Calcolo Numerico : Laboratorio

Gianna Del Corso <gianna.delcorso@unipi.it>

20 Ottobre 2021

Quantità di esercizi: in questa dispensa ci sono *più esercizi* di quanti uno studente medio riesce a farne durante una lezione di laboratorio, specialmente tenendo conto anche degli esercizi facoltativi. Questo è perché sono pensate per "tenere impegnati" per tutta la lezione anche quegli studenti che già hanno un solido background.
Quindi fate gli esercizi che riuscite, partendo da quelli *non* segnati come facoltativi, e non preoccupatevi se non li finite tutti! Questa dispensa è il risultato del contributo di varie persone che hanno proposto, elaborato e raffinato i vari esercizi.

1 Ottenere aiuto

Con il comando `help`, potete ottenere una breve documentazione su come si usa un comando.

```
>> help imag
imag Complex imaginary part.
imag(X) is the imaginary part of X.
See I or J to enter complex numbers.

See also real, isreal, conj, angle, abs.

Reference page for imag
Other functions named imag
```

È possibile aggiungere una breve documentazione (ed è buona pratica farlo) anche alle funzioni scritte dall'utente. Per farlo basta aggiungere un commento fra l'istruzione `function x = myfun(y)` ed il corpo della funzione. Provate, ad esempio, a modificare la funzione `pow(x,n)` in questo modo:

```
function y = pow(x,n)
% Usage: y = pow(x,n)
%
% Questa funzione calcola l'n-esima potenza di x.

 ...
```

```
end
```

Verifichiamo cosa succede usando il comando `help pow`:

```
>> help pow
'pow' is a function from the file /path/to/pow.m

 Usage: y = pow(x,n)
 Questa funzione calcola l'n-esima potenza di x.
```

2 Vettori e matrici

Matlab è pensato per lavorare con vettori e matrici; pertanto, ha una sintassi specifica e parecchi comandi dedicati, che rendono molto più semplice lavorare con i vettori rispetto a un linguaggio generico come il C.

2.1 Creare vettori e matrici

```
>> A=[1 2 3; 4 5 6]
A =

   1 2 3
   4 5 6
```

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

   0 0
   0 0
   0 0

>> ones(3,2)
ans =

   1 1
   1 1
   1 1

>> eye(3)
ans =

   1 0 0
   0 1 0
```

```
   0 0 1

>> randn(2,3)
ans =

   0.567178 -0.126397 -0.090664
  -0.678601 0.504481 0.754911
```

2.2 Il *range operator* :

Con la sintassi `a:t:b` creiamo un vettore (riga) che contiene gli elementi `a`, `a+t`, `a+2t`... fino a `b` (o fino all'ultimo che sia minore o uguale a `b`). Se `t`=1, può essere omesso.

```
 >> 1:0.5:4
ans =

   1.0000 1.5000 2.0000 2.5000 3.0000 3.5000 4.0000

>> 1:10
ans =

   1 2 3 4 5 6 7 8 9 10

>> 1:2:10
ans =

   1 3 5 7 9
```

Dove avete già usato l'operatore `:`?

2.3 Il comando `linspace`

Un'utile variante dell'operatore `:` è il comando `linspace` che, come il nome suggerisce, permette di dividere un intervallo in parti uguali. Supponiamo ad esempio di voler dividere $[0, 2\pi]$ in $n-1$ sotto-intervalli di eguale lunghezza. Possiamo dare il comando `t = linspace(0,2*pi,n)` ed ottenere un vettore `t` tale che

- `t(1) = 0`
- `t(n) = 2*pi`
- Per ogni i fra 2 ed n, `t(i) - t(i-1) = 2*pi / (n-1)`.

Spesso `linspace` è il modo più naturale per suddividere un intervallo su cui si desidera plottare una funzione. Utilizzate `help linspace` per approfondire l'utilizzo del comando.

2.4 Accedere agli elementi

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

   1 1 1
   1 1 1

>> A(1,2)=2
A =

   1 2 1
   1 1 1

>> A(1,2)
ans = 2
>> A(5,10)
error: invalid row index = 5
error: invalid column index = 10
>> A(5,10)=7
A =

   1 2 1 0 0 0 0 0 0 0
   1 1 1 0 0 0 0 0 0 0
   0 0 0 0 0 0 0 0 0 0
   0 0 0 0 0 0 0 0 0 0
   0 0 0 0 0 0 0 0 0 7
```

Se cerco di *leggere* un elemento che non esiste (perché la matrice è troppo piccola), ottengo un errore. Se cerco di *scrivere* un elemento che non esiste, la matrice viene automaticamente ingrandita.

2.5 Operazioni su vettori

```
>> a=1:3
a =

  1 2 3

>> b=4:6
b =

  4 5 6

>> a+b
ans =
```

```
   5 7 9

>> sin(a)
ans =

   0.84147 0.90930 0.14112

>> 2*a+1
ans =

   3 5 7

>> a.*b %operazioni elemento per elemento
ans =

    4 10 18

>> c=a' %matrice trasposta
c =

   1
   2
   3

>> C=a'*b %prodotto matrice-matrice
C =

    4 5 6
    8 10 12
   12 15 18

>> length(a) %lunghezza di un vettore
ans = 3

>> size(C) %dimensioni di una matrice - (righe, colonne)
ans =

   3 3
```

2.6 Costruire matrici particolari

Se vogliamo scrivere uno script che produce la matrice

$$A = \begin{bmatrix} 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 \\ 21 & 22 & 23 & 24 & 25 & 26 & 27 & 28 & 29 \\ 31 & 32 & 33 & 34 & 35 & 36 & 37 & 38 & 39 \\ 41 & 42 & 43 & 44 & 45 & 46 & 47 & 48 & 49 \\ 51 & 52 & 53 & 54 & 55 & 56 & 57 & 58 & 59 \\ 61 & 62 & 63 & 64 & 65 & 66 & 67 & 68 & 69 \\ 71 & 72 & 73 & 74 & 75 & 76 & 77 & 78 & 79 \\ 81 & 82 & 83 & 84 & 85 & 86 & 87 & 88 & 89 \\ 91 & 92 & 93 & 94 & 95 & 96 & 97 & 98 & 99 \end{bmatrix}$$

servono due cicli for annidati...

```
A = zeros(9, 9);
for i = 1:9
    !for j = 1:9!
        !A(i,j) = 10*i + j!
    !end!
end
```

Le linee in rosso sono il corpo del loop più esterno. Come vengono trasformate queste righe in istruzioni?

Indentazione (allineamento verticale delle istruzioni tramite spazi) rende tutto molto più leggibile.

Preallocare A come una matrice di zeri rende il codice leggermente più veloce e assicura risultati coerenti (anche se c'era già una variabile A in giro).

Esercizio 1. Scrivere le 'tabelline':

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 2 & 4 & 6 & 8 & 10 & 12 & 14 & 16 & 18 \\ 3 & 6 & 9 & 12 & 15 & 18 & 21 & 24 & 27 \\ 4 & 8 & 12 & 16 & 20 & 24 & 28 & 32 & 36 \\ 5 & 10 & 15 & 20 & 25 & 30 & 35 & 40 & 45 \\ 6 & 12 & 18 & 24 & 30 & 36 & 42 & 48 & 54 \\ 7 & 14 & 21 & 28 & 35 & 42 & 49 & 56 & 63 \\ 8 & 16 & 24 & 32 & 40 & 48 & 56 & 64 & 72 \\ 9 & 18 & 27 & 36 & 45 & 54 & 63 & 72 & 81 \end{bmatrix}$$

Esercizio 2. Creare la matrice

$$B = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 2 & 0 & 0 & 0 \\ 1 & 2 & 3 & 0 & 0 \\ 1 & 2 & 3 & 4 & 0 \\ 1 & 2 & 3 & 4 & 5 \end{bmatrix}.$$

Riuscite a farlo riempendo la matrice una riga per volta a partire dalla prima? Riuscite a farlo riempendo la matrice una colonna per volta?

Esercizio 3. Creare la matrice $n \times n$ (per esempio, per $n = 10$) contenente elementi

$$C = \begin{bmatrix} 0 & 1 & & & & & \\ 1 & 0 & 1 & & & & \\ & 1 & 0 & 1 & & & \\ & & 1 & 0 & 1 & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & 1 & 0 & 1 \\ & & & & & 1 & 0 \end{bmatrix}.$$

Gli elementi non visualizzati sono tutti zeri. (Vi servono davvero due `for` annidati qui?)

Esercizio 4. Scrivere una funzione `A=bidiag(a, b, n)` che costruisce la matrice di bidiagonale $n \times n$ definita nel seguente modo $A_{ij} = a$, per $i = j$, $A_{ij} = b$ per $i = 2, \ldots, n$ e $j = i - 1$. Ad esempio, la chiamata della funzione `A=bidiag(3, -1, 4)` deve restituire la matrice

$$A = \begin{bmatrix} 3 & 0 & 0 & 0 \\ -1 & 3 & 0 & 0 \\ 0 & -1 & 3 & 0 \\ 0 & 0 & -1 & 3 \end{bmatrix}.$$

Esercizio 5. Scrivere una funzione `function M=laplace(n)` che crea la matrice di dimensione $n \times n$ che ha 2 sulla diagonale e -1 sulla sopra- e sotto-diagonale:

```
>> laplace(5)
ans =

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

É possibile inserire direttamente le entrate di una matrice che giacciono su una certa diagonale con il comando `diag`. Piú precisamente digitando `A= diag(v,k)` si ottiene una matrice che ha gli elementi del vettore `v` sulla k-esima diagonale e 0 altrove. Il segno negativo o positivo di `k` corrisponde alle sottodiagonali e alle sopradiagonali, rispettivamente. Se viene omesso il parametro `k`, gli elementi di v vengono inseriti sulla diagonale principale. In questo modo risulta facilitata l'inizializzazione di matrici con struttura a banda: bidiagonali, tridiagonali ecc.

```
>> A=diag(1:10)+diag(ones(9,1),1)
A =
```

```
 1 1 0 0 0 0 0 0 0 0
 0 2 1 0 0 0 0 0 0 0
 0 0 3 1 0 0 0 0 0 0
 0 0 0 4 1 0 0 0 0 0
 0 0 0 0 5 1 0 0 0 0
 0 0 0 0 0 6 1 0 0 0
 0 0 0 0 0 0 7 1 0 0
 0 0 0 0 0 0 0 8 1 0
 0 0 0 0 0 0 0 0 9 1
 0 0 0 0 0 0 0 0 0 10
```

Esercizio 6. Scrivere una funzione `y=prodottoL(b)` che, dato un vettore b, calcola il prodotto $y = Lb$. Ad esempio, per $n = 4$,

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}.$$

Utilizzando il comando di Matlab `L * v`, il prodotto costa $O(n^2)$ operazioni aritmetiche (perché?). È possibile scrivere questa funzione in modo che utilizzi $O(n)$ operazioni aritmetiche?

3 Soluzione di sistemi triangolari

$$\begin{pmatrix} L_{11} & 0 & 0 & 0 & 0 \\ L_{21} & L_{22} & 0 & 0 & 0 \\ L_{31} & L_{32} & L_{33} & 0 & 0 \\ L_{41} & L_{42} & L_{43} & L_{44} & 0 \\ L_{51} & L_{52} & L_{53} & L_{54} & L_{55} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \end{pmatrix}$$

Possiamo risolverlo per sostituzione: a ogni passo, supponendo di avere calcolato $x_1, \ldots, x_{i-1}$ si ha

$$x_i = \frac{b_i - \sum_{j=1}^{i-1} L_{ij} x_j}{L_{ii}}$$

```
function x=inf_solve(L,b)
n=length(b)
x=zeros(n,1);
for i=1:n
       somma=0; % accumulatore
       for j=1:i-1
              somma=somma+L(i,j)*x(j);
       end
       x(i)=(b(i)-somma)/L(i,i);
```

```
        end
end
```

Testiamo la funzione con la matrice `L=tril(laplace(5))` e il termine noto `L*y`, dove y è un vettore semplice.

```
>> y=[1:5]'
y =

1
2
3
4
5

>> L=tril(laplace(5))
L =

2 0 0 0 0
-1 2 0 0 0
0 -1 2 0 0
0 0 -1 2 0
0 0 0 -1 2

>> inf_solve(L,L*y)
ans =

1
2
3
4
5
```

Esercizio 7. Scrivere una `function sup_solve(U,b)` che risolva un sistema $Ux = b$ con U triangolare superiore (hint: sostituire a partire dall'ultima riga). Testare su `U=triu(laplace(5))`, `b=U*y` (con `y` vettore opportuno).

Esercizio 8 (facoltativo). Modificare `inf_solve` e `sup_solve` in modo da sostituire il ciclo `for` più interno con operazioni su vettori.

Esercizio 9 (facoltativo). Scrivere due funzioni `inf_solve2` e `sup_solve2` che risolvano i sistemi $Ly = b$ e $Ux = y$ supponendo L e U bidiagonali (facendo meno calcoli di `inf_solve` e `sup_solve`!).

Quanto tempo impiegano le due funzioni?

```
>> L=tril(laplace(1000));
>> b=tril(L*[1:1000]';
>> tic; inf_solve(A, b); toc
```

```
Elapsed time is 0.560339 seconds.
>> tic; inf_solve2(A, b); toc
Elapsed time is 0.000768 seconds.
>>
```

Quando una matrice ha una struttura dobbiamo sempre cercare di sfruttarla nei calcoli; il guadagno di tempo (e a volte anche di precisione dei risultati) può essere notevole.