CE 395 Special Topics in Machine Learning

Assoc. Prof. Dr. Yuriy Mishchenko

Fall 2017

NUMERICAL OPTIMIZATION

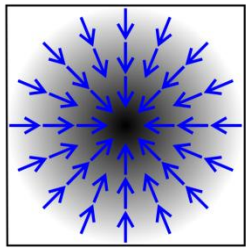
Gradient

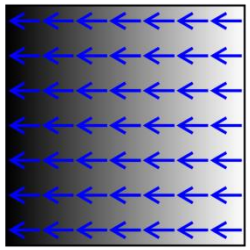
Definition: Gradient of a multi-parameter function *f*(*x*) is defined as the vector of its partial derivatives

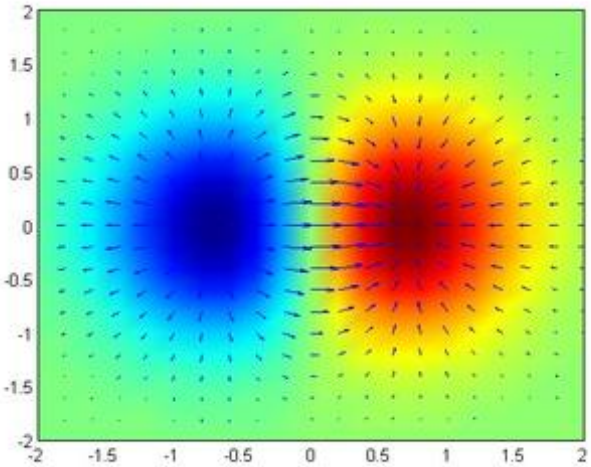
$$\nabla f = \left[\frac{\partial f(x)}{\partial x_i} \right] = \left[\frac{\partial f(x)}{\partial x_1}, \frac{\partial f(x)}{\partial x_2}, \ldots, \frac{\partial f(x)}{\partial x_p} \right]$$

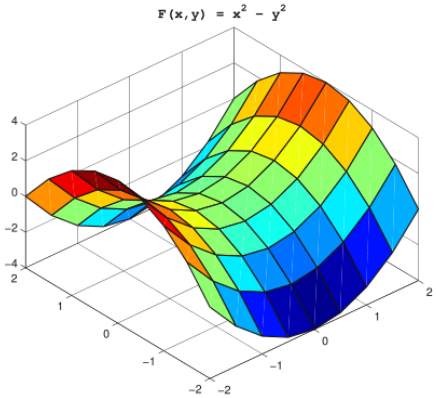
Gradient

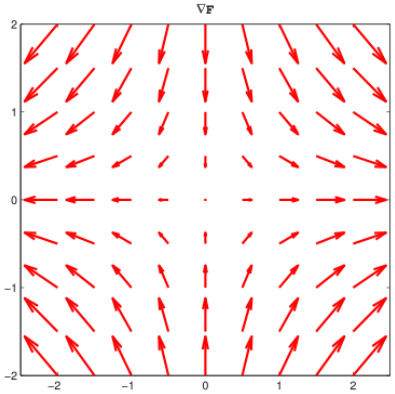
Gradient points towards the direction of function's fastest increase and has the length proportional to the function derivative in that direction











Gradient

Example of gradient calculations:

$$\nabla e^{-(x^2+y^2)/2} = -e^{-(x^2+y^2)/2}\begin{bmatrix} x \\ y \end{bmatrix}$$

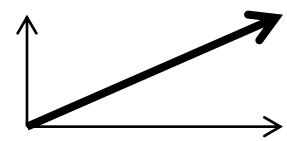
$$\nabla(x^2+y^2) = \begin{bmatrix} 2x \\ 2y \end{bmatrix}$$

$$\nabla(xy) = \begin{bmatrix} y \\ x \end{bmatrix}$$

$$\nabla e^{x^2-y^2} = e^{x^2-y^2}\begin{bmatrix} 2x \\ -2y \end{bmatrix}$$

$$\nabla f = [\partial f / \partial x, \partial f / \partial y]$$

$\partial f / \partial y$ Rate of growth in y



Rate of growth in x

$\partial f / \partial x$

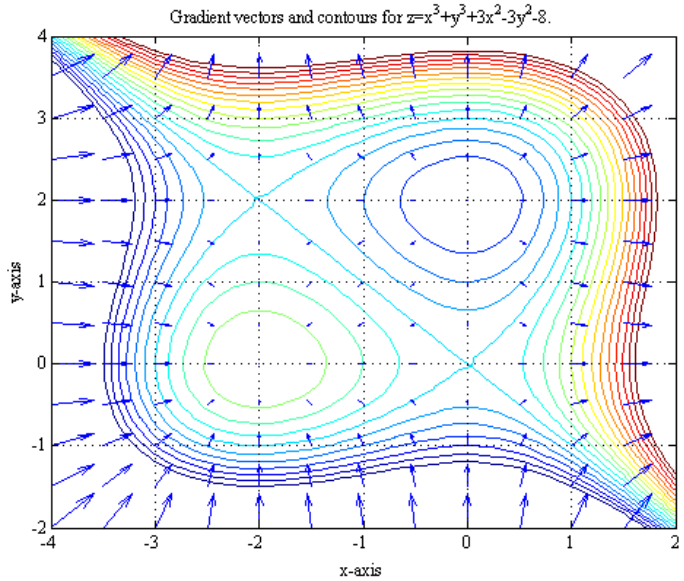
Directional derivative

$$\Delta f = \nabla f^T \cdot \Delta x = \sum_i \frac{\partial f(x)}{\partial x_i} \Delta x_i = |\nabla f| \, |\Delta x| \cos\varphi$$

$$\Delta f = \nabla f^T \cdot \Delta x = |\nabla f| \, |\Delta x|, \ \nabla f \uparrow\uparrow \Delta x$$

$$\Delta f = \nabla f^T \cdot \Delta x = -|\nabla f| \, |\Delta x|, \ \nabla f \uparrow\downarrow \Delta x$$

$$\Delta f = \nabla f^T \cdot \Delta x = 0, \ \nabla f \perp \Delta x \ \ (ie \ \varphi = \frac{\pi}{2})$$

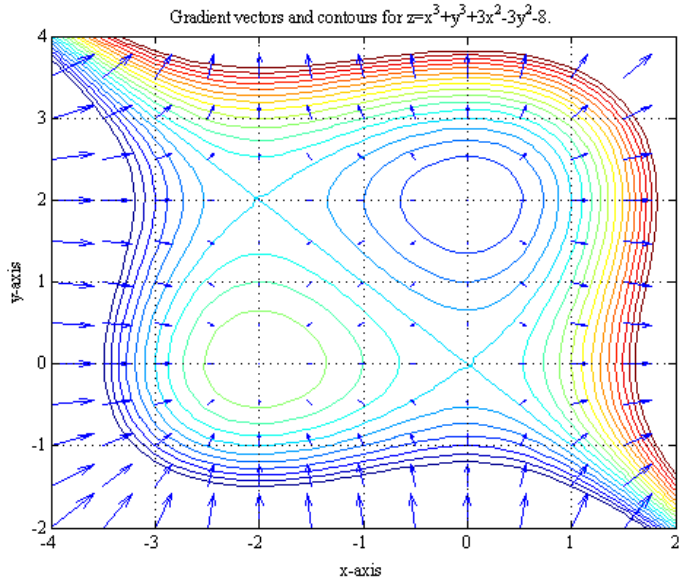


Level surface and the gradient

$$\Delta f = \nabla f^T \cdot \Delta x = \sum_i \frac{\partial f(x)}{\partial x_i} \Delta x_i = |\nabla f\,||\Delta x|\cos\varphi$$

Level surface is a surface S in p-dimensional parameter space where $f(x)$=const

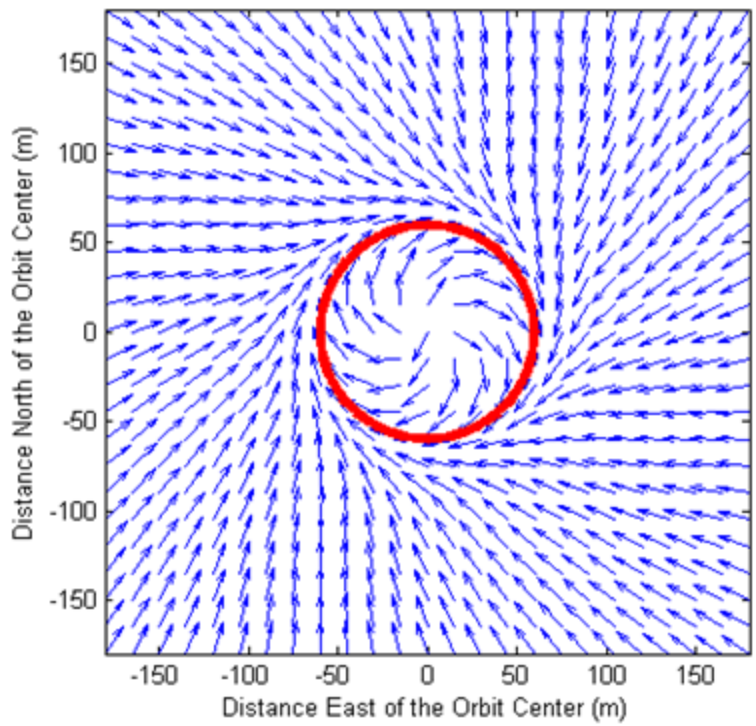
Gradient of a function is perpendicular to all its level surfaces – see figure to the right !



This is because must have

$\Delta f = \nabla f^T \cdot \Delta x = 0 \Rightarrow \nabla f \perp \Delta x$ for all displacements Δx within a level surface.

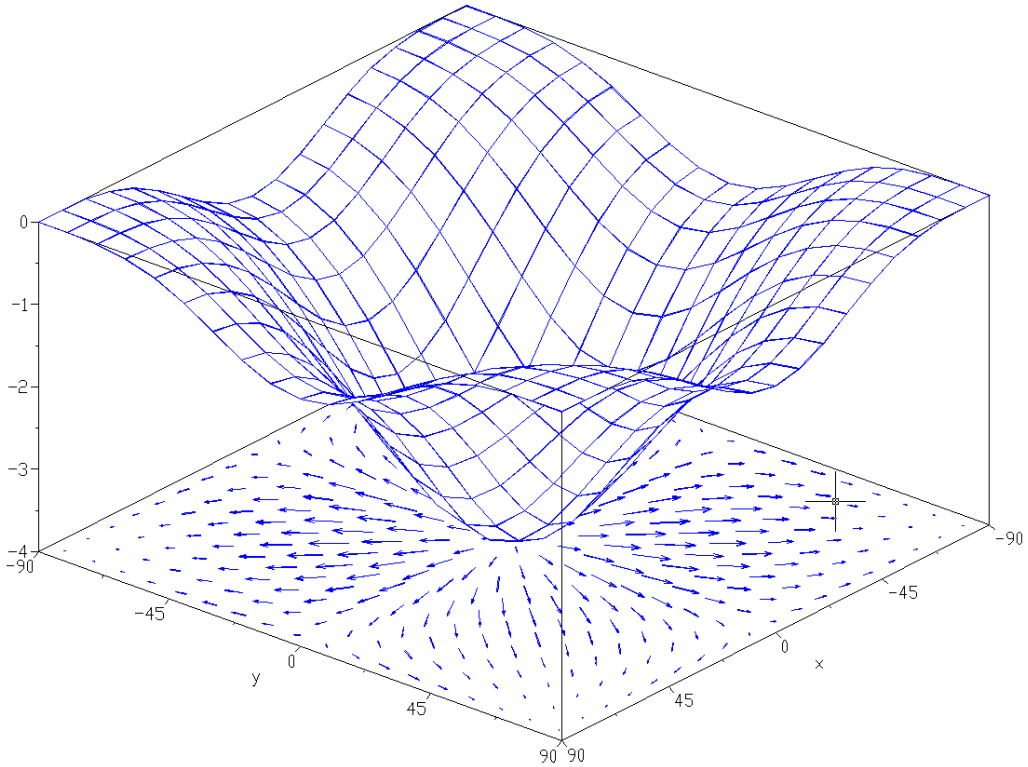
$$\Delta f = \nabla f^T \cdot \Delta x = \frac{\partial f(x)}{\partial x_1}\Delta x_1 + \frac{\partial f(x)}{\partial x_2}\Delta x_2 + \ldots + \frac{\partial f(x)}{\partial x_p}\Delta x_p$$



← DESCRIBE THIS FUNCTION FROM ITS *GRADIENT*
(*Is this even a function* ???)

Gradient descent

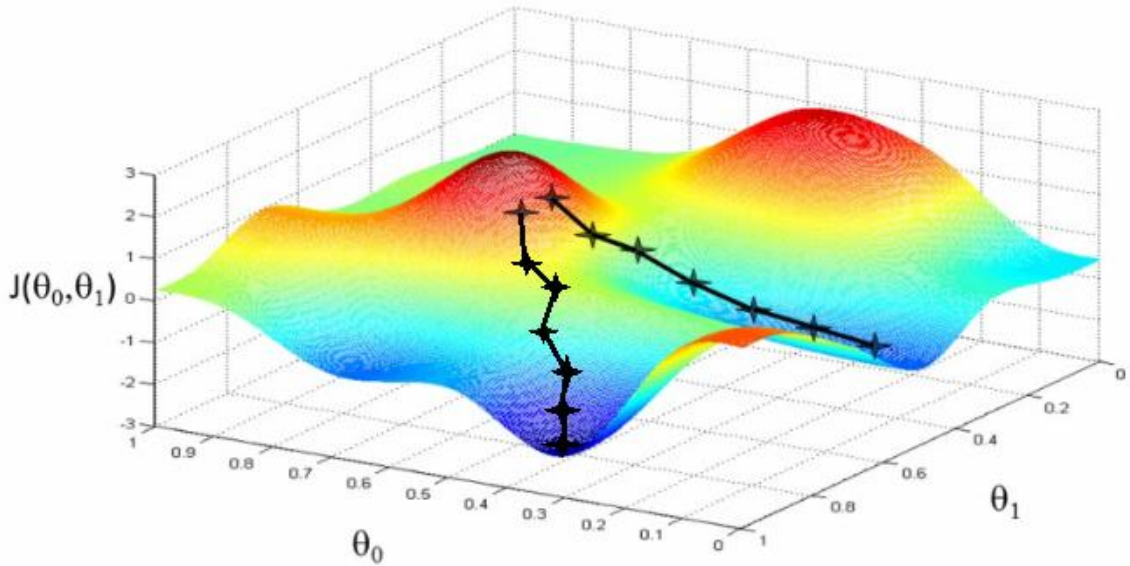
Anti-gradient is the vector exactly opposite to gradient, $-\nabla f$: follow the *anti-gradient* to reduce the value of a function (that is - descent !)



Gradient descent

Gradient Descent (GD) method:

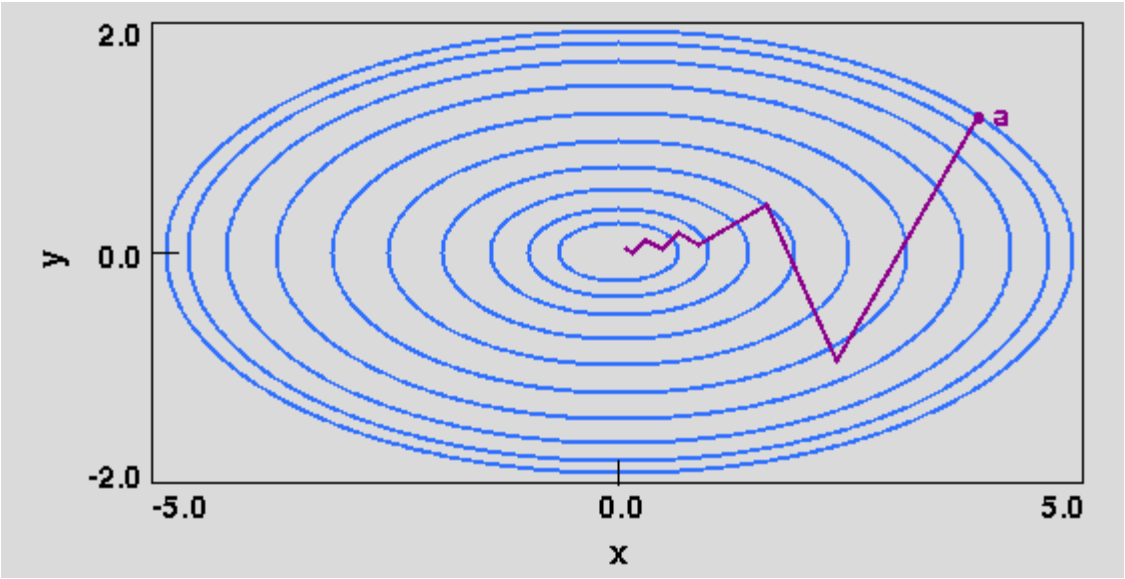
- Follow anti-gradient continuously until $-\nabla f$ turns 0, then no further descent is possible



Gradient descent

Gradient Descent (GD) method:

- ~~Follow anti-gradient continuously until $-\nabla f = 0$~~
- In practice – sequence of finite steps

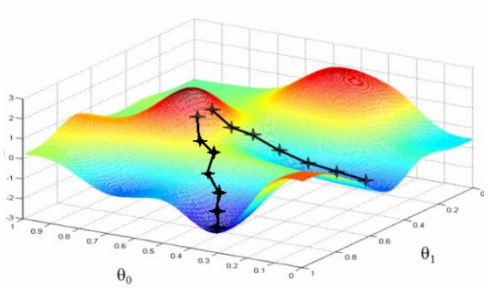


Gradient descent

Gradient descent algorithm with <u>learning rate</u>

$$\text{REPEAT till convergence}$$

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



Gradient descent algorithm with <u>line search</u>

$$\text{REPEAT till convergence}$$

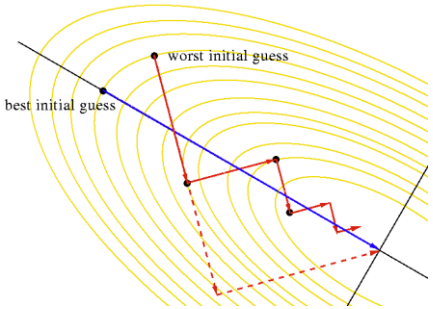
$$\text{set } \mathrm{p}_k = -\nabla f(x_k)$$

$$\text{set } g_k(x) = p_k^{\ T} \cdot \nabla f(x)$$

$$\text{REPEAT until } g_k(x) = 0 \text{ (Secant method)}$$

$$x_k^{i+1} = x_k^i - \frac{g_k(x_k^i)(x_k^i - x_k^{i-1})}{g_k(x_k^i) - g_k(x_k^{i-1})}$$

$$\text{set } x_{k+1} = x_k^{i.end}$$



Newton method

- **Newton method** is numerical optimization method that uses the Newton nonlinear equation solver to solve the nonlinear optimality conditions $\nabla f(x)=0$ directly
- **NM advantage against GD is dramatically faster convergence**
- The key disadvantage is the **need to invert large matrices** and **supply 2nd order derivatives** of objective function, in addition to the gradient

Hessian matrix

$$Hf = \begin{bmatrix} \frac{\partial^2 f(x)}{\partial x_1{}^2} & \frac{\partial^2 f(x)}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f(x)}{\partial x_1 \partial x_p} \\ \frac{\partial^2 f(x)}{\partial x_1 \partial x_2} & \frac{\partial^2 f(x)}{\partial x_2{}^2} & & \frac{\partial^2 f(x)}{\partial x_2 \partial x_p} \\ \cdots & & & \\ \frac{\partial^2 f(x)}{\partial x_1 \partial x_p} & \frac{\partial^2 f(x)}{\partial x_2 \partial x_p} & \cdots & \frac{\partial^2 f(x)}{\partial x_p{}^2} \end{bmatrix}$$

Newton method

Approximate optimality condition:

(nonlinear) *(linear)*

$$\nabla f(x_0 + \Delta x) = 0 \xrightarrow{\text{approximately}} \nabla f(x_0) + Hf(x_0)\Delta x \approx 0$$

In the future we will write instead

$$g_k + H_k \Delta x_k = 0$$

Remember this notation: $g_k = \nabla f(x_k),\ H_k = Hf(x_k)$

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

Newton method

Newton numerical optimization algorithm

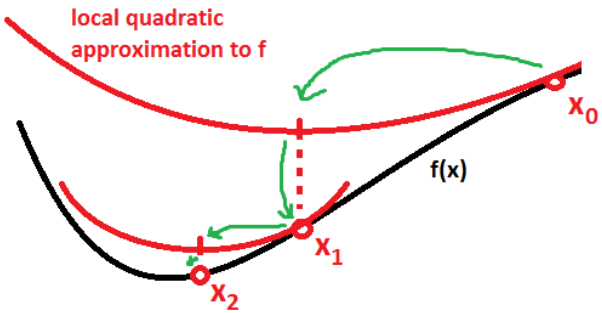
REPEAT intil converge

$$g_k = \nabla f(x_k)$$

$$H_k = Hf(x_k)$$

$$\Delta x_k = -H_k^{-1} g_k$$

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



Newton method works by constructing a series of local quadratic approximations to the objective and minimizing those analytically

Newton method

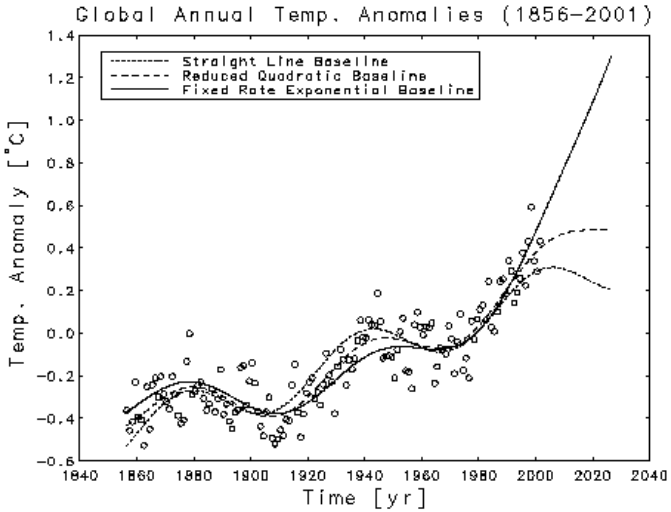
Because NM uses information about the curvature of the objective, it is dramatically faster than GD that only relies on the derivative (gradient) for next guess at the minimum

Newton method

- Because of the matrix inversion, NM is more computationally intensive than GD – O(p) vs $O(p^3)$ – which makes it very costly on large-scale optimization problems in fact
- NM may also suffer from peculiar behaviors of H and numerical instabilities of inversion H^{-1} in high dimensions
- GD is significantly more stable and well behaved in large scale problems

Levenberg-Marquardt Method

LM method is the Newton method
for Nonlinear Least Squares:



Nonlinear Least Squares Loss Function

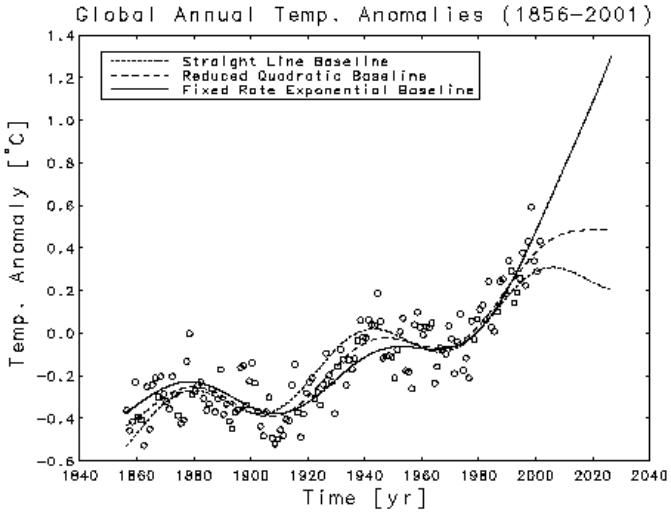
$$S(\theta) = \frac{1}{2}\sum_i \left(y_i - f(x_i;\theta)\right)^2$$

NLS gradient

$$\nabla S(\theta) = \sum_i g_i \cdot \left(f(x_i;\theta) - y_i\right)$$

Levenberg-Marquardt Method

LM method is the Newton method
for Nonlinear Least Squares:



Newton update

$$\sum_i g_i g_i^T \Delta\theta = \sum_i g_i \cdot (y_i - f(x_i;\theta))$$

Regularization or dampening (the Levenberg-Marquardt part)

$$\left[\sum_i g_i g_i^T + \lambda \operatorname{diag}\left(\sum_i g_i g_i^T\right)\right]\Delta\theta = \sum_i g_i \cdot (y_i - f(x_i;\theta))$$

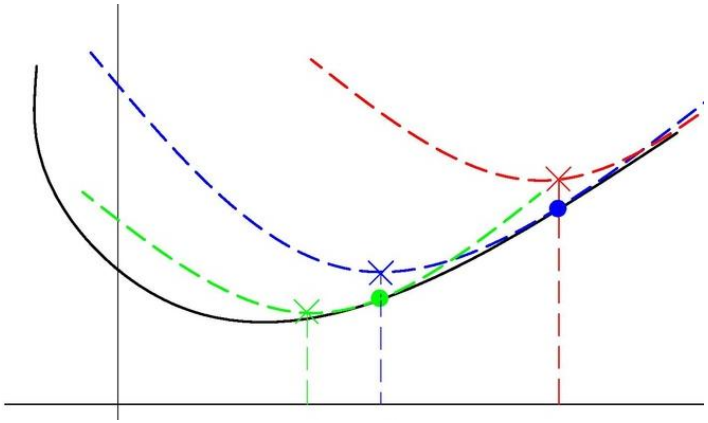
Quasi-Newton methods

- One of the huge downsides of NM is the requirement to have analytically the Hessian matrix, which may be very large (p^2) and difficult to construct, if at all possible
- **Quasi-Newton optimization methods** deal with this problem by maintaining an approximate representation of the Hessian's inverse H^{-1}
- The time complexity of QNM generally is $O(p^2)$ instead of $O(p^3)$ of original NM

Quasi-Newton methods

The concept of the model of objective function:

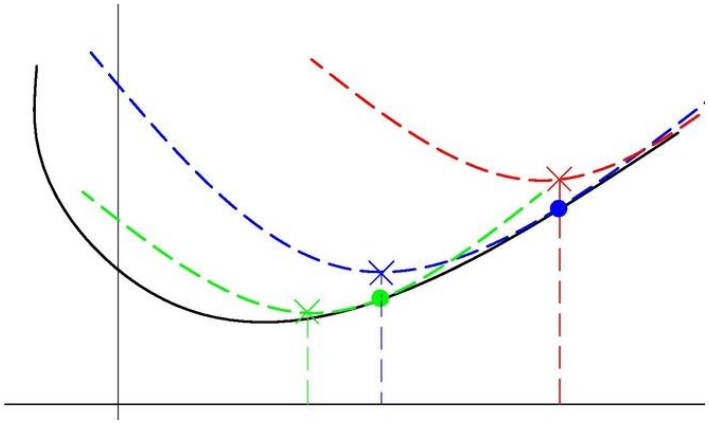
$$f(x) \approx m_k(x_k + p) = f_k + p^T g_k + \frac{1}{2} p^T B_k p$$



Quasi-Newton methods

Optimization step proceeds by minimizing the objective model $m_k(p)$ instead of $f(x)$

$$\min f(x) \rightarrow \min m_k(p)$$



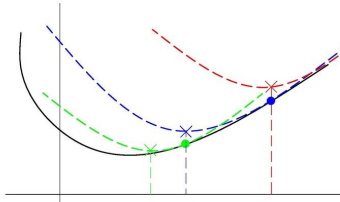
$$m_k(x_k + p) = f_k + p^T g_k + \frac{1}{2} p^T B_k p$$

Quasi-Newton methods

B_k is chosen by various techniques to mimic the Hessian property:

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

(more details in Advanced)



Trust-region methods

- Since we base our selection of the next minimum guess point $x_{k+1}=x_k+\Delta x_k$ on an **approximate model of the objective function** $m_k(x)$, we cannot expect such selection to be overly precise
- In fact, it is not a good idea to trust the model and its predictions too much generally; by doing so we may end up wasting significant computational resources just to be finding accurate predictions of inaccurate model

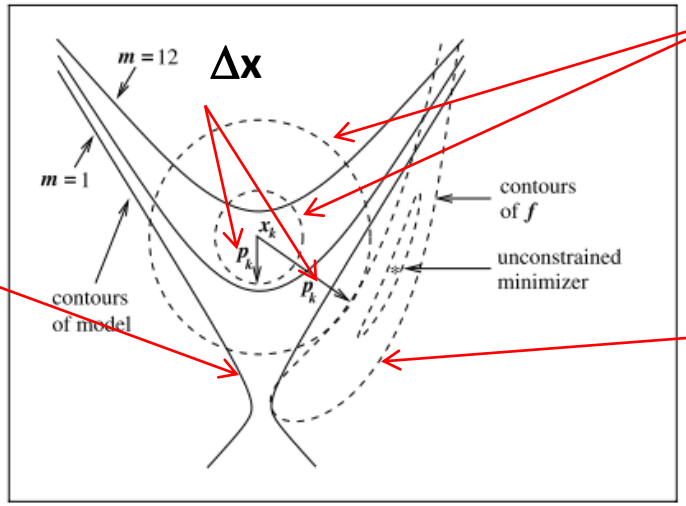
Trust-region methods

In short, Δx_k-search must be restricted to the part of the parameter space where the **model objective m_k can be trusted** to be accurate

$$\boxed{|\Delta x_k| \leq R \quad \leftarrow \textbf{Trust-region radius}}$$

Possible trust regions

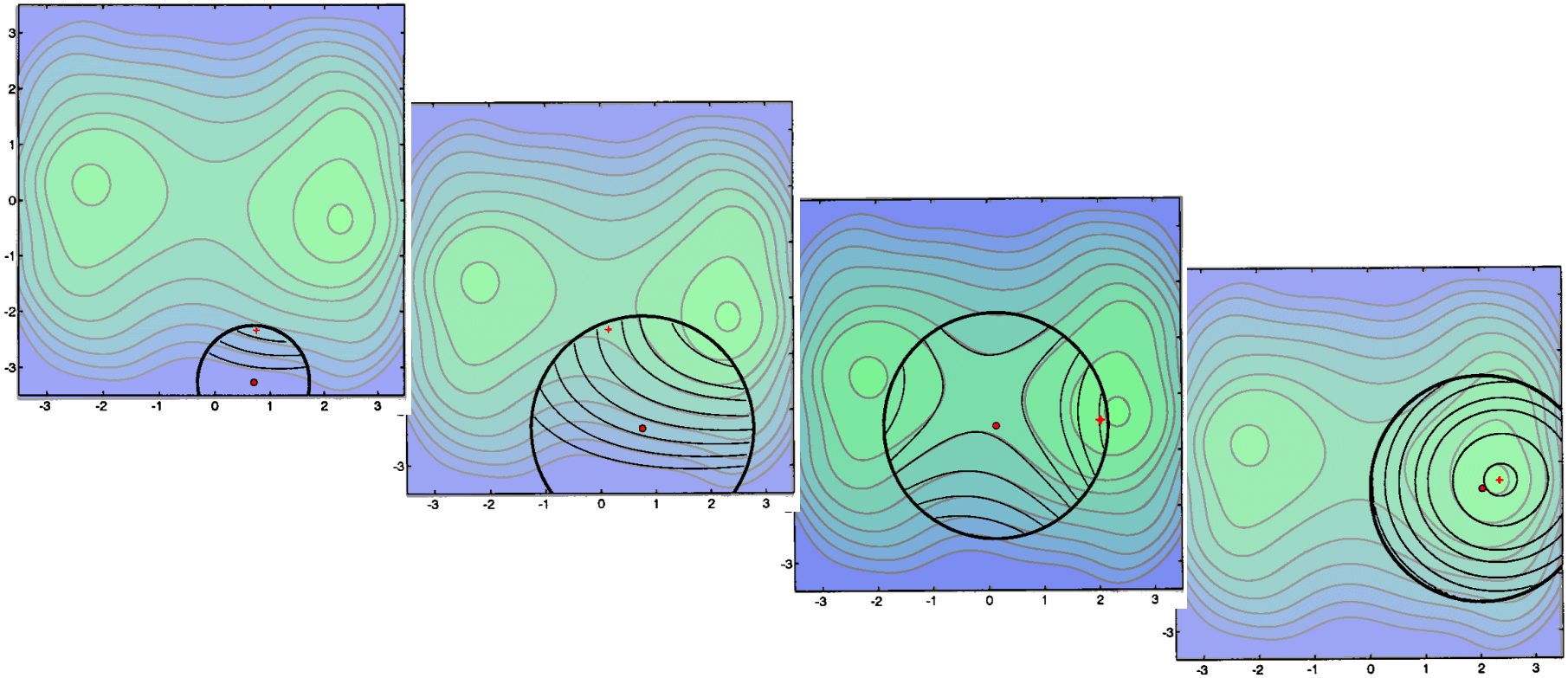
Model objective *m*



Real objective *f*

Trust-region methods

Example of an objective function optimization using trust region method:



Trust-region methods

- How to choose the trust-region: given step Δx_k, calculate the **<u>Actual Reduction Ratio</u>**

$$r_k = \frac{f(x_k) - f(x_k + \Delta x_k)}{m_k(0) - m_k(\Delta x_k)}$$

- The trust-region radius is kept constant or increased by a factor $\alpha>1$ if r_k is positive and close to 1; if r_k is close to zero or negative, trust region is shrank by a factor $\alpha^{-1}<1$ and **the step Δx_k is <u>re-done</u>**.

$$R_{k+1} \leftarrow \alpha^{-1} R_k \mid \alpha R_k$$

Trust-region methods

The step in the trust-region optimization methods is constrained to be within the trust-region:

$$\min\left(g_k{}^T p + \frac{1}{2} p^T B_k p \right)$$

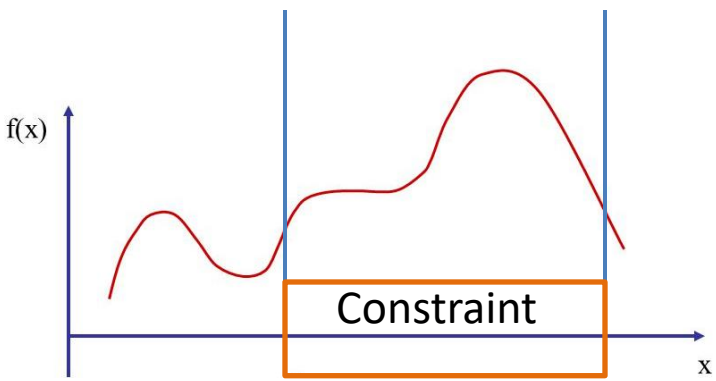
$$s.t. |\, p \,| < R_k$$

This constraint can affect both the size and the direction of the step

BASICS OF CONSTRAINED OPTIMIZATION

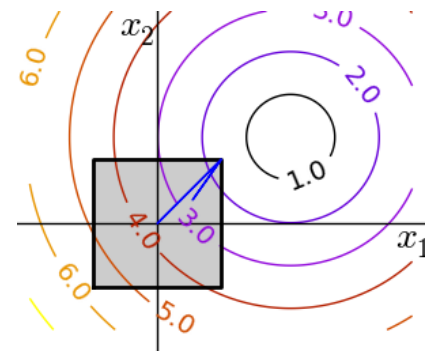
Constrained optimization

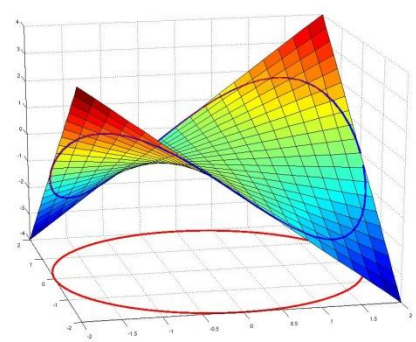
Unconstrained vs. constrained optimization



$$a \le x \le b$$

$$-a \le x_1 \le a, -b \le x_2 \le b$$





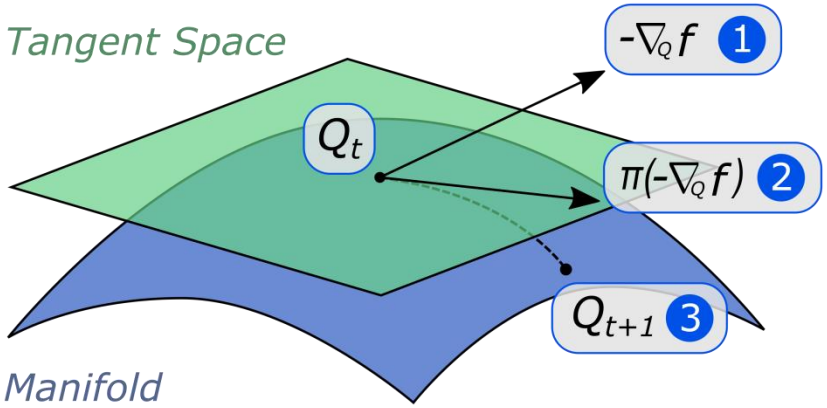
$$x_1^2 + x_2^2 \le r^2$$

Constrained optimization

Constrained optimization on manifolds (equality constraints):

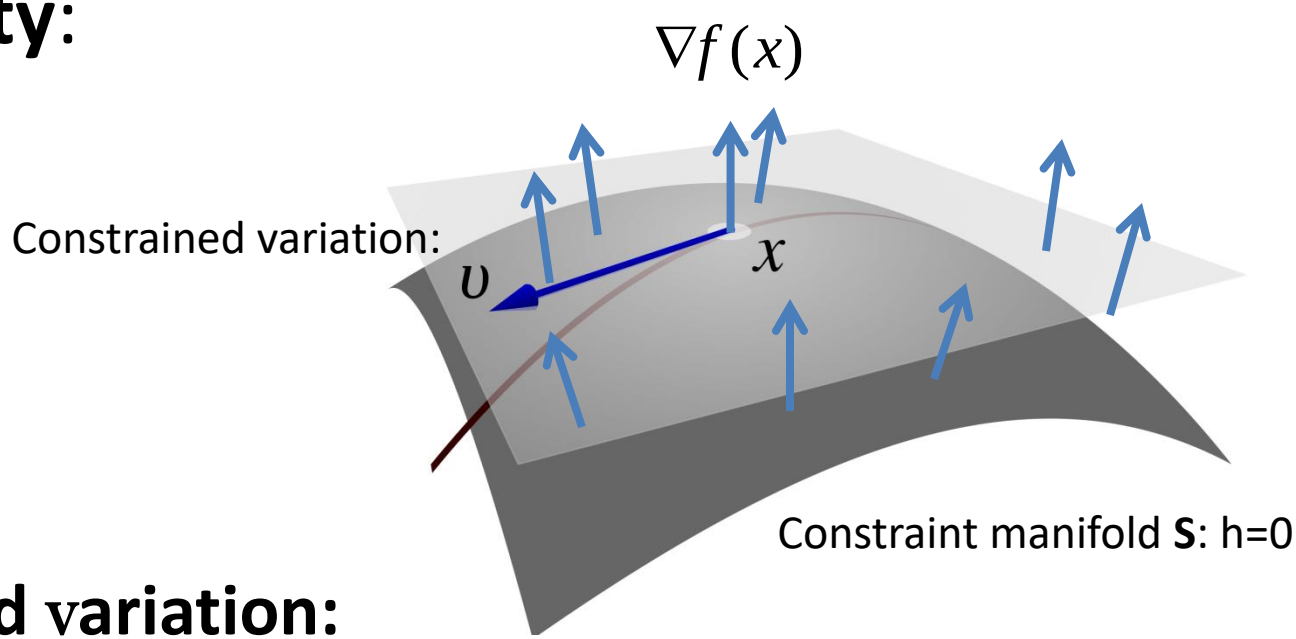
$$\min f(x)$$
$$s.t.$$
$$h_i(x) = 0, i = 1..q$$

Each equality constraint typically means a reduction of 1 degree of freedom from the parameter space producing a surface in $x_{1..p}$



Constrained optimization

Constrained optimization on manifolds – **optimality**:



<u>Constrained variation:</u>

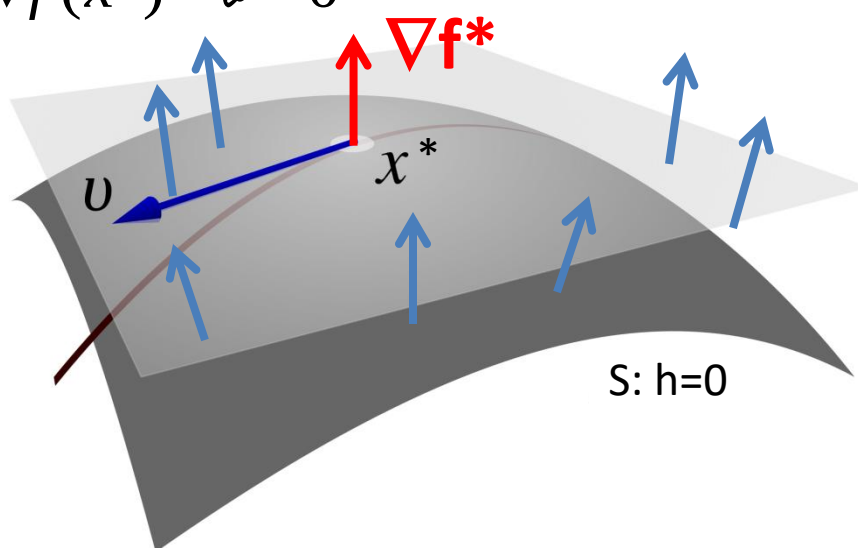
any υ s.t. $\Delta h = \nabla h(x)^T \cdot \upsilon = 0$ (that is υ is inside the constraint manifold)

must also result in $\Delta f = \nabla f(x)^T \cdot \upsilon \to 0$

Constrained optimization

Geometrically, ∇f must be perpendicular to the constraint's manifold at the of maximum or minimum – otherwise shifting along manifold in the direction of ∇f can be used to lower f

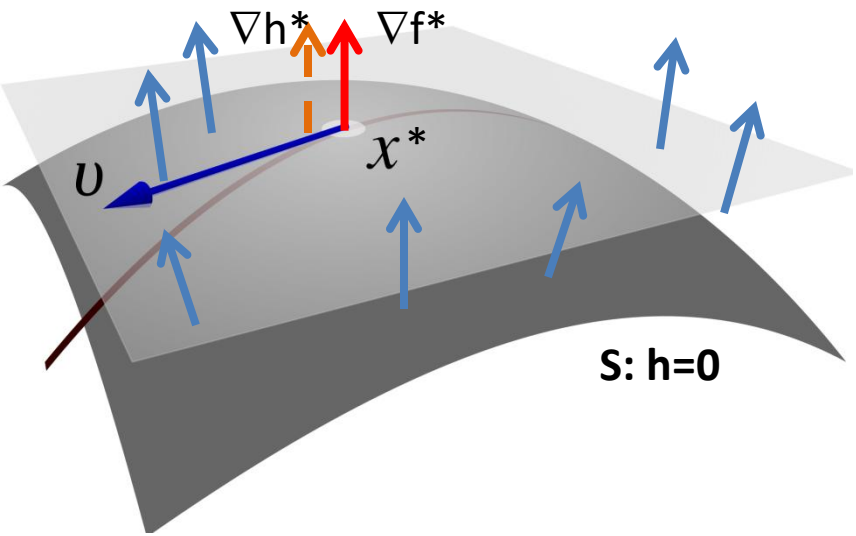
$$\text{any } u: \Delta h = \nabla h(x^*)^T \cdot u = 0 \Rightarrow \Delta f = \nabla f(x^*)^T \cdot u = 0$$



Constrained optimization

View the constraint as the **level surface of h(x)=0**. Then the direction perpendicular to S is ∇h, and we can say that ∇f must be parallel to ∇h, or in other words proportional to it:

$$\nabla f(x) = \lambda \nabla h(x)$$



Constrained optimization

Formalization – **Lagrange function *L*:**

These are called the **Lagrange multipliers**

$$L(x,\lambda) = f(x) + \sum_{i=1}^{q} \lambda_i h_i(x)$$

$$\min_x f(x) \quad \rightarrow \quad \underset{x,\lambda}{\text{extreme}}\, L(x,\lambda)$$

$$s.t.$$

$$h_i(x) = 0$$

Constrained optimization

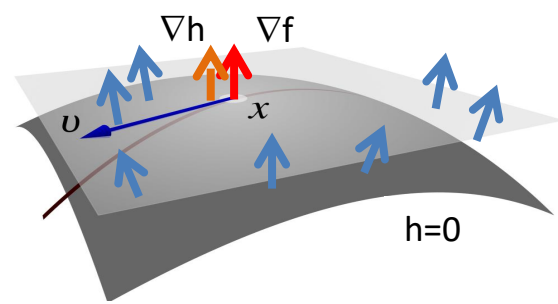
Write down the optimality conditions for *L* with respect to x and λ and make sure you get what we want:

$$L(x,\lambda) = f(x) + \sum_{i=1}^{q} \lambda_i h_i(x)$$

$$\text{extreme}\, L(x,\lambda):$$

$$\frac{\partial L}{\partial x} = 0$$

$$\frac{\partial L}{\partial \lambda} = 0$$



Note the solutions are the extreme points of L, not minimum nor maximum, because λ-terms in λh(x) <u>are linear in λ</u>

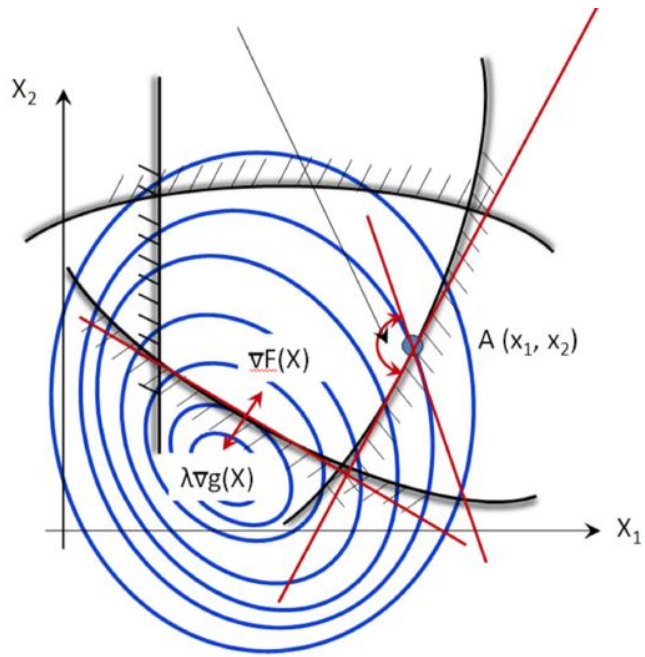
Constrained optimization

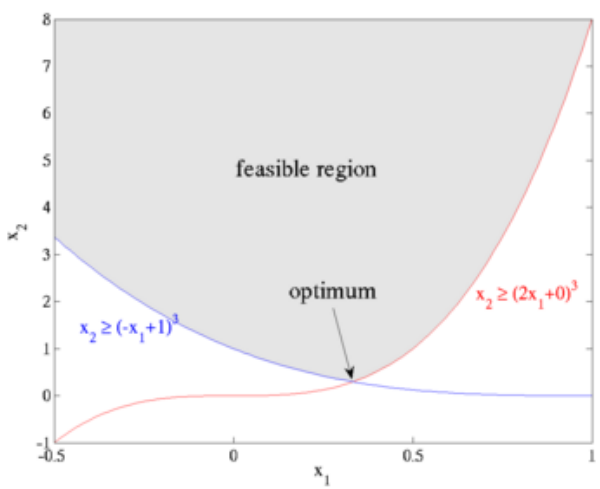
Optimization on sub-regions (inequality constraints)

$$\min f(x)$$

$$s.t.$$

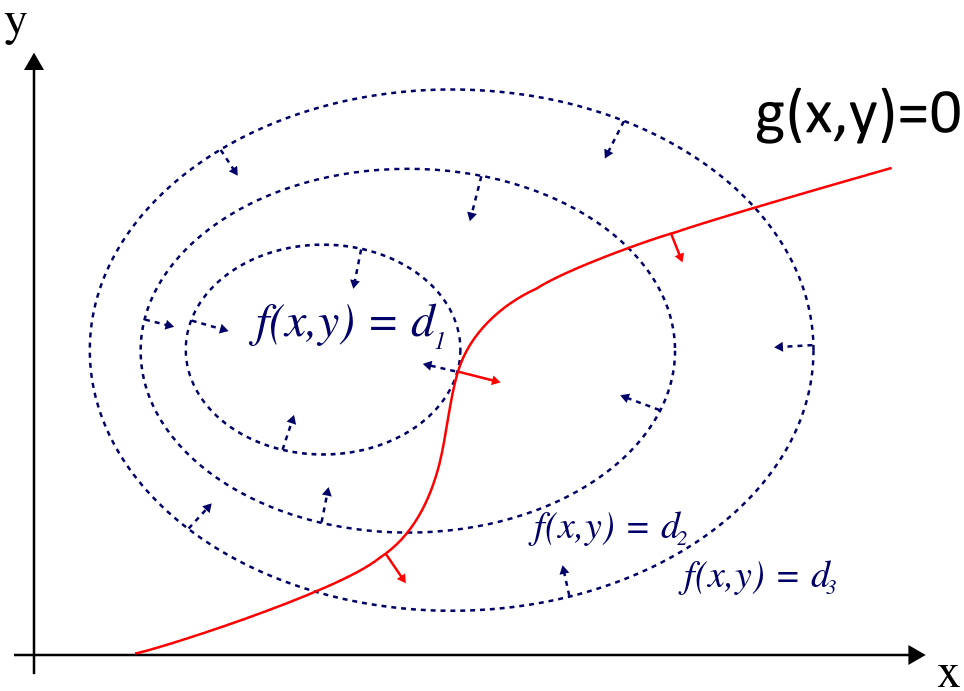
$$g_i(x) \le 0, i = 1..r$$





Constrained optimization

Optimization on sub-regions – optimality:

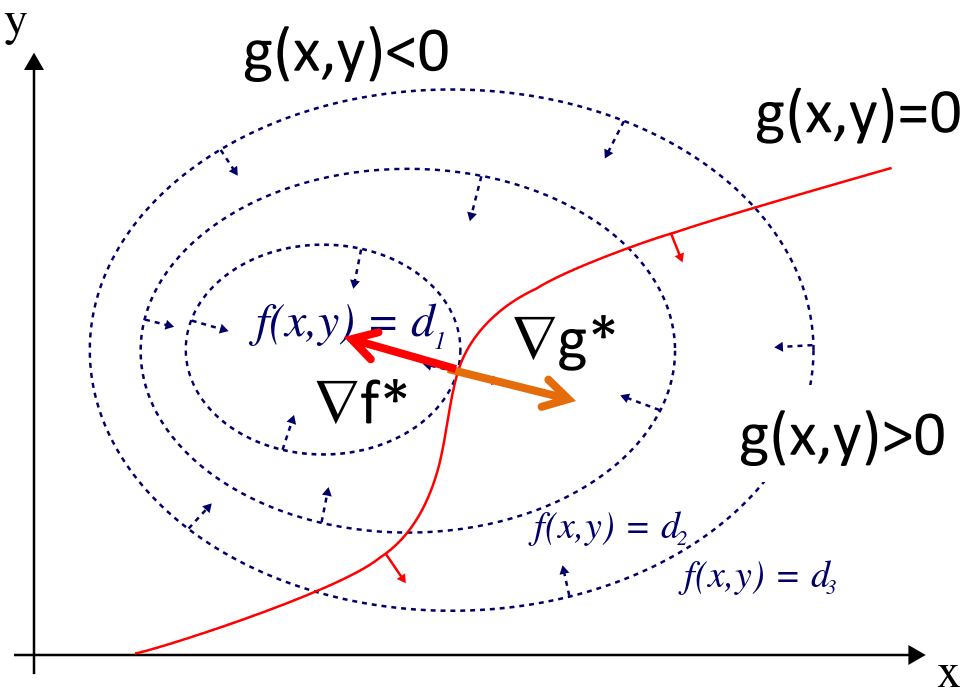


If optimal point is inside region $g(x)<0$, then opt. cond. are same as for unconstrained optimization…

If optimal point is on the boundary $g(x^*)=0$, then opt. cond. can be described by optimization on manifolds earlier, that is – must have ∇f perpendicular to the level surface $g(x)=0$

Constrained optimization

Optimization on sub-regions – optimality:



Additional condition must be met that f **is unable to decrease inwards the region** g(x)<0, since otherwise we can move away from the boundary inwards g(x)<0 and lower f(x)!

This implies $\nabla g = -\mu \nabla f$ with $\mu \geq 0$: note in the manifold constrained case there were no restrictions on the values of λ because it was forbidden to move away from the constraint's surface at all. Here we just can't move outside, so $\mu \geq 0$.

Constrained optimization

Formalization – **generalized Lagrange function**:

$$L(x,\lambda,\mu) = f(x) + \sum_{i=1}^{q} \lambda_i h_i(x) + \sum_{i=1}^{r} \mu_i g_i(x)$$

Lagrange multipliers

Optimality conditions:

$$\nabla_x L(x,\lambda,\mu) = \nabla f(x) + \sum_{i=1}^{q} \lambda_i \nabla h_i(x) + \sum_{i=1}^{r} \mu_i \nabla g_i(x) = 0$$

Additionally must have $\mu_i \geq 0$

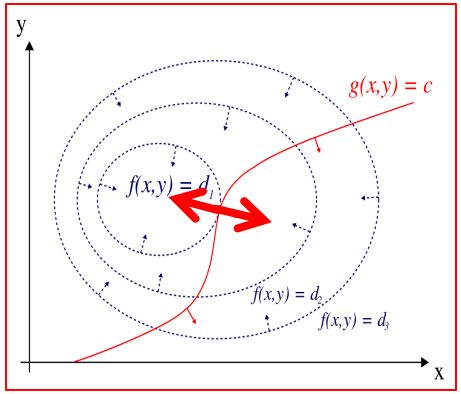
Karush-Kuhn-Tucker (KKT) equations

$$\nabla_x L(x^*, \lambda^*, \mu^*) = \nabla f(x^*) + \sum_{i=1}^{q} \lambda_i^* \nabla h_i(x^*) + \sum_{i=1}^{r} \mu_i^* \nabla g_i(x^*) = 0$$

$$\left.\begin{aligned} g_i(x^*) &\le 0, i = 1..r \\ h_i(x^*) &= 0, i = 1..q \end{aligned}\right\} \quad \textbf{primal feasibility}$$

$$\mu_i^* \ge 0, i = 1..r \quad \textbf{dual feasibility}$$

$$\mu_i^* g_i(x^*) = 0, i = 1..r \quad \textbf{complementary slackness}$$



Complementary slackness essentially selects either g(x)=0 (the boundary, then must have μ>0) or g(x)<0 (the interior, then μ=0 and minimum is unconstrained). Effectively, this is a 0-1 switch for selecting boundary-vs-interior in the KKT equations !

Karush-Kuhn-Tucker (KKT) equations

The **standard form** of constrained optimization problems:

$$\min f(x)$$

must be <u>min</u>

$$s.t.$$

$$h_i(x) = 0, i = 1..q$$

must be <u>=0</u>

$$g_i(x) \le 0, i = 1..r$$

must be <u>≤0</u>

If your problem has objective or constraints in any different form (for example max f(x), g(x)≥0, or h(x)=a), **<u>it must be converted to the standard form to use KKT</u>.**

Constrained optimization

Example:

$$\min 2x^2 + 4x + 2y^2 - 2y + 3$$
$$s.t.\ x \geq 0, y \geq 0, x + y \leq 1$$

$$\begin{cases} \nabla_x L(x^*, \lambda^*, \mu^*) = \nabla f(x^*) + \sum_{i=1}^{q} \lambda_i^* \nabla h_i(x^*) + \sum_{i=1}^{r} \mu_i^* \nabla g_i(x^*) = 0 \\ \left.\begin{array}{l} g_i(x^*) \leq 0, i = 1..r \\ h_i(x^*) = 0, i = 1..q \end{array}\right\} \text{primal feasibility} \\ \mu_i^* \geq 0, i = 1..r \quad \text{dual feasibility} \\ \mu_i^* g_i(x^*) = 0, i = 1..r \quad \text{complementary slackness} \end{cases}$$

QUESTIONS FOR SELF-CONTROL

- Describe the main idea of the gradient descent (GD) method.
- What is the difference between GD algorithm with learning rate and line search?
- Explain the geometric interpretation of the formula $\Delta f=\nabla f\cdot\Delta x=|\nabla f||\Delta x|\cos(\varphi)$.
- Define the level surface of a function and describe its relationship with the function's gradient?
- Give interpretation of the gradient field shown in slide 10.
- Define the function's Hessian.
- Calculate the Hessian of $f(x,y,z)=2x^2+y^2+4z^2-4xy+2yz+5x-3y+7$.
- Calculate the Hessian of $f(x,y)=x^4y^2-2x^2y^3+7y$.
- Calculate and express in matrix notation the Hessian of the general quadratic form x^TQx+g^Tx, where x and g are px1 column vectors and Q is pxp symmetric square matrix.
- Calculate and express in matrix notation the Hessian of the general Gaussian $\exp(-(x-g)^TQ(x-g))$, where x, g and Q are as in the previous question.

- Write down and explain the steps of the Newton method for numerical optimization.
- Describe what Levenberg-Marquardt optimization method is.
- What is the difference between Newton and quasi-Newton numerical optimization methods?
- Explicitly evaluate $B_{k+1}s_k$ in slide 29 and show that it equals y_k.
- Explain the ideas behind the trust regions in numerical optimization.
- Give definition of the trust region radius.
- Write down the general unconstrained optimality conditions by using only the symbol of the gradient of the objective function ∇f.
- Explain how the optimality condition $\nabla f=0$ changes if optimization is constrained to take place on a surface h(x)=0.
- Explain how the optimality condition $\nabla f=0$ changes if optimization is constrained to take place inside a region $g(x)\leq 0$.
- Write down the general KKT equations.
- Define the standard form of constrained optimization problems for KKT.

- What is the Lagrange function in constrained optimization?
- What are the Lagrange multipliers in constrained optimization?
- Derive the optimality conditions for Lagrange function in slide 45.
- Describe what feasibility region is.
- Identify primal feasibility conditions in KKT equations.
- Identify dual feasibility conditions in KKT equations.
- Identify the complementary slackness conditions in KKT equations.
- Write down the Lagrange functions for the following constrained optimization problems:

$$\min 2x^2 + 2y^2 + 4x - 2y$$
$$s.t.\ 2x - y = 2$$

$$\min 2x^2 + 2y^2 + 4x - 2y$$
$$s.t.\ x \geq 0, y \geq 0, x + y \leq 1, x - y = 0$$

$$\min \exp(-x^2 - y^2 - xy)$$
$$s.t.\ x^2 + y^2 = 2, x \geq 0, y \leq 0$$

$$\min\left[(x + y) \cdot \exp(-|x| - |y|)\right]$$
$$s.t.\ xy = 2, x + y \leq 0$$

ADVANCED

Quasi-Newton methods

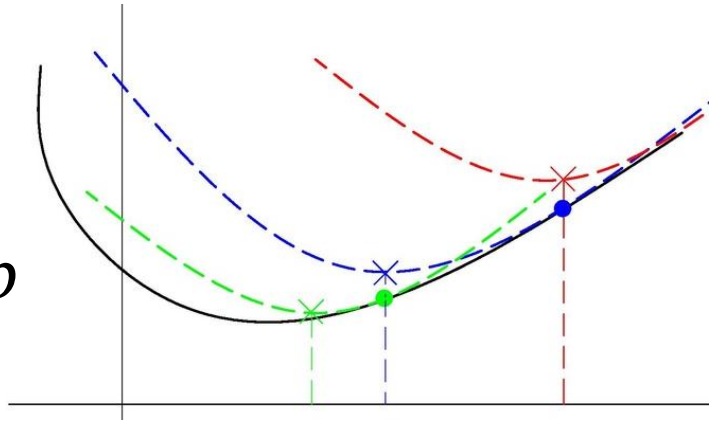
- One of the problems of NM in applications is the requirement to supply analytically the Hessian matrix of objective function, which may be large (p^2), difficult to construct, and expensive to compute
- Quasi-Newton methods work around this problem by keeping an approximate numerical representation of the Hessian H and/or its inverse H^{-1}

Quasi-Newton methods

QNM use a quadratic model for the objective with the true gradient g_k but model Hessian B_k

$$\min f(x) \rightarrow \min m_k(x_k + p)$$

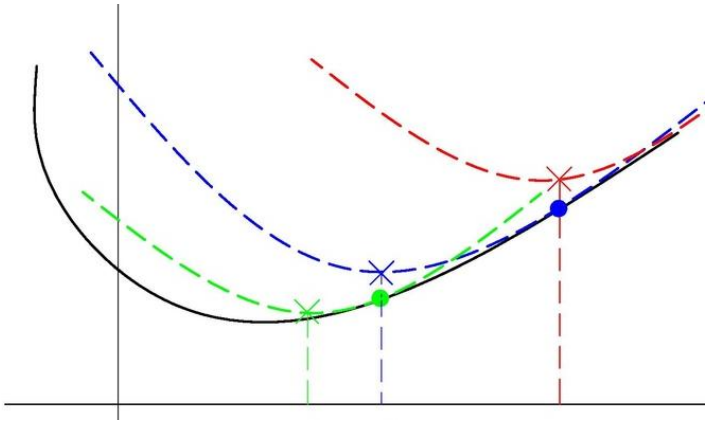
$$m_k(x_k + p) = f_k + p^T g_k + \frac{1}{2} p^T B_k p$$



Quasi-Newton methods

The optimization step then proceeds either by minimizing the objective function model …

$$m_k(x_k + p) = f_k + p^T g_k + \frac{1}{2} p^T B_k p$$

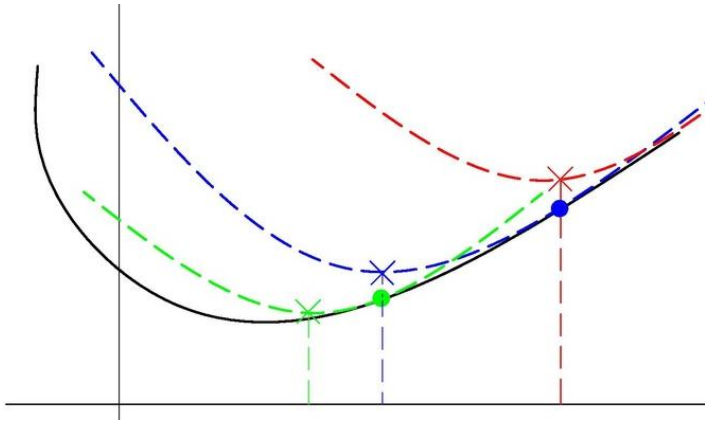


Exact Newton step:

$$p_k = -B_k^{-1} g_k$$

Quasi-Newton methods

… or by line-search in the direction defined by p_k. The idea behind that is that we don't expect the model to be exact, so why rely on the exact minimum of m_k, p_k, then?



Line search:

$$\min_a f(x_k + a\hat{p}_k), \hat{p}_k = -B_k^{-1} g_k$$

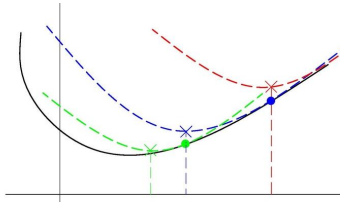
Quasi-Newton methods

How to choose B_k?

$$B_k \approx Hf(x_k)$$

We want B_k to mimic the Hessian for at least one last step:

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

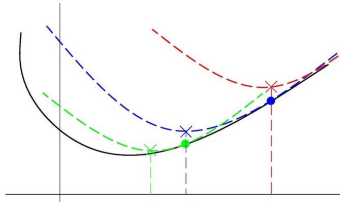


Quasi-Newton methods

We therefore require directly

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

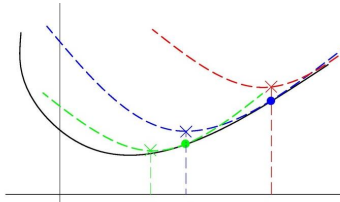
Mathematically, this is an underdetermined system – just p equations for p^2 matrix $B_k \Rightarrow$ thus much freedom exists in choosing B_k !



Quasi-Newton methods

the BFGS (Broyden, Fletcher, Goldfarb, Shanno) formula is one of the most successfull methods for updating B_k

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k}$$

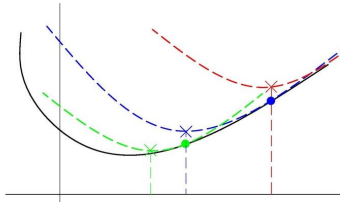


Quasi-Newton methods

Here we define as is conventional for quasi-Newton literature

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

$$y_k = \nabla f(x_{k+1}) - \nabla f(x_k) \qquad (ie\ \Delta g_k)$$



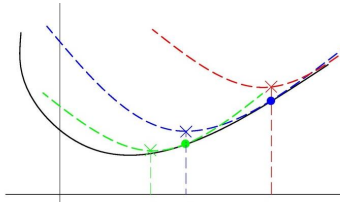
Quasi-Newton methods

The BFGS (Broyden, Fletcher, Goldfarb, Shanno) formula then yields the condition $y_k = B_{k+1} s_k$:

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k}$$

$$\text{from } \; y_k = B_{k+1} s_k$$

← **VERIFY THIS**

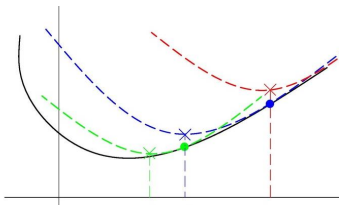


Quasi-Newton methods

While Bk is an approximation to the Hessian, it is further possible to invert B_k analytically, by using the so called Woodbury lemma (inverse of Hessian is needed for calculating p_k) (***note the confusion in the notation here regarding H_k – this is how these things are typically written in QNM literature***):

$$H_{k+1} = (I - \rho_k s_k y_k^T) H_k (I - \rho_k s_k y_k^T) + \rho_k s_k s_k^T$$

$$\rho_k = \frac{1}{y_k^T s_k}, \text{ here } H_k = B_k^{-1} \approx Hf(x_k)^{-1}!$$



Quasi-Newton methods

Observe that BFGS update does not involve explicit inversion of B_k, therefore, it costs at most $O(p^2)$ flops – a very significant saving over direct NM for large p!

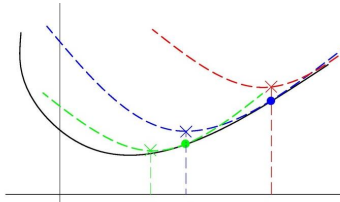
$$\text{QuasiNewton Algorithm:}$$

$$\hat{p}_k = -H_k g_k$$

$$x_{k+1} \leftarrow \min_a f(x_k + a\hat{p}_k)$$

$$y_k = g_{k+1} - g_k, s_k = x_{k+1} - x_k, \ \rho_k = (y_k^T s_k)^{-1}$$

$$H_{k+1} = (I - \rho_k s_k y_k^T) H_k (I - \rho_k s_k y_k^T) + \rho_k s_k s_k^T$$



The principle of exceeding precision

If a model or its input data are only accurate up to a given precision (for example 0.1), it is not meaningful to calculate the predictions or the outputs of the model with greater precision (for example 0.001)