Ottimizzazione Non Lineare

G. Liuzzi[1]

Giovedì 9 Novembre 2017

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

Chi sono io ?

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_2017-2018`

Chi siete Voi ?

- Studenti del 1° anno della Laurea Magistrale in **Ingegneria Gestionale** (? Altri?)
- che hanno già seguito la prima parte di questo corso con Il Prof. M. Caramia (?) su "Ottimizzazione nonlineare **non vincolata**"

Notizie utili (che corso è questo?)

Ottimizzazione Non Lineare

- per Ingegneria **Gestionale**
- Orario delle Lezioni:
  - **Lunedì** 9:30-11:15 (aula C2)
  - **Giovedì** 11:30-13:15 (aula C2)
  - **Venerdì** 11:30-13:15 (aula C2)
- Ricevimento: **Giovedì** o **Venerdì** subito dopo lezione
- 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 (?):
  - prova scritta
  - discussione

A questo punto...

A questo punto... vi presento Julia!

"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, ...

Perché Julia ?

Comparison Of Differential Equation Solver Software

Subject/Item	MATLAB	SciPy	deSolve	DifferentialEquations.jl	Sundials	Hairer	ODEPACK and Netlib	JitCODE	PyDSTool	FATODE	GSL	BOOST	Mathematica	Maple
Language	MATLAB	Python	R	Julia	C++ and Fortran	Fortran	Fortran	Python	Python	Fortran	C	C++	Mathematica	Maple
Selection of Methods for ODEs	Fair	Poor	Poor	Excellent	Good (including ARKCODE)	Good	Good	Poor	Poor	Good	Poor	Fair	Fair	Fair
Efficiency*	Poor	Poor	Poor	Excellent	Excellent (including ARKCODE)	Good	Good	Good	Good	Good	Fair	Fair	Fair	Good
Tweakability	Fair	Poor	Poor	Excellent	Good	Good	Good	Fair	Fair	Fair	Fair	Fair	Good	Fair
Event Handling	Good	None	Fair	Excellent	Good**	None	Good**	None	Fair	None	None	None	Good	Good
Symbolic Calculation of Jacobians and Autodifferentiation	None	None	None	Good	None	None	None	None	None	None	None	None	Excellent	Excellent
Complex Numbers	Excellent	Fair	None	Good	Good	None	None	None	None	None	None	Good	Excellent	Excellent
Arbitrary Precision Numbers	None	None	None	Excellent	None	None	None	None	None	None	None	Excellent	Excellent	Excellent
Control Over Linear/Nonlinear Solvers	None	None	None	Excellent	Excellent	Good	Depends on the solver	None	None	None	None	None	Fair	None
Built-in Parallelism	None	None	None	Excellent	Excellent	None	None	None	None	None	None	Fair	None	None
Differential-Algebraic Equation (DAE) Solvers	Good	None	Good	Excellent	Good	Excellent	Good	None	Fair	Good	None	None	Good	Good
Implicitly-Defined DAE Solvers	Good	None	Excellent	Fair	Excellent	None	Excellent	None	None	None	None	None	Good	None
Constant-Lag Delay Differential Equation (DDE) Solvers	Fair	None	Poor	Excellent	None	Good	Fair (via DDVERK)	Fair	None	None	None	None	Good	Excellent
State-Dependent DDE Solvers	Poor	None	Poor	Excellent	None	Excellent	Good	None	None	None	None	None	None	Excellent
Stochastic Differential Equation (SDE) Solvers	Poor	None	None	Excellent	None	None	None	Good	None	None	None	None	Fair	Poor
Specialized Methods for 2nd Order ODEs and Hamiltonians (and Symplectic Integrators)	None	None	None	Excellent	None	Good	None	None	None	None	None	Fair	Good	None
Boundary Value Problem (BVP) Solvers	Good	Fair	None	Good	None	None	Good	None	None	None	None	None	Good	Fair
GPU Compatibility	None	None	None	Excellent	None	None	None	None	None	None	None	Excellent	None	None
Analysis Addons (Sensitivity Analysis, Parameter Estimation, etc.)	None	None	None	Excellent	Good	None	Good (for some methods like DASPK)	None	Poor	Good	None	None	Excellent	None

*Efficiency takes into account not only the efficiency of the implementation, but the features of the implemented methods (advanced timestepping controls, existence of methods which are known to be more efficient, Jacobian handling)

**Event handling needs to be implemented yourself using basic rootfinding functionality

Scale	None	Poor	Fair	Good	Excellent
Explanation	Functionality does not exist	Functionality exists, but is feature-incomplete	The basic features exist	The basic features exist and some extra tweakability exists. May include extra methods for efficiency.	Has all of the basic features and more. Extra features for flexibility and efficiency.

For more detailed explainations and comparisons, see the following blog post:

http://www.stochasticlifestyle.com/a-comparison-between-differential-equation-solver-suites-in-matlab-r-julia-python-c-and-fortran

Cosa dovete fare Voi?

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: http://docs.julialang.org
   _ _   _| |_  __ _   |  Type "?help" for help.
  | | | | | | |/ _` |  |
  | | |_| | | | (_| |  |  Version 0.4.2 (2015-12-06 21:47 UTC)
 _/ |\__'_|_|_|\__'_|  |  Official http://julialang.org release
|__/                   |  x86_64-linux-gnu

julia>
```

Cosa dovete fare Voi?

- 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")
```

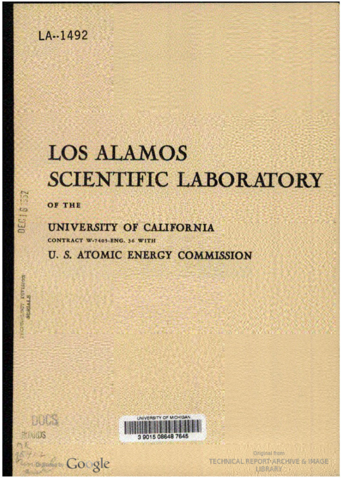
- Compilare (per favore) il questionario disponibile all'indirizzo:

`https://goo.gl/forms/7VpzMic0joxAgqVI3`

(c'è il link sulla pagina web del corso)

N.B. in base alle risposte che fornirete potrò decidere se dedicare una parte della prima esercitazione all'installazione di `Julia`

Un problema di minimo (nella fisica delle particelle)


LA--1492

LOS ALAMOS
SCIENTIFIC LABORATORY
OF THE
UNIVERSITY OF CALIFORNIA
CONTRACT W-7405-ENG. 36 WITH
U. S. ATOMIC ENERGY COMMISSION

DEC 16 1952
DOCS
UNIVERSITY OF MICHIGAN
3 9015 08648 7645
Digitized by Google
Original from
TECHNICAL REPORT ARCHIVE & IMAGE LIBRARY

LOS ALAMOS SCIENTIFIC LABORATORY
of the
UNIVERSITY OF CALIFORNIA

Report written:
November 19, 1952

Report distributed: DEC 3 1952

LA-1492

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

Tre processi di dispersione (scattering) per collisione $\pi - H$

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

I risultati degli esperimenti (cross sections) 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** $\alpha_1, \ldots, \alpha_6$).

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

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

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."

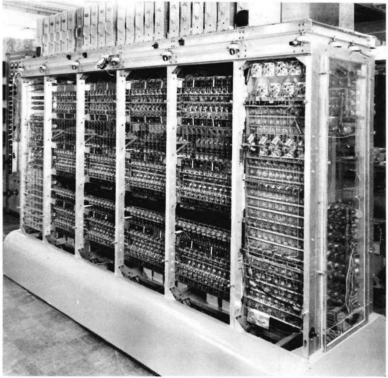
"In principle, problems of this type could be handled in two ways."

"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 $1/2^\circ$ $(\alpha_1 + 1/2^\circ,\ \alpha_1 + 1^\circ\ \dots)$ 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 $\alpha_2, \alpha_3, \dots, \alpha_6$ 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 $\pm 1/16^\circ$

Algoritmo "Fermi-Metropolis"

Homework

Scrivete l'algoritmo "Fermi-Metropolis" in (qualche) pseudo-codice oppure usando un linguaggio di programmazione già noto

Programma d'esame

- Ottimizzazione **senza** l'uso delle **derivate**
  - Esercitazioni in Julia
- Ottimizzazione **globale**
  - Esercitazioni in Julia
- Ottimizzazione in presenza di **vincoli**
  - Esercitazioni in Julia (?)

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`)

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 $k = 0, \ldots, \texttt{maxit}$

- Calcola $d_k = -\nabla f(x_k)$
- **If** $\|d_k\| \leq \texttt{tol}$ **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

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

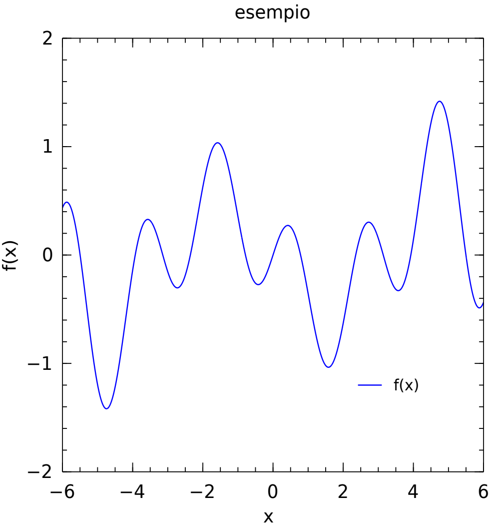
For $k = 0, \ldots,$`maxit`

- 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

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

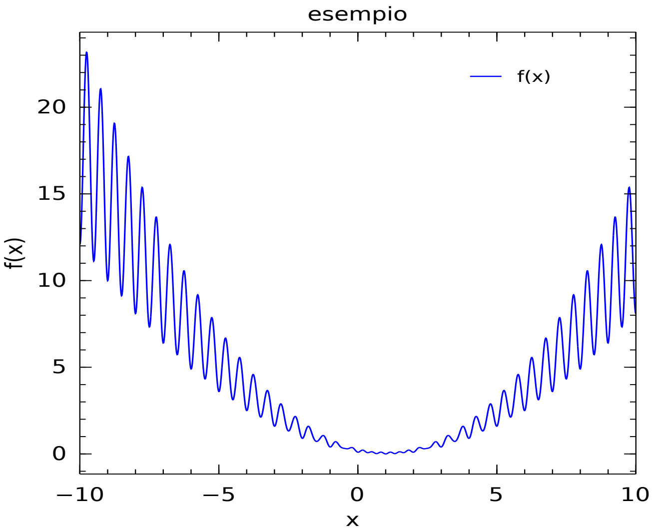
Differenze finite su $\tilde{f}$:

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

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:

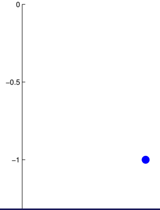
$$\min_{x \in \mathbb{R}^2} f(x)$$

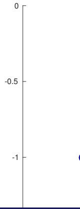
Punto iniziale: $x_0 = (-0.9; -1.0)^\top$
$f(x)$ iniziale: $f(x_0) = 11.3524$
Passo iniziale: $\Delta = 0.3$

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 $i = 1, 2, \dots, n$

    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)