Università degli studi di Cagliari

Facoltà di Ingegneria

Corso di Laurea Specialistica in Ingegneria per l'Ambiente e il Territorio

(Ord.509)

Seminario

Matlab-Simulink per l'Ingegneria

Relazione finale

A.A. 2012/2013

Docenti:

Prof. G.Rodriguez

Prof. E.Usai

Prof. A.Pisano

Allievo:

Antonio Dessena

INDICE

1. INTRODUZIONE A MATLAB

MATLAB è l'abbreviazione di *Matrix Laboratory*, nasce negli anni '80-'84, ed è un linguaggio di alto livello e un ambiente interattivo per il calcolo numerico e l'analisi statistica, l'analisi e la visualizzazione dei dati e la programmazione.

Matlab consente di creare e manipolare matrici, visualizzare funzioni e dati, sviluppare e implementare algoritmi, analizzare dati, creare modelli e applicazioni.

Il linguaggio, gli strumenti e le funzioni matematiche incorporate consentono di raggiungere una soluzione più velocemente rispetto all'uso di fogli di calcolo o di linguaggi di programmazione tradizionali, quali C, C++ o Java.

È possibile usare Matlab per diverse applicazioni, tra cui l'elaborazione di segnali e i sistemi di telecomunicazione, l'elaborazione di immagini e video, i sistemi di controllo, ecc..

1.1 Variabili e costanti speciali

La filosofia del programma è quella di richiamare librerie, come blas, linpack, lapack, all'interno delle quali vi sono i programmi e le funzioni per la risoluzione di operazioni tra scalari o tra matrici; quindi per la risoluzione delle operazioni matematiche non si creano blocchi o strutture di programmazione, ma si procede utilizzando semplicemente la sintassi matematica.

Il programma memorizza tutto quello che compare a video in un file di testo editabile e ricaricabile su Matlab.

`diary prova.txt` inizia la registrazione;

`diary off` chiude la registrazione;

`edit prova.txt` richiama il file di testo memorizzato e salvato.

`ans` variabile temporanea che contiene il risultato dell'operazione più recente;

`eps` epsilon, il più piccolo numero reale che addizionato ad 1 crea un numero maggiore di 1

$eps = 2.2204 \cdot e - 16 = 2.2204 \cdot e^{-16}$;

`realmin` il più piccolo numero reale che può essere utilizzato $\cong 2.2 \cdot e - 308$;

`realmax` il più grande numero reale che può essere utilizzato $\cong 2.2 \cdot e308$;

`i, j` unità immaginarie, indici rappresentativi di riga e colonna in una matrice;

`pi` indica il numero p-greco π .

I nomi delle variabili devono iniziare con un carattere alfabetico, e successivamente si possono utilizzare caratteri alfanumerici o underscores; la lunghezza del nome di una variabile non deve eccedere i 19 caratteri.

clear cancella le variabili dal foglio di lavoro, workspace, e rinizializza le variabili assegnate;

toolbox sono delle estensioni, applicazioni, librerie esterne utilizzate per implementare le funzionalità del programma.

Il programma può essere utilizzato come un interprete creando programmi per la risoluzione di calcoli, in questo caso i tempi di calcolo sono lunghi e non si utilizza il programma per la sua reale funzione e potenzialità.

Utilizzando la sintassi matematica, le formulazioni e le notazioni matriciali, il programma richiama le blas, i tempi di calcolo sono estremamente ridotti e si utilizza il programma nella sua reale connotazione e in maniera ottimale.

L'espressione matematica:

$$\sum_{i=1}^{n} a_i = S$$

Può essere espressa in Matlab come prodotto tra vettori:

$a = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix}$ vettore colonna;

$e = \begin{bmatrix} 1 \\ \vdots \\ \vdots \\ 1 \end{bmatrix}$ vettore colonna;

$a' = a^T$ vettore a trasposto, da vettore colonna diventa vettore riga; l'apice è rappresentativo della funzione di trasposto;

$$a' \cdot e = a^T \cdot e = [\, a_1, a_2, \dots, a_n] \cdot \begin{bmatrix} 1 \\ \vdots \\ \vdots \\ 1 \end{bmatrix} = S$$

Le variabili sono *case-sensitive* nel senso che:

$a \neq A$

a viene utilizzata per scalari o vettori;

A viene utilizzata per matrici;

who richiama le variabili usate, elenca le variabili presenti in memoria;

whos richiama le variabili usate, elenca le variabili presenti in memoria indicando anche le dimensioni, i byte occupati e il tipo di variabile;

↑ il tasto freccia richiama i comandi eseguiti in precedenza;

$a \uparrow, A \uparrow$ richiama le variabili specifiche.

Matlab ragiona con matrici, anche gli scalari sono visti come matrici definite da una matrice *1x1* una riga e una colonna.

$A = rand(m, n)$ crea una matrice di dimensione *(m,n)* di elementi casuali compresi tra [0,1]

m: definisce il numero di righe della matrice *A*;

n: definisce il numero delle colonne della matrice *A*;

$A = rand(m, n);$ il punto e virgola regola l'output, se non viene inserito compare la matrice a video, se invece viene inserito la matrice non compare a video, ma è presente in memoria, visualizzabile con le sue caratteristiche tramite il comando *whos*, chiaramente se la matrice è molto grande l'output a video richiede molto tempo.

1.2 Forme indeterminate

In analisi matematica esistono alcune forme indeterminate, impossibili da risolvere, come ad esempio il rapporto tra un numero e zero, Matlab segnala tali forme come errori e nel caso di calcoli matriciali procede ugualmente con i calcoli permettendo all'operatore di salvare i calcoli corretti e correggere gli errori che hanno determinato le forme indeterminate.

Inf infinito;

NaN Not a Number, risultato numerico indefinito.

1.3 Precisione singola e precisione doppia

I numeri possono essere memorizzati in precisione singola o in precisione doppia.

$d = single(89)$ 4 *bytes* 7 cifre decimali;

$d_1 = 89$ 8 *bytes* 16 cifre decimali;

La differenza tra i due numeri non è uguale a zero.

$d \neq d_1$

$d - d_1 \neq 0$

format long visualizza le cifre decimali;

format short riduce le cifre decimali;

$s = 'pippo'$ stringa;

$t = 'c'$ carattere.

La stringa e il carattere vengono entrambe memorizzate come *char*.

1.4 Numeri complessi

Matlab mette in evidenza i numeri complessi nella loro parte reale e immaginaria:

$z = sqrt(-1) = 0 + 1.0000i$

0 parte reale;

$1.0000i$ parte immaginaria;

I numeri complessi vengono riconosciuti immediatamente e occupano in memoria 16 *bytes*, attraverso il comando *whos* se ne evidenzia la loro natura complessa: *complex*.

abs(z) valore assoluto di *z*;

angle(z) fase di *z*;

conj(z) complesso coniugato di *z*;

real(z) restituisce la parte reale di *z*;

imag(z) restituisce la parte immaginaria di *z*;

$z = 2 + 3i$

$abs(z) = 3.6056$

$angle(z) = 0.9828$

$conj(z) = 2 - 3i$

$real(z) = 2$

$imag(z) = 3$

$z_1 = 1 + 1i$ $z_2 = 2 - 3i$

$z_1 \cdot z_2 = 5 - 1i$

$\frac{z_1}{z_2} = -0.0769 + 0.3846i$

1.5 Altre istruzioni

doc	richiama il manuale in linea;
help …	richiama un aiuto relativamente alla funzione digitata;
tic	fa partire il cronometro;
toc	legge e mostra a video il tempo trascorso tra lo start, cioè il *tic* e il comando *toc*.

1.6 Alcune funzioni

$[L\ U\ P] = lu\ (A)$	Algoritmo di Gauss, fattorizzazione *PA=LU*;
L	matrice triangolare inferiore;
U	matrice triangolare superiore;
P	matrice di permutazione.

1.7 Elementi di aritmetica di macchina

L'aritmetica dell'analisi è infinita, i numeri decimali possono estendersi all'infinito; l'aritmetica di macchina, cioè dei calcolatori, è finita, i numeri decimali che si possono registrare nei registri di memoria non sono infiniti, ma finiti.

$\frac{1}{3}, \sqrt{2}$: vengono memorizzati i numeri fino alla $16^{\wedge}$ cifra decimale, il resto viene arrotondato, o per difetto o per eccesso a seconda del caso.

Quindi tutti i numeri sono approssimati, questo implica che durante le operazioni tra numeri, siano essi scalari, vettori o matrici, si commette un errore tipico della macchina, la quale non può fisicamente memorizzare tutte le infinite cifre decimale.

Il massimo errore relativo di memorizzazione è pari a:

$$2 \cdot eps = 4.4409 \cdot e - 16$$

Un'altra considerazione va fatta relativamente alla trasformazione di numeri in base decimale, linguaggio matematico, in numeri in base binaria, linguaggio di macchina; tale trasformazione comporta sempre un arrotondamento e quindi un errore già in fase di memorizzazione:

$$(0{,}5)_{10} = (0{,}1)_2$$

$$(0{,}1)_{10} = (0{,}0001100110011)_2$$

$$\frac{1}{3} = (0{,}333\ \ldots\ \ldots)_{10} = (0{,}1)_3$$

Due numeri con 10 cifre decimali vengono entrambi memorizzati in maniera esatta, max 16 cifre decimali, ma il loro prodotto determina un errore, perché anziché avere 20 cifre decimali la

macchina restituisce sempre 16 cifre decimali che, come già detto, sono le uniche che può memorizzare.

$\left(\sqrt{2}\right) - 2 = 0$ matematicamente corretto;

$sqrt(2)^2 - 2 \neq 0 = 4.4409 \cdot e - 16$ linguaggio di macchina.

Tali considerazioni portano alla conclusione importante che nei calcolatori non vale *la proprietà associativa*:

$a + (b + c) \neq (a + b) + c$

1.8 Algoritmo

Un algoritmo è una sequenza univoca di un numero finito di operazioni elementari che stabilisce come calcolare la soluzione di un problema, assegnati certi dati iniziali.

sequenza univoca: non è sufficiente dare una formula, deve essere chiaro anche in che ordine bisogna eseguire le operazioni;

operazioni elementari: si tratta di una nozione relativa, devono essere elementari, cioè di semplice comprensione, per chi leggerà l'algoritmo;

numero finito: se l'algoritmo è iterativo deve essere fornito un criterio di arresto;

input/output: deve essere chiaro il numero ed il tipo dei dati richiesti dall'algoritmo e di quelli da esso generati.

Definiamo *stabile* un algoritmo nel quale la successione delle operazioni non amplifica eccessivamente gli errori presenti sui dati.

Definiamo *instabile* un algoritmo nel quale la successione delle operazioni amplifica eccessivamente gli errori presenti sui dati.

1.9 Variabile struttura

Sono del tipo:

$rec.nome = 'antonio'$

$rec.cognome = 'dessena'$

$rec.età = '33'$

$rec.mat = rand(3)$

In una variabile di tipo struttura possono essere inserite stringhe ma anche matrici.

1.10 Variabile cell array

Sono del tipo:

$qq = \{23, 'pippo', rand(3)\}$

doc cell richiama il manuale in linea relativamente alla variabile cell.

1.11 Operazioni elemento per elemento

+	somma	array+scalare	A+b
-	sottrazione	array-scalare	A-b
+	somma di array	A+B	
-	sottrazione di array		A-B
.*	moltiplicazione tra array		A.*B
.^	elevazione a potenza tra array		A.^B

Le operazioni elemento per elemento vengono svolte tra gli elementi che occupano posizioni omologhe.

Il prodotto standard *, che indica il prodotto scalare, è consentito solo tra vettori dimensionalmente coerenti, cioè un vettore riga per un vettore colonna.

1.12 Dichiarazione estensiva di vettori e matrici

Per una matrice si possono digitare gli elementi uno per volta separando gli elementi di ogni riga con uno spazio bianco o virgola e le righe con un punto e virgola, il tutto tra parentesi quadre.

$A = [1\ 2\ 3; 4\ 5\ 6; 7\ 8\ 9]$

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Le righe devono avere la stessa lunghezza, cioè lo stesso numero di elementi, altrimenti compare un messaggio di errore a video.

$v = [1; 2; 3]$ $v = 3x1$ 3 righe x 1 colonna

$$v = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$ vettore colonna

$w = [4\ 7\ 8]$ vettore riga

1.13 Operazioni tra matrici

$B = [2\ 3\ 1; 5\ 4\ 3; 7\ 6\ 5]$

$$B = \begin{bmatrix} 2 & 3 & 1 \\ 5 & 4 & 3 \\ 7 & 6 & 5 \end{bmatrix}$$

$$A + B = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} + \begin{bmatrix} 2 & 3 & 1 \\ 5 & 4 & 3 \\ 7 & 6 & 5 \end{bmatrix} = \begin{bmatrix} 3 & 5 & 4 \\ 9 & 9 & 9 \\ 14 & 14 & 14 \end{bmatrix}$$

$$3 \cdot A - 5 \cdot B = 3 \cdot \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} - 5 \cdot \begin{bmatrix} 2 & 3 & 1 \\ 5 & 4 & 3 \\ 7 & 6 & 5 \end{bmatrix} = \begin{bmatrix} -7 & -9 & 4 \\ -13 & -5 & 3 \\ -14 & -6 & 2 \end{bmatrix}$$

Per ogni operazione, con il linguaggio di programmazione C, avremo dovuto scrivere un programma di cicli *for*, in Matlab si utilizza direttamente la notazione matriciale.

I prodotti tra matrici non sono commutativi.

$A \cdot B$ prodotto righe x colonne

$A \cdot B \neq B \cdot A$

$$A \cdot B = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \cdot \begin{bmatrix} 2 & 3 & 1 \\ 5 & 4 & 3 \\ 7 & 6 & 5 \end{bmatrix} = \begin{bmatrix} 33 & 29 & 22 \\ 75 & 68 & 49 \\ 117 & 107 & 76 \end{bmatrix}$$

$$B \cdot A = \begin{bmatrix} 2 & 3 & 1 \\ 5 & 4 & 3 \\ 7 & 6 & 5 \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} = \begin{bmatrix} 21 & 27 & 33 \\ 42 & 54 & 66 \\ 66 & 84 & 102 \end{bmatrix}$$

$$A^2 = A \cdot A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} = \begin{bmatrix} 30 & 36 & 42 \\ 66 & 81 & 96 \\ 102 & 126 & 156 \end{bmatrix}$$

Il rapporto A/A in analisi matematica non ha senso, in Matlab tale operazione esegue una particolare funzione, che vedremo più avanti.

$A' = A^T = A^*$ matrice trasposta e coniugata di A;

$$A' = A^T = A^* = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$$

1.14 Matrici di elementi complessi

$Q = [2 \quad 3 + 2i; 3 - 4i \quad 1i]$

$$Q = \begin{bmatrix} 2 & 3 + 2i \\ 3 - 4i & i \end{bmatrix}$$

$Q' = \begin{bmatrix} 2 & 3 + 4i \\ 3 - 2i & -i \end{bmatrix}$ gli elementi vengono trasposti e cambiati di segno, coniugati.

I vettori e le matrici possono essere sommati, moltiplicati se hanno le stesse dimensioni, Matlab è coerente con le regole sulle operazioni tra vettori e matrici.

$v = [1; 2; 3]$ vettore colonna

$w = [4\ 7\ 8]$ vettore riga

$v + w =$ ERRORE vettore colonna + vettore riga = errore;

$v + w' = [5; 9; 11] = \begin{bmatrix} 5 \\ 9 \\ 11 \end{bmatrix}$ vettore colonna + vettore riga trasposto = vettore colonna;

$v' + w = [5\ 9\ 11]$ vettore colonna trasposto + vettore riga = vettore riga;

Il prodotto tra matrici si può svolgere se la dimensione delle righe di una matrice è uguale alla dimensione delle colonne dell'altra matrice:

$A = (m, n); \quad B = (n, q)$

$C = A \cdot B; \quad C = (m, q)$

$A + v =$ ERRORE matrice + vettore colonna trasposto = errore;

$A \cdot v = \begin{bmatrix} 14 \\ 32 \\ 50 \end{bmatrix}$ matrice x vettore colonna = vettore colonna;

$v \cdot A =$ ERRORE vettore colonna x matrice = errore;

$v' \cdot A = [30\ 36\ 42]$ vettore colonna trasposto x matrice = vettore riga;

$A \cdot v' =$ ERRORE matrice x vettore colonna trasposto = errore;

$A \cdot w =$ ERRORE matrice x vettore riga = errore;

$A \cdot w' = \begin{bmatrix} 42 \\ 99 \\ 156 \end{bmatrix}$ matrice x vettore riga trasposto = vettore colonna;

$w' \cdot A =$ ERRORE vettore riga trasposto x matrice = errore;

$w \cdot A = [88\ 107\ 126]$ vettore riga x matrice = vettore riga.

1.15 Dichiarazione intensiva di vettori e matrici

$s = [1 : 10]$ incremento unitario da 1 a 10;

$s = [1\ \ 2\ \ 3\ \ 4\ \ 5\ \ 6\ \ 7\ \ 8\ \ 9\ \ 10]$

$t = [2 : 2 : 10]$ [*start : step,passo : stop*] se si omette il numero centrale lo step è 1;

$due = [2 : 2 : 20]$ tabellina del due;

$tre = [3 : 3 : 30]$ tabellina del tre;

$quattro = [4 : 4 : 40]$ tabellina del quattro;

$v_n = [100 : 100 : 1000]'$ vai da 100 a 1000 con passo 100 e trasponi;

$$v_n = \begin{bmatrix} 100 \\ 200 \\ 300 \\ 400 \\ 500 \\ 600 \\ 700 \\ 800 \\ 900 \\ 1000 \end{bmatrix}$$

$sec = [0 : 0.1 : 2]'$ vai da 0 a 2 con passo 0.1 e trasponi;

$$v_n = \begin{bmatrix} 0.1 \\ 0.2 \\ \vdots \\ 2 \end{bmatrix}$$

1.16 Grafico di sinx tra $-\pi$ e $+\pi$

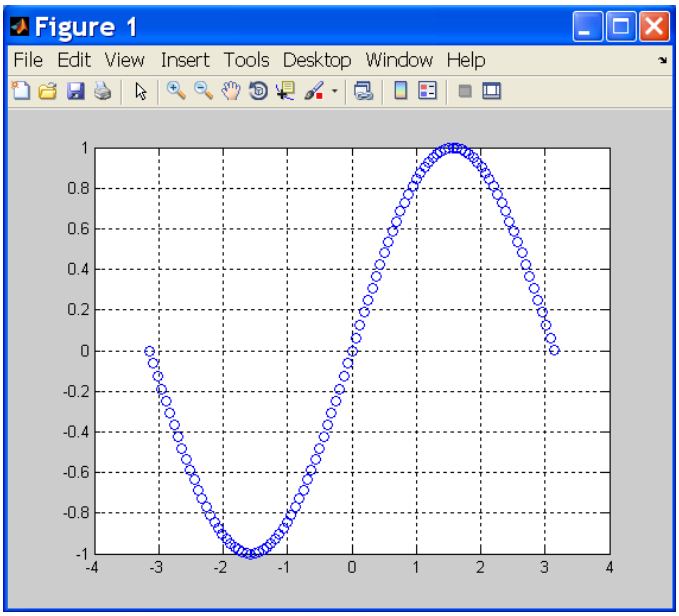
Il programma campiona la funzione e poi unisce i punti del dominio discreto; se non voglio visualizzare il dominio discreto inserisco il punto e virgola alla fine dell'istruzione:

```
x=[-pi:2*pi/100:pi]';
y=sin(x);
plot(x,y,'o'),grid
```

[x y] in output restituisce la coppie di valori *(x,y)*, che è una matrice [200x2];

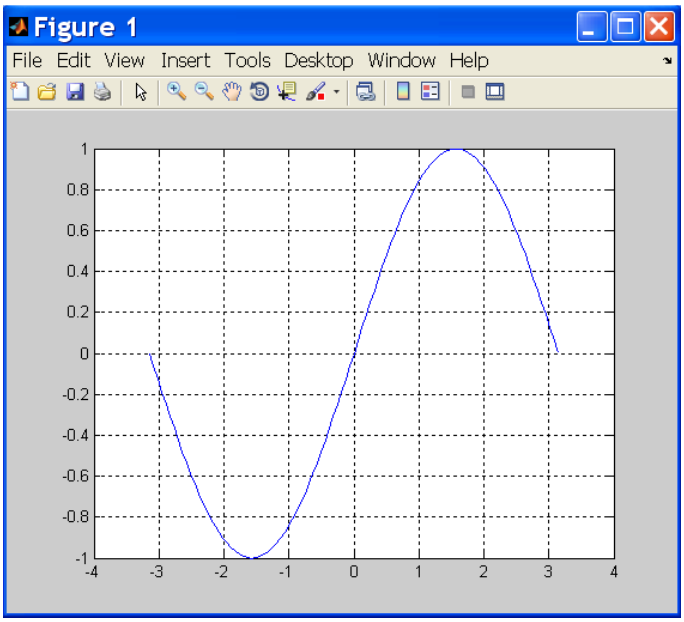
La funzione `plot` restituisce il grafico di *sin(x)* i cui valori sono definiti tramite pallini.

La funzione `grid` definisce una griglia all'interno del grafico.




```
x=[-pi:2*pi/100:pi]';
y=sin(x);
plot(x,y,'-'),grid
```

La funzione `plot` restituisce il grafico di *sin(x)* i cui valori sono definiti tramite una linea continua.



Nella parte dedicata alla grafica, vedremo nel dettaglio i vari tipi di linee utilizzabili.

1.17 Funzioni che creano matrici

rand(n) crea una matrice quadrata $(n \cdot n)$ di elementi casuali compresi tra [0,1];

rand(m,n) crea una matrice rettangolare $(m \cdot n)$ di elementi casuali compresi tra [0,1] ;

size(A) estrae la dimensione della matrice *A*;

zeros(n) crea una matrice identità $(n \cdot n)$ i cui elementi sono pari a 0;

zeros(m,n) crea una matrice $(m \cdot n)$ i cui elementi sono pari a 0;

zeros(size(A)) crea una matrice di elementi pari a 0 avente la stessa dimensione della matrice *A*;

ones(n) crea una matrice $(n \cdot n)$ i cui elementi sono pari a 1;

ones(m,n) crea una matrice $(m \cdot n)$ i cui elementi sono pari a 1;

ones(size(A)) crea una matrice di elementi pari a 1 avente la stessa dimensione della matrice *A*;

eye(n) crea una matrice identità $(n \cdot n)$;

eye(size(A)) crea una matrice identità con la stessa dimensione della matrice *A*;

length(b) estrae la lunghezza del vettore riga o colonna *v*, *w*.

Nelle applicazioni reali le matrici quadrate sono rare, di solito si ha a che fare con matrici rettangolari.

$D = rand(5)$ matrice 5x5;

$E = rand(10,4)$ matrice 10x4;

$size(D)$ 5 5;

$size(E)$ 10 4;

$F = zeros(5)$ matrice di zeri 5x5;

$G = zeros(10,4)$ matrice di zeri 10x4;

$size(F)$ 5 5;

$size(G)$ 10 4;

$zeros(size(A))$ matrice di zeri 3x3;

$H = ones(5)$ matrice di 1 5x5;

$I = ones(10,4)$ matrice di 1 10x4;

$M = magic(5)$ matrice-quadrato magico, la somma delle righe e delle colonne è sempre costante;

$$M = \begin{bmatrix} 17 & 24 & 1 & 8 & 15 \\ 23 & 5 & 7 & 14 & 16 \\ 4 & 6 & 13 & 20 & 22 \\ 10 & 12 & 19 & 21 & 3 \\ 11 & 18 & 25 & 2 & 9 \end{bmatrix}$$

$v = ones(5,1)$ vettore colonna 5x1;

$M \cdot v$ vettore colonna = 65;

$M' \cdot v$ vettore colonna = 65;

$diag(M)$ estrae la diagonale di *M*;

$sum(diag(M))$ somma gli elementi della diagonale di *M* = 65;

$rot90(M)$ ruota la matrice *M* di 90° in senso antiorario;

$diag(rot90(M))$ estrae la diagonale della matrice *M* ruotata di 90° in senso antiorario;

$sum(diag(rot90(M)))$ somma gli elementi dell'antidiagonale = 65

$lam = eig(M)$ calcola gli autovalori lambda della matrice *M*;

$$lam = \begin{bmatrix} 65.0000 \\ -21.2768 \\ -13.1263 \\ 21.2768 \\ 13.1263 \end{bmatrix}$$

2. PROGRAMMAZIONE IN MATLAB

Finora abbiamo utilizzato Matlab come un interprete, cioè tramite un utilizzo "*da prompt*" del programma, le varie operazioni sono state eseguite direttamente nel *workspace* ossia nello spazio di lavoro adibito alla digitazione.

Risulta comodo e conveniente definire dei files con estensione *.m* (i cosiddetti *m-files*) che contengono una sequenza di istruzioni Matlab che, all'atto dell'esecuzione di tali files, vengono eseguite in sequenza, come se le espressioni contenute all'interno di tali files vengano via via digitate ed eseguite nel workspace del programma.

L'inserimento dal workspace o l'esecuzione da files esterni sono difatto operazioni equivalenti.

Tali files si definiscono:

- *Script*
- *Function*

La differenza tra *script* e *function* sta nella modalità di esecuzione, in entrambi i casi si parte da un foglio di *editor* che si apre dal programma, tale editor ha la memoria vuota e si comunica con esso tramite comandi di input e di output.

Ad esempio:

$[L\ U\ P] = lu\ (A)$ Algoritmo di Gauss, fattorizzazione *PA=LU*;

La *function* per partire ha bisogno del dato di input *A*, quando la function termina restituisce in output il risultato, cioè le tre matrici.

L'utilizzo di script e function non determina *effetti collaterali*, non disturbano il programma, vengono eseguiti molto velocemente e snelliscono il lavoro sviluppato nel workspace.

Svolgiamo un esempio per definire la creazione e lo sviluppo di script e function.

Script

Es: creare uno script in cui si deve fissare un intero *n* e una matrice del tipo:

$M = \begin{bmatrix} A & C \\ O & B \end{bmatrix}$ *M*: matrice mosaico, formata da 4 matrici;

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

$$B = \begin{bmatrix} 4 & 2 & 0 & & 0 \\ 2 & & & & \\ 0 & & & & 0 \\ & & & & 2 \\ 0 & & 0 & 2 & 4 \end{bmatrix}$$

$C = rand$

$O = zeros$

Tutto l’esercizio può essere sviluppato tramite cicli *for*, ma risulta estremamente lungo e laborioso.

Apriamo un foglio di editor, lo nominiamo ad es. *esempi* e lo salviamo, la sua estensione sarà *.m*, abbiamo creato a tutti gli effetti lo script: *esempi.m* che risulta al momento essere vuoto.

Per ottenere la matrice mosaico *M* compiliamo lo script:

```
n=input('Dammi il valore di n: ')
A=diag(3*ones(n,1))+triu(ones(n),1)
B=diag(4*ones(n,1))+diag(2*ones(n-1,1),1)+diag(2*ones(n-1,1),-1)
C=rand(n)
O=zeros(n)
M=[A C;O B]
```

All’interno del workspace digitiamo *esempi*, richiamiamo cioè lo script che esegue i comandi richiesti.

Nello specifico avremo:

```
n=input('Dammi il valore di n: ')
```

lo script chiede di inserire la dimensione delle matrici, dopodiché lo script esegue in automatico la creazione di tutte le matrici.

Le matrici *A* e *B* si costruiscono come somma di diversi blocchi, che progressivamente costruiscono le matrici richieste.

Create le matrici *C* e *O* lo script restituisce in output la matrice mosaico *M*.

```
M=[A C;O B]
```

costituita da 4 blocchi, cioè dalle 4 matrici generate durante l’elaborazione dello script.

Function

Per scrivere la function devo solo modificare la prima riga dello script:

```
function M=mat1(n)
A=diag(3*ones(n,1))+triu(ones(n),1)
B=diag(4*ones(n,1))+diag(2*ones(n-1,1),1)+diag(2*ones(n-1,1),-1)
C=rand(n)
O=zeros(n)
M=[A C;O B]
```

La function viene salvata attribuendogli il nome *mat.1* .

Per richiamare la function basta digitare nel workspace:

`M=mat1(n)` dove *n* è un intero a scelta;

`help mat1` compare la prima riga di testo della function in cui è possibile inserire un commento per capire immediatamente quale processo, operazione svolge la function.

Quindi l'esercizio dovrebbe iniziare sempre con la creazione di uno script, tramite il comando editor, nominare e salvare lo script; durante l'elaborazione dello script è consigliato sempre salvare e richiamare nel workspace lo script, in modo da rendersi conto immediatamente se lo svolgimento dei processi prosegue in maniera corretta.

Quando si è concluso di scrivere lo script e si è sicuri della sua correttezza si può trasformare lo script in function modificando la prima riga dello script.

Pertanto in sintesi si può così riassumere:

1. Aprire il file editor per iniziare ad elaborare lo script.
2. Nominare e salvare lo script.
3. Elaborare i processi e le operazioni necessarie allo svolgimento del problema.
4. Concluso lo script trasformarlo in function modificando la prima riga dello script, nominare e salvare.
5. Completare la function aggiungendo in testa al file una riga di commento attraverso il simbolo % In modo tale da capire immediatamente i processi svolti dalla function richiamandola dal workspace tramite il comando help.

2.1 Operatori logici

Gli operatori logici si applicano tra scalari, tra vettori, o matrici, di dimensioni analoghe, oppure tra un vettore, o matrice, ed uno scalare; gli operatori testano elemento per elemento il soddisfacimento della relazione.

I primi quattro operatori verificano solo la parte reale, gli ultimi due, == e ~=, anche la parte immaginaria.

Per confrontare stringhe, si utilizza la funzione *strcmp*

I risultati di operazioni logiche possono essere attribuiti a variabili.

`>` maggiore;

`<` minore;

`>=` maggiore o uguale;

`<=` minore o uguale;

`==` uguale;

`~=` diverso;

`~` not;

`&&` and;

|| or;

x1=… assegnazione;

x2==… fai un confronto e dimmi se è vero o falso;

$a = b \quad \neq a == b$

Una operazione logica può essere vera o falsa:

$q = 3 == 6$ dimmi se 3==6, il risultato mettilo in q;

$q = 0$ FALSO;

$q = 3 <= 6$ dimmi se 3<=6, il risultato mettilo in q;

$q = 1$ VERO.

In Matlab un risultato logico dà un valore numerico: FALSO=0 VERO=1;

Se chiamiamo una variabile *for*, *abs*, *sin* le corrispondenti istruzioni non possono essere utilizzate.

2.2 Istruzioni di controllo

Il linguaggio Matlab possiede i classici costrutti iterativi e condizionali:

for ripetizione di un insieme di istruzioni per un numero predeterminato di iterazioni;

while ripetizione di un insieme di istruzioni fintantoché una determinata condizione rimane vera;

if istruzione condizionale, può utilizzare *else* e *else if*;

else identifica un blocco di istruzioni alternative;

else if esegue un blocco di istruzioni se è soddisfatta una condizione alternativa;

end termina le istruzioni *f*or, *while* e *if* ;

break termina l'esecuzione di un ciclo *for* o *while*;

switch indirizza il controllo di un programma confrontando l'espressione di input con le espressioni associate alle clausole *case*;

case utilizzato con *switch* per controllare l'esecuzione di un programma.

2.3 Ciclo *for*

Consideriamo la serie di Taylor per l'esponenziale:

$$e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!}$$

$$x = 1 \quad e = \sum_{k=0}^{\infty} \frac{1}{k!} \approx \sum_{k=0}^{N} \frac{1}{k!}$$

consideriamo ≈ un'approssimazione perché non possiamo sommare infiniti valori.

$k! = 1 \cdot 2 \cdot 3 \cdot \ldots \ldots \ldots \cdot k$ *k* fattoriale.

Costruiamo uno script per eseguire questa sommatoria, all'interno è possibile visualizzare de cicli *for* annidati:

```
N=15;

% 1° ciclo for

s=1;     % 1/0!
for k=1:N
   factk=1;
   for i=1:k
      factk=factk*i;
   end
   s=s+1/factk;
end
s
s-exp(1)
```

s-exp(1) ci dà l'errore, la precisione fino all'8^ cifra decimale;

Costruiamo un secondo script per eseguire il calcolo precedente in maniera differente, sempre tramite un ciclo *for*:

```
% 2° ciclo for

ss=1;
for k=1:N
   vk=[1:k]';
   factd=prod(vk);
   ss=ss+1/factd;
end
ss
```

I valori finali del primo script *s* e del secondo script *ss* devono essere uguali.

2.4 Ciclo *while*

Consideriamo la stessa serie di Taylor per l'esponenziale precedente e costruiamo uno script in cui la risoluzione viene eseguita tramite un ciclo *while*:

```
% ciclo while

tau=1e-123;
sss=1;
k=0;
addendo=123;
while addendo>=tau
   k=k+1;
   factk=1;
   for i=1:k
      factk=factk*i;
   end
   addendo=+1/factk;
   sss =sss+addendo;
end
sss
k
```

k posto alla fine permette di visualizzare i cicli effettuati per raggiungere il risultato.

2.5 Espressioni intensive ed estensive

L'espressione:

$for\ i{:}\ N$

$\vdots$

end

In Matlab genera un vettore riga.

$for\ i = 2{:}\,2{:}\,30$ genera tutti i numeri pari da 2 a 30 con passo 2, espressione intensiva;

$for\ i = 0{:}\,0.1{:}\,10$ genera tutti i numeri da 0 a 10 con passo 0.1, espressione intensiva;

$for\ i = [2\ \ 5\ \ 96\ \ 123]$ genera un vettore di 5 elementi assegnati, espressione estensiva;

Queste espressioni generano dei vettori riga; in Matlab non si è vincolati all'uso di numeri interi, Matlab legge sia i decimali che i numeri complessi, come già visto.

2.6 Algoritmi per la risoluzione di calcoli complessi

Matlab possiede Algoritmi per la risoluzione di calcoli complessi:

- calcolo del determinante di una matrice;
- calcolo dell'inversa di una matrice;
- calcolo di sistemi lineari;
- calcolo di autovalori e autovettori.

$det(A) = \cdots$ il determinante di una matrice *A* viene calcolato tramite un algoritmo molto veloce:

Algoritmo di Gauss: fattorizzazione $A=LU$;

$inv(A) = \cdots$ la matrice inversa di *A* viene calcolata tramite: fattorizzazione *LU* per *n* sistemi lineari;

$inv(A) = A^{-1}$

Consideriamo un sistema:

$Ax = b$

$e = ones(5,1)$

$b = A \cdot e$ con $x=e$;

$x = A^{-1} \cdot b$

$x = inv(A) \cdot b = e$ *x* ed *e* non sono perfettamente uguali; costo computazionale: $\frac{4}{3}n^3$.

$x - e \neq 0$

E' possibile risolvere il sistema senza calcolare l'inversa di *A*, ma utilizzando l'algoritmo di Gauss:

$x = A\backslash b = e$ questa funzione abilita l'algoritmo di Gauss e risolve senza dover calcolare l'inversa di *A*, costo computazionale: $\frac{n^3}{3}$.

Con questa funzione utilizzo le librerie di Matlab, sfrutto l'algoritmo di Gauss per risolvere il sistema, i calcoli vengono eseguiti più velocemente rispetto al caso in cui voglia risolvere il sistema obbligando il programma a calcolare l'inversa di *A*.

Vediamo alcune considerazioni su matrici:

$A\backslash B = A^{-1}B = inv(A)B$

$A/B = AB^{-1} = Ainv(B)$

Il prodotto matriciale non è commutativo:

$A\backslash B = A^{-1}B \quad \neq \quad B/A = BA^{-1}$

* operatore prodotto matriciale;

$lam = eig(A)$ calcola gli autovalori di *A*; per una matrice 5x5 sono5, ma non è detto che siano reali, possiamo avere anche autovalori complessi, mentre gli autovalori di una matrice simmetrica sono sempre reali;

$lam = eig(B)$ calcola gli autovalori di *B*;

$[U \quad D] = eig(B)$

U matrice degli autovettori per colonna;

D matrice degli autovalori;

Se conosco autovalori e autovettori di una matrice, la matrice è diagonalizzabile:

$A = UDU^{-1}$

$U^{-1}AU = D$

Non tutte le matrici sono diagonalizzabili, non tutte le matrici possiedono autovalori e autovettori, con le matrici simmetriche non si hanno problemi di questo tipo.

La matrice non è diagonalizzabile se il problema è mal posto.

Un problema è *ben posto* se esso possiede, in un prefissato campo di definizione, una e una sola soluzione e queste dipende con continuità dai dati, in caso contrario viene detto *mal posto*.

Quando il problema è *instabile* può accadere che una piccolissima perturbazione sui dati in ingresso può portare ad una soluzione molto differente da quella corrispondente ai dati esatti.

Anche quando il problema è *stabile* si cerca di dare una misura quantitativa di come la sua soluzione venga influenzata da una perturbazione sui dati; questa caratterizzazione viene detta *condizionamento del problema*.

Creiamo uno script per la risoluzione di un sistema in cui abilitiamo l'algoritmo di Gauss con pivoting:

```
% Calcoli matriciali con algoritmo di
% Gauss con pivoting

clear all
n=20;
A=rand(n);
e=ones(n,1);
b=A*e;
tic
x=A\b;            % Gauss con pivoting
toc
tempo=toc
e-x;
condizionamento=cond(A)
errore=norm(x-e)
```

Progressivamente, aumentando la dimensione *n* della matrice si verifica che:

- aumentano i tempi di calcolo;
- aumenta il condizionamento;
- aumenta l'errore.

Se consideriamo un problema mal condizionato, utilizzando ad esempio la matrice di Hilbert, anche se usiamo il miglior algoritmo otteniamo sempre risultati sbagliati, il programma segnala tali incongruenza tramite messaggi di *warning*.

2.7 Sottoindicizzazione

$v(1) = U(1,1)$ vettore con una componente;

$v(2) = U(2,1)$ vettore con due componenti;

$v = U(:,1)$ vettore colonna estratto dalla prima colonna di *U*: 1° autovettore;

$v = U(:,3)$ vettore colonna estratto dalla terza colonna di *U*: 3° autovettore;

$q = U(2,:)$ seconda riga della matrice *U*: 1° autovettore;

$q = U(2,3:5)$ 3:5 range, della seconda riga prendi dal 3° al 5° elemento;

$C = A(1:3,1:3)$ crea una sottomatrice 3x3 di *A*;

$C = A(4:5,2:5)$ crea una sottomatrice 2x4 di *A*;

$A(4:5,2:5) = zeros(2,4)$ crea una sottomatrice 2x4 di *zeri*;

$zeros(2,4) = A(4:5,2:5)$ ERRORE;

$A(4:5,2:5) = 7$ assegniamo al blocco un numero;

Queste funzionalità permettono di creare maschere su immagini, che in realtà sono matrici *i j*, in alcune celle.

$C = A(:,[1\ 3\ 4])$

E' possibile creare la sottoindicizzazione con vettori di 1 e di 0.

$v = [1:5]'$

$v = [11:15]'$

$z = v([1:4])$ crea un vettore con gli elementi dal 1° al 4° di *v;*

$z = v([1\ \ 4])$ crea un vettore di due elementi, il 1° e il 4° di *v;*

$z = v(logical[1\ 0\ 0\ 1\ 0])$ stiamo passando al vettore valori logici:

0: NO non lo prendi;

1:SI lo prendi;

$v = [11:25]'$ crea vettore colonna da 11:25;

$v > 0$ le operazioni logiche le posso fare direttamente sul vettore;

$v = 1$ crea vettore di tutti 1, cioè tutti gli elementi di *v* sono *>0*;

$C = v(v > 20)$

$C = v(v > 0)$

3. CREAZIONE DI GRAFICI 2D-3D IN MATLAB

Consideriamo una funzione $y = f(x)$ con $x \in [a, b]$.

Non si considera un dominio continuo, ma un dominio discreto, cioè costituito da un numero finito di punti, su cui ci si accontenta di avere una soluzione approssimata della funzione:

$[x_1, x_2, \dots, x_N]$ partizione equidistante del dominio $[a, b]$;

$x_1 = a$

$x_N = b$

$x_i < x_{i+1}$

La funzione di base prevede in Matlab la creazione di due vettori, *x* e *y*:

$x = [x_1, x_2, x_3, \dots, x_N]$

$y = [y_1, y_2, y_3, \dots, y_N]$

$y_i = f(x_i)$ $i = 1, 2, \dots, N$ con *N* sufficientemente grande.

3.1 Funzione plot

Dopo aver creato i vettori *x* e *y* è possibile utilizzare in Matlab la funzione `plot` e visualizzare una curva nel piano, in 2D, che *interpola*, per mezzo di una spezzata, i punti (x_i, y_i) realizzando il grafico desiderato.

Esempio

Consideriamo la funzione: $y = 5sin(x)cos^2(x)$ nell'intervallo $x \in [0, 4\pi]$.

Creiamo un primo vettore x_1 che esegue una grigliatura dell'intervallo di interesse con spaziatura 0.25:

```
x1=[0:0.25:4*pi];
```

Il vettore così realizzato ha 51 elementi `x1=[0 0.25 0.5 … 12.25 12.5]`.

L'istruzione `x1=linspace(0,4*pi,51);` fornisce il medesimo risultato.

Definiamo un secondo vettore x_2 che campioni l'intervallo di interesse con 1000 punti equispaziati:

```
x2=linspace(0,4*pi,1000);
```

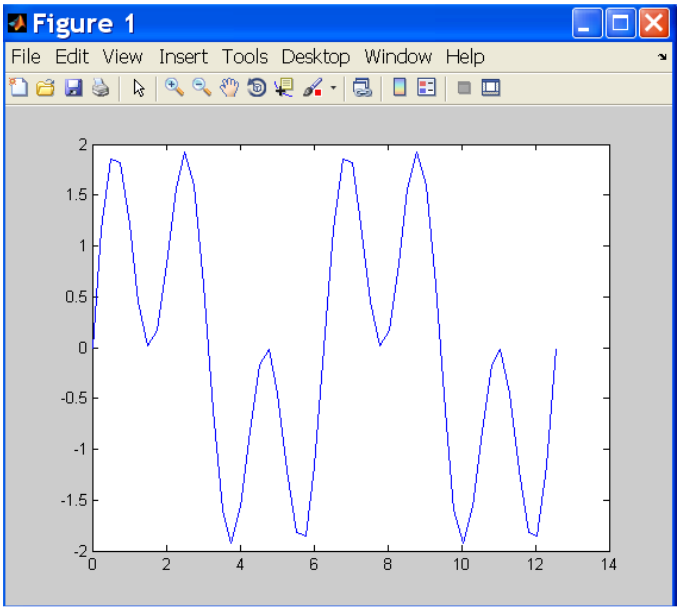
Ora generiamo i vettori y_1 e y_2 contenenti i valori della funzione nei punti individuati dai vettori x_1 e x_2:

```
y1=5*sin(x1).*cos(x1).^2;
```

```
y2=5*sin(x2).*cos(x2).^2;
```

I vettori x_1 e x_2 hanno rispettivamente dimensione 51 e 1000, alla fine è possibile generare i grafici, iniziando con il visualizzare la curva ottenuta interpolando i vettori x_1 e y_1:

```
x1=[0:0.25:4*pi];
x2=linspace(0,4*pi,1000);
y1=5*sin(x1).*cos(x1).^2;
y2=5*sin(x2).*cos(x2).^2;
plot(x1,y1)
```



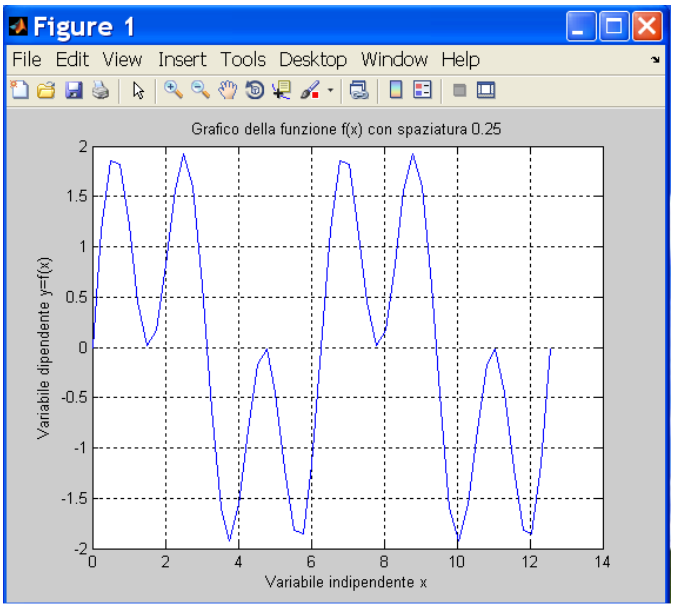
Il grafico generato di default è spoglio e privo di etichette.

3.2 Funzioni grid, label, title

Al grafico precedente aggiungiamo di seguito:

- una griglia utilizzando la funzione `grid`,
- il nome degli assi utilizzando le funzioni `xlabel` ed `ylabel`,
- il titolo del grafico utilizzando la funzione `title`:

```
grid
xlabel('Variabile indipendente x')
ylabel('Variabile dipendente y=f(x)')
title('Grafico della funzione f(x) con spaziatura 0.25')
```



Tale grafico risulta essere molto spigoloso perché abbiamo utilizzato un dominio discreto con un numero N di punti troppo piccolo.

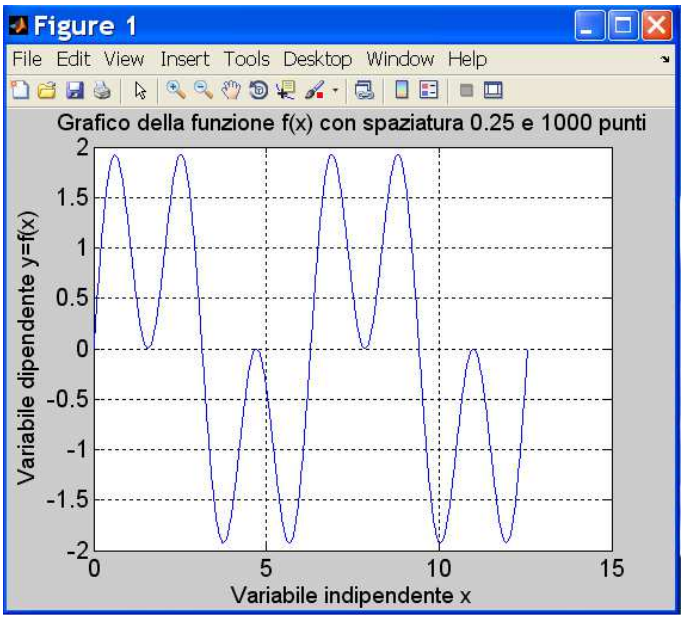
Ora generiamo un grafico corrispondente alla coppia di vettori (x_2, y_2), che hanno dimensione 1000.

Ci aspettiamo un grafico che riproduca in modo più fedele la funzione in esame.

Completiamo ulteriormente, rispetto all'esempio precedente, il set di istruzioni di formattazione introducendo la scelta del *font* e della dimensione dei caratteri nelle varie etichette.

Per mezzo dell'istruzione *set* impostiamo inoltre la dimensione delle cifre che compaiono negli assi delle ascisse e delle ordinate.

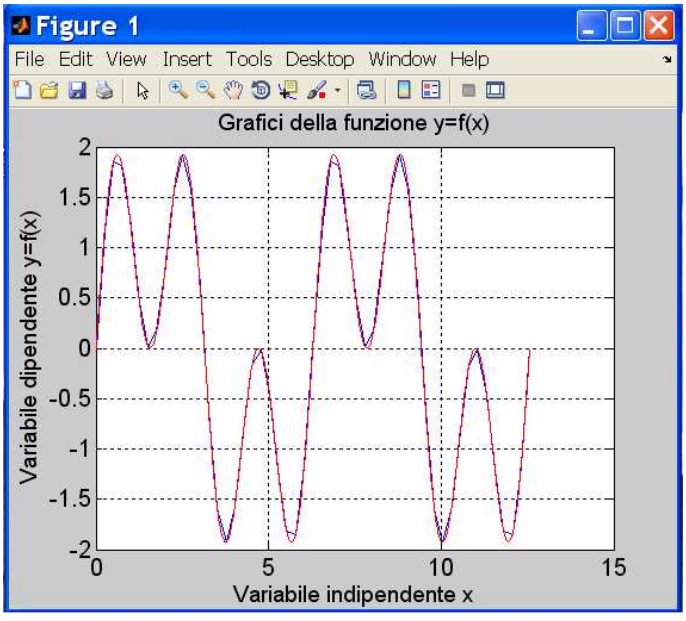
```
plot(x2,y2)
grid
xlabel('Variabile indipendente x','FontName','Arial','FontSize',14)
ylabel('Variabile dipendente y=f(x)','FontName','Arial','FontSize',14)
title('Grafico della funzione f(x) con spaziatura 0.25 e 1000 punti','FontName','Arial','FontSize',14)
set(gca,'FontName','Arial','FontSize',15)
```



3.3 Grafici multiplot

Ora desideriamo riportare in un unico grafico le due curve ottenute; il modo più immediato è quello di passare alla funzione *plot* un numero superiore di argomenti di input:

```
plot(x1,y1,'b', x2,y2,'r')
grid
xlabel('Variabile indipendente x','FontName','Arial','FontSize',14)
ylabel('Variabile dipendente y=f(x)','FontName','Arial','FontSize',14)
title('Grafici della funzione y=f(x)','FontName','Arial','FontSize',14)
set(gca,'FontName','Arial','FontSize',15)
```



Gli argomenti *'b'* ed *'r'* assegnano i colori blu e rosso alle linee dei rispettivi grafici; i colori disponibili sono diversi:

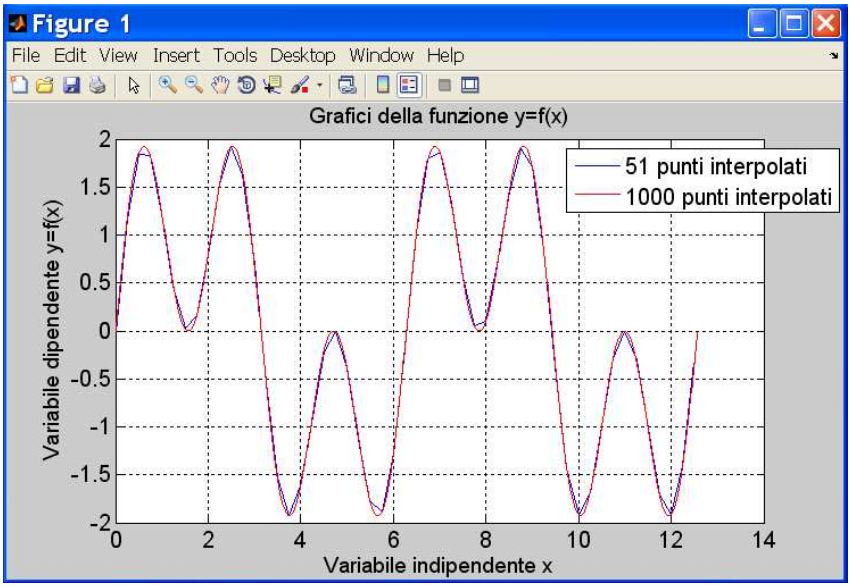
```
b      blu
g      verde
r      rosso
c      ciano
m      magenta
y      giallo
k      nero
```

E' possibile inoltre attribuire colori arbitrari.

3.4 Funzione legenda

Il comando `legend` genera una legenda con le stringhe di testo inserite come argomenti.

```
plot(x1,y1,'b', x2,y2,'r')
grid
xlabel('Variabile indipendente x','FontName','Arial','FontSize',14)
ylabel('Variabile dipendente y=f(x)','FontName','Arial','FontSize',14)
title('Grafici della funzione y=f(x)','FontName','Arial','FontSize',14)
legend('51 punti interpolati','1000 punti interpolati')
set(gca,'FontName','Arial','FontSize',15)
```



3.5 Funzione hold on

Un modo alternativo per sovrapporre più grafici nella stessa finestra è utilizzare la funzione `hold on`.

Per concludere si riporta lo script completo che genera le quattro figure su esposte:

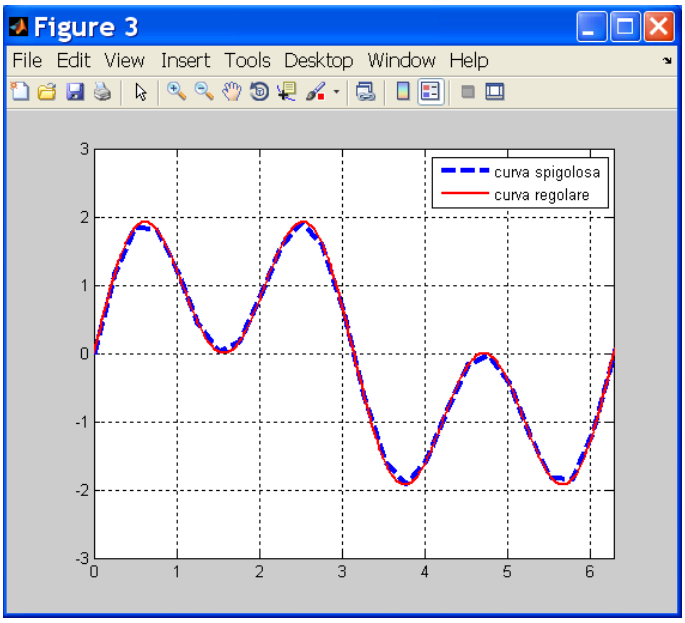
```
x1=linspace(0,4*pi,51);
x2=linspace(0,4*pi,1000);
y1=5*sin(x1).*cos(x1).^2;
y2=5*sin(x2).*cos(x2).^2;
figure(1)
plot(x1,y1)
figure(2)
plot(x2,y2),grid
xlabel('Variabile indipendente x','FontName','Arial','FontSize',14)
ylabel('Variabile dipendente y=f(x)','FontName','Arial','FontSize',14)
title('Grafico della funzione f(x) con spaziatura 0.25 e 1000
punti','FontName','Arial','FontSize',14)
legend('51 punti interpolati','1000 punti interpolati')
set(gca,'FontName','Arial','FontSize',15)
figure(3)
plot(x1,y1,'b','LineStyle','--','LineWidth',3)
hold on
plot(x2,y2,'r','LineStyle','-','LineWidth',2),legend('curva spigolosa','curva
regolare')
hold off
grid
figure(4)
plot(x1,y1,'b', x2,y2,'r')
grid
xlabel('Variabile indipendente x','FontName','Arial','FontSize',14)
ylabel('Variabile dipendente y=f(x)','FontName','Arial','FontSize',14)
title('Grafici della funzione y=f(x)','FontName','Arial','FontSize',14)
legend('51 punti interpolati','1000 punti interpolati')
set(gca,'FontName','Arial','FontSize',15)
```

3.6 Range degli assi coordinati

E’ possibile modificare il range degli assi coordinati:

```
axis([0 2*pi -3 3])
```

La prima coppia rappresenta gli estremi dell’asse *x*, la seconda coppia rappresenta gli estremi dell’asse *y*, con questa funzione è possibile ottenere uno zoom del grafico in esame:



3.7 Tipo di linea

E’ possibile modificare il tipo di linea:

```
'-'   solid      (linea a tratto continuo)
':'   dotted     (linea punteggiata)
'--'  dashed     (linea tratteggiata)
'-'   dashdot    (linea-punto-linea)
```

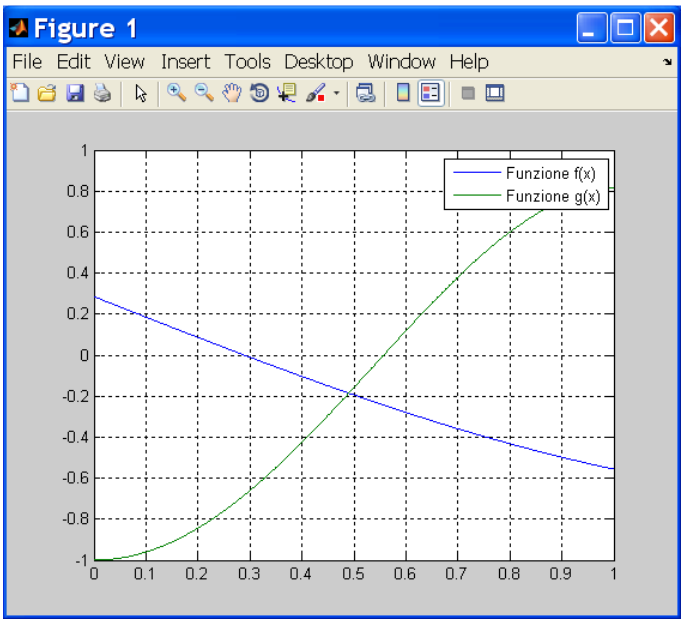
3.8 Creazione di grafici sovrapposti

E’ possibile creare grafici sovrapposti che rappresentano più curve differenti, creiamo ad esempio i grafici relativi alle curve:

$$f(x) = \frac{x^2}{1+x^2} - sen(x)$$

$$g(x) = 2xsen(x) - 1$$

```
x=[0:0.001:1];
f=((x.^2)/(1+x.^2))-sin(x);
g=2*x.*sin(2*x)-1;
figure(1)
plot(x,f,x,g),legend('Funzione f(x)','Funzione g(x)')
grid
```



3.9 Grafici di curve parametriche

E' possibile creare grafici di curve parametriche, la procedura di tracciamento di curve parametriche è analoga a quanto visto per i grafici di funzioni scalari di una variabile.

Si devono generare i punti (*x*,*y*) della curva ed una volta fatto ciò è possibile impiegare il comando standard `plot`.

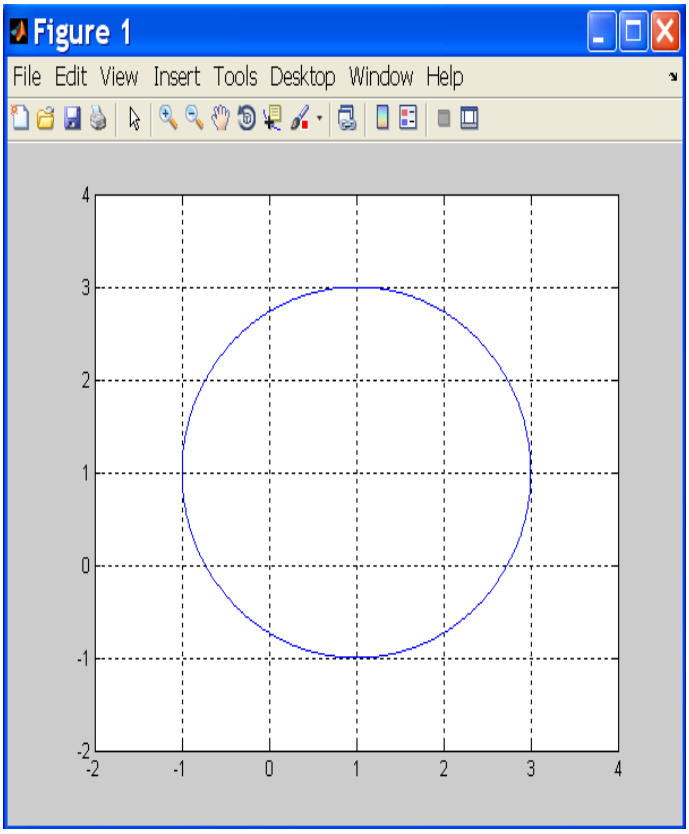
Il seguente codice di esempio, grafica nel piano la circonferenza centrata nel punto (1,1) e avente raggio pari a 2:

$$x = x_C + Rcos(t)$$

$$y = y_C + Rsen(t)$$

$t \in [0,2\pi]$ *R = raggio della circonferenza*

```
t=0:0.01:2*pi;
x_c1=1; % ascissa del centro
y_c1=1; % ordinata del centro
R=2; % raggio
x=x_c1+R*cos(t);
y=y_c1+R*sin(t);
plot(x,y), grid
axis([-2 4 -2 4])
```



3.10 Creazione di grafici multipli

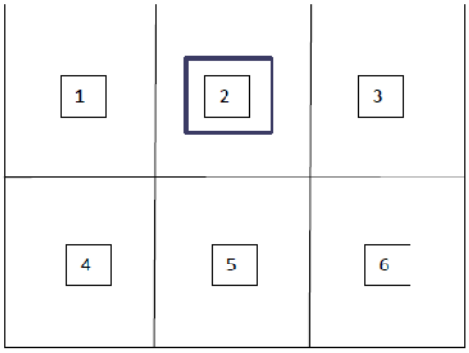
Per creare grafici multipli disposti secondo righe, colonne, o matricialmente, si utilizza la funzione `subplot`.

L'istruzione `subplot(n,m,h)` alloca una finestra grafica e la suddivide in sottofinestre realizzando una matrice con *n* righe ed *m* colonne.

Il terzo parametro h $(1 \leq h \leq n \cdot m)$ definisce la sottofinestra attiva tra le $n \cdot m$ sottofinestre disponibili.

L'indice h dapprima scandisce tutte le sottofinestre della prima riga, poi tutte le sottofinestre della seconda riga, ecc.

L’istruzione `subplot(2,3,2)` alloca una finestra grafica suddivisa come in figura e rende attiva la finestra numero 2:




```
t=0:0.01:5;
y1=sin(t)+cos(t).^2;
y2=sin(t);
y3=cos(t);
y4=sqrt(cos(t).^2+abs(sin(t)));

close all

figure(1)
subplot(1,2,1),plot(t,y1),grid
subplot(1,2,2),plot(t,y2),grid

figure(2)
subplot(2,1,1),plot(t,y1),grid
subplot(2,1,2),plot(t,y2),grid

figure(3)
subplot(2,2,1),plot(t,y1),grid
subplot(2,2,2),plot(t,y2),grid
subplot(2,2,3),plot(t,y3),grid
subplot(2,2,4),plot(t,y4),grid
```

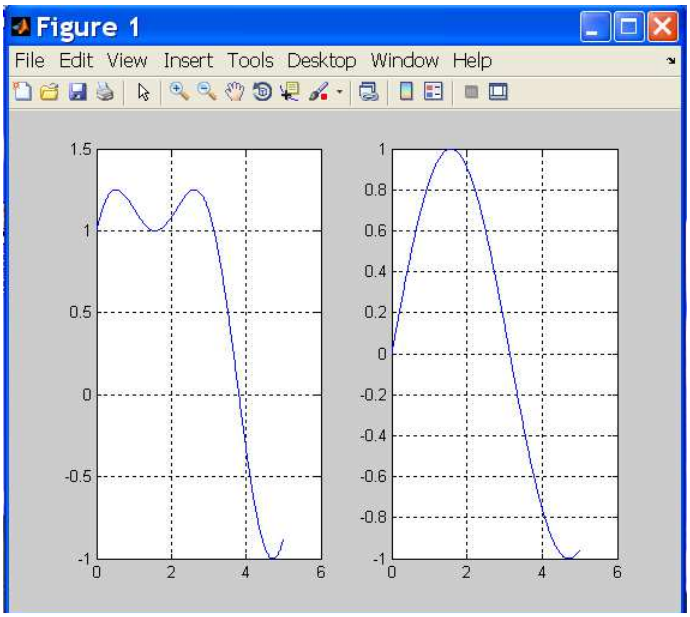
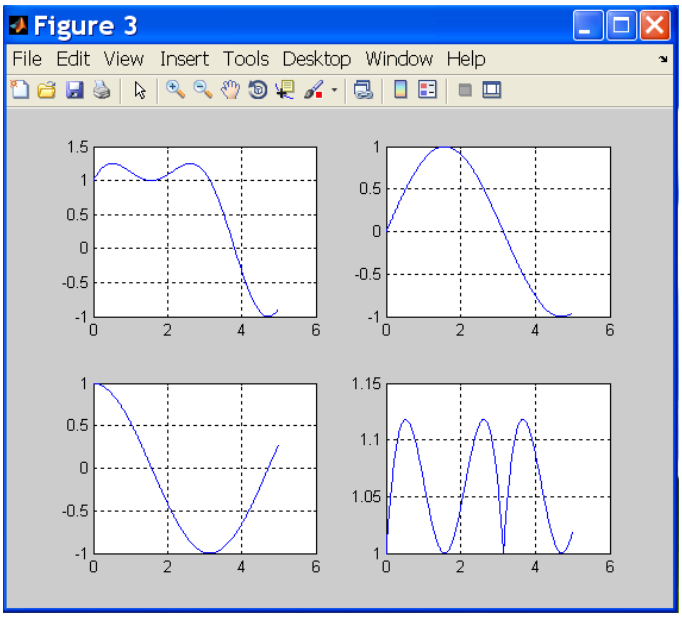


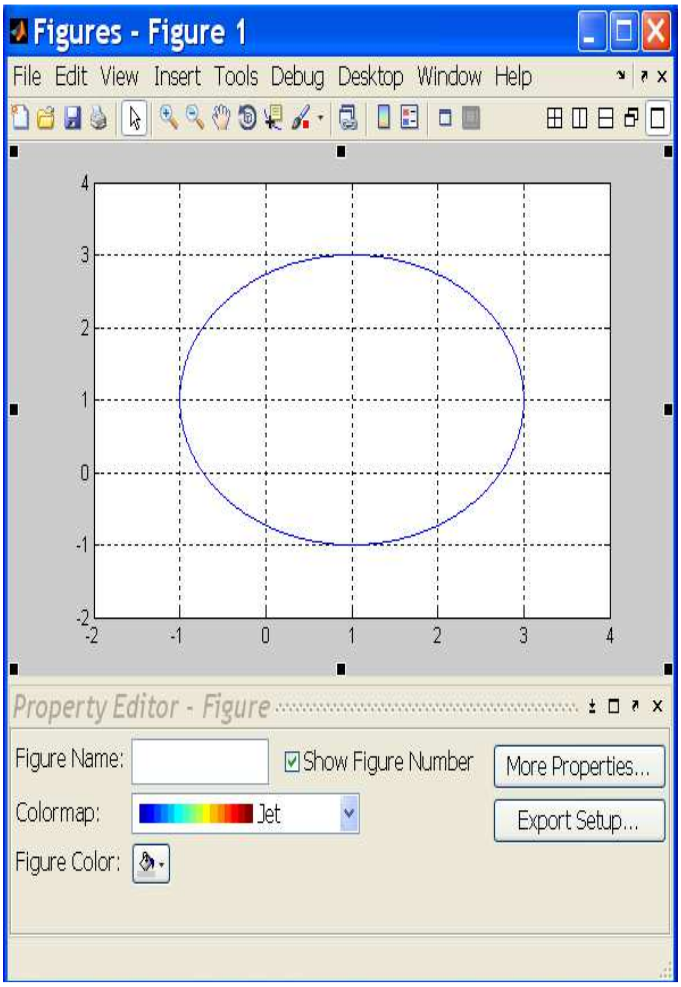
Figure 1
File Edit View Insert Tools Desktop Window Help
1.5
1
0.5
0
-0.5
-1
0 2 4 6
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0 2 4 6


Figure 2
File Edit View Insert Tools Desktop Window Help
1.5
1
0.5
0
-0.5
-1
0 0.5 1 1.5 2 2.5 3 3.5 4 4.5 5
1
0.5
0
-0.5
-1
0 0.5 1 1.5 2 2.5 3 3.5 4 4.5 5



3.11 Implementazione tools

Dalla versione 6.5 in poi la finestra associata al grafico corrente si arricchisce di una serie di menù per mezzo del quale si possono modificare in maniera semplice tutte le proprietà di una data figura.



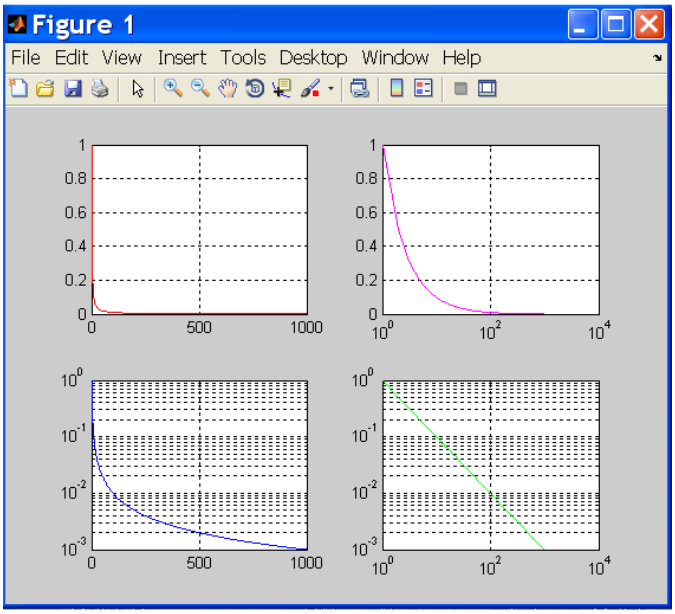
3.12 Grafici avanzati

E’ possibile creare diversi grafici avanzati:

- scale logaritmiche;
- stairs,
- stem,
- diagrammi a torta 2D,
- diagrammi a torta 3D,
- comet,

3.13 Scale logaritmiche

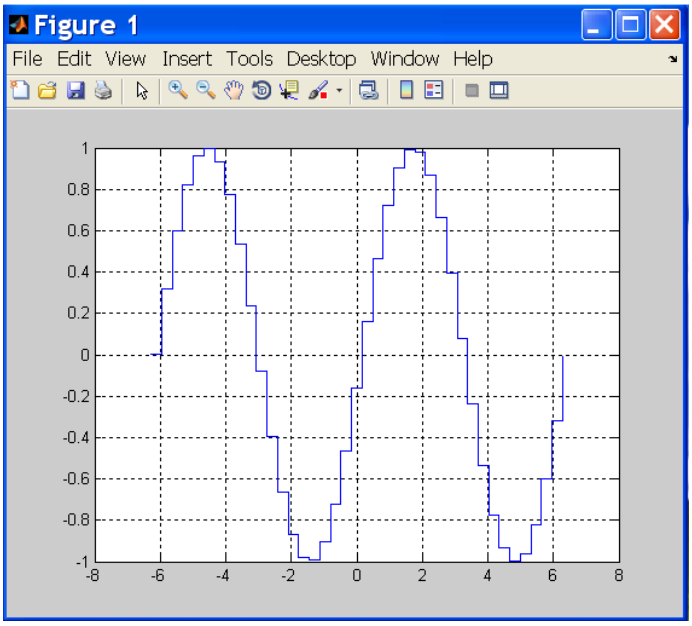
```
x=1:1000; y=1./x;
subplot(2,2,1),plot(x,y,'r'),grid
subplot(2,2,2),semilogx(x,y,'m'),grid
subplot(2,2,3),semilogy(x,y,'b'),grid
subplot(2,2,4),loglog(x,y,'g'),grid
```



3.14 Stairs

Utile per rappresentare segnali a tempo discreto, il grafico viene realizzato interpolando i punti (*x*,*y*) con una funzione costante a tratti (*zero-order-hold*).

```
x=linspace(-2*pi,2*pi,40)
Stairs(x,sin(x)),grid
```

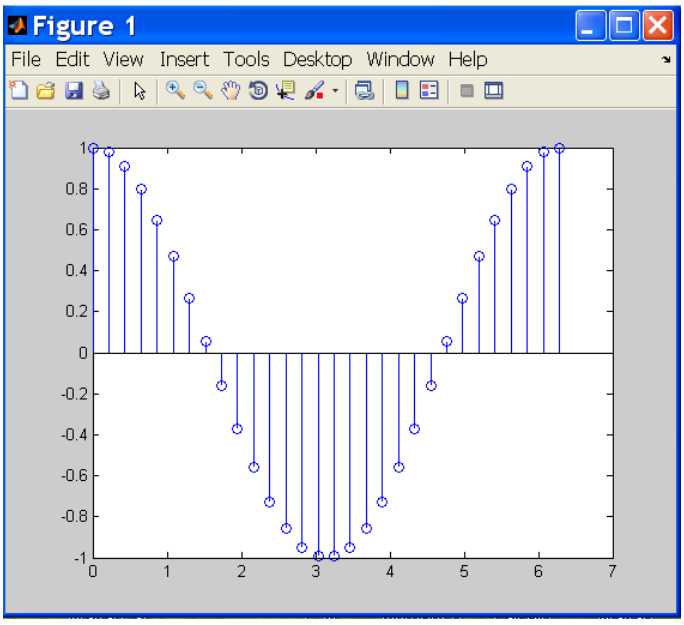


3.15 Stem

La funzione grafica `stem` non interpola i punti (x_i, y_i) per mezzo di una spezzata, ma in luogo di ciò traccia un segmento verticale che connette tali punti con l'asse orizzontale.

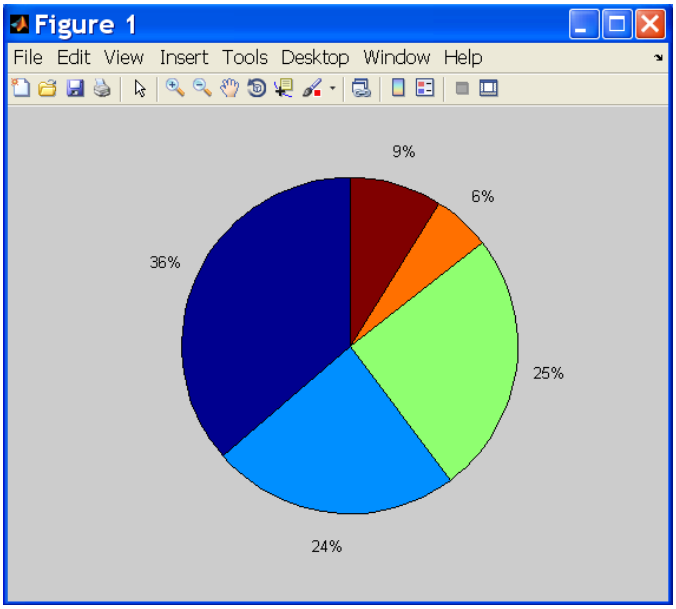
In corrispondenza dei punti viene visualizzato un simbolo marker, di default un piccolo cerchietto.

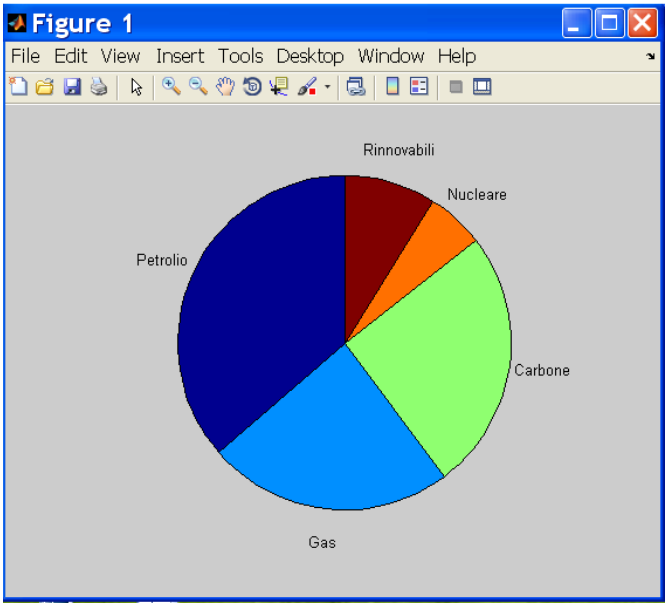
```
t=linspace(0,2*pi,30);
h=stem(t,cos(t));
```



3.16 Diagrammi a torta 2D

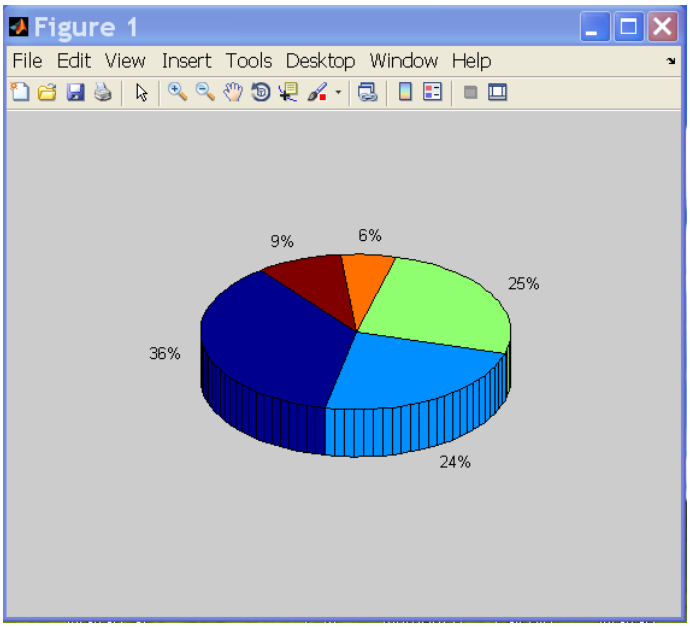
```
x=[4677 3052 3246 728 1139];
pie(x)

x=[4677 3052 3246 728 1139];
pie(x,{'Petrolio','Gas','Carbone','Nucleare','Rinnovabili'})
```





3.17 Diagrammi a torta 3D

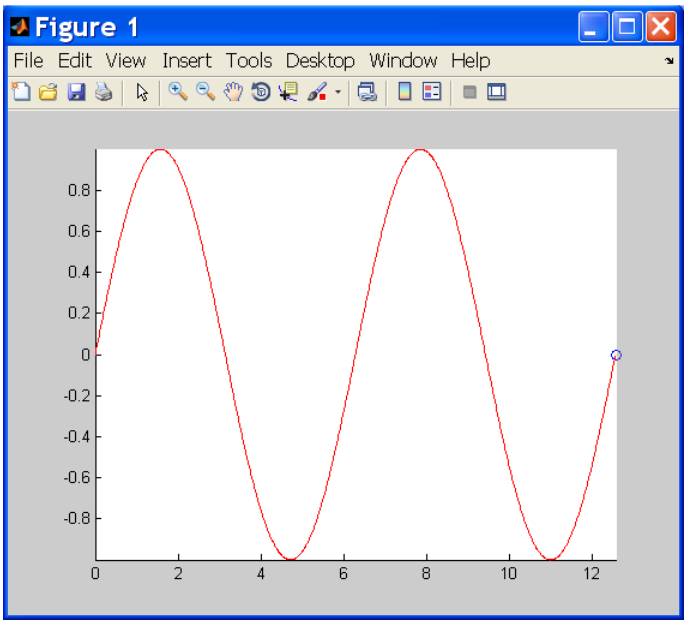
```
x=[4677 3052 3246 728 1139];
pie3(x)
```



3.18 Comet

La funzione `comet` abilita un grafico animato:

```
t=linspace(0,4*pi,1000);
comet(t,sin(t))
```



3.19 Grafici 3D

L'obiettivo è quello di realizzare il grafico tridimensionale di una superficie del tipo:

$z = f(x, y)$

Con le variabili indipendenti *x* e *y* appartenenti ad un dominio rettangolare del piano:

$x_{min} \leq x \leq x_{max}$ $\qquad$ $y_{min} \leq y \leq y_{max}$

La prima istruzione necessaria è la costruzione di due particolari matrici, *X* e *Y*, sulla base del dominio della soluzione di interesse e di uno o più *passi di discretizzazione*.

```
[X,Y]=meshgrid(xmin:stepx:xmax, ymin:stepy:ymax);
```

Sviluppiamo la trattazione riferendoci ad un esempio concreto:

$z \leq sinc\left(\sqrt{x^2 + y^2}\right) = \frac{\sin\left(\sqrt{x^2+y^2}\right)}{\sqrt{x^2+y^2}}$ $\qquad$ $-8 \leq (x, y) \leq 8$

Quindi il dominio è simmetrico rispetto all'origine:

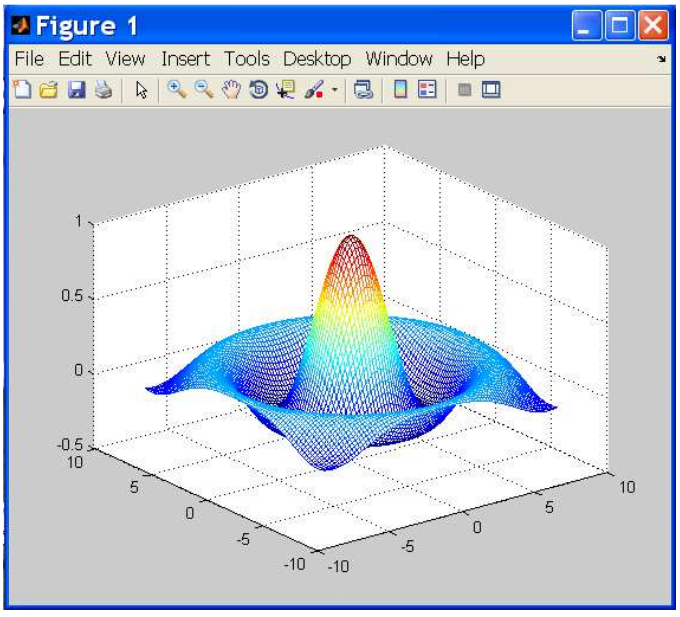
$x_{min} = y_{min} \leq A$ $\qquad$ $x_{max} = y_{max} = B$

Si può adottare la sintassi compatta:

```
[X,Y]=meshgrid(A:step:B);
```

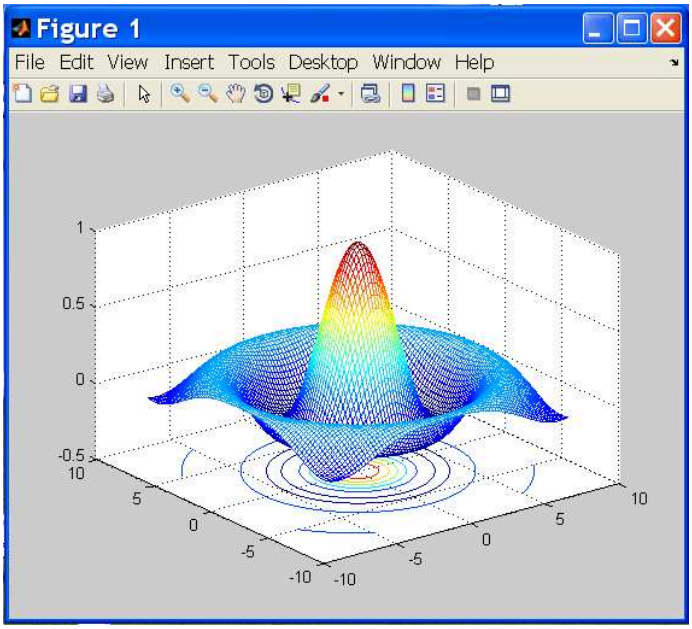
Il codice per graficare la funzione `sinc` è il seguente, scegliamo un passo di discretizzazione pari a 0.2:

```
[X,Y]=meshgrid(-8:.2:8);
R=sqrt(X.^2+Y.^2)+eps;
Z=sin(R)./R;
mesh(X,Y,Z)
```



L’istruzione `meshc(X,Y,Z)`in luogo dell’istruzione `mesh` mostra anche i *contours* della superficie sul piano *x*-*y*:

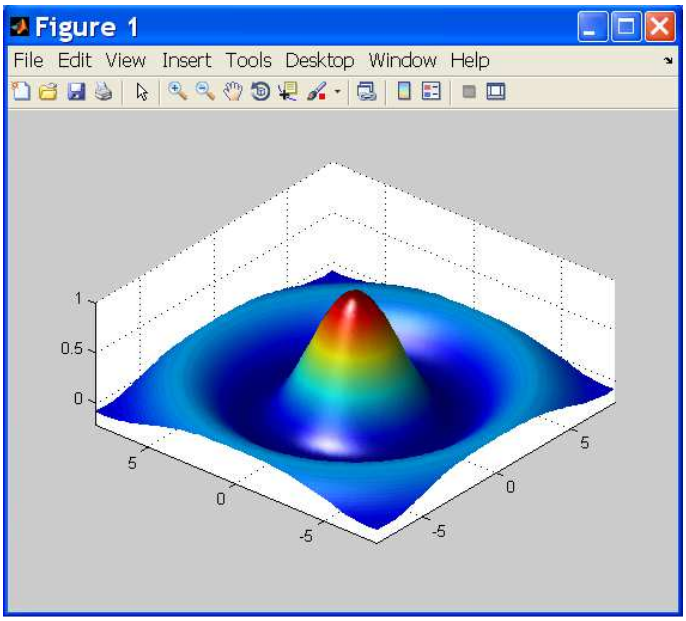
```
[X,Y]=meshgrid(-8:.2:8);
R=sqrt(X.^2+Y.^2)+eps;
Z=sin(R)./R;
meshc(X,Y,Z)
```



Nelle versione più recenti di Matlab è possibile ruotare il grafico in maniera tridimensionale, zoomare, cambiare la colorazione e inserire una *colorbar*.

Utilizzando la funzione `surf`, rispetto alla funzione `mesh` è possibile conferire un *rendering* migliore al grafico:

```
surf(X,Y,Z,'Facecolor','interp','Edgecolor','none','FaceLighting','phong')
daspect([5 5 1])
axis tight
view(-50,30)
camlight left
```



3.20 Curve parametriche in 3D

L'analogo 3D della funzione `plot` è la funzione `plot3`; se *x*, *y* e *z* sono tre vettori della medesima lunghezza, la funzione:

```
plot3(x,y,z)
```

genera una linea 3D attraverso i punti (*x(i)*, *y(i)*, *z(i)*), e produce quindi una sua proiezione 2D.

Consideriamo la curva parametrica:

$$\begin{cases} x = \sin(t) \\ y = \cos(t) \\ z = t \end{cases} \qquad 0 \leq t \leq 10\pi$$

```
t = 0:pi/50:10*pi;
plot3(sin(t),cos(t),t), grid
xlabel('X')
ylabel('Y')
zlabel('Z')
```

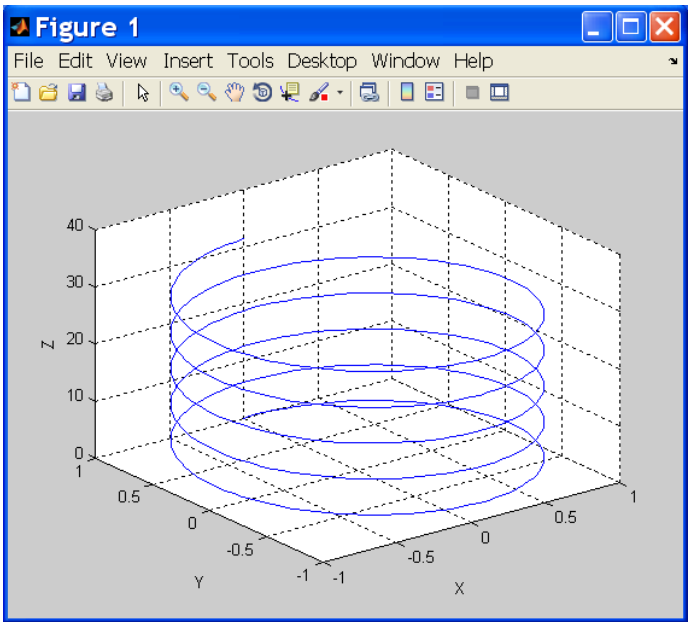


Figure 1
File Edit View Insert Tools Desktop Window Help
40
30
20
10
0
Z
1
0.5
0
-0.5
-1
Y
-1
-0.5
0
0.5
1
X

4. RISOLUZIONE DI EQUAZIONI PER VIA GRAFICA IN MATLAB

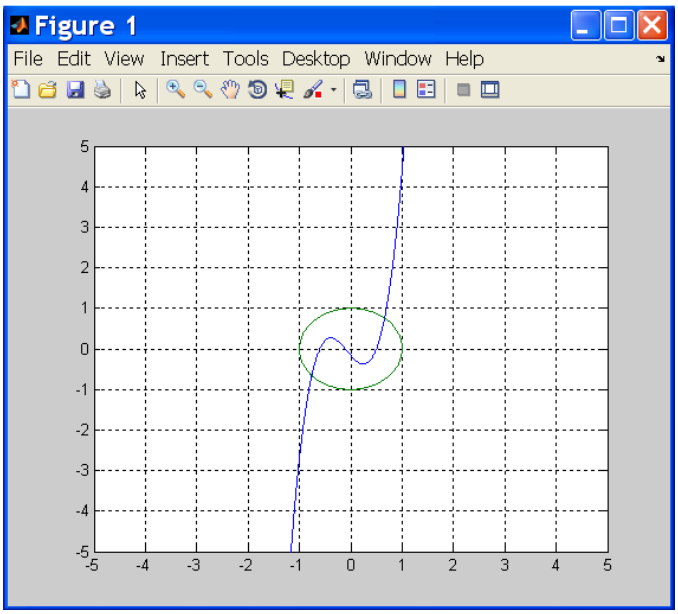
4.1 Esercizio 1

Calcolare per via grafica i punti di intersezione della funzione:

$$f(x) = 5\left(x + \frac{3}{5}\right)\left(x + \frac{1}{10}\right)\left(x - \frac{1}{2}\right)$$

con la circonferenza centrata nell'origine e di raggio unitario:

```
x=[-5:0.001:5];
f=5.*(x+3/5).*(x+0.1).*(x-0.5);
t=0:0.01:2*pi;
x_c=cos(t);
y_c=sin(t);
plot(x,f,x_c,y_c),grid
axis([-5 5 -5 5])
```



4.2 Esercizio 2

Date le curve:

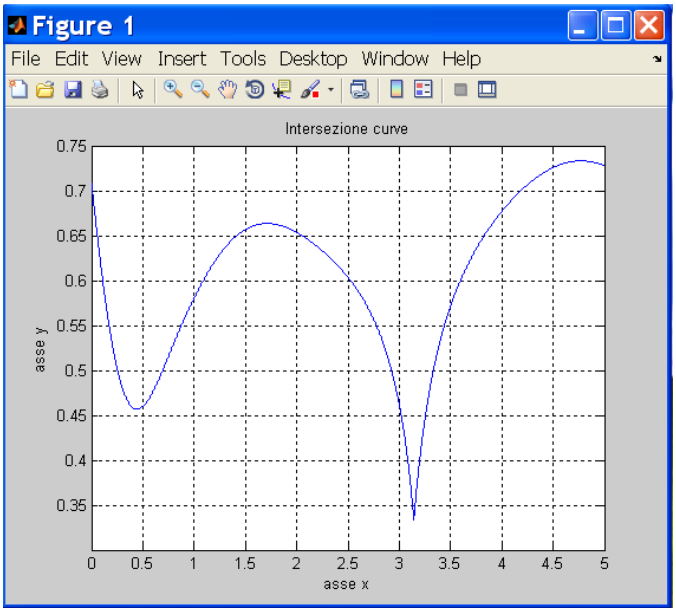
$$y_1 = log(1 + t|\sin(t)|) + \frac{1}{1 + cos^2(t)}$$

$$y_2 = cos[y_1^2(t)]e^{-t}$$

si realizzi il grafico della funzione:

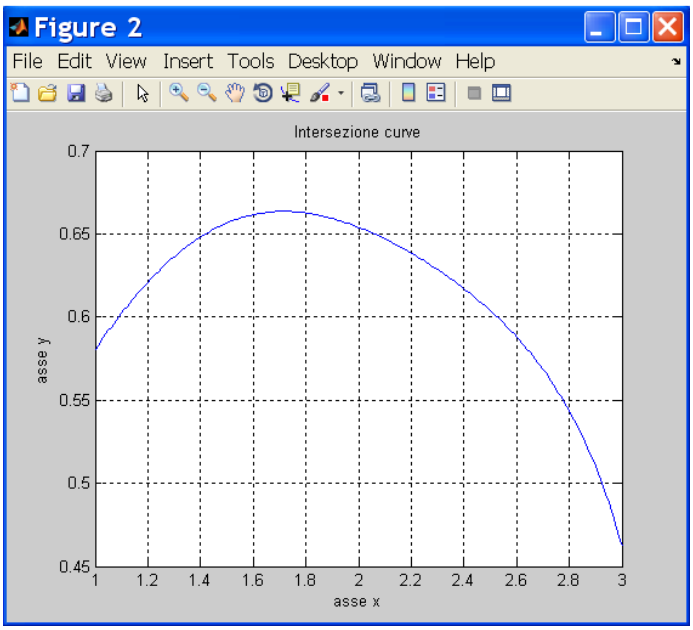
$$z = \frac{\sqrt{y_1^2(t) + y_2^4(t)}}{1 + |y_1(t)|}$$

```
t=[0:0.001:5];
y1=log(1+t.*abs(sin(t)))+1./(1+cos(t).^2);
y2=cos(y1.*y1).*exp(-t);
z=sqrt(y1.^2+y2.^4)./(1+abs(y1));
figure(1)
plot(t,z)
grid
xlabel('asse x'),ylabel('asse y'),title('Intersezione curve')
```



Determinare il valore massimo che la funzione $z(t)$ assume nell'intervallo $t \in [1,3]$.

```
t=[1:0.01:3];
y1=log(1+t.*abs(sin(t)))+1./(1+cos(t).^2);
y2=cos(y1.*y1).*exp(-t);
z=sqrt(y1.^2+y2.^4)./(1+abs(y1));
figure(2)
plot(t,z)
grid
xlabel('asse x'),ylabel('asse y'),title('Intersezione curve')
[massimo posizione]=max(z);
val_max=massimo
t_max=t(posizione)
```



Più piccolo è lo step, cioè l'intervallo di discretizzazione tra 1 e 3 più precisa sarà la soluzione.

4.3 Esercizio 3

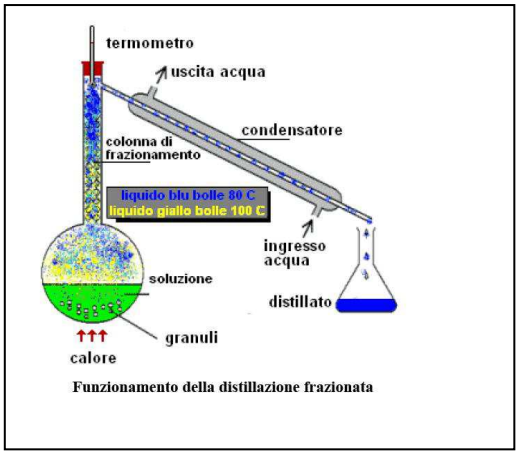
Modellando un processo chimico di distillazione si ottiene la seguente relazione:

$$L = 100\left(\frac{x}{0.6}\right)^{0.625}\left(\frac{1-x}{0.4}\right)^{-1.625}$$

in cui L rappresenta la quantità di liquido, in moli, che resta nel distillatore, ed x è la frazione di benzene nel residuo.

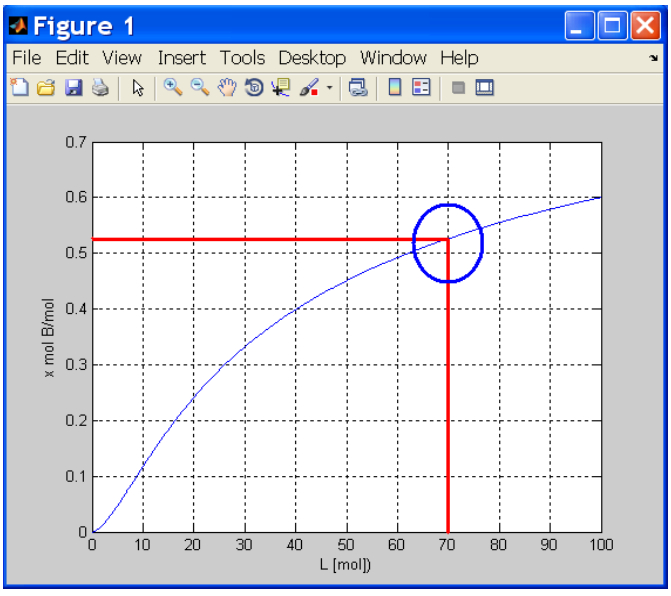
La variabile x assume valori nell'intervallo [0,0.6].

Si desidera individuare il valore della frazione residua x in corrispondenza del quale restano nel distillatore $L = 70$ moli di liquido.



Funzionamento della distillazione frazionata

```
x=[0:0.001:0.6];
L=100*(x/0.6).^(0.625).*((1-x)/0.4).^(-1.625);
figure(1)
plot(L,x),grid,xlabel('L [mol])'),ylabel('x mol B/mol')
figure(2)
plot(x,L),grid,xlabel('x'),ylabel('L ')
```



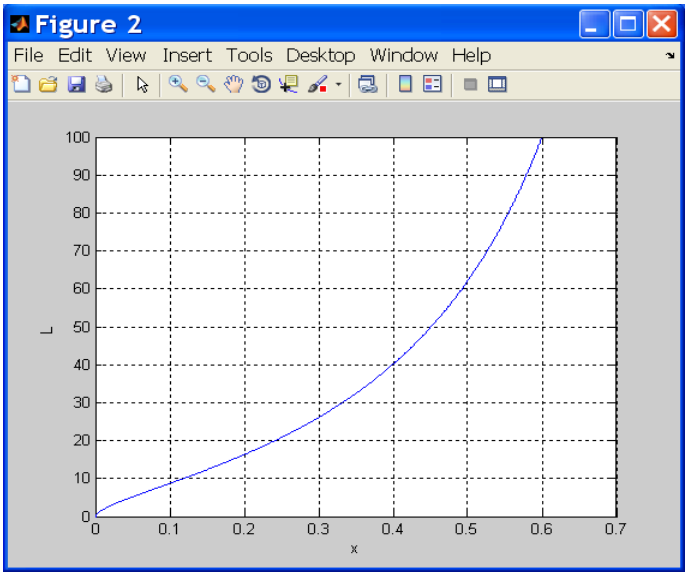


Figure 2
File Edit View Insert Tools Desktop Window Help
100
90
80
70
60
50
40
30
20
10
0
L
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
x

4.4 Esercizio 4

Determinare per via grafica l'intersezione tra le curve:

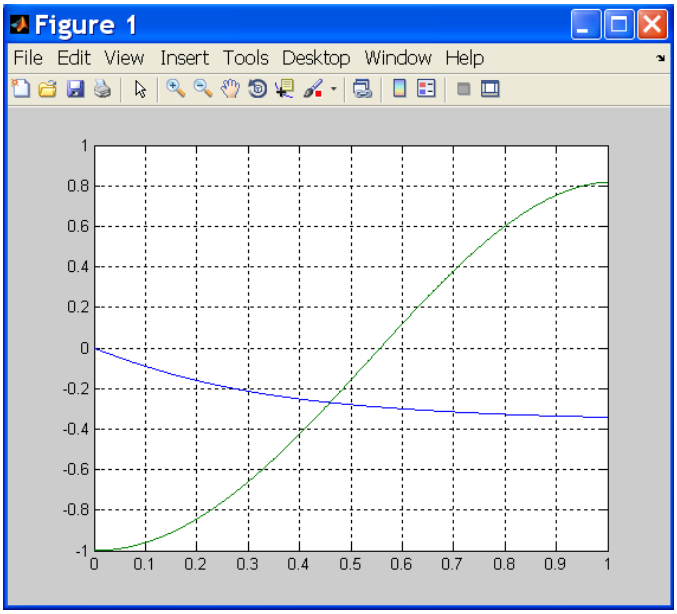
$$f(x) = \frac{x^2}{1+x^2} - \sin(x)$$

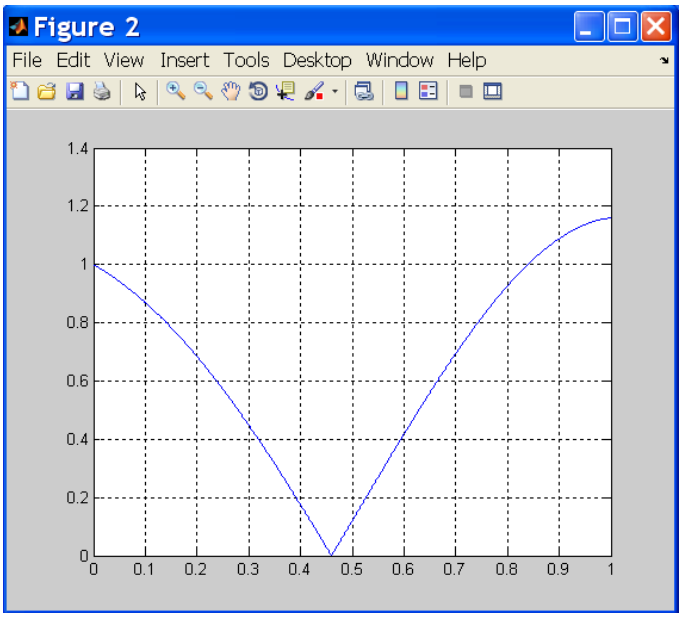
$$g(x) = 2xsin(2x) - 1$$

con $x \in [0,1]$.

Relativamente alla sola parte positiva calcolare il punto di minimo e visualizzare sul workspace di Matlab la soluzione.

```
x=0:0.001:1;
f=(x.^2)./(1+x.^2)-sin(x);
g=2*x.*sin(2*x)-1;
plot(x,f,x,g),grid
z=abs(f-g);
figure(2)
plot(x,z),grid
[valore posizione]=min(z)
disp('La soluzione è: ')
x(posizione)
```





La soluzione è:

$x = 0.46$

5. INTRODUZIONE A SIMULINK

Il software *SIMULINK* è un applicativo associato al programma di calcolo Matlab e ne costituisce una potente e intuitiva interfaccia grafica che ne semplifica grandemente l'impiego da parte dell'utente.

Mediante Simulink è possibile *programmare* l'esecuzione di calcoli in ambiente Matlab in maniera molto più rapida ed *error-free* rispetto alla scrittura dei lunghi e complessi *m-files* che sono necessari, ad esempio, per programmare in Matlab l'integrazione numerica di un sistema di equazioni differenziali di ordine elevato.

Sostanzialmente Simulink è un linguaggio di programmazione ad oggetti, attraverso cui è possibile risolvere equazioni differenziali e sistemi di equazioni differenziali.

Mediante gli oggetti disponibili in ambiente Simulink è possibile simulare dei sistemi anche molto complessi con uno sforzo da parte dell'utente che si limita al tracciamento, su un foglio di lavoro elettronico, di uno schema a blocchi rappresentativo del sistema in esame.

Concentreremo l'attenzione su alcuni casi di interesse:

- visualizzazione di un segnale;
- risoluzione di equazioni differenziali lineari a coefficienti costanti;
- risoluzione di sistemi di equazioni differenziali.

5.1 Generalità: equazione differenziale

In analisi matematica un'equazione differenziale è una relazione tra una funzione *f(x)* non nota ed alcune sue derivate.

Nel caso in cui *y* sia una funzione definita in un intervallo *I* dell'insieme dei numeri reali, ossia:

$y: I \to \mathbb{R}$

si parla di *equazione differenziale ordinaria*, abbreviata con *EDO*, o in alcuni testi *ODE*, acronimo di *ordinary differential equation*.

La scrittura generale di un'equazione differenziale ordinaria, in una variabile *x*, di ordine *n* può essere espressa nella forma:

$$f\big(x, y(x), y'(x), y''(x), \dots, y^n(x)\big) = 0$$

Si chiama *ordine* o *grado dell'equazione* il grado della più alta derivata presente; ad esempio:

$$y''(x) = y(x) + y'(x)$$

è un'equazione differenziale ordinaria del secondo ordine, la funzione incognita *y* è funzione solo di *x*.

Si chiama *soluzione dell'equazione differenziale* una funzione *y*, derivabile un certo numero di volte, che soddisfi la relazione definita dall'equazione.

La risoluzione si ottiene mediante ripetuti processi di integrazione:

$$y^n \to \cdots \to y''' \to y'' \to y' \to y$$

Molto spesso anziché trovare un'espressione analitica di una funzione che soddisfi un'equazione differenziale, è solitamente possibile studiarne l'andamento qualitativo, servendosi di computer in grado di effettuare approssimazioni tramite metodi di calcolo numerici.

Le equazioni differenziali sono uno dei più importanti strumenti che l'analisi matematica mette a disposizione nello studio di modelli matematici nei più disparati settori della scienza, dalla fisica all'ingegneria alla biologia all'economia.

Da questo punto di vista quindi un'equazione differenziale può essere intesa come un'espressione matematica che rappresenta un fenomeno fisico; durante lo studio del fenomeno fisico e della sua rappresentazione numerica sono due le domande che ci si deve porre:

- esiste la soluzione dell'equazione differenziale?
- l'equazione differenziale che si è scritto rappresenta in maniera corretta ed efficace il problema che si deve studiare?

Se il sistema rappresenta adeguatamente il fenomeno, non ci si mette il problema se esiste una soluzione, ma si cerca direttamente una soluzione.

L'ingegnere si accontenta di risolvere il sistema utilizzando metodi numerici e non analitici e di ottenere una soluzione approssimata, perché il problema di partenza è ben posto, sotto ipotesi molto ragionevoli.

Le soluzioni numeriche approssimate dipendono da:

- condizioni iniziali;
- intervallo di integrazione: finestra temporale in cui interessa studiare la simulazione del problema;
- precisione della soluzione approssimata, relativa alla scala del problema;
- metodi di integrazione numerica;
- passo di integrazione.

5.2 Teorema del campionamento di Nyquist-Shannon

In elettronica e telecomunicazioni il teorema del campionamento è uno dei teoremi base della teoria dei segnali e mette in relazione il contenuto di informazione di un segnale campionato con la frequenza di campionamento e le componenti minime e massime di frequenza del segnale analogico originale, definendo così la minima frequenza necessaria per campionare un segnale analogico senza perdere informazioni e per poter quindi ricostruire il segnale analogico originario, detta *frequenza di Nyquist*.

Il campionamento è il primo passo del processo di conversione analogico-digitale di un segnale, consiste nel prelievo di campioni da un segnale analogico e continuo nel tempo ogni Δt secondi.

Δt passo di campionamento;

$f_C = \frac{1}{\Delta t}$ frequenza di campionamento.

Il risultato è un segnale analogico in tempo discreto, che viene in seguito quantizzato, codificato e reso accessibile a qualsiasi elaboratore digitale.

Pertanto il teorema afferma che, sotto opportune ipotesi, in una conversione analogico-digitale la minima frequenza di campionamento necessaria per evitare ambiguità e perdita di informazione nella ricostruzione del segnale analogico originario, ovvero nella riconversione digitale-analogica, con larghezza di banda finita e nota è pari al doppio della sua frequenza massima:

$f_C = 2f_{max}$ *Teorema del campionamento di Nyquist*

Se la funzione da studiare è costante il passo di campionamento può essere grande, se invece la funzione non è molto lineare il passo di campionamento deve essere piccolo in modo tale da non perdere informazioni sulla funzione di partenza.

5.3 Formule alle differenze finite per la risoluzione di equazioni differenziali ordinarie

I metodi di questa classe consentono di determinare una soluzione discreta di un problema differenziale, cioè di approssimare numericamente la soluzione su un intervallo discreto di punti:

$\eta_j \cong y(x_j) \qquad j = 0,1, \dots \dots$

Si pone $\eta_0 = y_0$

I punti di valutazione possono essere equispaziati:

$x_j = x_{j-1} + h = x_0 + jh \qquad h = passo\ di\ discretizzazione$

oppure non equispaziati:

$x_j = x_{j-1} + h_j$

I valori η_j possono essere ottenuti approssimando in vario modo la derivata della soluzione $y(x)$ mediante rapporti incrementali cioè utilizzando differenze finite.

Il metodo più semplice è:

Metodo di Eulero-Cauchy: consiste nell'approssimare la derivata nel punto x_j col rapporto incrementale in avanti:

$$f(x_j, y(x_j)) \cong \frac{y_{j+1} - y_j}{h}$$

Sostituendo i valori incogniti $y_j = y(x_j)$ con le approssimazioni η_j si ottiene lo schema numerico:

$$\begin{cases} \eta_{j+1} = \eta_j + hf(x_j, \eta_j) \\ \eta_0 = y_0 \end{cases}$$

Questa formula viene detta *monostep* ed *esplicita*.

Monostep: la valutazione di η_{i+1} coinvolge solo η_i e non le approssimazioni precedenti η_{i-1} η_{i-2}.

Esplicita: il valore da calcolare ad ogni passo η_{i+1} compare solo a sinistra dell'uguale e non tra gli argomenti della funzione *f*.

Multistep: la valutazione di coinvolge η_{i+1} e anche le approssimazioni precedenti η_{i-1} η_{i-2}.

Implicita: il valore da calcolare ad ogni passo η_{i+1} compare sia a sinistra dell'uguale che a destra dell'uguale tra gli argomenti della funzione *f*.

Metodo di Eulero implicito: monostep implicita;

Metodo del punto medio: multistep esplicita;

Metodo di Crank-Nicolson: monostep implicita;

Metodo di Heun: monostep esplicita;

Metodo di Eulero modificato: monostep esplicita.

Metodi Predictor-Corrector: si forma di un metodo esplicito, il *predictor*, e un metodo implicito, il *corrector* .

5.4 Errore di discretizzazione globale

L'errore di discretizzazione globale è dato da due contibuti:

Errore di discretizzazione locale: tiene conto dell'errore introdotto dalla discretizzazione della derivata prima;

Errore di propagazione: contiene l'accumulo di tutti gli errori commessi ai passi precedenti del passo $i + 1_{esimo}$;

Metodo consistente: un metodo si definisce consistente se l'errore di discretizzazione locale tende a zero al tendere di *h* a zero;

Metodo stabile: un metodo si definisce stabile se l'errore di propagazione tende a zero al tendere di *h* a zero;

Metodo convergente: un metodo si definisce convergente se al tempo stesso è consistente e stabile.

La scelta del metodo influisce sul risultato finale, è necessario tenere sotto controllo gli errori di troncamento e di arrotondamento e al tempo stesso avere un'elevata precisione relativamente al risultato finale.

Non è immediato quindi scegliere la migliore configurazione relativamente al passo *h* e al metodo da utilizzare.

A volte la riduzione del passo di campionamento e la modifica del metodo di integrazione non bastano per rappresentare il fenomeno fisico, ossia non si verifica nessuna attinenza tra fenomeno fisico e modello numerico.

Le proprietà più importanti degli algoritmi di integrazione numerica sono:

- *stabilità*: relativa alla traiettoria del fenomeno fisico, tiene conto delle condizioni iniziali, se si commettono errori iniziali, tali errori non inficiano il risultato finale; nei moti caotici piccoli errori iniziali determinano grandi variazioni finali, si parla cioè di insatbilità;
- *convergenza*: per $k \to \infty$ la soluzione deve tendere al risultato esatto;
- *algoritmi a passo fisso*: con passi di discretizzazione regolari;
- *algoritmi a passo variabile*: con passi di discretizzazione non regolari, si adatta la frequenza del campionamento alla frequenza del segnale;
- *algoritmi per problemi di tipo stiff*: per questo genere di problemi i primi due algoritmi non vanno bene, sono problemi in cui le variabili evolvono in maniera differente tra loro.

Per uno stesso modello, cambiando il set di parametri cambia la convergenza dei risultati.

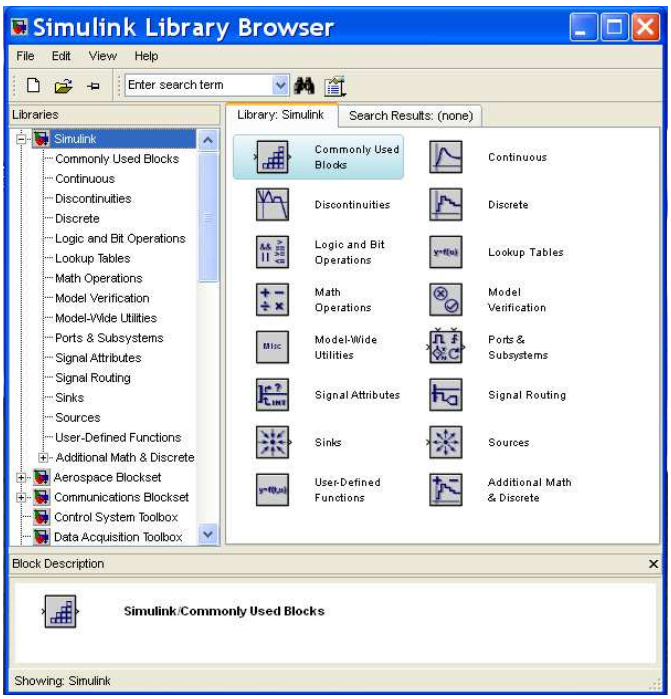
Una qualunque *ODE* di ordine *n* può essere ricondotta ad un sistema di *n* equazioni di ordine 1.

5.5 Generalità: librerie e blocchi elementari in Simulink

Simulink si compone di una serie di librerie che contengono dei blocchi elementari, i quali, opportunamente interconnessi, tramite grafi o rami, andranno a realizzare lo schema a blocchi che rappresenta la funzionalità desiderata.

Si accede alla lista delle librerie aprendo innanzitutto Matlab e quindi aprendo Simulink, tramite l'apposita icona presente sulla barra dei menù di Matlab.

All'apertura di Simulink compare una finestra: *Simulink Library Browser*, è la libreria di Simulink all'interno della quale sono contenuti i blocchi di Simulink suddivisi in sottolibrerie.



Gli oggetti della libreria non sono editabili, per rendere i blocchi funzionanti e modificabili è necessario aprire un foglio di lavoro di Simulink, cliccando sul pulsante *New model* posto a sinistra

nella finestra di apertura di Simulink, comparirà una finestra denominata *Untitled* all'interno della quale si dovrà costruire lo schema a blocchi corrispondente all'operazione desiderata trascinando i blocchi dalle sottolibrerie al foglio di lavoro.

I blocchi più comuni sono compresi in specifiche sottolibrerie:

LIBRERIA	CONTENUTO
Continuous	Componenti lineari a tempo continuo
Discrete	Componenti lineari a tempo discreto
Functions & Tables	Funzionalità programmabili e look-up tables
Math	Funzioni matematiche
Nonlinear	Componenti non lineari
Signals & Systems	Condizionamento di segnali
Sinks	Scope-pozzi. Visualizzatori di segnale
Sources	Sorgenti. Generatori di forme d'onda

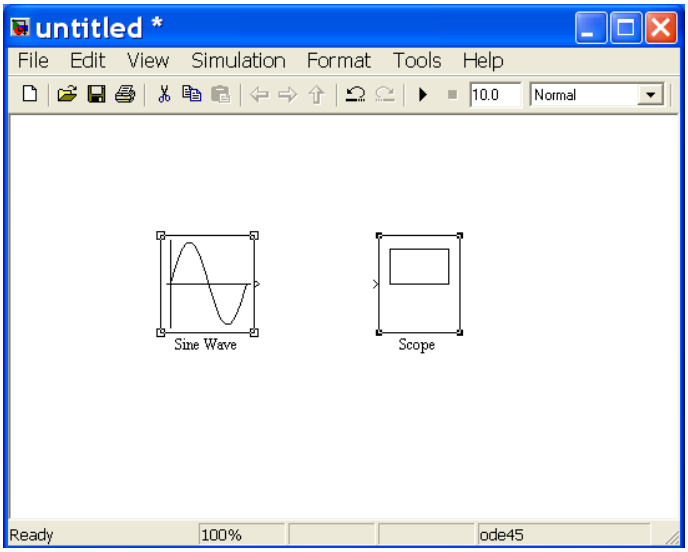
5.6 Visualizzazione di una sinusoide

In generale si devono importare nella pagina di lavoro un certo numero di blocchi elementari realizzanti le funzioni richieste che dovranno essere opportunamente interconnessi tra di loro al fine di realizzare il modello di simulazione.

Per visualizzare una sinusoide sono sufficienti due blocchi elementari: un blocco che generi il segnale desiderato ed un blocco che ne permetta la visualizzazione.

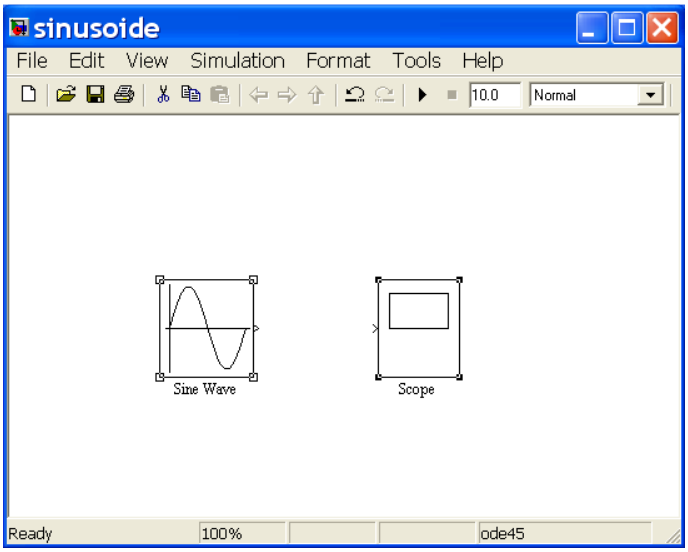
Il primo blocco lo troveremo nella libreria *Source*, blocco *Sine Wave*, il secondo blocco si trova nella libreria *Sinks*, blocco *Scope*.

I blocchi necessari vanno importati nella pagina di lavoro *Untitled* trascinando con il mouse, tramite la funzione *drag-and-drop*, l'icona del blocco all'interno della pagina di lavoro.



A questo punto si salva la pagina di lavoro, la scritta *Untitled* verrà sostituita dal nome assegnato al file.

Il file creato ha estensione *.mdl*, il file creato potrà essere chiamato *sinusoide.mdl*



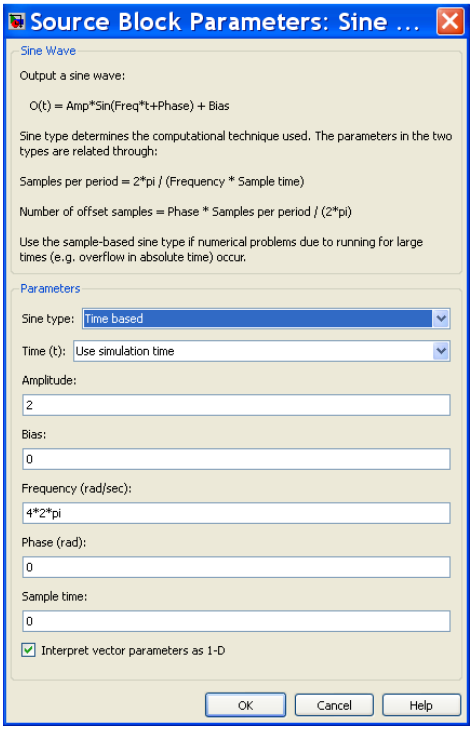
A questo punto i blocchi sono editabili, è possibile modificarne la dimensione, i font, i colori e lo sfondo; inoltre è possibile settare i parametri interni di funzionamento dei blocchi.

Source: *Sine Wave*

Possono essere impostati i parametri di ampiezza, frequenza e sfasamento che definiscono la particolare sinusoide che si desidera generare.

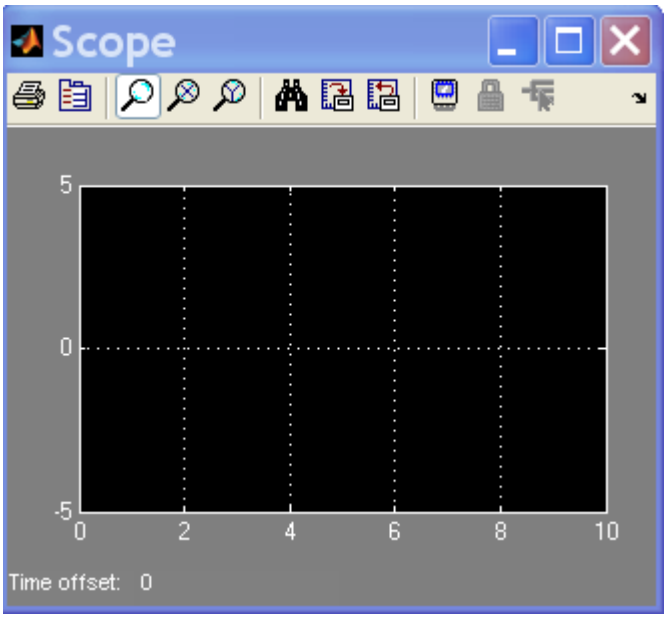
A tal fine è necessario fare doppio click sul blocco *Sine Wave*, e come risultato si apre una finestra di dialogo all'interno della quale vanno impostati i parametri di funzionamento:

- *Amplitude* 2;
- *Frequency* $4\,Hz = 4 \cdot 2\pi\, rad/sec$;



Sinks: *Scope*

Facendo doppio click sul blocco Scope si ottiene come risultato la comparsa della finestra di visualizzazione:



E' possibile scegliere il numero di assi, a seconda del numero di segnali che si vogliono registrare.

Cliccando sul pulsante *Properties*,il secondo da destra, affianco al pulsante di stampa, deselezioniamo nel menù *Data History* la check-box di default *Limit data points to last [5000]*

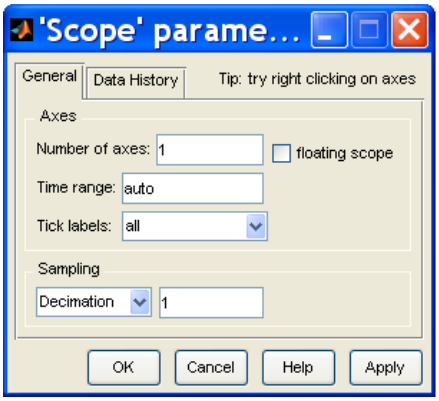
General

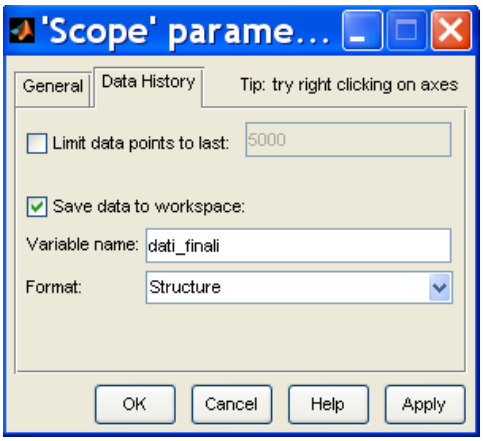
- *Number of axes* 1;
- *Tick labels* *all*;

Data History

- *Save data to workspace* *Variable name: dati_finali*;

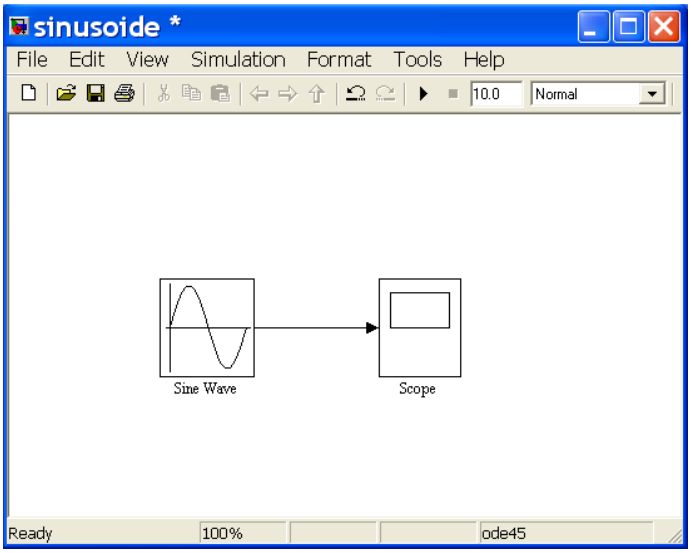
  Format: Structure;





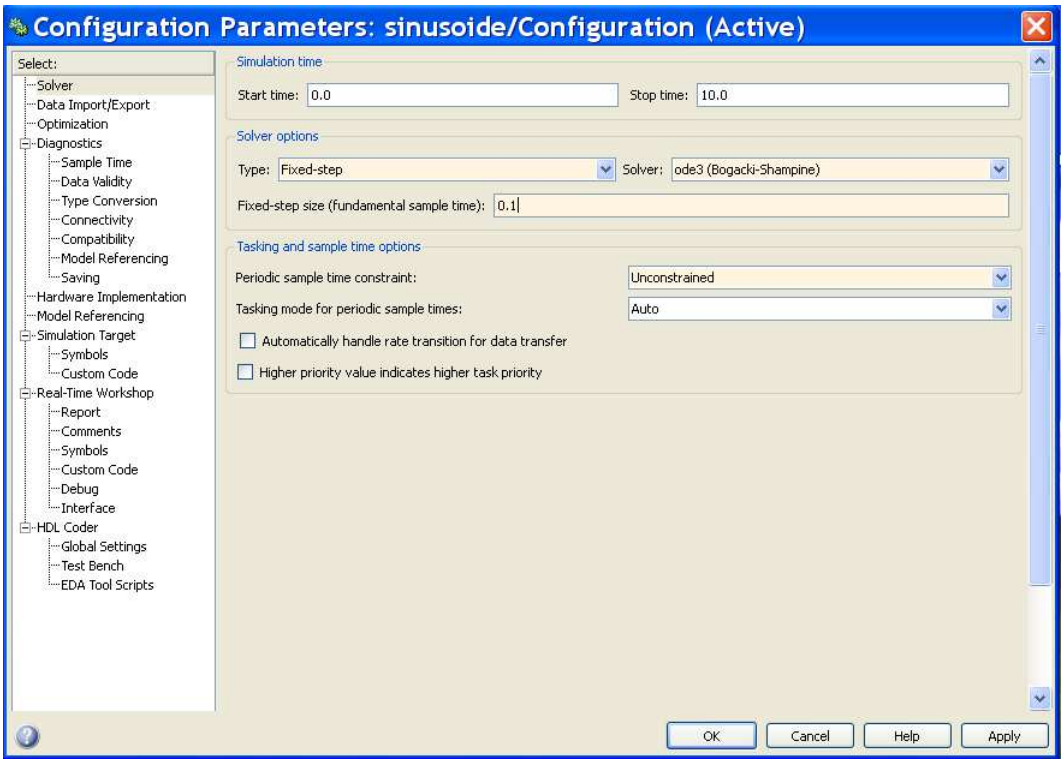
Si deve ora collegare l'uscita del generatore di funzione *Sine Wave* con l'ingresso del blocco di visualizzazione *Scope*.

Per effettuare il collegamento si deve portare la freccia del mouse nel punto di partenza dello stesso, e quindi *tracciare il collegamento* tenendo premuto il tasto sinistro del mouse e portandosi fino al punto di destinazione.



Resta da compiere un ultimo passo prima di avviare la simulazione: si devono impostare la durata, cioè l'intervallo temporale della simulazione, ed il metodo di integrazione numerica.

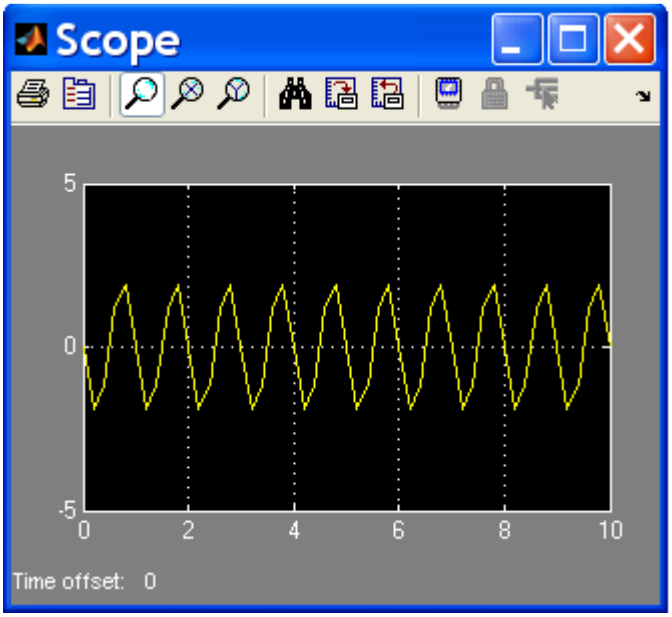
Tali parametri si impostano selezionando la voce *Simulation Parameters* dal menù' *Simulation* della finestra di lavoro, compare la seguente finestra:



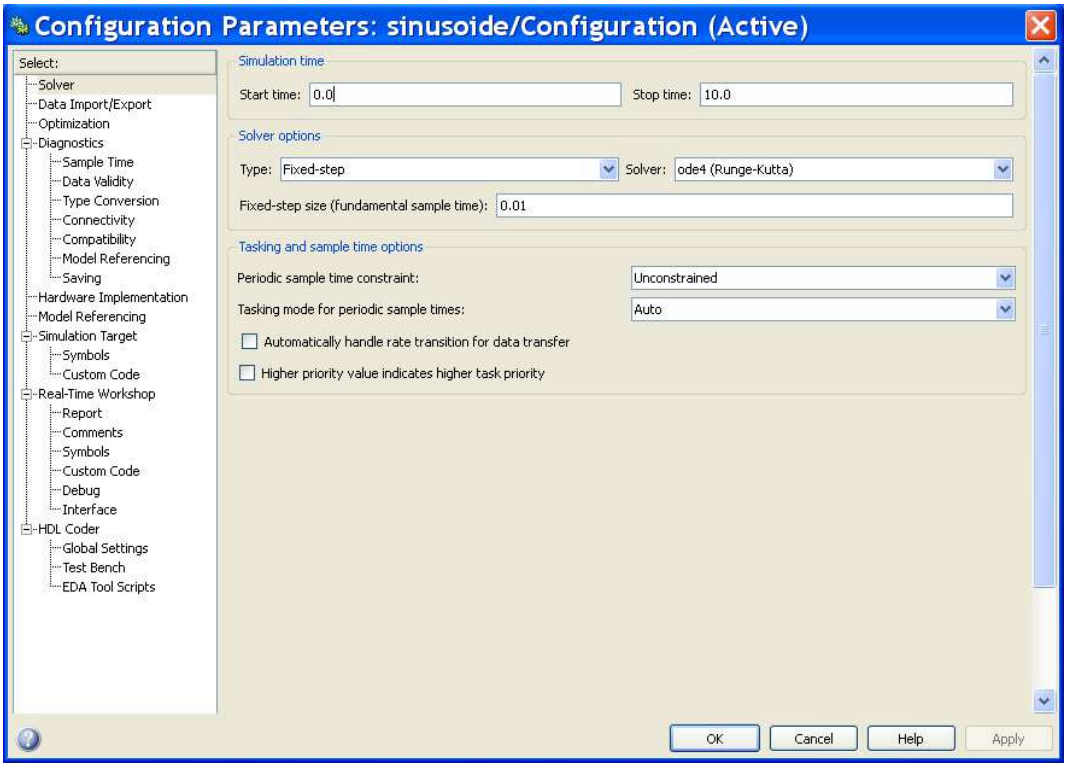
L'intervallo temporale della simulazione si imposta nella casella *Stop Time*.

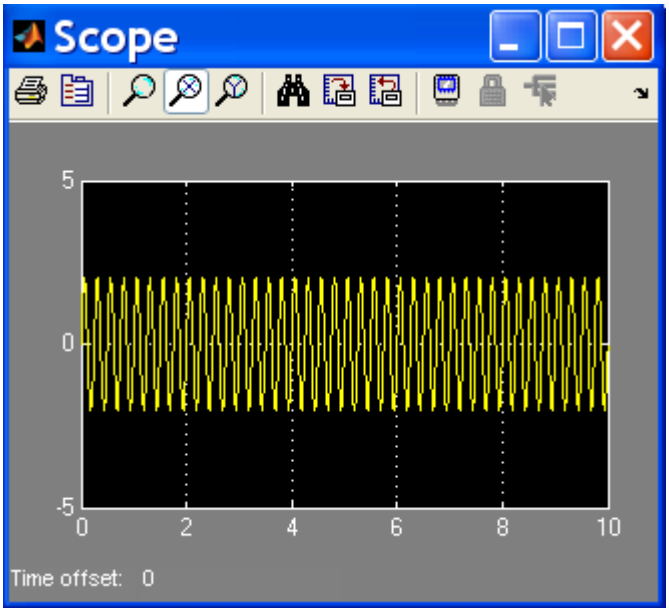
Selezionando le *Solver Options*, in *Fixed step size* si può modificare il passo di campionamento e in *Solver* si può scegliere il metodo di approssimazione.

A questo punto tutto è pronto per eseguire la simulazione, e' sufficiente cliccare il pulsante *Start Simulation* e nella finestra di visualizzazione dello *Scope* comparirà il segnale generato dal blocco:



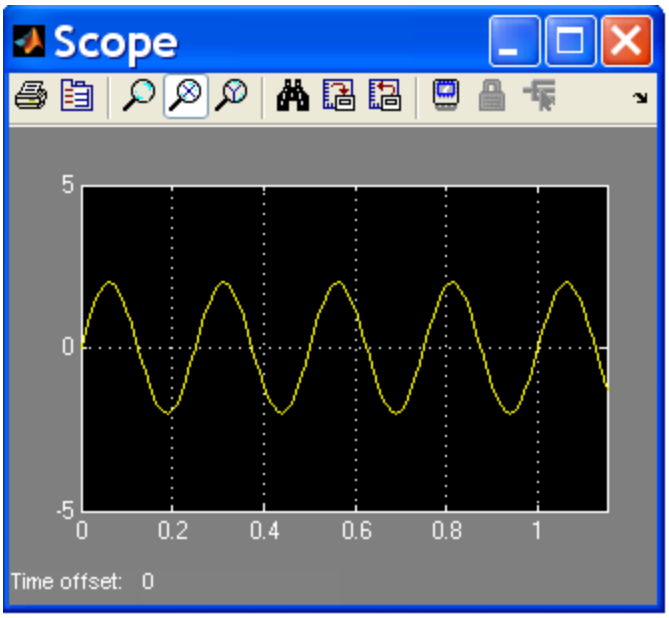
Nella finestra di visualizzazione dello *Scope* compare il segnale generato dal blocco, ma questo non è una sinusoide, in quanto di default Simulink utilizza un passo di campionamento che non soddisfa il teorema del campionamento, pertanto modifichiamo il passo di campionamento e il metodo di risoluzione:





A questo punto il teorema del campionamento è rispettato e abbiamo ottenuto una sinusoide.

Per visualizzare meglio la sinusoide facciamo uno zoom nella finestra di visualizzazione del segnale:



E’ possibile salvare il progetto in un file *prova_simulink.mdl* ed esportare i grafici ottenuti in vari formati, tra cui in formato *.pdf*

6. REALIZZAZIONE DI MODELLI IN SIMULINK

6.1 Equazione differenziale del 1° ordine

Come abbiamo già visto Simulink è un risolutore di equazioni differenziali, in ambito Simulink per soluzione di un equazione differenziale intendiamo semplicemente la visualizzazione della soluzione in un blocco di tipo *Scope*.

Scriviamo un programma per risolvere un'equazione differenziale del 1° ordine:

$$\begin{cases} a_1\dot{y} + a_0 y = b_0 u \\ y(t_0) = y_0 \end{cases}$$

Per risolvere in maniera numerica l'equazione differenziale è necessaria una condizione iniziale.

Per tradurre un'equazione differenziale in termini di una combinazione tra blocchi in Simulink si deve esplicitare l'equazione differenziale rispetto alla derivata di ordine più elevato:

$$\dot{y} = \frac{1}{a_1}(b_0 u - a_0 y)$$

Se integriamo i due membri otteniamo la soluzione $y(x)$.

$\dot{y}$ argomento di un integrale;

u termine noto, deve essere fornito esternamente.

Per risolvere il problema ci servono i seguenti blocchi:

- integratore 1 istanza;
- moltiplicatore 3 istanze;
- somma algebrica 1 istanza;
- variabile sorgente 1 istanza.

Dalla libreria di Simulink trasciniamo sul foglio di lavoro i seguenti blocchi:

LIBRERIA	CONTENUTO
Continuous	*Integrator*
Math Operation	*Gain*
Math Operation	*Sum*
Sources	*Step*
Sinks	*Scope*

Step è una funzione gradino, *Scope* visualizza la variabile $y(x)$ in uscita.

Trasciniamo all'interno del foglio di lavoro i blocchi necessari per lo sviluppo del problema, selezioniamo tutti i blocchi e con il comando *Format* presente nella barra dei menù cambiamo le dimensioni, i font e le caratteristiche grafiche dei blocchi.

Attraverso la funzione *Flip Block* presente all'interno del comando *Format* è possibile ruotare, se

necessario, un blocco.

Le variabili $a_0,\ a_1,\ b_0$ devono essere inizializzate, cioè gli si deve attribuire un valore, all'interno del workspace di Matlab; se non si inizializzano le variabili, unendo i blocchi tramite i grafi e facendo partire la simulazione, il programma fornisce un messaggio di errore evidenziando la necessità di inizializzare le variabili.

$a_0 = 5$

$a_1 = 2$

$b_0 = 3$

In corrispondenza dei grafi, e comunque in generale all'interno del foglio di lavoro di Simulink, è possibile aggiungere, facendo doppio click con il tasto sinistro del mouse, dei testi pro-memoria.

Modifichiamo i parametri di default dei blocchi:

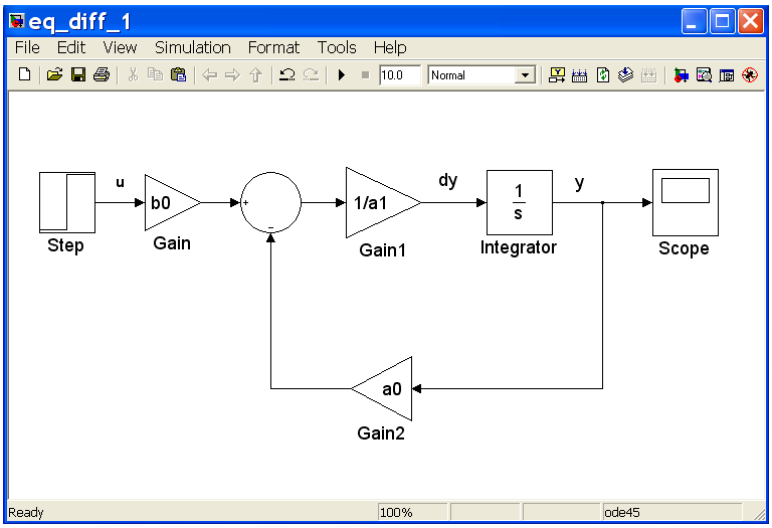
Integrator 0 *Inizial condition*

Gain b_0

Gain1 $1/a_1$

Gain2 a_0

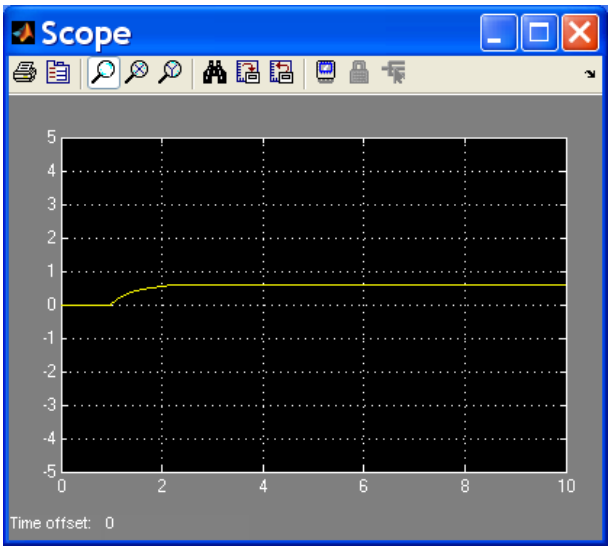
Rinominiamo e salviamo il foglio di lavoro: *eq_diff_1.mdl*



τ: costante di tempo

$\tau = \frac{a_1}{a_0} = \frac{2}{5} = 0.4$ cioè il risultato si ottiene dopo $4 \div 5$ costanti di tempo.

Mandiamo in esecuzione il modello e verifichiamo il segnale in uscita tramite lo *Scope*:



Dal segnale in uscita sembra che la funzione raggiunga il *plateau* in *T=2.5 sec* circa.

$0.4 \cdot 5 = 2$ $\tau = T$ la simulazione sembra corretta.

Inseriamo a questo punto un *Mux*:

LIBRERIA	**CONTENUTO**
Signal Routing	*Mux*

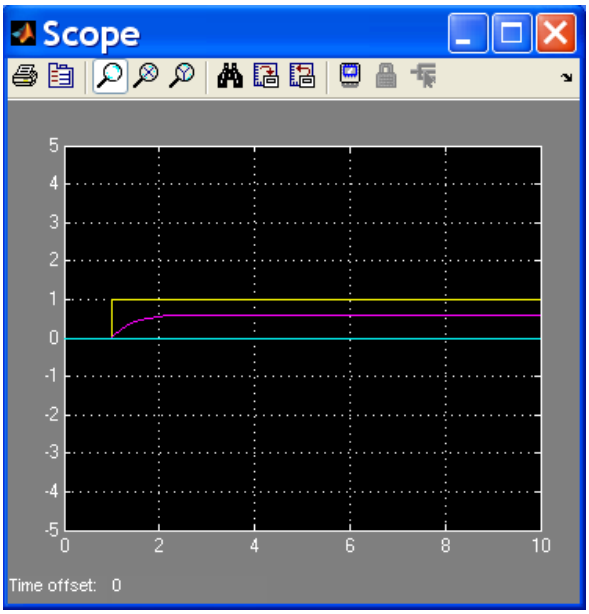
Mux: è un blocco che accetta in ingresso diversi canali ed esce con un solo canale.

Modifichiamo i canali di ingresso del *Mux* e consideriamo tre canali; il segnale in uscita dallo Scope registra 3 segnali, rappresentati da tre curve:

curva giallo: rappresenta il segnale *u* di ingresso che è costante ed è pari ad 1;

curva magenta: rappresenta il segnale della funzione $y(x)$;

curva ciano: -



Inseriamo a questo punto il blocco che definisce la Funzione di trasferimento, *Transfer Fcn*, il cui significato fisico sarà spiegato più avanti; il blocco lo troviamo in:

LIBRERIA	**CONTENUTO**
Continuous	*Transfer Fcn*

Attraverso la *Transfer Fcn* otteniamo una rappresentazione in frequenza dell'equazione differenziale di partenza; sfruttando la trasformata di Laplace si passa dal dominio del tempo al dominio della frequenza, matematicamente si passa da una rappresentazione tramite integrali ad una rappresentazione tramite polinomi nel campo complesso.

Data la funzione:

$a_1\dot{y} + a_0 y = b_0 u$

si ha:

$a_1 x + a_0 = 0$ *Polinomio caratteristico*, eq. caratteristica; si può costruire anche per il secondo membro: $b_0 u \rightarrow b$, $b_0 u + b_1 \dot{u} \rightarrow b_0 + b_1 z$

$(a_1 s + a_0) y(s) = b_0 u(s)$ *Funzione di Trasferimento*

$$y(s) = \frac{b_0}{(a_1 s + a_0)} u(s)$$

$y(s)$ Uscita - rappresentazione nel dominio della frequenza;

$u(s)$ Ingresso - rappresentazione nel dominio del tempo.

Uniamo il blocco relativo alla *Transfer Fcn* alla *u* in ingresso, e al *Mux* in uscita; a questo punto le tre curve in uscite dallo *Scope* sono rappresentative di:

curva giallo: rappresenta il segnale *u* di ingresso che è costante ed è pari ad 1;

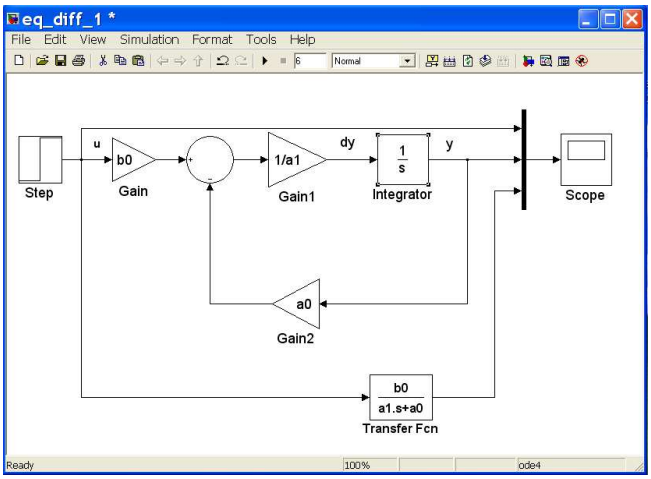
curva magenta: rappresenta il segnale della funzione $y(x)$;

curva ciano: rappresenta il segnale della *Transfer Fcn*;

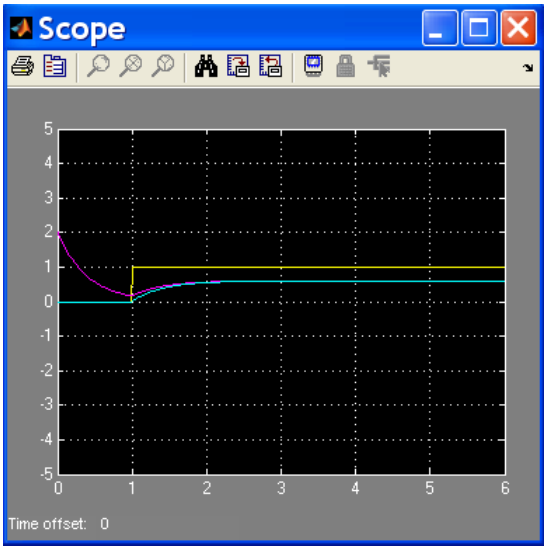
Stop time 6

Integrator 2 *Inizial condition*.

A questo punto la configurazione del modello è la seguente:



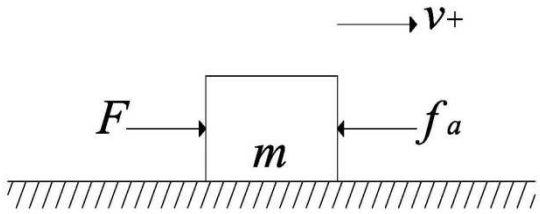
Mandando in esecuzione il modello si ottiene:



A questo punto è possibile fare diverse simulazioni con diversi passi di discretizzazione e con diversi metodi di integrazione; salvando le diverse simulazioni nel formato *Array* dal *Data History* della finestra dei parametri dello *Scope*, è possibile plottare tutte le curve delle varie simulazioni nel workspace di Matlab, in modo tale da apprezzare le variazioni delle curve al cambiare del passo di integrazione e del metodo numerico di risoluzione:

Prova 1:	Type: Variable-step	Solver: ODE45	
	Max step size: 0.1		
	Data History:	Variable name:	Yvar1
		Format:	Array
Prova 2:	Type: Variable-step	Solver: ODE45	
	Max step size: 0.01		
	Data History:	Variable name:	Yvar2
		Format:	Array
Prova 3:	Type: Fixed-step	Solver: ODE1(Euler)	
	Max step size: 0.1		
	Data History:	Variable name:	Yeuler1
		Format:	Array
Prova 4:	Type: Fixed-step	Solver: ODE1(Euler)	
	Max step size: 0.01		
	Data History:	Variable name:	Y euler2
		Format:	Array
Prova 5:	Type: Fixed-step	Solver: ODE4(Runge-Kutta)	
	Max step size: 0.1		
	Data History:	Variable name:	Yrk1
		Format:	Array
Prova 6:	Type: Fixed-step	Solver: ODE4(Runge-Kutta)	
	Max step size: 0.01		
	Data History:	Variable name:	Yrk2
		Format:	Array

L’equazione differenziale sviluppata tramite i blocchi di Simulink ha un significato fisico, può rappresentare la traiettoria di un proiettile o il moto di un corpo di massa *m* spinto da una forza *F* su un piano lubrificato dotato di attrito:



b attrito

$f_a = bv$ forza di attrito

$ma = F - f_a$

$m\dot{v} = F - bv$

$F = m\dot{v} + bv$

La massa può essere in movimento e cioè avere condizioni iniziali non nulle, ad es: *2m/s*.

Le condizioni iniziali è possibile modificarle all’interno dei parametri di settaggio del blocco *Integrator*.

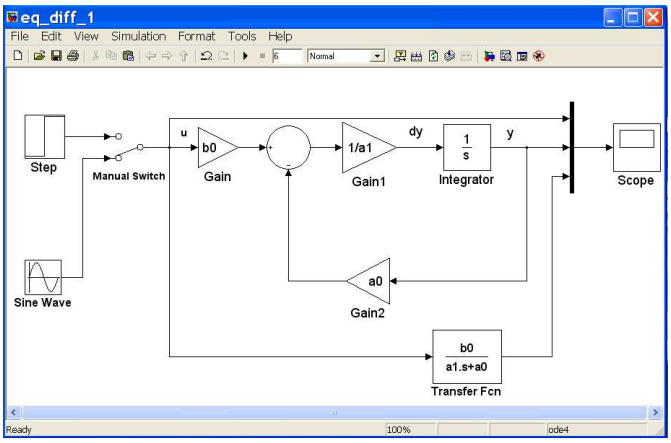
La curva viola, rappresentativa del segnale della funzione $y(x)$ rappresenta la realtà, cioè il caso della massa che si muove con velocità *v* iniziale non nulla; la curva ciano rappresentativa del segnale della *Transfer Fcn* ha condizioni iniziali nulle.

Il plateau della curva viola rappresenta il momento in cui *F* si equilibra con f_a.

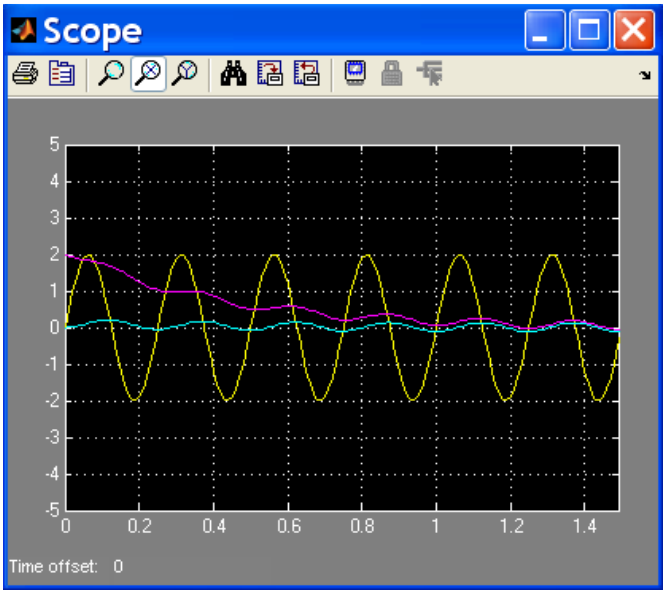
Simulink permette di configurare i parametri di input/output direttamente dalla finestra di *Configuration Parameters* tramite l’opzione *Data Import/Export*.

Oltre ad avere un segnale costante in ingresso è possibile avere in ingresso un segnale sinusoidale, ed eventualmente scegliere l’origine del segnale, procediamo introducendo due nuovi blocchi:

LIBRERIA	**CONTENUTO**
Sources	*Sine Wave*
Signal Routing	*Manual Switch*



In uscita otteniamo:



6.2 Equazione differenziale del 2° ordine

Scriviamo un programma per risolvere un'equazione differenziale del 2° ordine:

$$\begin{cases} 5\ddot{y} + 3\dot{y} + y = 2u \\ y(0) = 1 \\ \dot{y}(0) = -1 \end{cases}$$

Per risolvere in maniera numerica l'equazione differenziale sono necessarie due condizioni iniziali.

$$u = \begin{cases} 1 & t = 0 \div 2 \\ 2 & t = 2 \div 3 \\ 3 & t \geq 3 \end{cases}$$

$$\ddot{y} = \frac{1}{5}(2u - 3\dot{y} - y)$$

Se integriamo due volte i due membri otteniamo la soluzione $y(x)$.

$\ddot{y}$ argomento di un integrale;

$\dot{y}$ argomento di un integrale;

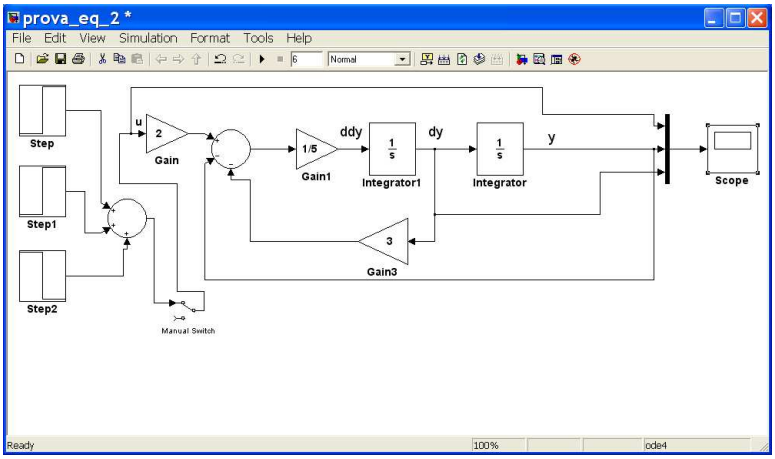
u termine noto, è stato fornito esternamente.

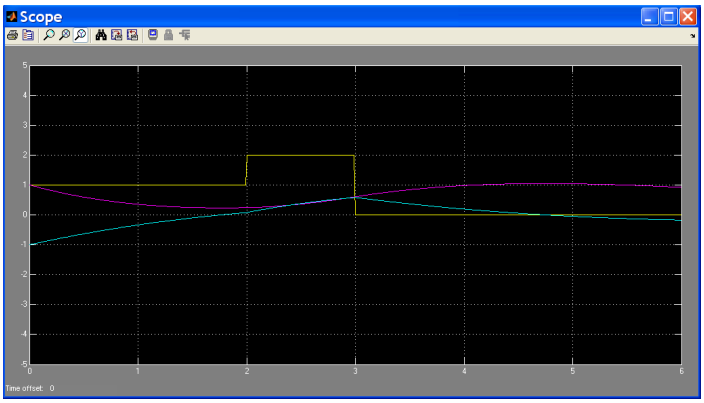
Per risolvere il problema ci servono i seguenti blocchi:

- integratore 2 istanze
- moltiplicatore 3 istanze
- somma algebrica 2 istanza
- variabile sorgente 3 istanza

Dalla libreria di Simulink trasciniamo sul foglio di lavoro i seguenti blocchi:

LIBRERIA	CONTENUTO
Continuous	*Integrator*
Math Operation	*Gain*
Math Operation	*Sum*
Sources	*Step*
Sinks	*Scope*





6.3 Equazione differenziale del 3° ordine

Scriviamo un programma per risolvere un'equazione differenziale del 3° ordine:

$$a_3\dddot{y} + a_2\ddot{y} + a_1\dot{y} + a_0 y = b_0 u$$

La soluzione richiederebbe 3 integratori per ottenere $\dddot{y}$, $\ddot{y}$ e $\dot{y}$, tuttavia proviamo a risolvere in modo differente; l'equazione differenziale di partenza può essere espressa mediante un sistema di 3 equazioni di ordine *n*, risolvere l'equazione differenziale di partenza o il sistema è la stessa cosa:

Consideriamo le seguenti variabili di stato:

$$\begin{cases} x_1 = y \\ x_2 = \dot{y} \\ x_3 = \ddot{y} \end{cases}$$

$$\begin{cases} \dot{x_1} = x_2 \\ \dot{x_2} = x_3 \\ \dot{x_3} = -\dfrac{a_0}{a_3}x_1 - \dfrac{a_1}{a_3}x_2 - \dfrac{a_2}{a_3}x_3 + \dfrac{b_0}{a_3}u \end{cases}$$

Esplicitiamo rispetto alla derivata di ordine più alto:

$$\dddot{y} = -\frac{a_0}{a_3}y - \frac{a_1}{a_3}\dot{y} - \frac{a_2}{a_3}\ddot{y} + \frac{b_0}{a_3}u$$

$\underline{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$ vettore delle 3 variabili di stato

$$\begin{cases} \underline{\dot{x}} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -\dfrac{a_0}{a_3} & -\dfrac{a_1}{a_3} & -\dfrac{a_2}{a_3} \end{bmatrix} \cdot \underline{x} + \begin{bmatrix} 0 \\ 0 \\ \dfrac{b_0}{a_3} \end{bmatrix} \cdot u \\ y = [1\ 0\ 0] \cdot \underline{x} \end{cases}$$

La prima equazione del sistema rappresentala forma $\underline{\dot{x}} = A\underline{x} + B\underline{u}$

La seconda equazione del sistema rappresentala forma $y = C\underline{x} + D\underline{u}$

$$\begin{cases} \underline{\dot{x}} = A\underline{x} + B\underline{u} \\ y = C\underline{x} + D\underline{u} \end{cases} \qquad (*)$$

L'ultimo sistema rappresenta la forma canonica controllata

$x \in \mathbb{R}^n$
$u \in \mathbb{R}^q$
$y \in \mathbb{R}^p$

$p, q \leq n$

$rank\{B\} = q;$

$rank\{C\} = p;$

In Simulink è presente un blocco che rappresenta il sistema (*) e si utilizza per sistemi di equazioni lineari, a volte si usa per sistemi non lineari.

$$\underline{\dot{x}} = A\underline{x} + B\begin{bmatrix} \underline{u} \\ \underline{f}(\underline{x}) \end{bmatrix}$$

$\begin{bmatrix} \underline{u} \\ \underline{f}(\underline{x}) \end{bmatrix}$ parte non lineare

A, *B*, *C* in generale non sono matrici strutturate, nel nostro caso manca la matrice *D*

LIBRERIA	CONTENUTO
Sources	*Repeating Sequence*
Continuous	*State-Space*
Continuous	*Transfer Fcn*
Sinks	*To Workspace*

Lo *Scope* rallenta sull'esecuzione della simulazione perché carica sulla parte grafica è utile per capire come sta procedendo la simulazione.

Inizialmente è bene usare lo *Scope* per visualizzare il comportamento della simulazione, tuttavia terminata la simulazione sostituisco lo *Scope* con il blocco *To Workspace*, tale blocco trasferisce le variabili della simulazione nella memoria di Matlab, da qui è possibile visualizzarle graficamente mediante la funzione `plot`.

Maggiori sono le variabili inserite nel modello maggiore sarà la lentezza della simulazione e quindi del file.

Le variabili possono essere inizializzate all'interno dell'ambiente di lavoro di Simulink oppure possono essere inizializzate dal workspace di Matlab:

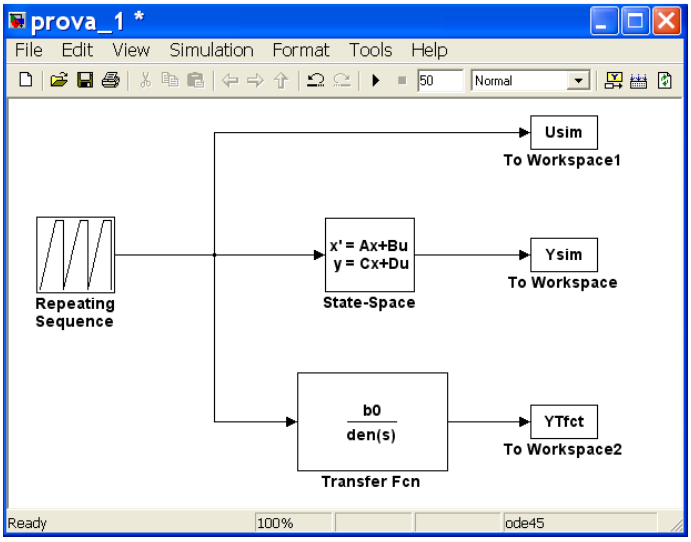
$a_3 = 1$

$a_2 = 3$

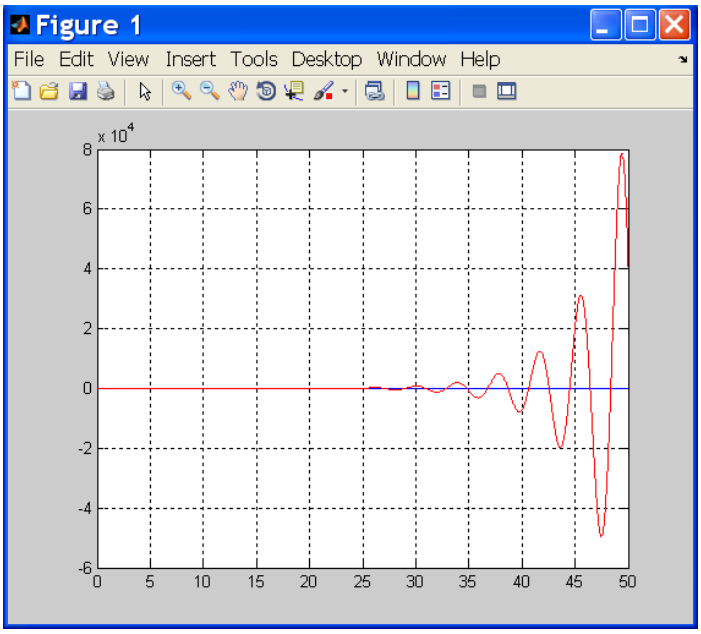
$a_1 = 2$

$a_0 = 4$

$b_0 = 2$




```
plot(tout,Usim,tout,YTfct,tout,Ysim),grid
```



Chiaramente è possibile modificare le condizioni in uscita o in entrata.

6.4 Equazione differenziale lineare a coefficienti costanti

Si visualizzi nei primi 15 secondi la soluzione $y(t)$ dell'equazione differenziale lineare del terzo ordine:

$3\dddot{y} + 2\ddot{y} + 3\dot{y} + 6y = 3 + \cos(2t)$

con condizioni iniziali:

$y(0) = 1$

$\dot{y}(0) = 2$

$\ddot{y}(0) = -3$

La procedura è sempre la stessa:

1. Riscriviamo l'equazione in forma esplicita, cioè isolando a sinistra dell'uguale la derivata di ordine più elevato:

$\dddot{y} = \left(\left[1 + \frac{1}{3}\cos(2t)\right]\right) - \frac{2}{3}\ddot{y} - \dot{y} - 2y$

Il contributo tra parentesi quadre definisce la parte funzione del tempo.

2. Trasciniamo nel foglio di lavoro di Simulink i blocchi necessari per la risoluzione del modello e li colleghiamo tra loro in maniera opportuna.

Dalla libreria di Simulink trasciniamo sul foglio di lavoro i seguenti blocchi:

LIBRERIA	**CONTENUTO**
Continuous	*Integrator*
Math Operation	*Gain*
Math Operation	*Sum*
Sources	*Clock*
User-Defined Functions	*Fcn*
Sinks	*Scope*

Configuriamo il sommatore con 4 nodi positivi e lo rendiamo di forma rettangolare anziché circolare, modifichiamo tali parametri all'interno della finestra di dialogo.

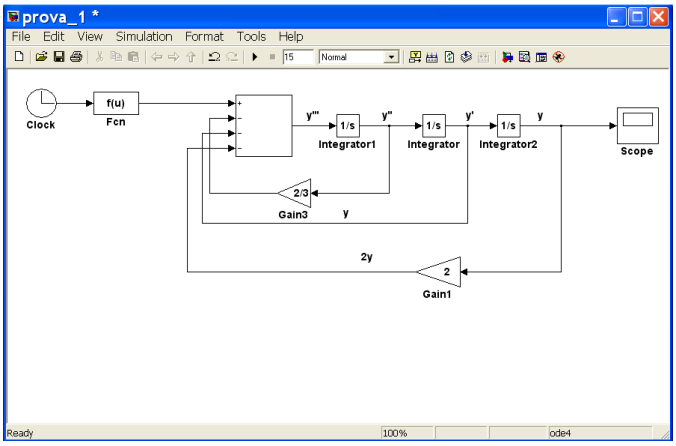
La funzione del tempo viene generata con un blocco *Clock* ed un blocco *Fcn*.

Dopo aver fatto tutti collegamenti bisogna impostare i guadagni dei tre blocchi *Gain* in accordo con l'equazione di partenza.

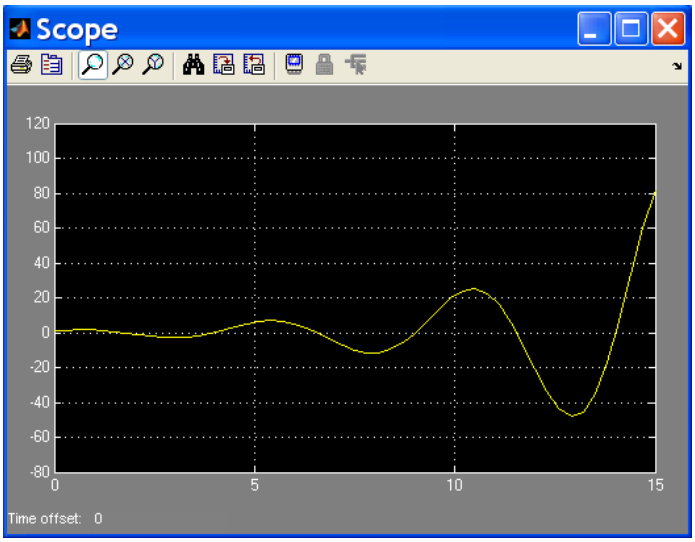
Prima di avviare la simulazione bisogna impostare le condizioni iniziali date dall'equazione differenziale di partenza; le condizioni iniziali si impostano mediante una finestra di dialogo che si apre facendo doppio click sui blocchi *Integrator*.

Non resta che impostare correttamente i Parametri della simulazione *Stop time* = 15 e i parametri dello *Scope* ed avviare la simulazione premendo il pulsante *Start*.

Lo schema finale è:



L'andamento temporale della soluzione cercata è il seguente:



6.5 Equazione differenziale lineare a coefficienti non costanti

Nelle equazioni differenziali a coefficienti non costanti almeno uno dei coefficienti che moltiplicano le derivate della soluzione *y* sono delle funzioni del tempo, anziché essere delle costanti come nell'esercizio precedente.

Si visualizzi nei primi 15 secondi la soluzione *y(t)* dell'equazione differenziale lineare tempovariante del secondo ordine:

$$\ddot{y} + \sin(t)\dot{y} + 4y = 2$$

con condizioni iniziali:

$$y(0) = 1$$

$\dot{y}(0) = -1$

La procedura è sempre la stessa:

1. Riscriviamo l'equazione in forma esplicita, cioè isolando a sinistra dell'uguale la derivata di ordine più elevato:

$\ddot{y} = 2 - \sin(t)\,\dot{y} - 4y$

2. Trasciniamo nel foglio di lavoro di Simulink i blocchi necessari per la risoluzione del modello e li colleghiamo tra loro in maniera opportuna.

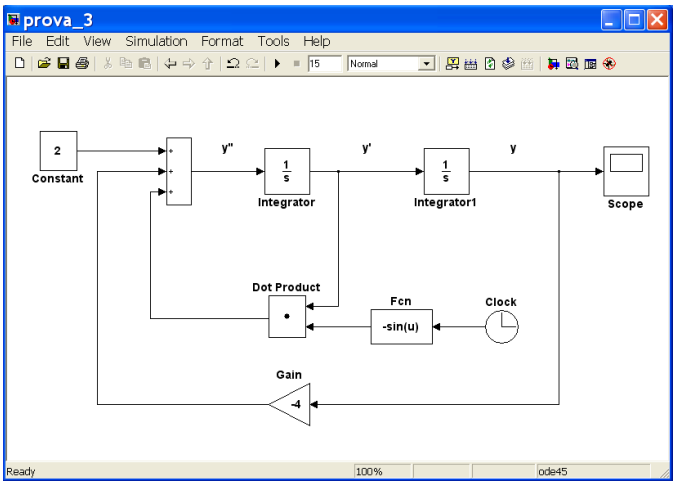
Dalla libreria di Simulink trasciniamo sul foglio di lavoro i seguenti blocchi:

LIBRERIA	**CONTENUTO**
Continuous	*Integrator*
Math Operation	*Gain*
Math Operation	*Sum*
Math Operation	*Dot Product*
Sources	*Clock*
Sources	*Constant*
User-Defined Functions	*Fcn*
Sinks	*Scope*

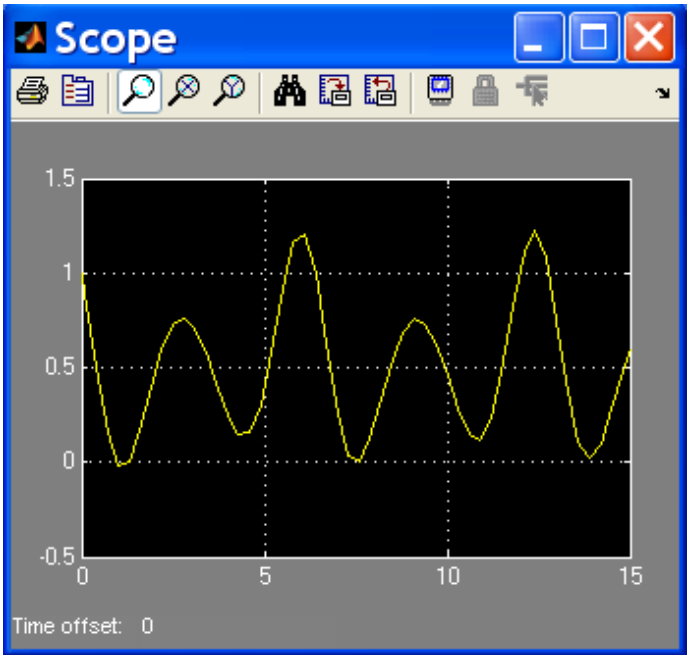
La componente $-\sin(t)\,\dot{y}$ a destra dell'uguale nell'equazione in forma esplicita non potrà più essere generata mediante un blocco di guadagno *Gain*; è possibile utilizzare il blocco *Dot Product*, dalla libreria *Math Operation*, un blocco con due ingressi che fornisce in uscita il prodotto degli stessi.

Al primo ingresso del blocco *Dot Product* si dovrà mandare il segnale $\dot{y}$ prelevato nel punto opportuno dello schema; al secondo ingresso si dovrà invece mandare il guadagno tempovariante $-\sin(t)$, sempre generato con i blocchi *Clock* ed *Fcn* come visto nell'esercizio precedente, nel blocco *Fcn* è inserita la stringa $-\sin(u)$.

Lo schema finale è:



L'andamento temporale della soluzione cercata è il seguente:



6.6 Sistema di equazioni differenziali lineari a coefficienti costanti

Si visualizzino nei primi 30 secondi le soluzioni *y(t)* ed *x(t)* del seguente sistema di equazioni differenziali lineari:

$$\begin{cases} 2\ddot{y} + 4\dot{y} + 6y + x = 3 \\ \dot{x} + \dot{y} + 4x = \cos(t) \end{cases}$$

con condizioni iniziali:

$x(0) = 1$

$y(0) = 1$

$\dot{y}(0) = -1$

Il sistema è complessivamente di ordine 3, l'ordine totale è la somma degli ordini delle singole equazioni.

La procedura è sempre la stessa:

1. Riscriviamo il sistema in forma esplicita:

$$\begin{cases} \ddot{y} = \frac{3}{2} - 2\dot{y} - 3y - \frac{x}{2} \\ \dot{x} = \cos(t) - \dot{y} - 4x \end{cases}$$

2. Trasciniamo nel foglio di lavoro di Simulink i blocchi necessari per la risoluzione del modello e li colleghiamo tra loro in maniera opportuna.

Dalla libreria di Simulink trasciniamo sul foglio di lavoro i seguenti blocchi:

LIBRERIA	**CONTENUTO**
Continuous	*Integrator*
Math Operation	*Gain*
Math Operation	*Sum*
Math Operation	*Dot Product*
Sources	*Clock*
Sources	*Constant*
User-Defined Functions	*Fcn*
Sinks	*Scope*

Si devono collocare in tutto 3 blocchi Integrator che però non dovranno essere disposti in cascata, ma disposti su 2 righe distinte, perché il sistema in esame si compone di 2 equazioni.

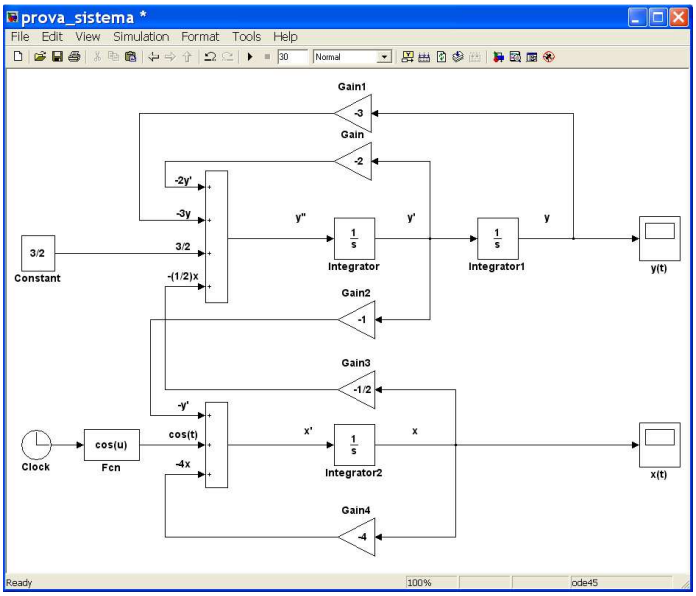
Nella prima riga, corrispondente alla prima equazione mettiamo due integratori in cascata, nella seconda riga metterò il terzo integratore restante.

Analogamente a prima, si collegano due sommatori in ingresso agli integratori più a sinistra, uno dei sommatori deve avere 4 ingressi, per l'altro ne bastano 3.

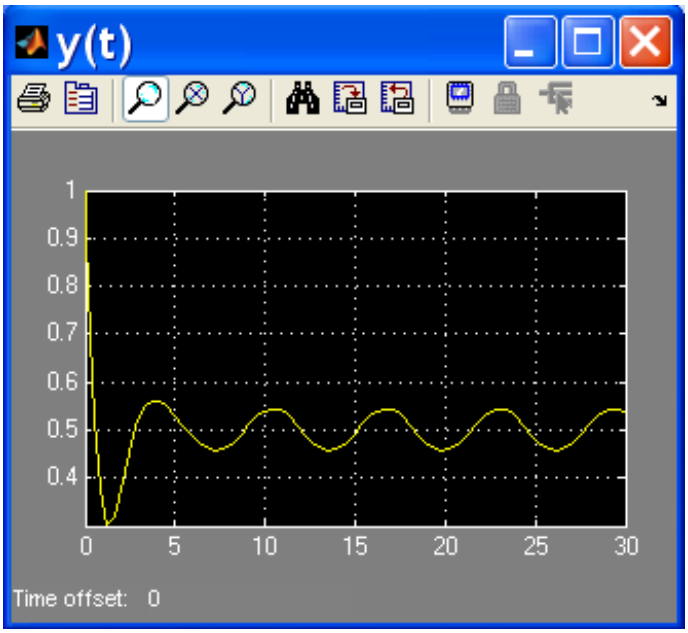
Per concludere lo schema si devono generare tutte le grandezze a destra dell'uguale nelle equazioni (11) e (12), e tali grandezze devono essere "convogliate" al nodo sommatore in accordo con le due equazioni.

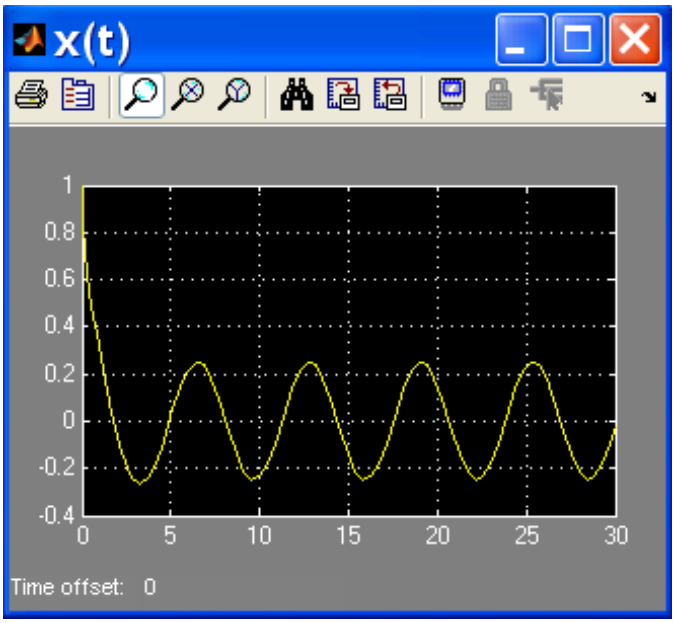
La funzione $\cos(t)$ è generata con i blocchi *Clock* ed *Fcn*, nel blocco *Fcn* è inserita la stringa $\cos(u)$

Lo schema finale è:



I grafici risultanti saranno:





6.7 Equazioni differenziali non lineari del primo ordine

Analizziamo la soluzione di una equazione differenziale non lineare del primo ordine.

Consideriamo per semplicità una equazione autonoma, la variabile tempo non compare esplicitamente:

$\dot{x} = f(x)$

$x(0) = x_0 \qquad\qquad x_0 \in \mathbb{R}$

In gergo matematico tale problema si chiama ***Problema di Cauchy ai valori iniziali***.

Va evidenziato che i grafici che si ottengono con Simulink sono sempre grafici approssimati per due motivi:

- un'equazione differenziale viene risolta via software impiegando algoritmi automatici di calcolo affetti da un intrinseco errore di approssimazione, l'errore si riduce diminuendo il parametro *Fixed step size* nei *Simulation parameters*;
- ogni operazione sui numeri reali effettuata da un programma software è affetta da un errore di troncamento, o arrotondamento dovuto al fatto che un calcolatore rappresenta i numeri con una quantità finita di cifre.

La risoluzione di un problema di Cauchy in ambiente Simulink è estremamente semplice, consideriamo la funzione:

$\dot{x} = -x + \sin(x) + xcos(x)$

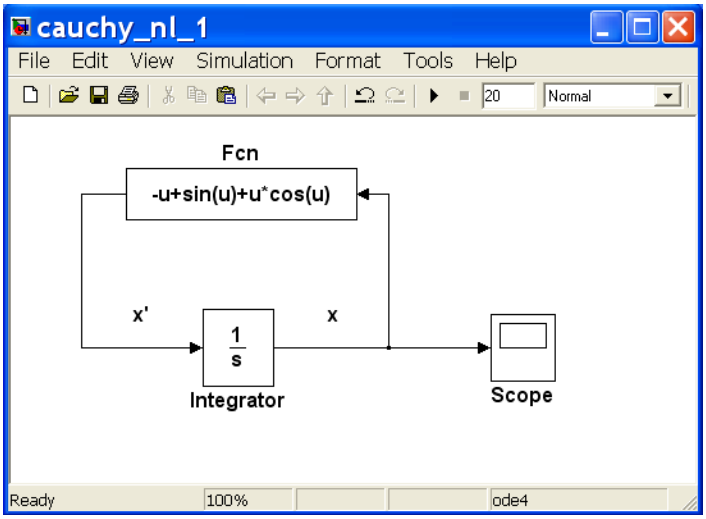
$x(0) = 1$

L'equazione è già in forma esplicita, è sufficiente un blocco *Integrator*, un blocco *Fcn* che realizzi la funzione *f(x)* e un blocco *Scope*.

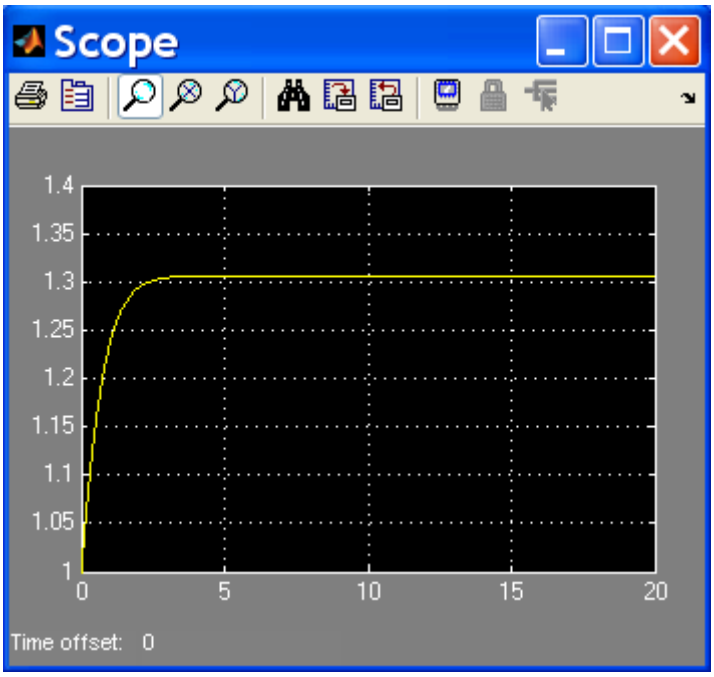
Dalla libreria di Simulink trasciniamo sul foglio di lavoro i seguenti blocchi:

LIBRERIA	CONTENUTO
Continuous	*Integrator*
User-Defined Functions	*Fcn*
Sinks	*Scope*

Lo schema finale è:



Il grafico risultante è:



6.8 Equazioni differenziali non lineari del secondo ordine

Analizziamo la soluzione di una equazione differenziale non lineare del secondo ordine.

Consideriamo per semplicità una equazione autonoma, la variabile tempo non compare esplicitamente:

$\ddot{x} = f(x, \dot{x})$

$x(0) = x_0 \qquad \dot{x}(0) = \dot{x}_0 \qquad x_0, \dot{x}_0 \in \mathbb{R}$

Stavolta servono due integratori, l'equazione è del secondo ordine, collegati in cascata, un blocco *Fcn* che dovrà realizzare la funzione $f(x, \dot{x})$ ed un blocco *Mux* (libreria "Signals & Systems").

Il blocco *Mux* ha un numero arbitrario di ingressi, il numero di ingressi si imposta dalla corrispondente finestra di dialogo attraverso il parametro intero *Number of inputs*.

La funzione del blocco *Mux* è quella di *compattare* tutti i segnali di ingresso in un vettore ad *N* componenti, dove *N = Number of Inputs*; l'utilità di questo blocco consiste nel fatto che se si manda ad un blocco *Fcn* l'uscita di un blocco *Mux* posso scrivere nel blocco *Fcn* una espressione, funzione, che dipende da tutti gli elementi del vettore in uscita dal *Mux*.

Il primo elemento del vettore sarà indicato con *u(1),* il secondo con *u(2)* e così via fino all'elemento *N-esimo*.

Consideriamo la funzione:

$\ddot{x} = -x^2 \cdot \sin(\dot{x}) - 2\dot{x}$

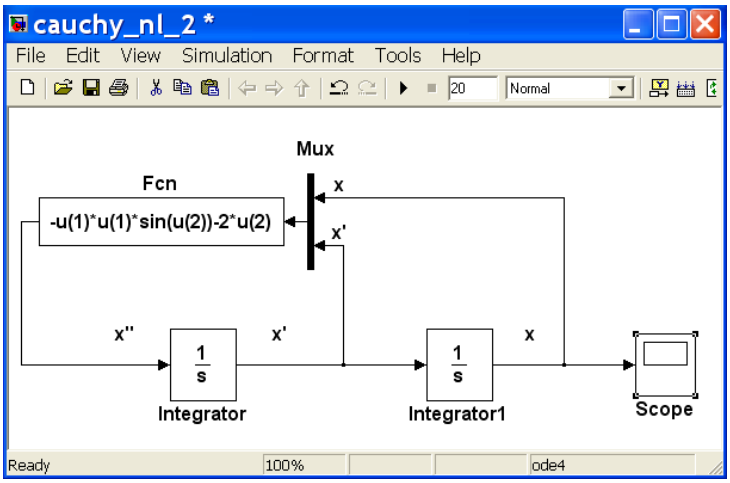
$x(0) = 1 \qquad \dot{x}(0) = 3$

L'equazione è già in forma esplicita, sono sufficienti due blocchi *Integrator*, un blocco *Mux*, un blocco *Fcn* che realizzi la funzione $f(x, \dot{x})$ e un blocco *Scope*.

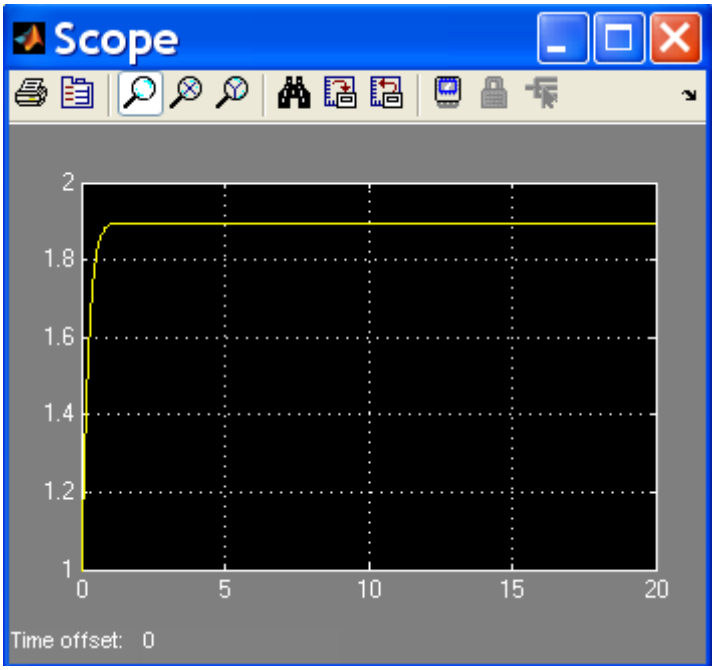
Dalla libreria di Simulink trasciniamo sul foglio di lavoro i seguenti blocchi:

LIBRERIA	**CONTENUTO**
Continuous	*Integrator*
Signal Routing	*Mux*
User-Defined Functions	*Fcn*
Sinks	*Scope*

Lo schema finale è:



Il grafico risultante è:



6.9 Equazioni differenziali non lineari del terzo ordine

Analizziamo la soluzione di una equazione differenziale non lineare del terzo ordine.

Consideriamo la funzione:

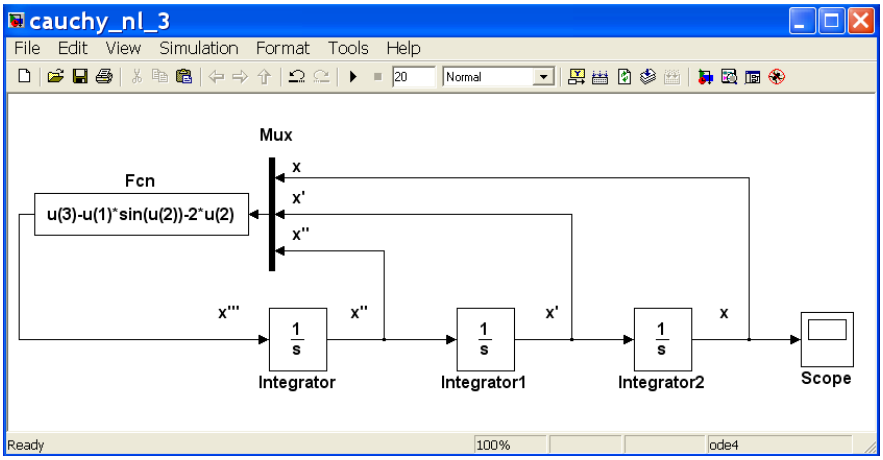
$$\dddot{x} = \ddot{x} - x \cdot \sin(\dot{x}) - 2\dot{x}$$

$$x(0) = 1 \qquad \dot{x}(0) = 3 \qquad \ddot{x}(0) = 3$$

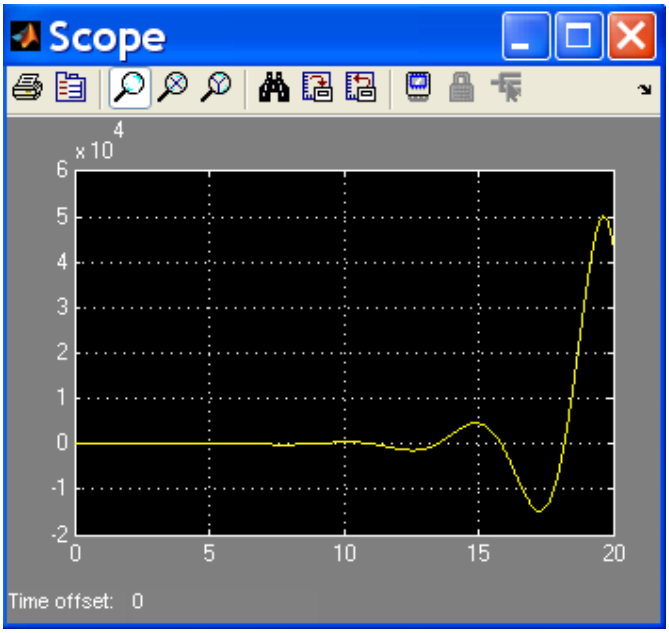
Dalla libreria di Simulink trasciniamo sul foglio di lavoro i seguenti blocchi:

LIBRERIA	CONTENUTO
Continuous	*Integrator*
Signal Routing	*Mux*
User-Defined Functions	*Fcn*
Sinks	*Scope*

Lo schema finale è:



Il grafico risultante è:



6.10 Equazione differenziale non lineare tempo-variante di secondo ordine

Analizziamo la soluzione di un'equazione differenziale non lineare e non autonoma, tempo variante, di secondo ordine in forma esplicita.

L'estensione ai sistemi di ordine superiore è immediata.

$\ddot{x}(t) = f(x(t), \dot{x}(t), t)$

$x(0) = x_0 \qquad \dot{x}(0) = \dot{x}_0 \qquad x_0, \dot{x}_0 \in \mathbb{R}$

E' sufficiente introdurre una semplicissima modifica allo schema relativo all'equazione differenziale del secondo ordine vista precedentemente.

Se si aggiunge un terzo ingresso al blocco *Mux* e si applica a tale ingresso l'uscita del blocco *Clock*, cioè la variabile tempo, si potrà includere nella funzione *f* , cioè nel contenuto del blocco *Fcn* anche la variabile tempo, che sarà indicata con *u(3)*.

Consideriamo la funzione:

$\ddot{x} = -x^2 \cdot \sin(\dot{x}) \cdot \cos(2t) - 2\dot{x}$

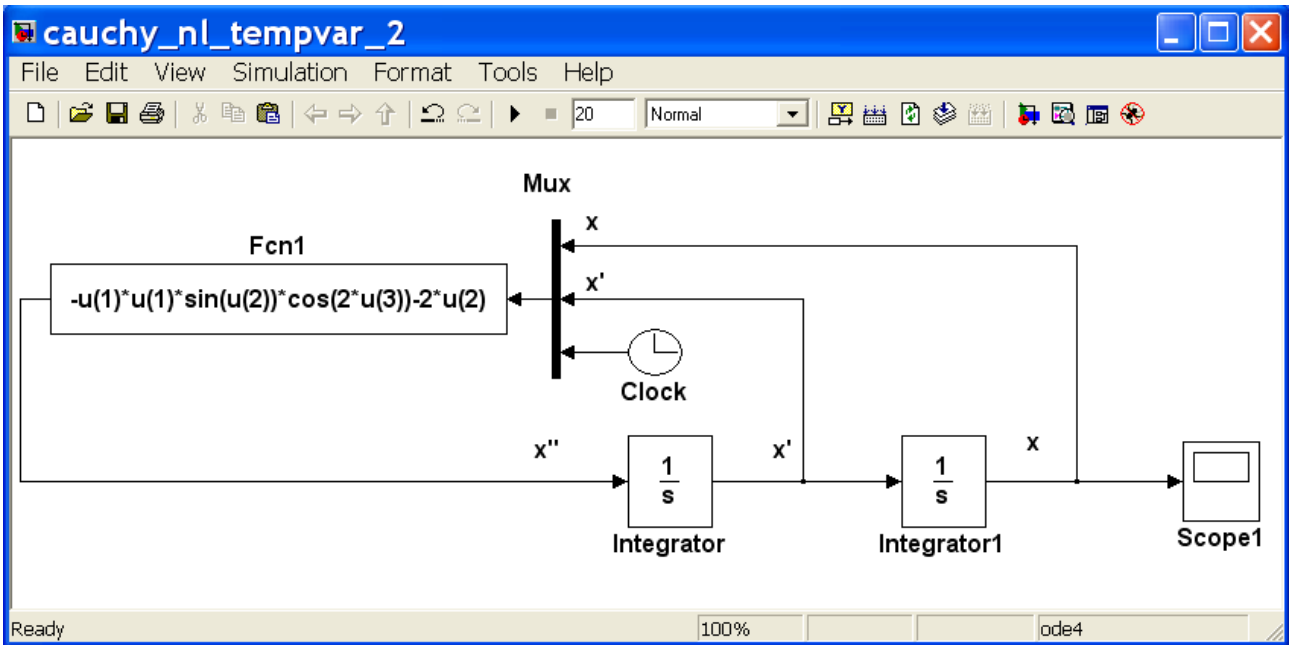
$x(0) = 1 \qquad \dot{x}(0) = 3$

L'equazione è già in forma esplicita, sono sufficienti due blocchi *Integrator*, un blocco *Mux*, un blocco *Fcn* che realizzi la funzione $f(x(t), \dot{x}(t), t)$ e un blocco *Scope*.

Dalla libreria di Simulink trasciniamo sul foglio di lavoro i seguenti blocchi:

LIBRERIA	**CONTENUTO**
Continuous	*Integrator*
Sources	*Clock*
Signal Routing	*Mux*
User-Defined Functions	*Fcn*
Sinks	*Scope*

Lo schema finale è:



Il grafico risultante è: