Laboratorio di Analisi Numerica
Lezione 2

Leonardo Robol <leonardo.robol@unipi.it>
Igor Simunec <igor.simunec@sns.it>

8 ottobre 2021

Quantità di esercizi: in questa dispensa ci sono *più esercizi* di quanti uno studente medio riesca a farne durante una lezione di laboratorio, specialmente tenendo conto anche degli esercizi facoltativi. Questo è perché è pensata per "tenere impegnati" per tutta la lezione anche quegli studenti che già hanno un solido background di programmazione.
Quindi fate gli esercizi che riuscite, partendo da quelli *non* segnati come facoltativi, e non preoccupatevi se non li finite tutti!

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.
```

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)
%POW Calcola la potenza di un numero floating point.
%
% Y = POW(X, N) calcola la potenza N-esima del numero floating point X.

 ...
```

```
end
```

Verifichiamo cosa succede usando il comando `help pow`:

```
>> help pow
POW Calcola la potenza di un numero floating point.

Y = pow(X, N) calcola la potenza N-esima del numero floating point 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 ad 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 =

   0.5377 -2.2588 0.3188
   1.8339 0.8622 -1.3077
```

2.2 Il *range operator* :

Con la sintassi `a:t:b` creiamo un vettore (riga) che contiene gli elementi `a`, `a+t`, `a+2t`, `a+3t`, ... fino a `b` (o fino all'ultimo che sia minore o uguale a `b`). Se `t`=1, si può semplicemente usare il comando `a:b`

```
>> 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)
??? Index exceeds matrix dimensions.

>> 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 =

    0.8415 0.9093 0.1411

>> 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 una matrice per diagonali

È possibile inserire direttamente le entrate di una matrice che giacciono su una certa diagonale con il comando `diag`. Più 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
```

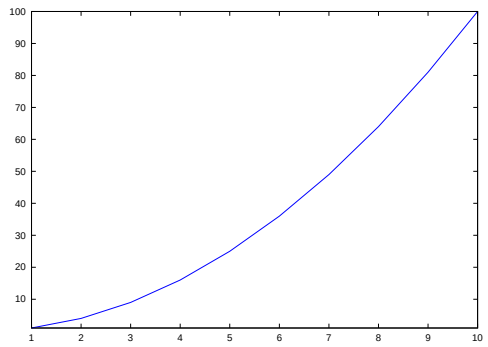
3 Grafici

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 1. 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+2*i,2)`. A quale ascissa/ordinata arriva il cerchio?

4 Cerchi di Gerschgorin

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

```
function gg(A)
  sz=size(A);
  n=sz(1); %cosa fanno queste due istruzioni?

  close all %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),'*');
  % '*': disegna solo i punti, non collegandoli con linee
  %'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 2. Testare la funzione `gg` su alcune matrici test: `rand(10)`, `randn(10)`, `hilb(10)`,

Esercizio 3. Scrivere una funzione `ggsecond(A)` che disegni i cerchi di Gerschgorin di A e mostri come variano gli autovalori quando gli elementi fuori dalla diagonale di `A` vengono ridotti pian piano fino a diventare zero, come nella dimostrazione del secondo teorema di Gerschgorin. Per farlo, generate tante matrici (diciamo $k = 100$) a intervalli uguali lungo il "segmento" che unisce A alla sua diagonale, e plottate i loro autovalori.

Esercizio 4. 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 5 (facoltativo). Scrivete una funzione `perturb(A,epsilon)` che, data una matrice A, generi 50 matrici i cui elementi sono gli elementi di A modificati con un valore casuale e piccolo (di grandezza circa `epsilon` ottenibile con le istruzioni `epsilon*(randn(n)+i*randn(n))`), e disegni 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? E per una matrice che ha un singolo blocco di Jordan di dimensione 5?

Esercizio 6 (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 scorsa lezione (`fact`, `pow`, `myexp`, `myexp2`) vettorizzando i cicli `for`, in modo che non siano più necessarie $O(n)$ istruzioni per calcolare un esponenziale?