INTRODUZIONE A MATLAB E SIMULINK

Vincenzo LIPPIELLO

www.docenti.unina.it/lippiell

vincenzo.lippiello@unina.it

PRISMA Lab

Dipartimento di Ingegneria Elettrica e Tecnologie dell'Informazione

Università degli Studi di Napoli Federico II

www.prisma.unina.it

Introduzione / 1

- Il programma MATLAB è nato principalmente come programma destinato alla gestione di matrici. Le versioni successive sono state completate con serie di funzioni che permettono le più complesse analisi numeriche, adatte ad esempio all'analisi e alla soluzione di problemi di controllo.

Introduzione / 2

- La linea di comando di MATLAB è indicata da un prompt come in DOS: >> . Accetta dichiarazioni di variabili, espressioni e chiamate a tutte le funzioni disponibili nel programma. Tutte le funzioni di MATLAB non sono altro che files di testo, simili a quelli che l'utente può generare con un text editor, e vengono eseguite semplicemente digitandone il nome sulla linea di comando. MATLAB permette inoltre di richiamare le ultime righe di comandi inseriti usando le frecce in alto e in basso.

Introduzione / 3

- Help di MATLAB
  - MATLAB presenta un help in linea con informazioni sulla sintassi di tutte le funzioni disponibili.
  - Per accedere a queste informazioni, basta digitare:
    - *help nome_funzione*
  - È anche possibile avere un help di tutte le funzioni di una certa categoria; ad esempio per sapere quali sono le funzioni specifiche per l'analisi ed il controllo di sistemi dinamici, basta digitare:
    - `help control`
  - Per sapere quali sono le varie categorie di funzioni disponibili (i cosiddetti toolbox), basta digitare:
    - *help*
  - Per accedere alla home-page della guida digitare:
    - *doc*

Introduzione / 4

- I files interpretati dal programma sono file di testo ASCII con estensione `.m` ; sono generati con un text editor e sono eseguiti in MATLAB semplicemente digitandone il nome sulla linea di comando (senza estensione!).
- È possibile inserire dei commenti al loro interno precedendo ogni linea di commento col percento %
- *Attenzione!* Può essere molto utile andare nelle directories dove si trova il programma ed analizzare come le varie funzioni sono state implementate. Ciò è possibile poiché ogni funzione ed ogni comando MATLAB richiama un file `.m`

Punteggiatura e variabili / 1

- Le istruzioni (siano esse contenute in un file .m lanciato da MATLAB, oppure digitate direttamente dalla linea di comando) vanno sempre terminate con un punto e virgola, altrimenti è visualizzato il risultato dell'applicazione dell'istruzione
  - Es: `var1=6;`
  - Es: `var2=linspace(-10,10,10000);`

Punteggiatura e variabili / 2

- Per visualizzare il contenuto di una variabile è sufficiente digitarne il nome senza punto e virgola sulla linea di comando.
- Tutti i calcoli effettuati in MATLAB sono eseguiti in doppia precisione, ma si possono visualizzare in un formato diverso usando i comandi:
  - `format short` Virgola fissa con 4 decimali
  - `format long` Virgola fissa con 15 decimali
  - `format short e` Notazione scientifica 4 dec.
  - `format long e` Notazione scientifica 15 dec.
- Il risultato dell'ultima operazione è memorizzato nella variabile `ans`.

Operatori scalari

- Gli operatori disponibili sono:
  - `+, -, *, /, ^`
  - `sin, cos, tan` (radianti)
  - `sind, cosd, tand` (gradi)
  - `asin, acos, atan`
  - `exp, log` (naturale), `log10` (in base 10)
  - `abs, sqrt, sign`

Numeri complessi

- L'unità complessa è `i` o `j` ed è predefinita
  - NON usare `i` o `j` come variabili o indici nei cicli se dovesse servirvi l'unità complessa
- Un numero complesso si scrive nella forma `a+j*b`
  - Es: `z=2+j*3`
- Operatori applicabili a numeri complessi:
  - `abs` : modulo, es. `abs(z)`
  - `angle` : fase, es. `angle(z)`
  - `real` : parte reale, es. `real(z)`
  - `imag` : parte immaginaria, es. `imag(z)`

Matrici e loro operatori / 1

- L'inserimento di un vettore o di una matrice in generale viene effettuato tra parentesi quadre, separando gli elementi delle righe con spazi o virgole, e le diverse righe con punti e virgola (oppure andando a capo ad ogni nuova riga)
  - Es. di vettore riga: `x = [1, 2, 3];`
  - Es. di vettore colonna: `y = [1; 4; 7];`
  - Es. di matrice: `A = [1 2 3; 4 5 6; 7 8 9];`
    oppure:
    ```
    A = [1 2 3
         4 5 6
         7 8 9 ];
    ```

Matrici e loro operatori / 2

- Per far riferimento agli elementi di una matrice A
  - l'elemento a_{mn} è indirizzato come `A(m,n);`
    - es. `A(2,3)` dà 6
  - la riga m-esima è indirizzata come `A(m,:)`, dove tutte le colonne sono indicate con due punti;
    - es. `A(2,:)` dà [4 5 6]
  - la colonna n-esima è indirizzata come `A(:,n)`, dove tutte le righe sono indicate con due punti;
    - es. `A(:,3)` dà [3; 6; 9]
  - la sottomatrice avente elementi a_{mn}, con $m_1<m<m_2$ e $n_1<n<n_2$, è indirizzata come A(m1:m2,n1:n2);
    - es. `A(1:2,2:3)` dà [2, 3; 5, 6]

Matrici e loro operatori / 3

- Gli operatori applicabili a matrici sono
  - `+ - * ^ / \ '`
  - Divisione a sinistra:`A\B=inv(A)*B`
  - Divisione a destra: `B/A=B*inv(A)`

Matrici e loro operatori / 4

- Altre funzioni operanti essenzialmente su vettori (riga o colonna) sono
  - `max, min`
  - `sort`
  - `sum, prod`
  - `median`
- Esistono poi particolari operatori (`.*, ./, .^`) che permettono di effettuare operazioni su vettori elemento per elemento, senza ricorrere a cicli. Ad esempio, se x è un vettore, per moltiplicare elemento per elemento i due vettori sin(x) e cos(x) basta fare
  - `y = sin(x) .* cos(x);`

Matrici e loro operatori / 5

- Altre funzioni che operano invece essenzialmente su matrici sono
  - `inv`
  - `det`
  - `size`
  - `rank`
  - `eig`
- Attenzione: tutte le funzioni che operano su matrici hanno dei vincoli sugli operandi introdotti. Ad esempio non si può invertire una matrice non quadrata. Per ulteriori spiegazioni sulla sintassi della funzione utilizzare il comando `help`.

Matrici e loro operatori / 6

- La funzione `eig` opera su matrici quadrate nel modo seguente
  - `y=eig(A);` produce un vettore y contenente gli autovalori della matrice A
  - `[U,D]=eig(A);` produce una matrice U avente per colonne gli autovettori della matrice A, ed una matrice D diagonale avente sulla stessa gli autovalori della matrice A

Matrici e loro operatori / 7

- Esistono poi varie funzioni predefinite per la creazione di matrici
  - `eye(n)` : matrice identità n righe n colonne
  - `zeros(m,n)`: matrice di 0 con m righe e n colonne
  - `ones(m,n)` : matrice di 1 con m righe e n colonne
  - `rand(m,n)` : matrice casuale di valori tra 0 e 1
  - `diag(X)`
    - se *X* è un vettore con *n* elementi, produce una matrice quadrata diagonale di dimensione *n* per *n* con gli elementi di *X* sulla diagonale
    - se invece *X* è una matrice quadrata di dimensione *n* per *n*, produce un vettore di *n* elementi pari a quelli sulla diagonale di *X*

Matrici e loro operatori / 8

- Il comando : può essere usato per generare vettori
  - senza specificare incremento
    - es. `t=1:5 => t=[1 2 3 4 5]`
  - con incremento positivo specificato
    - es. `t=0:0.2:1 => t=[0 0.2 0.4 0.6 0.8 1]`
  - con incremento negativo specificato
    - es. `t=2:-0.2:1 => t=[2 1.8 1.6 1.4 1.2 1]`

Polinomi / 1

- Alcune funzioni che verranno elencate in seguito necessitano di polinomi come parametri d'ingresso.
- MATLAB tratta i polinomi come particolari vettori riga, i cui elementi sono i coefficienti dei monomi del polinomio in ordine di potenza decrescente.
  - Es. il polinomio $s^4+111s^3+1100s^2+1000s+4$ viene rappresentato come: `num=[1 111 1100 1000 4];`

Polinomi / 2

- La funzione `conv` moltiplica due vettori e quindi due polinomi.
  - Es. il prodotto tra polinomi $(s^2+s+1)*(s^2+111s+1000)$ viene effettuato con:
    `prod = conv([1 1 1],[1 111 1000])` che dà come risultato il vettore `[1 112 1112 1111 1000]`

Polinomi / 3

- La funzione `roots` calcola le radici del polinomio
  - Es: `p = [1 3 5 2]; r = roots(p);`
    In `r` sono memorizzate le radici del polinomio `p`
- La funzione inversa è la funzione `poly`
  - `pp = poly(r);`
  - In `pp` viene ripristinato il polinomio originale `p`

Polinomi / 4

- La funzione `residue` calcola residui e poli di una funzione di trasferimento.
  - Es: `num = [1 30 1000]; den = conv([1 1 1],[1 111 1000]);`
    `[r,p,k] = residue(num,den);`
  - In `r` sono memorizzati i residui della funzione, in `p` i poli ed in `k` l'eventuale termine aggiuntivo nel caso in cui il numero di zeri sia maggiore o uguale al numero di poli della funzione.

Istruzione IF – THEN - ELSE

- La forma generale del costrutto IF-THEN-ELSE è la stessa di un qualsiasi linguaggio di programmazione
  - ```
    if      condizione1,
           operazioni1;
    elseif  condizione2,
          operazioni2;
    else
          operazioni3;
    end;
    ```
    ```

Istruzione IF – THEN – ELSE / 2

- `Condizione1,2` devono essere condizioni che restituiscono come risultato VERO o FALSO. Gli operatori disponibili per tali confronti sono

`<, >`	
`<=, >=`	
`==`	uguale
`~=`	diverso
`&`	and logico
`|`	or logico
`~`	not logico

Istruzioni per cicli / 1

- I cicli si possono fare con due diversi costrutti:
  - ```
    for k = 1:step:n,
     operazioni;
    end;
    ```
- Il ciclo esegue le operazioni incrementando la variabile `k` da `1` a `n` con il passo indicato in `step`
    ```

Istruzioni per cicli / 2

- Oppure
  - ```
    while condizione
     operazioni;
    end;
    ```
- Il ciclo esegue le operazioni fino a che la condizione è verificata. La condizione viene costruita con le stesse regole (vincoli ed operatori) di quella dell'IF-THEN-ELSE.
- Attenzione: prevedere una inizializzazione prima del ciclo che verifichi la condizione per far sì che il programma entri nel ciclo, ed inoltre inserire nelle operazioni qualcosa che possa interagire e quindi modificare la condizione, altrimenti il ciclo sarà ripetuto all'infinito.
    ```

Grafici / 1

- La funzione `plot` crea grafici bidimensionali: riceve in ingresso due vettori della stessa lunghezza e stampa i punti corrispondenti alle coordinate fornite dai due vettori. Ad esempio se si hanno due vettori `x` e `y`, il grafico corrispondente si ottiene come
  - `plot(x,y);`

Grafici / 2

- Per tracciare il grafico di una qualsiasi funzione, è perciò necessario crearsi un opportuno vettore da usare come ascisse, passarlo alla funzione per ricavare un vettore contenente le ordinate, ed usare la funzione `plot` sui due vettori così ottenuti. Ad esempio per tracciare la funzione `sin(x)` tra `-4` e `4` si può usare la serie di comandi
  - ```
    x=-4:0.01:4;
    y=sin(x);
    plot(x,y);
    ```
    ```

Grafici / 3

- Se si usa la funzione `plot` con un solo parametro complesso, il grafico rappresenterà la parte reale e la parte immaginaria degli elementi del vettore
  - `plot(y);`
    con `y` complesso, equivale a
    `plot(real(y),imag(y));`

Grafici / 4

- Per creare grafici di colori diversi o usando caratteri diversi dal punto si può specificare dopo le coordinate una stringa di 2 elementi. Il primo è il colore del grafico, il secondo il simbolo usato per contrassegnare i punti. Ad es.
  - `plot(x,y,'g+');`
  - Crea un grafico in verde usando dei + al posto dei punti. Questa opzione può essere usata nei casi di grafici sovrapposti da stampare (se la stampante a disposizione non è a colori e se non si cambia il tipo di simbolo, non si capisce più nulla ...)
- Per stampare più grafici contemporaneamente sugli stessi assi, allora usare il comando `hold on` e continuare con i vari plot, oppure
  - `plot(x,y,'g+',x,z,'r-')`

- L'insieme delle scelte possibili è il seguente

r	red	.	point
g	green	o	circle
b	blue	x	x-mark
w	white	+	plus
m	magenta	*	star
c	cyan	-	solid
y	yellow	:	dotted
k	black	--	dashed
		-.	dash-dot

Grafici / 6

- Altri comandi sono
  - `grid` : sovrappone al grafico un grigliato
  - `title` : aggiunge un titolo del disegno
  - `xlabel` : aggiunge una legenda per l'asse x
  - `ylabel` : aggiunge una legenda per l'asse y
  - `axis` : imposta la scala degli assi del grafico
  - `clf` : cancella il grafico corrente
- Il comando `figure` crea una nuova finestra grafica in cui far comparire il disegno; per spostarsi sulla *n*-esima finestra grafica, basta digitare `figure(n)`

- Per visualizzare più grafici sulla stessa schermata, ma con assi differenti, si può usare la funzione `subplot`. La funzione vuole 3 parametri:
  - il primo indica in quante parti verticali dividere lo schermo,
  - il secondo in quante parti orizzontali,
  - il terzo in quale parte eseguire il plot successivo.
    - ```
      subplot(2,1,1), plot(funz1);
      subplot(2,1,2), plot(funz2);
      ```
    - crea due finestre divise da una linea orizzontale, e visualizza in quella alta il grafico di `funz1`, e in quella bassa quello di `funz2`.
      ```

- Le due funzioni che possono essere utilizzate per creare vettori equispaziati o comunque tipici per gli assi delle ascisse sono
  - `x = linspace(0.01,100,1000);`
  - `x = logspace(-2,2,1000);`
- La `linspace` crea un vettore `x` di `1000` elementi compreso tra `0.01` e `100` separati linearmente.
- La `logspace` crea lo stesso vettore, con elementi separati logaritmicamente. Si osservi che i primi due parametri sono gli esponenti degli estremi dell'intervallo espressi in base `10`.

- Esempio di sovrapposizione di grafici

```
x=-2*pi:0.01:2*pi;
y=sin(x);
z=cos(x);
figure(1)
plot(x,y,'r',x,z,'b:');
figure(2)
plot(x,y,'r');
hold on
plot(x,z,'b:');
hold off
```

Grafici / 10

- La funzione `semilogx` genera grafici con scala delle ascisse logaritmica in base `10`. La sintassi è identica a quella della funzione `plot`
- La funzione `semilogy` genera grafici con scala delle ordinate logaritmica in base `10`
- La funzione `loglog` genera grafici con entrambe le scale logaritmiche in base `10`

Definizione di nuove funzioni / 1

- In MATLAB è possibile creare nuove funzioni. Basta creare un file con estensione `.m` e nome del file uguale a quella della funzione desiderata
- La prima riga del file deve contenere il nome della funzione e gli argomenti di ingresso e di uscita.
  - `function z = fun1(a,b)`
  - `function [x,y] = fun2(a,b)`
  - Risulta che `fun1` e `fun2` sono nomi di funzioni; `a` e `b` sono argomenti d'ingresso; `x`, `y` e `z` sono argomenti d'uscita

Definizione di nuove funzioni / 2

- Il blocco di linee di commento consecutive che eventualmente segue la prima linea del file viene visualizzato digitando il comando `help` seguito dal nome della funzione creata
- Le variabili utilizzate in una funzione sono locali e quindi indipendenti da quelle dell'ambiente chiamante
- È possibile utilizzare anche variabili globali, a patto che vengano definite come tali sia nell'ambiente chiamante sia nella funzione, utilizzando il comando `global` seguito dai nomi delle variabili, separati da spazi.
  - `global F G H`

Forme di rappresentazione del modello / 1

- Rappresentazione Guadagno-Zeri-Poli
  - La funzione `sys = zpk(z,p,k)` crea un modello guadagno-zeri-poli
  - Esempio
    - ```
      z = [];
      p = [0.1+i 0.1-i];
      k = [2];
      sys = zpk(z,p,k);
      ```

      corrisponde alla funzione di trasferimento

$$\frac{2}{(s-0.1+j)(s-0.1-j)}$$
      ```

Forme di rappresentazione del modello / 2

- Rappresentazione polinomiale
  - La funzione `sys = tf(num,den)` crea un modello con rappresentazione polinomiale della funzione di trasferimento
  - Esempio
    - ```
      num = [1 1];
      den = [1 2 2];
      sys = tf(num,den);
      ```

      corrisponde alla funzione di trasferimento

$$\frac{s+1}{s^2+2s+2}$$
      ```

Forme di rappresentazione del modello / 3

- Rappresentazione ingresso-stato-uscita
  - La funzione `sys = ss(A,B,C,D)` crea un modello ingresso-stato-uscita di sistema rappresentato nella forma matriciale

    $dx = A\,x + B\,u$

    $y = C\,x + D\,u$
  - Esempio
    - ```
      A = [1 0;-1 -0.5]; b = [1;2];
      c = [0 1]; d = [0];
      sys = ss(A,b,c,d);
      ```

      corrisponde alla funzione di trasferimento

$$\frac{2s-3}{s^2-\frac{1}{2}s-\frac{1}{2}}$$
      ```

Conversione del modello

- Il passaggio tra le diverse forme di rappresentazione può essere realizzato in modo diretto utilizzando le seguenti funzioni

  La funzione `sys2 = zpk(sys)` converte il modello sys nella forma di rappresentazione guadagno-zeri-poli

  La funzione `sys2 = tf(sys)` converte il modello sys nella forma di rappresentazione polinomiale

  La funzione `sys2 = ss(sys)` converte il modello sys nella forma di rappresentazione ingresso-stato-uscita

Dinamiche del modello

- Le principali proprietà dinamiche di un modello LTI possono essere direttamente valutate mediante le seguenti funzioni:
  - `fb = bandwidth(sys)` banda passante della risposta in frequenza
  - `k = dcgain(sys)` guadagno in bassa frequenza
  - `p = pole(sys)` valuta i poli del modello
  - `z = zero(sys)` valuta gli zeri del modello
  - `[p,z] = pzmap(sys)` valuta e disegna la mappa poli-zeri del modello

Interconnessione dei modelli LTI / 1

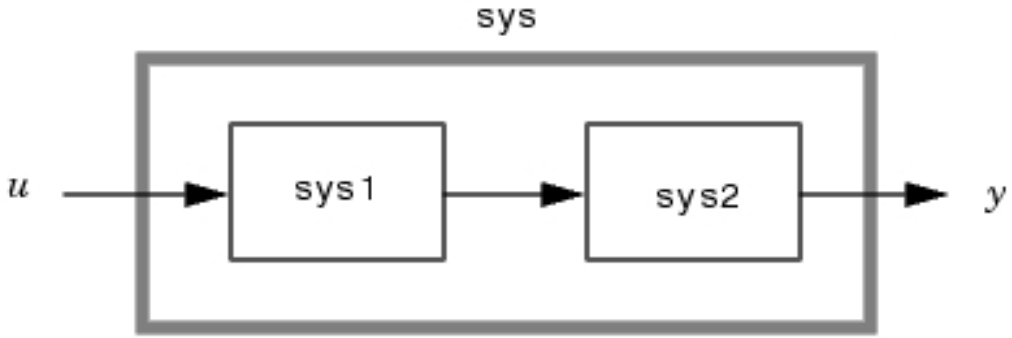
- La connessione in serie di due modelli LTI SISO può essere realizzata mediante il seguente comando

```
sys = series(sys1,sys2)
```

oppure moltiplicando direttamente i due sistemi

```
sys = sys1 * sys2
```

Il nuovo modello `sys` è il risultato della serie del modello `sys1` seguito dal modello `sys2`



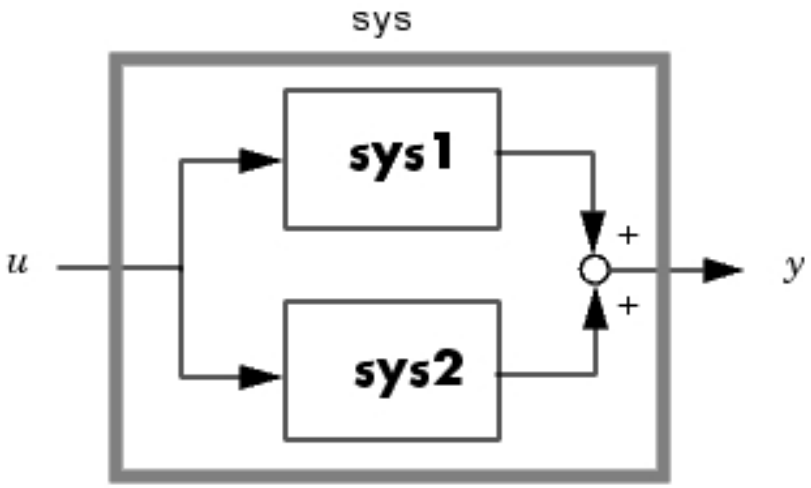
Interconnessione dei modelli LTI / 2

- La connessione in parallelo di due modelli LTI SISO può essere realizzata mediante il seguente comando

```
sys = parallel(sys1,sys2)
```

oppure sommando direttamente i due sistemi

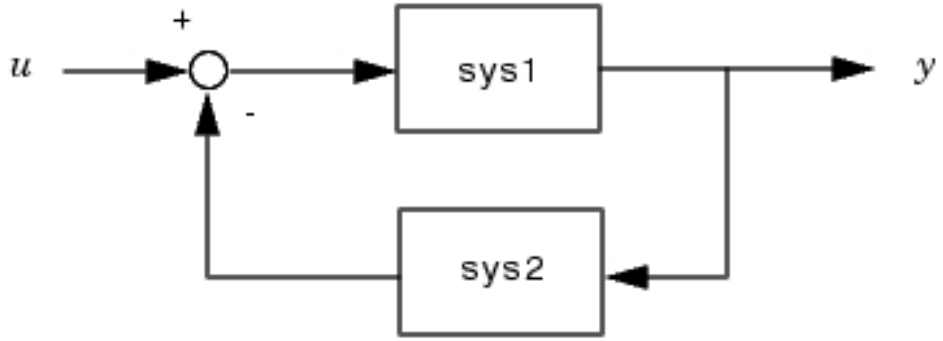
```
sys = sys1 + sys2
```



Interconnessione dei modelli LTI / 3

- La connessione in retroazione negativa di due modelli LTI SISO può essere realizzata mediante il seguente comando

```
sys = feedback(sys1,sys2)
```



- Sono anche ammessi casi in cui uni dei sue sistemi è un semplice guadagno (`sys1 = k` oppure `sys2 = h`)
- Con il comando `sys=feedback(sys1,sys2,+1)` si ottiene il modello corrispondente alla reazione positiva

Risposta in evoluzione libera

- La risposta in evoluzione libera di un modello LTI ingresso-stato-uscita, con stato iniziale $x(0)$ assegnato, può essere valutata e graficata(*) con il comando

```
[y,t,x] = initial(sys,x0)
```

dove `y` è il vettore dei valori assunti dall'uscita del modello, `t` è il vettore dei tempi e `x` è la traiettoria dello stato.

`sys` deve essere costruito con `ss`

- Esempio:

```
A = [-0.5572 -0.7814; 0.7814 0];
c = [1.9691 6.4493];
x0 = [1; 0]
sys = ss(A,[],c,[]);
initial(sys,x0)
```

Risposta ad un impulso

- La risposta all'impulso di un modello LTI può essere valutata e graficata con il comando

  `[y,t] = impulse(sys)`

  `[y,t,x] = impulse(sys)` (per i modelli i-s-u)

  dove `y` è il vettore dei valori assunti dall'uscita del modello, `t` è il vettore dei tempi e `x` è la traiettoria dello stato

- Esempio:

```
A = [-0.5572 -0.7814;0.7814  0];
b = [-1; 2];
c = [1.9691  6.4493];
sys = ss(A,b,c,0);
impulse(sys)
```

Risposta allo scalino / 1

- La risposta allo scalino di un modello LTI può essere valutata e graficata con il comando

```
[y,t] = step(sys)
```

```
[y,t,x] = step(sys)
```
(per i modelli i-s-u)

dove `y` è il vettore dei valori assunti dall'uscita del modello, `t` è il vettore dei tempi e `x` è la traiettoria dello stato

- Esempio:

```
A = [-0.5572 -0.7814;0.7814  0];
b = [1; 0];
c = [1.9691  6.4493];
sys = ss(A,b,c,0);
step(sys)
```

Risposta allo scalino / 2

- Menu contestuale
  - Cliccando con il tasto destro sul grafico, accedendo al menù delle caratteristiche, è possibile rilevare automaticamente le principali caratteristiche della risposta al gradino:
    - Ts: Tempo di salita (10-90%)
    - Ta: Tempo di assestamento (default al 2%)
    - s%: Sovraelongazione
    - y∞: Valore di regime

Risposta ad ingresso generico / 1

- La risposta ad un ingresso generico di un modello LTI può essere valutata e graficata con il comando

```
[y,t,x] = lsim(sys,u,t)
[y,t,x] = lsim(sys,u,t,x0)
```

(per i modelli i-s-u)

dove u è il vettore dei valori assunti dall'ingresso del modello, y è il vettore dei valori assunti dall'uscita, t è il vettore dei tempi e x è la traiettoria dello stato

- Esempio:

```
t=0:0.01:10;
u=2*sin(pi*t).*(t<4)+5*(t>=6);
sys = tf([2 5 1],[1 2 3]);
lsim(sys,u,t)
```

Risposta ad un ingresso generico / 2

- Il segnale di ingresso può essere composto in modo discreto, usando le funzioni elementari del Matlab, oppure mediante la funzione

```
[u,t]=gensig(type,tau)
```

che può generare un segnale periodico di tipo sinusoidale (`type='sin'`), onda quadra (`type='square'`) o impulso periodico (`type='impulse'`), di periodo `tau`

- Esempio:

```
[u,t] = gensig('square',4,10,0.1);
sys = tf([2 5 1],[1 2 3]);
lsim(sys,u,t)
```

Diagramma di Bode / 1

- Il diagramma di Bode di un modello LTI può essere valutata e graficata con il comando

```
[mag,phase,w] = bode(sys)
```

dove `mag` e `phase` sono il vettore dei valori assunti dal modulo e dalla fase della risposta armonica, rispettivamente, e `w` è il vettore delle pulsazioni

- Esempio:

```
sys = tf([1 0.1 7.5],[1 0.12 9 0 0])
bode(sys); grid
```

Diagramma di Bode / 2

- Menu contestuale
  - Cliccando con il tasto destro sul grafico, accedendo al menù delle caratteristiche, è possibile misurare automaticamente eventuali caratteristiche risonanti, ovvero frequenza e ampiezza del picco di risonanza

Diagramma di Nichols

- Il diagramma di Nichols di un modello LTI può essere valutata e graficata con il comando

  `[mag,phase,w] = nichols(sys)`

  dove `mag` e `phase` sono il vettore dei valori assunti dal modulo e dalla fase della risposta armonica, rispettivamente, e `w` è il vettore delle pulsazioni

- Esempio:

```
num = [-4 48 -18 250 600];
den = [1 30 282 525 60];
sys = tf(num,den)
nichols(sys); ngrid
```

Diagramma di Nyquist

- Il diagramma di Nyquist di un modello LTI può essere valutata e graficata con il comando

```
[re,im,w] = nyquist(sys)
```

dove `re` e `im` sono il vettore dei valori assunti dalla parte reale e dalla parte immaginaria della risposta armonica, rispettivamente, e `w` è il vettore delle pulsazioni

- Esempio:

```
sys = tf([2 5 1],[1 2 3])
nyquist(sys); grid
```

Ulteriori interfacce per sistemi LTI

- Potrebbero essere utili
  - `sisotool`
  - `ltiview`

Esercizio #1

Sia assegnato il sistema LTI rappresentato dalla seguente funzione di trasferimento

$$F(s) = \frac{50s+50}{s^4+8s^3+30s^2+76s+80}$$

Si valutino:

1) i parametri della risposta al gradino (t_s, $t_{a,2\%}$, $t_{a,5\%}$,$s_{\%}$, y_∞)
2) la risposta al segnale:

$$u(t) = 2\delta_{-1}(t) - \delta_{-2}(t-5)\delta_{-1}(7-t) - 2\delta_{-1}(t-7)$$

1) I parametri della risposta armonica (*banda*, *risonanza*, *ecc.*)

Esercizio #2

Sia assegnato il sistema LTI rappresentato dalle seguenti matrici

$$A = \begin{bmatrix} -0.5 & -0.8 \\ 0.8 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \qquad C = [1 \quad 3]$$

Si valutino:

1) la durata della risposta libera, valutata come il tempo di assestamento al di sotto del 5% del valore di picco, assegnato $\boldsymbol{x}(0) = [1 \quad 2]^T$
2) i parametri della risposta al gradino (t_s, $t_{a,2\%}$, $t_{a,5\%}$,$s_{\%}$, y_∞)
3) la risposta ad un'onda quadra di periodo 2s e ampiezza 10
4) I parametri della risposta armonica (*banda*, *risonanza*, *ecc.*)

Esercizio #3

Sia assegnato il sistema LTI avente i seguenti poli, zeri e guadagno di alta frequenza

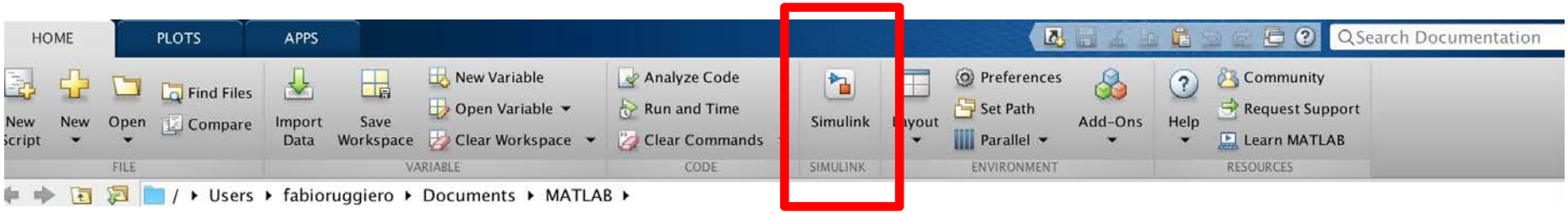
$$p = [\,-1, -4, 2 + j, -2 - j\,]$$
$$\mathrm{z} = [\,-3,\ \ 4\,]$$
$$k = -5$$

Si valutino:

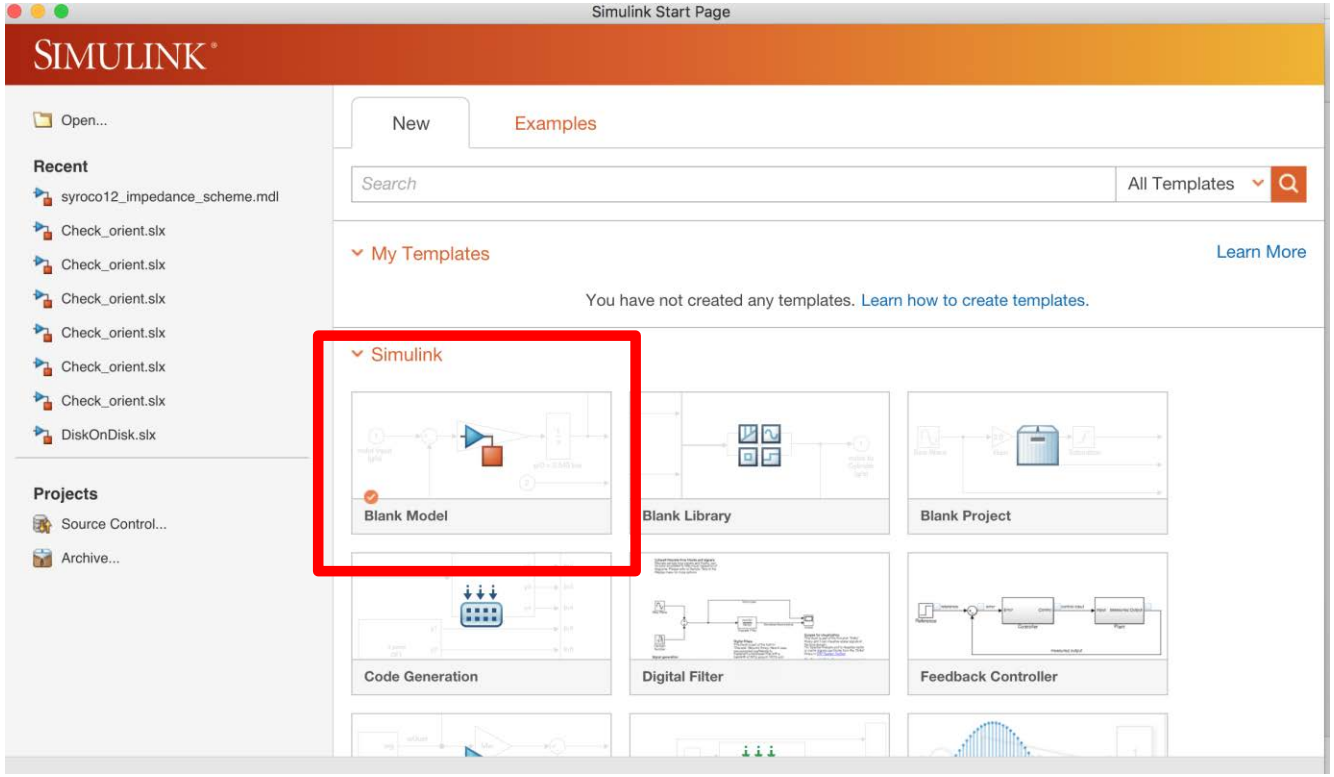
1) i parametri della risposta al gradino (t_s, $t_{a,2\%}$, $t_{a,5\%}$,$s_{\%}$, y_∞)
2) la durata della risposta all'impulso, valutata come il tempo di assestamento al di sotto del 2% del valore di picco
3) la risposta ad un'onda sinusoidale di periodo 1s e ampiezza 2

Simulink / 1

- Introduzione
  - È un software, per lo più grafico, per la modellazione, simulazione e analisi di sistemi dinamici
  - Strettamente integrato con Matlab
- Avvio
  - Digitare *`simulink`* dal prompt dei comandi di Matlab
  - Cliccare l'apposita icona

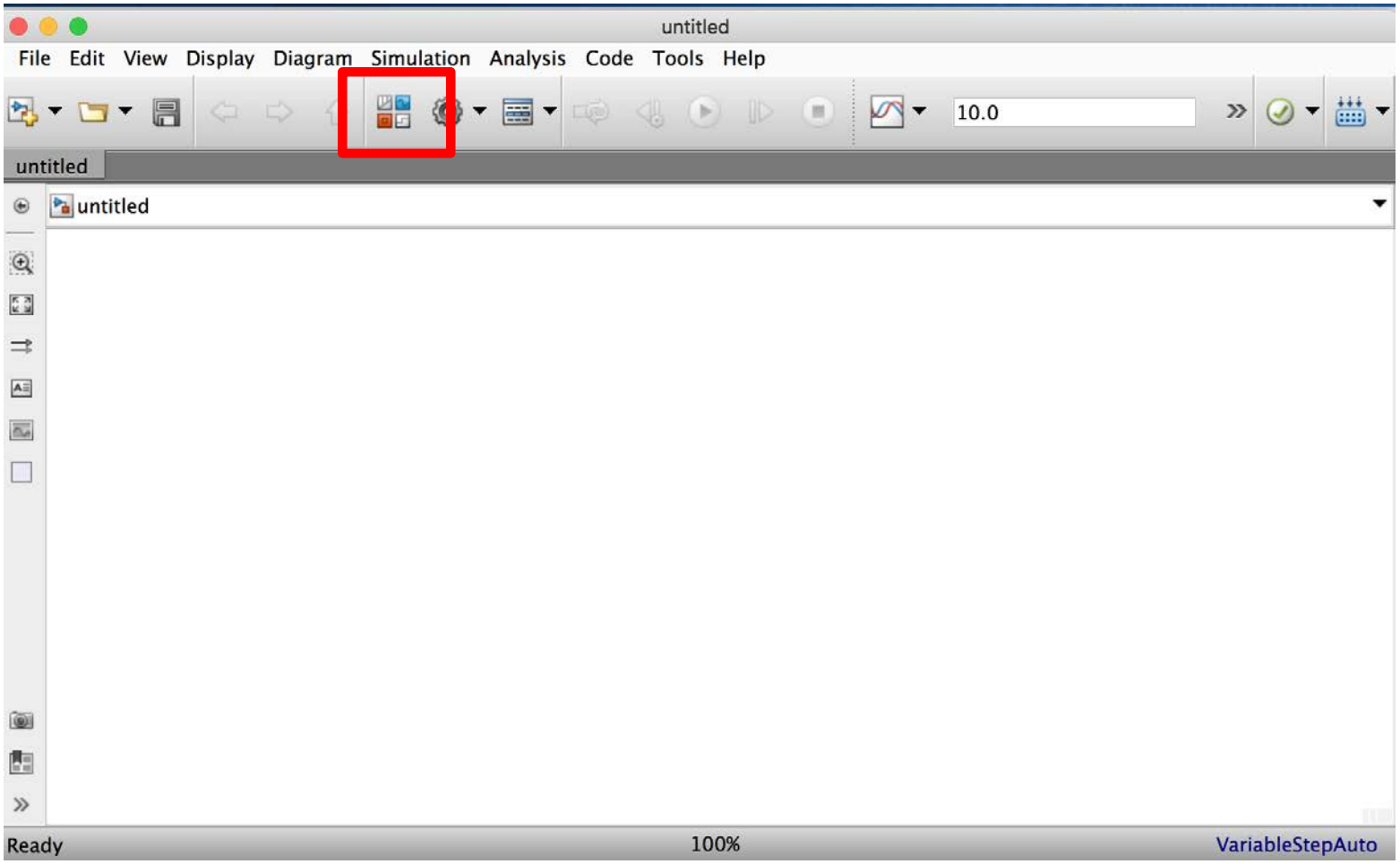


- Creare nuovo modello
  - Cliccare l'icona *Blank Model* per aprire un nuovo file di lavoro Simulink

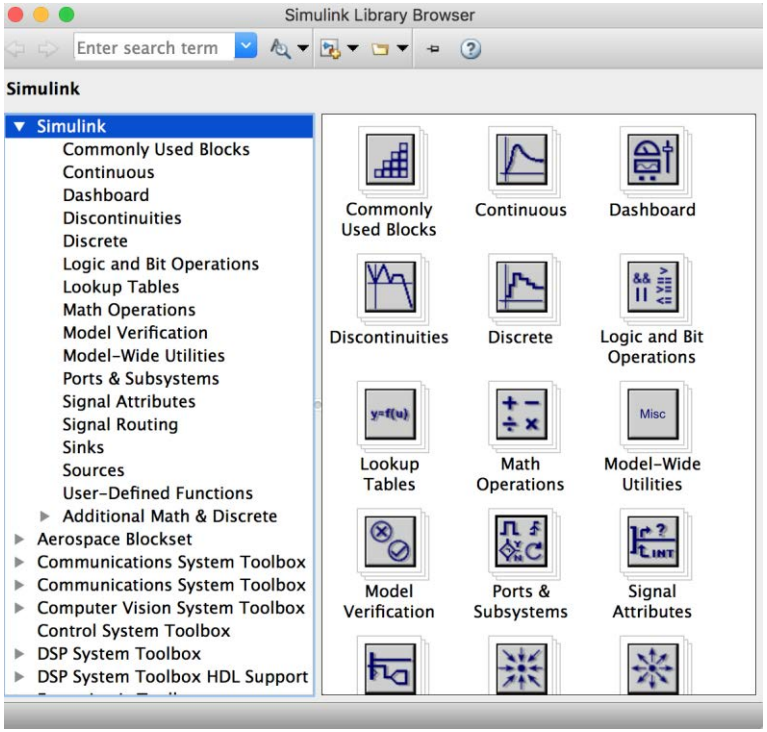


Simulink / 3

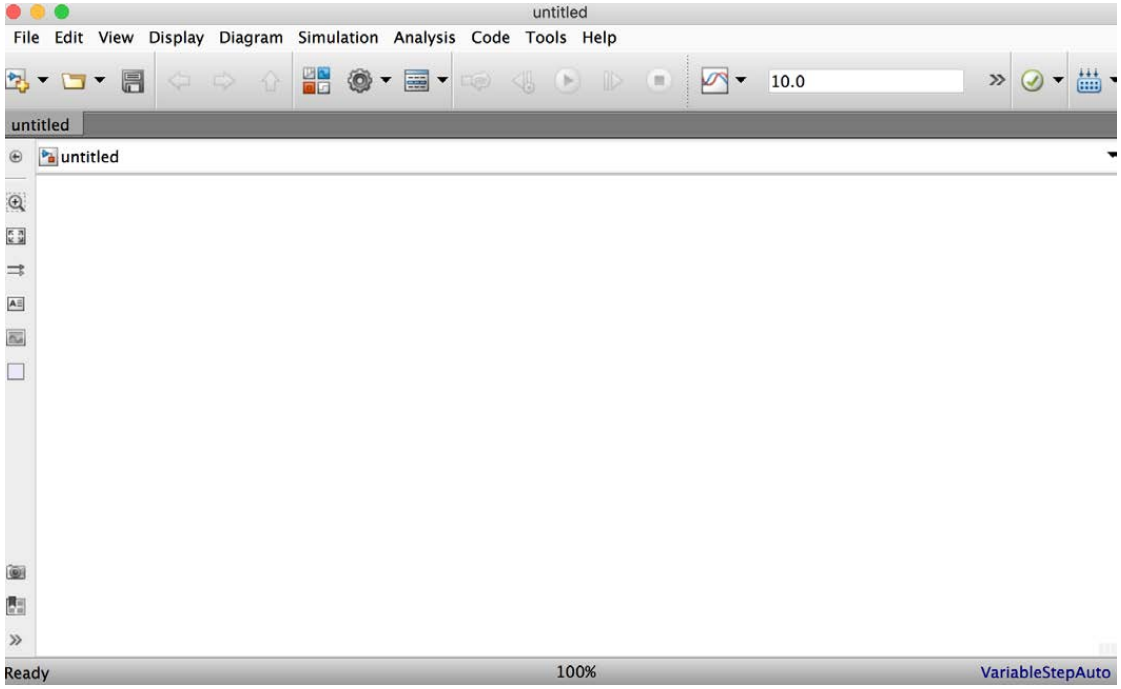
- Cliccare l'icona selezionata per aprire la libreria dei modelli



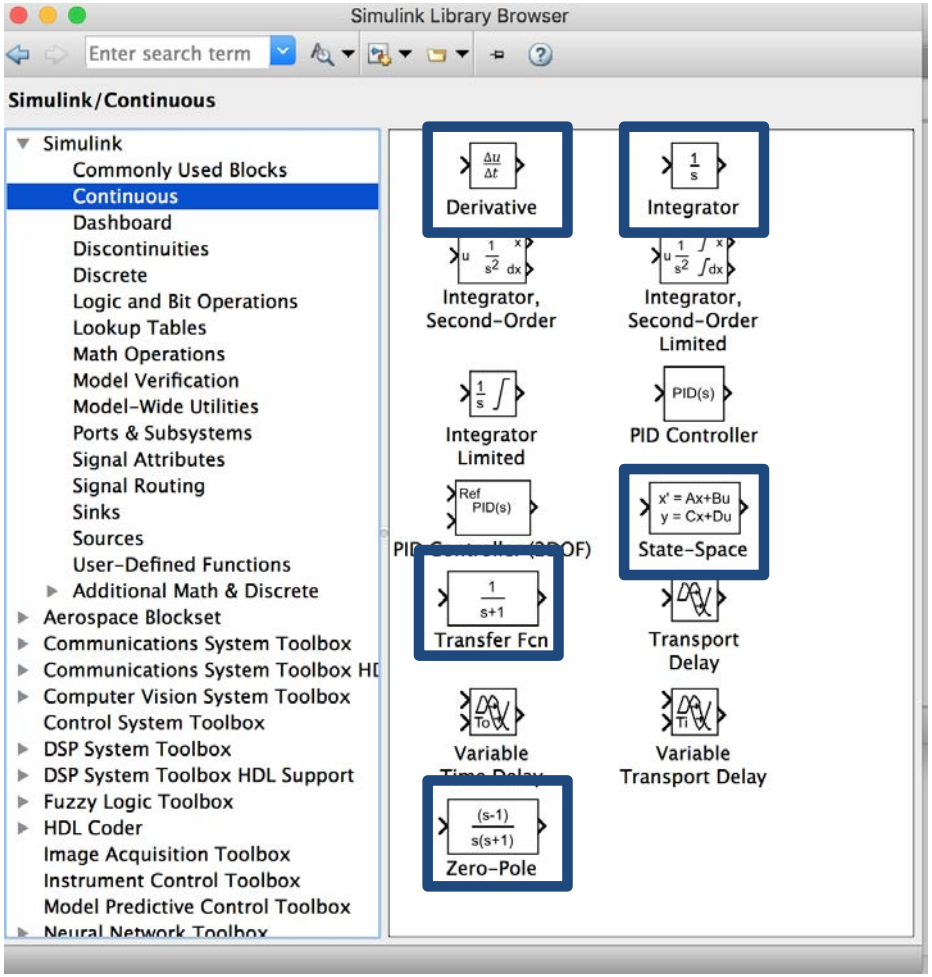
Libreria degli elementi



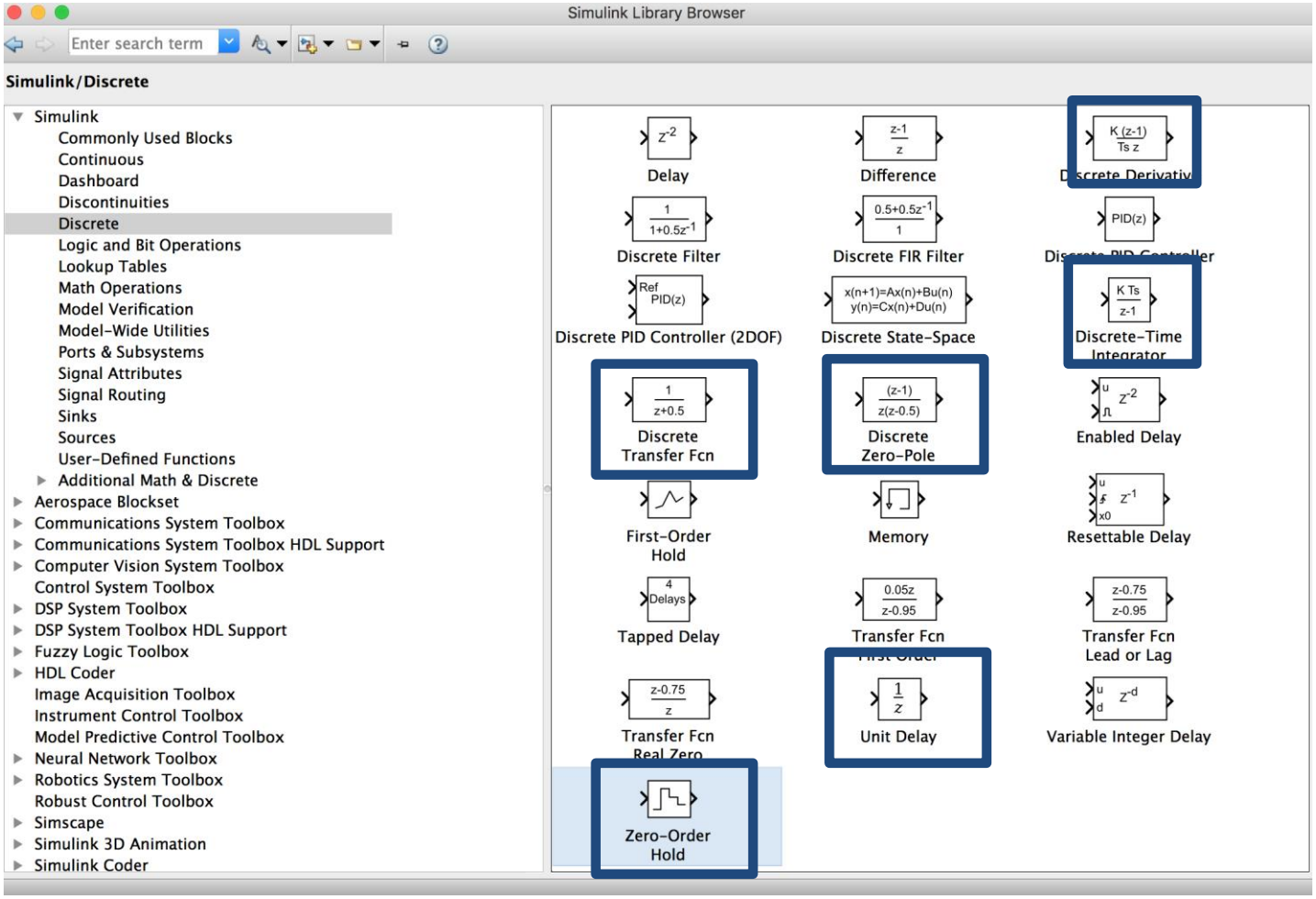
Il modello viene creato in questa finestra



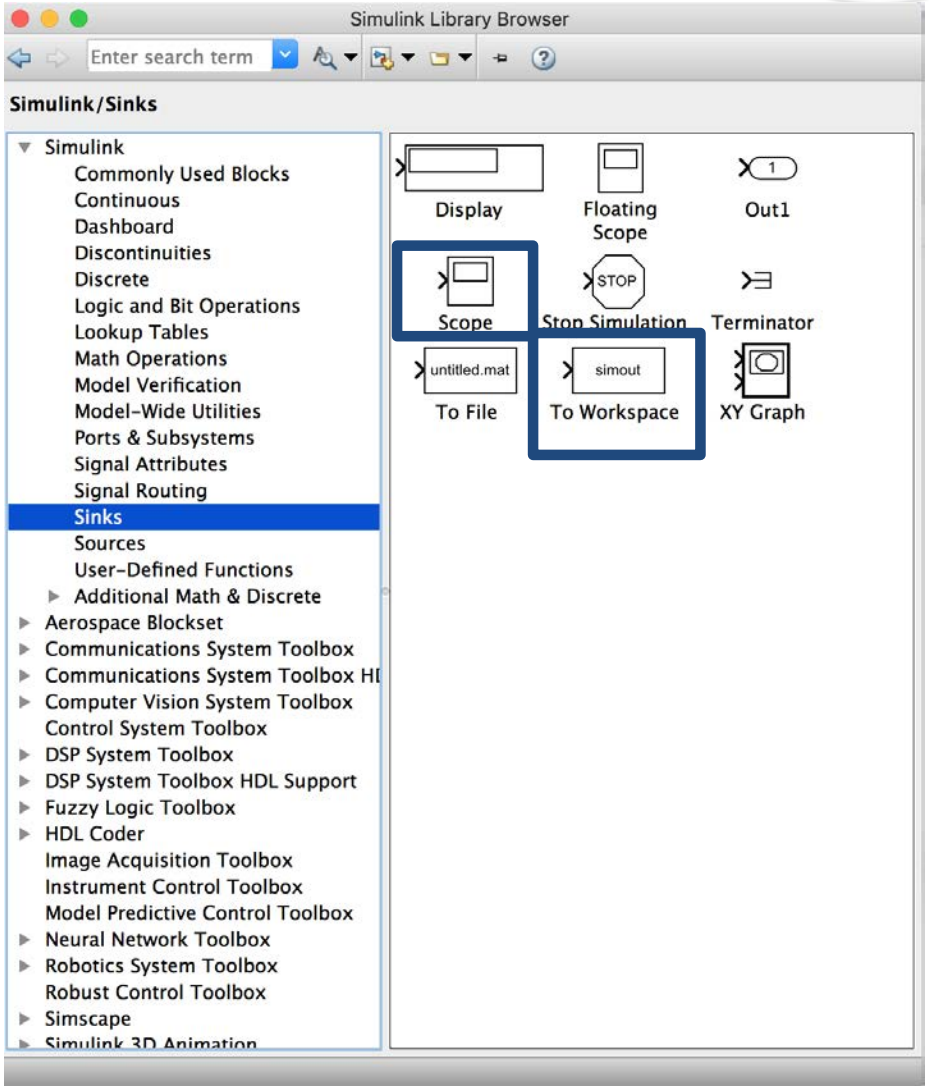
- Libreria dei modelli Simulink
  - Tempo continuo



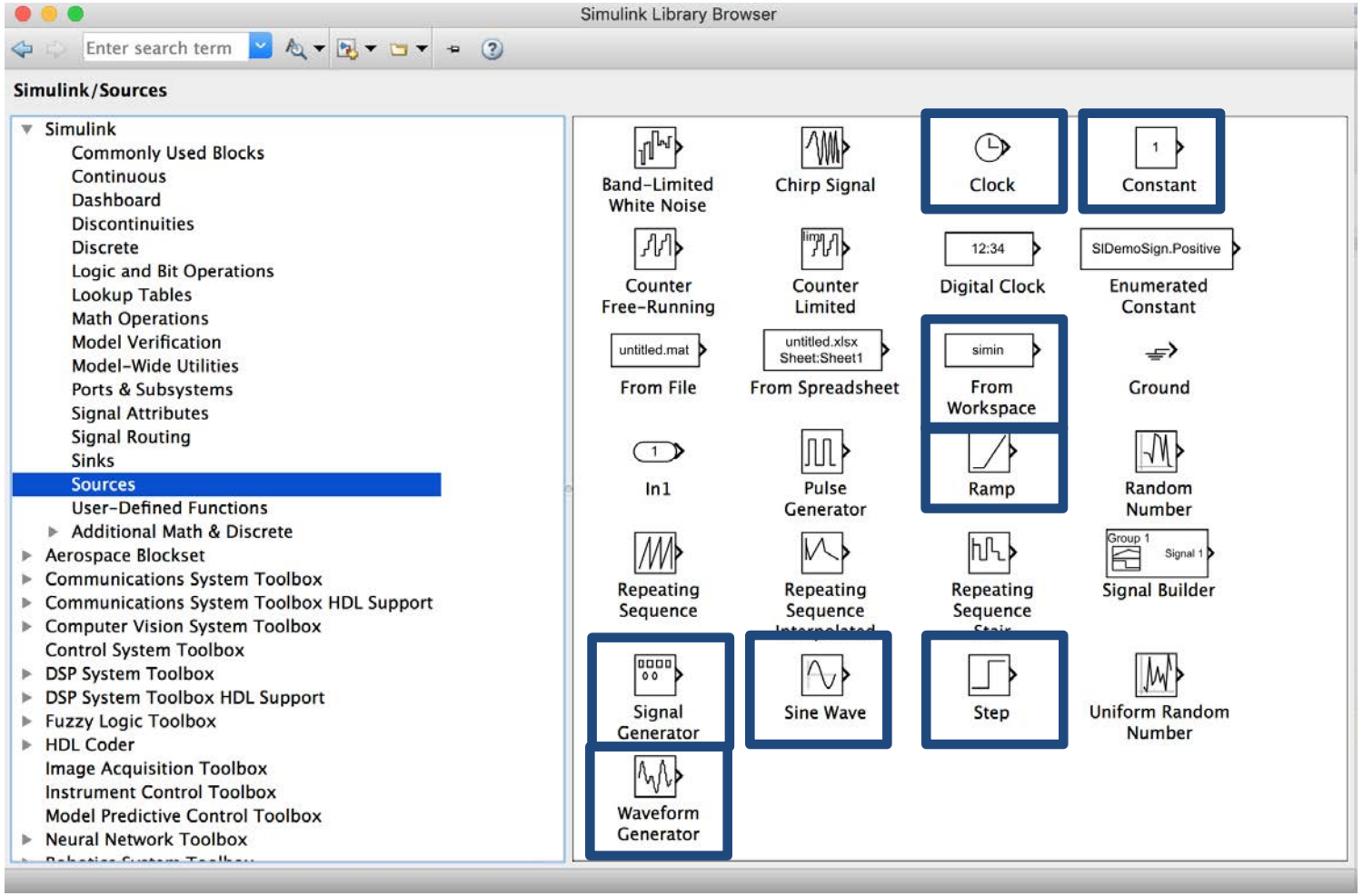
- Tempo discreto



- Visualizzazione segnali

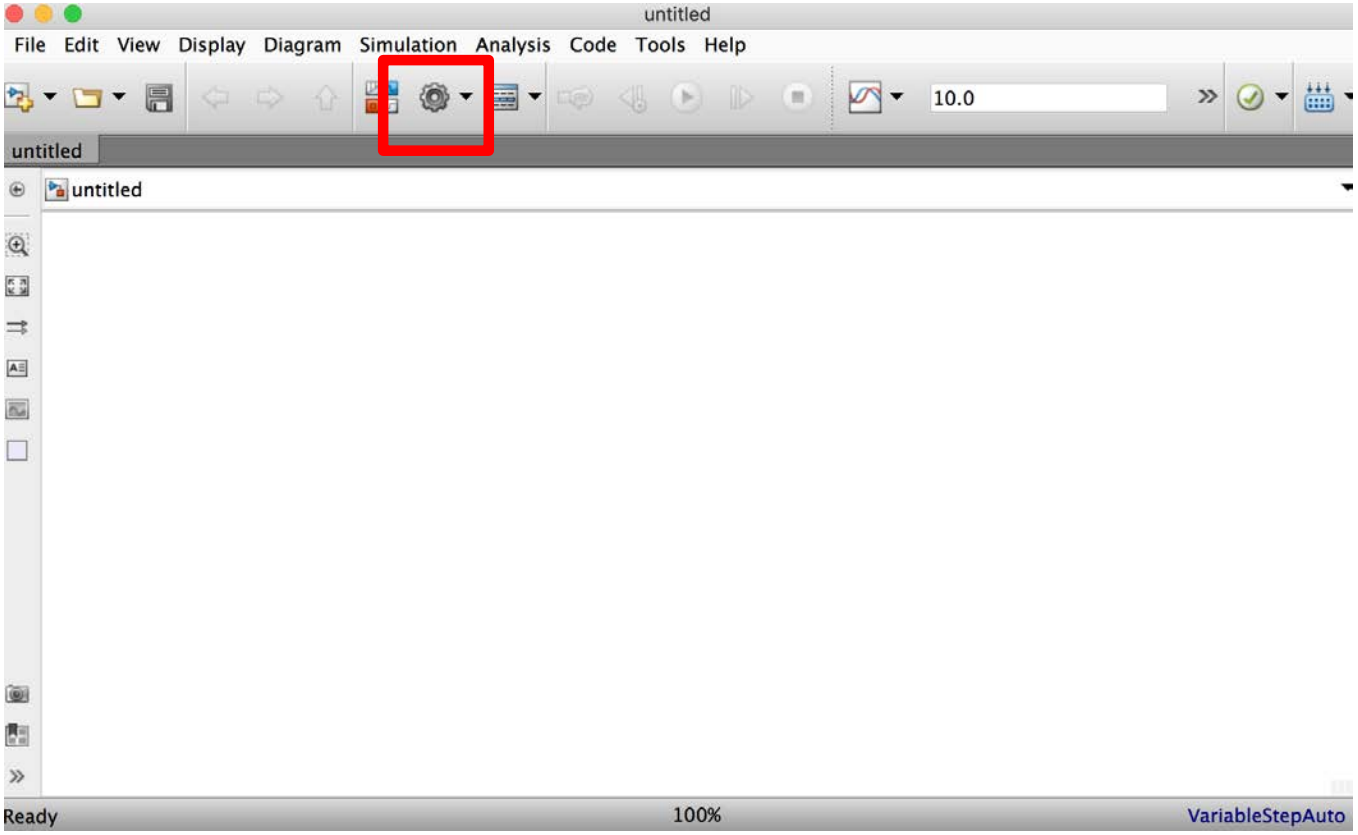


- Generazione segnali

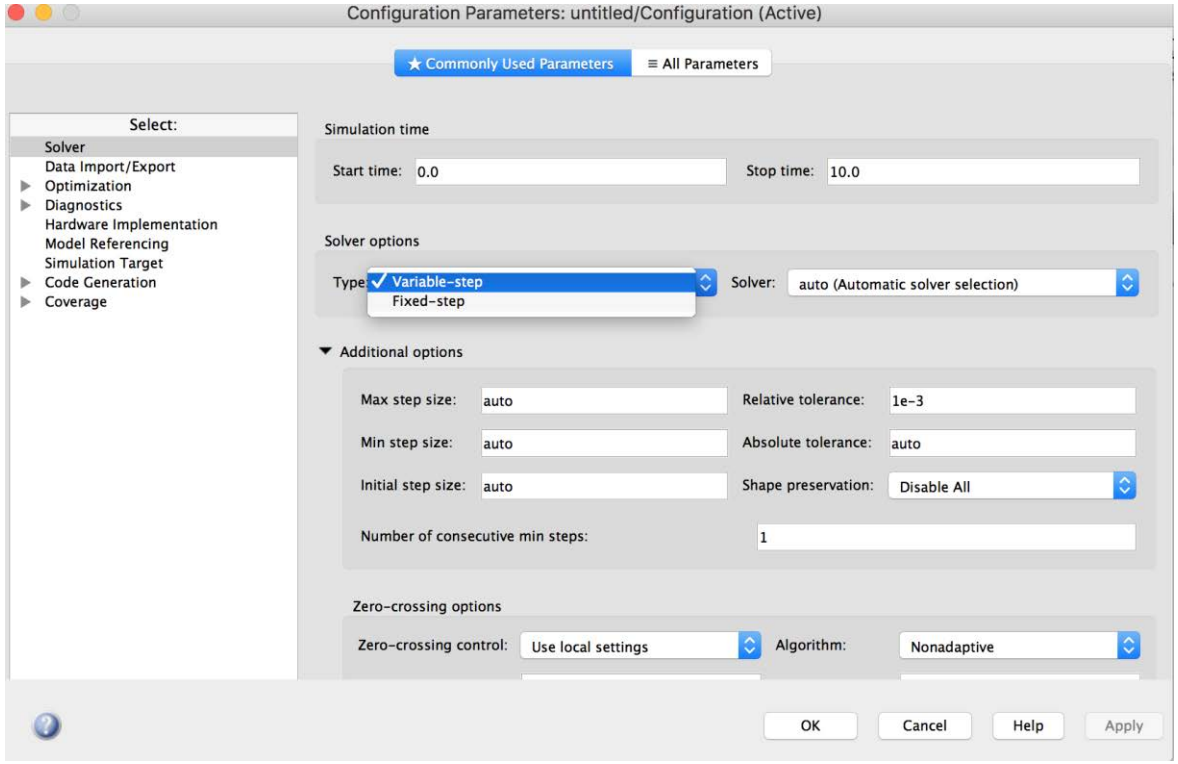


- Creazione del modello per via grafica
  - Trascinare il blocco desiderato dalla libreria alla finestra di lavoro, oppure tasto destro col mouse e cliccare *Add block to model [titolo del modello]*
  - Alcuni blocchi hanno porte di solo ingresso o sola uscita, altri hanno porte sia di ingresso che di uscita
    - Le porte possono essere collegate fra di loro “tirando” una linea fra una porta di uscita ed una di entrata tenendo premuto il pulsante sinistro del mouse
    - Le nuove versioni di Simulink permettono collegamenti veloci fra blocchi con tasti a scelta rapida, dipendenti dal sistema operativo, e funzioni di allineamento veloce

- Scelta delle configurazioni di simulazioni
  - Premere l'icona con il simbolo di ingranaggio



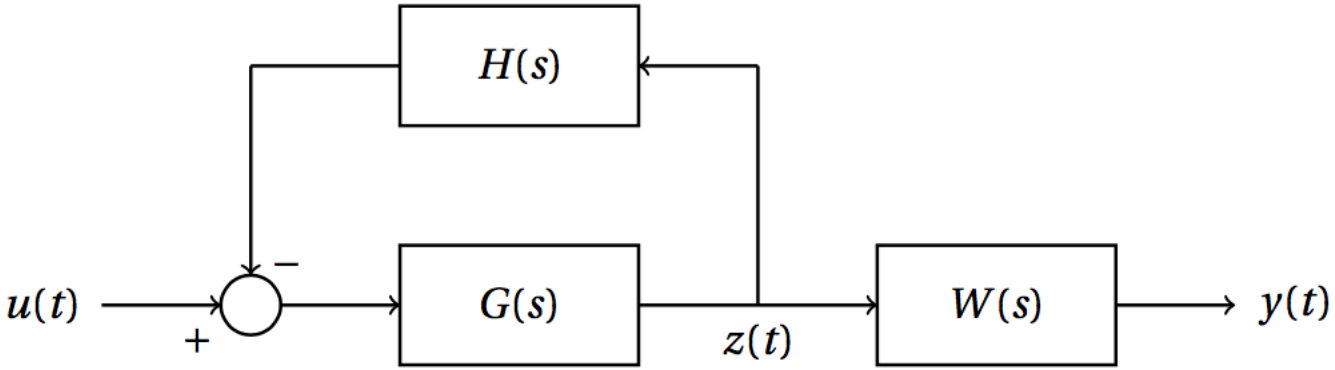
- Scelta del tipo di *solver*
  - *Variable-step* sistemi a tempo continuo
  - *Fixed-step* sistemi a tempo discreto (scegliere anche il *fundamental sample time*)



- Scelta dei parametri dei blocchi funzionali
  - Per ogni blocco all'interno della finestra di lavoro, un doppio clic col tasto sinistro del mouse apre la finestra per la scelta delle opzioni del relativo blocco
    - Ogni blocco ha una varietà di impostazioni
    - Consultare la documentazione per eventuali dubbi e per gli esempi

- Chiamare un modello Simulink da Matlab
  - Si può usare il comando *sim* all'interno del programma di Matlab (estensione *.m*)
  - Consultare la documentazione, e relativi esempi, per l'accesso ai dati di uscita
  - Esempio
    - `simout = sim ('nome_del_file_simulink');`

- Esempio
  - Simulare il seguente schema a blocchi



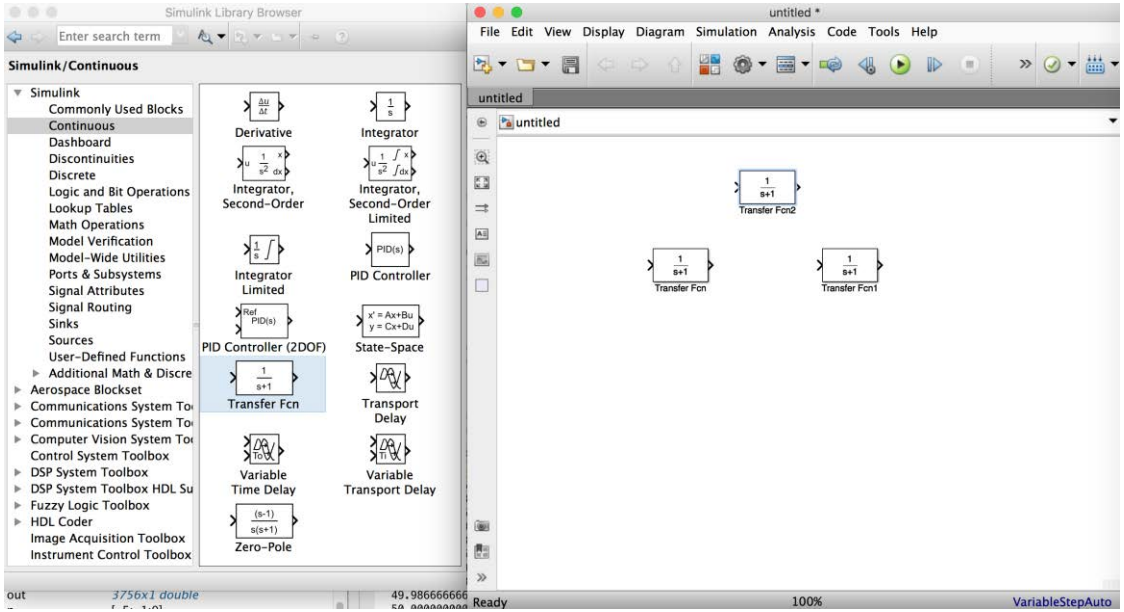
$$G(s) = \frac{1}{s},$$

$$H(s) = \frac{1}{s+10},$$

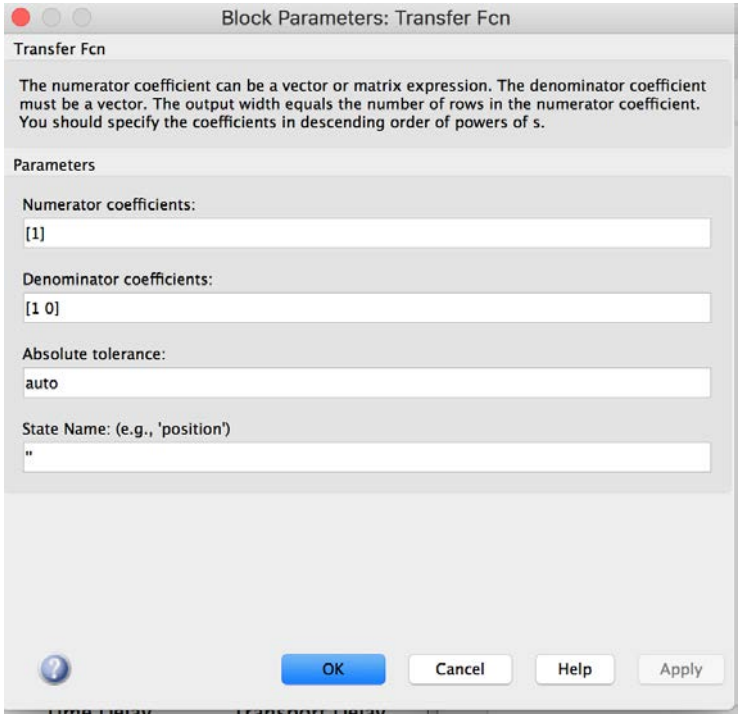
$$W(s) = \frac{s-1}{s+10}.$$

$$u(t) = 3\sin\left(2t + \frac{\pi}{8}\right)$$

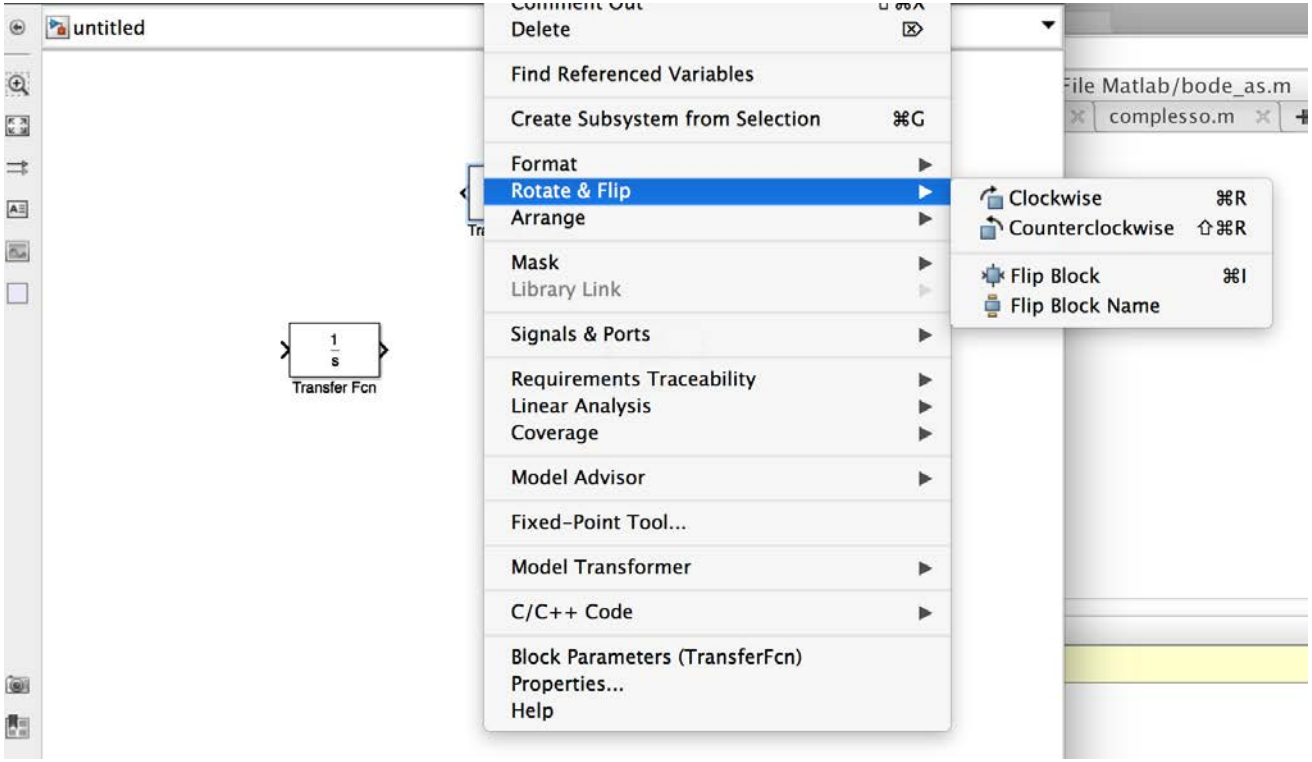
- Trascinare dalla libreria *Continous* tre blocchi *Transfer Fcn*
  - Se ne può importare anche uno, e duplicarlo poi nello schema di lavoro cliccando e trascinando il blocco col tasto destro del mouse
    - Così facendo verranno copiate nel nuovo blocco anche tutte le modifiche apportate alle impostazioni del blocco



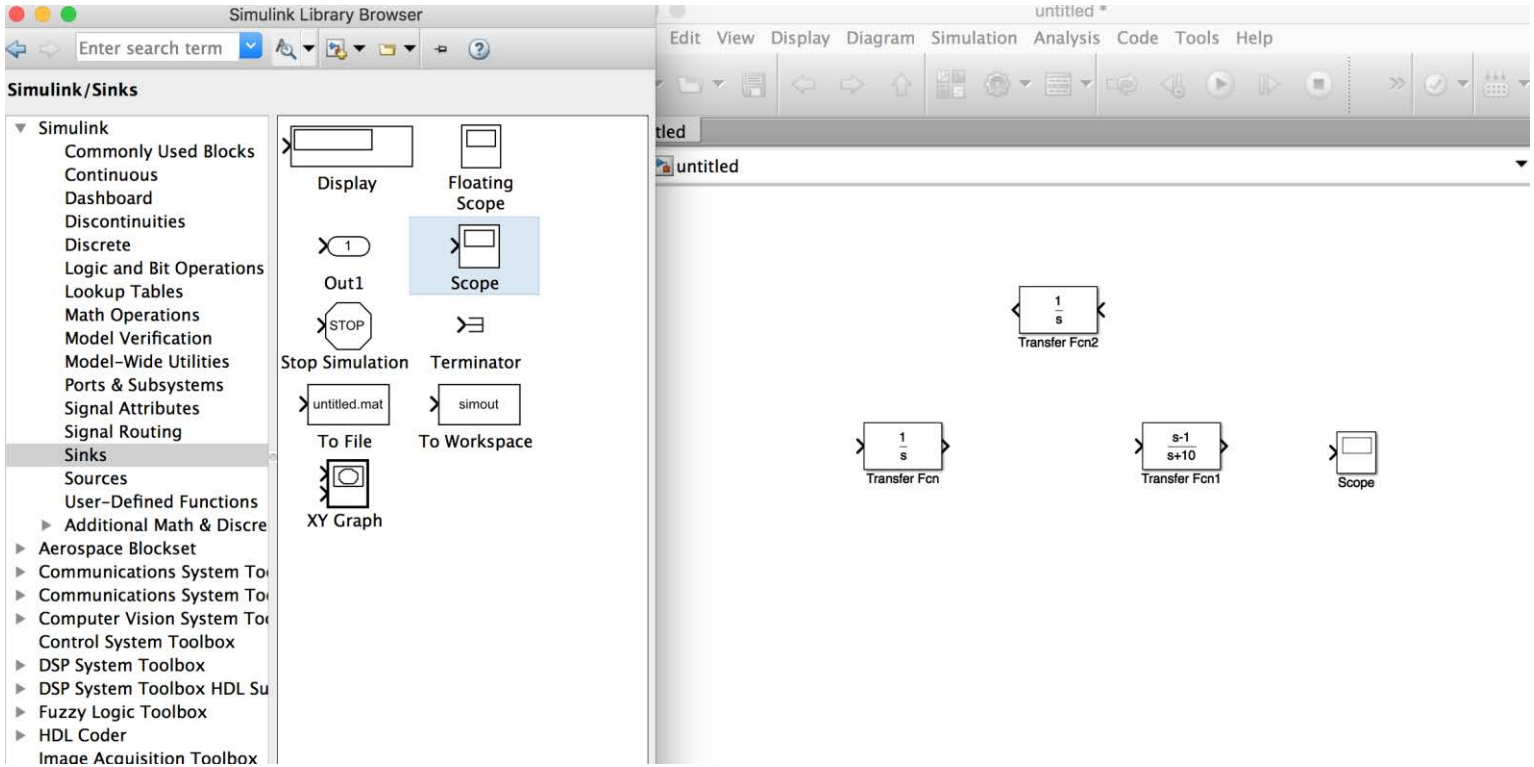
- Per ognuno dei blocchi, doppio clic col tasto sinistro del mouse per accedere alle configurazioni interne
  - Impostare i polinomi del numeratore e del denominatore di ogni singola funzione di trasferimento



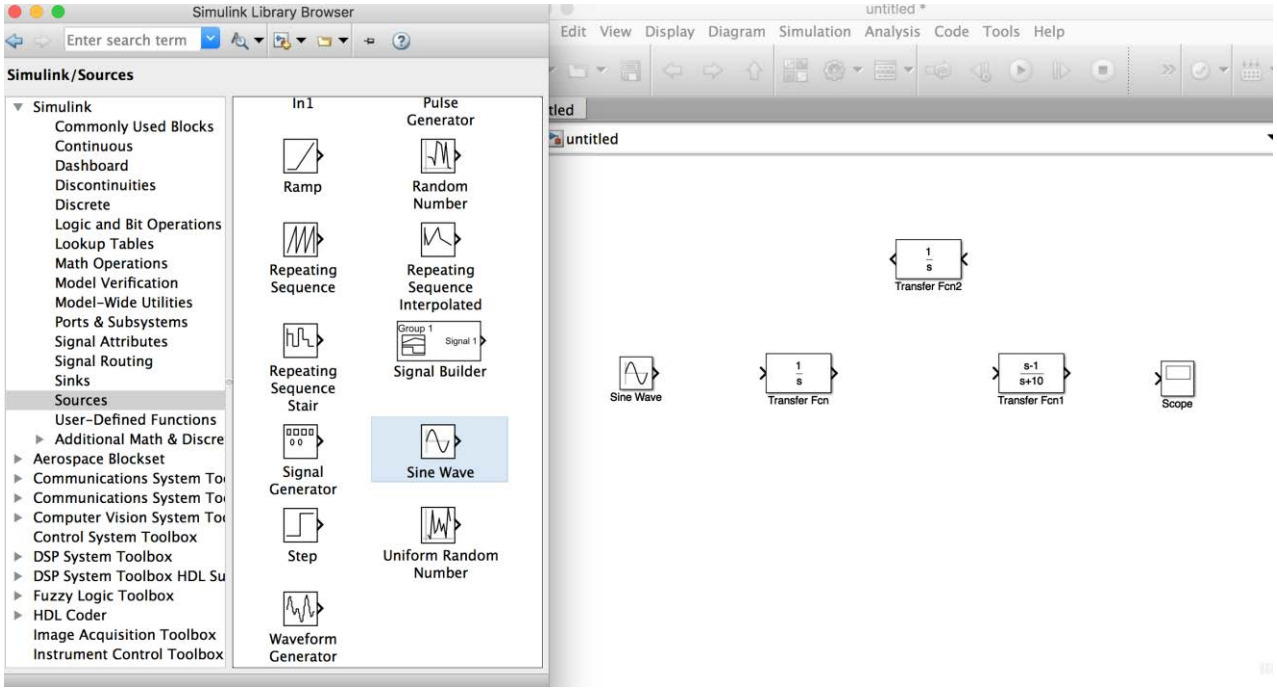
- Ruotare e specchiare un blocco
  - Per specchiare orizzontalmente H(s), ad esempio, cliccare col tasto destro del mouse sul blocco e cercare *`Rotate & Flip`* nel menu contestuale



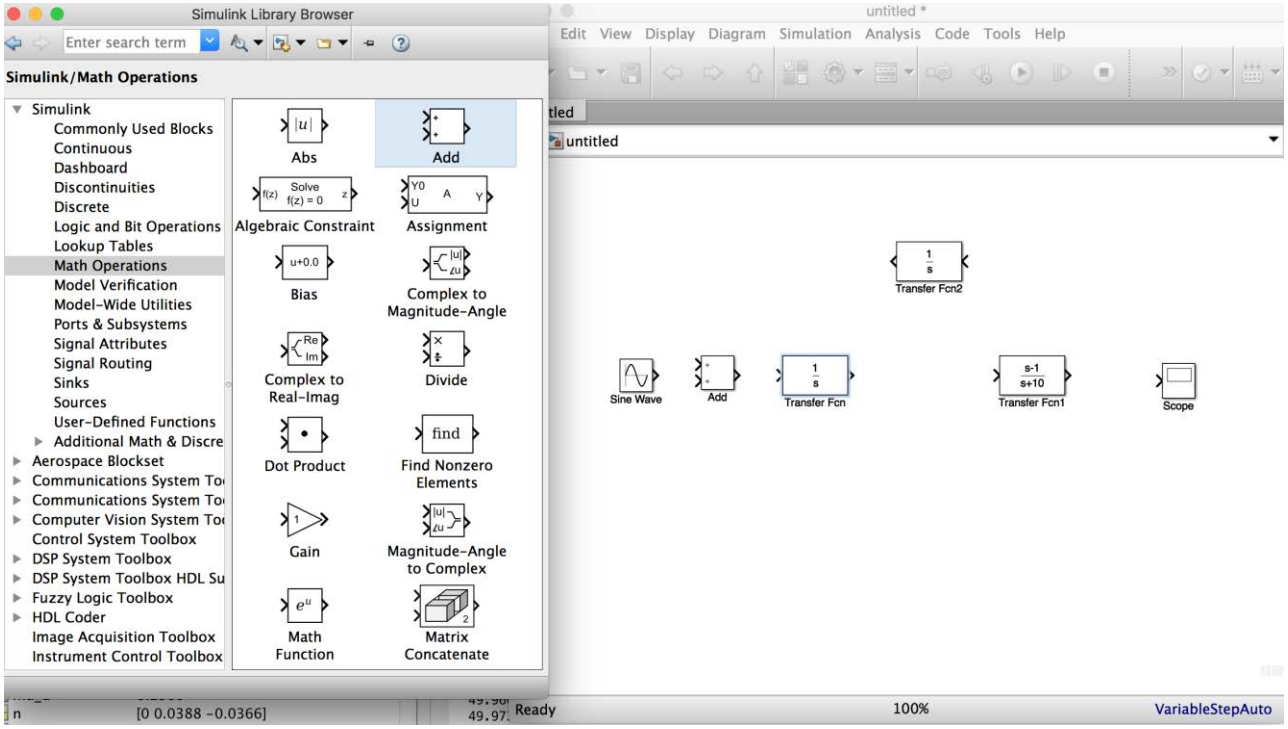
- Visualizzazione segnali
  - Importare uno *Scope* dal menu *Sinks* della libreria



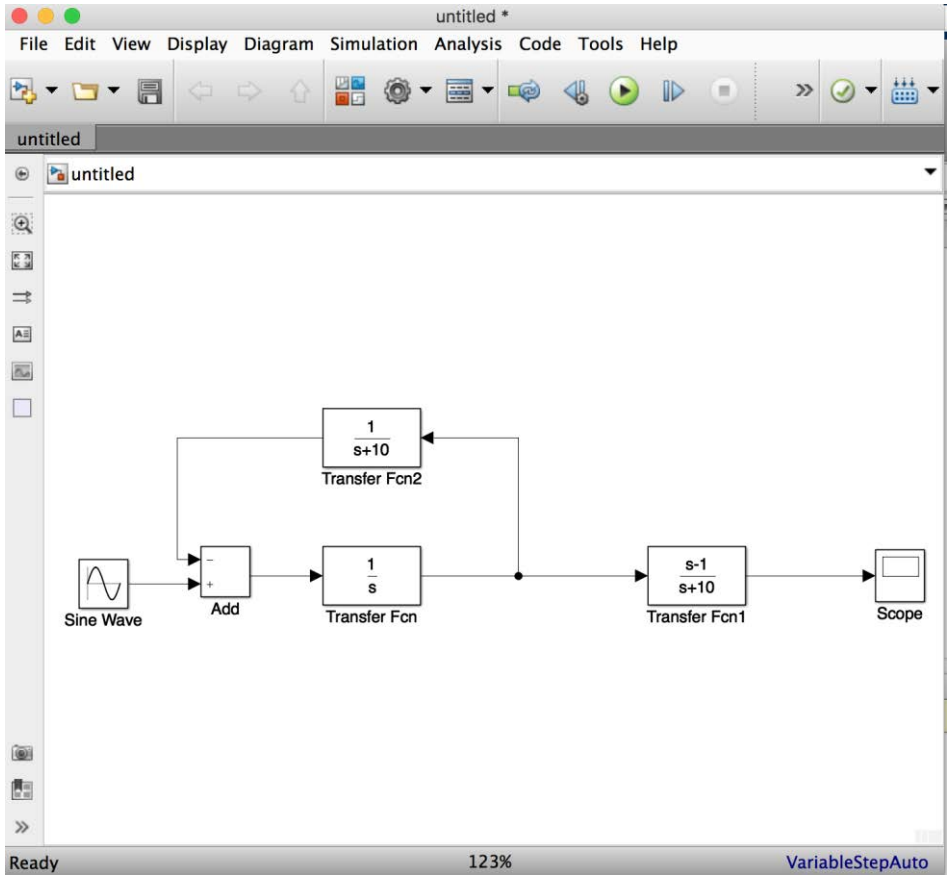
- Generazione dei segnali
  - Importare un blocco *Sine Wave* dalla libreria *Sources*
  - Doppio clic sul blocco col tasto sinistro del mouse per impostare i dati quali frequenza, ritardo, ampiezza, sfasamento, etc.



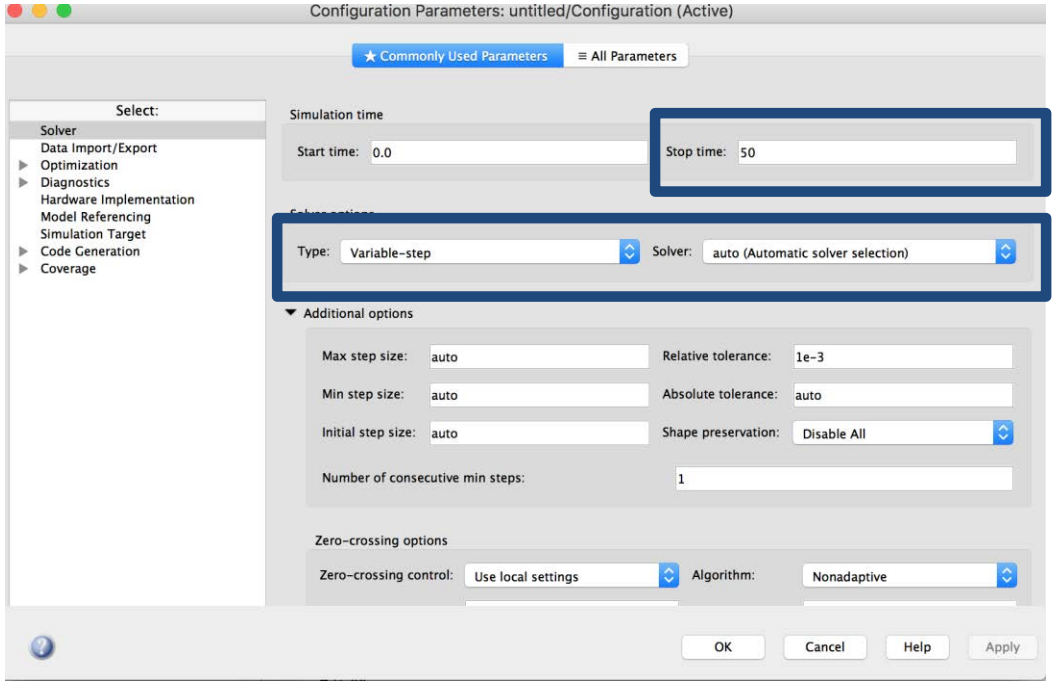
- Operazioni fra segnali
  - Inserire il blocco *Add* dalla libreria *Math Operations*
  - Doppio clic col tasto sinistro del mouse sul blocco per cambiare segno all'operazione e relativo ordinamento



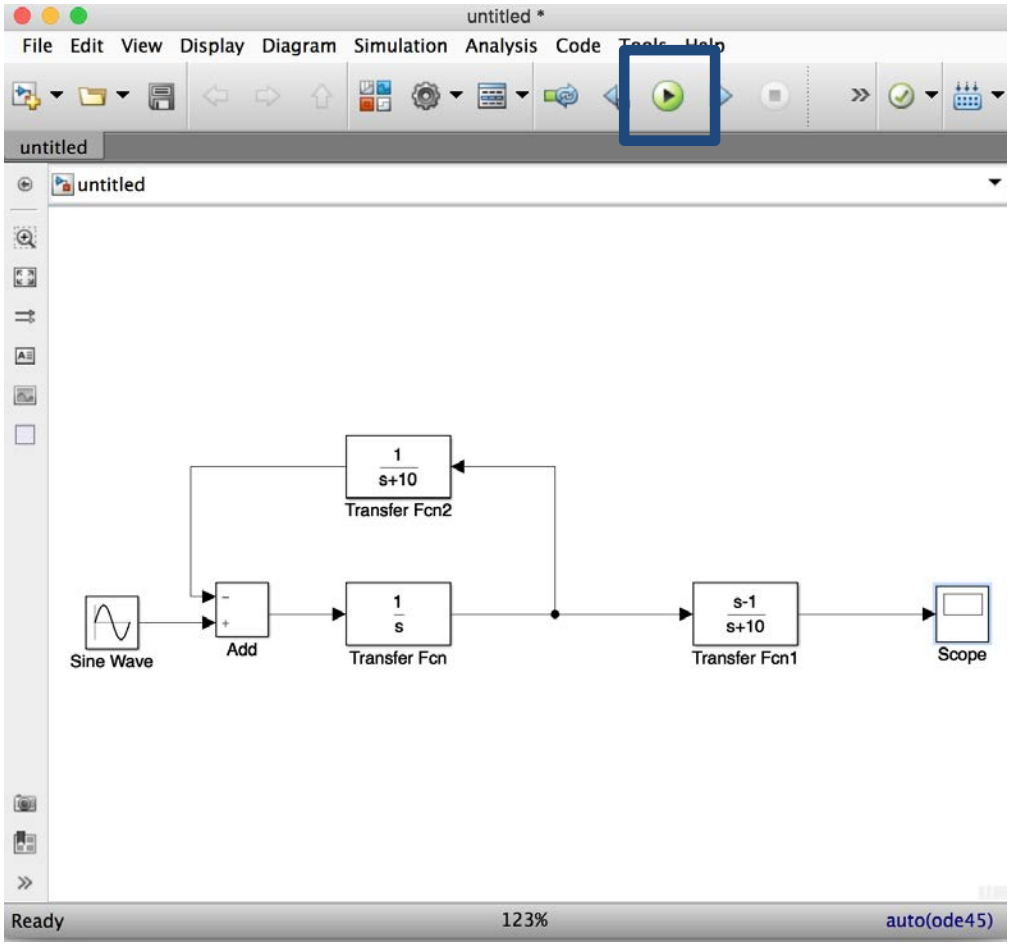
- Procedere ora al collegamento dei vari componenti dello schema come indicato nel testo dell'esempio



- Cliccare sull'icona dell'ingranaggio per le impostazioni dello schema e scegliere il *solver* per un sistema a tempo continuo (*Variable Step*)
- Impostare anche il tempo finale di simulazione (*Stop Time*) a 50 secondi



- Cliccare il tasto *Play* per avviare la simulazione



- Terminata la simulazione è possibile visualizzare il segnale di uscita, come qualunque altro segnale desiderato a cui sia stato collegato uno *Scope*, tramite doppio clic del tasto sinistro del mouse sul relativo blocco