The BFGS Optimization Algorithm

Pratham Lalwani

UC Merced

May 13, 2025

Outline

Problem Setup

- Given $f : \mathbb{R}^n \to \mathbb{R}$, say we are interested is minimizing the function, which is,
$$\min_{\boldsymbol{x} \in \mathbb{R}^n} f(\boldsymbol{x}).$$
- From calculus, $\nabla f(\boldsymbol{x}) = \mathbf{0}$ and solve analytically if it can be done.
- This problem arises everywhere especially nowadays with Machine Learning where $f(\boldsymbol{x})$ is usually a cost function we are trying to minimize.

Gradient Descent and Its Limitations

- With no way to compute a analytic solution one might turn a simple algorithm like Gradient Descent.

Gradient Descent and Its Limitations

- With no way to compute a analytic solution one might turn a simple algorithm like Gradient Descent.
- The next iterate is given by : $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \gamma \nabla f(\boldsymbol{x}_k)$, where γ is a fixed constant called step size or learning rate.

Gradient Descent and Its Limitations

- With no way to compute a analytic solution one might turn a simple algorithm like Gradient Descent.
- The next iterate is given by : $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \gamma \nabla f(\boldsymbol{x}_k)$, where γ is a fixed constant called step size or learning rate.
- Pros: simple, easy to implement and not computationally expensive (per step)

Gradient Descent and Its Limitations

- With no way to compute a analytic solution one might turn a simple algorithm like Gradient Descent.
- The next iterate is given by : $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \gamma \nabla f(\boldsymbol{x}_k)$, where γ is a fixed constant called step size or learning rate.
- Pros: simple, easy to implement and not computationally expensive (per step)
- Cons: slow convergence, sensitivity to step size.

Gradient Descent and Its Limitations

- With no way to compute a analytic solution one might turn a simple algorithm like Gradient Descent.
- The next iterate is given by : $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \gamma \nabla f(\boldsymbol{x}_k)$, where γ is a fixed constant called step size or learning rate.
- Pros: simple, easy to implement and not computationally expensive (per step)
- Cons: slow convergence, sensitivity to step size.

Gradient Descent

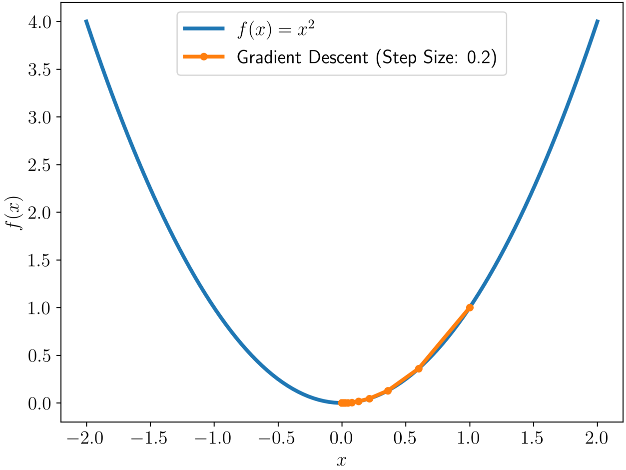


Figure: Gradient Descent on x^2

Gradient Descent

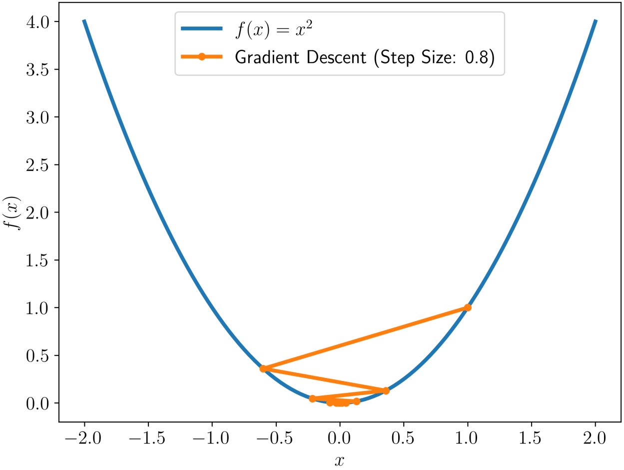


Figure: Gradient Descent on x^2

But wait a minute

Instead of having a fixed Learning Rate what if we have a adaptive learning rate, which is "greedy". If we have,

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}).$$

But wait a minute

Instead of having a fixed Learning Rate what if we have a adaptive learning rate, which is "greedy". If we have,

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}).$$

We define,

$$\phi(\alpha) := f(\boldsymbol{x}_k - \alpha \nabla f(\boldsymbol{x})).$$

But wait a minute

Instead of having a fixed Learning Rate what if we have a adaptive learning rate, which is "greedy". If we have,

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}).$$

We define,

$$\phi(\alpha) := f(\boldsymbol{x}_k - \alpha \nabla f(\boldsymbol{x})).$$

Now we solve the problem,

$$\min_{\alpha \in [0,1]} \phi(\alpha).$$

But wait a minute

Instead of having a fixed Learning Rate what if we have a adaptive learning rate, which is "greedy". If we have,

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}).$$

We define,

$$\phi(\alpha) := f(\boldsymbol{x}_k - \alpha \nabla f(\boldsymbol{x})).$$

Now we solve the problem,

$$\min_{\alpha \in [0,1]} \phi(\alpha).$$

The good this about this problem is that it is one-dimensional. But we still have to realize the function we are evaluating underneath might be expensive to evaluate.

Line Search

Now there are tons of techniques to find the minimum of a 1-D function. We shall use the one which is most familiar to us which is bisection method.

Line Search

Now there are tons of techniques to find the minimum of a 1-D function. We shall use the one which is most familiar to us which is bisection method. But, when should we terminate?

Line Search

Now there are tons of techniques to find the minimum of a 1-D function. We shall use the one which is most familiar to us which is bisection method. But, when should we terminate?
There is 2 conditions usually that we need to follow,
Sufficient Decrease:

$$f(\boldsymbol{x} - \alpha_k \nabla f(\boldsymbol{x}_k) \leq f(x_k) + c_1 \alpha_k \|\nabla f\|^2.$$

and curvature condition,

$$\nabla f\left(x_k + \alpha_k p_k\right)^T p_k \geq c_2 \nabla f_k^T p_k$$

These two together make the Wolfe conditions of sufficient decrease.

A picture is worth thousand words

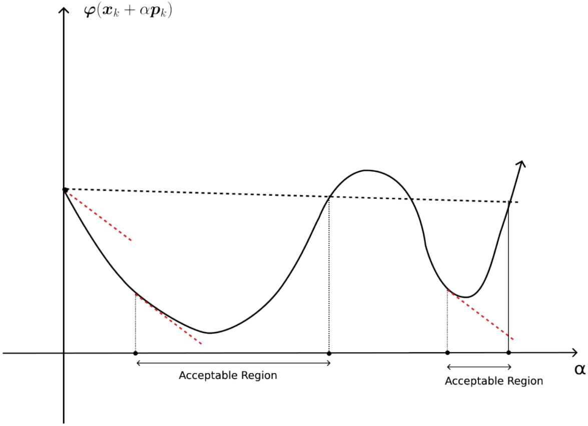


Figure: Strong Wolfe Condition acceptance regions

Steepest Descent & Newton

- As we know steepest descent has it's own problems such as "zig-zag" behaviour as discussed in class. Thus we would like a better method.
- We also learnt about Newton Update for minimizing scalar functions which is given by

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - H_f(\boldsymbol{x}_k)^{-1} \nabla f(\boldsymbol{x}_k).$$

  Where H_f is the hessian of f w.r.t $\boldsymbol{x}$
- This is computationally quite expensive as it requires solving a linear system which takes $\mathcal{O}(n^3)$ time to solve. Where n is problem dimension.

Quasi-Newton

- We would like to develop a Hessian which doesn't take $O(n^3)$ time to solve.
- Before we do that some notation. All the algorithms we saw before now take the form

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k.$$

  Where $\boldsymbol{p}_k$ is a descent direction. We define $y_k = \nabla f(\boldsymbol{x}_{k+1}) - \nabla f(\boldsymbol{x}_k)$ and $s_k = x_{k+1} - x_k$. We would like to mimic the Newton search direction, which is $\boldsymbol{p}_k^{Newton} = -H_f(x_k)^{-1}\nabla f(\boldsymbol{x})$. We would like a approximate hessian $B_k \approx H_f(x_k)$ which doesn't take $\mathcal{O}(n^2)$ to compute.

BFGS Derivation Outline

We start by defining a convex quadratic model as at step k as:

$$m_k(p) = f(\boldsymbol{x}_k) + \nabla f(\boldsymbol{x}_k)^\top + \frac{1}{2}\boldsymbol{p}^\top B_k \boldsymbol{p}.$$

The unique minimizer of this quadratic is

$$p_k = -B_k^{-1}\nabla f(\boldsymbol{x}_k).$$

Now instead of recomputing B_{k+1} for next iteration we proceed as follows,
We would like to have $\nabla m_{k+1}(\mathbf{0}) = \nabla f(\boldsymbol{x}_k)$ and
$\nabla m_{k+1}(-\alpha_k \boldsymbol{p}_k) = \nabla f(\boldsymbol{x}_k)$ to provide a good approximation to the objective function f around those points.
The first one we get for free,

$$\nabla m_{k+1}(\mathbf{0}) = \nabla f(\boldsymbol{x}_k).$$

The second one simplifies to,

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

This is the secant equation for the second derivative.

BFGS Derivation Outline

The key benefit of BFGS is that it computes the inverse Hessian directly.
Which means we directly find H_{k+1} such that,

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

We also would like to make the minimal update on H_k to get H_{k+1} and as mentioned earlier it is positive definite,
Which results in the following constrained optimization problem,

$$\min_{H \in \mathbb{R}^{n \times n}} \|W^{1/2}(H - H_k)W^{1/2}\|_F \qquad \text{subject to } \; H = H^T \; \text{ and } Hy_k = s_k.$$

Which gives the formula: $H_{k+1} = \left(I - \rho_k s_k y_k^T\right) H_k \left(I - \rho_k y_k s_k^T\right) + \rho_k s_k s_k^T$
Which is the BFGS update.
Great thing about it is that it is a small rank-2 update and still keeps the matrix positive definite.

Algorithm 6.1 (BFGS Algorithm)

Algorithm 1 (BFGS Algorithm)

Require: Given starting point x_0, convergence tolerance $\epsilon > 0$, inverse Hessian approximation H_0;

$k \leftarrow 0$;
while $\|\nabla f_k\| > \epsilon \quad \& \quad k < \text{maxIter}$ **do**
  Compute search direction
  $p_k = -H_k \nabla f_k$;
  Set $x_{k+1} = x_k + \alpha_k p_k$ where α_k is computed from a line search procedure to satisfy the Wolfe conditions;
  Define $s_k = x_{k+1} - x_k$ and $y_k = \nabla f_{k+1} - \nabla f_k$;
  Compute H_{k+1} ;
  $k \leftarrow k + 1$;
end while

Rosenbrock Example

- Test function: $f(x, y) = (a - x)^2 + b(y - x^2)^2$
- Typical parameters: $a = 1, b = 100$
- Illustrates curved valley and optimization challenge

Rosenbrock Example Results

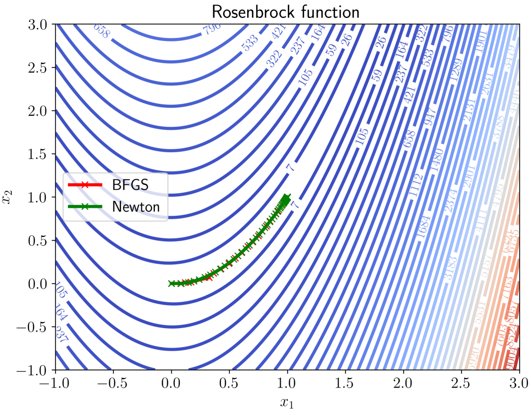
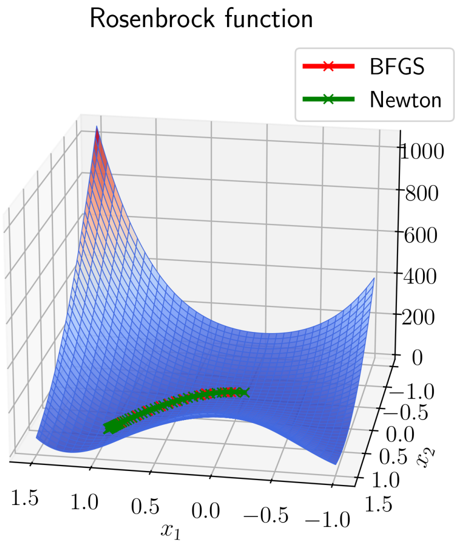


Figure: BFGS optimization path on Rosenbrock function

Rosenbrock Example Results



Results

Function	BFGS	Newton	LBFGS	GD
Adjiman Function (2-D)	nan	4.61e-05	5.50e-16	7.70e-01
Rosenbrock N-D (100-D)	9.28e-11	5.13e-04	9.24e-11	2.92e+00
Paviani Function (10-D)	nan	nan	nan	8.72e-02
Csendes Function (10-D)	9.57e-11	1.21e-03	9.16e-11	7.48e-02
Griewank Function (2-D)	nan	1.26e-15	6.65e-11	8.31e-09
Hosaki Function (2-D)	nan	3.17e-05	nan	8.66e-02
Brent Function (2-D)	9.00e-11	2.51e-05	9.90e-11	3.67e+00
Giunta Function (2-D)	2.22e-15	9.40e-05	2.67e-15	1.29e-08
Styblinski-Tang Function (2-D)	2.93e-14	1.12e-04	6.39e-14	9.96e-11
Trid 6 Function (6-D)	nan	2.47e-05	nan	4.08e+00

Convergence of BFGS

Conclusion

- Introduced BFGS as an quasi Newton optimzer.
- Provided description of Wolfe conditions, and an outline of BFGS derivation.
- Compared BFGS against other methods.