Laboratorio di Calcolo Numerico
Lezione 2

1 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.

1.1 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
```

1.2 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 :?

1.3 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.

1.4 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.

1.5 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
```

1.6 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
```

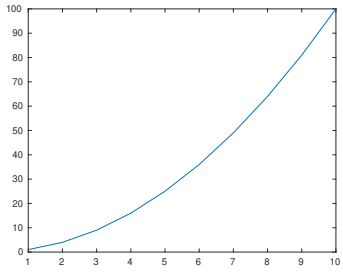
2 Grafici

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?

3 Sottomatrici e determinanti

Utilizzando l'operatore `:` (*range* operator), in MATLAB è possibile selezionare un'intera sottomatrice di una matrice:

```
>> A=[1 2 3; 4 5 6; 7 8 9]
A =

1 2 3
4 5 6
7 8 9

>> A(1:2,2:3)
ans =

2 3
5 6

>> A(2:end,1:end-2)
ans =

4
7

>> A(1:end,1:end)
ans =

1 2 3
4 5 6
7 8 9

>> A(1,:)
ans =

1 2 3
```

La sintassi `a:b` seleziona tutte le righe/colonne comprese tra `a` e `b` (estremi inclusi). Il valore `end` viene sostituito con il massimo indice disponibile. Il solo `:` è un'abbreviazione per `1:end`.

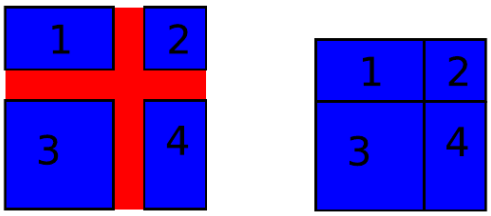
Possiamo anche assegnare un valore a una sottomatrice selezionata in questo modo:

```
>> A(1:2,1:2)=eye(2)
A =

1 0 3
0 1 6
7 8 9
```

Ovviamente le dimensioni devono essere compatibili: non posso selezionare una sottomatrice 2×2 e assegnarle il valore `eye(3)`!

La seguente function restituisce la sottomatrice corrispondente al *minore* (i, j) in A, cioè la matrice che si ottiene eliminando la i-esima riga e la j-esima colonna di A.




```
function B = submatrix_minor(A,i,j)
X=A(1:i-1,1:j-1);
Y=A(1:i-1,j+1:end);
Z=A(i+1:end,1:j-1);
W=A(i+1:end,j+1:end);
B=[X Y; Z W];
end
```

Abbiamo già visto che se `X,Y,Z,W` sono numeri, la riga di codice `B=[X Y; Z W]` crea la matrice

$$\begin{bmatrix} X & Y \\ Z & W \end{bmatrix}.$$

Ora vediamo che la stessa sintassi funziona anche se `X,Y,Z,W` sono matrici di dimensioni "compatibili" e crea la matrice formata accostando i quattro blocchi.

Esercizio 2. Creare una `function d=mydet(A)` che calcoli il determinante di una matrice quadrata A utilizzando la formula di Laplace sulla prima riga, cioè

$$\det(A) = A_{11} \det A^{(11)} - A_{12} \det A^{(12)} + A_{13} \det A^{(13)} - \cdots + (-1)^{n+1} A_{1n} \det A^{(1n)}$$

dove A_{ij} è l'elemento di A nella posizione (i, j) e $A^{(ij)}$ è la matrice minore di A rispetto a (i, j). [Hint: la funzione può essere *ricorsiva*, cioè chiamare se stessa al suo interno. Fate attenzione: bisogna definire un caso base!]

Poi testarla su alcune matrici, confrontandola con la funzione `det` di MATLAB, per esempio matrici con elementi casuali `V=randn(2)`, `V=randn(3)`, `V=randn(4)`...

Esercizio 3. La seguente function è equivalente a `submatrix_minor(A,i,j)`. Come funziona? [Hint: osservate cosa succede eseguendo `A(v,w)` dove `v` e `w` sono vettori contenenti interi.]

```
function B = submatrix_minor2(A,i,j)
B = A([1:i-1 i+1:size(A,1)],[1:j-1 j+1:size(A,2)]);
end
```

4 Tempi di calcolo

Le funzioni `tic` e `toc` misurano il tempo necessario ad eseguire più istruzioni. La prima fa partire il cronometro, la seconda lo ferma e restituisce il valore ottenuto. Per esempio, le istruzioni

```
tic;
A=randn(100);
B=randn(100);
A*B;
t=toc;
```

salvano in `t` il tempo necessario ad eseguire le due righe centrali.

La seguente funzione disegna un grafico del tempo impiegato per calcolare il prodotto di matrici $n \times n$, generate casualmente, con n crescente.

```
function plottimes();
n = [100, 200, 400, 800, 1600, 3200, 6400];
nn = length(n);
tempi=zeros(nn,1); %prepara un vettore vuoto con i tempi
for i=1:nn
       A=randn(n(i)); %le matrici vengono generate prima di ‘‘tic’’
       B=randn(n(i));
       tic;
       C=A*B;
       tempi(i)=toc;
end
plot(n,tempi);
end
```

I tempi di calcolo dovrebbero crescere come $\mathcal{O}(n^3)$, in particolare

$$\text{tempo}(2n) \approx 8 \cdot \text{tempo}(n).$$

Se mostrato in un grafico con scala logaritmica sugli assi x e y, ad esempio usando il comando

```
loglog(n, tempi)
```

l'andamento dovrebbe essere quello di una retta con coefficiente angolare 3 ($y = 3x + k$).

Esercizio 4. Calcolare i tempi di esecuzione del prodotto `A*B` e della somma `A+B` di matrici $n \times n$ per i valori crescenti di n considerati nell'esempio sopra. Mostrare i tempi su un grafico che utilizzi la scala logaritmica per entrambi gli assi (usare il comando `loglog` invece di `plot`), in modo da poter confrontare meglio tempi su scale molto diverse.

Per riferimento riportare anche i grafici di $y = n^3$ e $y = n^2$, per verificare che le pendenze delle rette rappresentanti i costi della moltiplicazione e n^3 coincidano, almeno asintoticamente, così come quelli della somma e della funzione n^2.

Per valori piccoli di n, ad esempio 100, per cui i tempi di esecuzione sono molto piccoli, ci possono essere delle oscillazioni nei tempi dovuti al fatto che MATLAB è un linguaggio interpretato. Nel caso si vogliano delle misurazioni più affidabili si può utilizzare la funzione `timeit` (si digiti `help timeit` per vedere come funziona) che fa una media dei tempi su un certo numero di esecuzioni.