Corso di Laurea in Informatica – Università di Ferrara
Calcolo Numerico
Prova scritta e pratica del 19/07/2023 – 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) Determinare la rappresentazione decimale del numero α la cui rappresentazione in formato ANSI standard IEEE a 32 bit è $\text{IEEE}_{32}(\alpha) = 11000101110110111101011101000000$.
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 $p_6(x)$ siano i logaritmi naturali dei 7 addendi più grandi che concorrono alla parte intera in base 10 del numero α del punto teorico precedente. Assegnare l'addendo più grande a c_0, il secondo più grande a c_1 e così via. Calcolare questi coefficienti **nel modo più conveniente, usando la sintassi vettoriale di Matlab**. Lo script visualizzi anche il grafico del polinomio nell'intervallo $[-1.4, 0.5]$. Provare lo script con il valore $x = 0.57$.

Esercizio 2

1) (T) (2 punti) Valutare l'errore inerente, l'errore algoritmico, l'indice di condizionamento e l'indice algoritmico nel calcolo dell'espressione:
$$\varphi(x) = \sqrt[3]{3.5 + \sin(2+x)}\,\ln(2+x)$$
   dove essa è definita. Esistono valori di x per i quali il problema tende ad essere mal condizionato? Motivare la risposta.
2) (M) (2 punti) Realizzare un M-function file per la valutazione della seguente funzione:
$$f_a(x) = 0.13 \cdot a^{4x} - 4.26 \cdot a^{3x} - 2.91 \cdot a^{2x} + 3.55 \cdot a^x + 9.14\,, \qquad a > 0,$$
   dotando la M-function di opportuni controlli sull'input. Per fissati valori scalari della variabile x e del parametro $a > 0$ (passati in input separatamente), lo script deve eseguire i seguenti passi:
   (a) calcolo di $f_a(x)$ mediante un ciclo `for`;
   (b) posto $y = a^x$, calcolo di $f_a(x)$ usando la funzione Matlab `polyval`;
   (c) calcolo di $f_a(x)$ mediante sintassi vettoriale, *senza* alcun ciclo.
   La M-function restituisca i valori calcolati utilizzando un *unico* parametro di uscita. Scrivere un M-script di test della M-function, che (i) acquisisca dall'utente i valori di x e di a mediante opportune funzioni di input, (ii) richiami la funzione di calcolo di $f_a(x)$, (iii) visualizzi a video i risultati ottenuti, utilizzando il formato di visualizzazione più esteso disponibile in Matlab e senza fare uso di istruzioni di stampa e, infine, (iv) determini

le differenze relative percentuali in modulo fra i valori ottenuti nei tre modi. Provare lo script con $a = 1.3$ e $x = 0.7$.

Opzionale (2 punti) *Se possibile*, scrivere una seconda versione della M-function, in modo tale che possa funzionare correttamente anche se in input viene passato un vettore `xx` di N componenti x_i, $i = 1, \ldots, N$, da usare come valori per la variabile x nel calcolo di $f_a(x)$, con a fissato.

Esercizio 3

1) (T) (4 punti) Risolvere con il metodo di eliminazione di Gauss con pivoting totale il seguente sistema lineare:

$$\begin{cases} (1/2)x_1 + (3/2)x_2 - (1/2)x_3 = -1 \\ x_1 + 2x_2 + 2x_3 = -2 \\ 4x_1 + 7x_2 - x_3 = 0 \end{cases}$$

Esplicitare tutte le matrici coinvolte nella fattorizzazione $PAQ = LR$ usata per risolvere il sistema. Calcolare il determinante di A, usando la fattorizzazione ottenuta. [*Suggerimento: conviene mantenere i valori in forma di frazione.*]

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 prime due righe e delle ultime due colonne. 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, sia utilizzando le M-function `gauss2`, `ltrisol` e `utrisol` in allegato, sia usando l'opportuna funzione predefinita di Matlab. Stampare a video soluzione e residuo normalizzato (in norma infinito). Calcolare poi la matrice $B = AA^T$ e verificare se essa è definita positiva, eseguendo la fattorizzazione di Cholesky mediante la M-function `chol` di Matlab. Stampare a video il risultato. Nel caso B sia definita positiva, risolvere il sistema $B\boldsymbol{x} = \boldsymbol{c}$, con $\boldsymbol{c} = (3/5, -2/3, -1/7)^T$, usando il fattore di Choleski e le M-function in allegato.

Esercizio 4

Si consideri la seguente matrice:

$$A = \begin{pmatrix} 2 & 0 & 1/2 \\ 0 & 4/3 & -5/4 \\ 2/3 & 0 & 1 \end{pmatrix}$$

1) (T) (4 punti) Disegnare i dischi di Gershgorin per righe e per colonne della matrice A e dedurre da questi se essa è invertibile oppure no, motivando la risposta. Determinare la riducibilità o meno di A. Stabilire *a priori* se i metodi iterativi di Jacobi e di Gauss-Seidel sono convergenti per un sistema lineare avente A come matrice dei coefficienti. Successivamente, determinare i rispettivi raggi spettrali e, qualora possibile, determinare le velocità asintotiche di convergenza. Scrivere l'espressione della matrice di iterazione per il metodo SOR relativa ad A. Se possibile, determinare il valore ottimale del parametro di rilassamento del metodo SOR ed il corrispondente raggio spettrale ottimale, motivando la risposta.
2) (M) (3 punti) Realizzare un M-script file che controlli i risultati ottenuti al punto precedente, determinando spettro, raggio spettrale e, se possibile, velocità asintotica di convergenza di entrambi i metodi. Visualizzare a monitor i risultati con istruzioni di stampa formattate. Successivamente, considerato il sistema lineare $A\boldsymbol{x} = \boldsymbol{b}$ con $\boldsymbol{b} = (1/3, -2, -1)^T$, lo script determini un'approssimazione $\boldsymbol{x}^{(\mathrm{J})}$ della soluzione $\boldsymbol{x}^*$ utilizzando la M-function `jacobi` (in allegato), con una tolleranza di arresto `tol` $= 10^{-5}$, un numero massimo di iterazioni pari a `maxit` $= 50$ e vettore iniziale $\boldsymbol{x}^{(0)} = (1, 1, 1)^T$. Lo script visualizzi a monitor il numero di iterazioni compiute e l'approssimazione calcolata. Infine, lo script costruisca la matrice di iterazione del metodo SOR utilizzando $\omega = 0.82$ come parametro di rilassamento e visualizzi a console la sesta iterazione, partendo da $\boldsymbol{x}^{(0)} = (1, 1, 1)^T$.

Esercizio 5

Si consideri la funzione: $f(x) = \pi - \ln(3x) + \sin(2x)$, $x \in [a, b]$.

1) (T) (4 punti) Ricavare l'espressione analitica del polinomio $p_3(x)$ di grado al più 3 di interpolazione di $f(x)$ in $[a, b] = [0.2, 5]$ su nodi di Chebychev, mediante il polinomio di Newton. Scrivere l'espressione dell'errore di interpolazione. Approssimare l'errore valutando tutte le quantità dell'espressione in $x^* = 3.9$. Confrontare tale valore con la differenza $f(x^*) - p_3(x^*)$.

Scrivere l'espressione dei polinomi di Lagrange su 3 punti equispaziati nell'intervallo $[a, b]$ dato sopra e la formula che fornisce il polinomio interpolante $p_2(x)$ di grado al più 2 su tali nodi nella forma di Lagrange.

2) (M) (4 punti) Scrivere un M-script file che calcoli il polinomio $p_n(x)$ di grado n interpolante la funzione $f(x)$ in $[a, b]$ sugli $n + 1$ nodi di Chebychev, utilizzando a tale scopo la function `polyLagrange` in allegato. Lo script generi poi in uno stesso grafico le due curve $f(x)$ e $p_n(x)$, evidenziando nodi e punti di interpolazione e dotando il grafico di titolo, etichette degli assi e legenda. Lo script valuti infine l'errore relativo percentuale $e_n(x) = |p_n(x) - f(x)|/|f(x)|$ negli N punti di un campionamento uniforme dell'intervallo $[a, b]$, disegnando il corrispondente grafico in una seconda finestra grafica. Utilizzare un campionamento di $[a, b]$ con $N \geq 201$. Evidenziare opportunamente in tale grafico i nodi di interpolazione.
Successivamente, lo script costruisca la spline cubica completa $s_3(x)$ interpolante $f(x)$ in $[a, b]$ sugli $n + 1$ nodi equispaziati utilizzando le functions `mySplineCompleta` e `valSpline` in allegato. Lo script aggiunga infine il grafico della spline e del corrispondente errore ai grafici già presenti nelle due figure.
Provare lo script con $n + 1 = 6$ ed $N = 201$.

Esercizio 6

1) (T) (3 punti) Si consideri la funzione $f(x)$ dell'esercizio 5 nell'intervallo $[a, b] = [4.5, 5.5]$. Dire se sono verificate le condizioni per l'esistenza di almeno uno zero di $f(x)$ in $[a, b]$. In caso affermativo, verificare le condizioni del teorema di convergenza globale del metodo di Newton in tale intervallo. Determinare poi quanti passi del metodo di bisezione sono necessari per approssimare uno zero di $f(x)$ in $[a, b]$ con un'accuratezza di almeno $\tau = 10^{-5}$. Calcolare esattamente due iterate dei seguenti metodi, esplicitando tutte le formule di aggiornamento: di Newton, partendo dal punto $x_0 = 5.3$; delle secanti, partendo da $x_0 = a$ e $x_1 = b$; di falsa posizione, prendendo dagli estremi dell'intervallo. Enunciare infine le condizioni sufficienti per l'esistenza e l'unicità di un punto fisso per una generica funzione $g(x)$ su un intervallo.
2) (M) (3 punti) Implementare il metodo di Newton in una M-function `myNewton.m`, dotata di opportuni controlli e prevedendo l'arresto sia rispetto ad una soglia τ_x sulla differenza degli iterati successivi, sia rispetto ad una soglia τ_f sul valore assoluto della funzione. Le soglie siano passate come parametri in ingresso. Costruire poi uno script di prova nel quale confrontare il calcolo della radice dell'equazione del punto precedente effettuato mediante tale M-function e mediante la funzione predefinita `fzero` di Matlab. Si fornisca ad entrambe le M-functions lo stesso punto iniziale $x_0 = 5.3$, le stesse soglie di arresto $\tau_x = 10^{-3}$ e $\tau_f = 10^{-4}$ e, infine, lo stesso numero massimo di iterazioni `maxit` $= 100$.

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 [L, R, p, deter] = gauss2(A)
% GAUSS2 - Fattorizzazione di Gauss con pivoting parziale (versione 2)
%  Esegue la fattorizzazione PA = LR con l'algoritmo di eliminazione di
%  Gauss con pivoting parziale (con aggiornamento mediante diadi)
% SYNOPSIS
%   [L, R, p, deter] = gauss2(A)
% INPUT
%   A (double array) - Matrice m x n
% OUTPUT
%   L (double array) - Matrice triangolare inferiore a diagonale unitaria
%   R (double array) - Matrice triangolare/trapezoidale superiore
%   p (double array) - Vettore delle permutazioni di righe
%   deter (double)   - determinante (della parte quadrata) di A
%
[m, n] = size(A); deter = 1;
temp = zeros(1, n); p = 1 : n;
tol = eps * norm(A, inf);  % tolleranza verso lo zero (trascurabilita')
for j = 1 : min(m-1,n)
   [amax, ind] = max( abs(A(j:n, j)) );
   ind = ind + j - 1;
   if (ind ~= j)  % pivot non in posizione diagonale: occorre scambiare righe
      % scambio di indici
      aux = p(j);  p(j) = p(ind);  p(ind) = aux;
      deter = -deter;
      % scambio di righe
      temp = A(ind,:); A(ind,:) = A(j,:); A(j,:) = temp;
   end
   deter = deter * A(j,j);
   if ( abs(A(j,j)) > tol )
      A(j+1:end, j) = A(j+1:end, j) / A(j,j);
      % operazione di base: aggiornamento mediante diadi
```

```
      A(j+1:end, j+1:end) = A(j+1:end, j+1:end) - A(j+1:end, j)*A(j, j+1:end);
   end
end
deter = deter * A(n,n);
R = triu(A);
L = eye(n) + tril(A(1:n,1:n), -1);
end % fine della function gauss2
```

```
function [x, k] = jacobi(A, b, x, maxit, tol)
%JACOBI - Metodo iterativo di Jacobi per sistemi lineari
n = size(A,1);
if ( issparse(A) )
  d = spdiags(A, 0);
else
  d = diag( A );
end
if ( any( abs(d) < eps*norm(d, inf) ) )
  error('Elementi diagonali troppo piccoli.');
end
b = b ./ d;
k = 0; stop = 0;
while ( ~stop )
   k = k + 1;
   xtemp = x;
   x = x - (A*x)./d + b; % istruzione vettoriale
   stop = ( norm(xtemp - x, inf) < tol*norm(x, inf) ) ...
          || ( k == maxit );
end
if ( k == maxit )
   warning('Raggiunto il numero massimo di iterazioni maxit = %d', ...
           maxit);
end
end % fine della M-function jacobi.m
```

```
function [p, coeff] = polyLagrange(x, y, punti)
% POLYLAGRANGE - Polinomio interpolante nella forma di Lagrange
% INPUT
%   x      (double array) - vettore dei nodi o punti di osservazione
%   y      (double array) - vettore delle osservazioni
%   punti  (double array) - vettore dei punti in cui calcolare il polinomio
%                           di Lagrange
% OUTPUT
%   p      (double array) - valore del polinomio nei punti
%   coeff  (double array) - coefficienti della base di Lagrange
%
  n1 = length(y); coeff = zeros(size(x)); p = zeros(size(punti));
  for k = 1 : n1
      coeff(k) = y(k) / prod( x(k) - x([1:k-1, k+1:n1]) );
  end
  for j = 1 : length(punti)
      ij = find( punti(j) == x );
      if isempty(ij)
          % calcolo del polinomio di Lagrange
          p(j) = prod( punti(j) - x ) * ...
                 sum( coeff ./ (punti(j) - x) );
      else
          p(j) = y( ij(1) );
      end
  end
end
```

```
function [C] = mySplineCompleta(x, y, z0, zf)
% MYSPLINECOMPLETA - Coefficienti della spline cubica completa interpolante
% INPUT
```

```
%  x  (double array) - vettore dei nodi di interpolazione
%  y  (double array) - vettore dei valori della funzione nei nodi
%  z0 (double)       - valori della derivata prima nel nodo iniziale
%  zf (double)       - valori della derivata prima nel nodo finale
% OUPUT
%  C  (double array) - matrice con tante righe quanti sono i
%                      sottointervalli e 4 colonne, contenenti i
%                      coeff. dei polinomi di grado 3, da quello
%                      della potenza 3 a quello della potenza 0
%
%  C(i,1)*(x-x_i)^3 + C(i,2)*(x-x_i)^2 + C(i,3)*(x-x_i) + C(i,4)
%
%  Rappresentazione 'pp' (piecewise polynomial)
%
  x = x(:); y = y(:);
  m   = length(x); mm1 = m - 1; mm2 = m - 2;  % m = num. totale nodi
  h   = diff(x); % m-1 valori di h
  d0  = 2 * ( h(1:mm2) + h(2:mm1) ); % diagonale principale
  d1  = [0; h(1:(m-3))]; % prima sopradiagonale della matrice
  dm1 = [h(3:(mm1)); 0]; % prima sottodiagonale della matrice
  T   = spdiags([dm1 d0 d1], [-1 0 1], mm2, mm2); % ordine m-2
  r   = ( ( y(2:mm1) - y(1:mm2) ) ./ h(1:mm2) ) .* h(2:mm1) + ...
        ( ( y(3:m)   - y(2:mm1) ) ./ h(2:mm1) ) .* h(1:mm2);
  r   = 3 * r;
  r(1)   = r(1)   - z0 * h(2);
  r(mm2) = r(mm2) - zf * h(mm2);
  z = T \ r; % risoluzione di un sistema tridiagonale
  z = [z0; z; zf];
  C = zeros(mm1, 4);
  C(:,4) = y(1:mm1); % coeff. del termine costante
  C(:,3) = z(1:mm1); % coeff. di x - x_i
  C(:,1) = ...       % coeff. di (x - x_i)^3
           ( z(2:m) + z(1:mm1) ) ./ ( h(1:mm1).^2 ) ...
           - 2*( y(2:m) - y(1:mm1) ) ./ ( h(1:mm1).^3 );
  C(:,2) = ...       % coeff. di (x - x_i)^2
           ( ( y(2:m) - y(1:mm1) ) ./ h(1:mm1) ...
             - z(1:mm1) ...
           ) ./ h(1:mm1) - C(:,1) .* h(1:mm1);
end
```

```
function [yy] = valSpline(C, x, xx)
% VALSPLINE - Valutazione di una spline cubica interpolante
%  Calcola il valore yy in xx della spline cubica interpolante sui
%  nodi x, con coeff. nelle colonne di C, usando lo schema di Horner
% INPUT
%  C  (double array) - matrice dei coefficienti della spline cubica
%                      sulla partizione x, nella forma 'pp' (in
%                      colonna 1 il coeff. della potenza 3).
%  x  (double array) - vettore dei nodi di interpolazione (deve essere gia'
%                      ORDINATO in ordine crescente)
%  xx (double array) - vettore dei punti in cui calcolare la spline
% OUTPUT
%  yy (double array) - vettore dei valori della spline nei punti xx
%
  x = x(:); xx = xx(:); m = size(C,1); yy = zeros(size(xx));
  for i = 1 : length(xx)
    if ( xx(i) < x(1) || xx(i) > x(end) )
      error(sprintf('xx(%d) esterno all''intervallo',i));
    end
    if ( xx(i) == x(end) )
      k = m;
    else
      k = min( find( x > xx(i) ) ) - 1;
    end
    yy(i) = ( ( C(k,1) * (xx(i) - x(k)) + C(k,2) ...
```

```
            ) * (xx(i) - x(k)) + C(k,3) ...
          ) * (xx(i) - x(k)) + C(k,4); % oppure yy(i) = ...
  end
end
```

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

$$\text{IEEE}_{32}(\alpha) = 1\,\underbrace{10001011}_{\widetilde{p}}\,\underbrace{10110111101011101000000}_{\widetilde{m}}$$

$$\begin{aligned}
s = 1 &\Rightarrow \alpha < 0 \\
\widetilde{p} &= (10001011)_2 = (+139)_{10} \\
&\Rightarrow p = \widetilde{p} - bias = 139 - 127 = 12 \\
&\Rightarrow 2^p = 2^{12} \\
\widetilde{m} &= (10110111101011101000000)_2 \\
&\Rightarrow m = (1.10110111101011101000000)_2 \\
|\alpha| &= m \cdot 2^p = (1.10110111101011101000000)_2 \cdot 2^{12} \\
&= 2^{12}\left(1 + 2^{-1} + 2^{-3} + 2^{-4} + 2^{-6} + 2^{-7} + 2^{-8} + 2^{-9} + 2^{-11} + 2^{-13} + 2^{-14} + 2^{-15} + 2^{-17}\right) \\
&= 2^{12} + 2^{11} + 2^9 + 2^8 + 2^6 + 2^5 + 2^4 + 2^3 + 2^1 + 2^{-1} + 2^{-2} + 2^{-3} + 2^{-5} \\
&= 4096 + 2048 + 512 + 256 + 64 + 32 + 16 + 8 + 2 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{32} \\
&= 7034 + 0.5 + 0.25 + 0.125 + 0.03125 = 7034.90625
\end{aligned}$$

Dunque $\alpha = (-7034.90625)_{10}$.

Matlab

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

```
% Cognome Nome
% Matricola
%----------------------------
%  esercizio 1 - 19/07/2023
%----------------------------
close all; clear all; clc;
disp('Esecizio 1');

p6 = log( 2.^([4 5 6 8 9 11 12]') );

x0 = input('Inserire il punto nel quale valutare il polinomio (double): x0 = ');
[r,    q0] = ruffiniHorner( p6, x0 );
[derp, q1] = ruffiniHorner( q0, x0 );
[ders, q2] = ruffiniHorner( q1, x0 );
fprintf('\nValore del polinomio in x0: p(%g) = %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);
fh = fplot(@(x)(polyval(p6,x)), [-1.4, 0.5]);
```

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

Punto di valutazione: $x_0 = 0.57$
Valori calcolati in x_0:
$p_6(x_0) \approx 16.4602$
$p_6'(x_0) \approx 26.0521$
$p_6''(x_0) \approx 69.2726$

I coefficienti del polinomio risultano:

$$\texttt{p6} \approx \begin{pmatrix} 2.7726 \\ 3.4657 \\ 4.1589 \\ 5.5452 \\ 6.2383 \\ 7.6246 \\ 8.3178 \end{pmatrix}$$

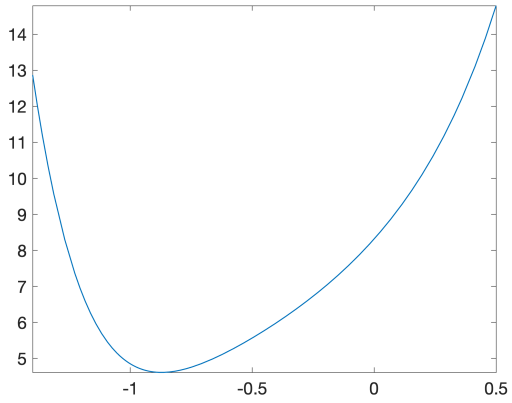


Figura 1: grafico del polinomio $p_6(x)$ dell'esercizio 1 nell'intervallo $[-1.00, 0.93]$.

Soluzione esercizio 2

Teoria

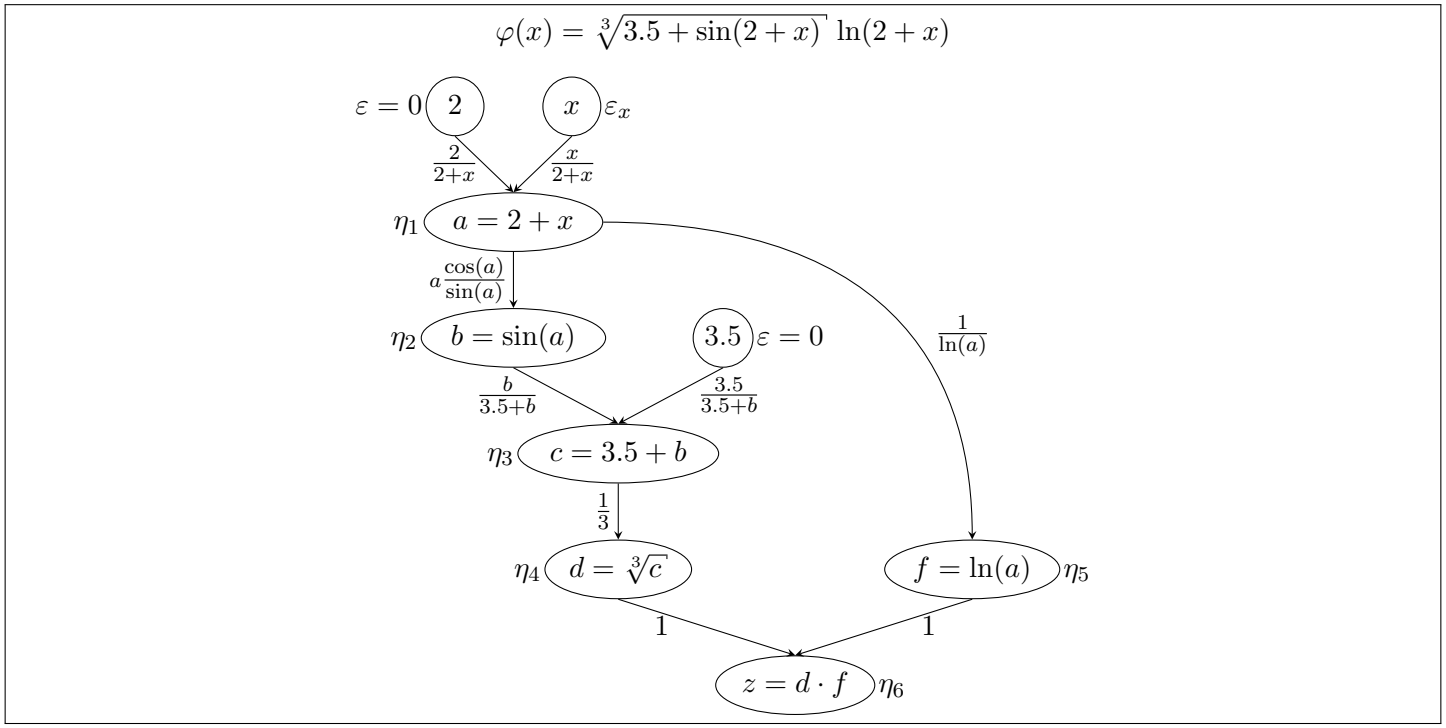


Figura 2: grafo della formulazione della funzione $\varphi(x)$ dell'esercizio 2.

Notiamo innanzitutto che la funzione $\varphi(x)$ è definita $\forall x \in \Omega =]-2, +\infty[$. Con riferimento al grafo in figura 2, l'errore inerente è dato da:

$$\begin{aligned}
\epsilon_{\text{dati}} &= \frac{x}{2+x}\left(\frac{a\cos(a)}{\sin(a)} \cdot \frac{b}{3.5+b} \cdot \frac{1}{3} + \frac{1}{\ln(a)}\right)\varepsilon_x \\
&= \frac{x}{2+x}\left(\frac{(2+x)\cos(2+x)}{\cancel{\sin(2+x)}} \cdot \frac{\cancel{\sin(2+x)}}{3.5+\sin(2+x)} \cdot \frac{1}{3} + \frac{1}{\ln(2+x)}\right)\varepsilon_x \\
&= \left(\frac{\cos(2+x)}{3.5+\sin(2+x)} \cdot \frac{1}{3} + \frac{1}{(2+x)\ln(2+x)}\right) x\varepsilon_x \approx \theta_1 \varepsilon_x
\end{aligned}$$

L'indice di condizionamento è:

$$I_{\text{cond}} = |\theta_1| = \left|\frac{\cos(2+x)}{3.5+\sin(2+x)} \cdot \frac{1}{3} + \frac{1}{(2+x)\ln(2+x)}\right| |x| \tag{1}$$

Il primo addendo dentro il modulo del primo fattore rimane limitato $\forall x \in \Omega$. Per $x \to +\infty$, il denominatore del secondo addendo diverge, quindi il secondo addendo converge a zero: dunque, $I_{\text{cond}} \to +\infty$ per $x \to +\infty$, per la presenza del secondo fattore $|x|$. Vediamo poi che il denominatore del secondo addendo dentro il modulo per $x \to -2^+$ porta ad una forma indeterminata $0 \cdot (-\infty)$. Tuttavia, con il cambio di variabile $2 + x = y$ si ha

$$\lim_{x \to -2^+} (2+x)\ln(2+x) = \lim_{y \to 0^+} y \ln(y) = \lim_{y \to 0^+} \frac{\ln(y)}{1/y} \overset{\text{H}}{=} \lim_{y \to 0^+} \frac{1/y}{-1/y^2} = \lim_{y \to 0^+} \frac{-y^2}{y} = 0^- .$$

Perciò, per $x \to -2^+$ il primo addendo tende a $-2/10.5$, mentre il secondo addendo diverge a $+\infty$. Inoltre, vediamo che per $x \to -1^{\mp}$ il primo addendo rimane limitato, mentre il secondo addendo diverge a $\pm\infty$. Pertanto

$$I_{\text{cond}} \xrightarrow[x \to -2^+, -1, +\infty]{} +\infty$$

ossia il problema è mal condizionato in prossimità di -2 e -1 e per x molto grande.
L'errore algoritmico è dato da:

$$\begin{aligned}
\epsilon_{\text{alg}} &= \frac{a\cos(a)}{\sin(a)} \cdot \frac{b}{3.5+b} \cdot \frac{1}{3} \cdot \eta_1 + \frac{b}{3.5+b} \cdot \frac{1}{3} \cdot \eta_2 + \frac{1}{3} \cdot \eta_3 + \eta_4 + \frac{1}{\ln(a)} \cdot \eta_1 + \eta_5 + \eta_6 \\
&= \left(\frac{(2+x)\cos(2+x)}{\cancel{\sin(2+x)}} \cdot \frac{\cancel{\sin(2+x)}}{3.5+\sin(2+x)} \cdot \frac{1}{3} + \frac{1}{\ln(2+x)} \right) \eta_1 + \frac{\sin(2+x)}{3.5+\sin(2+x)} \cdot \frac{1}{3} \cdot \eta_2 + \frac{1}{3} \cdot \eta_3 + \eta_4 + \eta_5 + \eta_6
\end{aligned}$$

e l'indice algoritmico è dunque

$$I_{\text{alg}} = \left| \frac{(2+x)\cos(2+x)}{10.5 + 3\sin(2+x)} + \frac{1}{\ln(2+x)} \right| + \frac{\left|\sin(2+x)\right|}{10.5 + 3\sin(2+x)} + \frac{10}{3} \tag{2}$$

Il primo addendo di (2) diverge (in modulo) per $x \to +\infty$, per $x \to -2^+$ e per $x \to -1$, mentre il secondo addendo rimane sempre limitato. Dunque, si ha instabilità della formulazione di $\varphi(x)$ per gli stessi x per i quali si ha mal condizionamento.

Matlab

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

```
% Cognome Nome
% Matricola
function [z] = esercizio2( x, a, c )
% ESERCIZIO2 - Esercizio 2 prova scritta di Calcolo Numerico del 19/07/2023
%   Valuta la funzione
%    f_a(t) = c(1)*a^((n-1)*t) + c(2)*a^(n-2)*t) + ... + c(n-1)*a^(t) + c(n)
%   nel punto x, dove 'a' e' un parametro scalare e i coefficienti
%   c(i) sono assegnati, i = 1,...,n.
% SYNOPSIS
%   [z] = esercizio2( x, a, c )
% INPUT
%   x (double)       - Punto nel quale valutare la funzione f_a(t)
%   a (double)       - Base delle funzioni esponenziali, a > 0
%   c (double array) - Vettore dei coefficienti della combinazione lineare
%                      di funzioni esponenziali.
%                      [Non espressamente richiesto, ma molto opportuno]
% OUTPUT
%   z (doube array)  - Vettore colonna di tre elementi, tali che:
%                      z(1) = valutazione di f_a(x) con un ciclo for
%                      z(2) = valutazione di f_a(x) con polyval
%                      z(3) = valutazione di f_a(x) con operazioni vettoriali

% Controlli sull'input
if ( isempty(x) || ~isnumeric(x) || ~isscalar(x) )
    error('Il parametro di input ''x'' deve essere uno scalare non vuoto');
end
if ( isempty(a) || ~isnumeric(a) || ~isscalar(a) || (a <= 0) )
    error('Il parametro di input ''a'' deve essere uno scalare positivo non vuoto');
end
```

```
if ( isempty(c) || any( ~isnumeric(c) ) )
    error('Il parametro di input ''c'' deve essere un vettore non vuoto di numeri');
end

% Esecuzione dei calcoli
n = numel(c); c = c(:);
z = zeros(3,1);
z(1) = c(end);
for k = n-1 : -1 : 1
    z(1) = z(1) + c(k) * a^((n-k)*x);
end
y = a^x;
z(2) = polyval(c, y);
z(3) = sum( c(1:(end-1)) .* (a.^([n-1:-1:1]'*x)) ) + c(end);
end
```

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

```
% Cognome Nome
% Matricola
%----------------------------
%  esercizio 2 - 19/07/2023
%----------------------------
close all; clear all; clc;
disp('Test esercizio 2');

coeffs = [0.13, -4.26, -2.91, 3.55, 9.14]';

a = input('Valore scalare positivo per la base dell''esponenziale: a = ');
x = input('Valore scalare per la variabile: x = ');

format long e % attiva il formato di visualizzazione di massima accuratezza
y = esercizio2( x, a, coeffs )
format % ripristina il formato di visualizzazione standard

fprintf('Differenze relative percentuali: \n');
fprintf('\n | y(2) - y(1) | / | y(1) |  = %g %%', ...
            abs( diff(y(1:2)) / y(1) )*100   );
fprintf('\n | y(3) - y(1) | / | y(1) |  = %g %%', ...
            abs( diff(y([1,3])) / y(1) )*100 );
fprintf('\n | y(3) - y(2) | / | y(2) |  = %g %%\n\n', ...
            abs( diff(y(2:3)) / y(2) )*100   );
```

Eseguendo l'M-script e inserendo come parametri $a = 1.6$ e $x = 0.5$, come richiesto, si ottiene il seguente output:

```
Test esercizio 2
Inserire un valore scalare positivo per la base dell'esponenziale: a = 1.3
Inserire un valore scalare per la variabile: x = 0.7

y =
     2.084317527288490e+00
     2.084317527288488e+00
     2.084317527288491e+00

Differenze relative percentuali:
 | y(2) - y(1) | / | y(1) |  = 1.06531e-13 %
 | y(3) - y(1) | / | y(1) |  = 2.13062e-14 %
 | y(3) - y(2) | / | y(2) |  = 1.27837e-13 %
```

Nota. Nel sorgente della M-function il terzo parametro opzionale contenente i coefficienti del polinomio esponenziale non è espressamente richiesto dal testo dell'esercizio, ma è **molto opportuno** per rendere la M-function di utilizzo generale. In tal modo, infatti, il codice può funzionare **senza alcuna modifica** per **qualunque** combinazione lineare di esponenziali, invece che per il solo caso dell'esercizio in questione.

Parte opzionale. Le modifiche da apportare al codice della M-funzion riguardano essenzialmente un controllo, il setup iniziale e i calcoli, come si vede dal contenuto dell'M-function file `esercizio2opz.m`:

```
% Cognome Nome
% Matricola
function [Z] = esercizio2opz( x, a, c )
% ESERCIZIO2OPZ - Esercizio 2 prova scritta di Calcolo Numerico del 19/07/2023
%   Valuta la funzione
%    f_a(t) = c(1)*a^((n-1)*t) + c(2)*a^(n-2)*t) + ... + c(n-1)*a^(t) + c(n)
%   nel punto x, dove 'a' e' un parametro scalare e i coefficienti
%   c(i) sono assegnati, i = 1,...,n.
% SYNOPSIS
%   [z] = esercizio2( x, a, c )
% INPUT
%   x (double array) - Vettore di N punti nel quale valutare la funzione f_a(t)
%   a (double)       - Base delle funzioni esponenziali, a > 0
%   c (double array) - Vettore dei coefficienti della combinazione lineare
%                      di funzioni esponenziali.
%                      [Non espressamente richiesto, ma molto opportuno]
% OUTPUT
%   Z (doube array)  - Matrice 3 x N, tale che:
%                      Z(1, :) = valutazione di f_a(x) con un ciclo for
%                      Z(2, :) = valutazione di f_a(x) con polyval
%                      Z(3, :) = valutazione di f_a(x) con operazioni vettoriali

% Controlli sull'input
if ( isempty(x) || ~isnumeric(x) )
    error('Il parametro di input ''x'' deve essere uno scalare non vuoto');
end
if ( isempty(a) || ~isnumeric(a) || ~isscalar(a) || (a <= 0) )
    error('Il parametro di input ''a'' deve essere uno scalare positivo non vuoto');
end
if ( isempty(c) || any( ~isnumeric(c) ) )
    error('Il parametro di input ''c'' deve essere un vettore non vuoto di numeri');
end

% Esecuzione dei calcoli
n = numel(c); c = c(:);
N = numel(x); x = x(:)';
Z = zeros(3,N);
Z(1, :) = c(end);
for k = n-1 : -1 : 1
    Z(1, :) = Z(1, :) + c(k) * a.^((n-k)*x);
end
y = a.^x;
Z(2, :) = polyval(c, y);
Z(3, :) = sum( c(1:(end-1)) .* (a.^([n-1:-1:1]'*x)) ) + c(end);
end
```

Per una generalizzazione diretta ad un vettore $\boldsymbol{x}$ di N componenti occorre infatti prevedere che la M-function restituisca una matrice di tre righe ed N colonne, una per ciascuna componente x_j di $\boldsymbol{x}$. Inoltre, nei calcoli occorre assicurarsi che il vettore $\boldsymbol{x}$ sia un vettore riga, non colonna, in modo che, post-moltiplicato per il vettore colonna $(n-1, n-2, \ldots, 2, 1)^T$, dia una diade di dimensioni $(n-1) \times N$: la j-esima colonna di tale matrice contiene i valori degli esponenti non nulli delle esponenziali, ossia $(n-1)x_j$, $(n-2)x_j$, $\ldots$, $2x_j$, x_j. A questo punto, per ottenere i corrispondenti addendi del polinomio esponenziale per tutte le N componenti del vettore $\boldsymbol{x}$, occorre utilizzare nuovamente la sintassi vettoriale per il calcolo dell'esponenziale con base il parametro a: usiamo infatti l'intera diade all'esponente, usando l'esponenziazione vettoriale "`.^`" di Matlab. Questa operazione dà ancora una matrice di dimensioni $(n-1) \times N$, in cui gli elementi della j-esima colonna sono le esponenziali di cui dobbiamo fare la combinazione lineare, valutate nelle componente x_j di $\boldsymbol{x}$, $j = 1, \ldots, N$. Il passo successivo è creare, per ciascuna componente x_j di $\boldsymbol{x}$, tutti gli $n-1$ addendi non costanti del polinomio esponenziale: ancora una volta, questo si ottiene vettorialmente utilizzando il prodotto componente-per-componente di Matlab con l'operatore "`.*`" fra i primi $n-1$ coefficienti del vettore colonna `c` dei coefficienti del polinomio (escludendo quindi l'ultima componente di `c`, che contiene il termine costante del polinomio) e la matrice $(n-1) \times N$ delle esponenziali appena creata. Questa operazione costruisce la matrice $(n-1) \times N$ nella quale la j-esima colonna contiene tutti

gli $(n-1)$ addendi non costanti del polinomio esponenziale valutato nella j-esima componente x_j di $\boldsymbol{x}$. L'ultimo passo è dunque sommare lungo le colonne di questa matrice, ottenendo così un vettore riga. Aggiungendo a questo vettore riga l'ultima componente del vettore `c`, ossia il termine noto del polinomio, otteniamo esattamente quanto richiesto: ogni componente j-esima di questo vettore riga contiene il valore richiesto della combinazione lineare di esponenziali, valutate nella componente x_j del vettore $\boldsymbol{x}$.

In quanto sopra descritto, è lecito ed efficiente costruire la diade **senza** considerare il valore 0 nel vettore degli interi, perchè questo dà origine ad un'ultima riga completamente nulla nella diade, che nel passo successivo risulta in un'ultima riga formata da soli 1 nei valori delle esponenziali, che quindi poi origina nella matrice finale un'intera ultima riga di valori tutti uguali al termine noto del polinomio (ossia tutti uguali all'ultima componente del vettore `c` dei coefficienti). Pertanto, tutte queste operazioni sono inutili e possono essere evitate posticipando la somma del termine noto del polinomio alla fine, al solo vettore riga che si ha dopo la somma per colonne.

Come si vede, usando **opportunamente** la sintassi vettoriale di Matlab, **tutte** queste operazioni sono correttamente implementabili **in un'unica riga di codice**.

Per provare anche la parte opzionale e constatare che funziona correttamente, si possono aggiungere all'M-script file di test `testEsercizio2.m` poche righe di codice, usando ad esempio un vettore $\boldsymbol{x}$ generato in modo casuale:

```
% Test della parte opzionale
% x = input('Vettore per la variabile: x = ');
rng(5);
xx = [x; rand(4,1)];

format long e % attiva il formato di visualizzazione di massima accuratezza
Y = esercizio2opz( xx, a, coeffs )
format % ripristina il formato di visualizzazione standard

fprintf('Differenze relative percentuali MASSIME: \n');
fprintf('\n max( | Y(2,:) - Y(1,:) | / | Y(1,:) | ) = %g %%', ...
            max( abs( diff(Y(1:2, :)) ./ Y(1, :) )'*100 ) );
fprintf('\n max( | Y(3,:) - Y(1,:) | / | Y(1,:) | ) = %g %%', ...
            max( abs( diff(Y([1,3], :)) / Y(1, :) )'*100 ) );
fprintf('\n max( | Y(3,:) - Y(2,:) | / | Y(2,:) | ) = %g %%\n\n', ...
            max( abs( diff(Y(2:3, :)) / Y(2, :) )'*100 ) );
```

L'esecuzione di questa seconda parte del codice di test fornisce i seguenti risultati:

```
Y =

  Columns 1 through 4

    2.084317527288490e+00     4.724165799923271e+00     8.761152523374690e-01     4.793233388232859e+00
    2.084317527288488e+00     4.724165799923271e+00     8.761152523374705e-01     4.793233388232860e+00
    2.084317527288491e+00     4.724165799923272e+00     8.761152523374687e-01     4.793233388232860e+00

  Column 5

    5.076400290595988e-01
    5.076400290596048e-01
    5.076400290596013e-01

Differenze relative percentuali MASSIME:

 max( | Y(2,:) - Y(1,:) | / | Y(1,:) | ) = 1.181e-12 %
 max( | Y(3,:) - Y(1,:) | / | Y(1,:) | ) = 2.03837e-14 %
 max( | Y(3,:) - Y(2,:) | / | Y(2,:) | ) = 1.26126e-14 %
```

In questo caso, si è inserita come prima componente del vettore $\boldsymbol{x}$ dei punti lo stesso valore 0.7 richiesto per la prova della versione non vettoriale della M-function, in modo da poter osservare dalla prima colonna della matrice Y di output che in questo punto si ottengono **esattamente** gli stessi risultati del caso precedente.

La valutazione delle differenze percentuali, trattandosi di N punti diversi x_j, è stata fatta considerando il caso peggiore. Le differenze relative percentuali osservabili manifestano la diversa propagazione degli errori dell'aritmetica finita nei tre diversi modi di eseguire il calcolo. Tuttavia, le differenze rimangono tutte limitate a valori molto piccoli, al massimo dell'ordine di 10^{-12}, quindi del tutto accettabili.

Soluzione esercizio 3

Teoria

N.B. In ciò che segue, i numeri in trasparenza di ciascuna colonna rappresentano i moltiplicatori di quella stessa colonna, memorizzati nelle posizioni degli zeri sottodiagonali del fattore R finale. Questi valori **non entrano** nei calcoli successivi, ma vengono permutati insieme alle righe. Al termine, tutti i valori in trasparenza rappresentano la parte triangolare inferiore della matrice L della fattorizzazione di A.

Nel seguito indicheremo con $\widehat{A}$ la matrice completa $\left[A^{(k)}|\boldsymbol{b}^{(k)}\right]$ della generica iterazione k-esima, omettendo in essa l'indice d'iterazione k per semplicità di notazione. Per il sistema dato si ha:

$$\left[A^{(1)}|\boldsymbol{b}^{(1)}\right] = \left(\begin{array}{ccc|c} 1/2 & 3/2 & -1/2 & -1 \\ 1 & 2 & 2 & -2 \\ 4 & 7 & -1 & 0 \end{array}\right), \quad \max_{i,j=1,2,3}\left|a_{i,j}^{(1)}\right| = 7\,,\ i_* = 3,\ j_* = 2 \ \Rightarrow\ P_1 = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix},\ Q_1 = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$P_1\left[A^{(1)}Q_1|\boldsymbol{b}^{(1)}\right] = \left(\begin{array}{ccc|c} 7 & 4 & -1 & 0 \\ 2 & 1 & 2 & -2 \\ 3/2 & 1/2 & -1/2 & -1 \end{array}\right), \quad \boldsymbol{m}^{(1)} = \begin{pmatrix} 0 \\ 2/7 \\ 3/14 \end{pmatrix}, \quad L_1 = I_3 - \boldsymbol{m}^{(1)}\mathbf{e}_1^T = \begin{pmatrix} 1 & 0 & 0 \\ -2/7 & 1 & 0 \\ -3/14 & 0 & 1 \end{pmatrix}$$

$$\xrightarrow[\widehat{A}_{3,*}\leftarrow\widehat{A}_{3,*}-(3/14)\widehat{A}_{1,*}]{\widehat{A}_{2,*}\leftarrow\widehat{A}_{2,*}-(2/7)\widehat{A}_{1,*}} \left[A^{(2)}|\boldsymbol{b}^{(2)}\right] = L_1P_1\left[A^{(1)}Q_1|\boldsymbol{b}^{(1)}\right] = \left(\begin{array}{ccc|c} 7 & 4 & -1 & 0 \\ 2/7 & -1/7 & 16/7 & -2 \\ 3/14 & -5/14 & -2/7 & -1 \end{array}\right),$$

$$\max_{i,j=2,3}\left|a_{i,j}^{(2)}\right| = \frac{16}{7}\,,\ i_* = 2,\ j_* = 3 \quad \Rightarrow \quad P_2 = I_3\,, \quad Q_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix},$$

$$P_2\left[A^{(2)}Q_2|\boldsymbol{b}^{(2)}\right] = \left(\begin{array}{ccc|c} 7 & -1 & 4 & 0 \\ 2/7 & 16/7 & -1/7 & -2 \\ 3/14 & -2/7 & -5/14 & -1 \end{array}\right) \quad \boldsymbol{m}^{(2)} = \begin{pmatrix} 0 \\ 0 \\ -1/8 \end{pmatrix}, \quad L_2 = I_3 - \boldsymbol{m}^{(2)}\mathbf{e}_2^T = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1/8 & 1 \end{pmatrix}$$

$$\xrightarrow{\widehat{A}_{3,*}\leftarrow\widehat{A}_{3,*}+(1/8)\widehat{A}_{2,*}} \left[A^{(3)}|\boldsymbol{b}^{(3)}\right] = L_2P_2\left[A^{(2)}Q_2|\boldsymbol{b}^{(2)}\right] = \left(\begin{array}{ccc|c} 7 & -1 & 4 & 0 \\ 2/7 & 16/7 & -1/7 & -2 \\ 3/14 & -1/8 & -3/8 & -5/4 \end{array}\right) = [R|\boldsymbol{c}]$$

$$\boldsymbol{c} = L^{-1}P\boldsymbol{b} = \begin{pmatrix} 0 \\ -2 \\ -5/4 \end{pmatrix}, \qquad S_2 = P_2 = I_3 \ \Rightarrow\ \widetilde{L}_1^{-1} = S_2L_1^{-1}S_2^T = L_1^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 2/7 & 1 & 0 \\ 3/14 & 0 & 1 \end{pmatrix}$$

$$P = S_1 = P_2P_1 = P_1 = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}, \quad Q = Q_1Q_2 = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

$$L = \widetilde{L}_1^{-1}L_2^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 2/7 & 1 & 0 \\ 3/14 & -1/8 & 1 \end{pmatrix}, \quad R = \begin{pmatrix} 7 & -1 & 4 \\ 0 & 16/7 & -1/7 \\ 0 & 0 & -3/8 \end{pmatrix}$$

$$\left.\begin{aligned} x_3 &= \frac{c_3}{r_{3,3}} = \frac{-5/4}{-3/8} = \frac{10}{3} \\ x_2 &= \frac{c_2 - r_{2,3}x_3}{r_{2,2}} = \frac{-2-(-1/7)10/3}{16/7} = -\frac{2}{3} \\ x_1 &= \frac{c_1 - r_{1,2}x_2 - r_{1,3}x_3}{r_{1,1}} = \frac{0-(-1)(-2/3)-4(10/3)}{7} = -2 \end{aligned}\right\} \ \Rightarrow\ \boldsymbol{x} = \begin{pmatrix} -2 \\ -2/3 \\ 10/3 \end{pmatrix} \ \Rightarrow\ \boldsymbol{x}^* = Q\boldsymbol{x} = \begin{pmatrix} 10/3 \\ -2 \\ -2/3 \end{pmatrix}$$

$$\det(A) = \det(P)\det(Q)\det(R) = (-1)\cdot 1\cdot \cancel{7}\cdot \frac{\cancel{16}_2}{\cancel{7}}\cdot\frac{-3}{\cancel{8}} = 6\,.$$

Nota. Gli elementi sottodiagonali della matrice finale L sono esattamente gli elementi riportati in trasparenza nella parte strettamente triangolare inferiore nella matrice $A^{(3)}$: l'utilità della strategia di memorizzare i moltiplicatori al posto degli elementi resi nulli delle trasformazioni di Gauss è esattamente questa, cioà fornire direttamente la parte strettamente sottodiagonale della matrice triangolare inferiore finale L.

Per l'ultimo punto, chiamiamo F la sottomatrice di ordine 2 di A formata dall'intersezione delle prime due righe e delle ultime due colonne di A: con la rotazione elementare di Givens dobbiamo annullare l'elemento di posizione $(2,1)$ di F (che coincide con l'elemento in posizione $(2,2)$ di A), ossia $f_{2,1} = a_{2,2} = 2$, utilizzando per la rotazione l'elemento in posizione $(1,1)$ di F (che coincide con l'elemento in posizione $(1,2)$ di A), ossia $f_{1,1} = a_{1,2} = 3/2$.

Usando le formule numericamente più stabili per il calcolo della matrice di rotazione di Givens si ha:

$$F = \begin{pmatrix} 3/2 & -1/2 \\ 2 & 2 \end{pmatrix}, \qquad 3/2 = |f_{1,1}| < |f_{2,1}| = 2 \;\Rightarrow\; t = \frac{f_{1,1}}{f_{2,1}} = \frac{3}{4}$$

$$\Rightarrow\; s = \frac{1}{\sqrt{1+t^2}} = \frac{1}{\sqrt{1+(3/4)^2}} = \frac{4}{5}, \quad c = ts = \frac{3}{\cancel{4}} \cdot \frac{\cancel{4}}{5} = \frac{3}{5}$$

$$G_{1,2} = \frac{1}{5}\begin{pmatrix} 3 & 4 \\ -4 & 3 \end{pmatrix} \;\Rightarrow\; R_2 = G_{1,2}A_2 = \frac{1}{5}\begin{pmatrix} 3 & 4 \\ -4 & 3 \end{pmatrix}\begin{pmatrix} 3/2 & -1/2 \\ 2 & 2 \end{pmatrix} = \begin{pmatrix} 5/2 & 13/10 \\ 0 & 8/5 \end{pmatrix}$$

$$\det(F) = \det(R_2) = \frac{\cancel{5}}{\cancel{2}} \cdot \frac{\cancel{8}}{\cancel{5}} = 4\,.$$

Matlab

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

```
% Cognome Nome
% Matricola
%-----------------------------
%  esercizio 3 - 19/07/2023
%-----------------------------
close all; clear all; clc;
disp('Esercizio 3');

A = [0.5 1.5 -0.5; 1 2 2; 4 7 -1]; b = [-1 -2 0]';
xTeoria = [10/3 -2 -2/3]';

% prima parte
[L, R, p, detA] = gauss2(A);
x1 = utrisol( R, ltrisol(L, b(p)) )
% oppure:  y1 = ltrisol(L, b(p)); x1 = utrisol(R, 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) );
x2 = A \ b

% seconda parte
B = A * A';
disp('Fattorizzazione di Cholesky:');
[Lchol, p] = chol(B, 'lower') % Lchol e' triangolare inferiore
if ( ~p )
    fprintf('\nB e'' definita positiva\n');
else
    error('\nB non e'' definita positiva\n');
end
c  = [3/5, -2/3, -1/7]';
x3 = utrisol( Lchol', ltrisol(Lchol, c) )
```

Eseguendo lo script si ottiene il seguente output:

```
Esercizio 3

x1 =
    3.3333
   -2.0000
   -0.6667

Massima differenza in modulo dalla soluzione teorica:
   1.1102e-16

Residuo normalizzato (in norma infinito):
   1.0e-15 *

   -0.0555
         0
   -0.3331

x2 =
    3.3333
   -2.0000
   -0.6667

Fattorizzazione di Cholesky:
Lchol =
    1.6583         0         0
    1.5076    2.5937         0
    7.8393    1.6123    1.3950

p =
     0

B e' definita positiva
x3 =
    5.1426
    0.5308
   -1.1438
```

Soluzione esercizio 4

Teoria

La matrice A è **non tridiagonale** e **non simmetrica**. Per la matrice A, i dischi di Gershgorin per righe sono (figura 3):

$$\mathcal{H}_1 = \left\{ x \in \mathbb{C} \,\middle|\, |x-2| \le 1/2 \right\}, \qquad \mathcal{H}_2 = \left\{ x \in \mathbb{C} \,\middle|\, |x-4/3| \le 5/4 \right\}, \qquad \mathcal{H}_3 = \left\{ x \in \mathbb{C} \,\middle|\, |x-1| \le 2/3 \right\},$$

mentre i dischi per colonne sono

$$\mathcal{K}_1 = \left\{ x \in \mathbb{C} \,\middle|\, |x-2| \le 2/3 \right\}, \qquad \mathcal{K}_2 = \left\{ x \in \mathbb{C} \,\middle|\, |x-4/3| \le 0 \right\}, \qquad \mathcal{K}_3 = \left\{ x \in \mathbb{C} \,\middle|\, |x-1| \le 7/4 \right\}.$$

Pertanto (si veda la figura 3):

$$\mathcal{U}_r = \bigcup_{j=1}^{3} \mathcal{H}_j = \mathcal{H}_2\,, \quad \mathcal{U}_c = \bigcup_{j=1}^{3} \mathcal{K}_j = \mathcal{K}_3 \quad \Rightarrow \quad \mathcal{S} = \mathcal{U}_r \cap \mathcal{U}_c = \mathcal{H}_2 \cap \mathcal{K}_3 = \mathcal{H}_2$$

dove l'ultima uguaglianza segue dal fatto che $\mathcal{H}_2 \subset \mathcal{K}_3$.

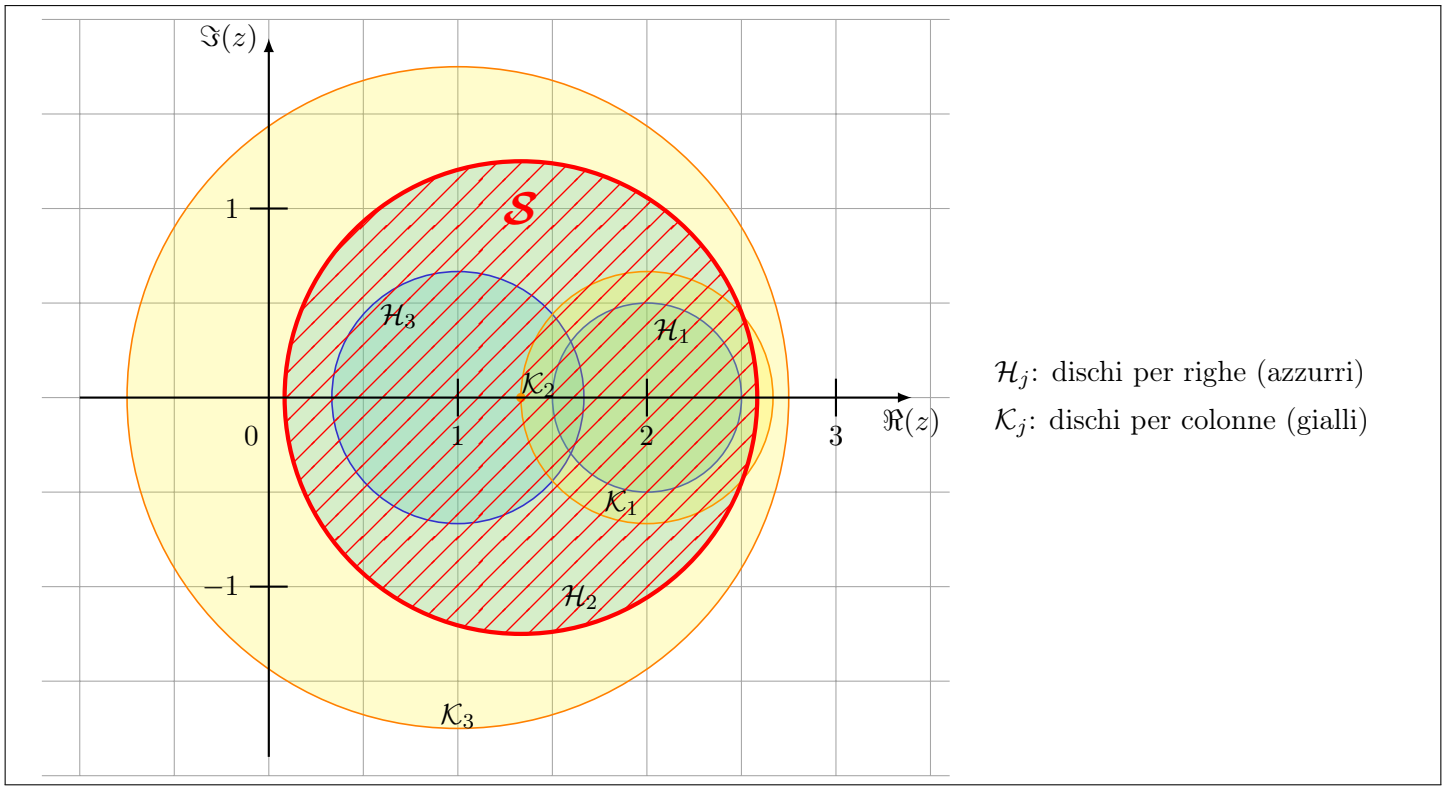
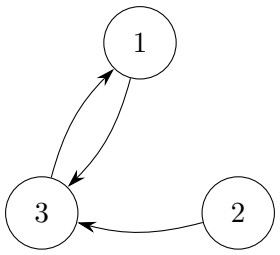


Figura 3: dischi di Ghershgorin per righe e per colonne della matrice A dell'Esercizio 4. La regione tratteggiata $\mathcal{S}$ contiene *tutti* gli autovalori di A. Si noti che il disco $\mathcal{K}_2$ ha raggio zero, quindi si riduce ad un punto, esattamente il centro $(4/3,\, 0)$ del disco stesso.

Si vede immediatamente che A è una matrice **strettamente diagonale dominante per righe**, perché $|a_{i,i}| > \sum_{j \neq i} |a_{i,j}|$ $\forall i = 1, 2, 3$, ma **per colonne non è nemmeno diagonale dominante**: infatti, nella terza colonna, $1/2 + 5/4 = 7/4 > 1$.

Pertanto, essendo verificata la condizione di stretta diagonale dominanza almeno per righe, i metodi Jacobi e di Gauss-Seidel 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 $\rho(\mathcal{G}) < \rho(J)$). Inoltre, la matrice A **è invertibile**, in quanto strettamente diagonale dominante per righe (e dunque certamente 0 **non** appartiene alla regione $\mathcal{S}$).

La matrice è non tridiagonale, quindi per determinare la riducibilità o meno di A costruiamo il grafo della matrice:



Si vede immediatamente che il grafo di A **non è** strettamente connesso, in quanto partendo dal nodo 1 o dal nodo 3 non si arriva al nodo 2. Pertanto, la matrice A è **riducibile**.

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

Inoltre, essendo A **non tridiagonale**, non possiamo utilizzare i risultati che legano la convergenza del metodo di Jacobi al metodo SOR, nè conosciamo un'espressione per il parametro ottima di rilassamento ω^* di SOR.

Pertanto, senza eseguire ulteriori calcoli, **non** è possibile dedurre *a priori* quale dei due fra il metodo di Jacobi e il metodo di Gauss-Seidel converga più velocemente.

Determiniamo la matrice J di iterazione di Jacobi:

$$J = D^{-1}(L+U) = I_3 - D^{-1}A = \begin{pmatrix} 0 & 0 & -1/4 \\ 0 & 0 & 15/16 \\ -2/3 & 0 & 0 \end{pmatrix} \approx \begin{pmatrix} 0 & 0 & -0.25 \\ 0 & 0 & 0.9375 \\ -0.6667 & 0 & 0 \end{pmatrix}.$$

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

Determiniamo gli autovalori della matrice J:

$$\det\big(J - \lambda I_3\big) = \begin{vmatrix} -\lambda & 0 & -1/4 \\ 0 & -\lambda & 15/16 \\ -2/3 & 0 & -\lambda \end{vmatrix} = -\lambda^3 - \left(\frac{-2}{3}\cdot\frac{-1}{4}\right)(-\lambda) = -\lambda\left(\lambda^2 - \frac{1}{6}\right)$$

$$\lambda_1(J) = 0 \qquad \lambda^2 - \frac{1}{6} = 0 \;\Rightarrow\; \lambda_{2,3}(J) = \mp\frac{1}{\sqrt{6}} \quad\Rightarrow\quad \rho(J) = \max_{i=1,\dots,3}\big|\lambda_i(J)\big| = \frac{1}{\sqrt{6}} \approx 0.4082\,.$$

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:

$$T = \big[D - L \,\big|\, I_3\big] = \left(\begin{array}{ccc|ccc} 2 & 0 & 0 & 1 & 0 & 0 \\ 0 & 4/3 & 0 & 0 & 1 & 0 \\ 2/3 & 0 & 1 & 0 & 0 & 1 \end{array}\right) \xrightarrow{T_{3,*}\leftarrow T_{3,*}-(1/3)T_{1,*}} \left(\begin{array}{ccc|ccc} 2 & 0 & 0 & 1 & 0 & 0 \\ 0 & 4/3 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & -1/3 & 0 & 1 \end{array}\right) \to$$

$$\to \left(\begin{array}{ccc|ccc} 1 & 0 & 0 & 1/2 & 0 & 0 \\ 0 & 1 & 0 & 0 & 3/4 & 0 \\ 0 & 0 & 1 & -1/3 & 0 & 1 \end{array}\right) = \big[I_3 \,\big|\, (D-L)^{-1}\big]\,.$$

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

$$\mathcal{G} = (D-L)^{-1}U = \begin{pmatrix} 1/2 & 0 & 0 \\ 0 & 3/4 & 0 \\ -1/3 & 0 & 1 \end{pmatrix}\begin{pmatrix} 0 & 0 & -1/2 \\ 0 & 0 & 5/4 \\ 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & -1/4 \\ 0 & 0 & 15/16 \\ 0 & 0 & 1/6 \end{pmatrix}.$$

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

$$\det\big(\mathcal{G} - \lambda I_4\big) = \begin{vmatrix} -\lambda & 0 & -1/4 \\ 0 & -\lambda & 15/16 \\ 0 & 0 & (1/6) - \lambda \end{vmatrix} = \lambda^2\left(\frac{1}{6} - \lambda\right) \;\Rightarrow\; \lambda_{1,2}(\mathcal{G}) = 0,\; \lambda_3(\mathcal{G}) = \frac{1}{6} \;\Rightarrow\; \rho(\mathcal{G}) = \max_{i=1,\dots,3}\big|\lambda_i(\mathcal{G})\big| = \frac{1}{6}\,.$$

Il fatto che $\rho(\mathcal{G}) = 1/6 \approx 0.1667 < 1$ conferma che anche il metodo di Gauss-Seidel è convergente. Inoltre, osserviamo che $\rho(\mathcal{G}) = \rho(J)^2$: se ne deduce che il metodo di Gauss-Seidel converge più velocemente del metodo di Jacobi e precisamente con velocità asintotica *doppia* rispetto al metodo di Jacobi:

$$R_\infty(\mathcal{G}) = -\ln\big(\rho(\mathcal{G})\big) = -\ln\big(\rho^2(J)\big) = -2\ln\big(\rho(J)\big) = 2R_\infty(J) \qquad R_\infty(\mathcal{G}) \approx 1.7918 \qquad R_\infty(J) \approx 0.8958\,.$$

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

$$\mathcal{G}(\omega) = (D-\omega L)^{-1}\big((1-\omega)D+\omega U\big) = \begin{pmatrix} 2 & 0 & 0 \\ & 4/3 & 0 \\ (2/3)\omega & 0 & 1 \end{pmatrix}^{-1} \begin{pmatrix} 2(1-\omega) & 0 & -\omega/2 \\ 0 & 4(1-\omega)/3 & 5\omega/4 \\ 0 & 0 & 1-\omega \end{pmatrix}.$$

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

Matlab

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

```
% Cognome Nome
% Matricola
%----------------------------
%  esercizio 4 - 19/07/2023
%----------------------------
close all; clear all; clc
disp('Esercizio 4');

A = [2 0 1/2; 0 4/3 -5/4; 2/3 0 1];
b = [1/3, -2, -1]'; d = diag(A);
invD    = diag( 1 ./ d );
J       = -invD * (tril(A,-1) + triu(A,1));
GS      = -tril(A) \ triu(A,1);
eigsJ   = eig( J )
eigsGS  = eig( GS )
rhoJ    = max(abs( eigsJ ));  RinfJ  = -log( rhoJ );
rhoGS   = max(abs( eigsGS )); RinfGS = -log( rhoGS );
fprintf('\nrho(J) = %e,  rho(GS) = %e', rhoJ, rhoGS);
fprintf('\nRinf(J) = %f,  Rinf(GS) = %f\n', RinfJ, RinfGS);
% Soluzione con il metodo di Jacobi
tol = 1.0e-5; maxit = 50; x0 = ones(size(b));
[xJ, iterJ] = jacobi(A, b, x0, maxit, tol);
fprintf('\nNumero di iterazioni di Jacobi: iterJ = %d', iterJ);
fprintf('\nSoluzione calcolata con il metodo di Jacobi:');
xJ
% Soluzione con il metodo di SOR
omega = 0.82;
SOR = (diag(d) + omega*tril(A,-1)) ...
      \ [( (1-omega)*diag(d) - omega*triu(A,1) ), omega*b]
cSOR = SOR(:,end); SOR(:,end) = []; xSOR = x0;
for k = 1:6, xSOR = SOR*xSOR + cSOR; end
xSOR
```

Eseguendo lo script, il metodo di Jacobi si arresta dopo 15 iterazioni con un'approssimazione della soluzione data da $\boldsymbol{x}_{\mathrm{J}} \approx (0.5000, -2.7500, -1.3333)^T$, che è l'approssimazione a quattro cifre decimali della soluzione $\boldsymbol{x}^* = \big(1/2, -11/4, -4/3\big)^T$ del sistema $A\boldsymbol{x} = \boldsymbol{b}$ (questa può facilmente essere calcolata da linea di comando con `xs = A\b` e impostando il formato di visualizzazione a numeri razionali mediante `format rat`).

D'altro canto, l'approssimazione raggiunta dal metodo SOR con $\omega = 0.82$ dopo 6 iterazioni partendo dallo stesso punto iniziale $\boldsymbol{x}^{(0)} = (1,1,1)^T$ è $\boldsymbol{x}^{(6)}_{\mathrm{SOR}} \approx (0.4951, -2.7316, -1.3284)^T$. Si osserva che le componenti di quest'approssimazione si stanno avvicinando a quelle della soluzione $\boldsymbol{x}^*$ e la loro accuratezza è già abbastanza buona: l'errore relativo percentuale in norma euclidea per questa approssimazione è infatti $\big\|\boldsymbol{x}^{(6)}_{\mathrm{SOR}} - \boldsymbol{x}^*\big\|_2 / \big\|\boldsymbol{x}^*\big\|_2 \approx 0.64\%$.

Soluzione esercizio 5

Teoria

Notiamo innanzitutto che la funzione è ben definita nell'intervallo $[a,b] = [0.2, 5] = [1/5, 5]$ perchè l'argomento del logaritmo è sempre positivo. Dato che si sta cercando un polinomio interpolante di grado al più $n = 3$, i punti

di interpolazione devono essere $n+1=4$. Costruiamo il polinomio interpolante per la funzione $f(x)$ sull'intero intervallo $[0.2,\ 5]$. Siano t_j i nodi di Chebychev in $[-1,1]$ e w_j i nodi di Chebychev in $[0.2,5]$, $j=0,1,2,3$, ottenuti usando la trasformazione che mappa $[-1,1]$ in $[a,b]$:

$$t_{3-j}=\cos\left(\frac{2j+1}{2(n+1)}\pi\right)\quad\forall j=3,2,1,0$$

$$\Rightarrow\quad t_0=\cos(7\pi/8)\approx-0.9239\,,\ t_1=\cos(5\pi/8)\approx-0.3827\,,\ t_2=\cos(3\pi/8)=-\cos(5\pi/8)\approx0.3827\,,$$
$$t_3=\cos(\pi/8)=-\cos(7\pi/8)\approx0.9239\,,$$

$$w_j=\frac{a+b}{2}+\frac{b-a}{2}t_j=2.6+2.4t_j=\frac{13}{5}+\frac{12}{5}t_j\quad\forall j=0,1,2,3\quad\Rightarrow\quad\boldsymbol{w}\approx(0.3827,\ 1.6816,\ 3.5184,\ 4.8173)^T\,.$$

Calcoliamo ora per questi quattro punti di interpolazione la relativa tabella delle differenze divise:

w_k	$f(w_k)$	$f[w_k,w_{k+1}]$	$f[w_0,w_1,w_2]$	$f[w_0,w_1,w_2,w_3]$
0.3827	3.6963			
1.6816	1.3035	−1.8422		
3.5184	1.4693	0.0902	0.6163	
4.8173	0.2625	−0.9291	−0.3251	−0.2123

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

$$\begin{aligned}p_3^{\mathrm{C}}(x)&\approx3.6963+(-1.8422)(x-w_0)+0.6163(x-w_0)(x-w_1)+(-0.2123)(x-w_0)(x-w_1)(x-w_2)\\&=3.6963-1.8422(x-0.3827)-0.6163(x-0.3827)(x-1.6816)-0.2123(x-0.3827)(x-1.6816)(x-3.5184)\,.\end{aligned}$$

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

$$p_3^{\mathrm{C}}(x)\approx-0.2123x^3+1.8013x^2-4.7926x+5.2785\,.$$

Nota per lo studente. La tabella delle differenze divise e i coefficienti del polinomio $p_3^{\mathrm{C}}(x)$ 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 $x_0,\ldots,x_k$. 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 $\omega_{n-1}(x)=(x-x_0)(x-x_1)\cdots(x-x_{n-1})$ 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 $p_3^{\mathrm{C}}(x)$ 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:

$$f'(x) = 2\cos(2x) - \frac{1}{x}\,, \quad f''(x) = -4\sin(2x) + \frac{1}{x^2}\,, \quad f'''(x) = -8\cos(2x) - \frac{2}{x^3}\,, \quad f^{(4)}(x) = 16\sin(2x) + \frac{6}{x^4}$$

$$\Rightarrow \quad R_3(x) = (x-0.3827)(x-1.6816)(x-3.5184)(x-4.8173)\frac{1}{4!}\left(16\sin(2\xi_x) + \frac{6}{\xi_x^4}\right) \quad \text{con } \xi_x \in]0.2,\, 5[\,.$$

I valori in $x^* = 3.9$ della funzione $f(x)$, di $p_3^{\mathrm{C}}(x)$ e della loro differenza sono:

$$f\big(x^*\big) = f(3.9) \approx 1.6805\,,\ p_3^{\mathrm{C}}\big(x^*\big) = p_3^{\mathrm{C}}(3.9) \approx 1.3934 \ \Rightarrow\ f\big(x^*\big) - p_3^{\mathrm{C}}\big(x^*\big) = 1.6805 - 1.3934 \approx 0.2872\,.$$

Il valore dell'espressione dell'errore valutata prendendo $\xi_x = x = x^*$ è ≈ 1.8210

N.B. <u>**Non**</u> possiamo scrivere $R_3\big(x^*\big) \approx 1.8210$ perchè, com'è ben noto, il punto ξ_{x^*} nell'argomento della derivata **è non noto** e, pur dipendendo da x^*, **non è lecito supporre** che sia $\xi_x^* = x^*$.

Infine, gli $n+1 = 3$ nodi equispaziati di interpolazione in $[a, b]$ e i corrispondenti valori di f e sono:

$$h = \frac{b-a}{n} = \frac{5 - 1/5}{2} = 2.4\,, \quad x_j = a + jh \quad \forall j = 0, 1, 2$$

$$\Rightarrow \quad \boldsymbol{x} = \big(a,\, (a+b)/2,\, b\big)^T = \big(1/5,\, 13/5,\, 5\big)^T = \big(0.2,\, 2.6,\, 5.0\big)^T\,,$$

$$\Rightarrow \quad f(\boldsymbol{x}) \approx (4.0418,\, 0.2040,\, -0.1105)^T$$

I tre polinomi di Lagrange per questi punti sono:

$$L_0(x) = \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} = \frac{(x-13/5)(x-5)}{(1/5-13/5)(1/5-5)} = \frac{x^2 - (38/5)x + 13}{288/25} \approx 0.0868x^2 - 0.6597x + 1.1285\,,$$

$$L_1(x) = \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_0-x_2)} = \frac{(x-1/5)(x-5)}{(13/5-1/5)(13/5-5)} = \frac{x^2 - (26/5)x + 1}{-144/25} \approx -0.1736x^2 + 0.9028x - 0.1736\,,$$

$$L_2(x) = \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)} = \frac{(x-1/5)(x-13/5)}{(5-1/5)(5-13/5)} = \frac{x^2 - (14/5)x + (13/25)}{288/25} \approx 0.0868x^2 - 0.2431x + 0.0451\,.$$

La formula che fornisce il polinomio interpolante $p_2(x)$ sui tre punti equistaziati mediante i polinomi di Lagrange è:

$$\begin{aligned} p_2(x) &= \sum_{i=0}^{2} f(x_i)L_i(x) \\ &\approx 4.0418\Big(0.0868x^2 - 0.6597x + 1.1285\Big) + 0.2040\Big(-0.1736x^2 + 0.9028x - 0.1736\Big) \\ &\qquad - 0.1105\Big(0.0868x^2 - 0.2431x + 0.0451\Big) \\ &\approx 0.3509x^2 - 2.6665x + 4.5611 - 0.0354x^2 + 0.1842x - 0.0354 - 0.0096x^2 + 0.0269x - 0.0050 \\ &\approx 0.3058x^2 - 2.4555x + 4.5207 \end{aligned}$$

Matlab

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

```
% Cognome Nome
% Matricola
%----------------------------
%  esercizio 5 - 19/07/2023
%----------------------------
close all; clear all; clc;
disp('Esercizio 5');

f = @(x)( pi - log(3*x) + sin(2*x) )
f1 = @(x)( 2*cos(2*x) - (1 ./ x) )
a = 0.2; b = 5; nNodi = 6; n = nNodi - 1; N = 201;
t = cos( (2*[nNodi-1:-1:0]' + 1)*pi ./ (2*nNodi) ); % nodi di Chebychev in [-1,1]
x = 0.5*( (b - a)*t + (a+b) ); % nodi di Chabychev in [a,b]
y = f(x);
xx = linspace(a, b, N)';
```

```
ff = f(xx);
[yy, d1] = polyLagrange(x, y, xx);
errRelPerc = abs( yy - ff ) ./ abs ( ff ) * 100;

figure(1);
ah1 = ...
plot([xx(1); xx(end)],[0; 0],'k-', ... % asse delle ascisse
      x,zeros(size(x)),'bo', ...       % nodi di interpolazione di Chebychev
      x,y,'b*' );                      % valori della funzione nei nodi di Chebychev
hold on;
ph(1:2) = plot( xx,ff,'k-', ...        % grafico della funzione
                xx,yy,'b-' );          % interpolante con nodi di Chebychev
hold off;
th(1) = title(sprintf('Funzione e polinomio interpolante di grado %d', n));
xh(1) = xlabel('Intervallo di interpolazione');
yh(1) = ylabel('Valori di f(x) e p_n(x)');

figure(2);
ah2 = ...
plot([xx(1); xx(end)],[0; 0],'k-', ... % asse delle ascisse
      x,zeros(size(x)),'bo' );         % nodi di interpolazione
hold on;
ph(3) = plot( xx, errRelPerc,'b-');    % errore relativo percentuale in [a,b]
hold off;

th(2) = title(sprintf('Errore relativo percentuale con polinomio di grado %d', n));
xh(2) = xlabel('Intervallo di interpolazione');
yh(2) = ylabel('Errore relaivo percentuale');

%
% seconda parte
%
xEq = linspace(a,b,nNodi); yEq = f(xEq);
z0 = f1(a); zf = f1(b);
[C] = mySplineCompleta(xEq, yEq, z0, zf);
[ww] = valSpline(C, xEq, xx);
figure(1);
hold on;
ph(4) = plot(xx,ww,'r-'); % spline naturale su nodi equidistanti
hold off;
lh1 = legend(ph([1:2,4])','Funzione','Interp. Cheb.','Spline nat.');

figure(2);
hold on;
ph(5) = plot(xx, 100*abs(ff-ww)./abs(ff), 'r-'); % err. rel. perc. con la spline
hold off;
lh2 = legend(ph([3,5])','Interp. Cheb.','Spline nat.');
```

L'esecuzione dell'M-script genera i grafici in fig. 4 (che qui, per esigenze di stampa, sono stati leggermente migliorati).

Soluzione esercizio 6

Teoria

Notiamo innanzitutto che la funzione è ben definita nell'intervallo $[a, b] = [4.5, 5.5] = [9/2, 11/2]$ perchè l'argomento del logaritmo è sempre positivo. Procediamo alle verifica dell'esistenza di almeno uno zero di $f(x)$ in $[a, b]$, verificando se i valori che $f(x)$ assume agli estremi dell'intervallo sono discordi: $f(a) = f(9/2) = \pi - \ln(27/2) + \sin(9) \approx 0.9510 > 0$, $f(b) = f(11/2) = \pi - \ln(33/2) + \sin(11) \approx -0.6618 < 0$, il che consente di affermare che la funzione possiede almeno uno zero in $[a, b]$ (per il Teorema degli zeri).

Per verificare le ipotesi del teorema di convergenza globale del metodo di Newton in tale intervallo, occorre calcolare le derivate prima e seconda (già calcolate al punto precedente) e valutarne il segno. Si ha:

$$f'(x) = -\frac{1}{x} + 2\cos(2x)\,, \qquad f''(x) = \frac{1}{x^2} - 4\sin(2x)\,.$$

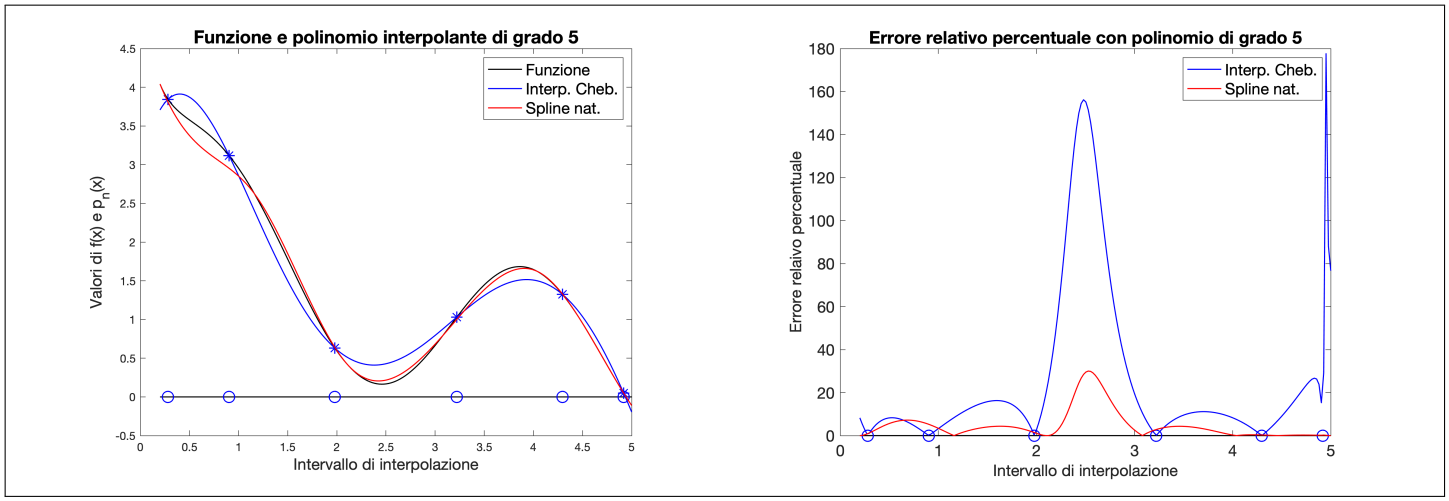


Figura 4: grafici generati dall'M-script `esercizio5.m` (figure leggermente migliorate per esigenze di stampa).

Notiamo innanzitutto che per $x \in [9/2,\, 11/2]$ si ha $2x \in [9,\, 11] \subset \big[(5/2)\pi,\, (7/2)\pi\big] \approx [7.85,\, 11.00]$ dove il coseno è negativo. Inoltre $-1/x < -5/11 \approx -0.4545 < 0$ in $[a, b]$. Concludiamo quindi che $f'(x) < 0\ \forall x \in [a, b]$, quindi certamente il metodo di Newton è ben definito perchè la derivata prima non si annulla. Essendo essa sempre negativa, deduciamo che la funzione $f(x)$ è monotona strettamente decrescente in $[a, b]$.

Per quanto riguarda il segno della derivata seconda, invece, notiamo che la funzione seno è monotona decrescente in $\big[(5/2)\pi,\, (7/2)\pi\big]$ e ivi cambia segno: poiché $1/x^2$ è sempre positiva, la prima verifica che possiamo fare è valutare $f''(x)$ agli estremi dell'intervallo. Si ha $f''(a) = f''(9/2) = -4\sin(9) + 4/81 \approx -1.5991 < 0$, mentre $f''(b) = f''(11/2) = -4\sin(11) + 4/121 \approx 4.0330 > 0$: dunque, $f''(x)$ ha segno **non** costante in $[a, b]$ e pertanto **non** sono ivi verificate le ipotesi del Teorema di convergenza globale del metodo di Newton (che, lo ricordiamo, sono condizioni solo sufficienti) e non possiamo affermare, ma nemmeno escludere, che esso sia convergente.

I grafici di $f(x)$, $f'(x)$ ed $f''(x)$ sono riportati in fig. 5 e confermano quanto già stabilito analiticamente.

Per quanto riguarda il metodo di bisezione, il numero minimo di iterazioni da compiere per avere un'accuratezza di almeno 10^{-5} nell'approssimare lo zero della funzione è

$$k = \left\lceil \log_2\left((b-a)/10^{-5}\right)\right\rceil = \left\lceil \log_2\left((5.5-5.4)\cdot 10^5\right)\right\rceil \approx \lceil 16.6096\rceil = 17\,.$$

Calcoliamo ora due iterazioni dei vari metodi:

- metodo di Newton con $x_0 = 5.3$:

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} = 5.3 - \frac{\pi - \ln(3\cdot 5.3) + \sin(2\cdot 5.3)}{2\cos(2\cdot 5.3) - 1/5.3} \approx 5.3 - \frac{-0.5475}{-0.9594} \approx 4.7293$$

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)} \approx 4.7293 - \frac{\pi - \ln(3\cdot 4.7293) + \sin(2\cdot 4.7293)}{2\cos(2\cdot 4.7293) - 1/4.7293} \approx 4.7293 - \frac{0.4554}{-2.2103} \approx 4.9353$$

- metodo delle secanti con $w_0 = a = 4.5$ e $w_1 = b = 5.5$:

$$w_2 = w_1 - \frac{w_1 - w_0}{f(w_1) - f(w_0)} f(w_1)$$

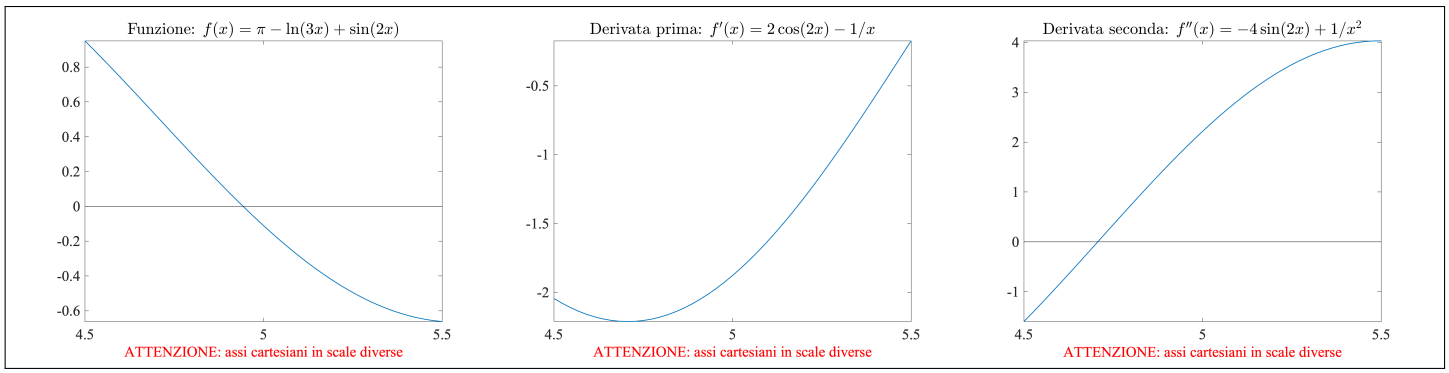


Figura 5: grafici della funzione $f(x)$ dell'esercizio 6 e delle sue derivate prima e seconda nell'intervallo $[a, b] = [4.5,\, 5.5]$.

$$= 5.5 - \frac{5.5 - 4.5}{\pi - \ln(3 \cdot 5.5) + \sin(2 \cdot 5.5) - \big(\pi - \ln(3 \cdot 4.5) + \sin(2 \cdot 4.5)\big)} \big(\pi - \ln(3 \cdot 5.5) + \sin(2 \cdot 5.5)\big)$$

$$\approx 5.5 - \frac{1}{-0.6618 - 0.9510} \cdot (-0.6618) \approx 5.0897$$

$$w_3 = w_2 - \frac{w_2 - w_1}{f(w_2) - f(w_1)} f(w_2)$$

$$\approx 5.0897 - \frac{(5.0897 - 5.5) \cdot \big(\pi - \ln(3 \cdot 5.0897) + \sin(2 \cdot 5.0897)\big)}{\pi - \ln(3 \cdot 5.0897) + \sin(2 \cdot 5.0897) - \big(\pi - \ln(3 \cdot 5.5) + \sin(2 \cdot 5.5)\big)}$$

$$\approx 5.0897 - \frac{-0.4103}{-0.2692 - (-0.6618)} \approx 4.8083$$

- metodo di falsa posizione con $a_0 = a = 4.5$ e $b_0 = b = 5.5$:

$$z_0 = b_0 - f(b_0)(b_0 - a_0)/\big(f(b_0) - f(a_0)\big) \approx 5.5 - (-0.6618)\frac{5.5 - 4.5}{-0.6618 - 0.9510} \approx 5.0897\,,$$

$$f(z_0) = \pi - \ln(3 \cdot 5.0897) + \sin(2 \cdot 5.0897) \approx -0.2692 < 0 \quad \Rightarrow \quad a_1 = a_0 = 4.5\,, \quad b_1 = x_0 = 5.0897$$

$$z_1 = b_1 - f(b_1)(b_1 - a_1)/\big(f(b_1) - f(a_1)\big) \approx 5.0897 - (-0.2692)\frac{5.0897 - 4.5}{-0.2692 - 0.9510} \approx 4.9596$$

Dalla parte Matlab del presente esercizio, utilizzando la funzione predefinita `fzero` di Matlab, si ottiene per lo zero x^* di $f(x)$ in $[4.5, 5.5]$ la stima numerica $x^* \approx 4.9430$. Possiamo quindi stimare l'accuratezza dell'approssimazione ottenuta dai tre metodi dopo le due iterazioni di ciascuno appena calcolate. Gli errori relativi percentuali nei tre casi sono, rispettivamente:

$$\text{metodo di Newton:} \qquad \mathrm{err}_2^{(\mathrm{N})} = \frac{|x_2 - x^*|}{|x^*|} \approx \frac{|4.9353 - 4.9430|}{4.9430} \approx 0.15\%\,,$$

$$\text{metodo delle secanti:} \qquad \mathrm{err}_2^{(\mathrm{S})} = \frac{|w_3 - x^*|}{|x^*|} \approx \frac{|4.8083 - 4.9430|}{4.9430} \approx 2.73\%\,,$$

$$\text{metodo di falsa posizione:} \qquad \mathrm{err}_2^{(\mathrm{FP})} = \frac{|z_1 - x^*|}{|x^*|} \approx \frac{|4.9596 - 4.9430|}{4.9430} \approx 0.34\%\,.$$

Si nota che, in questo caso, il metodo di Newton è il più accurato dei tre dopo le due iterazioni iniziali. Comunque, anche il metodo di falsa posizione raggiunge un'accuratezza paragonali a quella del metodo di Newton.

Infine, le condizioni sufficienti per l'esistenza e l'unicità di un punto fisso di una generica funzione $g(x)$ in un generico intervallo $[a, b]$ sono le seguenti:

- $g(x) \in C^0\big([a, b]\big)$, ossia che $g(x)$ sia almeno continua in $[a, b]$;
- $g\big([a, b]\big) \subseteq [a, b]$, ossia che $g(x)$ assuma valori in $[a, b]$ per ogni $x \in [a, b]$;
- $g(x)$ sia una contrazione su $[a, b]$, ossia che $\exists L \in [0, 1[$ tale che $\big|g(x_1) - g(x_2)\big| \leq L\big|x_1 - x_2\big|$ $\forall x_1, x_2 \in [a, b]$, $x_1 \neq x_2$.

Se $g(x) \in C^1\big([a, b]\big)$, la terza condizione può essere sostituita da $\big|f'(x)\big| < 1$ $\forall x \in [a, b]$.

Matlab

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

```
function [xs, it] = myNewton(fname, fpname, xs, tolx, tolf, maxit)
% INPUT:  fname  (string o fhandle) - Function della funzione
%         fpname (string o fhandle) - Function della derivata prima
%         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: xs     (double)   - Approssimazione della soluzione
%         it     (integer)  - Numero di iterazioni eseguite

  % soglia di controllo sull'ordine di grandezza della derivata prima
  tolfp = min( tolf, 10*eps );
  % Metodo di Newton
  % N.B.: avendo messo 'xs' come parametro di ingresso corrispondente
  %       all'argomento di input 'x0' (punto iniziale), la sua inizializzazone
  %       e' gia' effettuata e in uscita verra' sovrascritto con l'approssimazione
  %       calcolata dello zero della funzione
  fxs = feval(fname, xs); it = 0;
```

```
  stop = ( abs(fxs) < tolf );
  while ( ~stop )
    it   = it + 1;
    fpx  = feval(fpname, xs);
    if (abs(fpx) < tolfp), error('|f''(xk)| troppo piccolo.'); end
    d    = fxs / fpx;  xs = xs - d;
    fxs   = feval(fname, xs);
    stop = ( (abs(fxs) < tolf) || (abs(d) < tolx*abs(xs)) ...
             || (it == maxit) );
  end
  if ( it >= maxit )
    fprintf('\nRaggiunto il massimo numero di iterazioni.\n');
  end
end
```

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

```
% Cognome Nome
% Matricola
%----------------------------
%  esercizio 6 - 19/07/2023
%----------------------------
close all; clear all; clc;
disp('Esercizio 6');

funzione = @(x)( pi - log(3*x) + sin(2*x) );
derivata = @(x)( 2*cos(2*x) - (1.0 ./ x) );
a = 4.5; b = 5.5; taux = 1.0e-3; tauf = 1.0e-4; maxit = 100;
x0 = 5.3;

[xN, itN] = myNewton(funzione, derivata, x0, taux, tauf, maxit);

myOptions = optimset('TolX',taux, 'TolFun',tauf, 'MaxIter',maxit);
[xfz, fxfz, exitflag, output] = fzero(funzione, x0, myOptions);
if (exitflag ~= 1); warning('zero della funzione non trovato.'); end

fprintf('\nApprossimazioni calcolate:\n');
fprintf('\n\tM-function myNewton:')
fprintf('\n\txs = %g, f(xs) = %g  (iterazioni = %d, x0 = %g)', ...
         xN, funzione(xN), itN, x0);
fprintf('\n\tFunzione predefinita fzero con gli stessi parametri:')
fprintf('\n\txs = %g, f(xs) = %g  (iterazioni = %d, x0 = %g)', ...
         xfz, fxfz, output.iterations, x0);
fprintf('\n\n');
```

Si noti che basta interrogare l'help in linea di Matlab per la funzione `fzero` per scoprire che i parametri di ingresso addizionali τ_x, τ_f e `maxit` che questo esercizio chiede di passare alla funzione predefinita, vanno passati utilizzando la struttura di opzioni restituita dal comando `optimset`. I nomi dei tre campi di tale struttura da utilizzare per i parametri richiesti sono immediatamente individuati proprio nell'help in linea di `optimset` e sono `TolX`, `TolFun` e `MaxIter`, rispettivamente.

Eseguendo lo script si ottiene il seguente output:

```
Approssimazioni calcolate:


  M-function myNewton:
  xs = 4.94296, f(xs) = 5.18992e-05  (iterazioni = 3, x0 = 5.3)
  Funzione predefinita fzero con gli stessi parametri:
  xs = 4.94472, f(xs) = -0.00346741  (iterazioni = 3, x0 = 5.3)
```

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

Tuttavia, in questo caso, il valore assoluto della funzione nel punto ottenuto con il metodo di Newton è migliore (più vicino a zero) di quello nel punto ottenuto da `fzero`.