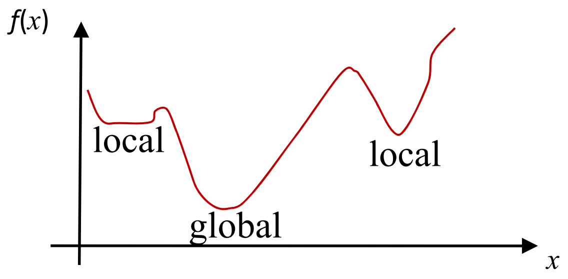
UNCONSTRAINED OPTIMIZATION

(USING DERIVATIVES)

Consider the problems Minimize $f(\boldsymbol{x})$, st $\boldsymbol{x}\in\mathbb{R}^n$.

DEFINITION 1: A point $\boldsymbol{x}^*\in\mathbb{R}^n$ is said to be a *LOCAL MINIMUM POINT* of f over $\mathbb{R}^n$ if there exists $\varepsilon>0$ such that $f(\boldsymbol{x}) \geq f(\boldsymbol{x}^*)$ for all $\boldsymbol{x}$ such that $\|\boldsymbol{x}-\boldsymbol{x}^*\| < \varepsilon$ (i.e., within a distance ε of $\boldsymbol{x}$). If $f(\boldsymbol{x}) > f(\boldsymbol{x}^*)$ then it is *STRICT* local minimum point.

DEFINITION 2: A point $\boldsymbol{x}^*\in\mathbb{R}^n$ is said to be a *GLOBAL MINIMUM POINT* of f over $\mathbb{R}^n$ if $f(\boldsymbol{x}) \geq f(\boldsymbol{x}^*)$ for all $\boldsymbol{x}\in\mathbb{R}^n$. If $f(\boldsymbol{x})>f(\boldsymbol{x}^*)$ then it is a *STRICT* global minimum point.



Assuming that f is differentiable, the gradient vector $\nabla \boldsymbol{f}(\boldsymbol{x})$ is the vector

$$\nabla \boldsymbol{f}(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(\boldsymbol{x}) \\ \frac{\partial f}{\partial x_2}(\boldsymbol{x}) \\ \vdots \\ \frac{\partial f}{\partial x_n}(\boldsymbol{x}) \end{bmatrix}$$

<u>THEOREM 1</u>: **Necessary Conditions**

If f is smooth real-valued function then a <u>necessary</u> condition for the point $\boldsymbol{x}^*$ to be a local minimizer (or maximizer) is that $\nabla \boldsymbol{f}(\boldsymbol{x}^*) = \mathbf{0}$, i.e.,

$$\begin{cases} \dfrac{\partial f}{\partial x_1}(\boldsymbol{x}^*) = 0 \\ \dfrac{\partial f}{\partial x_2}(\boldsymbol{x}^*) = 0 \\ \qquad \vdots \\ \dfrac{\partial f}{\partial x_n}(\boldsymbol{x}^*) = 0 \end{cases}$$

(in general, a nonlinear system of n equations in n unknowns…)

Furthermore, the Hessian $\nabla^2 \boldsymbol{f}(\boldsymbol{x}^*)$ is positive (negative) semidefinite at a minimum (maximum).

<u>THEOREM 2</u>: **Sufficient Conditions**

Assuming that f is twice continuously differentiable,

If 1) $\nabla \boldsymbol{f}(\boldsymbol{x}^*) = \mathbf{0}$ and,

2) $\boldsymbol{y}^T \nabla^2 \boldsymbol{f}(\boldsymbol{x}^*) \boldsymbol{y} > 0$ for all $\boldsymbol{y} \neq \mathbf{0}$

then $\boldsymbol{x}^*$ is a local minimizer.

NOTE:

1. Condition 2 states that the Hessian at $\boldsymbol{x}^*$ should be positive definite
2. If the gradient vanishes and the Hessian is <u>negative</u> definite at $\boldsymbol{x}^*$, then $\boldsymbol{x}^*$ is a local maximum

Note that there may be points other than $\boldsymbol{x}^*$ that satisfy the <u>necessary</u> conditions; however, these will not satisfy the <u>sufficient</u> conditions unless they are local optima. (e.g. $f(x)=x^3$ at $x=0$).

EXAMPLE: Consider the function $f(x,y,z) = 2x^2 + xy + y^2 + yz + z^2 - 6x - 7y - 8z + 9$

Points satisfying the **necessary** *conditions* ($[\nabla \boldsymbol{f}(\boldsymbol{x})]=[\boldsymbol{0}]$)

$$\partial f/\partial x = 4x + y - 6 = 0$$

$$\partial f/\partial y = x + 2y + z - 7 = 0$$

$$\partial f/\partial z = y + 2z - 8 = 0$$

which yields $[x=6/5;\ y=6/5;\ z=17/5]$

Do these satisfy the ***sufficient*** *conditions*?

$$\boldsymbol{H} = \nabla^2 \boldsymbol{f}(\boldsymbol{x}^*) = \begin{bmatrix} 4 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 2 \end{bmatrix}$$

$$|4| = 4 > 0; \quad \begin{vmatrix} 4 & 1 \\ 1 & 2 \end{vmatrix} = 7 > 0; \quad \begin{vmatrix} 4 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 2 \end{vmatrix} = 10 > 0$$

Since all principal minors are strictly positive, it follows that $\boldsymbol{H}$ is positive definite over $\mathbb{R}^3$ and hence the sufficient conditions are satisfied and $\boldsymbol{x}^*$ is a local minimum.

Is the point also a **global minimum**?

By the Taylor series expansion around $\boldsymbol{x}^*$ we have

$$f(\boldsymbol{x}^* + \boldsymbol{d}) = f(\boldsymbol{x}^*) + \nabla \boldsymbol{f}(\boldsymbol{x}^*)^T \boldsymbol{d} + \tfrac{1}{2}\boldsymbol{d}^T \nabla^2 \boldsymbol{f}(\boldsymbol{x})\boldsymbol{d},$$

But $\nabla^2 \boldsymbol{f}(\boldsymbol{x})$ is positive definite and hence, for all $\boldsymbol{d} \neq \boldsymbol{0}$ we have $\tfrac{1}{2}\boldsymbol{d}^T \nabla^2 \boldsymbol{f}(\boldsymbol{x})\boldsymbol{d} > 0$.

Further, $\nabla \boldsymbol{f}(\boldsymbol{x}^*) = 0 \Rightarrow f(\boldsymbol{x}^* + \boldsymbol{d}) - f(\boldsymbol{x}^*) > 0$

$$\Rightarrow f(\boldsymbol{x}^* + \boldsymbol{d}) > f(\boldsymbol{x}^*) \text{ for all } \boldsymbol{d} \neq \boldsymbol{0}.$$

Therefore $\boldsymbol{x}^*$ is also a **global** minimum.

Recall that a **convex** function has a positive semidefinite Hessian. Thus (along the same lines as the example above) the necessary conditions for a local minimum are also sufficient. Furthermore, the local minimum is also a **global** minimum. If the function is **strictly** convex then this global minimum is also unique. (WHY?). Similar remarks hold for **concave** functions and their **maxima**.

Most techniques are designed to find local minima (maxima); these of course, being adequate for convex (concave) functions.

CONVERGENCE OF ALGORITHMS

Before getting into algorithms, we briefly examine the general conditions under which an algorithm will converge to an optimum solution.

DEFINITION 1: An algorithm $\mathbb{A}$ is a point-to-point mapping defined on a space $\mathbb{X}$ that assigns to every point $\boldsymbol{x}\in\mathbb{X}$, a point $\boldsymbol{y}\in\mathbb{X}$.

(An algorithm may also be viewed more generally as a point-to-**set** mapping; however we'll keep it simple here…). Thus an algorithm $\mathbb{A}$ generates a sequence of points; i.e., given a starting point $\boldsymbol{x}^{0}$, an iterative sequence is generated from

$$\boldsymbol{x}^{k+1} = \mathbb{A}(\boldsymbol{x}^{k})$$

An Example

Minimize x-5, st $x \geq 0$

The optimum solution is at $x^*=0$

One algorithm might be defined by the following point-to-point map:

$$x^{k+1}=\mathbb{A}(x^k)=x^k/2$$

So for $x^0 = 5$ the sequence generated would be {5, 2.5, 1.25, 0.625, ...}

Clearly, this converges to the optimum

Another algorithm might use the point-to-set map:

$$x^{k+1}= \mathbb{A}(x^k)=[x^k/4, x^k/2]$$

i.e., each new point is somewhere within the closed interval above. The exact sequence cannot be predicted now, but starting at the same $x^0 = 5$ we might generate the following sequence: {5, 2, 0.8, 0.36, 0.10,…}. Again, this is a sequence that converges to x^* (and actually, at least as fast the previous method).

Finally, consider the following algorithm:

$$x^{k+1}= \mathbb{A}(x^k) = \begin{cases} (x^k/2) & if\ \ x^k < 1 \\ 1+(x^k-1)/2 & if\ \ x^k \geq 1 \end{cases}$$

For $x^0<1$ this converges to x^* but for $x^0 \geq 1$ it converges to the sub-optimal point x=1

Associated with $\mathbb{A}$ are (i) **a solution set**, and (ii) a **descent** (or **ascent**) **function**.

<u>DEFINITION 2</u>: A solution set Ω is defined as

$$\Omega=\{\boldsymbol{x} \mid \boldsymbol{x} \text{ is a solution to our problem}\}.$$

For instance consider the problem: Minimize $f(\boldsymbol{x})$, st $\boldsymbol{x}\in\mathbb{R}^n$

Among other possibilities, we could define Ω as

1) $\Omega = \{\boldsymbol{x}\in\mathbb{R}^n \mid \boldsymbol{x}$ is a local minimum point of $f\}$,
2) $\Omega = \{\boldsymbol{x}\in\mathbb{R}^n \mid \nabla \boldsymbol{f}(\boldsymbol{x}) = \mathbf{0}\}$
3) $\Omega = \{\boldsymbol{x}\in\mathbb{R}^n \mid f(\boldsymbol{x}) \le b\}$, where b is some acceptable value
4) $\Omega = \{\boldsymbol{x}\in\mathbb{R}^n \mid \nabla \boldsymbol{f}(\boldsymbol{x}) = \mathbf{0}$ and $\nabla^2 \boldsymbol{f}(\boldsymbol{x})$ is positive definite$\}$. etc. etc.

<u>DEFINITION 3</u>: Let $\Omega \subset \mathbb{X}$ be a given solution set and let $\mathbb{A}$ be an algorithm on $\mathbb{X}$. A continuous real valued function z on $\mathbb{X}$ is said to be a DESCENT FUNCTION for Ω and $\mathbb{A}$ if it satisfies

(i) if $\boldsymbol{x}\notin\Omega$ and $\boldsymbol{y}= \mathbb{A}(\boldsymbol{x})$, then $z(\boldsymbol{y})< z(\boldsymbol{x})$

(ii) if $\boldsymbol{x}\in\Omega$ and $\boldsymbol{y}= \mathbb{A}(\boldsymbol{x})$, then $z(\boldsymbol{y})\le z(\boldsymbol{x})$

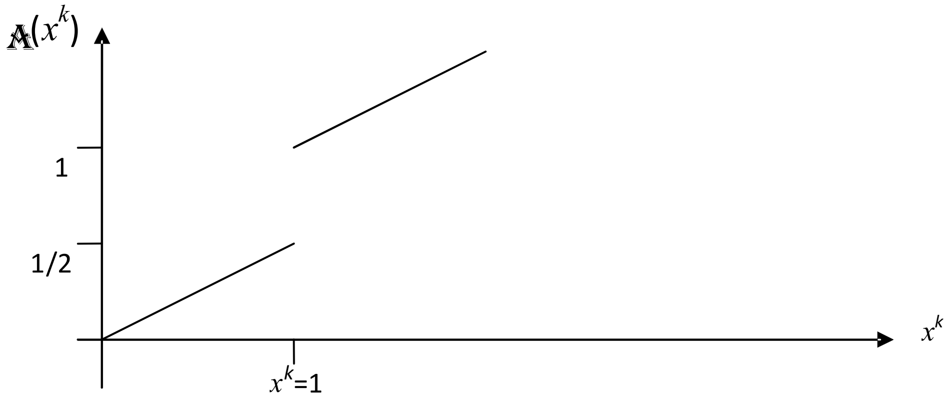
In other words, a descent function decreases in value as each new point is generated by $\mathbb{A}$, and once an acceptable solution is found new points can never be any worse. For example, if Ω is defined as in 2), we could use $\|\nabla \boldsymbol{f}(\boldsymbol{x})\|$ as a descent function. For Ω as in 3) we could simply use $f(\boldsymbol{x})$ itself.

Closed Maps

Another important concept is continuity (or closedness) of algorithmic maps.

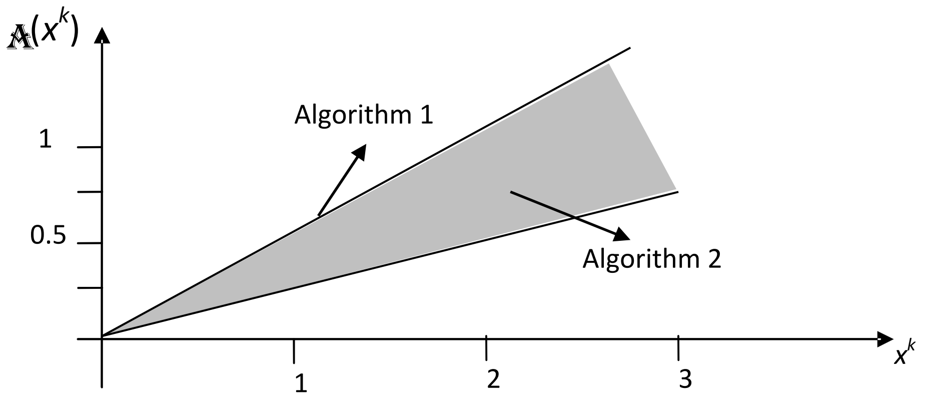
<u>DEFINITION 4</u>: An algorithm $\mathbb{A}$ on $\mathbb{X}$ is closed (or continuous) at $\boldsymbol{x} \in \mathbb{X}$, if $\{\boldsymbol{x}^{\boldsymbol{k}}\} \to \boldsymbol{x}$ implies that $\{\mathbb{A}(\boldsymbol{x}^{\boldsymbol{k}})\} \to \mathbb{A}(\boldsymbol{x})$. The map (or algorithm) is closed on $\mathbb{Z} \subset \mathbb{X}$ if it is closed at each point in $\mathbb{Z}$.

Consider the map for the third algorithm we saw earlier: As shown below, the map is discontinuous at $x^k=1$. This is why it doesn't converge to the optimum



For example, the sequence with $x^k=1-(1/k)$ converges to $x=1$ but the sequence $\{\mathbb{A}(x^k)\}$ with $\mathbb{A}(x^k)=x^k/2$ converges to the value 1/2, which is not $\mathbb{A}(x)= \mathbb{A}(1)=1$.

On the other hand, the maps for the first two algorithmic maps are both closed



Note that closedness of algorithmic maps is analogous to continuity in functions.

We may now state the main theorem of this section:

CONVERGENCE THEOREM: Let $\mathbb{A}$ be an algorithm on $\mathbb{X}$ and suppose that given $\boldsymbol{x^0}$, the sequence $\{x^k\}_{k=0\ldots\infty}$ is generated via $\boldsymbol{x^{k+1}} = \mathbb{A}(\boldsymbol{x^k})$. Let the optimal solution set $\Omega\subset\mathbb{X}$ be given, and suppose that,

a) all points $\boldsymbol{x^k}$ are contained in a compact set $\mathbb{S}\subset\mathbb{X}$

b) there exists a continuous descent function z for $\mathbb{A}$ and Ω

c) the map of algorithm $\mathbb{A}$ is closed at points outside Ω.

Then the sequence $\{x^k\}_{k=0\ldots\infty}$ converges to a point in Ω, or the limit of any convergent subsequence of $\{\boldsymbol{x^k}\}$ is a solution, i.e., the limit lies within Ω. Also, $z(\boldsymbol{x^k}) \rightarrow z(\boldsymbol{x})$ for some $\boldsymbol{x}\in\Omega$.

Note that the convergent subsequence may be the sequence itself. If not, at least some subset of the points generated converges to a solution.

<u>NOTES ON THE CONVERGENCE THEOREM</u>:

1. It states "sufficient" conditions for convergence. It might be possible to prove convergence under different (perhaps weaker) assumptions. However, the notions of a continuous algorithmic map and of a descent function form the backbone of all proofs.
2. By convergence it means that the algorithm will converge to a solution irrespective of the starting point $\boldsymbol{x}^0$. (This is not to be confused with trying to find a global minimum). Thus proof of convergence is an indication of robustness.
3. In most engineering applications, the design variables are usually bounded as $x_{lower} \leq x \leq x_{upper}$. This ensures that $\{\boldsymbol{x}^{\boldsymbol{k}}\}$ is generated in a compact set as per requirement (a).
4. If Ω is the singleton $\boldsymbol{x}^*$ then the whole sequence $\{\boldsymbol{x}^{\boldsymbol{k}}\}$ converges to $\boldsymbol{x}^*$.
5. The theorem is very useful for modifying existing algorithms so as to make them convergent.

RATES OF CONVERGENCE

This refers to the "speed" with which convergence takes place.

Suppose $\{\boldsymbol{x}^{\boldsymbol{k}}\} \rightarrow \boldsymbol{x}^*$. We say that $\{\boldsymbol{x}^{\boldsymbol{k}}\}$ converges to $\boldsymbol{x}^*$ with **order** (or **rate**) r and a **rate constant** C, if r is the supremum of the nonnegative values satisfying

$$0 \leq \lim_{k \to \infty} \left(\frac{\left\| \boldsymbol{x}^{k+1} - \boldsymbol{x}^* \right\|}{\left\| \boldsymbol{x}^{k} - \boldsymbol{x}^* \right\|^r} \right) = C < \infty$$

e.g. if $(\boldsymbol{x}^{\boldsymbol{k}}\text{-}\boldsymbol{x}^*)$ is the "error" $\boldsymbol{e}^{\boldsymbol{k}}$ at iteration k (with $\lim_{k\to\infty} \boldsymbol{e}^{\boldsymbol{k}}=\mathbf{0}$), then this says

$$\lim_{k \to \infty} \left(\frac{\left\| \boldsymbol{e}^{k+1} \right\|}{\left\| \boldsymbol{e}^{k} \right\|^r} \right) = C.$$

- If r=1, C must be <1 for convergence; if 0<C<1 the sequence is then said to have a ***linear*** convergence rate.
- If r=1, but C=0 then the sequence is said to have ***superlinear*** convergence.

<u>THEOREM</u>: If r>1 then convergence is superlinear.

<u>Proof</u>: Suppose we have $\lim_{k \to \infty} \left(\frac{\left\| \boldsymbol{e}^{k+1} \right\|}{\left\| \boldsymbol{e}^{k} \right\|^r} \right) = C$ where $C<\infty$ and r>1.

Then $$\lim_{k \to \infty} \left(\frac{\left\| \boldsymbol{e}^{k+1} \right\|}{\left\| \boldsymbol{e}^{k} \right\|} \right) = \lim_{k \to \infty} \left(\frac{\left\| \boldsymbol{e}^{k+1} \right\|}{\left\| \boldsymbol{e}^{k} \right\|^r} \left\| \boldsymbol{e}^{k} \right\|^{r-1} \right) = C \cdot 0 = 0$$

In particular, if r =2 and C>0 the sequence is said to have quadratic convergence; it will eventually always converge faster than a linearly convergent one…

EXAMPLE 1: Suppose $\left(\frac{\|\boldsymbol{e}^{k+1}\|}{\|\boldsymbol{e}^{k}\|^{r}}\right)=C$ for all k, i.e., $\|\boldsymbol{e}^{k+1}\| = C\|\boldsymbol{e}^{k}\|^{r}$. Note that with this simplification, we don't need to worry about limits…

Case 1: r=1, 0<C<1 (note that if C>1, then the sequence diverges!).

So if we start with $\|\boldsymbol{e}^{0}\|$=1 and C=10^{-1}, then we have $\{\|\boldsymbol{e}^{k}\|\}$={1, 10^{-1}, 10^{-2}, 10^{-3},…}.

If C=0.99 then we have $\{\|\boldsymbol{e}^{k}\|\}$={1, .99, .9801, .9703,…} – not so good!

Case 2: r=2

If we start with $\|\boldsymbol{e}^{0}\|$=1 and C=10^{-1}, then $\{\|\boldsymbol{e}^{k}\|\}$={1, 10^{-1}, 10^{-3}, 10^{-7},…}.

If C=0.99 then we have $\{\|\boldsymbol{e}^{k}\|\}$={1, .99, .9703, 0.932, 0.86…}

Note that convergence is much faster: for C=10^{-1} we can get the error to 10^{-7} in 4 steps now as opposed to eight with linear convergence!

With linear convergence rates the reduction $\|\boldsymbol{x}^{k} - \boldsymbol{x}^{*}\|$ at each step will depend significantly on the rate constant C. For example, when C is small linear convergence can compare favorably with superlinear convergence.

EXAMPLE 2: Consider the sequence $\left\{x^{\left(2^{-k}\right)}\right\}$ where $x \geq 0$ (each element is the square root of the previous element). The limit of this sequence is 1.

If r=1 then $\lim_{k \to \infty}\left(\frac{\|\boldsymbol{e}^{k+1}\|}{\|\boldsymbol{e}^{k}\|}\right) = \lim_{k \to \infty}\left(\frac{x^{2^{-(k+1)}} - 1}{x^{2^{-k}} - 1}\right)$=0.5. Thus the sequence has a linear convergence rate with rate constant C=0.5

EXAMPLE 3: Consider the sequence $\{x^k\}$ with $x^k = a^k$ where $0<a<1$. The limit of this sequence is 0.

If r=1 then $\lim_{k \to \infty}\left(\frac{\|\boldsymbol{e}^{k+1}\|}{\|\boldsymbol{e}^{k}\|}\right) = \lim_{k \to \infty}\left(\frac{a^{k+1} - 0}{a^{k} - 0}\right)$=$a$. Thus the sequence has a linear convergence rate with rate constant a.

EXAMPLE 4: Consider the sequence $\left\{x^{\left(2^{k}\right)}\right\}$ where $0 \leq x < 1$ (each element is the square of the previous element). The limit of this sequence is 0.

If r=2 then $\lim_{k \to \infty}\left(\frac{\|\boldsymbol{e}^{k+1}\|}{\|\boldsymbol{e}^{k}\|^2}\right) = \lim_{k \to \infty}\left(\frac{x^{2^{(k+1)}}}{\left(x^{2^k}\right)^2}\right)$=1. Thus the sequence has a quadratic (and hence, superlinear) convergence rate with rate constant C=1

Stopping Criteria

The algorithm $\mathbb{A}$ terminates when we reach a point in Ω. Usually, convergence to a point in Ω occurs only in a limiting sense, and we need practical stopping criteria. Some common examples are,

1. Stop if the distance moved is less than ε after N successive iterations, that is, $\|\boldsymbol{x}^{k+N}-\boldsymbol{x}^{k}\|<\varepsilon$.
2. Stop if the change in $\boldsymbol{x}$ from one iteration to the next is less than ε%, i.e.,
$$\frac{\|x^{k+1}-x^{k}\|}{\|x^{k}\|}<\varepsilon$$
3. Stop when the improvement in the descent function z after N iterations is less than ε, i.e., $z(\boldsymbol{x}^{k}) - z(\boldsymbol{x}^{k+N}) < \varepsilon$.
4. Stop if the improvement in the descent function from one iteration to the next is less than ε%, i.e., $\frac{z(x^{k})-z(x^{k+1})}{|z(x^{k})|} < \varepsilon$.
5. If $z(\boldsymbol{x}^{*})$ is known in advance then stop when we are less than ε units from this, i.e., $|z(\boldsymbol{x}^{k}) - z(\boldsymbol{x}^{*})| < \varepsilon$.

ALGORITHMS FOR UNCONSTRAINED OPTIMIZATION

We have already seen simple search methods for unimodal functions. We now look at other methods.

NEWTON'S METHOD

Looks for a point satisfying the necessary conditions, i.e., it tries to solve $\nabla \boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0}$; a system of nonlinear equations. To do this it uses the Newton-Raphson method for each equation $\partial f / \partial x_i = 0$ for each i,

Recall that to solve $g(x) = 0$ by the Newton-Raphson method we used the Taylor series expansion around x^k at iteration k. Now consider a generalization to solve

$$g(\mathbf{x}) = g(x_1, x_2, \ldots, x_n) = 0$$

The Taylor series expansion around $\boldsymbol{x}^{\boldsymbol{k}}$ is

$$g(\boldsymbol{x}^{\boldsymbol{k}}+\Delta\boldsymbol{x}) = g(\boldsymbol{x}^{\boldsymbol{k}}) + [\nabla \boldsymbol{g}(\boldsymbol{x}^{\boldsymbol{k}})]^T [\Delta\boldsymbol{x}] \underbrace{+ \tfrac{1}{2} [\Delta\mathbf{x}]^T \nabla^2 \boldsymbol{g}(\boldsymbol{x}^{\boldsymbol{k}}) [\Delta\boldsymbol{x}] + \ldots}_{\text{(negligible terms…)}}$$

To choose $\Delta\boldsymbol{x}$ so that $g(\boldsymbol{x}^{\boldsymbol{k}}+\Delta\boldsymbol{x}) \cong 0$ and then move from the current point $\boldsymbol{x}^{\boldsymbol{k}}$ to a better point $\boldsymbol{x}^{\boldsymbol{k+1}}=\boldsymbol{x}^{\boldsymbol{k}} + \Delta\boldsymbol{x}$, we thus want $g(\boldsymbol{x}^{\boldsymbol{k+1}})= g(\boldsymbol{x}^{\boldsymbol{k}}+\Delta\boldsymbol{x})= g(\boldsymbol{x}^{\boldsymbol{k}}) + [\nabla \boldsymbol{g}(\boldsymbol{x}^{\boldsymbol{k}})]^T [\Delta\boldsymbol{x}] \cong 0$

Consider now when we have n simultaneous nonlinear equations $g_1 = g_2 = g_n = 0$. With each i we have the following system:

$[\nabla \boldsymbol{g_1}(\boldsymbol{x^k})]^T\, [\Delta \boldsymbol{x}] = -g_1(\boldsymbol{x^k})$

$[\nabla \boldsymbol{g_2}(\boldsymbol{x^k})]^T\, [\Delta \boldsymbol{x}] = -g_2(\boldsymbol{x^k})$

:

$[\nabla \boldsymbol{g_n}(\boldsymbol{x^k})]^T\, [\Delta \boldsymbol{x}] = -g_n(\boldsymbol{x^k})$

where $\Delta\boldsymbol{x} = \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \vdots \\ \Delta x_n \end{bmatrix}$. Thus we solve $\boldsymbol{J}\Delta\boldsymbol{x} = -\boldsymbol{g}$ to obtain $\Delta\boldsymbol{x} = -\boldsymbol{J}^{-1}\boldsymbol{g}$, i.e.,

$$\begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \vdots \\ \Delta x_n \end{bmatrix} = -\begin{bmatrix} \partial g_1/\partial x_1 & \partial g_1/\partial x_2 & \dots & \partial g_1/\partial x_n \\ \partial g_2/\partial x_1 & \partial g_2/\partial x_2 & \dots & \partial g_2/\partial x_n \\ \vdots & \vdots & \dots & \vdots \\ \partial g_n/\partial x_1 & \partial g_n/\partial x_2 & \dots & \partial g_n/\partial x_n \end{bmatrix}^{-1} \begin{bmatrix} g_1(\boldsymbol{x^k}) \\ g_2(\boldsymbol{x^k}) \\ \vdots \\ g_n(\boldsymbol{x^k}) \end{bmatrix}$$

where each partial derivative above is evaluated at $\boldsymbol{x^k}$,

Thus we have $\boxed{\Delta\boldsymbol{x} = -[\boldsymbol{J}(\boldsymbol{x^k})]^{-1}\, [\boldsymbol{g}(\boldsymbol{x^k})]}$ where the matrix $\boldsymbol{J}$ is called the JACOBIAN matrix and $\boldsymbol{J}(\boldsymbol{x^k})$ indicates that it is evaluated at the point $\boldsymbol{x^k}$. The $\boldsymbol{x^{k+1}}=\boldsymbol{x^k}+\Delta\boldsymbol{x}$

Let us apply this to solve $\nabla \boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0}$, i.e.,

$$\begin{aligned} g_1(\boldsymbol{x}) &\equiv \frac{\partial f}{\partial x_1}(\boldsymbol{x}) = 0 \\ g_2(\boldsymbol{x}) &\equiv \frac{\partial f}{\partial x_2}(\boldsymbol{x}) = 0 \\ &\vdots \\ g_n(\boldsymbol{x}) &\equiv \frac{\partial f}{\partial x_n}(\boldsymbol{x}) = 0 \end{aligned}$$

Thus we have $\boldsymbol{g}(\boldsymbol{x}) \equiv \nabla \boldsymbol{f}(\boldsymbol{x})$; [i.e., $g_i(\boldsymbol{x}) = \frac{\partial f}{\partial x_i}(\boldsymbol{x})$]

$$\Rightarrow \quad \frac{\partial g_i}{\partial x_j}(\boldsymbol{x}) \equiv \frac{\partial^2 f}{\partial x_j \partial x_i}(\boldsymbol{x})$$

So the Jacobian matrix for $\boldsymbol{g}(\boldsymbol{x})$ is

$$\begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix} = \boldsymbol{\nabla^2 f}$$

Notice that this is nothing but the Hessian matrix ($\boldsymbol{H}$) of the function f!

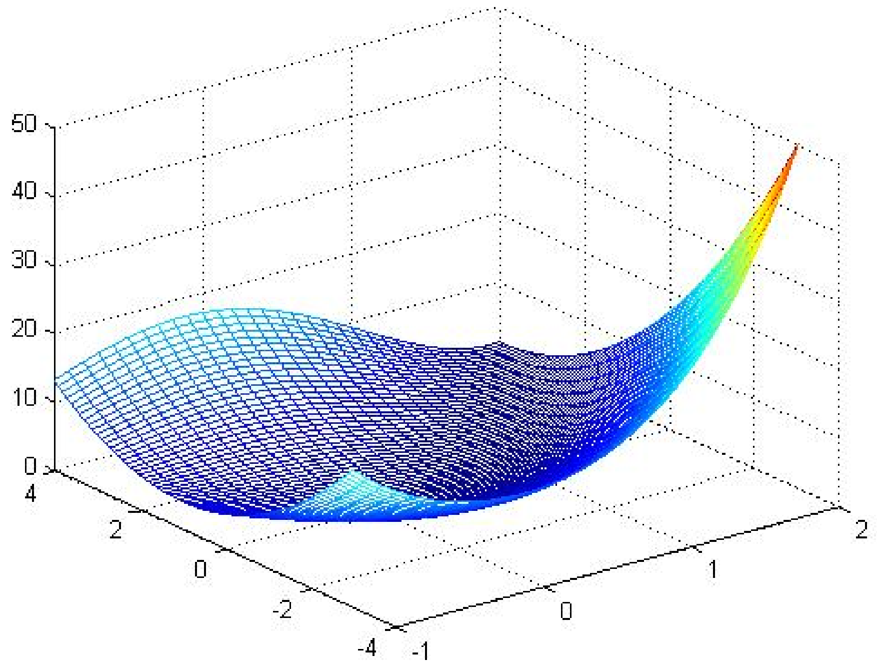
Thus if $\boldsymbol{x^i}$ is the guess at iteration i

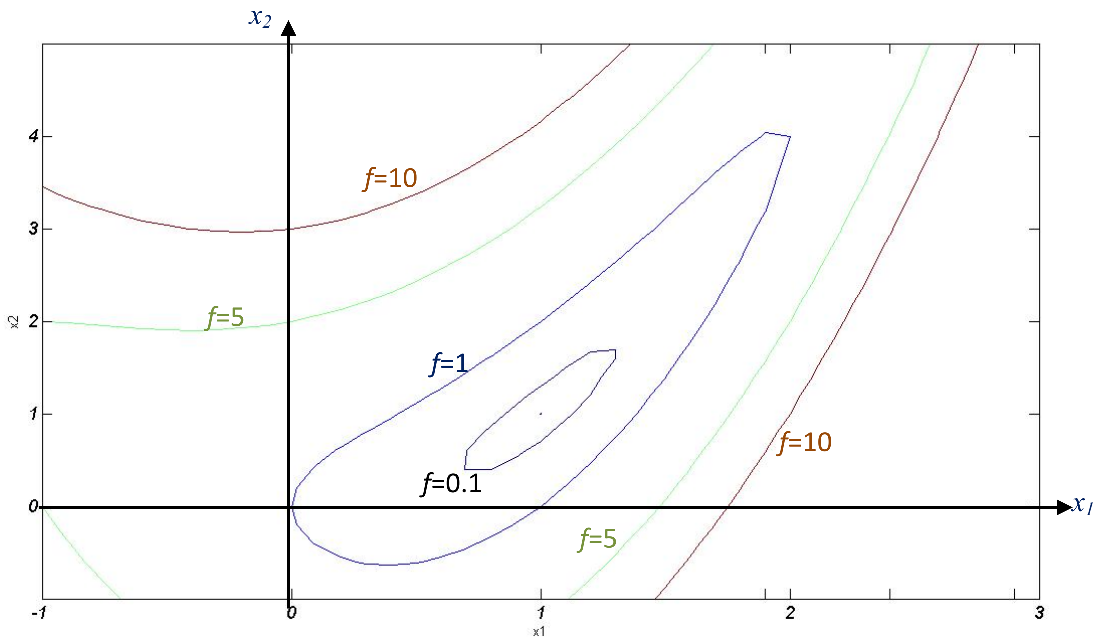
then $\Delta\boldsymbol{x} = -[\boldsymbol{J}(\boldsymbol{x^i})]^{-1}\ [\boldsymbol{g}(\boldsymbol{x^i})]$ implies that

$$\Delta\boldsymbol{x} = -[\nabla^2\boldsymbol{f}(\boldsymbol{x^i})]^{-1}\ [\nabla\boldsymbol{f}(\boldsymbol{x^i})] \ = \ -\boldsymbol{H}^{-1}\nabla\boldsymbol{f}$$

and $\boldsymbol{x^{i+1}} = \boldsymbol{x^i} + \Delta\boldsymbol{x}$ is the improved guess at iteration i+1.

AN EXAMPLE: Minimize $f(x_1, x_2) = (x_2 - x_1^2)^2 + (1-x_1)^2$





Contours of *f*(*x*) for various values

For this problem $\boldsymbol{x}^* = [1,1]$ and $f(\boldsymbol{x}^*)=0$,

$$\boldsymbol{\nabla f}(\boldsymbol{x}) = \begin{bmatrix} {}^{\partial f}/_{\partial x_1} \\ {}^{\partial f}/_{\partial x_2} \end{bmatrix} = \begin{bmatrix} 4x_1^3 - 4x_1x_2 + 2x_1 - 2 \\ 2(x_2 - x_1^2) \end{bmatrix}$$

and

$$\boldsymbol{\nabla^2 f}(\boldsymbol{x}) = \begin{bmatrix} \dfrac{\partial^2 f}{\partial x_1^2} & \dfrac{\partial^2 f}{\partial x_1\,\partial x_2} \\ \dfrac{\partial^2 f}{\partial x_1\,\partial x_2} & \dfrac{\partial^2 f}{\partial x_2^2} \end{bmatrix} = \begin{bmatrix} 12x_1^2 - 4x_2 + 2 & -4x_1 \\ -4x_1 & 2 \end{bmatrix}$$

ITERATION 1

Start with $\boldsymbol{x^0} = \begin{bmatrix}0\\0\end{bmatrix}$ where $f(\boldsymbol{x^0})=1$

$$\boldsymbol{\nabla f}(\boldsymbol{x^0}) = \begin{bmatrix}-2\\0\end{bmatrix}, \qquad \boldsymbol{H} = \boldsymbol{\nabla}^2\boldsymbol{f}(\boldsymbol{x^0}) = \begin{bmatrix}2 & 0\\0 & 2\end{bmatrix} \Rightarrow \boldsymbol{H^{-1}} = \begin{bmatrix}1/2 & 0\\0 & 1/2\end{bmatrix}$$

Therefore $\Delta\boldsymbol{x} = -\boldsymbol{H^{-1}\nabla f} = -\begin{bmatrix}1/2 & 0\\0 & 1/2\end{bmatrix}\begin{bmatrix}-2\\0\end{bmatrix} = \begin{bmatrix}1\\0\end{bmatrix}$

and $\boldsymbol{x^1} = \boldsymbol{x^0}+\Delta\boldsymbol{x} = \begin{bmatrix}0\\0\end{bmatrix} + \begin{bmatrix}1\\0\end{bmatrix} = \begin{bmatrix}1\\0\end{bmatrix}$

ITERATION 2 [$f(\boldsymbol{x^1})=1$]

$$\boldsymbol{\nabla f}(\boldsymbol{x^1}) = \begin{bmatrix}4\\-2\end{bmatrix}, \qquad \boldsymbol{H} = \boldsymbol{\nabla}^2\boldsymbol{f}(\boldsymbol{x^1}) = \begin{bmatrix}14 & -4\\-4 & 2\end{bmatrix} \Rightarrow \boldsymbol{H^{-1}} = \begin{bmatrix}1/6 & 1/3\\1/3 & 7/6\end{bmatrix}$$

$$\Delta\boldsymbol{x} = -\boldsymbol{H^{-1}\nabla f} = -\begin{bmatrix}1/6 & 1/3\\1/3 & 7/6\end{bmatrix}\begin{bmatrix}4\\-2\end{bmatrix} = \begin{bmatrix}0\\1\end{bmatrix}$$

and $\boldsymbol{x^2} = \boldsymbol{x^1}+\Delta\boldsymbol{x} = \begin{bmatrix}1\\0\end{bmatrix} + \begin{bmatrix}0\\1\end{bmatrix} = \begin{bmatrix}1\\1\end{bmatrix}$

ITERATION 3 [$f(\boldsymbol{x^2})=0$]

$\boldsymbol{\nabla f}(\boldsymbol{x^2}) = \begin{bmatrix}0\\0\end{bmatrix} \Rightarrow \begin{bmatrix}1\\1\end{bmatrix}$ is a stationary point. **STOP**

Suppose we had used a different starting point say $\boldsymbol{x}^0 = \begin{bmatrix}2\\1\end{bmatrix}$ where $f(\boldsymbol{x}^0)=10$

ITERATION 1 $[f(\boldsymbol{x}^0)=10]$

$$\boldsymbol{\nabla f}(\boldsymbol{x}^\mathbf{0}) = \begin{bmatrix}26\\-6\end{bmatrix}, \qquad \boldsymbol{H} = \begin{bmatrix}46 & -8\\-8 & 2\end{bmatrix} \Rightarrow \boldsymbol{H}^{-\mathbf{1}} = \begin{bmatrix}0.0714 & 0.2857\\0.2857 & 1.6428\end{bmatrix}$$

$$\Delta\boldsymbol{x} = -\boldsymbol{H}^{-\mathbf{1}}\boldsymbol{\nabla f} = -\begin{bmatrix}0.0714 & 0.2857\\0.2857 & 1.6428\end{bmatrix}\begin{bmatrix}26\\-6\end{bmatrix} = \begin{bmatrix}-0.1428\\2.4286\end{bmatrix}$$

and $\boldsymbol{x}^2 = \boldsymbol{x}^1 + \Delta\boldsymbol{x} = \begin{bmatrix}2\\1\end{bmatrix} + \begin{bmatrix}-0.1428\\2.4286\end{bmatrix} = \begin{bmatrix}1.8572\\3.4286\end{bmatrix}$

ITERATION 2 $[f(\boldsymbol{x}^1)=0.735]$

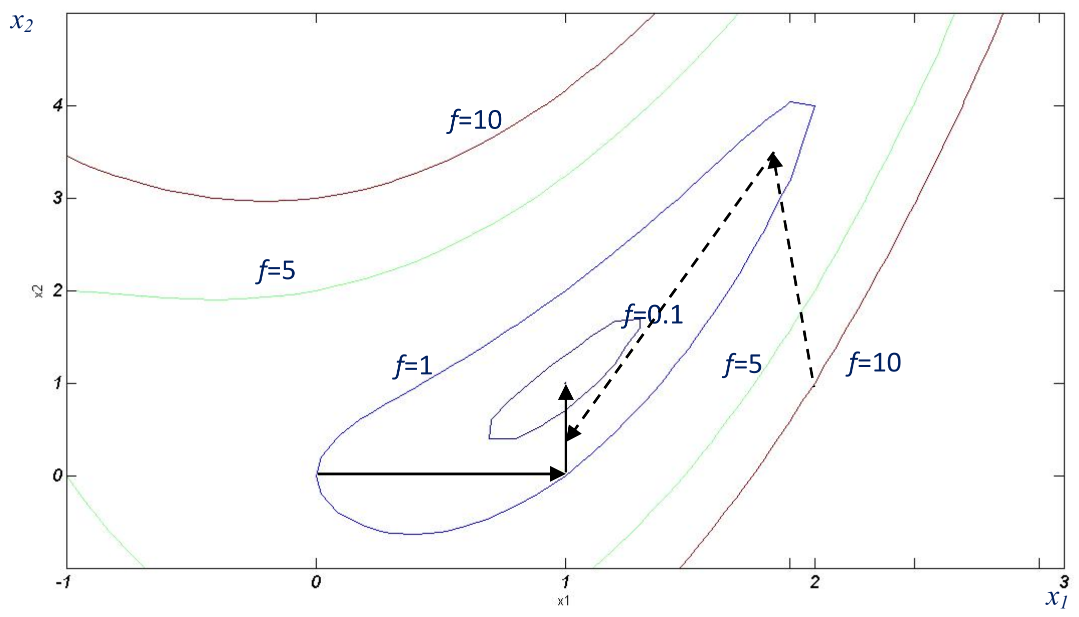
$$\boldsymbol{\nabla f}(\boldsymbol{x}^\mathbf{1}) = \begin{bmatrix}1.8569\\-0.0408\end{bmatrix} \ \boldsymbol{H} = \begin{bmatrix}29.6735 & -7.4286\\-7.4286 & 2\end{bmatrix} \Rightarrow \boldsymbol{H}^{-\mathbf{1}} = \begin{bmatrix}0.4804 & 1.7843\\1.7843 & 7.1275\end{bmatrix}$$

$$\Delta\boldsymbol{x} = -\boldsymbol{H}^{-\mathbf{1}}\boldsymbol{\nabla f} = -\begin{bmatrix}0.4804 & 1.7843\\1.7843 & 7.1275\end{bmatrix}\begin{bmatrix}1.8569\\-0.0408\end{bmatrix} = \begin{bmatrix}-0.8235\\-3.0384\end{bmatrix}$$

and $\boldsymbol{x}^2 = \boldsymbol{x}^1 + \Delta\boldsymbol{x} = \begin{bmatrix}1.8572\\3.4286\end{bmatrix} + \begin{bmatrix}-0.8235\\-3.0384\end{bmatrix} = \begin{bmatrix}1.0336\\0.3902\end{bmatrix}$

Worked well with $\begin{bmatrix}0\\0\end{bmatrix}$ as the starting point but with $\boldsymbol{x}^0 = \begin{bmatrix}2\\1\end{bmatrix}$ it gets messy!!

Solution Paths from two different starting points



$\boldsymbol{x^0} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ $\longrightarrow$

$\boldsymbol{x^0} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$ $\dashrightarrow$

Newton's method is simple to use and understand but it has several drawbacks,

- Convergence depends strongly on starting point
- Requires matrix inversion
- Requires second partial derivatives to be computed
- Since it only searches for a stationary point, it may sometimes not converge to a minimum

Newton's Method – An Alternative Viewpoint

Consider an approximation for f using the first three terms of the Taylor series expansion for f about the current iterate $\boldsymbol{x}^{\boldsymbol{k}}$:

$$f(\boldsymbol{x}^{\boldsymbol{k}}+\boldsymbol{d}) \approx Q(\boldsymbol{d}) = f(\boldsymbol{x}^{\boldsymbol{k}}) + \boldsymbol{d}^{\mathrm{T}}\nabla \boldsymbol{f}(\boldsymbol{x}^{\boldsymbol{k}}) + \tfrac{1}{2}\, \boldsymbol{d}^{\mathrm{T}}\nabla^2 \boldsymbol{f}(\boldsymbol{x}^{\boldsymbol{k}})\boldsymbol{d}$$

At every iteration Newton's method actually approximates $f(\boldsymbol{x})$ as above. Now suppose we try to minimize w.r.t. $\boldsymbol{d}$ the above approximation to find an optimal $\boldsymbol{d}$ by setting its gradient to zero (i.e. the necessary conditions $\nabla \boldsymbol{Q}(\boldsymbol{d})=\mathbf{0}$). We get

$$\nabla \boldsymbol{f}(\boldsymbol{x}^{\boldsymbol{k}}) + \nabla^2 \boldsymbol{f}\,(\boldsymbol{x}^{\boldsymbol{k}})\boldsymbol{d} = \mathbf{0},\ \text{ i.e., }\ \nabla^2 \boldsymbol{f}\,(\boldsymbol{x}^{\boldsymbol{k}})\boldsymbol{d} = -\nabla \boldsymbol{f}(\boldsymbol{x}^{\boldsymbol{k}})$$

which yields the same value for $\boldsymbol{d}$ that we saw earlier (the above system of linear equations is called the Newton Equations…).

This yields more insight into Newton's method:

- First, the approximation above may not be great as we move further away from $\boldsymbol{x}^{\boldsymbol{k}}$, or if the function is highly nonlinear, and thus if we start too far off from the optimum we may not converge at all.
- Second, even if we do converge, there's no guarantee we will converge to a minimum since we only look for a stationary point (i.e., one that satisfies the necessary conditions) – so we could wind up at a maximum or a saddle point.
- When the method DOES converge though it has a very good quadratic convergence rate; it works best when we're in the vicinity of the optimum.

Unconstrained Optimization

There are two basic strategies: (i) Line Search, and (ii) Trust Region

Line Search

Here we choose a **search direction** $\boldsymbol{d}^k$ along which the function f will improve, and search along this direction from the current iterate $\boldsymbol{x}^k$ for a better iterate. The distance to move along this direction (**step size**) is chosen by (approximately) solving the one-dimensional optimization problem

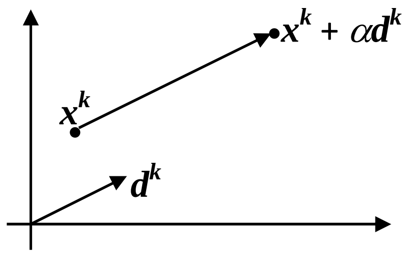
$$\text{Minimize } f(\boldsymbol{x}^k + \alpha \boldsymbol{d}^k), \text{ st } \alpha > 0$$

The value of α that solves the above (say α^*) yields $\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \alpha^* \boldsymbol{d}^k$ and the process continues from $\boldsymbol{x}^{k+1}$

Also, a search direction $\boldsymbol{d}^k$ often has the form

$$\boldsymbol{d}^k = -\boldsymbol{B}_k^{-1} \nabla \boldsymbol{f}(\boldsymbol{x}^k)$$

where $\boldsymbol{B}_k$ is some symmetric and nonsingular matrix. In Newton's method $\boldsymbol{B}_k$ is the exact Hessian $\nabla^2 \boldsymbol{f}(\boldsymbol{x}^k)$. For the steepest descent method it is simply the identity matrix $\boldsymbol{I}_k$ and for quasi-Newton methods it is an approximation of the Hessian that is updated at each iteration.



Trust Region

In this strategy information about f is used to construct a "model" m_k that approximates the original function well in some limited area around the current iterate $\boldsymbol{x^k}$ (called the "trust region") where we "trust" the approximation. We then try to minimize the approximation in the trust region, i.e., we solve for $\boldsymbol{d}$ via

"Minimize $m_k(\boldsymbol{x^k} + \boldsymbol{d})$, st $(\boldsymbol{x^k} + \boldsymbol{d})$ lies within the trust region"

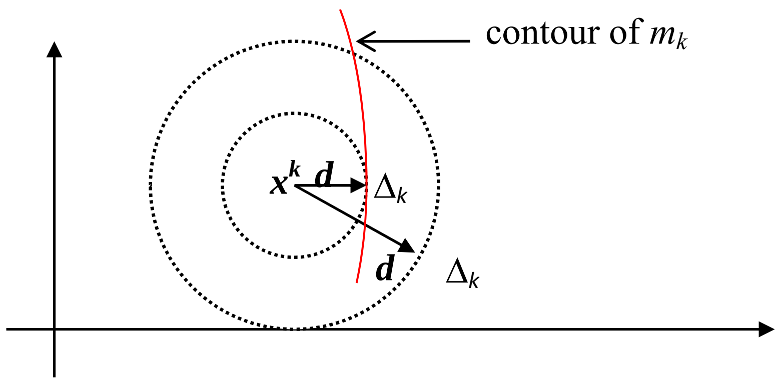
If the solution is not satisfactory we shrink the trust region and try again. Trust regions are typically balls, or boxes, or ellipsoids around $\boldsymbol{x^k}$.

A typical model m_k that approximates the function in the trust region around $\boldsymbol{x^k}$ is

$$m_k(\boldsymbol{x^k}+\boldsymbol{d}) \approx f(\boldsymbol{x^k}) + \boldsymbol{d}^{\mathrm{T}}\, \nabla \boldsymbol{f}(\boldsymbol{x^k}) + (\tfrac{1}{2})\boldsymbol{d}^{\mathrm{T}}\boldsymbol{B}(\boldsymbol{x^k})\boldsymbol{d}$$

where the matrix $\boldsymbol{B}$ is typically the Hessian (or some approximation of it) at $\boldsymbol{x^k}$.

A typical constraint for the trust region might be $\|\boldsymbol{d}\| \le \Delta_k$ so that we look inside a ball of radius Δ_k. If the optimum value $\boldsymbol{d}^*$ is unsatisfactory we reduce the value of Δ_k; otherwise we might choose to increase this value. Note that when we shrink the trust region the optimum $\boldsymbol{d}$ will in general, point in a different direction than the previous value of $\boldsymbol{d}$.



In some sense the two approaches differ in how we choose the **direction** to move along and the **distance** we go along this direction:

- In line search we first fix a direction $\boldsymbol{d}^k$ and then search along this direction for the distance α_k to move.
- In the trust region approach, we first choose a maximum distance Δ_k and then seek a direction and step that maximizes improvement, under the constraint that the step cannot be larger than the maximum distance Δ_k. If the performance in moving to a new point is unsatisfactory, we reduce the value of Δ_k and try again in a smaller trust region. Conversely if the performance is good we expand the size of the trust region. "Performance" here refers to the quality of the approximation, usually measured by a comparison of the predicted improvement in the objective using the approximation with the actual improvement in the true objective value.
- The actual "amount" of work at each step of the trust region approach could be a little more since we have a constrained optimization and since line search methods seldom try and find the exact step length. However, they still perform very well in practice and both approaches seem to be equally popular.

DESCENT METHODS

Suppose that f is a continuously differentiable function defined on $\mathbb{R}^n$.

<u>DEFINITION</u>: If $\nabla \boldsymbol{f}^{\mathrm{T}}(\bar{\boldsymbol{x}})\, \boldsymbol{d} < 0$, then $\boldsymbol{d}$ is called a *DIRECTION OF DESCENT* of the function f at the point $\bar{\boldsymbol{x}}$.

Suppose that $\boldsymbol{\delta}$ is some (sufficiently small) positive number. Then for all $\theta > 0$

$$(1) \qquad f(\bar{\boldsymbol{x}}+\theta\boldsymbol{d}) = f(\bar{\boldsymbol{x}}) + \nabla \boldsymbol{f}^{\mathrm{T}}(\bar{\boldsymbol{x}})\,(\theta\boldsymbol{d}) + \theta\,\|\boldsymbol{d}\|\,\beta(\boldsymbol{x}^{\boldsymbol{0}}, \theta\boldsymbol{d})$$

i.e., $\{f(\bar{\boldsymbol{x}}+\theta\boldsymbol{d}) - f(\bar{\boldsymbol{x}})\}/\theta = \nabla \boldsymbol{f}^{\mathrm{T}}(\bar{\boldsymbol{x}})\,(\boldsymbol{d}) + \|\boldsymbol{d}\|\,\beta(\boldsymbol{x}^{\boldsymbol{0}}, \theta\boldsymbol{d})$

Now, by definition $\|\boldsymbol{d}\|\,\beta(\boldsymbol{x}^{\boldsymbol{0}}, \theta\boldsymbol{d}) \rightarrow 0$ as $\theta \rightarrow 0$, and since $\nabla \boldsymbol{f}^{\mathrm{T}}(\bar{\boldsymbol{x}})(\boldsymbol{d}) < 0$, it follows that $\{f(\bar{\boldsymbol{x}}+\theta\boldsymbol{d}) - f(\bar{\boldsymbol{x}})\}/\theta < 0$, i.e., $f(\bar{\boldsymbol{x}}+\theta\boldsymbol{d}) < f(\bar{\boldsymbol{x}})$ for <u>all</u> values of θ that are smaller than some (sufficiently small) positive value δ, i.e. for all $\theta \in (0, \delta)$.

In other words, *by moving some finite distance* θ *along the direction of descent* $\mathbf{d}$ *from the current point* $\bar{\mathbf{x}}$ *to the new point* $\bar{\mathbf{x}}+\theta\mathbf{d}$*, we can reduce the value of* $f(\mathbf{x})$ *from* $f(\bar{\boldsymbol{x}})$ *to* $f(\bar{\boldsymbol{x}}+\theta\boldsymbol{d})$.

The question is how far to move, since if we move too far ($\theta > \delta$) we are no longer guaranteed a reduction in the value of f.

To formalize this:

Given the current approximation $\boldsymbol{x}^{i}$ we have to answer the following:

1. Is $\boldsymbol{x}^{i}$ "sufficiently" near optimal? If so, STOP.
2. In what direction $\boldsymbol{d}^{i}$ should the next point $\boldsymbol{x}^{i+1}$ be located?
3. How far along this direction $\boldsymbol{d}^{i}$ should we move from $\boldsymbol{x}^{i}$ to locate $\boldsymbol{x}^{i+1}$?

i.e., $\boldsymbol{x}^{i+1} = \boldsymbol{x}^{i} + \alpha_i \boldsymbol{d}^{i}$ where

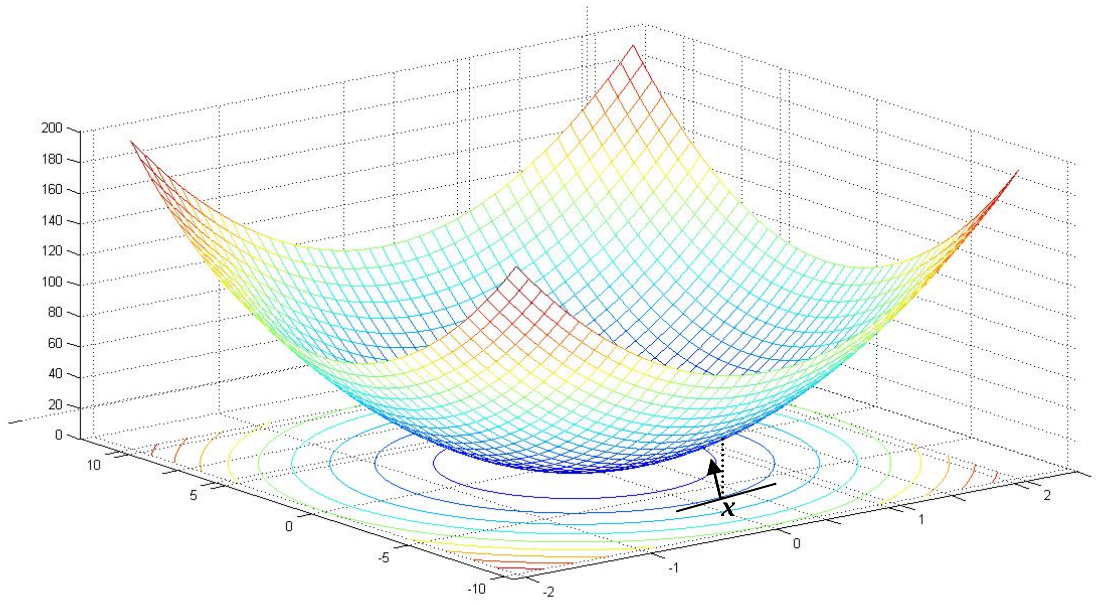
- α_i is a scalar quantity called the **step size**.
- $\boldsymbol{d}^{i}$ is a vector quantity called the **search direction**.

In Newton's method $\boldsymbol{x}^{i+1} = \boldsymbol{x}^{i} + [-\boldsymbol{H}^{-1}\ \nabla \boldsymbol{f}]$, so that α_i is fixed at 1.0, and $\boldsymbol{d}^{i} = -\boldsymbol{H}^{-1}\nabla \boldsymbol{f} = -[\nabla^2 \boldsymbol{f}(\boldsymbol{x}^{i})]^{-1}\ \nabla \boldsymbol{f}(\boldsymbol{x}^{i})$.

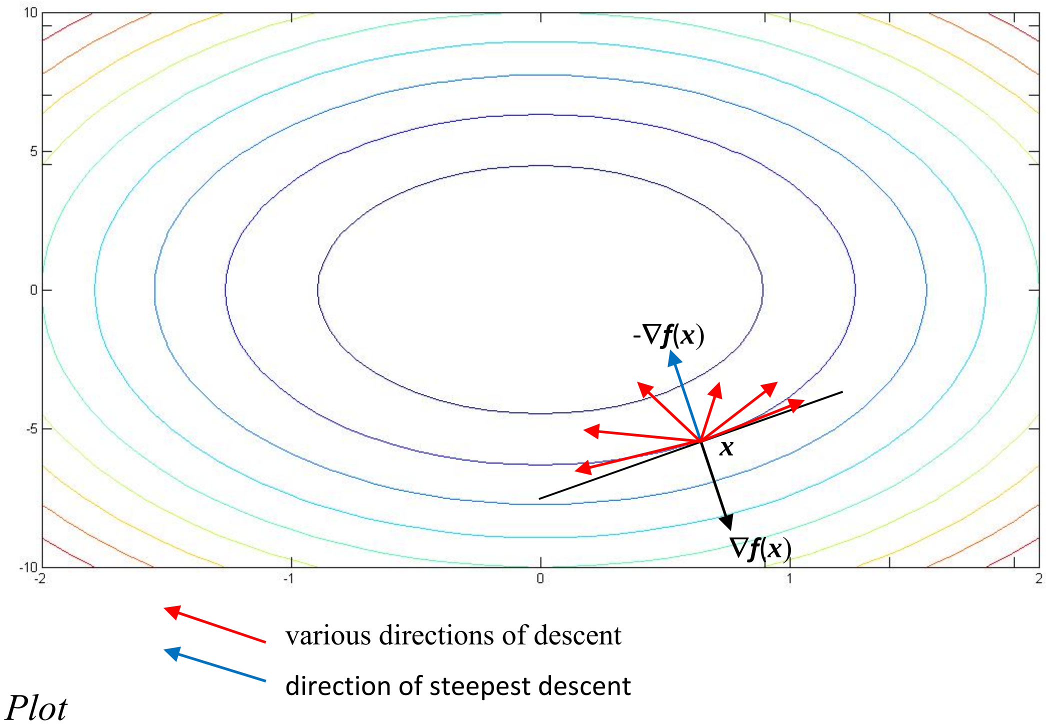
In the **STEEPEST DESCENT** method we choose $\boldsymbol{d}^{i}$ differently. If $\Delta f = f(\boldsymbol{x}^{i+1}) - f(\boldsymbol{x}^{i})$ and $[\Delta \boldsymbol{x}] = \boldsymbol{x}^{i+1} - \boldsymbol{x}^{i}$, then from (1), $\Delta f \approx \nabla \boldsymbol{f}^{T}(x^{i})\ [\Delta \mathbf{x}]$.

It is easy to see that $|\Delta f|$ can be maximized (i.e., the greatest reduction in f obtained) by choosing $\Delta \boldsymbol{x} = -\theta \nabla \boldsymbol{f}^{T}(\boldsymbol{x}^{i})$. (The gradient vector points in the direction of steepest ascent).

Thus we select $\boldsymbol{d}^{i} = -\nabla \boldsymbol{f}(\boldsymbol{x}^{i})$.



3-dimensional plot



Contour Plot

$$f(x) = 25x_1^2 + x_2^2$$

The step size α_i is usually found by means of a one-dimensional search method (with α_i as the decision variable and $\boldsymbol{x^i} + \alpha_i\boldsymbol{d^i}$ as the function to be minimized for $\boldsymbol{x^i}$ and $\boldsymbol{d^i}$ given), such as the Golden-Section or Fibonacci or quadratic interpolation etc. (start with some initial interval for α_i and find the point inside that interval that minimizes $f(\boldsymbol{x^i}+ \alpha_i\boldsymbol{d^i})$.

The method continues until $\left\|\nabla f(\boldsymbol{x^k})\right\| \leq \varepsilon$, where ε is some user-specified tolerance.

AN EXAMPLE: Minimize $f(x_1, x_2) = x_1^2 + 25x_2^2$

$$\boldsymbol{\nabla f(x)} = \begin{bmatrix} 2x_1 \\ 50x_2 \end{bmatrix}$$

<u>ITERATION 1</u>: Let $\boldsymbol{x^0} = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$

(Note that Newton's method would find the minimum $f(0,0)=0$ in one step since f is quadratic and $\nabla \boldsymbol{f}=\boldsymbol{0}$ is thus simply a set of linear equations!).

$$\boldsymbol{\nabla f(x^0)} = \begin{bmatrix} 4 \\ 100 \end{bmatrix} \quad \|\boldsymbol{\nabla f(x^0)}\| = 100.08 \quad \boldsymbol{d^0} = \begin{bmatrix} -4 \\ -100 \end{bmatrix}$$

$$\boldsymbol{x^1} = \boldsymbol{x^0} + \alpha\boldsymbol{d^0} = \begin{bmatrix} 2 \\ 2 \end{bmatrix} + \alpha\begin{bmatrix} -4 \\ -100 \end{bmatrix} = \begin{bmatrix} 2-4\alpha \\ 2-100\alpha \end{bmatrix}$$

$f(\boldsymbol{x^1}) = (2-4\alpha)^2 + 25(2-100\alpha)^2 = 250016\alpha^2 - 10016\alpha + 104$

Q. How large should α be to fix $\boldsymbol{x^1}$?

A. We try to choose α so that $f(\boldsymbol{x^1})$ is minimized.

The minimum of $f(2\text{-}4\alpha,\ 2\text{-}100\alpha)$ occurs at α_0=0.02003.

$$\therefore\ \boldsymbol{x}^1 = \boldsymbol{x}^0 + \alpha_0 \boldsymbol{d}^0 = \begin{bmatrix} 1.920 \\ -0.003 \end{bmatrix} \text{ and } f(\boldsymbol{x}^1) = 3.19.$$

<u>ITERATION 2</u>: $\boldsymbol{x}^1 = \begin{bmatrix} 1.920 \\ -0.003 \end{bmatrix}$

$$\boldsymbol{\nabla f}(\boldsymbol{x}^1) = \begin{bmatrix} 3.84 \\ -0.15 \end{bmatrix} \quad \|\boldsymbol{\nabla f}(\boldsymbol{x}^1)\| = 3.84 \quad \boldsymbol{d}^1 = \begin{bmatrix} -3.84 \\ 0.15 \end{bmatrix}$$

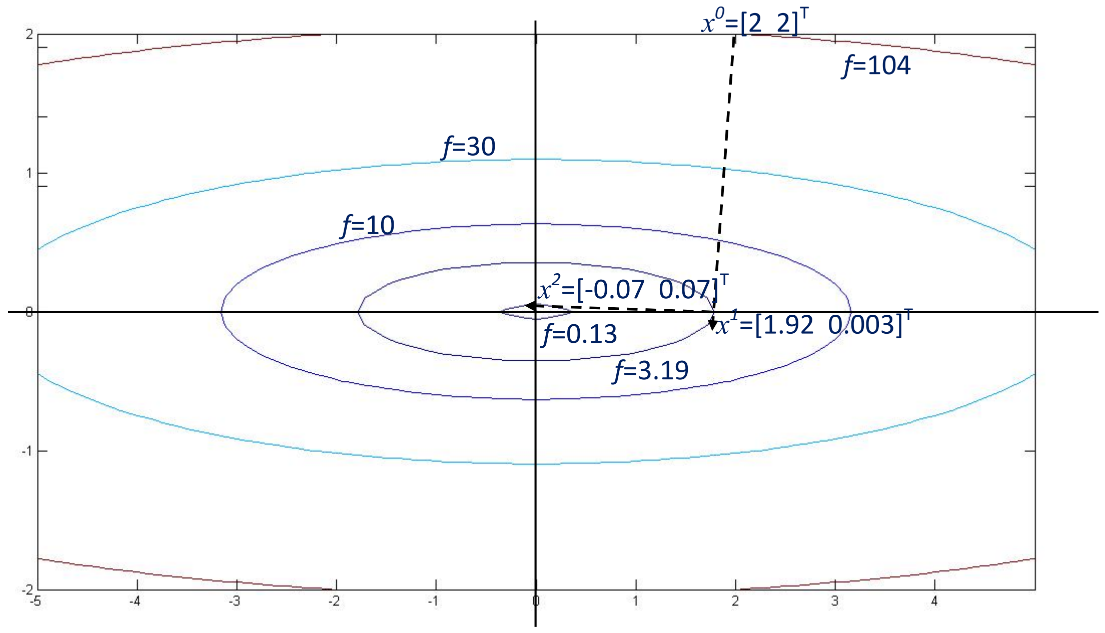
$$\boldsymbol{x}^2 = \boldsymbol{x}^1 + \alpha \boldsymbol{d}^1 = \begin{bmatrix} 1.92 - 3.84\alpha \\ -0.003 + 0.15\alpha \end{bmatrix}$$

$f(\boldsymbol{x}^2)$is minimized at $\alpha_1 \cong 0.48$.

$$\therefore\ \boldsymbol{x}^2 = \boldsymbol{x}^1 + \alpha_1 \boldsymbol{d}^1 = \begin{bmatrix} -0.07 \\ 0.07 \end{bmatrix} \text{ and } f(\boldsymbol{x}^2) = 0.13.$$

<u>ITERATION 3</u>: $\boldsymbol{x}^2 = \begin{bmatrix} -0.07 \\ 0.07 \end{bmatrix}$

$$\boldsymbol{\nabla f}(\boldsymbol{x}^2) = \begin{bmatrix} -0.14 \\ 3.50 \end{bmatrix} \quad \|\boldsymbol{\nabla f}(\boldsymbol{x}^2)\| = 3.50 \quad \text{etc. etc. etc.}$$



THE QUADRATIC CASE

Consider the convex quadratic function $f(x)= ½\, \boldsymbol{x}^{\mathrm{T}} [\boldsymbol{Q}]\boldsymbol{x} + \boldsymbol{x}^{\mathrm{T}}\boldsymbol{b}$, where $[\boldsymbol{Q}]$ is an $(n\times n)$ symmetric positive definite matrix (the previous example falls into this category). We have

$$\nabla f^{\mathrm{T}}(\boldsymbol{x}) = \boldsymbol{Qx} + \boldsymbol{b}$$

and the method of steepest descent is

$$\boldsymbol{x}^{i+1} = \boldsymbol{x}^{i} - \alpha\nabla \boldsymbol{f}(\boldsymbol{x}^{i}).$$

If we choose α so as to minimize the function w.r.t. α

$$f(\boldsymbol{x}^{i}-\alpha\nabla \boldsymbol{f}(\boldsymbol{x}^{i})) = ½\, [\boldsymbol{x}^{i}-\alpha\nabla \boldsymbol{f}(\boldsymbol{x}^{i})]^{\mathrm{T}} \boldsymbol{Q}\, [\boldsymbol{x}^{i}-\alpha\nabla \boldsymbol{f}(\boldsymbol{x}^{i})] - [\boldsymbol{x}^{i}-\alpha\nabla \boldsymbol{f}(\boldsymbol{x}^{i})]^{\mathrm{T}}\boldsymbol{b}$$

For the sake of convenience, let us denote $\nabla \boldsymbol{f}_{\mathrm{i}} = \nabla \boldsymbol{f}(\boldsymbol{x}^{i})$. It follows that

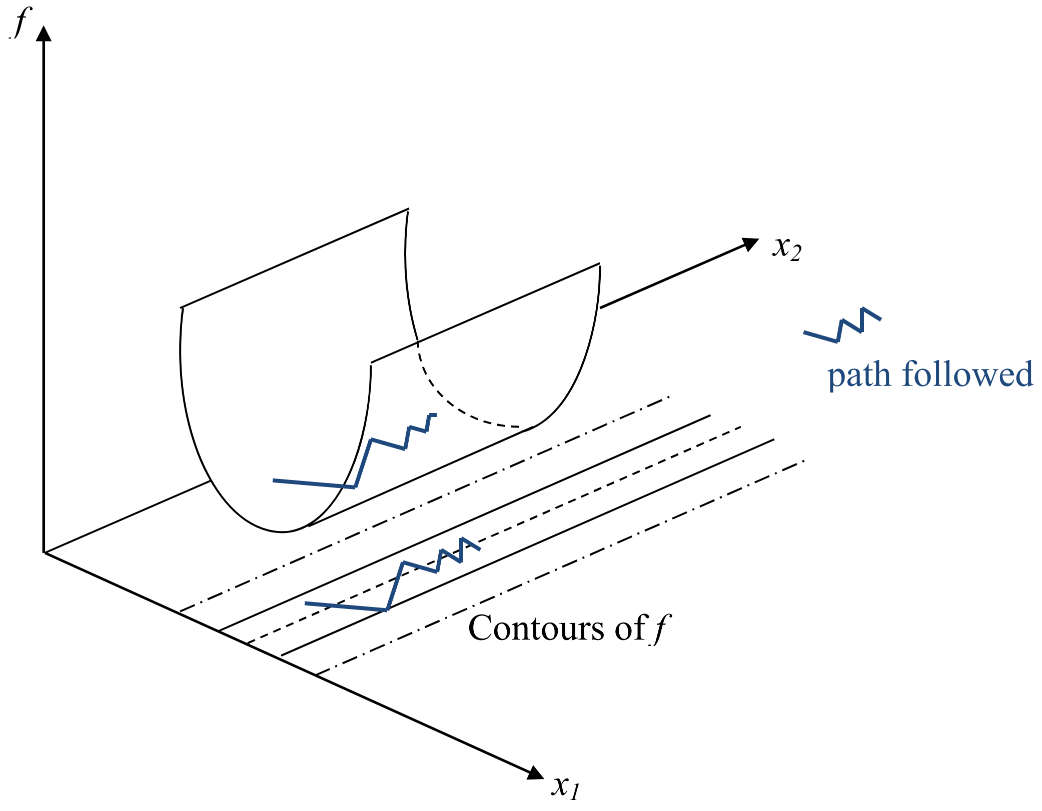
$$\alpha_i = \frac{\nabla \boldsymbol{f}_i^{\mathrm{T}} \nabla \boldsymbol{f}_i}{\nabla \boldsymbol{f}_i^{\mathrm{T}}\, \boldsymbol{Q} \nabla \boldsymbol{f}_i}$$

Thus the new point is,

$$\boldsymbol{x}^{i+1} = \boldsymbol{x}^{i} - \left(\frac{\nabla \boldsymbol{f}_i^{\mathrm{T}} \nabla \boldsymbol{f}_i}{\nabla \boldsymbol{f}_i^{\mathrm{T}}\, \boldsymbol{Q} \nabla \boldsymbol{f}_i}\right) \nabla \boldsymbol{f}(\boldsymbol{x}^{i})$$

PROPOSITION: For the quadratic case successive search directions are mutually orthogonal. (PROVE and VERIFY using the previous example...)

The main problem with steepest descent is the phenomenon of "zig-zagging" or "oscillation" when travelling along a narrow "valley", especially if α_k (the step size) is not exactly determined to minimize along the direction $\boldsymbol{d}^k$.



VARIANTS OF THE STEEPEST DESCENT METHOD

Alternate method for choosing step size α_k

Often it may be difficult to compute the step size based upon an exact line search, or the function may not be unimodal. Thus the step size may be chosen by some other logic:

STEP 0: Select an ε such that $0<\varepsilon<1$ (usually $\varepsilon \leq 0.5$).

Also, choose any $\alpha>0$.

STEP 1: Compute $\boldsymbol{x} = \boldsymbol{x}^{\boldsymbol{i}} - \alpha \nabla \boldsymbol{f}(\boldsymbol{x}^{\boldsymbol{i}})$,

STEP 2: Compute $f(x)$ and check if,

$$|f(\boldsymbol{x}^{\boldsymbol{i}}) - f(\boldsymbol{x})| \geq \varepsilon . \alpha . \| \nabla \boldsymbol{f}(\boldsymbol{x}^{\boldsymbol{i}}) \|^2$$

STEP 3: If the above inequality is satisfied, then set $\alpha_i = \alpha$ and end the line search.

Else, reduce α (perhaps by multiplying α by some $\lambda<1$) and return to Step 1.

Note: The initial choice of α is critical. Normally the initial values of α are relatively large during the initial iterations and then become successively smaller. This procedure ensures a "sufficient decrease" without exact computation of the minimum along the search direction.

Guaranteeing Descent

Recall that a valid search direction $\boldsymbol{d}$ is required to satisfy $\nabla \boldsymbol{f}^T \boldsymbol{d} < 0$.

With Newton's method $\boldsymbol{d} = -(\nabla^2 \boldsymbol{f})^{-1} \nabla \boldsymbol{f} = -\boldsymbol{H}^{-1} \nabla \boldsymbol{f}$

Thus we need to satisfy $\nabla \boldsymbol{f}^T \boldsymbol{d} = -\nabla \boldsymbol{f}^T \boldsymbol{H}^{-1} \nabla \boldsymbol{f} < 0$, i.e. $\nabla \boldsymbol{f}^T \boldsymbol{H}^{-1} \nabla \boldsymbol{f} > 0$

If $\boldsymbol{H}$ is positive definite this condition is always satisfied. But what if $\boldsymbol{H}$ is not positive definite (e.g., it is indefinite)? Note that requiring that $\boldsymbol{H}$ be positive definite is a stronger requirement than just requiring that $\nabla \boldsymbol{f}^T \boldsymbol{d} < 0$.

<u>STRATEGY</u>: Replace $\boldsymbol{H}$ with some "related" matrix ($\boldsymbol{A}$) that is positive definite!

<u>Justification</u>:

- This guarantees descent
- If done properly this guarantees convergence with a line search
- Since $\boldsymbol{H}$ is usually positive definite (and definitely, at least positive semidefinite) at the optimum $\boldsymbol{x}^*$ we usually need to do this only at points far away from the optimum
- The related matrix $\boldsymbol{A}$ is easily found with little extra effort

Consider the system $\boldsymbol{Hd} = -\nabla \boldsymbol{f}$ that is solved to find $\boldsymbol{d}$.

If $\boldsymbol{H}$ is positive definite it can always be factorized as $\boldsymbol{H}=\boldsymbol{LDL}^{\mathrm{T}}$, where $\boldsymbol{D}$ is a positive diagonal matrix and $\boldsymbol{L}$ is lower triangular so we just solve $\boldsymbol{LDL}^{\mathrm{T}}\boldsymbol{d} = -\nabla \boldsymbol{f}$.

This is easily done in two stages involving just back substitution (let $\boldsymbol{u}=\boldsymbol{DL}^{\mathrm{T}}\boldsymbol{d}$ and find $\boldsymbol{u}$ first, then solve $\boldsymbol{u}=\boldsymbol{DL}^{\mathrm{T}}\boldsymbol{d}$ for $\boldsymbol{d}$).

If $\boldsymbol{H}$ is not positive definite then some element d_{ii} of $\boldsymbol{D}$ is negative and we revise the factorization so that these values are replaced. It can be shown that this is equivalent to replacing $\boldsymbol{H}$ with a matrix $\boldsymbol{A} = \boldsymbol{H}+\boldsymbol{E}$ (where $\boldsymbol{E}$ is diagonal) so that $\boldsymbol{A}$ is positive definite. We then factor this matrix as usual.

<u>Example:</u>

Suppose $\boldsymbol{H} = \begin{bmatrix} -1 & 2 & 4 \\ 2 & -3 & 6 \\ 4 & 6 & 22 \end{bmatrix}$. This is not positive definite. However if we define

$\boldsymbol{E} = \begin{bmatrix} 5 & 0 & 0 \\ 0 & 12 & 0 \\ 0 & 0 & 0 \end{bmatrix}$, then $\boldsymbol{A}=\boldsymbol{H}+\boldsymbol{E} = \begin{bmatrix} 4 & 2 & 4 \\ 2 & 9 & 6 \\ 4 & 6 & 22 \end{bmatrix}$ is positive definite and we can factorize

it as

$$\boldsymbol{A} = \begin{bmatrix} 4 & 2 & 4 \\ 2 & 9 & 6 \\ 4 & 6 & 22 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1/2 & 1 & 0 \\ 1 & 1/2 & 1 \end{bmatrix} \begin{bmatrix} 4 & 0 & 0 \\ 0 & 8 & 0 \\ 0 & 0 & 16 \end{bmatrix} \begin{bmatrix} 1 & 1/2 & 1 \\ 0 & 1 & 1/2 \\ 0 & 0 & 1 \end{bmatrix}$$

And we would now use this to solve for $\boldsymbol{d}$ via

$$\begin{bmatrix} 1 & 0 & 0 \\ 1/2 & 1 & 0 \\ 1 & 1/2 & 1 \end{bmatrix} \begin{bmatrix} 4 & 0 & 0 \\ 0 & 8 & 0 \\ 0 & 0 & 16 \end{bmatrix} \begin{bmatrix} 1 & 1/2 & 1 \\ 0 & 1 & 1/2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} \nabla f_1 \\ \nabla f_2 \\ \nabla f_3 \end{bmatrix}$$

This is easily solved by using two back substitution stages…

Globalization Strategies

Typically we compute a search direction based on a Taylor series approximation. "Globalization strategies" ensure that even when the approximation is not great (e.g. far from the solution) we can use the direction to pick a new point in a fashion that ensures global convergence to a stationary point.

Consider the typical line search formula for a new iterate: $\boldsymbol{x}^{\boldsymbol{k+1}} = \boldsymbol{x}^{\boldsymbol{k}} + \alpha \boldsymbol{d}^{\boldsymbol{k}}$

Suppose $\boldsymbol{d}^{\boldsymbol{k}}$ is a descent direction, i.e., $(\boldsymbol{d}^{\boldsymbol{k}})^{\mathrm{T}}\nabla \boldsymbol{f}(\boldsymbol{x}^{\boldsymbol{k}}) < 0$. So ideally, we would like to exactly minimize $f(\boldsymbol{x}^{\boldsymbol{k}} + \alpha \boldsymbol{d}^{\boldsymbol{k}})$, st $\alpha>0$. But this could be too expensive, so in practice, we accept an *approximate* minimizer that reduces f "sufficiently."

Search Direction: To guarantee convergence we need to make two assumptions

A. It produces "sufficient" descent

Assuming we can always choose $\boldsymbol{d}^{\boldsymbol{k}}$ such that $(\boldsymbol{d}^{\boldsymbol{k}})^{\mathrm{T}}\nabla \boldsymbol{f}(\boldsymbol{x}^{\boldsymbol{k}}) < 0$, we want to be sure that the quantity $(\boldsymbol{d}^{\boldsymbol{k}})^{\mathrm{T}}\nabla \boldsymbol{f}(\boldsymbol{x}^{\boldsymbol{k}})$ is "sufficiently" negative, i.e., that it is not "too close" to zero and thus slows down progress substantially. Since we can write

$$(\boldsymbol{d}^{\boldsymbol{k}})^{\mathrm{T}}\nabla \boldsymbol{f}(\boldsymbol{x}^{\boldsymbol{k}}) = \left\|\boldsymbol{d}^{\boldsymbol{k}}\right\| \left\|\nabla \boldsymbol{f}(\boldsymbol{x}^{\boldsymbol{k}})\right\| \cos\theta$$

this is equivalent to requiring that the angle θ be sufficiently obtuse, i.e. that

$$\cos\theta = \frac{(\boldsymbol{d}^{\boldsymbol{k}})^T \nabla \boldsymbol{f}(\boldsymbol{x}^{\boldsymbol{k}})}{\left\|d^{\boldsymbol{k}}\right\| \cdot \left\|\nabla \boldsymbol{f}(\boldsymbol{x}^{\boldsymbol{k}})\right\|} \leq -\varepsilon < 0, \text{ i.e., } \qquad -\frac{(\boldsymbol{d}^{\boldsymbol{k}})^T \nabla \boldsymbol{f}(\boldsymbol{x}^{\boldsymbol{k}})}{\left\|d^{\boldsymbol{k}}\right\| \cdot \left\|\nabla \boldsymbol{f}(\boldsymbol{x}^{\boldsymbol{k}})\right\|} \geq \varepsilon > 0,$$

where $\varepsilon>0$ is some specified tolerance. This is called the ***Angle Condition***.

B. <u>It is "gradient related"</u>

The search direction is said to be gradient-related if

$$\|\boldsymbol{d}^{\boldsymbol{k}}\| \geq m \|\nabla \boldsymbol{f}(\boldsymbol{x}^{\boldsymbol{k}})\|$$

for all k, where $m>0$ is some constant. This ensures that the norm of the search direction is not too much smaller than that of the gradient. Note that for the steepest descent method this is true with $m=1$.

Step Size: Similarly, with the step size we need to make two assumptions as well

A. <u>It produces a "sufficient" decrease in the objective f</u>

Here "sufficiency" is related to the Taylor series: a linear (first order) approximation would be $f(\boldsymbol{x}^k+\alpha\boldsymbol{d}^k) \approx f(\boldsymbol{x}^k) + \alpha(\boldsymbol{d}^k)^T\nabla\boldsymbol{f}(\boldsymbol{x}^k)$. We will require that α produce a decrease that is *at least some fraction of the decrease* $-\alpha(\boldsymbol{d}^k)^T\nabla\boldsymbol{f}(\boldsymbol{x}^k)$ *predicted by the above approximation*, i.e,

$$f(\boldsymbol{x}^k) - f(\boldsymbol{x}^k+\alpha\boldsymbol{d}^k) \geq \varepsilon\{-\alpha(\boldsymbol{d}^k)^T\nabla\boldsymbol{f}(\boldsymbol{x}^k)\},$$

i.e., $$\phi(\alpha) = f(\boldsymbol{x}^k+\alpha\boldsymbol{d}^k) \leq f(\boldsymbol{x}^k) + \varepsilon\alpha(\boldsymbol{d}^k)^T\nabla\boldsymbol{f}(\boldsymbol{x}^k)$$

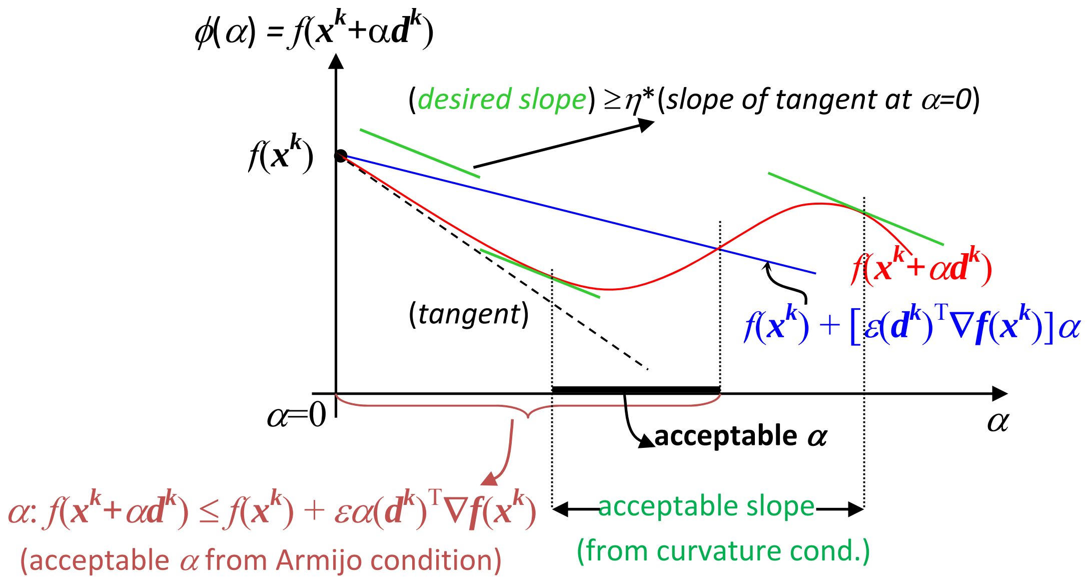
where $0<\varepsilon<1$. This is called the *Armijo* condition

B. <u>It is not "too small"</u>

In this condition we wish to ensure that α itself is not too small. One simple approach to ensure that we don't have unacceptably short steps is to start with some value of α (e.g., $\alpha=1$ corresponding to the ideal Newton step) and if it does not satisfy the Armijo condition, we reduce α by half and continue doing so until we find an α for which the decrease in f is sufficiently large.

There are other methods for setting α that are more efficient and do the same job. For example, the ***curvature condition*** which states that $(\boldsymbol{d}^k)^T\nabla f(\boldsymbol{x}^k+\alpha\boldsymbol{d}^k) \geq \eta(\boldsymbol{d}^k)^T\nabla \boldsymbol{f}(\boldsymbol{x}^k)$ where $0<\varepsilon<\eta<1$ and ε is the value used for the Armijo condition. Note that the LHS is the derivative of $\phi(\alpha)$, i.e., its slope and we just want this to be greater than η times the slope of $\phi(0)$.

The Armijo and curvature conditions together are called the ***Wolfe Conditions***


$\phi(\alpha) = f(\boldsymbol{x}^k+\alpha\boldsymbol{d}^k)$
(desired slope) $\geq \eta$*(slope of tangent at α=0)
$f(\boldsymbol{x}^k)$
$f(\boldsymbol{x}^k+\alpha\boldsymbol{d}^k)$
(tangent)
$f(\boldsymbol{x}^k) + [\varepsilon(\boldsymbol{d}^k)^T\nabla\boldsymbol{f}(\boldsymbol{x}^k)]\alpha$
α=0
acceptable α
α
α: $f(\boldsymbol{x}^k+\alpha\boldsymbol{d}^k) \leq f(\boldsymbol{x}^k) + \varepsilon\alpha(\boldsymbol{d}^k)^T\nabla\boldsymbol{f}(\boldsymbol{x}^k)$
(acceptable α from Armijo condition)
acceptable slope
(from curvature cond.)

<u>THEOREM</u>: If the above four conditions (two on search direction and two on step size) are satisfied for a general line-search type descent algorithm (along with some additional assumptions such as compactness, etc.), then it can be proved that the algorithm will converge to a stationary point having $\|\nabla \boldsymbol{f}(\boldsymbol{x})\|=0$

Alternate Stopping Criteria

The termination criterion of $\|\nabla \boldsymbol{f}(\boldsymbol{x})\| < \varepsilon$ is good only for small, well-scaled problems. Most engineering design problems are however, not well-scaled. For instance consider the problem

$$\text{Minimize } f(x) = 1 + \frac{10^{-9}}{2}(x-10^5)^2$$

At $x^0 = 0$, $f(x^0)=6$, $\|f'(x^0)\| = 10^{-4}$.

At the optimum point $x^* = 10^5$ we have $f(x^*)=1$,

Thus if the termination criterion had been $\|\nabla \boldsymbol{f}(\boldsymbol{x})\| < \varepsilon$, where, say $\varepsilon=10^{-3}$, the program would have stopped at x^0. However, by proceeding further a reduction of $f(x^0)-f(x^*) = (6-1)$ units, i.e., $(6-1)/1 = 500\%$ is possible!

It should be emphasized that no one set of termination criteria is suitable for all cases. Termination criteria depend on nonlinearity, scaling and magnitudes of functions and variables. From an engineering standpoint $|f_i-f|$ is more important $\| \boldsymbol{x}^i-\boldsymbol{x}^* \|$.

Consequently we may use:

$$|f(x^k) - f(x^{k-1})| < \varepsilon \text{ for } k=i,i-1,\ldots,i-m$$

i.e., change in objective is less than a specified tolerance for *m successive iterations*.

Gill, Murray & Wright suggest the following conditions be *simultaneously* satisfied:

(1) $f_{i-1} - f_i \leq \varepsilon(1+|f_i|)$

(2) $\|\boldsymbol{x}^{i-1} - \boldsymbol{x}^i\| \leq (\varepsilon)^{1/2}(1+\|\boldsymbol{x}^i\|)$

(3) $\|\nabla \boldsymbol{f}(\boldsymbol{x}^i)\| \leq (\varepsilon)^{1/3}(1+|f_i|)$

where $f_i \equiv f(\boldsymbol{x}^i)$ and ε is some prespecified tolerance.

(1) and (2) check whether the sequence $\{\boldsymbol{x}^i\}$ is converging while (3) is based on the necessary condition $\nabla \boldsymbol{f}^* = \boldsymbol{0}$. For well scaled problems satisfaction of (1) will imply satisfaction of (2); for poorly scaled problems (2) will force the algorithm to try harder for a better point. In engineering design we might even stop if the cost at the current point is (say) 70% of the starting cost, i.e., $f_i/f_0 \leq 0.7$

SCALING

Consider the problem: Minimize $f(\boldsymbol{x})$ st $\boldsymbol{x} \in \mathbb{R}^n$.

If we scale variables by the linear transformation using the $(n \times n)$ matrix Λ:

$$\underset{n\times n}{\Lambda^{-1}} \cdot \underset{n\times 1}{\boldsymbol{x}} = \underset{n\times 1}{\boldsymbol{y}} \qquad \text{or equivalently} \qquad \underset{n\times 1}{\boldsymbol{x}} = \underset{n\times n}{\Lambda} \cdot \underset{n\times 1}{\boldsymbol{y}}$$

We now have the problem: Minimize $g(\boldsymbol{y}) \equiv f(\Lambda \boldsymbol{y})$.

So

$$\nabla_y \boldsymbol{g} = \Lambda \cdot \nabla_y \boldsymbol{f} \qquad \text{and} \qquad \nabla_y^2 \boldsymbol{g} = \Lambda^T (\nabla_x^2 \boldsymbol{f}) \Lambda$$

Scaling is recommended when the variables x_i have widely differing magnitudes. It is also useful if the matrix $\nabla_y^2 \boldsymbol{g}$ has a better condition number (ratio of the largest to the smallest eigenvalue) than the matrix $\nabla_x^2 \boldsymbol{f}$.

Example:

Consider Minimize $f(\boldsymbol{x}) = 4x_1^2 + x_2^2$ (the optimum solution is at $\boldsymbol{x}^* = [0 \quad 0]^T$)

We have

$$\nabla_{\boldsymbol{x}} \boldsymbol{f} = [8x_1 \quad 2x_2]^T \qquad \text{and} \qquad \nabla_{\boldsymbol{x}}^2 \boldsymbol{f} = \begin{bmatrix} 8 & 0 \\ 0 & 2 \end{bmatrix}.$$

Thus $cond[\nabla_{\boldsymbol{x}}^2 \boldsymbol{f}] = \lambda_{max}/\lambda_{min} = 8/2 = 4$

So if $\boldsymbol{x}^0 = [0.5 \quad 1]^T$ then the steepest descent method generates the sequence

$$\boldsymbol{x}^1 = \begin{bmatrix} 0.5 \\ 1 \end{bmatrix} - \frac{5}{34}\begin{bmatrix} 4 \\ 2 \end{bmatrix} = \begin{bmatrix} -3/34 \\ 24/34 \end{bmatrix}, \qquad \boldsymbol{x}^2 = \begin{bmatrix} -3/34 \\ 24/34 \end{bmatrix} - \alpha \begin{bmatrix} -12/34 \\ 48/34 \end{bmatrix} = \begin{bmatrix} ?? \\ ?? \end{bmatrix},$$

etc., etc.

Now consider the scaling transform $y_1 = 2x_1$, $y_2 = x_2$, so that $g(\boldsymbol{y}) = y_1^2 + y_2^2$

i.e., $\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0.5 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$

Then $\boldsymbol{\nabla}_{\boldsymbol{y}}^{\boldsymbol{2}} \boldsymbol{g} = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$ with $cond[\boldsymbol{\nabla}_{\boldsymbol{y}}^{\boldsymbol{2}} \boldsymbol{g}] = 1.$

At $\boldsymbol{x}^0 = [0.5 \quad 1]^T$, i.e., $\boldsymbol{y} = [1 \quad 1]^T$ the steepest descent method yields

$$\boldsymbol{y}^1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} - \frac{8}{16}\begin{bmatrix} 2 \\ 2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \boldsymbol{y}^*$$

Single step convergence!

A Trust-Region Algorithm

Suppose we start at the initial point $\boldsymbol{x^0}$ with an initial trust region bound Δ_0.

Specify a tolerance ε. Set k=0

<u>Step 1</u>: If $\left\|\nabla \boldsymbol{f}(\boldsymbol{x^k})\right\| \le \varepsilon$ is optimal, stop.

<u>Step 2</u>: Solve the following problem to obtain the trial step $\boldsymbol{d^k}$:

$$\text{Minimize } m_k(\boldsymbol{d}) = f(\boldsymbol{x^k}) + \boldsymbol{d}^{\mathrm{T}}\nabla \boldsymbol{f}(\boldsymbol{x^k}) + (\tfrac{1}{2})\boldsymbol{d}^{\mathrm{T}}\nabla^2 \boldsymbol{f}(\boldsymbol{x^k})\boldsymbol{d}$$

$$\text{st} \quad \|\boldsymbol{d}\| \le \Delta_k$$

<u>Step 3</u>: Compute $\rho_k = \dfrac{f(\boldsymbol{x^k}) - f(\boldsymbol{x^k} + \boldsymbol{d}^k)}{f(\boldsymbol{x^k}) - m_k(\boldsymbol{d}^k)} = \dfrac{\text{actual reduction}}{\text{predicted reduction}}$

If $0 < \rho_k \le \eta$ set $\boldsymbol{x^{k+1}} = \boldsymbol{x^k} + \boldsymbol{d}$ (unsuccessful step) and Δ_{k+1}=0.5Δ_k,

If $\rho_k \ge \mu$ set $\boldsymbol{x^{k+1}} = \boldsymbol{x^k} + \boldsymbol{d}$ and Δ_{k+1}=2Δ_k

If $\eta < \rho_k < \mu$ set $\boldsymbol{x^{k+1}} = \boldsymbol{x^k} + \boldsymbol{d}$ and Δ_{k+1}=Δ_k

Set k=k+1 and return to Step 1.

*Typical values for η and μ might be 0.25 and 0.75.

Note that ρ_k indicates how well the model m_k predicts the reduction in the function f. If ρ_k is small (say <0.25) then the actual reduction is much smaller than the prediction and indicates the model cannot be "trusted" for a large Δ_k. One approach might be to altogether reject $\boldsymbol{d}$ in this case (in the above algorithm we accept $\boldsymbol{d}$ as long as there is any reduction in f). Conversely, if ρ_k is large (say > 0.75) then the model seems to be predicting the reduction well – so we expand our trust region.

A Trust-Region Algorithm (cont'd)

How do we solve the optimization problem in Step 2 to obtain the trial step $\boldsymbol{d}^k$?

Minimize $m_k(\boldsymbol{d}) = f(\boldsymbol{x}^k) + \boldsymbol{d}^T\nabla \boldsymbol{f}(\boldsymbol{x}^k) + (½)\boldsymbol{d}^T\nabla^2 \boldsymbol{f}(\boldsymbol{x}^k)\boldsymbol{d}$

st $\quad \|\boldsymbol{d}\| \leq \Delta_k$

In general, this is not a trivial problem to solve. Note that a point satisfying the necessary conditions for the unconstrained problem would be obtained from

$\nabla \boldsymbol{m}_k(\boldsymbol{d}) = \boldsymbol{0}$, i.e., $\nabla \boldsymbol{f}(\boldsymbol{x}^k) + \nabla^2 \boldsymbol{f}(\boldsymbol{x}^k)\boldsymbol{d} = \boldsymbol{0}$, i.e., the Newton step $\boldsymbol{d} = -[\nabla^2 \boldsymbol{f}(\boldsymbol{x}^k)]^{-1}\nabla \boldsymbol{f}(\boldsymbol{x}^k)$.

So if $\|\boldsymbol{d}\| \leq \Delta_k$ then we just accept this as our trial step. If on the other hand $\|\boldsymbol{d}\| > \Delta_k$, it may be shown that we need to find a λ such that the direction $\boldsymbol{d}$ obtained from

$$\boldsymbol{d} = -[\nabla^2 \boldsymbol{f}(\boldsymbol{x}^k) + \lambda \mathbf{I}]^{-1}\nabla \boldsymbol{f}(\boldsymbol{x}^k) \qquad (\otimes)$$

has $\|\boldsymbol{d}\| = \Delta_k$. This leads to a nonlinear equation in λ which we solve for λ and substituting this into ($\otimes$) then yields the step $\boldsymbol{d}$.

E.g., if $\nabla^2 \boldsymbol{f}(\boldsymbol{x}^k) = \begin{bmatrix} 120 & 0 \\ 0 & 36 \end{bmatrix}$, $\nabla \boldsymbol{f}(\boldsymbol{x}^k) = \begin{bmatrix} 152 \\ 28 \end{bmatrix}$ $\Delta_k = 1$, ($\|\boldsymbol{d}\| = 1.49$ here…) then we solve

$\left\| -\begin{bmatrix} 120+\lambda & 0 \\ 0 & 36+\lambda \end{bmatrix}^{-1} \begin{bmatrix} 152 \\ 28 \end{bmatrix} \right\| = 1$, i.e., $\left(\frac{152}{(120+\lambda)}\right)^2 + \left(\frac{28}{(36+\lambda)}\right)^2 = 1$ to get $\lambda = 42.655$, so that

$\boldsymbol{d} = \begin{bmatrix} -0.9345 \\ -0.356 \end{bmatrix}$ with $\|\boldsymbol{d}\| = 1$

In practice, most algorithms will find only approximate solutions to the subproblem.

VARIABLE-METRIC (OR *QUASI-NEWTON*) METHODS

Recall that for Newton's method, at each iteration we have to

(1) Compute the Hessian matrix H *and* (2) Invert the Hessian

In many situations, $\boldsymbol{H}$ is either not available or very expensive (in terms of CPU time) to compute. Also, inversion of a matrix involves a lot of computational effort (for an $(n\times n)$ matrix the CPU time required is of the order of n^3, which can become very large even for moderate values of n).

The main idea behind quasi-Newton methods is to never *actually* compute the Hessian or its inverse, but rather to gradually construct an approximation to the latter using information gathered as the descent process progresses. That is, obtain second derivative information using only first derivative information. Then if $\boldsymbol{A_i}$ is a matrix which is the current approximation (at iteration i) of the true Hessian inverse $\boldsymbol{H}^{-1}$, we take steps in the direction $\boldsymbol{d^i} = -\boldsymbol{A_i} \cdot \nabla \boldsymbol{f}(\boldsymbol{x^i})$.

MODIFIED NEWTON'S METHOD

Let $\boldsymbol{x^{k+1}} = \boldsymbol{x^k} - \alpha_k \boldsymbol{A_k}[\nabla \boldsymbol{f}(\boldsymbol{x^i})]$, where $\boldsymbol{A_k}$ is a symmetric $(n\times n)$ matrix and as usual we Minimize $f\{\boldsymbol{x^k} - \alpha_k \boldsymbol{A_k}[\nabla \boldsymbol{f}(\boldsymbol{x^i})]\}$ to obtain α_k.

Note that if $\boldsymbol{A_k} = \boldsymbol{H_k}^{-1}$, $\alpha_k=1$ this is Newton's method,

if $\boldsymbol{A_k} = \boldsymbol{I}_n$ this is steepest descent method.

$\boldsymbol{A_k}$ is required to be positive definite. Usually $\boldsymbol{A_k}$ should be “close to” $\boldsymbol{H_k}^{-1}$. In the "classical" modified Newton's method $\boldsymbol{A_k} = [\boldsymbol{H}(\boldsymbol{x^o})]^{-1}$, i.e., Hessian at $\boldsymbol{x^0}$ is used throughout.

Quasi-Newton Methods

Consider the Taylor series approximation for f about the current iterate $\boldsymbol{x}^{\boldsymbol{k}}$:

$$f(\boldsymbol{x}^{\boldsymbol{k}}+\boldsymbol{d}) \approx f_k + \boldsymbol{d}^{\mathrm{T}}\nabla \boldsymbol{f_k} + (\tfrac{1}{2})\boldsymbol{d}^{\mathrm{T}}\boldsymbol{H_k d}$$

For ease of notation, let $f_k = f(\boldsymbol{x}^{\boldsymbol{k}})$, $\nabla\boldsymbol{f_k} = \nabla\boldsymbol{f}(\boldsymbol{x}^{\boldsymbol{k}})$ and $\boldsymbol{H_k} = \nabla^2\boldsymbol{f}\,(\boldsymbol{x}^{\boldsymbol{k}})$…

Recall that the Newton search direction is obtained by minimizing the above to obtain the Newton equations $\boldsymbol{H_k d} = -\nabla\boldsymbol{f_k}$ and solving for ***d.***

In Quasi-Newton methods we either approximate $\boldsymbol{H_k}$ via $\boldsymbol{D_k}$ or, $\boldsymbol{H_k}^{-1}$ via $\boldsymbol{A_k}$.

To see how we get these approximations consider the following approximation for the second derivative of a univariate function:

$$f''(x_k) \approx \frac{f'(x_k) - f'(x_{k-1})}{x_k - x_{k-1}} \quad \Rightarrow \quad f''(x_k)(x_k - x_{k-1}) \approx f'(x_k) - f'(x_{k-1})$$

Extending this logic to n dimensions we might rewrite this as follows:

$$\boldsymbol{H_k}[\boldsymbol{x}^{\boldsymbol{k}} - \boldsymbol{x}^{\boldsymbol{k-1}}] \approx \nabla\boldsymbol{f_k} - \nabla\boldsymbol{f_{k-1}}$$

If we now approximate the matrix $\boldsymbol{H_k}$ by the matrix $\boldsymbol{D_k}$ we obtain the following **Secant Condition** that is used to define the Quasi-Newton approximation:

$$\boxed{\boldsymbol{D_k}[\boldsymbol{x}^{\boldsymbol{k}} - \boldsymbol{x}^{\boldsymbol{k-1}}] = \nabla\boldsymbol{f_k} - \nabla\boldsymbol{f_{k-1}}}$$

This is the basis for all Quasi-Newton methods. Note that in particular, if f is quadratic, so that we can write it as $f(\boldsymbol{x}) = (\tfrac{1}{2})\boldsymbol{x}^{\mathrm{T}}\boldsymbol{H_k x} + \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x}$, we have

$$\boldsymbol{H_k}[\boldsymbol{x}^{\boldsymbol{k}} - \boldsymbol{x}^{\boldsymbol{k-1}}] = (\boldsymbol{H_k x}^{\boldsymbol{k}} + \boldsymbol{c}) - (\boldsymbol{H_k x}^{\boldsymbol{k}-1} + \boldsymbol{c}) = \nabla\boldsymbol{f_k} - \nabla\boldsymbol{f_{k-1}}$$

i.e., $\boldsymbol{H_k}$ satisfies the Secant Condition!

In Quasi-Newton methods we start with some matrix that satisfies the secant condition and update it at each iteration so that it continues to satisfy it.

If we define

$$\boldsymbol{p}^k = \boldsymbol{x}^{k+1} - \boldsymbol{x}^k \quad \text{and} \qquad \boldsymbol{y}^k = \nabla \boldsymbol{f}_{k+1} - \nabla \boldsymbol{f}_k$$

i.e., $\boldsymbol{p}^k$ is the change in $\boldsymbol{x}$ while $\boldsymbol{y}^k$ is the change in the gradient when going from iteration k to iteration k+1, then the Quasi-Newton conditions are $\boldsymbol{D}_k\boldsymbol{p}^{k-1} = \boldsymbol{y}^{k-1}$, or more commonly

$$\boxed{\boldsymbol{D}_{k+1}\boldsymbol{p}^k = \boldsymbol{y}^k}$$

An alternative way of saying the same thing is (recall that $\boldsymbol{A}_k \approx \boldsymbol{H}_k^{-1}$)

$$\boxed{\boldsymbol{p}^k = \boldsymbol{A}_{k+1}\,\boldsymbol{y}^k}$$

<u>Note</u>: If we use a line search and get $\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \alpha \boldsymbol{d}^k$ then $\boldsymbol{p}^k = \alpha \boldsymbol{d}^k$!

A simple example of an update that satisfies the secant condition is the following (symmetric Rank-1 update):

$$\boldsymbol{D}_{k+1} = \boldsymbol{D}_k + \frac{(\boldsymbol{y}^k - \boldsymbol{D}_k\boldsymbol{p}^k)(\boldsymbol{y}^k - \boldsymbol{D}_k\boldsymbol{p}^k)^T}{(\boldsymbol{y}^k - \boldsymbol{D}_k\boldsymbol{p}^k)^T\,\boldsymbol{p}^k}$$

Note that

$$\boldsymbol{D}_{k+1}\boldsymbol{p}^k = \boldsymbol{D}_k\boldsymbol{p}^k + \frac{(\boldsymbol{y}^k - \boldsymbol{D}_k\boldsymbol{p}^k)(\boldsymbol{y}^k - \boldsymbol{D}_k\boldsymbol{p}^k)^T}{(\boldsymbol{y}^k - \boldsymbol{D}_k\boldsymbol{p}^k)^T\,\boldsymbol{p}^k}\,\boldsymbol{p}^k$$

$$= \boldsymbol{D}_k\boldsymbol{p}^k + \frac{(\boldsymbol{y}^k - \boldsymbol{D}_k\boldsymbol{p}^k)\left((\boldsymbol{y}^k - \boldsymbol{D}_k\boldsymbol{p}^k)^T\,\boldsymbol{p}^k\right)}{(\boldsymbol{y}^k - \boldsymbol{D}_k\boldsymbol{p}^k)^T\,\boldsymbol{p}^k} = \boldsymbol{D}_k\boldsymbol{p}^k + (\boldsymbol{y}^k - \boldsymbol{D}_k\boldsymbol{p}^k) = \boldsymbol{y}^k$$

This update maintains symmetry but not necessarily positive definiteness.

Better methods are based on the so-called Broyden class of updates that preserve symmetry as well as positive definiteness as long as $(\boldsymbol{p}^k)^{\mathrm{T}}\boldsymbol{y}^k>0$. Positive definiteness is important because presumably the Hessian is positive definite (and certainly, positive semidefinite) at the optimum.

In this class of updates we can use the following (rather messy!) formulae for updating either the approximation to the Hessian ($\boldsymbol{D_k}$) or the approximation to its inverse ($\boldsymbol{A_k}$), depending on which one we need. These are defined as below:

For the approximation to $\boldsymbol{H_k}$ we have

$$\boldsymbol{D_{k+1}} = \boldsymbol{D_k} + \frac{\boldsymbol{y^k}(\boldsymbol{y^k})^T}{(\boldsymbol{y^k})^T\boldsymbol{p^k}} - \frac{\boldsymbol{D_k p^k}(\boldsymbol{D_k p^k})^T}{(\boldsymbol{p^k})^T\boldsymbol{D_k p^k}} + \phi\left((\boldsymbol{p^k})^T\boldsymbol{D_k p^k}\right)\boldsymbol{u^k}(\boldsymbol{u^k})^T, \qquad (\perp)$$

where ϕ is a scalar and $\boldsymbol{u^k} = \dfrac{\boldsymbol{y^k}}{(\boldsymbol{y^k})^T\boldsymbol{p^k}} - \dfrac{\boldsymbol{D_k p^k}}{(\boldsymbol{p^k})^T\boldsymbol{D_k p^k}}$

<u>NOTE</u>: It is easily verified that $\boldsymbol{D_{k+1}p^k} = \boldsymbol{y^k}$

For the approximation to the $\boldsymbol{H_k^{-1}}$ we have

$$\boldsymbol{A_{k+1}} = \boldsymbol{A_k} + \frac{\boldsymbol{p^k}(\boldsymbol{p^k})^T}{(\boldsymbol{p^k})^T\boldsymbol{y^k}} - \frac{\boldsymbol{A_k y^k}(\boldsymbol{A_k y^k})^T}{(\boldsymbol{y^k})^T\boldsymbol{A_k y^k}} + \varepsilon\left((\boldsymbol{y^k})^T\boldsymbol{A_k y^k}\right)\boldsymbol{v^k}(\boldsymbol{v^k})^T \qquad (\perp\perp)$$

where ε is a scalar and $\boldsymbol{v^k} = \dfrac{\boldsymbol{p^k}}{(\boldsymbol{p^k})^T\boldsymbol{y^k}} - \dfrac{\boldsymbol{A_k y^k}}{(\boldsymbol{y^k})^T\boldsymbol{A_k y^k}}$

<u>NOTE</u>: It is easily verified that $\boldsymbol{A_{k+1}y^k} = \boldsymbol{p^k}$

Two of the best Quasi-Newton algorithms use the above schemes for updating the approximations to the Hessian (or its inverse) that is used in place of $\boldsymbol{H_k}$ (or $\boldsymbol{H_k}^{-1}$) when trying to solve the Newton equations $\boldsymbol{H_k}\,\boldsymbol{d} = -\nabla \boldsymbol{f_k}$ for the search direction $\boldsymbol{d}$, (or computing $\boldsymbol{d} = -(\boldsymbol{H_k}^{-1})\nabla \boldsymbol{f_k}$).

- In the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm we approximate $\boldsymbol{H_k}$ via $\boldsymbol{D_k}$ and solve the system $\boldsymbol{H_k d} = -\nabla \boldsymbol{f_k} \;\approx\; \boldsymbol{D_k d} = -\nabla \boldsymbol{f_k}$ for the vector $\boldsymbol{d}$

  To update $\boldsymbol{D_k}$ we use formula $(\perp)$ with $\phi=0$. It may be shown that this is equivalent to setting $\varepsilon=1$ in $(\perp\perp)$…

- In the Davidon-Fletcher-Powell (DFP) algorithm we approximate $\boldsymbol{H_k}^{-1}$ via $\boldsymbol{A_k}$ and compute $\boldsymbol{d} = -\boldsymbol{H_k}^{-1}\nabla \boldsymbol{f_k} \approx -\boldsymbol{A_k}\nabla \boldsymbol{f_k}$.

  To update $\boldsymbol{A_k}$ we use formula $(\perp\perp)$ with $\varepsilon=0$. It may be shown that this is equivalent to setting $\phi=1$ in $(\perp)$…

In fact, we may write

$$\boldsymbol{D_{k+1}} = (1-\phi)\boldsymbol{D_{k+1}^{BFGS}} + \phi \boldsymbol{D_{k+1}^{DFP}}$$

or

$$\boldsymbol{A_{k+1}} = \varepsilon \boldsymbol{A_{k+1}^{BFGS}} + (1-\varepsilon)\boldsymbol{A_{k+1}^{DFP}}$$

The above two algorithms from the Broyden class maintain both symmetry as well as positive semidefiniteness - BFGS and DFP are considered among the best methods for unconstrained optimization…

BROYDEN-FLETCHER-GOLDFARB-SHANNO METHOD

At iteration i the method is as follows:

STEP 1: Let $\boldsymbol{D_i} \cong \boldsymbol{H_i}$ (i.e., an approximation to the true Hessian at the current point $\boldsymbol{x^i}$ (at $\boldsymbol{x^0}$ choose $\boldsymbol{D_1} = \boldsymbol{I_n}$ or any symmetric positive definite matrix).

STEP 2: If $\boldsymbol{x^i}$ is optimal stop; otherwise obtain the search direction $\boldsymbol{d^i}$ by solving

$$\boldsymbol{D_i d} = -\nabla \boldsymbol{f_i}$$

STEP 3: Minimize f in the direction $\boldsymbol{d^i}$, i.e.,

$$\text{Minimize}_\alpha \{f(\boldsymbol{x^i}+\alpha \boldsymbol{d^i})\} \text{ to find } \alpha \text{ and hence } \boldsymbol{x^{i+1}}.$$

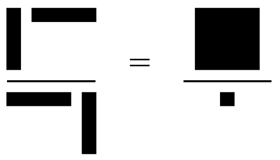
STEP 4: Define

$$\boldsymbol{p^i} = \boldsymbol{x^{i+1}} - \boldsymbol{x^i} = \Delta \boldsymbol{x^i} \text{ (change in } \boldsymbol{x}\text{)}$$

$$\boldsymbol{y^i} = \nabla \boldsymbol{f_{i+1}} - \nabla \boldsymbol{f_i} = \Delta \boldsymbol{g^i} \text{ (change in gradient)}$$

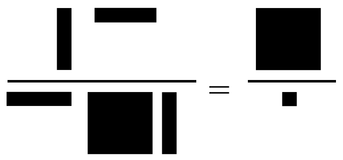
STEP 5: Compute

$$\boldsymbol{B_i} = \frac{\boldsymbol{y^i}(\boldsymbol{y^i})^T}{(\boldsymbol{y^i})^T \boldsymbol{p^i}}$$



$$= \frac{Matrix}{Scalar}$$

$$\boldsymbol{C_i} = -\frac{\boldsymbol{D_i p^i}(\boldsymbol{D_i p^i})^T}{(\boldsymbol{p^i})^T \boldsymbol{D_i p^i}}$$



$$= \frac{Matrix}{Scalar}$$

STEP 6: UPDATE the approximation to $\boldsymbol{H}$ via

$$\boldsymbol{D_{i+1}} = \boldsymbol{D_i} + \boldsymbol{B_i} + \boldsymbol{C_i}$$

Set i=i+1, and return to Step 2

THE DAVIDON-FLETCHER-POWELL METHOD

<u>STEP 1</u>: Let $\boldsymbol{A_i} \cong \boldsymbol{H_i} = [\,\boldsymbol{\nabla f(x^i)}]^{-1}$ (an approximation to the true Hessian at the current point $\boldsymbol{x^i}$ (at $\boldsymbol{x^0}$ choose $\boldsymbol{A_0 = I_n}$ or any symmetric positive definite matrix).

<u>STEP 2</u>: If $\boldsymbol{x^i}$ is optimal stop; otherwise obtain the search direction $\boldsymbol{d^i}$ by solving

$$\boldsymbol{d^i} = -\boldsymbol{A_i}\nabla \boldsymbol{f_i}$$

<u>STEP 3</u>: Minimize f in the direction $\boldsymbol{d^i}$, i.e.,

$$\text{Minimize}_{\alpha}\ \{f(\boldsymbol{x^i}+\alpha\boldsymbol{d^i})\} \text{ to find } \alpha \text{ and hence } \boldsymbol{x^{i+1}}.$$

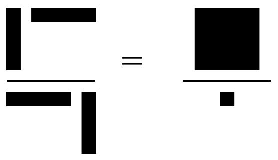
<u>STEP 4</u>: Define

$$\boldsymbol{p^i = x^{i+1} - x^i} = \Delta \boldsymbol{x^i} \text{ (change in } \boldsymbol{x}\text{)}$$

$$\boldsymbol{y^i} = \nabla \boldsymbol{f_{i+1}} - \nabla \boldsymbol{f_i} = \Delta \boldsymbol{g^i} \text{ (change in gradient)}$$

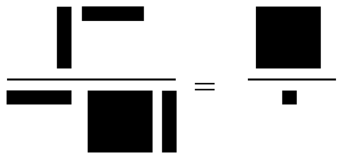
<u>STEP 5</u>: Compute

$$\boldsymbol{B_i} = \frac{\boldsymbol{p^i}(\boldsymbol{p^i})^{\mathrm{T}}}{(\boldsymbol{p^i})^{\mathrm{T}}\boldsymbol{y^i}}$$



$$= \frac{\mathit{Matrix}}{\mathit{Scalar}}$$

$$\boldsymbol{C_i} = -\frac{\boldsymbol{A_i y^i}(\boldsymbol{A_i y^i})^{\mathrm{T}}}{(\boldsymbol{y^i})^{\mathrm{T}}\boldsymbol{A_i y^i}}$$



$$= \frac{\mathit{Matrix}}{\mathit{Scalar}}$$

<u>STEP 6</u>: UPDATE the approximation to $\boldsymbol{H}^{-1}$ via

$$\boldsymbol{A_{i+1} = A_i + B_i + C_i}$$

Set i=i+1, and return to Step 2

If the gradient vector is readily available, this is generally regarded as one of the best algorithms available).

AN EXAMPLE: Minimize $f(\boldsymbol{x}) = (x_2 - x_1)^2 + (1-x_1)^2$

Thus $$\nabla f(x) = \begin{bmatrix} -2(x_2 - x_1) - 2(1 - x_1) \\ 2(x_2 - x_1) \end{bmatrix} = \begin{bmatrix} 4x_1 - 2x_2 - 2 \\ 2x_2 - 2x_1 \end{bmatrix}$$

ITERATION 1

1) Start with $\boldsymbol{x^0} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ and $\boldsymbol{A_0} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cong \boldsymbol{H}^{-1}(\boldsymbol{x^0})$

2) $\boldsymbol{d^0} = -\boldsymbol{A_0}\boldsymbol{\nabla f}(\boldsymbol{x_0}) = -\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} -2 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$

3) $\text{Min}_\alpha f(\boldsymbol{x^0}+\alpha\boldsymbol{d^0}) = \text{Min}_\alpha f\{(0+2\alpha),(0+0\alpha)\} = \text{Min}_\alpha\, 4\alpha^2+(1-2\alpha)^2 = 8\alpha^2+1-4\alpha$

This yields $\alpha_0=0.25$, so that $\boldsymbol{x^1} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} + 0.25\begin{bmatrix} 2 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0 \end{bmatrix}$, with $f(\boldsymbol{x^1})=0.5$

4) $\boldsymbol{p^0} = \Delta\boldsymbol{x^0} = \begin{bmatrix} 0.5 \\ 0 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0 \end{bmatrix}$

$\boldsymbol{y^0} = \Delta\boldsymbol{g^0} = \boldsymbol{\nabla f}(\boldsymbol{x^1}) - \boldsymbol{\nabla f}(\boldsymbol{x^0}) = \begin{bmatrix} 0 \\ -1 \end{bmatrix} - \begin{bmatrix} -2 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ -1 \end{bmatrix}$

5) $\boldsymbol{B_0} = \dfrac{\boldsymbol{p^0}[\boldsymbol{p^0}]^{\mathrm{T}}}{[\boldsymbol{p^0}]^{\mathrm{T}}\boldsymbol{y^0}} = \dfrac{\begin{bmatrix} 0.5 \\ 0 \end{bmatrix}[0.5 \;\; 0]}{[0.5 \;\; 0]\begin{bmatrix} 2 \\ -1 \end{bmatrix}} = \dfrac{\begin{bmatrix} 0.25 & 0 \\ 0 & 0 \end{bmatrix}}{1} = \begin{bmatrix} 0.25 & 0 \\ 0 & 0 \end{bmatrix}$

$\boldsymbol{C_0} = -\dfrac{[\boldsymbol{A_0}\boldsymbol{y^0}]\cdot[\boldsymbol{A_0}\boldsymbol{y^0}]^{\mathrm{T}}}{[\boldsymbol{y^0}]^{\mathrm{T}}\boldsymbol{A_0}\boldsymbol{y^0}};\ \boldsymbol{A_0}\boldsymbol{y^0} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} 2 \\ -1 \end{bmatrix} = \begin{bmatrix} 2 \\ -1 \end{bmatrix}$

$\therefore \boldsymbol{C_0} = -\dfrac{\begin{bmatrix} 2 \\ -1 \end{bmatrix}[2 \;\; -1]}{[2 \;\; -1]\begin{bmatrix} 2 \\ -1 \end{bmatrix}} = \begin{bmatrix} -0.8 & 0.4 \\ 0.4 & -0.2 \end{bmatrix}$

6) $\boldsymbol{A_1} = \boldsymbol{A_0} + \boldsymbol{B_0} + \boldsymbol{C_0}$

$= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + \begin{bmatrix} 0.25 & 0 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} -0.8 & 0.4 \\ 0.4 & -0.2 \end{bmatrix} = \begin{bmatrix} 0.45 & 0.4 \\ 0.4 & 0.8 \end{bmatrix}$

ITERATION 2

2) $\boldsymbol{d^1} = -\boldsymbol{A_1}\boldsymbol{\nabla f}(\boldsymbol{x_0}) = -\begin{bmatrix} 0.45 & 0.4 \\ 0.4 & 0.8 \end{bmatrix}\begin{bmatrix} 0 \\ -1 \end{bmatrix} = \begin{bmatrix} 0.4 \\ 0.8 \end{bmatrix}$

3) $\text{Min}_\alpha f(\boldsymbol{x^1}+\alpha\boldsymbol{d^1}) = \text{Min}_\alpha f\{(0.5+0.4\alpha),(0+0.8\alpha)\}$

$=\text{Min}_\alpha\,(0.8\alpha - 0.5 - 0.4\alpha)^2 + (1 - 0.5 - 0.4\alpha)^2$

This yields $\alpha_1=1.25$, and the optimal solution ($\boldsymbol{\nabla f}(\boldsymbol{x}^2)=0$) with

$\boldsymbol{x^2} = \begin{bmatrix} 0.5 \\ 0 \end{bmatrix} + 1.25\begin{bmatrix} 0.4 \\ 0.8 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$, with $f(\boldsymbol{x}^2)=0$

CONJUGATE DIRECTIONS

Better descent methods usually use *conjugate search directions.*

<u>DEFINITION</u>: Two directions $\hat{\boldsymbol{d}}$ and $\bar{\boldsymbol{d}}$ are conjugate with respect to the symmetric positive definite matrix $\boldsymbol{Q}$ if,

$$(\hat{\boldsymbol{d}})^{\mathrm{T}}\boldsymbol{Q}\bar{\boldsymbol{d}} = 0$$

Conjugate directions are also linearly independent.

NOTE: If $\boldsymbol{Q}=\boldsymbol{I}$ then $(\hat{\boldsymbol{d}})^{\mathrm{T}}\boldsymbol{Q}\bar{\boldsymbol{d}} = (\hat{\boldsymbol{d}})^{\mathrm{T}}\bar{\boldsymbol{d}} = 0$, i.e., the conditions for $\hat{\boldsymbol{d}}$ and $\bar{\boldsymbol{d}}$ to be conjugate is that they should be orthogonal;

<u>Geometric Interpretation</u>

Suppose $f(\boldsymbol{x}) = ½\, \boldsymbol{x}^{\mathrm{T}}\boldsymbol{Q}x + \boldsymbol{b}^{\mathrm{T}}\boldsymbol{x}$

Thus $\boldsymbol{Q}$ is the Hessian of $f(\boldsymbol{x})$ and of order $(n\times n)$

<u>Example</u>: n=2 and say $\boldsymbol{Q}$ is positive definite (and of course, symmetric)

e.g., $f(\boldsymbol{x}) = ax_1^2 + bx_2^2 + cx_1x_2 = \frac{1}{2}\begin{bmatrix} x_1 & x_2 \end{bmatrix}\begin{bmatrix} 2a & c \\ c & 2b \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$.

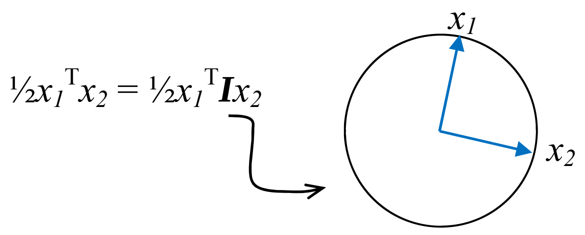
Here the contours of f are ellipses centered at (0,0). In general, the contours of a quadratic function are n-dimensional ellipsoids with axes in the directions of the n mutually orthogonal eigenvectors of the Hessian. The axis corresponding to the ith eigenvector is proportional to $1/\lambda_i$.

Verify that the eigenvalues are

$\lambda_1 = (a+b) + \sqrt{(a-b)^2 + c^2}$ and $\lambda_2 = (a+b) - \sqrt{(a-b)^2 + c^2}$

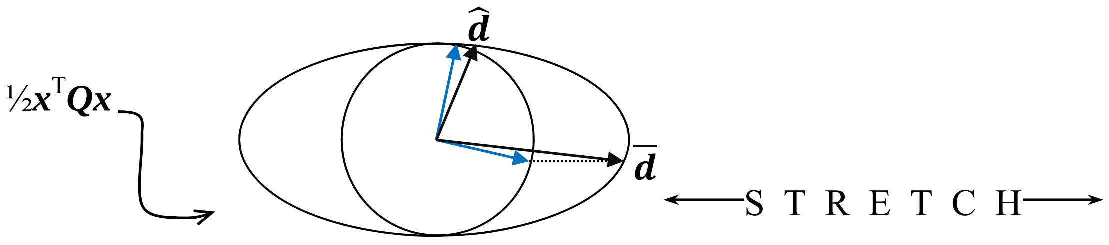
E.g., if a=b=0.5, and c=0, then λ_1=1, λ_2=1, so we get a circle centered at (0,0).

Now, if we start with


$\frac{1}{2}x_1^T x_2 = \frac{1}{2}x_1^T \boldsymbol{I} x_2$
x_1
x_2

x_1 and x_2 are orthogonal radii vectors

and we "stretch" this circle along one diameter as shown below

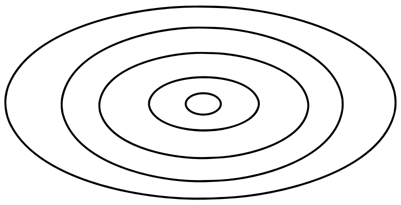


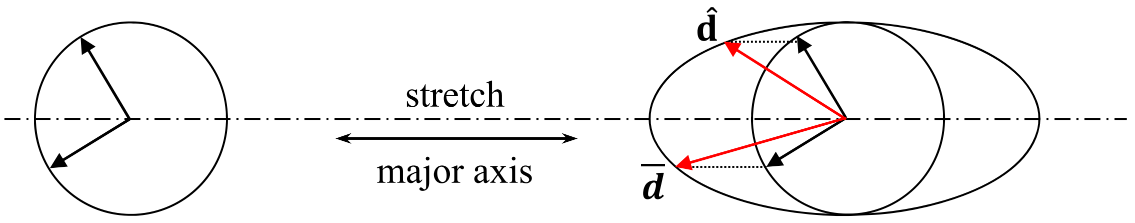
The two orthogonal radii vectors become conjugate with respect to some matrix $\boldsymbol{Q}$.

Question. How do conjugate directions relate to minimization?

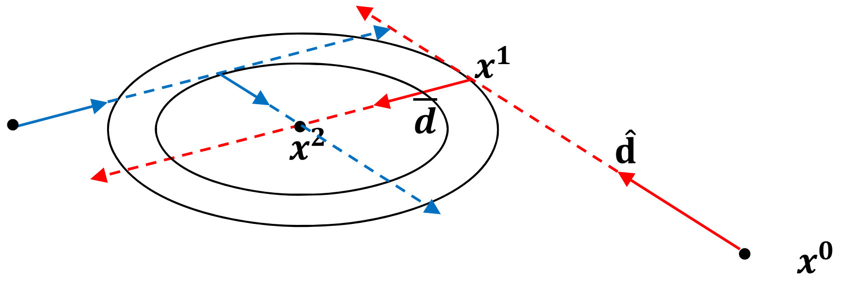
<u>THEOREM</u>: The minimum of a convex quadratic function of n variables $f(x_1, x_2, \ldots, x_n) = \frac{1}{2}\mathbf{x}^T\boldsymbol{H}x + \mathbf{c}^T\mathbf{x} + k$, (where $\boldsymbol{H}$ is the positive definite Hessian) can be found in **at most** n steps from **any** point via a series of minimizations along each of a set of $\boldsymbol{H}$-conjugate directions.

Example: n=2; contours of $f(\mathbf{x})$ are as shown below:





Directions $\hat{d}$ and $\bar{d}$ are H-conjugate



(1) Start at x^0 and search along $\hat{d}$ and minimize to obtain x^1

(2) Search along $\bar{d}$ and minimize to get x^2 (the optimum point) **(2 steps!)**

(Same thing in second case (but first along $\bar{d}$ and then along $\hat{d}$)

Normally

- A set of H-conjugate directions $d^1, d^2, \ldots, d^n$ is not usually known beforehand, but generated by some (often complicated) scheme which depends upon available information (Hessian, gradient vectors, function values at previous points in the search, etc.).
- Conjugate direction methods are "designed" for convex, quadratic functions
- Best for large problems: storage needs are low & “work” per iteration is low
- Unless f is actually quadratic the Hessian matrix is not constant, but changes as the search proceeds -- thus the "conjugate" directions computed are seldom actually H-conjugate for any H. Despite this, even for non-quadratic functions, the so-called "conjugate" directions generated are good search directions.

CONJUGATE GRADIENT METHOD

(For solving $\boldsymbol{Ax}=\boldsymbol{b}$ when $\boldsymbol{A}$ is symmetric & positive definite)

First note that solving $\boldsymbol{Ax}=\boldsymbol{b}$, is equivalent to minimizing $f(\boldsymbol{x}) = ½\boldsymbol{x}^T\boldsymbol{Ax} - \boldsymbol{b}^T\boldsymbol{x}$ (since the necessary conditions $-\nabla \boldsymbol{f}(\boldsymbol{x}^0) = \boldsymbol{Ax} - \boldsymbol{b} = \boldsymbol{0}$ are sufficient).

Now consider a set of vectors $\boldsymbol{p}^0, \boldsymbol{p}^1, \ldots, \boldsymbol{p}^m$ that are conjugate w.r.t. $\boldsymbol{A}$, i.e., $(\boldsymbol{p}^i)^T\boldsymbol{Ap}^j=\boldsymbol{0}$ if $i \neq j$, and suppose we define $\boldsymbol{y}= \sum_{i=0}^{m} \alpha_i \boldsymbol{p}^i$.

Then $$\begin{aligned} f(\boldsymbol{y}) &= ½(\textstyle\sum_{i=0}^{m} \alpha_i \boldsymbol{p}^i)^T\boldsymbol{A}(\sum_{j=0}^{m} \alpha_j \boldsymbol{p}^j) - \boldsymbol{b}^T(\sum_{i=0}^{m} \alpha_i \boldsymbol{p}^i) \\ &= ½(\textstyle\sum_{i=0}^{m} \sum_{j=0}^{m} \alpha_i\alpha_j (\boldsymbol{p}^i)^T\boldsymbol{Ap}^j) - (\sum_{i=0}^{m} \alpha_i \boldsymbol{b}^T\boldsymbol{p}^i) \\ &= ½(\textstyle\sum_{i=0}^{m} \alpha_i^2 (\boldsymbol{p}^i)^T\boldsymbol{Ap}^i) - (\sum_{i=0}^{m} \alpha_i \boldsymbol{b}^T\boldsymbol{p}^i) \qquad \text{(from conjugacy)} \\ &= (\textstyle\sum_{i=0}^{m} \left(\frac{1}{2}\alpha_i^2(\boldsymbol{p}^i)^T\boldsymbol{Ap}^i - \alpha_i\boldsymbol{b}^T\boldsymbol{p}^i\right) \end{aligned}$$

Minimizing this is easy since each term is separable and setting the derivative w.r.t. each α_i to 0 we get $\alpha_i(\boldsymbol{p}^i)^T\boldsymbol{Ap}^i - \boldsymbol{b}^T\boldsymbol{p}^i = 0$, i.e. $\alpha_i = \frac{\boldsymbol{b}^T\boldsymbol{p}^i}{(\boldsymbol{p}^i)^T\boldsymbol{Ap}^i}$

Since we don't know the conjugate vectors $\boldsymbol{p}^i$ in advance the conjugate gradient method <u>determines these iteratively, along with their coefficients</u> α_i.

Suppose we define $\boldsymbol{r}^i = \boldsymbol{b} - \boldsymbol{Ax}^i$ as the residual vector at iteration i and a scalar quantity β_i that will be used to determine $\boldsymbol{p}^i$.

Then the conjugate gradient algorithm may be defined as follows:

1. Start with $\boldsymbol{x}^0=\boldsymbol{0}$ so that $\boldsymbol{r}^0=\boldsymbol{b}$. Let $\beta_0=0$, $\boldsymbol{p}^0=\boldsymbol{b}$ and $\alpha_0 = \frac{\boldsymbol{b}^{\mathrm{T}}\boldsymbol{p}^0}{(\boldsymbol{p}^0)^{\mathrm{T}}\boldsymbol{A}\boldsymbol{p}^0}$
2. Find $\boldsymbol{x}^1=\boldsymbol{x}^0+\alpha_0\,\boldsymbol{p}^0$ and compute $\boldsymbol{r}^1=\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}^1$. Then set iteration counter $i=1$
3. STOP if $\|\boldsymbol{r}^i\| < \varepsilon$ (some suitable tolerance)
4. Set $\beta_i=\frac{(\boldsymbol{r}^i)^{\mathrm{T}}\boldsymbol{r}^i}{(\boldsymbol{r}^{i-1})^{\mathrm{T}}\boldsymbol{r}^{i-1}}$
5. Set $\boldsymbol{p}^i = \boldsymbol{r}^i + \beta_i\,\boldsymbol{p}^{i-1}$
6. Set $\alpha_i = \frac{(\boldsymbol{r}^i)^{\mathrm{T}}\boldsymbol{r}^i}{(\boldsymbol{p}^i)^{\mathrm{T}}\boldsymbol{A}\boldsymbol{p}^i}$
7. Set $\boldsymbol{x}^{i+1}=\boldsymbol{x}^i+\alpha_i\,\boldsymbol{p}^i$
8. Set $\boldsymbol{r}^{i+1}=\boldsymbol{r}^i-\alpha_i\,\boldsymbol{A}\boldsymbol{p}^i$ and return to Step 3.

Notes:

1) If we wished to, we could also compute (using the original formulae...)

   $\boldsymbol{r}^{i+1}=\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}^{i+1}$, because $\boldsymbol{r}^i-\alpha_i\,\boldsymbol{A}\boldsymbol{p}^i = (\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}^i)-\alpha_i\,\boldsymbol{A}\boldsymbol{p}^i = \boldsymbol{b}-\boldsymbol{A}(\boldsymbol{x}^i+\alpha_i\boldsymbol{p}^i)=\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}^{i+1}$

   $\alpha_i = \frac{\boldsymbol{b}^{\mathrm{T}}\boldsymbol{p}^i}{(\boldsymbol{p}^i)^{\mathrm{T}}\boldsymbol{A}\boldsymbol{p}^i}$, because $\boldsymbol{b}^{\mathrm{T}}\boldsymbol{p}^i = (\boldsymbol{r}^i)^{\mathrm{T}}\boldsymbol{r}^i$

   We use the formulae in (6) and (8) only for computational efficiency.

2) $\boldsymbol{x}^{i+1}=\boldsymbol{x}^i+\alpha_i\,\boldsymbol{p}^i+\alpha_{i-1}\,\boldsymbol{p}^{i-1}+\alpha_{i-2}\,\boldsymbol{p}^{i-2}\ldots+\alpha_0\,\boldsymbol{p}^0$

   So these approximate solutions at each step have the same form as the vector $\boldsymbol{y}$ on the previous page.

THEOREM: For $i>j$, the vectors $\{\boldsymbol{p}^i\}$ and $\{\boldsymbol{r}^i\}$ computed in the above algorithm satisfy

$(\boldsymbol{r}^i)^{\mathrm{T}}\boldsymbol{r}^j=0$	(residuals are orthogonal to each other)
$(\boldsymbol{r}^i)^{\mathrm{T}}\boldsymbol{p}^j=0,$	(residuals are orthogonal to the search directions)
$(\boldsymbol{p}^i)^{\mathrm{T}}\boldsymbol{A}\boldsymbol{p}^j=0$	(search directions are conjugate w.r.t. $\boldsymbol{A}$)

THE FLETCHER REEVES ALGORITHM

(also called the nonlinear CONJUGATE-GRADIENT method)

This is an algorithm that uses conjugate search directions:

Here $\boldsymbol{r}^{\boldsymbol{i}} \equiv \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{\boldsymbol{i}} = -\nabla \boldsymbol{f}(\boldsymbol{x}^{\boldsymbol{0}})$ if the function is quadratic ($\frac{1}{2}\, \boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}^{\mathrm{T}}\boldsymbol{x}$)

Start with $\boldsymbol{d}^{\boldsymbol{0}} = -\nabla \boldsymbol{f}(\boldsymbol{x}^{\boldsymbol{0}})$ (steepest descent dir.),

Minimize along $\boldsymbol{d}^{\boldsymbol{0}}$, i.e.,

Minimize $f(\alpha) = \{f(\boldsymbol{x}^{\boldsymbol{0}} + \alpha\, \boldsymbol{d}^{\boldsymbol{0}})\}$ to obtain α_0 and hence $\boldsymbol{x}^{\boldsymbol{1}} = \boldsymbol{x}^{\boldsymbol{0}} + \alpha_0 \boldsymbol{d}^{\boldsymbol{0}}$

The next search direction is computed by

$$\boldsymbol{d}^{\mathbf{1}} = \quad -\nabla \boldsymbol{f}(\boldsymbol{x}^{\mathbf{1}}) \qquad + \qquad \frac{\left\|\nabla \boldsymbol{f}(\boldsymbol{x}^{\mathbf{1}})\right\|^2}{\|\nabla \boldsymbol{f}(\boldsymbol{x}^{\mathbf{0}})\|^2} \cdot \boldsymbol{d}^{\mathbf{0}}$$

(steepest descent dir.) *plus* (a multiple of the previous search dir.)

Here $\|\nabla \boldsymbol{f}(\boldsymbol{x})\|$ denotes the length of the vector $\nabla \boldsymbol{f}(\boldsymbol{x})$,

i.e., $\|\nabla \boldsymbol{f}(\boldsymbol{x})\| = \sqrt{\left(\frac{\partial f}{\partial x_1}\right)^2 + \left(\frac{\partial f}{\partial x_2}\right)^2 + \ldots + \left(\frac{\partial f}{\partial x_n}\right)^2}$-

The procedure could be generalized (at iteration *i*),

$$\boldsymbol{d}^{\boldsymbol{i}+\mathbf{1}} = -\nabla \boldsymbol{f}\left(\boldsymbol{x}^{\boldsymbol{i}+\mathbf{1}}\right) + \frac{\left\|\nabla \boldsymbol{f}\left(\boldsymbol{x}^{\boldsymbol{i}+\mathbf{1}}\right)\right\|^2}{\|\nabla \boldsymbol{f}(\boldsymbol{x}^{\boldsymbol{i}})\|^2} \cdot \boldsymbol{d}^{\boldsymbol{i}}$$

How does this relate to conjugate directions?

Suppose that $f(x)$ is quadratic with Hessian $\boldsymbol{H}$.

Then for the vectors $\boldsymbol{d}^{0}, \boldsymbol{d}^{1},\ldots, \boldsymbol{d}^{n-1}$

$$[\boldsymbol{d}^{i}]^{\mathrm{T}}\, \boldsymbol{H}\, \boldsymbol{d}^{j} = 0 \qquad \text{for } i\neq j,\ i,j \in\{0,1,\ldots,n\text{-}1\},$$

i.e., the vectors are all $\boldsymbol{H}$-conjugate. Note that the directions are not specified beforehand; rather they are determined sequentially at each step.

<u>Some Features of the method</u>

(1) $\nabla\boldsymbol{f}(\boldsymbol{x}^{i})$ is always non-zero and linearly independent of previous directions, unless the solution is obtained in less than n steps (in which case of course, we are done!)

(2) The formula for finding $\boldsymbol{d}^{i+1}$ is the simplest of all conjugate directions methods.

(3) The path taken usually shows *uniformly* good progress, and very often avoids "zig-zagging."

(4) The minimization procedure along direction $\boldsymbol{d}^{k}$ has a closed form solution with

$\alpha_k \equiv$ step size in direction $\boldsymbol{d}^{k}$ at iteration k,

$$= \frac{-[\nabla \boldsymbol{f}(\boldsymbol{x}^{k})]^{\mathrm{T}}\boldsymbol{d}^{k}}{[\boldsymbol{d}^{k}]^{\mathrm{T}}\boldsymbol{H}\boldsymbol{d}^{k}} \qquad \textit{as long as } f(\boldsymbol{x}) \textit{ is quadratic}$$

(5) It is well-suited to large problems because there are no matrix operations and storage requirements are not very high (we only need the last two gradients).

(6) If a “bad” direction is generated at some stage, there is a tendency for the next direction to also be bad; so the algorithm will tend to crawl when this happened. To avoid this, there is always some “restart” strategy used when this method is implemented (e.g., restart with the steepest descent dir. if we seem to be stuck).

<u>AN EXAMPLE</u> Consider the previous example solved by the steepest descent method: Min $f(x) = x_1^2 + 25x_2^2$

Recall that $\boldsymbol{\nabla f}(\boldsymbol{x}) = \begin{bmatrix} 2x_1 \\ 50x_2 \end{bmatrix}$ and $\boldsymbol{H} = \begin{bmatrix} 2 & 0 \\ 0 & 50 \end{bmatrix}$

Start with $[2 \;\; 2]^T$

<u>ITERATION 1</u>

$$\boldsymbol{x^0} = \begin{bmatrix} 2 \\ 2 \end{bmatrix} \qquad \boldsymbol{d^0} = -\boldsymbol{\nabla f}(\boldsymbol{x}) = \begin{bmatrix} -4 \\ -100 \end{bmatrix}; \quad \boldsymbol{x^1} = \boldsymbol{x^0} + \alpha_0 \boldsymbol{d^0} = \begin{bmatrix} 2 - 4\alpha_0 \\ 2 - 100\alpha_0 \end{bmatrix}$$

To obtain α_0 we minimize $f(2\text{-}4\alpha_0, 2\text{-}100\alpha_0)$. Thus,

$$\alpha_0 = \frac{-[\boldsymbol{\nabla f}(\boldsymbol{x^0})]^T]\boldsymbol{d^0}}{[\boldsymbol{d^0}]^T\boldsymbol{H}\boldsymbol{d^0}} = \frac{-[4 \;\; 100]\begin{bmatrix} -4 \\ -100 \end{bmatrix}}{[-4 \;\; -100]\begin{bmatrix} 2 & 0 \\ 0 & 50 \end{bmatrix}\begin{bmatrix} -4 \\ -100 \end{bmatrix}} = 0.02003$$

$$\therefore \; \boldsymbol{x^1} = \begin{bmatrix} 2 \\ 2 \end{bmatrix} + 0.02003 \begin{bmatrix} -4 \\ -100 \end{bmatrix} = \begin{bmatrix} 1.91988 \\ -0.00307 \end{bmatrix}$$

To summarize: $\boldsymbol{x^0} = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$; $\boldsymbol{d^0} = \begin{bmatrix} -4 \\ -100 \end{bmatrix}$; $\alpha_0 = 0.02003$

<u>ITERATION 2</u>

$$\boldsymbol{x^1} = \begin{bmatrix} 1.91988 \\ -0.00307 \end{bmatrix} \qquad \boldsymbol{d^1} = -\boldsymbol{\nabla f}(\boldsymbol{x^1}) + \frac{\|\nabla f(x^1)\|^2}{\|\nabla f(x^0)\|^2} \cdot \boldsymbol{d^0}, \text{ i.e.,}$$

$$= -\begin{bmatrix} 2 * 1.91988 \\ 50 * (-0.00307) \end{bmatrix} + \left\{\frac{\{2(1.91988)\}^2 + \{50(-0.00307)\}^2}{4^2 + 100^2}\right\} \cdot \begin{bmatrix} -4 \\ -100 \end{bmatrix}$$

$$= \begin{bmatrix} -3.84565 \\ 0.00615 \end{bmatrix}$$

$$\alpha_1 = \frac{-[\boldsymbol{\nabla f}(\boldsymbol{x^1})]^T]\boldsymbol{d^1}}{[\boldsymbol{d^1}]^T\boldsymbol{H}\boldsymbol{d^1}} = 0.499$$

$$\therefore \; \boldsymbol{x^2} = \begin{bmatrix} 1.91988 \\ -0.00307 \end{bmatrix} + 0.499 \begin{bmatrix} -3.84565 \\ 0.00615 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

This must be the optimum solution since $\nabla \boldsymbol{f}(\boldsymbol{x}^2)=0$.

To summarize:

Iteration (i)	$\boldsymbol{x}^{i-1}$	$\boldsymbol{d}^{i-1}$	α_{i-1}	$\boldsymbol{x}^1$
1	$\begin{bmatrix}2\\2\end{bmatrix}$	$\begin{bmatrix}-4\\-100\end{bmatrix}$	0.02003	$\begin{bmatrix}1.91988\\-0.00307\end{bmatrix}$
2	$\begin{bmatrix}1.91988\\-0.00307\end{bmatrix}$	$\begin{bmatrix}-3.84565\\0.00615\end{bmatrix}$	0.499	$\begin{bmatrix}0\\0\end{bmatrix}$

Note that with n=2, only 2 search directions were required. (Try with a different $\boldsymbol{x}^0$**...**)

THE POLAK-RIBIERE METHOD

This is similar to the Fletcher-Reeves method except that the multiple of the old search direction used is

$$\frac{\left[\nabla \boldsymbol{f}(\boldsymbol{x}^{i+1}) - \nabla \boldsymbol{f}(\boldsymbol{x}^{i})\right]^{\mathrm{T}} \nabla \boldsymbol{f}(\boldsymbol{x}^{i+1})}{\|\nabla \boldsymbol{f}(\boldsymbol{x}^{i})\|^2}$$

instead of $\dfrac{\left\|\nabla \boldsymbol{f}(\boldsymbol{x}^{i+1})\right\|^2}{\|\nabla \boldsymbol{f}(\boldsymbol{x}^{i})\|^2}$

THE HESTENES-STEIFEL METHOD

This is also similar but the multiple of the old search direction used is

$$\frac{\left[\nabla \boldsymbol{f}(\boldsymbol{x}^{i+1}) - \nabla \boldsymbol{f}(\boldsymbol{x}^{i})\right]^{\mathrm{T}} \nabla \boldsymbol{f}(\boldsymbol{x}^{i+1})}{[\nabla \boldsymbol{f}(\boldsymbol{x}^{i+1}) - \nabla \boldsymbol{f}(\boldsymbol{x}^{i})]^{\mathrm{T}} \nabla \boldsymbol{f}(\boldsymbol{x}^{i})}$$

instead of $\dfrac{\left\|\nabla \boldsymbol{f}(\boldsymbol{x}^{i+1})\right\|^2}{\|\nabla \boldsymbol{f}(\boldsymbol{x}^{i})\|^2}$