Quasi-Newton Methods

Thus far we have worked with first order methods — those that use first derivative information to determine good descent directions.

Method	Pros	Cons
Gradient Descent	Simple implementation best local descent ok for small cond #	poor global convergence rate easily stalls
Conjugate Gradient	Superlinear convergence minimal extra computations minimal memory footprint	requires periodic restarts
Newton's Method (full second order)	quadratic convergence	Computationally expensive large memory footprint requires 2^{nd} derivative information

We next turn our attention to approximate second order methods that seek to gain good convergence without direct second order information.

Nesterov acceleration was a hopeful method but has not shown superior performance to conjugate gradient or quasi-Newton methods.

quasi-Newton methods approximate the hessian using curvature information derived from recent gradient computations.

We can see the concept by considering a small 1D example.

$$\min f(x) = \sin^2 x \quad (\text{near } x=0)$$

We know the solution is $x^* = 0$. First employ gradient descent with backtracking and the Armijo condition, starting from $x_0 = 0.2$ we find:

k	x_k	$f(x_k)$	$f'(x_k)$
0	+0.2	0.03947	+0.38942
1	−0.18942	0.03545	−0.36984
2	+0.18042	0.03200	+0.35306
3	−0.17264	0.29510	−0.33846

In this example, the performance is poor! Now consider a quasi-Newton method. We begin with a gradient descent step, so $k=0$ and $k=1$ iterations are exactly as above. But now we consider a quadratic model function near x_1

$$x_2 = x_1 + p \quad \text{and } p \text{ minimizes } m(p) = ap^2 + bp + c \quad (\text{hopefully with } a>0)$$

and require that $m(0) = f(x_1)$, $m'(0) = f'(x_1)$ and $m'(x_0 - x_1) = f'(x_0)$. That is, we ask that the model correctly recover the recent first order information. We have:

$$m(0) = c \overset{\text{set}}{=} 0.03545$$

$$m'(0) = b \overset{\text{set}}{=} -0.36984$$

$$m'(x_0 - x_1) = (2a)(0.38942) + b \overset{\text{set}}{=} 0.38942$$

which has unique solution $a = 0.97486$, $b = -0.36984$, $c = 0.03545$

$$m(p) = (0.97486)p^2 - (0.36984)p + 0.03545$$

Using this model, we try a Newton step as the initial linesearch step:

$$x_2 = x_1 + p \quad \text{where } p = -b/2a = 0.18975$$

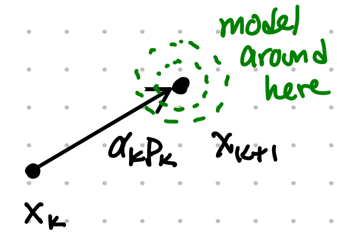
$$x_2 = -0.18942 + 0.18975 = 0.00033$$

With this idea, we have

$$x_2 = 3.3 \times 10^{-4} \qquad f(x_2) = 1.1 \times 10^{-7} \qquad f'(x_2) = 6.6 \times 10^{-4}$$

Dramatically better than gradient descent

How can we implement this idea in $\mathbb{R}^n$? When seeking p_{k+1}:



$$m_{k+1}(p) = \frac{1}{2} p^T B_{k+1} p + \nabla f(x_{k+1})^T p + f(x_{k+1})$$

$$\Rightarrow \nabla m_{k+1}(p) = B_{k+1} p + \nabla f(x_{k+1})$$

$$\nabla m_{k+1}(0) = \nabla f(x_{k+1}) \quad \text{(satisfied trivially)}$$

$$\nabla m_{k+1}(-\alpha_k p_k) = \nabla f(x_k) \quad \text{(enforce)}$$

$$\Downarrow$$

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

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

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

Secant equation

We force B to be symmetric positive definite. When p_k satisfies the curvature condition (second Wolfe condition) then the secant equation has a solution. But the solution is not unique. This fact leads to the possibility of having infinitely many possible update algorithms. The one that has proven to be best overall (and over decades of use) is the BFGS update (Broyden Fletcher Goldfarb Shanno).

Instead, use the inverse matrix $H = B^{-1}$ and find

$$H_{k+1} = \underset{J}{\operatorname{argmin}} \ \| J - H_k \|_F$$

$$\text{s.t. } J = J^T, \ J y_k = s_k$$

The analytic result is:

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

new updated inverse hessian

Previous inverse hessian

Now, we can try an approximate Newton Step!

BFGS Algorithm

Given : $x_0 \in \mathbb{R}^n$, $f : \mathbb{R}^n \to \mathbb{R}$, $\varepsilon > 0$

Set : H_0 (initial hessian approximation) — $H_0 = I_n$ is easy and works*

$k \leftarrow 0$

Step 1: If $\|\nabla f(x_k)\| < \varepsilon$ then stop. — Or other stopping conditions

Step 2: Search direction $P_k = -H_k \nabla f(x_k)$ — Newton step with approx hessian

Step 3: Solve $\alpha_k = \underset{\beta}{\operatorname{argmin}} \; f(x_k + \beta P_k)$ — line search with strong Wolfe

Step 4: Updates :

$s_k = x_{k+1} - x_k \quad (= \alpha_k P_k)$ — step vector

$y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$ — 2nd order information!

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

$k \leftarrow k+1$

Go to step 1.

H update may not result in positive definite property unless strong Wolfe is used!

(technically $y^T s > 0$ is the guarantee)

* A simple alternative H_0 :

Set $H_0 = |f(x_0)| \, I_n$ or $H_0 = |\text{typf}| \, I_n$

Hopefully captures some initial scale information.

User supplied value

Best when $f(x) \geq 0$ and $f(x^*) \sim 0$

Example: $n = 10$ Rosenbrock Function with steepness parameter 10.

$$x_0 = \left(\tfrac{1}{10}, \tfrac{2}{10}, \dots, \tfrac{9}{10}, 1\right) \qquad x^* = (1, 1, 1, \dots, 1)$$

	GD (Armijo)	GD (SW)	ConjGrad	BFGS
# iter	620	665	78	26
# f	4534	5563	648	57
# g	620	1368	194	51
"evals"	10,734	19,243	2588	567
$\lVert x_k - x_* \rVert$	6.1 E-3	3.2 E-3	5.5E-5	8.4 E-5
$\lVert g_k \rVert$	8.8 E-3	4.5E-3	6.0 E-4	1.1 E-6

Conj Grad and BFGS have more expensive computational updates.

BFGS must store an $n \times n$ matrix.