Unconstraint Optimization

Prof. Cesar de Prada
Dpt. Systems Engineering and Automatic Control
UVA
prada@autom.uva.es

Outline

- Theoretical solution
- Optimizing a function of one variable
  - Newton type methods
  - Bracketing methods
  - Polynomial approximation methods
- Multivariate methods
  - Gradient based algorithms
  - Newton type algorithms
  - Gradient free algorithms
- Software

There exist many methods. Only some of them will be considered in the course

Extremum analytical conditions

$$\min_{x} J(x)$$

$$x \in R^{n}$$

In unconstraint optimization problems there exist a set of analytical conditions for a point being the solution

Necessary condition

$$\left.\frac{\partial J(x)}{\partial x}\right|_{x^*} = 0$$

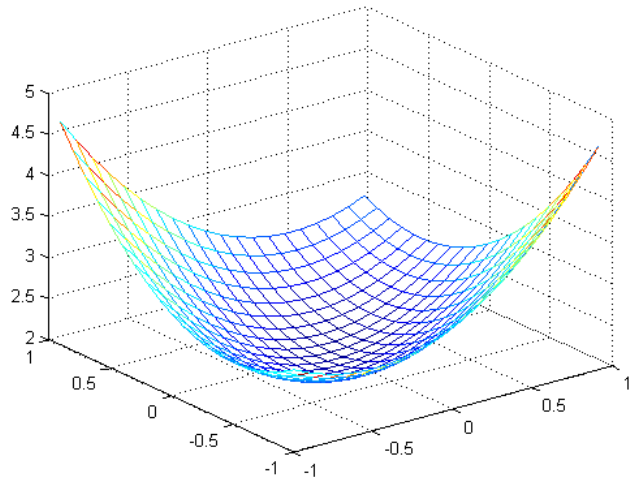
The hessian H determines the character of the possible optimum

$$H = \left.\frac{\partial^2 J(x)}{\partial x^2}\right|_{x^*}$$

Multivariable Optimization

$$\min_{x} J(x)$$

$$x \in R^{n}$$



- Example

$$J(x_1, x_2) = x_1^2 + x_2^2 - x_1 x_2 + 2$$

$$J(x_1, x_2) = \frac{1}{2}(x_1, x_2)' \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + 2$$

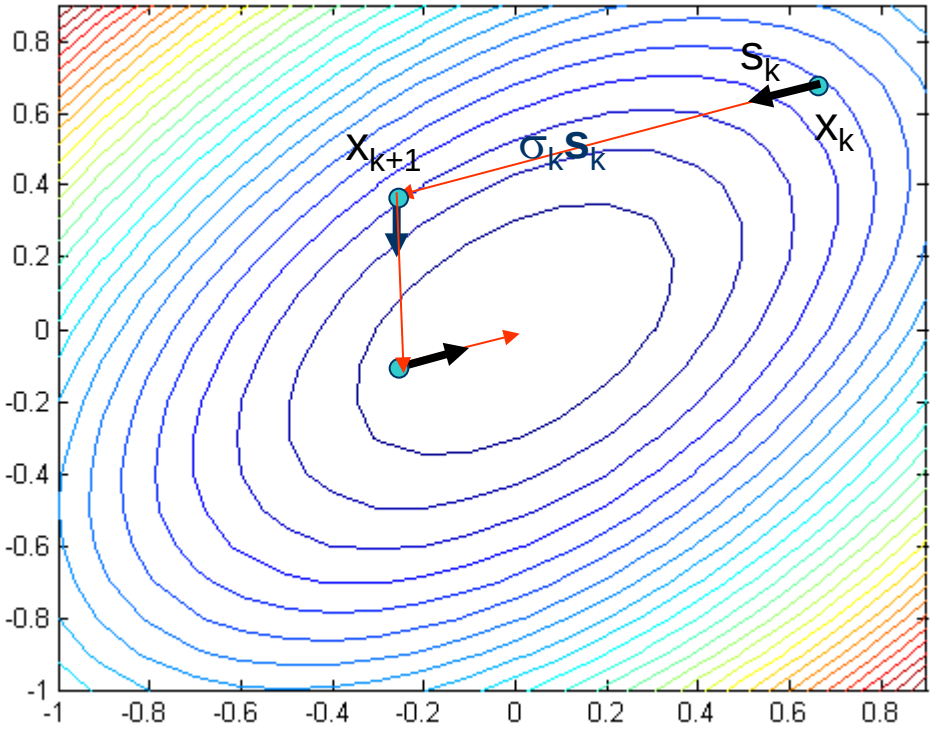
$$g(x)' = \begin{bmatrix} \frac{\partial J}{\partial x_1} & \frac{\partial J}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 2x_1 - x_2 & 2x_2 - x_1 \end{bmatrix}$$

$$g(x^*) = 0 \Rightarrow$$

$$x_1^* = 0; x_2^* = 0$$

x_k = value of vector x in the stage k

Iterative methods



Contours of J(x)

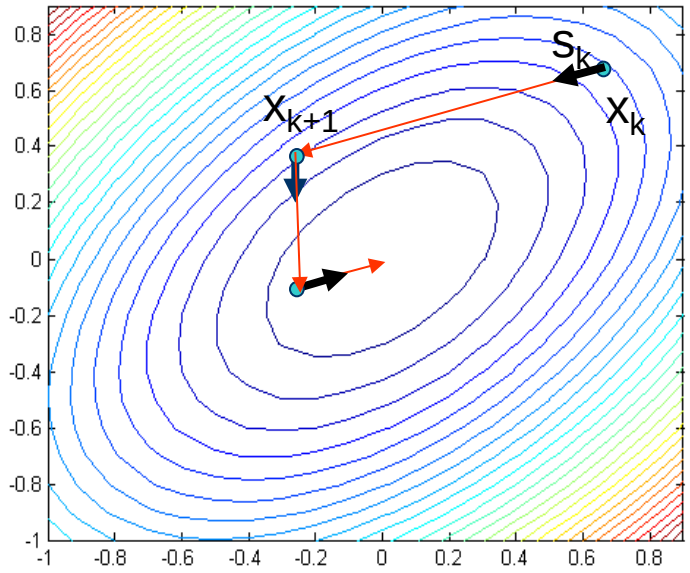
$$x_{k+1} = x_k + \Delta x_k = x_k + \sigma_k s_k$$

Iterative methods:

Starting from an initial guess x_0 , the algorithm provides a new point located in a searching direction s_k that provides a better value of J.

The algorithm continues iterating until x_k is closed enough to the optimum

Criteria for stopping the iterations



1 El gradient is small enough

$$\left\| \frac{\partial J(x_k)}{\partial x} \right\| \leq \varepsilon_1$$

2 The solution does not move in a significant way

$$\frac{\|x_{k+1} - x_k\|}{\varepsilon_0 + \|x_k\|} \leq \varepsilon_2$$

3 The cost function does not improve in a significant way

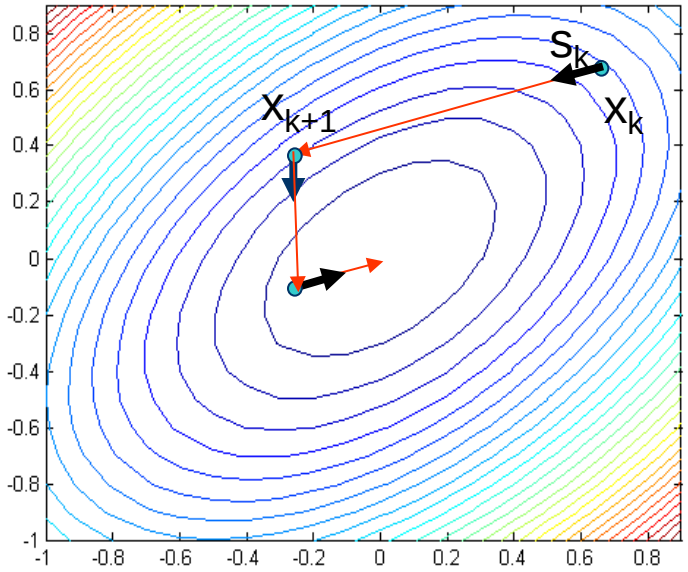
$$\frac{|J(x_{k+1}) - J(x_k)|}{\varepsilon_0 + |J(x_k)|} \leq \varepsilon_3$$

4 The number of iterations exceeds a certain maximum number N

ε Sets the precision or tolerance

$\varepsilon_0 > 0$ avoids divisions by zero

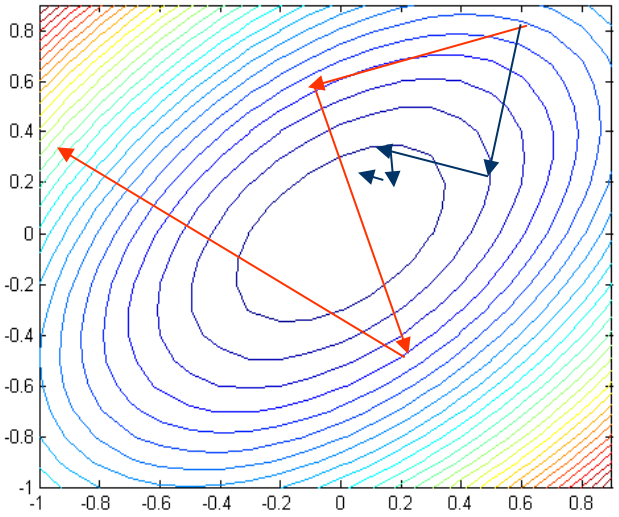
Properties of an iterative algorithm



Stability / Convergence

The step length of every iteration is decreasing

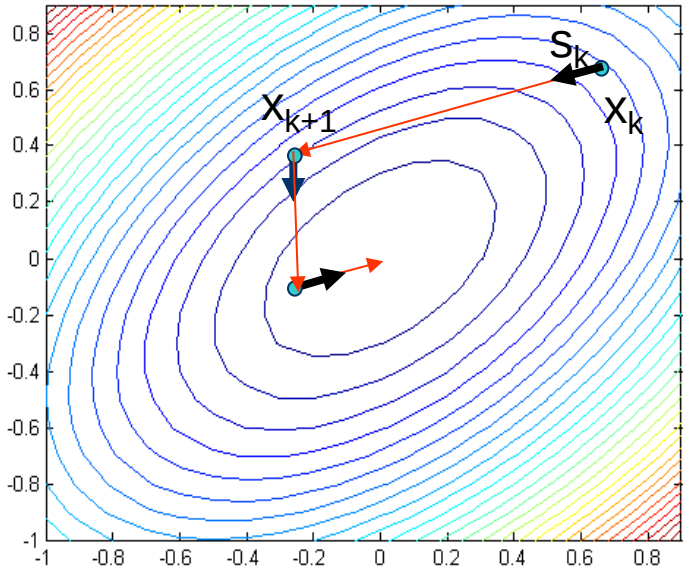
Convergence to the optimum



Local / Global convergence

Iterative algorithms are discrete dynamical systems and can be studied

Properties of an iterative algorithm



Speed of convergence to the optimum.

c speed of convergence p order of convergence

$$\frac{\left\| x_{k+1} - x^* \right\|}{\left\| x_k - x^* \right\|^p} \leq c \qquad k \text{ large}$$

$$0 < c < 1$$

Superlineal speed of convergence

$$\lim_{k \to \infty} \frac{\left\| x_{k+1} - x^* \right\|}{\left\| x_k - x^* \right\|^p} = 0$$

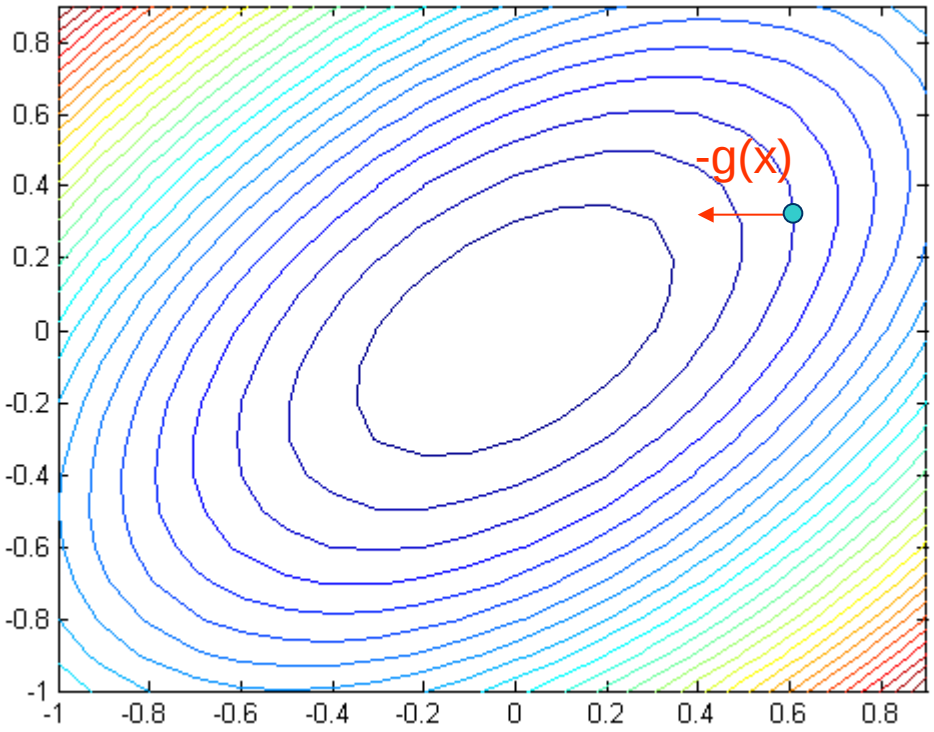
Multivariable Optimization

$$\min_{x} J(x)$$

$$x \in R^{n}$$

- Many approaches:
  - Gradient based methods
  - Newton type methods
  - Gradient free methods

Gradient based methods



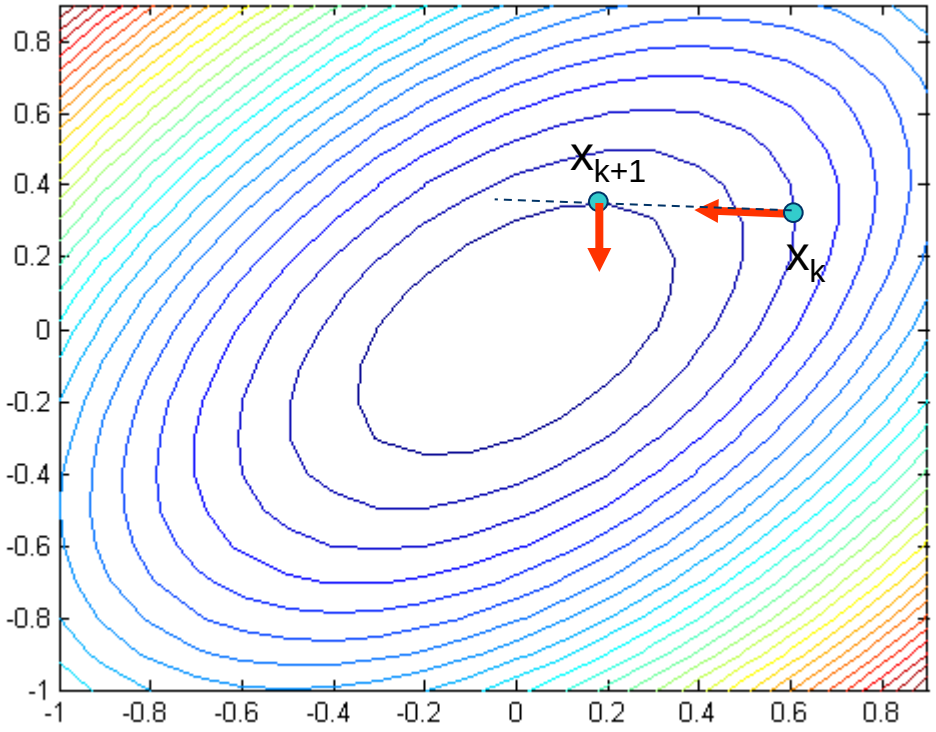
Contours of J(x)

The gradient vector of J(x) at x points to the direction where the function J has the bigger increase.

The opposite direction is the one with the maximum decrease of the function J, and can be considered as a good searching direction

$$g(x)' = \frac{\partial J}{\partial x}$$

Steepest descent method



Contours of J(x)

$$x_{k+1} = x_k - \sigma_k \frac{\partial J(x_k)}{\partial x}' =$$

$$= x_k - \sigma_k g(x_k)$$

$$\min_{\sigma_k} J(x_k - \sigma_k g(x_k))$$

$$\text{parar si } \|g(x_k)\| \leq \varepsilon$$

Move as much as possible in the direction of maximum decrease performing an (scalar) optimization of the step length σ_k

Quadratic functions

Any function continuously differentiable can be approximated by a quadratic one near the optimum:

$$J(x) = J(x^*) + \left.\frac{\partial J}{\partial x}\right|_{x^*} (x - x^*) + \frac{1}{2}(x - x^*)' \left.\frac{\partial^2 J(x)}{\partial x^2}\right|_{x^*} (x - x^*) + ...$$

$$J(x) = a + b'x + \frac{1}{2}x'Cx \qquad C = \left.\frac{\partial^2 J(x)}{\partial x^2}\right|_{x^*}$$

The region $x'Cx \leq 1$ is convex if C is PSD

They are fairly easy functions, so that if a method does not work well with quadratic functions, likely it will not work well with other functions.

Steepest descend algorithm applied to quadratic functions

$$J(x) = a + b'x + \frac{1}{2}x'Cx$$

$$g(x) = b + Cx$$

$$\begin{cases} x_{k+1} = x_k - \sigma_k g(x_k) \\ \min\limits_{\sigma_k} J(x_k - \sigma_k g(x_k)) \end{cases}$$

C, Symmetric definite positive

A quadratic function is a good candidate for testing the method because many functions can be approximated by quadratic ones near the optimum, they are easy to deal with and have analytical solutions.

Converges to the optimum when $k \to \infty$?

Speed of convergence

Steepest descend algorithm applied to quadratic functions

$$J(x) = a + b'x + \frac{1}{2}x'Cx \qquad g(x) = b + Cx$$

$$J(x_k - \sigma_k g(x_k)) = a + b'(x_k - \sigma_k g(x_k)) +$$

$$+ \frac{1}{2}(x_k - \sigma_k g(x_k))'C(x_k - \sigma_k g(x_k)) =$$

$$= J(x_k) - 2\frac{b'}{2}\sigma_k g(x_k) + \frac{1}{2}[-x_k'\sigma_k Cg(x_k) - \sigma_k g(x_k)'Cx_k +$$

$$+ \frac{1}{2}\sigma_k^2 g(x_k)'Cg(x_k)] = J(x_k) - \frac{1}{2}\begin{bmatrix}(b + Cx_k)'\sigma_k g(x_k) - \\ -\sigma_k g(x_k)'(b + Cx_k) + \sigma_k^2 g(x_k)'Cg(x_k)\end{bmatrix} =$$

$$= J(x_k) - \sigma_k \|g(x_k)\|^2 + \frac{1}{2}\sigma_k^2 g(x_k)'Cg(x_k)$$

Steepest descend algorithm applied to quadratic functions

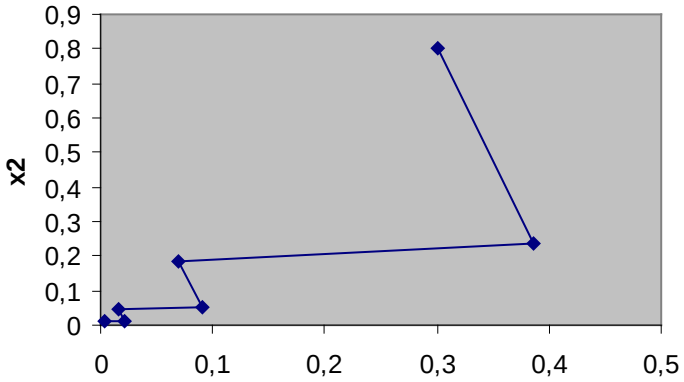
$$J(x_k - \sigma_k g(x_k)) = J(x_k) - \sigma_k \left\| g(x_k) \right\|^2 + \frac{1}{2}\sigma_k^2 g(x_k)' C g(x_k)$$

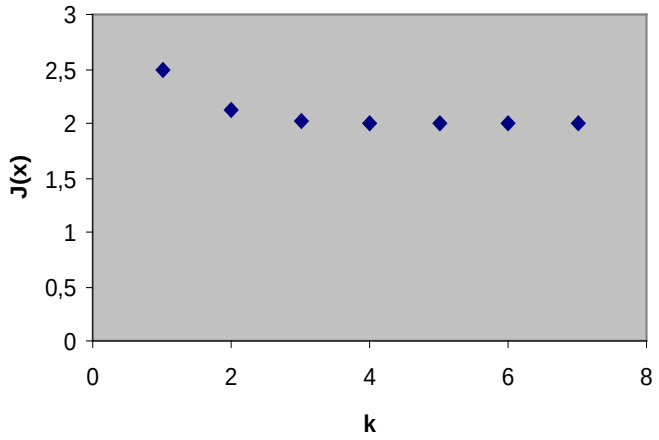
$$\min_{\sigma_k} J(x_k - \sigma_k g(x_k)) \Rightarrow \left. \frac{\partial J(x_k - \sigma_k g(x_k))}{\partial \sigma_k} \right|_{\sigma_k^*} = 0$$

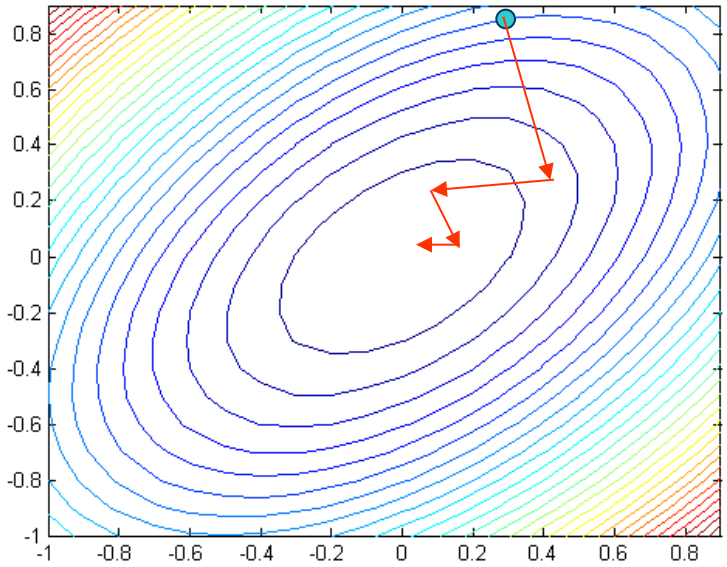
$$-\left\| g(x_k) \right\|^2 + \sigma_k^* g(x_k)' C g(x_k) = 0$$

$$\sigma_k^* = \frac{\left\| g(x_k) \right\|^2}{g(x_k)' C g(x_k)} \qquad x_{k+1} = x_k - \frac{\left\| g(x_k) \right\|^2}{g(x_k)' C g(x_k)} g(x_k)$$

Example







$$J(x_1, x_2) = x_1^2 + x_2^2 - x_1 x_2 + 2$$

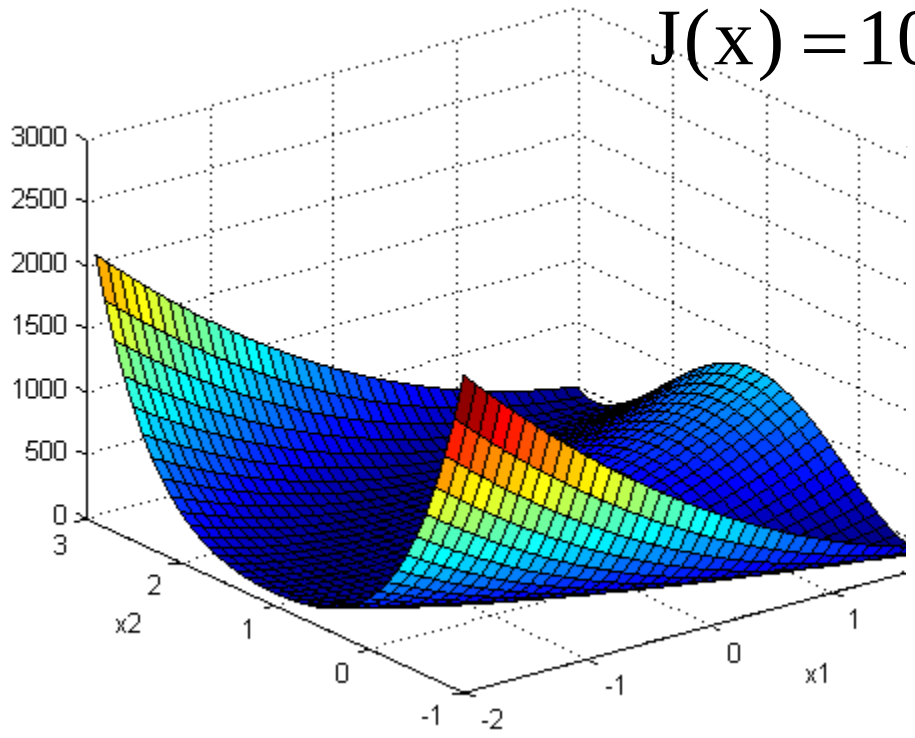
$$g(x)' = \begin{bmatrix} 2x_1 - x_2 \\ 2x_2 - x_1 \end{bmatrix} \qquad C = \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix}$$

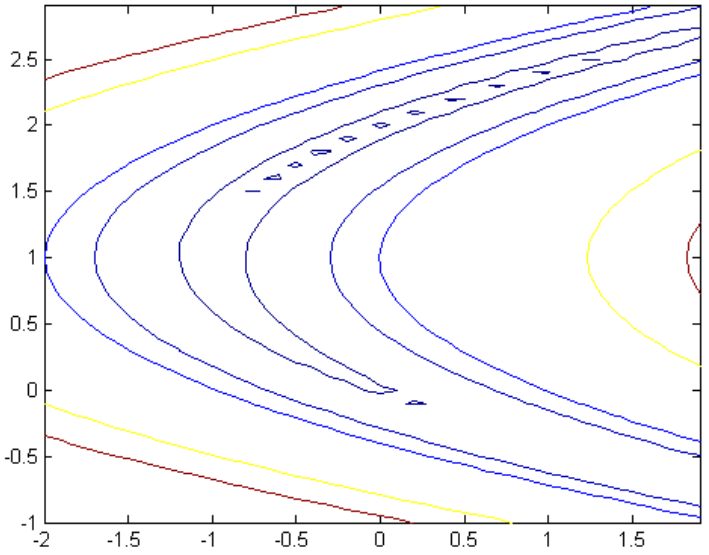
$$\mathbf{x}_0 = \begin{bmatrix} 0.8 \\ 0.3 \end{bmatrix} \qquad x_{k+1} = x_k - \sigma_k g(x_k)$$

Excel

Banana Function (Rosenbrock)

$$J(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$





Steepest descend algorithm applied to quadratic functions

Convergence? The exact optimum is reached when $|g(x)|=0$

With quadratic functions, the steepest descend either reaches the optimum in the first step or never

$$x_{k+1} = x_k - \sigma_k g(x_k) \qquad g(x) = b + Cx$$

$$g(x_{k+1}) = b + Cx_{k+1} = b + Cx_k - C\sigma_k g(x_k) =$$

$$= g(x_k) - \sigma_k C g(x_k)$$

Assume that $|g(x_0)| \neq 0$, Then, it may happens that $g(x_0)$ is or not an eigenvector of C.

Steepest descend algorithm applied to quadratic functions

If $g(x_0)$ is an eigenvector of C:

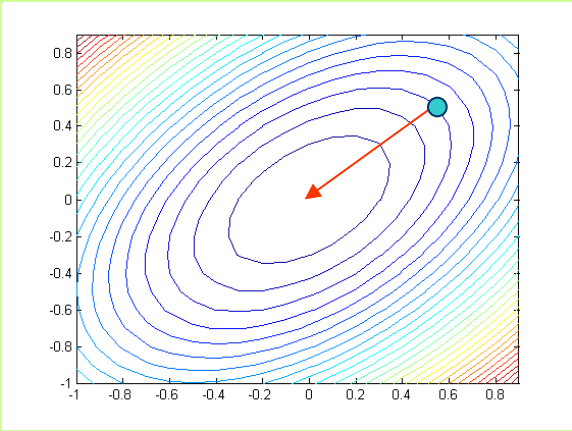
$$g(x_{k+1}) = g(x_k) - \sigma_k^* C g(x_k)$$

$$Cg(x_0) = \lambda g(x_0)$$

$$g(x_1) = g(x_0) - \sigma_k^* C g(x_0) = g(x_0) - \sigma_k^* \lambda g(x_0) =$$

$$= g(x_0) - \frac{\|g(x_0)\|^2}{g(x_0)' \lambda g(x_0)} \lambda g(x_0) = 0$$

And the optimum is reached in the first iteration of the algorithm



Steepest descend algorithm applied to quadratic functions

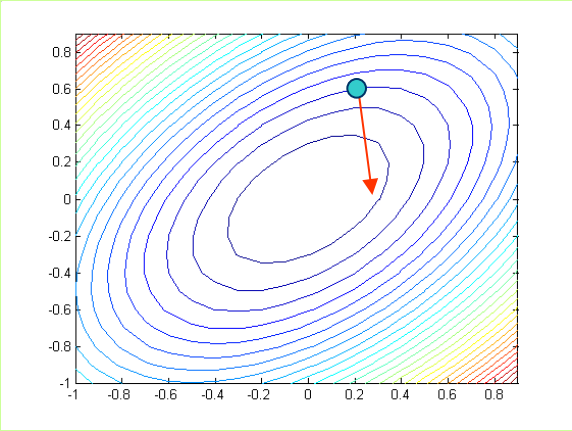
If $g(x_0)$ is not an eigenvector of C, then:

$$Cg(x_0) = \alpha g(x_0) + z \qquad z \neq 0 \quad \alpha \neq 0 \quad z \perp g(x_0)$$

$$g(x_1) = g(x_0) - \sigma_k^* Cg(x_0) = g(x_0) - \sigma_1^* (\alpha g(x_0) + z) =$$

$$= g(x_0) - \frac{\|g(x_0)\|^2}{g(x_0)'(\alpha g(x_0) + z)} (\alpha g(x_0) + z) =$$

$$= g(x_0) - \frac{1}{\alpha}(\alpha g(x_0) + z) = -\frac{z}{\alpha} \neq 0$$



And the optimum is not reached in the next iteration

Steepest descend algorithm applied to quadratic functions

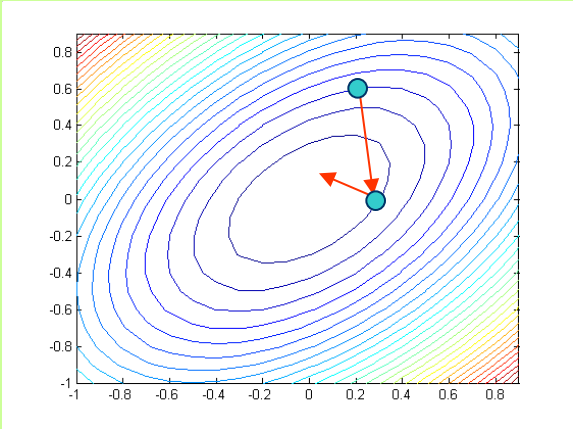
In addition, $g(x_1)$ is not an eigenvector of C:
In fact, if it were true, then there exist a λ such that:

$$Cg(x_1) = \lambda g(x_1) \Rightarrow C(-\frac{z}{\alpha}) = \lambda(-\frac{z}{\alpha}) \Rightarrow Cz = \lambda z \Rightarrow z'C = \lambda z'$$

$$z'Cg(x_0) = \lambda z'g(x_0) = 0 \qquad (*)$$

but

$$z'Cg(x_0) = z'(\alpha g(x_0) + z) = \|z\|^2 \neq 0$$



Which contradicts the expression (*), hence , by induction, it is proved that the optimum will never be reached, even if $k \to \infty$

Newton type methods

Approach: Design a perfect method for a quadratic function and extend it to other functions.

Assume that J(x) is a quadratic function, Which should be Δx so that the optimum is reached in a step?

$$J(x) = a + b'x + \frac{1}{2}x'Cx \qquad g(x) = b + Cx \quad H = C$$

C is the hessian, or matrix of second derivatives of J

$$x_{k+1} = x_k + \Delta x_k$$

$$g(x_k + \Delta x_k) = b + C(x_k + \Delta x_k) = g(x_k) + C\Delta x_k$$

$$g(x_k + \Delta x_k) = 0 \quad \Rightarrow g(x_k) + C\Delta x_k = 0$$

$$\Delta x_k = -C^{-1}g(x_k) \qquad x_{k+1} = x_k - C^{-1}g(x_k)$$

Newton's method

By analogy, when J(x) is any twice differentiable function, we could use the algorithm:

$$s_k = -\left[\frac{\partial^2 J(x_k)}{\partial x^2}\right]^{-1} g(x_k) = -H(x_k)^{-1} g(x_k)$$

$$\min_{\sigma_k} J(x_k - \sigma_k H(x_k)^{-1} g(x_k))$$

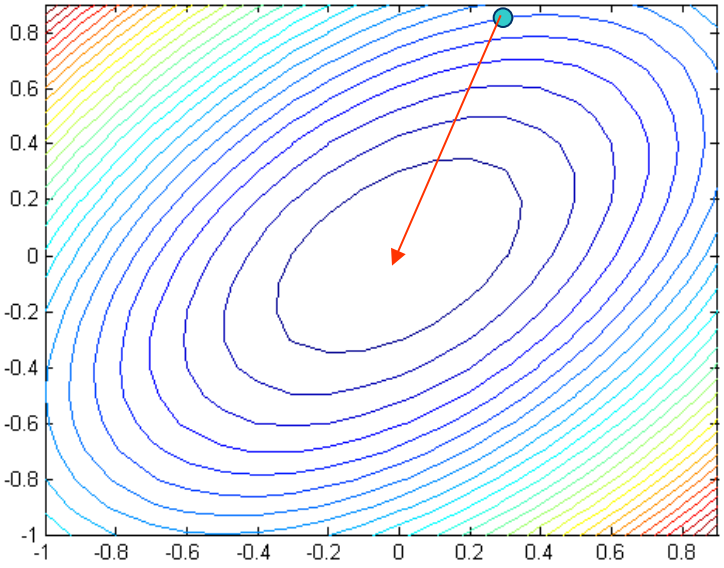
s_k search direction in the step k

$$x_{k+1} = x_k + \sigma_k s_k$$

Second order method

As the algorithm progresses and J(x) approaches the optimum, then J(x) will be more similar to a quadratic function and the algorithm will converge faster to the optimum.

Example



$$J(x_1, x_2) = x_1^2 + x_2^2 - x_1 x_2 + 2$$

$$g(\mathbf{x})' = \begin{bmatrix} 2x_1 - x_2 \\ 2x_2 - x_1 \end{bmatrix} \quad C = \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix}$$

As J is quadratic:

$$x_{k+1} = x_k - C^{-1} g(x_k)$$

$$\mathbf{x}_0 = \begin{bmatrix} 0.8 \\ 0.3 \end{bmatrix}$$

$$\mathbf{x}_1 = \begin{bmatrix} 0.8 \\ 0.3 \end{bmatrix} - \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix}^{-1} \begin{bmatrix} 1.3 \\ -0.2 \end{bmatrix} = \begin{bmatrix} 0.8 \\ 0.3 \end{bmatrix} - \begin{bmatrix} 0.8 \\ 0.3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$g(\mathbf{x}_1)' = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Excel

Convergence

$$s_k = -H(x_k)^{-1} g(x_k)$$

$$\min_{\sigma_k} J(x_k - \sigma_k H(x_k)^{-1} g(x_k))$$

$$x_{k+1} = x_k + \sigma_k s_k$$

In a first order approximation:

$$J(x_{k+1}) \approx J(x_k) + \left.\frac{\partial J(x_k)}{\partial x}\right| \Delta x_k =$$

$$= J(x_k) + g(x_k)' \sigma_k s_k =$$

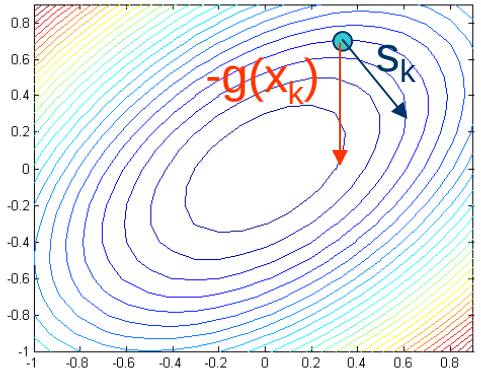
$$= J(x_k) - \sigma_k g(x_k)' \left[\frac{\partial^2 J(x_k)}{\partial x^2}\right]^{-1} g(x_k)$$

Verification: s_k is a descent direction if:

$$-g(x_k)' s_k > 0$$

If the Hessian is not PD, then there is no guarantee that J(x) decreases every step

Only if J(x) is convex we can guarantee that H is PD



Advantages / disadvantages of the Newton's method

Advantages:

Usually less iterations are required to reach the optimum

Disadvantages:

✓The Hessian and the gradient of J(x) are required

✓The Hessian must be inverted

✓There is no guarantee that in a certain step the Hessian is PD and the method converges

The gradient and the Hessian can be approximated by finite differences

Instead of inverted the Hessian, it is possible to solve a linear set of equations to compute s_k:

$$\left[\frac{\partial^2 J(x_k)}{\partial x^2}\right] s_k = -g(x_k)$$

Marquardt-Levenberg's Algorithm

It modifies the Hessian in order to guarantee that it is PD every step

$$x_{k+1} = x_k + \sigma_k s_k$$

$$s_k = -\left[\frac{\partial^2 J(x_k)}{\partial x^2} + \beta_k I\right]^{-1} g(x_k) \qquad \beta_k \geq 0$$

$$\text{choose } \beta_k \text{ so that } \left[\frac{\partial^2 J(x_k)}{\partial x^2} + \beta_k I\right] \text{ is PD}$$

$$\min_{\sigma_k} J(x_k - \sigma_k \left[\frac{\partial^2 J(x_k)}{\partial x^2} + \beta_k I\right]^{-1} g(x_k))$$

Marquardt-Levenberg's Algorithm

Choose $x_0, \quad \beta_0$

solve in $s_k \quad \left[\frac{\partial^2 J(x_k)}{\partial x^2} + \beta_k I\right] s_k = -g(x_k)$

if $\quad -g(x_k)' s_k \le 0 \Rightarrow \ \beta_k = 2\beta_k \quad$ recompute s_k

$$\min_{\sigma_k} J\left(x_k - \sigma_k \left[\frac{\partial^2 J(x_k)}{\partial x^2} + \beta_k I\right]^{-1} g(x_k)\right)$$

$$x_{k+1} = x_k + \sigma_k s_k$$

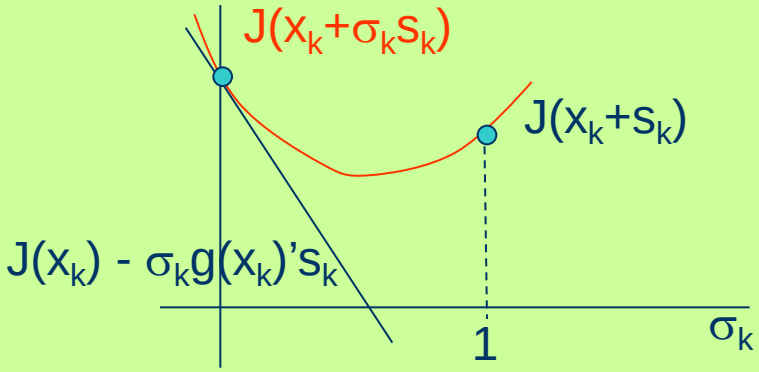
Minimization with respect to σ_k

Any of the single variable optimization methods can be applied

Sometimes, for simplicity, a value of σ_k is chosen with the only condition that $J(x'_k)$ decreases in the corresponding step.

σ_k must be ≥ 0 and the value of the pure Newton's method corresponds to σ_k =1

J can be evaluated at σ_k = 0 and 1, if it does not decreases, then a quadratic interpolation can be computed, as two points $J(x_k+s_k)$, $J(x_k)$ plus $g(x_k)'s_k$ are known, and σ_k can be obtained as its minimum:



$$\hat{\sigma}_k = \frac{-g(x_k)'s_k}{2[J(x_k+s_k)-J(x_k)-g(x_k)s_k]}$$

Quasi-Newton Methods

They try to avoid the computation of the inverse of the Hessian, which is a time consuming tasks, substituting it by a matrix $\hat{H}_k$ definite positive that approaches $H(x_k)^{-1}$ after some steps.

Taylor series expansion of J(x) at x_k :

$$J(x_{k+1}) = J(x_k + \Delta x_k) = J(x_k) + \nabla J(x_k)\Delta x_k + \frac{1}{2}\Delta x_k' \frac{\partial^2 J(x_k)}{\partial x^2}\Delta x_k + ...$$

Any approximation B (of second order) to the Hessian should verify:

$$J(x_{k+1}) = J(x_k) + \nabla J(x_k)\Delta x_k + \frac{1}{2}\Delta x_k' B \Delta x_k$$

Quasi-Newton Methods

$$J(x_{k+1}) = J(x_k) + \nabla J(x_k)\Delta x_k + \frac{1}{2}\Delta x_k' B \Delta x_k$$

Computing the gradient of $J(x_{k+1})$ with respect to Δx:

$$g(x_{k+1}) = g(x_k) + B\Delta x_k \quad \Rightarrow \quad \Delta g(x_k) = B\Delta x_k \quad \Rightarrow$$

$$B^{-1}\Delta g(x_k) = \Delta x_k$$

Hence, a matrix $\tilde{H}$ that be a second order approximation of the inverse of the Hessian should verify:

$$\boxed{\tilde{H}\Delta g(x_k) = \Delta x_k}$$

Quasi-Newton Methods

We start with an initial PD matrix $\tilde{H}_0$ and compute the first step in the search direction: $s_0 = -\tilde{H}_0 g(x_0)$ Then, we look for a correction T_0 such that $\tilde{H}_1 = \tilde{H}_0 + T_0$ verifies the above mentioned condition:

$$\tilde{H}_1 \Delta g(x_0) = \Delta x_0$$

In general:

$$s_k = -\tilde{H}_k g(x_k)$$ Search direction

$$\tilde{H}_{k+1} = \tilde{H}_k + T_k$$ Update formula with T_k a correction matrix sartisfying:

$$(\tilde{H}_k + T_k)\Delta g(x_k) = \Delta x_k$$ There are several choices of T_k satisfying this relation

Quasi-Newton Methods

There are several choices of T_k satisfying:

$$(\tilde{H}_k + T_k)\Delta g(x_k) = \Delta x_k \quad \text{p.e.}: \forall \alpha, \beta \in R^n, \neq 0$$

$$T_k = \frac{\Delta x_k \alpha'}{\alpha' \Delta g(x_k)} - \frac{\tilde{H}_k \Delta g(x_k) \beta'}{\beta' \Delta g(x_k)}$$

$$(\tilde{H}_k + T_k)\Delta g(x_k) = \tilde{H}_k \Delta g(x_k) + \frac{\Delta x_k \alpha'}{\alpha' \Delta g(x_k)} \Delta g(x_k) -$$

$$- \frac{\tilde{H}_k \Delta g(x_k) \beta'}{\beta' \Delta g(x_k)} \Delta g(x_k) = \tilde{H}_k \Delta g(x_k) + \Delta x_k - \tilde{H}_k \Delta g(x_k) = \Delta x_k$$

DFP Algorithm (Davidon, Fletcher, Powell)

In particular, the choice:

$$\alpha = \Delta x_k \quad \beta = \tilde{H}_k \Delta g(x_k)$$

$$T_k = \frac{\Delta x_k \alpha'}{\alpha' \Delta g(x_k)} - \frac{\tilde{H}_k \Delta g(x_k) \beta'}{\beta' \Delta g(x_k)}$$

$$T_k = \frac{\Delta x_k \Delta x_k'}{\Delta x_k' \Delta g(x_k)} - \frac{\tilde{H}_k \Delta g(x_k) (\tilde{H}_k \Delta g(x_k))'}{\Delta g(x_k)' \tilde{H}_k \Delta g(x_k)}$$

Leads to the DFP method, which, when applied to quadratic functions, gives an exact estimation of the hessian after n steps. n = size of x

DFP Algorithm (Davidon, Fletcher, Powell)

Choose x_0 $\tilde{H}_0$ PD, simmetric

$$\min_{\sigma_k} J(x_k - \sigma_k \tilde{H}_k g(x_k))$$

$$x_{k+1} = x_k - \sigma_k \tilde{H}_k g(x_k)$$

$$\Delta x_k = x_{k+1} - x_k \qquad \Delta g(x_k) = g(x_{k+1}) - g(x_k)$$

$$\tilde{H}_{k+1} = \tilde{H}_k + \frac{\Delta x_k \Delta x_k'}{\Delta x_k' \Delta g(x_k)} - \frac{\tilde{H}_k \Delta g(x_k)(\tilde{H}_k \Delta g(x_k))'}{\Delta g(x_k)' \tilde{H}_k \Delta g(x_k)}$$

BFGS Algorithm (Broyden, Fletcher, Goldfarb, Shanno) 1970

The Hessian is recursively estimated

Δx y Δg play symmetric roles in relation with DFP

$$B_{k+1}\Delta x_k = \Delta g(x_k)$$

$$\Delta x_k = \tilde{H}_{k+1}\Delta g(x_k)$$

swap Δx_k with $\Delta g(x_k)$ in the expression of T_k

$$B_{k+1} = B_k + \frac{\Delta g(x_k)\Delta g(x_k)'}{\Delta x_k'\Delta g(x_k)} - \frac{B_k \Delta x_k \Delta x_k' B_k}{\Delta x_k' B_k \Delta x_k}$$

if J(x) is convex, then B_k is always PD

$\tilde{H}_{k+1} = B_{k+1}^{-1}$ can be estimated using

$$(A + zv')^{-1} = A^{-1} - \frac{A^{-1}zv'A^{-1}}{1 + v'A^{-1}z}$$

If B_k es PD and $\Delta x_k' \Delta g(x_k) > 0$ then B_{k+1} is PD. If not, B_k is not updated

BFGS Algorithm (Broyden, Fletcher, Goldfarb, Shanno) 1970

choose x_0, $\tilde{H}_0$ PD, symmetric

$$s_k = -\tilde{H}_k g(x_k)$$

Usually is more efficient than DFP

$$\min_{\sigma_k} J(x_k + \sigma_k s_k)$$

$$x_{k+1} = x_k + \sigma_k s_k$$

$$\Delta x_k = x_{k+1} - x_k \qquad \Delta g(x_k) = g(x_{k+1}) - g(x_k)$$

$$\tilde{H}_{k+1} = \left[I - \frac{\Delta x_k \Delta g(x_k)'}{\Delta x_k' \Delta g(x_k)} \right] \tilde{H}_k \left[I - \frac{\Delta x_k \Delta g(x_k)'}{\Delta x_k' \Delta g(x_k)} \right] + \frac{\Delta x_k \Delta x_k'}{\Delta x_k' \Delta g(x_k)}$$

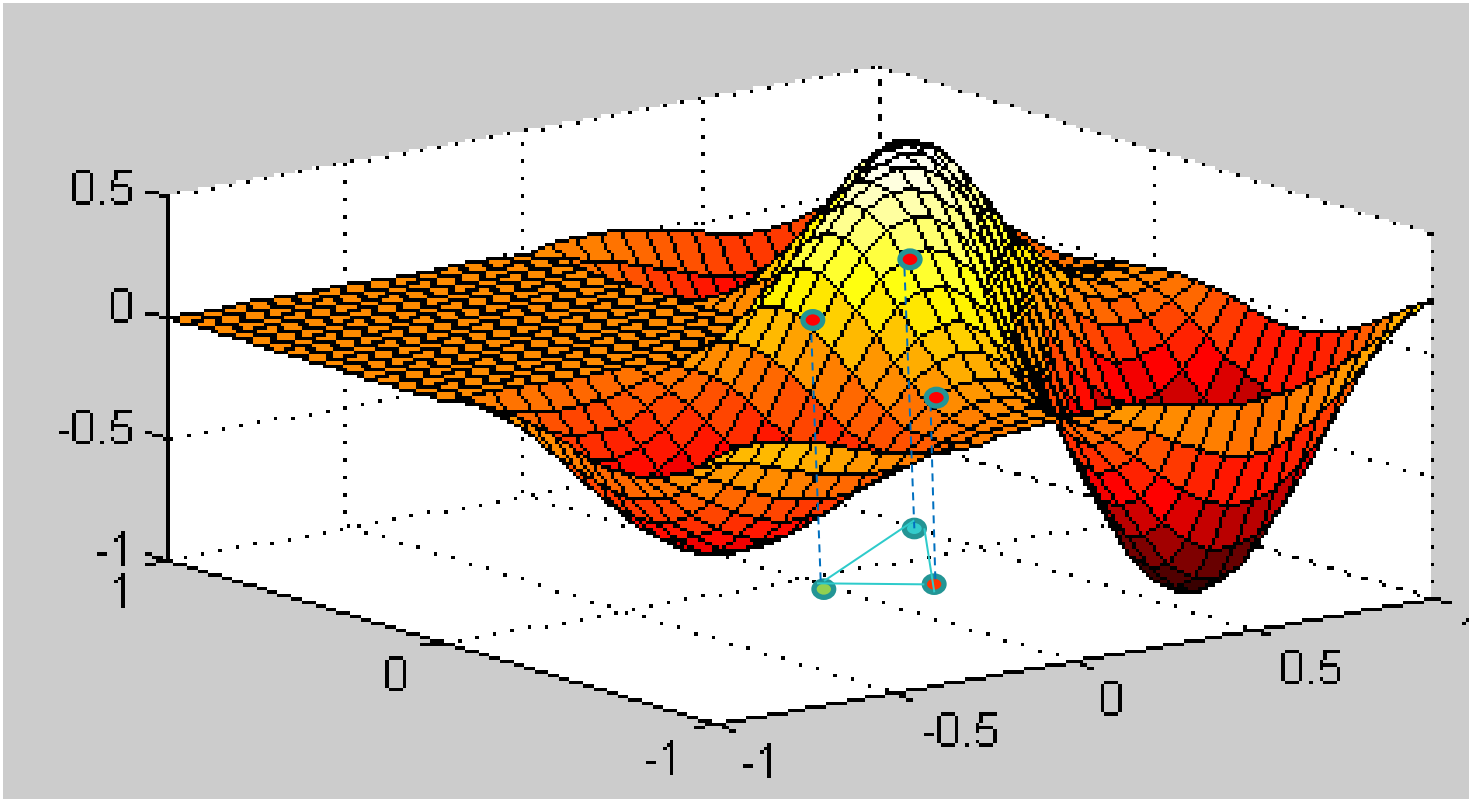
Methods using values of J(x) only (Direct search)

- Methods based on the use of the gradient of J(x) work well when applied to “smooth” functions, even if they are many decision variables.
- Nevertheless, in practice, the computation of the gradient can be difficult, or even impossible, due to discontinuities, complex non-linearities, etc.
- Very often, numerical estimations of the gradient based on finite differences are time consuming.
- An alternative for those situations where the gradient is difficult to obtain is to relay on optimization methods that only use values of J(x), e.g.:
  - Simplex
  - Powell’s conjugate directions

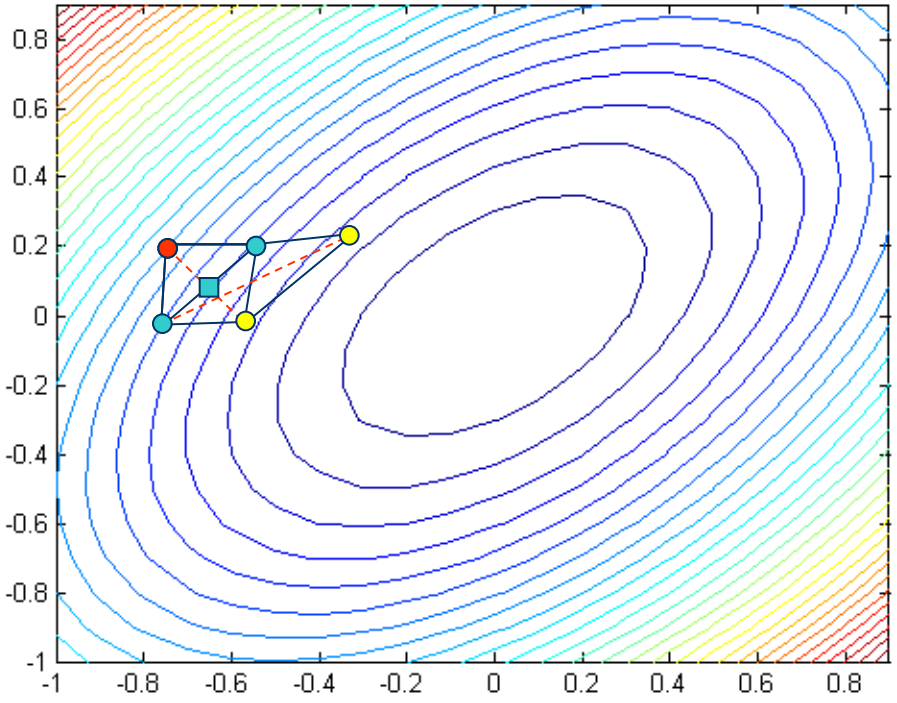
Simplex search method

- ✓ This type of methods uses sets of points where the value of J(x) is evaluated, located on places that form a certain pattern, employing these values to evolve towards a new pattern closer to the optimum.
- ✓ The easiest geometrical figure in a n-dimensional space is called a simplex and has n + 1 vertices. For instance, a simplex in R^2 is an equilateral triangle, in R^3 un tetrahedron, etc.
- ✓ The simplex search method employs the values of the function in the n+1 vertices of this geometrical figure to generate another simplex located closer to the optimum and continues the iteration until the optimum is found within the required precision.
- ✓ Excepting the name, it has nothing in common with the LP Simplex method.

Simplex



Simplex search method



vertex

centroid

1 J(x) is computed in the n+1 vertices of the simplex

2 The vertex with the worst value is selected and projected a certain distance through the centroid formed by the remaining vertices.

3 A new simplex is formed with the projected vertex and the remaining ones

4 If there is an improvement, the iteration continues until the required tolerance is met

Simplex search method,

When the iterations advance, either the optimum is reached or it is possible that, before reaching the optimum with the required precision, a cyclical situation appears between two or more simplexes. In order to avoid these cycles, three rules are applied:

1 If the worst vertex was already generated in the previous iteration, then change to the second worst vertex.

2 If a vertex remain in the same value for more than M iterations, then, reduce the size of the simplex by a factor, using as a base the point with smaller value of J(x). Advise: $M = \text{int}\,(1.65n + 0.05n^2)$

3 The iterations are finished when the simplex is small enough or the standard deviation of the values of J(x) evaluated in the vertices is small enough

Generation of points of the simplex

Starting from the base point $x^{(0)}$ and a given scale factor α, the coordinates of the remaining vertices $x^{(i)}$, i= 1,...,n of an initial regular simplex can be computed by:

$$x_j^{(i)} = \begin{cases} x_j^{(0)} + \left[\dfrac{\sqrt{n+1}+n-1}{n\sqrt{2}}\right]\alpha & \text{if } j = i \\ x_j^{(0)} + \left[\dfrac{\sqrt{n+1}-1}{n\sqrt{2}}\right]\alpha & \text{if } j \neq i \end{cases}$$

If $x^{(j)}$ is the vertex to be reflected, then, the centroid x_c of the remaining points and the new location of the reflected point are:

$$x_c = \frac{1}{n}\sum_{\substack{i=0 \\ i\neq j}}^{n} x^{(i)} \qquad x_{new}^{(j)} = 2x_c - x_{old}^{(j)}$$

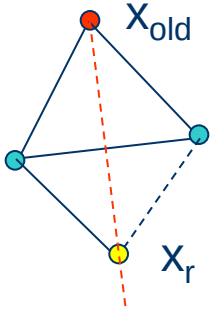
J_m lowest value of J on simplex

J_M second highest value of J on the simplex

Nelder – Mead's method

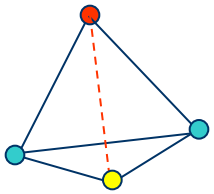
Instead of using a regular simplex, it expands or contracts it according to a set of rules in order to improve the convergence.

Normal reflection



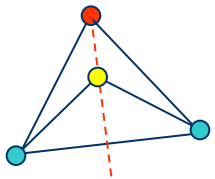
$J_m < J(x_r) < J_M$

$\theta = 1$



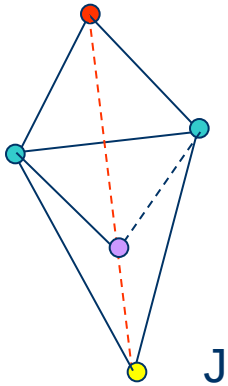
$J_M < J(x_r) < J(x_{old})$

$\theta = 0.5$



$J_M < J(x_r) \geq J(x_{old})$

$\theta = - 0.5$



$J(x_r) < J_m$

$\theta = 2$

$$x_c = \frac{1}{n}\sum_{\substack{i=0 \\ i \neq j}}^{n} x^{(i)} \qquad x_{new}^{(j)} = x_{old}^{(j)} + (1+\theta)(x_c - x_{old}^{(j)}) \qquad -1 \leq \theta < 1$$

Nelder – Mead's method

- Advantages:
  - Easy to implement, requiring small storage resources and evaluations of the function only.
  - Few adjustable parameters
  - Robust againt noises and errors in the computation of the function as it uses the worst value
- Disadvantages:
  - Scaling of the variables id required
  - Slow convergence as it does not use neither pass iterations information nor structural one

Powell’s conjugate directions method

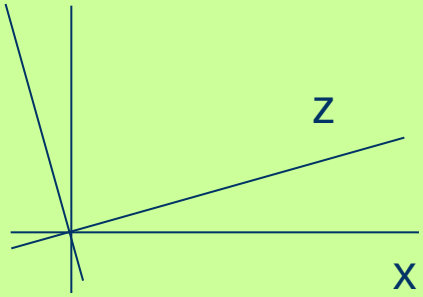
In a similar way as other methods, the design of the method is made with reference to a quadratic function, applying it later on to any J(x)

$$J(x) = a + b'x + \frac{1}{2}x'Cx$$

How to find the minimum of J(x) without using the gradient or the Hessian?

The core idea is to look for the minimum along each of the so-called C-conjugate directions, on which the function J(x) only depends on a single component of vector x, so that the search can be performed with mono-dimensional methods

C conjugate directions



If a matrix S diagonalizes C, so that S'CS = D is diagonal, then, using the coordinate system given by: $z = S^{-1}x$

$$J(x) = a + b'x + \frac{1}{2}x'Cx =$$

$$= a + b'Sz + \frac{1}{2}z'S'CSz = a + b'Sz + \frac{1}{2}z'Dz$$

As there are no cross terms in z because D is diagonal, the function J(Sz) is separable and its minimum can be computed as a sequence of n minimization problems with respect to every component z_j of z

$$\mathrm{J(x)} = \mathrm{J(Sz)} = 3 + \begin{bmatrix} 2 & 1 \end{bmatrix}\begin{bmatrix} z_1 \\ z_2 \end{bmatrix} + \begin{bmatrix} z_1 & z_2 \end{bmatrix}\begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix}\begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = 3 + 2z_1 + z_2 + 3z_1^2 + 2z_2^2 =$$

$$= \left(3 + 2z_1 + 3z_1^2\right) + \left(z_2 + 2z_2^2\right) = \mathrm{J}_1(z_1) + \mathrm{J}_2(z_2)$$

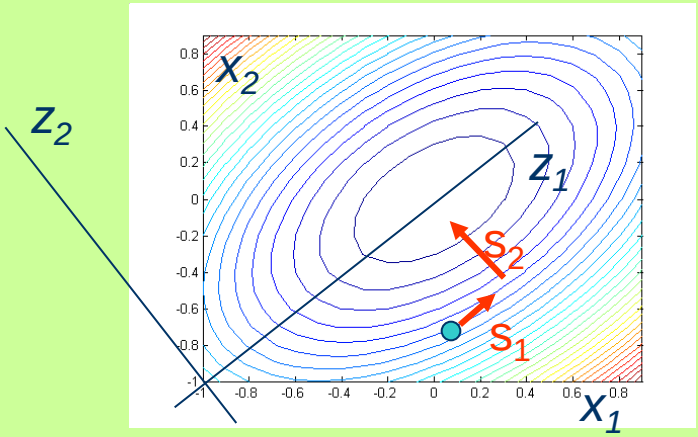
C conjugate directions

$$x = Sz = \left[s_1 \vdots \quad s_2 \vdots \quad \ldots \quad s_n \vdots \right] \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix} = z_1 s_1 + z_2 s_2 + \ldots + z_n s_n$$

Minimize J(Sz) over every component z_j of z is equivalent to minimize J(x) over every one of the n directions called C-conjugates

The new axis coincide with the main directions of J(x)

Using this method, after n iterations we will reach the optimum of a quadratic function



C conjugate directions

Condiction S'CS = D diagonal can be formulated as:

$$\begin{bmatrix} s_1' \\ s_2' \\ \vdots \\ s_n' \end{bmatrix} C \begin{bmatrix} s_1 \mid & s_2 \mid & \dots & \mid s_n \end{bmatrix} = \begin{bmatrix} a_{11} & 0 & \dots & 0 \\ 0 & a_{22} & \dots & 0 \\ 0 & 0 & \dots & 0 \\ 0 & \dots & 0 & a_{nn} \end{bmatrix}$$

$$s_i' C s_j = 0 \quad i \neq j$$

Definition: Given C (n x n) symmetric, the directions s_1, s_2, ...s_r $r \leq n$ are C-conjugates if they are lineary independent and verify:

$$s_i' C s_j = 0 \quad i \neq j$$

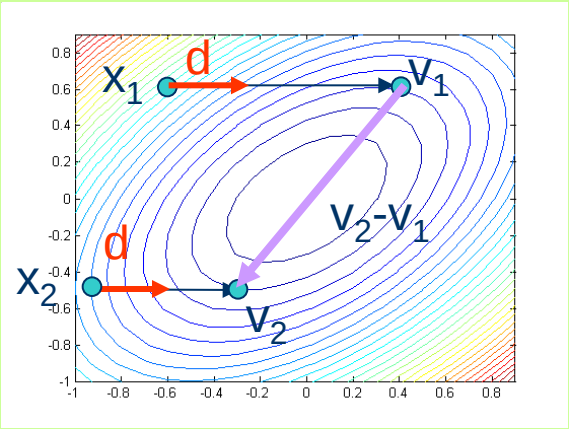
Parallel subspace property

Given a quadratic function J(x) and a direction d, $\forall\ x_1 \neq x_2 \in R^n$ it happens that if v_1 is the solution of

$$\min_{\sigma} J(x_1 + \sigma d)$$

And if v_2 is the solution of $\min_{\sigma} J(x_2 + \sigma d)$

Then, the direction $v_2 - v_1$ is C-conjugate to d



C conjugate directions

Proof:

$$J(x) = a + b'x + \frac{1}{2}x'Cx \qquad g(x) = b + Cx$$

$$J(w) = J(x + \sigma d) \qquad \text{en el óptimo:}$$

$$\frac{\partial J}{\partial \sigma} = \frac{\partial J}{\partial w}\frac{\partial w}{\partial \sigma} = (b' + w'C)d = 0$$

$$\left.\begin{aligned}(b' + v_2'C)d = 0\\ (b' + v_1'C)d = 0\end{aligned}\right\} (v_2' - v_1')Cd = 0$$

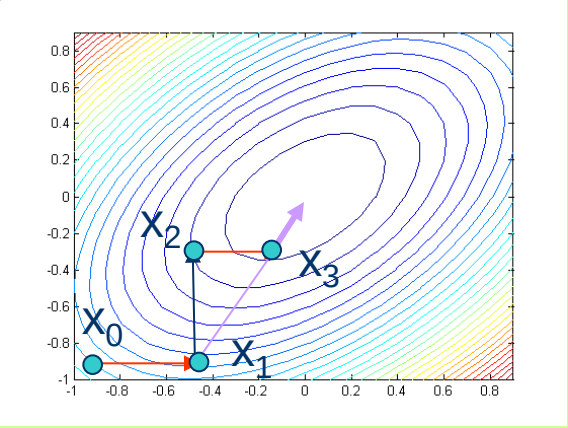
So, direction $v_2 - v_1$ is C conjugate with d

The idea can be extended to n directions: If starting from x_1 and x_2 we obtain v_1 and v_2 after $m < n$ searches over the m conjugate directions $s_1, s_2, \ldots, s_m$, then $v_2 - v_1$ is C-conjugate with all the $s_1, s_2, \ldots, s_m$ directions

Powell's conjugate directions method

For the generation of two conjugate directions, the parallel subspace property uses two stating points and two minimizations in a common direction d. The same result can be obtained with one single starting point and more minimizations:

As we can see in the Figure, minimizing J(x) successively over the n directions of the axis of x, the n+1 minimization is parallel to the first one, so that vector $x_{n+1} - x_1$ is C-conjugate to the first axis. Minimizing in this direction and repeating the procedure successively, the minimization over the n C-conjugate directions can be performed and the optimum reached, without computing the diagonalization of C



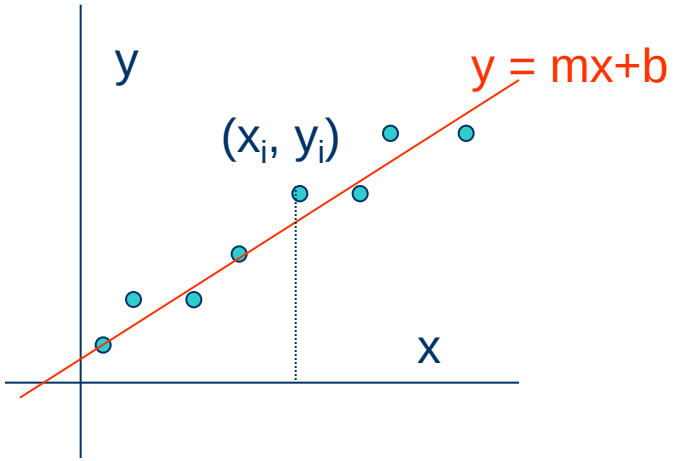
Powell's conjugate directions method

1. Choose x_0 and n linearly independent directions, e.g. $s_i = e_i$
2. Built the set of n+1 search directions $s_n, s_1, s_2, s_3, ..., s_n$
3. Minimize J(x) over the n+1 search directions successively. Be v_j the optimum in the j-iteration
4. Compute a new search direction as $s_{n+1} = v_{n+1} - v_1$ that will be C conjugate to s_n (and to the previous ones)
5. Use as new set of n+1 search directions $s_{n+1}, s_2, s_3, ..., s_n, s_{n+1}$ where s_1 has been scratched and $s_{n+1} = v_{n+1} - v_1$ has been added
6. Check if the optimum has been reached as well as the linear independence of the n different s_i
7. Go back to 3

Powell’s conjugate directions method

- If J(x) is quadratic, after n loops, the n+1 searches are made over conjugate directions and the optimum is reached exactly
- If J(x) is not quadratic, it can be proved that the algorithm has superlinear convergence to the optimum
- Is an efficient and reliable method

Fitting a curve to a set of data by least squares (LS)



Find the linear relation that better fits to a set of N couples (x_i, y_i) of experimental data. The problem can be formulated as an optimization one: Look for the straight line parameters (m , b) that provide a minimum value to the sum of the squares of the deviations between the data and the formula

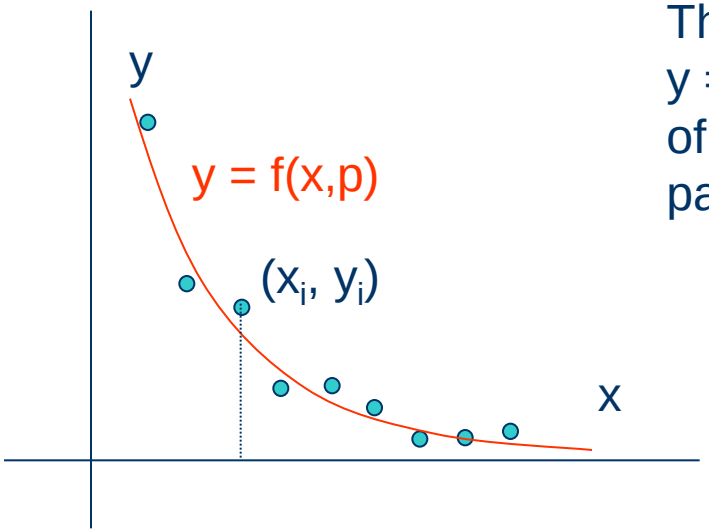
$$\min_{m,b} \sum_{i=1}^{N} (y_i - (mx_i + b))^2$$

There exist an analytical solution

$$\frac{\partial \sum_{i=1}^{N} (y_i - (mx_i + b))^2}{\partial m} = 0$$

$$\frac{\partial \sum_{i=1}^{N} (y_i - (mx_i + b))^2}{\partial b} = 0$$

Data fit



The idea can be extended to any function $y = f(x, p)$ that should fit a set of N couples of data (x_i , y_i). Here p are the unknown parameters that must be estimated.

$$\min_{p} \sum_{i=1}^{N} (y_i - f(x_i, p))^2$$

The problem can be formulated as the minimization of the sum of squares of the residuals $y_i – f(x_i,p)$ with respect to the function parameters p

Redlich-Kwong's equation

Empirical relation among:

Pressure P

Temperature T

Molar volume v

of a real gas

$$P = \frac{RT}{v-b} - \frac{a}{v(v+b)\sqrt{T}}$$

a and b are unknown coefficients that must be estimated using experimental data

Example: CO_2 data

Excel

volumen molar v	Temperatura T	Presión P
500	273	33
500	323	43
600	373	45
700	273	26
600	323	37
700	373	39
400	272	38
400	373	63,6

Solving algebraic equations

In many problems it is necessary to solve equations such as:

$$f(x) = 0$$

Or sets of equations :

$$\begin{cases} f(x, y) = 0 \\ g(x, y) = 0 \end{cases}$$

There are several methods available:

- ✓Newton
- ✓Secant
- ✓Bisection

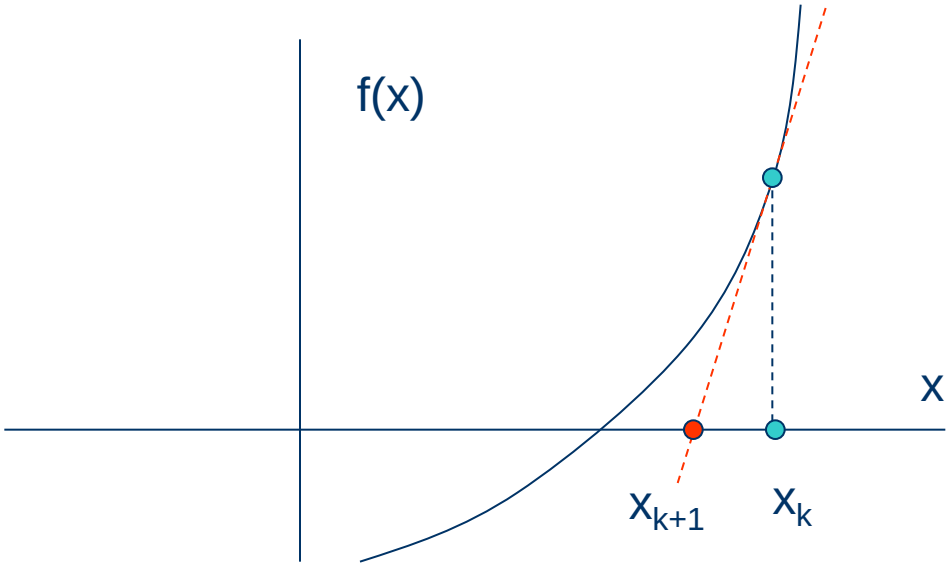
But also can be formulated as optimization problems

Newton's method

$$f(x) = 0$$

$$f(x_{i+1}) = f(x_i) + \left.\frac{\partial f}{\partial x}\right|_{x_i} (x_{i+1} - x_i) + \ldots = 0$$

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$



Newton-Raphson

$$F(x) = 0$$

$$F(x_{i+1}) = F(x_i) + \left.\frac{\partial F}{\partial x}\right|_{x_i} (x_{i+1} - x_i) + \ldots = 0$$

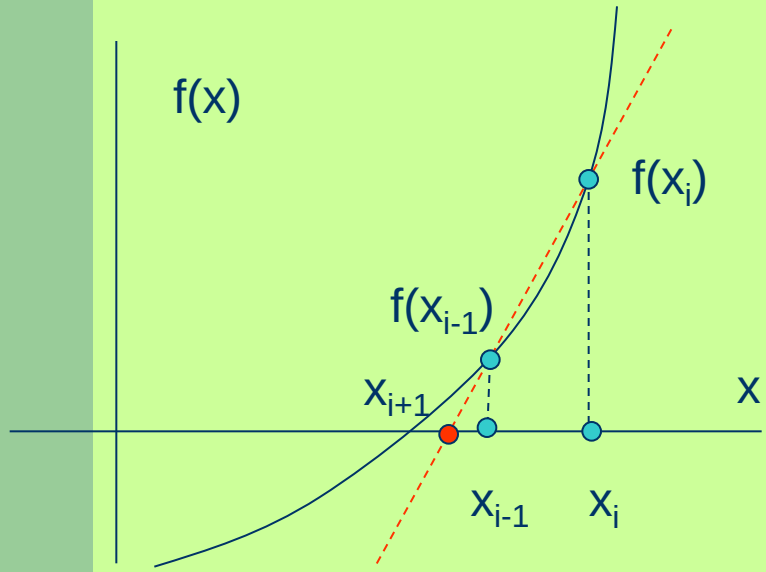
$$x_{i+1} = x_i - \left[\left.\frac{\partial F}{\partial x}\right|_{x_i}\right]^{-1} F(x_i)$$

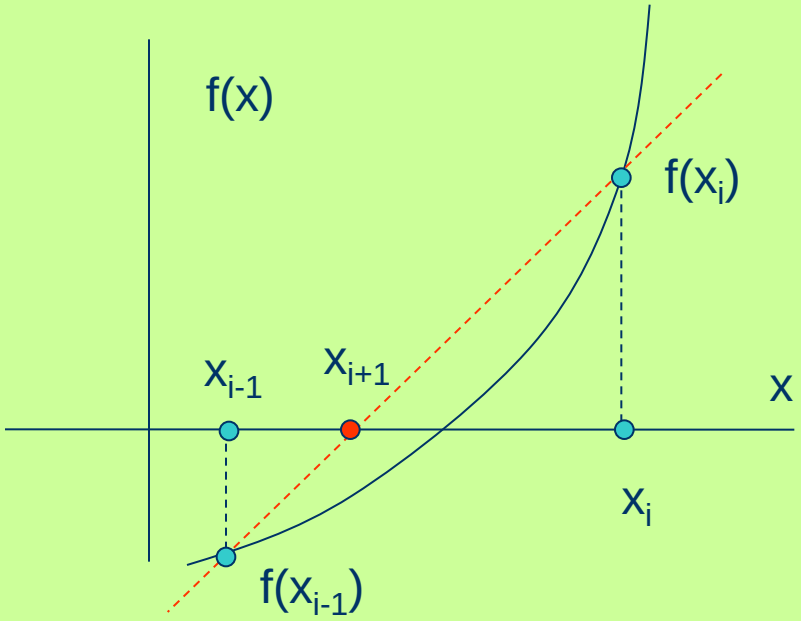
It is necessary to compute and estimate the Jacobian every step

Secant method

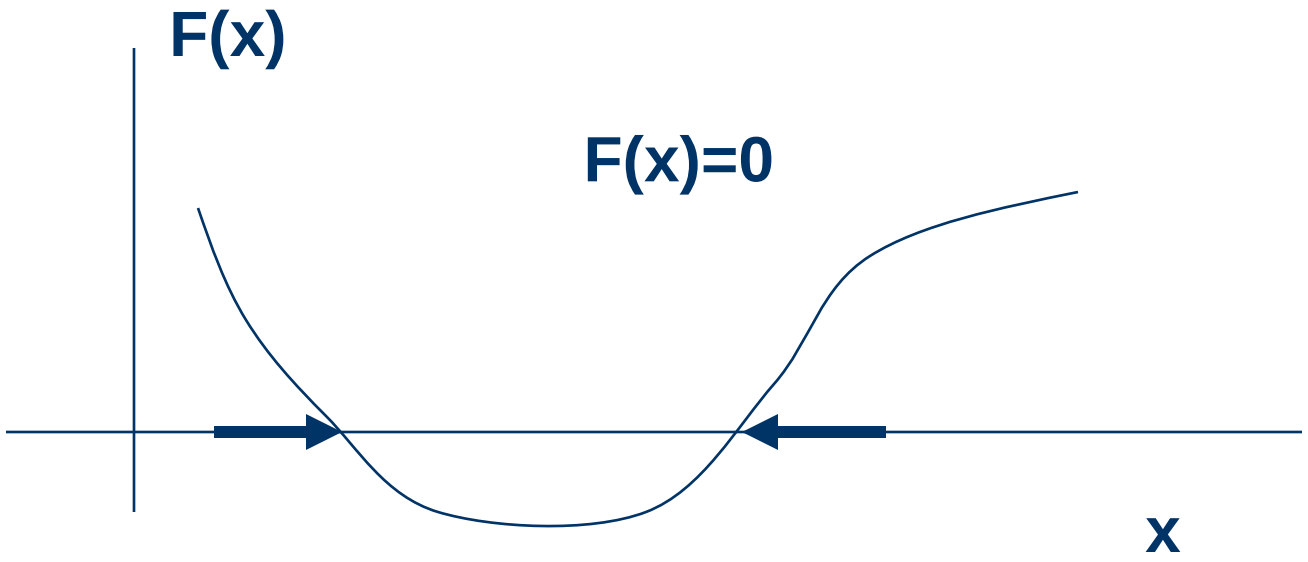
$$x_{i+1} = \frac{x_{i-1} f(x_i) - x_i f(x_i)}{f(x_i) - f(x_{i-1})}$$

It avoids the computation of the derivatives

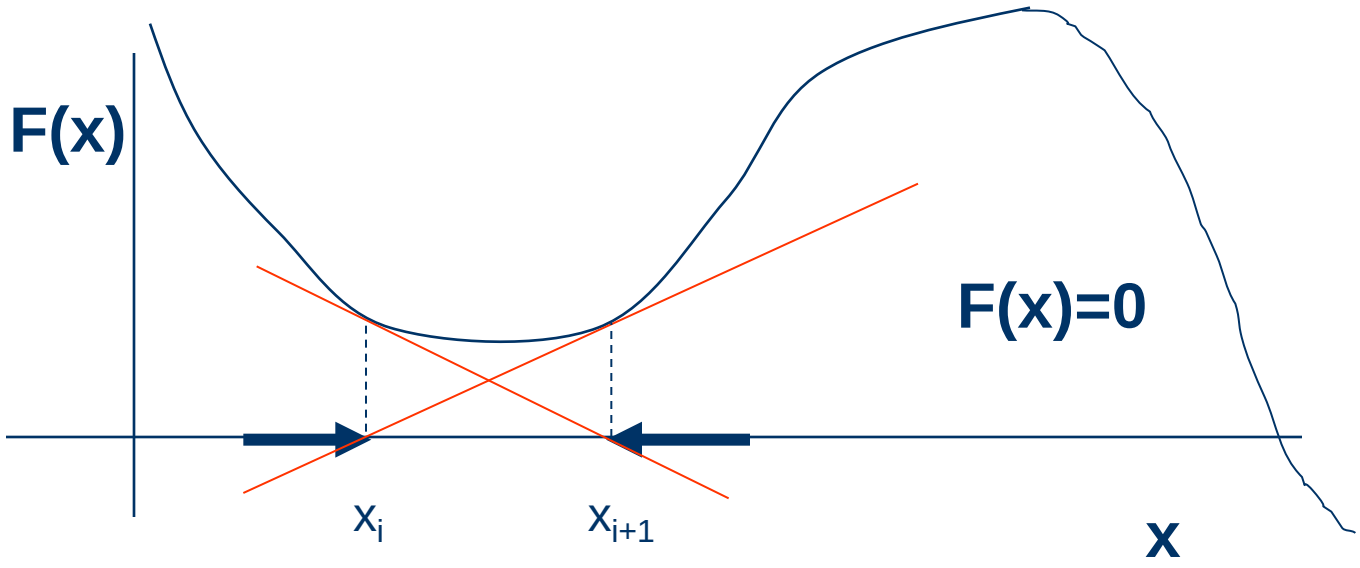




Initialization problem



Oscilations



Formulation as an optimization problem

The problem can be formulated as:

$$\begin{cases} f(x,y) = 0 \\ g(x,y) = 0 \end{cases}$$

$$\min_{x,y,\varepsilon_1,\varepsilon_2} \quad \varepsilon_1^2 + \varepsilon_2^2$$

$$\begin{cases} f(x,y) = \varepsilon_1 \\ g(x,y) = \varepsilon_2 \end{cases}$$

If the problem is feasible, the minimum of $\varepsilon_1^2 + \varepsilon_2^2$, is (0,0), so that x and y will verify the set of equations

$$\min_{x,y} \quad f(x,y)^2 + g(x,y)^2$$

Some important points with regard to the numerical solution of optimization problems

Once an optimization problem has been formulated, it is convenient to reshape it in order to facilitate its numerical solution and the search of the optimum.

Among possible changes in the formulation we can mention:

- ✓Scaling the independent variables
- ✓Changes to avoid computations out of the admisible range in functions such as: log(x), $x^{½}$, …
- ✓Changes to avoid non-differenciable expressions
- ✓Changes to improve the convexity of the problem

In addition, it is important an adequate adjustment of the precision, tolerances, number of steps, etc. of the optimization algorithm

Scaling

Scaling refers to the relative order of magnitude of the problem variables, which should not be very different in order to avoid numerical problems created by wide different sensibilities in different directions.

Example: x_1 takes values around 100 and x_2 around 0.1

$$J(x_1, x_2) = 10x_1 + 5x_2 - x_1 x_2$$

It can be reformulated in terms of the new scaled variables u_1, u_2

$$J(u_1, u_2) = 1000\left(\frac{x_1}{100}\right) + 0.5\left(\frac{x_2}{0.1}\right) - 10\left(\frac{x_1}{100}\right)\left(\frac{x_2}{0.1}\right) = 1000u_1 + 0.5u_2 - 10u_1 u_2$$

$$u_1 = \frac{x_1}{100} \quad u_2 = \frac{x_2}{0.1}$$

Now, u_1 and u_2 both have values around 1

Convexification

$$J(x_1, x_2) = x_1 x_2$$

Non convex function in x

$$x_1 = e^{v_1} \qquad x_2 = e^{v_2}$$

Change of variables

$$x_1 x_2 = e^{v_1} e^{v_2} = e^{v_1 + v_2}$$

$$\min_{x_1, x_2} J(x_1, x_2) = \min_{v_1, v_2} J(v_1, v_2)$$

Convex function in v

Range problem!