Newton and Quasi-Newton algorithms

V. Leclère (ENPC)

April 16th, 2021

Why should I bother to learn this stuff ?

- Newton algorithm is, in theory, the best black box algorithm for smooth strongly convex function. It is used in practice as well as a stepping step for more advanced algorithm.
- Quasi-Newton algorithms (in particular L-BFGS) are the actual by default algorithm for most smooth black-box optimization library. Used in large scale application (e.g. weather forecast) for decades.
- $\Longrightarrow$ useful for
  - understanding the optimization software you might use as an engineer
  - understanding more advanced methods (e.g. interior points methods)
  - getting an idea of why the convergence might behave strangely in practice

Contents

Contents

Newton algorithm

Let f be $\mathcal{C}^2$ such that $\nabla^2 f(x) \succ 0$ for all x (so in particular strictly convex).
The Newton algorithm is a descent direction algorithm with :

- $d^{(k)} = -[\nabla^2 f(x^{(k)})]^{-1} \nabla f(x^{(k)})$
- $t^{(k)} = 1$

Note that

$$\nabla f(x^{(k)})^\top d^{(k)} = -\nabla f(x^{(k)})^\top [\nabla^2 f(x^{(k)})]^{-1} \nabla f(x^{(k)}) < 0$$

(unless $\nabla f(x^{(k)}) = 0$)
$\rightsquigarrow$ $d^{(k)}$ is a descent direction.

We are now going to give multiple justifications to this direction choice.

Newton algorithm

Let f be $\mathcal{C}^2$ such that $\nabla^2 f(x) \succ 0$ for all x (so in particular strictly convex).
The Newton algorithm is a descent direction algorithm with :

- $d^{(k)} = -[\nabla^2 f(x^{(k)})]^{-1} \nabla f(x^{(k)})$
- $t^{(k)} = 1$

Note that

$$\nabla f(x^{(k)})^\top d^{(k)} = -\nabla f(x^{(k)})^\top [\nabla^2 f(x^{(k)})]^{-1} \nabla f(x^{(k)}) < 0$$

(unless $\nabla f(x^{(k)}) = 0$)
$\rightsquigarrow$ $d^{(k)}$ is a descent direction.

We are now going to give multiple justifications to this direction choice.

Newton algorithm

Let f be $\mathcal{C}^2$ such that $\nabla^2 f(x) \succ 0$ for all x (so in particular strictly convex).
The Newton algorithm is a descent direction algorithm with :

- $d^{(k)} = -[\nabla^2 f(x^{(k)})]^{-1} \nabla f(x^{(k)})$
- $t^{(k)} = 1$

Note that

$$\nabla f(x^{(k)})^\top d^{(k)} = -\nabla f(x^{(k)})^\top [\nabla^2 f(x^{(k)})]^{-1} \nabla f(x^{(k)}) < 0$$

(unless $\nabla f(x^{(k)}) = 0$)
$\rightsquigarrow$ $d^{(k)}$ is a descent direction.

We are now going to give multiple justifications to this direction choice.

Second-order approximation minimization

We have

$$f(x^{(k)} + d) = f(x^{(k)}) + \nabla f(x^{(k)})^\top d + \frac{1}{2} d^\top \nabla^2 f(x^{(k)}) d + o(\|d\|^2)$$

The Newton method choose the direction d (with step 1) that minimize this second order approximation, which is given by

$$\nabla f(x^{(k)}) + \nabla^2 f(x^{(k)}) d^{(k)} = 0$$

$\rightsquigarrow$ The Newton method can be seen as a model-based method, where the model at iteration k is simply the second orde approximation.

$\rightsquigarrow$ A trust region method with confidence radius $+\infty$ is simply the Newton method.

Second-order approximation minimization

We have

$$f(x^{(k)} + d) = f(x^{(k)}) + \nabla f(x^{(k)})^\top d + \frac{1}{2} d^\top \nabla^2 f(x^{(k)}) d + o(\|d\|^2)$$

The Newton method choose the direction d (with step 1) that minimize this second order approximation, which is given by

$$\nabla f(x^{(k)}) + \nabla^2 f(x^{(k)}) d^{(k)} = 0$$

$\rightsquigarrow$ The Newton method can be seen as a model-based method, where the model at iteration k is simply the second orde approximation.

$\rightsquigarrow$ A trust region method with confidence radius $+\infty$ is simply the Newton method.

Second-order approximation minimization

We have

$$f(x^{(k)} + d) = f(x^{(k)}) + \nabla f(x^{(k)})^\top d + \frac{1}{2} d^\top \nabla^2 f(x^{(k)}) d + o(\|d\|^2)$$

The Newton method choose the direction d (with step 1) that minimize this second order approximation, which is given by

$$\nabla f(x^{(k)}) + \nabla^2 f(x^{(k)}) d^{(k)} = 0$$

$\rightsquigarrow$ The Newton method can be seen as a model-based method, where the model at iteration k is simply the second orde approximation.

$\rightsquigarrow$ A trust region method with confidence radius $+\infty$ is simply the Newton method.

Steepest descent with adaptative norm

- The Newton direction $d^{(k)}$ is the steepest descent direction for the quadratic norm associated to $\nabla^2 f(x^{(k)})$:

$$d^{(k)} = \arg\min_{d} \left\{ \nabla f(x^{(k)})^\top d \quad | \quad \|d\|_{\nabla f(x^{(k)})} \leq 1 \right\}$$

- Recall that the steepest gradient descent for a quadratic norm $\|\cdot\|_P$ converges rapidly if the condition number of the Hessian, after change of coordinate, is small.
- In particular a good choice near $x^\sharp$ is $P = \nabla f(x^\sharp)$.

$\rightsquigarrow$ fast around $x^\sharp$

Solution of linearized optimality condition

The optimality condition is given by

$$\nabla f(x^\sharp) = 0$$

We can linearize it as

$$\nabla f(x^{(k)} + d) \approx \nabla f(x^{(k)}) + \nabla^2 f(x^{(k)}) d = 0$$

And the Newton step $d^{(k)}$ is the solution of this linearization.

Affine invariance

- Recall that gradient and conjugate gradient method can be accelarated through smart affine change of variables (pre-conditionning).
- It is not the same for the Newton method:
  - Let A be an invertible matrix, and denote $y = Ax + b$, and $\tilde{f} : x \mapsto f(Ax + b)$.
  - $\nabla \tilde{f}(y) = A \nabla f(x)$ and $\nabla^2 \tilde{f}(y) = A^\top \nabla^2 f(x) A$
  - The Newton step for $\tilde{f}$ is thus

$$d_y = -(A^\top \nabla^2 f(x) A)^{-1} A \nabla f(x) = -A^{-1} (\nabla^2 f(x))^{-1} \nabla f(x) = A^{-1} d_x$$

  - Consequently

$$x^{(k+1)} - x^{(k)} = A(y^{(k+1)} - y^{(k)})$$

Contents

Damped Newton algorithm

Data: Initial point $x^{(0)}$, Second order oracle, error $\varepsilon > 0$.
while $\|\nabla f(x^{(k)})\| \geq \varepsilon$ **do**
  Solve for $d^{(k)}$

$$\nabla^2 f(x^{(k)}) d^{(k)} = -\nabla f(x^{(k)})$$

  Compute $t^{(k)}$ by backtracking line-search, starting from $t = 1$;
  $x^{(k+1)} = x^{(k)} + t^{(k)} d^{(k)}$

Algorithm 1: Damped Newton algorithm

- The Newton algorithm with fixed step size $t = 1$ is too numerically unstable, and you should always use a backtracking line-search.
- If the function is not strictly convex the Newton direction is not necessarily a descent direction, and you should check for it (and default to a gradient step).

Convergence idea

Assume that f is strongly convex, such that $mI \preceq \nabla^2 f(x) \preceq MI$, and that the Hessian $\nabla^2 f$ is L-Lipschitz.
We can show that there exists $0 < \eta \leq m^2/L$ and $\gamma > 0$ such that

- If $\|\nabla f(x^{(k)})\|_2 \geq \eta$, then

$$f(x^{(k+1)}) - f(x^{(k)}) \leq -\gamma$$

- If $\|\nabla f(x^{(k)})\|_2 < \eta$, then $t^{(k)} = 1$ and

$$\frac{L}{2m^2}\|\nabla f(x^{(k+1)})\|_2 \leq \left(\frac{L}{2m^2}\|\nabla f(x^{(k)})\|_2\right)^2$$

Newton is fast around the solution

We have, if $\|\nabla f(x^{(k)})\|_2 < \eta$, then $t^{(k)} = 1$ and

$$\frac{L}{2m^2}\|\nabla f(x^{(k+1)})\|_2 \leq \left(\frac{L}{2m^2}\|\nabla f(x^{(k)})\|_2\right)^2$$

Let $k = k_0 + \ell$, $\ell \geq 1$, with k_0 such that $\|\nabla f(x^{(k_0)})\|_2 < \eta$. Then $\|\nabla f(x^{(k)})\|_2 < \eta$, and,

$$\frac{L}{2m^2}\|\nabla f(x^{(k)})\|_2 \leq \left(\frac{L}{2m^2}\|\nabla f(x^{(k-1)})\|_2\right)^2$$

Recursively,

$$\frac{L}{2m^2}\|\nabla f(x^{(k)})\|_2 \leq \left(\frac{L}{2m^2}\|\nabla f(x^{(k_0)})\|_2\right)^{2^\ell} \leq \frac{1}{2^{2^\ell}}$$

And thus

$$f(x^{(k)}) - v^\sharp \leq \frac{1}{2m}\|\nabla f(x^{(k)})\|_2^2 \leq \frac{2m^3}{L^2}\frac{1}{2^{2^{\ell-1}}}$$

$\rightsquigarrow$ in the quadratic convergence phase, Newton's algorithm get the result in a few iterations (5 or 6).

Newton is fast around the solution

We have, if $\|\nabla f(x^{(k)})\|_2 < \eta$, then $t^{(k)} = 1$ and

$$\frac{L}{2m^2}\|\nabla f(x^{(k+1)})\|_2 \leq \left(\frac{L}{2m^2}\|\nabla f(x^{(k)})\|_2\right)^2$$

Let $k = k_0 + \ell$, $\ell \geq 1$, with k_0 such that $\|\nabla f(x^{(k_0)})\|_2 < \eta$. Then $\|\nabla f(x^{(k)})\|_2 < \eta$, and,

$$\frac{L}{2m^2}\|\nabla f(x^{(k)})\|_2 \leq \left(\frac{L}{2m^2}\|\nabla f(x^{(k-1)})\|_2\right)^2$$

Recursively,

$$\frac{L}{2m^2}\|\nabla f(x^{(k)})\|_2 \leq \left(\frac{L}{2m^2}\|\nabla f(x^{(k_0)})\|_2\right)^{2^\ell} \leq \frac{1}{2^{2^\ell}}$$

And thus

$$f(x^{(k)}) - v^\sharp \leq \frac{1}{2m}\|\nabla f(x^{(k)})\|_2^2 \leq \frac{2m^3}{L^2}\frac{1}{2^{2^{\ell-1}}}$$

$\rightsquigarrow$ in the quadratic convergence phase, Newton's algorithm get the result in a few iterations (5 or 6).

Newton is fast around the solution

We have, if $\|\nabla f(x^{(k)})\|_2 < \eta$, then $t^{(k)} = 1$ and

$$\frac{L}{2m^2}\|\nabla f(x^{(k+1)})\|_2 \leq \left(\frac{L}{2m^2}\|\nabla f(x^{(k)})\|_2\right)^2$$

Let $k = k_0 + \ell$, $\ell \geq 1$, with k_0 such that $\|\nabla f(x^{(k_0)})\|_2 < \eta$. Then $\|\nabla f(x^{(k)})\|_2 < \eta$, and,

$$\frac{L}{2m^2}\|\nabla f(x^{(k)})\|_2 \leq \left(\frac{L}{2m^2}\|\nabla f(x^{(k-1)})\|_2\right)^2$$

Recursively,

$$\frac{L}{2m^2}\|\nabla f(x^{(k)})\|_2 \leq \left(\frac{L}{2m^2}\|\nabla f(x^{(k_0)})\|_2\right)^{2^\ell} \leq \frac{1}{2^{2^\ell}}$$

And thus

$$f(x^{(k)}) - v^\sharp \leq \frac{1}{2m}\|\nabla f(x^{(k)})\|_2^2 \leq \frac{2m^3}{L^2}\frac{1}{2^{2^{\ell-1}}}$$

$\rightsquigarrow$ in the quadratic convergence phase, Newton's algorithm get the result in a few iterations (5 or 6).

Newton is fast around the solution

We have, if $\|\nabla f(x^{(k)})\|_2 < \eta$, then $t^{(k)} = 1$ and

$$\frac{L}{2m^2}\|\nabla f(x^{(k+1)})\|_2 \leq \left(\frac{L}{2m^2}\|\nabla f(x^{(k)})\|_2\right)^2$$

Let $k = k_0 + \ell$, $\ell \geq 1$, with k_0 such that $\|\nabla f(x^{(k_0)})\|_2 < \eta$. Then $\|\nabla f(x^{(k)})\|_2 < \eta$, and,

$$\frac{L}{2m^2}\|\nabla f(x^{(k)})\|_2 \leq \left(\frac{L}{2m^2}\|\nabla f(x^{(k-1)})\|_2\right)^2$$

Recursively,

$$\frac{L}{2m^2}\|\nabla f(x^{(k)})\|_2 \leq \left(\frac{L}{2m^2}\|\nabla f(x^{(k_0)})\|_2\right)^{2^\ell} \leq \frac{1}{2^{2^\ell}}$$

And thus

$$f(x^{(k)}) - v^\sharp \leq \frac{1}{2m}\|\nabla f(x^{(k)})\|_2^2 \leq \frac{2m^3}{L^2}\frac{1}{2^{2^{\ell-1}}}$$

$\rightsquigarrow$ in the quadratic convergence phase, Newton's algorithm get the result in a few iterations (5 or 6).

Convergence speed - Wrap-up

The Newton algorithm, for strongly convex function, have two phases :

- The damped phase, where $t^{(k)}$ can be less than 1. Each iteration yield an absolute improvement of $-\gamma < 0$.
- The quadratic phase, where each step $t^{(k)} = 1$.

Thus the total number of iteration to get an ε solution is bounded above by

$$\frac{f(x^{(0)}) - v^\sharp}{\gamma} + \underbrace{\log_2(\log_2(\varepsilon_0/\varepsilon))}_{\lesssim 6}$$

where $\varepsilon_0 = 2m^3/L^2$.

Note that, in 6 iterations in the quadratic convergent phase we get an error $\varepsilon \approx 5.10^{-20}\varepsilon_0$.

Convergence speed - Wrap-up

The Newton algorithm, for strongly convex function, have two phases :

- The damped phase, where $t^{(k)}$ can be less than 1. Each iteration yield an absolute improvement of $-\gamma < 0$.
- The quadratic phase, where each step $t^{(k)} = 1$.

Thus the total number of iteration to get an ε solution is bounded above by

$$\frac{f(x^{(0)}) - v^\sharp}{\gamma} + \underbrace{\log_2(\log_2(\varepsilon_0/\varepsilon))}_{\lesssim 6}$$

where $\varepsilon_0 = 2m^3/L^2$.

Note that, in 6 iterations in the quadratic convergent phase we get an error $\varepsilon \approx 5.10^{-20}\varepsilon_0$.

Convergence speed - Wrap-up

The Newton algorithm, for strongly convex function, have two phases :

- The damped phase, where $t^{(k)}$ can be less than 1. Each iteration yield an absolute improvement of $-\gamma < 0$.
- The quadratic phase, where each step $t^{(k)} = 1$.

Thus the total number of iteration to get an ε solution is bounded above by

$$\frac{f(x^{(0)}) - v^\sharp}{\gamma} + \underbrace{\log_2(\log_2(\varepsilon_0/\varepsilon))}_{\lesssim 6}$$

where $\varepsilon_0 = 2m^3/L^2$.

Note that, in 6 iterations in the quadratic convergent phase we get an error $\varepsilon \approx 5.10^{-20}\varepsilon_0$.

Contents

The main idea

Newton's step is the very efficient (near optimality) but have three drawbacks:

- having a second order oracle to compute the Hessian
- storing the Hessian (n^2 values)
- solving a (dense) linear system : $\nabla^2 f(x^{(k)})d = -\nabla f(x^{(k)})$

The main idea of Quasi Newton method is to construct, from first order informations, a sequence of matrix $M^{(k)}$, if possible sparse, that approximate the (inverse of) the Hessian.

Conditions on the approximate Hessian I ♢

We want to construct $M^{(k)}$ an approximation of $\nabla^2 f(x^{(k)})$, leading to a quadratic model of f at iteration k

$$f^{(k)}(x) := f(x^{(k)}) + \left\langle \nabla f(x^{(k)}), x - x^{(k)} \right\rangle + \frac{1}{2}(x - x^{(k)})^\top M^{(k)}(x - x^{(k)})$$

We ask that the gradient of the model $f^{(k)}$ and the through function matches in current and last iterates:

$$\begin{cases} \nabla f^{(k)}(x^{(k)}) = \nabla f(x^{(k)}) \\ \nabla f^{(k)}(x^{(k-1)}) = \nabla f(x^{(k-1)}) \end{cases}$$

This simply write as the Quasi-Newton equation

$$M^{(k)} \underbrace{(x^{(k)} - x^{(k-1)})}_{\delta_x^{(k-1)}} = \underbrace{\nabla f(x^{(k)}) - \nabla f(x^{(k-1)})}_{\delta_g^{(k-1)}}$$

♣ Exercise: prove it

Conditions on the approximate Hessian I ♢

We want to construct $M^{(k)}$ an approximation of $\nabla^2 f(x^{(k)})$, leading to a quadratic model of f at iteration k

$$f^{(k)}(x) := f(x^{(k)}) + \left\langle \nabla f(x^{(k)}), x - x^{(k)} \right\rangle + \frac{1}{2}(x - x^{(k)})^\top M^{(k)}(x - x^{(k)})$$

We ask that the gradient of the model $f^{(k)}$ and the through function matches in current and last iterates:

$$\begin{cases} \nabla f^{(k)}(x^{(k)}) = \nabla f(x^{(k)}) \\ \nabla f^{(k)}(x^{(k-1)}) = \nabla f(x^{(k-1)}) \end{cases}$$

This simply write as the Quasi-Newton equation

$$M^{(k)} \underbrace{(x^{(k)} - x^{(k-1)})}_{\delta_x^{(k-1)}} = \underbrace{\nabla f(x^{(k)}) - \nabla f(x^{(k-1)})}_{\delta_g^{(k-1)}}$$

♣ Exercise: prove it

Conditions on the approximate Hessian I ♢

We want to construct $M^{(k)}$ an approximation of $\nabla^2 f(x^{(k)})$, leading to a quadratic model of f at iteration k

$$f^{(k)}(x) := f(x^{(k)}) + \left\langle \nabla f(x^{(k)}), x - x^{(k)} \right\rangle + \frac{1}{2}(x - x^{(k)})^\top M^{(k)}(x - x^{(k)})$$

We ask that the gradient of the model $f^{(k)}$ and the through function matches in current and last iterates:

$$\begin{cases} \nabla f^{(k)}(x^{(k)}) = \nabla f(x^{(k)}) \\ \nabla f^{(k)}(x^{(k-1)}) = \nabla f(x^{(k-1)}) \end{cases}$$

This simply write as the Quasi-Newton equation

$$M^{(k)} \underbrace{(x^{(k)} - x^{(k-1)})}_{\delta_x^{(k-1)}} = \underbrace{\nabla f(x^{(k)}) - \nabla f(x^{(k-1)})}_{\delta_g^{(k-1)}}$$

♣ Exercise: prove it

Conditions on the approximate Hessian II ♢

We are looking for a matrix M such that

- $M \succ 0$
- $M\delta_x = \delta_g$ (only possible if $\delta_g^\top \delta_x > 0$ ♣ Exercise: prove it)
- $M^\top = M$
- M is constructed from first order informations only
- If possible, M is sparse

$\rightsquigarrow$ an infinite number of solutions as we have $n(n+1)/2$ variables and n constraints.

$\rightsquigarrow$ a large number of quasi-Newton algorithms developped and tested between 1960-1980.

Conditions on the approximate Hessian II ♢

We are looking for a matrix M such that

- $M \succ 0$
- $M\delta_x = \delta_g$ (only possible if $\delta_g^\top \delta_x > 0$ ♣ Exercise: prove it)
- $M^\top = M$
- M is constructed from first order informations only
- If possible, M is sparse

$\rightsquigarrow$ an infinite number of solutions as we have $n(n+1)/2$ variables and n constraints.

$\rightsquigarrow$ a large number of quasi-Newton algorithms developped and tested between 1960-1980.

Conditions on the approximate Hessian II ♢

We are looking for a matrix M such that

- $M \succ 0$
- $M\delta_x = \delta_g$ (only possible if $\delta_g^\top \delta_x > 0$ ♣ Exercise: prove it)
- $M^\top = M$
- M is constructed from first order informations only
- If possible, M is sparse

$\rightsquigarrow$ an infinite number of solutions as we have $n(n+1)/2$ variables and n constraints.

$\rightsquigarrow$ a large number of quasi-Newton algorithms developped and tested between 1960-1980.

Choosing the approximate Hessian $M^{(k)}$

At the end of iteration k we have determined

- $x^{(k+1)}$ and $\delta_x^{(k)} = x^{(k+1)} - x^{(k)}$
- $g^{(k+1)} = \nabla f(x^{(k)})$ and $\delta_g^{(k)} = g^{(k+1)} - g^{(k)}$

and we are looking for $M^{(k+1)} \approx \nabla^2 f(x^{(k+1)}$ satisfying the previous requirement.

The idea is to choose $M^{(k+1)}$ close to $M^{(k)}$, that is to solve (analytically)

$$\begin{aligned} \underset{M \in S_{++}^n}{\text{Min}} \quad & d(M, M^{(k)}) \\ s.t. \quad & M\delta_x^{(k)} = \delta_g^{(k)} \end{aligned}$$

for some distance d.

Choosing the approximate Hessian $M^{(k)}$

At the end of iteration k we have determined

- $x^{(k+1)}$ and $\delta_x^{(k)} = x^{(k+1)} - x^{(k)}$
- $g^{(k+1)} = \nabla f(x^{(k)})$ and $\delta_g^{(k)} = g^{(k+1)} - g^{(k)}$

and we are looking for $M^{(k+1)} \approx \nabla^2 f(x^{(k+1)}$ satisfying the previous requirement.

The idea is to choose $M^{(k+1)}$ close to $M^{(k)}$, that is to solve (analytically)

$$\begin{aligned} \underset{M \in S_{++}^n}{\text{Min}} \quad & d(M, M^{(k)}) \\ s.t. \quad & M\delta_x^{(k)} = \delta_g^{(k)} \end{aligned}$$

for some distance d.

BFGS

Broyden-Fletcher-Goldfarb-Shanno chose

$$d(A, B) := \mathrm{tr}(AB) - \ln\det(AB)$$

A few remarks

- $\Psi : M \mapsto \mathrm{tr}\, M - \ln\det(M)$ is convex on S^n_{++}
- For $M \in S^n_{++}$, $\mathrm{tr}\, M - \ln\det(M) = \sum_{i=1}^n \lambda_i - \ln(\lambda_i)$
- Ψ is minimized in the identity matrix
- $d(A, B) - n$ is the Kullback-Lieber divergence between $\mathcal{N}(0, A)$ and $\mathcal{N}(0, B)$

BFGS update

One of the pragmatic reason for this choice of distance is that the optimal solution can be found analytically.
We have (to alleviate notation we drop the index k on $\delta_x^{(k)}$ and $\delta_g^{(k)}$)

$$M^{(k+1)} = M^{(k)} + \frac{\delta_g \delta_g^\top}{\delta_g^\top \delta_x} - \frac{M^{(k)} \delta_x \delta_x^\top M^{(k)}}{\delta_x^\top M^{(k)} \delta_x}$$

Even better, denoting $W = M^{-1}$,

$$W^{(k+1)} = \left(I - \frac{\delta_x \delta_g^\top}{\delta_g^\top \delta_x}\right) W^{(k)} \left(I - \frac{\delta_g \delta_x^\top}{\delta_g^\top \delta_x}\right) + \frac{\delta_x \delta_x^\top}{\delta_g^\top \delta_x}$$

BFGS update

One of the pragmatic reason for this choice of distance is that the optimal solution can be found analytically.
We have (to alleviate notation we drop the index k on $\delta_x^{(k)}$ and $\delta_g^{(k)}$)

$$M^{(k+1)} = M^{(k)} + \frac{\delta_g \delta_g^\top}{\delta_g^\top \delta_x} - \frac{M^{(k)} \delta_x \delta_x^\top M^{(k)}}{\delta_x^\top M^{(k)} \delta_x}$$

Even better, denoting $W = M^{-1}$,

$$W^{(k+1)} = \left(I - \frac{\delta_x \delta_g^\top}{\delta_g^\top \delta_x}\right) W^{(k)} \left(I - \frac{\delta_g \delta_x^\top}{\delta_g^\top \delta_x}\right) + \frac{\delta_x \delta_x^\top}{\delta_g^\top \delta_x}$$

BFGS algorithm

Data: Initial point $x^{(0)}$, First order oracle, error $\varepsilon > 0$.
$W^{(0)} = I$; **while** $\|\nabla f(x^{(k)})\| \geq \varepsilon$ **do**
  $g^{(k)} := \nabla f(x^{(k)})$;
  $d^{(k)} := -W^{(k)} g^{(k)}$;
  Compute $t^{(k)}$ by backtracking line-search, starting from $t = 1$;
  $x^{(k+1)} = x^{(k)} + t^{(k)} d^{(k)}$;
  $\delta_g = g^{(k+1)} - g^{(k)}$, $\delta_x = x^{(k+1)} - x^{(k)}$;
  $W^{(k+1)} = \left(I - \frac{\delta_x \delta_g^\top}{\delta_g^\top \delta_x}\right) W^{(k)} \left(I - \frac{\delta_g \delta_x^\top}{\delta_g^\top \delta_x}\right) + \frac{\delta_x \delta_x^\top}{\delta_g^\top \delta_x}$;
  $k = k + 1$;

Algorithm 2: BFGS algorithm

- First order oracle only
- No need to solve a linear system
- Still large memory requirement

Limited-memory BFGS (L-BFGS)

- For $n \geq 10^3$ storing the matrices is a difficulty.
- Instead of storing and updating the matrix $W^{(k)}$ we store (δ_x, δ_g) pairs.
- We can then compute $d^{(k)} = -W^{(k)} g^{(k)}$ directly from the last 5 to 20 pairs, using recursively the update rule and never computing $W^{(k)}$.

$\rightsquigarrow$ an algorithm with rougly the same storage requirement as gradient algorithm, and convergence almost equivalent to Newton method.

$\rightsquigarrow$ this is the "go to" algorithm when you want high level precision for strongly convex smooth problem. It is the default choice in a lot of optimization libraries.

Limited-memory BFGS (L-BFGS)

- For $n \geq 10^3$ storing the matrices is a difficulty.
- Instead of storing and updating the matrix $W^{(k)}$ we store (δ_x, δ_g) pairs.
- We can then compute $d^{(k)} = -W^{(k)} g^{(k)}$ directly from the last 5 to 20 pairs, using recursively the update rule and never computing $W^{(k)}$.

$\rightsquigarrow$ an algorithm with rougly the same storage requirement as gradient algorithm, and convergence almost equivalent to Newton method.

$\rightsquigarrow$ this is the "go to" algorithm when you want high level precision for strongly convex smooth problem. It is the default choice in a lot of optimization libraries.

Limited-memory BFGS (L-BFGS)

- For $n \geq 10^3$ storing the matrices is a difficulty.
- Instead of storing and updating the matrix $W^{(k)}$ we store (δ_x, δ_g) pairs.
- We can then compute $d^{(k)} = -W^{(k)} g^{(k)}$ directly from the last 5 to 20 pairs, using recursively the update rule and never computing $W^{(k)}$.

$\rightsquigarrow$ an algorithm with rougly the same storage requirement as gradient algorithm, and convergence almost equivalent to Newton method.

$\rightsquigarrow$ this is the "go to" algorithm when you want high level precision for strongly convex smooth problem. It is the default choice in a lot of optimization libraries.

What you have to know

- At least one idea behind Newton's algorithm.
- The Newton step.
- That quasi-Newton methods are almost as good as Newton, without requiring a second order oracle.

What you really should know

- Newton's algorithm default step is 1, but you should use backtracking step anyway.
- Newton's algorithm converges in two phases : a slow damped phase, and a very fast quadratically convergent phase close to the optimum (at most 6 iterations).
- BFGS is the by default quasi-Newton method. It work by updating an approximation of the inverse of the Hessian close to the precedent approximation and satisfying some natural requirement.
- L-BFGS limit the memory requirement by never storing the matrix but only the step and gradient updates.

What you have to be able to do

- Implement a damped Newton method.

What you should be able to do

- Implement a BFGS method (with the update formula in front of your eyes)