

# LABORATORIO CALCOLO

**Come si digita un vettore / matrice?**

Il vettore è identificato dalle parentesi quadre [...], si distingue a questo punto tra:

1. Vettore Riga ---> separo le componenti con le virgole

2. Vettore Colonna ---> separo con il punto e virgola

Le matrici si scrivono come i vettori, quando voglio andare a capo metto il punto e virgola.

Esempio: Matrice identità (3x5)

$A = [1,0,0;0,1,0;0,0,1]$

1 0 0  
0 1 0  
0 0 1

**Comandi per matrici identità e matrici di zeri o uni**

1) eye (n) ci da la matrice identità di ordine n

2) zeros (n,m) ci da la matrice di zeri di n righe e m colonne

3) ones (n,m) ci da la matrice di uno

Per sostituire un qualsiasi elemento di una matrice con un valore:

$A(i,j) = a$

**Operatori per Matrici**

1) det (A) ci da il determinante di una matrice quadrata

2) eig (A) fornisce gli autovalori

3) rank (A) ci da il rango della matrice

**Come risolvere un sistema lineare**

Consideriamo una matrice B e una colonna di termini noti b, per risolvere il sistema lineare si usa il comando di backslash

$B \backslash b$  ---> trova un vettore che risolva il sistema  $B * x = b$

**Errore di cancellazione:**

Le operazioni non sono commutative infatti se provo a fare  $a=1e10$   $b=1e4$  e  $c=(a+b)^2$ , quando faccio  $c-a^2-b^2-2*a*b$  non viene zero ma se faccio  $c-b^2-a^2-2*a*b$  questo fa zero. Il motivo è che nel primo caso ho tolto un numero molto grande da un numero grande mentre nel secondo ho sottratto prima un numero piccolo e poi uno grande.

**Cosa significa il ; a fine riga**

Il punto e virgola nasconde la riga in modo che non venga stampata, mentre il % all'inizio della riga fa sì che il programma non legga il comando (è solo una nota)

**Operatore "for"**

Serve per fare una iterazione, in particolare un ciclo del tipo:

`for k =a:b`

`"istruzioneI";`

`end`

`ripete l'istruzione per a, a+1, a+2, ..., b`

*esempio: Matrice identità di ordine n*

```
n = ...;  
A = zeros (n);  
for k = 1:n;  
A(k,k)=1  
end  
A
```

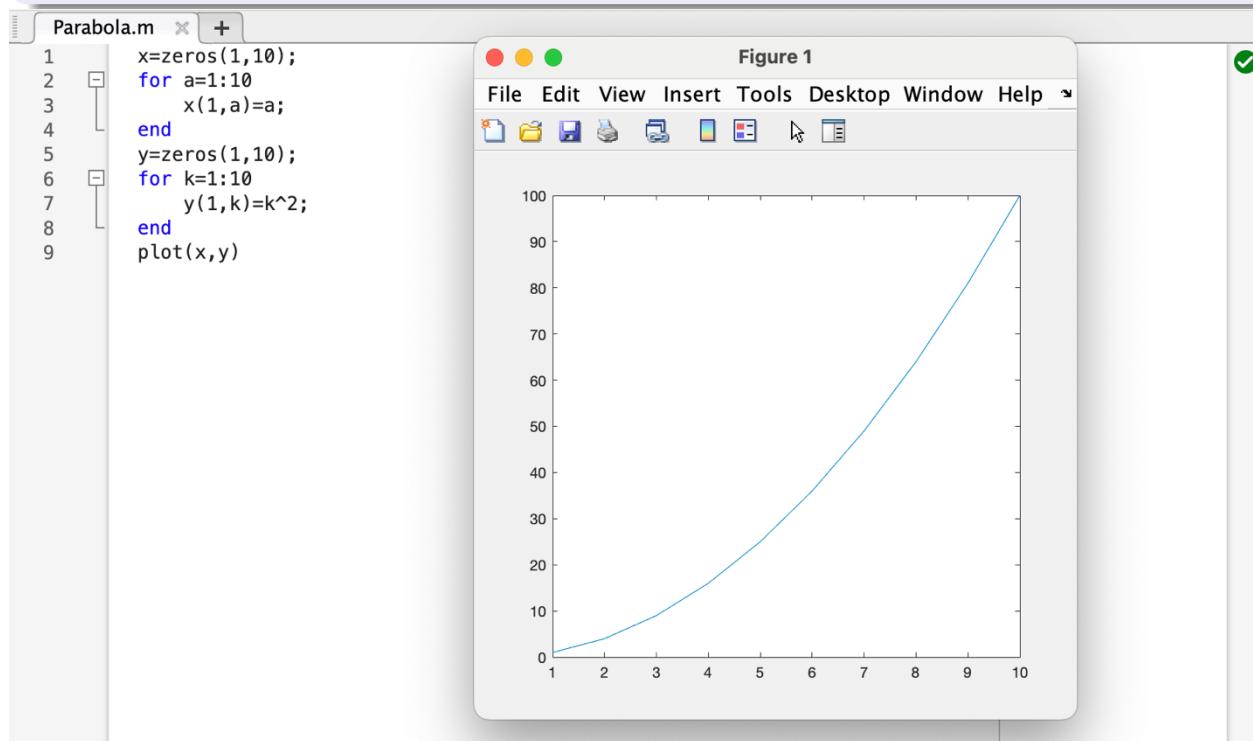
### Comando plot

*Il comando plot serve per graficare dei punti, con grafici a dispersione prendendo due vettori della stessa lunghezza*

*plot(x,y) ---> fa il grafico unendo le coppie di punti  $(x_i, y_i)$*

### Esercizio

Generare due vettori contenenti rispettivamente  $x = (1, 2, \dots, 10)$  e  $y = (1, 4, 9, 16, 25, \dots, 100)$ , e utilizzarli per tracciare sullo schermo il grafico della funzione  $y = x^2$ .



### Comando length

*Ci serve per scrivere la dimensione di un vettore x, ad esempio se prendo un vettore del tipo [1,2,3,4,5] ho che  $\text{length}(x) = 5$*

**Come faccio a infittire i punti del grafico?**

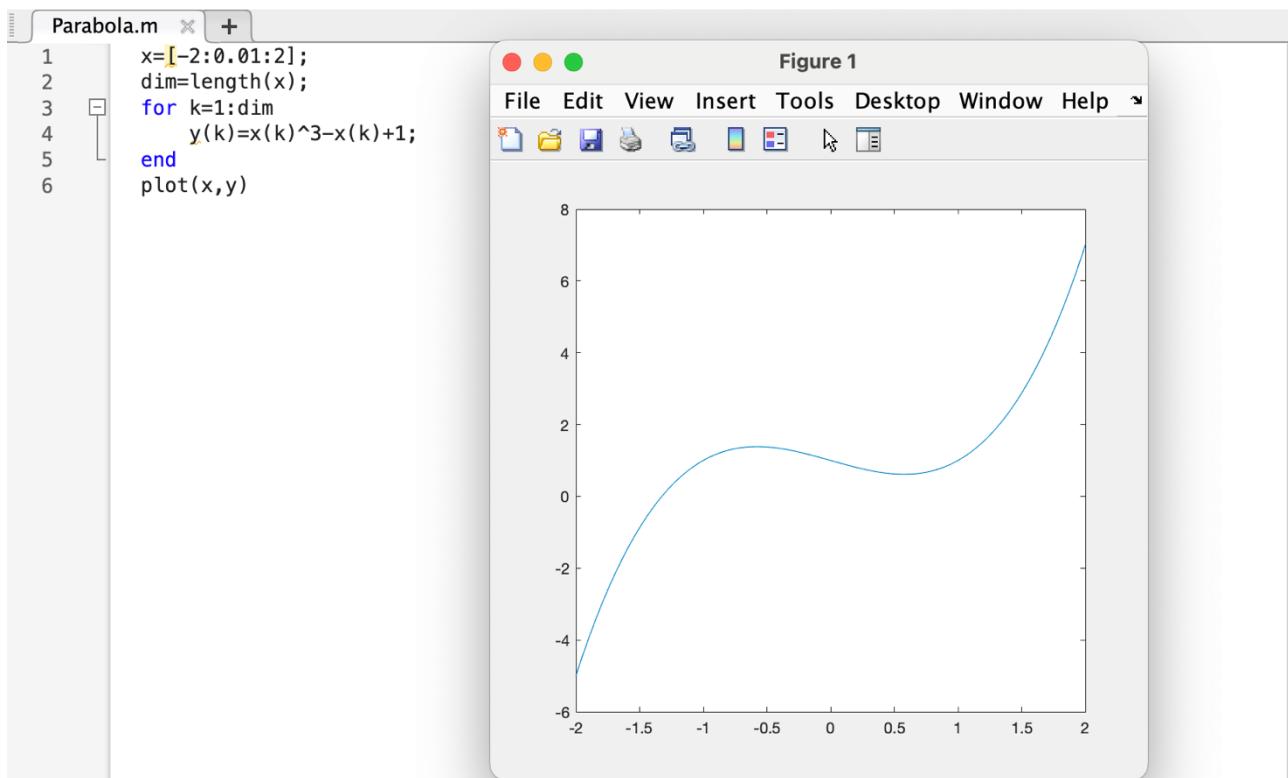
Per aumentare i passi che fa il *for*, posso usare il comando del passo  
 $for k = a:t:b \rightarrow k$  va da  $a$  fino a  $b$  compiendo ogni volta un passo  $t$

### Esercizio

Tracciare sullo schermo il grafico della funzione  $y(x) = x^3 - x + 1$ , disegnando la funzione 'per punti' su un numero sufficientemente alto di punti in  $[-2, 2]$ .

In questo caso non posso ripetere il procedimento di prima, poiché i vettori devono avere dimensione finita. Per far sì che un vettore assuma tutti i valori compresi in un compatto con un passo  $t$  devo scrivere

$$x = [a:t:b]$$



**Cicli for annidati (uno dentro l'altro)**

Ad esempio se voglio scrivere una matrice, avrò bisogno di due indici, uno che mi scorre le righe e uno che mi scorre le colonne.

Esempio: scrivere uno script che stampi la seguente matrice:

A =

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

Metto un for per le righe ( $k$ ) e uno per le colonne ( $t$ ), per poi usare due end per chiudere:

```

1 A = zeros(9,9);
2   for k=1:9
3     for t=1:9
4       A(k,t)=10*k+t;
5     end
6   end
7   A
8

```

Se metto nel primo for l'indice riga, riempie prima le righe e poi riempie le colonne; se inverti i for mi riempie prima le colonne. (si può vedere eseguendo il programma senza ;)

Esercizio: scrivere la matrice delle tabelline

```

1 dim =10
2 A = zeros(dim,dim);
3   for k=1:dim
4     for t=1:dim
5       A(k,t)= k*t;
6     end
7   end
8   A
9

```

Esercizio: scrivere la matrice triangolare

B =

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

```

1 B=zeros(5,5);
2   for i=1:5
3     for j=1:i
4       B(i,j)=j;
5     end
6   end
7   B

```

Esercizio: scrivere la matrice

```

0 1 0 0 0 0 0 0
1 0 1 0 0 0 0 0
0 1 0 1 0 0 0 0
0 0 1 0 1 0 0 0
0 0 0 1 0 1 0 0
0 0 0 0 1 0 1 0
0 0 0 0 0 1 0 1
0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0

```

```

1 n=10
2 B=zeros(n)
3   for i=1:n-1
4     B(i+1,i)=1;
5     B(i,i+1)=1;
6   end
7   B

```

## Istruzione if

Serve per porre condizioni vere o false, ad esempio se voglio un programma che stampa un numero minimo:

```
if a < b  
    minimo = a  
else  
    minimo = b  
end
```

Le condizioni logiche che posso porre sono le seguenti:

### Operatori logici

Uguale	<code>==</code>	<code>3 == 2 + 1</code>	
Diverso	<code>~=</code>	<code>3 ~= 5</code>	o anche <code>not(3 == 5)</code>
Minore	<code>&lt;</code>	<code>2 &lt; 3</code>	
Minore o uguale	<code>&lt;=</code>	<code>2 &lt;= 3</code>	
Maggiore	<code>&gt;</code>	<code>5 &gt; 3</code>	
Maggiore o uguale	<code>&gt;=</code>	<code>5 &gt;= 3</code>	
Oppure	<code>  </code>	<code>k&lt;1    k&gt;5</code>	
Entrambi	<code>&amp;&amp;</code>	<code>k&gt;=1 &amp;&amp; k&lt;=5</code>	

Esempio: crea la matrice di prima usando if e gli operatori logici

```
1 n=10  
2 C=zeros(n,n);  
3 for i=1:n  
4     for j=1:n  
5         if i==j+1 || i+1==j  
6             C(i,j)=1  
7         end  
8     end  
9 end
```

### Metodo del Punto Fisso

Quali sono i punti fissi della funzione  $\Phi(x) = \frac{1}{4} + \frac{3}{4}x^2$ ?

Per il metodo di punto fisso con  $x_1 = 0$ , quanto valgono  $x_2$  e  $x_3$ ?

Scrivere uno *script* che esegue 20 iterazioni del metodo, e salva i risultati in un vettore  $X = [x_1, x_2, \dots, x_{21}]$ . Controllare che  $x_2, x_3$  coincidano con quelli calcolati sopra. A quale dei punti fissi converge il metodo?

I-2) Calcolo  $f(x)$  e i suoi punti fissi, successivamente calcolo  $x_2$  e  $x_3$  partendo da  $x_1=0$

$$f(x) = \Phi(x) - x = \frac{1}{4} + \frac{3}{4}x^2 - x$$

$$\alpha_1 = 1/3 \quad \alpha_2 = 1$$

$$x_1 = 0 \Rightarrow x_2 = \frac{1}{4} \Rightarrow x_3 = 19/64$$

3) Eseguo un ciclo for per una matrice di 21 componenti, con la prima componente uguale a  $x_1$ :

```

1 %Metodo del Punto Fisso
2 X(1,1)=0;
3 X = zeros(1,21);
4 for k=1:20
5   X(1,k+1)=1/4+3/4*X(1,k)^2;
6 end
7 X

```

	X =
Columns 1 through 10	0 0.2500 0.2969 0.3161 0.3249 0.3292 0.3313 0.3323 0.3328 0.3331
Columns 11 through 20	0.3332 0.3333 0.3333 0.3333 0.3333 0.3333 0.3333 0.3333 0.3333 0.3333
Column 21	0.3333

### Metodo di Newton

Scrivere uno *script* che esegue 10 iterazioni del metodo di Newton sulla funzione  $f(x) = \frac{1}{4} + \frac{3}{4}x^2 - x$ , salvando i risultati in un vettore  $X = [x_1, x_2, \dots, x_{11}]$ .

I) Calcolo la derivata e scrivo  $\phi(x)$  secondo il metodo di newton:

```

1 %Metodo del Punto Fisso
2 X(1,1)=0;
3 X = zeros(1,11);
4 for k=1:10
5   X(1,k+1)=X(1,k)-((0.25+0.75*X(1,k)^2-X(1,k))/(1.5*X(1,k)-1));
6 end
7 X

```

```

X =
Columns 1 through 10

    0    0.2500    0.3250    0.3332    0.3333    0.3333    0.3333    0.3333    0.3333    0.3333

Column 11

    0.3333

```

### **Funzione degli errori**

*Guardando i metodi precedenti, abbiamo che la funzione:*

*\*abs (X - α)\* da l'errore rispetto alla distanza per ogni iterazione dallo zero effettivo della funzione*

### **Comando grafico in scala logaritmica**

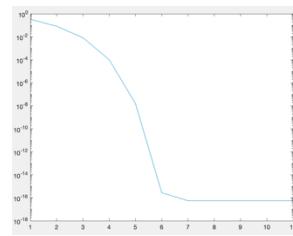
*semilogy(X - α) mi da la funzione delle distanze in scala logaritmica*

*Esempio:*

```

1 %Metodo del Punto Fisso
2 X(1,1)=0;
3 X = zeros(1,11);
4 for k=1:10
5     X(1,k+1)=X(1,k)-((0.25+0.75*X(1,k)^2-X(1,k))/(1.5*X(1,k)-1));
6 end
7 X;
8 semilogy(abs(X-1/3))

```



### **Come si definisce implicitamente una funzione?**

*Prendiamo ad esempio il metodo del punto fisso, voglio definire la funzione in modo da poterla cambiare quando mi pare. Si utilizza il comando “@(x)”*

*Esempio:*

$$\Phi = @(x) 1/4 + 3/4 * x^2$$

```

N=20;
Phi = @(x) 1/4 +3/4 * x^2;
punto_iniziale = 0;

```

```

X=zeros(N+1,1);
X(1) = punto_iniziale;
for n = 1:N
    X(n+1) = Phi(X(n));
end
X

```

*Osservazione: se metto punto\_iniziale = 2 ho che per il teorema del punto fisso la successione diverge e infatti i valori tendono a Inf*

## Come impostare un criterio di arresto?

È possibile crearlo con un if, ad esempio se voglio che  $|x_n - x| < 10^{-5}$ , metto un ciclo if con la mia condizione di arresto e se è verificata uso il comando "break", che mi termina l'esecuzione dell'algoritmo.

```
1 N=20;                                X =
2 Phi = @(x) 1/4 +3/4 * x^2;
3 punto_iniziale = 0;
4 epsilon = 10^-5;
5
6 X=zeros(N+1,1);
7 X(1) = punto_iniziale;
8 for n = 1:N
9     X(n+1) = Phi(X(n));
10    if abs (X(n+1)-X(n)) < epsilon
11        break %mi termina l'esecuzione
12    end
13 end
14 X
```

11 ITERAZIONI →

X =	0
	0.2500
	0.2969
	0.3161
	0.3249
	0.3292
	0.3313
	0.3323
	0.3328
	0.3331
	0.3332
	0.3333
	0.3333
	0.3333
	0
	0
	0
	0
	0
	0

Se volessi che ad ogni passaggio mi butti via le iterazioni precedenti, prendendo solo l'ultima (dato che per calcolare  $X(n)$  ho bisogno solo di  $X(n-1)$ ).

Ho bisogno quindi di cambiare il valore di scalare ogni volta i valori in modo da usare un cassetto unico e non  $X(1), X(2), \dots$

Uso solo due valori  $X_{old}$  e  $X_{new}$ :

```
Phi = @(x) 1/4 +3/4 * x^2;
punto_iniziale = 0;
epsilon = 10^-5;
N=20; %numero massimo di iterazione
```

```
Xold = punto_iniziale;
for n = 1:N
    Xnew = Phi(Xold);
    if abs (Xnew - Xold) < epsilon
        break
    else
        Xold = Xnew;
    end
end
Xnew
```

Osservazione: se voglio che mi dia tutte e 16 le cifre scrivo "format long".

## **Comando “while”**

*Lavorare con l'N numero di iterazioni non è il top, quindi posso sostituire il for e l'if con un unico comando.*

```
1 Phi = @(x) 1/4 +3/4 * x^2;
2 punto_iniziale = 0;
3 epsilon = 10^-5;
4 errore = Inf; → Fa funzionare la
5 prima iterazione
6
7 Xnew = punto_iniziale;
8 while errore >= epsilon
9     Xold = Xnew;
10    Xnew = Phi(Xold);
11    errore = abs(Xnew - Xold);
12 end
13 Xnew
```

*“Fintanto che il criterio di arresto non è verificato, esegui comandi (sostituisci Xold con Xnew e calcola il nuovo Xnew con la funzione phi)”*

*I tre comandi sono sequenziali, ossia sostituiscono i valori fissati all'inizio*

### **Comandi “and” e “or”**

*“and” si usa per imporre più condizioni contemporaneamente, ad esempio (esercizio che aveva lasciato per casa)*

## *Metodo di Bisezione*

*Supponiamo di avere una funzione continua, prendo un intervallo  $[a,b]$  in cui  $f(a) * f(b) < 0$ , ossia ho uno zero.*

Prendo il punto intermedio tra  $a$ ,  $b$  e calcolo  $f(c)$ , restringo l'insieme.

```

1 f = @(x)1/4 +0.75*x^2 - x;
2 A = zeros(20,1);
3 B = zeros(20,1);
4 C = zeros (20,1);
5 A(1)= 0;
6 B(1)= 1;
7 N = 20;
8
9 for k=1:N
10    C(k)= 1/2*(A(k)+B(k));
11    if f(C(k))==0
12        break
13    else
14        if f(C(k))*f(A(k))<0
15            A(k+1)=A(k);
16            B(k+1)=C(k);
17        else
18            A(k+1)= C(k);
19            B(k+1)= B(k);
20        end
21    end
22 end
23 C(N)
ans =
0.333333015441895

```

*Il metodo di bisezione permette di sapere il numero di iterazioni da applicare per avere una data precisione.*

*Mettiamo che io voglia una precisione epsilon = 10^-4*

```
epsilon = 10e-4  
N=ceil(log2((B(1)-A(1))/epsilon));
```

Il comando “ceil” mi restituisce un’approssimazione all’intero

## **Funzioni su MatLab**

*Sono modi per organizzare diversamente il codice, quando il programma incontra la funzione, calcola i valori di input ed esegue tutto il codice risultante dopo la riga function.*

Importante: Bisogna salvare lo script con lo stesso nome della funzione per far si che possa riconoscerla

Esempio: Fattoriale

```
fattoriale.m  
function F = fattoriale (n)  
% F = 1*2*3*...*n  
F = 1;  
for k=1:n  
    F= F*k;  
end
```

>> fattoriale(3)  
ans =  
6

## **Accumulatori**

*Sono delle particolari variabili che vengono aggiornate ogni iterazione, ad esempio se si vuole fare la somma:*

```
function [s] = somma(v)  
% v un vettore del quale vogliamo  
% calcolare la somma delle componenti  
n=length(v);  
s=0; %accumulatore della somma  
for i=1:n  
    s=s+v(i);  
end  
end
```

s=s+v(i); → Accumulatore

## **Esercizio**

Si scriva una funzione `function y=potenzaintera(x,n)` che utilizzando un accumulatore calcoli  $y = x^n$  ottenuto moltiplicando  $n$ -volte  $x$  per se stesso.

```
1 function potenzaintera(x,n)  
2 y=x;  
3 for k=1:n-1  
4     x=x*y;  
5 end  
6  
7 x
```

## Approssimazione del Polinomio di Taylor

$f(x) = \exp(x)$  è  $C^\infty(\mathbb{R})$

Il polinomio di Taylor per  $x_0 = 0$ , arrestato al termine  $n$ -esimo della funzione esponenziale è

$$EXP(x, n) = \sum_{k=0}^n \frac{x^k}{k!} = 1 + x + \frac{x^2}{2} + \frac{x^3}{3!} + \cdots + \frac{x^n}{n!}.$$

### Esercizio

Si scriva una funzione `function y=myexp(x,n)` che approssima il valore di  $\exp(x)$  utilizzando il polinomio di Taylor di grado  $n$  calcolato in  $x$ . Si possono utilizzare le funzioni `fattoriale(n)` e `potenzaintera`.

```
1 [-] function y = myexp(x,n)
2 [-] V=zeros(1,n);
3 [-] for k=1:n;
4 [-]     V(1,k)= potenzaintera(x,k)/fattoriale(k);
5 [-] end
6 [-] y=1+somma(V);
7 [-]
8 [-]
```

Questa in realtà costa troppo, poiché per ogni indice mi deve calcolare la potenza e il fattoriale, vediamo un modo meno impegnativo, con un costo lineare.

```
1 [-] function y = myexp(x,n)
2 [-] %calcola un'approssimazione della funzione esponenziale usando il polinomio
3 [-] %di Taylor di grado n
4 [-] y=1;
5 [-] t=1;
6 [-] for i=1:n
7 [-]     t=t*x/i;
8 [-]     y=y+t;
9 [-] end
10 [-]
```

Osserviamo inoltre che l'algoritmo si comporta male con i numeri negativi, come fare? Metto un If:

```
1 [-] function y = myexp3(x)
2 [-] %calcola un'approssimazione della funzione esponenziale usando il polinomio
3 [-] %di Taylor di grado n
4 [-] n=500;
5 [-] if x>=0
6 [-]     y=myexp(x,n);
7 [-] else
8 [-]     y=1/myexp(-x,n);
9 [-] end
```

## **Dimensione di una matrice**

L'operatore “size( $A$ )” mi da un vettore che dà il numero di righe e di colonne.

## **Moltiplicazione Matrice x Vettore**

MatLab in realtà ha già la sua funzione per il prodotto vettore, ma noi vogliamo farlo per conto nostro.

```
1 [-] function y = mat_vec(A,b)
2     l=length(b);
3     [n,m]=size(A);
4     y=zeros(l,1);
5     if not(m==l)
6         error('dimensioni non compatibili')
7     end
8 [-] for i=1:l
9 [-]     for j=1:n
10        z=A(i,j)*b(j,1);
11        y(i)=y(i)+z;
12    end
13 end
14
```

## **Risoluzione di un Sistema Diagonale**

### Esercizio

Si scriva una funzione `function y = diag_solve(A, b)` che preso in ingresso la matrice  $A$  **diagonale** e il vettore  $b$  dei termini noti risolve il sistema  $Ax = b$ .

```
1 [-] function y = diag_solve(A,b)
2     [n,m]=size(A);
3     y=zeros(m,1);
4     l=length(b);
5     if not(l==n)
6         error('impossibile')
7     end
8 [-] for i=1:m
9     y(i)=b(i)/A(i,i);
10    end
```

## Sistema Triangolare Superiore (metodo di sostituzione)

### Esercizio

Scriviamo una function `x = sup_solve(A)` che risolve un sistema lineare triangolare superiore

Devo optare per una sostituzione all'indietro, quindi l'ultima riga sarà quella da cui partono le mie iterazioni.

```
1 function x=sup_solve(A,b)
2 [n,m]=size(A);
3 n=length(b);
4 x=zeros(n,1);
5 x(n)=b(n)/A(n,n);
6 for i=n-1:-1:1
7     y=0;
8     for j=i+1:n
9         y=y+A(i,j)*x(j);
10    end
11    x(i)=(b(i)-y)/A(i,i);
12 end
13
14
15
```

## Sistema Triangolare Inferiore (metodo di sostituzione)

Stavolta il sistema è ancora più semplice poiché la matrice viene letta dall'alto verso il basso:

```
1 function x=inf_solve(A)
2 [n,m]=size(A);
3 b=A(:,m);
4 lb=length(b);
5 x=b;
6 x(1)=b(1)/A(1,1);
7 for i=2:lb
8     for j=1:i-1
9         x(i)=x(i)-A(i,j)*x(j);
10    end
11    x(i)=x(i)/A(i,i);
12 end
13
14
15
```

### **Funzioni “speye” e “sprandn”**

Creano matrici con elementi sparsi, in particolare la funzione “speye( $n$ )” crea una matrice identità con elementi non nulli sulla diagonale.

La funzione “sprandn( $n,m,k$ )” crea una matrice  $n \times m$  con una percentuale di  $k$  elementi diversi da zero.

Esempio: sprandn(4,5,0.2)

```
>> sprandn(4,5,0.2)
```

ans =

(4,1)	2.7694
(1,2)	3.5784
(4,3)	-1.3499
(4,4)	3.0349

### **Fattorizzazione $LDL^T$**

È una fattorizzazione per matrici simmetriche, difatti si nota che la matrice  $U$  moltiplicata a sinistra per la matrice  $E_1$  e a destra per la trasposta di  $E_1$  produce una matrice simmetrica con zeri sia sotto la colonna del pivot che nella riga a destra di esso.

Al passo 1 ad esempio si ha:

$$U_2 = E_1 \cdot U_1 \cdot E_1^T$$

$$U_2 = \begin{bmatrix} A_{11} & 0 & 0 & 0 \\ 0 & \star & \star & \star \\ 0 & \star & \star & \star \\ 0 & \star & \star & \star \end{bmatrix} . \quad \xrightarrow{\hspace{10em}} \boxed{\text{Simmetrica, ciò significa che anche la matrice } U_2 \text{ è simmetrica così come tutte quelle successive}}$$

Alla fine della fattorizzazione si ottiene una matrice diagonale  $D$

$$D = E_{n-1} E_{n-2} \dots E_2 E_1 A E_1^T E_2^T \dots E_{n-2}^T E_{n-1}^T.$$

La fattorizzazione per cui non sarà  $LU$  ma  $LDL^T$ . Su Matlab esiste una funzione apposita che calcola la matrice  $L$  e la matrice  $D$ .

$[L, D] = Idl(A)$

### **Fattorizzazione di Cholesky**

È una riscrittura della fattorizzazione  $LDL^T$ , in pratica riscrive la matrice  $D$  come prodotto tra  $D^{1/2} D^{1/2}$  in questo modo:

$$A = LDL^T = (LD^{1/2})(D^{1/2}L^T) = R^T R,$$

Dove  $R$  è una matrice triangolare superiore. In pratica scrivo  $A$  come prodotto tra una matrice triangolare superiore e inferiore.

Anche in questo caso esiste una funzione in Matlab che calcola la matrice  $R$ :

$$R = \text{chol}(A)$$

### Esercizio:

Consideriamo la matrice  $5 \times 5$  che ha 5 sulla diagonale e  $-1$  fuori

$$A = \begin{bmatrix} 5 & -1 & -1 & -1 & -1 \\ -1 & 5 & -1 & -1 & -1 \\ -1 & -1 & 5 & -1 & -1 \\ -1 & -1 & -1 & 5 & -1 \\ -1 & -1 & -1 & -1 & 5 \end{bmatrix}.$$

- Scrivere due function `x = risolvi_ldl(L, D, b)` e `function x = risolvi_chol(R, b)` che prendono in input  $\mathbf{b} \in \mathbb{R}^n$  e i fattori di queste due fattorizzazioni, rispettivamente, e li usano per risolvere  $A\mathbf{x} = \mathbf{b}$  (potete usare funzioni già scritte, ad es. `sup_solve`).
- Scrivere uno script che genera un  $\mathbf{x}$  casuale, costruisce  $\mathbf{b} = A\mathbf{x}$  in modo che il vettore  $\mathbf{x}$  sia la soluzione del sistema lineare  $A\mathbf{x} = \mathbf{b}$ , risolve il sistema lineare con le due fattorizzazioni, e visualizza l'errore relativo delle due soluzioni calcolate.

Per `risolvi_ldl` considero un sistema composto da tre equazioni matriciali, ossia risolvo tre sistemi in modo successivo; per `risolvi_chol` faccio la stessa cosa risolvendo due sistemi relativi a  $R$  e alla sua trasposta.

$$\begin{cases} L\mathbf{u} = \mathbf{b} \\ D\mathbf{v} = \mathbf{u} \\ L^T\mathbf{x} = \mathbf{v} \end{cases} \quad \begin{cases} R^T\mathbf{u} = \mathbf{b} \\ R\mathbf{x} = \mathbf{u} \end{cases}$$

```
function x=risolvi_ldl(L,D,b)
n=length(b);
x=zeros(n,1);
u=inf_solve(L,b);
v=diag_solve(D,u);
x=sup_solve(L',v);
```

```
function x=risolvi_chol(R,b)
n=length(b);
x=zeros(n,1);
y=inf_solve(R',b);
x=sup_solve(R,y);
```

```
x=randn(5,1);
b=A*x;
[L,D]=ldl(A);
R=chol(A);
x_ldl=risolvi_ldl(L,D,b);
x_chol=risolvi_chol(R,b);
errore_ldl=norm(x-x_ldl)/norm(x)
errore_chol=norm(x-x_chol)/norm(x)
```

errore\_ldl =

1.6504e-16

errore\_chol =

6.9232e-16

## Metodo Gauss – Seidel

Risolviamo il seguente esercizio con il metodo di Gauss – Seidel:

- Scrivere una function `xnew = gs_step(A, b, xold)` che esegue un passo del metodo di Gauss–Seidel; non creando le matrici  $M$  ed  $N$  ma con la formula

$$x_i^{(new)} = \frac{b_i - \sum_{j < i} A_{ij}x_j^{(new)} - \sum_{j > i} A_{ij}x_j^{(old)}}{A_{ii}}.$$

- Verificare che per  $A = \begin{bmatrix} 3 & -1 & -1 \\ -1 & 3 & -1 \\ -1 & -1 & 3 \end{bmatrix}$ ,  $b = \begin{bmatrix} 1 \\ 4 \\ 6 \end{bmatrix}$ ,  $x^{(old)} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$  si ha  $x^{(new)} = \begin{bmatrix} 1 \\ \frac{1}{2} \\ 3 \end{bmatrix}$ . Usare questi valori per controllare il buon funzionamento. Verificare, con Matlab, che vale la condizione di convergenza  $\rho(H) < 1$ .
- Scrivere una function `x = gs(A, b, x1, m)` che (chiamando `gs_step`) esegue  $m$  passi del metodo di Gauss–Seidel.
- Modificarla in function `[x, v] = gs_errore(A, b, x1, m, z)`, in modo che restituisca anche la norma- $\infty$  delle differenze tra iterate e uno  $z \in \mathbb{R}^n$  dato:

$$v = [v_1, v_2, \dots, v_{m+1}] = [\|x^{(1)} - z\|_\infty, \|x^{(2)} - z\|_\infty, \dots, \|x^{(m+1)} - z\|_\infty].$$

- Plottare  $v$  (con  $z = A \setminus b$  la soluzione esatta) in scala logaritmica, e verificare che  $\frac{v_{k+1}}{v_k} \rightarrow \rho(H)$ .
- Scrivere function `x = gs_arresto(A, b, x1, epsilon)` che si arresti quando  $\|Ax^{(new)} - b\|_\infty \leq \epsilon$ .
- Ripetere tutto con il metodo di Jacobi. Quale è il metodo più veloce?

- Punto 1:

```
function xnew = gs_step(A,b,xold)
n=length(xold);
xnew=xold;
for i=1:n
    sum_new=0;
    sum_old=0;
    for j=1:n
        if j<i
            sum_new=sum_new+A(i,j)*xnew(j);
        end
        if j>i
            sum_old=sum_old+A(i,j)*xold(j);
        end
    end
    xnew(i)=(b(i)-sum_new-sum_old)/A(i,i);
end
```

>> gs\_step(A,b,xold)  
ans =  
1  
2  
3

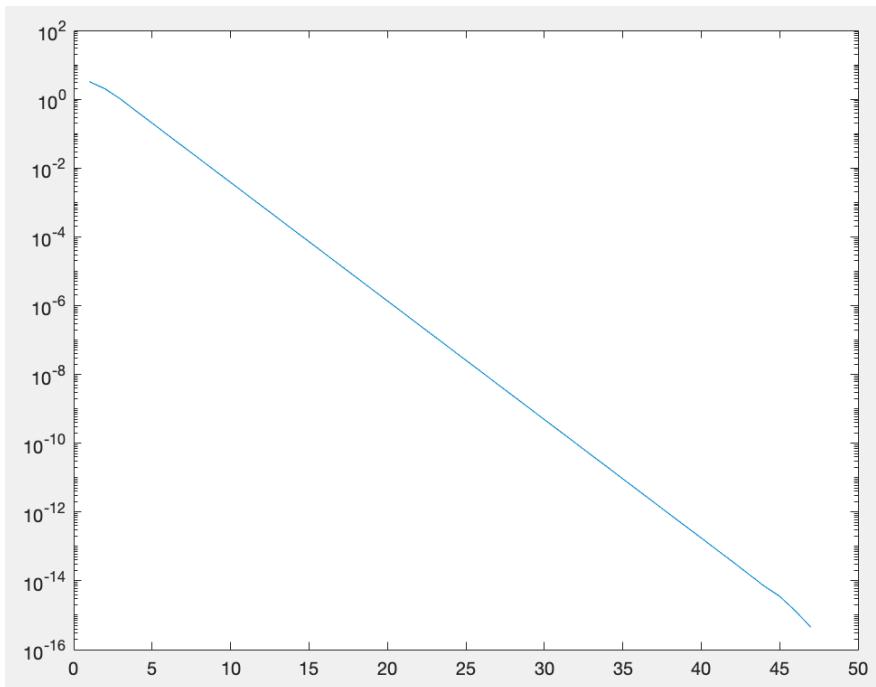
- Punto 2:

```
function x=gauss_seidel(A,b,x1,m)
xold=x1;
for k=1:m
    x=gs_step(A,b,xold)
    xold=x
end
```

- Punto 3:

```
function [x,v]=gs_errore(A,b,x1,m,z)
x=x1;
v=norm(x1-z,inf);
for k=1:m
    x=gs_step(A,b,x);
    v(k+1)=(norm((x-z),Inf));
end
semilogy(v)
```

- Punto 4: Ho plottato con  $m=60$



- Punto 5:

```
function x=gs_arresto(A,b,x1,epsilon)
x=x1;
while norm(A*x-b,Inf)>=epsilon
    x=gs_step(A,b,x);
end
```

### Metodo di Jacobi

Ripeti lo stesso esercizio fatto per il metodo Gauss – Seidel con Jacobi, qual è il metodo più veloce?

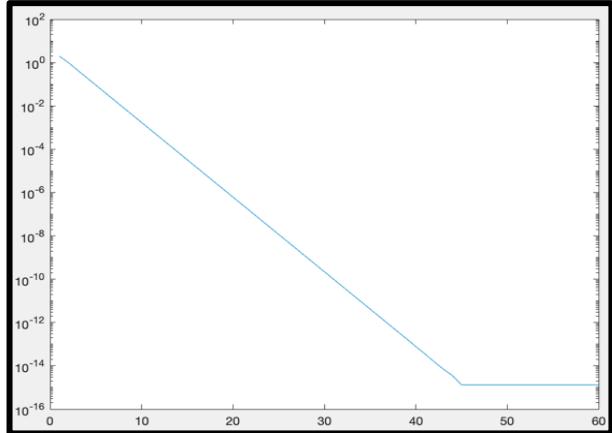
```
function x=jacobi(A,b,x1,m)
x=x1;
for k=1:m
x=jacobi_step(A,b,x);
end
```

```
function [x,v]=jacobi_errore(A,b,x1,m,z)
x=x1;
v=norm(x-z,Inf);
for k=1:m
x=jacobi_step(A,b,x);
v(k)=norm(x-z,Inf);
end
semilogy(v)
```

```

function x=jacobi_arresto(A,b,x1,epsilon)
x=x1;
while norm(A*x-b)>=epsilon
    x=jacobi_step(A,b,x);
end

```



```

function xnew=jacobi_step(A,b,xold)
n=length(b);
xnew=xold;
for i=1:n
    sum=b(i);
    for j=1:i-1
        sum=sum-A(i,j)*xnew(j);
    end
    for j=i+1:n
        sum=sum-A(i,j)*xnew(j);
    end
    xnew(i)=sum/A(i,i);
end
>> jacobi_step(A,b,xold)
ans =
    1
    2
    3

```

# SISTEMI NON LINEARI

## Metodo di Newton Multivariato

- Scrivere nella forma  $F(\mathbf{x}) = 0$  il sistema di equazioni non-lineari

$$2u^2 + v + w = 1.$$

$$u + 3v^2 + w = 1,$$

$$u + v + 4w^2 = 1,$$

con  $\mathbf{x} = [u, v, w]^T \in \mathbb{R}^3$  and  $F : \mathbb{R}^3 \rightarrow \mathbb{R}^3$ , e calcolarne lo Jacobiano  $J_F$ .

- Scrivere  $\mathbf{F} = @(x) [2*x(1)^2 + \dots]$  e  $J\mathbf{F} = @(x) [4*x(1) \dots]$  che calcolano  $\mathbf{F}$  e  $J_F$  su un vettore  $\mathbf{x}$  in input.
- Per controllare che lo Jacobiano sia corretto, generare vettori casuali  $\mathbf{x} = \text{randn}(3,1)$ ,  $\mathbf{h} = 1e-5 * \text{randn}(3,1)$  e controllare che  $\mathbf{F}(\mathbf{x} + \mathbf{h}) - \mathbf{F}(\mathbf{x}) - J_F(\mathbf{x})\mathbf{h} = O(\|\mathbf{h}\|^2)$ .
- Scrivere una function  $\mathbf{x}_2 = \text{multinewton\_step}(\mathbf{F}, J\mathbf{F}, \mathbf{x}_1)$  che esegue un passo del metodo di Newton multivariato. Testarla sul sistema precedente con  $\mathbf{x}^{(1)} = [1, 1, 1]^T$ , che dovrebbe produrre  $\mathbf{x}^{(2)} = [\frac{1}{2}, \frac{1}{2}, \frac{1}{2}]^T$ .
- Scrivere function  $[\mathbf{x}, \mathbf{e}] = \text{multinewton}(\mathbf{F}, J\mathbf{F}, \mathbf{x}_1, m, z)$  che esegue  $m$  passi del metodo e restituisce anche il vettore degli errori  $e_k = \|\mathbf{x}^{(k)} - \mathbf{z}\|_\infty$ ,  $k = 1, 2, \dots, m + 1$ .

1) Come prima cosa devo calcolare a mano il Jacobiano di  $F$  facendo quindi le derivate; impostiamo che  $x$  sia il vettore colonna  $x = [u; v; w]$ , per cui:

$$u = x(1)$$

$$v = x(2)$$

$$z = x(3)$$

$$F = \begin{bmatrix} 2x_1^2 + x_2 + x_3 - 1 \\ x_1 + 3x_2^2 + x_3 - 1 \\ x_1 + x_2 + 4x_3^2 - 1 \end{bmatrix} \quad JF = \begin{bmatrix} 4x_1 & 1 & 1 \\ 1 & 6x_2 & 1 \\ 1 & 1 & 8x_3 \end{bmatrix}$$

2) La verifica facciamola con ad esempio 20 vettori generati a caso

```
for k =1:20
x=randn(3,1);
h=(1e-5)*randn(3,1);
F = @(x) [2*x(1)^2+x(2)+x(3)-1;x(1)+3*x(2)^2+x(3)-1;x(1)+x(2)+4*x(3)^2-1];
JF = @(x) [4*x(1),1,1;1,6*x(2),1;1,1,8*x(3)];
if F(x+h)-F(x)-JF(x)*h >= norm(h)^2
error('errore')
else
    'corretto'
end
end
```

3) Al passo  $k+1$  avremo che  $x(k+1)$  sarà il vettore generato da:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - (J_F(\mathbf{x}^{(k)}))^{-1} F(\mathbf{x}^{(k)})$$

Scriviamo ad esempio il programma di step che calcola il vettore del passo successivo

```
function x2=multinewton_step(F,JF,x1)
x2 = x1-((JF(x1))^-1)*F(x1);
```

Verifichiamo i valori richiesti dal problema:

```
>> multinewton_step(F,JF,[1;1;1])
```

```
ans =
```

```
0.5000
0.5000
0.5000
```

4) A questo punto mi basta inizializzare  $x$  e iterare  $m$  volte con un ciclo for:

```
function [x,e]=multinewton(F,JF,x1,m,z)
x=x1;
e=zeros(m+1,1);
e(1)=norm(x-z,inf);
for k=1:m
    x=multinewton_step(F,JF,x);
    e(k+1)=norm(x-z,inf);
end
x
e
```

### Metodo delle corde multivariato

Scrivere function  $[x, e] = \text{multicorde}(F, A, x_1, m, z)$  che esegue  $m$  passi del metodo delle corde, calcolando una volta sola la fattorizzazione LU della matrice  $A$ . Testare il metodo con  $A = J_F(\mathbf{x}^{(1)})$  e  $A = J_F(\mathbf{x}_*)$ .

Il sistema è sempre quello precedente, per cui:

$$F = \begin{bmatrix} 2x_1^2 + x_2 + x_3 - 1 \\ x_1 + 3x_2^2 + x_3 - 1 \\ x_1 + x_2 + 4x_3^2 - 1 \end{bmatrix} \quad JF = \begin{bmatrix} 4x_1 & 1 & 1 \\ 1 & 6x_2 & 1 \\ 1 & 1 & 8x_3 \end{bmatrix}$$

Come prima cosa occorre calcolare la matrice  $A$  come jacobiano di  $F$  e fattorizzarlo tramite la funzione  $[L, U, P] = \text{lu}(A)$

Successivamente si va a impostare il sistema:

$$\begin{cases} A = JF(\mathbf{x}^{(k)}) = LUP^T \\ A * \mathbf{h} = F(\mathbf{x}^{(k)}) \end{cases}$$

Da cui si ottiene che al passo k:

$$\mathbf{h} \cdot (LUP^{-1}) = F(\mathbf{x}^{(k)})$$

Con la funzione backslash vado a moltiplicare a sinistra per le inverse di L e U e per P così com'è:

$$\mathbf{h} = L^{-1}U^{-1}P \cdot F(\mathbf{x}^{(k)})$$

A quel punto vado ad aggiornare il valore di x:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \mathbf{h}$$

```
function x=multicorde(F,A,x1,m)
x=x1;
[L,U,P]=lu(A);
for k=1:m
    h=U\ (L\ (P*F(x)));
    x=x-h;
end
```

## INTERPOLAZIONE E APPROXIMAZIONE

### Polinomio di Interpolazione

- Scrivere una function  $X = \text{vandermonde}(x, n)$  che dato un vettore di nodi  $x = [x_1, x_2, \dots, x_m]$  e un grado massimo  $n$  costruisce

$$X = \begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^n \\ 1 & x_2 & x_2^2 & \dots & x_2^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_m & x_m^2 & \dots & x_m^n \end{bmatrix} \in \mathbb{R}^{m \times (n+1)}.$$

- Scrivere una function  $\text{alpha} = \text{interpol}(x, y)$  che, dati in input vettori  $x, y \in \mathbb{R}^{n+1}$ , restituisce i coefficienti del loro polinomio di interpolazione. Per testarla, costruire un esempio con soluzione  $\phi(x) = x^2 + 2x + 3$ .
- Scrivere uno script che calcola i coefficienti del polinomio di interpolazione a  $f(x) = \sin(x)$  nei nodi  $(0, \frac{\pi}{2}, \pi)$ , e, presa una griglia  $z=0:0.01:\pi$ , calcola  $\max_i |\sin(z_i) - \phi(z_i)|$  e traccia un grafico delle due funzioni ( $\text{plot}(z, fz, z, phiz)$ ). Per calcolare le due funzioni, potete usare i comandi  $fz = \sin(pi*z)$  e  $phiz = \text{polyval}(\text{alpha}(\text{end}:1), z)$ . Come diminuiscono gli errori prendendo invece 5 o 10 nodi equispaziati in  $[0, \pi]$ ? Qual è la limitazione dell'errore prevista dalla teoria?
- Ripetere l'esperimento con la funzione  $f(x) = \frac{1}{25x^2+1}$  e 3, 5, 10 nodi in  $[-1, 1]$ . Per questa funzione le interpolanti polinomiali  $\phi(x)$  hanno un comportamento oscillatorio e non si avvicinano a  $f$ . Sulla stessa funzione, provare a risolvere anche il problema di approssimazione con  $m = 10$  punti e un polinomio di grado 4.

## SOLUZIONE:

1) La matrice di Vandermonde si genera facilmente con la seguente funzione:

```
function X = vandermonde(x,n)
m=length(x);
X = ones(m,n+1);
for i=1:m
    for j=1:n+1
        X(i,j)=x(i)^(j-1);
    end
end
```

2) La funzione che calcola i coefficienti dovrà includere la matrice precedentemente creata; bisogna controllare attentamente le dimensioni.

$$x = (x_0, x_1, \dots, x_m)$$

Dato che ci troviamo davanti a un problema di interpolazione avremo bisogno che  $n = m$ , per cui vado a inizializzare  $n$  (N.B.  $x$  ha dimensione  $m+1$ )

Successivamente creo la matrice e risolvo il sistema con la funzione di backslash:

```
function alpha = interpola(x,y)
n=length(x)-1;
X=vandermonde(x,n);
alpha=X\y;
```

Per testare sulla funzione data, basta che mi calcoli  $n+1$  punti sostituendo valori a piacere, ad esempio 0, 1, 2 e successivamente vado a utilizzare la funzione (dovrà restituirmi  $\alpha = [3; 2; 1]$ )

```
>> interpola([0;1;2], [3;6;11])
```

```
ans =
```

```
3
2
1
```

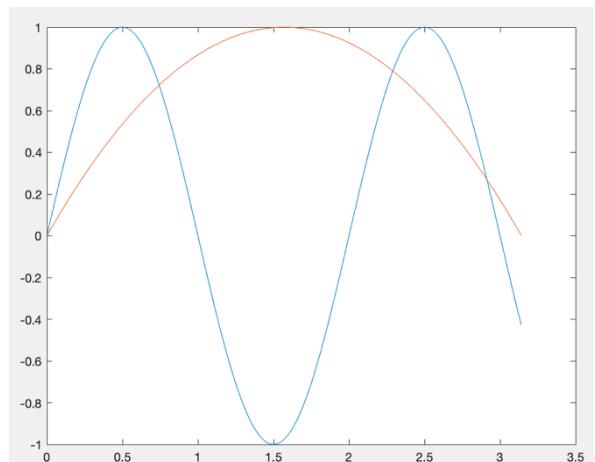
3) Inizializzo la funzione  $f = \sin(x)$  per poi andare a scrivere il vettore  $y$  come valore di  $f$  calcolato nei nodi. Una volta che ho  $y$  mi basta di usare la funzione  $\alpha = \text{interpola}(x, y)$  per calcolare i coefficienti del polinomio.

N.B: Il polinomio ha grado uguale al numero di nodi – 1.

```
f = @(x) sin(x);
x=[0;pi/2;pi];
y=f(x);
alpha=interpola(x,y);
z=0:0.01:pi;
l=length(z);
err=zeros(l,1);
phi = @(x) alpha(1) + alpha(2)*x + alpha(3)*x^2;
for i=1:l
    err(i)=abs(sin(z(i))-phi(z(i)));
end
Max_err=max(err)
fz=sin(pi*z);
phiz=polyval(alpha(end:-1:1),z);
plot(z,fz,z,phiz)
>> interpolazione_seno
```

```
Max_err =
```

```
0.0560
```



Proviamo a prendere ora 5 nodi spaziati equamente, ci basta scrivere il programma di prima cambiando solo il grado di alpha:

```
err =
```

**0.0018**

Con 10 nodi invece si ottiene:

```
err =
```

**3.0066e-07**

La limitazione prevista dalla teoria segue la seguente formula di errore massimo:

$$|f(x) - \phi(x)| \leq \frac{C_{n+1}}{(n+1)!} (b-a)^{n+1} = \frac{1}{(n+1)!} \pi^{n+1}$$

$$C_{n+1} = \max_{x \in [a,b]} |f^{(n+1)}(x)|$$

Per cui con  $C = 1$ , si ha che ad esempio per 5 nodi il metodo ha un errore massimo di:

```
>> 1/factorial(5)*pi^(5+1)
ans =
8.0116
```

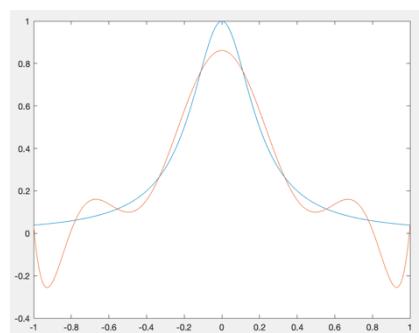
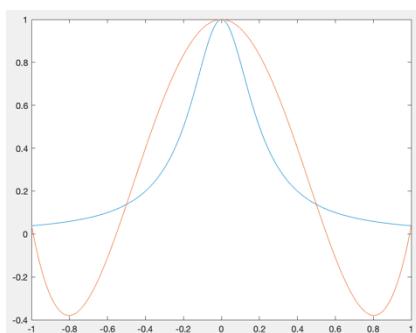
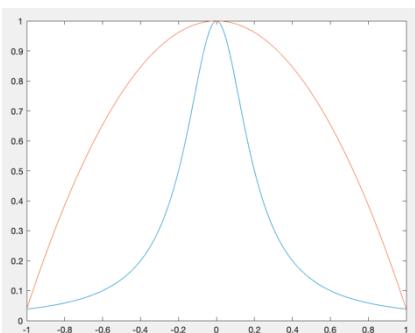
### COMANDO Linspace

Crea un vettore riga spaziando un intervallo, ad esempio:

```
>> linspace(-1,1,5)
ans =
-1.0000    -0.5000         0    0.5000    1.0000
```

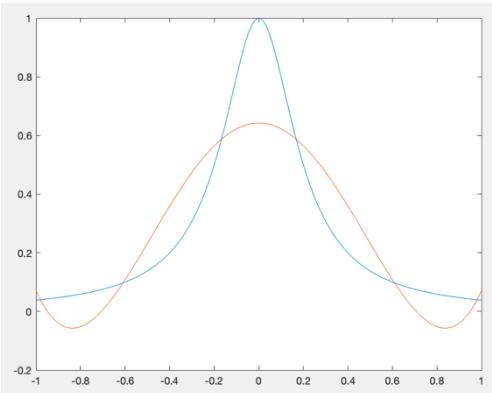
4) Scriviamo il programma analogo a prima, con 3, 5 e 10 nodi osservando i grafici:

```
f = @(x) 1/(25*(x)^2+1);
x = (linspace(-1,1,3))';
fx=x;
for k=1:length(x)
    fx(k)=f(x(k));
end
alpha = interp1(x,fx);
z=-1:0.01:1;
fz=z;
for k=1:length(z)
    fz(k)=f(z(1,k));
end
phiz=polyval(alpha(end:-1:1),z);
plot(z,fz,z,phiz)
```



Effettivamente si nota che il polinomio di interpolazione non si avvicina alla funzione ma ci oscilla intorno, proviamo quindi con un polinomio di approssimazione di grado  $n = 4$ , creando la funzione approssima(x, y, n):

```
function alpha = approssima(x,y,n)
X = vandermonde(x,n);
alpha = X\y;
```



```
f = @(x) 1/(25*(x)^2+1);
x = (linspace(-1,1,10))';
fx=x;
for k=1:length(x)
    fx(k)=f(x(k));
end
alpha = approssima(x,fx,4);
z=-1:0.01:1;
fz=z;
for k=1:length(z)
    fz(k)=f(z(1,k));
end
phiz=polyval(alpha(end:-1:1),z);
plot(z,fz,z,phiz)
```

Osserviamo che con un polinomio di grado 4, la funzione è approssimata meglio che con un polinomio di interpolazione

## INTEGRAZIONE NUMERICA

### Metodo dei Trapezi e del Punto Medio

- Scrivere `function IT = trapezi(f, a, b, N)` e `function IM = puntomedio(f, a, b, N)` che implementano il metodo del punto medio e dei trapezi

$$I_{T,N} = \frac{b-a}{N} \sum_{i=0}^{N-1} \frac{f(x_i) + f(x_{i+1})}{2}, \quad I_{M,N} = \frac{b-a}{N} \sum_{i=0}^{N-1} f\left(\frac{x_i + x_{i+1}}{2}\right).$$

Per il primo, usare  $\frac{1}{2}f(x_0) + f(x_1) + f(x_2) + \dots + f(x_{N-1}) + \frac{1}{2}f(x_N)$  in modo da effettuare solo  $N+1$  valutazioni di funzione.

- Dopo aver scritto ognuna delle due funzioni, testarla su  $N = 10$  punti e  $\int_0^1 (x+1) dx = \frac{3}{2}$ ,  $\int_1^3 x^2 dx = \frac{26}{3}$ ,  $\int_0^{\pi/2} \cos(x) dx = 1$ . Quale dovrebbe essere l'errore sul primo esempio? Verificare che  $E_{20}/E_{10} \approx 1/4$  negli altri due.
- Controllare le stime dell'errore: è vero che esiste  $\xi \in (a, b)$  tale che  $I_{T,N} - T = \frac{1}{24} f''(\xi) \dots ?$
- Scrivere una function `[IN, I2N, E2N] = trapezi2(f, a, b, N)` che calcola le approssimazioni  $I_{T,N}$  e  $I_{T,2N}$  e la stima dell'errore  $E_{2N} = \frac{1}{3} |I_{T,N} - I_{T,2N}|$ . Scrivere la funzione senza ricalcolare la funzione più volte negli stessi punti, in modo che richieda solo  $2N+1$  valutazioni di funzione. Quanto differisce  $E_{2N}$  dall'errore vero sui nostri esempi?

1) È sufficiente scrivere  $x$  come vettore di  $N+1$  valori equispaziati dell'intervallo  $[a, b]$ , chiamiamo  $h$  la spaziatura:  $\frac{b-a}{N}$ , il vettore  $x$  si scrive come:

$$x = [a, a + h, a + 2h, \dots, a + (N - 1)h, b]$$

Per applicare la formula dei trapezi in forma composita ci basta quindi usare un accumulatore:

```
function IT = trapezi(f,a,b,N)
x = linspace(a,b,N+1)';
sum = 0;
for k=1:N
    sum = sum + (f(x(k))+f(x(k+1)))/2;
end
IT = ((b-a)*sum)/N;
```

Stessa cosa si può fare con il metodo del punto medio:

```
function IM = puntomedio(f,a,b,N)
x = linspace(a,b,N+1)';
sum = 0;
for k=1:N
    sum = sum + f((x(k)+x(k+1))/2);
end
IM = ((b-a)*sum)/N;
```

2) Testiamo i programmi sulle funzioni date:

```
>> trapezi(@(x) x+1, 0,1,10)      >> trapezi(@(x) x^2, 1,3,10)      >> trapezi(@(x) cos(x),0,pi/2,10)
ans =
1.5000
ans =
8.6800
ans =
0.9979

>> puntomedio(@(x) x+1, 0,1,10)  >> puntomedio(@(x) x^2, 1,3,10)  >> puntomedio(@(x) cos(x),0,pi/2,10)
ans =
1.5000
ans =
8.6600
ans =
1.0010
```

Verifichiamo solo che  $E10/E20 = \frac{1}{4}$  nei secondi due esempi (solo per il secondo):

```
>> (trapezi(@(x) x^2, 1,3,20)-26/3)/(trapezi(@(x) x^2, 1,3,10)-26/3)
```

```
ans =
```

```
0.2500
```

3) Scriviamo la funzione alternativa trapezi2 usando un accumulatore somma che valuta prima i valori di  $I_{IN}$  e poi quelli di  $I_{2N}$ , infatti in questo caso  $h$  è la metà rispetto a prima, perciò il vettore  $x$  è lungo  $2N+1$ .

1. Valuto la funzione nei nodi con indice dispari ( $2i + 1$ ) e calcolo  $I_{IN}$  con tali valori (nota che sono quelli che avrei con  $h_{IN} = 2h_{I2N}$ )

2. Calcolo i valori pari, ossia quelli relativa alla spaziatura  $h$  da cui posso calcolare  $I_{2N}$

```
function [IN,I2N, E2N] = trapezio2(f,a,b,N)
h = (b-a)/(2*N);
x = a:h:b;
S = f(a)/2+f(b)/2;
for i = 1:N-1
    S = S + f(x(2*i+1));
end
IN = 2*h*S;
for i = 1:N
    S = S + f(x(2*i));
end
I2N = h*S;
E2N = 1/3 * abs(IN-I2N);
```

### **Metodo di Cavalieri – Simpson**

In realtà non serve rifare tutto il programma poiché possiamo servirci della formula di Cavalieri – Simpson composita:

$$I_{2N} = \frac{2}{3} I_{M,N} + \frac{1}{3} I_{T,N}$$

Da cui si ottiene che:

```
function IN = cavalieri_simpson(f,a,b,N)
IM=puntomedio(f,a,b,N);
IT=trapezi(f,a,b,N);
IN = (2/3 * IM) + (1/3 * IT);
```

Possiamo ad esempio confrontare l'errore di questo con l'errore degli altri due metodi per il secondo esempio e per il terzo:

EN_puntomedio =	EN_puntomedio =
0.0067	0.0010
EN_trapezi =	EN_trapezi =
0.0133	0.0021
EN_cavalieri_simpson =	EN_cavalieri_simpson =
0	2.1155e-07

# EQUAZIONI DIFFERENZIALI

## *Metodo di Eulero Esplicito*

- Vogliamo risolvere il problema ai valori iniziali

$$\frac{d}{dt} \begin{bmatrix} y^{(1)}(t) \\ y^{(2)}(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} y^{(1)}(t) \\ y^{(2)}(t) \end{bmatrix}, \quad \begin{bmatrix} y^{(1)}(0) \\ y^{(2)}(0) \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad (1)$$

che ha soluzione (controllare!)  $y^{(1)}(t) = \cos t$ ,  $y^{(2)}(t) = -\sin t$ .

- Scrivere una function  $[t, Y] = \text{euleroesplicito}(f, a, b, y_0, N)$  che approssima la soluzione di un problema ai valori iniziali con il metodo di Eulero esplicito, restituendo  $t = [t_0, t_1, \dots, t_N] \in \mathbb{R}^{1 \times (N+1)}$  e  $Y = [y_0, y_1, \dots, y_N] \in \mathbb{R}^{m \times (N+1)}$ .
- Tracciare un grafico della soluzione calcolata di (1) (sull'intervallo  $[0, 2*\pi]$  e con  $N \in \{100, 200, 400\}$ ) e della soluzione esatta con il comando  $\text{plot}(t, Y, t, [\cos(t); -\sin(t)])$ .
- Calcolare l'errore globale  $E_N = \max_{n=1,2,\dots,N} \|y_n - y(t_n)\|_\infty$  tra la soluzione numerica calcolata dal metodo e quella esatta (suggerimento: usare le funzioni `max`, `abs`, `sin`, `cos` di Matlab su vettori/matrici).
- Verificare che raddoppiando  $N$  il valore di  $E_N$  si riduce di un fattore circa 2.
- Cosa succede con un intervallo  $[a, b]$  più ampio, ad es.  $[0, 20]$  o  $[0, 100]$ ?
- Risolvere il problema *stiff*  $y' = Ay$ ,  $A = \begin{bmatrix} -10 & -10 \\ -10 & -11 \end{bmatrix}$  con diversi valori di  $N$ . Cosa succede quando  $N$  diventa troppo piccolo, per esempio  $N = 50$ ?

Prima di scrivere il programma, ricordiamo che il metodo di Eulero esplicito è un metodo a un passo che calcola una serie di punti (o vettori) che approssimano la soluzione del problema di Cauchy e si risolve mediante la seguente formula:

$$y_{n+1} = y_n + h f_n, \quad n = 0, 1, 2, \dots, N-1,$$

Definiamo perciò tutto ciò che ci occorre per eseguire il metodo:

$$\begin{aligned} t &= (t_0, t_1, t_2, \dots, t_N) \quad h = \frac{b-a}{N} \\ t_0 &= a; \quad t_N = b; \quad y_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}; \quad f(t, y) = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} * y \\ f(t, y) &= \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} * y \\ Y &= [y_0, y_1, \dots, y_N] \end{aligned}$$

Partiamo quindi da  $n = 1$ , vogliamo quindi ricavare  $y_1$

$$y_1 = y_0 + h \cdot f(t_0, y_0)$$

$$y_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} + h \cdot \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

Abbiamo quindi aggiornato la seconda componente del vettore  $\mathbf{Y}$ :

$$\mathbf{Y} = \left[ \begin{matrix} 1 \\ 0 \end{matrix} , \left[ \begin{matrix} 1 \\ -1 \end{matrix} , 0, 0, 0, \dots \right] \right]$$

Avremo perciò che per  $n = k+1$ :

$$y_k = y_k + h \cdot f(t_k, y_k)$$

**1)** A questo punto possiamo scrivere il programma:

```
function [t,Y] = euleroesplicito(f,a,b,y0,N)
m = length(y0);
t = zeros(1,N+1);
Y = zeros(m,N+1);
Y(:,1) = y0;
t(1) = a;
h = (b-a)/N;

for n = 1:N
    t(n+1) = t(n) + h;
    Y(:,n+1) = Y(:,n) + h * f(t(n),Y(:,n));
end
```

**2)** Come prima cosa dobbiamo definire  $f$ :

$A =$

$$\begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$$

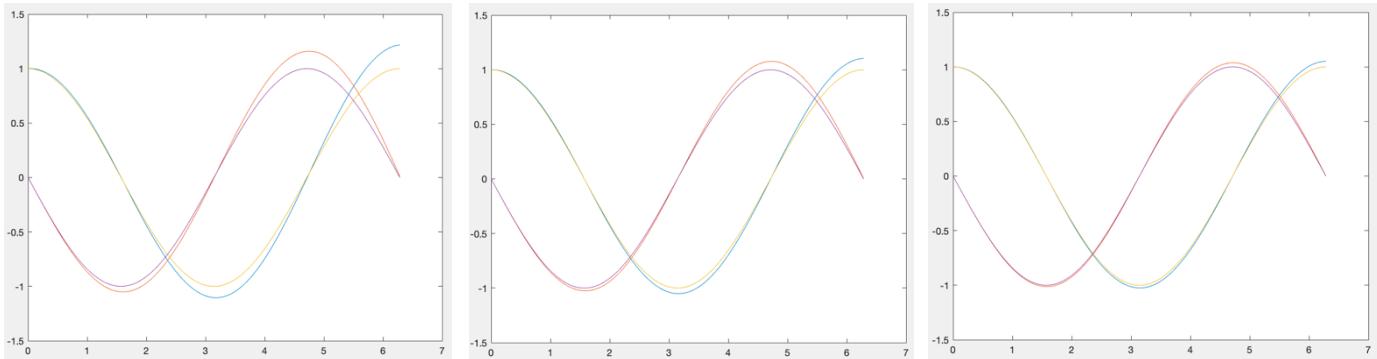
$\gg f = @(t,y) A*y$

$f =$

```
function handle with value:
@(t,y)A*y
```

Successivamente eseguiamo i comandi richiesti per  $N = 100, 200, 400$

$\gg [t,Y] = euleroesplicito(f,0,2*pi,[1;0],100)$   
 $\text{plot}(t,Y,t,[\cos(t);-\sin(t)])$



**3)** Calcoliamo l'errore massimo per i valori di  $N = 100, 200$  e verifichiamo che l'errore dimezza:

```
>> N=100;
[t,Y] = euleroesplicito(f,0,2*pi,y0,100);
E = zeros(1,N+1);
for n = 1:N+1
E(n)= norm(([cos(t(n));(-sin(t(n)))] - Y(:,n)),inf);
end
EN = max(E)
EN =
0.2177

>> N=200;
[t,Y] = euleroesplicito(f,0,2*pi,y0,N);
E = zeros(1,N+1);
for n = 1:N+1
E(n)= norm(([cos(t(n));(-sin(t(n)))] - Y(:,n)),inf);
end
EN = max(E)
EN =
0.1037

>> E200/E100
ans =
0.4762
```

4) Per un intervallo  $[0, 100]$  l'errore massimo è enorme ( $10^{15}$ )

5) Risolviamo con Eulero esplicito, sostituendo la matrice  $[-10, -10; -10, -11]$  e osserviamo graficamente che per  $N = 50$  la soluzione compie oscillazioni molto grandi; questo perché per un problema stiff i metodi espliciti non convergono mai.

## Metodo di Eulero Implicito

- Vogliamo risolvere con il metodo di Eulero *implicito* il problema lineare

$$y' = Ay, \quad y(a) = y_0 \in \mathbb{R}^m, \quad A \in \mathbb{R}^{m \times m}. \quad (2)$$

Trovare un'espressione per scrivere esplicitamente  $y_{n+1}$  in funzione di  $y_n$ .

- Scrivere una function  $[t, Y] = \text{euleroimplicito}(A, a, b, y_0, N)$  che utilizza il metodo di Eulero implicito per risolvere (2). Utilizzare \ (backslash) per risolvere il sistema lineare necessario per calcolare  $y_{n+1}$ .
- Testare la funzione sul problema (1), ponendo  $A = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$ .
- Come per il metodo di Eulero esplicito, tracciare un grafico della soluzione calcolata e di quella esatta; calcolare l'errore globale  $E_N$  e verificare che decresce come previsto per un metodo di ordine  $p = 1$ .
- Risolvere il problema stiff  $y' = Ay$ ,  $A = \begin{bmatrix} -10 & -10 \\ -10 & -11 \end{bmatrix}$  con diversi valori di  $N$ .

1) Per trovare un'espressione basta scrivere l'equazione del metodo e manipolare la funzione  $f = Ay$

$$y_{n+1} = y_n + f(t_{n+1}, y_{n+1})$$

$$f = A \cdot y_{n+1}$$

$$y_{n+1} = y_n + A \cdot y_{n+1}$$

$$y_{n+1} \cdot (I - A) = y_n$$

$$y_{n+1} = (I - A)^{-1} \cdot y_n$$

2) Scriviamo a questo punto la funzione:

```
function [t,Y,EN] = euleroimplicito(A,a,b,y0,N)
h = (b-a)/N;
m = length(y0);
t = zeros(1,N+1);
t(1)=a;
Y = zeros(m,N+1);
Y(:,1)=y0;
I = eye(m);
for n = 1:N
    t(n+1)=t(n)+h;
    Y(:,n+1)=(I-h*A)\Y(:,n);
end
plot(t,Y,t,[cos(t);(-sin(t))])
EN = max(max(abs(Y-[cos(t);-sin(t)])));
```

3) Per verificare che il metodo abbia ordine di convergenza  $p=1$  ci basta verificare che il massimo errore sia un  $O(h^p)$ , per cui basta graficare gli errori massimi e vedere che diminuiscono almeno linearmente

4) Essendo il metodo convergente per qualsiasi problema (quindi anche per lo stiff) l'errore viene molto meno di prima e infatti non si hanno grosse oscillazioni.

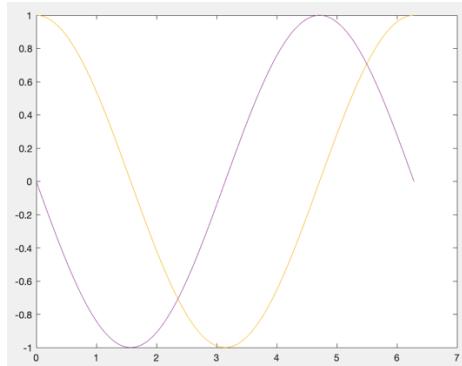
$$EN_{max} = 1.1153$$

### Metodo dei Trapezi

Scrivere una function  $[T, Y] = \text{trapezidiff}(A, a, b, y0, N)$  che utilizza il metodo dei trapezi per risolvere (2). Calcolare l'errore globale  $E_N$  ottenuto su (1) e verificare che la convergenza del metodo è di ordine  $p = 2$ .

```
function [t,Y,EN] = trapezidiff(A,a,b,y0,N)
h = (b-a)/N;
m = length(y0);
t = zeros(1,N+1);
t(1)=a;
Y = zeros(m,N+1);
Y(:,1)=y0;
I = eye(m);
for n = 1:N
    t(n+1)=t(n)+h;
    Y(:,n+1)=(I-h/2*A)\((I+h/2*A)*Y(:,n));
end
plot(t,Y,t,[cos(t);(-sin(t))])
EN = max(max(abs(Y-[cos(t);-sin(t)])));

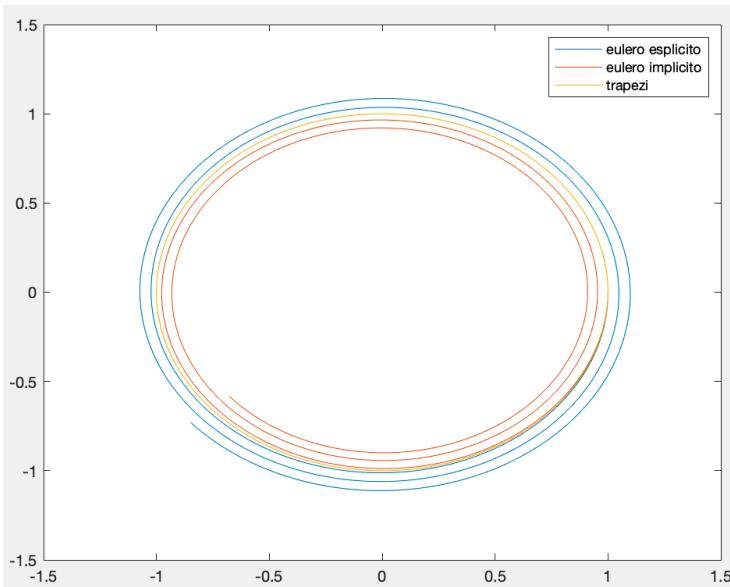
```



L'errore è molto basso, ossia  $E100 = 0.0021$

- Il problema (1) simula una particella in moto circolare uniforme sul piano  $(y_1, y_2)$ . Per i tre metodi, tracciare, con  $\text{plot}(Y(1, :), Y(2, :))$ , un grafico dell'orbita della particella. Tracciare anche un grafico dell'energia  $(y^{(1)}(t))^2 + (y^{(2)}(t))^2$  rispetto al tempo. Cosa osserviamo?

```
>> [t,YE]=euleroesplicito(@(t,y) A*y,0,15,[1;0],1000);
[t,YI]=euleroimplicito(A,0,15,[1;0],1000);
[t,YT]=trapezidiff(A,0,15,[1;0],1000);
plot(YE(1,:),YE(2,:),YI(1,:),YI(2,:),YT(1,:),YT(2,:))
legend('eulero esplicito','eulero implicito','trapezi')
```



# METODI A PIÙ PASSI

## Metodo Implicito BDF 2

Vogliamo risolvere equazioni scalari ( $m = 1$ ) con BDF(2), il metodo a due passi

$$p_1(z) = z^2 - \frac{4}{3}z + \frac{1}{3}, \quad p_2(z) = \frac{2}{3}z^2.$$

- Scrivere (con carta e penna) l'equazione  $g(y) = 0$  da risolvere per calcolare  $y = y_{n+2}$  dati  $y_{n+1}, y_n, t_{n+2}$ .
- Scrivere  $f = @(t, y) \dots$  e  $df = @(t, y) \dots$  che calcolano  $f$  e la sua derivata (Jacobiano)  $\frac{\partial f}{\partial y}(t, y)$  per il problema  $y' = -2ty^2$ ,  $y(a) = 1$ ,  $[a, b] = [0, 1]$ .
- Scrivere  $g = @(y) \dots$  e  $dg = @(y) \dots$  che calcolano  $g$  in un punto  $y$  e la sua derivata, utilizzando  $f$ ,  $df$ ,  $Y(n+1)$ ,  $Y(n)$ ,  $t(n+2)$ .
- Scrivere una function  $[t, Y] = bdf2(N)$  che risolve il problema dato sopra: calcola il primo valore incognito  $Y(2)$  con il metodo di Eulero esplicito, e ad ogni passo  $n$  fa 5 iterazioni del metodo di Newton  $y^{(k+1)} = y^{(k)} - g(y^{(k)})/dg(y^{(k)})$  per trovare il valore  $y_{n+2}$  che risolve  $g(y_{n+2}) = 0$ . Qual è un buon valore iniziale?  
*Suggerimento:* per controllare la corretta convergenza (quadratica) del metodo di Newton, farsi scrivere sullo schermo  $g(y^{(k)})$  dopo ogni passo.
- Controllare che il metodo converge con ordine  $p = 2$ , confrontando la soluzione calcolata con quella esatta  $y(t) = \frac{1}{1+t^2}$  (Matlab:  
1 ./ (1+t.^2)).

1) Da  $p_1(z)$  mi ricavo i coefficienti alpha e da  $p_2(z)$  ricavo invece i coefficienti beta:

$$\alpha_0 = \frac{1}{3}, \quad \alpha_1 = -\frac{4}{3}, \quad \alpha_2 = 1$$

$$\beta_0 = 0, \quad \beta_1 = 0, \quad \beta_2 = \frac{2}{3}$$

Da cui ho la mia espressione del metodo a più passi:

$$y_{n+2} - \frac{4}{3} \cdot y_{n+1} + \frac{1}{3} \cdot y_n = h \cdot \frac{2}{3} \cdot f_{n+2}$$

Imponendo  $g = y_{n+2}$  posso scrivere:

$$y - \frac{4}{3} \cdot y_{n+1} + \frac{1}{3} \cdot y_n = h \cdot \frac{2}{3} \cdot f(t_{n+2}, y)$$

$$g = -y + \frac{4}{3} \cdot y_{n+1} - \frac{1}{3} \cdot y_n + h \cdot \frac{2}{3} \cdot f(t_{n+2}, y) = 0$$

2)  $y' = -2 \cdot t \cdot y^2$ ,  $y(a) = 1$ ,  $t \in [a, b] = [0, 1]$

Calcoliamo quindi la derivata di  $f$  e scriviamo su Matlab le funzioni  $g$ ,  $dg$ ,  $f$ ,  $df$

$$f = -2ty^2$$

$$df = -4ty$$

$$g = -y + \frac{4}{3} \cdot y_{n+1} - \frac{1}{3} \cdot y_n + h \cdot \frac{2}{3} \cdot f(t_{n+2}, y)$$

$$dg = -1 + h \cdot \frac{2}{3} \cdot df(t_{n+2}, y)$$

$$f = @(t,y) -2*t*y^2;$$

$$df = @(t,y) -4*t*y;$$

$$g = @(y) 1/3*Y(n)-4/3*Y(n+1)+y-2/3*h*f(t(n+2),y);$$

$$dg = @(y) 1-2/3*h*df(t(n+2),y);$$

**3)** Il metodo richiesto prevede che ad ogni passo di n, ossia per ogni elemento di Y io vada a risolvere l'equazione  $g(y) = 0$  in modo da trovare y.

Per risolvere l'equazione è richiesto di utilizzare il metodo di Newton a 5 passi, per cui avrò che per ogni iterazione n:

- *Calcolo g e dg utilizzando i valori del passo precedente:*

```
g = @(y) 1/3*Y(n)-4/3*Y(n+1)+y-2/3*h*f(t(n+2),y);
dg = @(y) 1-2/3*h*df(t(n+2),y);
```

- *Impongo che y, sia uguale a Y(n+1) in modo da avere il passo zero del metodo di Newton*

- *Esegua 5 passi del metodo di newton*

La mia y trovata al termine dei 5 passi sarà proprio la Y(1, n+2)

```
function [t,Y] = bdf2(N)
Y = zeros(1,N+1);
Y(1) = 1;
h = 1/N;
t = 0:h:1;

f = @(t,y) -2*t*y^2;
df = @(t,y) -4*t*y;

Y(2) = Y(1) + h*f(t(1),Y(1));

for n=1:N-1
    g = @(y) 1/3*Y(n)-4/3*Y(n+1)+y-2/3*h*f(t(n+2),y);
    dg = @(y) 1-2/3*h*df(t(n+2),y);
    y = Y(n+1);
    for j = 1:5
        y = y - dg(y)\g(y);
    end
    Y(n+2) = y;
end
```

**4)** Verifichiamo infine l'ordine di convergenza attraverso gli errori E20 e E40:

```
>> esatta = @(t) 1./(1+t.^2);      >> esatta = @(t) 1./(1+t.^2);
[T,Y] = bdf2(20);                  [T,Y] = bdf2(40);
E20 = max(max(abs(Y-esatta(T))))  E40 = max(max(abs(Y-esatta(T)))))

E20 =
0.0037
E40 =
9.4273e-04

>> p = sqrt(E20/E40)

p =
1.9853
```

