Elaborato di

Calcolo Numerico

Anno Accademico 2021/2022

John *Smith* - 1234567 - *john.smith@stud.unifi.it*
Nice *Guy* - 9876543 - *nice.guy@stud.unifi.it*

June 25, 2022

Contents

List of Figures

List of Tables

1 Capitolo 1

1.1 Esercizio 1

Verificare che,

$$\frac{-f(x+2h)+8f(x+h)-8f(x-h)+f(x-2h)+O(h^4)}{12h}=f'(x)+O(h^4)$$

Soluzione:

Per verificare la equivalenza ci servirà applicare sviluppo di Taylor per tutti i funzioni. La funzione di Taylor definita cosi:

$$f(x)=f(x_0)+f'(x_0)(x-x_0)+\frac{f''(x_0)}{2!}(x-x_0)^2+...+\frac{f^{(n)}(x_0)}{n!}(x-x_0)^n+O((x-x_0)^n)$$

Calcoliamo le sviluppi di Taylor per tutti funzioni che sono presenti in formula da verificare. A variablie x_0 noi assumiamo il valore x, quindi $x_0=x$. Allora:

$$f(x+2h)=f(x)+f'(x)(x+2h-x)+O(x+2h-x)=f(x)+2hf'(x)+O(h)$$

$$f(x+h)=f(x)+f'(x)(x+h-x)+O(x+h-x)=f(x)+hf'(x)+O(h)$$

$$f(x-h)=f(x)+f'(x)(x-h-x)+O(x-h-x)=f(x)-hf'(x)+O(h)$$

$$f(x-2h)=f(x)+f'(x)(x-2h-x)+O(x-2h-x)=f(x)-2hf'(x)+O(h)$$

Se mettere tutto insieme otteniamo:

$$\frac{-(f(x)+2hf'(x))+8(f(x)+hf'(x))-8(f(x)-hf'(x))+(f(x)-2hf'(x))+O(h^4)}{12h}=$$

$$=\frac{\cancel{-f(x)}-2hf'(x)+8\cancel{f(x)}+8hf'(x)-\cancel{8f(x)}+8hf'(x)+\cancel{f(x)}-2hf'(x)+O(h^4)}{12h}=$$

$$=\frac{\cancel{12h}f'(x)+O(h^4)}{\cancel{12h}}=f'(x)+O(h^4)$$

1.2 Esercizio 2

Calcolare, motivandone i passaggi, la precisione di macchina della doppia precisione dello standard IEEE. Confrontare questa quantità con quanto ritornato dalla variabile eps di Matlab, commentando a riguardo.
Soluzione:

1.3 Esercizio 3

Eseguire il seguente script Matlab:

```
format long e
(1+(1e−14−1))*1e14
```

Spiegare i risultati ottenuti.
Soluzione:

Quando si esegue $a-a+b$ il risultato è 100 mentre quando si esegue $a+b-a$ si ottiene 0. La differenza dei risultati è dovuta al fenomeno della cancellazione numerica:

2 Capitolo 2

2.1 Eserczio 4

Scrivere una function Matlab, $radice(x)$ che, avendo in ingresso un numero x non negativo, ne calcoli la radice quadrata utilizzando solo operazioni algebriche elementari, con la massima precisione possibile. Confrontare con la function *sqrt* di Matlab per 20 valori di s, equispaziati logaritmicamente nell'intervallo $[1e-10, 1e10]$, evidenziando che si 'e ottenuta la massima precisione possibile.
Soluzione:

```
function answer = radice(x)
    % answer = radice(x)
    % This method is called Babilons method
    % It gets some value in input and apply the Babilons method
    % until I needed precision, but we need maximum precision possible,
    % so I apply this method until I get two equal values
    % one before another and this will mean this method arrived at a point,
    % in which value not changes, so it is a maximum precision.
    % INPUT:  x      - Is a value of which we want to get square root
    % OUTPUT: answer - This is the square root of the input
    format long e;

    if x < 0
        error('Wrong value in input! Should be not less then zero!');
    end

    answer = x / 2;
    guess = x;

    while not(answer == guess)
        guess = answer;
        answer = (guess + (x / guess)) / 2;
    end

end

for i = 1:0.5:10
    s = 1e-10 + 1e10 * log10(i);
    custom = radice(s);
    native = sqrt(s);

    if abs(custom - native) >= eps(abs(native))

        if abs(custom - native) > eps(abs(native))
            diff = abs(custom - native);
            fprintf([ ...
                    'result is not equal for %.' int2str(abs(int16(log10(eps(s)))) + 1) ...
                    'd, with values custom=%.' int2str(abs(int16(log10(eps(custom)))) + 20) ...
                    'd and native=%.' int2str(abs(int16(log10(eps(native)))) + 20) ...
                    'd, with diff=%.' int2str(abs(int16(log10(eps(diff)))) + 20) 'd\n' ...
                ], s, custom, native, diff);
        elseif abs(custom - native) == eps(abs(native))
            fprintf([ ...
                    'Difference betwen this numbers are equal to exponent ' ...
                    'for this numbers, and equals to %.30d\n'...
                ], eps(abs(native)));
```

```
        end

    end

    fprintf([ ...
            'custom=%.' int2str(abs(int16(log10(eps(custom)))) + 1) ...
            'd\nnative=%.' int2str(abs(int16(log10(eps(native)))) + 1) ...
            'd\n\n'
        ], custom, native);

end
```

2.2 Esercizio 5

Scrivere function Matlab distinte che implementino efficientemente i seguenti metodi per la ricerca degli zeri di una funzione $f(x)$:

- il metodo di Newton;
- il metodo delle secanti;
- il metodo di Steffensen:

$$x_{n+1} = x_n - \frac{f(x_n)^2}{f(x_n + f(x_n)) - f(x_n)}, \qquad \text{n=0,1,...} \tag{1}$$

Per tutti i metodi, utilizzare come criterio di arresto

$$|x_{n+1} - x_n| \leq tol * (1 + |x_n|)$$

essendo *tol* una opportuna tolleranza specificata in ingresso. Curare particolarmente la robustezza del codice.
Soluzione:

- Metodo di Newton

```
function [x, i] = newton(f, f1, x0, tolx, maxit)
    % [x, i] = newton(f, f1, x0, tolx, maxit)
    % Newton's method, used to calculate a root of the equation f(x)=0
    % Input:  f     — function that we can derive
    %         f1    — derived function of the 'f'
    %         x0    — initial approximation
    %         tolx  — tolerance
    %         maxit — maximum number of iterations (default = 100)
    % Output: x     — the approximation of the root function
    %         i     — number of iterations
    % CHECK THIS: bisezione, corde, secanti, aitken, newtonmod

    format long e;

    if nargin < 4
        error('Number of arguments must be at least 4!');
    elseif nargin == 4
        maxit = 100;
    end

    if tolx < eps
        error('Tolerance cannot be checked!');
    end

```

```
    x = x0;

    for i = 1:maxit
        fx = feval(f, x);
        f1x = feval(f1, x);
        x = x - fx / f1x;

        if abs(x - x0) <= tolx * (1 + abs(x0))
            break;
        end

        x0 = x;
    end

    if abs(x - x0) > tolx * (1 + abs(x0))
        error('The method does not converge!');
    end

end
```

- Metodo delle secanti

```
function [x, i] = secanti(f, x0, x1, tolx, maxit)
    % [x, i] = secanti(f, x0, x1, tolx, maxit)
    % Method of the secant, used to calculate a root of the equation f(x)=0
    % Input:  f     - function that we can derive
    %         x0    - initial approximation
    %         x1    - second initial approximation
    %         tolx  - tolerance
    %         maxit - maximum number of iterations (default = 100)
    % Output: x     - the approximation of the root function
    %         i     - number of iterations
    % CHECK THIS: bisezione, corde, secanti, aitken, newtonmod

    format long e;

    if nargin < 4
        error('Number of arguments must be at least 4!');
    elseif nargin == 4
        maxit = 100;
    end

    i = 0;
    f0 = feval(f, x0);

    for i = 1:maxit
        f1 = feval(f, x1);
        df1 = (f1 - f0) / (x1 - x0);
        x = x1 - (f1 / df1);

        if abs(x1 - x0) <= tolx * (1 + abs(x0))
            break;
        end

        x0 = x1;
        x1 = x;
        f0 = f1;

    end
```

```
    if abs(x - x0) > tolx * (1 + abs(x0))
        error('The method does not converge!');
    end

end
```

- Metodo di Steffensen

```
function [x, i] = steffensen(f, x0, tolx, maxit)
    % [x, i] = steffensen(f, x0, tolx, maxit)
    % Steffensen's method, used to calculate a root of the equation f(x)=0
    % Input:  f     - a fixed point iteration function
    %         x0    - initial guess to the fixed point
    %         tolx  - tolerance
    %         maxit - maximum number of iterations (default = 100)
    % Output: x     - the approximation of the root function
    %         i     - number of iterations
    % CHECK THIS: bisezione, corde, secanti, aitken, newtonmod

    format long e;

    if nargin < 3
        error('Number of arguments must be at least 3!');
    elseif nargin == 3
        maxit = 100;
    end

    for i = 1:maxit
        % get ready to do a large, but finite, number of iterations.
        % This is so that if the method fails to converge, we won't
        % be stuck in an infinite loop.
        fx = feval(f, x0); % calculate the next two guesses for the fixed point.
        f_fx_x = feval(f, fx + x0);
        x = x0 - (fx^2 / (f_fx_x - fx));
        % use Aitken's delta squared method to
        % find a better approximation to x0.
        if abs(x - x0) <= tolx * (1 + abs(x0))
            break; % if we are, stop the iterations, we have our answer.
        end

        x0 = x;
    end

    if abs(x - x0) > tolx * (1 + abs(x0))
        error(['Failed to converge in ' maxit ' iterations!']);
    end

end
```

2.3 Esercizio 6

Utilizzare le *function* del precedente esercizio per determinare una approssimazione della radice della funzione

$$f(x) = x - cos(\frac{\pi}{2}x),$$

per $tol = 10^{-3}, 10^{-6}, 10^{-9}, 10^{-12}$, partendo da $x_0 = 1$ (e $x_1 = 0.99$ per il metodo delle secanti). Tabulare i risultati, in modo da confrontare le iterazioni richieste da ciascun metodo. Commentare il relativo costo

computazionale, in termini di valutazioni funzionali richieste.
Soluzione: Eseguendo lo script es6.msi ottengono i risultati contenuti nella tabella 2 e nella figura 1. Come si può notare, il metodo di newton e il metodo delle secanti convergono molto più rapidamente del metodo di bisezione e del metodo delle corde.

Metodo	tolleranza= 10^{-3}	tolleranza= 10^{-6}	tolleranza= 10^{-9}	tolleranza= 10^{-12}
newton	5.94611646360541e-01	5.94611644056836e-01	5.94611644056836e-01	5.94611644056836e-01
secanti	5.94611646954077e-01	5.94611644056836e-01	5.94611644056836e-01	5.94611644056836e-01
steffensen	5.94611681141925e-01	5.94611644056837e-01	5.94611644056836e-01	5.94611644056836e-01

Table 1: valori approssimati con i metodi di Newton, secanti e Steffensen

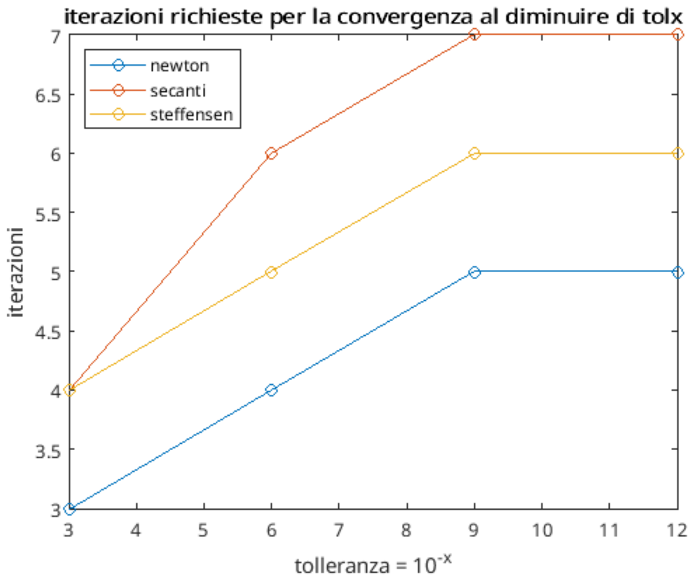


Figure 1: iterazioni richieste

2.4 Esercizio 7

Utilizzare le *function* del precedente esercizio per determinare una approssimazione della radice della funzione

$$f(x) = \left[x - cos(\frac{\pi}{2}x)\right]^3,$$

per $tol = 10^{-3}, 10^{-6}, 10^{-9}, 10^{-12}$, partendo da $x_0 = 1$ (e $x_1 = 0.99$ per il metodo delle secanti). Tabulare i risultati, in modo da confrontare le iterazioni richieste da ciascun metodo. Commentare i risultati ottenuti.
Soluzione: Eseguendo lo script es7.msi ottengono i risultati contenuti nella tabella **??** e nella figura 2. Come si può notare, il metodo di newton e il metodo delle secanti convergono molto più rapidamente del metodo di bisezione e del metodo delle corde.

Metodo	tolleranza= 10^{-3}	tolleranza= 10^{-6}	tolleranza= 10^{-9}	tolleranza= 10^{-12}
newton	5.94611646360541e-01	5.94611644056836e-01	5.94611644056836e-01	5.94611644056836e-01
secanti	5.94611646954077e-01	5.94611644056836e-01	5.94611644056836e-01	5.94611644056836e-01
steffensen	5.94611681141925e-01	5.94611644056837e-01	5.94611644056836e-01	5.94611644056836e-01

Table 2: valori approssimati con i metodi di Newton, secanti e Steffensen

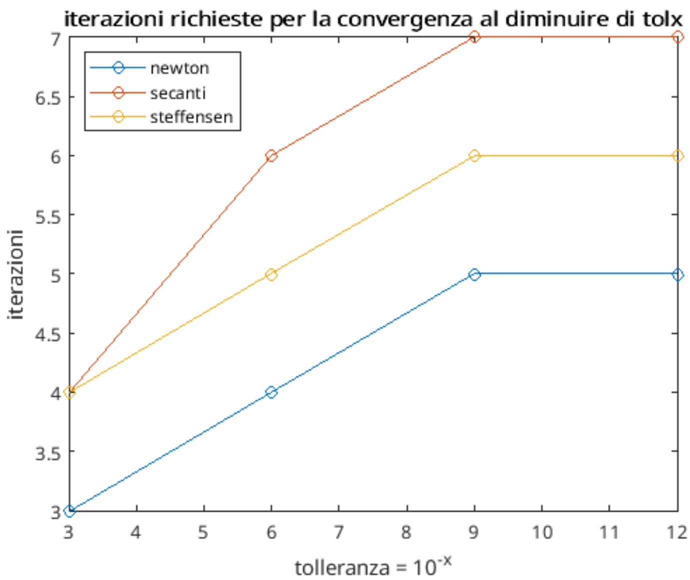


Figure 2: iterazioni richieste

3 Capitolo 3

3.1 Esercizio 8

Scrivere una function Matlab,

```
    function x = mialu(A, b)
```

che, data in ingresso una matrice A ed un vettore b, calcoli la soluzione del sistema lineare $Ax = b$ con il metodo di fattorizzazione LU con *pivoting* parziale. Curare particolarmente la scrittura e l'efficienza della function, e validarla su due esempi non banali, generati casualmente, di cui sia nota la soluzione.
Soluzione:

```
function x = mialu(A, b)
    % x = mialu(A, b)
    % Method of the secant, used to calculate a root of the equation f(x)=0
    % Input:  A     - a matrix raprresentation of the equations with unknown variables
    %         b     - result of unknown variables
    % Output: x     - array to what equals unknown variables

    % Calculate L and U matrixes
    n = length(A);
    L = zeros(n);
    U = zeros(n);
    % P = eye(n);

    for k = 1:n
        % find the entry in the left column with the largest abs value (pivot)
        [~, r] = max(abs(A(k:end, k)));
        r = n - (n - k + 1) + r;

        A([k r], :) = A([r k], :);
        % P([k r], :) = P([r k], :);
        L([k r], :) = L([r k], :);
        b([k r], :) = b([r k], :);

        % from the pivot down divide by the pivot
        L(k:n, k) = A(k:n, k) / A(k, k);

        U(k, 1:n) = A(k, 1:n);
        A(k + 1:n, 1:n) = A(k + 1:n, 1:n) - L(k + 1:n, k) * A(k, 1:n);

    end

    U(:, end) = A(:, end);

    x = zeros(n, 1);
    y = zeros(n, 1);

    % calcolo delle soluzioni di Ly = b
    for i = 1:1:n
        alpha = 0;

        for k = 1:1:i
            alpha = alpha + L(i, k) * y(k);
        end

        y(i) = b(i) - alpha;
    end

```

```
    % calcolo delle soluzioni di Ux = y
    for i = n:-1:1
        alpha = 0;

        for k = i + 1:1:n
            alpha = alpha + U(i, k) * x(k);
        end

        x(i) = (y(i) - alpha) / U(i, i);
    end

end
```

3.2 Esercizio 9

Scrivere una function Matlab,

```
    function x = mialdl(A, b)
```

che, dati in ingresso una matrice sdp A ed un vettore b, calcoli la soluzione del corrispondente sistema lineare utilizzando la fattorizzazione LDL^T. Curare particolarmente la scrittura e l'efficienza della function, e validarla su due esempi non banali, generati casualmente, di cui sia nota la soluzione.
Soluzione:

```
function x = mialdl(A, b)
    % x = mialdl(A, b)
    % Method of the secant, used to calculate a root of the equation f(x)=0
    % Input:  A     - a matrix rapresentation of the equations with unknown variables
    %                 It is assumed that A is symmetric and postive definite.
    %         b     - result of unknown variables
    % Output: x     - array to what equals unknown variables

    %  Figure out the size of A.
    n = size(A, 1);
    % The main loop.  See Golub and Van Loan for details.
    L = zeros(n, n);

    for j = 1:n,

        if (j > 1),
            v(1:j - 1) = L(j, 1:j - 1) .* d(1:j - 1);
            v(j) = A(j, j) - L(j, 1:j - 1) * v(1:j - 1)';
            d(j) = v(j);

            if (j < n),
                L(j + 1:n, j) = (A(j + 1:n, j) - L(j + 1:n, 1:j - 1) * v(1:j - 1)') / v(j
                    );
            end;

        else
            v(1) = A(1, 1);
            d(1) = v(1);
            L(2:n, 1) = A(2:n, 1) / v(1);
        end;

    end;

    %  Put d into a matrix.
    D = diag(d);
```

```
    %  Put ones on the diagonal of L.
    L = L + eye(n);

    % n = size(A,1);
    % L=zeros(n,n);
    % for j=1:n,
    %     if (j>1),
    %         v(1:j-1)=L(j,1:j-1).*d(1:j-1);
    %         v(j)=A(j,j)-L(j,1:j-1)*v(1:j-1)';
    %         d(j)=v(j);
    %         if(j<n),
    %             L(j+1:n,j)=(A(j+1:n,j)-L(j+1:n,1:j-1)*v(1:j-1)')/v(j);
    %         end;
    %     else
    %         v(1)=A(1,1);
    %         d(1)=v(1);
    %         L(2:n,1)=A(2:n,1)/v(1);
    %     end;
    % end;
    % D=diag(d);
    % L=L+eye(n);

    U = D * L';
    x = zeros(n, 1);
    y = zeros(n, 1);

    % calcolo delle soluzioni di Ly = b
    for i = 1:1:n
        alpha = 0;

        for k = 1:1:i
            alpha = alpha + L(i, k) * y(k);
        end

        y(i) = b(i) - alpha;
    end

    % calcolo delle soluzioni di Ux = y
    for i = n:-1:1
        alpha = 0;

        for k = i + 1:1:n
            alpha = alpha + U(i, k) * x(k);
        end

        x(i) = (y(i) - alpha) / U(i, i);
    end

end
```

3.3 Esercizio 10

Data la function Matlab

```
function [A, b] = linsis(n, k, simme)
    %
    % [A,b] = linsis(n,k,simme) Crea una matrice A nxn ed un termine noto b,
    % in modo che la soluzione del sistema lineare
```

```
    % A*x=b sia x = [1,2,...,n]'.
    % k `e un parametro ausiliario.
    % simme, se specificato, crea una matrice
    % simmetrica e definita positiva.
    %
    sigma = 10^(-2 * (1 - k)) / n;
    rng(0);
    [q1, r1] = qr(rand(n));

    if nargin == 3
        q2 = q1';
    else
        [q2, r1] = qr(rand(n));
    end

    A = q1 * diag([sigma 2 / n:1 / n:1]) * q2;
    x = [1:n]';
    b = A * x;
    return
```

che crea sistemi lineari casuali con soluzione nota, risolvere, utilizzando la *function mialu*, i sistemi lineari generati da $[A, b] = linsis(10, 1)$ e $[A, b] = linsis(10, 10)$. Commentare l'accuratezza dei risultati ottenuti, dandone spiegazione esaustiva.
Soluzione:

3.4 Esercizio 11

Risolvere, utilizzando la *function mialdlt*, i sistemi lineari generati da $[A, b] = linsis(10, 1, 1)$ e $[A, b] = linsis(10, 10, 1)$. Commentare l'accuratezza dei risultati ottenuti, dandone spiegazione esaustiva.
Soluzione:

3.5 Esercizio 12

Scrivere una function Matlab,

```
    function [x,nr] = miaqr(A, b)
```

che, data in ingresso la matrice A $m \times n$, con $m \geq n = rank(A)$, ed un vettore b di lunghezza m, calcoli la soluzione del sistema lineare $Ax = b$ nel senso dei minimi quadrati e, inoltre, la norma, nr, del corrispondente vettore residuo. Curare particolarmente la scrittura e l'efficienza della *function*. Validare la *function miaqr* su due esempi non banali, generati casualmente, confrontando la soluzione ottenuta con quella calcolata con l'operatore Matlab.
Soluzione:

```
function [x, nr] = miaqr(A, b)
    %   QR = myqr(A)
    %   calcola la fattorizzazione QR di Householder della matrice A
    %   Input:
    %           A= matrice quadrata da fattorizzare
    %
    %   Output:
    %           QR=matrice contenente le informazioni sui fattori Q e R della
    %           fattorizzazione QR di A
    %
    [m, n] = size(A);

    if n > m
        error('Dimensioni errate');
    end

```

```
    if length(b) ~= m
        error('Dati inconsistenti');
    end

    QR = A;

    for i = 1:n
        alfa = norm(QR(i:m, i));

        if alfa == 0
            error('la matrice non ha rango massimo');
        end

        if QR(i, i) >= 0
            alfa = -alfa;
        end

        v1 = QR(i, i) -alfa;
        QR(i, i) = alfa;
        QR(i + 1:m, i) = QR(i + 1:m, i) / v1;
        beta = -v1 / alfa;
        v = [1; QR(i + 1:m, i)];
        QR(i:m, i + 1:n) = QR(i:m, i + 1:n) - (beta * v) * (v' * QR(i:m, i + 1:n));
    end

    [m, n] = size(QR);
    k = length(b);

    if k ~= m
        error('Dati inconsistenti');
    end

    x = b(:);

    for i = 1:n
        v = [1; QR(i + 1:m, i)];
        beta = 2 / (v' * v);
        x(i:m) = x(i:m) - beta * (v' * x(i:m)) * v;
    end

    x = x(1:n);

    for j = n:-1:1

        if QR(j, j) == 0
            error('Matrice singolare');
        end

        x(j) = x(j) / QR(j, j);
        x(1:j - 1) = x(1:j - 1) - QR(1:j - 1, j) * x(j);
    end

end

%{

A = round (10 * rand (3))
b = round (10 * rand (3, 1))
```

```
%}
```

3.6 Esercizio 13

```
%Codice esercizio 13

A= [1, 2, 3; 1 2 4; 3 4 5; 3 4 6; 5 6 7];
b=[14 17 26 29 38];
QR=myqr(A);
ris=qrsolve(QR,b);
disp(ris);
```

Il risultato finale è $ris = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$

3.7 Esercizio 14

A\b	(A'*A)\(A'*b)
1.0000	3.5759
2.0000	-3.4624
3.0000	9.5151
4.0000	-1.2974
5.0000	7.9574
6.0000	4.9125
7.0000	7.2378
8.0000	7.9765

Table 3: valori approssimati

L'espressione $A\backslash b$ risolve in matlab, il sistema di equazioni lineari nella forma matriciale $A \cdot x = b$ per x. L'espressione $(A^T \cdot A)\backslash(A^T \cdot b)$, è matematicamente la stessa operazione dell'espressione precedente, solamente che si moltiplica le due componenti per la trasposta di A. La matrice A viene calcolata usando la funzione vander() che genera una matrice di tipo Vandermonde, la quale è mal condizionata. Usando la funzione cond() sulla matrice A si ottiene una condizionamento pari a: 1.5428e+09.Nella prima espressione, questo malcondizionamento non influisce sul risultato.Invece nella seconda espressione eseguendo la prima parentesi tonda, il condizionamento è pari a: 4.4897e+18. Questo fa si che eseguendo la divisione tra una matrice mal condizionata e il vettore, il risultato presenta degli errori.

4 Capitolo 4

4.1 Esercizio 15

Eseguendo il codice es15.m si ottengono i seguenti risultati:

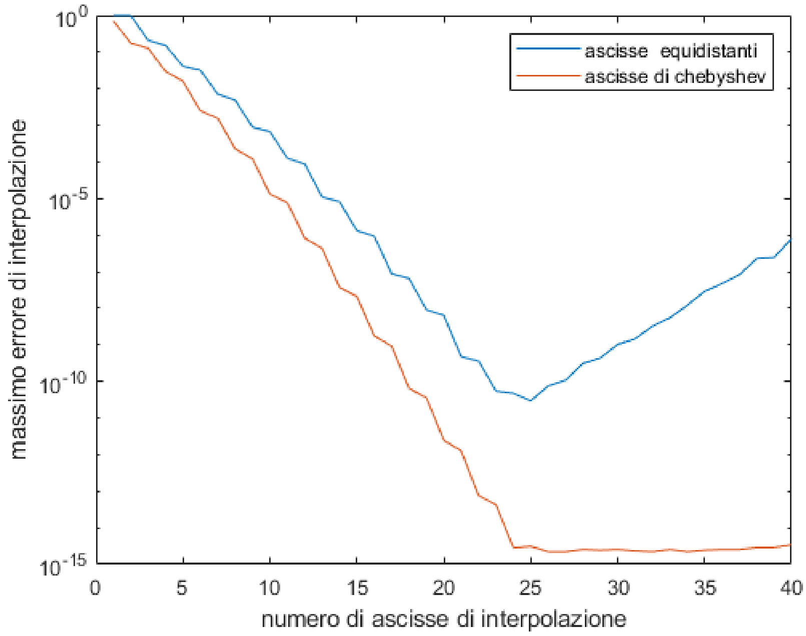


Figure 3: risultati interpolazione

Per le ascisse di chebyshev, si ha una decrescita esponenziale dell'errore massimo per $n \leq 25$ per poi assestarsi a circa $2 \cdot 10^{-15}$ per n successivi. Per quanto riguarda le ascisse equidistanti invece, si può notare come l'errore massimo torni a crescere esponenzialmente per $n > 25$. I risultati confermano il mal condizionamento del problema di interpolazione polinomiale quando vengono usate ascisse d'interpolazione equidistanti,

4.2 Esercizio 16

Eseguendo es16.m si ottiene:

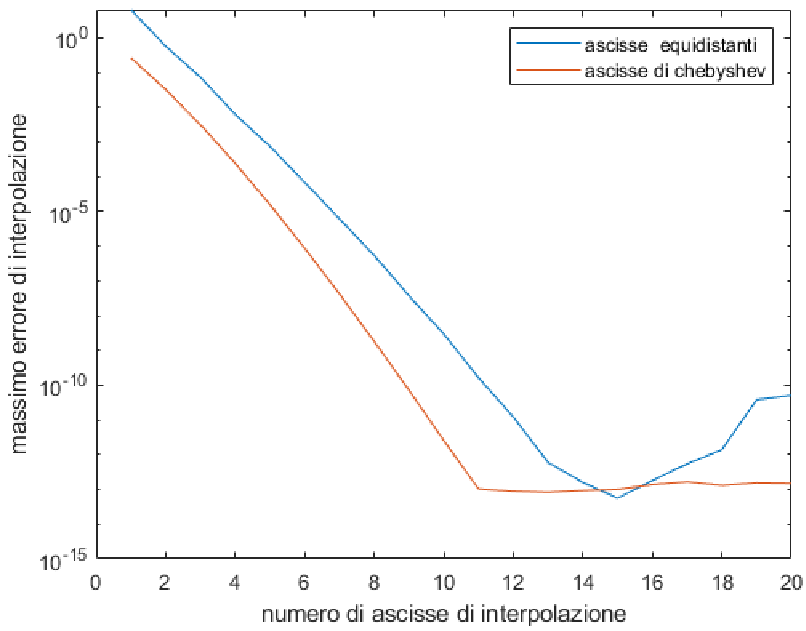


Figure 4: risultati interpolazione hermite

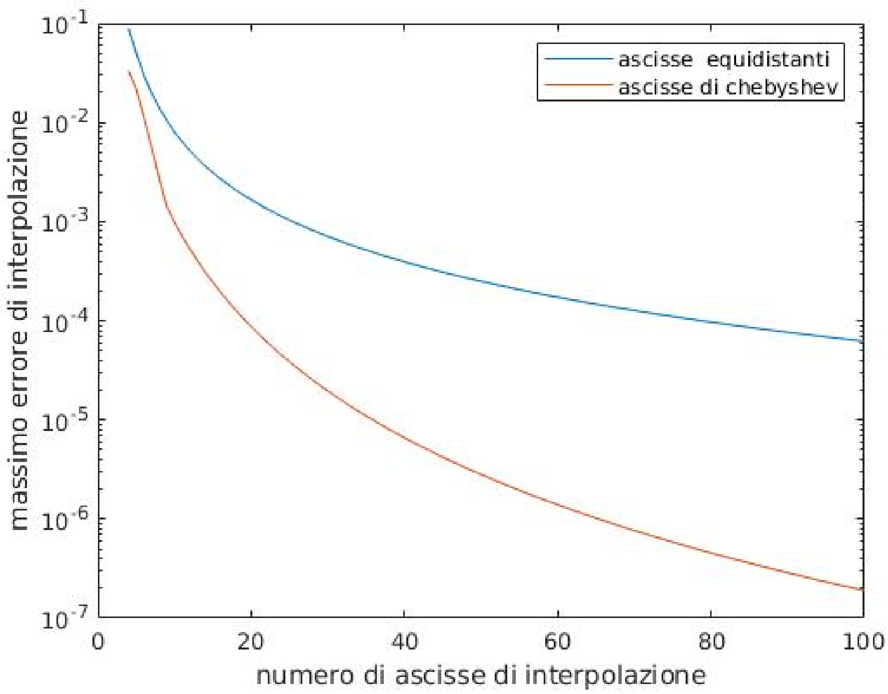
Si può notare come, rispetto all'interpolazione classica, l'errore decresca più rapidamente per $n \leq 15$. Anche in questo caso l'errore commesso usando le ascisse di chebyshev è migliore in confronto al caso delle ascisse equidistanti(eccetto per $n = 15$).

4.3 Esercizio 17

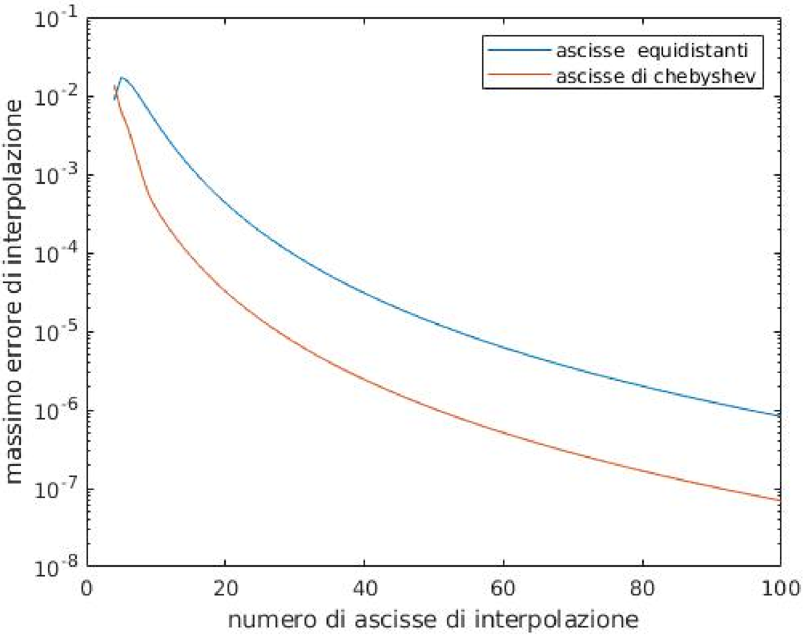
```
function output=splinenat(xi,fi,xq)
%
% output=splinenat(x,y,xq)
%funzione che calcola la spline cubica naturale.
%Input:
%   xi=vettore delle ascisse su cui calcolare la spline
%   fi=vettore dei valori di f(x), con x ascissa
%   xq= insieme delle ascisse di cui si vuole sapere il valore della spline
%Output:
%   output=vettore delle approssimazioni sulle ascisse xq
%

m = length(xi);
l=length(xq);

if m~=length(fi)
    error(dati errati);
end
for i = 1:m-1
    if any( find(xi(i+1:m)==xi(i)) ),
        error(ascisse non distinte),
    end
end
xi = xi(:);
```

```
fi = fi(:);
[xi,ind] = sort(xi);
fi = fi(ind);
% ordino le ascisse in modo crescente
hi = diff(xi);
n= m-1;
df = diff(fi)./hi;
hh = hi(1:n-1)+hi(2:n);
rhs = 6*diff(df)./hh;
phi = hi(1:n-1)./hh;
csi = hi(2:n)./hh;
% = 1-phi;
d= 2*ones(n-1,1);
phi = phi(2:n-1);
csi = csi(1:n-2);
mi = trisolve( phi, d, csi, rhs );
mi = [0;mi;0];
r=fi(1:n)-((hi(1:n).^2)/6).*mi(1:n);
q=df(1:n)-((hi(1:n)/6).*(mi(2:m)-mi(1:n)));
output=zeros(l,1);
for i=1:l
    indp=find(xq(i)>=xi(1:n),1,'last');
    indg=indp+1;
    output(i)=((((xq(i)-xi(indp)).^3).*mi(indg)+((xi(indg)-xq(i)).^3).*mi(indp))/(6*hi(indp)))+q(indp).*(xq(i)-xi(indp))+r(indp);
end
return
end
```

4.4 Esercizio 18



4.5 Esercizio 19

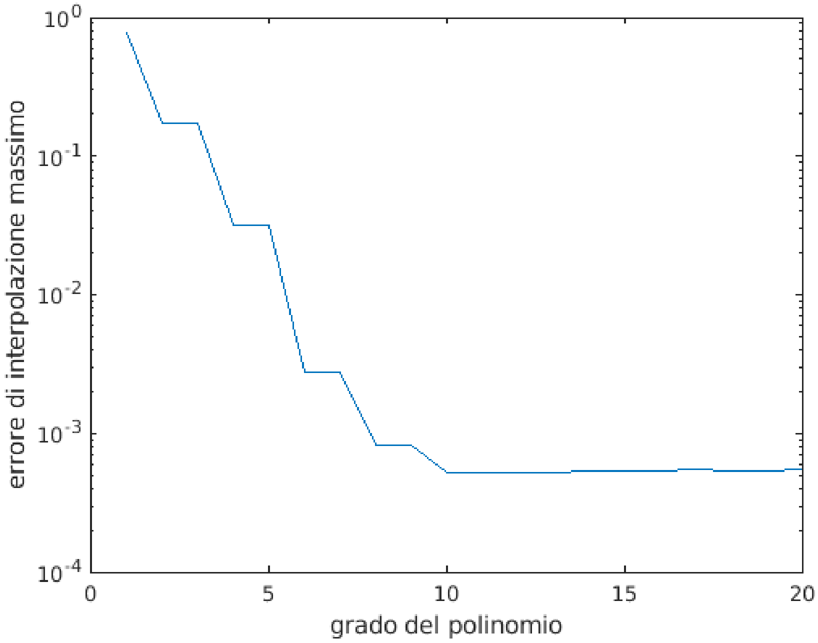


Si nota dai due grafici che la funzione spline riesce a mantenere su entrambe le tipologie di scelta delle ascissi, un basso errore di interpolazione, mentre la funzione splinenat, arriva ad avere nelle ascisse di chebyshev, un andamento simile a quello corrispettivo alla funzione spline, essendo comunque meno preciso. L'uso invece delle ascisse equidistanti nella funzione splinenat, fa si che si abbia un andamento sostanzialmente diverso, rispetto a quello delle corrispettive nella funzione spline, e anche rispetto alle ascisse di chebyshev calcolate dalla stessa funzione.

4.6 Esercizio 20

```
function y = minimiquadrati(xi, fi, m)
%
%
%  y = minimiquadrati(xi, fi, m)
%  calcola il valore del polinomio di approssimazione ai minimi quadrati di grado m
% sulle ascisse xi. fi contiene i valori approssimati di una funzione f valutata su xi
if length(unique(xi)) < m+1
    error('ascisse distinte non sufficienti');
end
fi = fi(:);
V = fliplr(vander(xi));
V = V(1:end, 1:m+1);
QR = myqr(V);
p = qrsolve(QR, fi);
y = p(m+1)*ones(size(xi));
for i = 0:m-1
    y = y.*xi+p(m-i);
end
end
```

Eseguendo es20.m si ottiene:



Si nota una decrescita delll'errore esponenziale fino a $m = 10$, dove si assesta tra 10^{-3} e 10^{-4}.

5 Capitolo 5

5.1 Esercizio 21

```matlab
function c = ncweights(n)
%
%
%  c = nc-weights(n)
%  calcola i pesi della formula di newton cotes di grado n;
%       n-      grado
%       c-      pesi calcolati
%
if n<=0
    error('grado della formula non positivo');
end
nvalues = floor(n/2 + 1);
c=zeros(1,nvalues);
for j = 1:nvalues
    temp = (0:n);
    temp(j)=[];
    f = @(x)(prod(x-temp) /prod(j-1-temp));
    c(j) = integral(f, 0, n, 'ArrayValued', true);
end
c = [c flip(c)]; %sfrutto la simmetria dei pesi
if mod(n,2)==0
    %elimino la copia del valore centrale prodotta da flip(c) e che risulta di troppo per n pari
    c(n/2+1) = [];
end
return
end
```

Eseguendo lo script es21.m si ottiene:

5.2 Esercizio 22

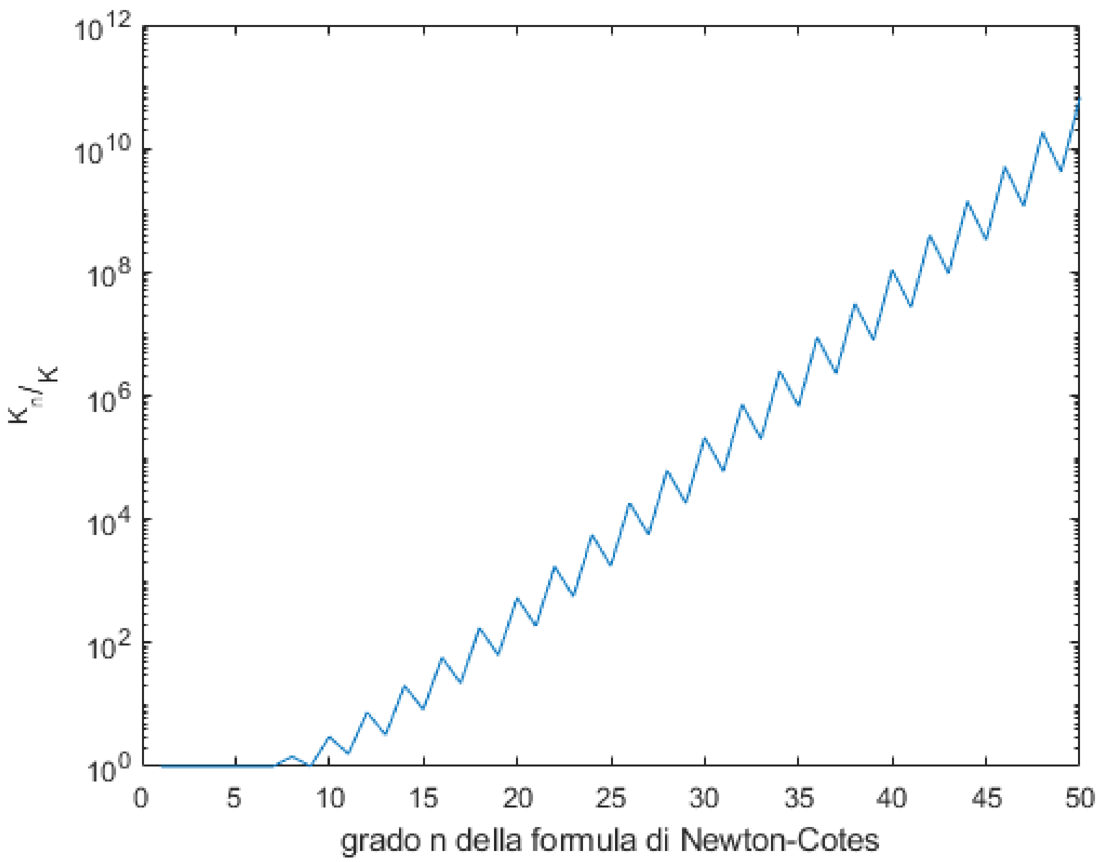
Sappiamo che $k = (b-a)$ e $k_n = (b-a)\frac{1}{n}\sum_{i=0}^{n}|c_{in}|$. Il rapporto sarà dunque dato da:

$$\frac{k_n}{k} = \frac{(b-a)\frac{1}{n}\sum_{i=0}^{n}|c_{in}|}{b-a} = \frac{1}{n}\sum_{i=0}^{n}|c_{in}|$$

Calcolando $\frac{k_n}{k}$ per $n = 1, \ldots.., 50$(es22.m) si ottiene:

n $\backslash c_{in}$	0	1	2	3	4	5	6	7
1	$\frac{1}{2}$	$\frac{1}{2}$						
2	$\frac{1}{3}$	$\frac{4}{3}$	$\frac{1}{3}$					
3	$\frac{3}{8}$	$\frac{9}{8}$	$\frac{9}{8}$	$\frac{3}{8}$				
4	$\frac{14}{45}$	$\frac{64}{45}$	$\frac{8}{15}$	$\frac{64}{45}$	$\frac{14}{45}$			
5	$\frac{95}{288}$	$\frac{125}{96}$	$\frac{125}{144}$	$\frac{125}{144}$	$\frac{125}{96}$	$\frac{95}{288}$		
6	$\frac{41}{140}$	$\frac{54}{35}$	$\frac{27}{140}$	$\frac{68}{35}$	$\frac{27}{140}$	$\frac{54}{35}$	$\frac{41}{140}$	
7	$\frac{108}{355}$	$\frac{810}{559}$	$\frac{343}{640}$	$\frac{649}{536}$	$\frac{649}{536}$	$\frac{343}{640}$	$\frac{810}{559}$	$\frac{108}{355}$

Table 4: pesi della formula di Newton-Cotes fino al settimo grado



5.3 Esercizio 23

```
function y = newtoncotes(f,a, b, n)
%
% y= newtoncotes(f,a,b, n)
% calcola l'approssimazione dell'integrale definito per la funzione f sull'intervallo [a,
    b],
% utilizzando la formula di newton cotes di grado n.
%
%       f-      funzione
%       a,b-    intervallo dei punti
%       n-      grado della formula di newton cotes
%       y-      approssimazione integrale definito di f(x)

if a > b || n < 0
    error('dati inconsistenti');
end
xi = linspace(a, b, n+1);
fi = feval(f, xi);
h = (b-a) / n;
c = ncweights(n);
y = h*sum(fi.*c);
return
end
```

RISULTATI PER N DA 1 A 9(es23.m):

grado della formula	valore integrale	errore
1	$4,28 \cdot 10^{-1}$	$2,53 \cdot 10^{-1}$
2	$2,13 \cdot 10^{-1}$	$3,8 \cdot 10^{-2}$
3	$1,96 \cdot 10^{-1}$	$2,1 \cdot 10^{-2}$
4	$1,80 \cdot 10^{-1}$	$5 \cdot 10^{-3}$
5	$1,79 \cdot 10^{-1}$	$4 \cdot 10^{-3}$
6	$1,76 \cdot 10^{-1}$	$1 \cdot 10^{-3}$
7	$1,76 \cdot 10^{-1}$	$1 \cdot 10^{-3}$
8	$1,75 \cdot 10^{-1}$	0
9	$1,75 \cdot 10^{-1}$	0

5.4 Esercizio 24

```
function I = trapecomp(f, a, b, n)
%
%
% I = trapecomp(f, a, b)
%
% Approssimazione dell'integrale definito di f(x) con estremi a e b,
% mediante la formula composita dei trapezi su n+1 ascisse equidistanti
%       f-      funzione
%       a,b-    estremi dell'intervallo
%       I-      approssimazione integrale definito di f(x)
%
if a==b
    I=0;
elseif n < 1 || n~=fix(n)
    error('numero di ascisse non valido');
else
    h=(b-a)/n;
```

```
    x=linspace(a, b, n+1);
    f = feval(f, x);
    I = h*(f(1)/2 + sum(f(2:n)) + f(n+1)/2);
end
return
end
```

```
function I = simpcomp(f, a, b, n)
    %myFun - Description
    %
    % I = simpcomp(f, a, b)
    %
    % Approssimazione dell'integrale definito di f(x) con estremi a e b,
    % mediante la formula composita di Simpson su n+1 ascisse equidistanti(n pari)
    %   f-      funzione
    %   a,b-    estremi dell'intervallo
    %   I-      approssimazione integrale definito di f(x)
    %
    %

    if a==b
        I=0;
    elseif n < 2 || n/2 ~= fix(n/2)
        error('numero di ascisse non valido');
    else
        h=(b-a)/n;
        x=linspace(a, b, n+1);
        f = feval(f, x);
        I = (h/3) * (f(1) + f(n+1) + 4*sum(f(2:2:n)) + 2*sum(f(3:2:n-1)));
    end
    return
    end
```

Approssimando $\int_{-1}^{1.1} tan(x)dx$ con le due formule si ottiene:

intervalli\formula	trapezi composita	simpson composita
2	0.266403558406035	0.266403558406035
4	0.203432804450016	0.182442553131343
6	0.188498346613972	0.177333443886033
8	0.182789408875225	0.175908277016961
10	0.180034803521960	0.175392868382289
12	0.178504015707472	0.175172572071972
14	0.177568218195411	0.175066546519247
16	0.176955413111201	0.175010747856527
18	0.176532709616469	0.174979254439942
20	0.176229037552030	0.174960448895386

Table 5: risultati di es24.m

La formula composita di simpson converge più rapidamente ed è più precisa rispetto alla formula dei trapezi

5.5 Esercizio 25

```
function [I2, points] = adaptrap(f, a, b, tol, fa, fb)
%
%
```

```
% Syntax: [I2, points] = adaptrap(f, a, b, tol, fa, fb)
%
% Approssimazione dell'integrale definito di f(x) con estremi a e b,
% mediante la formula adattiva dei trapezi
%       f-      funzione
%       a,b-    estremi intervallo
%       tol-    tolleranza
%       fa,fb-  valore della funzione valutata negli punti a,b
%
%       points- punti valutati
%       I2-     valore dell'integrale di f(x)
%
global points
delta = 0.5;
if nargin<=4
    fa = feval(f, a );
    fb = feval(f, b );
    if nargout==2
        points = [a fa; b fb];
    else
        points = [];
    end
end
h = b-a;
x1 = (a+b)/2;
f1 = feval(f, x1);
if ~isempty(points)
    points = [points; [x1 f1]];
end
I1 = .5*h*(fa+fb);
I2 = .5*(I1+h*f1);
e = abs(I2-I1)/3;
if e>tol || abs(b-a) > delta
    I2 = adaptrap(f, a, x1, tol/2, fa, f1) + adaptrap(f, x1, b, tol/2, f1, fb);
end
return
end
```

```
function [I2, points] = adapsim(f, a, b, tol, fa, f1, fb)
% [I2, points] = adapsim(f, a, b, tol, fa, f1, fb)
% Approssimazione dell'integrale definito di f(x) con estremi a e b,
% mediante la formula adattiva di simpson
%       f-      funzione
%       a,b-    estremi intervallo
%       tol-    tolleranza
%       fa,fb-  valore della funzione valutata negli punti a,b
%       f1-     valore della funzione valutata nel punto x1, intermedio ad a,b
%
%       points- punti valutati
%       I2-     valore dell'integrale di f(x)
%
global points
delta = 0.5;
x1 = (a+b)/2;
if nargin<=4
    fa = feval(f, a );
    fb = feval(f, b );
    f1 = feval(f, x1);
```

```
    if nargout==2
        points = [a fa;x1 f1; b fb];
    else
        points = [];
    end
end
h = (b-a)/6;
x2 = (a+x1)/2;
x3 = (x1+b)/2;
f2 = feval(f, x2);
f3 = feval(f, x3);
if ~isempty(points)
    points = [points; [x2 f2; x3 f3]];
end
I1 = h*(fa+fb+4*f1);
I2 = .5*h*(fa+4*f2+2*f1+4*f3+fb);
e = abs(I2-I1)/15;
if e>tol || abs(b-a) > delta
    I2 = adapsim(f, a, x1, tol/2, fa, f2, f1) + adapsim(f, x1, b, tol/2, f1, f3, fb);
end
return
end
```

Approssimando $\int_{-1}^{1} \frac{1}{1+10^2x^2}dx$ con le due formule si ottiene:

tolleranza\formula	trapezi adattiva	simpson adattiva
10^{-2}	0.295559711784128, punti = 21	0.281297643062670, punti = 17
10^{-3}	0.294585368185034 , punti = 93	0.281297643062670, punti = 17
10^{-4}	0.294274200873635, punti = 277	0.294259338419631, punti = 41
10^{-5}	0.294230142164878, punti = 793	0.294227809768005, punti = 81
10^{-6}	0.294226019603178, punti = 2692	0.294225764620384 , punti = 145

Table 6: risultati di es25.m

Per ciascuna formula, l'operazione che comporta maggior costo computazionale ad ogni chiamata è la valutazione funzionale dei punti di un sottointervallo. Poichè ogni punto viene valutato una sola volta, possiamo confrontare il costo delle due formule andando a vedere quanti punti aggiuntivi sono stati utilizzati. Osservando i dati riportati nella tabella 6, è palese come la formula di simpson adattiva convergà più rapidamente rispetto alla formula dei trapezi adattiva.

6 Codici ausiliari

6.1 Esercizio 6

Listing 1: es6.m

```
syms x;
f = @(x)(x - cos((pi / 2) * x));
f1 = matlabFunction(diff(f, x));

x0 = 1;
x1 = 9.9e-1;
x = zeros(3, 4);
y = zeros(3, 4);

for i = 3:3:12

    [x(1, i / 3), y(1, i / 3)] = es5_newton(f, f1, x0, 10^(-i));
    [x(2, i / 3), y(2, i / 3)] = es5_secanti(f, x0, x1, 10^(-i));
    [x(3, i / 3), y(3, i / 3)] = es5_steffensen(f, x0, 10^(-i));
end

row_names = {'newton', 'secanti', 'steffensen'};
colnames = {'10^-3', '10^-6', '10^-9', '10^-12'};
values = array2table(x, 'RowNames', row_names, 'VariableNames', colnames);
disp(values)
figure
plot([3, 6, 9, 12], y', 'o-')
title('iterazioni richieste per la convergenza al diminuire di tolx')
xlabel('tolleranza = 10^{-x}')
ylabel('iterazioni')
legend({'newton', 'secanti', 'steffensen'}, 'Location', 'northwest')
```

6.2 Esercizio 7

Listing 2: es7.m

```
f = @(x)(x^2*tan(x));
f1 = @(x)(2*x*tan(x) +(x^2)/(cos(x)^2));
m = 3;
x0 = 1;
y= zeros(3, 4);
x=-1*ones(3,4);
for i=3:3:12
  [x(1, i/3), y(1, i/3)] = newton(f, f1, x0, 10^(-i));
  [x(2, i/3), y(2, i/3)] = newtonmod(f, f1, x0, m, 10^(-i));
  [x(3, i/3), y(3, i/3)] = aitken(f, f1, x0, 10^(-i));
end
disp(x);
disp(y);
row_names = {'newton', 'newton modificato', 'aitken'};
colnames = {'10^-3','10^-6','10^-9','10^-12'};
values = array2table(x,'RowNames',row_names,'VariableNames',colnames)

format
iterations = array2table(y,'RowNames',row_names,'VariableNames',colnames)
plot([3, 6, 9, 12], y(1,1:end)','-o');
hold on;
```

```
plot([3, 6, 9, 12], y(2,1:end)','-o');
plot([3, 6, 9, 12], y(3,1:end)','--')
title('iterazioni richieste per la convergenza al diminuire di tolx')
xlabel('tolleranza = 10^{-x}')
ylabel('iterazioni')
legend({'newton','newtonmod','aitken'},'Location','northwest')
```

6.3 Esercizio 15

Listing 3: es15.m

```
%Codice esercizio 15

f = @(x)(cos((pi*x.^2)/2));
x = linspace(-1, 1, 100001);
linerrors = zeros(1, 40);
chebyerrors = zeros(1, 40);
for n = 1:40
    xlin = linspace(-1, 1, n+1);
    xcheby = chebyshev(-1,1,n+1);
    ylin = lagrange(xlin,f(xlin),x);
    ycheby = lagrange(xcheby,f(xcheby),x);
    linerrors(n) = norm(abs(f(x) - ylin), inf);
    chebyerrors(n) = norm( abs(f(x) - ycheby), inf);
end
semilogy(linerrors);
hold on;
semilogy(chebyerrors);
xlabel('numero di ascisse di interpolazione');
ylabel('massimo errore di interpolazione');
legend({'ascisse  equidistanti', 'ascisse di chebyshev'},'Location','northeast');
```

6.4 Esercizio 16

Listing 4: es16.m

```
%Codice esercizio 16

f = @(x)(cos((pi*x.^2)/2));
f1 = @(x)(-pi*x.*sin((pi*x.^2)/2));
x = linspace(-1, 1, 100001);
linerrors = zeros(1, 20);
chebyerrors = zeros(1, 20);
for n = 1:20
    xlin = linspace(-1, 1, n+1);
    xcheby = chebyshev(-1,1, n+1);
    ylin = hermite(xlin,f(xlin),f1(xlin),x);
    ycheby = hermite(xcheby,f(xcheby),f1(xcheby),x);
    linerrors(n) = norm(abs(f(x) - ylin), inf);
    chebyerrors(n) = norm( abs(f(x) - ycheby), inf);
end
semilogy(linerrors);
hold on;
semilogy(chebyerrors);
xlabel('numero di ascisse di interpolazione');
ylabel('massimo errore di interpolazione');
legend({'ascisse  equidistanti', 'ascisse di chebyshev'},'Location','northeast');
```

6.5 Esercizio 18

Listing 5: es18.m

```matlab
%Codice esercizio 18
f = @(x)(cos((pi*(x.^2))/2));
x = linspace(-1, 1, 100001);
linerrors = zeros(1, 40);
chebyerrors = zeros(1, 40);
for n = 4:100
    xlin = linspace(-1, 1, n+1);
    xcheby = chebyshev(-1,1,n+1);
    %xcheby(1)=-1;
    %xcheby(n+1)=1;
    ylin = splinenat(xlin,f(xlin),x);
    ycheby = splinenat(xcheby,f(xcheby),x);
    ylin=ylin';
    ycheby=ycheby';
    linerrors(n) = norm(abs(f(x) - ylin), inf);
    chebyerrors(n) = norm( abs(f(x) - ycheby), inf);
end
semilogy(linerrors);
hold on;
semilogy(chebyerrors);
xlabel('numero di ascisse di interpolazione');
ylabel('massimo errore di interpolazione');
legend({'ascisse  equidistanti', 'ascisse di chebyshev'},'Location','northeast');
```

6.6 Esercizio 19

Listing 6: es19.m

```matlab
%Codice esercizio 19
f = @(x)(cos((pi*(x.^2))/2));
x = linspace(-1, 1, 100001);
linerrors = zeros(1, 40);
chebyerrors = zeros(1, 40);
for n = 4:100
    xlin = linspace(-1, 1, n+1);
    xcheby = chebyshev(-1,1,n+1);
    ylin = spline(xlin,f(xlin),x);
    ycheby = spline(xcheby,f(xcheby),x);
    linerrors(n) = norm(abs(f(x) - ylin), inf);
    chebyerrors(n) = norm( abs(f(x) - ycheby), inf);
end
semilogy(linerrors);
hold on;
semilogy(chebyerrors);
xlabel('numero di ascisse di interpolazione');
ylabel('massimo errore di interpolazione');
legend({'ascisse  equidistanti', 'ascisse di chebyshev'},'Location','northeast');
```

6.7 Esercizio 20

Listing 7: es20.m

```matlab
%Codice esercizio 20

```

```
f = @(x)(cos((pi*x.^2)/2));
fp = @(x)(f(x) + 10^(-3)*rand(size(x)));
xi = -1 + 2*(0:10^4)/10^4;
fi = f(xi);
fpi = fp(xi);
errors=zeros(1, 20);
for m = 1:20
    y = minimiquadrati(xi, fpi, m);
    errors(m) = norm(abs(y-fi), inf);
end
semilogy(errors);
xlabel('grado del polinomio');
ylabel('errore di interpolazione massimo');
```

6.8 Esercizio 21

Listing 8: es21.m

```
%Codice esercizio 21

for i = 1:7
    weights= rats(ncweights(i))
end
```

6.9 Esercizio 22

Listing 9: es22.m

```
%Codice esercizio 22

rapp = zeros(1, 50);
for i = 1:50
    rapp(i) = sum(abs(ncweights(i)))/i;
end
semilogy(rapp);
xlabel('grado n della formula di Newton-Cotes');
ylabel('^{K_n}/_{K}');
```

6.10 Esercizio 23

Listing 10: es23.m

```
%Codice esercizio 23

value = log(cos(1)/cos(1.1));
x = zeros(1,9);
errors=zeros(1, 9);
for i = 1:9
    x(i) = newtoncotes(@tan, -1,1.1, i);
    errors(i) = abs(value-x(i));
end
```

6.11 Esercizio 24

Listing 11: es24.m

```
%Codice esercizio 24

a = -1;
b = 1.1;
n = 10;
itrap = zeros(1, n);
isimp = zeros(1, n);
for i = 1:n
    itrap(i) = trapecomp(@tan, a, b, i*2);
    isimp(i) = simpcomp(@tan, a, b, i*2);
end
integrali = [itrap; isimp];
row_names = {'trapezi composta', 'simpson composta'};
colnames = {'2','4','6','8','10','12','14','16','18','20'};
values = array2table(integrali,'RowNames',row_names,'VariableNames',colnames);
disp(values);
```

6.12 Esercizio 25

Listing 12: es25.m

```
%Codice esercizio 25

format long e
f = @(x)(1/(1+100*x.^2));
a = -1;
b = 1;
itrap = zeros(1, 5);
trap_points = zeros(1, 5);
isimp = zeros(1, 5);
simp_points = zeros(1, 5);
for i = 1:5
    [itrap(i), points] = adaptrap(f, a, b, 10^(-i-1));
    trap_points(i) =  length(points);
    [isimp(i), points] = adapsim(f, a, b, 10^(-i-1));
    simp_points(i) = length(points);
end
integrali = [itrap; isimp];
npoints = [trap_points; simp_points];
row_names = {'trapezi adattiva', 'simpson adattiva'};
colnames = {'10^-2','10^-3','10^-4','10^-5','10^-6'};
values = array2table(integrali,'RowNames',row_names,'VariableNames',colnames);
npoints = array2table(npoints,'RowNames',row_names,'VariableNames',colnames);
disp(values);
format
disp(npoints);
```