Lecture 3:

Conjugate Gradients and Quasi-2nd order Methods

Week of January 23

University California, Berkeley

Machine Learning Algorithms

MSSE 277B, 3 Units

Spring 2023

Prof. Teresa Head-Gordon

Departments of Chemistry, Bioengineering, Chemical and Biomolecular Engineering

Review Lecture 2

We examined the 0th order Golden search method guaranteed to find a minimum with optimal sectioning (univariate function).

$$f(\vec{x}_0) \Rightarrow f(\vec{x})$$ Golden Section

It picks new points x_{I+1} such that it divides the bracketed domain into uneven lengths so that search space is reduced by the largest fraction possible!

Steepest descents improves over 0th order methods by including gradient information

$$f(\vec{x}) = f(\vec{x}_0) + \vec{\nabla} f(\vec{x}) \Delta\vec{x} + O(\Delta\vec{x})^2$$ 1st order

…and steepest descent itself relies on a step size that is adapted,

$$x_1 = x_0 - \lambda \left. \frac{df}{dx} \right|_{x_0}$$ Steepest descent

or can be solved as a "line search" using methods such as golden section!

In multivariate optimization, the steepest descent now relies on two basic components: the step size size (or solved by GS) and the direction. In standard SD, the direction is the negative of the gradient.

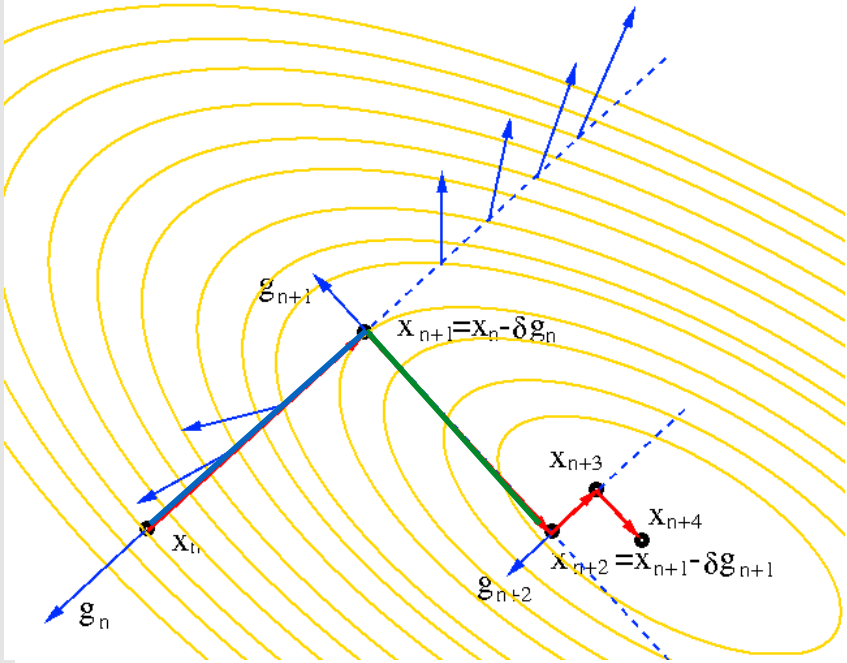
In SD the search direction based on the current gradient will destroy previous line minimizers.

See example here

Step 1: starting from (x_1^0, x_2^0) line search leads to zero gradient at (x_1^1, x_2^1) with residual at this point orthogonal to line

Step 2: starting from (x_1^1, x_2^1) line search in new direction (SD gradient) leads to zero gradient at (x_1^2, x_2^2) with residual at this point orthogonal to line

Step 3: starting from (x_1^2, x_2^2) line search in new direction (SD gradient) when projected onto blue line reintroduces a residual – we have destroyed that previous minimizer!



Review Lecture 2

Conjugate Gradients

Given poor rate of convergence of steepest descents for anything but homogeneous quadratic, can we do something better? In particular, are better search directions possible?

Conjugate gradients seeks to eliminate this problem by determining a new direction $d_{i+1}(\vec{x})$ that is a compromise between the new gradient direction $-\vec{\nabla} f(\vec{x}_{i+1})$ (standard steepest descent) and the previous step directions $d_i(\vec{x})$.

$$d_{i+1}(\vec{x}) = -\vec{\nabla} f(\vec{x}_{i+1}) + \beta d_i(\vec{x})$$

In particular, we want to find a value for β so that each new search direction does not spoil the minimization achieved by the previous ones. This requires us to form a basis set of search directions $\{d_i(\vec{x})\}$ all of which are "H orthogonal"

$$d_{i+1}(\vec{x})^T \underline{H} d_i(\vec{x}) = 0$$

And reducing it to n-1D optimizations!

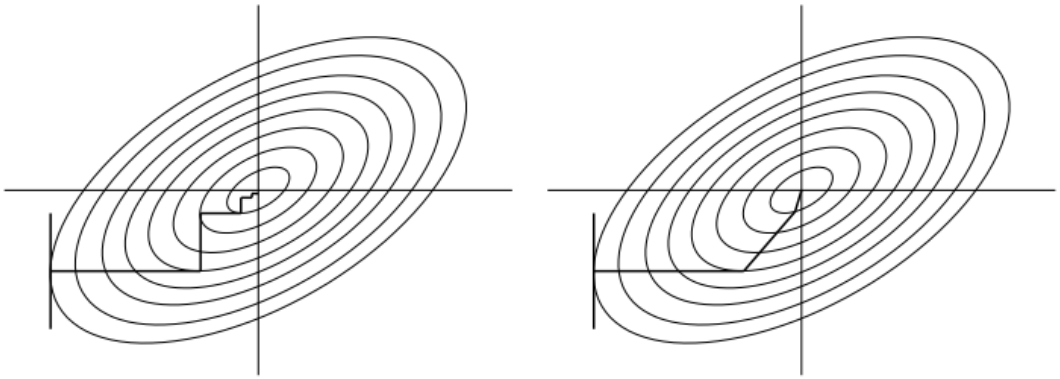


Figure 6.14: Steepest descent vs. conjugate gradient.

Multivariate Functions: Quadratic

Define multivariate quadratic function:

$$f(\vec{x}) = \frac{1}{2}\vec{x}^T \underline{\underline{H}}\vec{x} - \vec{b}\vec{x} + c$$

and at a minimum

$$-\vec{\nabla} f(\vec{x}_{min}) = -\underline{\underline{H}}\vec{x}_{min} + \vec{b} = 0$$

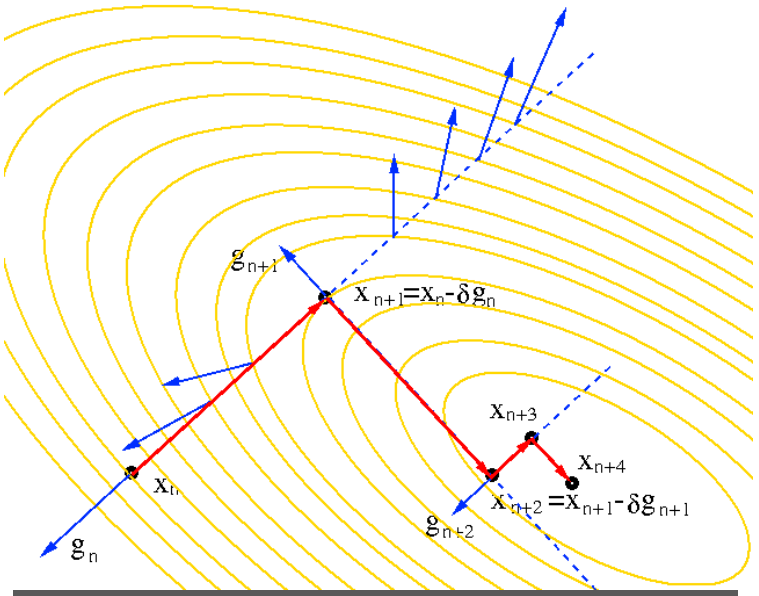
We are solving a set of linear equations, where H (the Hessian) and b are known, but x_{min} is unknown.

The "residual" r, measures how far from b we are

$$\vec{r}_i = -\nabla f(\vec{x}_i) = \vec{b} - \underline{\underline{H}}\vec{x}_i$$

The error measures how far we are from x_{min}, and is related to residual as

$$\vec{e}_i = \vec{x}_i - \vec{x}_{min} \longrightarrow \vec{r}_i = -\underline{\underline{H}}\vec{e}_i$$



Conjugate Gradients: Use of Golden Section

What we need is a way to determine these clever directions, and then 1D optimize for each independent direction.

We know that $\{d_i(\vec{x})\}$ spans the search space so solution $\vec{x}_{min}$ is formed from a linear combination

$$\vec{x}_{min} = \sum_k \lambda_k d_k(\vec{x})$$

And operating with H we get

$$\vec{b} = \underline{\underline{H}}\vec{x}_{min} = \sum_k \lambda_k \underline{\underline{H}} d_k(\vec{x})$$

Lets project onto one of the directions, $d_i(\vec{x})$ in order to find the λ_i's

$$d_i(\vec{x})^T \vec{b} = \sum_k \lambda_k d_i(\vec{x})^T \underline{\underline{H}} d_k(\vec{x})$$

"H orthogonal" $\quad d_i(\vec{x})^T \underline{\underline{H}} d_k(\vec{x}) = 0$

and thus $\quad d_i(\vec{x})^T \vec{b} = \lambda_i d_i(\vec{x})^T \underline{\underline{H}} d_i(\vec{x})$

Allowing us to solve for every λ_i

$$\lambda_i = \frac{d_i(\vec{x})^T \vec{b}}{d_i(\vec{x})^T \underline{\underline{H}} d_i(\vec{x})}$$

For simple (small n) quadratics H,b are easy to formulate. But for arbitrary large n function, we need not evaluate Hessian, but just solve for λ_i using SD or a line search (Golden Section) for new position x_{i+1}

$$\vec{x}_{i+1} = \vec{x}_i + \lambda_i d_i(\vec{x})$$

Conjugate Gradients: Search Directions

Our new search direction $d_{i+1}(\vec{x})$ must be orthogonal to any proceeding directions in order to not destroy the previous direction minimizer.

Therefore we will project previous search directions out of our current gradient direction $-\vec{\nabla} f(\vec{x}_{i+1})$!

$$d_{i+1}(\vec{x}) = -\vec{\nabla} f(\vec{x}_{i+1}) + \sum_{k=0}^{i} \beta_k d_k(\vec{x})$$

I.e. this operation with the right β_k's will define a new direction that remains H orthogonal.

Now lets solve for β_k by exploiting H orthogonal property! Multiply both sides by $\underline{\underline{H}}\, d_i(\vec{x})$

$$d_{i+1}(\vec{x})^T \underline{\underline{H}}\, d_i(\vec{x}) = -\vec{\nabla} f(\vec{x}_{i+1})^T \underline{\underline{H}}\, d_i(\vec{x}) + \sum_{k=0}^{i} \beta_k d_k(\vec{x})^T \underline{\underline{H}}\, d_i(\vec{x})$$

H orthogonal

$$0 = -\vec{\nabla} f(\vec{x}_{i+1})^T \underline{\underline{H}}\, d_i(\vec{x}) + \beta_i d_i(\vec{x})^T \underline{\underline{H}}\, d_i(\vec{x})$$

$$\beta_i = \frac{\vec{\nabla} f(\vec{x}_{i+1})^T \underline{\underline{H}}\, d_i(\vec{x})}{d_i(\vec{x})^T \underline{\underline{H}}\, d_i(\vec{x})}$$

Conjugate Gradients: Algorithm

While we could solve all search directions in one go using Gram-Schmidt orthogonalization, it is not that efficient. Instead we solve for them dynamically!

To get things started we use $d_0(\vec{x}) = -\nabla f(\vec{x}_0)$, i.e. steepest descent direction in first step. Line minimize along $d_0(\vec{x})$ to determine $\vec{x}_1$

$$\vec{x}_1 = \vec{x}_0 + \lambda_0 d_0(\vec{x})$$

Now we can evaluate $-\nabla f(\vec{x}_1)$, and create the new search direction by projecting out previous direction!

$$d_1(\vec{x}) = -\nabla f(\vec{x}_1) + \beta_0 d_0(\vec{x})$$

And line minimization along $d_1(\vec{x})$ lets us determine $\vec{x}_2$. By generalization ….

$$d_{i+1}(\vec{x}) = -\nabla f(\vec{x}_{i+1}) + \beta_i d_i(\vec{x})$$

Hence we are dynamically creating H-orthogonal $\{d_i(\vec{x})\}$

$$\beta_i = \frac{\vec{\nabla} f(\vec{x}_{i+1})^T \underline{\underline{H}}\, d_i(\vec{x})}{d_i(\vec{x})^T \underline{\underline{H}}\, d_i(\vec{x})}$$

Again, for a small n-quadratic H is easy to formulate! But for general efficiency we formulate β_i so we don't need H! Let's try and get rid of $\underline{\underline{H}} d_i(\vec{x})$

Remember that $\vec{x}_{i+1} = \vec{x}_i + \lambda_i d_i(\vec{x})$

Operate with H $\underline{\underline{H}}\vec{x}_{i+1} = \underline{\underline{H}}\vec{x}_i + \lambda_i \underline{\underline{H}} d_i(\vec{x})$

Remember residual $\vec{r}_i = -\nabla f(\vec{x}_i) = \vec{b} - \underline{\underline{H}}\vec{x}_i$

and use in above $\vec{b} - \vec{r}_i = \underline{\underline{H}}\vec{x}_i$

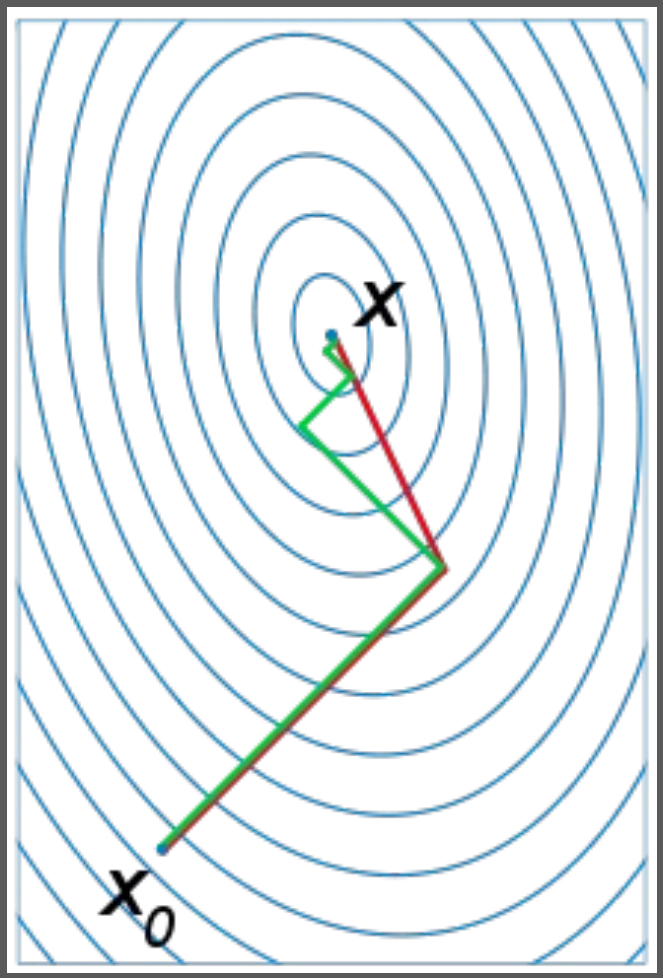
$$\vec{b} - \vec{r}_{i+1} = \vec{b} - \vec{r}_i + \lambda_i \underline{\underline{H}} d_i(\vec{x})$$

$$\underline{\underline{H}} d_i(\vec{x}) = \frac{\vec{r}_i - \vec{r}_{i+1}}{\lambda_i}$$

Use this in β_i

$$\beta_i = \frac{\vec{\nabla} f(\vec{x}_{i+1})^T (\vec{r}_i - \vec{r}_{i+1})/\lambda_i}{d_i(\vec{x})^T (\vec{r}_i - \vec{r}_{i+1})/\lambda_i}$$

Conjugate Gradients: Algorithm Efficiency



Because direction vectors are built from the residuals,

$$d_{i+1}(\vec{x}) = \vec{r}_{i+1} + \beta_i d_i(\vec{x})$$

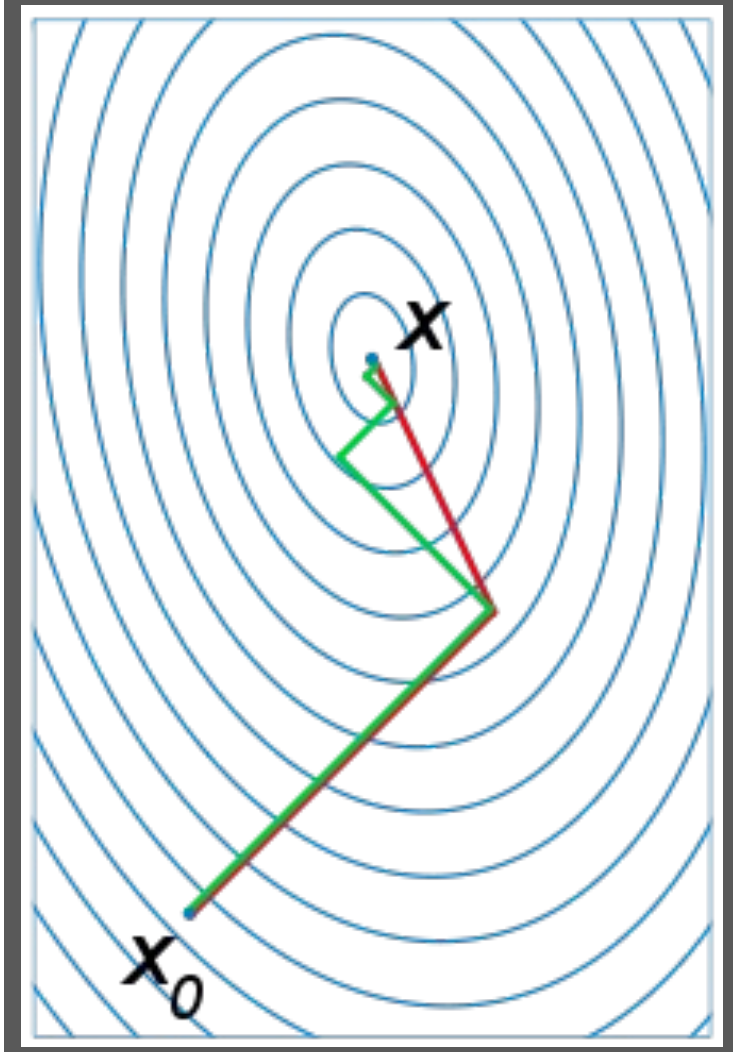
each new residual is orthogonal to any of the previous search directions and any preceding residual

$$d_j(\vec{x})^T \vec{r}_{i+1} = 0 \qquad \vec{r}_{i+1}^T \vec{r}_j = 0 \qquad \text{for j=0…,i}$$

This is a consequence of H-orthogonality of direction vectors, whose proof can be found in front page reading. Hence exploit these new H-orthogonality relationships! Write β_i in terms of residuals and direction vectors and cancel out λ

$$\beta_i = \frac{\vec{\nabla} f(\vec{x}_{i+1})^T (\vec{r}_i - \vec{r}_{i+1})/\lambda_i}{d_i(\vec{x})^T (\vec{r}_i - \vec{r}_{i+1})/\lambda_i} \qquad \beta_i = \frac{-\vec{r}_{i+1}^T (\vec{r}_i - \vec{r}_{i+1})}{d_i(\vec{x})^T (\vec{r}_i - \vec{r}_{i+1})}$$

Conjugate Gradients: Algorithm Efficiency



1st H orthogonality:

$$\beta_i = \frac{-\vec{r}_{i+1}^{T}(-\vec{r}_{i+1})}{d_i(\vec{x})^T(\vec{r}_i)}$$

Residual and search direction definitions:

$$\beta_i = \frac{\nabla f(\vec{x}_{i+1})\nabla f(\vec{x}_{i+1})}{\left(\vec{r}_i + \beta_{i-1} d_{i-1}(\vec{x})\right)^T \vec{r}_i}$$

2nd H orthogonality:

$$\beta_i = \frac{\nabla f(\vec{x}_{i+1})\nabla f(\vec{x}_{i+1})}{\vec{r}_i^{\,T}\vec{r}_i}$$

Residual definition:

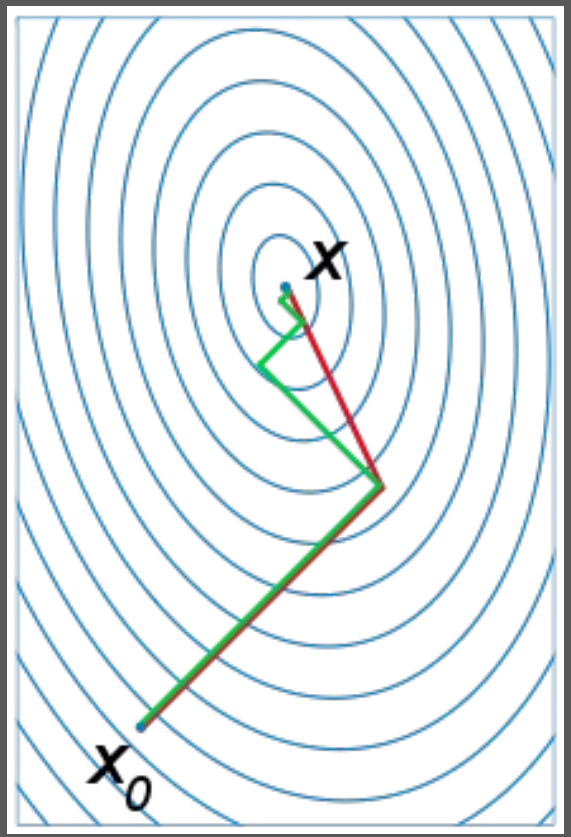
$$\beta_i = \frac{\nabla f(\vec{x}_{i+1})\nabla f(\vec{x}_{i+1})}{\nabla f(\vec{x}_i)\nabla f(\vec{x}_i)}$$

New directions are dynamically created with no H!

$$d_{i+1}(\vec{x}) = -\nabla f(\vec{x}_{i+1}) + \beta_i d_i(\vec{x})$$

$$d_{i+1}(\vec{x}) = -\nabla f(\vec{x}_{i+1}) + \frac{\nabla f(\vec{x}_{i+1})\nabla f(\vec{x}_{i+1})}{\nabla f(\vec{x}_i)\nabla f(\vec{x}_i)} d_i(\vec{x})$$

Conjugate Gradients: Algorithm Efficiency



2nd Order Methods: Newton Methods

$$f(\vec{x}_o + \Delta\vec{x}) = f(\vec{x}_0) + \vec{\nabla} f(\vec{x}_0)\,\Delta\vec{x} + \frac{1}{2}\Delta\vec{x}^T \underline{\underline{H}}(\vec{x}_0)\Delta\vec{x} + \cdots.$$

Let's differentiate above but truncate at quadratic

$$\vec{\nabla} f(\vec{x}_o + \Delta\vec{x}) = \vec{\nabla} f(\vec{x}_0) + \vec{\nabla}^2 f(\vec{x}_0)\,\Delta\vec{x}$$

$$\vec{\nabla} f(\vec{x}_o + \Delta\vec{x}) = \vec{\nabla} f(\vec{x}_0) + \underline{\underline{H}}(\vec{x}_0)\Delta\vec{x}$$

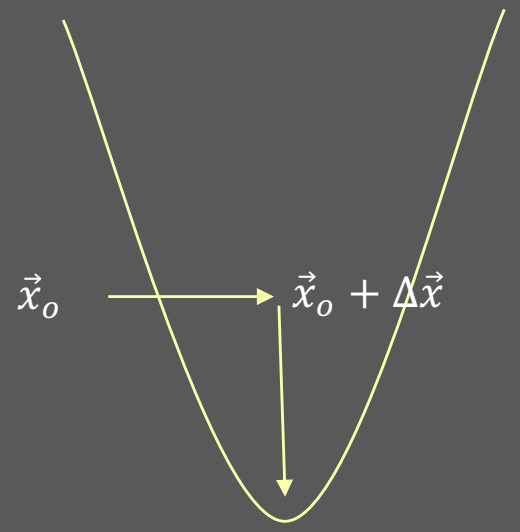
Ultimately we can determine $\Delta\vec{x}$ that minimizes quadratic function by condition $\vec{\nabla} f(\vec{x}_o + \Delta\vec{x})=0$

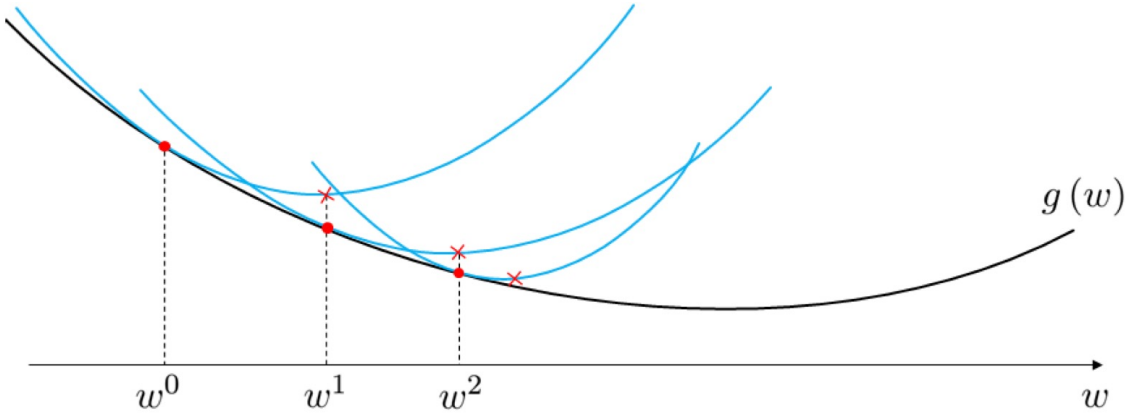
$$\underline{\underline{H}}(\vec{x}_0)\Delta\vec{x} = -\vec{\nabla} f(\vec{x}_0)$$

$$\Delta\vec{x} = \vec{\nabla} f(\vec{x}_0)\underline{\underline{H}}^{-1}(\vec{x}_0)$$

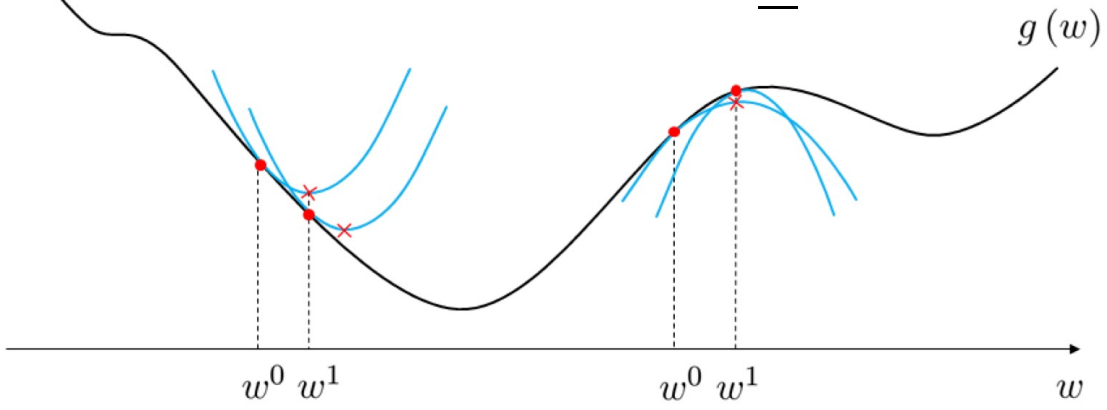
and Newton method converges in a single step for any quadratic (homogeneous or inhomogeneous)

Any function can be approximated over some local interval as a simple quadratic based on truncation of its Taylor expansion





$$\vec{x}_{k+1} = \vec{x}_k - \vec{\nabla} f(\vec{x}_k) \underline{\underline{H}}^{-1}(\vec{x}_k)$$



2nd Order Methods: Newton Methods

For non-quadratic functions Newton's method requires continued update of Hessian until convergence.

- Note that 2nd order method converges to a stationary point, so determine if a minimum, maximum, or saddle point by finding the eigenvalues.

- The primary problem with 2nd order method like Newton is that it requires calculation of matrix elements and taking inverse of N^2 Hessian matrix. This may be problematic if 2nd derivatives are expensive, you do not have memory to store, or because matrix inversion is typically an N^3 calculation.

- Quasi-second order methods correspond to ways of approximating the Hessian to avoid this burden.

David-Fletcher-Powell proposed

$$\underline{\underline{H}}(\vec{x}_{k+1}) \sim \underline{\underline{H}}(\vec{x}_k) + \frac{\Delta\vec{x}_k \Delta\vec{x}_k{}^T}{\Delta\vec{x}_k{}^T \Delta\vec{g}_k} - \frac{\underline{\underline{H}}(\vec{x}_k)\Delta\vec{g}_k \left[\underline{\underline{H}}(\vec{x}_k)\Delta\vec{g}_k\right]^T}{\Delta\vec{g}_k{}^T \underline{\underline{H}}(\vec{x}_k)\Delta\vec{g}_k}$$

Where

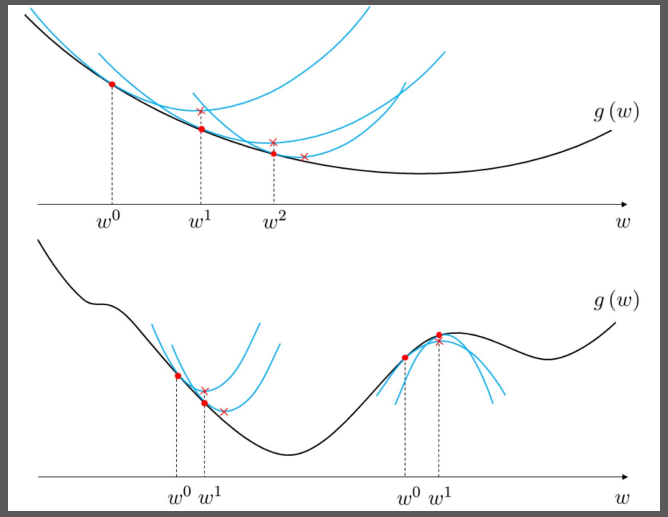
$$\Delta\vec{x}_k = \lambda_{min} d_i \qquad \text{and} \qquad \Delta\vec{g}_k = \left[\vec{\nabla} f(\vec{x}_{k+1}) - \vec{\nabla} f(\vec{x}_k)\right]$$

(Don't panic! Just get feel of the DFP algorithm!)

Position update formula satisfies Newton condition

$$\vec{x}_{k+1} = \vec{x}_k - \vec{\nabla} f(\vec{x}_k) \underline{\underline{H}}^{-1}(\vec{x}_k)$$

However, DFP requires inversion of Hessian which is $O(N^3)$ operation



Quasi-2nd Order Method: David-Fletcher-Powell

Broyden-Fletcher-Goldfarb-Shanno (BFGS)

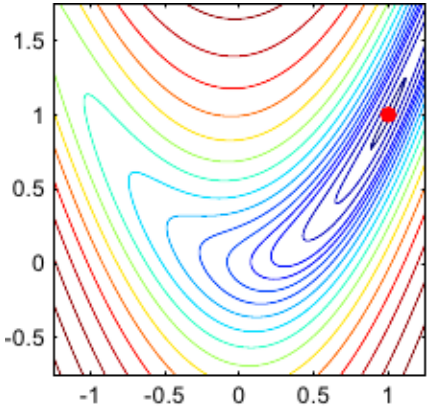
Considered to be the best quasi-2nd order method known, as it approximates inverse Hessian directly to avoid $O(N^3)$ step!

$$\underline{\underline{H}}^{-1}(\vec{x}_{k+1}) = \underline{\underline{H}}^{-1}(\vec{x}_k) + \left[1 + \frac{\Delta\vec{g}_k{}^T \underline{\underline{H}}^{-1}(\vec{x}_k)\Delta\vec{g}_k}{\Delta\vec{x}_k{}^T\Delta\vec{g}_k}\right] \times$$

$$\left[\frac{\Delta\vec{x}_k\Delta\vec{x}_k{}^T}{\Delta\vec{x}_k{}^T\Delta\vec{g}_k} - \frac{\underline{\underline{H}}^{-1}(\vec{x}_k)\Delta\vec{g}_k\Delta\vec{x}_k{}^T + \left[\underline{\underline{H}}^{-1}(\vec{x}_k)\Delta\vec{g}_k\Delta\vec{x}_k{}^T\right]^T}{\Delta\vec{g}_k{}^T\Delta\vec{x}_k}\right]$$

And now update Newton step directly!!

$$\vec{x}_{k+1} = \vec{x}_k - \vec{\nabla} f(\vec{x}_k)\underline{\underline{H}}^{-1}(\vec{x}_k)$$



We will explore this in canned SciPy code to see how many steps to the minimum of the Rosenbrock function!