Corso di Laurea in Informatica – Università di Ferrara
Calcolo Numerico
Prova scritta e pratica del 08/09/2023 – A.A. 2022–2023
Tempo a disposizione: 4 ore

Istruzioni

Gli esercizi sono di due tipologie:

- tipologia (T): esercizio che riguarda gli argomenti teorici del corso. **lo svolgimento di tale esercizio va effettuato sul foglio, che verrà consegnato a parte**;
- tipologia (M): esercizio di programmazione da svolgere in ambiente Matlab. Il codice di svolgimento di tale esercizio va scritto sulla propria postazione PC e verrà consegnato inserendolo nella cartella di lavoro dedicata, già predisposta nella postazione.

  N.B.: tutti gli M-files consegnati devono contenere, all'inizio, due righe di commento con l'indicazione dei propri cognome e nome (nella prima riga) e del proprio numero di matricola (nella seconda riga), pena l'esclusione dalla prova scritta.

Gli esercizi denotati con "(M + T)" riguardano argomenti sia teorici, che di programmazione Matlab: la parte teorica dovrà essere svolta sul foglio, mentre la parte di programmazione Matlab verrà consegnata insieme ai sorgenti degli altri esercizi.

Esercizio 1

1) (T) (2 punti) Dato il numero reale $\alpha = -(\texttt{2C5A7})_{16} + 2^{-6}$, determinarne la rappresentazione ANSI standard IEEE a 32 bit, mostrando tutti i calcoli della conversione..
2) (M) (2 punti) Realizzare un M-script di Matlab che, usando la M-function `ruffiniHorner` in allegato, calcoli e stampi il valore del polinomio $p_6(x) = c_6x^6 + \ldots + c_1x + c_0$ e delle sue derivate prima $p_6'(x)$ e seconda $p_6''(x)$ in un prefissato punto x_0 accettato in input.
   I coefficienti c_j, $j = 0, \ldots, 6$, del polinomio $p_6(x)$ sono i seguenti:

$$c_0 = \left|\sin(7\pi/9.076)\sqrt[3]{\mathrm{e}^2}\right|, \qquad c_1 = \cos\left(3\sqrt{2\mathrm{e}^{-3}}\right), \qquad c_2 = -\log_{10}\left(0.314/\pi^3\right), \qquad c_3 = 0,$$
$$c_4 = \min\left\{3.7\mathrm{E}2, -\tan(\mathrm{e}\pi), \left[0.25^3\right]\right\}, \qquad c_5 = -\tan(\pi/4) + \log_2\left(\left|\sin(-7.3)\right|\right), \qquad c_6 = -\ln\left(\sqrt{5!}\right),$$

   dove $[x]$ è la funzione parte intera di x. Lo script visualizzi anche il grafico del polinomio nell'intervallo $[-1.0, 0.85]$. Provare lo script con il valore $x = -0.31$.

Esercizio 2

1) (T) (3 punti) Dopo aver determinato l'insieme Ω_φ di definizione dell'espressione:

$$\varphi(x) = \frac{4\cos(3x)}{\ln(x/3) - 5^{-1.75x}} \qquad x \in \Omega_\varphi \subset \mathbb{R}$$

   valutare l'errore inerente, l'errore algoritmico, l'indice di condizionamento e l'indice algoritmico nel calcolo di $\varphi(x)$. Determinare inoltre gli eventuali valori di x per i quali il problema è mal condizionato e quelli per i quali il calcolo è instabile.
2) (M) (4 punti) Realizzare un M-function file per la valutazione della funzione

$$\varphi_{\alpha,\beta,\gamma,\delta}(x) = \alpha\cos(\beta x)/\big(\ln(x/\gamma) - 5^{\delta x}\big),$$

   generalizzazione della $\varphi(x)$ del punto precedente, nella quale $\alpha, \beta, \gamma, \delta \in \mathbb{R}$ sono costanti scelte dall'utente e passate con un'unico parametro d'ingresso, secondo argomento di input della funzione. Per ogni fissato vettore $\boldsymbol{x}$ di valori, la M-function esegua in modo efficiente il calcolo del valore della funzione per tutte le componenti x_i di $\boldsymbol{x}$ e restituisca in output il vettore $\boldsymbol{y}$ con i risultati delle valutazioni di $\varphi_{\alpha,\beta,\gamma,\delta}(x)$.

In un M-script di prova, si consideri un vettore $\boldsymbol{x}$ di N componenti equispaziate nell'intervallo $[x_A, x_B]$. **Nel minor numero possibile di istruzioni con sintassi vettoriale**, lo script controlli se ci sono componenti del vettore per le quali il valore assoluto del denominatore è minore di 100 volte la precisione di macchina ed elimini tali componenti dal vettore stesso.
Successivamente, lo script invochi la M-function passandole il vettore $\boldsymbol{x}$ risultante dopo i controlli, salvando i risultati nel vettore `fwhlx` e temporizzando l'esecuzione dell'istruzione. A seguire, lo script invochi la M-function in un ciclo, per ciascuna componente x_i di $\boldsymbol{x}$, salvando i risultati nel vettore `fcptx` e temporizzando il ciclo.
Senza usare altri cicli, lo script controlli e segnali se fra le componenti dei due vettori `fwhlx` e `fcptx` ci sono differenze maggiori di 10 volte la precisione di macchina.
Con istruzioni di stampa opportunamente formattate, lo script visualizzi a console i due tempi di calcolo, la loro differenza relativa e il loro rapporto in forma percentuale.
Infine, lo script disegni il grafico di $\varphi_{\alpha,\beta,\gamma,\delta}(x)$ nell'intervallo $[x_A, x_B]$ usando le ascisse $\boldsymbol{x}$ ed i valori in `fwhlx`.
Provare lo script con $[x_A, x_B] = [0.1, 2.8]$, $\alpha = 4.1$, $\beta = 2.7$, $\gamma = 3$, $\delta = -1.75$ ed $N = 30$ milioni di punti.

Esercizio 3

1) (T) (4 punti) Considerato il sistema lineare:

$$\begin{cases} 3.4x_1 + 1.4x_2 - 1.8x_3 + 2.7x_4 = 1.1 \\ 1.4x_1 + 1.9x_2 - 0.4x_3 - 1.0x_4 = -0.5 \\ -1.8x_1 - 0.4x_2 + 2.2x_3 - 3.4x_4 = 2.3 \\ 2.7x_1 - 1.0x_2 - 3.4x_3 + 7.8x_4 = 0.9 \end{cases}$$

applicare il metodo di fattorizzazione di Cholesky per determinare se la matrice A dei coefficienti è definita positiva. In caso affermativo, utilizzare la fattorizzazione $A = LL^T$ ottenuta per risolvere il sistema. In caso negativo, applicare il metodo di eliminazione di Gauss con pivoting parziale. Esplicitare tutti i calcoli e tutte le matrici coinvolte nella fattorizzazione usata per risolvere il sistema. Calcolare il determinante di A, usando la fattorizzazione ottenuta. Nei calcoli, mantenere almeno tre cifre decimali corrette.
Calcolare poi, usando le formule più stabili del metodo di Givens, la fattorizzazione QR della sottomatrice di ordine 2 di A formata dall'intersezione delle righe seconda e terza e delle colonne terza e quarta. Calcolare infine il determinante di tale sottomatrice, nel modo più conveniente.
2) (M) (3 punti) Realizzare un M-script file che controlli la soluzione ottenuta nel punto precedente, utilizzando le opportune M-function predefinite di Matlab e le M-functions `ltrisol` e `utrisol` in allegato. Stampare a video soluzione e residuo normalizzato (in norma infinito). Lo script calcoli poi la matrice $B = -L^T L$ e verifichi se è definita positiva. Lo script risolva il sistema $B\boldsymbol{x} = \boldsymbol{c}$, con $\boldsymbol{c} = (-1.5, 0.8, 3.1, -0.6)^T$, scegliendo in modo automatico il metodo di Cholesky o il metodo di Gauss con pivoting parziale a seconda che B sia o meno definita positiva, facendo opportuno uso delle funzioni predefinite di Matlab e delle le M-function in allegato.

Esercizio 4

Si considerino la seguente matrice M e il seguente vettore $\boldsymbol{z}$:

$$M = \begin{pmatrix} 3 & -1/3 & 0 \\ -1/2 & 2 & 1/6 \\ 0 & -1/5 & -3 \end{pmatrix} \qquad \boldsymbol{z} = \begin{pmatrix} -3/2 \\ 7/10 \\ 1/6 \end{pmatrix}$$

1. (T) (4 punti) Dato il sistema $M\boldsymbol{x} = \boldsymbol{z}$, stabilire *a priori* (cioè senza calcolare i rispettivi raggi spettrali) se i metodi iterativi di Jacobi e Gauss-Seidel risultano convergenti, motivando accuratamente le risposte. Stabilire se la matrice M è riducibile. Calcolare poi i raggi spettrali $\rho(J)$ e $\rho(\mathcal{G})$ delle matrici di iterazione di Jacobi e Gauss-Seidel e le rispettive velocità asintotiche di convergenza. Quale dei due metodi è più veloce? Scrivere l'espressione della generica matrice del metodo SOR con parametro ω nella forma di prodotto delle corrispondenti matrici triangolari inferiore e superiore.
2. (M) (4 punti) Realizzare un M-script file che calcoli in modo efficiente la matrice d'iterazione `J` ed il corrispondente vettore costante `cJ` del metodo di Jacobi per la matrice M e il termine noto $\boldsymbol{z}$ del punto 1. Analogamente, lo script calcoli in modo efficiente la matrice d'iterazione `G` ed il corrispondente vettore costante `cG` del metodo di Gauss-Seidel. Successivamente, lo script calcoli esattamente 5 iterazioni di entrambi i metodi, partendo dal vettore identicamente uguale a 2. Lo script calcoli poi la soluzione $\boldsymbol{x}^*$ del sistema $M\boldsymbol{x} = \boldsymbol{z}$ e confronti gli errori relativi rispetto ad $\boldsymbol{x}^*$ delle due approssimazioni ottenute alla quinta iterazione dei due metodi iterativi.

Si riportino nel foglio delle risposte queste differenze relative e si dica, motivando la risposta, se è lecito trarre da questi errori conclusioni riguardo il comportamento asintotico dei due metodi.

Esercizio 5

1) (T) (4 punti) Data la funzione $f(x) = (1/2)\cos(3x - 1)$ e dati i punti $a = x_0 = -1.1$, $x_1 = -0.2$, $x_2 = 0.5$, $b = x_3 = 1.5$, determinare l'espressione del polinomio interpolante $p_3(x)$ di grado al più 3 di $f(x)$ nei punti x_i, $i = 0, \ldots, 3$, usando il polinomio di Newton. Esplicitare tutti i calcoli. Scrivere l'espressione analitica dell'errore di interpolazione nel caso in esame e determinarne una maggiorazione nel punto $x_* = 0.9$, utilizzando le maggiorazioni delle derivate di $f(x)$ in $[a, b]$. Calcolare infine l'effettivo errore di interpolazione in x_* e nei tre punti $z_1 = -0.9$, $z_2 = -0.1$ e $z_3 = 1.2$: i valori di $p_3(x)$ in tali punti z_j possono essere considerati buone approssimazioni di $f(x)$ negli stessi punti? Motivare le risposte.
2) (M) (3 punti) Verificare i risultati del punto precedente mediante un M-script che utilizzi la M-function `tabDiff.m` in allegato. Lo script calcoli poi anche i coefficienti dei polinomi interpolanti $p_4(x)$ e $p_5(x)$ nella forma di Newton, che si ottengono aggiungendo ai punti di interpolazione i punti z_1 e z_2 precedentemente definiti.

Esercizio 6

1) (T) (3 punti) Costruire il polinomio $p_2(x)$ di migliore approssimazione nel senso dei minimi quadrati delle seguenti osservazioni sperimentali:

x_i	0.75	−0.85	−0.50	1.62	−1.33	−3.53	2.36	−1.98
y_i	−1.27	−1.92	4.04	−3.06	−4.52	5.02	−0.68	2.34

esplicitando le espressioni delle matrici e del termine noto coinvolti nel corrispondente sistema delle equazioni normali.
Risolvere tale sistema con un metodo numerico a scelta, mostrando i calcoli. Calcolare l'errore relativo commesso dal modello di secondo grado in ciascuno dei punti di osservazione, riportandolo anche in forma percentuale. Commentare i risultati ottenuti.
2) (M) (4 punti) Implementare nella M-function `myBisection` il metodo di bisezione. I parametri di ingresso siano: il nome di una function o un function handle, i due estremi dell'intervallo di ricerca, la tolleranza richiesta per l'approssimazione della soluzione. I parametri di uscita siano: l'approssimazione calcolata della radice dell'equazione, il valore della funzione in tale punto e il numero di iterazioni compiute. In un M-script file si provi `myBisection` per la ricerca di una radice dell'equazione $3x\sin(x+2) = \sqrt{5-x}$ nell'intervallo $[-4.5, 1]$, con tolleranza $\tau_x = 10^{-3}$. Lo script confronti poi tale approssimazione con quella che si ottiene usando la M-function `myNewton` in allegato, con punto iniziale $x_0 = 0.5$, tolleranze $\tau_x = \tau_f = 10^{-3}$ ed un massimo di 100 iterazioni.
Visualizzare i risultati a console. Disegnare in due finestre grafiche distinte i grafici della funzione utilizzata con le precedenti M-functions e della sua derivata prima. Dotare i grafici di opportuni titoli.
Riportare nel foglio delle risposte i risultati ottenuti, commentandoli brevemente.

Appendice: codici forniti

```
function [r, q] = ruffiniHorner(p, a)
% RUFFINIHORNER - Schema di Ruffini-Horner
%  Calcola il valore di un polinomio p(x) nel punto x = a e i coefficienti
%  del polinomio quoziente q(x) = p(x) / (x - a)
% SYNOPSIS
%   [r, q] = ruffiniHorner(p, a)
% INPUT
%   p (double array) - Vettore dei coefficienti del polinomio, ordinati
%                      da quello di grado piu' alto a quello di grado zero
%   a (double)       - Punto (numero reale) in cui calcolare il polinomio
% OUTPUT
%   r (double)       - Valore del polinomio nel punto x = a
%   q (double array) - Vettore dei coefficienti del polinomio quoziente
%                      q(x) = p(x) / (x - a)
%
```

```
r = [];
if ( isempty(p) )
    q = [];
    warning('Il vettore p dei coefficienti e'' vuoto');
    return
elseif ( isempty(a) )
    q = p;
    warning('Il punto ''a'' in cui valutare il polinomio e'' vuoto');
    return
else
    n = numel(p) - 1;  % grado del polinomio
    q = zeros(n, 1);
    q(1) = p(1);
    for i = 2 : n+1
        q(i) = q(i-1)*a + p(i);
    end
    r = q(n+1);
    q = q(1:n);
end
end % fine della function ruffiniHorner
```

```
function [x] = ltrisol(L, b)
% LTRISOL - Soluzione di sistemi triang. inf. (per colonne)
  n = length(b);
  x = b;
  x(1) = x(1)/L(1,1);
  for j = 2:n
     % SAXPY
     x(j:n) = x(j:n) - L(j:n, j-1)*x(j-1);
     x(j)   = x(j) / L(j,j);
  end
end
```

```
function [x] = utrisol(R,b)
% RTRISOL - Soluzione di sistema triang. sup. (per colonne)
  n = length(b);
  x = b;
  x(n) = x(n)/R(n,n);
  for j = n-1 : -1 : 1
     % SAXPY - BLAS1
     x(1:j) = x(1:j) - R(1:j, j+1)*x(j+1);
    x(j) = x(j)/R(j,j);
  end
end
```

```
function [d] = tabDiff(x, y)
% TABDIFF - Tabella delle differenze divise sui nodi x e i valori y
% INPUT
%   x  (double array) - vettore dei nodi o punti di osservazione
%   y  (double array) - vettore delle osservazioni
% OUTPUT
%   d  (double array) - coefficienti del polinomio di Newton (ordinati
%                       per grado crescente del termine corrispondente)
%
  x = x(:); y = y(:);
  n = length(x);
  d = y;
  for k = 2 : n
    d(k:n) = ( d(k:n) - d(k-1:n-1) ) ./ ( x(k:n) - x(1:n-k+1) );
  end
end % fine della function tabDiff
```

```
function [x, fx, it] = myNewton(fname, fpname, x0, tolx, tolf, maxit)
% MYNEWTON - Metodo di Newton per la soluzione di equazioni non lineari
```

```
%  Determina iterativamente un'approssimazione xs della soluzione
%  dell'equazione non lineare f(x) = 0  mediante il metodo di Newton.
% SYNOPSIS
%  [x, fx, it] = newton(fname, fpname, x0, tolx, tolf, maxit)
% INPUT
%  fname  (string o func. handle) - Function della funzione f
%  fpname (string o func. handle) - Function della derivata prima f' di f
%  x0     (double)                - Stima iniziale
%  tolx   (double)                - Distanza minima fra iterati successivi
%  tolf   (double)                - Soglia verso zero dei valori di f(x)
%  maxit  (integer)               - Numero massimo di interazioni
% OUTPUT
%  x      (double)  - Approssimazione della soluzione dell'equazione
%  fx     (double)  - Valore della funzione in x
%  it     (integer) - Numero di iterazioni compiute fino all'arresto
%
  tolfp = min( tolf, 10*eps );
  % Metodo di Newton
  x = x0; fx = feval(fname, x); it = 0;
  stop = ( abs(fx) < tolf );
  while ( ~stop )
    it   = it + 1;
    fpx  = feval(fpname, x);
    if (abs(fpx) < tolfp), error('|f''(xk)| troppo piccolo.'); end
    d    = fx / fpx;  x = x - d;  fx = feval(fname, x);
    stop = ( (abs(fx) < tolf && abs(d) < tolx*abs(x)) ...
             || (fx == 0) || (it == maxit) );
  end
  if ( it >= maxit )
    fprintf('\nRaggiunto il massimo numero di iterazioni.\n');
  end
end % fine della function myNewton.m
```

Soluzioni e commenti

Nel seguito sono riportate le soluzioni degli esercizi del compito, includendo commenti che aiutino il più possibile la comprensione della soluzione. Qualora uno stesso esercizio possa eventualmente essere risolto in più modi, in alcuni casi si riportano le descrizioni anche di qualche modo alternativo. In generale, questo rende il testo delle soluzioni degli esercizi inevitabilmente molto più lungo di quanto non sia effettivamente richiesto a stedentesse e studenti nel momento della prova scritta, pertanto **la lunghezza delle soluzioni qui riportate è da considerarsi NON RAPPRESENTATIVA della lunghezza del compito**. Si invitano studentesse e studenti a contattare il docente per eventuali dubbi o chiarimenti.

Nella trascrizione delle soluzioni è stata posta la massima cura. Tuttavia, nel caso si rilevino sviste e/o errori di vario genere, si invitano studentesse e studenti a comunicarlo via e-mail al docente.

Soluzione esercizio 1

Teoria

In questo caso, la conversione in binario del numero α è molto semplice e si può fare rapidamente a mano. Il valore assoluto della prima parte è il numero intero esadecimale $(\texttt{2C5A7})_{16}$, la cui rappresentazione binaria si ottiene immediatamente sostituendo ad ogni cifra esadecimale il quartetto di bit che la rappresenta:

$$(\texttt{2})_{16} = (0010)_2, \quad (\texttt{C})_{16} = (1100)_2, \quad (\texttt{5})_{16} = (0101)_2, \quad (\texttt{A})_{16} = (1010)_2, \quad (\texttt{7})_{16} = (0111)_2,$$
$$\Rightarrow \qquad (\texttt{2C5A7})_{16} = (\underbrace{00101100010110100111}_{\texttt{2}\quad\texttt{C}\quad\texttt{5}\quad\texttt{A}\quad\texttt{7}})_2 = (10110001011010011 1)_2 \,.$$

D'altro canto, la seconda parte di α è data già come potenza (negativa) di 2, quindi la rappresentazione binaria è immediata: $2^{-6} = (0.000001)_2$.

ATTENZIONE: in questo caso i due addendi la cui somma fornisce il numero α **hanno segno opposto**: sarebbe quindi **errato** aggiungere semplicemente la rappresentazione binaria di 2^{-6} in coda a quella di $(\texttt{2C5A7})_{16}$!

Perché questo equivarrebbe a fare la somma dei loro moduli. Invece, avendo segno opposto, occorre fare la differenza dei loro moduli: $(\texttt{2C5A7})_{16} - 2^{-6}$. Tuttavia, in questo caso, anche la differenza dei loro moduli è estremamente semplice e fattibile rapidamente a mano. Infatti, essendo $0 < 2^{-6} < 1$, per avere la rappresentazione della differenza dei moduli basta scrivere la parte intera diminuita di 1 e aggiungere la parte frazionaria della differenza $1 - 2^{-6}$. Essendo $2^{-6} = (0.000001)_2$, la rappresentazione binaria di quest'ultima differenza si ottiene immediatamente invertendo tutti i bit della parte frazionaria di 2^{-6}, tranne l'ultimo: $1 - 2^{-6} = (0.111111)_2$. Nel caso in esame, poi, dato che l'intero $(\texttt{2C5A7})_{16} > 1$ ha rappresentazione binaria che termina con un 1 (perché si tratta di un intero dispari), nella parte intera basta semplicemente sostituire 0 come ultimo bit:

$$(\texttt{2C5A7})_{16} - 1 = (10110001011010011 0)_2, \quad 1 - 2^{-6} = (0.111111)_2$$

$$\Rightarrow \qquad \alpha = -(\texttt{2C5A7})_{16} + 2^{-6} = -\Big((\texttt{2C5A7})_{16} - 2^{-6}\Big) = -(10110001011010011 0.111111)_2 \,.$$

Assai più lungo a dirsi, che a farsi! In questo modo, si ottiene la rappresentazione binaria del numero α senza necessità di eseguire né l'algoritmo delle divisioni successive, né quello delle moltiplicazioni successive. Si noti che, trattandosi di un intero sommato ad una potenza (negativa) di 2, la rappresentazione binaria di α è finita. Possiamo allora procedere con i successivi passi per ottenere la rappresentazione IEEE di α in precisione semplice:

$$\begin{aligned}
\alpha &= -(10110001011010011 0.111111)_2 \\
&= -(1.01100010110100110111111)_2 \cdot 2^{17} \\
&= -(1.01100010110100110111111)_2 \cdot 2^{(+10001)_2}
\end{aligned}$$

$$\begin{aligned}
&\alpha < 0 \Rightarrow s = 1 \\
&p = (+10001)_2 = (+17)_{10} \\
&\quad \Rightarrow \widetilde{p} = p + bias = 17 + 127 = 144 = 128 + 16 = (10010000)_2 \\
&m = (1.01100010110100110111111)_2 \\
&\quad \Rightarrow \widetilde{m} = 01100010110100110111111 \quad \text{(attenzione al troncamento della mantissa alla 24-esima cifra binaria!)}
\end{aligned}$$

$$\text{IEEE}_{32}(\alpha) = \overbrace{\underbrace{1}_{s}\underbrace{10010000}_{\widetilde{p}}\underbrace{01100010110100110111111}_{\widetilde{m}}}^{\text{32 bit = 4 byte}}$$

(1° byte | 2° byte | 3° byte | 4° byte)

Ovviamente, si arriva allo stesso risultato anche seguendo la strada standard, ossia mediante gli algoritmi delle divisioni successive e delle moltiplicazioni successive. L'intero $-(\texttt{2C5A7})_{16}$ va convertito in decimale per eseguire la somma con 2^{-6}. Tuttavia, spendo già che, nel caso in esame, il modulo della parte intera di α sarà inferiore di 1 al modulo del valore esadecimale dato, si può applicare l'algoritmo delle divisioni successive direttamente al modulo del valore esadecimale diminuito di 1, ossia a $(\texttt{2C5A6})_{16}$, avendo cura in tal caso di eseguire tutte le operazioni in aritmetica esadecimale:

$$(\texttt{2C5A7})_{16} = (181671)_{10}\,, \quad 2^{-6} = 0.015625 \quad \Rightarrow \quad \alpha = -181671 + 0.015625 = -181670.984375$$

parte intera:

181670	/2
90835	0
45417	1
22708	1
11354	0
5677	0
2838	1
1419	0
709	1
354	1
177	0
88	1
44	0
22	0
11	0
5	1
2	1
1	0
0	1

oppure

$(\texttt{2C5A6})_{16}$	/2
$(\texttt{162D3})_{16}$	0
$(\texttt{B169})_{16}$	1
$(\texttt{58B4})_{16}$	1
$(\texttt{2C5A})_{16}$	0
$(\texttt{162D})_{16}$	0
$(\texttt{B16})_{16}$	1
$(\texttt{58B})_{16}$	0
$(\texttt{2C5})_{16}$	1
$(\texttt{162})_{16}$	1
$(\texttt{B1})_{16}$	0
$(\texttt{58})_{16}$	1
$(\texttt{2C})_{16}$	0
$(\texttt{16})_{16}$	0
$(\texttt{B})_{16}$	0
$(\texttt{5})_{16}$	1
$(\texttt{2})_{16}$	1
$(\texttt{1})_{16}$	0
$(\texttt{0})_{16}$	1

parte frazionaria:

0.984375	×2
0.968750	1
0.937500	1
0.875000	1
0.750000	1
0.500000	1
0.000000	1

$\Downarrow$

$0.984375 = (0.111111)_2$

$\Rightarrow \; 181670 = (10110001011010011 0)_2$

$\Downarrow$

$\alpha = -(10110001011010011 0.111111)_2$

ATTENZIONE!! Se si eseguono i calcoli con Matlab, è **fondamentale** non perdere i decimali dei valori! Il valore decimale esatto della parte frazionaria di α è 0.984375 e quello di 2^{-6} è 0.015625. Se si mantiene la visualizzazione di default di Matlab (che usualmente è il formato "`short`"), a console vengono visualizzati valori **approssimati** al più a 4 cifre decimali: pertanto, il valore visualizzato di 2^{-6} è `0.0156`, mentre il valore visualizzato di α dopo l'operazione di somma `-hex2dec('2C5A7') + 2^(-6)` è `-1.8167e+05`! In RAM Matlab ha calcolato il valore corretto, ma se si trascrivono i valori approssimati visualizzati a console e si eseguono poi i calcoli partendo da questi, si ha una **pesante perdita di accuratezza**, che si ripercuote quasi certamente su una errata conversione in binario, sia della parte intera, che di quella frazionaria.

Quindi, nell'effettuare i calcoli delle conversioni con Matlab, si raccomanda di usare sempre, per la visualizzazione dei dati e dei risultati intermedi, il formato più esteso ("`long`" oppure "`long e`"). In questo modo, si riduce moltissimo la possibilità di commettere errori dovuti ad una trascrizione approssimata dei valori.

Matlab

Contenuto dell'M-function file `esercizio1.m`:

```
% Cognome Nome
% Matricola
%-----------------------------
%  esercizio 1 - 08/09/2023
%-----------------------------
close all; clear all; clc;
disp('Esecizio 1');

p6    = zeros(7,1);
p6(1) = -log( sqrt( factorial(5) ) );                          % c6
p6(2) = -tan( pi/4 ) + log2( abs( sin(-7.3) ) );               % c5
p6(3) = min( [ 3.7E2; -tan( exp(1)*pi ); fix( 0.25^3 ) ] );    % c4
p6(5) = -log10( 0.314 / pi^3 );                                % c2
p6(6) = cos( 3 * sqrt( 2*exp(-3) ) );                          % c1
p6(7) = abs( sin(7*pi / 9.076) * exp(2/3) );                   % c0

x0 = input('Inserire il punto nel quale valutare il polinomio (double): x0 = ');

[r   , q ] = ruffiniHorner( p6, x0 );
[derp, q1] = ruffiniHorner(  q, x0 );
[ders, q2] = ruffiniHorner( q1, x0 );
fprintf('\nValore del polinomio in x0 = %g: p(x0) = %g', x0, r);
fprintf('\nDerivata prima del polinomio in x0: p''(%g) = %g', x0, derp);
fprintf('\nDerivata seconda del polinomio in x0: p''''(%g) = %g\n\n', x0, 2*ders);
ph = fplot( @(x)(polyval(p6,x)), [-1.00,0.85] );
```

Eseguendo lo script si ottengono i seguenti valori a console e una figura simile alla seguente (quella presentata qui è stata leggermente migliorata per esigenze di stampa):

Punto di valutazione: $x_0 = -0.31$
Valori calcolati in x_0:
$p_6(x_0) \approx \quad 1.2942$
$p_6'(x_0) \approx -0.6681$
$p_6''(x_0) \approx \quad 4.0609$

I coefficienti del polinomio risultano:

$$\mathrm{p6} \approx \begin{pmatrix} -2.3937 \\ -1.2337 \\ 0 \\ 0 \\ 1.9945 \\ 0.5844 \\ 1.2822 \end{pmatrix}$$

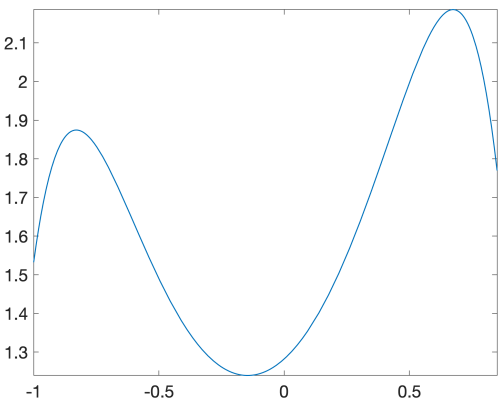


Figura 1: grafico del polinomio $p_6(x)$ dell'esercizio 1 nell'intervallo $[-1.00, 0.85]$.

Soluzione esercizio 2

Teoria

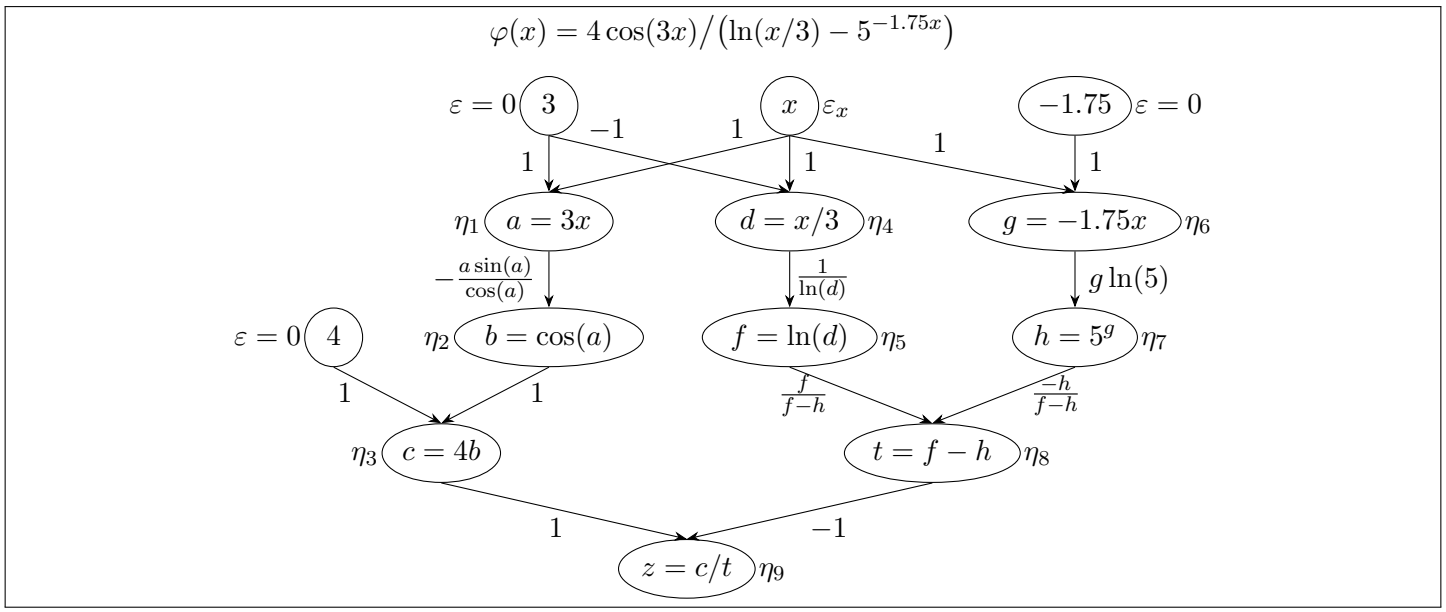


Figura 2: grafo della formulazione della funzione $\varphi(x)$ dell'esercizio 2.

Notiamo innanzitutto che la funzione $\varphi(x)$ non è definita fuori da $]0, +\infty[$ per la presenza del logaritmo naturale. Inoltre, il denominatore della funzione $\varphi(x)$ si annulla certamente in un punto $x_* \in \mathbb{R}$, perché $f(x) = \ln(x/3)$ è una funzione monotona strettamente crescente da $-\infty$ a $+\infty$ in $]0, +\infty[$, mentre $h(x) = 5^{-1.75x}$ (esponenziale negativa di base maggiore di 1) è monotona strettamente decrescente da 1 a 0 nello stesso intervallo $]0, +\infty[$. Per la monotonìa di $f(x)$ e $h(x)$, il loro punto d'intersezione $x_* \in]0, +\infty[$ è unico. Per avere una stima numerica di x_0 possiamo utilizzare la funzione predefinita `fzero` di Matlab sulla funzione del denominatore $t(x) = f(x) - h(x)$: le due semplici righe di codice

```
>> f = @(x)( log(x/3) ); h = @(x)( 5.^(-1.75*x) );  t = @(x)( f(x) - h(x) );
>> xs = fzero(t, 3.0)
```

forniscono il valore approssimato $x_* \approx 3.0006$. Non essendoci altre situazioni critiche nell'espressione di $\varphi(x)$, concludiamo che il suo dominio di definizione è $\Omega_\varphi =]0, +\infty[\setminus \{x_*\}$.

Nota. Si sarebbe potuto evitare di definire tre function handle e passare direttamente nel primo argomento di `fzero` la definizione del denominatore, come function anonima contestuale. Tuttavia, definire i due singoli function handle per le funzioni $f(x)$ e $h(x)$ consente di visualizzare graficamente il comportamento delle due funzioni, oltre che della funzione $t(x) = f(x) - h(x)$ (il denominatore), consentendo così di dedurre un buon punto iniziale da fornire a `fzero` come stima di partenza di x_*: si è scelto infatti $x_0 = 3.0$. Si veda la figura 4. Con riferimento al grafo della figura 2, l'errore inerente è dato da:

$$\begin{aligned}
\epsilon_{\text{dati}} &= \left(1 \cdot \left(-\frac{a\sin(a)}{\cos(a)}\right) \cdot 1 \cdot 1 + 1 \cdot \frac{1}{\ln(d)} \cdot \frac{f}{f-h} \cdot (-1) + 1 \cdot g\ln(5) \cdot \left(\frac{-h}{f-h}\right) \cdot (-1)\right) \varepsilon_x \\
&= -\left(3x\tan(3x) + \frac{1}{\cancel{\ln(x/3)}} \cdot \frac{\cancel{\ln(x/3)}}{\ln(x/3) - 5^{-1.75x}} + 1.75x \cdot \ln(5) \cdot \frac{5^{-1.75x}}{\ln(x/3) - 5^{-1.75x}}\right) \varepsilon_x \\
&= -\left(3x\tan(3x) + \frac{1 + 1.75\ln(5)x5^{-1.75x}}{\ln(x/3) - 5^{-1.75x}}\right) \varepsilon_x = -\left(3x\tan(3x) + \frac{5^{1.75x} + 1.75\ln(5)x}{5^{1.75x}\ln(x/3) - 1}\right) \varepsilon_x = \theta_x \varepsilon_x
\end{aligned}$$

L'indice di condizionamento risulta:

$$I_{\text{cond}} = |\theta_x| = \left|3x\tan(3x) + \frac{1 + 1.75\ln(5)x5^{-1.75x}}{\ln(x/3) - 5^{-1.75x}}\right| = \left|3x\tan(3x) + \frac{5^{1.75x} + 1.75\ln(5)x}{5^{1.75x}\ln(x/3) - 1}\right|. \tag{1}$$

Dato che $\tan(3x)$ diverge nei pressi di tutti i multipli dispari di $\pi/6$, dove invece il secondo addendo di θ_x rimane limitato, si ha $I_{\text{cond}} \xrightarrow{x \to x_k} +\infty$ per tutti gli x_k tali che $x_k = (2k+1)\pi/6$ con $k \in \mathbb{N}_0$, con $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$. Qui si considera solo $k \in \mathbb{N}_0$ e non $k \in \mathbb{Z}$ perché $\varphi(x)$ è non definita in $]-\infty, 0]$. Nei pressi di ciascuno di tali punti x_k

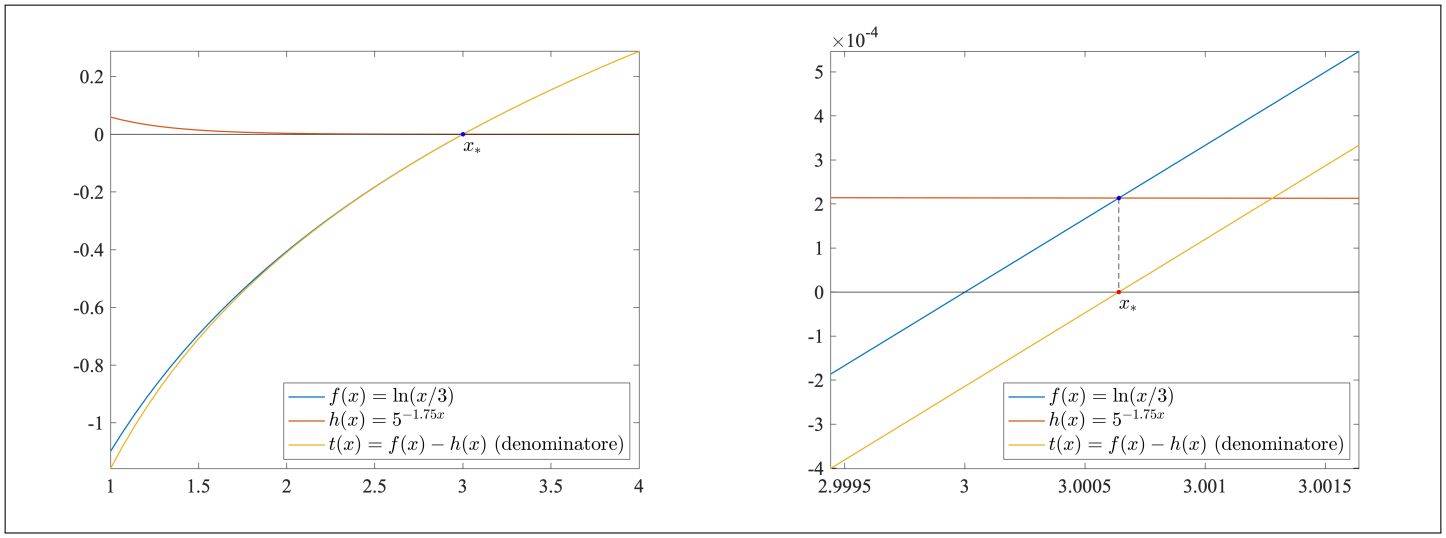


Figura 3: andamento delle funzioni $f(x)$, $h(x)$ e $t(x)$ a denominatore della funzione $\varphi(x)$ dell'esercizio 2. Come mostra il dettaglio a destra, si noti che il punto d'intersezione delle curve $f(x)$ e $h(x)$ non appartiene all'asse delle ascisse, pur essendo ad esso molto vicino, ma ha ascissa x_*, che è esattamente lo zero di $t(x) = f(x) - h(x)$ (ossia, del denominatore di $\varphi(x)$).

si ha quindi malcondizionamento. Si noti che $x_* \in]5\pi/6, 7\pi/6[$ e dunque $x_* \neq x_k \ \forall k \in \mathbb{N}_0$, ossia x_* non coincide con alcun x_k. D'altro canto, il denominatore del secondo addendo di θ_x è lo stesso di $\varphi(x)$ e dunque si annulla nel solo punto $x_* \approx 3.0006 \notin \Omega_\varphi$, nel quale il primo addendo vale $3x_* \tan(3x_*) \approx -4.0509 \in \mathbb{R}$ (ossia rimane limitato). Pertanto, $I_{\text{cond}} \xrightarrow[x \to x_*]{} +\infty$. Per $x \to 0^+$ si ha $I_{\text{cond}} \to 0$. Concludiamo quindi che il calcolo di $\varphi(x)$ è malcondizionato nei pressi di tutti i punti $x_k = (2k+1)\pi/6 \in \Omega_\varphi$, $k \in \mathbb{N}_0$, e nei pressi di $x_* \notin \Omega_\varphi$.

Ancora con riferimento al grafo della figura 2, l'errore algoritmico e l'indice algoritmico sono dati da:

$$\epsilon_{\text{alg}} = \left(\left(-\frac{a\sin(a)}{\cos(a)}\right)\cdot 1 \cdot 1\right)\eta_1 + 1\cdot 1\cdot\eta_2 + 1\cdot\eta_3 + \frac{1}{\cancel{\ln(d)}}\cdot\frac{\cancel{f}}{f-h}\cdot(-1)\eta_4 + \frac{f}{f-h}\cdot(-1)\eta_5 \tag{2}$$

$$+ g\ln(5)\cdot\left(\frac{-h}{f-h}\right)\cdot(-1)\eta_6 + \left(\frac{-h}{f-h}\right)\cdot(-1)\eta_7 + (-1)\eta_8 + \eta_9 \tag{3}$$

$$= -3x\tan(3x)\eta_1 + \eta_2 + \eta_3 - \frac{1}{\ln(x/3) - 5^{-1.75x}}\left(\eta_4 + \ln(x/3)\eta_5 + 5^{-1.75x}\left(1.75\ln(5)x\eta_6 - \eta_7\right)\right) - \eta_8 + \eta_9 \tag{4}$$

$$I_{\text{alg}} = \left|3x\tan(3x)\right| + \frac{1}{\left|\ln(x/3) - 5^{-1.75x}\right|}\left(1 + \left|\ln(x/3)\right| + 5^{-1.75x}\left(1.75\ln(5)x + 1\right)\right) + 4 \tag{5}$$

Come si vede, l'indice algoritmico tende a divergere nei pressi degli stessi punti in cui diverge l'indice di condizionamento, ossia abbiamo instabilità dell'algoritmo nei pressi di tutti i punti $x_k = (2k+1)\pi/6 \in \Omega_\varphi$, $k \in \mathbb{N}_0$, e nei pressi di $x_* \notin \Omega_\varphi$. In più per l'indice algoritmico dobbiamo considerare anche cosa accade in un intorno destro di zero (ricordando che $0 \notin \Omega_\varphi$), a causa della presenza di $\left|\ln(x/3)\right|$ a numeratore. Per $x \to 0^+$ il primo addendo di I_{alg} tende a zero e il numeratore del secondo addendo si comporta come $\left|\ln(x/3)\right| + 2$, mentre il denominatore si comporta come $\left|\ln(x/3) - 1\right|$. Allora, dato che $\left|\ln(x/3)\right| \to +\infty$ per $x \to 0^+$, concludiamo che il secondo addendo tende a 1 per $x \to 0^+$ e dunque, in conclusione, che $I_{\text{cond}} \to 5 < +\infty$ per $x \to 0^+$, ossia che non c'è instabilità nei pressi di un intorno destro di zero.

Matlab

Contenuto dell'M-function file `esercizio2.m`:

```
% Cognome Nome
% Matricola
function [y] = esercizio2( x, params)
%ESERCIZIO2 - Esercizio 2 prova scritta di Calcolo Numerico dell'8/09/2023
%  Valuta la funzione
%    f(t; a,b,c,d) =  a*cos( b*x ) ./ ( log( x/c ) - 5.^( d*x )  )
%  nel punto x, per fissati valori dei parametri reali 'a', 'b', 'c' e 'd'.
% SYNOPSIS
```

```
%   y = esercizio2( x, params )
% INPUT
%   x      (double array) - Vettore di valori nei quali valutare la funzione f(t)
%   params (double array) - Vettore dei parametri della funzione f(t):
%                           params(1:2) = [a,b] : parametri numeratore
%                           params(3:4) = [c,d] : parametri denominatore
% OUTPUT
%   y      (double array) - Vettore dei valori della funzione f(t) nei punti x

  % Si assume che i controlli sui paramteri di input siano effettuati prima
  % della chiamata della funzione
  y = params(1)*cos( params(2)*x ) ./ ( log( x/params(3) ) - 5.^( params(4)*x )  );
end
```

Contenuto dell'M-script file `testEsercizio2.m`:

```
% Cognome Nome
% Matricola
%-----------------------------
%  esercizio 2 - 08/09/2023
%-----------------------------
close all; clear all; clc
disp('Test esercizio 2');

N = 3.0e7; params = [4.1; 2.7; 3; -1.75]; xA = 0.1; xB = 2.8;
x = linspace(xA, xB, N)';
x( find( abs( log( x/params(3) ) - 5.^( params(4)*x ) ) < 100*eps ) ) = [];
N = numel(x); % nel caso il controllo precedente abbia eliminato componenti di x
tic; fwhlx = esercizio2( x, params );
tVect = toc;
fcptx = zeros(N,1);
tic;
for i = 1:N
    fcptx(i) = esercizio2( x(i), params );
end
tLoop = toc;
if ( any( abs(fwhlx - fcptx) > 10*eps ) )
    warning('Differenze maggiori do 10*eps nella valutazione della funzione');
end
fprintf('\ntempo dell''invocazione vettoriale: tVect = %g secondi', tVect);
fprintf('\ntempo del ciclo: tLoop = %g secondi', tLoop);
fprintf('\ndifferenza relativa: |tLoop - tVect| / |tLoop| = %g', ...
        abs(tLoop - tVect)/abs(tLoop) );
fprintf('\nincidenza percentuale: |tVect| / |tLoop| = %g %%', ...
        abs(tVect)*100/abs(tLoop) );
fprintf('\n\n');
ph = plot(x,fwhlx);
```

Eseguendo l'M-script con parametri $[x_A, x_B] = [0.1, 2.8]$ e $\alpha = 4.1$, $\beta = 2.7$, $\gamma = 3$, $\delta = -1.75$ (quali elementi del vettore `params`, nell'ordine) ed infine $N = 3 \cdot 10^7$, come richiesto dal testo, si ottiene un output simile al seguente:

```
Test esercizio 2

tempo dell'invocazione vettoriale: tVect = 3.62843 secondi
tempo del ciclo: tLoop = 5.72888 secondi
differenza relativa: |tLoop - tVect| / |tLoop| = 0.366643
incidenza percentuale: |tVect| / |tLoop| = 63.3357 %
```

Chiaramente, i valori effettivi sono quasi certamente diversi da quelli riportati (dipendono sia dalla macchina, che dal grado di utilizzo della CPU da parte di altri processi, mentre gira l'M-script). Anche run successivi sulla stessa macchina possono dare valori diversi. Tuttavia, i valori di della differenza percentuale e dell'incidenza relativa dovrebbero avvicinarsi abbastanza a quelli riportati qui. Essenzialmente, il tempo impiegato dal ciclo è circa doppio rispetto al tempo impiegato dalla valutazione con sintassi vettoriale.

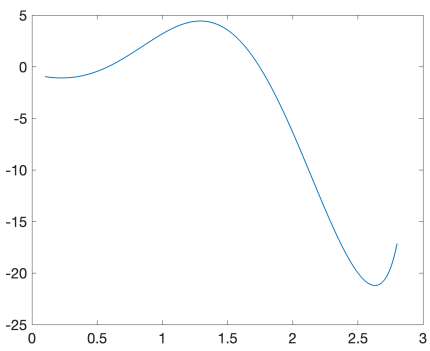


Figura 4: grafico della funzione $\varphi_{\alpha,\beta,\gamma,\delta}(x)$ dell'esercizio 2 nell'intervallo $[0.1, 2.8]$ con $\alpha = 4.1$, $\beta = 2.7$, $\gamma = 3$, $\delta = -1.75$.

Soluzione esercizio 3

Teoria

La matrice A ed il termine noto $\boldsymbol{b}$ del sistema sono

$$A = \begin{pmatrix} 3.4 & 1.4 & -1.8 & 2.7 \\ 1.4 & 1.9 & -0.4 & -1.0 \\ -1.8 & -0.4 & 2.2 & -3.4 \\ 2.7 & -1.0 & -3.4 & 7.8 \end{pmatrix}, \qquad \boldsymbol{b} = \begin{pmatrix} 1.1 \\ -0.5 \\ 2.3 \\ 0.9 \end{pmatrix}$$

Vediamo immediatamente che $A = A^T$, ossia che A è quadrata e simmetrica: possiamo indagare la sua definita positività.

Applichiamo l'algoritmo di fattorizzazione di Cholesky con la strategia di "pavimentazione" per colonne. L'algoritmo è in grado di rilevare l'eventuale non definita positività di A (alla precisione di macchina): se al passo k-esimo il radicando dell'elemento diagonale risulta non positivo, la matrice A è non definita positiva e l'algoritmo si arresta prematuramente, tranne al più nel caso in cui a diventare nullo sia l'ultimo elemento della diagonale principale, nell'ultimo passo. L'algoritmo per il calcolo degli elementi del fattore $\mathcal{L}$ di Cholesky di A per colonne e la sua versione con sintassi vettoriale sono:

per $k = 1, 2, \ldots, n$
- $\widetilde{\ell}_{k,k} = a_{k,k} - \sum_{j=1}^{k-1} \ell_{k,j}^2$
- se $\widetilde{\ell}_{k,k} < 0$ allora stop
- $\ell_{k,k} = \sqrt{\widetilde{\ell}_{k,k}}$
- per $i = k+1, k+2, \ldots, n$
  - $\ell_{i,k} = \left(a_{i,k} - \sum_{j=1}^{k-1} \ell_{k,j}\ell_{i,j}\right) \Big/ \ell_{k,k}$

per $k = 1, 2, \ldots n$
- $\widetilde{\ell}_{k,k} = a_{k,k} - \mathcal{L}_{k,\,1:(k-1)}\mathcal{L}_{k,\,1:(k-1)}^T$
- se $\widetilde{\ell}_{k,k} < 0$ allora stop
- $\ell_{k,k} = \sqrt{\widetilde{\ell}_{k,k}}$
- $\mathcal{L}_{(k+1):n,\,k} = \left(A_{(k+1):n,\,k} - \mathcal{L}_{(k+1):n,\,1:(k-1)}\mathcal{L}_{k,\,1:(k-1)}^T\right) \Big/ \ell_{k,k}$

N.B.: nella tecnica compatta, gli elementi strettamente sottodiagonali del fattore $\mathcal{L}$ di Choleshy riempiono il triangolo inferiore di A, mentre gli elementi diagonali di $\mathcal{L}$ vengono salvati in un vettore. Qui, non essendo richiesta la tecnica compatta, utilizziamo per $\mathcal{L}$ una matrice diversa da A.

Procediamo con l'algoritmo:

$k = 1$: $a_{1,1} = 3.4 > 0 \Rightarrow$ possiamo procedere con il calcolo della prima colonna di $\mathcal{L}$:

$$\ell_{1,1} = \sqrt{a_{1,1}} = \sqrt{3.4} \approx 1.8439, \qquad \mathcal{L}_{2:4,\,1} = A_{2:4,\,1} \Big/ \ell_{1,1} = \frac{1}{\sqrt{3.4}} \begin{pmatrix} 1.4 \\ -1.8 \\ 2.7 \end{pmatrix} \Rightarrow \mathcal{L}^{(1)} \approx \begin{pmatrix} 1.8439 & 0 & 0 & 0 \\ 0.7593 & 0 & 0 & 0 \\ -0.9762 & 0 & 0 & 0 \\ 1.4643 & 0 & 0 & 0 \end{pmatrix}$$

$k = 2$: $\widetilde{\ell}_{2,2} = a_{2,2} - \ell_{2,1}^2 = 1.9 - (1.4/\sqrt{3.4})^2 \approx 1.9 - 0.5765 \approx 1.3225 > 0 \Rightarrow$ possiamo procedere con il calcolo della seconda colonna di $\mathcal{L}$:

$$\ell_{2,2} = \sqrt{\widetilde{\ell}_{2,2}} = \sqrt{1.3225} \approx 1.1504$$

$$\mathcal{L}_{3:4,\,2} = \left(A_{3:4,\,2} - \mathcal{L}_{3:4,\,1}^{(1)} \cdot \mathcal{L}_{2,\,1}^{(1)}\right) \Big/ \ell_{2,2} \approx \left(\begin{pmatrix} -0.4 \\ -1.0 \end{pmatrix} - \begin{pmatrix} -0.9762 \\ 1.4643 \end{pmatrix} \cdot 0.7593\right) \Big/ 1.1504 \approx \begin{pmatrix} 0.2966 \\ -1.8356 \end{pmatrix}$$

$$\Rightarrow \mathcal{L}^{(2)} \approx \begin{pmatrix} 1.8439 & 0 & 0 & 0 \\ 0.7593 & 1.1504 & 0 & 0 \\ -0.9762 & 0.2966 & 0 & 0 \\ 1.4643 & -1.8356 & 0 & 0 \end{pmatrix}$$

$k = 3$: $\widetilde{\ell}_{3,3} = a_{3,3} - \left(\ell_{3,1}^2 + \ell_{3,2}^2\right) \approx 2.2 - \left((-0.9762)^2 + 0.2966^2\right) \approx 1.1591 > 0 \Rightarrow$ possiamo procedere con il calcolo della terza colonna di $\mathcal{L}$:

$$\ell_{3,3} = \sqrt{\widetilde{\ell}_{3,3}} \approx \sqrt{1.1591} \approx 1.0766$$

$$\ell_{4,\,3} = \left(a_{4,\,3} - \mathcal{L}_{4,\,1:2}^{(2)}\left(\mathcal{L}_{3,\,1:2}^{(2)}\right)^T\right) \Big/ \ell_{3,3} \approx \left(-3.4 - \begin{pmatrix} 1.4643, & -1.8356 \end{pmatrix} \begin{pmatrix} -0.9762 \\ 0.2966 \end{pmatrix}\right) \Big/ 1.0766$$

$$\approx -1.3247$$

$$\Rightarrow \mathcal{L}^{(3)} \approx \begin{pmatrix} 1.8439 & 0 & 0 & 0 \\ 0.7593 & 1.1504 & 0 & 0 \\ -0.9762 & 0.2966 & 1.0766 & 0 \\ 1.4643 & -1.8356 & -1.3247 & 0 \end{pmatrix}$$

$k = 4$: $\widetilde{\ell}_{4,4} = a_{4,4} - \left(\ell_{4,1}^2 + \ell_{4,2}^2 + \ell_{4,3}^2\right) \approx 4 - \left(1.4643^2 + (-1.8356)^2 + (-1.3247)^2\right) \approx 0.5316 > 0 \Rightarrow$ possiamo procedere con il calcolo della quarta (e ultima) colonna di $\mathcal{L}$:

$$\ell_{4,4} = \sqrt{\widetilde{\ell}_{4,4}} \approx \sqrt{0.5316} \approx 0.7291$$

$$\Rightarrow \mathcal{L} = \mathcal{L}^{(4)} \approx \begin{pmatrix} 1.8439 & 0 & 0 & 0 \\ 0.7593 & 1.1504 & 0 & 0 \\ -0.9762 & 0.2966 & 1.0766 & 0 \\ 1.4643 & -1.8356 & -1.3247 & 0.7291 \end{pmatrix}$$

Dato che l'algoritmo è arrivato al termine e la matrice ottenuta è triangolare inferiore con elementi diagonali tutti positivi, concludiamo che A **è definita positiva** (e quindi, in particolare, invertibile) e il suo fattore di Cholesky è $\mathcal{L}$, ossia $A = \mathcal{L}\mathcal{L}^T$. Abbiamo immediatamente che

$$\det(A) = \big(\det(\mathcal{L})\big)^2 = \left(\prod_{k=1}^{4} \ell_{k,k}\right)^2 \approx (1.8439 \cdot 1.1504 \cdot 1.0766 \cdot 0.7291)^2 \approx 3.56$$

Dato che A è definita positiva, per una delle proprietà delle matrici definite positive, tutti i suoi minori principali di testa sono positivi, quindi in particolare non nulli: ne discende allora che A può essere fattorizzata in forma LR anche con l'algoritmo di fattorizzazione di Gauss con strategia diagonale.

Avendo la fattorizzazione di Cholesky, la soluzione $\boldsymbol{x}^*$ del sistema $A\boldsymbol{x} = \boldsymbol{b}$ si trova immediatamente risolvendo in successione i due sistemi triangolari $\mathcal{L}\boldsymbol{y} = \boldsymbol{b}$ e $\mathcal{L}^T\boldsymbol{x} = \boldsymbol{y}^*$, dove $\boldsymbol{y}^*$ è la soluzione del sistema triangolare inferiore. Utilizzando la M-function `ltrisol`, per il primo sistema otteniamo $\boldsymbol{y}^* \approx (0.5966, -0.8283, 2.9054, 3.2299)^T$. Utilizzando poi la M-function `utrisol`, per il secondo sistema otteniamo $\boldsymbol{x}^* \approx (-0.6290, 4.2476, 8.1496, 4.4301)^T$, che è la soluzione del sistema di partenza.

Nota. Tutti i calcoli precedenti sono fatti in Matlab, conservando i risultati intermedi nelle rispettive variabili, in modo da mantenere la maggior accuratezza possibile. I valori riportati qui sono le approssimazioni con 4 cifre frazionarie dei vari risultati delle operazioni. Effettuare i calcoli utilizzando queste approssimazioni come operandi, invece di conservare i valori nelle variabili, porta ad un accumulo degli errori di arrotondamento e a valori sempre meno accurati, man mano che si procede con i passaggi.

Per l'ultimo punto, chiamiamo F la sottomatrice di ordine 2 di A formata dall'intersezione delle righe seconda e terza e delle colonne terza e quarta:

$$F = \begin{pmatrix} -0.4 & -1.0 \\ 2.2 & -3.4 \end{pmatrix} = \begin{pmatrix} -2/5 & -1 \\ 11/5 & -17/5 \end{pmatrix}$$

Con la rotazione elementare di Givens dobbiamo annullare l'elemento di posizione $(2,1)$ di F (che coincide con l'elemento in posizione $(3,3)$ di A), ossia $f_{2,1} = a_{3,3} = 2.2$, utilizzando per la rotazione l'elemento in posizione $(1,1)$ di F (che coincide con l'elemento in posizione $(2,3)$ di A), ossia $f_{1,1} = a_{2,3} = -0.4$. Usando le formule numericamente più stabili per il calcolo della matrice di rotazione di Givens si ha:

$$0.4 = |f_{1,1}| < |f_{2,1}| = 2.2 \;\Rightarrow\; t = \frac{f_{1,1}}{f_{2,1}} = -\frac{0.4}{2.2} \approx -0.1818$$

$$\Rightarrow\; s = \frac{1}{\sqrt{1+t^2}} = \frac{1}{\sqrt{1+(-0.1818)^2}} \approx 0.9839\,, \quad c = ts \approx -0.1818 \cdot 0.9839 \approx -0.1789$$

$$G_{1,2} \approx \begin{pmatrix} -0.1789 & 0.9839 \\ -0.9839 & -0.1789 \end{pmatrix} \;\Rightarrow\; R_2 = G_{1,2}A_2 \approx \begin{pmatrix} -0.1789 & 0.9839 \\ -0.9839 & -0.1789 \end{pmatrix}\begin{pmatrix} -0.4 & -1.0 \\ 2.2 & -3.4 \end{pmatrix} \approx \begin{pmatrix} 2.2361 & -3.1663 \\ 0 & 1.5921 \end{pmatrix}$$

$$\det(F) = \det(R_2) \approx 2.2361 \cdot 1.5921 \approx 3.56\,.$$

Matlab

Contenuto dell'M-script file `esercizio3.m`:

```
% Cognome Nome
% Matricola
%----------------------------
```

```
%  esercizio 3 - 08/09/2023
%-----------------------------
close all; clear all; clc;
disp('Esercizio 3');

A = [ 3.4    1.4   -1.8    2.7;
      1.4    1.9   -0.4   -1.0;
     -1.8   -0.4    2.2   -3.4;
      2.7   -1.0   -3.4    7.8 ];
b = [1.1   -0.5   2.3   0.9]';
xTeoria = [-0.6290, 4.2476, 8.1496, 4.4301]'; % N.B.: valori approssimati

% prima parte
[L, flag] = chol(A, 'lower') % L e' triangolare inferiore
x1 = utrisol( L', ltrisol(L, b) )
% oppure:  y1 = ltrisol(L, b); x1 = utrisol(L', y1)
disp('Massima differenza in modulo dalla soluzione teorica:');
disp( max( abs( x1 - xTeoria ) ) );
disp('Residuo normalizzato (in norma infinito):');
disp( (b - A*x1) / norm(b, inf) );
% NON RICHIESTO: controllo della soluzione mediante operatore '\' di Matlab
x2 = A \ b

% seconda parte
B = -L' * L; c = [-1.5, 0.8, 3.1, -0.6]';
disp('Fattorizzazione di Cholesky di B:');
[L1, p] = chol(B, 'lower')
if ( ~p )
    fprintf('\nB e'' definita positiva\n');
    x3 = utrisol( L1', ltrisol(L1, c) )
else
    fprintf('\nB non e'' definita positiva\n');
    x3 = B \ c % Basta questo: "\" e' implementato con il m. di Gauss con piv. par.
end
```

Eseguendo lo script si ottiene il seguente output:

```
Esercizio 3
L =
    1.8439         0         0         0
    0.7593    1.1504         0         0
   -0.9762    0.2966    1.0766         0
    1.4643   -1.8356   -1.3247    0.7291

flag =
     0

x1 =
   -0.6290
    4.2476
    8.1496
    4.4301

Massima differenza in modulo dalla soluzione teorica:
   3.7596e-05

Residuo normalizzato (in norma infinito):
   1.0e-14 *

    0.0193
   -0.0965
    0.1158
   -0.2462

x2 =
   -0.6290
    4.2476
    8.1496
    4.4301

Fattorizzazione di Cholesky di B:

L1 =
     []

p =
     1

B non e' definita positiva

x3 =
   -0.6946
    1.4136
   -2.7531
    1.0805
```

Il valore nullo di `flag` conferma la definita positività della matrice A. Possiamo notare come la differenza in modulo fra la soluzione "teorica" e la soluzione calcolata dallo script sia piuttosto grande: dell'ordine di 10^{-5}. Questo non è sintomo di errori di approssimazione nel calcolo della soluzione $\boldsymbol{x}^*$ effettuata nella parte teorica, bensì è coerente con il fatto che nello script abbiamo riportato (a mano) l'approssimazione alla quarta cifra frazionaria di

tale soluzione! Dunque, le differenze fra la soluzione $\boldsymbol{x}^*$ e i valori approssimati nel vettore `xTeoria` devono essere al più sulla quinta cifra decimale, ossia dell'ordine di 10^{-5}, come in effetti è. Per constatare che la soluzione calcolata nella parte teorica coincide con quella nel vettore `x1` calcolato nello script, dovremmo riportare in `xTeoria` tutte le cifre decimali (avendo altresì avuto cura di effettuare i calcoli solamente in Matlab, non scrivendo valori intermedi approssimati e ripartendo usando quelli nei calcoli successivi). Invece di trascrivere tutti i decimali, la stessa cosa si può ottenere salvando il vettore $\boldsymbol{x}^*$ in un file `.mat` al termine della parte teorica e ri-caricandolo al termine dell'esecuzione dello script `Esercizio3`, per effettuare il confronto. Se si procede in tal modo, si riscontrano infatti differenze in modulo al più dell'ordine della precisione di macchina (circa $8.9 \cdot 10^{-16}$).

Soluzione esercizio 4

Teoria

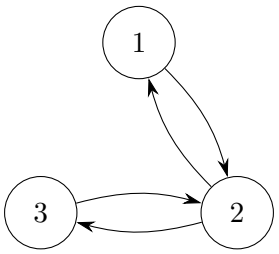
La matrice M è **tridiagonale**, ma **non simmetrica**. Si vede immediatamente che M è **strettamente diagonale dominante** sia per righe, che per colonne:

$$\begin{aligned} &\text{per righe:} && 3 > |-1/3|\,, && 2 > |-1/2| + 1/6 = 2/3\,, && |-3| > |-1/5|\,, \\ &\text{per colonne:} && 3 > |-1/2|\,, && 2 > |-1/3| + |-1/5| = 8/15\,, && |-3| > 1/6\,. \end{aligned}$$

Pertanto, possiamo già affermare che i metodi di Jacobi e di Gauss-Seidel per il sistema lineare $M\boldsymbol{x} = \boldsymbol{z}$ sono entrambi convergenti e $\|\mathcal{G}\|_\infty < \|J\|_\infty$ (ma da questo non possiamo concludere che uno dei due sia asintoticamente più veloce dell'altro, in quanto la disuguaglianza delle norme **non** implica che $\rho(\mathcal{G}) < \rho(J)$). Tuttavia, essendo M tridiagonale a diagonale non nulla ed essendo convergenti entrambi i metodi di Jacobi e di Gauss-Seidel, possiamo già affermare che il metodo di Gauss-Seidel converge con velocità asintotica **doppia** rispetto a quella del metodo di Jacobi. Dovremo quindi aspettarci che il raggio spettrale $\rho(\mathcal{G})$ della matrice $\mathcal{G}$ di Gauss-Seidel risulti il quadrato del raggio spettrale $\rho(J)$ della matrice di Jacobi.

Inoltre, la matrice M **è invertibile**, in quanto strettamente diagonale dominante (e dunque certamente 0 **non** appartiene allo spettro di M).

Poiché la matrice M è tridiagonale con elementi tutti non nulli nella diagonale principale e nella prima sopra-diagonale, possiamo subito affermare che M è **irriducibile**. Possiamo facilmente avere conferma di questa conclusione costruendo il grafo della matrice M:



Si vede immediatamente che il grafo di M **è strettamente connesso**, in quanto partendo da qualunque nodo si arriva in qualune altro nodo del grafo. Pertanto, la matrice M è **irriducibile**.

La matrice M è **non simmetrica**, quindi non possiamo applicare i teoremi che legano i metodi di Jacobi e Gauss-Seidel alle proprietà di definita positività (di M e di $2D - M$, con $D = \mathrm{diag}(M)$).

Tuttavia, essendo M **tridiagonale**, possiamo utilizzare i risultati che legano la convergenza del metodo SOR a quella del metodo di Jacobi e possiamo anche utilizzare l'espressione per il parametro ottimale di rilassamento ω^* di SOR.

Determiniamo la matrice J di iterazione di Jacobi:

$$\begin{aligned} J = D^{-1}(L+U) = I_3 - D^{-1}M &= \begin{pmatrix} 1/3 & 0 & 0 \\ 0 & 1/2 & 0 \\ 0 & 0 & -1/3 \end{pmatrix} \begin{pmatrix} 0 & 1/3 & 0 \\ 1/2 & 0 & -1/6 \\ 0 & 1/5 & 0 \end{pmatrix} \\ &= \begin{pmatrix} 0 & 1/9 & 0 \\ 1/4 & 0 & -1/12 \\ 0 & -1/15 & 0 \end{pmatrix} \approx \begin{pmatrix} 0 & 0.1111 & 0 \\ 0.2500 & 0 & -0.0833 \\ 0 & -0.0667 & 0 \end{pmatrix}. \end{aligned}$$

Vediamo che **non vale** la condizione $J \geq 0$: ne discende che **non** sono verificate la ipotesi del Teorema di Stein-Rosenberg. Per stabilire le rispettive velocità asintotiche di convergenza procediamo al calcolo esplicito dei raggi spettrali delle due matrici di iterazione.

Determiniamo gli autovalori della matrice J:

$$\det\big(J-\lambda I_3\big)=\begin{vmatrix}-\lambda & 1/9 & 0\\ 1/4 & -\lambda & -1/12\\ 0 & -1/15 & -\lambda\end{vmatrix}=-\lambda^3-(-\lambda)\left(\frac{1}{4}\cdot\frac{1}{9}+\frac{1}{12}\cdot\frac{1}{15}\right)=-\lambda\left(\lambda^2-\frac{1}{30}\right)$$

$$\lambda_1(J)=0\qquad \lambda^2-\frac{1}{30}=0\ \Rightarrow\ \lambda_{2,3}(J)=\mp\frac{1}{\sqrt{30}}\quad\Rightarrow\quad \rho(J)=\max_{i=1,\dots,3}\big|\lambda_i(J)\big|=\frac{1}{\sqrt{30}}\approx 0.1826\,.$$

Il fatto che $\rho(J)<1$ conferma che il metodo di Jacobi è convergente.

Calcoliamo ora la matrice di iterazione di Gauss-Seidel. Per prima cosa occorre determinare la matrice inversa di $D-L$. In questo caso l'applicazione dell'algoritmo di Gauss-Jordan è molto semplice e veloce:

$$T=\big[D-L\,\big|\,I_3\big]=\left(\begin{array}{ccc|ccc}3 & 0 & 0 & 1 & 0 & 0\\ -1/2 & 2 & 0 & 0 & 1 & 0\\ 0 & -1/5 & -3 & 0 & 0 & 1\end{array}\right)\xrightarrow{T_{2,*}\leftarrow T_{2,*}+(1/6)T_{1,*}}\left(\begin{array}{ccc|ccc}3 & 0 & 0 & 1 & 0 & 0\\ 0 & 2 & 0 & 1/6 & 1 & 0\\ 0 & -1/5 & -3 & 0 & 0 & 1\end{array}\right)\to$$

$$\xrightarrow{T_{3,*}\leftarrow T_{3,*}+(1/10)T_{2,*}}\left(\begin{array}{ccc|ccc}3 & 0 & 0 & 1 & 0 & 0\\ 0 & 2 & 0 & 1/6 & 1 & 0\\ 0 & 0 & -3 & 1/60 & 1/10 & 1\end{array}\right)\to\left(\begin{array}{ccc|ccc}1 & 0 & 0 & 1/3 & 0 & 0\\ 0 & 1 & 0 & 1/12 & 1/2 & 0\\ 0 & 0 & 1 & -1/180 & -1/30 & -1/3\end{array}\right)$$

$$(D-L)^{-1}=\begin{pmatrix}1/3 & 0 & 0\\ 1/12 & 1/2 & 0\\ -1/180 & -1/30 & -1/3\end{pmatrix}\approx\begin{pmatrix}0.3333 & 0 & 0\\ 0.0833 & 0.5000 & 0\\ -0.0056 & -0.0333 & -0.3333\end{pmatrix}$$

Possiamo ora calcolare la matrice d'iterazione di Gauss-Seidel:

$$\mathcal{G}=(D-L)^{-1}U=\begin{pmatrix}1/3 & 0 & 0\\ 1/12 & 1/2 & 0\\ -1/180 & -1/30 & -1/3\end{pmatrix}\begin{pmatrix}0 & 1/3 & 0\\ 0 & 0 & -1/6\\ 0 & 0 & 0\end{pmatrix}=\begin{pmatrix}0 & 1/9 & 0\\ 0 & 1/36 & -1/12\\ 0 & -1/540 & 1/180\end{pmatrix}$$

$$\approx\begin{pmatrix}0 & 0.1111 & 0\\ 0 & 0.0278 & -0.0833\\ 0 & -0.0019 & -0.0056\end{pmatrix}.$$

Determiniamo gli autovalori della matrice $\mathcal{G}$:

$$\det\big(\mathcal{G}-\lambda I_4\big)=\begin{vmatrix}-\lambda & 1/9 & 0\\ 0 & (1/36)-\lambda & -1/12\\ 0 & -1/540 & (1/180)-\lambda\end{vmatrix}=-\lambda\left(\frac{1}{36}-\lambda\right)\left(\frac{1}{180}-\lambda\right)+\lambda\cdot\frac{1}{12}\cdot\frac{1}{540}$$

$$=-\lambda\left(\lambda^2-\left(\frac{1}{36}+\frac{1}{180}\right)\lambda+\cancel{\frac{1}{36}\cdot\frac{1}{180}}-\cancel{\frac{1}{12}\cdot\frac{1}{540}}\right)=-\lambda^2\left(\lambda-\frac{1}{30}\right)$$

$$\Rightarrow\ \lambda_{1,2}(\mathcal{G})=0,\ \lambda_3(\mathcal{G})=\frac{1}{30}\ \Rightarrow\ \rho(\mathcal{G})=\max_{i=1,\dots,3}\big|\lambda_i(\mathcal{G})\big|=\frac{1}{30}\approx 0.0333\,.$$

Il fatto che $\rho(\mathcal{G})=1/30\approx 0.0333<1$ conferma che anche il metodo di Gauss-Seidel è convergente. Inoltre, osserviamo che $\rho(\mathcal{G})=\rho(J)^2$, come atteso: se ne deduce che il metodo di Gauss-Seidel converge con velocità asintotica *doppia* rispetto al metodo di Jacobi:

$$R_\infty(\mathcal{G})=-\ln\big(\rho(\mathcal{G})\big)=-\ln\big(\rho^2(J)\big)=-2\ln\big(\rho(J)\big)=2R_\infty(J)\qquad R_\infty(\mathcal{G})\approx 3.4012\qquad R_\infty(J)\approx 1.7006\,.$$

Esprimiamo infine la matrice di iterazione $\mathcal{G}(\omega)$ del metodo SOR come prodotto delle due matrici triangolari, per un generico valore di $\omega\in\,]0,2[$:

$$\mathcal{G}(\omega)=(D-\omega L)^{-1}\big((1-\omega)D+\omega U\big)=\begin{pmatrix}3 & 0 & 0\\ -\omega/2 & 2 & 0\\ 0 & -\omega/5 & -3\end{pmatrix}^{-1}\begin{pmatrix}3(1-\omega) & \omega/3 & 0\\ 0 & 2(1-\omega) & -\omega/6\\ 0 & 0 & 3(\omega-1)\end{pmatrix}.$$

Come osservato in precedenza, essendo M tridiagonale, è possibile utilizzare i risultati visti a lezione per la determinazione del parametro ottimale ω^* di rilassamento ed il corrispondente raggio spettrale ottimale $\rho\big(\mathcal{G}(\omega^*)\big)$:

$$\omega^*=\frac{2}{1+\sqrt{1-\rho^2(J)}}=\frac{2}{1+\sqrt{1-1/30}}\approx 1.0085\qquad\Rightarrow\qquad \rho^*=\rho\big(\mathcal{G}(\omega^*)\big)=1-\omega^*\approx 0.0085$$

Matlab

Contenuto dell'M-script file `esercizio4.m`:

```
% Cognome Nome
% Matricola
%-----------------------------
%  esercizio 4 - 08/09/2023
%-----------------------------
close all; clear all; clc
disp('Esercizio 4');

M = [3 -1/3 0; -0.5 2 1/6; 0 -1/5 -3];
z = [-1.5; 0.7; 1/6];
x0 = 2*ones(size(z));

disp('Metodo di Jacobi:');
d = 1 ./ diag(M);
J = -diag( d ) * (tril(M, -1) + triu(M, 1))
cJ = z .* d
rhoJ = abs(eigs(J,1)) % oppure rhoJ = max( abs(eig(J)) ) perche' J e' piccola
RinfJ = -log( rhoJ )

disp('Metodo di Gauss-Seidel:');
GS = tril(M) \ [-triu(M, 1) z];
cGS = GS(:, end), GS(:, end) = []
rhoGS = abs(eigs(GS,1)) % oppure rhoJ = max( abs(eig(J)) ) perche' GS e' piccola
RinfGS = -log( rhoGS )

xJk = x0;
for k = 1:5
    xJk = J*xJk + cJ;
end
fprintf('\nApproxs. a 5 iterazioni con il metodo di Jacobi:')
xJk

xGSk = x0;
for k = 1:5
    xGSk = GS*xGSk + cGS;
end
fprintf('\nApproxs. a 5 iterazioni con il metodo di Gauss-Seidel:')
xGSk

fprintf('\nSoluzion del sistema e differenze relative con xJk e xGSk:')
xs = M \ z

errRelJ = norm(xJk - xs) / norm(xs)
errRelGS = norm(xGSk - xs) / norm(xs)
```

Eseguendo lo script, si ottengono le stesse matrici J e $\mathcal{G}$ determinate nella parte teorica (impostando la visualizzazione a "`format rational`" si possono controllare i valori anche in forma di frazione). Lo stesso accade per i raggi spettrali e le velocità asintotiche di convergenza. Abbiamo quindi conferma che i calcoli svolti a mano nella parte teorica sono corretti.

La soluzione de sistema $M\boldsymbol{x} = \boldsymbol{z}$ risulta $\boldsymbol{x}^* \approx (-0.4736,\, 0.2375,\, -0.0714)^T$. Il metodo di Jacobi avviato da $\boldsymbol{x}^{(0)} = (2\,,2\,,2)^T$ e arrestato dopo esattamente 5 iterazioni determina un'approssimazione della soluzione data da $\boldsymbol{x}_{\mathrm{J}}^{(5)} \approx (-0.4734,\, 0.2380,\, -0.0715)^T$. L'approssimazione raggiunta dal metodo di Gauss-Seidel dopo esattamente 5 iterazioni, partendo dallo stesso punto iniziale, è $\boldsymbol{x}_{\mathrm{GS}}^{(5)} \approx (-0.4736,\, 0.2375,\, -0.0714)^T$. I valori delle rispettive differenze relative rispetto a $\boldsymbol{x}^*$ sono:

$$\left\|\boldsymbol{x}_{\mathrm{J}}^{(5)} - \boldsymbol{x}^*\right\|_2 / \left\|\boldsymbol{x}^*\right\|_2 \approx 0.001 = 0.1\%\,, \qquad \left\|\boldsymbol{x}_{\mathrm{GS}}^{(5)} - \boldsymbol{x}^*\right\|_2 / \left\|\boldsymbol{x}^*\right\|_2 \approx 9.939 \cdot 10^{-7} \approx 0.0001\%\,.$$

Si osserva che in entrambi i casi le componenti delle approssimazione si stanno avvicinando a quelle della soluzione $\boldsymbol{x}^*$ (il che è coerente con la convergenza dei due metodi già dimostrata nella parte teorica), ma l'accuratezza raggiunta dal metodo di Gauss-Seidel in sole 5 iterazioni è decisamente migliore di quella raggiunta dal metodo di Jacobi dopo lo stesso numero di iterazioni.

Tuttavia, anche se le differenze relative mostrano chiaramente per entrambi i metodi un avvicinamento delle approssimazioni alla soluzione, **non è lecito** trarre da ciò conclusioni riguardo la convergenza e la velocità asintotica di convergenza di ciascun metodo! Infatti, i risultati di convergenza visti non stabiliscono un avvicinamento monotòno delle iterazioni, anche qualora siano convergenti. Soprattutto, il comportamento **al finito** (ossia dopo un numero finito di iterazioni) delle due successioni di approssimazioni può essere alternato: per un certo numero di iterazioni potrebbe essere più veloce il metodo di Jacobi, poi potrebbe diventarlo il metodo di Gauss-Seidel, poi di nuovo il metodo di Jacobi... Non è detto che quest'alternanza si verifichi necessariamente, ma può accadere. Questo perchè i risultati di convergenza non danno informazioni riguardo la monotonìa delle iterazioni e soprattutto sono di carattere **asintotico**, ossia valgono per un numero di iterazioni che tende all'infinito. Pertanto, da un certo numero di iterazioni (non noto) in poi, siamo certi che il metodo di Gauss-Seidel convergerà alla soluzione più rapidamente di quello di Jacobi (nel caso in esame, con velocità esattamente doppia). Tuttavia, solo osservando il comportamento dopo poche iterazioni non possiamo dedurre che questo sia vero. Sono solo i risultati teorici che ne danno la garanzia.

Soluzione esercizio 5

Teoria

Notiamo innanzitutto che la funzione $f(x) = (1/2)\cos(3x-1)$ è ben definita su tutto $\mathbb{R}$. Essendo stati assegnati 4 punti distinti $x_j \in [-1.1, 1.5]$, $j = 0, \dots, 3$, possiamo costruire un **unico** polinomio interpolante di grado al più $n = 3$. Costruiamo il polinomio interpolante nella forma di Newton per la funzione $f(x)$ sull'intero intervallo $[-1.1, 1.5]$ usando la tabella delle differenze divise:

x_k	$f(x_k)$	$f[x_k, x_{k+1}]$	$f[x_k, x_{k+1}, x_{k+2}]$	$f[x_0, x_1, x_2, x_3]$
-1.1	-0.2004			
-0.2	-0.0146	0.2064		
0.5	0.4388	0.6477	0.2758	
1.5	-0.4682	-0.9070	-0.9145	-0.4578

Dalla tabella ricaviamo l'espressione del polinomio di interpolazione nella forma di Newton:

$$\begin{aligned} p_3^{\mathrm{N}}(x) &\approx -0.2004 + 0.2064(x-x_0) + 0.2758(x-x_0)(x-x_1) + (-0.4578)(x-x_0)(x-x_1)(x-x_2) \\ &= -0.2004 + 0.2064\big(x-(-1.1)\big) + 0.2758\big(x-(-1.1)\big)\big(x-(-0.2)\big) \\ &\qquad - 0.4578\big(x-(-1.1)\big)\big(x-(-0.2)\big)(x-0.5) \\ &= -0.2004 + 0.2064(x+1.1) + 0.2758(x+1.1)(x+0.2) - 0.4578(x+1.1)(x+0.2)(x-0.5)\,. \end{aligned}$$

Svolgendo i calcoli, si arriva molto rapidamente alla forma canonica del polinomio interpolante sui nodi dati:

$$\begin{aligned} p_3(x) &\approx -0.2004 + 0.2064(x+1.1) + 0.2758(x+1.1)(x+0.2) - 0.4578(x+1.1)(x+0.2)(x-0.5) \\ &\approx -0.2004 + 0.2064x + 0.2270 + 0.2758\big(x^2 + 1.3x + 0.22\big) - 0.4578\big(x^3 + 0.8x^2 - 0.43x - 0.11\big) \\ &\approx -0.2004 + 0.2064x + 0.2270 + 0.2758x^2 + 0.3310x + 0.0607 - 0.4578x^3 - 0.3662x^2 + 0.1966x + 0.0504 \\ &\approx -0.4578x^3 - 0.0905x^2 + 0.7618x + 0.1377\,. \end{aligned}$$

Nota. I valori riportati sono ottenuti eseguendo tutte le operazioni in Matlab, mantenendo in memoria i risultati intermedi e riportando solo le prime quattro cifre frazionarie. Se invece si eseguono le operazioni partendo dai valori approssimati riportati sul foglio, i valori dei coefficienti ottenuti al termine delle operazioni possono essere leggermente diversi, a causa dell'accumulo degli errori di approssimazione dei valori intermedi.

Nota. La tabella delle differenze divise e i coefficienti del polinomio $p_3(x)$ si ottengono facilmente con poche righe di Matlab. In un ciclo di n iterazioni, con n il grado del polinomio interpolante, ad ogni k-esima iterazione si generano la colonna $(k+2)$-esima della tabella, il prodotto dei polinomi $(x-x_0)\cdots(x-x_k)$ e la forma canonica del polinomio $p_k(x)$ di grado k di interpolazione sui primi $k+1$ nodi $x_0, \dots, x_k$. Per il prodotto dei polinomi si usa la funzione predefinita `conv` (*convoluzione*, facilmente rintracciabile nella documentazione di Matlab mediante `help poly`). Sia dunque `y` il vettore colonna che contiene gli $n+1$ valori della funzione da interpolare (ossia `y` è la seconda colonna della tabella delle differenze divise), partendo da `y(1)` $= f(x_0)$. Allora:

```
d = y; s = [1]; p = zeros(1, (n+1)); p(end) = d(1);
for k = 1 : n
```

```
    d((k+1) : end) = diff( d(k : end) ) ./ ( x((k+1) : end) - x(1 : (end-k)) )
    s = conv(s, [1, -x(k)]);
    p((end-k) : end) = p((end-k) : end) + d(k+1)*s;
end
```

Al termine, il vettore colonna `d` contiene gli elementi diagonali della tabella delle differenze divise (iniziando con `d(1)` $= f(x_0)$), il vettore riga `s` contiene i coefficienti della forma canonica del polinomio $\omega_n(x) = (x - x_0)(x - x_1)\cdots(x - x_{n-1})$ e il vettore riga `p` contiene esattamente i coefficienti della forma canonica del polinomio interpolante $p_n(x)$ di grado n su tutti i nodi $x_0, x_1, \ldots, x_n$.

Evitando di concludere con il punto e virgola la prima riga del ciclo (ed inserendo eventualmente un `pause` al termine del corpo del ciclo), nelle componenti dalla $(k+1)$-esima all'ultima del vettore `d` si leggono i valori da inserire nella $(k+2)$-esima colonna della tabella.

Nel caso del presente esercizio, essendo $n = 3$, per il polinomio interpolante $p_3(x)$ i vettori `y`, `d` e `p` hanno quattro componenti e il ciclo effettua tre sole iterazioni. Alla prima iterazione, i valori in `d(2:4)` sono quelli della terza colonna della tabella delle differenze divise, alla seconda iterazione in `d(3:4)` si leggono i due elementi della quarta colonna della tabella e alla terza (e ultima) iterazione il valore in `d(4)` contiene l'unico valore diverso da zero nella quinta (e ultima) colonna della tabella.

L'espressione generale dell'errore di interpolazione di un polinomio interpolante di grado al più n su $n+1$ punti distinti x_j, $j = 0, \ldots, n$, di un generico intervallo $[a, b]$ per una funzione $f \in C^{n+1}\big([a, b]\big)$ si scrive come

$$R_n(x) = \omega_{n+1}(x)\frac{f^{(n+1)}(\xi_x)}{(n+1)!} \qquad \text{con} \quad \xi_x \in]a, b[\quad \text{e} \quad \omega_{n+1}(x) = (x - x_0)(x - x_1)\cdots(x - x_n)\,.$$

Nel caso in esame si ha:

$$\begin{gathered} f'(x) = -(3/2)\sin(3x - 1)\,, \quad f''(x) = -(9/2)\cos(3x - 1) \\ f'''(x) = (27/2)\sin(3x - 1)\,, \quad f^{(4)}(x) = (81/2)\cos(3x - 1) \\ \Rightarrow \quad R_3(x) = (x + 1.1)(x + 0.2)(x - 0.5)(x - 1.5)\frac{1}{4!}\Big((81/2)\cos(3\xi_x - 1)\Big) \quad \text{con } \xi_x \in]-1.1,\, 1.5[\,. \end{gathered}$$

Per avere una maggiorazione dell'errore di interpolazione nel punto $x_* = 0.9$, dobbiamo maggiorare il modulo della derivata quarta: dato che $|\cos(t)| \le 1\ \forall t \in \mathbb{R}$, possiamo scrivere $\big|f^{(4)}(x)\big| \le 81/2\ \forall x \in \mathbb{R}$, in particolare $\forall x \in [-1.1, 1.5]$. Possiamo dunque scrivere

$$\big|R_3(x_*)\big| \le \big|(0.9 + 1.1)(0.9 + 0.2)(0.9 - 0.5)(0.9 - 1.5)\big|\frac{81}{2} = \big|2.0 \cdot 1.1 \cdot 0.4 \cdot (-0.6)\big|\frac{81}{2} = 0.528 \cdot 40.5 = 21.384\,.$$

Infine, i moduli degli errori (effettivi) di interpolazione in $x_* = 0.9$ e nei tre punti $z_1 = -0.9$, $z_2 = -0.1$ e $z_3 = 1.2$ sono

$$\begin{array}{rcl} f(0.9) \approx -0.0644\,, \quad p_3(0.9) \approx 0.4163 & \Rightarrow & \big|f(0.9) - p_3(0.9)\big| \approx |-0.0644 - 0.4163| \approx 0.4808 \\ f(-0.9) \approx -0.4241\,, \; p_3(-0.9) \approx -0.2875 & \Rightarrow & \big|f(-0.9) - p_3(-0.9)\big| \approx |-0.4241 - -0.2875| \approx 0.1366 \\ f(-0.1) \approx 0.1337\,, \quad p_3(-0.1) \approx 0.0611 & \Rightarrow & \big|f(-0.1) - p_3(-0.1)\big| \approx |0.1337 - 0.0611| \approx 0.0727 \\ f(1.2) \approx 1.6805\,, \quad p_3(1.2) \approx 0.1305 & \Rightarrow & \big|f(1.2) - p_3(1.2)\big| \approx |1.6805 - 0.1305| \approx 0.5590 \end{array}$$

In senso assoluto, gli errori effettivi sembrano valori "piccoli" e si sarebbe quindi tentati di dedurre che le stime fornite dal polinomio di interpolazione siano buone stime dei valori veri di $f(x)$ nei punti dati. Tuttavia, questa conclusione è affrettata e, in effetti, non può essere dedotta semplicemente dagli errori assoluti, perchè gli errori assoluti **non** tengono conto dell'ordine di grandezza dei valori confrontati. Possiamo ancora notare che il modulo dell'errore di interpolazione in $x_* = 0.9$, che vale circa 0.4808, è effettivamente inferiore alla stima ottenuta maggiorando l'espressione analitica dell'errore di interpolazione nello stesso punto x_*.

Per avere una valutazione ragionevole dell'accuratezza con cui il polinomio interpolante approssima la funzione $f(x)$ nei punti dati occorre considerare i moduli degli **errori relativi** effettivi di interpolazione: $\big|f(x) - p_3(x)\big|/\big|f(x)\big|$. Calcolando queste quantità ed esprimendole in percentuale si riscontra allora che in z_1, z_2 e z_3 si hanno moduli degli errori relativi effettivi del 32.3%, 54.3% e addirittura del 130.5%, rispettivamente: significa che le approssimazioni date dal polinomio in questi punti sono **cattive**. Ancor peggio accade in $x_* = 0.9$, dove la percentuale di oltre il 746% indica che la distanza fra l'approssimazione fornita dal polinomio e il valore della funzione nello stesso punto è quasi sette volte e mezza il valore stesso della funzione!!

In figura 5 sono mostrati i comportamenti della funzione $f(x)$ e del polinomio interpolante $p_3(x)$ nell'intervallo $[-1.1, 1.5]$ (nel pannello di sinistra), oltre al grafico dell'errore di interpolazione sullo stesso intervallo (nel pannello di destra).

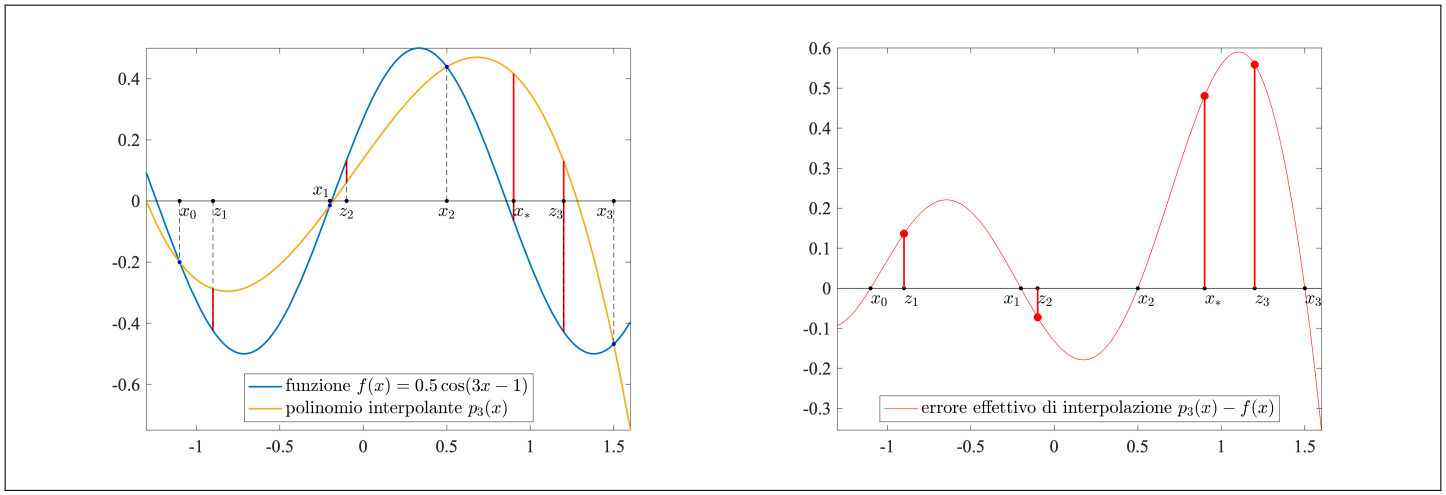


Figura 5: a sinistra: grafici della funzione $f(x) = (1/2)\cos(3x-1)$ e del polinomio interpolante $p_3(x)$ di grado 3 sui nodi x_j, $j = 0, \ldots, 3$; a destra: grafico dell'errore di interpolazione(figure leggermente migliorate per esigenze di stampa).

Matlab

Contenuto dell'M-script file `esercizio5.m`:

```
% Cognome Nome
% Matricola
%-----------------------------
%  esercizio 5 - 08/09/2023
%-----------------------------
close all; clear all; clc
disp('Esercizio 5');

xj = [-1.1, -0.2, 0.5, 1.5]';
f = @(x)(0.5*cos(3*x-1)); yj = f(xj);

fprintf('\nCoeff. pol. di interp. di f(x) in xj nella forma di Newton:\n')
d = tabDiff(xj, yj)

xs = [0.9, -0.9, -0.1, 1.2]';  % xs = (x*, z1, z2, z3)
ys = d(1); t = ones(size(xs));
for k = 2 : numel(d)
    t = t .* (xs - xj(k-1));
    ys = ys + d(k)*t;
end
fprintf('\nValori del pol. interp. p3(x) in x*, z1, z2, z3:\n')
disp(ys)

% in cio' che segue, si potrebbero riutilizzare i vettori xj e yj, sovrascrivendoli

z1 = -0.9; z2 = -0.1; x2 = [xj; z1; z2]; y2 = [yj; f(z1); f(z2)];
fprintf('\nCoeff. pol. interp. di f(x) in x2 nella forma di Newton:\n')
d2 = tabDiff(x2, y2)
```

Eseguendo lo script si ottiene il seguente output, che conferma i risultati della parte teorica:

```
Esercizio 5
Coeff. pol. interp. di f(x) in xj nella forma di Newton:
d =
   -0.2004
    0.2064
    0.2758
   -0.4578

Valori del pol. interp. p3(x) in x*, z1, z2, z3:
    0.4163
   -0.2875
    0.0611
    0.1305

Coeff. pol. interp. di f(x) in x2 nella forma di Newton:
d2 =
   -0.2004
    0.2064
    0.2758
   -0.4578
    0.2904
    0.5831
```

Soluzione esercizio 6

Teoria

Per impostare il sistema delle equazioni normali, costruiamo dapprima la matrice di Vandermonde relativa ai nodi x_i, $i = 0, \dots, 6$:

$$A = \begin{pmatrix} x_0^2 & x_0 & 1 \\ x_1^2 & x_1 & 1 \\ \vdots & \vdots & \vdots \\ x_6^2 & x_6 & 1 \end{pmatrix} \approx \begin{pmatrix} 0.5625 & 0.7500 & 1.0000 \\ 0.7225 & -0.8500 & 1.0000 \\ 0.2500 & -0.5000 & 1.0000 \\ 2.6244 & 1.6200 & 1.0000 \\ 1.7689 & -1.3300 & 1.0000 \\ 12.4609 & -3.5300 & 1.0000 \\ 5.5696 & 2.3600 & 1.0000 \\ 3.9204 & -1.9800 & 1.0000 \end{pmatrix}$$

Costruiamo poi le matrici $B = A^T A$ e il vettore $A^T \boldsymbol{y}$, dove $\boldsymbol{y} = (y_0, \dots, y_6)^T$:

$$B = A^T A \approx \begin{pmatrix} 212.5814 & -37.0235 & 27.8792 \\ -37.0235 & 27.8792 & -3.4600 \\ 27.8792 & -3.4600 & 8.0000 \end{pmatrix} \qquad \boldsymbol{b} = A^T \boldsymbol{y} \approx \begin{pmatrix} 50.8225 \\ -24.2447 \\ -0.0500 \end{pmatrix}$$

Ora, i coefficienti del polinomio $p_2^*(x)$ di grado al più 2 di migliore approssimazione nel senso dei minimi quadrati dei dati forniti sono le componenti della soluzione $\boldsymbol{\alpha}^*$ del sistema delle equazioni normali $B\boldsymbol{\alpha} = \boldsymbol{b}$.

Dato che si tratta di un sistema pieno e piccolo, possiamo utilmente applicare appena due passi del metodo di Gauss con pivoting parziale:

$$\big[B\big|\boldsymbol{b}\big] = \big[B^{(1)}\big|\boldsymbol{b}^{(1)}\big] \approx \left(\begin{array}{ccc|c} 212.5814 & -37.0235 & 27.8792 & 50.8225 \\ -37.0235 & 27.8792 & -3.4600 & -24.2447 \\ 27.8792 & -3.4600 & 8.0000 & -0.0500 \end{array}\right)$$

$$P_1 = I_3, \quad \boldsymbol{m}^{(1)} \approx \begin{pmatrix} 0.0000 \\ -0.1742 \\ 0.1311 \end{pmatrix}, \quad L_1 \approx \begin{pmatrix} 1.0000 & & \\ 0.1742 & 1.0000 & \\ -0.1311 & 0.0000 & 1.0000 \end{pmatrix}$$

$$\big[B^{(2)}\big|\boldsymbol{b}^{(2)}\big] = L_1 P_1 \big[B^{(1)}\big|\boldsymbol{b}^{(1)}\big] \approx \left(\begin{array}{ccc|c} 212.5814 & -37.0235 & 27.8792 & 50.8225 \\ 0.0000 & 21.4311 & 1.3955 & -15.3934 \\ 0.0000 & 1.3955 & 4.3438 & -6.7152 \end{array}\right)$$

$$P_2 = I_3, \quad \boldsymbol{m}^{(2)} \approx \begin{pmatrix} 0.0000 \\ 0.0000 \\ 0.0651 \end{pmatrix}, \quad L_2 \approx \begin{pmatrix} 1.0000 & & \\ 0.0000 & 1.0000 & \\ 0.0000 & -0.0651 & 1.0000 \end{pmatrix}$$

$$\big[R\big|\boldsymbol{c}\big] = L_2 P_2 \big[B^{(2)}\big|\boldsymbol{b}^{(2)}\big] \approx \left(\begin{array}{ccc|c} 212.5814 & -37.0235 & 27.8792 & 50.8225 \\ 0.0000 & 21.4311 & 1.3955 & -15.3934 \\ 0.0000 & 0.0651 & 4.2529 & -5.7128 \end{array}\right)$$

$$\alpha_3^* = c_3 / r_{33} \approx \frac{-5.7128}{4.2529} \approx -1.3433$$

$$\alpha_2^* = \frac{c_2 - r_{23}\alpha_3^*}{r_{22}} \approx \frac{-15.3934 - 1.3955 \cdot (-1.3433)}{21.4311} \approx -0.6308$$

$$\alpha_1^* = \frac{c_1 - r_{12}\alpha_2^* - r_{13}\alpha_3^*}{r_{11}} \approx \frac{50.8225 - (-37.0235) \cdot (-0.6308) - 27.8792 \cdot (-1.3433)}{212.5814} \approx 0.3054$$

$$\Rightarrow \ \boldsymbol{\alpha}^* \approx \begin{pmatrix} 0.3054 \\ -0.6308 \\ -1.3433 \end{pmatrix}$$

Dunque, il polinomio $p_2^*(x)$ richiesto è

$$p_2^*(x) = \alpha_1^* x^2 + \alpha_2^* x + \alpha_3^* \approx 0.3054x^2 - 0.6308x - 1.3433$$

Il vettore dei moduli degli errori relativi nei punti x_i, $i = 0, \ldots, 6$, in forma standard e in forma percentuale, è:

$$\mathrm{err}_{\mathrm{rel}} = \begin{pmatrix} \dfrac{|y_0 - p_2^*(x_0)|}{|y_0|} \\ \dfrac{|y_1 - p_2^*(x_1)|}{|y_1|} \\ \vdots \\ \dfrac{|y_6 - p_2^*(x_6)|}{|y_6|} \end{pmatrix} \approx \begin{pmatrix} |-1.27 - 0.0750| \,/\, |-1.27| \\ |-1.92 - (-1.3705)| \,/\, |-1.92| \\ |4.04 - (-1.2773)| \,/\, |4.04| \\ |-3.06 - (-0.3825)| \,/\, |-3.06| \\ |-4.52 - 2.6877| \,/\, |-4.52| \\ |5.02 - 6.6557| \,/\, |5.02| \\ |-0.68 - 14.6320| \,/\, |-0.68| \\ |2.34 - 14.6320| \,/\, |2.34| \end{pmatrix} \approx \begin{pmatrix} 0.2950 \\ 0.6946 \\ 1.2355 \\ 0.4890 \\ 1.0079 \\ 0.0660 \\ 0.6635 \\ 0.5287 \end{pmatrix} \approx \begin{pmatrix} 29.45\% \\ 69.46\% \\ 123.55\% \\ 48.90\% \\ 100.79\% \\ 6.60\% \\ 66.35\% \\ 52.87\% \end{pmatrix}$$

Trattandosi di approssimazione e non di interpolazione, vediamo che l'errore relativo nei punti dati può essere anche molto grande e arrivare oltre l'ordine di grandezza dell'osservazione stessa, come accade per x_2.

Nota. Tutti i calcoli precedenti si svolgono molto facilmente in Matlab: una volta definiti i due vettori

```
x = [ 0.75  -0.85  -0.50   1.62  -1.33  -3.53   2.36  -1.98]';
y = [-1.27  -1.92   4.04  -3.06  -4.52   5.02  -0.68   2.34]';
```

le matrici richieste per il sistema delle equazioni normali e il vettore soluzione (che contiene i coefficienti $\boldsymbol{\alpha}^*$ del polinomio approssimante $p_2^*(x)$ richiesto) si ottengono con i semplici comandi

```
A = [x.*x, x, ones(size(x))], B = A'*A, b = A'*y, p2 = B \ b
```

Il calcolo delle matrici e dei passaggi intermedi del metodo di Gauss con pivoting parziale è altrettanto veloce (si noti che non sono necessari scambi di righe):

```
B1 = [A, b], m1 = [0; B1(2:3,1) / B1(1,1)], L1 = eye(3) - m1*[1, 0, 0]
B2 = L1*B1, m2 = [0; 0; B2(3,2) / B2(2,2)], L2 = eye(3) - m2*[0, 1, 0]
R = L2*B2; c = R(:,4), R = triu(R(:,1:3)), alpha = R \ c
```

Le M-function `ltrisol` e `utrisol` possono essere eventualmente utilizzate per controllare i risultati.

Il vettore `alpha` **DEVE** coincidere con il vettore `p2` precedentemente ottenuto, a meno di valori vicini alla precisione di macchina. Per controllare questa cosa basta usare il comando `abs(p2 - alpha)`, che dovrebbe dare un vettore di valori nulli o con ordini di grandezza non superiori a 10^{-15}.

Infine, per avere i valori dell'errore relativo nei punti dati, basta usare il comando

```
errRel = abs(y - polyval(p2, x)) ./ abs(y)
```

e la corrispondente rappresentazione percentuale si ha con `errRel*100`.

Matlab

Contenuto dell'M-function file `myBisection.m`:

```
function [c, fc, it] = myBisection(fname, a, b, tol)
% MYBISECTION - Implementa il metodo di bisezione per equazioni non lineari
% SYNOPSIS:
%  [c, fc, it] = myBisection(fname, a, b, tol)
% INPUT:
%  fname (string/fhandle) - Nome function o function handle che calacola i
%                           valori della funzione
%  a, b  (double)         - Estremi dell'intervallo
%  tol   (double)         - Tolleranza nell'approssimazione della soluzione
% OUTPUT:
%  c  (double)  - Approssimazione della soluzione
%  fc (double)  - Valore della funzione nell'approssimazione della soluzione
%  it (integer) - Numero di iterazioni eseguite
  maxit = ceil( log2( (b - a)/tol ) );  it = 0;
  fa = feval(fname, a);  fb = feval(fname, b);
  if ( sign(fa)*sign(fb) > 0 ), error('Intervallo non corretto');
  elseif (fa == 0), c = a; fc = fa; return;
```

```
  elseif (fb == 0), c = b; fc = fb; return;
  else
    % Metodo di bisezione
    soglia = tol + eps*max( abs([a; b]) );  arresto = 0;
    while ( ~arresto )
      it = it + 1;    c  = a + (b - a)*0.5;
      fc = feval(fname, c);
      if ( fc == 0 ), break; end
      if ( sign(fc)*sign(fa) > 0 )
        a = c;  fa = fc;
      else
        b = c;  fb = fc;
      end
      arresto = ( abs(b - a) < soglia ) || ( it == maxit );
    end
  end
end
```

Contenuto dell'M-script file `esercizio6.m`:

```
% Cognome Nome
% Matricola
%----------------------------
%  esercizio 6 - 08/09/2023
%----------------------------
close all; clear all; clc

a = -4.5; b = 1; tolx = 1.0e-3;
f  = @(x)( 3*x.*sin(x + 2) - sqrt(5 - x) );

[xsB,fsB,itB] = myBisection(f,a,b,tolx);
fprintf('\nApprossimazione della radice con metodo di bisezione:');
fprintf('\nxs = %g   f(xs) = %g   iter = %d', xsB, fsB, itB);

f1 = @(x)( 3*(sin(x + 2) + x.*cos(x + 2)) + 1 ./ (2*sqrt(5 - x)) );
x0 = 0.5; tolf = tolx; maxit = 100;

[xsNewt,fsNewt,itNewt] = myNewton(f,f1,x0,tolx,tolf,maxit);
fprintf('\nApprossimazione della radice con metodo di Newton:');
fprintf('\nxsN = %g   f(xsN) = %g   iterN = %d\n\n', xsNewt, fsNewt, itNewt);

xx = linspace(a,b,201)';
fh(1) = figure;
ph(1) = plot(xx,f(xx));
th(1) = title('Funzione f(x) = 3xsin(x+2) - sqrt(5-x)');
fh(2) = figure;
ph(2) = plot(xx,f1(xx));
th(2) = title('Funzione f''(x) = 3(sin(x+2) + xcos(x+2)) + 1/(2sqrt(5-x))');
```

Eseguendo lo script si ottiene un output simile al seguente e i grafici di figura 6:

```
Approssimazione della radice con metodo di bisezione:
xs = -2.38849   f(xs) = -0.00396884   iter = 13
Approssimazione della radice con metodo di Newton:
xsN = -2.38901   f(xsN) = 8.17354e-08   iterN = 12
```

Come si vede, le approssimazioni dello zero di $f(x)$ in $[a, b]$ determinate dalle M-function `myNewton` e `myBisection` sono molto vicine e sono ottenute quasi con lo stesso numero di iterazioni.

Tuttavia, in questo caso, il valore assoluto della funzione ottenuto nell'approssimazione della radice dell'equazione con il metodo di Newton è migliore (più vicino a zero) di quello ottenuto il metodo di bisezione. Questo fatto è concorde con il fatto che la velocità di convergenza del metodo di Newton è **quadratica**, ossia, nelle vicinanze della radice dell'equazione, ogni nuova iterazione del metodo fornisce due ulteriori cifre corrette di accuratezza.

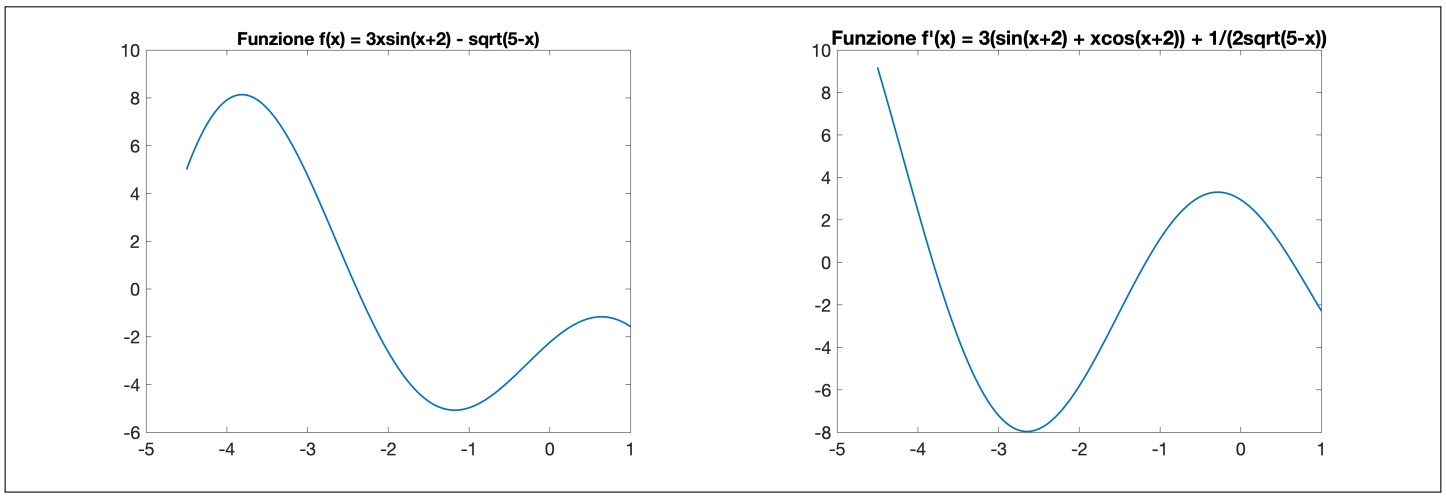


Figura 6: a sinistra: grafico della funzione $f(x) = 3x\sin(x+2)\sqrt{5-x}$ nell'intervallo $[-4.5, 1]$; a destra: grafico della derivata prima $f'(x) = 3\big(\sin(x+2) + x\cos(x+2)\big) + 1\big/\big(2\sqrt{5-x}\big)$ di $f(x)$ nello stesso intervallo (figure leggermente migliorate per esigenze di stampa).