1 Le funzioni e la visualizzazione grafica

Supponiamo che venga data una **funzione** reale a variabile reale, eventualmente definita su un intervallo (a, b). Generalmente siamo interessati a

- ⋆ calcolarne i suoi **zeri**
- ⋆ calcolare il suo **integrale** e la sua **derivata**
- ⋆ conoscerne in maniera approssimata l'**andamento grafico**

Come punto di partenza supponiamo che ci venga data una funzione definita in maniera analitica tramite la sua equazione, vogliamo conoscere le istruzioni MATLAB (Octave) per definirla, valutarla in un punto o in un insieme di punti e rappresentarla graficamente.

1.1 Anonymous function

Una **anonymous function** definisce una funzione matematica con l'ausilio di una variabile di tipo function handle @ (*puntatore alla funzione*) secondo la sintassi

```
f=@(arg1,arg2,...,argn)expr
```

Da questo deduciamo che una *anonymous function* può contenere solo una istruzione eseguibile, può accettare variabili in input: `arg1,arg2,...,argn`, che sono le variabili indipendenti su cui essa agisce, mentre `expr` è l'espressione della funzione che vogliamo definire e, volendo, può essere racchiusa tra parentesi tonde o quadre.

Si osservari che una *anonymous function* è una funzione che non viene memorizzata in un file, al contrario delle *functions* definite tramite un file di estensione `.m`.

Esempio 1.1. *Si definisca la funzione* $f(x) = 1/(1+x^2)$ *mediante una anonymous function e la si valuti nel punto* $x = 3$*.*

```
f=@(x) 1/(1+x^2); y=f(3)
```

Se la variabile `x` in cui dobbiamo definire la funzione è un array, le operazioni `* / ^` utilizzate per definire una funzione a valori scalari devono essere sostituite dalle corrispondenti operazioni `.* ./ .^` che lavorano elemento per elemento.

Esempio 1.2. *Si ridefinisca l'istruzione della funzione* $f(x)$ *dell'Esempio 1.1 per valutarla nell'array* `x=[1, 2, -3]`*.*

```
f=@(x) 1./(1+x.^2); x=[1,2,-3]; y=f(x)
```

L'espressione `expr` può contenere dei parametri che non rientrano nell'elenco delle variabili, ma che devono essere in ogni caso assegnati prima di definire la funzione. Fare attenzione perché, se modifichiamo il valore del parametro, dobbiamo ridefinire il *function handle*.

Esempio 1.3. *Si definisca la funzione* $g(x) = a/(1-x^2)$ *mediante una anonymous function, con* $a = 4$ *e la si valuti nel punto* $x = 3$*. Si esegua nuovamente la valutazione modificando il valore del parametro in* $a = -2$*.*

```
a=4; g=@(x) a/(1+x^2); y=g(3)
a=-2; g=@(x) a/(1+x^2); y=g(3) %ridefiniamo il function hande
```

I *function handle* vengono utilizzati anche per definire **funzioni vettoriali**, seguendo la sintassi comune per definire vettori e matrici. In altre parole, gli spazi oppure le virgole sono usate per separare gli elementi di una riga, e punto e virgola per separare le righe. In questo caso MATLAB (Octave) interpreta lo spazio come elemento separatore. Di conseguenza, c'è da fare attenzione alle definizioni errate di funzioni ed evitare spazi bianchi quando è possibile. Si può, ad esempio, sbagliare di definire una funzione scalare racchiusa tra parentesi quadre separando con uno spazio i suoi addendi. In questo caso MATLAB (Octave) interpreterà gli addendi come componenti diverse di una funzione vettoriale.

Esempio 1.4. *Si definisca la funzione vettoriale* $h : \mathbb{R}^2 \to \mathbb{R}^2$*, con* $h(x, y) = [e^x sin(y), x^2 + y]^t$ *mediante una anonymous function e la si valuti in un punto a piacere.*

```
h=@(x,y) [exp(x).*sin(y); x.^2+y]; h(1,pi)
```

Esempio 1.5. *Si definisca la funzione scalare* $k(x) = x^2 - sin(x) - cos(x)$ *mediante una anonymous function, racchiudendo l'espressione* `expr` *tra parentesi quadre, senza lasciare uno spazio bianco tra gli addendi. Successivamente la si riscriva lasciando uno spazio bianco tra gli addendi. La si valuti in* $x = \pi/4$ *in entrambi i casi per provare come nel secondo caso MATLAB (Octave) la abbia interpretata come una funzione* $k : \mathbb{R} \to \mathbb{R}^3$*.*

```
k=@(x) [x^2-sin(x)-cos(x)];   k(pi/4)
k_prova=@(x) [x^2  -sin(x)  -cos(x)];   k_prova(pi/4)
```

2 La visualizzazione grafica

Supponiamo di avere una funzione `f` che è stata definita come un *function handle* e un array di due elementi `interv=[`x_0 x_1`]` i cui valori sono gli estemo di un intervallo di definizione della funzione. Per visualizzare il grafico di $f(x)$ nell'intervallo `interv` si utilizza il comando `fplot(f,interv)`. In alternativa, si può direttamente invocare l'anonymous function all'interno del comando `fplot`.

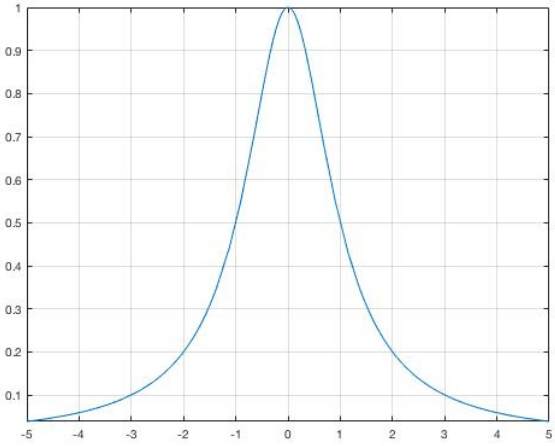

Figure 1: Grafico della funzione $f(x) = 1/(1+x^2)$ nell'intervallo $[-5, 5]$.

Esempio 2.1. *Si definisca la funzione* $f(x) = 1/(1+x^2)$ *mediante una anonymous function e si visualizzi il suo grafico nell'intervallo* $[-5, 5]$ *facendo uso del comando* `fplot`*, come rappresentato nella Figura 1.*

```
f=@(x) 1/(1+x^2); interv =[-5 5]; fplot(f,interv);
%alternativamente fplot(@(x) 1/(1+x^2),[-5 5]);
```

I grafici prodotti da MATLAB (Octave) sono una rappresentazione approssimata, a meno dello 0.2%, del grafico di `f` e sono ottenuti campionando la funzione su un opportuno insieme non equispaziato di ascisse. Per aumentare l'accuratezza della rappresentazione è sufficiente richiamare `fplot` come `fplot(f,int,tol,n,LineSpec)`. In questo caso `tol` indica la tolleranza relativa richiesta, parametro $n \geq 1$ indica che il grafico della funzione sia disegnato utilizzando almeno $n+1$ punti, mentre `LineSpec` è una stringa che specifica invece il tratto grafico (od il colore) della linea utilizzata nel tracciamento del grafico. Per usare i valori di default (ovvero preassegnati) per una qualsiasi di tali variabili si usa la matrice vuota (`[ ]`). Il comando `grid on` dopo il comando `tol`, introduce un griglia di riferimento.

Per rappresentare graficamente sull'intervallo `interv=[`x_0`,` x_1`]` una funzione memorizzata nel file miafunzione.m, la sintassi corretta da utilizzare è `fplot(@miafunzione,interv)`, laddove il carattere @ genera il *function handle* associato alla *user-defined function* miafunzione.m In alternativa, si può utilizzare l'istruzione `fplot('miafunzione',interv)`.

Per visualizzare funzioni in una variabile o grafici $2D$ MATLAB (Octave) utilizza il comando `plot`. Dati due array `x=` $[x_1, \ldots, x_n]$ e `x=` $[y_1, \ldots, y_n]$ della stessa lunghezza, il comando `plot(`x, y`)` stampa in output le coppie di punti corrispondenti alle coordinate dei due vettori (x_i, y_i) (per $i = 1, \ldots, n$), collegati con una linea continua ottenuta mediante interpolazione lineare. Comandi utili per generare vettori riga sono

- ⋆ `linspace(a,b,n)` genera un vettore riga di n punti equispaziati da a a b, con gli estremi a e b compresi

- `:` è usato per generare vettori riga:
  - senza specificare l'incremento: `x=a:b`
  - con incremento specificato: `x=a:h:b`
  - con decremento specificato: `x=b:-h:a`

Il comando `plot` può essere usato in alternativa a `fplot`, per visualizzare il grafico di una funzione matematica. Le sue istruzioni richiedono che funzione sia stata prima valutata su un insieme di ascisse, generata, ad esempio, dei comandi citati sopra. Prima di vedere degli esempi ed esercizi, richiamiamo come speficare elementi dello **stile del grafico**. In questo caso si utilizza il comando `plot(x,y,'stile')`, laddove 'stile' è una stringa opzionale che contiene al massimo tre caratteri che descrivono

- il colore
- i marcatori di punti
- lo stile della linea

selezionando tra le opzioni

r	red	.	punti	-	linea solida
g	green	o	cerchietti	:	linea puntinata
b	blue	x	crocette	–	linea tratteggiata
w	white	+	più	-.	linea tratto-puntinata
m	magenta	*	stelline	(none)	nessuna linea
c	cyan	s	quadrato		
y	yellow	d	diamante		
k	black	v	triangolo (sotto)		
		∧	triangolo (sopra)		
		<	triangolo (sinistra)		
		>	triangolo (destra)		
		p	pentagramma		
		h	esagramma		

Altri comandi utili sono i seguenti

- `xlabel('...')` aggiunge testo sotto l'asse delle x
- `ylabel('...')` aggiunge testo lungo l'asse delle y
- `title('...')` aggiunge un titolo sopra la figura
- `legend(...)` aggiunge una leggenda al grafico
- `axis(...)` riscala gli assi del grafico
- `grid` sovrappone al grafico un grigliato

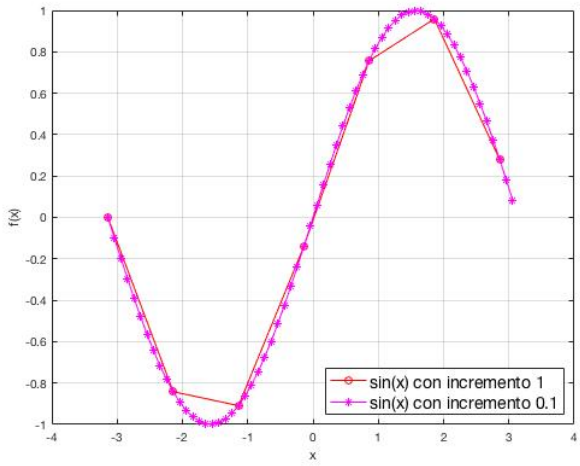

Figure 2: Grafico della funzione $y = sin(x)$ nell'intervallo $[-\pi, \pi]$ costruito su insiemi di punti equispaziati con incrementi diversi.

È possibile visualizzare più funzioni contemporaneamente sullo stesso grafico utilizzando tramite il comando `plot(x1,y1,x2,y2....xn,yn)`, laddove `x1,y1` sono gli array contententi rispettivamente le ascisse e ordinate del primo grafico e così via fino a `xn,yn`.

Esempio 2.2. *Produrre il grafico della funzione* $f(x) = sin(x)$ *utilizzando il comando* `plot`*, dopo averla definita su un insieme equispaziati di punti tra* $-\pi$ *ed* π*, specificando come incremento* 1 *e successivamente* 0.1*. Si commentino le differenze osservate tra i due grafici.*

```
x1= -pi:1:pi;
x2= -pi:0.1:pi;
y1=sin(x1);
y2=sin(x2);
plot(x1,y1,'ro-',x2,y2,'m*-')
xlabel('x');
ylabel('f(x)');
grid on
legend('sin(x) con incremento 1','sin(x) con incremento 0.1',
    ... 'Location','SouthEast','FontSize',14)
```

Chiaramente, dalla Figura 2 si osserva che campionando la funzione su un insieme di poche ascisse, la rappresentazione grafica sarà meno accurata.

Esempio 2.3. *Produrre il grafico della funzione* $g(x) = e^x sin^2(x) - x$ *utilizzando il comando* `plot`*, dopo averla definita su un insieme di* 100 *ascisse equispaziate tra* -2 *e* 3 *facendo uso del comando* `linspace`*.*

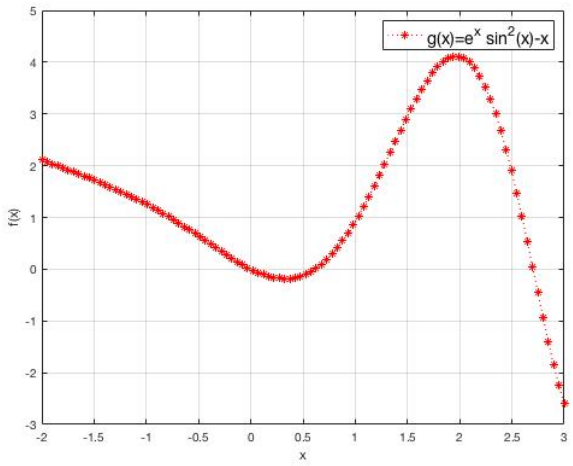

Figure 3: Grafico della funzione $g(x) = e^x sin^2(x) - x$ nell'intervallo $[-2, 3]$.

```
x=linspace(-2,3,100);
y= exp(x).*sin(x).^2- x;
plot(x,y,'r*:')
xlabel('x')
ylabel('f(x)')
grid on
legend('g(x)=e^x sin^2(x)-x','FontSize',14)
```

Esercizio 2.1. *Si scriva uno script per disegnare il grafico della funzione*

$$f(x) = \begin{cases} e^{-x} & \text{se} \quad -2 \leq x \leq 0 \\ 1 - 3x^3 & \text{se} \quad 0 < x \leq 2. \end{cases} \tag{1}$$

Soluzione:

```
x1=linspace(-2,0);
y1=exp(-x1);
x2=linspace(0,2);
y2=1-3*x2.^3;
x=[x1 x2];
y=[y1 y2];
plot(x,y,'b-.')
xlabel('x')
ylabel('f(x)')
grid on
```

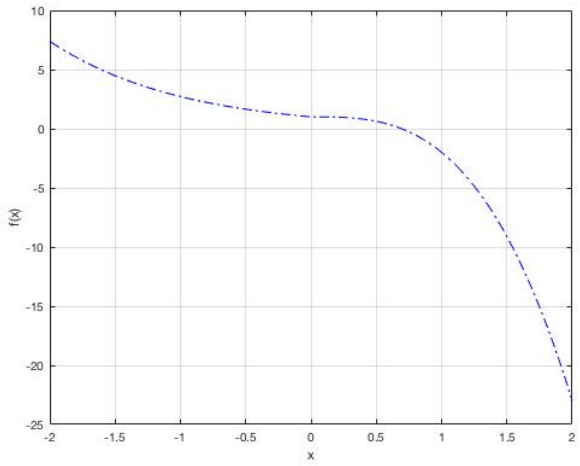

Figure 4: Grafico della funzione definita dall'Equazione (1).

Il comando `xlabel(subplot(m,n,p))` si utilizza per visualizzare più grafici sulla stessa finestra ma ognuno a se stante. Il paramentro m indica in quante parti verticali dividere lo schermo, il parametro n indica in quante parti orizzontali, il parametro p indica in quale parte eseguire il plot successivo. Richiamiamo inoltre altre funzioni che torneranno utili per creare grafici sono

- ⋆ `loglog` plot x-y in scala logaritmica
- ⋆ `semilogx` plot x-y in scala logaritmica solo per le x
- ⋆ `semilogy` plot x-y in scala logaritmica solo per le y
- ⋆ `polar` polar plot in cordinate polari

Esercizio 2.2. *Utilizzando un ciclo* for*, scrivere uno script che disegni i grafici delle funzioni* $(x-2k)^3$*, per* $k = 1, \ldots, 5$ *nell'intervallo* $[0, 15]$*, come rappresentato nella Figura 5.*

Soluzione:

```
x=linspace(0,15);
for i=1:5
    plot(x,(x-2*i).^3)
    hold on %mantiene lo stesso plot e sovrappone i grafici
            %successivi al grafico corrente
end
grid on
xlabel('x')
ylabel('f(x)')
legend('(x-2)^3','(x-4)^3', '(x-6)^3', '(x-8)^3', ...
       '(x-10)^3','Location','NorthWest','FontSize',14)
```

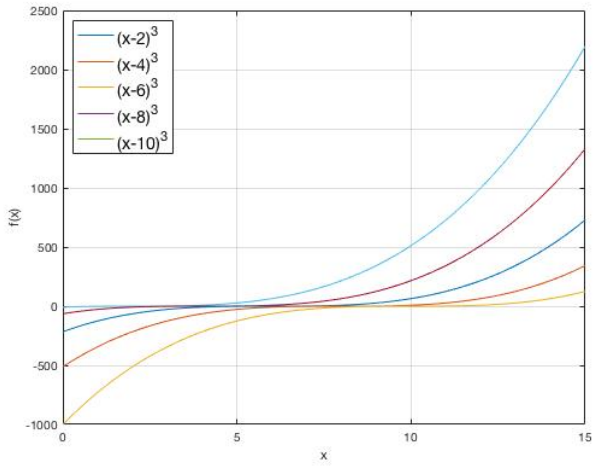

Figure 5: Grafici delle funzioni $(x-2k)^3$, per $k = 1, \ldots, 5$.

2.1 Gli zeri

Un numero reale o complesso α viene chiamato **zero** della funzione $f(x)$ se $f(\alpha) = 0$, ovvero se α soddisfa l'equazione $f(x) = 0$. Uno zero viene detto *semplice* se $f'(\alpha) \neq 0$, oppure, *multiplo* in caso contrario. Conoscere il grafico di una funzione è utile per dedurre informazioni sui suoi zeri. In particolare, dal grafico di una funzione è possibile ricavare informazioni, seppur in maniera approssimata, dei suoi zeri reali.

Calcolare in modo diretto tutti gli zeri di una data funzione non è sempre possibile. Fanno eccezione i polinomi a coefficienti reali di grado $n \leq 2$, per cui abbiamo le formule esplicite. Tuttavia, è anche noto che, se $n \geq 5$, non esistono formule generali che con un numero finito di operazioni consentano di calcolare le radici di un polinomio qualunque di grado n. Il numero di zeri di una funzione neanche è determinabile in modo elementare a priori. Fanno eccezione i polinomi per i quali il numero di radici (reali o complesse) è uguale al grado del polinomio stesso. Si ricordi che se un polinomio a coefficienti reali di grado $n \geq 2$ ammette una radice complessa $\alpha = x + iy$ con $y \neq 0$, allora esso deve presentare come radice anche il complesso coniugato $\bar{\alpha} = x - iy$ di α.

Il comando di MATLAB (Octave) `fzero(f,x0)` calcola uno zero (non tutti) di una funzione `f` vicino ad un certo valore `x0`, reale o complesso. L'intervallo in cui viene cercato lo zero dal programma viene visualizzato come output, assieme allo zero.

Il comando `fzero(f,[x0,x1])` cerca uno zero di `f` nell'intervallo di estremi `[x0,x1]`, purché `f` cambi di segno tra `x0` e `x1`.

Esempio 2.4. *Si rappresenti graficamente la funzione* $f(x) = x^2 - 1 + e^x$ *per verificare che essa presenta due zeri nell'intervallo* $(-1, 1)$*, come si può osservare dalla Figura 6. Si faccia uso del comando* `fzero(f,x0)` *per calcolare lo zero poù vicino a* -1 *e quello più vicino a* 1*.*

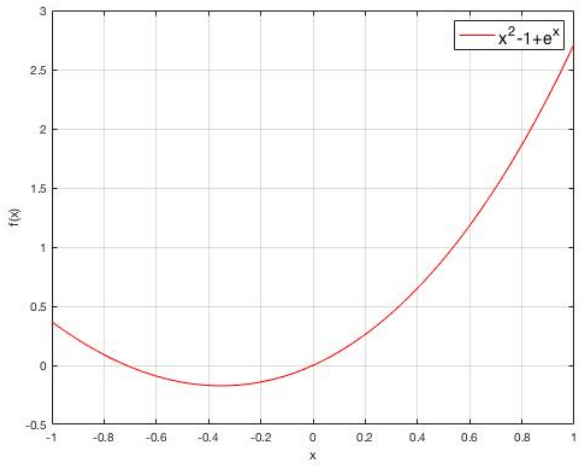

Figure 6: Grafico della funzione $f(x) = x^2 - 1 + e^x$ nell'intervallo $(-1, 1)$.

Dopo aver osservato graficamente che uno zero si trova in $(-1, -0.2)$ *e l'altro in* $(-0.2, 1)$*, si calcolino utilizzando il comando* `fzero(f,[x0,x1])`.

```
f=@(x) x.^2-1+exp(x);
x=linspace(-1,1);
plot(x,f(x),'r')
xlabel('x')
ylabel('f(x)')
legend('x^2-1+e^x','FontSize',14)
grid on
primo_zero=fzero(f,-1)
secondo_zero=fzero(f,1)
primo_zero_altern=fzero(f,[-1 -0.2])
secondo_zero_altern=fzero(f,[-0.2 1])
```

Esercizio 2.3. *Si utilizzi il comando* `fzero` *per trovare i punti di intersezione tra i grafici delle funzioni* $f(x) = ln(x)$ *e* $g(x) = e^{-x}$*, che risolvono l'equazione*

$$ln(x) - e^{-x} = 0. \tag{2}$$

Soluzione:

```
x=linspace(0.5, 2.5, 1000)
y=log(x);
z=exp(-x);
d=y-z
```

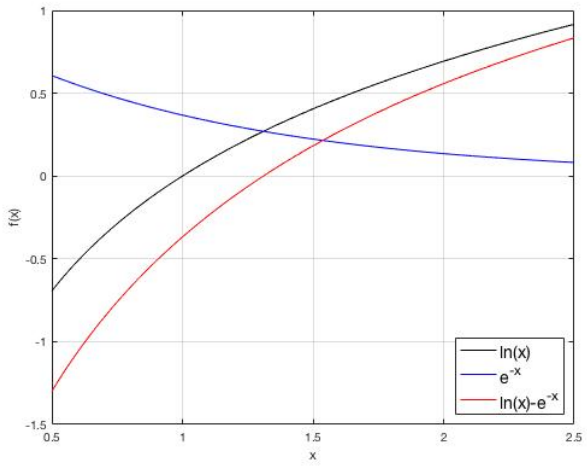

Figure 7: Intersezione tra i grafici delle funzioni $ln(x)$ ed e^{-x}.

```
plot(x,y,'k',x,z,'b',x,d,'r')
grid on
hold on
xlabel('x')
ylabel('f(x)')
legend('ln(x)','e^{-x}','ln(x)-e^{-x}', 'Location','SouthEast',
       ... 'FontSize',14)
```

Ultima versione aggiornata: 22 Aprile 2020

Link alle lezioni precedenti:
Lezioni 1-2

Referenze

1. MATLAB® The language of technical computing: computation, visualization, programming, The MathWorks Inc
2. MATLAB: An introduction with applications, A. Gilat, Wiley
3. Scientific Computing with MATLAB and Octave, A. Quarteroni, , F. Saleri, P. Gervasio, Springer
4. MATLAB Guide1 D. J. Higham and N. J. Higham, SIAM
5. Numerical Computing with MATLAB, C. Moler, SIAM