

Metodo di bisezione con zero mal condizionato

Istruzioni trattate: poly, ones, horner, gcf, gca.

Nella prima parte di questa esercitazione vedremo un esempio concreto di *problema mal condizionato* relativo al calcolo di uno zero di una funzione. Applicheremo la procedura **Bisezione**, in una versione leggermente modificata rispetto a quella presentata a lezione, a due realizzazioni, *diverse ma vicine*, della funzione:

$$f(x) = (x - 2)^{13}$$

Nella seconda parte descriveremo, in relazione ad un disegno prodotto nella prima parte, alcune nozioni relative agli *oggetti grafici* di Scilab.

Prima parte

Consideriamo la seguente realizzazione del metodo di *bisezione*:

```
function [z,s,d,iter] = Bisezione(f,a,b)
//
// Applica il metodo di bisezione alla funzione f a partire dall'intervallo
// [a,b]. L'iterazione si arresta quando si è trovato uno zero di f oppure
// l'intervallo costruito ha estremi elementi consecutivi di F(2,53).
//
// z: approssimazione suggerita.
// s,d: estremo sinistro e destro, rispettivamente, dell'ultimo intervallo
//      calcolato.
// iter: numero di iterazioni effettuate.
//
iter = 0;
if ( f(a) == 0 | f(b) == 0 | sign(f(a)) == sign(f(b)) ) then
    error('la funzione non assume valori di segno opposto agli estremi');
else
    s = a; d = b;
    z = (s+d)/2;
    while ( f(z) ~= 0 & d > nearfloat('succ',s) ),
        if sign(f(s)) ~= sign(f(z)) then d = z;
        else s = z; end;
        z = (s+d)/2;
        iter = iter + 1;
    end;
end;
endfunction
```

In questa realizzazione si è scelto di arrestare la costruzione della successione se *si è trovato uno zero della funzione f* oppure *l'intervallo costruito ha estremi elementi consecutivi di $M = F(2,53)$* . In entrambi i casi è inutile proseguire l'iterazione.

Definiamo le funzioni F e G alle quali applicheremo la procedura **Bisezione**:

```
function y = F(x)
    y = (x - 2) .^ 13;
endfunction
//
function y = G(x)
    p = poly(2*ones(1,13),'x');
    y = horner(p,x);
endfunction
```

La prima, `F`, è la *realizzazione ingenua* della funzione f . Per capire cosa è `G`, descriviamo i tre comandi utilizzati nella sua definizione.

- `poly`

Questa *funzione predefinita* è una realizzazione della funzione che restituisce *il polinomio* che ha *coefficienti* assegnati o *il polinomio monico* che ha *radici* assegnate. Precisamente, assegnato un vettore v , riga o colonna, a componenti $v_1, \dots, v_n$:

$$\text{poly}(v, 'X')$$

restituisce *un'approssimazione del polinomio monico in X di grado n che ha radici $v_1, \dots, v_n$* ¹
 $(X - v_1) \cdots (X - v_n)$;

$$\text{poly}(v, 'X', 'coeff')$$

restituisce *il polinomio in X di grado al più $n-1$ e coefficienti $v_1, \dots, v_n$: $v_1 + v_2 X + \cdots + v_n X^{n-1}$* .
 Ad esempio:

```
-->poly([1,2,3], 'X')
ans =
```

$$-6 + 11X - 6X^2 + X^3$$

```
-->poly(3*[1,2,3], 's', 'coeff')
ans =
```

$$3 + 6s + 9s^2$$

Si osservi che l'oggetto restituito dalla funzione `poly` è di tipo *polinomio*, che *non è una funzione*. In particolare: il valore della funzione associata ad un oggetto di tipo *polinomio* si calcola, in *Scilab*, con una specifica *funzione predefinita* (la *funzione predefinita horner*, descritta sotto).

- `ones`

Questa *funzione predefinita* restituisce, quando applicata ad una coppia di numeri interi positivi r e c , la *matrice di dimensione $r \times c$ in cui ciascun elemento ha valore 1*.

- `horner`

Questa *funzione predefinita* è una realizzazione della funzione che, dati *un polinomio p* ed una matrice x , restituisce la matrice, della stessa dimensione di x , di elemento i, j il valore del polinomio p in x_{ij} calcolato con il *metodo di Horner*². Ad esempio:

```
-->p = poly([1,2,3], 's')
p =
```

$$-6 + 11s - 6s^2 + s^3$$

```
-->horner(p, [1,2,3])
ans =
```

$$0. \quad 0. \quad 0.$$

```
-->horner(p, eye(2,2))
ans =
```

¹La stringa `'X'` può essere sostituita da una stringa di caratteri quasi arbitraria.

²Per approfondire, vedere: https://en.wikipedia.org/wiki/Horner's_method#Description_of_the_algorithm.

```

      0.   - 6.
    - 6.    0.

-->p(1)
ans  =

      2    3
    - 6 + 11s - 6s + s

-->p(2)
!--error 21
Indice non valido.

```

Dunque: si verifica facilmente che anche G è una realizzazione della funzione f .

Applichiamo la procedura `Bisezione` ad F .

```

-->a = 0; b = 3; [z,s,d,iter] = Bisezione(F,a,b);

-->printf('\n   z = %10.9e , F(z) = %3.2e , iter = %d , ampiezza = %3.2e\n', ...
        z,F(z),iter,d-s);

z = 2.000000000e+00 , F(z) = 0.00e+00 , iter = 51 , ampiezza = 1.33e-15

```

La procedura dichiara (correttamente) che $z = 2$ è uno zero della funzione F . Ci domandiamo se sia possibile dedurre che 2 è un'approssimazione dello zero di f con errore assoluto non superiore a circa $1.33 \cdot 10^{-15}$.

Questa seconda deduzione è corretta: secondo il nostro modello le *funzioni predefinite* $\ominus$ e $\cdot^\wedge$ restituiscono l'arrotondato del valore esatto in $F(2, 53)$ e quindi per ogni $\xi \in F(2, 53)$ esiste $\theta \in \mathbb{R}$ tale che:

$$F(\xi) = f(\xi)(1 + \theta) \quad \text{e} \quad |\theta| < (1 + u)^{14} - 1 \approx 14u \approx 1.554 \cdot 10^{-15}$$

Dunque: F è un'approssimazione sufficientemente accurata di f da garantire che F ed f hanno sempre lo stesso *segno*. Se il calcolatore fosse in grado di calcolare i valori necessari della funzione f , otterremmo:

$$\text{Bisezione}(f, a, b) = \text{Bisezione}(F, a, b)$$

Applichiamo adesso la procedura `Bisezione` a G .

```

-->[z,s,d,iter] = Bisezione(G,a,b);

-->printf('\n   z = %10.9e , G(z) = %3.2e , iter = %d , ampiezza = %3.2e\n', ...
        z($),G(z($)),iter,d-s);

z = 2.177268961e+00 , G(z) = -4.66e-10 , iter = 52 , ampiezza = 4.44e-16

```

La procedura dichiara (correttamente) che la funzione G assume valori di segno opposto agli estremi dell'intervallo $[s, d]$ contenente z e di ampiezza circa $4.44 \cdot 10^{-16}$. Ci domandiamo se sia possibile dedurre che z è un'approssimazione di uno zero di f con errore assoluto non superiore a circa $4.44 \cdot 10^{-16}$.

La distanza tra 2 (zero di f) e z è circa $0.177 > 4.44 \cdot 10^{-16}$ e quindi, se la deduzione fosse corretta, lo zero di f approssimato da z sarebbe *certamente* diverso da 2 . Ma f non ha zeri diversi da 2 : la deduzione non può essere corretta.

Ciò che rende non corretta la deduzione è che (a) *il calcolo dello zero di f è mal condizionato*: esistono perturbazioni di f di misura arbitrariamente piccola che generano funzioni con qualche zero che dista da quello di f molto più della misura della perturbazione, e (b) la funzione G è la restrizione ad M di una di tali perturbazioni.

Si può avere un'idea della misura della perturbazione di f calcolata usando G , nei pressi di 2 , disegnando un'approssimazione del grafico della funzione $|F \ominus G|$:

```
-->xi = linspace(-0.23,0.23,700)'; xi = xi + 2;

-->clf(); plot2d(xi,abs(F(xi)-G(xi)),style = 5); xgrid();

-->xlabel('xi'); ylabel('| F(xi) - G(xi) |');
```

Si ottiene il disegno riportato in Figura 1. Dal disegno risulta che per molti elementi $\xi \in M$ la

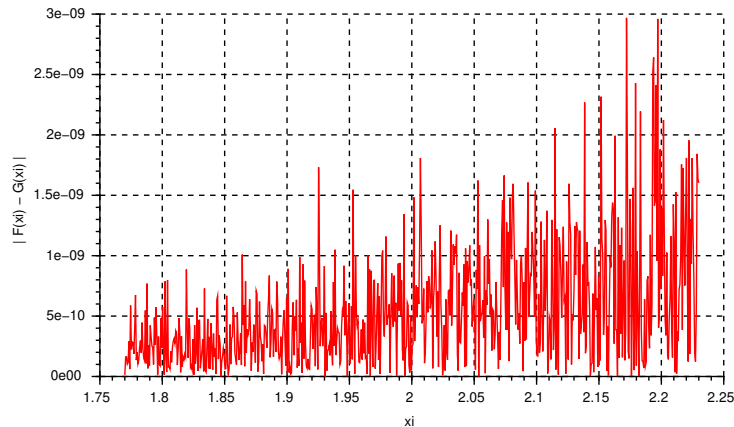


Figura 1: Approssimazione del grafico di $|F(\xi) - G(\xi)|$.

distanza tra $F(\xi)$ e $G(\xi)$ supera 10^{-9} . Dallo studio di F fatto sopra si può dedurre che per ogni $\xi \in [2 - 0.23; 2 + 0.23] \cap M$ si ha: $|F(\xi) - f(\xi)| \leq 10^{-23}$ e quindi, essendo $10^{-9} \gg 10^{-23}$, si può concludere che per molti valori di ξ la distanza tra $G(\xi)$ e $f(\xi)$ supera 10^{-9} . Anche supponendo che la misura effettiva della perturbazione di f calcolata usando G sia 10^{-9} non è ragionevole sperare di approssimare lo zero di f con errore assoluto inferiore a $\sqrt[13]{10^{-9}} \approx 0.2$. Questa conclusione si conferma disegnando approssimazioni del grafico di $F \approx f$ e di G nei pressi di 2 su uno stesso piano cartesiano (il significato dei comandi dell'ultima riga sarà spiegato nella seconda parte):

```
-->clf(); plot2d(xi,[G(xi),F(xi)],style = [5,2]); xgrid();

-->xlabel('xi'); legend('G(xi)', 'F(xi)');
```

```
-->assi = gca(); assi.children(2).children(1).thickness = 2;
```

Si ottiene il disegno riportato in Figura 2. Dal disegno risulta che la funzione G assume certamente valori positivi e valori negativi negli intervalli $[1.8; 2)$ e $(2; 2.2]$, e questo spiega il risultato della procedura *Bisezione* applicata a G .

In conclusione, il calcolo dello zero di f è mal condizionato e:

- esistono *piccole* perturbazioni di f con *qualche* zero *distante* da quello di f , e G fornisce valori di una di esse;
- esistono *piccole* perturbazioni di f che hanno *solo* zeri *vicini* a quello di f , e F fornisce valori di una di esse.

Seconda parte

Scilab consente all'utilizzatore di *creare* e *gestire* molti *oggetti grafici* ciascuno dei quali possiede varie *proprietà* che possono essere *visualizzate* e *modificate*. Ad ogni istante, gli oggetti esistenti (che l'utilizzatore ha creato) sono organizzati in una o più *strutture ad albero*, una per ogni finestra grafica esistente, e con una opportuna *visita* è possibile accedere alle proprietà di ciascun oggetto. Gli oggetti grafici sono creati con appositi comandi che agiscono su quella che è la *finestra grafica corrente* all'istante in cui il comando è eseguito.

Consideriamo, ad esempio, il seguente dialogo:

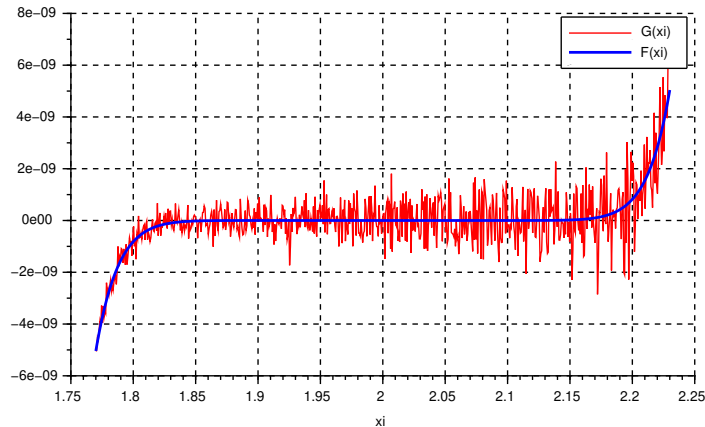
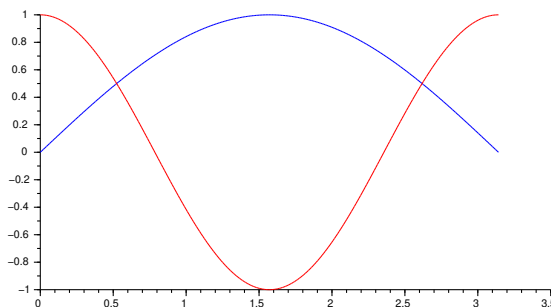


Figura 2: Approssimazioni del grafico di $F(\xi)$ e di $G(\xi)$.

```
-->x = linspace(0,%pi,300)';
```

```
-->plot2d(x,[sin(x),cos(2*x)],style = [2,5]);
```

Il comando `plot2d`: (a) crea la Finestra grafica numero 0, (b) la dichiara *finestra grafica corrente*, e (c) in essa crea il disegno riportato, a sinistra, nella Figura 3. Nella stessa Figura, a destra, è riportato schematicamente l'albero che organizza gli oggetti grafici esistenti nella Finestra grafica numero 0 (vedremo tra poco come ottenere l'albero associato ad una finestra grafica).



```
Figure
  Axes
    Compound
      Polyline(1) , Polyline(2)
```

Figura 3: Contenuto della Finestra grafica numero 0 e albero associato.

L'albero ha per *radice* un oggetto di tipo **Figure** che ha un figlio, oggetto di tipo **Axes**. Quest'ultimo ha un figlio, oggetto di tipo **Compound** che, infine, ha due figli, oggetti di tipo **Polyline**. I figli di un oggetto sono *numerati* in ordine di *anzianità crescente*: il figlio numero uno è l'ultimo creato, il numero due l'ultimo creato dei rimanenti ecc. Dunque gli oggetti **Polyline(1)** e **Polyline(2)**, sono, rispettivamente, la curva rossa (creata per seconda) e quella blu (creata per prima).

Per accedere alle proprietà di un oggetto grafico occorre creare un *puntatore all'oggetto grafico*. In *Scilab* un puntatore di questo tipo è chiamato *graphic handle*.

- **gcf**

Questa *funzione predefinita* restituisce un puntatore all'oggetto di tipo **Figure** radice dell'albero associato alla *finestra grafica corrente*.

- **gca**

Questa *funzione predefinita* restituisce un puntatore all'oggetto di tipo **Axes** corrispondente agli *assi correnti*.

Il comando:

```
FG0 = gcf();
```

crea la variabile di nome **FGO**, puntatore alla radice dell'albero associato alla *finestra grafica corrente*: la Finestra grafica numero 0. Se chiediamo a *Scilab* di mostrare il valore della variabile **FGO** otteniamo *un elenco di tutte le proprietà dell'oggetto puntato con relativo valore*. L'elenco inizia con:

```
FGO =
```

```
Handle of type "Figure" with properties:
```

```
=====
```

```
children: "Axes"
```

Le proprietà di un oggetto di tipo **Figure** sono elencate e descritte nella pagina di *help* relativa al termine *figure properties*. A ciascuna delle proprietà dell'oggetto puntato dalla variabile **FGO** è associata una variabile di nome **FGO.<nome della proprietà>** il cui valore controlla la proprietà. Ad esempio:

```
-->FGO.figure_size
```

```
ans =
```

```
1442.    852.
```

```
-->FGO.children.grid
```

```
ans =
```

```
- 1.    - 1.
```

Il valore della variabile **FGO.figure_size** è una riga di due numeri interi che rappresentano, in *pixel*, la *larghezza* e l'*altezza* della finestra grafica. La variabile **FGO.children.grid**, invece, controlla la proprietà di nome **grid** del *figlio* dell'oggetto puntato da **FGO**. La variabile **FGO.children** esiste – infatti un oggetto di tipo **Figure** ha una proprietà di nome **children** – ed il valore è un puntatore al figlio dell'oggetto puntato da **FGO**: un oggetto di tipo **Axes**. Quest'ultimo ha una proprietà di nome **grid** che controlla la presenza della griglia. Il valore della variabile **FGO.children.grid** è una riga di due numeri interi ciascuno dei quali controlla la parte di griglia parallela ad uno degli assi con la seguente codifica: -1 significa griglia assente, $n \neq 0$ significa griglia presente disegnata nel colore opportuno. Ad esempio, il comando:

```
-->FGO.children.grid = [3,1];
```

aggiunge la griglia al disegno come mostrato in Figura 4.

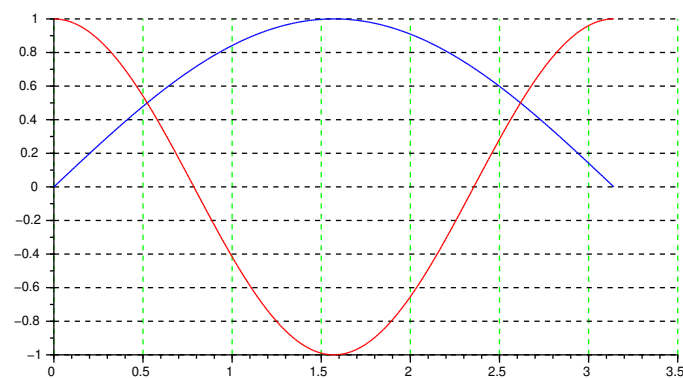


Figura 4: **FGO.children.grid = [3,1]**.

Una delle proprietà di un oggetto di tipo **Polyline** si chiama **thickness** ed il valore è un numero intero che specifica, in *pixel*, lo spessore della linea. Lo spessore della linea rossa della figura nella Finestra grafica numero 0 vale:

```
-->FG0.children.children.children(1).thickness
ans =

    1.
```

e per cambiarlo:

```
-->FG0.children.children.children(1).thickness = 2;
```

L'effetto è visibile nella Figura 5 che riproduce il contenuto della Finestra grafica 0. Si osservi

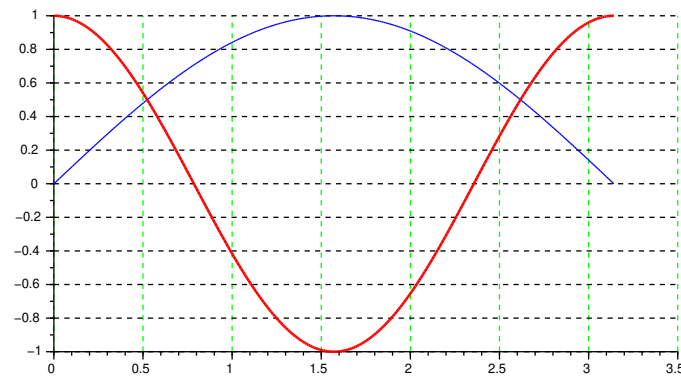


Figura 5: `FG0.children.children.children(1).thickness = 2.`

che la curva rossa è sovrapposta a quella blu ed entrambe sono sovrapposte alla griglia: *ad ogni istante l'ultimo oggetto creato è sovrapposto a quelli esistenti*. La griglia, visibile o no, è creata contestualmente alla finestra grafica.

Per ottenere l'albero associato alla *finestra grafica corrente* possiamo utilizzare due metodi:

- Con pazienza, si esplora l'albero nodo per nodo, a partire dalla radice, facendosi mostrare ogni volta le proprietà dell'oggetto e leggendo il valore della proprietà `children`.
- Si seleziona la finestra grafica che interessa e dal menu a tendina *Modifica* (rispettivamente: *Edit* se "Scilab parla inglese") in alto a sinistra si seleziona *Proprietà della figura* (rispettivamente: *Figure properties*). Si apre una finestra *Figure editor* dove, insieme ad altre cose, è visualizzato l'albero associato alla finestra grafica. Tramite questa stessa finestra è possibile gestire le proprietà degli oggetti grafici presenti nella finestra grafica.

Come esempio finale, esaminiamo il disegno riportato in Figura 2. La Figura 6 riporta, schematicamente, l'albero associato alla finestra grafica che contiene il disegno. La proprietà modificata con i comandi:

```
assi = gca(); assi.children(2).children(1).thickness = 2;
```

è lo spessore della linea blu (il grafico di F). Infatti il figlio numero *due* dell'oggetto di tipo `Axes` è quello di tipo `Compound` – creato *prima* dell'altro figlio – ed il figlio numero *uno* dell'oggetto di tipo `Compound` è il grafico di `F` – creato *dopo* l'altro figlio.

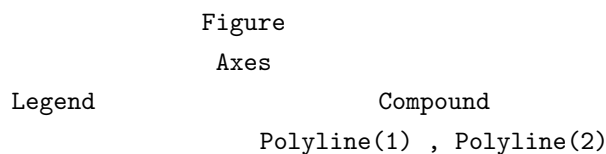


Figura 6: Albero relativo al disegno riportato in Figura 2.

Esercizi

1. Riguardo alla procedura **Bisezione**: (a) spiegare perché se l'intervallo costruito ha estremi elementi consecutivi di $F(2, 53)$ allora è inutile proseguire l'iterazione e (b) discutere l'efficacia del criterio d'arresto.
2. Sia u la precisione di macchina in $F(2, 53)$. Verificare che dopo l'assegnamento:

$$\mathbf{p} = \text{poly}([1, u, -1], 'x')$$

il valore di $\mathbf{p}$ *non* è il polinomio monico q di radici $1, u, -1$. Discutere poi, utilizzando l'opportuna pagina dell'*help* di *Scilab*, il seguente dialogo:

```
-->coeff(p,0) == u
```

```
ans =
```

```
T
```

```
-->coeff(p,1) == -1
```

```
ans =
```

```
F
```

Infine, calcolare analiticamente $\mathbf{p} - q$ e verificare che $q = \text{poly}([1, -1, u], 'x')$.

3. Si consideri il polinomio $q(t) = 2t^7 - 6t^4 + 8$. Calcolare, utilizzando il comando **horner**, i valori $p(2)$ e $p(-8)$.
4. Verificare che per ogni $\xi \in F(2, 53)$ esiste $\theta \in \mathbb{R}$ tale che:³

$$F(\xi) = f(\xi)(1 + \theta) \quad \text{e} \quad |\theta| < (1 + u)^{14} - 1$$

Dimostrare poi che:⁴ $(1 + u)^{14} - 1 < 1$ (e quindi che $F(\xi)$ e $f(\xi)$ hanno lo stesso segno).

5. Verificare che: se il calcolatore fosse in grado di calcolare i valori necessari della funzione f , otterremmo:

$$\text{Bisezione}(f, \mathbf{a}, \mathbf{b}) = \text{Bisezione}(F, \mathbf{a}, \mathbf{b})$$

6. Sia $g : \mathbb{R} \rightarrow \mathbb{R}$ una funzione continua tale che:

$$\max_{x \in [1.77; 2.23]} |f(x) - g(x)| \leq 10^{-9}$$

Stimare la massima distanza tra lo zero di f (che vale 2) ed uno zero di g (certamente esistente: spiegare perché). Utilizzare poi *Scilab* per tracciare grafici approssimati di f , $f + 10^{-9}$ e $f - 10^{-9}$ su $[1.77; 2.23]$ e spiegare come questi grafici possono essere utilizzati per stimare il risultato analitico ottenuto.

7. Modificare il disegno riportato in Figura 4 in modo da ottenere una griglia orizzontale di colore blu, la curva corrispondente al grafico di $\sin x$ di colore verde, spessore 3 pixel e sovrapposta a quella corrispondente al grafico di $\cos 2x$.

³Per la disuguaglianza, osservare che la funzione $g(x) = (1 + x)^{14}$ è convessa e utilizzare la retta tangente al grafico di $g(x)$ per $x = 0$.

⁴Dimostrare per induzione che: per ogni $k \in \mathbb{N}$ tale che $ku < 1$ si ha: $(1 + u)^k < 1/(1 - ku)$, e poi utilizzare la disuguaglianza provata, con $k = 14$.