POLITECNICO DI MILANO

Lab





Esercitazione 1
Sistemi e Equazioni Differenziali

Corso di Strumentazione e Controllo di Impianti Chimici

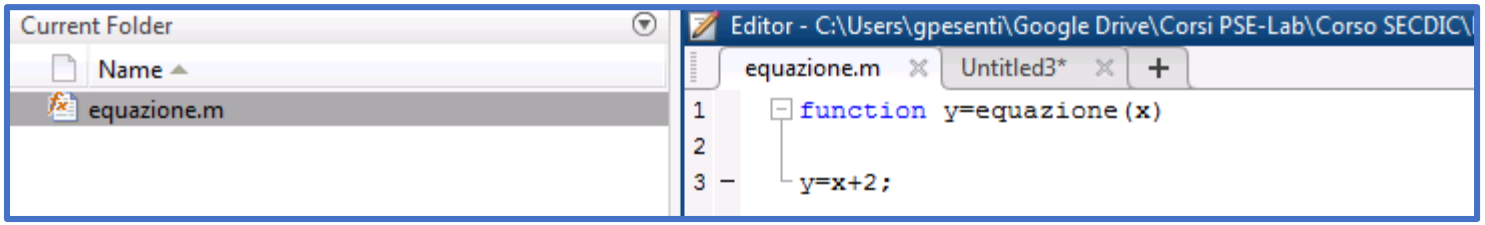
Prof. Davide Manca

Tutor: Giuseppe Pesenti

Funzioni definite all'interno di file

Definire e chiamare funzioni:

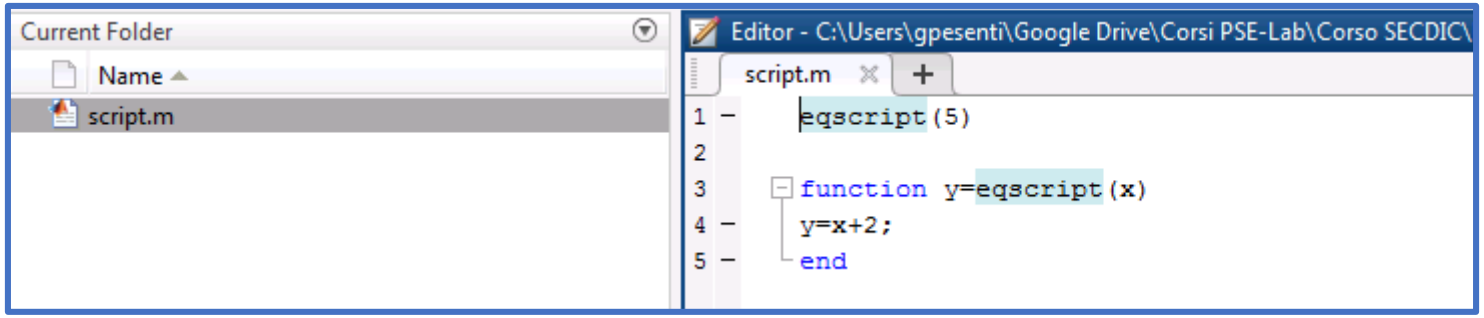
- **Definire la funzione in file .m con lo stesso nome della funzione**



  - Accessibili se il file della funzione è nella Current Folder di Matlab

```
>> equazione(5)
```

- **Definire la funzione in uno script**



  - Solo lo script stesso può chiamare la funzione

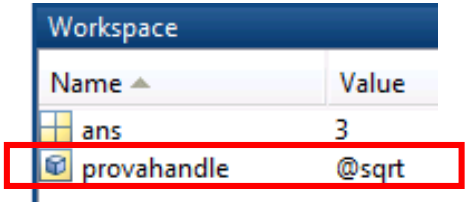
```
>> eqscript(1)
Undefined function or variable 'eqscript'.
```

Function handle

- **Function handle**

```
>> provahandle=@sqrt;
>> provahandle(9)
```

→ ans = 3



L'**operatore @** memorizza nella variabile (**handle**) l'**associazione a una funzione**

```
equazione.m
1   function y=equazione(x)
2
3 -     y=x+2;
```

```
>> handleeq=@equazione;
>> handleeq(7)
```

→ ans = 9

Il function handle memorizza la **specifica istanza della funzione**

```
equazione.m
1   function y=equazione(x,a)
2
3 -     y=x+a;
```

```
>> a=5;
>> handle2=@(x)equazione(x,a);
>> handle2(9)
```

→ ans = 14

```
>> a=0;
>> handle2(9)
```

←

- **Memorizza** anche eventuali **parametri**
- **Anche se i parametri cambiano** o non sono più nello scope
- Il function handle **è una variabile**: può essere usato e passato come una variabile

Funzioni anonime

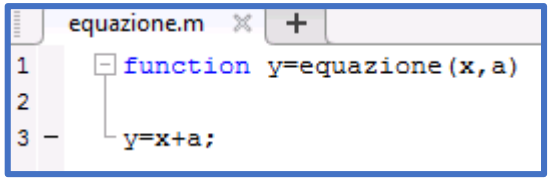
E' possibile **associare un function handle a funzioni non definite in un file**

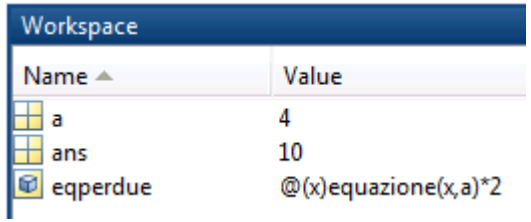
handle=@(input) espressione

```
>> cubomenouno=@(x) x^3-1;
>> cubomenouno(2)
```

ans = 7

- **Funzioni anonime**
  - Sono funzioni non definite in un file, **associate a un function handle**
  - La funzione consiste di una **singola espressione**
  - L'espressione può contenere **altre funzioni**






```
>> a=4;
>> eqperdue=@(x) equazione(x,a)*2;
>> eqperdue(1)
```

ans = 10

- **Nota:** un handle a una funzione può essere definito con una funzione anonima equivalente

```
>> handle1=@sqrt;
```

```
>> handle2=@(x) sqrt(x);
```

Azzeramento di equazioni – fzero

- **fzero**

  ≈ metodo di **bisezione**

  Cerca uno zero **solo se** l'equazione **attraversa lo zero**

  Può essere utilizzato solo per azzerare una **singola variabile**

```
>> fzero(handlefunzione, valoreiniziale)
```

Esempio: azzerare la funzione f(x) = 3^x – 15

script1.m / es1.m

```
function y=es1(x)

y=3^x-15;
```

```
handlees1 = @(x) es1(x);    % creazione handle
```

script1.m / es1.m

```
handlees1 = @es1;    % creazione handle
x_iniziale = 5;      % valore iniziale per fsolve
fzero (handlees1,x_iniziale)
```

```
fzero (@es1,5)  % senza definire variabili per handle e x_iniziale
```

ans = 2.4650

Esercizio: trovare lo zero della retta y = 2x – 3 utilizzando il comando fzero

retta.m / script2.m

```
function y=retta(x)

y=2*x-3;
```

```
>> fzero(@retta,0)
```

ans = 1.5000

```
>> fzero(@(x) 2*x-3,0)
```

Azzeramento di equazioni – fsolve

- **fsolve**

  ≈ metodo della **tangente**

  Cerca uno zero anche se l'equazione non attraversa lo zero

  Può essere utilizzato per **sistemi di equazioni in più variabili**

  >> fsolve(handlefunzione, valoreiniziali)

Esempio: azzerare la funzione f(x) = |x^3 – 15|

script2.m / es2.m

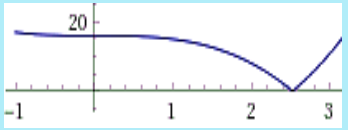
```
function y=es2(x)

y=abs(x^3-15);
```

```
fsolve (@es2,1)
```

→ ans = 2.4662

```
fsolve (@es2,0)
```

→ ans = 0



Nelle risoluzioni numeriche:

- **metodo/risolutore**
- **valore iniziale**
- **opzioni del risolutore**

Esercizio: trovare lo zero della parabola y = (x-5)^2 utilizzando il comando fsolve

script3.m / es3.m

```
function y=es3(x)

y=(x-5)^2;
```

```
fsolve (@es3,0)
```

```
fsolve (@(x) (x-5)^2,0)
```

→ ans = 4.9922

```
options = optimoptions('fsolve');
options.FunctionTolerance=1e-10;
options.OptimalityTolerance=1e-12;
fsolve (@es3,0,options)
```

→ ans = 4.9999

Risoluzione di sistemi di equazioni

Risolvere sistemi di equazioni:

- **fsolve**

```
>>  fsolve(handlefunzione, valoriiniziali)
```

L'obiettivo è trovare le variabili che azzerano un sistema di equazioni.

Si definisce un'**unica funzione contenente tutte le equazioni** del sistema:

- la funzione riceve **più di un valore come input**
- si azzera una funzione che ha **più di un valore come output**

Poiché i risolutori considerano **solo una** variabile in entrata e **una** in uscita per la funzione:

- **non** usare più variabili in ingresso o in uscita
- utilizzare un **unico vettore di elementi** per le variabili in input e per le equazioni in output

```
function [eq1, eq2] = sistemaeq(x1, x2)
```

→

```
function output = sistemaeq(x)
```

Esempio: azzerare il seguente sistema di equazioni

$$e^{3x} + 20y = 0$$
$$4x - 33y = 0$$

sistemaeq.m

```
1   function output = sistemaeq(x)
2
3 -   output(1)=exp(3*x(1))+20*x(2);
4 -   output(2)=4*x(1)-33*x(2);
```

```
>> fsolve(@sistemaeq,[0, 0])

ans =

   -0.2159   -0.0262
```

Risoluzione di equazioni differenziali

$$\begin{cases} y'(t) = f\,(t, y(t)) & \textit{Equazione differenziale} \\ y(t = t_0) = y_0 & \textit{Condizione al contorno} \end{cases}$$

Risolvere equazioni differenziali:

- **Analiticamente**

$$y'(t) = y(t) \longrightarrow y(t) = c \cdot e^t$$

Tuttavia, spesso la soluzione analitica **non esiste**

- **Numericamente**: risoluzione approssimata con **metodi numerici**

$$y'(t) = f\,(t, y(t))$$

$$y'(t) \approx \frac{y(t+h) - y(t)}{h} \longrightarrow y(t+h) \approx y(t) + h \cdot y'(t)$$

$$y(t+h) \approx y(t) + h \cdot f(t, y(t))$$

Metodo di Eulero esplicito: $y_{n+1} \approx y_n + h \cdot f(t_n, y_n)$

Metodo di Eulero esplicito

$$y_{n+1} \approx y_n + h \cdot f(t_n, y_n)$$

valore di y allo **step successivo**

step attuale: valori noti

- Partendo dai valori della **condizione iniziale**

$$y_0 = y(t = t_0) \qquad t_0$$

$$y_1 \approx y_0 + h \cdot f(t_0, y_0) \qquad t_1$$

$$y_2 \approx y_1 + h \cdot f(t_1, y_1) \qquad t_2$$

$$\dots$$

$$y_{n+1} \approx y_n + h \cdot f(t_n, y_n) \qquad t_n$$

$$y_N \approx y_{N-1} + h \cdot f(t_{N-1}, y_{N-1}) \qquad t_N$$

- **Risoluzione iterativa** lungo il dominio di integrazione

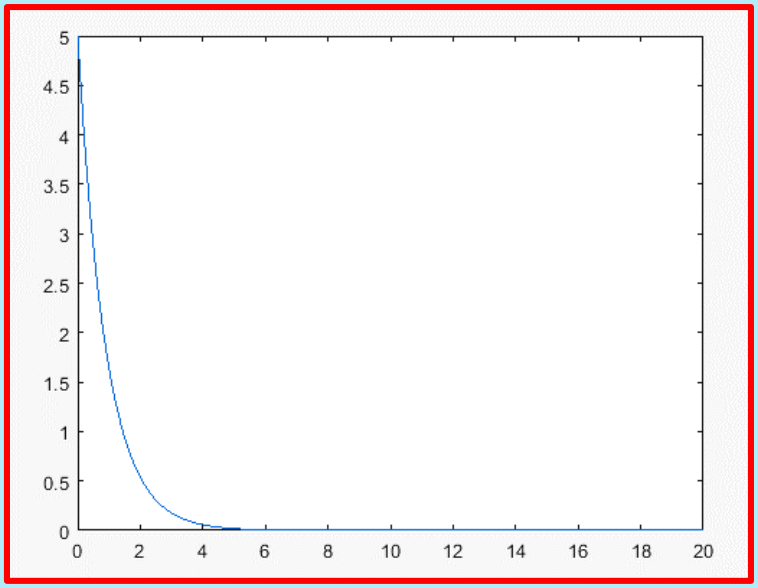
Esercizio 1.1 – Risoluzione con Eulero esplicito

Integrare l'equazione differenziale:

$$\begin{cases} \dot{y} = -\lambda \cdot y & \textit{Equazione differenziale} \\ y(t=0) = 5 & \textit{Condizione al contorno} \end{cases}$$

tra t = 0 e t = 20, con $\lambda = 1$. Utilizzare il metodo di **Eulero esplicito**: $y_{n+1} \approx y_n + h \cdot f(t_n, y_n)$

- **Diagrammare y nel tempo**




```
function dydt = eqdiff(t,y,lambda)

dydt=-lambda*y;
```

```
lambda=1;
h=0.1;
t=0:h:20;
nstep=length(t);
y=zeros(1,nstep);
y(1)=5;
for i=1:nstep-1
   y(i+1)=y(i)+h*eqdiff(t(i),y(i),lambda);
end
plot(t,y)
```

Esercizio 1.1 – Confronto con soluzione analitica

Integrare l'equazione differenziale:

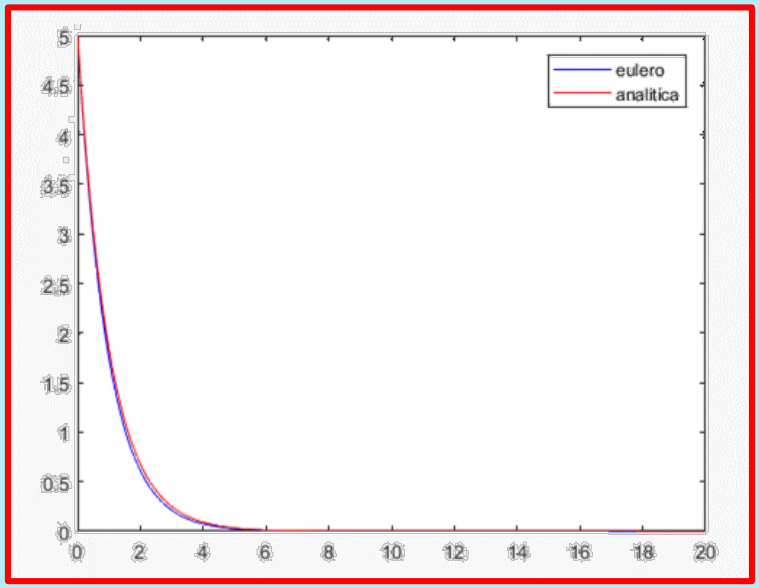
$$\begin{cases} \dot{y} = -\lambda \cdot y \\ y(t=0) = 5 \end{cases}$$

Equazione differenziale

Condizione al contorno

tra t = 0 e t = 20, con $\lambda = 1$. Utilizzare il metodo di **Eulero esplicito**: $y_{n+1} \approx y_n + h \cdot f(t_n, y_n)$

- **Confrontare i risultati con la soluzione analitica:** $y(t) = 5 \cdot \exp(-\lambda \cdot t)$



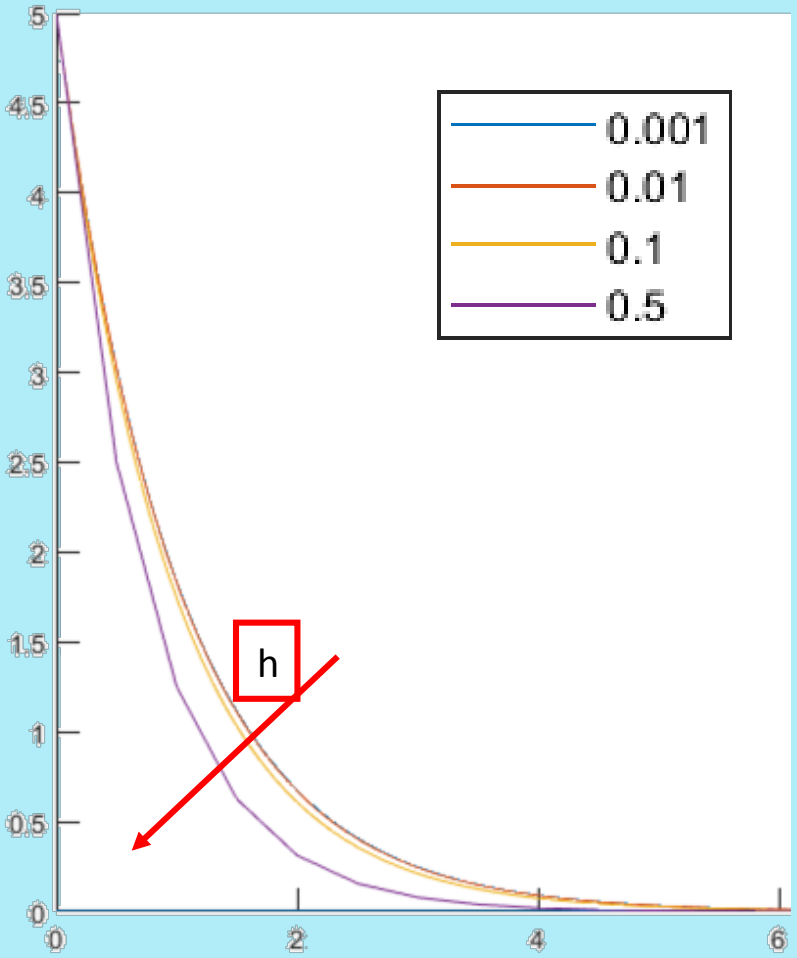

```
...
yan=zeros(1,nstep);
yan(1)=5;
for i=2:nstep
   yan(i)= 5*exp(-lambda*t(i));
end
hold on
plot(t,yan)
legend('eulero','analitica')
```

Esercizio 1.1 – Step di integrazione

Cosa succede variando il passo di integrazione h?

```
lambda=1;
h=[0.001, 0.01, 0.1, 0.5];
for j=1:length(h)
   t=0:h(j):20;
   nstep=length(t);
   y=zeros(1,nstep);
   y(1)=5;
   for i=1:nstep-1
      y(i+1)=y(i)+h(j)*eqdiff(t(i), y(i), lambda);
   end
   hold on
   plot(t,y)
end
legend('0.001','0.01','0.1','0.5')
```

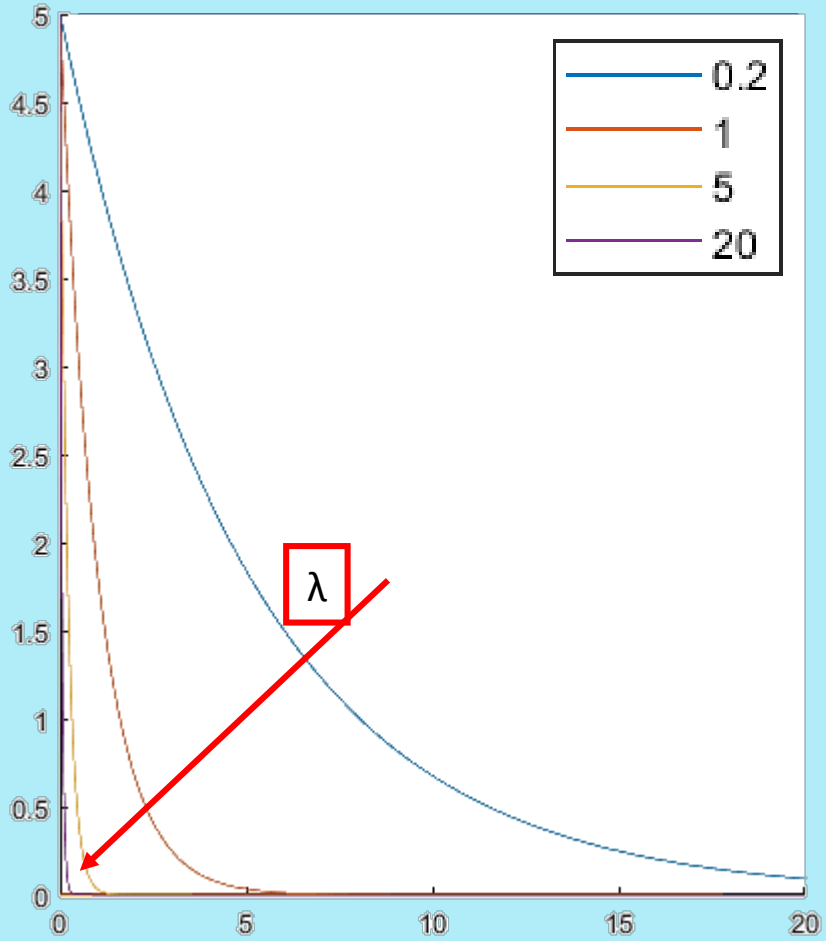
ad es. 0.001 vs 0.01 vs 0.1 vs 0.5



Esercizio 1.1 – Parametro lambda

Cosa succede variando lambda? ad es. 0.2 vs 1 vs 5 vs 20

```
lambda=[0.2, 1, 5, 20];
h=0.001
t=0:h:20;
nstep=length(t);
y=zeros(1,nstep);
y(1)=5;
for k=1:length(lambda)
   for i=1:nstep-1
      y(i+1)=y(i)+h*eqdiff(t(i),y(i),lambda(k));
   end
   hold on
   plot(t,y)
end
legend('0.2','1','5','20')
```



Esercizio 1.1 – Integratore ode45

E' possibile usare integratori numerici di equazioni differenziali ordinarie disponibili in Matlab:

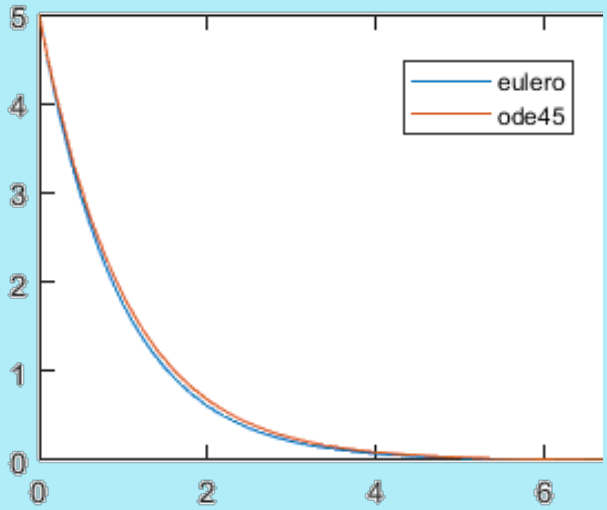
ode suite (**ode45**, ode23, ode113, **ode15s**, ode23s, …)

```
[tout,yout] = ode45( handleeqdiff, estremiintegrazione, valoreiniziale)
```

ad es: `[tode,yode] = ode45( @(t,y)eqdiff(t,y,lambda) , [0 20], 5)`

Integrare la stessa equazione differenziale utilizzando l'integratore di Matlab ode45.

```
handleeqdiff=@(t,y)eqdiff(t,y,lambda);
tspan=[0 20];
yiniziale=5;
[tode,yode]=ode45(handleeqdiff,tspan,yiniziale);
hold on
plot(tode,yode)
legend('eulero','ode45')
```



Esercizio 1.2 – Traccia

In un reattore batch:

$$\mathbf{A} \xrightarrow{r_1 = k_1 c_A} \mathbf{B} \xrightarrow{r_2 = k_2 c_B} \mathbf{C}$$

$$\begin{cases} \dfrac{dc_A}{dt} = -k_1 c_A & c_A(t = t_0) = 3\ mol/m^3 \\ \dfrac{dc_B}{dt} = k_1 c_A - k_2 c_B & c_B(t = t_0) = 0\ mol/m^3 \\ \dfrac{dc_C}{dt} = k_2 c_B & c_C(t = t_0) = 0\ mol/m^3 \end{cases}$$

$$k_1 = 0.5$$

$$k_2 = 0.3$$

Dopo quanto tempo si ottiene la massima produttività di B?

$$\bar{t}\,? \quad c_B(t = \bar{t}) = max$$