Introduzione ad un ambiente per il calcolo scientifico

Dott. Marco Caliari

a.a. 2008/2009

Queste note sono rivolte agli studenti del Dottorato in Neuroscienze e Scienze psicologiche e psichiatriche dell'Università degli Studi di Verona.

Capitolo 1

Aritmetica floating point

1.1 I numeri macchina

Data la capacità finita di un calcolatore, solo alcuni dei numeri reali possono essere rappresentati. I calcolatori usano la notazione *a virgola mobile (floating point)*: ogni numero $x = \pm x_1 x_2 \dots x_p . x_{p+1} x_{p+2} \dots$ (ove x_i è una cifra compresa tra 0 e 9) è rappresentato internamente come $\pm 0.x_1 x_2 \dots x_t \cdot 10^p$ ove p è una opportuna potenza (eventualmente negativa). Questa notazione sottintende che si sta usando la base 10 per la rappresentazione. In realtà, i calcolatori usano la base 2. La capacità finita implica un limite sul numero, diciamo t, di cifre utilizzabili ("precisione") e sull'esponente p ("ordine di grandezza") che si assume variare tra $L < 0$ e $U > 0$. Inoltre, questa rappresentazione non è unica: per esempio, $0.1 \cdot 10^{-1} = 0.01 \cdot 10^0 = 0.001 \cdot 10^1$. Si assume quindi $x_1 \neq 0$ (rappresentazione *normalizzata*). L'insieme dei *numeri macchina* è allora

$$\{0\} \cup \{x \colon x = \text{sign}(x)(0.x_1 x_2 \dots x_t)_B \cdot B^p,\ 0 \leq x_i < B,\ x_1 \neq 0,\ L \leq p \leq U\}$$

ove B è la *base*, t il numero di *cifre significative*, $x_1 x_2 \dots x_t$ la *mantissa* e p la *caratteristica*. La scrittura $(0.x_1 x_2 \dots x_t)_B$ è una abbreviazione per

$$\sum_{i=1}^{t} x_i B^{-i} = \frac{x_1}{B^1} + \frac{x_2}{B^2} + \ldots + \frac{x_t}{B^t}$$

L'*arrotondamento* di un numero con più di t cifre significative avviene in maniera standard: il numero $x = \text{sign}(x)(0.x_1 x_2 \dots x_t x_{t+1} \dots)_B \cdot B^p$ viene approssimato come

$$\text{sign}(x)(0.x_1 x_2 \dots \tilde{x}_t)_B \cdot B^p,\ \tilde{x}_t = \begin{cases} x_t & \text{se } x_{t+1} < B/2 \\ x_t + 1 & \text{se } x_{t+1} \geq B/2 \end{cases}$$

(per semplicità, assumiamo B pari). Il numero macchina $\varepsilon = \frac{B}{2} \cdot B^{-t}$ è detto *precisione di macchina*. Si ha

$$\begin{aligned} 1+\varepsilon &= (0.\underbrace{10\ldots00}_{t \text{ cifre}})_B \cdot B^1 + \left(0.\underbrace{00\ldots0\frac{B}{2}}_{t \text{ cifre}}\right)_B \cdot B^0 = \\ &= (0.\underbrace{10\ldots000}_{t+1 \text{ cifre}})_B \cdot B^1 + \left(0.\underbrace{00\ldots00\frac{B}{2}}_{t+1 \text{ cifre}}\right)_B \cdot B^1 = \\ &= (0.\underbrace{10\ldots00\frac{B}{2}}_{t+1 \text{ cifre}})_B \cdot B^1 \end{aligned}$$

e, per la regola di arrotondamento, quest'ultimo numero è arrotondato al numero macchina

$$(0.\underbrace{10\ldots01}_{t \text{ cifre}})_B \cdot B^1 > 1$$

È facile rendersi conto, allora, che ε è il più piccolo numero macchina che, sommato a 1, dà un numero macchina maggiore di 1. Infatti, se invece di $\frac{B}{2}$ l'ultima sua cifra fosse stata minore, per la regola di arrotondamento il risultato della somma con 1 sarebbe stato ancora 1. In un normale calcolatore, $B = 2$ e $t = 52$ e dunque $\varepsilon = 2^{-52} \approx 2.2204 \cdot 10^{-16}$. Dato un numero x e il suo arrotondamento a numero macchina $\mathrm{fl}(x)$, si ha

$$|x - \mathrm{fl}(x)| \leq \frac{B}{2} \cdot B^{-(t+1)} \cdot B^p = \frac{B}{2} \cdot B^{-t} \cdot B^{p-1} \leq \frac{B}{2} \cdot B^{-t} |x|$$

da cui

$$\frac{|x - \mathrm{fl}(x)|}{|x|} \leq \varepsilon$$

Dunque, l'*errore relativo* che si commette arrotondando un numero con un numero macchina è minore o uguale alla precisione di macchina. Questo vale ovviamente se la caratteristica di x è p, con $L \leq p \leq U$. Il più grande numero rappresentabile come numero macchina è

$$\begin{aligned} (0.\underbrace{B-1B-1\ldots B-1B-1}_{t \text{ cifre}})_B \cdot B^U &= (1 - (0.\underbrace{00\ldots01}_{t \text{ cifre}})_B) \cdot B^U = \\ &= (1 - B^{-t}) \cdot B^U \end{aligned}$$

In un normale calcolatore $U = 1024$ e dunque $(1 - B^{-t}) \cdot B^U \approx 1.7977 \cdot 10^{308}$. Il più piccolo numero rappresentabile come numero macchina è

$$(0.\underbrace{10\ldots00}_{t \text{ cifre}})_B \cdot B^L = B^{L-1}$$

In un normale calcolatore $L = -1021$ e dunque $B^{L-1} \approx 2.2251 \cdot 10^{-308}$.

Consideriamo ora una qualunque operazione aritmetica tra due numeri, per esempio la somma. La somma di due numeri x e y viene eseguita al calcolatore come

$$\mathrm{fl}\left(\mathrm{fl}\left(x\right) + \mathrm{fl}\left(y\right)\right)$$

I due numeri vengono prima arrotondati in numeri macchina, ne viene eseguita la somma e, infine, il risultato è arrotondato a numero macchina. La stessa cosa succede per ogni altro tipo di operazione. Le due limitazioni dei numeri macchina, sulla precisione e sull'ordine di grandezza, possono portare a risultati sorprendenti, anche per operazioni apparentemente banali.

1.2 Cancellazione

Consideriamo la seguente espressione

$$\frac{(1+x)-1}{x}$$

ove $x = 0.1234 \cdot 10^{-2}$. Il calcolo con numeri macchina con $t = 4$ (e $B = 10$) produce

$$(1+x) = (0.1000000 + 0.0001234) \cdot 10 = 0.1001234 \cdot 10$$

che, arrotondato a t cifre decimali, è $0.1001 \cdot 10$. Poi

$$(0.1001 - 0.1000) \cdot 10 = 0.0001 \cdot 10 = 0.1000 \cdot 10^{-2}$$

E infine

$$(0.1000/0.1234) \cdot 10^{-2} = 0.8104$$

L'errore relativo commesso è $|1 - 0.8104| = 0.1896 \approx 20\%$. Il fatto che t sia comunemente molto più grande, non esclude l'insorgere di questo tipo di errori, detti di *cancellazione* (poiché alcune cifre sono state cancellate durante gli arrotondamenti).

1.3 Overflow e underflow

Si consideri la media aritmetica dei numeri $a = (1 - B^{-t}) \cdot B^{U}$ (il più grande numero macchina) e $b = (1 - B^{-t}) \cdot B^{U-1}$. Il risultato è minore di a e quindi può essere rappresentato, eventualmente in maniera approssimata, come numero macchina. Se però si calcola la media come una somma $a + b$

seguita dalla divisione per 2, si vede come $a + b > a$ e dunque maggiore del più grande numero macchina. L'insieme dei numeri maggiori del più grande numero macchina (o minori del suo opposto) si chiama regione di *overflow*. Il risultato dell'operazione $a+b$ viene comunemente denotato con `Inf`. Qualora il risultato finale di una successione di operazioni sia un numero macchina, solitamente esiste un modo per condurla a termine, anche qualora i risultati parziali risultassero in overflow. Per la media aritmetica, per esempio, basta calcolarla come $(a + b)/2 = b + (a - b)/2$, oppure $a/2 + b/2$.

Analogamente, l'insieme dei numeri diversi da zero e minori del più piccolo numero macchina (o maggiori del suo opposto) si chiama regione di *underflow*. Un risultato in regione di underflow viene comunemente arrotondato a zero.

1.4 Alcuni comandi di Matlab®

Il numero $t = 52$ cifre (binarie) significative che un calcolatore può usare corrisponde a circa 16 cifre decimali significative. Il numero intero 2 viene visualizzato in MATLAB® come

```
>> 2

ans =

     2
```

mentre 10.2 viene visualizzato in MATLAB® come

```
>> 10.2

ans =

   10.2000
```

È una notazione a virgola *fissa*, con quattro cifre decimali. Mentre il numero 0.001 viene visualizzato come

```
>> 0.001

ans =

   1.0000e-03
```

dunque in virgola mobile. La notazione `1.0000e-03` equivale a $1.0000 \cdot 10^{-3}$. In pratica, MATLAB® sceglie automaticamente il formato di *visualizzazione* più opportuno, eventualmente arrotondando il numero, come si vede con 1.23456

```
>> 1.23456

ans =

    1.2346
```

In ogni caso, *tutti* i numeri vengono sempre rappresentati internamente come floating point a (circa) 16 cifre decimali. Per rendersene conto, basta dare il comando `format`

```
>> format long e
```

Si ha, ad esempio,

```
>> 1.23456

ans =

     1.234560000000000e+00
```

È possibile immettere numeri in virgola mobile, come ad esempio `e`

```
>> 1.2e-5

ans =

     1.200000000000000e-05
```

I numeri complessi vengono rappresentati nella forma $a + b$i, `i`

```
>> 3+5i

ans =

      3.000000000000000e+00 + 5.000000000000000e+00i
```

Qualora il risultato di un'operazione sia un numero complesso, questo viene calcolato automaticamente, come ad esempio

```
>> sqrt(-1)

ans =

                    0 + 1.000000000000000e+00i
```

o

```
>> log(-1)

ans =

                    0 + 3.141592653589793e+00
```

Il risultato di operazione "impossibili" (come `0/0`) è `NaN` ("Not a Number").
`pi` Il numero $\pi = 3.14159\ldots$ è approssimato dal comando `pi`.

1.4.1 Risultati "anomali"

Tutte le funzioni elementari sono implementate e la loro accuratezza è garan-
`eps` tita alla precisione di macchina (che si ottiene con il comando `eps`). Significa che, se f è una funzione e `f` la sua implementazione in MATLAB®, allora

$$\frac{|f(x) - \mathtt{f}(\mathrm{fl}\,(x))|}{|f(x)|} \leq \varepsilon$$

Tuttavia, alcune proprietà delle funzioni elementari non sono rispettate. Per esempio,

```
>> sin(pi)

ans =

     1.224646799147353e-16
```

oppure

```
(1/49)*49

ans =

     9.999999999999999e-01
```

Capitolo 2

Operazioni vettoriali

Mentre per Pitagora "tutto è numero", per MATLAB® "tutto è matrice". Per rendersene conto, basta infatti definire

```
>> A = 1

A =

     1
```

e poi `size`

```
>> size(A)

ans =

     1     1
```

per vedere che la variabile `A` è intesa come matrice di dimensione 1×1. Vediamo allora alcuni comandi per la manipolazione di matrici. Innanzi tutto, è utile il comando, per esempio,

```
>> 1:4

ans =

     1     2     3     4
```

che è un'abbreviazione di

```
>> [1,2,3,4]
```

```
ans =

     1     2     3     4
```

È possibile specificare un passo diverso da 1

```
>> 1:2:5

ans =

     1     3     5
```

o, addirittura, non intero

```
>> 1:0.5:3

ans =

    1.0000    1.5000    2.0000    2.5000    3.0000
```

`linspace` In tal caso, può essere conveniente usare il comando `linspace`

```
>> linspace(1,3,5)

ans =

    1.0000    1.5000    2.0000    2.5000    3.0000
```

2.0.2 Costruzione, manipolazione ed estrazione di sottomatrici

```
>> A = [1,2,3,4;5,6,7,8;9,10,11,12;13,14,15,16]

A =

     1     2     3     4
     5     6     7     8
     9    10    11    12
    13    14    15    16

>> A(3,2)
```

```
ans =

    10

>> A(3,[1,2,3,4])

ans =

     9    10    11    12

>> A(3,1:4)

ans =

     9    10    11    12

>> A(3,:)

ans =

     9    10    11    12

>> A(3,1:2)

ans =

     9    10

>> A(3,[2,1,3,4])

ans =

    10     9    11    12
```

end

```
>> A(1,end)

ans =

     4
```

```
>> A(1:2,2:3)

ans =

     2     3
     6     7

>> A(:,2)

ans =

     2
     6
    10
    14
```

diag

```
>> diag(A)

ans =

     1
     6
    11
    16

>> diag(A,1)

ans =

     2
     7
    12

>> diag(A,-1)

ans =

     5
    10
    15
```

reshape

```
>> reshape(A,2,8)

ans =

     1     9     2    10     3    11     4    12
     5    13     6    14     7    15     8    16

>> reshape(A,8,2)

ans =

     1     3
     5     7
     9    11
    13    15
     2     4
     6     8
    10    12
    14    16

>> A(:)

ans =

     1
     5
     9
    13
     2
     6
    10
    14
     3
     7
    11
    15
     4
     8
    12
```

```
    16
```

repmat

```
>> repmat([1,2;3,4],2,2)

ans =

     1     2     1     2
     3     4     3     4
     1     2     1     2
     3     4     3     4
```

diag

```
>> diag([1,2,3,4])

ans =

     1     0     0     0
     0     2     0     0
     0     0     3     0
     0     0     0     4

>> diag([1,2,3],1)

ans =

     0     1     0     0
     0     0     2     0
     0     0     0     3
     0     0     0     0
```

La costruzione di matrici con elementi nulli avviene mediante il comando zeros

zeros

```
>> A=zeros(2,3)

A =

     0     0     0
     0     0     0
```

ones Analogamente per matrici con elementi pari ad uno

```
>> A=ones(3,2)

A =

     1     1
     1     1
     1     1
```

La costruzione di matrici identità avviene mediante il comando `eye` `eye`

```
>> eye(3)

ans =

     1     0     0
     0     1     0
     0     0     1
```

La costruzione di matrici di elementi random (distribuiti uniformemente tra 0 e 1) avviene mediante il comando `rand` `rand`

```
>> rand(2,3)

ans =

    0.9575    0.1576    0.9572
    0.9649    0.9706    0.4854
```

Per distribuzioni gaussiane (di media zero e deviazione standard uno) di elementi random, si usa invece il comando `randn`. La costruzione delle matrici di *Toeplitz* `randn`

$$\begin{bmatrix} c_1 & r_2 & r_3 & \dots & r_n \\ c_2 & c_1 & r_2 & \ddots & r_{n-1} \\ \vdots & c_2 & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & r_2 \\ c_n & c_{n-1} & \dots & c_2 & c_1 \end{bmatrix}$$

avviene mediante il comando `toeplitz` `toeplitz`

```
>> toeplitz([0,1,2,3],[0,-1,-2,-3])

ans =
```

```
     0    -1    -2    -3
     1     0    -1    -2
     2     1     0    -1
     3     2     1     0
```

2.0.3 Trasposizione e altre manipolazioni

.’ Il comando per ottenere la trasposta di una matrice è .’

```
>> A.'

ans =

     1     5     9    13
     2     6    10    14
     3     7    11    15
     4     8    12    16
```

’ Il comando ’ calcola la trasposta *coniugata* di una matrice

```
>> [1+i,2-i;3,4+i]

ans =

   1.0000 + 1.0000i   2.0000 - 1.0000i
   3.0000             4.0000 + 1.0000i

>> [1+i,2-i;3,4+i]'

ans =

   1.0000 - 1.0000i   3.0000
   2.0000 + 1.0000i   4.0000 - 1.0000i
```

I comandi `triu` e `tril` estraggono la parte triangolare superiore *(upper)* e
triu inferiore *(lower)*, rispettivamente

```
>> triu(A)

ans =

     1     2     3     4
     0     6     7     8
```

```
     0     0    11    12
     0     0     0    16
```

`tril`

```
>> tril(A)

ans =

     1     0     0     0
     5     6     0     0
     9    10    11     0
    13    14    15    16
```

2.0.4 Matrici in formato sparso

Si chiamano matrici *sparse* quelle in cui il numero di elementi diversi da zero è proporzionale a n piuttosto che a n^2 (se n è l'ordine della matrice). Per esempio, una matrice *tridiagonale* è una matrice con elementi diversi da zero solo in tre diagonali (di solito, la principale e le due adiacenti). Dunque il numero di elementi diversi da zero è $3n$ e quindi la matrice è sparsa. Una matrice triangolare superiore ha circa $n^2/2$ elementi diversi da zero (per la precisione $(n^2 - n)/2 + n$) e dunque *non* è sparsa.

Per matrici sparse è più conveniente memorizzare solo gli elementi diversi da zero e la loro posizione. Per esempio, invece di memorizzare tutti i 25 elementi della matrice

$$\begin{bmatrix} 10 & 0 & 0 & 0 & 0 \\ 20 & 30 & 0 & 0 & 0 \\ 0 & 0 & 40 & 0 & 0 \\ 0 & 0 & 50 & 60 & 0 \\ 0 & 0 & 0 & 0 & 70 \end{bmatrix}$$

si potrebbero memorizzare i tre vettori

$$\begin{aligned} \text{elementi (per colonna):} &\ [10, 20, 30, 40, 50, 60, 70] \\ \text{indici di colonna:} &\ [1, 1, 2, 3, 3, 4, 5] \\ \text{indici di riga:} &\ [1, 2, 2, 3, 4, 4, 5] \end{aligned}$$

Ovviamente, per questo piccolo esempio, le due tecniche di memorizzazione risultano praticamente equivalenti. MATLAB® utilizza un formato di memorizzazione delle matrici sparse simile a quello elemento/colonna/riga (in realtà, leggermente più efficiente). La conversione dalla memorizzazione in formato *pieno* alla memorizzazione in formato sparso avviene con il comando `sparse`.

`sparse`

```
>> A=[10,0,0,0,0;20,30,0,0,0;0,0,40,0,0;0,0,50,60,0;0,0,0,0,70]

A =

    10     0     0     0     0
    20    30     0     0     0
     0     0    40     0     0
     0     0    50    60     0
     0     0     0     0    70

>> B = sparse(A)

B =

   (1,1)       10
   (2,1)       20
   (2,2)       30
   (3,3)       40
   (4,3)       50
   (4,4)       60
   (5,5)       70
```

È possibile operare su una matrice sparsa come al solito, solo l'output ne risulterà eventualmente diverso:

```
>> B(2,1)

ans =

    20

>> B(2,:)

ans =

   (1,1)       20
   (1,2)       30
```

spdiags

Il più importante comando per la creazione di matrici direttamente in formato sparso è `spdiags`. Permette di creare matrici specificandone gli elementi e la posizione delle diagonali:

```
>> a=[1;2;3;4];b=[0;10;20;30];c=[100;200;0;0];
>> A=spdiags([c,a,b],[-2,0,1],4,4)

ans =

   (1,1)        1
   (3,1)      100
   (1,2)       10
   (2,2)        2
   (4,2)      200
   (2,3)       20
   (3,3)        3
   (3,4)       30
   (4,4)        4
```

`full`

```
>> full(A)

ans =

     1    10     0     0
     0     2    20     0
   100     0     3    30
     0   200     0     4
```

Oltre ad una maggior efficienza nell'occupazione di memoria, il formato sparso può risultare vantaggioso anche nella velocità di esecuzione, in quanto tutte le operazioni che coinvolgerebbero gli zeri vengono semplificate.

2.1 Operazioni vettoriali e puntuali

Siccome tutto è matrice, l'operatore di prodotto `*` è l'operatore di prodotto `*` matriciale. Il prodotto matrice-matrice è definito come

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \dots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \dots & \vdots \\ b_{n1} & b_{n2} & \dots & b_{nn} \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} a_{1i}b_{i1} & \sum_{i=1}^{n} a_{1i}b_{i2} & \dots & \sum_{i=1}^{n} a_{1i}b_{in} \\ \sum_{i=1}^{n} a_{2i}b_{i1} & \sum_{i=1}^{n} a_{2i}b_{i2} & \dots & \sum_{i=1}^{n} a_{2i}b_{in} \\ \vdots & \vdots & \dots & \vdots \\ \sum_{i=1}^{n} a_{ni}b_{i1} & \sum_{i=1}^{n} a_{ni}b_{i2} & \dots & \sum_{i=1}^{n} a_{ni}b_{in} \end{bmatrix}$$

Analogamente, l'operatore di prodotto * tra una matrice e un vettore (colonna) è il classico prodotto matrice-vettore e l'operatore di prodotto * tra un vettore riga e un vettore colonna è il prodotto scalare.

```
>> A=[1,2;3,4]

A =

     1     2
     3     4

>> B=[5,6;7,8]

B =

     5     6
     7     8

>> A*B

ans =

    19    22
    43    50

>> v=[5;6]

v =

     5
     6

>> A*v

ans =

    17
    39

>> w=[1,2]
```

```
w =

     1     2

>> w*v

ans =

    17
```

Ciò significa che alcune operazioni (con dimensioni incompatibili) non sono permesse

```
>> v*A
??? Error using ==> mtimes
Inner matrix dimensions must agree.
```

Analogamente, l'operatore di elevamento a potenza ^ è l'elevamento a potenza matriciale ^

```
>> A*A

ans =

     7    10
    15    22

>> A^2

ans =

     7    10
    15    22
```

e dunque si può applicare solo a matrici quadrate

```
>> v^2
??? Error using ==> mpower
Matrix must be square.
```

Gli operatori di divisione / e \ verranno trattati nel capitolo successivo.

In MATLAB® esistono anche operatori *puntuali*. Date due matrici, è possibile calcolare la matrice i cui elementi sono il prodotto degli elementi corrispondenti delle due matrici .*

```
>> A.*B

ans =

     5    12
    21    32
```

./, .^ Analogamente per la divisione puntuale e l'elevamento a potenza

```
>> A.^2

ans =

     1     4
     9    16
```

Anche in questo caso sono necessarie dimensioni compatibili

```
>> w'.*v

ans =

     5
    12

>> v'./w

ans =

     5     3

>> w.*v
??? Error using ==> times
Matrix dimensions must agree.
```

Tutte le funzioni elementari e tutti gli operatori di confronto di MATLAB® sono puntuali: ad esempio

```
>> exp(A)

ans =

    2.7183    7.3891
```

```
   20.0855   54.5982

>> A>2

ans =

     0     0
     1     1
```

2.2 Il comando `find`

Uno dei comandi più utili in MATLAB® è `find`. Consideriamo un vettore `find`

```
>> v=10:19

v =

    10    11    12    13    14    15    16    17    18    19
```

e supponiamo di voler sapere quali sono gli elementi del vettore maggiori o uguali a 15. È sufficiente il seguente comando

```
>> find(v>=15)

ans =

     6     7     8     9    10
```

A questo punto, è possibile eseguire operazioni solo sugli elementi specificati

```
>> index=find(v>=15)

index =

     6     7     8     9    10

>> v(index)

ans =

    15    16    17    18    19
```

```
>> v(index)=v(index)-15

v =

    10    11    12    13    14     0     1     2     3     4
```

Consideriamo ora il caso matriciale

```
>> A=[10,11;12,13]

A =

    10    11
    12    13

>> index=find(A<13)

index =

     1
     2
     3
```

Il risultato del comando `find` non è una matrice, come ci si potrebbe aspettare, ma un vettore colonna. Non c'è però alcun problema ad eseguire operazioni solo sugli elementi specificati

```
>> A(index)=0

A =

     0     0
     0    13
```

Per costruire, per esempio, una matrice che abbia elementi pari ad uno solo nelle posizioni specificate, è necessario prima inizializzarla con le dimensioni opportune

```
>> B=zeros(2)

B =

     0     0
     0     0
```

e poi assegnare i valori

```
>> B(index)=1

B =

     1     1
     1     0
```

Capitolo 3

Algebra lineare

3.1 Sistemi lineari

Il problema più comune dell'algebra lineare è la soluzione di un sistema lineare

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \\ \qquad\qquad \vdots \qquad\qquad\qquad \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = b_n \end{cases}$$

che può essere scritto, in forma compatta

$$Ax = b$$

ove

$$A = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{12} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & \ldots & \vdots \\ a_{n1} & a_{n2} & \ldots & a_{nn} \end{bmatrix} \quad x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

3.2 Il metodo di eliminazione di Gauss

Consideriamo il seguente sistema lineare

$$\begin{bmatrix} 1 & 2 & 1 \\ 1 & 1 & 1 \\ 2 & 5 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

e applichiamo la seguente *operazione elementare*: alla seconda riga sottraiamo la prima. Si ottiene

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & -1 & 0 \\ 2 & 5 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 3 \end{bmatrix}$$

Adesso, alla terza riga sottraiamo la prima moltiplicata per due. Si ottiene

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & -1 & 0 \\ 0 & 1 & -2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

Infine, alla terza riga aggiungiamo la seconda. Si ottiene

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & -1 & 0 \\ 0 & 0 & -2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix}$$

Allora, con la tecnica delle *sostituzioni all'indietro*, si ricava $x_3 = -1$ dall'ultima riga, poi $x_2 = -1$ dalla penultima e $x_1 = 1-2x_2-x_3 = 1+2+1 = 4$ dalla prima.

Dunque, il *metodo di eliminazione di Gauss* permette di riscrivere il sistema lineare in maniera equivalente (attraverso operazioni elementari) in forma *triangolare superiore*, facilmente risolvibile. Ci sono casi in cui questo processo sembra non essere possibile e casi in cui effettivamente non lo è. Per esempio, se consideriamo il sistema

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 0 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 4 \\ 2 \end{bmatrix}$$

e sottraiamo alla seconda riga la prima moltiplicata per due, otteniamo

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & -2 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix}$$

Ora, alla terza riga sottraiamo la prima, per ottenere

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & -2 \\ 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}$$

A questo punto, non è possibile ottenere un sistema triangolare superiore, a meno di *scambiare* la seconda e la terza riga (in inglese, questa procedura si chiama *pivoting*).

Rimangono ancora due casi: sistemi lineari che ammettono infinite soluzioni e sistemi lineari che non ammettono soluzione. Per esempio, per il sistema lineare

$$\begin{bmatrix} 1 & 2 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \tag{3.1}$$

è facile rendersi conto che $x = [\lambda, 0, 1-\lambda]^{\mathrm{T}}$ è soluzione, qualunque sia $\lambda \in \mathbb{R}$. Tra tutte le infinite soluzioni, una di interesse particolare è quella di norma euclidea ($\sqrt{x_1^2 + x_2^2 + x_3^2}$) minima *(least squares)*, corrispondente a $\lambda = 1/2$, $x = [1/2, 0, 1/2]^{\mathrm{T}}$ (la parabola $\lambda^2 + (1-\lambda)^2$ assume il valore minimo in corrispondenza del suo vertice). Di fatto, il sistema (3.1) è equivalente al *sistema sottodeterminato*

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{3.2}$$

(la prima riga di (3.1) è equivalente alla somma delle altre due). Un'altra soluzione di interesse è $x = [1, 0, 0]^{\mathrm{T}}$ che massimizza il numero di elementi pari a zero.

Consideriamo invece il sistema

$$\begin{bmatrix} 1 & 2 & 1 \\ 1 & 2 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix} \tag{3.3}$$

È facile rendersi conto che non ammette soluzioni (il sistema è *singolare*), in quanto le prime due righe richiedono che una stessa combinazione lineare di x_1, x_2 e x_3 assuma simultaneamente due diversi valori, 1 e 2. Una "soluzione" (chiamata soluzione *ai minimi quadrati*) è quella che minimizza la norma euclidea del *residuo*

$$r(x_1, x_2, x_3) = \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix} - \begin{bmatrix} 1 & 2 & 1 \\ 1 & 2 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Analogamente, si può cercare una soluzione ai minimi quadrati per sistemi *sovradeterminati*, quali

$$\begin{bmatrix} 1 & 2 & 1 \\ 1 & 3 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 0 \\ 1 \end{bmatrix} \tag{3.4}$$

Infine, può succedere che la soluzione ai minimi quadrati non sia unica: consideriamo il sistema lineare (singolare)

$$\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \tag{3.5}$$

La soluzione ai minimi quadrati dovrebbe minimizzare

$$r(x_1, x_2) = (1 - x_1)^2 + 1$$

e dunque $x_1 = 1$. Tra le infinite scelte di x_2 quella che minimizza la norma euclidea della soluzione (e massimizza il numero di elementi pari a zero) è $x_2 = 0$.

3.2.1 Sui minimi quadrati

Dato un insieme di coppie $\{(x_i, y_i)\}_{i=1}^n$, consideriamo un polinomio di grado uno $p_1(x) = a_1 x + a_2$ tale che $p_1(x_i) = y_i$. Naturalmente, sarà possibile trovare tale polinomio solo se tutte le coppie (x_i, y_i) appartengono ad una medesima retta. In tal caso, si dovrebbe risolvere il sistema lineare

$$\begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

Qualora il sistema risulti sovradeterminato, è possibile cercare la soluzione ai minimi quadrati, cioè minimizzare

$$\|r(a_1, a_2)\|_2^2 = (y_1 - (a_1 x_1 + a_2))^2 + (y_2 - (a_1 x_2 + a_2))^2 + \ldots + (y_n - (a_1 x_n + a_2))^2$$

Pertanto, il polinomio trovato è la *retta dei minimi quadrati* per l'insieme di coppie $\{(x_i, y_i)\}_{i=1}^n$.

In generale, dato il sistema sovradeterminato

$$Ax = b, \quad A \in \mathbb{R}^{n \times m},\ b \in \mathbb{R}^{n \times 1},\ n > m$$

il residuo $r(x) = b - Ax$ ha norma euclidea minima quando x è soluzione delle *equazioni normali*

$$A^{\mathrm{T}} A x = A^{\mathrm{T}} b$$

3.3 Sistemi lineari in Matlab®

Data una matrice A *non singolare* di ordine n e un termine noto b il comando

```
>> x=A\b
```

restituisce la soluzione x del sistema lineare $Ax = b$, calcolata con il metodo di eliminazione di Gauss con pivoting. Nel caso di termine noto a più colonne $B = [b_1, b_2, \ldots, b_m]$, la scrittura $AX = B$ significa

$$Ax_1 = b_1,\ Ax_2 = b_2,\ \ldots,\ Ax_m = b_m$$

ove $X = [x_1, x_2, \ldots, x_m]$. Si avrà ancora

```
>> X=A\B
```

Il comando `\` (equivalente a `mldivide`) può essere usato anche per calcolare prodotti matrice-matrice particolari, quali $A^{-1}B$. Infatti, si ha

$$X = A^{-1}B \iff AX = B$$

e dunque

```
>> X=A\B
```

Nel caso di prodotti quali AB^{-1}, si ha

$$X = AB^{-1} \iff XB = A$$

e X può essere calcolato mediante il comando `/` (equivalente a `mrdivide`) `/`

```
>> A/B
```

Nel caso di sistemi sovradeterminati (quali (3.4)), il comando `\` calcola automaticamente la soluzione ai minimi quadrati

```
>> A=[1,2,1;1,3,1;0,1,1;1,1,1]

A =

     1     2     1
     1     3     1
     0     1     1
     1     1     1
```

```
>> b=[1;2;0;1]

b =

     1
     2
     0
     1

>> A\b

ans =

    0.8333
    0.5000
   -0.5000

>> (A'*A)\(A'*b)

ans =

    0.8333
    0.5000
   -0.5000
```

Nel caso invece di sistemi sottodeterminati (quali (3.1)), il comando \ calcola automaticamente la soluzione con più elementi pari a zero

```
>> A=[1,1,1;0,1,0]

A =

     1     1     1
     0     1     0

>> b=[1;0]

b =

     1
     0
```

```
>> A\b

ans =

     1
     0
     0
```

3.3.1 SVD

La decomposizione SVD *(Singular Value Decomposition)* fattorizza una matrice $A \in \mathbb{R}^{n\times m}$ (anche rettangolare) nel prodotto

$$A = USV^{\mathrm{T}}, \quad U \in \mathbb{R}^{n\times n}, \quad S \in \mathbb{R}^{n\times m}, \quad V \in \mathbb{R}^{m\times m}$$

Gli elementi nella "diagonale" di S sono chiamati *valori singolari* di A. Tutti gli altri elementi di S sono nulli. Le matrici U e V sono *ortogonali*, cioè $U^{\mathrm{T}}U = I_n$ e $V^{\mathrm{T}}V = I_m$, ove I_n e I_m indicano le matrici identità di ordine n e m, rispettivamente. Il comando `svd` calcola la decomposizione SVD (per esempio di (3.4))

`svd`

```
>> A=[1,2,1;1,3,1;0,1,1;1,1,1]

A =

     1     2     1
     1     3     1
     0     1     1
     1     1     1

>> [U,S,V]=svd(A)

U =

   -0.5346    0.1091   -0.1889   -0.8165
   -0.7186   -0.5603   -0.0547    0.4082
   -0.2738    0.2608    0.9258    0.0000
   -0.3505    0.7786   -0.3230    0.4082


S =
```

```
    4.5732         0         0
         0    0.7952         0
         0         0    0.6736
         0         0         0


V =

   -0.3507    0.4117   -0.8411
   -0.8417   -0.5323    0.0903
   -0.4105    0.7397    0.5332
```

A partire dai fattori U, S e V è possibile calcolare, ancora, la soluzione ai minimi quadrati di un sistema sovradeterminato

```
>> b=[1;2;0;1]

b =

     1
     2
     0
     1

>> d=U'*b

d =

   -2.3223
   -0.2329
   -0.6213
    0.4082

>> s=diag(S)

s =

    4.5732
    0.7952
    0.6736
```

length

```
>> y=d(1:length(s))./s
```

```
y =

   -0.5078
   -0.2929
   -0.9224

>> x=V*y

x =

    0.8333
    0.5000
   -0.5000
```

In maniera analoga, è possibile calcolare la soluzione ai minimi quadrati di un sistema *quadrato* singolare, come (3.3)

```
>> A=[1,2,1;1,2,1;0,1,0]

A =

     1     2     1
     1     2     1
     0     1     0

>> b=[1;2;0]

b =

     1
     2
     0

>> [U,S,V]=svd(A)

U =

   -0.6873   -0.1664   -0.7071
   -0.6873   -0.1664    0.7071
   -0.2353    0.9719         0
```

```
S =

    3.5616         0         0
         0    0.5616         0
         0         0         0


V =

   -0.3859   -0.5925   -0.7071
   -0.8379    0.5458         0
   -0.3859   -0.5925    0.7071
```

Il terzo valore singolare vale 0 e dunque non è possibile calcolare $y_3 = d_3/s_3$. Basta però porre $y_3 = 0$. Si può fare automaticamente con il comando `find`:

```
>> d=U'*b

d =

   -2.0618
   -0.4991
    0.7071

>> s=diag(S)

s =

    3.5616
    0.5616
         0

>> y=zeros(size(d))

y =

     0
     0
     0
```

```
>> index=find(s~=0)

index =

     1
     2

>> y(index)=d(index)./s(index)

y =

   -0.5789
   -0.8888
         0

>> x=V*y

x =

    0.7500
   -0.0000
    0.7500
```

Si procede in maniera del tutto analoga per sistemi sottodeterminati (sia quadrati, come (3.1) che rettangolari, come (3.2)), nel caso in cui si desideri la soluzione di norma euclidea minima, piuttosto che quella con il maggior numero di zeri. Analogamente quando si desideri la soluzione ai minimi quadrati di norma euclidea minima di sistemi singolari quali (3.5).

3.4 Condizionamento di un sistema lineare

Quando si risolve un sistema lineare, le cose non vanno sempre bene. (Tristemente) famosi sono i sistemi lineari con matrici di Hilbert A definite da

$$A = (a_{ij}), \quad a_{ij} = \frac{1}{i+j-1}$$

Consideriamo infatti una matrice di Hilbert A di ordine 20 (si ottiene tramite il comando `hilb`) e un termine noto b in modo che il sistema lineare $Ax = b$ abbia come soluzione $x = [1, 1, \ldots, 1]^{\mathrm{T}}$ (dovrà essere $b = A * [1, 1, \ldots, 1]^{\mathrm{T}}$) e risolviamo il sistema

`hilb`

```
>> A=hilb(20);
>> b=A*ones(20,1);
>> x=A\b
Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 1.155429e-19.

x =

    1.0000
    1.0001
    0.9955
    1.0635
    0.5635
    2.4236
    0.2518
   -9.1734
   36.9718
  -53.5696
   43.8449
  -37.5198
   72.3107
  -51.0217
  -57.6111
  100.4214
    5.1570
  -86.7209
   62.3377
  -12.7251
```

Otteniamo un risultato molto lontano da quello aspettato $[1, 1, \ldots, 1]^{\mathrm{T}}$, per altro annunciato dal messaggio di *warning* di MATLAB®. Esiste un numero che predice l'accuratezza della soluzione di un sistema lineare, il *numero di condizionamento*, definito da $\|A\|_2 \|A^{-1}\|_2$ e che si può calcolare con il comando `cond`

cond

```
>> cond(A)

ans =

   1.8458e+18
```

Più è grande il numero di condizionamento, meno è accurata la soluzione del sistema lineare. Non esiste una soglia per dire se un numero di condizionamento è grande o piccolo: ma 10^{18} è certamente grande e l'errore tra la soluzione esatta e la soluzione che si ottiene è all'incirca proporzionale al numero di condizionamento. Per misurare la distanza tra due vettori, si calcola la norma (euclidea, per esempio) della differenza, tramite il comando `norm` `norm`

```
>> norm(x-ones(20,1))

ans =

  201.1761
```

Capitolo 4

Polinomi

Useremo la notazione polinomiale

$$p_{n-1}(x) = a_1 x^{n-1} + a_2 x^{n-2} + \ldots + a_{n-1} x + a_n$$

per un polinomio di grado $n-1$.

4.1 Valutazione di un polinomio

Consideriamo il polinomio

$$p_2(x) = 3x^2 + 4x - 1$$

e l'insieme di punti $\{-10, 0, 10\}$. Per valutare il polinomio sull'insieme di punti, è possibile dare il comando

```
>> x=linspace(-10,10,3)

x =

   -10     0    10

>> 3*x.^2+4*x-1

ans =

   259    -1   339
```

oppure, *preferibilmente*, usare il comando `polyval` `polyval`

```
>> polyval([3,4,-1],x)

ans =

   259    -1   339
```

4.2 Interpolazione polinomiale

Dato un insieme di n coppie $\{(x_i, y_i)\}_{i=1}^n$, esiste un unico polinomio $p_{n-1}(x)$ di grado $n-1$ *interpolante*, cioè tale che $p_{n-1}(x_i) = y_i$, $i = 1, 2, \ldots, n$. Per calcolarne i coefficienti, è sufficiente risolvere il sistema lineare *di Vandermonde*

$$\begin{bmatrix} x_1^{n-1} & x_1^{n-2} & \ldots & x_1 \\ x_2^{n-1} & x_2^{n-2} & \ldots & x_2 \\ \vdots & \vdots & \ldots & \vdots \\ x_n^{n-1} & x_n^{n-2} & \ldots & x_n \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

polyfit

Consideriamo il seguente insieme di dati $\{(x_i, y_i)\}_{i=1}^6$, $x_i = 2(i-1)\pi/5$, $y_i = \sin(x_i)$. Il comando `polyfit` imposta e risolve il sistema di Vandermonde

```
>> x=linspace(0,2*pi,6);
>> y=sin(x)

y =

         0    0.9511    0.5878   -0.5878   -0.9511   -0.0000
>> a=polyfit(x,y,length(x)-1)

a =

   -0.0060    0.0938   -0.4292    0.3431    0.8323    0.0000
```

Se poi valutiamo il polinomio negli stessi $\{x_i\}$ (chiamati *nodi di interpolazione*)

```
>> polyval(a,x)

ans =

    0.0000    0.9511    0.5878   -0.5878   -0.9511    0.0000
```

troviamo gli stessi valori $\{y_i\}$. Se invece valutiamo il polinomio in un punto $\bar{x} \notin \{x_i\}$

```
>> xbar=1;
>> polyval(a,xbar)

ans =

    0.8340

>> sin(xbar)

ans =

    0.8415
```

troviamo un valore "sperabilmente" vicino al valore della funzione che abbiamo interpolato. In realtà le cose possono andare molto male. Famoso è l'esempio di Runge:

```
>> x=linspace(-5,5,11);

x =

    -5    -4    -3    -2    -1     0     1     2     3     4     5

>> y=1./(1+x.^2);
>> a=polyfit(x,y,length(x)-1);
>> xbar=-4.5;
>> polyval(a,xbar)

ans =

    1.5787

>> 1./(1+xbar.^2)

ans =

    0.0471
```

L'errore relativo commesso è di circa $|0.0471 - 1.5787| / 0.0471 = 32.5180$. Bisogna dunque procedere diversamente.

4.3 Interpolazione polinomiale a tratti

Invece di costruire un unico polinomio di grado elevato, si possono costruire tanti polinomi, uno per ogni intervallo, di grado basso, in modo che risultino interpolanti (e dunque si raccordino tra loro) e magari abbiano qualche proprietà in più. Per esempio, è possibile usare le *splines cubiche*. Ristrette ad ogni intervallo, sono dei polinomi di grado tre. Oltre ad essere interpolanti, hanno anche la stessa pendenza e la stessa concavità in ogni nodo. Si costruiscono in MATLAB® con il comando `spline`

spline

```
>> spline(x,y,xbar)

ans =

    0.0484
```

Esistono varie famiglie di splines cubiche. Quelle implementate dal comando `spline` sono le *not-a-knot*.

4.4 Approssimazione ai minimi quadrati

Un altro approccio è quello dell'approssimazione ai minimi quadrati. Invece di cercare un polinomio interpolante, si cerca un polinomio di grado "basso" (solitamente molto più basso del numero di nodi) che passi "vicino" ai valori $\{y_i\}$ (che potrebbero essere affetti anche da errore), nel senso dei minimi quadrati (vedi paragrafo 3.2.1). I coefficienti di tale polinomio si ottengono sempre con il comando `polyfit`, specificando, come terzo argomento, il grado desiderato.

4.4.1 Approssimazione non polinomiale

Vediamo un esempio di approssimazione ai minimi quadrati non polinomiale. Supponiamo di voler approssimare un insieme di dati $\{(x_i, y_i)\}_{i=1}^n$ per mezzo di una funzione

$$f(x) = a_1 \sin(x) + a_2 \cos(x) + a_3$$

Siccome vi è dipendenza *lineare* della funzione $f(x)$ dai coefficienti a_1, a_2 e a_3, è sufficiente impostare il sistema di Vandermonde

$$\begin{bmatrix} \sin(x_1) & \cos(x_1) & 1 \\ \sin(x_2) & \cos(x_2) & 1 \\ \vdots & \vdots & \vdots \\ \sin(x_n) & \cos(x_n) & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

e calcolarne la soluzione ai minimi quadrati con il comando \ per la risoluzione di sistemi lineari.