CALCOLO NUMERICO

Vallierick Facunla - Mariani Dario - Puocci Ubaldo

Contents

1 Errori ed aritmetica finita

Soluzione 1.1 Per la convergenza del metodo iterativo si ha che

$$\lim_{k\to\infty} x_k = x^*.$$

Essendo $\Phi(x_k)$ continua:

$$x^* = \lim_{k\to\infty} x_k = \lim_{k\to\infty} \Phi(x_{k-1}) = \Phi(\lim_{k\to\infty} x_{k-1}) = \Phi(x^*).$$

Soluzione 1.2 Il Fortran utilizza una rappresentazione circolare dei numeri di macchina. Al passo 3 la variabile *numero* ha come valore il più grande intero rappresentabile in macchina. Al passo 4, venendo incrementata ulteriormente, assume quello del più piccolo intero rappresentabile in macchina. Un caso analogo si ha nel secondo ciclo dal passo 4 al passo 5 dove, decrementando il più piccolo intero, si passa al più grande intero rappresentabile.

Soluzione 1.3 Nel caso in cui un'aritmetica finita implementi l'arrotondamento, la precisione di macchina u corrispondente è:

$$u = \frac{1}{2} b^{1-m}$$

dove b è la base ed m il numero di cifre per la mantissa. La precisione di macchina è dunque:

$$u = \frac{1}{2} 8^{-4} \approx 1.2 \times 10^{-4}$$

Soluzione 1.4

h	$\Psi_h(0)$
10^{-1}	1.051709180756477
10^{-2}	1.005016708416795
10^{-3}	1.000500166708385
10^{-4}	1.000050001667141
10^{-5}	1.000005000006965
10^{-6}	1.000000499962184
10^{-7}	1.000000049433680
10^{-8}	$9.999999939225290 \times 10^{-1}$
10^{-9}	1.000000082740371
10^{-10}	1.000000082740371
10^{-11}	1.000000082740371
10^{-12}	1.000088900582341

I valori sono giusti? manca la spiegazione.

Soluzione 1.5 Se h è una quantità piccola ma definita, possiamo approssimare la funzione $f(x)$ tramite lo sviluppo di Taylor. Per la prima eguaglianza possiamo fermarci al secondo ordine di approssimazione:

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + f''(x_0)\frac{(x - x_0)^2}{2} + O((x - x_0)^3)$$

dal quale otteniamo che:

$$f(x_0 + h) = f(x_0) + hf'(x_0) + \frac{h^2}{2} f''(x_0) + O(h^3),$$

$$f(x_0 - h) = f(x_0) - hf'(x_0) + \frac{h^2}{2} f''(x_0) + O(h^3).$$

Quindi, effettuando le dovute sostituzioni nell'eguaglianza data:

$$\frac{f(x_0+h)-f(x_0-h)}{2h} =$$

$$= \frac{f(x_0)+hf'(x_0)+\frac{h^2}{2}f''(x_0)+O(h^3)-f(x_0)+hf'(x_0)-\frac{h^2}{2}f''(x_0)+O(h^3)}{2h} =$$

$$= \frac{2hf'(x_0)+O(h^3)}{2h} = f'(x_0)+O(h^2)$$

Per la secondo eguaglianza possiamo fermarci al terzo ordine di approssimazione:

$$f(x) = f(x_0)+f'(x_0)(x-x_0)+f''(x_0)\frac{(x-x_0)^2}{2}+f'''(x_0)\frac{(x-x_0)^3}{6}+O((x-x_0)^4)$$

dal quale otteniamo che:

$$f(x_0+h) = f(x_0)+hf'(x_0)+\frac{h^2}{2}f''(x_0)+\frac{h^3}{6}f'''(x_0)+O(h^4),$$

$$f(x_0-h) = f(x_0)-hf'(x_0)+\frac{h^2}{2}f''(x_0)-\frac{h^3}{6}f'''(x_0)+O(h^4).$$

Quindi, effettuando le dovute sostituzioni nell'eguaglianza data:

$$\frac{f(x_0+h)-2f(x_0)+f(x_0-h)}{h^2} =$$

$$= \frac{\frac{h^2}{2}f''(x_0)+O(h^4)+\frac{h^2}{2}f''(x_0)+O(h^4)}{h^2} = f''(x_0)+O(h^2).$$

Soluzione 1.6

n	x_n	ϵ_{x_n}
0	2.00000000000	5.857864×10^{-1}
1	1.50000000000	8.578643×10^{-2}
2	1.42857142857	1.435786×10^{-2}
3	1.41463414634	4.205839×10^{-4}
4	1.41421568627	2.123901×10^{-6}
5	1.41421356268	3.157739×10^{-10}
6	1.41421356237	2.220446×10^{-16}

Denotando con x_n l'approssimazione del metodo alla $n-esima$ iterazione e con ϵ_{x_n} l'errore di convergenza rispetto a $\sqrt{2}$, si evince dalla tabella che per aver un errore di $\approx 10^{-12}$ dobbiamo arrivare almeno ad $n = 6$.

Soluzione 1.7 Nel caso in cui un'aritmetica finita implementi l'arrotondamento, la precisione di macchina u corrispondente è:

$$u = \frac{1}{2}b^{1-m}$$

dove b è la base ed m il numero di cifre per la mantissa. Ponendo $b = 2$ e $u = 4.66 \times 10^{-10}$ otteniamo:

$$4.66 \times 10^{-10} = 2^{-m} \Rightarrow m = -log_2(4.66 \times 10^{-10}) \approx 31$$

Solzione 1.8 Dobbiamo distinguere due casi: il caso in cui l'aritmetica finita implementi il troncamento ed il caso in cui implementi l'arrotondamento.

- **Troncamento:** sappiamo che la precisione di macchina nel caso in cui un'aritmetica finita implementi il troncamento è: $u = b^{1-m}$, da cui si ottiene che:

$$u = \frac{b}{b^m} \Rightarrow b^m = \frac{b}{u} \Rightarrow m = log_b\left(\frac{b}{u}\right) \Rightarrow m = log_b b - log_b u \Rightarrow m = 1 - log_b u$$

  Essendo che il numero di cifre decimali nella mantissa è $m-1$, si ha che $m = -log_b u$. Ponendo infine $b = 10$ si ha il risultato cercato.

- **Arrotondamento:** sappiamo che la precisione di macchina nel caso in cui un'aritmetica finita implemento il troncamento è: $u = \frac{1}{2}b^{1-m}$, da cui si ottiene che:

$$2u = b^{1-m} \Rightarrow 1 - m = log_b(2u) \Rightarrow m = 1 - log_b(2u) \Rightarrow m = 1 - log_b 2 - log_b u,$$

  Ponendo $b = 2$ si ha che $m = -log_2 u$. Applicando infine la formula del cambiamento di base:

$$m = -\frac{log_{10} u}{log_{10} 2} \approx -log_{10} u.$$

Soluzione 1.9 Il non funzionamento del codice è dovuto alla rappresentazione in floating point del valore di delta pari a $0.0\overline{0011}$ la quale è affetta da errore di round-off. In questo modo il valore effettivo di delta è circa 0.0999.... Se $\frac{1}{10}$ non fosse affetto da questo errore, ci aspetteremmo che il ciclo compia dieci iterazioni, ma per l'errore osservato per $i = 10$, $x < 1$ mentre per $i = 11$, $x > 1$. Pertanto il ciclo non termina.

Soluzione 1.10 Il calcolo dell'espressione $\sqrt{x^2 + y^2}$ potrebbe presentare errori di overflow in quanto x e y potrebbero essere numeri rappresentabili in macchina, mentre i rispettivi quadrati troppo grandi per essere rappresentati. Per evitare questo problema si calcola $m = max(|x|, |y|)$, si raccoglie all'interno della radice il valore ottenuto, e portandolo fuori si ottiene:

$$m\sqrt{\left(\frac{x}{m}\right)^2 + \left(\frac{y}{m}\right)^2}.$$

Ponendo ad esempio $|x| > |y|$ otteniamo:

$$x\sqrt{1 + \left(\frac{y}{x}\right)^2},$$

dove $\frac{y}{x} < 1$ per ipotesi, quindi sicuramente il suo quadrato è rappresentabile in macchina.

Soluzione 1.11 Siano $R = (x + y) + z$ e $\hat{R} = (x \oplus y) \oplus z$, si ha che:

$$\hat{R} = (x \oplus y) \oplus z = fl(fl(fl(x) + fl(y)) + fl(z)) =$$
$$= fl(fl(x(1+\epsilon_x) + y(1+\epsilon_y)) + z(1+\epsilon_z)) =$$
$$= fl((x(1+\epsilon_x) + y(1+\epsilon_y))(1+\epsilon_1) + z(1+\epsilon_z)) =$$
$$= ((x(1+\epsilon_x) + y(1+\epsilon_y))(1+\epsilon_1) + z(1+\epsilon_z))(1+\epsilon_2) =$$
$$= (x + y + z + x\epsilon_x + y\epsilon_y + z\epsilon_z + x\epsilon_x\epsilon_1 + y\epsilon_y\epsilon_1 + (x+y)\epsilon_1)(1+\epsilon_2) =$$
$$= x + y + z + (x+y+z)\epsilon_2 + x\epsilon_x + y\epsilon_y + z\epsilon_z + x\epsilon_x\epsilon_2 + y\epsilon_1\epsilon_2 + z\epsilon_z\epsilon_2 + (x+y)\epsilon_1 + (x+y)\epsilon_1\epsilon_2 =$$
$$= R + R\epsilon_2 + x\epsilon_x + y\epsilon_y + z\epsilon_z + (x+y)\epsilon_1.$$

Sono stati trascurati gli infinitesimi di ordine superiore al primo. L'errore relativo su $\hat{R}$ è quindi:

$$\epsilon_R = \frac{\hat{R} - R}{R} = \frac{R + R\epsilon_2 + x\epsilon_x + y\epsilon_y + z\epsilon_z + (x+y)\epsilon_1 - R}{R} =$$

$$= \frac{R\epsilon_2 + x\epsilon_x + y\epsilon_y + z\epsilon_z + (x+y)\epsilon_1}{R} \Rightarrow$$

$$\Rightarrow |\epsilon_R| = \frac{|R\epsilon_2 + x\epsilon_x + y\epsilon_y + z\epsilon_z + (x+y)\epsilon_1|}{|R|} \leq$$

$$\leq \frac{|R||\epsilon_2| + |x||\epsilon_x| + |y||\epsilon_y| + |z||\epsilon_z| + |x+y||\epsilon_1|}{|R|}.$$

Sia U la precisione di macchina, ricordando che:

$$|\epsilon_x|, |\epsilon_y|, |\epsilon_z|, |\epsilon_1|, |\epsilon_2| \leq U$$

si ha:

$$|\epsilon_R| \leq \frac{|R|U + |x|U + |y|U + |z|U + |x+y|U}{|R|} =$$

$$= U\left(\frac{|x| + |z| + |x+y| + |R|}{|R|}\right) =$$

$$= U\left(\frac{|x| + |y| + |z|}{|R|} + \frac{|x+y|}{|R|} + 1\right) =$$

$$= U\left(\frac{|x| + |y| + |z|}{|x+y+z|} + \frac{|x+y|}{|x+y+z|} + 1\right).$$

Quindi l'errore relativo su $\hat{R}$ sarà tale che:

$$|\epsilon_R| \leq U\left(\frac{|x| + |y| + |z|}{|x+y+z|} + \frac{|x+y|}{|x+y+z|} + 1\right).$$

Ponendo $R = x + (y+z)$, $\hat{R} = x \oplus (y \oplus z)$ ed eseguendo il medesimo ragionamento si ottiene una stima dell'errore relativo pari a

$$|\epsilon_R| \leq U\left(\frac{|x| + |y| + |z|}{|x+y+z|} + \frac{|y+z|}{|x+y+z|} + 1\right).$$

Da questi risultati si evince che non vale la proprietà associativa per la somma algebrica in aritmetica finita.

Soluzione 1.12 Considerando un problema matematico formalizzato come

$$y = f(x)$$

il numero di condizionamento del problema è dato da:

$$\kappa = \left| f'(x)\frac{x}{y} \right|.$$

In questo caso

$$y = \sqrt{x}, \qquad f'(x) = \frac{1}{2\sqrt{x}}$$

si ha quindi

$$\kappa = \left| \frac{1}{2\sqrt{x}} \cdot \frac{x}{\sqrt{x}} \right| = \frac{1}{2}.$$

Soluzione 1.13 Poichè $\frac{4}{3}$ è un numero razionale periodico, il suo floating sarà affetto da un certo errore di rappresentazione. Se valutiamo l'argomento del logaritmo al numeratore per $x = \frac{4}{3}$, il cui valore esatto è 0, otteniamo $2.220446049250313e-16$. Quindi valutando la funzione in $fl\left(\frac{4}{3}\right)$ si ottiene un valore finito.

2 Radici di un'equazione

Soluzione 2.1 Ricordando che il metodo di Newton per calcolare la radice di un'equazione è:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}.$$

Come prima cosa, dobbiamo scegliere una $f(x)$ che abbia come radice $\sqrt{\alpha}$. Scegliamo quindi $f(x) = x^2 - \alpha$ e calcoliamo la sua derivata prima: $f'(x) = 2x$. Sostituendo nel metodo di Newton abbiamo che:

$$x_{i+1} = x_i - \frac{x_i^2 - \alpha}{2x_i} = \frac{2x_i^2 - x_i^2 + \alpha}{2x_i} = \frac{x_i^2 + \alpha}{2x_i} = \frac{1}{2}\left(\frac{x_i^2}{x_i} + \frac{\alpha}{x_i}\right) =$$

$$= \frac{1}{2}\left(x_i + \frac{\alpha}{x_i}\right).$$

Il codice Matlab che esprime questo risultato è il seguente:

```
function [] = metodoNewton2_1(alpha, x0, imax, tolx)
    format long e
    fprintf('x0 =\n'), disp(x0)
    x=(x0+alpha/x0)/2;
    fprintf('x1 =\n'), disp(x)
    i=1;
    while(i<imax) && (abs(x-x0)>tolx)
        i = i+1;
        x0 = x;
        x =(x0+alpha/x0)/2;
        fprintf('x%d =\n', i), disp(x)
    end
    if abs(x-x0)>tolx
        disp('Il metodo non converge')
    else
        disp('Il metodo converge')
    end
```

Dal quale si estraggono i seguenti risultati nel caso in cui $\alpha = x_0 = 3$:

i	x_i
0	3.000000000000000
1	2.000000000000000
2	1.750000000000000
3	1.732142857142857
4	1.732050810014727
5	1.732050807568877
6	1.732050807568877

Dalla tabella si evince che già con $i = 5$ si ha un errore di convergenza nell'ordine di 10^{-16}.

Soluzione 2.2 Per generalizzare il risultato precedente dobbiamo considerare la funzione generica $f(x) = x^n - \alpha$ che ha come radice $\sqrt[n]{\alpha}$. La derivata prima è: $f'(x) = nx^{n-1}$. Sostituendo nel metodo di Newton abbiamo che:

$$x_{i+1} = x_i - \frac{x_i^n - \alpha}{nx_i^{n-1}} = \frac{nx_i^n - x_i^n + \alpha}{nx_i^{n-1}} = \frac{x_i n(n-1) + \alpha}{nx_i^{n-1}} =$$

$$= \frac{1}{n}\left(\frac{x_i^n(n-1)}{x_i^{n-1}} + \frac{\alpha}{x_i^{n-1}}\right) =$$

$$= \frac{1}{n}\left(x_i(n-1) + \frac{\alpha}{x_i^{n-1}}\right).$$

Consideriamo adesso i casi:

- $\alpha = 3, \qquad n = 3$

$$x_{i+1} = \frac{1}{3}\left(2x_i + \frac{3}{x_i^2}\right)$$

```
function [] = metodoNewton2_2_3(alpha, x0, imax, tolx)
    format long e
    fprintf('x0 =\n'), disp(x0)
    x=(2*x0+alpha/x0^2)/3;
    fprintf('x1 =\n'), disp(x)
    i=1;
    while(i<imax) && (abs(x-x0)>tolx)
        i = i+1;
        x0 = x;
        x =(2*x0+alpha/x0^2)/3;
        fprintf('x%d =\n', i), disp(x)
    end
    if abs(x-x0)>tolx
        disp('Il metodo non converge')
    else
        disp('Il metodo converge')
    end
```

- $\alpha = 3, \qquad n = 4$

$$x_{i+1} = \frac{1}{4}\left(3x_i + \frac{3}{x_i^3}\right)$$

```
function [] = metodoNewton2_2_4(alpha, x0, imax, tolx)
    format long e
    fprintf('x0 =\n'), disp(x0)
    x=(3*x0+alpha/x0^3)/4;
    fprintf('x1 =\n'), disp(x)
    i=1;
    while(i<imax) && (abs(x-x0)>tolx)
        i = i+1;
        x0 = x;
        x =(3*x0+alpha/x0^3)/4;
        fprintf('x%d =\n', i), disp(x)
    end
    if abs(x-x0)>tolx
        disp('Il metodo non converge')
    else
        disp('Il metodo converge')
    end
```

- $\alpha = 3, \qquad n = 5$

$$x_{i+1} = \frac{1}{5}\left(4x_i + \frac{3}{x_i^4}\right)$$

```
function [] = metodoNewton2_2_4(alpha, x0, imax, tolx)
    format long e
    fprintf('x0 =\n'), disp(x0)
    x=(3*x0+alpha/x0^3)/4;
    fprintf('x1 =\n'), disp(x)
    i=1;
    while(i<imax) && (abs(x-x0)>tolx)
        i = i+1;
        x0 = x;
        x =(3*x0+alpha/x0^3)/4;
        fprintf('x%d =\n', i), disp(x)
    end
    if abs(x-x0)>tolx
        disp('Il metodo non converge')
    else
        disp('Il metodo converge')
    end
```

Soluzione 2.3 Ricordiamo che il metodo delle secanti per calcolare la radice di un'equazione è:

$$x_{i+1} = \frac{f(x_i)x_{i-1} - f(x_{i-1})x_i}{f(x_i) - f(x_{i-1})}$$

Come prima cosa, dobbiamo scegliere una $f(x)$ che abbia come radice $\sqrt{\alpha}$. Scegliamo quindi $f(x) = x^2 - \alpha$. Sostituendo nel metodo delle secanti abbiamo che:

$$x_{i+1} = \frac{(x_i^2 - \alpha)x_{i-1} - (x_{i-1}^2 - \alpha)x_i}{(x_i^2 - \alpha) - (x_{i-1}^2 - \alpha)} =$$

$$= \frac{x_i^2 x_{i-1} - x_{i-1}\alpha - x_{i-1}^2 x_i + x_i\alpha}{x_i^2 - \alpha - x_{i-1}^2 + \alpha} =$$

$$= \frac{x_i x_{i-1}(x_i - x_{i-1}) + \alpha(x_i - x_{i-1})}{x_i^2 - x_{i-1}^2} =$$

$$= \frac{(x_i x_{i-1} + \alpha)(x_i - x_{i-1})}{(x_i - x_{i-1})(x_i + x_{i-1})} =$$

$$= \frac{x_i x_{i-1} + \alpha}{x_i + x_{i-1}}.$$

Il codice Matlab che esprime questo risultato è il seguente:

```
function [] = metodoSecanti2_3(alpha, x0, imax, tolx)
    format long e
    fprintf('x0 =\n'), disp(x0)
%   calcolo la prima approssimazione con il metodo di Newton
    x=(x0+alpha/x0)/2;
    fprintf('x1 =\n'), disp(x)
    i=1;
    while(i<imax) && (abs(x-x0)>tolx)
        i = i+1;
        x1 = (x*x0 + alpha)/(x+x0);
        x0 = x;
        x = x1;
        fprintf('x%d =\n', i), disp(x)
    end
    if abs(x-x0)>tolx
        disp('Il metodo non converge')
    else
        disp('Il metodo converge')
    end
```

Dal quale si estraggono i seguenti risultati nel caso in cui $\alpha = x_0 = 3$:

i	x_i
0	3.000000000000000
1	2.000000000000000
2	1.800000000000000
3	1.736842105263158
4	1.732142857142857
5	1.732050934706042
6	1.732050807572256
7	1.732050807568877
8	1.732050807568877

I dati in tabella ci mostrano che il metodo delle secanti converge leggermente più lentamente rispetto al metodo di Newton.

Soluzione 2.4

```
function [y, i] = Newton(f, f1, x0, imax, tolx)
    format long;
    fx = feval(f, x0);
    f1x = feval(f1, x0);
    x = x0-fx/f1x;
    i=0;
    while((i < imax) && (abs(x-x0)>tolx))
        x0 = x;
        i = i+1;
        fx = feval(f, x0);
        f1x = feval(f1, x0);
        x = x0 - fx / f1x;
    end
    y = x;
    if (abs(x-x0)>tolx)
```

```
        disp('il metodo non converge')
    end
end
```

```
function [y, i] = NewtonM(f, f1, x0, imax, tolx, m)
    format long;
    i=0;
    vai = 1;
    while((i < imax) && vai)
        i = i+1;
        fx = feval(f, x0);
        f1x = feval(f1, x0);
        x = x0 - m*fx / f1x;
        x0 = x;
        if (abs(x-x0)<tolx)
            vai = 0;
        end
    end
    y = x;
    if (abs(x-x0)>tolx)
        disp('il metodo non converge')
    end
end
```

```
function [y,i] = Aitken( f, f1, x0, imax, tolx )
    format long;
    i = 0;
    x=x0;
    vai = 1;
    while((i < imax) && vai)
        x0 = x;
        x1 = Newton(f, f1, x0, imax, tolx);
        x = Newton(f, f1, x1, imax, tolx);
        i = i+1;
        if((x0 - 2*x1 +x) == 0)
            disp('Errore, impossibile dividere per 0');
            break;
        end
        x = (x*x0 - x1^2)/(x0 - 2*x1 +x);
        if (abs(x-x0)>tolx)
            vai = 0;
        end
    end
    y = x;
end
```

```
f = @(x) (x - pi)^10;
f1 = @(x) 10 * (x - pi)^9;

[a, b] = Newton(f, f1, 5, 1000, 10^-12);
[a, b] = NewtonM(f, f1, 5, 1000, 10^-12, 10);
[a, b] = Aitken(f, f1, 5, 1000, 10^-12);

```

```
g = @(x) ((x - pi)^10) * (exp(1)^(10*x));
g1 = @(x) (exp(1)^(10*x)*(10*x-10*pi+1))

[a, b] = Newton(g, g1, 5, 1000, 10^-12);
[a, b] = NewtonM(g, g1, 5, 1000, 10^-12, 10);
[a, b] = Aitken(g, g1, 5, 1000, 10^-12);
```

Per la funzione f_1 e tolx=10^{-12} i risultati sono i seguenti:

- Newton:
  $x = 3.141592653598136$ in 247 iterazioni
- Newton modificato:
  $x = 3.141592653589793$ in 1 iterazione
- Aitken:
  $x = 3.141592653597301$ in 1 iterazione

Per la funzione f_2 e tolx=10^{-12} i risultati sono i seguenti:

- Newton:
  $x = 3.141592653598037$ in 251 iterazioni
- Newton modificato:
  $x = 3.645161148540612$ in 1 iterazione
- Aitken:
  $x = 3.141592653597213$ in 1 iterazione

Soluzione 2.5 No, non è possibile utilizzare il metodo di bisezione per determinare lo zero delle funzioni del precedente esercizio poichè, per entrambe, si ha che $f(x) > 0, \forall x \in \mathbb{R}$. Viene meno, dunque, la condizione necessaria per l'applicazione del metodo di bisezione: $f(x)$ deve infatti essere continua in un intervallo $[a, b]$ tale che $f(a)f(b) < 0$.

Soluzione 2.6 Applicando i metodi di Newton, corde e secanti per determinare lo zero della funzione

$$f(x) = 1 - x - \left(1 + \frac{\cos 10x}{2}\right) \sin x.$$

con punto iniziale $x_0 = 0$, si ottengono i seguenti risultati:

`tolx`	Metodo di Newton	Metodo delle corde	Metodo delle secanti
10^{-1}	$x^* = 4.889619218476237\text{e-}01$, $i = 3$	$x^* = 4.562704641042116\text{e-}01$, $i = 3$	$x^* = 4.890581175677273\text{e-}01$, $i = 4$
10^{-2}	$x^* = 4.889437945327571\text{e-}01$, $i = 4$	$x^* = 4.850806992335395\text{e-}01$, $i = 9$	$x^* = 4.890581175677273\text{e-}01$, $i = 4$
10^{-3}	$x^* = 4.889437945327571\text{e-}01$, $i = 4$	$x^* = 4.892902238113615\text{e-}01$, $i = 16$	$x^* = 4.889437290085025\text{e-}01$, $i = 5$
10^{-4}	$x^* = 4.889437945327571\text{e-}01$, $i = 4$	$x^* = 4.889126087612141\text{e-}01$, $i = 23$	$x^* = 4.889437943738723\text{e-}01$, $i = 6$
10^{-5}	$x^* = 4.889437943773993\text{e-}01$, $i = 5$	$x^* = 4.889398354775753\text{e-}01$, $i = 29$	$x^* = 4.889437943738723\text{e-}01$, $i = 6$
10^{-6}	$x^* = 4.889437943773993\text{e-}01$, $i = 5$	$x^* = 4.889441506693744\text{e-}01$, $i = 36$	$x^* = 4.889437943738723\text{e-}01$, $i = 6$
10^{-7}	$x^* = 4.889437943773993\text{e-}01$, $i = 5$	$x^* = 4.889437623118317\text{e-}01$, $i = 43$	$x^* = 4.889437943738723\text{e-}01$, $i = 6$
10^{-8}	$x^* = 4.889437943773993\text{e-}01$, $i = 5$	$x^* = 4.889437903067187\text{e-}01$, $i = 49$	$x^* = 4.889437943773993\text{e-}01$, $i = 7$
10^{-9}	$x^* = 4.889437943773993\text{e-}01$, $i = 5$	$x^* = 4.889437947437524\text{e-}01$, $i = 56$	$x^* = 4.889437943773993\text{e-}01$, $i = 7$
10^{-10}	$x^* = 4.889437943773993\text{e-}01$, $i = 6$	$x^* = 4.889437943444282\text{e-}01$, $i = 63$	$x^* = 4.889437943773993\text{e-}01$, $i = 7$
10^{-11}	$x^* = 4.889437943773993\text{e-}01$, $i = 6$	$x^* = 4.889437943803667\text{e-}01$, $i = 70$	$x^* = 4.889437943773993\text{e-}01$, $i = 7$
10^{-12}	$x^* = 4.889437943773993\text{e-}01$, $i = 6$	$x^* = 4.889437943777760\text{e-}01$, $i = 76$	$x^* = 4.889437943773993\text{e-}01$, $i = 8$

Data una $f(x)$ con radice x^* il numero di condizionamento κ è:

$$\kappa = \frac{1}{|f'(x^*)|}$$

Nel nostro caso:

$$f'(x) = 5 \sin 10x \sin x - \cos x \left(\frac{\cos 10x}{2} + 1 \right) - 1$$

e $x^* \approx 0.48894$. Quindi:

$$\kappa_{x^*} \approx \frac{1}{|-4.27233|} \approx 0.23406.$$

Soluzione 2.7

```
function [a,j]=bisezione(f,a,b,tolx,imax)
    nit=imax;
    j=0;
    if(subs(f,a)*subs(f,b)>0)
        disp('errore')
    else
        while(j<=imax)
            sol=(a+b)/2;
            ff=subs(f,sol);
            if(abs(ff)<=tolx)
                nit=j;
                break;
        end
        fa=subs(f,a);
        fm=subs(f,sol);
        if(fa*fm<=0)
            b=sol;
        else
            a=sol;
        end
        j=j+1;
    end
end
```

Il codice sopra riportato trova il risultato $x = 0.488943794376610$ in 39 iterazioni con una tolleranza di 10^{-12}.

Soluzione 2.8 Dopo 245 iterazioni il metodo trova il risultato $x = 3.141592653581401$.

3 Risoluzione di sistemi lineari

Soluzione 3.1 $A, B \in \mathbb{R}^{n \times n}$ sono triangolari superiori (inferiori) quindi $a_{ij} = b_{ij} = 0$ per $i > j$ $(i < j)$.

- Somma
  Sia $C = A + B$, ovvero $c_{ij} = a_{ij} + b_{ij}$ per $1 \leq i, j \leq n$. Quindi, per $i > j$ $(i < j)$:

  $$c_{ij} = a_{ij} + bij = 0 + 0 = 0$$

  Concludiamo dunque che la struttura di C non cambia.

- Prodotto
  Sia $C = A \times B$ ovvero $c_{ij} = \sum_{K=1}^{n} a_{ik} b_{kj}$ per $1 \leq i, j \leq n$. Per ricavare c_{ij} dobbiamo considerare i termini della sommatoria da j a i: $c_{ij} = \sum_{K=j}^{i} a_{ik} b_{kj}$. Se $i < j$, $c_{ij} = 0$, ovvero C è una matrice triangolare inferiore. Per matrici triangolari superiori il procedimento è analogo.

Soluzione 3.2 Per l'esercizio 3.1 abbiamo che il prodotto di matrici triangolari inferiori (superiori) è ancora una matrice triangolare inferiore (superiore). Rimane da osservare gli elementi sulla diagonale. Siano A, $B \in \mathbb{R}^{n\times n}$ due matrici triangolari inferiori (superiori) a diagonale unitaria. La matrice prodotto $P = A \times B$ ha elementi sulla diagonale della forma:

$$p_{ii} = a_{ii}b_{ii} = 1 \cdot 1 = 1 \qquad \forall i = 1 \dots n$$

Quindi P ha diagonale unitaria.

Soluzione 3.3 $A \in \mathbb{R}^{n\times n}$ una matrice triangolare inferiore non singolare ($a_{ii} \neq 0$, $i = 1 \dots n$).
Per induzione su n:

- $n = 1$: $A = a_{11}$, A è uno scalare e la proprietà è verificata.
- $n - 1 \to n$:

$$A = \left(\begin{array}{c|c} A_{n-1} & \mathbf{0} \\ \hline \boldsymbol{a}^\top & a_{nn} \end{array}\right)$$

utilizzando lì'ipotesi induttiva per A_{n-1} si ha $A^{-1} = \left(\begin{array}{c|c} A_{n-1}^{-1} & \mathbf{0} \\ \hline \boldsymbol{v}^\top & w \end{array}\right) \to AA^{-1} = I$:

$$AA^{-1} = \left(\begin{array}{c|c} A_{n-1} & \mathbf{0} \\ \hline \boldsymbol{a}^\top & a_{nn} \end{array}\right) \cdot \left(\begin{array}{c|c} A_{n-1}^{-1} & \mathbf{0} \\ \hline \boldsymbol{v}^\top & w \end{array}\right) =$$

$$= \left(\begin{array}{c|c} A_{n-1}A_{n-1}^{-1} + \mathbf{0}\boldsymbol{v}^\top & A_{n-1}\mathbf{0} + \mathbf{0}w \\ \hline \boldsymbol{a}^\top A_{n-1}^{-1} + a_{nn}\boldsymbol{v}^\top & \boldsymbol{a}^\top\mathbf{0} + a_{nn}w \end{array}\right) =$$

$$= \left(\begin{array}{c|c} I & \mathbf{0} \\ \hline \boldsymbol{a}^\top A_{n-1}^{-1} + a_{nn}\boldsymbol{v}^\top & a_{nn}w \end{array}\right) = I \to$$

$$\to \begin{cases} \boldsymbol{a}^\top A_{n-1}^{-1} + a_{nn}\boldsymbol{v}^\top = \mathbf{0} \\ a_{nn}w = 1 \end{cases} \to \begin{cases} \boldsymbol{v}^\top = -\frac{\boldsymbol{a}^\top A_{n-1}^{-1}}{a_{nn}} \\ w = \frac{1}{a_{nn}} \end{cases}$$

Se la matrice ha diagonale unitaria è anche la sua inversa: $w = \frac{1}{a_{nn}} = 1$. Procedimento analogo per le matrici triangolari superiori.

Soluzione 3.4 Il generico passo di fattorizzazione è descritto dal seguente codice:

```
1        for i=1:n-1
2            if A(i,i)==0 disp('La matrice non e` fattorizzabile'),
                 break, end
3            A(i+1:n,i)=A(i+1:n,i)/A(i,i)
4            A(i+1:n,i)=A(i+1:n, i+1:n)-A(i+1:n,i)*A(i,i+1:n)
5        end
```

Ad ogni passo, la riga 3 esegue una divisione per $n - i$ componenti. La riga 4 invece esegue due operazioni per $n - 1$ componenti, quindi otteniamo:

$$\sum_{i=1}^{n-1}(n-i) + 2[(n-i)^2] = \sum_{i=1}^{n-1}(i + 2i^2) = \frac{n(n-1)}{2} + 2\frac{n(n-1)(2n-1)}{6} = \dots = \frac{2}{3}n^3 + \frac{1}{2}n^2 - \frac{1}{6}n \approx \frac{2}{3}n^3$$

Soluzione 3.5

```
function [L,U,P]=es3_5(A)
[m,n]=size(A);
if m~=n
    error('matrice non quadrata');
end
L=eye(n);
P=eye(n);
U=A;
for k=1:n
    [pivot, m]=max(abs(U(k:n,k)));
    if pivot==0
        error('Errore: Matrice singolare');
    end
    m=m+k-1;
        if m~=k
        % scambio le righe m e k
        U([k,m], :) = U([m, k], :);
        P([k,m], :) = P([m, k], :);
        if k >= 2
            L([k,m],1:k-1) = L([m,k],1:k-1);
        end
    end
    L(k+1:n,k)=U(k+1:n,k)/U(k,k);
    U(k+1:n,:)=U(k+1:n,:)-L(k+1:n,k)*U(k,:);
end
```

Soluzione 3.6

```
function [b] = solveLinearLUP(L, U, P, b)
    b = TriangolareInf(L,P*b);
    b = TriangolareSup(U,b);
end
```

La funzione restituisce la soluzione del sistema lineare $A\boldsymbol{x} = b$ con A matrice triangolare inferiore utilizzando il metodo di backsubstitution. Il vincolo sulla matrice è che deve essere nonsingolare.

```
function [b] = TriangolareInf(A, b)
    for j=1:length(A)
        if A(j,j) ~= 0
            b(j) = b(j)/A(j,j);
        else
            error('le matrice e` singolare')
        end
        for i=j+1:length(A)
            b(i) = b(i)-A(i,j)*b(j);
        end
    end
end
```

La funzione risolve i sistemi triangolari superiori per colonna.

```
function [b] = TriangolareSup(A, b)
    for j=size(A):-1:1
```

```
        if A(j,j)==0
            error('Matrice non singolare')
        else
            b(j)=b(j)/A(j,j);
        end
        for i=1:j-1
            b(i)=b(i)-A(i,j)*b(j);
        end
    end
end
```

Soluzione 3.7 Una matrice $A \in \mathbb{R}^{n\times n}$ è sdp se è simmetrica e se $\forall \boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{x} \neq \mathbf{0}$, risulta $\boldsymbol{x}^\top A\boldsymbol{x} > 0$. Poiché si ha che $det(A) = det(A^\top)$:

$$det(A^\top A) = det(AA^\top) = det(A)det(A^\top) = det(A)^2$$

Quindi se A è non singolare anche $A^\top A$ e $AA^\top$ sono non singolari. Inoltre:

$$(AA^\top)^\top = (A^\top)^\top A^\top = AA^\top,$$

$$(A^\top A)^\top = A^\top (A^\top)^\top = A^\top A,$$

quindi $AA^\top$ e $A^\top A$ sono simmetriche. Affinché siano sdp si deve anche avere che, $\forall \boldsymbol{v} \in \mathbb{R}^n$, $\boldsymbol{v} \neq \mathbf{0}$:

$$\boldsymbol{v}^\top AA^\top \boldsymbol{v} = (A^\top \boldsymbol{v})^\top (A^\top \boldsymbol{v}) > 0,$$

$$\boldsymbol{v}^\top A^\top A\boldsymbol{v} = (A\boldsymbol{v})^\top (A\boldsymbol{v}) > 0,$$

le quali sono verificate, infatti le quantità $A\boldsymbol{v}$ e $A^\top \boldsymbol{v}$ sono maggiori di zero perché A è non singolare.

Soluzione 3.8 Sia $A \in \mathbb{R}^{m\times n}$. Se $m \geq n = \text{rank}(A)$ allora la matrice ha tutte le colonne indipendenti, quindi ha rango massimo, quindi $det(A) \neq 0$, cioè è non singolare. Per la dimostrazione dell'esercizio 3.7, se A è una matrice non sigolare, $AA^\top$ è sdp.

Soluzione 3.9 Applichiamo la proprietà distributiva e la somma tra matrici:

$$A = \frac{1}{2}(A + A^\top) + \frac{1}{2}(A - A^\top) = \frac{1}{2}A + \frac{1}{2}^\top + \frac{1}{2}A - \frac{1}{2}A^\top.$$

Dobbiamo dimostrare che A_s è simmetrica, cioè che $A_s = A_s^\top$:

$$\frac{1}{2}(A + A^\top) = \left[\frac{1}{2}(A + A^\top)\right]^\top = \frac{1}{2}\left(A + A^\top\right)^\top = \frac{1}{2}\left(A^\top + \left(A^\top\right)^\top\right) = \frac{1}{2}\left(A^\top + A\right) = \frac{1}{2}\left(A + A^\top\right)$$

Non ci resta che dimostrare che A_a è antisimmetrica, cioè che $A_a = -A_a^\top$:

$$\frac{1}{2}(A - A^\top) = -\left[\frac{1}{2}(A - A^\top)\right]^\top = -\frac{1}{2}\left(A - A^\top\right)^\top = -\frac{1}{2}\left(A^\top - \left(A^\top\right)^\top\right) = -\frac{1}{2}\left(A^\top - A\right) = \frac{1}{2}\left(A - A^\top\right)$$

$$\boldsymbol{x}^\top A\boldsymbol{x} = \boldsymbol{x}^\top A_s\boldsymbol{x} \rightarrow \boldsymbol{x}^\top A_a\boldsymbol{x} = 0 \left(\boldsymbol{x}^\top A\boldsymbol{x} = \boldsymbol{x}^\top A_s\boldsymbol{x} + \boldsymbol{x}^\top A_a\boldsymbol{x}\right).$$

Non ci resta che calcolare:

$$\boldsymbol{x}^\top A_a\boldsymbol{x} = \boldsymbol{x}^\top \left(\frac{1}{2}\left(A - A^\top\right)\right)\boldsymbol{x} = \frac{1}{2}\left(\boldsymbol{x}^\top A\boldsymbol{x} - \boldsymbol{x}^\top A^\top \boldsymbol{x}\right) = \frac{1}{2}\left(\boldsymbol{x}^\top A\boldsymbol{x} - (A\boldsymbol{x})^\top \boldsymbol{x}\right) = \frac{1}{2}\left(\boldsymbol{x}^\top \boldsymbol{y} - \boldsymbol{x}^\top \boldsymbol{y}\right).$$

Poiché $\boldsymbol{x}^\top \boldsymbol{y} = \boldsymbol{x}^\top \boldsymbol{y} \rightarrow \boldsymbol{x}^\top A_a\boldsymbol{x} = 0$.

Soluzione 3.10 Il calcolo di L rappresenta il maggior contributo al costo computazionale, infatti l'algoritmo esegue:

- $j-1$ somme di 2 prodotti
- 1 sottrazione
- 1 divisione

Il costo è dunque di $2\,(j-1)+2 = 2j$ `flops`. Sappiamo inoltre che L è una matrice triangolare, quindi questo costo va moltiplicato $n-j$ volte per ciascuna colonna. Il costo totale dell'algoritmo è quindi:

$$\sum_{j=1}^{n} 2j(n-j) = 2n\sum_{j=1}^{n} j - 2\sum_{j=1}^{n} j^2 = 2n\frac{n(n+1)}{2} - 2\frac{n(n+1)(2n+1)}{6} \approx n^3 - \frac{2}{3}n^3 = \frac{1}{3}n^3 \quad \texttt{flops}$$

Soluzione 3.11

```
function [L,D] = es3_11(A)
    [m,n]=size(A);
    if m~=n
        error('la matrice non quadrata')
    end
    if A(1,1)<=0
        error('la matrice non sdp!')
    end
    A(2:n,1)=A(2:n,1)/A(1,1);
    for j=2:n
        v = (A(j,1:(j-1))').*diag(A(1:(j-1),1:(j-1)));
        A(j,j) = A(j,j)-A(j,1:(j-1))*v;
        if A(j,j)<=0
            error('la matrice non sdp!');
        end
        A((j+1):n,j)=(A((j+1):n,j)-A((j+1):n,1:(j-1))*v)/A(j,j);
    end
    if nargout==1
        L=A;
    else
    for j=1:n
      for i=1:n
           if i==j
                D(i,j) = A(i,j);
                L(i,j) = 1;
          end
            if i>j
                D(i,j) = 0;
                L(i,j) = A(i,j);
            end
             if i<j
                D(i,j) = 0;
                L(i,j) = 0;
            end
        end
    end
    end
end
```

Soluzione 3.12

```
function [b] = es3_12(L, D, b)
    b = TriangolareInf(L,b');
    b = diagonal(diag(D), b');
    b = TriangolareSup(L',b');
end
```

Soluzione 3.13

$$A_1 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 2 & 2 \\ 1 & 2 & 3 & 3 \\ 1 & 2 & 3 & 4 \end{pmatrix}, \qquad A_2 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 2 & 2 \\ 1 & 2 & 3 & 3 \\ 1 & 2 & 3 & 2 \end{pmatrix}$$

Applicando il codice dell'esercizio 3.11 riceviamo il messaggio di errore "Matrice non sdp" soltanto per la matrice A_2. La matrice A_1 è quindi sdp.

Soluzione 3.14 La matrice A scelta per l'esercizio è:

$$A = \begin{pmatrix} 1 & -2 & 9 \\ 0 & 9 & 8 \\ 2 & 4 & 1 \end{pmatrix}$$

Il vettore $\boldsymbol{b}$ è:

$$\boldsymbol{b} = (4, 2, 3)^\top$$

Utilizzando il codice implementato negli esercizi 3.5 e 3.6 otteniamo il vettore delle incognite:

$$\boldsymbol{x} = (1.414746543778802, -0.02764976958525346, 0.2811059907834101)^\top$$

Quindi il vettore residuo è:

$$\boldsymbol{r} = A\boldsymbol{x} - \boldsymbol{b} = (0, 0, 0)^\top$$

Quindi il risultato è corretto. Ora utilizziamo la matrice:

$$B = \begin{pmatrix} 9 & 2 & 3 \\ 2 & 9 & 4 \\ 3 & 4 & 6 \end{pmatrix}$$

Utilizzando il codice implementato negli esercizi 3.11 e 3.12 otteniamo il vettore delle incognite:

$$\boldsymbol{x} = (0.1446456521739130, -0.02176521739130439, 0.6126934782608696)$$

Quindi il vettore residuo è:

$$\boldsymbol{r} = (0, -4.440892098500626 \times 10^{-16}, 0)^\top$$

```
% prima parte
A = [1 -2 9; 0 9 8; 2 4 1]
[L, U, P] = es3_5(A)
b = [4 2 3]'
x = es3_6(L, U, P, b)
r = A*x' -b

% seconda parte
B = [14 5 2;5 8 1;2 1 4]
[L, D] = es3_11(B)
b = [4 2 3]'
x = es3_12(L, D, b)
r = B*x' -b
```

Soluzione 3.15

```
v = [1, 1, 1, 1, 1, 1, 1, 1, 1]
A = (diag(v*(-100),-1))+eye(10)
norm(A, inf)
norm(inv(A), inf)
norm(A, 1)
norm(inv(A), 1)
cond(A, inf)
```

$\kappa_\infty(A) = 101 \times 1.01 \times 10^{18}$. Analiticamente abbiamo che $||A||_\infty = ||A||_1 = 101$, $||A^{-1}||_\infty = ||A^{-1}||_1 = 1.01 \times 10^{18}$. Quindi $\kappa_\infty(A) = \kappa_1(A)$. Poiché $\kappa_\infty(A) >> 1$ il problema è mal condizionato, infatti con l'istruzione `cond` di Matlab si ottiene un warning per cui il reciproco del numero di condizionamento `RCOND` $= 9.801980 \times 10^{-21}$, confermando quanto detto.

Soluzione 3.16 Il confronto fra il vettore $\boldsymbol{x}$ con il vettore residuo $\boldsymbol{r}$ e fra il vettore $\boldsymbol{y}$ con il vettore residuo $\boldsymbol{r1}$ mostra che la soluzione proposta è corretta.

```
v = [1, 1, 1, 1, 1, 1, 1, 1, 1];
A = (diag(v*(-100),-1))+eye(10);

b = [1 -99*ones(1,9)]';
c = 0.1*[1 99*ones(1,9)]';
x = ones(10, 1);
y = 0.1*x;

r = linesolve(A, b);
r1 = linesolve(A,c);

x(1)=b(1);
for i=2:10
        x(i)=b(i)+100 * x(i - 1);
end
x=x(:);

y(1)=c(1);
for i=2:10
        y(i)=c(i)+100 * y(i - 1)
end
y=y(:);
```

Nel calcolo del secondo sistema lineare proposto, si arriva ad avere un errore dell'ordine di 10^{17} a causa del mal condizionamento di A e dell'errore di rappresentazione in aritmetica finita di 0.1.

Soluzione 3.17

```
function A = es3_17(A)
    [m,n] = size(A);
    for i=1:n
        alpha = norm(A(i:m, i));
        if alpha==0
            error('La matrice A ha rango non massimo')
        end
        if A(i,i)>=0
            alpha = -alpha;
        end
```

```
        v = A(i,i) - alpha;
        A(i,i) = alpha;
        A(i+1:m,i) = A(i+1:m,i)/v;
        beta = -v/alpha;
        A(i:m,i+1:n) = A(i:m, i+1:n) - (beta*[1; A(i+1:m,i)])*([1 A(i+1:m,
            i)']*A(i:m,i+1:n));
    end
end
```

Soluzione 3.18

```
function [x] = es3_18( A, b )
    [m,n] = size(A);
    Qt = eye(m);
    for i=1:n
        Qt= [eye(i-1) zeros(i-1,m-i+1); zeros(i-1, m-i+1)' (eye(m-i+1)-(2/
            norm([1; A(i+1:m, i)], 2)^2)*([1; A(i+1:m, i)]*[1 A(i+1:m, i)'
            ]))]*Qt;
    end
    x = TriangolareSup(triu(A(1:n,:)), Qt(1:n,:)*b);
end
```

Soluzione 3.19 Applicando il codice degli esercizi 3.17 e 3.18 otteniamo i seguenti risultati:

$$\boldsymbol{x} = (0.72, 1.44, 0.72)^\top$$

$$\boldsymbol{r} = A\boldsymbol{x} - \boldsymbol{b} = (-0.24, -0.24, 0.32, 0.32)^\top$$

$$||r||_2^2 = 0.32$$

```
A = [3 2 1; 1 2 3; 1 2 1; 2 1 2]
b = [6 6 4 4 ]'
QR = es3_17(A)
x = es3_18(QR, b)
r = A*x-b
norm(r, 2)^2
```

Soluzione 3.20 Consideriamo la funzione

$$F(x_1, x_2) = \begin{cases} x_2 - \cos(x_1) \\ x_1 x_2 - \frac{1}{2} \end{cases}$$

Il Jacobiano della funzione F è:

$$J = \begin{pmatrix} \sin(x_1) & 1 \\ x_1 & x_2 \end{pmatrix}$$

Applicando il metodo di Newton risolviamo:

$$\begin{cases} J_F(\boldsymbol{x}^k)\boldsymbol{d}^k = -F(\boldsymbol{x}^k) \\ \boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \boldsymbol{d}^k \end{cases}$$

```
function [x] = NonLinearNewton(F,J, x, imax, tolx)
    i=0;
    xold = x+1;
    while (i< imax )&&( norm (x-xold )> tolx )
        i=i+1;
        xold =x;
        [L,U,P] = es3_5(feval(J,x));
        x=x+es3_6(L,U, P, -feval(F,x));
        disp(norm(x - xold));
        disp(x);
    end
end
```

Troviamo così la soluzione:

$$x_1 = 0.610031273504773$$

$$x_2 = 0.819630103953308$$

iterazione	x_1, x_2	Norma dell'incremento
1	$0.7458, 0.7542$	0.5001
2	$0.5531, 0.8653$	0.3145
3	$0.6042, 0.8241$	0.0929
4	$0.6100, 0.8197$	0.0102
5	$0.6100, 0.8196$	0.00013

Soluzione 3.21 Determinare il minimo della funziona significa risolvere il sistema $F(\boldsymbol{x}) = \mathbf{0}$ con

$$F = \begin{pmatrix} 4x_1^3 + 2x_1 + x_2 \\ x_1 + 2x_2 - 2 \end{pmatrix}$$

Lo Jacobiano di F è:

$$J = \begin{pmatrix} 12x_1^2 + 2 & 1 \\ 1 & 2 \end{pmatrix}$$

Una volta che abbiamo risolto il sistema lineare si valuta $f(x_1, x_2)$ nei punti calcolati e otteniamo il minimo.

```
imax = 1000;
tolx = 10^-4;
F= @(x) [4 * x(1)^3+2 * x(1)+ x(2); x(1)+2 * x(2) - 2];
J = @(x) [12 * x(1)^2+2, 1; 1, 2];
[x] = NonLinearNewton(F, J, x, imax, tolx);

f = @(x1, x2) (x1^4+ x1 * ( x1+ x2)+(1 - x2)^2)
```

Eseguendo il codice sopra riportato otteniamo il punto di minimo $f(x_1, x_2) = -0.257319110855833$ in $(-0.439807440386867, 1.219903720193434)$

4 Approssimazione di funzioni

Soluzione 4.1

```
function [pval] = es4_1(x, f, xval)
  dd = diffdivise(x, f);
  pval = horner(x,dd,xval);
end
```

```
function divDiff = diffdivise(f, x)
        divDiff = f ;
        n = length (x) -1;
    for j = 1:n
        for i = n+1:-1: j+1
        divDiff ( i ) = ( divDiff ( i )-divDiff ( i -1)) /(x( i )-x( i-j )
            ) ;
        end
        end
end
```

```
function pval = horner(x, f, xval)
    n = length (x) -1;
        ntab = length ( xval ) ;
        pval = f (n+1)*ones (1 , ntab) ;
        for j = n:-1:1
        pval = pval .*( xval-x( j ) )+f ( j ) ;
        end
end
```

Soluzione 4.2

```
function [ pval , err ] = es4_2 (a ,b, chebyshev , f ,n )
    if chebyshev
        x = zeros (1 ,n+1);
        for i = n+1:-1:1
            x(1 , i ) = (a+b)/2+(b-a)/2*cos (((2*(n-i+1)+1)*pi ) /(2*(n+1)
                ) ) ;
        end
        y = f(x);
    else
        x = linspace (a ,b,n+1);
        y = f(x);
    end
    divDiff = diffdiv(y,x);
    xx = ( linspace (a ,b,30))';
    pval = hornerGeneralizzato(x , divDiff , xx) ;
    err = norm(pval-f(xx) , inf ) ;

    hold on ;
    fplot ( f , [ a ,b] , 'g' ) ;
    plot (xx , pval , 'Color' , 'r' ) ;
end
```

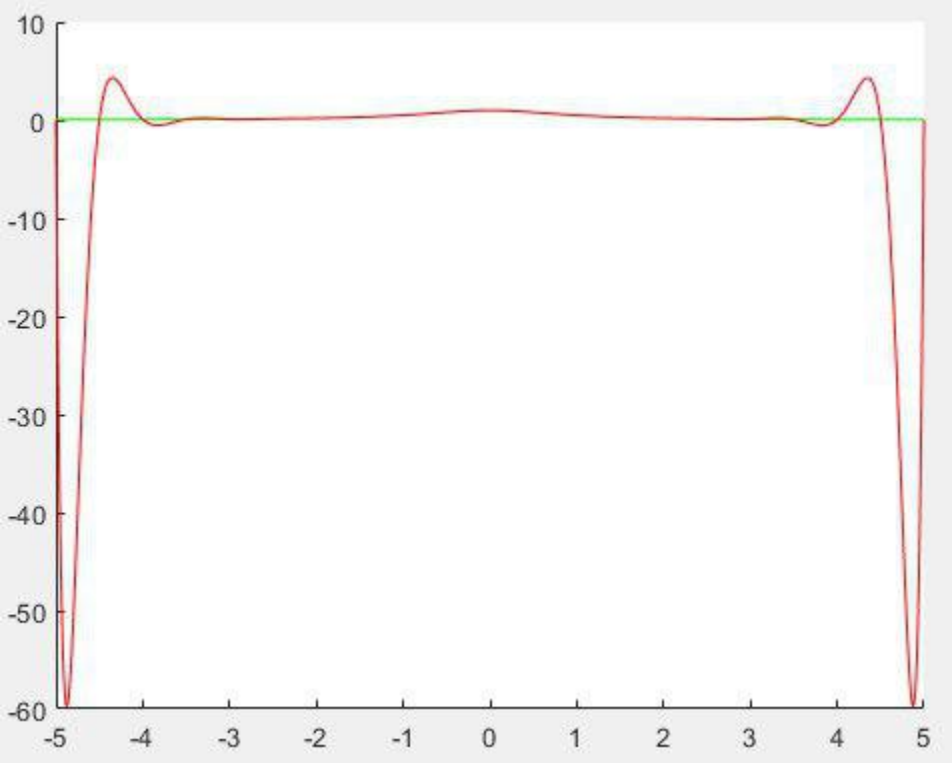
Per la funzione f_1 otteniamo i seguenti risultati:

n	f_1 linespace	f_1 Chebyshev
2	19.3236	18.0102
4	13.1501	11.8749
6	18.1323	7.7484
8	30.9578	5.1055
10	56.6897	3.1901
12	98.9538	1.8141
14	165.3159	1.3905
16	264.5756	0.8489
18	404.5909	0.6742
20	586.8901	0.4173

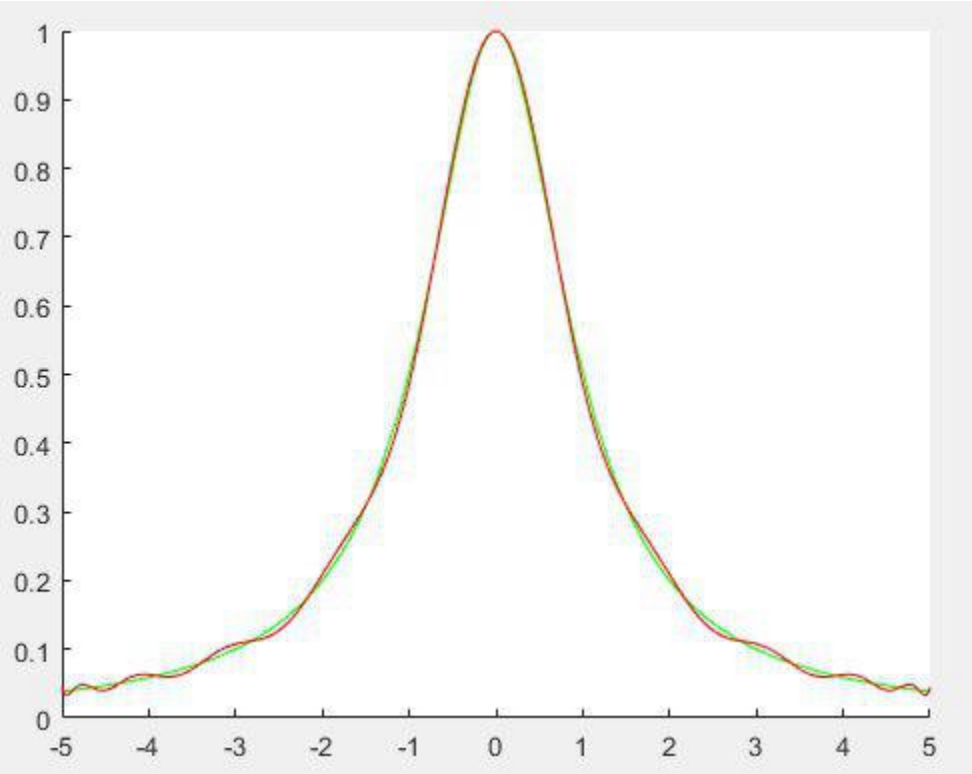
Per la funzione f_2 otteniamo i seguenti risultati:

n	f_2 linespace	f_2 Chebyshev
2	19.1297	13.0890
4	1.2187	0.6860
6	0.0386	0.0142
8	7.7163e-04	1.5969e-04
10	9.5057e-06	1.1062e-06
12	7.5808e-08	5.1858e-09
14	4.1907e-10	1.7709e-11
16	2.3664e-12	1.0251e-13
18	3.1308e-13	2.6645e-14
20	2.9193e-12	7.3275e-14

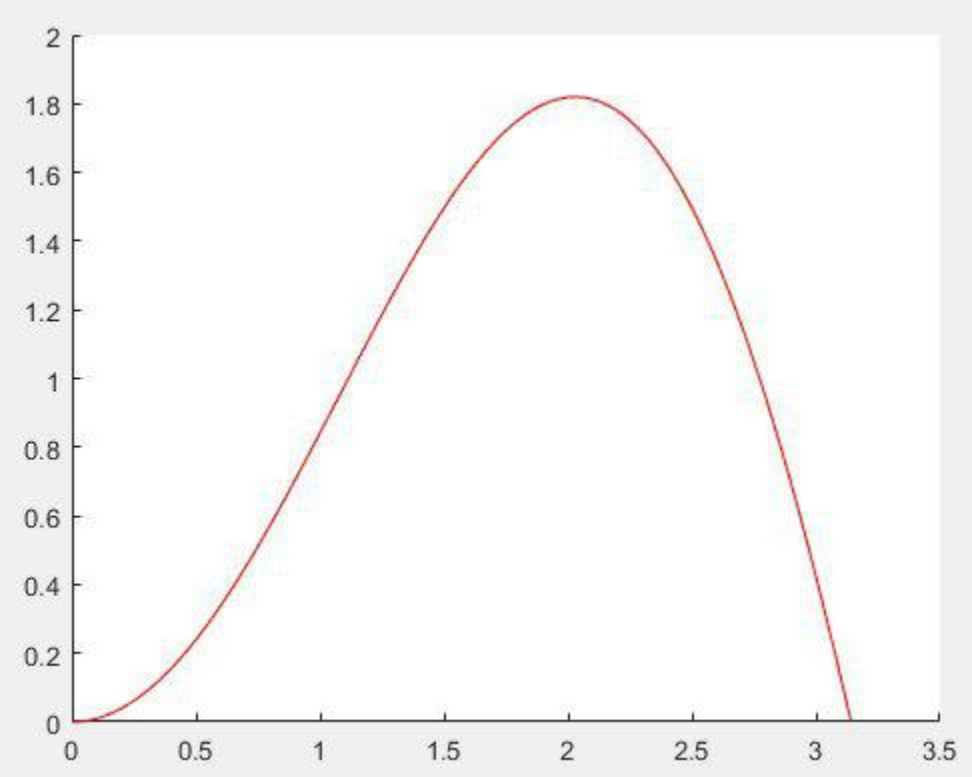
Osserviamo che utilizzando ascisse equidistanti per approssimare f_1, al crescere del numero degli intervalli, l'errore aumenta. Negli altri casi l'errore diminuisce.
Prima funzione calcolata con $n = 20$ con ascisse equidistanti:



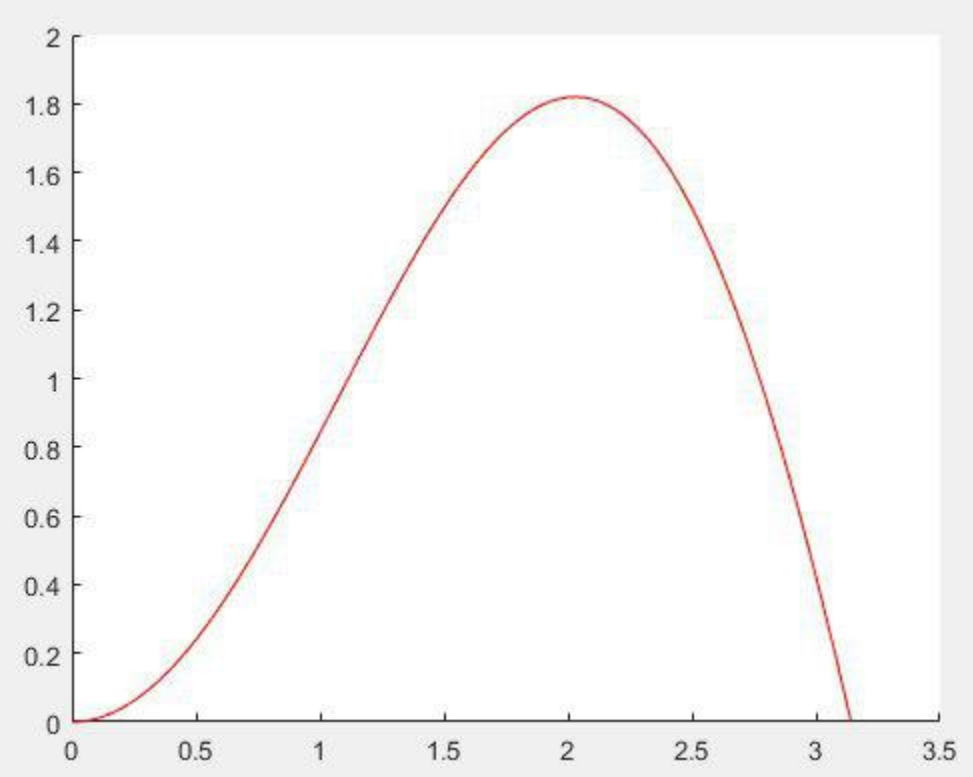
Prima funzione calcolata con $n = 20$ con ascisse di Chebyshev:



Seconda funzione calcolata con $n = 20$ con ascisse equidistanti:



Seconda funzione calcolata con $n = 20$ con ascisse di Chebyshev:



Soluzione 4.3 Questa funzione ritorna il vettore dei momenti $\boldsymbol{M}, \boldsymbol{r}, \boldsymbol{q}$.

```
function [M, r ,q] = es4_3 (x , f )

    if length ( f ) == length (x)
        n = length (x) ;
        q = zeros (1 ,n-1);
        r = zeros (1 ,n-1);
        phi = zeros (1 ,n-2);
        xi = zeros (1 ,n-2);
        for i = 1:n-2
            phi ( i ) = (x( i+1)-x( i ) ) /(x( i+2)-x( i ) ) ;
            xi ( i ) = (x( i+2)-x( i+1)) /(x( i+2)-x( i ) ) ;
        end
        divDiff = es4_3DD( f ,x) ;
        M = es4_3M( divDiff , phi , xi ) ;
        for i = 2:n
            h = x( i )-x( i -1);
            q( i ) = ( f ( i )-f ( i -1))/h-((h/6) *(M( i )-M( i -1)) ) ;
            r ( i ) = f ( i -1)-(h.^2/6)*M( i -1);
        end
    else
        error ( 'x e f devono avere la stessa lunghezza ') ;
    end
end
```

Funzione per il calcolo delle differenze divise.

```
function divDiff = es4_3DD( f ,x)
    divDiff = f ;
        n = length (x) -1;
    for j = 1:n
        for i = n+1:-1: j+1
        divDiff ( i ) = ( divDiff ( i )-divDiff ( i -1)) /(x( i )-x( i-j )
            ) ;
        end
    end
        divDiff = divDiff (3: length ( divDiff ) )';
end
```

Funzione per il calcolo dei momenti relativi alla spline cubica naturale che interpola nei nodi.

```
function [M] = es4_3M( divDiff, phi , xi )

n = length(xi);

for i = 1 : n
    diagonal(i) = 2 ;
end

diagonal = diagonal(1 : n)';
divDiff = 6*divDiff;

M = ex4_3ST (diagonal,divDiff,phi,xi);
M = [ 0 ;M; 0 ] ;
end
```

Funzione per la risoluzione di un sistema tridiagonale a diagonale dominante.

```
function M = es4_3ST( diagonal , divDiff , phi , xi )
        n = length ( divDiff ) ;
        v = zeros (n,1) ;
        w = diagonal (1) ;
    M = v;
        M(1) = divDiff (1)/w;
    for i = 2:n
        v( i -1) = xi ( i -1)/w;
        w = diagonal ( i )-phi ( i )*v( i -1);
        M( i ) = ( divDiff ( i )-phi ( i )*M( i -1))/w;
    end
    for j = n-1:-1:1
        M( j ) = M( j )-v( j )*M( j+1);
        end
end
```

Soluzione 4.4

```
function [ sval ] = Es4_4( x,f,M,xval )
    n = length (x) -1;
    if n == length ( xval )
        for j = 1:n
            for i = 2:n+1
                if xval ( j ) >= x( i -1) && xval ( j ) <= x( i )
```

```
                    h = x( i )-x( i -1);
                    q = ( f ( i )-f ( i -1))/h-((h/6) *(M( i )-M( i -1)) )
                        ;
                    r = f ( i -1)-(h.^2/6)*M( i -1);
                    sval ( j ) = (( xval ( j )-x( i -1)) .^3*M( i )+(x( i
                       )-xval ( j ) ) .^3*M( i -1))/6*h;
                    sval ( j ) = sval ( j )+q*( xval ( j )-x( i -1))+r ;
                end
            end
        end
    else
        error ( 'lunghezza sbagliata ' ) ;
    end
end
```

Soluzione 4.5

```
function [naturalError,notAKnotError] = es4_5(f,a,b,n, flag)
xval = linspace(a,b,n);
x = linspace( a,b,n+1);
f1 = f(x);
fvalx=f(xval);
[M,r,q]= es4_3(x,f1)
naturalSpline=Es4_4(x,f1,M,xval)
naturalError=norm((fvalx- naturalSpline),inf);
notAKnotSpline=spline(x,f1,xval);
notAKnotError=norm((fvalx-notAKnotSpline),inf);
points=linspace(a,b,501);
fval=f(points);
if flag
    plot(points,fval,'b',xval,naturalSpline,'r')
else
    plot(points,fval,'b',xval,notAKnotSpline,'r')
end
end
```

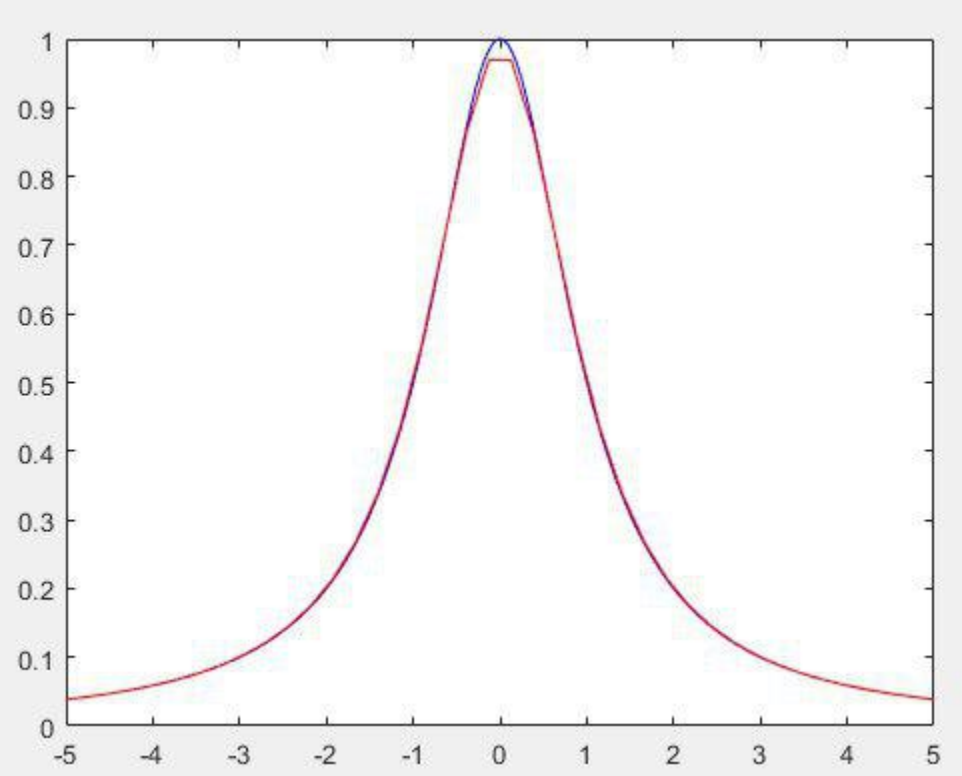
Per la funzione f_1 otteniamo i seguenti risultati:

n	f_1 naturale	f_1 not-a-knot
2	1.3878e-16	0
4	0.4478	0.2601
6	0.0648	0.1269
8	0.0305	0.0561
10	0.0421	0.0218
12	0.0524	0.0065
14	0.0531	0.0023
16	0.0498	0.0034
18	0.0452	0.0035
20	0.0405	0.0030

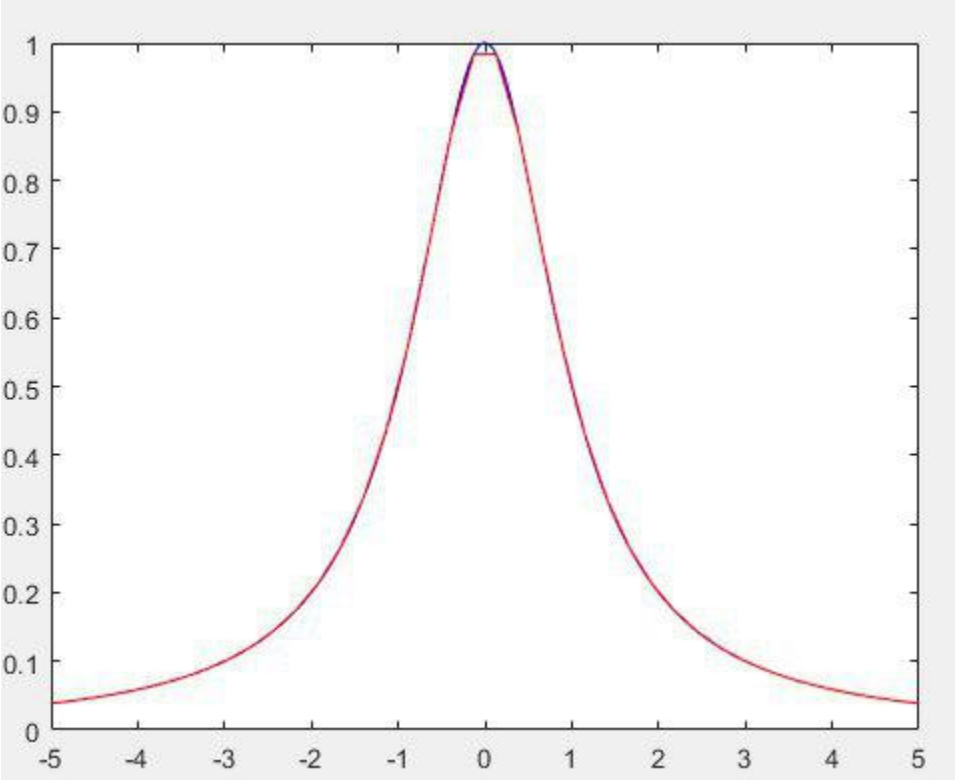
Per la funzione f_2 otteniamo i seguenti risultati:

n	f_2 naturale	f_2 not-a-knot
2	5.0344e-16	3.1284e-16
4	0.1494	0.0187
6	0.0833	0.0028
8	0.0460	6.7156e-04
10	0.0305	2.1966e-04
12	0.0243	8.7457e-05
14	0.0194	4.0031e-05
16	0.0158	2.0322e-05
18	0.0130	1.1173e-05
20	0.0108	6.5426e-06

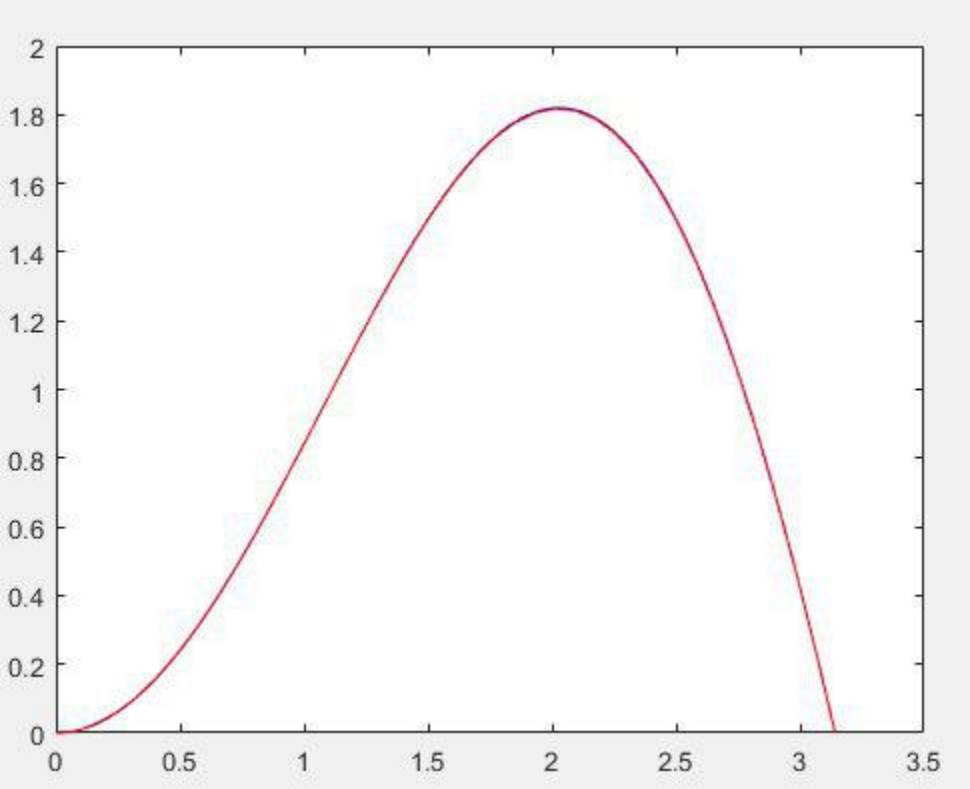
Per i seguenti grafici abbiamo utilizzato 40 punti di disegno.
Funzione f_1 calcolata con spline naturale:



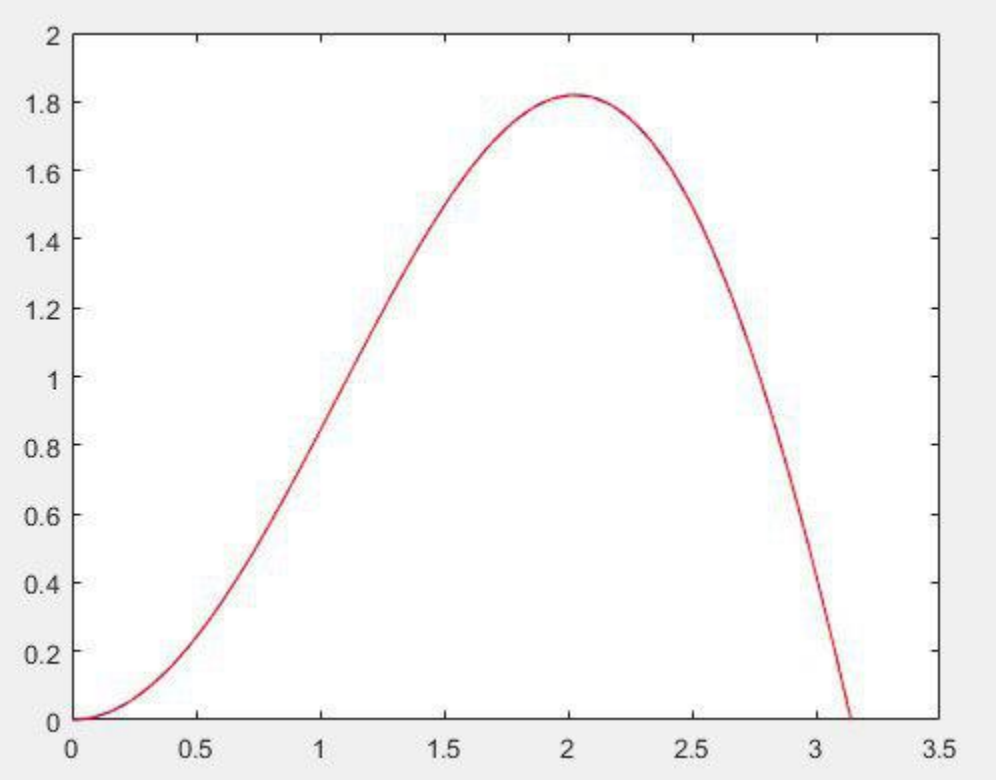
Funzione f_1 calcolata con spline not-a-knot:



Funzione f_2 calcolata con spline naturale:



Funzione f_2 calcolata con spline not-a-knot:



Soluzione 4.6 Questa funzione ritorna il vettore dei momenti $\boldsymbol{M}, \boldsymbol{r}, \boldsymbol{q}$.

```
function [M, r ,q] = es4_6 (x , f )

    if length ( f ) == length (x)
        n = length (x) ;
        q = zeros (1 ,n-1);
        r = zeros (1 ,n-1);
        phi = zeros (1 ,n-1);
        xi = zeros (1 ,n-2);
        for i = 1:n-2
            phi ( i ) = (x( i+1)-x( i ) ) /(x( i+2)-x( i ) ) ;
            xi ( i ) = (x( i+2)-x( i+1)) /(x( i+2)-x( i ) ) ;
        end
        divDiff = es4_6DD( f ,x) ;
        M = es4_6M( divDiff , phi , xi ) ;
        for i = 2:n
            h = x( i )-x( i -1);
            q( i ) = ( f ( i )-f ( i -1))/h-((h/6) *(M( i )-M( i -1)) ) ;
            r ( i ) = f ( i -1)-(h.^2/6)*M( i -1);
        end
    else
        error ( 'x e f devono avere stessa lunghezza ') ;
    end
end
```

Funzione per il calcolo delle differenze divise.

```
function divDiff = es4_6DD( f ,x)
    divDiff = f ;
        n = length (x) -1;
    for j = 1:n
        for i = n+1:-1: j+1
        divDiff ( i ) = ( divDiff ( i )-divDiff ( i -1)) /(x( i )-x( i-j )
            ) ;
        end
    end
        divDiff = divDiff (2: length ( divDiff ) )';
end
```

Funzione per il calcolo dei momenti relativi alla spline cubica naturale che interpola nei nodi.

```
function [M] = es4_6M( divDiff, phi , xi )

n = length(xi);

for i = 1 : n
    diagonal(i) = 2 ;
end

diagonal = diagonal(1 : n)';
divDiff = 6*divDiff;

M = ex4_3ST (diagonal,divDiff,phi,xi);
M = [ M; M(1) ] ;
end
```

Funzione per la risoluzione di un sistema tridiagonale a diagonale dominante.

```
function M = es4_3ST( diagonal , divDiff , phi , xi )
        n = length ( divDiff ) ;
        v = zeros (n,1) ;
        w = diagonal (1) ;
    M = v;
        M(1) = divDiff (1)/w;
    for i = 2:n
        v( i -1) = xi ( i -1)/w;
        w = diagonal ( i )-phi ( i )*v( i -1);
        M( i ) = ( divDiff ( i )-phi ( i )*M( i -1))/w;
    end
    for j = n-1:-1:1
        M( j ) = M( j )-v( j )*M( j+1);
        end
end
```

Soluzione 4.7

Soluzione 4.8

```
function [a] = es4_8 (m,x ,y)
        n = length (x) ;
        V = zeros (n,m+1);
        for j = 1:m+1
```

```
                V(: , j ) = x.^(m+1-j ) ;
        end
        a = V\y;
        z = V*a ;
end
```

Soluzione 4.9

- Esempio 1: Funzione $5x_i + 2 + \epsilon\gamma_i$, m=1, c=0.1. Valori random:

$$(0.0540, 0.5308, 0.7792, 0.9340, 0.1299, 0.5688, 0.4694, 0.0119, 0.3371, 0.1622)^\top$$

  Ordinate y:

$$(-2.9946, -1.8358, -0.6999, 0.4267, 1.4574, 2.6124, 3.7136, 4.7790, 5.9226, 7.0162)^\top$$

  Coefficienti del polinomio:

$$(4.9859, 2.0398)^\top$$

- Esempio 2: Funzione $3x_i^2 + 2x_i + 1 + \epsilon\gamma_i$, m = 2, c = 0.2. Valori random:

$$(0.7943, 0.3112, 0.5285, 0.1656, 0.6020, 0.2630, 0.6541, 0.6892, 0.7482, 0.4505)^\top$$

  Ordinate y:

$$(2.1589, 1.3215, 0.9205, 0.6998, 0.9352, 1.3119, 2.1308, 3.1749, 4.5200, 6.0901)^\top$$

  Coefficienti del polinomio:

$$(3.0376, 2.0103, 1.0888)^\top$$

```
f1 = @(x,e,l) 5 * x+ 2 +e * l;
e= 0.1;
s= linspace( - 1,1,10);
y1 = zeros(10,1);
l=rand(10,1);
for i=1:10
        y1(i) = f1(s(i),e,l(i));
end
res1 = es4_8(1,s, y1);
disp(res1`)
disp(y1`)
disp(l`)


f2 = @(x,e,l) 3 * x^2 + 2 * x +1 + e * l;
e= 0.2;
s= linspace( - 1,1,10);
y2 = zeros(10,1);
l2=rand(10,1);
for i=1:10
        y2(i) = f2(s(i),e,l2(i));
end
res2 = es4_8(2,s,y2);
disp(res2')
disp(y2')
disp(l2')
```

Soluzione 4.10

Funzione	Coefficienti polinomiali
$y_i = 5x_i + 2 + \epsilon\gamma_i$	$(4.9859, 2.0398)^\top$
$x_i = \frac{y_i - (2+\epsilon\gamma_i)}{5}$	$(5.0586, 2.0682)^\top$

```
m = 1;
n = 10;
c = 0.1

randomValues = rand(n,1) ;
x = linspace (-1,1,n)`;
y = x;

funct01 = @(x , a) (5*x)+2+(c*a) ;
funct02 = @(y , a) (y-2-(c*a) ) /5;

y01 = funct01 (x , randomValues) ;
x01 = funct02 (y , randomValues) ;

[ a01] = es4_8(m,x, y01);
disp(a01)
[ a02] = es4_8(m, x01 ,y) ;
disp(a02)
```

5 Formule di quadratura - Google Pagerank

Soluzione 5.1

```

function [integral] = formulaTrapeziComposita(f, a, b, n)
h = (b-a)/n;
integral = 0;
for i=1:n-1
    integral = integral+f(a+i*h);
end
integral = (h/2)*(2*integral + f(a) + f(b));
end
```

```

function [integral] = formulaSimpsonComposita(f, a, b, n)
h = (b-a)/n;
integral = f(a);
for i=1:n/2
    integral = integral + 4*f(a+(2*i-1)*h);
end
for i=1:n/2-1
    integral = integral +2*f((a+2*i*h));
end
integral = (f(b)+integral)*(h/3);
end
```

Soluzione 5.2

n	$\lvert I[f] - I_1^{(n)}[f]\rvert$	TR_n	$\lvert I[f] - I_2^{(n)}[f]\rvert$	SR_n
2^1	5.6706e-01	na	7.0308e-01	na
2^2	2.3483e-01	-3.3223e-01	5.0212e-01	-2.0096e-01
2^3	5.6353e-02	-1.7848e-01	3.1390e-03	-4.9899e-01
2^4	1.3274e-02	-4.3079e-02	1.0853e-03	-2.0537e-03
2^5	3.2632e-03	-1.0011e-02	7.3810e-05	-1.0115e-03
2^6	8.1229e-04	-2.4509e-03	4.6819e-06	-6.9128e-05
2^7	2.0285e-04	-6.0944e-04	2.9360e-07	-4.3883e-06
2^8	5.0699e-05	-1.5215e-04	1.8365e-081	-2.7524e-07

```
format long
F = @(x) x * exp(1)^ - x * cos(2 * x);
y = (3 * (exp(1)^( - 2 * pi) - 1) - 10 * pi * exp(1)^( - 2 * pi))/25;
nmax = 8;
err = zeros(nmax,2);
rap = zeros(nmax - 1,2);
for i=1:8
        err(i,1) = abs(y - formulaTrapeziComposita(F,0,2 * pi,2^i));
        err(i,2) = abs(y - formulaSimpsonComposita(F,0,2 * pi,2^i));
        if i>1
        rap(i - 1,:) = err(i,:)./err(i - 1,:);
        end
end
```

Soluzione 5.3

```
function [integral] = SimpsonAdattativa(f, a, b, tol)
        [integral] = recursiveSimpsonSimpson(f, a, b, tol);
end
function [integral, p] = recursiveSimpsonSimpson(f, a, b, tol)
        h = (b-a)/6;
        m = (a+b)/2;
        m1 = (a+m)/2;
        m2 = (m+b)/2;
        int1 = h*(feval(f, a) +4*feval(f, m) + feval(f, b));
        integral = int1/2 + h*(2*feval(f, m1) + 2*feval(f, m2)-feval(f, m)
           );
        err = abs(integral-int1)/15;
        if err>tol
                [intSx] = recursiveSimpsonSimpson(f, a, m, tol/2);
                [intDx] = recursiveSimpsonSimpson(f, m, b, tol/2);
                integral = intSx+intDx;
        end
end
```

```
function [integral] = TrapeziAdattativa(f, a, b, tol)
        [integral] = recursiveTrapezi(f, a, b, tol);
end

function [integral] = recursiveTrapezi(f, a, b, tol)
        h = (b-a)/2;
        m = (a+b)/2;
```

```
        int1 = h*(feval(f, a) + feval(f, b));
        integral = int1/2 + h*feval(f, m);
        err = abs(integral-int1)/3;
        if err>tol
                [intSx] = recursiveTrapezi(f, a, m, tol/2);
                [intDx] = recursiveTrapezi(f, m, b, tol/2);
                integral = intSx+intDx;
        end
end
```

Con la function di Simpson adattativa otteniamo: $I[f] = -0.122122876187005$.
Con la function di Trapezi adattativa otteniamo: $I[f] = -0.122122922578069$.

Soluzione 5.4

```
function [xn, i, err] = jacobi(A, b, x0, tol, nmax)
        D = diag(diag(A));
        J = - inv(D) * (A - D);
        q = D\b;
        xn = J * x0 + q;
        i = 1;
        err(i) = norm(xn - x0)/norm(xn);
        while (i<=nmax && err(i)>tol)
                x0 = xn;
                xn = J * x0+q;
                i = i+1;
                err(i) = norm(xn - x0)/norm(xn);
        end
        if i>nmax
                disp('Jacobi non converge ');
        end
end
```

```
function [xn, i, err] = gaussSeidel(A, b, x0, tol, nmax)
        D=diag(diag(A));
        L=tril(A) - D;
        U=triu(A) - D;
        DI=inv(D+L);
        GS= - DI * U;
        b1=(D+L)\b;
        xn=GS * x0+b1;
        i=1;
        err(i)=norm(xn - x0,inf)/norm(xn);
        while(err(i)>tol && i<=nmax)
                x0=xn;
                xn=GS * x0+b1;
                i=i+1;
                err(i)=norm(xn - x0,inf)/norm(xn);
        end
        if i>nmax
                error('GaussSeidel non converge');
        end
        i=i - 1;
```

```
end
```

Soluzione 5.5

```
A = [ - 5,2,1;1,7,2;2, - 3,7];
b = [2,1,3]';
x0 = [0,0,0]';
[z,j,jerr] = es5_4J(A,b,x0,1.e-3,25)
[y,i,gerr] = es5_4GS(A,b,x0,1.e-3,25)
```

tol	iterazioni Jacobi	iterazioni GaussSeidel
10^{-1}	4	3
10^{-2}	8	5
10^{-3}	11	7
10^{-4}	14	9
10^{-5}	18	12
10^{-6}	21	14

Si evince dalla tabella che il metodo di GaussSeidel converge alla soluzione con un numero di iterazioni minore rispetto al metodo di Jacobi.

Soluzione 5.6

```
H = [1 0 1 0 0; 1 0 0 0 1; 1 1 0 0 0; 0 1 1 0 0; 0 1 1 0 1]
p = 0.85
[m,n] = size (H)
S = makeS(H)
A = eye (m,n)-(p*S)
b = ones (m,1) *((1-p)/n)
x0 = ones(n,1)

[l, ip] = Potenze(H, 0.85, 10^-15)
[z,ij,jerr] = es5_4J(A,b,x0,10^-15,500)
[y,igs,gerr] = es5_4GS(A,b,x0,10^-15,500)
```

tol	iterazioni Jacobi	iterazioni GaussSeidel	iterazioni Potenze
10^{-1}	9	4	18
10^{-2}	22	10	32
10^{-3}	34	14	46
10^{-4}	45	19	60
10^{-5}	56	23	75
10^{-6}	67	28	89
10^{-7}	79	33	103
10^{-8}	90	37	117
10^{-9}	101	42	131
10^{-10}	112	46	145

Si osserva che il metodo di GaussSeidel converge in un numero inferiore di iterazioni rispetto ai metodi di Jacobi e delle potenze.

```
function [x,i] = Potenze(H, p, tol)
  [m,n] = size(H);
  if m~=n
```

```
    error('H non quadrata');
  end
  delta = zeros(n, 1);
  for j=1:n
    for i=1:n
      delta(j) = delta(j) + H(i, j);
    end
  end
  for j=1:n
    for i=1:n
      if delta(j)==0
        H(i, j) = 0;
      else
        H(i, j) = H(i, j)/delta(j);
      end
    end
    if delta(j)==0
      delta(j)=1;
    else
      delta(j)=0;
    end
  end
  x = rand(n,1);
  x = x/norm(x);
  imax = (log(tol)-log(2))/(log(p));
  for i=1:imax
    x = p*(H*x)+((1+p*(delta'*x -1))/n)*ones(n,1);
  end
end
```