Tutorial

Nonlinear Optimization

F. Javier Heredia

(heredia@eio.upc.es)

Dept. of Statistics and Operational Research

Technical University of Catalonia

Motivation : Nonlinear Optimization in Statistics (I)

*"**All statistical procedures are, in the ultimate analysis, solutions to suitably formulated optimization problems**. Whether it is designing a scientific experiment, or planning a large-scale survey for collection of data, or choosing a stochastic model to characterize observed data, or drawing inference from available data, such as estimation, testing of hypotheses, and decision making, **one has to choose an objective function and minimize or maximize it subject to given constraints on unknown parameters and inputs** such as the costs involved."*

C.R. Rao, in "*Mathematical Programming in Statistics*", Arthanary and Dodge 1993

Motivation : Nonlinear Optimization in Statistics (II)

- This tutorial will be restricted to **Nonlinear Optimization** techniques, useful for:
  - Maximization of a likelihood function.
  - Nonlinear regression.
  - ...
- We will not cover other optimization techniques as:
  - Combinatorial optimization.
  - Global optimization.
  - Nondifferentiable optimization.
  - Heuristics.
- Let's go now to the contents of this tutorial... ▶

Maximization of the likelihood function

- **Current status Covariates in a simple linear model (Langohr, Gómez)**
  - **Model :** $Y = \alpha + \beta Z + \varepsilon \quad ; \quad Y \in \Re^n, Z \in \Re, \varepsilon \in \Re^n$

    $\varepsilon \sim N(0, \sigma^2 I_n) \Rightarrow Y \mid Z \sim N(\alpha + \beta Z, \sigma^2 I_n)$

    Decision variables è $= [\alpha \quad \beta \quad \sigma]^T \in \Re^3$, $\sigma \geq 0$ **Constraints**
  - Z is the ***Current Status Covariate*** with cumulative distribution $W(z)$: the only observation is whether Z exceeds the observed value z_i or not; δ_i is the corresponding indicator variable: $\delta_i = 1_{\{Z \leq z_i\}}$
  - **Observations :** $[y_i \quad z_i \quad \delta_i], i = 1, 2, \ldots, n$
  - **Covariate :** Z is supposed to be discrete with possible (ordered) values $s_1, s_2, \ldots, s_m$ and corresponding probabilities $\omega_j = P(Z = s_j)$, $j = 1, \ldots, m$:

    Decision variables ù $= [\omega_1, \ldots, \omega_m]^T$, **Constraints** $\omega_j \geq 0$, $\sum_{j=1}^{m} \omega_j = 1$

Maximization of the likelihood function

- **Maximum likelihood function :**

$$L_n(\text{ù},\text{è}) = \prod_{i=1}^{n}\sum_{j=1}^{m}\gamma_{ij} f(y_i \mid s_j;\text{è})\omega_j =$$

Objective function $f(x)$

$$= \prod_{i=1}^{n}\sum_{j=1}^{m}\gamma_{ij}\frac{1}{\sigma\sqrt{2\pi}}\left(e^{-\frac{1}{2}\frac{(y_i-\alpha-\beta s_j)^2}{\sigma^2}}\right)\omega_j$$

$$\gamma_{ij} = 1_{\{s_j \in I_i\}} \quad , \quad I_i = \begin{cases} [s_1 , z_i] & \text{if } \delta_i = 1 \\ (z_i , s_m] & \text{otherwise} \end{cases}$$

- The problem above corresponds to a *Linearly Constrained Nonlinear Optimization Problem* (LCNOP):

(LCNOP) $\max_x \{ f(x) \mid x \in X \}$; $X = \{ x \in \Re^n \mid Ax = b, l \leq x \leq u \}$

Nonlinear regression: SIDS

- **Sudden Infant Death Syndrome:**
  - The following nonlinear model was considered by Murphy and Campbell (1987) as a part of their study of the Sudden Infant Death Syndrome (SIDS).
  - Given a data series of the daily temperature t_{ij} in day j of year i (j=1,2,...,365 , i=1,2,...,5), the authors proposed the following harmonic model:

$$t_{ij}(a_i, b_i, c_i) = a_i \cos\left(\frac{2\pi}{365} j - b_i\right) + c_i$$

Decision variables x

$$i = 1,2,3,4,5$$

$$j = 1,2,\ldots,365$$

Nonlinear regression: SIDS

- **Nonlinear optimization problem associated to the SIDS model:**

Objective function $f(x)$

$$\min_{\substack{a_i, b_i, c_i \\ i=1,2,\ldots,5}} \frac{1}{2}\sum_{i=1}^{5}\sum_{j=1}^{365}\left[t_{ij}(a_i, b_i, c_i) - t_{ij}\right]^2$$

(Nonlinear Least-Squares Problem)

- The problem above corresponds to an ***Unconstrained Nonlinear Optimization Problem (UNOP)*** :

(UNOP) $\min_{x \in \Re^n} f(x)$

- If *smoothness conditions* are added to the model, the problem becomes a ***Generally Constrained NOP*** :

(GCNOP) $\min_x \{ f(x) \mid x \in X \}$; $X = \{ x \in \Re^n \mid h(x) = 0,\ g(x) \leq 0 \}$

Summary

Nonlinear Optimization Problem (NOP)

- The general (standard) form of the NOP is :

$$(\text{NOP})\begin{cases}\min\limits_{x\in\Re^n} & f(x) & \textbf{Objective function}\\ \text{subject to:} & h(x)=0 & \textbf{Equality constraints}\\ & g(x)\le 0 & \textbf{Inequality constraints}\end{cases}$$

where $x\in\Re^n$ are the **decision variables**, or simply, **variables**, and

$$f:\Re^n\to\Re \qquad h:\Re^n\to\Re^m \qquad g:\Re^n\to\Re^l$$

- Usually, f, h and g are required to be differentiable and "smooth" (*Lipschitz continuous*, or so) to guarantee good properties of the algorithms.
- Of course, $\max f(x) \equiv \min -f(x)$

Classification of the NOP accordingly with the solution

- Consider the NOP expressed in the following way:

$$(\text{NOP})\ \min_x\{f(x)\mid x\in X\}\ ;\ X=\{x\in\Re^n \mid h(x)=0,\ g(x)\le 0\}$$

(X is known as the **feasible set**)

- **NOP with optimal solution:** The set $\{f(x)|x\in X\}$ is bounded below
  $$\min_x\{f(x)=x_1^2+x_2^2 \mid x\ge 0\}$$
- **Infeasible problem** : the feasible set X is empty:
  $$\min_x\{f(x)=x_1^2+x_2^2 \mid x_1+x_2\le -1,\ x\ge 0\}$$
- **Unbounded problem:** The set $\{f(x)|x\in X\}$ is unbounded below
  $$\min_x\{f(x)=-x_1^2-x_2^2 \mid x\ge 0\}$$
- **Existence of at least one global minimum**: guaranteed if f is continuous and $X\subseteq\Re^n$ compact (*Weierstrass Theorem*)

Classification of the NOP accordingly with the formulation

- **Unconstrained NOP:**
  $$(\text{UNOP})\ \min_{x\in\Re^n} f(x)$$
- **NOP with Simple Bounds:**
  $$(\text{SBNOP})\ \min_x\{f(x)|l\le x\le u\}$$
- **Linearly Constrained NOP:**
  $$(\text{LCNOP})\ \min_x\{f(x)|x\in X\}\ ;\ X=\{x\in\Re^n|Ax=b, l\le x\le u\}$$
- **Generally Constrained NOP:**
  $$(\text{GCNOP})\ \min_x\{f(x)\mid x\in X\}\ ;\ X=\{x\in\Re^n \mid h(x)=0,\ g(x)\le 0\}$$

General strategy of the NO algorithms

- Given the feasible bounded NOP:

$$(\text{NOP})\ \min_x\{f(x)\mid x\in X\}\ ;\ X=\{x\in\Re^n \mid h(x)=0,\ g(x)\le 0\}$$

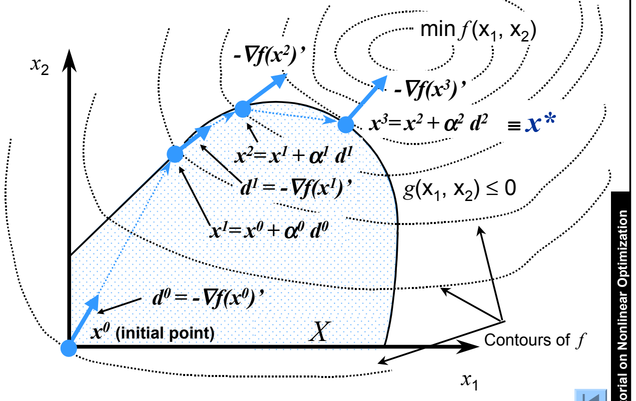
the general strategy followed by most of the NO alg. is:

1. Find a first feasible solution $x\in X$ (current solution).

2. If the current solution x satisfies the optimality conditions, then STOP: $x^*:=x$

3. If the current solution x does not satisfies the optimality conditions, find, using the local information available on x, a new feasible iterate $x\in X$ that improves the value of some merit function related with the objective function $f(x)$, or the objective function itself. Go to 2 with the new current iterate.

General strategy of the NO algorithms

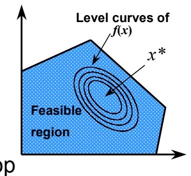


Desirable properties of the NO algorithms

- **Robustness**: they should perform well...
  - … on a wide variety of problems in their class
  - … for all reasonable choices of the initial variables
  - …without the need of "tuning".
- **Efficiency**: low execution time and memory requirements
- **Accuracy**: they should be able to identify the solution with precision without being affected by errors in the data or arithmetic rounding errors.

Local and Global optimization

- The methods presented here only seek for **local optima**
  - They converge to a point satisfying the first order optimality conditions (or second order, depending on the algorithm).
- **Convexity** : any local optima is global if the NOP is **convex**, that is:
  - If the objective function $f(x)$ is convex.
  - If the feasible set X is convex.
  - Example: minimization of a quadratic pos. def $f(x)$ over a politop



Bibliography and interesting web sites

- Arthanary, T.S., Dodge Y. (1993) "*Mathematical Programming in Statistics*". John Wiley and Sons
- Luenberger, D.G. (1984) "*Linear and Nonlinear Programming*". 2nd Edition. Addison Wesley.
- Nocedal, J., Wrigth, S.J. (1999) "*Numerical Optimization*". Springer Series in Operations Research. Springer Verlag.
- Moré, J.J., Wrigth, S.J. (1993) "*Optimization Software Guide*". Frontiers in Applied Mathematics. SIAM.
- Murphy, M.F.G., Campbell, M.J. (1987) "*Sudden infant death syndrome and environmental temperature: an analysis using vital statistics*". *J. of Epidemiology and Community Health*, March 1987, Vol. 41, No. 1, pages. 63-71.
- NEOS Guide : www-fp.mcs.anl.gov/otc/Guide/index.html
- NAG Numerical Libraries: www.nag.co.uk/numeric/numerical_libraries.asp
- PROC NLP (SAS Optimization Software) www-fp.mcs.anl.gov/otc/Guide/SoftwareGuide/Blurbs/procnlp.html

1BWSA Barcelona June 2002

Tutorial on Nonlinear Optimization

1BWSA-Tutorial-NLO-17

Algorithms for Unconstrained Nonlinear Optimization

Unconstrained Nonlinear Optimization

- **Fundamentals**
  - General framework: optimality conditions; descent directions; linesearch
  - Measures of performance of the algorithms: global convergence; local convergence.
- **Methods that use first derivatives**
  - Steepest Descent method (SD)
  - Conjugate Gradient method (CG)
  - Quasi-Newton method (QN)
- **Methods that use second derivatives**
  - Newton and Modified Newton methods (N, MN)
- **Nonderivative methods**
  - Finite differences, coordinate descent and direct search.
- **Nonlinear Least-squares problems**:
  - Gauss-Newton method (GN).

Fundamentals:
General Framework

Given (UNOP) $\min_{x\in\Re^n} f(x)$ **, generate a sequence** $\{x^k\}_{k=0}^{\infty}$ **that converges to the optimal solution** x^*

1. Initialize $x^k \in X \equiv \Re^n$ (current solution). $k:=0$

2. If the current solution x^k satisfies the **stopping criterium**, then $x^* := x^k$. **STOP**:

3. If x^k is not the optimal solution, find a new iterate that improves enough the value of the objective function, and take it as the new iterate. This is performed through the following steps:

- **3.1.** Computation of a **descent direction** d^k
- **3.2.** Computation of a **steplength** α^k
- **3.3.** Update: $x^{k+1} := x^k + \alpha^k d^k$, $k:=k+1$. Go to 2

Fundamentals:
Stopping criterium: optimality conditions

- **First-Order Necessary Conditions** — **Usual stopping criterium**

  "If x^* is a local minimizer and f is continuously differentiable in an open neighbourhood of x^*, then $\nabla f(x^*)=0$"

- **Second-Order Necessary Conditions**

  *"If x^* is a local minimizer and f and $\nabla^2 f$ is continuous in an open neighbourhood of x^*, then $\nabla f(x^*)=0$ and $\nabla^2 f(x^*)$ is **positive semidefinite**"*

- **Second-Order Sufficient Conditions.**

  *"Suppose that $\nabla^2 f$ is continuous in an open neighbourhood of x^* and that $\nabla f(x^*)=0$ and $\nabla^2 f(x^*)$ is **positive definite**. Then x^* is a **strict** local minimizer and f"*

- **The role of convexity.**

  *"When **f is convex**, any local minimizer x^* is a global minimizer of f. If, in addition, f is **differentiable**, then **any stationary point x^* is a global minimizer of f**"*

Fundamentals:

Practical stopping criterium

- The robust numerical implementation of the stopping criterium $\|\nabla f(x^k)\|\approx \mathbf{0}$ could be quite sophisticated. The algorithm will stop either ...

  ... If the measure of the relative size of the gradient at x^k is small:

$$\max_{1\le i\le n}\left|\frac{\max\{|x_i^k|,1\}}{\max\{|f(x^k)|,1\}}\nabla f(x^k)_i\right|\le \varepsilon^{\frac{1}{3}}$$

  ... or the measure of the relative change of the variables x^k in the last step is small :

$$\max_{1\le i\le n}\left|\frac{|x_i^{k+1}-x_i^k|}{\max\{x_i^k,1\}}\right|\le \varepsilon^{\frac{2}{3}}$$

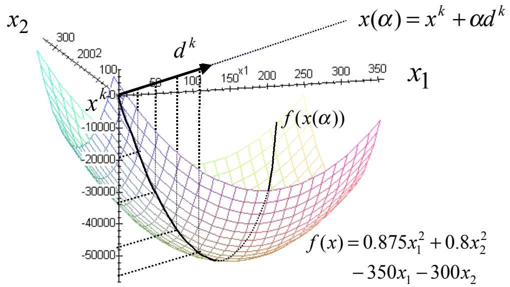
Where ε is the machine precision

Fundamentals:

Descent directions : definition

- **Descent direction :**

  $d^k\in\Re^n$ is a descent direction for the problem UNOP over x^k if: $\exists\,\overline{\alpha}\in\Re^+\ \big|\ \forall\alpha\in[0,\overline{\alpha}]:\ f(x^k+\alpha d^k)<f(x^k)$

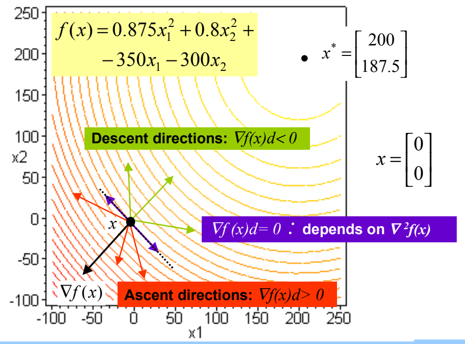


Fundamentals:

Descent directions : identification

- **Sufficient condition of descent:**

  If $\nabla f(x)d<0$ then d is a descent direction for $f(x)$ at x.



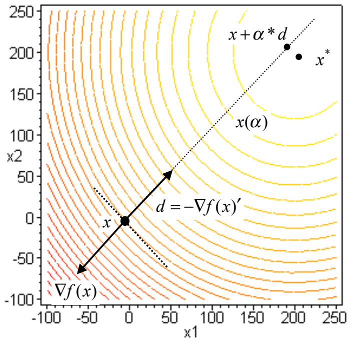
Fundamentals:

Descent directions : computation

- **Methods that use firsts derivatives:**
  - **Steepest descent:** $d^k=-\nabla f(x^k)$
  - **Conjugate Gradient:** $d^k=-\nabla f(x^k)+\beta^k d^{k-1}$
  - **Quasi-Newton Methods:** $d^k=-B^k\nabla f(x^k)$, B^k simetric, pos. def.
- **Methods that use second derivatives:**
  - **Newton Method :** $d^k=-[\nabla^2 f(x^k)]^{-1}\nabla f(x^k)$
- The properties of these methods will be studied later.

Fundamentals:

Steplength



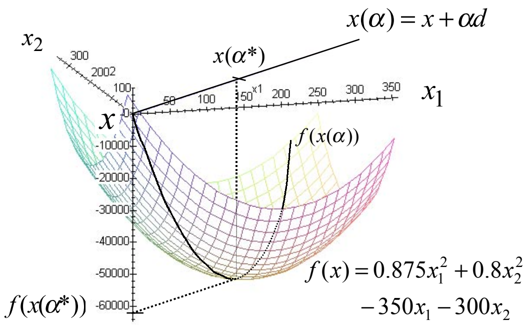
- x is the current iterate
- d is the current search descent direction (for instance $d=-\nabla f(x)'$)
- The next iterate will lie in the half line $x(\alpha)= x+\alpha d$, $\alpha \geq 0$
- We must decide how far to move from x along d:

steplength α^*

Fundamentals:

Computing the steplength: linesearch

- **Linesearch** : $\alpha^*=\text{argmin}\{ f(x+\alpha d) \mid \alpha \geq 0 \}$



Fundamentals:

Exact linesearch (quadratic functions)

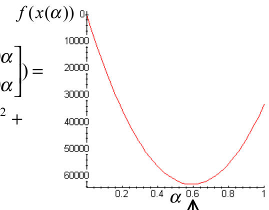
- **Linesearch** : $\alpha^*=\text{argmin}\{ f(x+\alpha d) \mid \alpha \geq 0 \}$
- In our example: $\min_{x\in\Re^n} f(x)=0.875x_1^2+0.8x_2^2-350x_1-300x_2$

$$x=\begin{bmatrix}0\\0\end{bmatrix} \; ; \; d=-\nabla f(x)'=\begin{bmatrix}350\\300\end{bmatrix} \quad \left(\nabla f(x)d=-\|\nabla f(x)\|<0\right)$$

$$f(x(\alpha))=f(x+\alpha d)=$$

$$=f\left(\begin{bmatrix}0\\0\end{bmatrix}+\alpha\begin{bmatrix}350\\300\end{bmatrix}\right)=f\left(\begin{bmatrix}350\alpha\\300\alpha\end{bmatrix}\right)=$$

$$=0.875(350\alpha)^2+0.8(300\alpha)^2+$$
$$-350(350\alpha)-300(300\alpha)$$
$$=179187.5\alpha^2-212500\alpha$$



$$\frac{df(x(\alpha^*))}{d\alpha}=358375\alpha^*-212500=0 \; ; \; \alpha^*=0.5939$$

Fundamentals:

Inexact linesearch (general functions)

- **Problem:** when $f(x)$ is not quadratic the optimal steplength α^* must be estimated numerically.
- **Purpose of the inexact linesearch** :
  to identify $\alpha^k \approx \text{argmin}\{ f(x^k+\alpha^k d^k) \mid \alpha \geq 0 \}$ such that...
  … provides significant reduction of $f(x)$
  … without spending too much time in the computation.
- **Rationale of the inexact linesearch methods**:
  - Define a criterium for the *sufficient decrease* in $f(x)$
  - Find somehow an interval [α_{min} , α_{max}] containing α^*.
  - Set $\alpha := \alpha_{max}$, the trial steplength.
  - Repeat Until $f(x^k+\alpha d^k)$ satisfies the ***sufficient decrease condition***
    Find (bisection, interpolation) a trial steplength $\alpha \in$ [α_{min} , α_{max}]
    Update [α_{min} , α_{max}] .
    End Repeat
  - Set $\alpha^k := \alpha$.

Fundamentals:

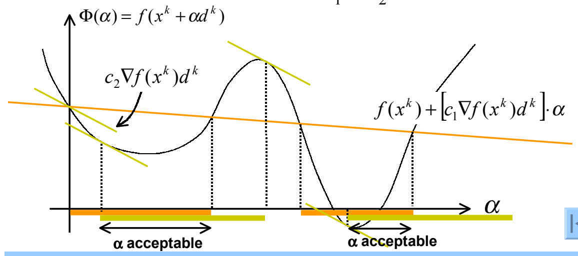
Inexact linesearch (general functions)

- **Sufficient decrease condition:** (Wolfe conditions)

$$f(x^k+\alpha d^k) \le f(x^k) + [c_1 \nabla f(x^k) d^k]\cdot\alpha \quad \text{(W1)}$$

$$\nabla f(x^k+\alpha d^k) d^k \ge c_2 \nabla f(x^k) d^k \quad \text{(W2)}$$

$$0 < c_1 < c_2 < 1$$



Fundamentals:

Measures of performance of an algorithm

- Global convergence: an algorithm is said to be "globally convergent" if

$$\lim_{k\to\infty} \left\| \nabla f(x^k) \right\| = 0$$

  that is, if we can assure that the method converges to a **stationary point**.
  - Introducing second order information, we can strengthen the result to include **convergence to a local minimum**.

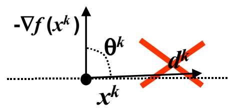
- Local convergence: how fast the sequence $\{x^k\}$ approaches to the optimal solution x^*.

Fundamentals:

Global convergence

- To ensure "global" convergence, several assumptions must be imposed to the objective function, the search direction and the steplength. Roughly speaking:
  - The objective f must be **bounded below** and **continuously differentiable**.
  - The gradient ∇f must be smooth (**Lipschitz continuous**).
  - The search direction d^k must be a **descent direction**.
  - The steplengths α^k must satisfy the **Wolfe conditions**.
  - The **angle** θ^k between the search directions d^k and the steepest descent direction $-\nabla f(x^k)$ must be **bounded away from 90°**

$$\cos\theta^k = \frac{-\nabla f(x^k) d^k}{\left\|\nabla f(x^k)\right\| \left\|d^k\right\|}$$



Fundamentals:

Conditions for "global" convergence

- **Zoutendijk's theorem:**

  *Consider any iteration of the form $x^{k+1} := x^k + \alpha^k d^k$ where d^k **is a descent direction** and α^k **satisfies the Wolfe conditions**. Suppose that f **is bounded below** in $\Re^n$ and that f **is continuously differentiable** in an open set N containing the level set*

$$L = \left\{ x : f(x) \le f(x^0) \right\}$$

  *where x^0 is the starting point of the iteration. Assume also that the **gradient is Lipschitz continuous** in N, that is, there exists a constant $L > 0$ such that:*

$$\left\| \nabla f(x) - \nabla f(\tilde{x}) \right\| \le L \left\| x - \tilde{x} \right\|, \quad \text{for all } x, \tilde{x} \in N$$

  Then

$$\sum_{k\ge 0} \cos^2\theta^k \left\| \nabla f(x^k) \right\|^2 < \infty \quad (\textit{Zoutendijk condition})$$

Fundamentals:

Interpretation of the Zoutendijk condition

- The Zoutendijk condition implies:

$$\cos^2\theta^k \left\|\nabla f(x^k)\right\|^2 \to 0 \qquad (1)$$

- If the method for choosing the search direction d^k ensures that the angle θ^k is bounded away from 90°, then there is a positive constant δ such that:

$$\cos\theta^k \geq \delta > 0 \quad \forall k$$

then, it follows from (1) that:

$$\lim_{k\to\infty}\left\|\nabla f(x^k)\right\| = 0$$

Fundamentals:

Linear and superlinear order of convergence

- **Linear convergence**:

*let $\{x^k\}$ be a sequence in $\Re^n$ that converges to x^*. We say that the convergence is **Q-linear** (or simply linear) if there is a constant $r\in(0,1)$ (ratio of convergence) such that*

$$\frac{\left\|x^{k+1}-x^*\right\|}{\left\|x^k-x^*\right\|} \leq r \quad \text{, for all } k \text{ sufficiently large}$$

- *The convergence is said to be **Q-superlinear** if*

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

Fundamentals:

Quadratic order of convergence

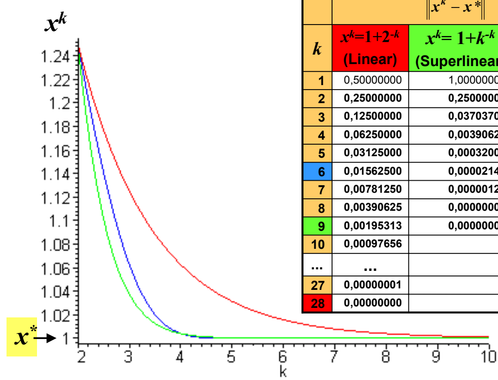
- **Quadratic convergence**:

*let $\{x^k\}$ be a sequence in $\Re^n$ that converges to x^*. We say that the convergence is **Q-quadratic** (or simply quadratic) if there is a constant $M>0$ such that :*

$$\frac{\left\|x^{k+1}-x^*\right\|}{\left\|x^k-x^*\right\|^2} \leq M \quad \text{, for all } k \text{ sufficiently large}$$

Fundamentals:

Order of convergence: example



k	$\left\|x^k - x^*\right\|$ $x^k=1+2^{-k}$ (Linear)	$x^k=1+k^{-k}$ (Superlinear)	$x^k=1+2^{-2^{k-1}}$ (Quadratic)
1	0,50000000	1,00000000	0,50000000
2	0,25000000	0,25000000	0,25000000
3	0,12500000	0,03703704	0,06250000
4	0,06250000	0,00390625	0,00390625
5	0,03125000	0,00032000	0,00001526
6	0,01562500	0,00002143	0,00000000
7	0,00781250	0,00000121	
8	0,00390625	0,00000006	
9	0,00195313	0,00000000	
10	0,00097656		
...	...		
27	0,00000001		
28	0,00000000		

Fundamentals
Order of convergence of the UNOP alg.

Algorithm	Order of convergence	
	f quadratic	f general
Steepest descent	Linear (r depends on $\nabla^2 f(x^*)$)	
Quasi-Newton	Superlinear	
Conjugate Gradients	$\leq n$ iterations	Quadratic (sub-sequence $\{x^{k+n}\}$)
Newton	1 iteration	Quadratic

Methods that use first derivatives
The Steepest Descent method (SD)

- **Search direction**: $d_{SD}^k = -\nabla f(x^k)'$
- **Global convergence:**
  - **Descent direction**: $\nabla f(x^k)\, d^k = -\|\nabla f(x^k)\| < 0$
  - **Zoutendijk condition**: angle $\theta^k = 0°$, $\cos(\theta^k) = 1\ \forall k$
- **Local convergence:**
  - Linear convergence.
  - The rate of convergence r depends on the properties of $f(x)$
- **Computational requirements:**
  - **Low memory requirements** : only needs to store several vectors of dimension n
  - Very easy to implement.

Methods that use first derivatives
Rate of convergence of the SD

- **Upper bound to the rate of convergence**

"Suppose that $f : \Re^n \to \Re$ is twice continuously differentiable, and that the iterates generated by the steepest descent method with exact line search converges to a point x^* where the Hessian matrix $\nabla^2 f(x^*)$ is positive definite. Then:

$$f(x^{k+1}) - f(x^*) \leq \beta\left[f(x^k) - f(x^*)\right] \ ; \ \beta = \left(\frac{\lambda_n - \lambda_1}{\lambda_n + \lambda_1}\right)^2$$

where $\lambda_1 \leq \ldots \leq \lambda_n$ are the eigenvalues of $\nabla^2 f(x^*)$ "

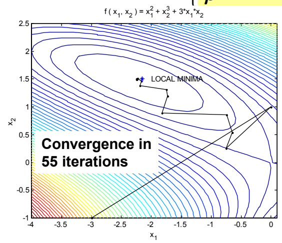
Methods that use first derivatives
Rate of convergence of the SD: example

- ***f(x)* with "slow" rate of conv.**

$$f(x_1, x_2) = x_1^2 + x_2^3 + 3x_1x_2$$

$$x^0 = [-3 \quad -1]' \quad x^* = [-2.25 \quad 1.5]'$$

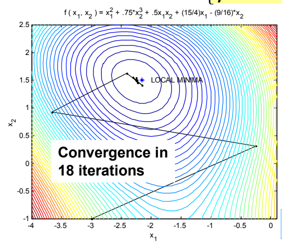
$$\nabla^2 f(x^*) = \begin{bmatrix} 2 & 3 \\ 3 & 9 \end{bmatrix} \ ; \ \begin{cases} \lambda_1 = 0.8902 \\ \lambda_2 = 10.1098 \\ \beta = 0.7025 \end{cases}$$



- ***f(x)* with "fast" rate of conv.**

$$f(x_1, x_2) = x_1^2 + \tfrac{3}{4}x_2^3 + \tfrac{1}{2}x_1x_2 + + \tfrac{15}{4}x_1 - \tfrac{9}{16}x_2$$

$$\nabla^2 f(x^*) = \begin{bmatrix} 2 & .5 \\ .5 & 2.25 \end{bmatrix} \ ; \ \begin{cases} \lambda_1 = 1.6096 \\ \lambda_2 = 2.6404 \\ \beta = 0.0588 \end{cases}$$

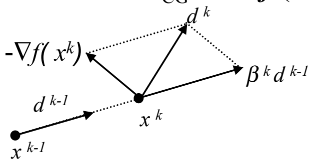


Methods that use first derivatives

The Conjugate Gradient method (CG)

1BWSA Barcelona June 2002

- **Search direction:** $d_{\mathrm{CG}}^{k} = -\nabla f(x^{k})' + \beta^{k} d^{k-1}$



- **Choices of β^k:**

Fletcher-Reeves : $\beta_{\mathrm{FR}}^{k} = \dfrac{\nabla f(x^{k})\nabla f(x^{k})'}{\left\|\nabla f(x^{k-1})\right\|^{2}}$

(best theoretical prop.)

Polak-Ribière : $\beta_{\mathrm{PR}}^{k} = \dfrac{\nabla f(x^{k})\left(\nabla f(x^{k}) - \nabla f(x^{k-1})\right)'}{\left\|\nabla f(x^{k-1})\right\|^{2}}$

(best practical behaviour)

Unconstrained Nonlinear Optimization

1BWSA-Tutorial-UNO-25

Methods that use first derivatives

Properties of the CG method (I)

1BWSA Barcelona June 2002

- **Global convergence** :
  - **Descent direction**: d^k_{GC} is a descent direction if the steplength α^k satisfies the *strong Wolfe conditions* :

$$f(x^k + \alpha d^k) \le f(x^k) + [c_1 \nabla f(x^k) d^k]\alpha \quad \text{(SW1)}$$
$$\left|\nabla f(x^k + \alpha d^k) d^k\right| \le c_2 \left|\nabla f(x^k) d^k\right| \quad \text{(SW2)}$$
$$0 < c_1 < c_2 < \tfrac{1}{2}$$

  - **Zoutendijk condition** : can be proved if the method is periodically restarted setting:

$$d_{\mathrm{CG}}^{l} = -\nabla f(x^{l})' = d_{\mathrm{SD}}^{l} \;,\; l = n, 2n, 3n, \ldots$$

Unconstrained Nonlinear Optimization

1BWSA-Tutorial-UNO-26

Methods that use first derivatives

Properties of the CG method (II)

1BWSA Barcelona June 2002

- **Local convergence** :
  - The CG method with restart has ***n*-step quadratically** convergence, that is:

$$\left\|x^{k+n} - x^*\right\| = O\left(\left\|x^{k} - x^*\right\|^{2}\right)$$

- **Computational requirements** :
  - **Low memory consumption**: only needs to store several vectors of dim n.
  - Almost as simple to implement as SD (the only difficulty is the computation of constant β^k).

Unconstrained Nonlinear Optimization

1BWSA-Tutorial-UNO-27

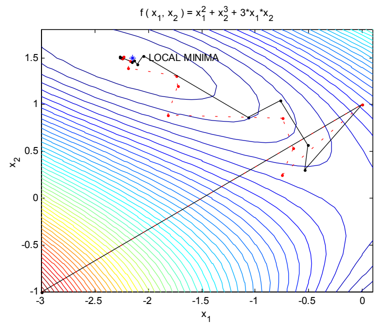
Methods that use first derivatives

Example of the CG method

1BWSA Barcelona June 2002

- **Exemple** : $f(x_1, x_2) = x_1^{\,2} + x_2^{\,3} + 3x_1x_2$

$x^0 = [-3 \;\; -1]' \qquad x^* = [-2.25 \;\; 1.5]'$



CG method (38 iterations)

SD method (55 iterations)

Unconstrained Nonlinear Optimization

1BWSA-Tutorial-UNO-28

Methods that use first derivatives

Quasi-Newton methods (QN)

- **Rationale of the Newton method**:
  To find the next iterate x^{k+1} as the minimizer of the quadratic model $m^k(p)$ of $f(x)$ around the current iterate x^k:

$$f(x^k+p) \approx f(x^k) + \nabla f(x^k)p + \frac{1}{2}p'\nabla^2 f(x^k)p \equiv m^k(p)$$

$$p^k \leftarrow \operatorname{argmin}\{m^k(p)\}$$

$$\nabla m^k(p^k) = \nabla^2 f(x^k)p^k + \nabla f(x^k)' = 0 \rightarrow p^k = -\nabla^2 f(x^k)^{-1}\nabla f(x^k)$$

$$x^{k+1} = x^k + p^k$$

- **Quasi-Newton method**:
  Applies a Newton strategy *avoiding the need of second derivatives* by substituiting the Hessian matrix $\nabla^2 f(x^k)$ by an approximation B^k

Methods that use first derivatives

Quasi-Newton methods (QN)

- **Quasi-Newton direction:** $d^k_{\text{QN}} = -\left[B^k\right]^{-1}\nabla f(x^k)'$
- **Choices of B^k**: given a symmetric pos. def. matrix B^0 , and: $s^k = x^{k+1} - x^k$; $y^k = \nabla f(x^{k+1})' - \nabla f(x^k)'$

Broyden-Fletcher-Goldfarb-Shanno (BFGS) :

$$B^{k+1}_{\text{BFGS}} = B^k_{\text{BFGS}} - \frac{B^k_{\text{BFGS}} s^k {s^k}^T B^k_{\text{BFGS}}}{{s^k}^T B^k_{\text{BFGS}} s^k} + \frac{y^k {y^k}^T}{{y^k}^T s^k}$$

Davidon-Fletcher-Powell (DFP) : $H^k = [B^k]^{-1}$

$$H^{k+1}_{\text{DFP}} = H^k_{\text{DFP}} - \frac{H^k_{\text{DFP}} y^k {y^k}^T H^k_{\text{DFP}}}{{y^k}^T H^k_{\text{DFP}} y^k} + \frac{s^k {s^k}^T}{{y^k}^T s^k}$$

Methods that use first derivatives

Quasi-Newton methods (QN)

- The BFGS formula is considered to be the most effective of all quasi-Newton updating formulae.
- **Properties of matrix B^{k+1}** : given B^k, $n\times n$ symetric, positive definite matrix, then the BFGS update provides B^{k+1} that :
  - Is symetric
  - Is positive definite if $s^{k\prime} y^k > 0$.
    (guaranteed if α^k satisfies the Wolfe conditions)
  - Satisfies the ***secant equation*** : $B^{k+1}s^k = y^k$
    (this is how we force $B^{k+1} \approx \nabla^2 f(x^{k+1})$.)

Methods that use first derivatives

Global convergence of the BFGS method

- **Global convergence** :
  - **Descent direction**: we check the descent condition:

$$\left.\begin{array}{l} \nabla f(x^k)d^k_{\text{BFGS}} = -\nabla f(x^k)\left[B^k_{\text{BFGS}}\right]^{-1}\nabla f(x^k)' \\ B^k_{\text{BFGS}} \text{ pos. def.} \Rightarrow \left[B^k_{\text{BFGS}}\right]^{-1} \text{ pos. def} \end{array}\right\} \nabla f(x^k)d^k_{\text{BFGS}} < 0$$

  - **Zoutendijk condition** : can be proved if the matrices B^k have an uniformly bounded condition number, that is, if there is a constant M such that:

$$\operatorname{cond}(B^k) = \left\|B^k\right\|\left\|{B^k}^{-1}\right\| \leq M, \quad \text{for all } k$$

1BWSA Barcelona June 2002

Methods that use first derivatives

Local convergence of the BFGS method

- **Local convergence** : under the following assumptions:
  - *The objective function f is twice continuously differentiable*
  - *The level set $\Omega=\{\, x\in\Re^n : f(x) \le f(x^0) \,\}$ is convex*
  - *The objective function f has a unique minimizer x^* in Ω*
  - *The Hessian matrix $\nabla^2 f(x^{k+1})$ is Lipschitz continuous at x^* and positive definite on Ω.*

  It can be shown that the iterates generated by the BFGS algorithm converges superlinearly to the minimizer x^*

Unconstrained Nonlinear Optimization

1BWSA-Tutorial-UNO-33

1BWSA Barcelona June 2002

Methods that use first derivatives

Local convergence of the BFGS method

- **Computational issues** :
  - The efficient implementation of the BFGS method does not store B^k explicitily, but the Cholesky factorization:
    $$B^k = L^k D^k L^{k'}$$
  - **Memory consumption**: $n(n+1)/2$ = O(n^2) elements of the Cholesky factors.
  - **Computational cost per iteration** : O(n^2) operations necessary to ...

    ... update the Cholesky factors .

    ... find the solution of the linear system $B^k d^k = -\nabla f(x^k)$.
  - The development of an efficient implementation of the BFGS method is quite difficult.

Unconstrained Nonlinear Optimization

1BWSA-Tutorial-UNO-34

1BWSA Barcelona June 2002

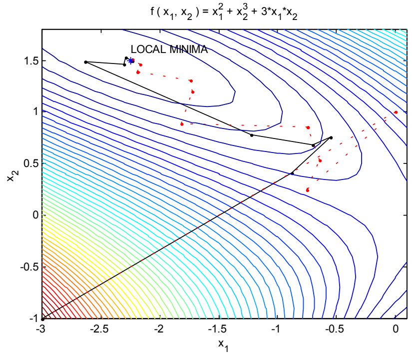
Methods that use first derivatives

Example of the BFGS method

- **Exemple** : $f(x_1,x_2)=x_1^{\,2}+x_2^{\,3}+3x_1x_2$

$x^0=[-3 \quad -1]'$

$x^*=[-2.25 \quad 1.5]'$



BFGS
(11 iterations)

SD method
(55 iterations)

Unconstrained Nonlinear Optimization

1BWSA-Tutorial-UNO-35

1BWSA Barcelona June 2002

Methods that use first and second derivatives

The Newton method (N)

- **Rationale of the Newton method:**

  To find the next iterate x^{k+1} as the minimizer of the quadratic model $m^k(p)$ of $f(x)$ around the current iterate x^k:

$$f(x^k+p) \approx f(x^k)+\nabla f(x^k)p+\frac{1}{2}p'\nabla^2 f(x^k)p \equiv m^k(p)$$

$$p^k \leftarrow \operatorname{argmin}\{m^k(p)\} = -\left[\nabla^2 f(x^k)\right]^{-1}\nabla f(x^k)$$

$$x^{k+1} = x^k + p^k$$

Unconstrained Nonlinear Optimization

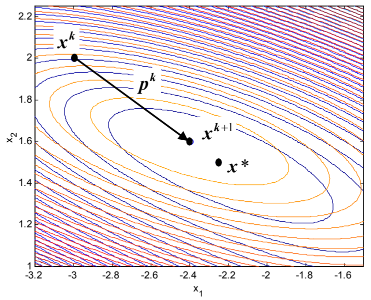
1BWSA-Tutorial-UNO-36

Methods that use first and second derivatives

The Newton method (N)

- **Exemple** : $f(x_1,x_2)=x_1^{\,2}+x_2^{\,3}+3x_1x_2$ $\quad x^k=[-3 \quad 2]'$

  $m^k(x_1,x_2)=x_1^{\,2}+6x_2^{\,2}+3x_1x_2-12x_2+21$



$$x^*=\begin{bmatrix}-2.25\\1.5\end{bmatrix}$$

(minimizer of f)

$$p^k=-\left[\nabla^2 f(x^k)\right]^{-1}\nabla f(x^k)=\begin{bmatrix}0.6\\-0.4\end{bmatrix}$$

$$x^{k+1}=x^k+p^k=\begin{bmatrix}-2.4\\1.6\end{bmatrix}$$

(minimizer of m^k)

Methods that use first and second derivatives

The Newton method (N)

- **Search direction:** $d_N^k=-\nabla^2 f(x^k)^{-1}\nabla f(x^k)^T$
- **Global convergence** :
  - **Descent direction**: the descent nature of d^k_N ***can only be guaranteed if*** the Hessian matrix $\nabla^2 f(x^k)$ is ***positive definite***:

    $$\text{If } \nabla^2 f(x^k) \text{ pos. def. then}$$
    $$\nabla f(x^k)d_N^k=-\nabla f(x^k)\nabla^2 f(x^k)^{-1}\nabla f(x^k)^T<0$$

    otherwise, ***the global convergence of the Newton method cannot be guaranteed***.

Methods that use first and second derivatives

Losing of the global convergence

- **Example** : $f(x_1,x_2)=x_1^{\,2}+x_2^{\,3}+3x_1x_2$

  Over x^0 = [-3 -1]', the Hessian matrix is:

  $$\nabla^2 f(x^0)=\begin{bmatrix}2 & 3\\3 & -6\end{bmatrix}$$

  which is indefinite (λ_1=3, λ_2=-7). Therefore, the descent property of the Newton direction cannot be guaranteed.In fact, after 1 iteration, the method finds an ascent direction:

k	$\nabla^2 f(x^k)$	$\nabla f(x^k)\ d_N^k$	
0	Indefinite	$-35.1428 < 0$	Descent direction
1	Indefinite	$0.37699 > 0$	ASCENT direction

Methods that use first and second derivatives

Local convergence of the Newton method

- **Local convergence** :

  "Suppose that
  - *The solution x^* satisfies the sufficient optimality condition .*
  - *The function f is twice differentiable.*
  - *The Hessian $\nabla^2 f(x)$ is Lipschitz continuous in a neighbourhood of a solution x^**

  Consider the Newton iteration $x^{k+1}=x^k+d_N^k$. Then:
  - *if the starting point x^0 is sufficiently close to x^*, the sequence of iterates converges to x^*;*
  - *the order of convergence of $\{x^k\}$ is quadratic; and*
  - *the sequence of gradient norms $\{\|\nabla f(x^k)\|\}$ converges quadratically to zero."*

1BWSA Barcelona June 2002

Methods that use first and second derivatives

Quadratic order of convergence

- **Example :** $f(x_1, x_2) = x_1^2 + x_2^3 + 3x_1x_2$

  The newton method converges from x^0 = [-3 2]' to x^*=[-2.25 1.5]^T in 4 iterations, **reducing the error $\|x^k - x^*\|$ quadratically** at each step :

k	$\|x^k-x^*\|$	$\le$	$\|x^{k-1}-x^*\|^2$	$\|\nabla f(x^k)\|$
0	9.013×10^{-1}			3.0
1	1.803×10^{-1}	$\le$	8.125×10^{-1}	4.8×10^{-1}
2	1.060×10^{-2}	$\le$	3.250×10^{-2}	2.657×10^{-2}
3	4.126×10^{-5}	$\le$	1.124×10^{-4}	1.029×10^{-4}
4	6.296×10^{-10}	$\le$	1.702×10^{-9}	1.571×10^{-9}

Unconstrained Nonlinear Optimization

1BWSA Barcelona June 2002

Methods that use first and second derivatives

Modified Newton methods (MN)

- **Search direction:**

$$d^k_{\mathrm{MN}} = -{B^k_{\mathrm{MN}}}^{-1} \nabla f(x^k)^T$$

where $B^k_{\mathrm{MN}} = \nabla^2 f(x^k) + E^k$, with

  - $E^k = 0$ if $\nabla^2 f(x^k)$ is sufficiently positive definite;
  - otherwise E^k is chosen to ensure that B^k_{MN} is sufficiently positive definite.

- **Methods to compute B^k_{MN}** : based on the modification of
  - The *spectral decomposition* of $\nabla^2 f(x^k) = Q\Lambda Q^T$.
  - The *Cholesky factorization* of $\nabla^2 f(x^k) = LDL^T$.

Unconstrained Nonlinear Optimization

1BWSA Barcelona June 2002

Methods that use first and second derivatives

Global convergence of the MN method

- **Global convergence** :
  - **Descent direction**: as B^k_{MN} is always positive definite,therefore, d^k_{MN} is a descent search direction.
  - **Zoutendijk condition** : can be proved if the matrices B^k_{MN} have an uniformly bounded condition number, that is, if there is a constant M such that :

$$\mathrm{cond}(B^k_{\mathrm{MN}}) = \left\| B^k_{\mathrm{MN}} \right\| \left\| {B^k_{\mathrm{MN}}}^{-1} \right\| \le M, \quad \text{for all } k$$

Unconstrained Nonlinear Optimization

1BWSA Barcelona June 2002

Methods that use first and second derivatives

Local convergence of the MN method

- **Local convergence** :
  - If the sequence of iterates $\{x^k\}$ converges to a point x^* where $\nabla^2 f(x^*)$ is sufficiently positive definite (i.e. $E^k = 0$ for k large enough), then the MN method reduces to the Newton methods, and the convergence is quadratic.
  - If $\nabla^2 f(x^*)$ is close to singular (that is, there is not guarantee that E^k=0) the convergence rate may only be linear.

Unconstrained Nonlinear Optimization

Methods that use first and second derivatives

Other aspects of the MN methods

- **Computational issues** :
  - The efficient implementation of the MN methods computes and store the modified Cholesky factorization of B^k_{MN}.
  - **Memory consumption**: $n(n+1)/2$ = O(n^2) elements of the Cholesky factors.
  - **Computational cost per iteration** : O(n^3) operations necessary to ...

    ... compute the modified Cholesky factors of $\nabla^2 f(x^*)$.

    ... find the solution of the linear system $B^k_{MN}\, d^k_{MN} = -\nabla f(x^k)$

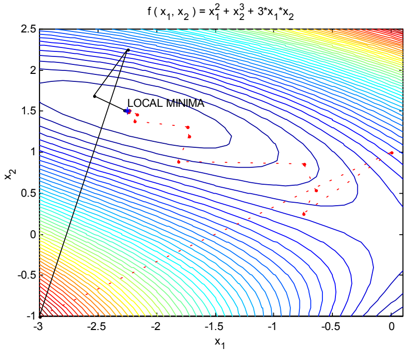
    plus the effort of computing the second derivatives
  - The efficient implementation of the MN method is quite difficult.

1BWSA Barcelona June 2002
Unconstrained Nonlinear Optimization

Methods that use first and second derivatives

Example of MN method (modified Chol. fac.)

- **Example** : $f(x_1,x_2)=x_1^{\,2}+x_2^{\,3}+3x_1x_2$ $\quad x^0=[-3\quad -1]'$ $\quad x^*=[-2.25\quad 1.5]'$



MN method (4 iterations)

SD method (55 iterations)

1BWSA Barcelona June 2002
Unconstrained Nonlinear Optimization

Nonderivative methods

Motivation and classification

- **Motivation:** in many problems, either the derivatives are not available in explicit form or they are given by very complicated expressions, prone to produce coding errors.
- **Example:** a Log-Likelihood function like

$$
\begin{aligned}
l(\breve{u},\alpha,\beta,\sigma)=\sum_{i=1}^{n}\Bigg[&\varepsilon_i\xi_{1i}\ln\left(\sum_{j=1}^{m}\frac{\gamma_{ij}}{\sigma}\exp\left[\frac{\ln(y_{obs,i}+z_{obs,i}-s_j)-\alpha-\beta\ln(s_j)}{\sigma}-e^{\frac{\ln(y_{obs,i}+z_{obs,i}-s_j)-\alpha-\beta\ln(s_j)}{\sigma}}\right]\omega_j\right)\\
&+\varepsilon_i\xi_{2i}\ln\left(\sum_{j=1}^{m}\gamma_{ij}\exp\left[-e^{\frac{\ln(y_{obs,i}+z_{obs,i}-s_j)-\alpha-\beta\ln(s_j)}{\sigma}}\right]\omega_j\right)\\
&+\varepsilon_i(1-\xi_{1i})(1-\xi_{2i})\ln\left(\sum_{j=1}^{m}\gamma_{ij}\left(1-\exp\left[-e^{\frac{\ln(y_{obs,i}+z_{obs,i}-s_j)-\alpha-\beta\ln(s_j)}{\sigma}}\right]\right)\omega_j\right)+(1-\varepsilon_i)\ln\left(\sum_{j=1}^{m}\gamma_{ij}\omega_j\right)\Bigg]
\end{aligned}
$$

1BWSA Barcelona June 2002
Unconstrained Nonlinear Optimization

Nonderivative methods

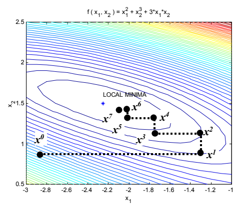
Classification

- **Finite differences** : to use a first derivative method (SD,CG,QN), computing the gradient as:

$$\frac{\partial f(x^k)}{\partial x_i}\approx\frac{1}{\varepsilon}\left(f(x^k+\varepsilon e_i)-f(x^k)\right)$$

  with ε a small positive scalar and e_i the unit vector

- **Coordinate descent:** the obj. function is minimized along one coordinate direction at each iteration.

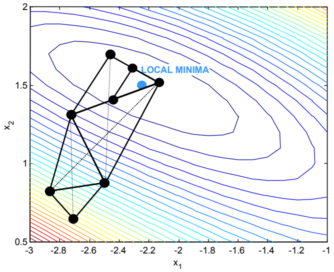


Nonderivative methods

Classification

– **Nelder & Mead simplex method** :

– Not to be confused with the simplex method for linear programming.

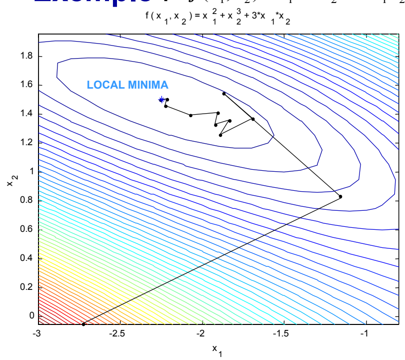


- Start with an initial **simplex** (convex hull of $n+1$ points).
- Select a new point that improves the worst point of the current simplex.
- Update de current simplex.

Nonderivative methods

Nelder & Mead method

- **Exemple** : $f(x_1,x_2)=x_1^2+x_2^3+3x_1x_2$ $\quad x^*=[-2.25 \quad 1.5]^t$



NM method (11 iterations)

Methods for Unconstrained Nonlinear Optimization

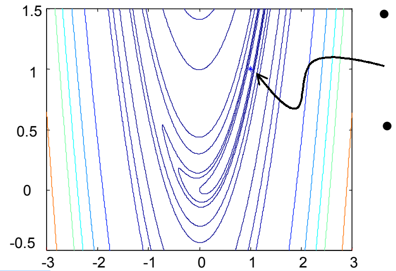
Computational comparison

Rosenbrock function (n=4)	Iter.	Execution time (seconds)	$f(x^*)$	$\|\| \nabla f(x^*) \|\|$
Steepest Descent	4760	120.34	1.038×10^{-11}	9.222×10^{-6}
Nelder & Mead	222	3.84	4.586×10^{-12}	8.828×10^{-5}
Conjugate Gradient	42	1.43	7.513×10^{-13}	7.820×10^{-7}
Quasi-Newton	27	0.33	4.980×10^{-17}	2.793×10^{-7}
Modified Newton	14	0.22	3.344×10^{-26}	2.506×10^{-12}
Newton	14	0.16	3.344×10^{-26}	2.506×10^{-12}

The extended Rosenbrock function

$$f(x)=\sum_{i=1}^{n/2}\left[10\left(x_{2i}-x_{2i-1}^2\right)^2+\left(1-x_{2i-1}\right)^2\right]$$

The contours lines for $n=2$:



- Unique global minimizer at $x^*=[\ 1\ \ 1\ ]^T$
- It is considered a difficult function to minimize.

Nonlinear Least-Squares Problems

- We will study now the solution for the Nonlinear Least-Squares Problem

$$\text{(NLSP)} \quad \min_{x \in \Re^n} f(x) = \frac{1}{2} \sum_{j=1}^{m} r_j^2(x) = \frac{1}{2} \| r(x) \|_2^2$$

where $r_j(x)$ represents the residuals of the model to be adjusted, and the decision variables x are the coefficients of the model.

Nonlinear Least-Squares Problems

- For instance, in the SIDS model:

$$\min_{\substack{a_i, b_i, c_i \\ i=1,2,\ldots,5}} \frac{1}{2} \sum_{i=1}^{5} \sum_{k=1}^{365} \left[t_{ik}(a_i, b_i, c_i) - t_{ik} \right]^2$$

we have:

- Decision variables: $x = [a_i \quad b_i \quad c_i]_{i=1,2,\ldots,5} \in \Re^{15}$
- Residuals: $r_j(x) = t_{ik}(a_i, b_i, c_i) - t_{ik}$
- Objective function:

$$f(x) = \frac{1}{2} \sum_{i=1}^{5} \sum_{k=1}^{365} \left[t_{ik}(a_i, b_i, c_i) - t_{ik} \right]^2$$

NLSP through the Newton method

- Remember that the **Newton search direction** was defined as :

$$d_N^k = -\nabla^2 f(x^k)^{-1} \nabla f(x^k)^T$$

therefore, in order to solve the NLSP with the Newton method, we need the **first and second derivatives of the objective function**

$$f(x) = \frac{1}{2} \sum_{j=1}^{m} r_j^2(x)$$

NLSP through the Newton method

- The derivatives of $f(x)$ can be expressed in terms of the **Jacobian of the residuals r**:

$$J(x) = \left[\frac{\partial r_j}{\partial x_i} \right]_{\substack{j=1,2,\ldots,m \\ i=1,2,\ldots,n}}$$

The gradient is: $\nabla f(x) = \sum_{j=1}^{m} r_j(x) \nabla r_j(x) = J(x)^T r(x)$

and the Hessian :

$$\nabla^2 f(x) = \sum_{j=1}^{m} \nabla r_j(x) \nabla r_j(x)^T + \sum_{j=1}^{m} r_j(x) \nabla^2 r_j(x) =$$

$$= J(x)^T J(x) + \sum_{j=1}^{m} r_j(x) \nabla^2 r_j(x)$$

The Gauss-Newton (G-N) method (I)

- The **Gauss-Newton** method applies a **Newton** method to the NLSP, **substituting the true Hessian**

A

$$\nabla^2 f(x) = J(x)^T J(x) + \sum_{j=1}^{m} r_j(x) \nabla^2 r_j(x)$$

by the approximation that neglects the term **A** : $\nabla^2 f(x) \approx J(x)^T J(x)$

,that is the **Gauss-Newton search direction** is :

$$d^K_{\text{G-N}} = -\left[J(x^k)^T J(x^k)\right]^{-1} J(x^k)^T r(x^k)$$

The G-N method (II)

- **Considerations**: the approximation $\nabla^2 f(x) \approx J(x)^T J(x)$ is appropiate when the term $J(x)^T J(x)$ dominates over $\sum_{j=1}^{m} r_j(x) \nabla^2 r_j(x)$ in the expression of the Hessian. This happends:
  - when the residuals r_j are small.
  - when each r_j is nearly a linear function, so that $\|\nabla^2 r_j(x)\|$ is small.

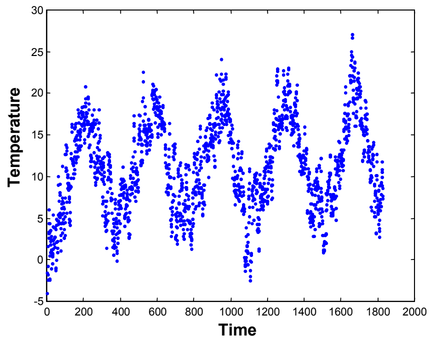
The G-N method (III)

- **Advantages**:
  - There is **no need to compute** the second derivatives $\nabla^2 r_j(x)$.
  - The G-N method can be shown to be **globally convergent** under certain conditions over the rank of $J(x^k)$.
  - The speed of convergence depends on how much the leading term $J(x)^T J(x)$ dominates.

  When $\sum_{j=1}^{m} r_j(x^*) \nabla^2 r_j(x^*) = 0$ the convergence is quadratic.

Example: the SIDS problem

- Real data (high dispersion):



Nonlinear Least-squares problems

Example: the SIDS problem, large residuals

- The residuals at the optimal solution are large:

  $\| r(x^*) \|_2 = 117.105, \quad \| r(x^*) \|_\infty = 8.280$

  nevertheless, the approximation $\nabla^2 f(x^k) \approx J(x^k)^T J(x^k)$ is very good near the solution:

  $\| \nabla^2 f(x^0) - J(x^0)^T J(x^0) \|_F = 1665.46$

  $\| \nabla^2 f(x^*) - J(x^*)^T J(x^*) \|_F = 4.129\times10^{-5}$

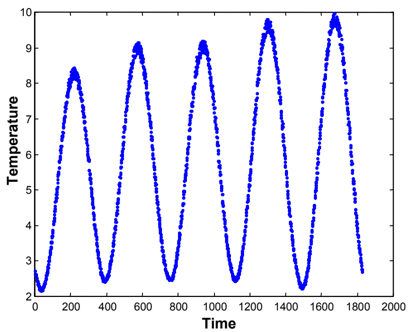
SIDS, large residuals	Exec. time (seconds)	$f(x^*)$	$\|\nabla f(x^*)\|$
Steep. Descent	42.90	6856.886	2.899×10^{-3}
Quasi-Newton	20.15	6856.886	1.192×10^{-4}
Gauss-Newton	16.63	6856.886	2.564×10^{-5}
Modified Newton	3.07	6856.886	9.524×10^{-5}

Graph

Nonlinear Least-squares problems

Example: the SIDS problem, small residuals

- Simulated data with low dispersion:



Nonlinear Least-squares problems

Example: the SIDS problem, small residuals

- The residuals are small:

  $\| r(x^*) \|_2 = 3.108, \ \| r(x^*) \|_\infty = 0.190$

- The approximation $\nabla^2 f(x^k) \approx J(x^k)^T J(x^k)$ is very good near the solution:

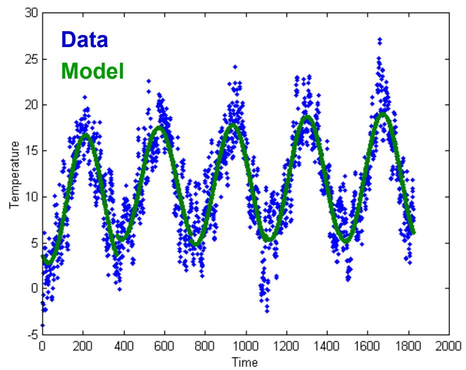
  $\| \nabla^2 f(x^0) - J(x^0)^T J(x^0) \|_F = 849.169$

  $\| \nabla^2 f(x^*) - J(x^*)^T J(x^*) \|_F = 5.813\times10^{-7}$

SIDS, small residuals	Exec. time (seconds)	$f(x^*)$	$\|\nabla f(x^*)\|$
Steep. Descent	18.34	4.832	3.209×10^{-6}
Quasi-Newton	15.49	4.832	7.137×10^{-7}
Gauss-Newton	16.81	4.832	4.912×10^{-7}
Modified Newton	3.25	4.832	5.689×10^{-9}

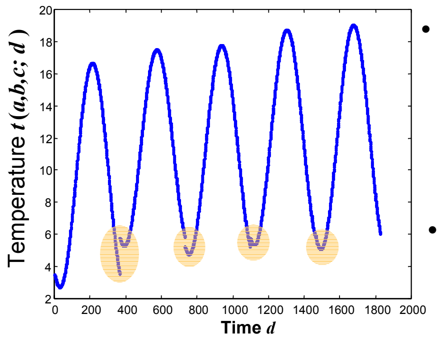
Nonlinear Least-squares problems

Solution for the SIDS problem



Nonlinear Least-squares problems

Adjusted model for temperature



- The adjusted model for the temperature presents discontinuities in the connecting points between different years.
- This problem can be avoided by **introducing constraints on the SIDS problem**

Algorithms for Constrained Nonlinear Optimization

Constrained Nonlinear Optimization

Fundamentals:

Formulation of the NOP

- The general (standard) form of the NOP is :

$$(\text{NOP})\begin{cases}\min\limits_{x\in\Re^n} & f(x) & \textbf{Objective function}\\ \text{subject to:} & h(x)=0 & \textbf{Equality constraints}\\ & g(x)\le 0 & \textbf{Inequality constraints}\end{cases}$$

where x are the **decision variables**, or simply, **variables**, and

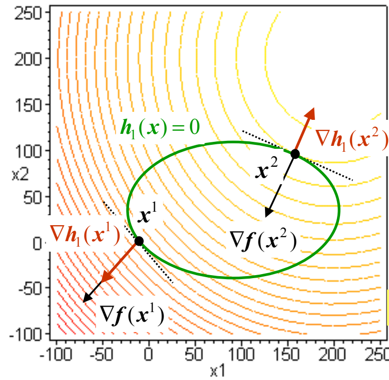
$$f:\Re^n\to\Re \qquad h:\Re^n\to\Re^m \qquad g:\Re^n\to\Re^l$$

- (NOP) will also be expressed as:

$$(\text{NOP})\ \min_x\{f(x)\mid x\in X\}\ ;\ \ X=\{x\in\Re^n\mid h(x)=0,\ g(x)\le 0\}$$

Fundamentals:

Geometry of the optimality conditions (I)

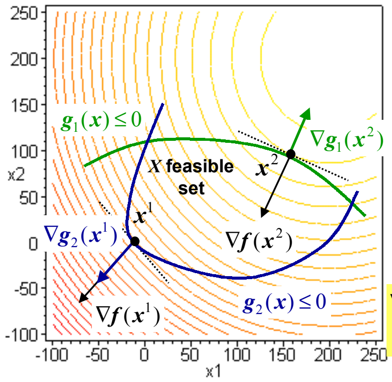


- One equality constraint
- Two stationary points x^1 and x^2 (x^2 minimum if f convex.)
- Relation between $\nabla f(x^k)$ and $\nabla h_1(x^k)$ at x^k, stationary point:

$$\nabla f(x^k)+\lambda_1\nabla h_1(x^k)=0$$

Fundamentals:

Geometry of the optimality conditions (II)



- Two inequality constraints defining X
- One stationary point, x^2 (x^1 is no longer stationary)
- Relation between $\nabla f(x^k)$ and $\nabla g_j(x^k)$ at x^k, stationary point:

$$\nabla f(x^k) + \mu_j \nabla g_j(x^k) = 0$$
$$\boxed{\mu_j \ge 0}$$

for all g_j active at x^k

Fundamentals:

Optimality conditions : the KKT conditions

Usual stopping criterium

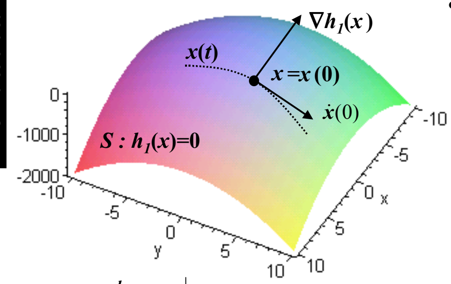
- **First-Order Necessary Conditions**
  "*Suppose that x^* is a local minimizer of NOP and that x^* is* regular, *then there are Lagrange multipliers vectors $\lambda \in \Re^m$ and $\mu \in \Re^l$ such that :*

$$i) \quad \nabla f(x^*) + \lambda^{*T} \nabla h(x^*) + \mu^{*T} \nabla g(x^*) = 0$$
$$ii) \quad \mu^{*T} g(x^*) = 0$$
$$iii) \quad \mu^* \ge 0$$

- This are the famous **Karush-Kuhn-Tucker conditions** (**KKT** for short).

Fundamentals:

Another interpretation of the KKT conditions



- Consider one equality constraint and a feasible point x
- Let $x(t)$ be *differentiable curve* over the feasible surface S at x.
- The derivative of $x(t)$ at t=0 is:

$$\dot{x}(0) = \frac{d}{dt} x(t)\Big|_{t=0}$$

- Then : $\frac{d}{dt} f(x(t))\Big|_{t=0} = \nabla f(x)\dot{x}(0)$. The **KKT** conditions impose that, **if x is a minimizer, this derivative must be nonnegative for all possible diff. curves $x(t)$.**

Fundamentals:

Optimality conditions

- **Regularity condition:**
  any feasible point $x \in X$ is said to be regular if the gradient vectors:

$$\nabla h_i(x) \quad , \quad i = 1,2,\ldots,m$$
$$\nabla g_j(x) \quad , \quad j \in J = \{ j \mid g_j(x) = 0 \}$$

Active constraints

are linearly independent.

Motivation

- Remember the likelihood maximization problem introduced previously:

$$\text{(LCNOP)}\quad \begin{cases} \max\limits_{\substack{\grave{u}\in\Re^m\\ \alpha,\beta,\sigma}} & L_n(\grave{u},\alpha,\beta,\sigma)=\prod\limits_{i=1}^{n}\sum\limits_{j=1}^{m}\gamma_{ij}\dfrac{1}{\sigma\sqrt{2\pi}}\left(e^{-\frac{1}{2}\frac{(y_i-\alpha-\beta s_j)^2}{\sigma^2}}\right)\omega_j \\ \text{subject to:} & \sum\limits_{j=1}^{m}\omega_j=1 \\ & \omega_j\ge 0,\ j=1,\dots,m \\ & \sigma\ge 0 \end{cases}$$

This is an exemple of *Linearly Constrained NOP Problem (LCNOP).*

- We will use the Reduced Gradient method to illustrate the rationale of the algorithms for (LCNOP).

The Reduced Gradient (RG) method

- The Reduced Gradient method solves the (LCNOP) problem:

$$\text{(LCNOP)}\ \min_x\{f(x)|x\in X\}\ ;\ X=\{x\in\Re^n|Ax=b, l\le x\le u\}$$

using the following strategy:

1. Find a first feasible solution $x^k\in X$ (current solution).

2. If the current solution satisfies the KKT conditions, x^k, STOP

3. Otherwise, find a new feasible solution that improves the current objective function value, and take this new point as the current solution:

- **3.1.** Find a **feasible descent** search direction d^k
- **3.2.** Perform a linesearch from x^k along d^k : α^k
- **3.3.** Update the current solution: $x^{k+1} := x^k + \alpha^k d^k$. Goto 2

Feasible direction

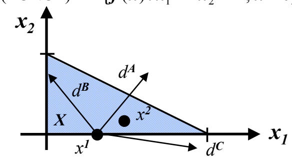
- **Feasible direction :** $d^k\in\Re^n$ is a feasible direction at x^k for the problem

$$\text{(LCNOP)}\ \min_x\{f(x)|x\in X\}\ ;\ X=\{x\in\Re^n|Ax=b, l\le x\le u\}$$

If :

$$\exists\bar{\alpha}\in\Re^+\,|\,\forall\alpha\in[0,\bar{\alpha}]: x^k+\alpha d^k\in X$$

$$\text{(LCNOP)}\min\{f(x): x_1+2x_2\le 2,\ x\ge 0\}$$



Over x^1 :	Over x^2 :
d^A: FEASIBLE d^B: FEASIBLE	$\forall d\in\Re^n$ FEASIBLE
d^C: INFEASIBLE	

Feasible descent direction of the RG

- **Nondegeneracy assumption:** at each iterate x^k, the RG method assumes that there exists a partition of the variables and columns of the coefficient matrix A:

$$x^k=\left[\frac{y}{z}\right]\quad \begin{array}{ll} y\in\Re^m & \text{(dependent variables)}\\ z\in\Re^{n-m} & \text{(independent variables)}\end{array}$$

$$Ax=[A_y\ |\ A_z]\left[\frac{y}{z}\right]=A_y y+A_z z=b$$

such that:

$$\begin{array}{ll} i. & A_y\in\Re^{m\times m}\ \text{is non-singular}\\ ii. & l_y<y<u_y\end{array}$$

Linearly constrained NOP

Feasible descent direction of the RG

- **Reduced gradient**: given the partition $x^k=[y\ z]^T$, the reduced gradient is the vector $r\in\Re^{n-m}$ defined as:

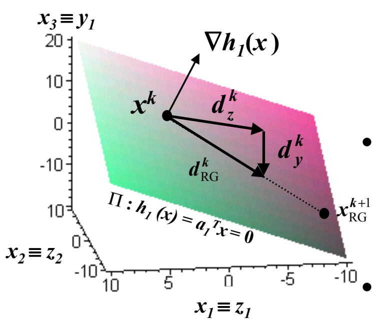
$$r^T = \nabla_z f(y,z) - \nabla_y f(y,z) A_y^{-1} A_z$$

- **Search direction**: it is obtained after the reduced gradient as:

$$d_{\mathrm{RG}}^k = \begin{bmatrix} d_y^k \\ d_z^k \end{bmatrix} = \begin{bmatrix} -A_y^{-1}A_z(-r) \\ \hline -r \end{bmatrix}$$

Linearly constrained NOP

Geometrical interpretation of d^k_{RG}



- Consider one linear constraint $a_1^T x = 0$ defining the feasible plane Π, and the feasible point x^k, where $z=[x_1\ x_2]^T$ and $y=x_3$
- The step d_z^k moves away from Π...

  ... and the step d_y^k corrects the step so that d^k_{RG} lies in Π
- And finally, a linesearch is performed to find x^{k+1}

Linearly constrained NOP

Feasible descent direction of the RG

- **Properties of d^k_{RG}:**
  - **d^k_{RG} is a descent direction**:

$$\nabla f(x^k) d_{RG}^k = [\nabla_y f(y,z) \mid \nabla_z f(y,z)] \begin{bmatrix} -A_y^{-1}A_z(-r) \\ \hline -r \end{bmatrix} =$$
$$= -\nabla_y f(y,z) A_y^{-1} A_z (-r) + \nabla_z f(y,z)(-r) =$$
$$= \underbrace{[\nabla_z f(y,z) - \nabla_y f(y,z) A_y^{-1} A_z]}_{r}(-r) = -r^T r =$$
$$= -\|r\| < 0 \quad \text{if} \quad r \neq 0$$

  - **d^k_{RG} is feasible for $Ax=b$:**

$$A(x^k + \alpha d_{RG}^k) = b + \alpha [A_y \mid A_z] \begin{bmatrix} -A_y^{-1}A_z(-r) \\ \hline -r \end{bmatrix} = b + \alpha \overbrace{\begin{bmatrix} -\overbrace{A_y A_y^{-1}}^{I} A_z + A_z \end{bmatrix}}^{0}(-r) = b$$

Linearly Constrained NOP

Feasible descent direction of the RG

- **Properties of d^k_{RG} (cont.):**
  - **d^k_{RG} may be infeasible for some bound**:

    this problem can be avoided by a slightly modification in the definition of the search direction.

Motivation : nonlinear regression with constraints

- Remember the SIDS problem

$$\min_{\substack{a_i, b_i, c_i \\ i=1,2,\ldots,5}} \frac{1}{2} \sum_{i=1}^{5} \sum_{j=1}^{365} \left[t_{ij}(a_i, b_i, c_i) - t_{ij} \right]^2$$

where the model to be adjusted was:

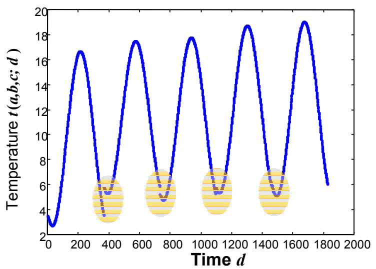
$$t_{ij}(a_i, b_i, c_i) = a_i \cos\left(\frac{2\pi}{365} j - b_i \right) + c_i$$

$$i = 1,2,3,4,5$$

$$j = 1,2,\ldots,365$$

Motivation : nonlinear regression with constraints

- The solution of this problem through unconstrained nonlinear optimization techniques presented discontinuities between the harmonic models for each year:



Motivation : nonlinear regression with constraints

- This problem can be overcame by introducing a set of **nonlinear constraints** that forces the continuity of the adjusted models:

$$h_i(x) = t_{i365}(a_i, b_i, c_i) - t_{i+1,1}(a_{i+1}, b_{i+1}, c_{i+1}) = 0 \quad , \quad i = 1,2,3,4$$

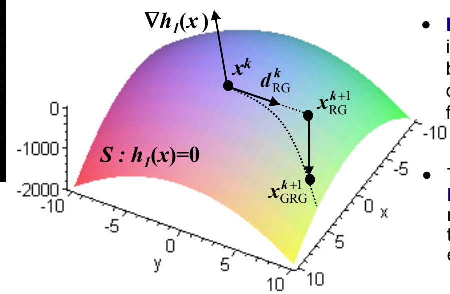
that is:

$$h_i(x) = a_i \cos\left(\frac{2\pi}{365} 365 - b_i \right) + c_i - a_{i+1} \cos\left(\frac{2\pi}{365} - b_{i+1} \right) - c_{i+1} = 0$$

$$i = 1,2,3,4$$

which are nonlinear w.r.t. the decision variables b_i

Generalized Reduced Gradient



- **Problem** : the iterate x^{k+1}_{RG} found by the RG method does not lie on the feasible surface S
- The **Generalized Reduced Gradient** method **(GRG)** tries to keep feasibility at each iterated point.
- The GRG method finds the **new feasible point x^{k+1}_{GRG}, solving numerically (Newton-Raphson)**, at each iteration, the nonlinear system of equations $h(x)=0$, starting at x^{k+1}_{RG} .

Generally Constrained NOP:

Augmented Lagrangian Methods (I)

- Consider, for simplicity, the (GCNOP) with only equality constraints : (GCNOP) $\min_{x} \{ f(x) \mid h(x) = 0 \}$
- The **Augmented Lagrangian** function is defined as :

$$L_A(x, \lambda; c) = f(x) + \lambda^T h(x) + \frac{c}{2} \| h(x) \|^2$$

This Augmented Lagrangian is formed with :

- the Lagrangian function $L(x, \lambda) = f(x) + \lambda^T h(x)$, plus
- the quadratic term $(c/2)\|h(x)\|^2$ that penalizes the infeasibilities of the solution x

($(c/2)\|h(x)\|^2 = 0$ if x is a feasible solution)

Generally Constrained NOP:

Augmented Lagrangian Methods (II)

- The key idea of the Augmented Lagrangian method is to solve the original problem by solving the sequence of **Unconstrained Subproblems**:

$$(\text{US})^k \quad \min_{x} L_A(x, \lambda^k; c^k)$$

, with an increasing sequence $\{c^k\}$, in such a way that **the sequence $\{x^k, \lambda^k\}$ converges to** $\{x^*, \lambda^*\}$ a solution that satisfies the KKT conditions of the original problem.

Generally Constrained NOP:

Augmented Lagrangian Methods (III)

- Note that, for a sufficiently large c^k, the penalty term will dominate in the minimization of $(\text{US})^k$, and then $L_A(x^k, \lambda^k) \approx f(x^k) + \lambda^{k\text{T}} h(x^k)$.
- In this case, the first order optimality conditions of the $(\text{US})^k$ at x^k will be:

$$\nabla L_A(x^k, \lambda^k; c^k) \approx \nabla f(x^k) + \lambda^{k^T} \nabla h(x^k)$$

which are nothing but the KKT conditions for the original problem.

Generally Constrained NOP:

Augmented Lagrangian Methods (III)

- **Framework of the Augmented Lagrangian algorithm**

Given c^0 and starting points x^0 and λ^0, $k:=0$

Do Until (x^k, λ^k) satisfies the KKT conditions.

Compute the new iterate as $x^{k+1} := \operatorname{argmin} L_A(x, \lambda^k, c^k)$

Update the Lagrange multipliers to obtain λ^{k+1}

Choose new penalty parameter $c^{k+1} \geq c^k$

$k:=k+1$

End Do

Generally Constrained NOP:

Projected Lagrangian Methods (I)

- Consider, again, the (GCNOP) with only equality constraints : (GCNOP) $\min_{x}\{f(x) \mid h(x)=0\}$
- **Projected Lagrangian** methods solve the original problem by solving the sequence of **Linearly Constrained Subproblems**:

$$(\text{LCS})^k \begin{cases} \min_x & L_P(x,\lambda^k;c^k) \\ \text{subj. to:} & \nabla h(x^k)(x-x^k)+h(x^k)=0 \end{cases}$$

where the linear constraints comes from the Taylor's series expansion of $h(x)$ around a given point x^k.

Generally Constrained NOP:

Projected Lagrangian Methods (II)

- The most effective expression of the objective function of subproblems (LCS)^k is the **Modified Augmented Lagrangian**:

$$L_P^k(x;\lambda^k,x^k,c)=f(x)+\lambda^{k^T}h^k(x)+\frac{c}{2}\left\|h^k(x)\right\|^2$$

that resembles the Augmented Lagrangian function, excepts that the expression of the nonlinear constraints $h(x)$ has been substituted by $h^k(x)$, defined as:

$$h^k(x)=h(x)-\underbrace{\left[\nabla h(x^k)(x-x^k)+h(x^k)\right]}_{\text{Linear approximation of } h(x) \text{ around } x^k}$$

- Near the solution ($(x^k, \lambda^k) \approx (x^*, \lambda^*)$) the method presents quadratic order of convergence if c=0.

Generally Constrained NOP:

Projected Lagrangian Methods (III)

- **Projected Lagrangian algorithm**

Given c^0 and starting points x^0 and λ^0, $k:=0$

Do Until (x^k, λ^k) satisfies the KKT conditions.

  Solve (LCS)^k to obtain x^{k+1}

  Take λ^{k+1} as the Lag. mult. at the opt. sol. of (LCS)^k

  If $(x^k, \lambda^k) \approx (x^*, \lambda^*)$ **then** set $c^{k+1} = 0$

  Else choose $c^{k+1} \geq c^k$

  $k:=k+1$

End Do

Generally Constrained NOP:

Sequential Quadratic Programming (I)

- Consider, the problem : (GCNOP) $\min_{x}\{f(x) \mid h(x)=0\}$
- **Sequential Quadratic Programming** solves the original problem by solving the sequence of **Quadratic Linearly Constrained Subproblems**:

$$(\text{QLCS})^k \begin{cases} \min_x & \frac{1}{2}(x-x^k)^T W^k (x-x^k)+\nabla f(x^k)(x-x^k) \\ \text{subj. to:} & \nabla h(x^k)(x-x^k)+h(x^k)=0 \end{cases}$$

where the matrix $W^k \in \Re^{n \times n}$ denotes the **Hessian of the Lagrangian function** at (x^k, λ^k) :

$$W^k=\nabla^2_{xx}L(x^k,\lambda^k)=\nabla^2 f(x^k)+\sum_{j=1}^{m}\lambda_j^k\nabla^2 h(x^k)$$

Generally Constrained NOP:

Sequential Quadratic Programming (II)

- **Framework of the Sequential Quadratic Programming**

Given starting points x^0 and λ^0, $k:=0$

Do Until (x^k , λ^k) satisfies the KKT conditions.

Solve (QLCS) ^{k} to obtain x^{k+1}.

Take λ^{k+1} as the Lag. mult. at the opt. sol. of (QLCS) ^{k}

$k:=k+1$

End Do

Generally Constrained NOP:

SQP and Projected Lagrangian

- The strategy of the **SQP** is similar to the one used in the **Projected Lagrangian** methods:
  - **Advantages of SQP**: it is easier to optimize the quadratic subproblem (QLCS)^k than the general (LCS)^k, due to the existence of specialised quadratic programming techniques.
  - **Disadvantages of SQP**: the computation of the quadratic objective function needs the second derivatives (or its numerical approximation) of the objective function $f(x)$ and constraints $h(x)$.
- Both methods can be proved to converge quadratically near the solution.

Solvers for Nonlinear Optimization

Solvers for nonlinear optimization

- **Optimization libraries.**
  - Solving the SIDS problem with the NAG Library
    - Without smoothness constraints.
    - With smoothness constraints.
- **Modeling languages.**
  - Maximization of the constrained likelihood function with AMPL

Optimization libraries:

Description

- Optimization libraries provides subroutines that can be called from the user's own code (mainly in FORTRAN, C or MATLAB).
- In order to solve a problem with an optimization library, the user must provide:
  - The **data structure** with the relevant information about the problem (matrix A in the (LCNOP), lower and upper bounds, etc.)
  - **Subroutines** that, **given a vector** x^k, **returns** all the information needed by the algorithm. This information could be:
    - **At least** $f(x^k)$ and the constraints value $h(x^k)$ and $g(x^k)$.
    - **Usually** the gradients $\nabla f(x^k)$ and Jacobians $\nabla h(x^k)$ and $\nabla g(x^k)$.
    - **Rarely**, the Hessians $\nabla^2 f(x^k)$, $\nabla^2 h(x^k)$ and $\nabla^2 g(x^k)$.

Optimization libraries:

Optimization libraries

- Some of most outstanding optimization libraries are:
  - For Unconstrained Optimization:
    - The optimization subroutines in the NAG and **HARWELL** libraries.
  - For Constrained Optimization:
    - **GRG**, **CONOP**: Generalized Reduced Gradient.
    - **LANCELOT** : Augmented Lagrangians.
    - **MINOS** : Projected Lagrangians.
    - **SNOPT**: Sequential Quadratic Programming.
- But, if you really are interested in knowing all the available optimization software, visit the *NEOS Guide* at www-fp.mcs.anl.gov/otc/Guide/ (a really impressive site in optimization!!)

Optimization libraries:

Solving the SIDS problem with the NAG libraries

- Subroutines for the unconstrained SIDS problem :
  - E04JAF : **Quasi-Newton**, using function values only.
  - E04GCF: **Gauss-Newton**, using function values and first derivatives.
- Subroutines for the constrained SIDS problem:
  - E04UCF: **Sequential Quadratic Programming**, using function values and first derivatives.
- We will see how to solve the SIDS problem calling these subroutines from MATLAB.

Optimization libraries:

Subroutine E04JAF (QN method)

- **Main program**: file SIDS_e04jaf.m

```
% Solving the SIDS problem with the NAG Foundation Library
% Routine E04jaf : Quasi-Newton method using function values only

'Example program for the NAG Foundation Library Routine e04jaf'
SIDS_t;                  % Here the observed data t_{ij} is loaded
x=zeros(15,1);           % Initial point
time=cputime;            % To know the total execution time.
[xQN,f] = e04jaf(x);     % This is the call to the subroutine
Optimal_Objective_Function=f % We print f(x*)...
At_the_Point_X=xQN           % ... the optimal solution x*...
time = cputime-time          % ... and the execution time
```


Optimization libraries:

Subroutine E04JAF (QN method)

- **User's subroutines**: computation of $f(x^k)$ file funct1.m

```
% Objective function for the SIDS problem
function [fc] = funct1(n,xc)
global obs;
fc=0;
ww=(2*pi)/365;
for i=0:4
   aux = 3*i;
   a = xc(1+aux);
   b = xc(2+aux);
   c = xc(3+aux);
   aux2 = 365*i;
    for j=1:365
      calc=a*cos(ww*j-b)+c;
      fc=fc+(calc-obs(j+aux2))^2;
   end
end
fc=fc/2;
```


Optimization libraries:

Subroutine E04JAF (QN method)

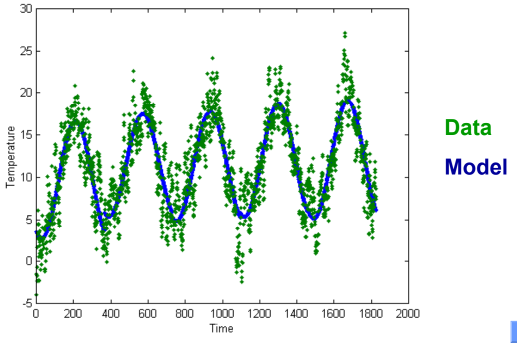
- **Output:**

```
» sids_e04jafe

ans =

Example program for the NAG
Foundation Library Routine
e04jaf


Optimal_Objective_Function =

  6.8569e+003
```

```
At_the_Point_X =
   -6.9691
    0.4920
    9.7255
   -6.0992
    6.7046
   11.4230
   -6.4842
    0.3844
   11.2986
   -6.7013
   -5.8949
   12.0551
   -6.9693
    6.8088
   12.0827

time =
   18.8900
```

Optimization libraries:
Subroutine E04JAF (QN method)

- **Graphical representation :**



Optimization libraries:
Subroutine E04GCF (GN method)

- **Main program**: file SIDS_e04gcf.m

```
% Solving the SIDS problem with the NAG Foundation Library
% Routine E04gcf : Gauss-Newton method using function values
% and first derivatives.

'Example program for NAG Foundation Library routine e04gcf'

SIDS_t;             % Here the observed data t_{ij} is loaded
x = zeros(15,1);    % Initial point
m=length(obs);      % Number of observations
time = cputime;     % To know the total execution time.
[xGN,fsumsq,ifail] = e04gcf(m,x);  % Call to the routine
The_Sum_of_Squares=fsumsq     % We print f(x*)...
At_the_Point_X=xGN            % ... the optimal solution x*...
time = cputime-time           % ... and the execution time
```


Optimization libraries:
Subroutine E04GCF (GN method)

- **User's subroutines**: file lsfun2.m

```
%
% Residuals and its derivatives for the
SIDS problem.
%
% Input :
%
% m           : number of observations.
% n           : number of coefficients.
% xc (n)      : coefficients.
% ljc         : num. Of rows of the Jac..
%
% Output:
%
% fvecc (m)      : vector of residuals.
% fjacc (ljc,n) : Jacobian of the vector
of residual.
%
```

- The output is similar to the previous routine.

graph

```
function [fvecc,fjacc] = lsfun1(m,n,xc,ljc)
global obs;

ww=(2.*pi)/365.;
for i=0:4
   aux = 3*i;
   i_a = aux + 1;
   i_b = aux + 2;
   i_c = aux + 3;
   a = xc(i_a);
   b = xc(i_b);
   c = xc(i_c);
   aux2 = 365*i;
   for j=1:365
     k = aux2 + j;
     calc  = a*cos(ww*j-b)+c;
     fvecc( k) = calc-obs(k);
     fjacc( k, i_a) = cos(ww*j-b);
     fjacc( k, i_b) = a*sin(ww*j-b);
     fjacc( k, i_c) = 1;
   end
end
```

$r_j(x^k)$

$\left.\frac{\partial r_j(x)}{\partial x_i}\right|_{x^k}$

Optimization libraries:
Constrained SIDS with subroutine E04UCF

- **Main program**: file SIDS_e04ucf.m

```
% Solving the constrained SIDS problem with the NAG Foundation Library
% Routine E04ucf : SQP method using function values and first derivatives.

'Example program for the NAG Foundation Library Routine e04ucf'
sids_t;           % Loading the observed data
n=15;             % Number of variables.
nclin=0;          % Number of linear constraints.
ncnln=4;          % Number of nonlinear constraints.
a=ones(nclin,1); % Coefficient matrix A (dummy).
bl=-ones(19,1)*1.0E+25; % Default lower bounds for variables and constraints.
bu= ones(19,1)*1.0E+25; % Default upper bounds for variables and constraints.
bl(16:19) = zeros(4,1); % Lower bound for each constraint.
bu(16:19) = zeros(4,1); % Upper bound for each constraint.
x=zeros(15,1);          % Initial point.
confun='SIDS_e04ucf_confune'; % User's subroutine for the constraints and Jacobian.
objfun='SIDS_e04ucf_objfune'; % User's subroutine for the o.f. and gradient.
```

Optimization libraries:
Constrained SIDS with subroutine E04UCF

- **Main program**: file SIDS_e04ucf.m (cont.)

```
% Optimization parameters:
    string = ' Infinite Bound Size = 1.0e25 ';
    e04uef(string);
    string = ' Print Level = 1 ';
    e04uef(string);
    string = ' Verify Level = -1 ';
    e04uef(string);
% Call to the optimizer:
    [iter,c,objf,objgrd,x,cjac,istate,clamda,r,ifail] = ...
    e04ucf(bl,bu,confun,objfun,x,ncnln,a);
```


1BWSA Barcelona June 2002

Optimization libraries:
Constrained SIDS with subroutine E04UCF

- **User's subroutines**: computation of $f(x^k)$ and $\nabla f(x^k)$: file SIDS_e04ucf_objfune.m

```
%
% Function "objfun" for the constrained SIDS problem
%
% Input:
%
% mode      : information required
% n         : number of variables
% x (n)     : current iterate x^k
% nstate    : information about the current iterate
%
% Output
%
% objf       : objective function at x^k.
% objgrd (n) : gradient vector at x^k
%
 function [mode,objf,objgrd] = objfun(mode,n,x,objf,objgrd,nstate)
```


1BWSA Barcelona June 2002

Optimization libraries:
Constrained SIDS with subroutine E04UCF

- **User's subroutines**: computation of $f(x^k)$ and $\nabla f(x^k)$: file SIDS_e04ucf_objfune.m

```
function [mode,objf,objgrd] = objfun(mode,n,x,objf,objgrd,nstate)
%
global obs;
ww=(2*pi)/365;
%
if mode==0 | mode==2 % Evaluation of f(x^k)
  F=0;
  for i=0:4
    aux = 3*i;
    a = x(1+aux);
    b = x(2+aux);
    c = x(3+aux);
    aux2 = 365*i;
    for j=1:365
        res = a*cos(ww*j-b)+c - obs(j+aux2);
        F=F+res^2;
    end
  end
  objf=F/2;
end
```


1BWSA Barcelona June 2002

Optimization libraries:
Constrained SIDS with subroutine E04UCF

- **User's subroutines**: computation of $f(x^k)$ and $\nabla f(x^k)$: file SIDS_e04ucf_objfune.m

```
if mode==1 | mode==2 % Evaluation of the gradient
  G=zeros(1,15);
  for i=0:4
    aux1 = 3*i;
    i_a = 1 + aux1;
    i_b = 2 + aux1;
    i_c = 3 + aux1;
    a = x(i_a);
    b = x(i_b);
    c = x(i_c);
    aux2 = i*365;
    for j=1:365
      res = a*cos(ww*j-b)+c - obs(j+aux2);
      G(i_a)=G(i_a)+res*cos(ww*j-b) ;
      G(i_b)=G(i_b)+res*sin(ww*j-b);
      G(i_c)=G(i_c)+res;
    end
    G(i_b) = a*G(i_b);
  end
end
objgrd = G;
```

Optimization libraries:
Constrained SIDS with subroutine E04UCF

- **User's subroutines**: computation of $h(x^k)$ and $\nabla h(x^k)$: file SIDS_e04ucf_confune.m

```
%
% Function "objfun" for the constrained SIDS problem
%
% Input:
%
% mode           : information required
% ncnln          : number of rows of the Jacobian.
% n              : number of variables
% nrows          : max(1,ncnln)
% needc(ncnln)   : flag to indicate the constraints to be evaluated.
% x (n)          : current iterate x^k
% nstate         : information about x^k.
%
% Output
%
% c (ncnln)      : value of the constraints over x^k
% cjac (nrowj,n) : Jacobian over x^k
%
 function [mode,c,cjac] = confun(mode,ncnln,n,nrowj,needc,x,c,cjac,nstate)
%
```


Optimization libraries:
Constrained SIDS with subroutine E04UCF

- **User's subroutines**: computation of $h(x^k)$ and $\nabla h(x^k)$: file SIDS_e04ucf_confune.m

```
if nstate==1
  cjac=zeros(ncnln,n);
end

ww = 2*pi/365;

for i=1:4
   aux = 3*(i-1);
   a0 = x(aux+1);
   b0 = x(aux+2);
   c0 = x(aux+3);
   a1 = x(aux+4);
   b1 = x(aux+5);
   c1 = x(aux+6);
   aux0 = ww*365.;
   aux1 = ww;
   cos0 = cos(aux0 - b0);
   sin0 = sin(aux0 - b0);
   cos1 = cos(aux1 - b1);
   sin1 = sin(aux1 - b1);
    if needc(i)>0
    % Continuity of y(j,t) at the "i"-th connect point
    if mode==0 | mode==2
          c(i) = a0*cos0 + c0 - ( a1*cos1 + c1);
      end
      if mode==1 | mode==2
          cjac(i,aux+1) = cos0;
          cjac(i,aux+2) = a0*sin0;
          cjac(i,aux+3) = 1.;
          cjac(i,aux+4) = -cos1;
          cjac(i,aux+5) = -a1*sin1;
          cjac(i,aux+6) = -1.;
       end
    end
end
```

$$h_i(x) = a_i \cos\left(\frac{2\pi}{365}365 - b_i\right) + c_i - a_{i+1}\cos\left(\frac{2\pi}{365} - b_{i+1}\right) - c_{i+1}$$

$$\left.\frac{\partial h_i(x)}{\partial x_j}\right|_{x^k}$$

Optimization libraries:
Constrained SIDS with subroutine E04UCF

- **Output (I):**

```
» sids_e04ucfE

ans =

Example program for the NAG Foundation Library Routine e04ucf



 Calls to E04UEF
 ---------------

       Infinite Bound Size = 1.0e25
       Print Level = 1
       Verify Level = -1

 *** E04UCF
 *** Start of NAG Library implementation details ***

 Implementation title: Microsoft Windows NT Powerstation
            Precision: FORTRAN Double Precision
         Product Code: FLNTI17DI
                 Mark: 17A

 *** End of NAG Library implementation details ***
```


Optimization libraries:
Constrained SIDS with subroutine E04UCF

- **Output (II) :**

```
Parameters
 ----------

 Linear constraints.....         0      Variables.............        15
 Nonlinear constraints..         4

 Infinite bound size....  1.00D+25      COLD start.............
 Infinite step size.....  1.00D+25      EPS (machine precision)  1.11D-16
 Step limit.............  2.00D+00      Hessian................        NO

 Linear feasibility.....  1.05D-08      Crash tolerance........  1.00D-02
 Nonlinear feasibility..  1.05D-08      Optimality tolerance...  3.26D-12
 Line search tolerance..  9.00D-01      Function precision.....  4.38D-15

 Derivative level.......         3      Monitoring file........        -1
 Verify level...........        -1

 Major iterations limit.        85      Major print level......         1
 Minor iterations limit.        57      Minor print level......         0

 Workspace provided is     IWORK(      53),  WORK(     954).
 To solve problem we need  IWORK(      53),  WORK(     954).
```

Optimization libraries:
Constrained SIDS with subroutine E04UCF

- **Output (III):**

```
Exit from NP problem after    33 major iterations,
                              34 minor iterations.

 Varbl State      Value       Lower Bound   Upper Bound  Lagr Mult   Slack

 V   1   FR   -6.01843          None          None
 V   2   FR    6.86197          None          None
 V   3   FR   10.2776           None          None
 V   4   FR   -6.45854          None          None
 V   5   FR    6.68295          None          None
 V   6   FR   11.2311           None          None
 V   7   FR   -6.46699          None          None
 V   8   FR    6.67074          None          None
 V   9   FR   11.3104           None          None
 V  10   FR   -7.00733          None          None
 V  11   FR    6.65565          None          None
 V  12   FR   11.8928           None          None
 V  13   FR    7.33845          None          None
 V  14   FR    9.92259          None          None
 V  15   FR   11.8731           None          None
```

x^*

Optimization libraries:
Constrained SIDS with subroutine E04UCF

- **Output (IV):**

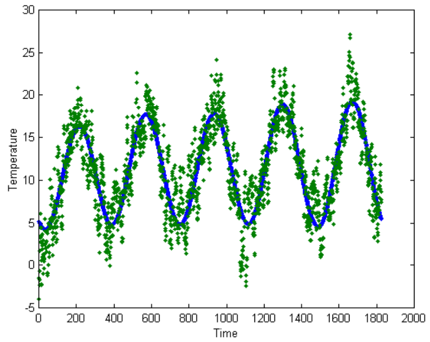
```
N Con State      Value        Lower Bound   Upper Bound   Lagr Mult    Slack

 N   1   EQ   -1.267431E-11       .             .           201.5     1.2674E-11
 N   2   EQ    2.171596E-11       .             .           131.5    -2.1716E-11
 N   3   EQ    7.942980E-12       .             .           135.8    -7.9430E-12
 N   4   EQ   -3.250733E-13       .             .            76.53    3.2507E-13

 Exit E04UCF - Optimal solution found.

 Final objective value =      7082.574
»
```

$h(x^*)$ λ^*

$f(x^*)$

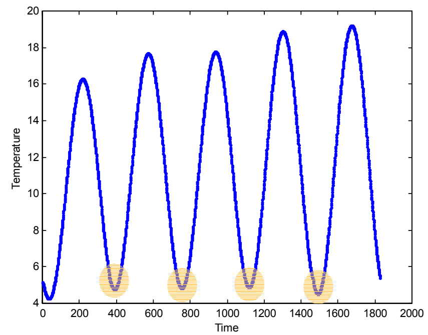
Optimization libraries:
Constrained SIDS with subroutine E04UCF

- **Graphical representation of the solution :** data and model



Optimization libraries:
Constrained SIDS with subroutine E04UCF

- **Graphical representation of the solution :** model



Now the model is continuous

Modeling languages: Introduction

- **Modeling languages** can be saw as a friendly interface between the user and the optimization libraries.
- The way this applications work is :
  - The user **defines the optimization problem** to be solved (objective function and constraints) in a notation very similar to the natural mathematical notation.
  - Then, he **selects the solver** to be used (MINOS, LANCELOT, CONOPT, etc).
  - The application **automatically translates the model defined by the user** to the specific input data structure needed by the selected solver.

Modeling languages: Advantages/Disadvantages

- **Advantages:** the developing time is shortened, because
  - The definition of the model is very easy, because the syntax of the modeling language resembles the usual mathematical notation.
  - The model definition is independent of the solver to be used. That means that, after defining just once the optimization problem, the user are able to solve it with a great variety of solvers, forgetting all the annoying issues related with the specific data structure of each solver.
- **Disadvantages:** the execution time increases compared with the one obtained directly using the optimization libraries.
- **Conclusion**: this approach is appropriate:
  - For small scale problems, where the execution time is not critical.
  - To develop prototype implementations to achieve a deeper comprehension of the model, before its implementation in FORTRAN o C.

Modeling languages: GAMS/AMPL

- **Modeling languages** : the two modeling languages most widely used are:
  - **GAMS** (www.gams.com) :
    General Algebraic Modeling System
  - **AMPL** (www.ampl.com) :
    A Modeling Language for Mathematical Programming
- We will use AMPL to illustrate the use of this sort of software, solving the constrained likelihood maximization problem

Modeling languages: A Log-Likelihood Function

(Langohr, Gómez, 1BWSA Poster Session, thursday)

$$
\text{(LCNOP)}\left\{
\begin{aligned}
\max_{\grave{u},\alpha,\beta,\sigma}\quad & l(\grave{u},\alpha,\beta,\sigma)=\sum_{i=1}^{n}\Bigg[\varepsilon_i\xi_{1i}\ln\left(\sum_{j=1}^{m}\frac{\gamma_{ij}}{\sigma}\exp\left[\frac{\ln(y_{obs,i}+z_{obs,i}-s_j)-\alpha-\beta\ln(s_j)}{\sigma}-e^{\frac{\ln(y_{obs,i}+z_{obs,i}-s_j)-\alpha-\beta\ln(s_j)}{\sigma}}\right]\omega_j\right)\\
&\quad+\varepsilon_i\xi_{2i}\ln\left(\sum_{j=1}^{m}\gamma_{ij}\exp\left[-e^{\frac{\ln(y_{obs,i}+z_{obs,i}-s_j)-\alpha-\beta\ln(s_j)}{\sigma}}\right]\omega_j\right)\\
&\quad+\varepsilon_i(1-\xi_{1i})(1-\xi_{2i})\ln\left(\sum_{j=1}^{m}\gamma_{ij}\left(1-\exp\left[-e^{\frac{\ln(y_{obs,i}+z_{obs,i}-s_j)-\alpha-\beta\ln(s_j)}{\sigma}}\right]\right)\omega_j\right)\\
&\quad+(1-\varepsilon_i)\ln\left(\sum_{j=1}^{m}\gamma_{ij}\omega_j\right)\Bigg]\\
\text{subj.to:}\quad & \sum_{j=1}^{m}\omega_j=1\\
& \omega_j\ge 0\quad,\quad j=1,2,\ldots,m\\
& \sigma\ge 0
\end{aligned}
\right.
$$

With α, β, σ, and ω decision variables, and ε, ξ and γ known parameters

Modeling languages:
User's data files with AMPL

- In order to solve the (LCNOP) Log-Likelihood problem with AMPL, the user must first define:
  - A Model file with:
    - The declaration of the decision variables $\omega,\alpha,\beta,\sigma$ and its bounds.
    - The mathematical expressions of the o.f. $l(\omega,\alpha,\beta,\sigma)$
    - The mathematical expression of the linear constraint.
  - A Data file with the definition of all the know parameters of the model (m, n and ε, ξ and γ).
  - A Run file which is a script file, a sort of main program, with the list of commands to be executed to solve the defined problem.
- And then, solve the problem with AMPL

Modeling languages:
The model file for the Log-Likelihood problem

- **Definition of the model:** file 1bwsa.mod

```
# Parameters of the model ##############################
set V;          # set for variables
param Lb{V};    # bounds for variables
param Ub{V};
param M ;       # number of possible covariate values
param N ;       # number of observations
param pi := 3.14159265;
param gamma{1..N,1..M};   # matrix for censoring pattern of covariate
param y{1..N};          # time from HIV+ to observed value for AIDS
param epsi{1..N};       # observation indicator for y
param xi_1{1..N};       # exact observation indicator for y
param xi_2{1..N};       # right-censored observation indicator for y
param hivn{1..N};       # time to HIV: left endpoint
param hivp{1..N};       # time to HIV: right endpoint
param s{1..M};          # values of time till HIV
param logsum{1..N,1..M}; # possible values for log(time from HIV till AIDS)
param RR_zeta;          # relative risk for current status covariate (csc)
param acc_fac_z;        # accelerating factor for csc
```


Modeling languages:
The model file for the Log-Likelihood problem

- **Definition of the model:** file 1bwsa.mod (cont.)

```
# Decision variables ################################
    var alpha;
    var beta;
    var sigma >= 0, :=1;
    var omega{j in 1..M} >=0;
# Objective function #################################
    maximize logLikelihood:
      sum{i in 1..N}
        (epsi[i]*xi_1[i]*log(1/sigma*sum{j in 1..M}
         gamma[i,j]*exp((logsum[i,j]-alpha-beta*log(s[j]))/sigma-
    exp((logsum[i,j]-alpha-beta*log(s[j]))/sigma))*omega[j])+
         epsi[i]*xi_2[i]*log(sum{j in 1..M} gamma[i,j]*exp(-
    exp((logsum[i,j]-alpha-beta*log(s[j]))/sigma))*omega[j])+
         epsi[i]*(1-xi_1[i])*(1-xi_2[i])*log(sum{j in 1..M}
      gamma[i,j]*(1-exp(-exp((logsum[i,j]-alpha-
    beta*log(s[j]))/sigma)))*omega[j])+
        (1-epsi[i])*log(sum{j in 1..M} gamma[i,j]*omega[j]));
# Linear constraint ##################################
    subject to sum1:
       sum {i in 1..M} omega[i] = 1;
```


Modeling languages:
The data file for the Log-Likelihood problem

- **Data:** file 1bwsa.dat

```
set V:= 1,2,3;

param:  Lb  Ub :=
   1    -15   15
   2    -15   15
   3      0   15;

param M := 215;
param N := 361;

param:  hivn  hivp   y    epsi  xi_1  xi_2:=
   1     1     45    36     1     1    0
   2     1     83    36     1     0    1
   3     1     112   1      1     1    0
   4     1     84    9      1     1    0
   5     1     37    76     1     0    1
   ....................................
 359   107     215   9999   0     0    0
 360    83     215   9999   0     0    0
 361     8     215   9999   0     0    0;
```

215+3 = 218 decision variables

Modeling languages:

The script file for the Log-Likelihood problem

- **Execution file :** file 1bwsa.run

```
# First, the model and data are loaded :
    model 1bwsa.mod;
    data 1bwsa.dat;
# Now the fixed parameters of the model are computed :
    for{j in 1..M}{
     let omega[j] := 1/M;
     let s[j] := j;
    }
    for{i in 1..N}{
      for{j in 1..M}{
        if hivn[i] <= s[j] and s[j] <= hivp[i]
        then let gamma[i,j]:= 1;
        else let gamma[i,j]:= 0;
      }
    }
    for{i in 1..N}{
      for{j in 1..M}{
        if gamma[i,j]==1 and epsi[i]==1
        then let logsum[i,j]:= log(y[i]+hivp[i]-s[j]);
        else let logsum[i,j]:= 0;
      }
    }
```


Modeling languages:

The script file for the Log-Likelihood problem

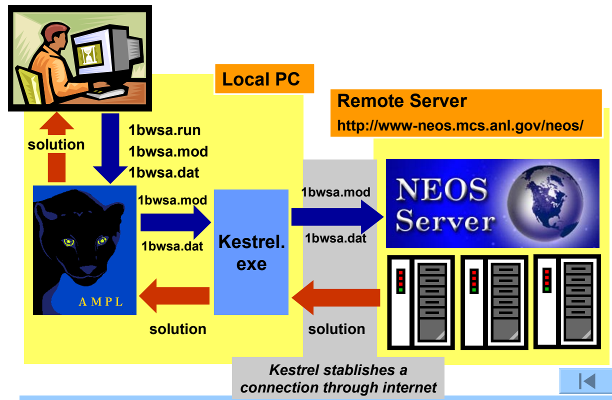
- **Execution file :** file 1bwsa.run (cont.)

```
# Here we choose the optimizer
  option solver kestrel; # remote optimization at the NEOS Server...
  option kestrel_options 'solver=snopt'; #...with the SNOPT optimizer
  option snopt_options "version";
# and now, we solve ...
  solve;
# ...and print the solution :
  let RR_zeta:= exp(-beta/sigma);
  let acc_fac_z:= exp(beta);
  printf "The estimated values of the model are (alpha, beta, sigma)
  = (%6.4f, %6.4f, %6.4f).\n",alpha, beta, sigma;
  printf "The relative risk amounts to %5.4f,", RR_zeta; printf " the
  accelerating factor is %5.4f.\n", acc_fac_z;
  option omit_zero_rows 1;
  display omega;
```

What means "*remote optimization*"?

Modeling languages:

Remote optimization through the NEOS Server



Modeling languages:

Resolution of the Log-Likelihood problem

- **Results :**

Execution command line

```
DOS> ampl 1bwsa.run
```

Tracking information of the remote server

```
Job has been submitted to Kestrel
Kestrel/NEOS Job number    : 163559
Kestrel/NEOS Job password  : irQEgUIl
Check the following URL for progress report :
     http://www-neos.mcs.anl.gov/neos/neos-cgi/check-
status.cgi?job=163559&pass=irQEgUIl
In case of problems, e-mail :
     neos-comments@mcs.anl.gov
```

Information about the remote optimization process

```
Intermediate Solver Output:
Checking the AMPL files
Executing algorithm...
SNOPT 6.1-1(5)(Nov 2001): version
Finished call
```

Modeling languages:

Resolution of the Log-Likelihood problem

- **Results : (cont.)**

Information sent by the remote optimization server

```
SNOPT 6.1-1(5)(Nov 2001):
Optimal solution found.
1882 iterations, objective -444.6446856
Nonlin evals: obj = 201, grad = 200.
```

Solution printed by the local 1bwsa.run file

```
The estimated values of the model are (alpha, beta, sigma) =
(4.8218, 0.1374, 0.4752).
The relative risk amounts to 0.7490, the accelerating factor
is 1.1472.
omega [*] :=
  1  0.0407
  6  0.156988
 10  0.0827764
 24  0.167377
 36  0.0762982
 48  0.126317
 59  0.0737118
 60  0.0258302
 83  0.0796388
 95  0.0172243
119  0.073086
202  0.0800525
```

Only the nonzero variables ω_j are shown