consider the update*

$$\begin{cases} d_k = -B_k g_k \ , \\ x_{k+1} = x_k + \rho_k B g_k, \ \ \rho_k \ \textit{optimal} \end{cases}$$

*where $B_0 > 0$ is provided as well as $x_0$. Then the matrices $B_k$ defined as in (2) are positive definite for all $k > 0$.*

PROOF. Let $x \in \mathbb{R}^n$. One has:

$$x^T B_{k+1} x = x^T B_k x + \frac{(s_k^T x)^2}{s_k^T y_k} - \frac{(y_k^T B_k x)^2}{y_k^T B_k y_k} \ . \ = \frac{y_k^T B_k y_k x^T B_k x - (y_k^T B_k x)^2}{y_k^T B_k y_k} + \frac{(s_k^T x)^2}{s_k^T y_k} \ . \tag{3}$$

If one defines the dot product $\langle x, y \rangle$ as $x^T B_k y$ the equation above reads:

$$x^T B_{k+1} x = \frac{\langle y_k, y_k \rangle \langle x, x \rangle - \langle y_k, x \rangle^2}{\langle y_k, y_k \rangle} + \frac{(s_k^T x)^2}{s_k^T y_k} \ .$$

The first term is positive by Cauchy-Schwartz inequality. Regarding the second term, as the step size is optimal one has:

$$g_{k+1}^T d_k = 0,$$

which implies

$$s_k^T y_k = \rho_k (g_{k+1} - g_k)^T d_k = \rho_k g_k^T B_k g_k > 0,$$

and so $x^T B_{k+1} x \geq 0$. Both terms being positive, the sum is zero only if both terms are 0. This implies that $x = \lambda y_k$ with $\lambda \neq 0$. In this case the second term cannot be zero as $s_k^T x = \lambda s_k^T y_k$. This implies that $B_{k+1} > 0$.

*Remark.* In the proof we used the fact that an optimal step size is used. The result still holds if one uses an approximate line search strategy for example using Wolf and Powell's rule. In this case the point $x_{k+1}$ is such that:

$$\phi'(\rho_k) = \nabla f(x_{k+1})^T d_k \geq m_2 \nabla f(x_k)^T d_k, 0 < m_2 < 1,$$

which guarantees:

$$g_{k+1}^T \frac{x_{k+1} - x_k}{\rho_k} > g_k^T \frac{x_{k+1} - x_k}{\rho_k},$$

and therefore $(g_{k+1} - g_k)^T (x_{k+1} - x_k) > 0$.

**Algorithm 1** Davidon-Fletcher-Powell algorithm

**Require:** $\varepsilon > 0$ (tolerance), $K$ (maximum number of iterations)

$x_0 \in \mathbb{R}^n$, $B_0 > 0$ (for example $I_n$)
**for** $k = 0$ to $K$ **do**
  **if** $\|g_k\| < \varepsilon$ **then**
    break
  **end if**
  $d_k = -B_k \nabla f(x_k)$
  Compute optimal step size $\rho_k$
  $x_{k+1} = x_k + \rho_k d_k$
  $s_k = \rho_k d_k$
  $y_k = g_{k+1} - g_k$
  $B_{k+1} = B_k + \frac{s_k s_k^T}{s_k^T y_k} - \frac{B_k y_k y_k^T B_k}{y_k^T B_k y_k}$
**end for**
**return** $x_{k+1}$

## 2.5 Davidon-Fletcher-Powell algorithm

One can know use the later formula in algorithm 1.

This algorithm has a remarkable property when the function $f$ is quadratic.

**Theorem 4.** *When $f$ is a quadratic form, the algorithm 1 generates a sequence of directions $s_0, \ldots, s_k$ which verify:*

$$\begin{aligned} s_i A^T s_j = 0, \quad & 0 \leq j < j \leq k+1, \\ B_{k+1} A s_i = s_i, \quad & 0 \leq i \leq k. \end{aligned} \tag{4}$$

PROOF. TODO

This theorem says that in the quadratic case, the algorithm 1 is like a conjugate gradient method, which therefore converges in at most $n$ iterations. One can also notice that for $k = n - 1$ the equalities

$$B_n A s_i = s_i, i = 0, \ldots, n-1,$$

and the fact that all $s_i$ are linearly independent imply that $B_n = A^{-1}$.

*Remark.* One can show that in the general case (non-quadratic), if the direction $d_k$ is reinitialized to $-g_k$ periodically, this algorithm converges to a local minimum $\hat{x}$ of $f$ and that:

$$\lim_{k \to \infty} B_k = \nabla^2 f(\hat{x})^{-1} \quad .$$

The implies that close to the optimum, if the line search is exact, the method behaves like a Newton method. This justifies the use of $\rho_k = 1$ when using approximate line search.

## 2.6 Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm

The BFGS formula is a correction formula of rank 2 which is derived from the formula of DFP by swapping the roles of $s_k$ and $y_k$. BFGS is named for the four people who independently discovered it in 1970: Broyden, Fletcher, Goldfarb and Shanno. The formula obtained allows to maintain an approximation $H_k$ of the Hessian which satisfies the same properties: $H_{k+1} > 0$ if $H_k > 0$ and satisfying the quasi-Newton relation:

$$y_k = H_k s_k.$$

The formula therefore reads:

$$H_{k+1} = H_k + \frac{y_k y_k^T}{y_k^T s_k} - \frac{H_k s_k s_k^T H_k}{s_k^T H_k s_k} \quad .$$

The algorithm is detailed in algorithm 2.

Note that the direction $d_k$ is obtained by solving a linear system. However in practice the update of $H_k$ is done on Cholesky factorization of $H_k = C_k C_k^T$ which implies that the complexity of BFGS is the same as DFP. The use of a Cholesky factorization is useful to check that $H_k$ states numerically positive definite, as this property can be lost due to numerical errors.

**Algorithm 2** Broyden-Davidon-Goldfarb-Shanno (BFGS) algorithm

**Require:** $\varepsilon > 0$ (tolerance), $K$ (maximum number of iterations)
$x_0 \in \mathbb{R}^n$, $H_0 > 0$ (for example $I_n$)
**for** $k = 0$ to $K$ **do**
  **if** $\|g_k\| < \varepsilon$ **then**
    break
  **end if**
  $d_k = -H_k^{-1} \nabla f(x_k)$
  Compute optimal step size $\rho_k$
  $x_{k+1} = x_k + \rho_k d_k$
  $s_k = \rho_k d_k$
  $y_k = g_{k+1} - g_k$
  $H_{k+1} = H_k + \frac{y_k y_k^T}{y_k^T s_k} - \frac{H_k s_k s_k^T H_k}{s_k^T H_k s_k}$
**end for**
**return** $x_{k+1}$

*Remark.* The BFGS algorithm has the same property as the DFP method: in the quadratic case it produces conjugate directions, converges in less than n iterations and $H_n = A$. Compared to DFP, BFGS convergence speed is much less sensitive to the use of approximate step size. It is therefore a particularly good candidate when combined with Wolfe and Powell's or Goldstein's rule.

*Remark.* The BFGS method is available in scipy as `scipy.optimize.fmin_bfgs`.

## 2.7 Limited-memory BFGS (L-BFGS) algorithm

The limited-memory BFGS (L-BFGS) algorithm is a variant of the BFGS algorithm that limits memory usage. While BFGS requires to store in memory a matrix of the size of the Hessian, $n \times n$, which can be prohibitive in applications such as computer vision or machine learning, the L-BFGS algorithm only stores a few vectors that are used to approximate the matrix $H_k^{-1}$. As a consequence the memory usage is linear in the dimension of the problem.

The L-BFGS is an algorithm of the quasi-Newton family with $d_k = -B_k \nabla f(x_k)$. The difference is in the computation of the product between $B_k$ and $\nabla f(x_k)$. The idea is to keep in memory the last few low rank corrections, more specifically the last $m$ updates $s_k = x_{k+1} - x_k$ and $y_k = g_{k+1} - g_k$. Often in practice $m < 10$, yet it might be necessary to modify this parameter on specific problems to speed up convergence. We denote by $\mu_k = \frac{1}{y_k^T s_k}$. The algorithm to compute the descent direction is given in algorithm 3.

**Algorithm 3** Direction finding in L-BFGS algorithm

**Require:** $m$ (memory size)
$q = g_k$
**for** $i = k - 1$ to $k - m$ **do**
  $\alpha_i = \mu_i s_i^T q$
  $q = q - \alpha_i y_i$
**end for**
$z = B_k^0 q$
**for** $i = k - m$ to $k - 1$ **do**
  $\beta = \mu_i y_i^T z$
  $z = z + s_i(\alpha_i - \beta)$
**end for**
$d_k = -z$

Commonly, the inverse Hessian $B_k^0$ is represented as a diagonal matrix, so that initially setting $z$ requires only an element-by-element multiplication. $B_k^0$ can change at each iteration but has however to be positive definite.

Like BFGS, L-BFGS does not need exact line search to converge. In machine learning, it is almost always the best approach to solve $\ell_2$ regularized Logistic regression and conditional random fields (CRF).

*Remark.* L-BFGS is for smooth unconstrained problem. Yet, it can be extended to handle simple box constraints (a.k.a. bound constraints) on variables; that is, constraints of the form $l_i \leq x_i \leq u_i$ where $l_i$ and $u_i$ are per-variable constant lower and upper bounds, respectively (for each $x_i$, either or both bounds may be omitted). This algorithm is called $L - BFGS - B$.

*Remark.* The L-BFGS-B algorithm is available in scipy as `scipy.optimize.fmin_l_bfgs_b`.

# 3 Methods specific to least squares

## 3.1 Gauss-Newton method

When considering least square problems the function to minimize reads:

$$f(x) = \frac{1}{2}\sum_{i=1}^{m} f_i(x)^2 \ .$$

Newton method can be applied to the minimization of $f$. The gradient and the Hessian matrix read in this particular case:

$$\nabla f(x) = \sum_{i=1}^{m} f_i(x)\nabla f_i(x) \ ,$$

and

$$\nabla^2 f(x) = \sum_{i=1}^{m} \nabla f_i(x)\nabla f_i(x)^T + \sum_{i=1}^{m} f_i(x)\nabla^2 f_i(x) \ .$$

If we are close to the optimum, where the $f_i(x)$ are assumed to be small the second term can be ignored. The matrix obtained reads:

$$H(x) = \sum_{i=1}^{m} \nabla f_i(x)\nabla f_i(x)^T \ .$$

This matrix is always positive. Furthermore when $m$ is much larger than $n$, this matrix is often positive definite. The Gauss-Newton method consists in using $H(x)$ in place of the Hessian. The method reads:

$$\begin{cases} x_0 = \text{fixed}, \\ H_k = \displaystyle\sum_{i=1}^{m} \nabla f_i(x_k)\nabla f_i(x_k)^T, \\ x_{k+1} = x_k - H_k^{-1}\nabla f(x_k) \ . \end{cases}$$

## 3.2 Levenberg-Marquardt method

In order to guarantee the convergence of the Gauss-Newton method, it can be combined with a line search procedure:

$$x_{k+1} = x_k - \rho_k H_k^{-1}\nabla f(x_k) \ .$$

However in order to guarantee that the $H_k$ stay positive definite a modified method, known as Levenberg-Marquardt, is often used. The idea is simply to replace $H_k$ by $H_k + \lambda I_n$. One can notice that if $\lambda$ is large, this method is equivalent to a simple gradient method. The Levenberg-Marquardt method reads:

$$\begin{cases} x_0 = \text{fixed}, \\ H_k = \displaystyle\sum_{i=1}^{m} \nabla f_i(x_k)\nabla f_i(x_k)^T, \\ d_k = -(H_k + \lambda I_n)^{-1}\nabla f(x_k) \\ x_{k+1} = x_k + \rho_k d_k \ . \end{cases}$$

*Remark.* The Levenberg-Marquardt method is available in scipy as `scipy.optimize.leastsq`.