Corso di Laurea in Informatica – Università di Ferrara
Calcolo Numerico
Prova scritta e pratica dell'8/02/2023 – 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 del proprio cognome (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 a quale numero reale (razionale) α corrisponde la seguente rappresentazione nel formato ANSI standard IEEE a 32 bit: $\text{IEEE}_{32}(\alpha) = 01000111100010011000001110000110$. Mostrare tutti i calcoli della conversione.
2) (M) (2 punti) Siano dati i seguenti coefficienti:

$$c_0 = \min\Big\{\mathrm{e}^{-2},\, \max\big\{\sin(7.2\pi),\, 3.1\big\} + (1/2)\Big\}, \qquad c_1 = \sqrt{\log(7!)}\,, \qquad c_2 = \tan\Big(\pi/3 - \log_2(4.17)\Big),$$
$$c_4 = \frac{55}{100}\Big|\sin(5\pi/8)\sqrt[3]{\mathrm{e}}\Big|, \qquad c_5 = \max\Big\{3.7\cdot 10^{-2},\, \tan(2\pi),\, -\log_{10}(0.314),\, 0.7/\pi^3\Big\},$$
$$c_6 = \cos\Big(3\arcsin\big(-\sqrt{\mathrm{e}/\pi}\big)\Big)$$

del polinomio $p_6(x) = c_6x^6 + \ldots + c_1x + c_0$. Realizzare un M-script file Matlab che, usando la M-function `ruffiniHorner` in allegato, calcoli e stampi il valore del polinomio $p_6(x)$ e delle sue derivate prima $p_6'(x)$ e seconda $p_6''(x)$ in un prefissato punto x_0 accettato in input. Lo script visualizzi anche il grafico del polinomio nell'intervallo $[-0.5, 1.4]$. Provare lo script con il valore $x_0 = 0.7$, riportando nel foglio delle risposte i valori ottenuti.

Esercizio 2

1) (T) (3 punti) Determinare l'insieme $\Omega_\varphi \subseteq \mathbb{R}$ di definizione, l'errore inerente e l'errore algoritmico nel calcolo dell'espressione:

$$\varphi(x) = \frac{5\mathrm{e}^x - 1}{5 + \mathrm{e}^{-x}} \qquad x \in \Omega \subseteq \mathbb{R}$$

Valutare l'indice di condizionamento e l'indice algoritmico. 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) Si consideri la funzione $\varphi_k(x;\boldsymbol{c}) = \Big(c_1\mathrm{e}^{kx/2} - c_2\Big)\Big/\Big(c_3 + c_4\mathrm{e}^{-kx}\Big)$, $x \in \mathbb{R}$ in cui $c_3, c_4 > 10^{-4}$ e il parametro k sia un intero positivo (cioè $k \in \mathbb{N}$). Realizzare una M-function che accetti in input un vettore `x` di ascisse, un vettore `c` contenente le costanti $c_1, \ldots, c_4$ e un parametro intero positivo N. Per tutti i k da 1 a N, la M-function calcoli per tutte le ascisse x_i il valore della funzione $\varphi_k(x;\boldsymbol{c})$. La funzione restituisca poi tutti i valori calcolati nelle ascisse, per tutti i k, mediante un unico parametro Y di uscita, in modo tale che sia semplice individuare i valori calcolati per ciascuno specifico k. Si effettui questo calcolo con **sintassi vettoriale**.
Programmare la M-function in modo tale che possa essere chiamata con un secondo parametro **opzionale** di uscita: in tal caso, e **solo** in tal caso, la M-function ripeta il calcolo dei valori di $\varphi_k(x;\boldsymbol{c})$ per tutti i k e per

tutte le ascisse x_i, ma **senza** utilizzare la sintassi vettoriale. In questo caso, poi, la M-function rilevi anche i tempi di esecuzione dell'intero calcolo nei due distinti modi (vettoriale e non vettoriale), restituendo tali tempi nel parametro opzionale di uscita, come elementi di un vettore colonna. Dotare la M-function di opportuni controlli sui parametri di ingresso e di corrispondenti messaggi per l'utente in caso di errore.
Scrivere infine un M-script di prova della M-function su un campionamento uniforme dell'intervallo $[-0.2, 0.3]$ con 3001 punti. Si provi la M-function due volte: la prima usando i parametri $c_1 = -2$, $c_2 = 5.2$, $c_3 = 0.41$, $c_4 = 3.7$, $N = 6$ e senza temporizzazione, la seconda con gli stessi valori delle ascisse, ma con parametri $c_1 = \mathrm{e}^{-2}$, $c_2 = 0.7$, $c_3 = \pi/3$, $c_4 = 1.4$, $N = 20$ e con temporizzazione del calcolo. Dopo ciascuna chiamata, lo script disegni in una nuova finestra grafica, su un unico piano cartesiano, i grafici delle N curve corrispondenti ai valori di k da 1 a N, ricavando il corrispondente *graphics handle*. Dopo la seconda chiamata, lo script visualizzi a console anche i tempi di esecuzione ed il loro rapporto percentuale.

Esercizio 3

Si consideri la seguente matrice:

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

1) (T) (4 punti) Definire una matrice di rotazione elementare e dire di quali proprietà gode. Determinare la fattorizzazione QR di A mediante trasformazioni di Givens, esplicitando tutti i calcoli ed utilizzando le formule più stabili. Risolvere inoltre il sistema lineare $A\boldsymbol{x} = \boldsymbol{b}$ dove $\boldsymbol{b} = (0, -1/3, 1, 1/4)^T$ (nel caso si usino valori decimali, mantenere almeno 3 cifre significative corrette nei calcoli).
2) (M) (4 punti) Realizzare una M-function Matlab, denominata `miaDiade`, che accetti in ingresso due vettori colonna $\boldsymbol{r}$ e $\boldsymbol{s}$, controllando che siano non vuoti e accertandosi che siano colonne, prima di continuare. La function costruisca poi la diade $\boldsymbol{r}\boldsymbol{s}^T$, ne ricavi le dimensioni e ne determini il rango numerico (tenendo conto della precisione di macchina) e determinante, sfruttando la fattorizzazione QR di Matlab. La function restituisca in output il rango numerico, il determinante e la norma infinito della diade.
Scrivere poi un M-script file che calcoli la fattorizzazione QR della matrice A di cui sopra e risolva il sistema lineare $A\boldsymbol{x} = \boldsymbol{b}$ con termine noto il vettore $\boldsymbol{b}$ di cui sopra, sfruttando la fattorizzazione ottenuta e la funzione `utrisol` in allegato. Controllare i risultati con quelli ottenuti al punto 1. Calcolare inoltre la norma infinito della differenza fra la soluzione del sistema ottenuta con la fattorizzazione QR e la soluzione che si ottiene sfruttando la fattorizzazione LU di Matlab e le M-functions `ltrisol` e `utrisol` in allegato. Lo script visualizzi a console il valore di tale norma con un'istruzione di stampa formattata. Le due soluzioni coincidono? Motivare il risultato, considerando anche la differenza relativa.
Successivamente, lo script provi la M-function `miaDiade` di cui sopra, passandole come vettori di ingresso, rispettivamente, la prima colonna della matrice Q e la prima riga della matrice R della fattorizzazione QR di A precedentemente ottenuta. Quanto risulta il rango della diade? È corretto? Motivare la risposta.

Esercizio 4

Si considerino la seguente matrice e il seguente vettore:

$$M = \begin{pmatrix} 9/4 & 0 & -1/3 & 1/2 \\ 0 & -3/2 & 0 & 3/2 \\ -1/2 & 1/3 & 7/4 & 0 \\ 1/2 & 0 & 0 & -3/2 \end{pmatrix} \qquad \boldsymbol{z} = \begin{pmatrix} z_1 \\ \vdots \\ z_4 \end{pmatrix} \text{ con } z_j = \frac{\cos(j+1)}{j},\ j = 1, \ldots, 4.$$

1) (4 punti) Disegnare i dischi di Gershgorin della matrice M. Senza effettuare il calcolo dei rispettivi raggi spettrali, stabilire se la matrice M gode o no di semplici proprietà che consentano di affermare qualcosa *a priori* riguardo la convergenza o meno dei metodi iterativi di Jacobi e Gauss-Seidel e, nel caso, riguardo la loro velocità asintotica di convergenza, motivando accuratamente le risposte.
Determinare poi le matrici di iterazione J di Jacobi e $\mathcal{G}$ di Gauss-Seidel, esplicitando *tutti* i calcoli. Scrivere infine per la matrice M l'espressione della matrice $\mathcal{G}(\omega)$ del metodo SOR con parametro ω, nella forma di prodotto delle corrispondenti matrici triangolari inferiore e superiore.
Opzionale (2 punti) Esplicitare i polinomi caratteristici delle matrici di iterazione di Jacobi e Gauss-Seidel. Con l'aiuto di Matlab, determinarne i raggi spettrali $\rho(J)$ e $\rho(\mathcal{G})$ e, se possibile, le rispettive velocità asintotiche di convergenza.
2) (M) (4 punti) Realizzare un M-script che calcoli le matrici d'iterazione `J` e `GS` ed i rispettivi vettori costanti `cJ` e `cGS` dei metodi di Jacobi e di Gauss-Seidel, rispettivamente, per la matrice M e il termine noto $\boldsymbol{z}$. Lo

script calcoli poi un'approssimazione `xJ` della soluzione $\boldsymbol{x}^*$ del sistema $M\boldsymbol{x} = \boldsymbol{z}$ implementando le iterazioni del metodo di Jacobi mediante `J` e `cJ` (*senza* chiamare una M-function), partendo del vettore $\boldsymbol{x}^{(0)} = (1, -1, 1, -1)^T$, con numero massimo di iterazioni `maxit` $= 50$ e tolleranza `tol` $= 10^{-5}$. Lo script visualizzi a console `xJ` e il numero `iterJ` di iterazioni compiute.
Inoltre, con gli stessi parametri e con $\omega = 1.03$, lo script calcoli un'approssimazione `xSOR` della soluzione $\boldsymbol{x}^*$ usando la M-function `sor` in allegato, visualizzandola a console insieme al numero `iterSOR` di iterazioni compiute.
Infine, calcolata la soluzione `xs` del sistema $M\boldsymbol{x} = \boldsymbol{z}$ con l'apposito comando Matlab, lo script calcoli le distanze relative in norma infinito di `xJ` e `xSOR` da `xs`, visualizzandole a console.

Esercizio 5

Si consideri la seguente funzione: $f(x) = \cos(0.7\pi + 2x)/\left(\mathrm{e}^{1.3x} - 2\right)$, $x \in [a, b]$ con $[a, b] = [-4.5,\, 0.15]$.

1) (T) (3 punti) Ricavare l'espressione analitica della forma canonica del polinomio $p_3(x)$ di interpolazione di $f(x)$ nei nodi $x_k = (31/20) \cdot k - (9/2)$, $k = 0, \ldots, 3$, mediante il polinomio di Newton. Determinare poi i 4 nodi di interpolazione di Chebychev nello stesso intervallo e i corrispondenti valori della funzione. Infine, scrivere la formula analitica del resto dell'interpolazione per una funzione sufficientemente regolare.
2) (M) (4 punti) Scrivere un M-script file che realizzi un ciclo (temporizzato) sul grado n del polinomio interpolante $f(x)$ negli $n + 1$ nodi equispaziati in $[a, b]$. Per ogni valore di n, il ciclo deve:
   (a) calcolare i valori del polinomio interpolante $p_n(x)$ mediante la M-function `polyLagrange` fornita in allegato;
   (b) visualizzare in un unico grafico, sempre nella stessa finestra grafica (aperta prima del ciclo), sull'intervallo $[a, b]$, la curva $f(x)$ con un tratto continuo blu ed il polinomio interpolante $p_n(x)$ con un tratto continuo rosso, usando un campionamento di $[a, b]$ con almeno 201 punti equidistanti;
   (c) evidenziare in tale grafico l'asse delle ascisse ($y = 0$) con una linea nera, i nodi di interpolazione con cerchietti neri ed i valori del polinomio nei nodi con cerchietti rossi;
   (d) attendere 2 secondi prima di passare all'iterazione successiva.

   Il valore massimo $N \in \mathbb{N}$ del grado del polinomio sia impostato prima del ciclo. L'aggiornamento dei grafici da un'iterazione all'altra deve avvenire senza chiudere e riaprire la finestra grafica. Dotare i grafici di etichette degli assi e di opportuni titoli, che riportino di volta in volta il corrente valore del grado del polinomio visualizzato nel grafico sottostante. Per provare lo script, usare $N = 12$.

Esercizio 6

Si consideri la funzione $f(x) = -x \sin\left(4x - \pi/3\right) + \mathrm{e}^{-x^3}$ con $x \in [a, b] = [-1,\, 1.5]$.

1) (T) (2 punti) Si determini il polinomio algebrico $p_3(x)$ di grado al più $n = 3$ che meglio approssima la funzione $f(x)$ in $[a, b]$ nel senso dei minimi quadrati, utilizzando i suoi valori in 6 punti equispaziati x_j, $j = 0, \ldots, 5$.
   Si esplicitino le matrici V, $B = V^T V$ e il termine noto $\boldsymbol{z} = V^T \boldsymbol{y}$ del sistema delle equazioni normali per il caso in esame, dove si è indicato con $\boldsymbol{y}$ il vettore colonna dei valori di $f(x)$ nei 6 punti equispaziati.
   Si risolva poi il sistema in Matlab con un appropriato metodo di soluzione e si determini il vettore soluzione $\boldsymbol{\alpha}^*$. Si scriva infine l'espressione di $p_n^*(x)$, fornendo gli errori assoluti e relativi che si commettono nei punti equispaziati x_j, $j = 0, \ldots, n$.
2) (M) (3 punti) In un M-script file, siano assegnate le due variabili intere positive N ed n. La prima rappresenta il numero di punti di osservazione rispetto ai quali costruire il polinomio algebrico di migliore approssimazione, la seconda rappresenta il grado di tale polinomio. Dopo aver controllato che $n < N$, lo script generi il vettore colonna `x` di N punti scelti casualmente nell'intervallo $[a, b]$, con distribuzione uniforme e con seme iniziale fissato. Definita come funzione *in-line* la funzione $f(x)$ del punto teorico precedente, lo script assegni al vettore `y` la valutazione della funzione nei punti di `x` e poi costruisca il polinomio $p_n^*(x)$, di grado al più n, di migliore approssimazione nel senso dei minimi quadrati, in tre modi distinti: (i) costruendo esplicitamente la matrice V di Vandermonde e risolvendo il corrispondente sistema delle equazioni normali; (ii) risolvendo direttamente il sistema sovradeterminato $V\boldsymbol{\alpha} = \boldsymbol{y}$; (iii) utilizzando l'apposita funzione di Matlab per l'interpolazione/approssimazione polinomiale.
   Calcolare la norma infinito delle differenze fra le tre soluzioni calcolate. Infine, lo script costruisca, in una stessa finestra grafica, i plot della funzione $f(x)$ e del polinomio $p_n^*(x)$ determinato con la soluzione (ii), nell'intervallo $[a, b]$, tracciando anche l'asse delle ascisse. Dotare il grafico del titolo e di opportune etichette per gli assi.
   Provare lo script impostando a 17 il seme iniziale del generatore di numeri (pseudo)casuali, $N = 15$, $n = 6$ e usando 2001 punti equispaziati per la generazione del grafico.

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

$$\text{IEEE}_{32}(\alpha) = \overset{\overset{32 \text{ bit} = 4 \text{ byte}}{\boxed{1^\circ \text{ byte}}\boxed{2^\circ \text{ byte}}\boxed{3^\circ \text{ byte}}\boxed{4^\circ \text{ byte}}}}{\underbrace{0}_{s}\underbrace{10001111}_{\widetilde{p}}\underbrace{00010011000001110000110}_{\widetilde{m}}}$$

$$\begin{aligned}
s = 0 &\Rightarrow \alpha > 0 \\
\widetilde{p} &= (10001111)_2 = (+143)_{10} \\
&\Rightarrow p = \widetilde{p} - bias = 143 - 127 = 16 \\
&\Rightarrow 2^p = 2^{16} \\
\widetilde{m} &= (00010011000001110000110)_2 \\
&\Rightarrow m = (1.00010011000001110000110)_2 \\
|\alpha| &= m \cdot 2^p = (1.00010011000001110000110)_2 \cdot 2^{16} \\
&= 2^{16}\left(1 + 2^{-4} + 2^{-7} + 2^{-8} + 2^{-14} + 2^{-15} + 2^{-16} + 2^{-21} + 2^{-22}\right) \\
&= 2^{16} + 2^{12} + 2^9 + 2^8 + 2^2 + 2^1 + 2^0 + 2^{-5} + 2^{-6} \\
&= 65536 + 4096 + 512 + 256 + 4 + 2 + 1 + \frac{1}{32} + \frac{1}{64} \\
&= 70407 + 3.125 \cdot 10^{-2} + 1.5625 \cdot 10^{-2} = 70407.046875
\end{aligned}$$

Matlab

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

```
% Cognome Nome
% Matricola
%----------------------------
%  esercizio 1 - 08/02/2023
%----------------------------
close all; clear all; clc;
disp('Esecizio 1');

p6    = zeros(7,1);
p6(1) = cos( 3*asin( -sqrt( exp(1)/pi ) ) );                    % c6
p6(2) = max( [ 3.7E-2; tan(2*pi); -log10(0.314); 0.7/pi^3 ] );  % c5
p6(3) = 0.55 * abs( sin(5*pi/8) * exp(1/3) );                   % c4
p6(5) = tan( pi/3 - log2( 4.17 ) );                             % c2
p6(6) = sqrt( log( factorial(7) ) );                            % c1
p6(7) = min( exp(-2), max( sin(7.2*pi), 3.1 ) + 0.5 );          % 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)), [-0.5,1.4] );
```

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

Punto di valutazione: $x_0 = 0.7$
Valori calcolati in x_0:
$p_6(x_0) \approx \quad 1.5426$
$p_6'(x_0) \approx \quad 1.3424$
$p_6''(x_0) \approx -2.0908$

Valori approssimati dei coefficienti del polinomio:

$$p_6(x) \approx \begin{pmatrix} -0.9034 \\ 0.5031 \\ 0.7092 \\ 0 \\ -1.6023 \\ 2.9198 \\ 0.1353 \end{pmatrix}$$

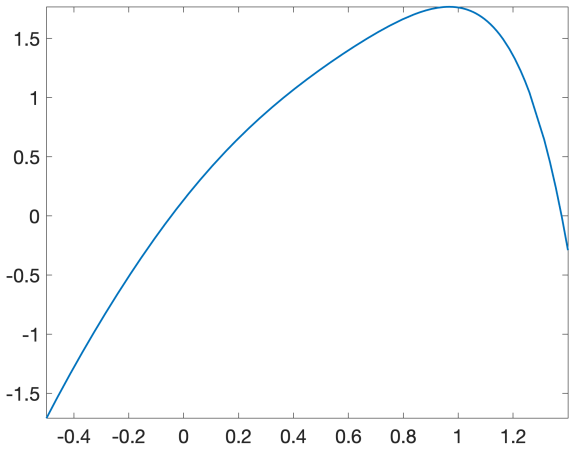


Figura 1: grafico del polinomio $p_6(x)$ dell'esercizio 1 nell'intervallo $[-0.5, 1.4]$.

Nota. Si richiama l'attenzione sul fatto che nei grafici di Matlab sia sempre MOLTO importante tenere presente in quale scala vengono visualizzati gli assi. In particolare, il motore grafico di Matlab è impostato di default nella modalità che prevede di scalare automaticamente gli assi delle ascisse e delle ordinate per far sì che l'immagine del grafico "si adatti" alle dimensioni della finestra grafica, che sono generalmente rettangolari, quasi quadrate.

Nel caso in esame le scale degli assi sono abbastanza simili, quindi il disegno è abbastanza coerente anche con i valori numerici ottenuti per le derivate (i valori della funzione sono sempre coerenti).

Soluzione esercizio 2

Teoria

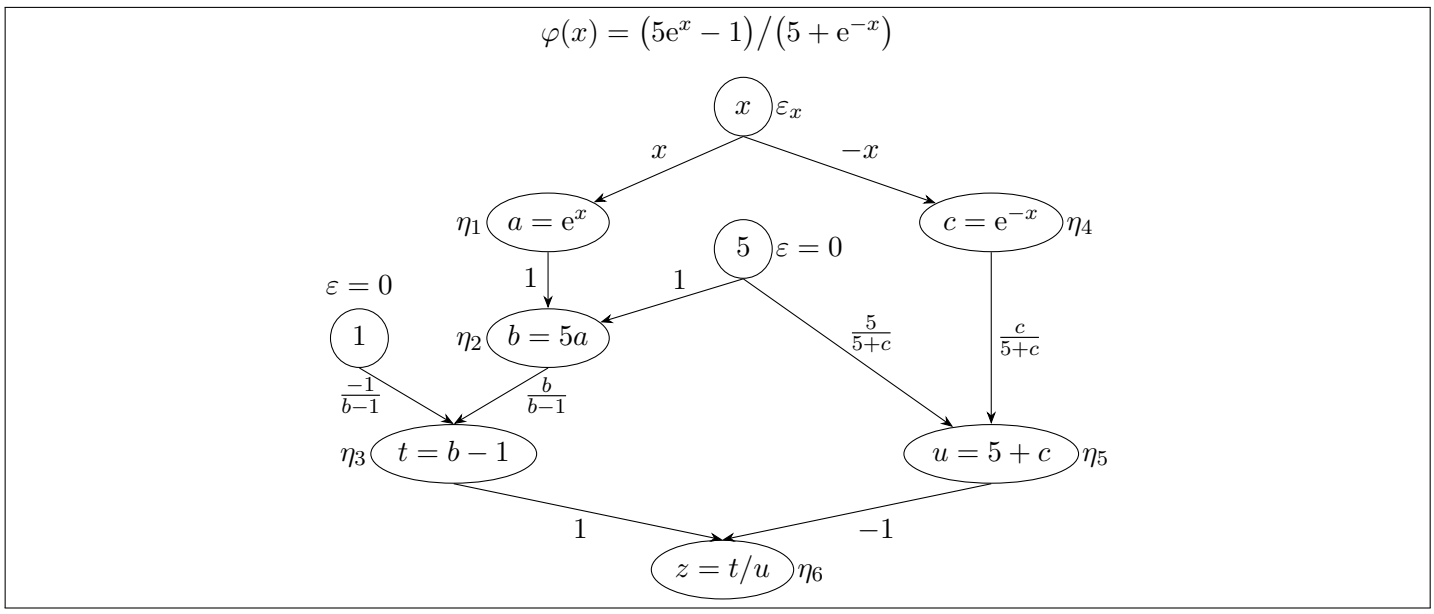


Figura 2: grafo della funzione $\varphi(x)$ dell'esercizio 2.

Notiamo che la funzione è definita $\forall x \in \mathbb{R}$, ossia $\Omega = \mathbb{R}$, in quanto il denominatore è sempre non nullo. Con riferimento al grafo della figura 2, l'errore inerente è dato da:

$$\epsilon_{\text{dati}} = \left(x \frac{b}{b-1} + (-x)\frac{c}{5+c}(-1) \right) \varepsilon_x = \left(x \frac{5e^x}{5e^x - 1} + x \frac{e^{-x}}{5 + e^{-x}} \right) \varepsilon_x = \left(\frac{5e^x}{5e^x - 1} + \frac{e^{-x}e^x}{5e^x + 1} \right) x\varepsilon_x$$
$$= \frac{5e^x(1 + 5e^x) + (5e^x - 1)}{(5e^x - 1)(5e^x + 1)} x\varepsilon_x = \frac{25e^{2x} + 10e^x - 1}{25e^{2x} - 1} x\varepsilon_x = \left(1 + \frac{10e^x}{25e^{2x} - 1} \right) x\varepsilon_x = \theta\varepsilon_x$$

L'indice di condizionamento risulta:

$$I_{\text{cond}} = |\theta| = \left|1 + \frac{10\mathrm{e}^x}{25\mathrm{e}^{2x} - 1}\right| |x| \,.$$

Nel denominatore si ha:

$$25\mathrm{e}^{2x} - 1 = 0 \quad \Leftrightarrow \quad \mathrm{e}^{2x} = 1/25 \quad \Leftrightarrow \quad x = \frac{1}{2}\ln(1/5^2) = \ln(1/5) = -\ln(5)\,.$$

Poiché per $x_0 = -\ln(5)$ il numeratore è non nullo (infatti $\mathrm{e}^{x_0} = \mathrm{e}^{\ln(1/5)} = 1/5 > 0$) si conclude che

$$\left|1 + \frac{10\mathrm{e}^x}{25\mathrm{e}^{2x} - 1}\right| \xrightarrow[x \to x_0]{} +\infty$$

e quindi che $I_{\text{cond}} \xrightarrow[x \to x_0]{} +\infty$, ossia nei pressi di x_0 si ha malcondizionamento del problema. Inoltre, dato che

$$\lim_{x \to +\infty} \frac{10\mathrm{e}^x}{25\mathrm{e}^{2x} - 1} \overset{t=\mathrm{e}^x}{=} \lim_{t \to +\infty} \frac{10t}{25t^2 - 1} = 0 \quad \text{e anche} \quad \lim_{x \to -\infty} \frac{10\mathrm{e}^x}{25\mathrm{e}^{2x} - 1} \overset{t=\mathrm{e}^x}{=} \lim_{t \to 0^+} \frac{10t}{25t^2 - 1} = 0$$

discende che $I_{\text{cond}} \xrightarrow[|x| \to \infty]{} +\infty$. Pertanto, si ha malcondizionamento del problema anche per valori (positivi o negativi) di modulo molto grande.

L'errore algoritmico e l'indice algoritmico sono dati da:

$$\epsilon_{\text{alg}} = \frac{b}{b-1}(\eta_1 + \eta_2) + \eta_3 + \frac{c}{5+c}\eta_4 - \eta_5 + \eta_6 = \frac{5\mathrm{e}^x}{5\mathrm{e}^x - 1}(\eta_1 + \eta_2) + \eta_3 + \frac{\mathrm{e}^{-x}}{5 + \mathrm{e}^{-x}}\eta_4 - \eta_5 + \eta_6$$
$$I_{\text{alg}} = \left|\frac{5\mathrm{e}^x}{5\mathrm{e}^x - 1}\right| 2 + \left|\frac{1}{5\mathrm{e}^x + 1}\right| + 3 = \frac{10\mathrm{e}^x}{\left|5\mathrm{e}^x - 1\right|} + \frac{1}{5\mathrm{e}^x + 1} + 3$$

Si vede immediatamente che il secondo addendo di I_{alg} è limitato per qualunque valore $x \in \mathbb{R}$. Resta pertanto da considerare solo il primo addendo, il quale, come abbiamo visto prima, diverge per $x \longrightarrow \ln(1/5)$. Dunque nei pressi del punto $x_0 = \ln(1/5)$ si ha anche instabilità dell'algoritmo per il calcolo di $\varphi(x)$, oltre che malcondizionamento del problema. Si noti tuttavia che I_{alg} rimane limitato per $|x| \longrightarrow +\infty$, perché

$$\lim_{x \to -\infty} \frac{10\mathrm{e}^x}{\left|5\mathrm{e}^x - 1\right|} = 0 \qquad \text{e} \qquad \lim_{x \to +\infty} \frac{10\mathrm{e}^x}{\left|5\mathrm{e}^x - 1\right|} = 2\,.$$

Pertanto $x_0 = \ln(1/5)$ rimane l'*unico* punto di instabilità per il calcolo di $\varphi(x)$.

Matlab

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

```
% Cognome Nome
% Matricola
%-----------------------------
%  esercizio 2 - 08/02/2023
%-----------------------------
function [Y, varargout] = esercizio2( x, c, N )
% ESERCIZIO2 - Prova scritta di Calcolo Numerico del 08/02/2023
%   Valuta la funzione
%    f_k(t) = ( c(1)*exp(k*t/2) - c(2) ) / ( c(3) + c(4)*exp(-k*t) )
%   nel punto t = x per tutti i k da 1 a N
% SYNOPSIS
%   [Y[, tempo]] = esercizio2( x, c, N )
% INPUT
%   x (double)         - Punto o vettore colonna di punti in cui valutare la
%                        funzione f_k(t) per tutti i k da 1 a N
%   c (double array)   - Vettore dei coefficienti della funzione, c(3) e c(4) non
%                        entrambi nulli
%   N (integer)        - Numero di funzioni diverse da valutare (intero
%                        positivo maggiore di zero)
% OUTPUT
%   Y (doube array)    - Vettore riga o matrice di N colonne, contenente i
%                        valori di f_k(x) per tutti i k da 1 a N nel/i
```

```
%                         punto/i x
%   tempo (opzionale) - Se richiesto in output, vettore di due elementi
%                         contenente il tempo di valutazione delle funzioni
%                         f_k(t) in tutti i punti x con sitassi vettoriale
%                         (tempo(1)) e senza sintassi vettoriale (tempo(2))

% Controllo dei parametri di ingresso
if ( (N <= 0) || (N ~= fix(N)) )
    error('Il parametro N deve essere intero positivo'); end
c = c(:);
if ( (numel(c) ~= 4) || (~isnumeric(c)) )
    error('Il parametro c deve essere un vettore di quattro valori reali'); end
if ( any(c(3:4) <= 1.0E-4) )
    error('I parametri c(3) e c(4) devono essere entrambi > 1.0E-4'); end

% Calcolo della funzione
x = x(:); n = numel(x);
tic;
Y = x * [1:N]; % diade
Y = ( c(1)*exp(0.5*Y) - c(2) ) ./ ( c(3) + c(4)*exp(-Y) );
% Oppure (ma meno efficientemente):
% Y = zeros(n,N);
% for k = 1 : N
%     Y(:,k) =  ( c(1)*exp(0.5*k*x) - c(2) ) ./ ( c(3) + c(4)*exp(-Y) );
% end
if (nargout == 1), return; end
tempo(1) = toc; tic;
Y = zeros(n,N);
for k = 1 : N
    for i = 1 : n
        Y(i,k) = ( c(1)*exp(0.5*k*x(i)) - c(2) ) / ( c(3) + c(4)*exp(-k*x(i)) );
    end
end
tempo(2) = toc;
varargout{1} = tempo;

end
```

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

```
% Cognome Nome
% Matricola
%---------------------------------
%  test esercizio 2 - 08/02/2023
%---------------------------------
close all; clear all; clc
disp('Test esercizio 2');

xx = linspace(-0.2, 0.3, 3001);
coeff = [-2; 5.2; 0.41; 3.7]; N = 6;
Y1 = esercizio2( xx, coeff, N );
figure(1); ph1 = plot(xx,Y1');
coeff = [exp(-2); 0.7; pi/3; 1.4]; N = 20;
[Y2, tempo] = esercizio2( xx, coeff, N );
figure(2); ph2 = plot(xx,Y2');
fprintf('\nTempi di esecuzione con c = [%g; %g; %g; %g] ed N = %d:',coeff,N);
fprintf('\n- sintassi vettoriale: %.3g secondi', tempo(1));
fprintf('\n- sintassi non vettoriale: %.3g secondi', tempo(2));
fprintf('\nRapporto percentuale vettoriale/scalare: %5.1f%%\n\n', ...
        100*tempo(1)/tempo(2));
```

L'esecuzione dell'M-script di prova `testEsercizio2.m` fornisce un output simile al seguente:

```
Test esercizio 2

Tempi di esecuzione con c = [-2.3; 1.5708] ed N = 20:
- sintassi vettoriale: 0.00103 secondi
- sintassi non vettoriale: 0.00435 secondi
Rapporto percentuale vettoriale/scalare:  23.8%
```

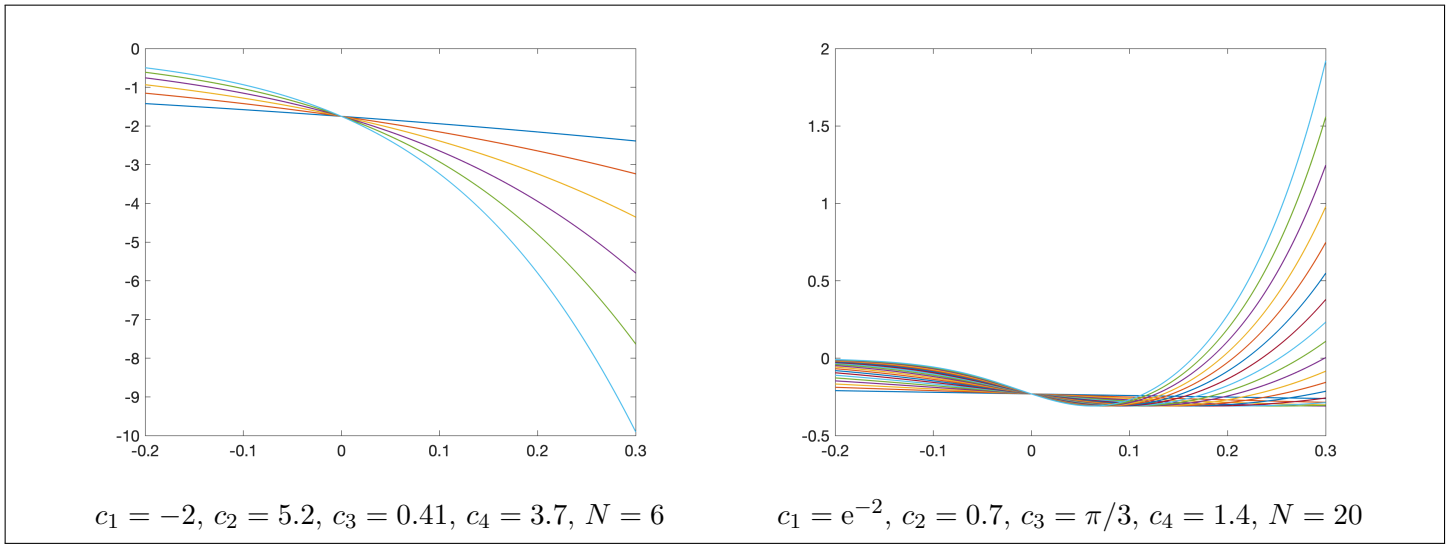


Figura 3: grafici generati dall'M-script `esercizio2.m` con i valori dei parametri indicati nel testo dell'esercizio (figure leggermente migliorate per esigenze di stampa).

e i grafici riportati in figura 3. Naturalmente, gli effettivi valori dei tempi sono quasi certamente differenti da quelli riportati qui, in quanto dipendono fortemente dalla macchina su cui si esegue il codice. Per una valutazione più affidabile occorrerebbe fare una statistica su più esecuzioni del codice. Quello che dovrebbe rimanere vero è che, in questo esercizio, il tempo con sintassi vettoriale sia circa un quarto di quello senza sintassi vettoriale.

Si noti anche che se per la parte con sintassi vettoriale si utilizza il ciclo `for` delle righe commentate nel sorgente, il tempo impiegato è un po' maggiore di quello con la diade e risulta circa la metà del tempo impiegato con sintassi non vettoriale.

Si noti altresì che, essendo forniti in secondi, è poco rilevante riportare i tempi di esecuzione con un'accuratezza eccessiva: per questo il formato di stampa `%.3g` specifica di usare al più tre cifre significative (nel più corto fra i due formati `%f` e `%e`).

Nota. Poiché la funzione da implementare nella M-function `Esercizio2.m` prevede il calcolo di una quantità frazionaria, un'attenta gestione dei controlli sui parametri di ingresso dovrebbe prevedere anche un controllo per eventuali denominatori nulli o, meglio, sotto una soglia prefissata τ legata alla precisione di macchina, prima di eseguire il calcolo effettivo. Tale controllo puiò facilmente essere implementato in modo vettoriale con poche istruzioni, come mostrato più sotto. La scelta della soglia τ verso zero è abbastanza arbitraria e tipicamente dipende dal problema. Inoltre, nel caso di presenza di denominatori nulli (o, come detto, sotto la soglia τ), è molto importante gestire correttamente anche l'output. Fra i diversi modi, le due alternative seguenti sono certamente fra i più consigliabili:

1) memorizzare direttamente nella corrispondente posizione del parametro di output un valore predefinito di "allerta", che molto spesso è `NaN`, **senza** eseguire il corrispondente calcolo. In tal modo, l'array `Y` di output mantiene la stessa dimensione per righe del parametro di ingresso `x`, ma le posizioni "critiche" sono facilmente individuabili;
2) rimuovere **prima** dei calcoli, dal vettore `x` in input, quelle componenti che provocano i valori nulli (o sotto soglia) a denominatore. In tal caso, però, il parametro `Y` di uscita avrà una dimensione per righe **minore** di quella del vettore `x` di input e quindi diventa necessario restituire al codice chiamante l'informazione di **quali** componenti del parametro `x` di ingresso sono state rimosse, restituendo in output un ulteriore parametro di uscita con gli indici di tali componenti. Inoltre, potrebbe accadere che la stessa componente x_i provochi un denominatore sotto soglia per certi valori di k, ma non per altri: con l'approccio di questa alternativa, diversamente dall'alternativa 1, verrebbero però esclusi dal calcolo tutti i valori di tale i-esima componente, anche quelli per i quali eventualmente il denominatore fosse sopra la soglia τ.

Per quanto detto, quindi, in generale l'alternativa preferibile è la 1. Una sua possibile implementazione è la seguente: si inserisce subito dopo i controlli sui parametri di ingresso il calcolo della soglia τ verso zero con una riga del tipo

```
tau = norm(x, 'inf')*eps;  % soglia verso zero
```

che tiene conto della precisione di macchina e dell'ordine di grandezza delle componenti di `x`; dopo la riga di costruzione della diade `Y`, si sostituisce il calcolo immediato dell'array `Y` di output con il codice seguente:

```
denom = c(3) + c(4)*exp(-Y);              % calcolo vettoriale di tutti i denominatori
indNullDenom = find( abs(denom) < tau );  % indici lineari dei denominatori sotto soglia
```

```
if ( isempty(indNullDenom) )
   Y = ( c(1)*exp(0.5*Y) - c(2) ) ./ ( c(3) + c(4)*exp(-Y) );
else
   Y(indNullDenom) = NaN; indDenom = setdiff([1:numel(Y)]', indNullDenom);
   Y(indDenom) = ( c(1)*exp(0.5*Y(indDenom)) - c(2) ) ./ ( c(3) + c(4)*exp(-Y(indDenom)) );
end
```

che esegue la stessa istruzione della versione precedente nel caso non vi siano denominatori troppo piccoli, ma conserva la sintassi vettoriale anche nel caso di denominatori sotto soglia; infine, nella parte di calcolo non vettoriale, si sostituisce il corpo del doppio ciclo `for` con il codice seguente:

```
    if ( isempty(indNullDenom) )
        Y(i,k) = ( c(1)*exp(0.5*k*x(i)) - c(2) ) / ( c(3) + c(4)*exp(-k*x(i)) );
    else
        ind = sub2ind([n, N], i, k);  pos = find( indNullDenom == ind );
        if ( isempty(pos) )
            Y(i,k) = ( c(1)*exp(0.5*k*x(i)) - c(2) ) / ( c(3) + c(4)*exp(-k*x(i)) );
        else
            Y(ind) = NaN; indNullDenom(pos) = [];
        end
    end
```

In quest'ultimo frammento di codice, man mano che si incontrano gl'indici delle componenti con denominatori sotto soglia, tali indici vengono rimossi dal vettore `indNullDenom`, così da rendere minimo il numero di volte in cui si esegue il ramo `else` del costrutto condizionale `if` più esterno, che fa operazioni addizionali. Infine, in quest'ultimo frammento di codice, per il calcolo dei valori delle singole componenti sopra soglia dell'output si potrebbe certamente utilizzare il valore contenuto nella corrispondente posizione della matrice `denom`, calcolata all'inizio per il test:

```
Y(i,k) = ( c(1)*exp(0.5*k*x(i)) - c(2) ) / denom( i, k );
```

Tuttavia, questo comporterebbe un significativo risparmio di tempo rispetto al calcolo esplicito dell'operazione del denominatore, falsando dunque la temporizzazione di questa seconda parte della M-function, perchè il tempo di calcolo di `denom` (per di più in modalità vettoriale) è conteggiato solo nella prima parte del codice. Pertanto, poiché lo scopo di questa seconda parte di codice è effettivamente di temporizzare il tempo di calcolo in modalità **non vettoriale** per confrontarlo con quello in modalità vettoriale, il calcolo dei denominatori sopra soglia viene rifatto.

Soluzione esercizio 3

Teoria

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

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

Una matrice $G_{ij}(\varphi) \in \mathcal{M}_n(\mathbb{R})$ con $i < j$ è di rotazione elementare (di angolo φ) se coincide con la matrice identità di ordine n tranne nei quattro elementi di posizioni (i,i), (i,j), (j,i) e (j,j), i quali valgono rispettivamente c, s, $-s$ e c con $c = \cos(\varphi)$ ed $s = \sin(\varphi)$. Una matrice di rotazione elementare è una matrice ortogonale, quindi invertibile e tale che $G_{ij}(\varphi)^{-1} = G_{ij}(\varphi)^T$. Ricordiamo inoltre che, dato un vettore $\boldsymbol{x} = (x_1, \ldots, x_i, \ldots, x_j, \ldots, x_n)^T \in \mathbb{R}^n$ con $x_i, x_j \neq 0$, $i < j$, esiste una matrice di rotazione elementare G_{ij} tale che $\boldsymbol{y} = G_{ij}\boldsymbol{x}$ ha la j-esima componente nulla, ossia $y_j = 0$. I valori c ed s per tale matrice di rotazione sono

$$c = x_i \Big/ \sqrt{x_i^2 + x_j^2} \qquad \text{e} \qquad s = x_j \Big/ \sqrt{x_i^2 + x_j^2}\,.$$

Gli stessi valori possono essere calcolati in modo stabile usando le seguenti formule:

$$\text{se } |x_j| \leq |x_i| \text{ allora } t = \frac{x_j}{x_i},\ c = \frac{1}{\sqrt{1+t^2}},\ s = t \cdot c\,; \qquad \text{se } |x_j| > |x_i| \text{ allora } t = \frac{x_i}{x_j},\ s = \frac{1}{\sqrt{1+t^2}},\ c = t \cdot s\,.$$

Applicando dunque il metodo di Givens per risolvere il sistema di equazioni si ha:

$$\left[A^{(1)}|\boldsymbol{b}^{(1)}\right] = \left(\begin{array}{cccc|c} 1/3 & 0 & 1/2 & -1 & 0 \\ 0 & 3 & 1/4 & 0 & -1/3 \\ 1/2 & 0 & 2 & 0 & 1 \\ 1 & 0 & -1 & 1/3 & 1/4 \end{array}\right)$$

$$a_{31}^{(1)} = 1/2 > a_{11}^{(1)} = 1/3 \;\Rightarrow\; t = \frac{a_{11}^{(1)}}{a_{31}^{(1)}} = \frac{2}{3} \;\Rightarrow\; s^{(1)} = \frac{1}{\sqrt{1+t^2}} = \frac{3}{\sqrt{13}}, \quad c^{(1)} = t \cdot s^{(1)} = \frac{2}{\cancel{3}}\frac{\cancel{3}}{\sqrt{13}} = \frac{2}{\sqrt{13}}$$

$$G_{13} = \begin{pmatrix} 2/\sqrt{13} & 0 & 3/\sqrt{13} & 0 \\ 0 & 1 & 0 & 0 \\ -3/\sqrt{13} & 0 & 2/\sqrt{13} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \approx \begin{pmatrix} 0.5547 & 0 & 0.8321 & 0 \\ 0 & 1 & 0 & 0 \\ -0.8321 & 0 & 0.5547 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\left[A^{(2)}|\boldsymbol{b}^{(2)}\right] = G_{13}\left[A^{(1)}|\boldsymbol{b}^{(1)}\right] = \left(\begin{array}{cccc|c} \sqrt{13}/6 & 0 & 7/\sqrt{13} & -2/\sqrt{13} & 3/\sqrt{13} \\ 0 & 3 & 1/4 & 0 & -1/3 \\ 0 & 0 & 5/\left(2\sqrt{13}\right) & 3/\sqrt{13} & 2/\sqrt{13} \\ 1 & 0 & -1 & 1/3 & 1/4 \end{array}\right)$$

$$\approx \left(\begin{array}{cccc|c} 0.6009 & 0 & 1.9415 & -0.5547 & 0.8321 \\ 0 & 3 & 0.2500 & 0 & -0.3333 \\ 0 & 0 & 0.6934 & 0.8321 & 0.5547 \\ 1 & 0 & -1 & 0.3333 & 0.2500 \end{array}\right)$$

$$\left|a_{41}^{(2)}\right| = 1 > \frac{\sqrt{13}}{6} = \left|a_{11}^{(2)}\right| \;\Rightarrow\; t = \frac{a_{11}^{(2)}}{a_{41}^{(2)}} = a_{11}^{(2)} = \frac{\sqrt{13}}{6} \;\Rightarrow\; \begin{cases} s^{(2)} = \dfrac{1}{\sqrt{1+t^2}} = \dfrac{6}{\sqrt{49}} = \dfrac{6}{7} \\ c^{(2)} = t \cdot c^{(2)} = \dfrac{\sqrt{13}}{\cancel{6}}\dfrac{\cancel{6}}{7} = \dfrac{\sqrt{13}}{7} \end{cases}$$

$$G_{14} = \begin{pmatrix} \sqrt{13}/7 & 0 & 0 & 6/7 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -6/7 & 0 & 0 & \sqrt{13}/7 \end{pmatrix} \approx \begin{pmatrix} 0.5151 & 0 & 0 & 0.8571 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -0.8571 & 0 & 0 & 0.5151 \end{pmatrix}$$

$$\left[A^{(3)}|\boldsymbol{b}^{(3)}\right] = G_{14}\left[A^{(2)}|\boldsymbol{b}^{(2)}\right] = \left(\begin{array}{cccc|c} 7/6 & 0 & 1/7 & 0 & 9/14 \\ 0 & 3 & 1/4 & 0 & -1/3 \\ 0 & 0 & 5/\left(2\sqrt{13}\right) & 3/\sqrt{13} & 2/\sqrt{13} \\ 0 & 0 & -55/\left(7\sqrt{13}\right) & 7/\left(3\sqrt{13}\right) & -59/\left(28\sqrt{13}\right) \end{array}\right)$$

$$\approx \left(\begin{array}{cccc|c} 1.1667 & 0 & 0.1429 & 0 & 0.6429 \\ 0 & 3 & 0.2500 & 0 & -0.3333 \\ 0 & 0 & 0.6934 & 0.8321 & 0.5547 \\ 0 & 0 & -2.1792 & 0.6472 & -0.5844 \end{array}\right)$$

$$\left|a_{43}^{(3)}\right| = \frac{55}{7\sqrt{13}} > \frac{5}{2\sqrt{13}} = \left|a_{33}^{(3)}\right| \;\Rightarrow\; t = \frac{a_{33}^{(3)}}{a_{43}^{(3)}} = \frac{-\cancel{5}}{\cancel{2\sqrt{13}}} \cdot \frac{7\cancel{\sqrt{13}}}{\cancel{55}_{11}} = \frac{-7}{22} \;\Rightarrow\; \begin{cases} s^{(3)} = \dfrac{1}{\sqrt{1+t^2}} = \dfrac{22}{\sqrt{533}} \\ c^{(3)} = t \cdot c^{(3)} = \dfrac{-7}{\cancel{22}}\dfrac{\cancel{22}}{\sqrt{533}} = \dfrac{-7}{\sqrt{533}} \end{cases}$$

$$G_{34} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -7/\sqrt{533} & 22/\sqrt{533} \\ 0 & 0 & -22/\sqrt{533} & -7/\sqrt{533} \end{pmatrix} \approx \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -0.3032 & 0.9529 \\ 0 & 0 & -0.9529 & -0.3032 \end{pmatrix}$$

$$\left[A^{(4)}|\boldsymbol{b}^{(4)}\right] = G_{34}\left[A^{(3)}|\boldsymbol{b}^{(3)}\right] = \left(\begin{array}{cccc|c} 7/6 & 0 & 1/7 & 0 & 9/14 \\ 0 & 3 & 1/4 & 0 & -1/3 \\ 0 & 0 & -5\sqrt{41}/14 & 7/\left(3\sqrt{41}\right) & -65/\left(14\sqrt{41}\right) \\ 0 & 0 & 0 & -19/\left(3\sqrt{41}\right) & -9/\left(4\sqrt{41}\right) \end{array}\right)$$

$$\approx \left(\begin{array}{cccc|c} 1.1667 & 0 & 0.1429 & 0 & 0.6429 \\ 0 & 3 & 0.2500 & 0 & -0.3333 \\ 0 & 0 & -2.2868 & 0.3644 & -0.7251 \\ 0 & 0 & 0 & -0.9891 & -0.3514 \end{array}\right)$$

$$= [R|\boldsymbol{d}] \text{ con } \boldsymbol{d} = Q^T\boldsymbol{b}$$

$$Q = G_{13}^T G_{14}^T G_{34}^T = \begin{pmatrix} 2/7 & 0 & -9/\left(7\sqrt{41}\right) & 6/\sqrt{41} \\ 0 & 1 & 0 & 0 \\ 3/7 & 0 & -38/\left(7\sqrt{41}\right) & -2/\sqrt{41} \\ 6/7 & 0 & 22/\left(7\sqrt{41}\right) & -1/\sqrt{41} \end{pmatrix} \approx \begin{pmatrix} 0.2857 & 0 & -0.2008 & 0.9370 \\ 0 & 1 & 0 & 0 \\ 0.4286 & 0 & -0.8478 & -0.3123 \\ 0.8571 & 0 & 0.4908 & -0.1562 \end{pmatrix}$$

$$R = \begin{pmatrix} 7/6 & 0 & 1/7 & 0 \\ 0 & 3 & 1/4 & 0 \\ 0 & 0 & -5\sqrt{41}/14 & 7/\left(3\sqrt{41}\right) \\ 0 & 0 & 0 & -19/\left(3\sqrt{41}\right) \end{pmatrix} \approx \begin{pmatrix} 1.1667 & 0 & 0.1429 & 0 \\ 0 & 3 & 0.2500 & 0 \\ 0 & 0 & -2.2868 & 0.3644 \\ 0 & 0 & 0 & -0.9891 \end{pmatrix}$$

$$\boldsymbol{d} = \begin{pmatrix} 9/14 \\ -1/3 \\ -65/\left(14\sqrt{41}\right) \\ -9/\left(4\sqrt{41}\right) \end{pmatrix} \approx \begin{pmatrix} 0.6429 \\ -0.3333 \\ -0.7251 \\ -0.3514 \end{pmatrix}$$

$$\left.\begin{aligned} x_4^* &= \frac{d_4}{r_{44}} = \frac{-9/\left(4\sqrt{41}\right)}{-19/\left(3\sqrt{41}\right)} = \frac{27}{76} \\ x_3^* &= \frac{d_3 - r_{34}x_4^*}{r_{33}} = \frac{-65/\left(14\sqrt{41}\right) - \left(7/(3\sqrt{41})\right)\cdot(27/76)}{-5\sqrt{41}/14} = \frac{71}{190} \\ x_2^* &= \frac{d_2 - r_{23}x_3^* - r_{24}x_4^*}{r_{22}} = \frac{(-1/3) - (1/4)\cdot(-5\sqrt{41}/14) - 0\cdot(27/76)}{3} = -\frac{973}{6840} \\ x_1^* &= \frac{d_1 - r_{12}x_2^* - r_{13}x_3^* - r_{14}x_4^*}{r_{11}} = \frac{(9/14) - 0\cdot(-973/6840) - (1/7)(71/190) - 0\cdot(27/76)}{7/6} = \frac{48}{95} \end{aligned}\right\} \Rightarrow$$

$$\Rightarrow \boldsymbol{x}^* = \begin{pmatrix} 48/95 \\ -973/6840 \\ 71/190 \\ 27/76 \end{pmatrix} \approx \begin{pmatrix} 0.5053 \\ -0.1423 \\ 0.3737 \\ 0.3553 \end{pmatrix}$$

Non richiesto: $\det(A) = \det(R) = \frac{7}{6}\cdot 3 \cdot \frac{-5\sqrt{41}}{14_2}\cdot\frac{-19}{3\sqrt{41}} = \frac{95}{12} \approx 7.9167$

Matlab

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

```
% Cognome Nome
% Matricola
%-----------------------------
%  esercizio 3 - 08/02/2023
%-----------------------------
function [rango, deter, normaInf] = miaDiade(r, s)
% MIADIADE - Function esercizio 3 prova di Calcolo Numerico dell'08/02/2023
%   Calcola la diade r*s' e ne ricava rango, determinante e norma infinito.
% SYNOPSIS:
%   [rango, deter, normaInf] = miaDiade(r, s)
% INPUT
%   r, s  (double vectors) - Vettori di numeri reali
% OUTPUT
%   rango    (integer) - Rango numerico della diade
%   deter    (double)  - Determinante della diade
%   normaInf (double)  - Norma infinito della diade
%
  if ( isempty(r) || isempty(s) )
      error('I due vettori di input devono essere non vuoti');
  elseif ( ~isvector(r) || ~isvector(s) )
      warning('Array in input non vettori colonna: trasformati in vettori colonna');
  end
  r = r(:); s = s(:); % assicura che r e s siano vettori colonna
  A = r*s'; [m,n] = size(A);
  [Q,R] = qr(A);
  diagonaleR = diag(R);
  normaInf = norm(A, inf);
  rango = numel( find(abs(diagonaleR) > eps*normaInf) );
```

```
  deter = prod( diagonaleR );
end % fine della function miaDiade
```

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

```
% Cognome Nome
% Matricola
%----------------------------
%  Esercizio 3 - 08/02/2023
%----------------------------
close all; clear all; clc
disp('Esercizio 3');

%
% Prima parte
%
A = [1/3, 0, 0.5, -1; 0, 3, 0.25, 0; 0.5, 0, 2, 0; 1, 0, -1, 1/3];
b = [0, -1/3,  1, 0.25]';
[Q, R] = qr( A )
xQR = utrisol(R, Q'*b)
[L,U,p] = lu( A, 'vector' );
xLU = utrisol(U, ltrisol(L, b(p)) )
normaDiffSol = norm( xLU - xQR, inf );
fprintf('\nNorma infinito differenza soluzioni: %g\n\n', normaDiffSol);
normaRelDiffSol = normaDiffSol / norm( xQR, inf )
%
% Seconda parte
%
[rankDiade, detDiade, normInfDiade] = miaDiade(Q(:,1), R(1,:)')
```

Eseguendo lo script si ottiene il seguente output a console:

```
Esercizio 3

Q =
   -0.2857         0   -0.2008   -0.9370
         0    1.0000         0         0
   -0.4286         0   -0.8478    0.3123
   -0.8571         0    0.4908    0.1562


R =
   -1.1667         0   -0.1429   -0.0000
         0    3.0000    0.2500         0
         0         0   -2.2868    0.3644
         0         0         0    0.9891

xQR =
    0.5053
   -0.1423
    0.3737
    0.3553

xLU =
    0.5053
   -0.1423
    0.3737
    0.3553

Norma infinito differenza soluzioni: 2.22045e-16

normaRelDiffSol =
   4.3946e-16

rankDiade =
     1

detDiade =
     0

normInfDiade =
    1.1224
```

Osservazione 1. La norma della differenza delle soluzioni non viene esattamente zero, bensì di circa $2.22 \cdot 10^{-16}$: questo ha senso ed è lecito, in quanto le due soluzioni, che sono identiche in aritmetica esatta, vengono calcolate numericamente con algoritmi differenti (fattorizzazione QR una, metodo di Gauss con pivoting parziale l'altra) e quindi si ha un diverso accumulo degli errori dell'aritmetica finita. In ogni caso, la differenza assoluta sulle singole componenti è dell'ordine della precisione di macchina ed è accettabile, in quanto la differenza relativa ha anch'essa norma dell'ordine della precisione di macchina: circa $4.39 \cdot 10^{-16}$. Inoltre, le matrici Q ed R ritornate dalla routine predefinita `qr` di Matlab hanno alcune differenze nei segni di alcuni elementi, sebbene i valori assoluti siano essenzialmente gli stessi (differenze dell'ordine della precisione di macchina). Questo è dovuto al fatto che la routine `qr` di Matlab utilizza per la fattorizzione QR un algoritmo diverso da quello di Givens (usa i cosiddetti "riflettori di Householder"). Tuttavia, questo non modifica la soluzione del sistema.

Nell'esecuzione dello script Matlab di questo esercizio, la M-function `miaDiade` riporta rango 1 per la diade $T = Q_{*,1}R_{1,*}$: questo risultato è corretto. Infatti, sia la prima colonna di Q (che è una matrice ortogonale, quindi di rango massimo perché invertibile), che la prima riga di R sono vettori non nulli e dunque di rango 1: per la relazione $\text{rank}(MN) \leq \min\{\text{rank}(M), \text{rank}(N)\}$ si ha $\text{rank}(T) \leq \min\{\text{rank}(Q_{*,1}), \text{rank}(R_{1,*})\} = 1$ ed avendo T almeno un elemento non nullo, si conclude che $\text{rank}(T) = 1$.

Osservazione 2. Il rango di una diade formata dal prodotto di vettori non identicamente nulli è **sempre** 1. Tuttavia, l'accumulo degli errori dell'aritmetica finita può portare ad un **rango numerico** differente. Nel caso in esame, ad esempio, se indichiamo con Q_d ed R_d i fattori della fattorizzazione QR della diade, è facile vedere che nel fattore R_d ci sono ben due elementi diagonali non identicamente nulli, oltre a quello di posizione $(1,1)$. Tali elementi hanno valori di modulo molto piccolo, circa $2.037 \cdot 10^{-17}$ e $4.107 \cdot 10^{-49}$, addirittura molto più piccoli della precisione di macchina, ma sono comunque **non nulli**. Quindi, se non si tenesse conto della precisione di macchina nel determinare quanti elementi **numericamente** non nulli ci sono sulla diagonale di R_d, il rango della diade risulterebbe 3, che è sbagliato.

Soluzione esercizio 4

Teoria

La matrice M è **non simmetrica**, sparsa, ma **non tridiagonale**. Per la matrice M, i dischi di Gershgorin per righe sono (figura 4):

$$\mathcal{H}_1 = \left\{ x \in \mathbb{C} \,\middle|\, |x - 9/4| \le 5/6 \approx 0.8333 \right\}, \qquad \mathcal{H}_2 = \left\{ x \in \mathbb{C} \,\middle|\, |x + 3/2| \le 3/2 \right\},$$
$$\mathcal{H}_3 = \left\{ x \in \mathbb{C} \,\middle|\, |x - 7/4| \le 5/6 \approx 0.8333 \right\}, \qquad \mathcal{H}_4 = \left\{ x \in \mathbb{C} \,\middle|\, |x + 3/2| \le 1/2 \right\},$$

mentre i dischi per colonne sono

$$\mathcal{K}_1 = \left\{ x \in \mathbb{C} \,\middle|\, |x - 9/4| \le 1 \right\}, \qquad \mathcal{K}_2 = \left\{ x \in \mathbb{C} \,\middle|\, |x + 3/2| \le 1/3 \approx 0.3333 \right\},$$
$$\mathcal{K}_3 = \left\{ x \in \mathbb{C} \,\middle|\, |x - 7/4| \le 1/3 \approx 0.3333 \right\}, \qquad \mathcal{K}_4 = \left\{ x \in \mathbb{C} \,\middle|\, |x + 3/2| \le 2 \right\}.$$

Pertanto

$$\mathcal{U}_r = \bigcup_{j=1}^{4} \mathcal{H}_j = \mathcal{H}_1 \cup \mathcal{H}_2 \cup \mathcal{H}_3, \qquad \mathcal{U}_c = \bigcup_{j=1}^{4} \mathcal{K}_j = \mathcal{K}_1 \cup \mathcal{K}_4 \quad \Rightarrow \quad \mathcal{S} = \mathcal{U}_r \cap \mathcal{U}_c$$

La regione $\mathcal{S}$ intersezione delle due unioni è descritta in modo dettagliato nella didascalia della figura 4. Si noti che sia per righe, che per colonne, l'unione dei dischi è composta da due unioni disgiunte, ciascuna delle quali

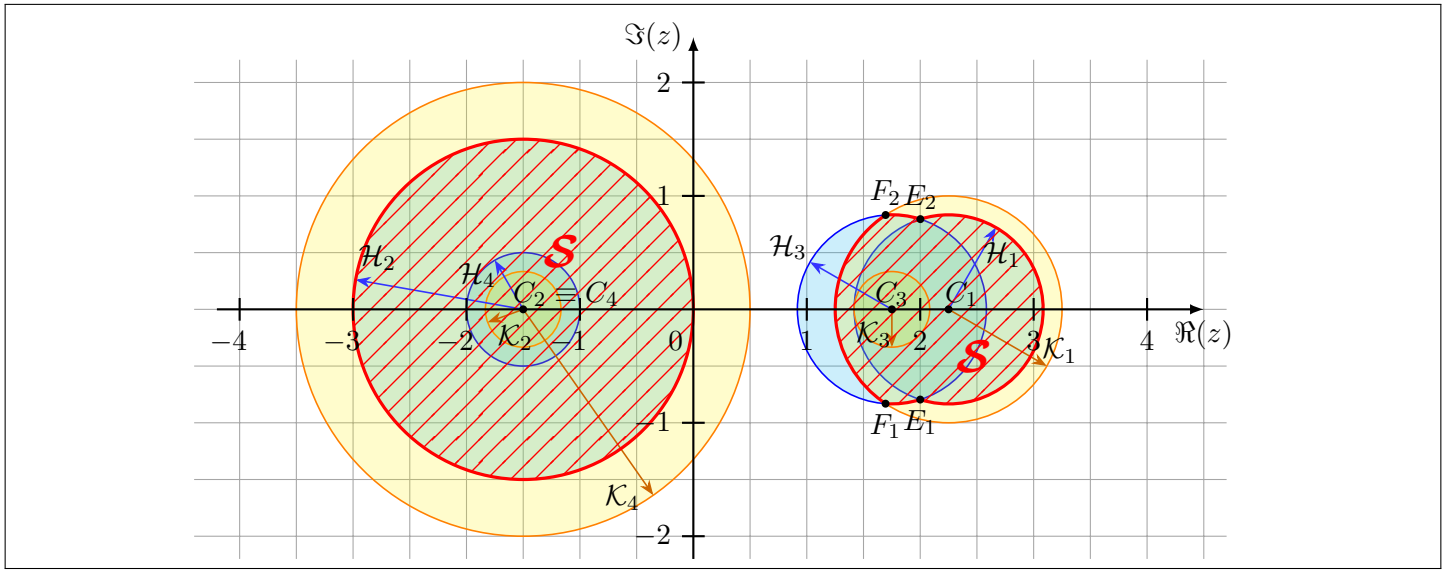


Figura 4: dischi di Ghershgorin per righe ($\mathcal{H}_1, \ldots, \mathcal{H}_4$, in azzurro) e per colonne ($\mathcal{K}_1, \ldots, \mathcal{K}_4$, in giallo) della matrice M dell'Esercizio 4. La regione tratteggiata $\mathcal{S}$ contiene *tutti* gli autovalori di M. Si noti che in questo caso la regione $\mathcal{S}$ è **non connessa**: è costituita da due parti disgiunte. Il bordo della regione $\mathcal{S}$ è composto dall'unione della circonferenza del disco $\mathcal{H}_2$ e dai **quattro diversi archi** di cerchio della parte a destra: i punti E_1 ed E_2 sono i punti di intersezione fra la circonferenza del disco $\mathcal{H}_1$ (di centro $C_1 = (9/4, 0) = (2.25, 0)$ e raggio $5/6 \approx 0.8333$) e quella del disco $\mathcal{H}_3$ (di centro $C_3 = (7/4, 0) = (1.75, 0)$ e raggio $5/6$); i punti F_1 ed F_2 sono i punti di intersezione fra la circonferenza del disco $\mathcal{H}_3$ e quella del disco $\mathcal{K}_1$ (di centro $C_1 = (9/4, 0) = (2.25, 0)$ e raggio 1). Percorrendo dunque il bordo di $\mathcal{S}$ in senso orario a partire da E_1, l'arco di cerchio che va da E_1 a F_1 appartiene alla circonferenza che è la frontiera del disco $\mathcal{H}_3$, l'arco di cerchio che va da F_1 a F_2 appartiene alla circonferenza che è la frontiera del disco $\mathcal{K}_1$, l'arco di cerchio che va da F_2 a E_2 appartiene alla circonferenza che è la frontiera del disco $\mathcal{H}_3$, e, infine, l'arco di cerchio che va da E_2 a E_1 appartiene alla circonferenza che è la frontiera del disco $\mathcal{H}_1$.

contenente due dischi:

$$\mathcal{U}_r = \bigcup_{j=1}^{4} \mathcal{H}_j = \underbrace{(\mathcal{H}_1 \cup \mathcal{H}_3)}_{\mathcal{U}_{r,1}} \cup \underbrace{(\mathcal{H}_2 \cup \mathcal{H}_4)}_{\mathcal{U}_{r,2}} \quad \text{con } (\mathcal{H}_1 \cup \mathcal{H}_3) \cap (\mathcal{H}_2 \cup \mathcal{H}_4) = \emptyset$$

$$\mathcal{U}_c = \bigcup_{j=1}^{4} \mathcal{K}_j = \underbrace{(\mathcal{K}_1 \cup \mathcal{K}_3)}_{\mathcal{U}_{c,1}} \cup \underbrace{(\mathcal{K}_2 \cup \mathcal{K}_4)}_{\mathcal{U}_{c,2}} \quad \text{con } (\mathcal{K}_1 \cup \mathcal{K}_3) \cap (\mathcal{K}_2 \cup \mathcal{K}_4) = \emptyset$$

Pertanto, discende dal terzo Teorema di Gershgorin che due autovalori di M si trovano in $\mathcal{U}_{r,1}$, la prima unione dei dischi per righe, e gli altri due autovalori si trovano in $\mathcal{U}_{r,2}$, la seconda unione dei dischi per righe. Analogamente accade con le due unioni $\mathcal{U}_{c,1}$ e $\mathcal{U}_{c,2}$ dei dischi per colonne. Pertanto, due autovalori si trovano nell'intersezione delle prime due unioni, ossia in $\mathcal{U}_{r,1} \cap \mathcal{U}_{c,1}$, mentre gli altri due autovalori si trovano nell'intersezione delle seconde due unioni, ossia in $\mathcal{U}_{r,2} \cap \mathcal{U}_{c,2}$.

Ora, poiché $\mathcal{H}_4 \subset \mathcal{H}_2 \Rightarrow \mathcal{H}_2 \cup \mathcal{H}_4 = \mathcal{H}_2$, $\mathcal{K}_3 \subset \mathcal{K}_1 \Rightarrow \mathcal{K}_1 \cup \mathcal{K}_3 = \mathcal{K}_1$ e, infine, $\mathcal{H}_2 \subset \mathcal{K}_4$, si ha che le intersezioni delle due coppie di unioni diventano

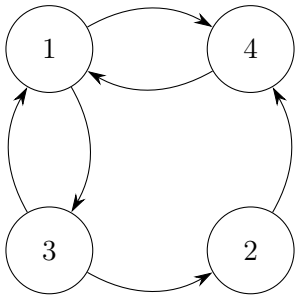
$$\mathcal{U}_{r,1} \cap \mathcal{U}_{c,1} = (\mathcal{H}_1 \cup \mathcal{H}_3) \cap (\mathcal{K}_1 \cup \mathcal{K}_3) = (\mathcal{H}_1 \cup \mathcal{H}_3) \cap \mathcal{K}_1 \quad \text{e} \quad \mathcal{U}_{r,2} \cap \mathcal{U}_{c,2} = (\mathcal{H}_2 \cup \mathcal{H}_4) \cap (\mathcal{K}_2 \cup \mathcal{K}_4) = \mathcal{H}_2 \cap \mathcal{K}_4 = \mathcal{H}_2$$

Quindi, concludiamo che due autovalori sono contenuti in $(\mathcal{H}_1 \cup \mathcal{H}_3) \cap \mathcal{K}_1$ e gli altri due in $\mathcal{H}_2$.

Nel caso in esame, vediamo immediatamente che M è una matrice diagonale dominante sia righe, perché $|m_{i,i}| \geq \sum_{j \neq i} |m_{i,j}| \ \forall i = 1, \dots, 4$, ma **non** per colonne, perché $|m_{4,4}| = 3/2 < 2 = \sum_{i \neq 4} |m_{i,4}|$. Tuttavia, la dominanza diagonale per righe **non vale in senso stretto**, perché nella seconda riga è $|m_{2,2}| = 3/2 = \sum_{j \neq 2} |m_{2,j}|$. Pertanto, la condizione di stretta diagonale dominanza non è verificata, nè per righe, nè per colonne: non possiamo quindi da qui **concludere nulla a priori** riguardo l'eventuale convergenza dei metodi iterativi di Jacobi e di Gauss-Seidel.

Inoltre, poiché $0 \notin \mathcal{S}$ (sta sulla frontiera di $\mathcal{S}$) non possiamo escludere a priori che sia un autovalore di M e quindi **non si può escludere a priori** che M sia singolare.

Controlliamo la riducibilità o meno di M costruendo il grafo della matrice:



Si vede immediatamente che il grafo di M **è strettamente connesso**: ne discende che la matrice M **è irriducibile**.

Concludiamo quindi che M è irriducibilmente diagonale dominante per righe: da qui possiamo quindi concludere **a priori** che i metodi di Jacobi e di Gauss-Seidel **convergono entrambi**, sebbene non sia possibile dire *a priori* quale dei due metodi converga più velocemente.

Osservazione: 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.

Determiniamo ora la matrice J di iterazione di Jacobi, ricordando che qui la matrice L e la matrice U sono, rispettivamente, le **opposte** della parte strettamente triangolare inferiore e della parte strettamente triangolare superiore della matrice M:

$$J = D^{-1}(L+U) = I_4 - D^{-1}M = \begin{pmatrix} 4/9 & 0 & 0 & 0 \\ 0 & -2/3 & 0 & 1 \\ 0 & 0 & 4/7 & 0 \\ 0 & 0 & 0 & -2/3 \end{pmatrix} \begin{pmatrix} 0 & 0 & 1/3 & -1/2 \\ 0 & 0 & 0 & -3/2 \\ 1/2 & -1/3 & 0 & 0 \\ -1/2 & 0 & 0 & 0 \end{pmatrix}$$

$$= \begin{pmatrix} 0 & 0 & 4/27 & -2/9 \\ 0 & 0 & 0 & 1 \\ 2/7 & -4/21 & 0 & 0 \\ 1/3 & 0 & 0 & 0 \end{pmatrix} \approx \begin{pmatrix} 0 & 0 & 0.1481 & -0.2222 \\ 0 & 0 & 0 & 1.0000 \\ 0.2857 & -0.1905 & 0 & 0 \\ 0.3333 & 0 & 0 & 0 \end{pmatrix}$$

Vediamo che **non** vale la condizione $J \geq 0$: ne discende che **non** sono verificate la ipotesi del Teorema di Stein-Rosenberg.

Calcoliamo ora la matrice di iterazione di Gauss-Seidel, ricordando che anche qui le matrici L ed U sono quelle definite anche per la matrice J di Jacobi. Per prima cosa occorre determinare la matrice inversa di $D - L$ (che

coincide con la parte triangolare inferiore di M, diagonale inclusa). In questo caso l'applicazione dell'algoritmo di Gauss-Jordan è molto semplice e veloce:

$$T = \big[D - L \,\big|\, I_4\big] = \left(\begin{array}{cccc|cccc} 9/4 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & -3/2 & 0 & 0 & 0 & 1 & 0 & 0 \\ -1/2 & 1/3 & 7/4 & 0 & 0 & 0 & 1 & 0 \\ 1/2 & 0 & 0 & -3/2 & 0 & 0 & 0 & 1 \end{array}\right)$$

$$\xrightarrow[T_{4,*}\leftarrow T_{4,*}-(2/9)T_{1,*}]{T_{3,*}\leftarrow T_{3,*}+(2/9)T_{1,*}} \left(\begin{array}{cccc|cccc} 9/4 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & -3/2 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1/3 & 7/4 & 0 & 2/9 & 0 & 1 & 0 \\ 0 & 0 & 0 & -3/2 & -2/9 & 0 & 0 & 1 \end{array}\right)$$

$$\xrightarrow{T_{3,*}\leftarrow T_{3,*}+(2/9)T_{2,*}} \left(\begin{array}{cccc|cccc} 9/4 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & -3/2 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 7/4 & 0 & 2/9 & 2/9 & 1 & 0 \\ 0 & 0 & 0 & -3/2 & -2/9 & 0 & 0 & 1 \end{array}\right)$$

$$\xrightarrow{T_{i,*}\leftarrow(1/t_{i,i})T_{i,*},\ i=1,\dots,4} \left(\begin{array}{cccc|cccc} 1 & 0 & 0 & 0 & 4/9 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -2/3 & 0 & 0 \\ 0 & 0 & 1 & 0 & 8/63 & 8/63 & 4/7 & 0 \\ 0 & 0 & 0 & 1 & 4/27 & 0 & 0 & -2/3 \end{array}\right) = \big[I_4 \,\big|\, (D-L)^{-1}\big]$$

$$(D-L)^{-1} = \begin{pmatrix} 4/9 & 0 & 0 & 0 \\ 0 & -2/3 & 0 & 0 \\ 8/63 & 8/63 & 4/7 & 0 \\ 4/27 & 0 & 0 & -2/3 \end{pmatrix} \approx \begin{pmatrix} 0.4444 & 0 & 0 & 0 \\ 0 & -0.6667 & 0 & 0 \\ 0.1270 & 0.1270 & 0.5714 & 0 \\ 0.1481 & 0 & 0 & -0.6667 \end{pmatrix}$$

Nota. Nel caso in esame, in alternativa ad applicare il metodo di Gauss-Jordan, dato che le dimensioni di $D - L$ sono ridotte e la parte sottodiagonale ha diversi elementi nulli, può essere praticabile anche il calcolo esplicito degli elementi sottodiagonali dell'inversa di una triangolare inferiore non singolare. Possiamo ora calcolare la matrice d'iterazione di Gauss-Seidel:

$$\mathcal{G} = (D-L)^{-1}U = \begin{pmatrix} 4/9 & 0 & 0 & 0 \\ 0 & -2/3 & 0 & 0 \\ 8/63 & 8/63 & 4/7 & 0 \\ 4/27 & 0 & 0 & -2/3 \end{pmatrix} \begin{pmatrix} 0 & 0 & 1/3 & -1/2 \\ 0 & 0 & 0 & -3/2 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$= \begin{pmatrix} 0 & 0 & 4/27 & -2/9 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 8/189 & -16/63 \\ 0 & 0 & 4/81 & -2/27 \end{pmatrix} \approx \begin{pmatrix} 0 & 0 & 0.1481 & -0.2222 \\ 0 & 0 & 0 & 1.0000 \\ 0 & 0 & 0.0423 & -0.2540 \\ 0 & 0 & 0.0494 & -0.0741 \end{pmatrix}$$

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

$$\mathcal{G}(\omega) = (D-\omega L)^{-1}\big((1-\omega)D + \omega U\big)$$

$$= \begin{pmatrix} 9/4 & 0 & 0 & 0 \\ 0 & -3/2 & 0 & 0 \\ -\omega/2 & \omega/3 & 7/4 & 0 \\ \omega/2 & 0 & 0 & -3/2 \end{pmatrix}^{-1} \begin{pmatrix} (1-\omega)9/4 & 0 & \omega/3 & -\omega/2 \\ 0 & -(1-\omega)3/2 & 0 & -3\omega/2 \\ 0 & 0 & (1-\omega)7/4 & 0 \\ 0 & 0 & 0 & -(1-\omega)3/2 \end{pmatrix}$$

Parte opzionale.

Determiniamo il polinomio caratteristico della matrice J (sviluppo del determinante con la regola di Laplace, ad esempio lungo la quarta colonna):

$$p_4^J(\lambda) = \det\big(J - \lambda I_4\big) = \begin{vmatrix} -\lambda & 0 & 4/27 & -2/9 \\ 0 & -\lambda & 0 & 1 \\ 2/7 & -4/21 & -\lambda & 0 \\ 1/3 & 0 & 0 & -\lambda \end{vmatrix}$$

$$= -(-2/9)\begin{vmatrix} 0 & -\lambda & 0 \\ 2/7 & -4/21 & -\lambda \\ 1/3 & 0 & 0 \end{vmatrix} + 1\begin{vmatrix} -\lambda & 0 & 4/27 \\ 2/7 & -4/21 & -\lambda \\ 1/3 & 0 & 0 \end{vmatrix} - \lambda\begin{vmatrix} -\lambda & 0 & 4/27 \\ 0 & -\lambda & 0 \\ 2/7 & -4/21 & -\lambda \end{vmatrix}$$

$$= \frac{2}{9}\cdot\frac{1}{3}\lambda^2 + \frac{16}{63\cdot 27} - \lambda\left(-\lambda^3 + \frac{8}{7\cdot 27}\lambda\right) = \lambda^4 + \frac{2}{63}\lambda^2 + \frac{16}{27\cdot 63} \approx \lambda^4 + 0.0317\lambda^2 + 0.0094$$

Mediante Matlab è immediato ottenere le radici di questo polinomio di quarto grado, che sono gli autovalori della matrice J: la singola istruzione

```
>> eigsJ = roots( [1, 0, 2/63, 0, 16/(63*27)] )
```

restituisce i quattro valori complessi (coniugati a coppie)

$$\lambda_1^{(J)} = \texttt{eigsJ[1]} \approx -0.2014 + 0.2375i \qquad \lambda_2^{(J)} = \texttt{eigsJ[2]} = \overline{\lambda_1}^{(J)} \approx -0.2014 - 0.2375i$$
$$\lambda_3^{(J)} = \texttt{eigsJ[3]} \approx 0.2014 - 0.2375i \qquad \lambda_4^{(J)} = \texttt{eigsJ[4]} = \overline{\lambda_3}^{(J)} \approx 0.2014 + 0.2375i$$

da cui si ottiene immediatamente

$$\rho(J) = \max_{i=1,\dots,4} \left|\lambda_i^{(J)}\right| = \left|0.2014 + 0.2375i\right| \approx 0.311425 < 1 \quad \Rightarrow \quad R_\infty(J) = -\ln\big(\rho(J)\big) \approx 1.1666$$

che conferma la previsione iniziale di convergenza del metodo di Jacobi. Si noti che, in questo caso, **tutti** gli autovalori di J hanno lo stesso modulo.

Determiniamo il polinomio caratteristico della matrice $\mathcal{G}$ (sviluppo del determinante con la regola di Laplace, a partire dalla porima colonna):

$$p_4^{\mathcal{G}}(\lambda) = \det\big(\mathcal{G} - \lambda I_4\big) = \begin{vmatrix} -\lambda & 0 & 4/27 & -2/9 \\ 0 & -\lambda & 0 & 1 \\ 0 & 0 & (8/189) - \lambda & 0 \\ 0 & 0 & 4/81 & (-2/27) - \lambda \end{vmatrix} = \lambda^2 \left(\left(\frac{8}{189} - \lambda \right) \left(-\frac{2}{27} - \lambda \right) \right)$$
$$= \lambda^2 \left(\lambda^2 + \frac{2}{63}\lambda - \frac{16}{81 \cdot 63} \right) = \lambda^4 + \frac{2}{63}\lambda^3 - \frac{16}{81 \cdot 63}\lambda^2 \approx \lambda^4 + 0.0317\lambda^3 + 0.0094\lambda^2$$

Si vede immediatamente che questo polinomio ha due radici nulle, ossia due autovalori di $\mathcal{G}$ sono nulli: $\lambda_1(\mathcal{G}) = 0 = \lambda_2(\mathcal{G})$. Mediante Matlab è immediato ottenere le due radici non nulle di questo polinomio di quarto grado, che sono gli altri due autovalori della matrice $\mathcal{G}$: la singola istruzione

```
>> roots( [1, 2/63, 16/(81*63)] )
```

restituisce i due valori complessi coniugati

$$\lambda_3^{(\mathcal{G})} \approx -0.0159 - 0.0957i \qquad \lambda_4^{(\mathcal{G})} = \overline{\lambda_3}^{(\mathcal{G})} \approx -0.0159 + 0.0957i$$

da cui si ottiene immediatamente

$$\rho(\mathcal{G}) = \max_{i=1,\dots,4} \left|\lambda_i^{(\mathcal{G})}\right| = \left|-0.0159 + 0.0957i\right| \approx 0.0969857 < 1 \quad \Rightarrow \quad R_\infty(\mathcal{G}) = -\ln\big(\rho(\mathcal{G})\big) \approx 2.3332$$

che conferma la previsione iniziale di convergenza del metodo di Gauss-Seidel. Da $\rho(\mathcal{G}) \approx 2.3332 > 1.1666 \approx \rho(J)$ discende subito che il metodo di Gauss-Seidel è asintoticamente più veloce del metodo di Jacobi. Per la precisione, il metodo di Gauss-Seidel è il doppio più veloce del metodo di Jacobi, perché $\rho(\mathcal{G})/\rho(J) \approx 2.3332/1.1666 = 2$.

Matlab

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

```
% Cognome Nome
% Matricola
%----------------------------
%  esercizio 4 - 08/02/2023
%----------------------------
close all; clear all; clc;
disp('Esercizio 4');

n = 4;
M = [9/4, 0, -1/3, 1/2; 0, -3/2, 0, 3/2; -1/2, 1/3, 7/4, 0; 1/2, 0, 0 -3/2];
z = cos([2 : (n+1)]') ./ [1 : n]';
d = diag(M);

if any( ~d ), error('presenza di elementi diagonali nulli in M'); end
invD = diag( 1 ./ d );
J    = -invD * (tril(M,-1) + triu(M,1));
cJ   = z ./ d;   % oppure cJ = invD*z, ma questo fa molte moltiplicazioni inutili
GS   = tril(M) \ [-triu(M,1) z];
cGS  = GS(:,end);   GS(:,end) = [];

x0 = [1; -1; 1; -1]; maxit = 50; tol = 1.0e-5;
```

```
% Soluzione con il metodo di Jacobi, sfruttando i calcoli precedenti
iterJ = 0; xJ = x0; stop = 0;
while ( ~stop )
    xJold = xJ; iterJ = iterJ + 1;
    xJ = J * xJ + cJ;
    stop = (   ( norm(xJ - xJold, inf) < tol * norm(xJold,inf) ) ...
             || ( iterJ >= maxit ) );
end
xJ, iterJ

% Metodo SOR con omega = 1.03
omega = 1.03;
[xSOR, iterSOR] = sor(M, z, x0, maxit, tol, omega)
xs = M \ z;
% Distanza relativa in norma infinito delle approssimazioni dalla soluzione
rJ   = norm( xJ   - xs, inf )  / norm( xs, inf );
rSOR = norm( xSOR - xs, inf )  / norm( xs, inf );

fprintf('\nDistanze relative in norma infinito dalla soluzione:');
fprintf('\n  Jacobi: %g,\n  SOR: %g\n\n', rJ, rSOR);
```

Eseguendo lo script si dovrebbero ottenere i seguenti risultati: il metodo di Jabobi converge in 14 iterazioni, il metodo SOR in 8 iterazioni. Al momento dell'arresto, le distanze relative dalla soluzione, in norma infinito, risultano circa $9.94 \cdot 10^{-7}$ per il metodo di Jacobi e circa $3.49 \cdot 10^{-6}$ per il metodo SOR con $\omega = 1.03$.

Nota per lo studente. Volendo, si può sfruttare facilmente questo sorgente per **controllare** i risultati ottenuti nella parte teorica (non per sostituirne i calcoli). A questo scopo, è sufficiente aggiungere le seguenti due righe di codice dopo il calcolo delle matrici `J` e `GS` e poi rimuoverle (o commentarle) prima di consegnare il sorgente:

```
eigsJ  = eig( J ),  rhoJ  = max(abs( eigsJ )),  RinfJ  = -log( rhoJ )
eigsGS = eig( GS ), rhoGS = max(abs( eigsGS )), RinfGS = -log( rhoGS )
```

Soluzione esercizio 5

Teoria

Notiamo innanzitutto che la funzione è ben definita nell'intervallo $[a, b] = [-4.5, 0.15]$ perchè il denominatore è sempre negativo: infatti, la funzione $g(x) = \mathrm{e}^{1.3x} - 2$ a denominatore è la traslata (lungo le ordinate) di una esponenziale positiva, quindi è una funzione monotona strettamente crescente su $[a, b]$, con massimo assunto nell'estremo destro $b = 0.15$ in cui vale $g(b) = \mathrm{e}^{1.3 \cdot .15} - 2 \approx -0.7847 < 0$.

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 $[-4.5, 0.15] = [-9/2, 3/20]$, utilizzando i punti equispaziati assegnati: $x_k = (31/20) \cdot k - (9/2)$, $k = 0, \ldots, 3$, e la tabella delle differenze divise per il polinomio nella forma di Newton:

x_k	$f(x_k)$	$f[x_k, x_{k+1}]$	$f[x_k, x_{k+1}, x_{k+2}]$	$f[x_0, x_1, x_2, x_3]$
$-9/2 = -4.50$	-0.4351			
$-59/20 = -2.95$	0.4284	0.5571		
$-7/5 = -1.40$	-0.4488	-0.5659	-0.3623	
$3/20 = 0.15$	1.0203	0.9478	0.4883	0.1829

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

$$\begin{aligned} p_3^{\mathrm{E}}(x) &\approx -0.4351 + 0.5571(x - x_0) + (-0.3623)(x - x_0)(x - x_1) + 0.1829(x - x_0)(x - x_1)(x - x_2) \\ &\approx -0.4351 + 0.5571(x + 4.5) - 0.3623(x + 4.5)(x + 2.95) + 0.1829(x + 4.5)(x + 2.95)(x + 1.4)\,. \end{aligned}$$

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

$$p_3^{\mathrm{E}}(x) \approx 0.1829x^3 + 1.2566x^2 + 2.1943x + 0.6623\,.$$

Siano t_j i nodi di Chebychev in $[-1, 1]$ e w_j i nodi di Chebychev in $[a, b] = [-4.5, 0.15]$, $j = 0, 1, 2, 3$, ottenuti usando la trasformazione lineare che mappa $[-1, 1]$ in $[a, b]$:

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

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

$$t_3 = \cos(\pi/8) = -\cos(7\pi/8) \approx 0.9239\,,$$

$$w_j = \frac{a+b}{2} + \frac{b-a}{2} t_j = -2.325 + 2.175 t_j \quad \forall j = 0,1,2,3 \quad \Rightarrow \quad \boldsymbol{w} \approx (-4.3230,\, -3.0647,\, -1.2853,\, -0.0270)^T\,.$$

I valori della funzione $f(x)$ nei punti w_j sono allora:

$$f(\boldsymbol{w}) \approx (-0.4942,\, 0.3557,\, -0.5143,\, 0.5252)^T\,.$$

Nota. La tabella delle differenze divise e i coefficienti del polinomio $p_3^{\mathrm{E}}(x)$ 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 $\omega_n(x) = (x-x_0)(x-x_1)\cdots(x-x_{n-1})$ 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 $p_3^{\mathrm{E}}(x)$ 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

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

Matlab

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

```
% Cognome Nome
% Matricola
%-----------------------------
%  esercizio 5 - 08/02/2023
%-----------------------------
close all; clear all; clc;
disp('Esercizio 5');

f = @(x)( cos(0.7*pi + 2*x) ./ (exp(1.3*x) - 2) );
a = -4.5; b = 0.15;
xx = linspace(a, b, 201)';
yy = f(xx);
fig1 = figure(1);
N = 12;
for n = 1 : N
    x = linspace(a, b, n+1)';
    y = f(x);
    [zz, ~] = polyLagrange(x, y, xx);
    ph1 = plot([xx(1); xx(end)], [0; 0], 'k-', ... % asse delle ascisse
               xx, yy, 'b-', ...                   % grafico della funzione
               xx, zz, 'r-', ...                   % grafico del polinomio interp.
```

```
                x, zeros(size(x)), 'ko', ...        % nodi di interpolazione
                x, y, 'ro' ...                      % valori del polinomio nei nodi
              );
    hold on; ph2 = plot([x, x]', [zeros(size(x)), y]', '--r'); hold off;
    th = title(sprintf('Funzione e polinomio interpolante di grado n = %d', n));
    xh = xlabel('Intervallo di interpolazione');
    yh = ylabel('Valori di f(x) e p_n(x)');
    lh = legend(ph1(2:3)', {'funzione f(x)','polinomio p_n(x)'}, ... % NON RICHIESTO
                           'Location', 'northwest');                 % NON RICHIESTO
    pause(2)
end
```

L'esecuzione dell'M-script genera esattamente N grafici, alcuni dei quali sono riportati come riferimento in fig. 5 (qui leggermente migliorati per esigenze di stampa).

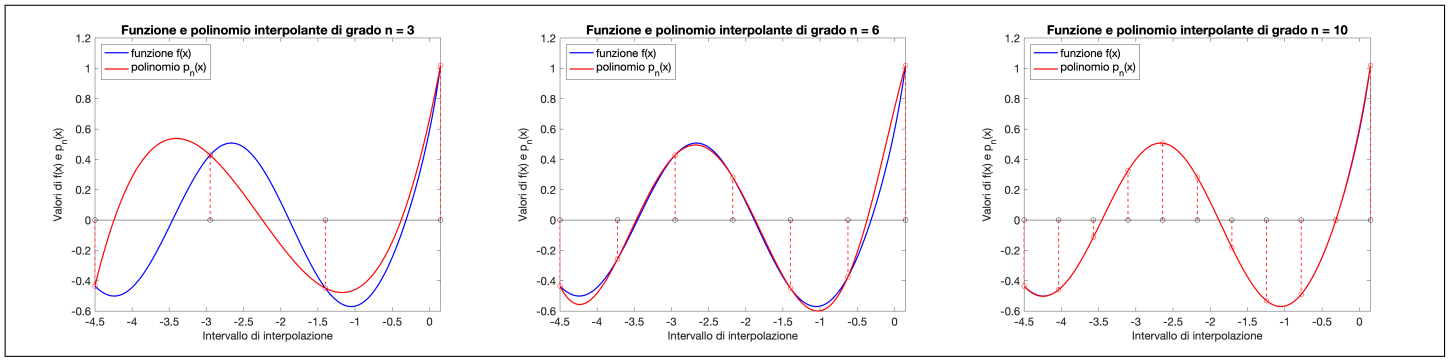


Figura 5: alcuni grafici generati dal ciclo dell'M-script `esercizio5.m`, per i gradi $n = 3, 6, 10$ del polinomio interpolante (figure leggermente migliorate per esigenze di stampa).

Soluzione esercizio 6

Teoria

Consideriamo dunque la funzione $f(x) = -x\sin(4x - \pi/3) + \mathrm{e}^{-x^3}$ nell'intervallo $[a, b] = [-1,\ 1.5]$. Gli $m = 6$ punti equispaziati sono

$$x_j = a + \frac{j}{m-1}(b-a), \quad j = 0, \ldots, m-1 \quad \Rightarrow \quad x_j = -1 + (j/5)\cdot 2.5 = 0.5\cdot j - 1, \quad j = 0, \ldots, 5$$
$$\Rightarrow \quad \boldsymbol{x} = \begin{pmatrix} -1, & -0.5, & 0, & 0.5, & 1.0, & 1.5 \end{pmatrix}^T = \begin{pmatrix} -1, & -1/2, & 0, & 1/2, & 1, & 3/2 \end{pmatrix}^T$$

da cui si ottiene subito il vettore dei valori della funzione in tali punti:

$$y_j = f(x_j), \quad j = 0, \ldots, 5 \quad \Rightarrow \quad \boldsymbol{y} \approx \begin{pmatrix} 3.6628, & 1.0860, & 1.0000, & 0.4750, & 0.1802, & 1.4911 \end{pmatrix}^T .$$

Costruiamo ora la matrice V di Vandermonde per il polinomio di grado $n = 3$ sui punti $\boldsymbol{x}$, la matrice $B = V^T V$ e il termine noto $\boldsymbol{z} = V^T \boldsymbol{y}$ del sistema delle equazioni normali:

$$V = \begin{pmatrix} -1 & 1 & -1 & 1 \\ -1/8 & 1/4 & -1/2 & 1 \\ 0 & 0 & 0 & 1 \\ 1/8 & 1/4 & 1/2 & 1 \\ 1 & 1 & 1 & 1 \\ 27/8 & 9/4 & 3/2 & 1 \end{pmatrix} = \begin{pmatrix} -1 & 1 & -1 & 1 \\ -0.125 & 0.25 & -0.5 & 1 \\ 0 & 0 & 0 & 1 \\ 0.125 & 0.25 & 0.5 & 1 \\ 1 & 1 & 1 & 1 \\ 3.375 & 2.25 & 1.5 & 1 \end{pmatrix}$$

$$B = V^T V = \begin{pmatrix} 859/64 & 243/32 & 115/16 & 27/8 \\ 243/32 & 115/16 & 27/8 & 19/4 \\ 115/16 & 27/8 & 19/4 & 3/2 \\ 27/8 & 19/4 & 3/2 & 6 \end{pmatrix} \approx \begin{pmatrix} 13.4220 & 7.5938 & 7.1875 & 3.3750 \\ 7.5938 & 7.1875 & 3.3750 & 4.7500 \\ 7.1875 & 3.3750 & 4.7500 & 1.5000 \\ 3.3750 & 4.7500 & 1.5000 & 6.0000 \end{pmatrix}$$

$$\boldsymbol{z} = V^T \boldsymbol{y} \approx \begin{pmatrix} 1.4735 \\ 7.5881 \\ -1.5515 \\ 7.8950 \end{pmatrix}$$

Il sistema lineare $B\boldsymbol{\alpha} = \boldsymbol{z}$ è un sistema pieno, di piccole dimensioni: potremmo certamente risolverlo efficientemente in Matlab utilizzando l'operatore "backslash" (\), che richiama il metodo di Gauss con pivoting parziale, ossia calcolare la soluzione con `B\z`. Tuttavia, dalla teoria sappiamo che, essendo i punti x_j tutti **distinti**, la matrice di Vandermonde V è di rango massimo e dunque la matrice B dei coefficienti del sistema delle equazioni normali è **definita positiva**: allora, il metodo numerico certamente migliore per risolvere il sistema è il metodo di **Cholesky**, sia dal punto di vista della complessità computazionale (pari a $O(n^3/6) = O(4^3/6) \approx O(10)$ operazioni floating point), che da quello della stabilità numerica (perché è incondizionatamente stabile). Pertanto, con la riga di istruzioni

```
>> [L, flag] = chol(B, 'lower'); alpha = utrisol( ltrisol(L, z) )
```

otteniamo la soluzione

$$\boldsymbol{\alpha}^* \approx \begin{pmatrix} -0.1791 \\ 1.4631 \\ -1.2776 \\ 0.5777 \end{pmatrix} \quad \Rightarrow \quad p_3^*(x) \approx -0.1791x^3 + 1.4631x^2 - 1.2776x + 0.5777\,.$$

Calcoliamo i valori del polinomio $p_3^*(x)$ di migliore approssimazione di $f(x)$ nei punti x_j nel senso dei minimi quadrati e i corrispondenti errori assoluti e relativi:

$$\widetilde{y}_j = p_3^*(x_j) \quad j = 0, \ldots, 5 \quad \Rightarrow \quad \widetilde{\boldsymbol{y}} \approx \begin{pmatrix} 3.4975 \\ 1.6047 \\ 0.5777 \\ 0.2823 \\ 0.5841 \\ 1.3488 \end{pmatrix} \quad \boldsymbol{e} = |\widetilde{\boldsymbol{y}} - \boldsymbol{y}| \approx \begin{pmatrix} 0.16523 \\ -0.51866 \\ 0.42230 \\ 0.19271 \\ -0.40386 \\ 0.14227 \end{pmatrix} \quad \boldsymbol{r} = \frac{|\widetilde{\boldsymbol{y}} - \boldsymbol{y}|}{|\boldsymbol{y}|} \approx \begin{pmatrix} 0.0451 \\ -0.4776 \\ 0.4223 \\ 0.4057 \\ -2.2411 \\ 0.0954 \end{pmatrix}$$

Matlab

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

```
% Cognome Nome
% Matricola
%-----------------------------
%  esercizio 6 - 08/02/2023
%-----------------------------
close all; clear all; clc;
disp('Esercizio 6');

rng(17);
N = 15; % numero di punti di osservazione per costruire il polinomio approssimante
n =  6; % grado del polinomio di migliore approssimazione ai minimi quadrati
nn = 2001; a = -1 ; b = 1.5;
f = @(x)( -x.*sin(4*x - pi/3) + exp(-x.^3) );

if (N <= n)
    error( sprintf(...
      'Numero N = %d di punti non mggiore del grado n = %d del polinomio', ...
       N, n));
end

x = sort(rand(N,1)*(b - a) + a); y = f(x);

% calcolo esplicito della matrice di Vandermonde
V = x.^(n:-1:0);
% parte (i): calcolo dei coefficienti risolvendo il sist. delle eq. normali
B = V'*V; z = V'*y;  % possibile soluz. (INVECE della riga seguente): alpha1 = B\z
[L, flag] = chol(B, 'lower'); alpha1 = utrisol( L', ltrisol(L, z) );
% parte (ii): calcolo dei coefficienti risolvendo il sist. sovradeterminato
alpha2 = V \ y;
% parte (iii): calcolo dei coefficienti mediante la function polyfit
alpha3 = polyfit(x, y, n)';

% confronto fra le soluzioni
differenza1e2 = norm(alpha1 - alpha2, inf)
differenza1e3 = norm(alpha1 - alpha3, inf)
differenza2e3 = norm(alpha2 - alpha3, inf)
```

```
% costruzione della rappresentazione grafica
xx = linspace(a, b, nn)'; yy = f(xx);
ph = plot(xx,yy);
hold on;
ph1 = plot([a;b],[0;0],'-k',x,zeros(size(x)),'+k',x,y,'.r','MarkerSize',20);
ph2 = plot(xx, polyval(alpha2, xx), '-m');
hold off
alh(1) = xlabel('Intervallo delle ascisse');
alh(2) = ylabel('Valori della funzione e del polinomio');
tlh = title(sprintf('Polinomio ai minimi quadrati di grado %d per f(x)',n));
lh  = legend([ph; ph2], ...                                          % NON RICHIESTO
             {'funzione f(x)', 'polinomio approssimante'}, ... % NON RICHIESTO
             'Location', 'northeast');                               % NON RICHIESTO
```

Eseguendo lo script si ottiene un output simile a quello della figura 6 (per esigenze di stampa, è stata aggiunta la legenda e leggermente migliorata la visibilità delle curve). I valori del confronto delle tre soluzioni calcolate dovrebbe risultare simile al seguente:

$$\|\widetilde{\boldsymbol{\alpha}}_1^* - \widetilde{\boldsymbol{\alpha}}_2^*\|_\infty = 3.3893 \cdot 10^{-12}, \quad \|\widetilde{\boldsymbol{\alpha}}_1^* - \widetilde{\boldsymbol{\alpha}}_3^*\|_\infty = 3.4062 \cdot 10^{-12}, \quad \|\widetilde{\boldsymbol{\alpha}}_2^* - \widetilde{\boldsymbol{\alpha}}_3^*\|_\infty = 1.6875 \cdot 10^{-14}.$$

Nota. In questo caso, per la costruzione esplicita della matrice di Vandermonde **non conviene usare l'istruzione predefinita** `vander` di Matlab: essa, infatti, restituisce una matrice quadrata di dimensioni pari al numero dei punti, dunque risolvendo il sistema lineare associato si avrebbe un polinomio approssimante di grado $N - 1 = 5$, quindi nel caso in esame di grado più alto del grado $n = 3$ richiesto.

Un modo diverso da `x.^([n:-1:0])` di calcolare la stessa matrice V è il seguente:

```
V = ones(N, n+1);
for k = n : -1 : 1
    V(:,k) = x .* V(:,k+1);
end
```

In questo modo Matlab esegue il numero minimo di operazioni aritmetiche necessarie, perché per ciascuna potenza k-esima di x_j si parte dalla potenza $(k-1)$-esima e non dall'inizio, ossia, $\forall j = 1, \ldots, N$, per ciascun $k = 1, \ldots, n$ si calcola $x_j^k = x_j \cdot x_j^{k-1}$ invece di $x_j^k = \underbrace{x_j \cdots x_j}_{k \text{ volte}}$.

Tuttavia, questa seconda implementazione utilizza necessariamente un ciclo `for` esterno: sebbene l'istruzione all'interno del ciclo sia vettoriale, è lecito attendersi che l'overhead di esecuzione del ciclo sia penalizzante rispetto alla sola sintassi vettoriale dell'unica istruzione `x.^([n:-1:0])`, soprattutto per piccole dimensioni.

Si noti comunque che, per le dimensioni in gioco nel caso in esame, i tempi di esecuzione sono talmente piccoli che le differenze sono totalmente trascurabili.

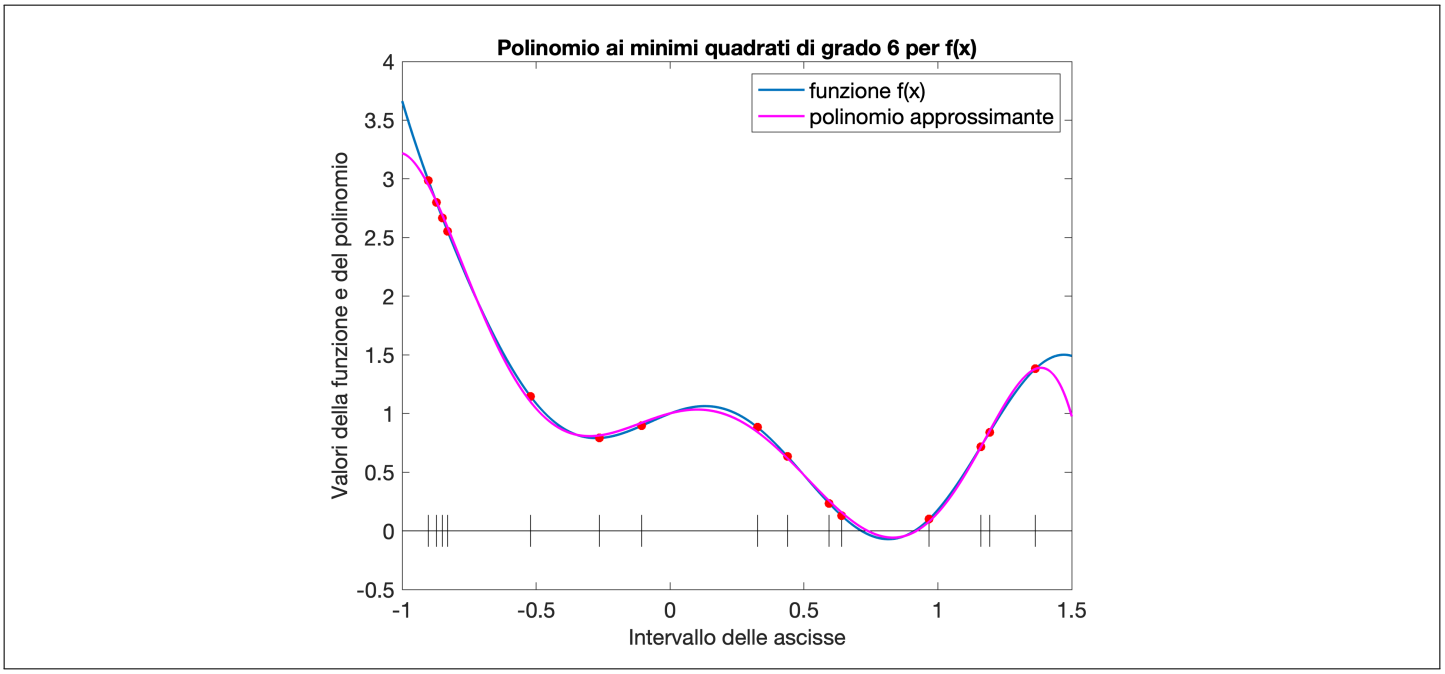


Figura 6: output grafico dell'M-script `Esercizio6.m` (l'immagine è stata leggermente migliorata per esigenze di stampa).