

# COMPLEMENTI DI RICERCA OPERATIVA

Prof. Marco Trubian  
6 CFU

**Luca Cappelletti**

Lecture Notes  
Year 2017/2018



Magistrale Informatica  
Università di Milano  
Italy  
28 dicembre 2017

# Indice

<b>1 Chapter 2</b>	<b>2</b>
1.1 Modelli quadratici . . . . .	2
1.1.1 Casi Possibili . . . . .	2
1.1.2 Esempio . . . . .	2
1.2 Introduzione agli algoritmi . . . . .	2
1.2.1 Quanto è buono un algoritmo di ottimizzazione? . . . . .	3
1.2.2 Come determiniamo un punto di minimo . . . . .	3
1.2.3 Condizioni di Wolfe . . . . .	3
1.2.4 Condizione di curvatura . . . . .	3
<b>2 Ampl</b>	<b>5</b>
2.1 Introduzione alla programmazione lineare . . . . .	5
2.1.1 Tips & Tricks . . . . .	5
2.1.2 Primo esempio . . . . .	5
2.1.3 Secondo esempio con separazione dei dati dal model . . . . .	6
2.1.4 Primo laboratorio . . . . .	7
2.2 Secondo laboratorio . . . . .	9
2.2.1 Primo esercizio . . . . .	9
2.2.2 Secondo esercizio . . . . .	10

## 1.1 Modelli quadratici

Algoritmi di ottimizzazione che approssimano localmente  $f$  con modelli quadratici:

$$\min f(x) = \frac{1}{2} x^T Q x - b^T x, \text{ t.c. } x \in \mathbb{R}^n$$

dove  $Q$  è una matrice quadrata di ordine  $n$ .

### 1.1.1 Casi Possibili

-  $Q$  non è semi-definita positiva:  $f$  non ha un minimo. -  $Q$  è definita positiva:  $x^* = Q^{-1}b$  è l'unico minimo locale. Il punto  $x^*$  è il **punto di ottimo globale**. -  $Q$  è definita semi-positiva: -  $Q$  non è singolare:  $x^* = Q^{-1}b$  è l'unico minimo globale. -  $Q$  è singolare: - non ho soluzioni. - ho infinite soluzioni.

### 1.1.2 Esempio

$$f(x, y) = \frac{1}{2}(ax^2 + by^2) - x$$

Riscrivo nei termini della formula per l'algoritmo:

$$f(x) = \frac{1}{2} \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

## 1.2 Introduzione agli algoritmi

Metodi di ottimizzazione continua:

- Dato un punto di inizio  $x_0$ , generiam una sequenza  $x_{k=0}^{\infty}$ . - Terminato l'algoritmo, quando le condizioni necessarie sono soddisfatte con una certa precisione, per esempio  $\|\nabla f(x_k)\| \leq \epsilon$  - Monotone algorithms requires that  $f(x_k) < f(x_{k-1}) \forall k$

### 1.2.1 Quanto è buono un algoritmo di ottimizzazione?

Un algoritmo è **decente** se **converge**.

**Definizione 1.2.1** (Convergente globalmente). Un algoritmo è chiamato convergente globalmente se converge a un punto  $x^*$

// PERSE COSE DA SLIDE QUI

Un algoritmo è **buono** se **converge rapidamente**

Chiamando  $x_k$  una sequenza in  $\mathbb{R}^n$  che converge a  $x^*$ . La **convergenza** è chiamata:

- **Q-lineare** se  $\exists r \in (0, 1)$  s.t.  $\frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|} \leq r$ , for  $k \geq \bar{k}$

- **Q-superlineare** se  $\lim_{k \rightarrow \infty} \frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|} = 0$

- **Q-quadratica** se  $\exists C > 0$  s.t.  $\frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|^2} \leq C$ , for  $k \geq \bar{k}$

Q-quadratica  $\Rightarrow$  superlineare  $\Rightarrow$  lineare.

### 1.2.2 Come determiniamo un punto di minimo

#### Line search

Dato il punto corrente determino la direzione e dopo di che determino di quanto muovermi.

#### Trust Region

Costruisco un modello quadratico in base alle informazioni locali, quindi scelgo un parametro  $\Delta k$ , un raggio, e scelgo la direzione risolvendo un problema di ottimizzazione vincolato al parametro  $\Delta k$ .

### 1.2.3 Condizioni di Wolfe

Per essere efficiente, la **linesearch inesatta** richiede alcune condizioni:

#### Decremento sufficiente

Sono accettabili solo i valori di  $\phi(\alpha)$  che siano inferiori a quelli della funzione.

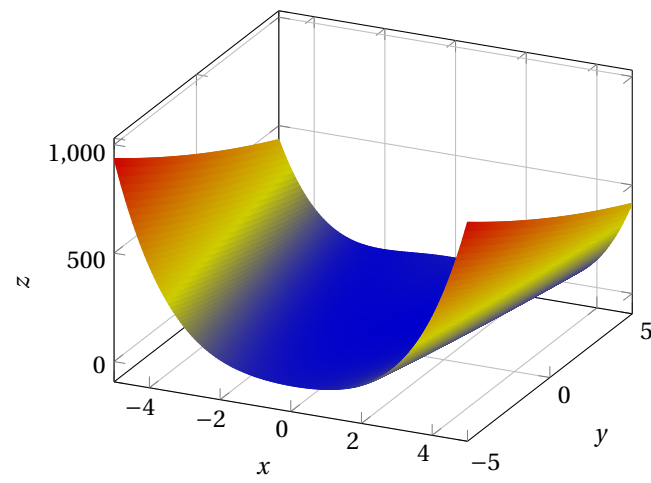
$$f(x + \alpha d) \leq f(x) + c_1 \alpha \nabla f(x)^T d, c \in (0, 1)$$

$$\phi(\alpha) \leq \phi(0) + \alpha c_1 \phi'(0)$$

### 1.2.4 Condizione di curvatura

$$\nabla f(x + \alpha d)^T d \geq c_2 \nabla f(x)^T d$$

STUDIARE BENE TEOREMA DI ZOUTENDIJK CON DIMOSTRAZIONE



# 2

## Ampl

### 2.1 Introduzione alla programmazione lineare

Per risolvere un problema utilizzando ampl è necessario utilizzare 3 tipi diversi di file:

1. Model file (.mod)
2. Data file (.dat)
3. Command file (.run)

Ampl carica questi file e li invia al *solver* (cplex, minos, ...), che quindi legge ed elabora il *Command file*.

*Gli esempi che seguono sono tratti dal canale youtube "Yong Wang":* [https://www.youtube.com/channel/UCXEnJBeaJx3P87A\\_UfZpd0Q](https://www.youtube.com/channel/UCXEnJBeaJx3P87A_UfZpd0Q)

#### 2.1.1 Tips & Tricks

1. Quando si hanno problemi ad identificare il path dei file *.mod*, *.dat* e *.run* è sufficiente fare click destro e selezionare **AMPL commands**.
2. Volendo stampare i valori di una variabile si può usare **display nome-variabile**.

#### 2.1.2 Primo esempio

Esempio di Model file

```
1  # PART 1: DECISION VARIABLES
2  var x1 >= 0; # first variable
3  var x2 >= 0; # second variable
4
5  # PART 2: OBJECTIVE FUNCTION
6  maximize z: 300*x1 + 200*x2;
7
8  # PART 3: CONSTRAINTS
9  s.t. M1:    2*x1 +   x2 <= 8; #s.t. significa "subject to"
10 s.t. M2:     x1 + 2*x2 <= 8;
```

**Esempio di Command file**

```

1  #RESET THE AMPL ENVIROMENT
2  reset;
3
4  #LOAD THE MODEL
5  model example1.mod;
6
7  #CHANGE THE SOLVER (optional)
8  option solver cplex;
9
10 #SOLVE
11 solve;
12
13 #SHOW RESULTS
14 display x1, x2, z;

```

**2.1.3 Secondo esempio con separazione dei dati dal model****Data file**

```

1  param n := 4;
2  param m := 4;
3
4  param    C :=
5      1    50
6      2    20
7      3    30
8      4    80;
9  param    A:  1    2    3    4:=
10     1  400    200   150   500
11     2    3    2    0    0
12     3    2    2    4    4
13     4    2    4    1    5;
14  param    B :=
15     1    500
16     2     7
17     3    10
18     4     8;

```

**Model file**

```

1  param n;
2  param m;
3  set J := {1..n}; #set of decision variables
4  set I := {1..m}; #set of constraints
5
6  param C {J} >= 0; #objective function coefficients
7  param A {I,J} >= 0; #constraint coefficients matrix
8  param B {I} >= 0; #rhs of the constraints
9
10 var X {J} >= 0; #decision variables

```

```

11
12 minimize z:  sum {j in J} C[j] * X[j];
13
14 s.t. Constraint {i in I}:
15     sum {j in J} A[i,j] * X[j] >= B[i];

```

### Command file

```

1  #RESET THE AMPL ENVIROMENT
2  reset;
3
4  #LOAD THE MODEL
5  model example2.mod;
6
7  #LOAD THE DATA
8  data example2.dat;
9
10 #DISPLAY THE PROBLEM FORMULATION
11 expand z, Constraint;
12
13 #CHANGE THE SOLVER (optional)
14 option solver cplex;
15
16 #SOLVE
17 solve;
18
19 #SHOW RESULTS
20 display X, z;

```

### 2.1.4 Primo laboratorio

#### Data file

```

1  data;
2
3  set PROD := bands coils;
4
5  param:    rate  profit  market :=
6    bands   200    25     6000
7    coils   140    30     4000 ;
8
9  param avail := 40;

```

#### Model file

```

1  set PROD;  # products
2
3  param rate {PROD} > 0;      # tons produced per hour

```



```
4 param avail >= 0;           # hours available in week
5
6 param profit {PROD};       # profit per ton
7 param market {PROD} >= 0;  # limit on tons sold in week
8
9 var Make {p in PROD} >= 0, <= market[p]; # tons produced
10
11 maximize Total_Profit: sum {p in PROD} profit[p] * Make[p];
12
13     # Objective: total profits from all products
14
15 subject to Time: sum {p in PROD} (1/rate[p]) * Make[p] <= avail;
16
17     # Constraint: total of hours used by all
18     # products may not exceed hours available
```

## 2.2 Secondo laboratorio

### 2.2.1 Primo esercizio

#### Data file

```

1 data;
2
3 param: ORIG: supply := # defines set "ORIG" and param "supply"
4     GARY    1400
5     CLEV    2600
6     PITT    2900 ;
7
8 param: DEST: demand := # defines "DEST" and "demand"
9     FRA     900
10    DET     1200
11    LAN     600
12    WIN     400
13    STL     1700
14    FRE     1100
15    LAF     1000 ;
16
17 param cost:
18     FRA DET LAN WIN STL FRE LAF :=
19     GARY 39 14 11 14 16 82 8
20     CLEV 27 9 12 9 26 95 17
21     PITT 24 14 17 13 28 99 20 ;

```

#### Model file

```

1 set ORIG; # origins
2 set DEST; # destinations
3
4 param supply {ORIG} >= 0; # amounts available at origins
5 param demand {DEST} >= 0; # amounts required at destinations
6
7     check: sum {i in ORIG} supply[i] = sum {j in DEST} demand[j];
8
9 param cost {ORIG,DEST} >= 0; # shipment costs per unit
10 var Trans {ORIG,DEST} >= 0; # units to be shipped
11
12 minimize Total_Cost:
13     sum {i in ORIG, j in DEST} cost[i,j] * Trans[i,j];
14
15 subject to Supply {i in ORIG}:
16     sum {j in DEST} Trans[i,j] = supply[i];
17
18 subject to Demand {j in DEST}:
19     sum {i in ORIG} Trans[i,j] = demand[j];

```

Una volta inclusi i due file va eseguito il comando *solve* e si ottiene:

```

1 MINOS 5.51: optimal solution found.
2 13 iterations, objective 196200

```

Un possibile Command file che va ad includere ed eseguire i file potrebbe essere:

```

1 model transp.mod;
2 data transp.dat;
3 solve;

```

## 2.2.2 Secondo esercizio

### Data file

```

1 data;
2
3 set ORIG := GARY CLEV PITT ;
4 set DEST := FRA DET LAN WIN STL FRE LAF ;
5 set PROD := bands coils plate ;
6
7 param supply (tr):  GARY  CLEV  PITT :=
8     bands    400    700    800
9     coils    800   1600   1800
10    plate    200    300    300 ;
11
12 param demand (tr):
13     FRA  DET  LAN  WIN  STL  FRE  LAF :=
14     bands 300 300 100  75  650 225 250
15     coils 500 750 400 250 950 850 500
16     plate 100 100  0  50  200 100 250 ;
17
18 param limit default 625 ;
19
20 param cost :=
21
22     [*,*,bands]:  FRA  DET  LAN  WIN  STL  FRE  LAF :=
23     GARY    30   10   8   10   11   71   6
24     CLEV    22    7   10   7   21   82  13
25     PITT    19   11  12   10   25   83  15
26
27     [*,*,coils]:  FRA  DET  LAN  WIN  STL  FRE  LAF :=
28     GARY    39   14  11   14   16   82   8
29     CLEV    27    9  12    9   26   95  17
30     PITT    24   14  17   13   28   99  20
31
32     [*,*,plate]:  FRA  DET  LAN  WIN  STL  FRE  LAF :=
33     GARY    41   15  12   16   17   86   8
34     CLEV    29    9  13    9   28   99  18
35     PITT    26   14  17   13   31  104  20 ;

```

### Model file

```

1 set ORIG;  # origins
2 set DEST;  # destinations

```

```

3  set PROD;      # products
4
5  param supply {ORIG,PROD} >= 0;  # amounts available at origins
6  param demand {DEST,PROD} >= 0;  # amounts required at destinations
7
8      check {p in PROD}:
9          sum {i in ORIG} supply[i,p] = sum {j in DEST} demand[j,p];
10
11 param limit {ORIG,DEST} >= 0;
12
13 param cost {ORIG,DEST,PROD} >= 0;  # shipment costs per unit
14 var Trans {ORIG,DEST,PROD} >= 0;  # units to be shipped
15
16 minimize Total_Cost:
17     sum {i in ORIG, j in DEST, p in PROD}
18         cost[i,j,p] * Trans[i,j,p];
19
20 subject to Supply {i in ORIG, p in PROD}:
21     sum {j in DEST} Trans[i,j,p] = supply[i,p];
22
23 subject to Demand {j in DEST, p in PROD}:
24     sum {i in ORIG} Trans[i,j,p] = demand[j,p];
25
26 subject to Multi {i in ORIG, j in DEST}:
27     sum {p in PROD} Trans[i,j,p] <= limit[i,j];

```