UNIVERSITÀ DEGLI STUDI DI ROMA TOR VERGATA

MACROAREA DI SCIENZE MATEMATICHE, FISICHE E NATURALI

CORSO DI LAUREA IN INFORMATICA

A.A. 2020/2021

Progetto MATLAB

PROFESSORE

Carlo Garoni

STUDENTI

Gabriele Benedetti
Giulia Pascale

Indice

Introduzione

Verranno inizialmente risolti dei problemi e successivamente, in appendice, spiegati tutti le funzioni utilizzate nel dettaglio.

Per la stesura del sequente elaborato abbiamo utilizzato LaTeX, per la risoluzione dei problemi MATLAB.

Capitolo 1

Problemi assegnati

1.1 Problema 1

Si consideri la funzione $\sqrt{x}$

(a) Sia *p(x)* il polinomio d'interpolazione di $\sqrt{x}$ sui nodi
$x_0 = 0, \quad x_1 = \frac{1}{64}, \quad x_2 = \frac{4}{64}, \quad x_3 = \frac{9}{64}, \quad x_4 = \frac{16}{64},$
$x_5 = \frac{36}{64}, \quad x_6 = \frac{36}{64}, \quad x_7 = \frac{49}{64}, \quad x_8 = 1$

Calcolare il vettore (colonna)

$$[p(\zeta_1) - \sqrt{\zeta}_1 \qquad p(\zeta_2) - \sqrt{\zeta}_2 \qquad \cdots \qquad p(\zeta_{20}) - \sqrt{\zeta}_{20}]^T$$

dove $\zeta_i = \frac{i-1}{20}$ per $i = 1, ..21$, e osservare in che modo varia la differenza $p(\zeta_i) - \sqrt{\zeta}_i$ al variare di i da 1 a 21.

(b) Tracciare il grafico di $\sqrt{x}$ e di $p(x)$ sull'intervallo [0,1], ponendo i due grafici su un'unica figura e inserendo una legenda che ci dica qual è la funzione $\sqrt{x}$ e qual è il polinomio $p(x)$.

1.1.1 Risoluzione problema 1

```
1 -  x = [0, 1/64, 4/64, 9/64, 16/64, 25/64, 36/64, 49/64, 1];
2
3 -  y = sqrt(x);
4 -  zeta = (0:20)/20;
5
6 -  p = Esercizio1_11(x,y,zeta);
7
8    %punto a: calcolo il vettore colonna k
9 -  k = p' - zeta';
10 - disp(k);
11
12   %punto b: traccio i grafici
13 - x1 = linspace(0,1,1000);
14 - y1 = sqrt(x1);
15 - p1 = Esercizio1_11(x,y,x1);
16 - plot(x1,p1,'linewidth',5)
17 - hold on
18 - plot(x1,y1,'linewidth',5);
19 - legend('p(x)','sqrt(x)')
20 - hold off;
```

Nella prima parte del codice, viene inizializzato il vettore x contenente i nodi $x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8$, si calcola il vettore y contenente i valori della funzione calcolata sul vettore x.

Quindi viene calcolato un vettore *zeta* contenente i valori

$[p(\zeta_1) - \sqrt{\zeta_1} \qquad p(\zeta_2) - \sqrt{\zeta_2} \qquad \dots \qquad p(\zeta_{20}) - \sqrt{\zeta_{20}}]^T$ dove $\zeta_i = \dfrac{i-1}{20}$ per $i = 1, ..21$.

Infine si calcola p, il polinomio d'interpolazione sui nodi contenuti nel vettore x, tramite la funzione $Esercizio1-11$. Quello che si ottiene è un vettore p contenente i valori del polinomio calcolato.

Risoluzione punto (a) - Righe (8 - 10)

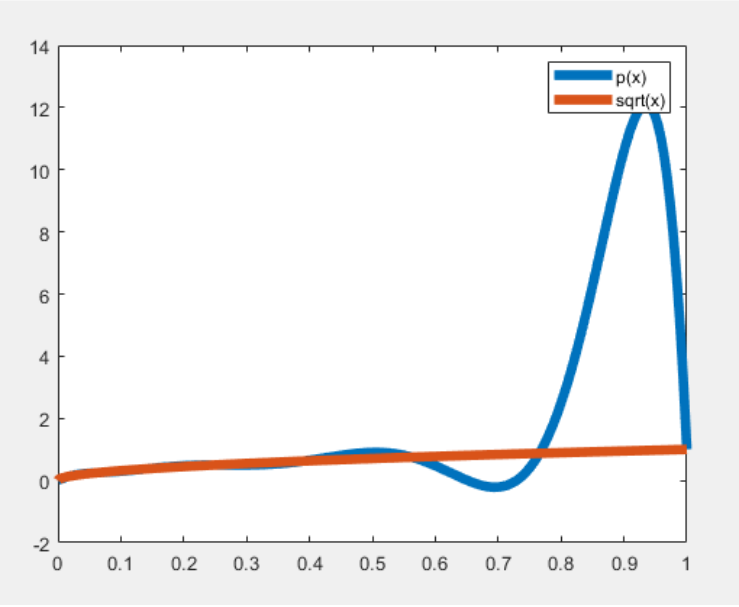
Viene calcolato il vettore colonna k, per osservare in che modo varia la differenza $p(\zeta_i) - \sqrt{\zeta_i}$ al variare di i da 1 a 21. Essendo k un vettore colonna, servono le trasposte p' e $zeta'$ dei relativi vettori p e $zeta$. I risultati verranno poi visualizzati nella

command Window di MATLAB come segue:

```
>> Problema1
         0
    0.1830
    0.1996
    0.2436
    0.2733
    0.2500
    0.2009
    0.1888
    0.2515
    0.3575
    0.4031
    0.2618
   -0.1241
   -0.6376
   -0.9112
   -0.3457
    1.6945
    5.4096
    9.6974
   10.7562
   -0.0000
```

Risoluzione punto (b) - Righe (12 - 20)

Vengono visualizzati i grafici di $\sqrt{x}$ e di $p(x)$ sull'intervallo [0,1].



1.2 Problema 2

Si consideri la funzione

$$f(x) = e^x$$

Per ogni intero $n \geq 1$ indichiamo con I_n la formula dei trapezi di ordine n per approssimare

$$I = \int_0^1 f(x)dx = 1,182818284590...$$

(a) Per ogni fissato $\epsilon > 0$ determinare un $n = n(\epsilon)$ tale che $|I - I_n| \leq \epsilon$

(b) Costruire una tabella che riporti vicino ad ogni $\epsilon \in 10^{-1}, 10^{-2}, ..., 10^{-10}$:

- il numero $n(\epsilon)$;
- il valore I_n per $n = n(\epsilon)$;
- il valore esatto I (in modo da confrontarlo con In);
- l'errore $|I - I_n|$ (che deve essere $\leq \epsilon$).

(c) Calcolare le approssimazioni di I ottenute con le formule dei trapezi I_2, I_4, I_8, I_16 e confrontarle con il valore esatto I.

(d) Sia $p(x)$ il polinomio d'interpolazione dei valori I_2, I_4, I_8, I_{16} sui nodi $h_2^2, \quad h_4^2, \quad h_8^2, \quad h_{16}^2$, dove $h_2 = \frac{1}{2}, h_4 = \frac{1}{4}, h_8 = \frac{1}{8}, \quad h_{16} = \frac{1}{16}$,

1.2.1 Risoluzione problema 2

Il punto (a) viene risolto senza utilizzare MATLAB, ma utilizzando $|\int_a^b f(x)dx - I_n| = \left| -\frac{(b-a)f''(\eta)}{12}h^2 \right| = \left|\frac{e^x}{12n^2}\right| \leq \epsilon$. Di conseguenza si ottiene che $n \geq \sqrt{\frac{e}{12\epsilon}}$. Si inserisce questo valore nel codice e si procede con la risoluzione degli altri punti.

```
format long;
f = @(x)exp(x);
x = sym('x');
integrale = integral(f,0,1); %integrale = int(f,x,0,1)

%punto b
fprintf('\n');
disp('punto b:');
fprintf('\n');

e = 10.^(-(1:10));
n = ceil(sqrt(exp(1)./(12*e)));

vettoreIn = zeros(1,10);
for i = 1:10
    vettoreIn(i) = integrazioneNumerica(0,1,n(i),f);
end
differenzaB = abs(integrale - vettoreIn);
Tb = table(e',n',vettoreIn',differenzaB','VariableNames',{'Soglia e' 'n(e)' 'In' '|I-In|'});
Tb.Properties.VariableNames;
disp(Tb);

%punto c
fprintf('\n');
disp('punto c:');
fprintf('\n');

i = [2,4,8,16];
for k = 1:4
    r(k) = integrazioneNumerica(0,1,i(k),f);
end
differenzaC = abs(integrale - r);
Tc = table(r',differenzaC','VariableNames',{'Nuove approssimazioni' 'differenza'});
Tc.Properties.VariableNames;
disp(Tc);

%punto d
fprintf('\n');
disp('punto d:');
fprintf('\n');

h = [(1/2)^2,(1/4)^2,(1/8)^2,(1/16)^2];
p = interpolazione(h,r,0)
fprintf('\n');
disp(['differenza = ', num2str(abs(p-integrale))]);
```

```
>> Problema2

punto b:

    Soglia e    n(e)          In                |I-In|
    ________    ____    ______________    ____________________

        0.1        2    1.75393109246483    0.0356492640057804
       0.01        5    1.72400561978279    0.0057237913237429
      0.001       16    1.71884112857999    0.000559300120949402
     0.0001       48    1.71834397651311    6.21480540687891e-05
      1e-05      151    1.71828810844886    6.27998981217459e-06
      1e-06      476    1.71828246043305    6.31974003795222e-07
      1e-07     1506    1.71828189159303    6.31339851508983e-08
      1e-08     4760    1.71828183477879    6.31974028664217e-09
      1e-09    15051    1.71828182909114    6.32093488661667e-10
      1e-10    47595    1.71828182852225    6.32081054163791e-11

punto c:

    Nuove approssimazioni          differenza
    _____________________    ____________________

      1.75393109246483        0.0356492640057804
      1.72722190455752        0.00894007609847147
       1.7205185921643        0.00223676370525672
      1.71884112857999       0.000559300120949402

punto d:

p =
   1.718281828460389

differenza = 1.3438e-12
```

1.3 Problema 3

Si consideri il sistema lineare Ax = b, dove

$$A = \begin{bmatrix} 5 & 1 & 1 \\ -1 & 7 & 1 \\ 0 & 1 & -3 \end{bmatrix}, b = \begin{bmatrix} 13 \\ 16 \\ -7 \end{bmatrix}$$

(a) Si calcoli la soluzione x del sistema dato con MATLAB.

(b) La matrice A è a diagonale dominante in senso stretto per cui il metodo di Jacobi è convergente ossia partendo da un qualsiasi vettore d'innesco $x^{(0)}$ la successione prodotta dal metodo di Jacobi converge (componente per componente) alla soluzione x del sistema dato. Calcolare le prime 10 iterazioni $x^{(1)}, ..., x^{(10)}$ del metodo di Jacobi partendo dal vettore nullo $x^{(0)} = [0, 0, 0]^T$ e confrontarle con la soluzione esatta x ponendo iterazioni e soluzione esatta in un'unica matrice S di dimensioni 3x12 le cui colonne sono nell'ordine $x^{(0)}, x^{(1)}, ..., x^{(10)}, x$.

(c) Consideriamo il metodo di Jacobi per risolvere il sistema dato. Conveniamo d'innescare il metodo di Jacobi con il vettore nullo $x^{(0)} = [0, 0, 0]^T$. Costruire una tabella che riporti vicino ad ogni $\epsilon \in \ 10^{-1}, 10^{-2}, ..., 10^{-10}$:

- il numero d'iterazioni K_ϵ necessarie al metodo di Jacobi per convergere entro la precisione ϵ;
- la soluzione approssimata x_ϵ calcolata dal metodo di Jacobi;
- la soluzione esatta x (in modo da confrontarla con la soluzione approssimata x_ϵ);
- la norma ∞ dell'errore $||x - x_\epsilon||_\infty$

1.3.1 Risoluzione problema 3

```
%punto a:
A = [5,1,2;-1,7,1;0,1,-3];
b = [13;16;-7];
x = A\b;

%punto b:
D=diag(diag(A));
xk = [0;0;0];
S = zeros(3,12);
S(:,1) = xk;

for k = 2:11

    xk = xk + D\(b - A*xk);
    S(:,k) = xk;

end

S(:,12) = x;
disp('Matrice S del punto B:');
disp(S);

%punto c:
e = 10.^(-(1:10));
x0 = [0;0;0];
Matrice2 = zeros(3,10);
MatriceX = zeros(10,3);

for i = 1:10
    [Matrice2(:,i),k(i),norma2] = jacobi(A,b,x0,e(i),100);
    normaInf(i) = norm(x-Matrice2(:,i),Inf);
    MatriceX(i,:) = x;
end

T = table(k',Matrice2',MatriceX,normaInf','VariableNames',{'K' 'x_e' 'x' 'Norma Infinito |x-x(e)|'});
T.Properties.VariableNames;
disp(T);
```

Nel punto (a) viene risolto il sistema. Nel punto (b) vengono calcolate le prime 10 iterazioni $x^{(1)}, ..., x^{(10)}$ del metodo di Jacobi partendo dal vettore nullo $x^{(0)} = [0, 0, 0]^T$, quindi viene costruita una matrice S 3x12 in cui l'ordine dei valori Ã¨: $x^{(0)}, x^{(1)}, ..., x^{(10)}, x$.

Infine il codice si occupa di risolvere il punto (c) e costruire una tabella che ne riporti i risultati nella Command Window.

```
>> Problema3
Matrice S del punto B:
  Columns 1 through 3
                  0   2.600000000000000   1.209523809523810
                  0   2.285714285714286   2.323809523809524
                  0   2.333333333333333   3.095238095238095
  Columns 4 through 6
   0.897142857142857   0.953560090702948   1.003845804988662
   2.016326530612245   1.969886621315193   1.992588273404600
   3.107936507936508   3.005442176870748   2.989962207105064
  Columns 7 through 9
   1.005497462477054   1.000591555987474   0.999507892287679
   2.001983371126228   2.001138291144122   1.999990061754105
   2.997529424468200   3.000661123708743   3.000379430381374
  Columns 10 through 12
   0.999850215496630   1.000026226187844   1.000000000000000
   1.999875494558043   1.999979075463609   2.000000000000000
   2.999996687251368   2.999958498186015   3.000000000000000
```

K	x_e			x			Norma Infinito |x-x(e)|
4	0.897142857142857	2.01632653061224	3.10793650793651	1	2	3	0.107936507936508
6	1.00384580498866	1.9925882734046	2.98996220710506	1	2	3	0.010037792894936
8	1.00059155598747	2.00113829114412	3.00066112370874	1	2	3	0.00113829114412178
10	0.99985021549663	1.99987549455804	2.99999668725137	1	2	3	0.000149784503370443
12	1.00002078563287	2.00000967542883	2.99999302515454	1	2	3	2.07856328726663e-05
14	0.999997916786298	1.99999966138707	3.00000132192754	1	2	3	2.08321370165354e-06
16	1.00000014243814	1.99999995026036	2.99999983785042	1	2	3	1.62149583093907e-07
18	0.999999997929445	2.00000001305538	3.00000001450418	1	2	3	1.45041774146648e-08
20	0.999999998732847	1.99999999817649	2.99999999921073	1	2	3	1.82350579081003e-09
22	1.00000000025679	2.00000000018404	2.99999999997724	1	2	3	2.56787702213046e-10

Appendice
Funzioni utilizzate

Le seguenti funzioni sono state utilizzate per risolvere i problemi nei primi capitoli. Segue spiegazione delle funzioni e dei relativi codici.

Prima di tutto occorre dire che ogni codice è strutturato come segue:

1. Spiegazione della funzione e degli input richiesti. Sono presenti esempi di input per facilitare la comprensione al lettore. Questa parte è consultabile digitando *help* *nomefunzione* sulla Command Window di MATLAB.

2. Il codice vero e proprio che viene eseguito quando la funzione viene invocata.

.1 Esercizio 1.11

```
1   function p = Esercizio1_11(x,y,t)
2
3   %Esercizio1_11(x,y,t)
4   %Input:
5   %x: vettore di valori distinti sull'asse delle ascisse,
6   %y: vettore di valori sull'asse delle ordinate,
7   %t: vettore di valori in cui si vuole valutare il polinomio;
8   %nota: x e y devono avere la stessa lunghezza.
9   %Output:
10  %p: vettore contenente i valori del polinomio di interpolazione calcolato
11  %nei punti contenuti nel vettore t.
12  %Esempio input: Valutarlo in un punto: Esercizio1_11([-1,0,2],[0,1,5],1)
13  %input: Valutarlo in due punti: Esercizio1_11([-1,0,2],[0,1,5],[1,3])
14
15
16  for i = 1:length(t)
17      p(i) = interpolazione(x,y,t(i));
18  end
19  end
```

$Esercizio1-11(x,y,t)$ è una funzione che restituisce un vettore p contenente i valori del polinomio di interpolazione calcolato nei punti contenuti nel vettore t. Questa prima parte di codice si occupa di prendere in input tre vettori di valori. In particolare x è un vettore di valori distinti sull'asse delle ascisse, y è un vettore di valori sull'asse delle ordinate, t è il vettore di valori di cui si vuole valutare il polinomio; x e y devono avere la stessa lunghezza.

.2 interpolazione

```
function p = interpolazione(x,y,t)
%interpolazione(x, y, t)
%Input:
%x: vettore di valori distinti sull'asse delle ascisse,
%y: vettore di valori sull'asse delle ordinate,
%t punto in cui si vuole valutare il polinomio;
%Nota: x e y devono avere la stessa lunghezza.
%Output:
%p: valore del polinomio d'interpolazione dei punti(x(i),y(i)) nel punto t
%inserito in input
%Esempio input: Valutarlo in un punto: interpolazione([-1,0,2],[0,1,5],1)

    A = differenzeDivise(x,y);  %calcolo matrice differenze divise
    d = diag(A);
    h = zeros(1,length(d));
    h(end) = d(end);
    for i = length(d)-1:-1:1
        h(i) = d(i) + (t-x(i))*h(i+1);
        p = h(i);
    end
end
```

$interpolazione(x, y, t)$ è una funzione che restituisce il valore del polinomio di interpolazione dei punti (x_i, y_i) nel punto t inserito in input.

Nella prima parte di codice è presente la spiegazione della funzione $interpolazione(x, y, z)$ e degli input da inserire. In particolare x è un vettore di valori distinti sull'asse delle ascisse, y è un vettore di valori sull'asse delle ordinate, t è un punto in cui si vuole valutare il polinomio; x e y devono avere la stessa lunghezza.

.3 differenze divise

```
1   function A = differenzeDivise(x,y)
2   %differenzeDivise(x,y) restituisce una matrice triangolare inferiore
3   %contenente i valori delle differenze divise.
4   %Input:
5   %x: lista di valori distinti sull'asse delle x,
6   %y: lista di valori sull'asse delle y;
7   %Nota: x e y devono avere la stessa dimensione
8   %Esempio input: differenzeDivise([-1,0,2],[0,1,5])
9
10  n = length(x);
11  A = zeros(n);
12  for i = 1:n
13      for j = 1:n
14          if j == 1
15              A(i,j) = y(i);
16          end
17          if (j ~= 1 && j <= i && i>1)
18              A(i,j) = (A(i,j-1)-A(j-1,j-1))/(x(i)-x(j-1));
19          end
20      end
21  end
22  end
```

$differenzeDivise(x, y)$ è una funzione che restituisce una matrice triangolare inferiore contenente i valori delle differenze divise.

.4 integrazioneNumerica

```
1   function r = integrazioneNumerica(a,b,n,f)
2   %integrazioneNumerica(a,b,n,f)
3   %Input:
4   %a, b: valori d'inizio e fine dell'intervallo
5   %n numeri di sottointervalli
6   %f: funzione
7   %Condizioni: a < b, n >= 1,
8   %Output:
9   %valore approssimato dell'integrale di una funzione f calcolato
10  %nell'intervallo [a,b] suddiviso in n sottointervalli.
11  %esempio input: integrazioneNumerica(0,1,6, @(x) 1./log(x+2))
12
13  h = (b-a)/n;
14  x = linspace(a,b,n+1); %x = a + (0:n)*h;
15  r = h * ( (f(a) + f(b)) / 2 + sum(f(x(2:n))) );
16
17  end
```

$integrazioneNumerica(a, b, n, f)$ è una funzione che restituisce il valore approssimato dell'integrale di una funzione $f(x)$ calcolato nell'intervallo $[a, b]$ suddiviso in n sottointervalli. In particolare è specificato che, essendo un intervallo che va da a fino a b, a deve essere un valore minore rispetto a b ed inoltre n deve essere un numero maggiore di 1, altrimenti non si può dividere l'intervallo in sottointervalli.

.5 jacobi

```
function[xk,k,norma2] = jacobi(A,b,innesco,e,nmax)

%jacobi(A,b,innesco,e,nmax)
%Input:
%A: matrice,
%b: vettore colonna,
%innesco: vettore colonna da cui partire
%e: valore soglia di precisione,
%nmax: numero massimo d'iterazioni consentite
%Output:
%xk: primo vettore che, calcolato con il metodo di Jacobi, soddisfa la condizione
%di arresto del residuo norma(r) <= e*norma2(b)
%Esempio input: jacobi([1 -3 -1; 0 9 0; 0 0 1], [-1 ; 9; 1], [1; 0; 0], 1, 3)

v = diag(A);
D = diag(v);

%calcolo le iterazioni
xk = innesco;
norma2 = norm(b - A*xk ,2);
    for k = 1:nmax
        if norma2 <= e*norm(b,2)
            break
        end
        xk = xk + D\(b - A*xk);
        norma2 = norm(b - A*xk ,2);
    end
end
```

$jacobi(A, b, innesco, e, nmax)$ è una funzione che restituisce il primo vettore xk che, calcolato dal metodo di Jacobi, soddisfa la condizione di arresto del residuo $||r^{(K)}||_2 \leq \epsilon||b||_2$