Ottimizzazione Non Lineare

G. Liuzzi[1]

Venerdì 9 Novembre 2018

[1] Istituto di Analisi dei Sistemi ed Informatica IASI - CNR

Informazioni

Giampaolo Liuzzi

- studio: IASI (CNR), Via dei Taurini 19 (00185, Roma), V piano, stanza 514 (**poco utile**)
- tel: 06 49937129 (**poco utile**)
- email: `giampaolo.liuzzi@iasi.cnr.it` (**utile**)
- didattica: (**utilissimo!!**)
  `http://www.iasi.cnr.it/~liuzzi/teachita.htm`
  `https://groups.google.com/d/forum/onl_tor_vergata_2018-2019`

Informazioni

Ottimizzazione Non Lineare

- Orario delle Lezioni:
  - **Mercoledì** 11:30-13:30 (aula C2)
  - **Giovedì** 11:30-13:30 (aula C2)
  - **Venerdì** 11:30-13:30 (aula C2)
- Ricevimento: **Mercoledì**, **Giovedì** o **Venerdì** subito prima della lezione c/o stanza Prof. V. Piccialli
- Materiale didattico:
  - slides delle lezioni
  - dispense
  - `http://www.iasi.cnr.it/~liuzzi/teachita.htm`
  - `http://people.uniroma2.it/veronica.piccialli/ /Ottimizzazione.html`
- Modalità d'esame:
  - esonero oppure
  - prova scritta

A questo punto...

A questo punto... vi presento

"Julia is a high-level, high-performance dynamic programming language for technical computing"

- `http://julialang.org/`
- Disponibile per
  1. Windows (32/64 bit) self-extracting archive (.exe)
  2. Mac OS (ver. $>$ 10.7, 64 bit) package (.dmg)
  3. Ubuntu (32/64 bit) packages (.deb)
  4. Fedora/RHEL/CentOS/SL (32/64 bit) packages (.rpm)
  5. Generic Linux (32/64 bit) binaries
- Utilizzabile in tre modi
  1. riga di comando
  2. Juno IDE
  3. online su `https://www.juliabox.org/`

A questo punto... vi presento Julia!

"Julia is a high-level, high-performance dynamic programming language for technical computing"

Alcune caratteristiche

- gratis e open-source
- script-type molto simile a Matlab® e R
- ideato per il parallelismo e il "distributed computing"
- estendibile tramite uso di moduli
- JuliaOpt (Optimization packages for the Julia language)
  - JuMP (Julia for Mathematical Programming)
  - CPLEX, GUROBI, GLPK, COIN-Clp, ...
  - Ipopt, KNITRO, NLopt, ...

Julia

Se avete intenzione di utilizzare Julia sul vostro PC:

- Scaricare ed installare Julia da `http://julialang.org/downloads/`
- Avviare una sessione interattiva di Julia (p.es. da un terminale/prompt DOS). Dovreste ottenere:

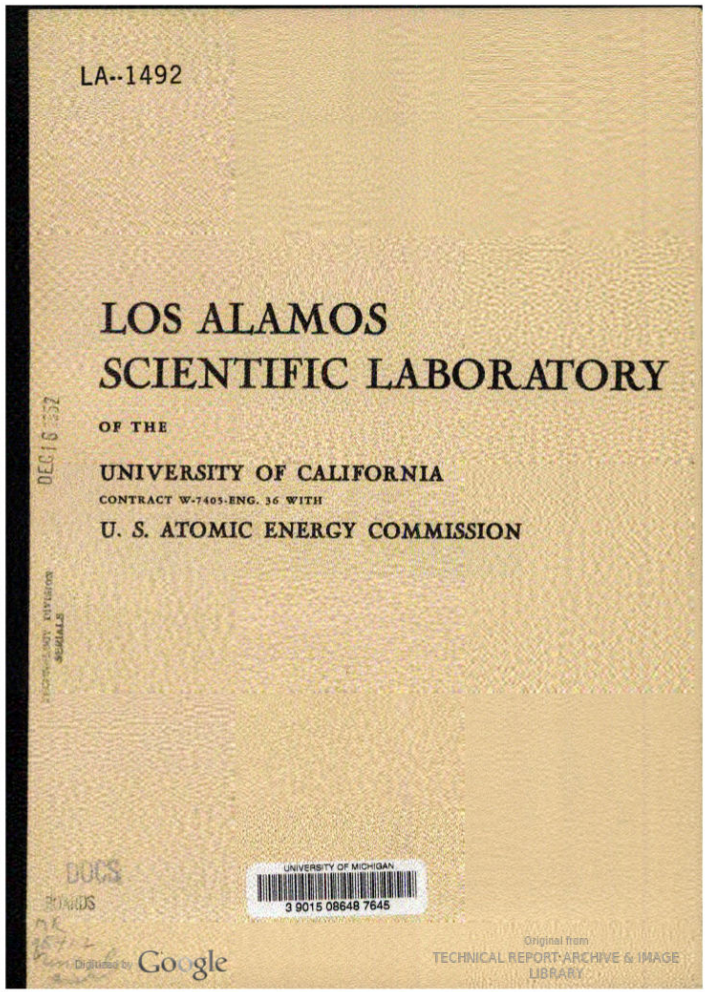
```
               _
   _       _ _(_)_     |  A fresh approach to technical computing
  (_)     | (_) (_)    |  Documentation: https://docs.julialang.org
   _ _   _| |_  __ _   |  Type "?help" for help.
  | | | | | | |/ _` |  |
  | | |_| | | | (_| |  |  Version 0.6.2 (2017-12-13 18:08 UTC)
 _/ |\__'_|_|_|\__'_|  |  Official http://julialang.org/ release
|__/                   |  x86_64-pc-linux-gnu
```

Julia

- al prompt di Julia eseguire i comandi:

```
julia> Pkg.update() \# Get latest package info
julia> Pkg.add("Optim")
julia> Pkg.add("JuMP")
julia> Pkg.add("NLopt")
```

Un problema di minimo (nella fisica delle particelle)



LOS ALAMOS SCIENTIFIC LABORATORY

of the

UNIVERSITY OF CALIFORNIA

Report written:
November 19, 1952

LA-1492

Report distributed:
DEC 3 1952

NUMERICAL SOLUTION OF A MINIMUM PROBLEM

Work done by:
E. Fermi
N. Metropolis

Report written by:
E. Fermi

Un problema di minimo (nella fisica delle particelle)

Abstract

"A particular non-linear function of six independent variables is minimized, using the Los Alamos electronic computer. The values of the variables at the minimum correspond to the phase shift angles in the scattering of pions by hydrogen"

mesone = particella composta da quark + antiquark

pione = mesone π (il più leggero dei mesoni)

Ci sono tre tipi di pione distinti per la loro carica elettrica:

$$\pi^+, \quad \pi^0, \quad \pi^-$$

Dispersione (scattering) $\pi - H$

Due livelli di energia: 113 MeV e 135 MeV.

Per ciascun livello di energia si considerano tre processi di dispersione (scattering) per collisione $\pi - H$

- $\pi^- + p \rightarrow \pi^- + p$
- $\pi^+ + p \rightarrow \pi^+ + p$
- $\pi^- + p \rightarrow \pi^0 + n \rightarrow \gamma\gamma + n$

I risultati degli esperimenti (cross sections, densità) sono acquisiti tramite rivelatori posti ad angoli di: $45°$, $90°$ e $135°$ attorno alla camera a idrogeno (dove avviene la dispersione).

- $\sigma_-^1, \sigma_-^2, \sigma_-^3$
- $\sigma_+^1, \sigma_+^2, \sigma_+^3$
- $\sigma_\gamma^1, \sigma_\gamma^2, \sigma_\gamma^3$

Stima dei parametri

La teoria della dispersione vuole che le **cross sections** $(\sigma_1, \ldots, \sigma_9)$ siano funzione di certi parametri teorici incogniti (**phase shifts**, sfasamenti delle onde s e p di irradiamento dei pioni) $\alpha_1, \ldots, \alpha_6$.

$$\begin{aligned} \sigma_1 &= f_1(\alpha_1, \ldots, \alpha_6) \\ \vdots &\quad \vdots \\ \sigma_9 &= f_9(\alpha_1, \ldots, \alpha_6) \end{aligned}$$

$$\sigma = F(\alpha)$$

Stima dei parametri

Quindi, possiamo scrivere il sistema non-lineare seguente

$$\sigma = F(\alpha),$$

definire la funzione di errore:

$$M(\alpha) = \sum_{i=1}^{9} \left(\frac{\sigma_i - F_i(\alpha)}{\epsilon_i} \right)^2$$

risolvere il problema

$$\min_{\alpha} M(\alpha)$$

Soluzione del problema

Dal rapporto LA-1492, pp. 12-13

"The problem that has been here discussed is an example of a minimum problem for a function of many variables."

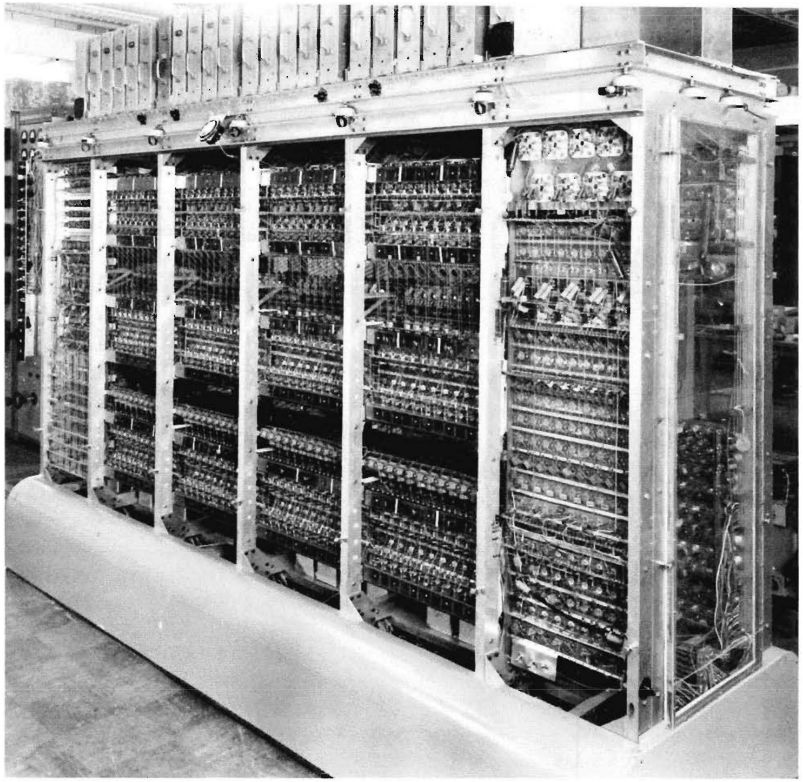
"In principle, problems of this type could be handled in two ways."

"One involves standard mathematical procedure of equating to 0 all the partial derivatives of the function and obtaining thereby a system of n equations with n unknowns."

"The second procedure is the one chosen in the present example: to search for the minimum value by computing the function at very many points until the minimum is attained"

MANIAC I

Fermi e Metropolis usarono il computer MANIAC (**M**athematical **A**nalyzer, **N**umerical **I**ntegrator, **A**nd **C**omputer, 1952-1958) del Los Alamos Laboratory per calcolare la funzione $M(x)$



MANIAC I

"The coding of this part of the problem $[M(x + \Delta x)]$ requires approximately 150 memory positions ... the machine computes its value in approximately **4/10 of a second**, whereas a hand computation of the same function takes about **20 minutes**"

Algoritmo "Fermi-Metropolis"

1. calcola il valore di $M(\alpha)$ per gli angoli iniziali
2. aumenta α_1 con passi di $1/2^\circ$ ($\alpha_1 + 1/2^\circ$, $\alpha_1 + 1^\circ \ldots$) finché il valore di M diminuisce
3. se $\alpha_1 + 1/2^\circ$ produce un aumento di M, diminuisci α_1 con passi di $-1/2^\circ$ finché M diminuisce
4. ripeti i passi 2 e 3 con $\alpha_2, \alpha_3, \ldots, \alpha_6$ al posto di α_1
5. ripeti 2, 3 e 4 finché per due cicli consecutivi il valore di M non si riduce

Al termine, ripeti lo stesso procedimento con passi di $\pm 1/16^\circ$

Programma d'esame

- Ottimizzazione **senza** l'uso delle **derivate**
- Ottimizzazione **globale**
- Ottimizzazione in presenza di **vincoli**

Materiale didattico

- Ottimizzazione **senza** l'uso delle **derivate**
  - Dispense e trasparenze delle lezioni (`www.iasi.cnr.it/∼liuzzi/teachita.htm`)
- Ottimizzazione **globale**
  - Trasparenze (`http://www.dis.uniroma1.it/∼lucidi/didattica/Ott-Glob.html`)
- Ottimizzazione in presenza di **vincoli**
  - Dispense e trasparenze delle lezioni (`www.iasi.cnr.it/∼liuzzi/teachita.htm`)

Cosa ci servirà/cosa dovete ricordare

- Un insieme "compatto" è un insieme chiuso e limitato
- Una funzione continua su un compatto ammette sempre massimo e minimo globale
- $\nabla f(x)^\top d = \lim_{\epsilon \to 0} \frac{f(x + \epsilon d) - f(x)}{\epsilon}$
- $\frac{\partial f(x)}{\partial x_i} = \lim_{\epsilon \to 0} \frac{f(x + \epsilon e_i) - f(x)}{\epsilon}$

Cosa ci servirà/cosa dovete ricordare

$$\begin{array}{lll} \min\ f(x) & \min\ f(x) & \min\ f(x) \\ x \in \mathbb{R}^n & s.t.\ x \in S \subseteq \mathbb{R}^n & s.t.\ g(x) \leq 0 \end{array}$$

dove

- $f : \mathbb{R}^n \to \mathbb{R}$, $f \in C^1$
- S insieme convesso
- $g : \mathbb{R}^n \to \mathbb{R}^m$, $m \geq 1$, $g_i \in C^1$

Condizioni di ottimo:

- caso non vincolato, $\nabla f(x^\star) = 0$
- caso vincoli "facili", $\nabla f(x^\star)^\top (x - x^\star) \geq 0$, per ogni $x \in S$
- caso vincolato, Fritz-JohnKarush-Kuhn-Tucker, $\exists \lambda_0, \lambda_i \exists \lambda_i$:
  - $\lambda_0 \nabla f(x^\star) + \sum_{i=1}^m \lambda_i \nabla g_i(x^\star) = 0_n$
  - $\nabla f(x^\star) + \sum_{i=1}^m \lambda_i \nabla g_i(x^\star) = 0_n$
  - $\lambda_0, \lambda_i \geq 0$ e non tutti nulli
  - $\lambda_i \geq 0$, $x^\star$ "regolare"
  - $\lambda_i g_i(x^\star) = 0$,

Metodo del Gradiente ...

... anche noto come metodo **Steepest Descent** o **Gradient Descent** per la soluzione di

$$\min_{x \in \mathbb{R}^n} f(x)$$

In cosa consiste?

For $k = 0, \ldots,$`maxit`

- Calcola $d_k = -\nabla f(x_k)$
- **If** $\|d_k\| \leq$`tol` **STOP**
- Definisci $x_{k+1} = x_k + \alpha_k d_k$

  α_k ottenuto mediante una ricerca di linea "tipo Armijo"

End For

Se non possiamo usare ∇f?

Soluzione: Invece di $\nabla f(x_k)$ possiamo utilizzare $\nabla_\epsilon f(x_k)$, cioè

$$\nabla_\epsilon f(x_k) = \begin{bmatrix} \frac{f(x_k+\epsilon e_1)-f(x_k)}{\epsilon} \\ \vdots \\ \frac{f(x_k+\epsilon e_n)-f(x_k)}{\epsilon} \end{bmatrix}$$

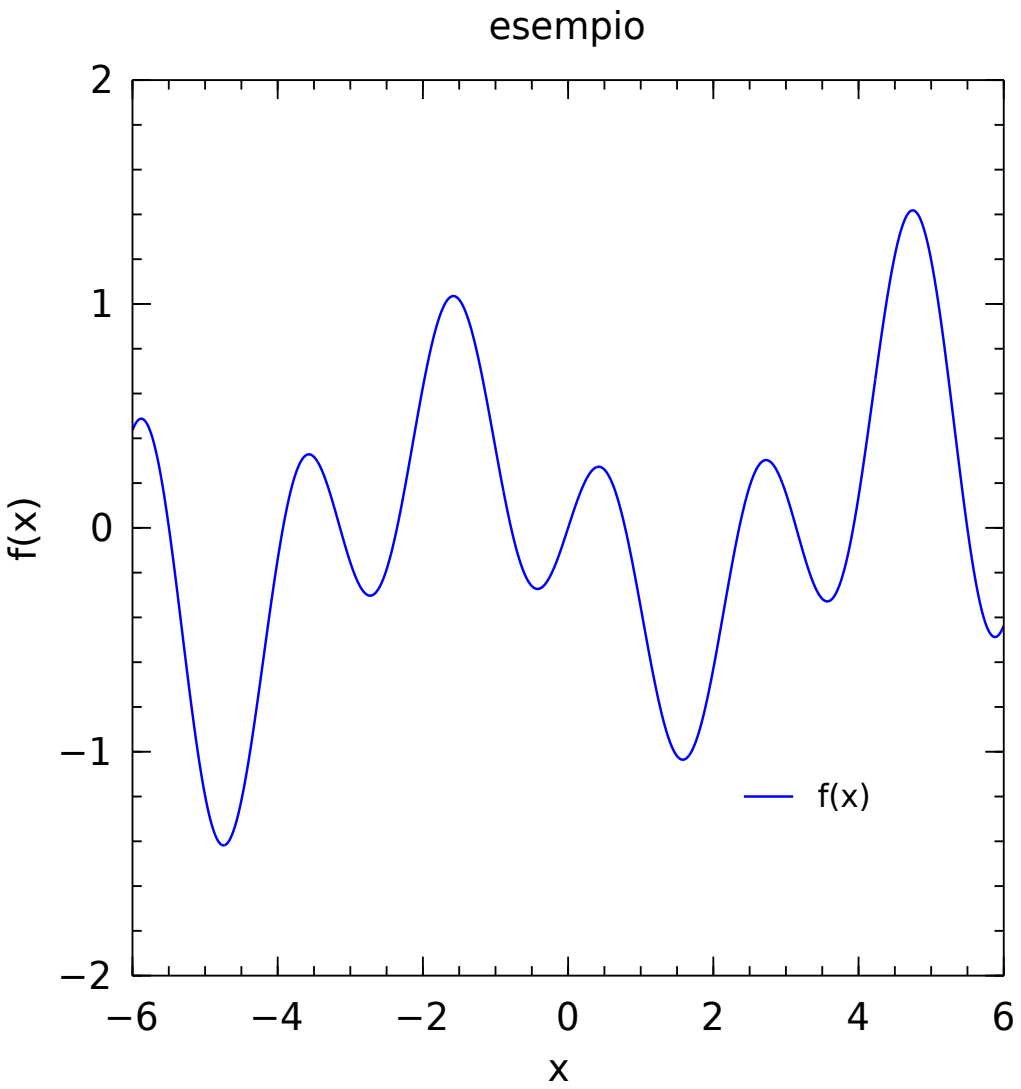
For $k = 0, \ldots,$`maxit`

- Calcola $d_k = -\nabla_\epsilon f(x_k)$
- **If** $\|d_k\| \leq$`tol` **STOP**
- Definisci $x_{k+1} = x_k + \alpha_k d_k$

  α_k ottenuto mediante una ricerca di linea "tipo Armijo"

End For

Quale è il problema?



Perché?

Supponiamo $f(x)$ sia affetta da rumore additivo quindi

$$\tilde{f}(x) = f(x) + \epsilon$$

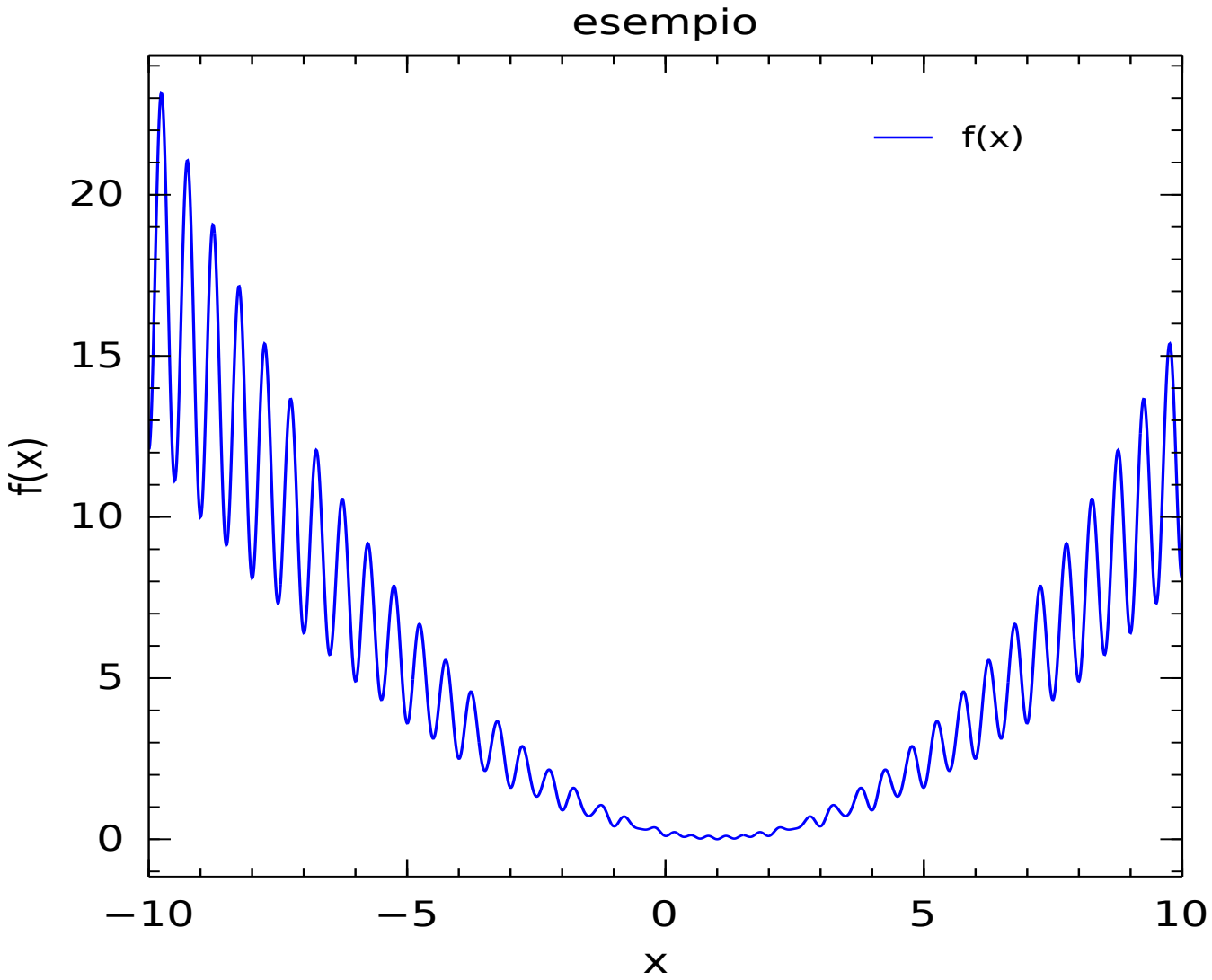
Differenze finite su $\tilde{f}$:

$$\begin{aligned} \frac{\partial \tilde{f}}{\partial x_i}(x) &\cong \frac{\tilde{f}(x + \Delta e_i) - \tilde{f}(x)}{\Delta} = \frac{f(x + \Delta e_i) + \epsilon_1 - f(x) - \epsilon_2}{\Delta} \\ &\cong \frac{\partial f}{\partial x_i}(x) + \frac{\epsilon_1 - \epsilon_2}{\Delta} \end{aligned}$$

Problema: per avere una buona approssimazione devo usare $\Delta \ll 1$ **ma** in questo caso

$$\frac{\epsilon_1 - \epsilon_2}{\Delta} \gg 1!!!$$

Un altro esempio (no rumore, ma molti minimi locali



Voi come fareste?

Obiettivo: minimizzare una funzione $f(x)$ di n variabili reali per la quale non è possibile/conveniente utilizzare derivate prime ne di ordine superiore

Algoritmo di Fermi-Metropolis

Fermi-Metropolis

Consideriamo il problema:

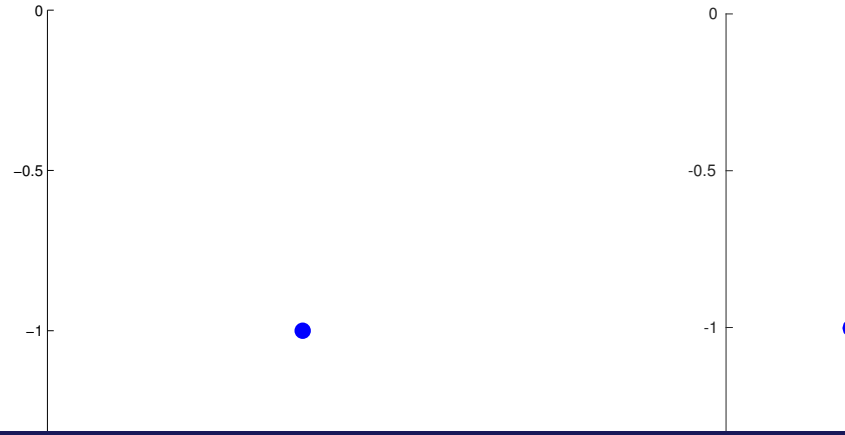
$$\min_{x \in \mathbb{R}^2} f(x)$$

Punto iniziale: $x_0 = (-0.9; -1.0)^\top$
$f(x)$ iniziale: $f(x_0) = 11.3524$
Passo iniziale: $\Delta = 0.3$

Punto corrente: $x_k = (-0.9; -0.7)^\top$
$f(x)$ corrente: $f(x_k) = 5.0788$
Passo corrente: $\Delta = 0.3$

Punto corrente: $x_k = (-0.6; -0.7)^\top$
$f(x)$ corrente: $f(x_k) = 2.2048$
Passo corrente: $\Delta = 0.3$

Punto corrente: $x_k = (-0.6; -0.4)^\top$
$f(x)$ corrente: $f(x_k) = 0.5248$



Ricapitolando

$f(x_k)$	Δ_k
11.352400	0.300000
5.078800	0.300000
0.524800	0.300000
0.524800	0.150000
0.006925	0.150000
0.006925	0.075000
0.006925	0.037500
0.006925	0.018750
0.004715	0.018750
0.004715	0.009375
0.000671	0.009375
0.000671	0.004687
0.000033	0.004687
0.000033	0.002344
0.000033	0.001172
0.000005	0.001172
0.000005	0.000586

Pseudo-code del metodo "Fermi-Metropolis"

INPUT: x_0, Δ_0, Δ_{min}, maxit

$k \leftarrow 0$, $x \leftarrow x_0$, $\Delta \leftarrow \Delta_0$

while $k \leq$maxit **and** $\Delta \geq \Delta_{min}$ **do**

  $k \leftarrow k+1$, $\tilde{x} \leftarrow x$

  for $i = 1, 2, \ldots, n$

    if $f(\tilde{x} + \Delta e_i) < f(\tilde{x})$ **then**

      while $f(\tilde{x} + \Delta e_i) < f(\tilde{x})$ **do** $\tilde{x} \leftarrow \tilde{x} + \Delta e_i$ **end while**

    else if $f(\tilde{x} - \Delta e_i) < f(\tilde{x})$ **then**

      while $f(\tilde{x} - \Delta e_i) < f(\tilde{x})$ **do** $\tilde{x} \leftarrow \tilde{x} - \Delta e_i$ **end while**

    end if

  end for

  if $f(\tilde{x}) = f(x)$ **then** $\Delta \leftarrow \Delta/2$

  else $x \leftarrow \tilde{x}$

  end if

end while

RETURN: x (miglior punto determinato)