Calcolo Numerico - Corso A: Laboratorio Lezione 3

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

3 Marzo 2023

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 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.

2.4 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
```

2.5 Calcolo di norme e condizionamento

Matlab mette a disposizione dei comandi per il calcolo delle norme (fare `help norm`), allo stesso modo si può stimare il condizionamento in norma 2 con il comando `cond(A )`, ed aggiungendo il parametro $p \in \{1, 2, \text{inf}, \text{'fro'}\}$ cioè scrivendo `cond(A, p)` otteniamo il condizionamento nella norma desiderata.

```
>> a=hilb(10);
>> cond(a)

ans =

1.6025e+13

>> cond(a,1)

ans =
```

```
3.5353e+13

>> cond(a,inf)

ans =

3.5353e+13
```

Sebbene i valori restituiti siano tutti diversi, qualitativamente si ottengono risultati confrontabili (questo per l'equivalenza delle norme).

2.6 Costruire matrici particolari

Esercizio 1. 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 2. 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 3. 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 4. Si esegua il seguente frammento di codice e si cerchi di interpretare il risultato

```
n=10^3;
A=bidiag(2, -1, n);
y=rand(n, 1);
b=A*y;
x=inf_solve(A, b);
err=norm(x-y, 'inf')
```

Si esegua poi il seguente frammento di codice

```
n=10^3;
A=bidiag(1, -2, n);
y=rand(n,1);
b=A*y;
x=inf_solve(A, b);
```

```
err=norm(x-y, 'inf')
```

Esercizio 5. 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 6 (facoltativo). Modificare `inf_solve` e `sup_solve` in modo da sostituire il ciclo `for` più interno con operazioni su vettori.

Esercizio 7 (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.

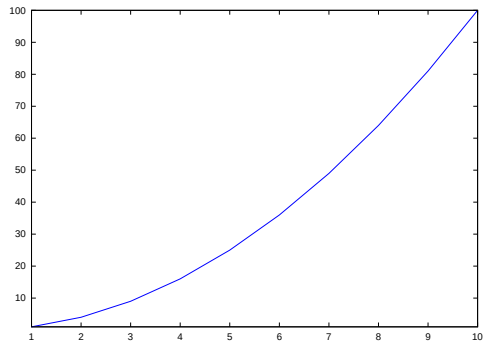
4 Grafici

Il comando `plot(x,y)` prende come argomenti due vettori della stessa lunghezza `x` e `y` e disegna sul piano cartesiano i punti `x(i),y(i)` collegandoli con una linea.

```
>> r=1:10
r =

1 2 3 4 5 6 7 8 9 10

>> plot(r,r.^2)
```



Il seguente comando disegna un cerchio.

```
>> t=0:.001:2*pi;
>>> plot(cos(t),sin(t))
```

Suggerimento: Il comando `t = 0:.001:2*pi` può essere sostituito da un comando `linspace`. Provate ad utilizzarlo nei prossimi esercizi.

Esercizio 8. Scrivere una funzione `circle(z,r)` che, dato un complesso `z` e un reale `r`≥ 0, disegni il cerchio di centro `(real(z),imag(z))` e raggio `r`. Testare con `circle(1+2i,2)`. A quale ascissa/ordinata arriva il cerchio?

5 Cerchi di Gerschgorin

La seguente funzione disegna gli autovalori e i cerchi di Gerschgorin di una matrice quadrata `A`.

```
function gg(A)
n=size(A, 1); %cosa fa questa istruzione?

clf; %elimina i disegni preesistenti
hold on; %fa si' che ogni disegno non cancelli il precedente
axis('equal'); %forza la stessa scala su x e y

autovalori=eig(A);
plot(real(autovalori),imag(autovalori),'r*');
% 'r*': disegna solo i punti, non collegandoli con linee di colore rosso (red)
%"help plot" per altre stringhe magiche
for k=1:n
center=A(k,k);
radius=0; %accumulatore
for j=1:n
if(j~=k)
radius=radius+abs(A(k,j));
end
```

```
end
circle(center,radius);
end
end
```

Esercizio 9. Testare la funzione `gg` su alcune matrici test: `rand(10)`, `randn(10)`, `hilb(10)`,

Esercizio 10. Scrivete una funzione `perturb(A,epsilon)` che, data una matrice A genera 50 matrici i cui elementi sono gli elementi di A modificati con un piccolo valore casuale di magnitudine circa `epsilon` (con le istruzioni `epsilon*randn(n) + 1i*epsilon*randn(n)`), e disegna i loro autovalori su un grafico. Di quanto si spostano gli autovalori per una perturbazione di grandezza `epsilon=1e-2`? Per una matrice casuale? Per l'identità?

Esercizio 11. Scrivere una funzione `ggsecond(A)` che

1. disegni i cerchi di Gerschgorin di `A`
2. usando il compando `t=linspace(0,1)` discretizzi l'intervallo $[0, 1]$ con 100 punti $t(k), k = 1, 2, \ldots, 100$.
3. per ogni punto `t(k)` cacoli la matrice `M=t(k)*A+(1-t(k))*D` con `D`, matrice diagonale che ha per elementi gli elementi diagonali `A`.
4. calcoli gli autovalori di$M$ e li plotti nello stesso grafico dove sono rappresentati i cerchi di Gerschgorin.

Esercizio 12. Nei grafici dell'esercizio precedente, noterete che per molte matrici (fate qualche esempio!) alcuni autovalori si "muovono" secondo questo schema: due autovalori reali si muovono nella stessa direzione e si avvicinano fino a coincidere, poi, una volta uniti, si staccano dall'asse reale e vanno in due direzioni opposte del piano complesso. Verificate questo fenomeno. Sapete spiegare perché questo accade?

Esercizio 13 (facoltativo). Poiché Matlab è un linguaggio *interpretato*, eseguire ogni singola istruzione ha un "costo" non trascurabile (il computer deve leggere la riga, interpretarla e trasformarla in codice macchina). Per questo se si riesce a riscrivere le funzioni utilizzando delle operazioni sui vettori invece che dei cicli `for`, il programma gira molto più velocemente. Per esempio, è molto più veloce

```
        s=sum(abs(v));
```

(una istruzione da interpretare) rispetto a

```
        s=0;
        for k=1:length(v)
        s=s+abs(v(k));
        end
```

($O(n)$ istruzioni da interpretare, dove n è la lunghezza del vettore `v`).

L'operazione di sostituire tante istruzioni su scalari con una singola istruzione su un vettore si chiama *vectorization*. Riuscite a riscrivere i programmi della prima lezione (`fattoriale`, `pow`, `myexp`, `myexp2`) vettorizzando i cicli `for`, in modo che non siano più necessarie $O(n)$ istruzioni per calcolare un esponenziale? Potrebbe esservi utile guardare i comando `help sum` e `help product`.