Esercizio 4.1 *Scrivere una function Matlab che implementi il calcolo del polinomio interpolante di grado n in forma di Lagrange.*
La forma della function deve essere del tipo: `y = lagrange( xi, fi, x )`
Soluzione:

```
function y = lagrange(xi, fi, x)
% function y = lagrange( xi, fi, x )
% xi vettore dei punti di ascissa
% fi vettore dei valori di f(x)
% x vettore di punti in cui calcolare f(x)
% y valore di f(x)
if length(xi) ~= length(fi)
    error('xi e fi hanno lunghezza diversa, deve essere uguale'
       )
end
n = length(xi)-1;
m = length(x);
y = zeros(size(x));
for i=1:m
    for j=1:n+1
        p = 1;
        for k=1:n+1
            if j~=k
                p = p*(x(i)-xi(k))/(xi(j)-xi(k));
            end
        end
        y(i) = y(i)+fi(j)*p;
    end
end
return
end
```

Esercizio 4.2 *Scrivere una function Matlab che implementi il calcolo del polinomio interpolante di grado n in forma di Newton.*
La forma della function deve essere del tipo: `y = newton( xi, fi, x )`
Soluzione:

```
function y = newton(xi,fi,x)
% function y = newton(xi,fi,x)
% Implementa il calcolo del polinomio interpolante di grado n
    in forma di Newton
%  xi vettore delle ascisse di interpolazione
```

```
%  fi vettore dei valori della funzione in x
%  x vettore dei punti in cui valutare il polinomio
%  y vettore dei valori del polinomio valutato sui punti x.
if length(xi) ~= length(fi)
    error('xi e fi hanno lunghezza diversa!')
end
dd = diff_div(xi, fi);
y = dd(length(dd));
for k = length(dd)-1:-1:1
    y = y*(x-xi(k))+dd(k);
end
end

function [fi] = diff_div(xi, fi)
for i = 1:length(xi) - 1
    for j = length(xi):-1:i+1
        fi(j) = (fi(j) - fi(j-1))/(xi(j)-xi(j-i));
    end
end
end
```

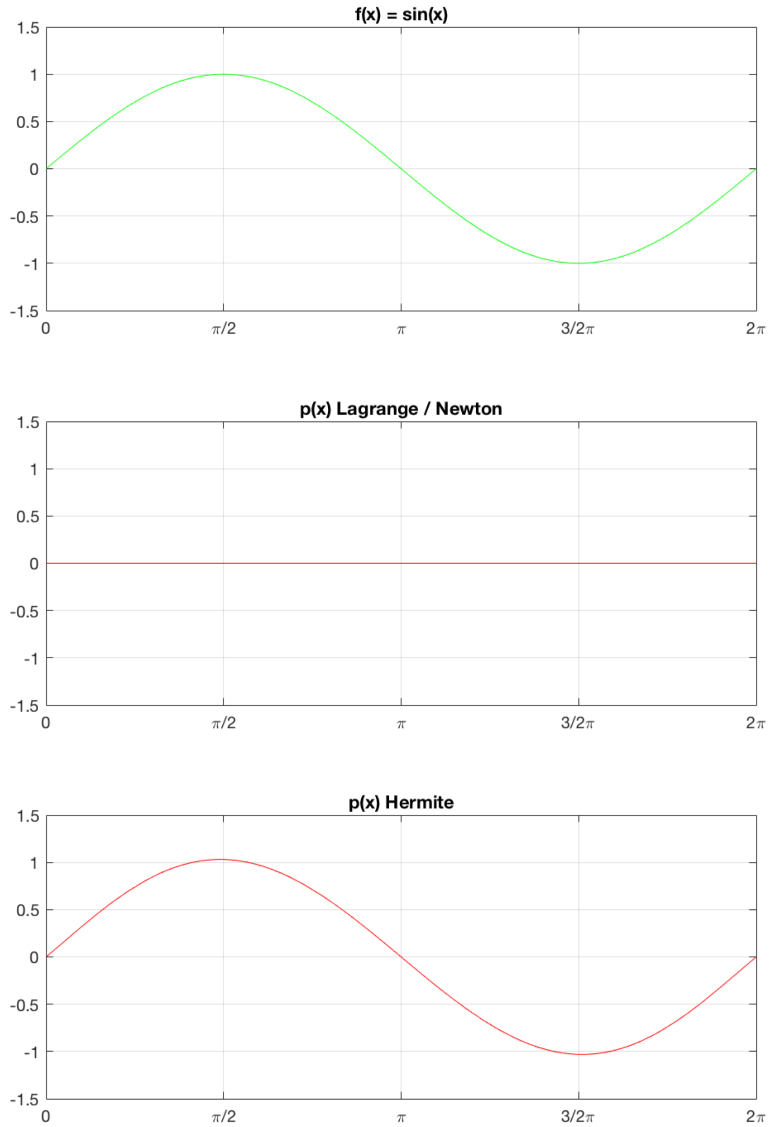
Esercizio 4.3 *Scrivere una function Matlab che implementi il calcolo del polinomio interpolante di Hermite.*
La forma della function deve essere del tipo:
`y = hermite( xi, fi, f1i, x )`
Soluzione:

```
function y = hermite(xi, fi, f1i, x)
% function y = hermite(xi, fi, f1i, x)
%   xi vettore delle ascisse
%   fi vettore delle valutazioni di f(x)
%   f1i vettore delle valutazioni di f'(x)
%   x punto da valutare
%   y vettore riscrito con le differenze divise
if length(xi) ~= length(fi) || length(f1i) ~= length(fi)
    error('xi, fi e f1x devono avere la stessa lunghezza')
end
% combino opportunamente i vettori
xih = zeros(length(xi)*2, 1);
fih = zeros(length(fi)*2, 1);
for i = 1:length(xi)
```

```
    xih(i+i-1) = xi(i);
    xih(i+i) = xi(i);
    fih(i+i-1) = fi(i);
    fih(i+i) = f1i(i);
end
ddh = diffDiviseHermite(xih, fih);
y = ddh(1);
for i = 2 : length(dd)
    prod = ddh(i);
    for j=1:i-1
        prod = prod*(x-xih(j));
    end
    y = y + prod;
end
end

function [fi] = diffDiviseHermite(xi, fi)
%  function [f] = diffDiviseHermite(x, f)
%  xi vettore delle ascisse
%  fi vettore con f(x0), f'(x0),..., f(xn), f'(xn)
%  fi vettore riscrito con le differenze divise
    n = length(xi)-1;
    for i = n:-2:3
        fi(i) = (fi(i)-fi(i-2))/(xi(i)-xi(i-1));
    end
    for j = 2:n
        for i = n+1:-1:j+1
            fi(i) = (fi(i)-fi(i-1))/(xi(i)-xi(i-j));
        end
    end
return
end
```

Esercizio 4.4 *Utilizzare le functions degli esercizi precedenti per disegnare l'approssimazione della funzione* $\sin(x)$ *nell'intervallo* $[0, 2\pi]$*, utilizzando le ascisse di interpolazione* $x_i = i\pi$*,* $i = 0, 1, 2$*.*
Soluzione:



Possiamo notare come il polinomio di Lagrange e Newton genera una retta $y = 0$ essendo $f_i = 0$ in tutte le ascisse.

```
% funzione esatta
f = @(x) sin(x);
f1 = @(x) cos(x);
interval = [0, 2*pi];

% polinomi interpolanti, function es 1, 2, 3
xi = [0, pi, 2*pi];
fi = [f(0), f(pi), f(2*pi)];
f1i = [f1(0), f1(pi), f1(2*pi)];

pn = @(x) newton(xi, fi, x); %Lagrange genererÃ  lo stesso polinomio
ph = @(x) hermite(xi, fi, f1i, x);

% plots
subplot(3,1,1)
fplot(f, interval, 'g')
grid on
title('f(x) = sin(x)')
xlim([0, 2*pi])
xticks([0 pi/2 pi pi+pi/2 2*pi])
xticklabels({'0','\pi/2','\pi','3/2\pi','2\pi'})
ylim([-1.5, 1.5])

subplot(3,1,2)
fplot(pn, interval, 'r')
grid on
title('p(x) Lagrange / Newton')
xlim([0, 2*pi])
xticks([0 pi/2 pi pi+pi/2 2*pi])
xticklabels({'0','\pi/2','\pi','3/2\pi','2\pi'})
ylim([-1.5, 1.5])

subplot(3,1,3)
fplot(ph, interval, 'r')
grid on
title('p(x) Hermite')
xlim([0, 2*pi])
xticks([0 pi/2 pi pi+pi/2 2*pi])
xticklabels({'0','\pi/2','\pi','3/2\pi','2\pi'})
```

```
ylim([−1.5, 1.5])
```

Esercizio 4.5 *Scrivere una function Matlab che implementi la spline cubica interpolante (naturale o not-a-knot, come specificato in ingresso) delle coppie di dati assegnate. La forma della funcion deve essere del tipo:*
`y = spline3(xi, fi, x, tipo)`
Soluzione: Il seguente codice Matlab implementa la function richiesta. Per rendere il codice più leggibile sono state craete varie sottofunzioni. Il codice è stato testato con un banale esempio:

```
% esempio con f(x) = (pi*x)/(x+1)
xi = [0,1,2,3];
fi = [0, 1.570796, 2.094395, 2.3561944];
x = 1.5;

% not−a−knot
y_nan = spline3(xi, fi, x, true);
% naturale
y_nat = spline3(xi, fi, x, false);
y = (pi∗x)/(x + 1);


function y = spline3(xi, fi, x, isNotAKnot)
% function y = spline3(xi, fi, x, isNotAKnot)
% xi vettore delle ascisse
% fi vettore delle valutazioni di f(x)
% punto da valutare
% isNotAKnot true se not−a−knot, false se naturale
% y risultato approssimazione di f(x)
    s = p_spline3(xi, fi, isNotAKnot);
    n = 0;
    for i = 1:xi(length(xi))
        if x > xi(i) && x <= xi(i+1)
            n = i;
            break
        end
    end

    y = double(subs(s(n), x));
end

```

```
function s = p_spline3(xi, fi, tipo)
    n = length(xi) - 1;
    xis = zeros(1, n - 1);
    phi = zeros(1, n - 1);
    for i = 1 : n - 1
        phi(i) = ( xi(i + 1) - xi(i) ) / ( xi(i + 2) - xi(i) );
        xis(i) = ( xi(i + 2) - xi(i + 1) ) / ( xi(i + 2) - xi(i ) );
    end
    dd = diff_div_spline3(xi, fi);
    if tipo
        m = vettore_sistema_spline3(phi, xis, dd);
    else
        m = sistema_spline3(phi, xis, dd);
    end

    s = espressione_spline3(xi, fi, m);
end

function fi = diff_div_spline3(xi, fi)

    n = length(xi) - 1;

    for j = 1 : 2
        for i = n + 1 : - 1 : j + 1
            fi(i) = ( fi(i) - fi(i - 1) )/(xi(i) - xi(i - j) );
        end
    end

    fi = fi(3 : length(fi))';
end

function m = sistema_spline3(phi, xi, dd)
    n = length(xi) + 1;
    u = zeros(1, n - 1);
    l = zeros(1, n - 2);
    u(1) = 2;
    for i = 2 : n - 1
        l(i) = phi(i) / u(i - 1);
        u(i) = 2 - l(i) * xi(i - 1);
    end
```

```
    dd = 6 * dd;
    y = zeros(1, n - 1);
    y(1) = dd(1);
    for i = 2 : n - 1
        y(i) = dd(i) - l(i) * y(i - 1);
    end
    m = zeros(1, n - 1);
    m(n - 1) = y(n - 1) / u(n - 1);
    for i = n - 2 : - 1 : 1
        m(i) = (y(i) - xi(i) * m(i + 1)) / u(i);
    end
    m = [0 m 0];
end

function m = vettore_sistema_spline3(phi, xi, dd)
    n = length(xi) + 1;
    if n + 1 < 4
        error('Not-A-Knot con meno di 4 ascisse!');
    end
    dd = [6 * dd(1); 6 * dd; 6 * dd(length(dd))];
    w = zeros(n, 1);
    u = zeros(n + 1, 1);
    l = zeros(n, 1);
    y = zeros(n + 1, 1);
    m = zeros(n + 1, 1);
    u(1) = 1;
    w(1) = 0;
    l(1) = phi(1);
    u(2) = 2 - phi(1);
    w(2) = xi(1) - phi(1);
    l(2) = phi(2) / u(2);
    u(3) = 2 - ( l(2) * w(2) );
    w(3) = xi(2);
    for i = 4 : n - 1
        l(i - 1) = phi(i - 1) / u(i - 1);
        u(i) = 2 - l(i - 1) * w(i - 1);
        w(i) = xi(i - 1);
    end
    l(n - 1) = ( phi(n - 1) - xi(n - 1) ) / u(n - 1);
    u(n) = 2 - xi(n - 1) - l(n - 1) * w(n - 1);
    w(n) = xi(n - 1);
```

```
    l(n) = 0;
    u(n + 1) = 1;
    y(1) = dd(1);
    for i = 2 : n + 1
        y(i) = dd(i) - l(i - 1) * y(i - 1);
    end
    m(n + 1) = y(n + 1) / u(n + 1);
    for i = n : -1 : 1
        m(i) = (y(i) - w(i) * m(i + 1))/u(i);
    end
    m(1) = m(1) - m(2) - m(3);
    m(n + 1) = m(n + 1) - m(n) - m(n - 1);
end


function s = espressione_spline3(xi, fi, m)
    n = length(xi) - 1;
    s = sym('x' , [n 1]);
    syms x;
    for i = 2 : n + 1
        hi = xi(i) - xi(i - 1);
        ri = fi(i - 1) - hi^2/6 * m(i - 1);
        qi = (fi(i) - fi(i - 1))/hi - hi/6 * (m(i) - m(i - 1));
        s(i - 1) = ( (x - xi(i - 1))^3 * m(i) + (xi(i) - x)^3 *
            m(i - 1) ) / (6 * hi) + qi * (x - xi(i - 1)) + ri;
    end
end
```

Esercizio 4.6 *Scrivere una function Matlab che implementi il calcolo delle ascisse di Chebyshev per il polinomio interpolante di grado n, su un generico intervallo $[a, b]$.*
La function deve essere del tipo: `xi = ceby( n, a, b )`
Soluzione:

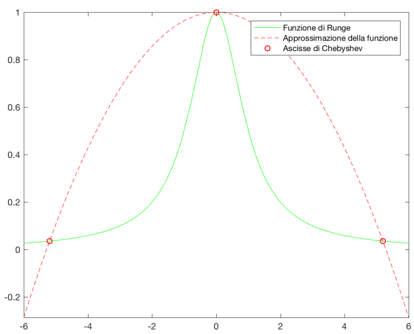
```
function xi = ceby(n, a, b)
% function xi = ceby(n, a, b)
% n numbero di ascisse da cercare
% intervallo da a a b
% xi vettore con le ascisse cercate di Chebyshev
  xi = zeros(n+1, 1);
    for i = 0:n
```

```
        xi(n+1-i) = (a+b)/2 + cos(pi*(2*i+1)/(2*(n+1)))*(b-a)/2;
    end
end
```

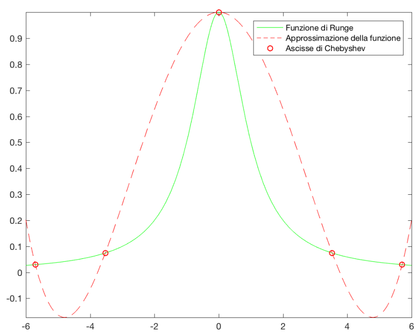
Esercizio 4.7 *Utilizzare le function degli Esercizi 4.1 e 4.6 per graficare l'approssimazione della funzione di Runge sull'intervallo* $[-6, 6]$, *per* $n = 2, 4, 6, \ldots, 40$. *Stimare numericamente l'errore commesso in funzione del grado* n *del polinomio interpolante.*

Soluzione: Di seguito i grafici che mostrano i polinomi interpolanti di grado n calcolati utilizzando come punti di interpolazione quelli corrispondenti alle n ascisse di Chebyshev, sovrapposti al grafico della funzione di Runge: $f(x) = \frac{1}{1+x^2}$.

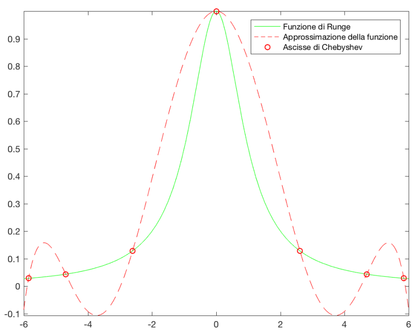
$n = 2$



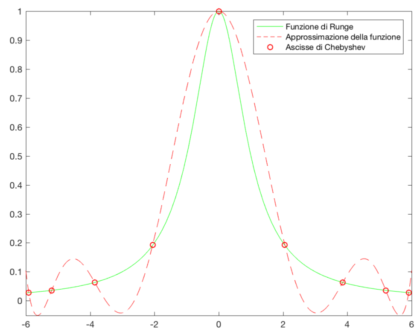
$n = 4$



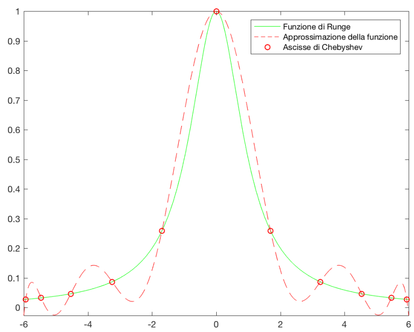
$n = 6$



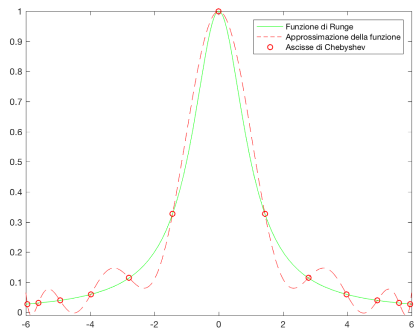
$n = 8$



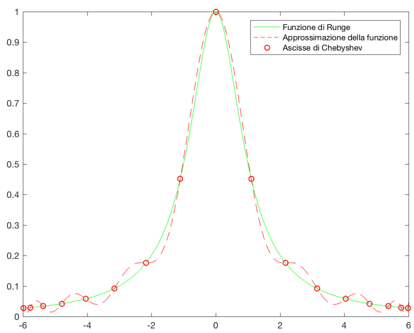
$n = 10$



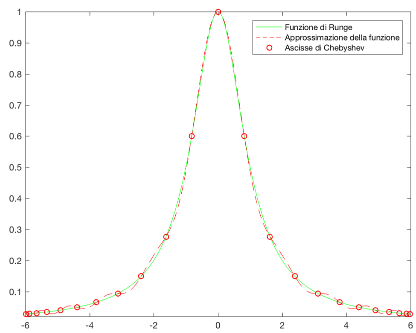
$n = 12$



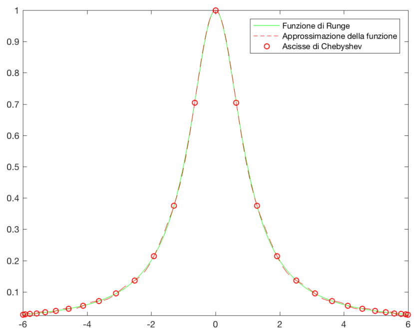
$n = 16$



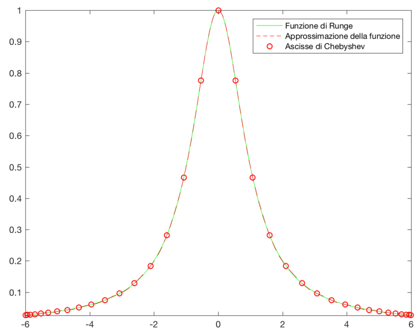
$n = 22$



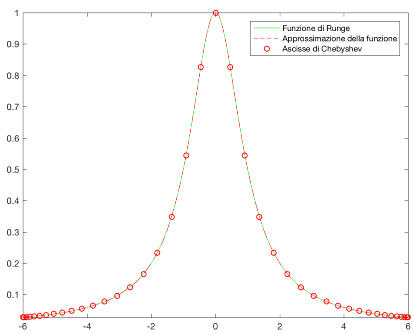
$n = 28$



$n = 30$



$n = 40$



L'errore è stato calcolato seguendo la seguente formula:

$$||e|| \approx ||f(x) - p_n(x)||_{\inf}$$

Dove f è intesa come la funzione di Runge e p il suo polinomio interpolante.

L'errore stimato è visibile nella seguente tabella:

n	errore
2	0.6577
4	0.4741
6	0.3371
8	0.2365
10	0.1640
12	0.1129
14	0.0772
16	0.0534
18	0.0399
20	0.0295
22	0.0216
24	0.0157
26	0.0113
28	0.0081
30	0.0058
32	0.0041
34	0.0029
36	0.0021
38	0.0015
40	0.0011

Notiamo che, utilizzando le ascisse di Chebyshev, aumentando il numero di punti otteniamo un'approssiamazione sempre più vicina alla funzione di Runge. Infatti l'errore diminuisce all'aumentare di n tendendo a 0 quando n tende all'infinito.

Esercizio 4.8 *Relativamente al precedente esercizio, stimare numericamente la crescita della costante di Lebesgue.*
Soluzione: La stima della costante di Lebesgue mediante le ascisse di Chebyshev è data dalla seguente formula:

$$\Lambda_n \approx \frac{2}{\pi} \log n$$

Ci si aspetta quindi che abbia una crescita logaritmica al crescere di n.

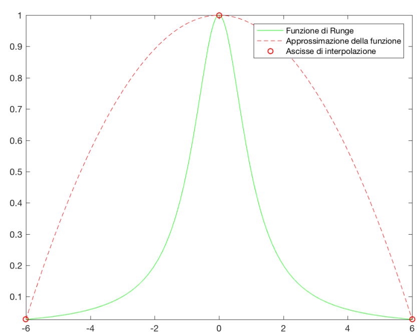
La tebella seguente mette in evidenza tale comportamento:

n	lebesgue
2	0.4413
4	0.8825
6	1.1407
8	1.3238
10	1.4659
12	1.5819
14	1.6801
16	1.7651
18	1.8401
20	1.9071
22	1.9678
24	2.0232
26	2.0742
28	2.1213
30	2.1653
32	2.2064
34	2.2450
36	2.2813
38	2.3158
40	2.3484

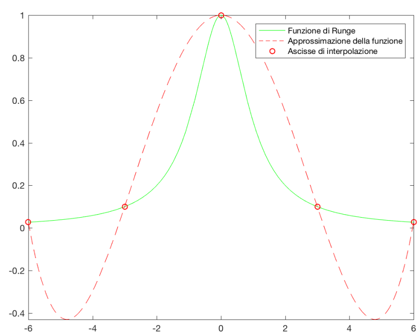
Esercizio 4.9 *Utilizzare la function ell'Esercizio 4.1 per approssimare la funzione di Runge sull'intervallo* $[-6, 6]$*, su una partizione uniforme di* $n + 1$ *ascisse per* $n = 2, 4, 6, \dots, 40$*. Stimare le corrispondenti costanti di Lebesgue.*

Soluzione:

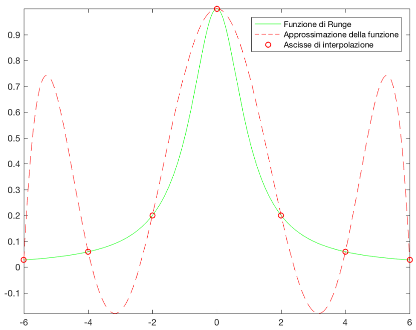
$n = 2$



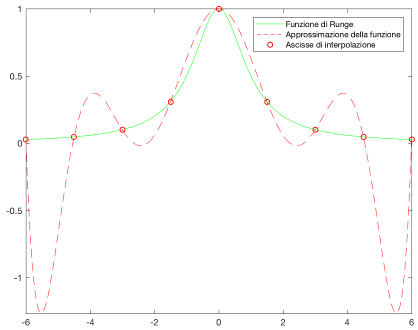
$n = 4$



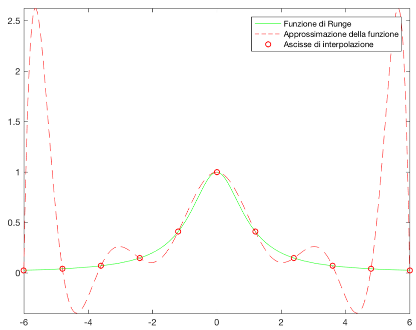
$n = 6$



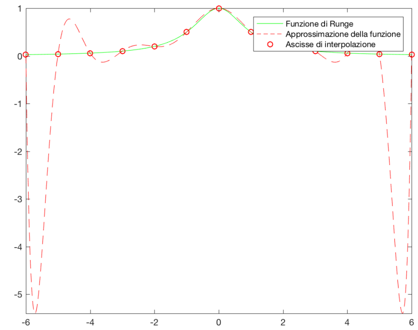
$n = 8$



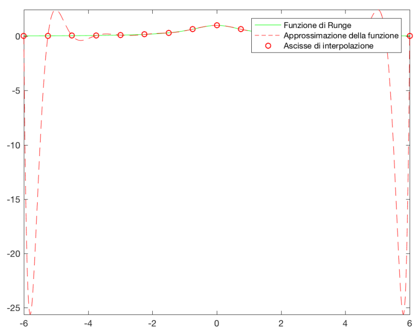
$n = 10$



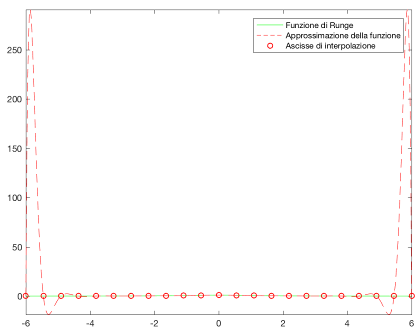
$n = 12$



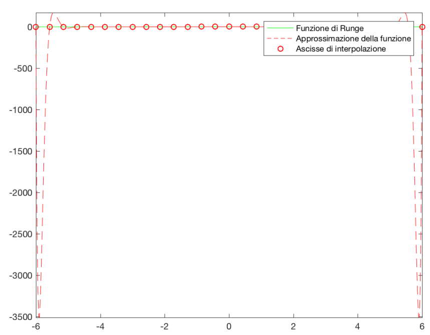
$n = 16$



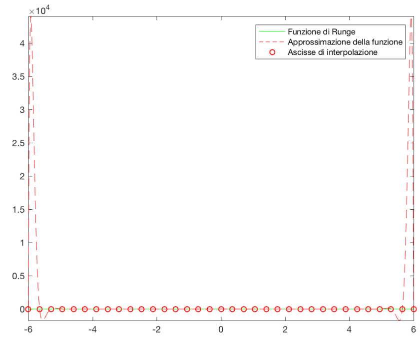
$n = 22$



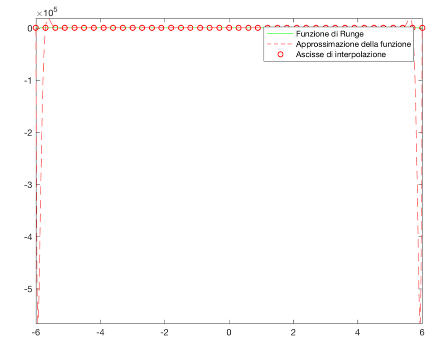
$n = 28$



$n = 34$



$n = 40$



Di seguito riportiamo la tabella con le stime degli errori e della costante di Lebesgue in funzione del grado n:

n	errore
2	0.6577
4	0.4741
6	0.3371
8	0.2365
10	0.1640
12	0.1129
14	0.0772
16	0.0534
18	0.0399
20	0.0295
22	0.0216
24	0.0157
26	0.0113
28	0.0081
30	0.0058
32	0.0041
34	0.0029
36	0.0021
38	0.0015
40	0.0011

Esercizio 4.10 *Stimare, nel senso dei minimi quadrati, posizione, velocità iniziale ed accelerazione relative ad un moto rettilineo uniformemente accelerato per cui sono note le seguenti misurazioni dele coppie (tempo, spazio):*
$(1, 2.9)$ $(1, 3.1)$ $(2, 6.9)$ $(2, 7.1)$ $(3, 12.9)$ $(3, 13.1)$ $(4, 20.9)$ $(4, 21.1)$ $(5, 30.9)$ $(5, 31.1)$
Soluzione: Il problema posto consiste nel risolvere

$$y = s(t) = x_0 + v_0 t + \frac{1}{2}at^2, \qquad \text{con } a, x_0 \text{ e } v_0 \text{ costanti}$$

La stima, nel senso dei minimi quadrati, equivale alla risoluzione del sistema lineare sovradeterminato Ax=b:

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 2 & 4 \\ 1 & 3 & 9 \\ 1 & 3 & 9 \\ 1 & 4 & 16 \\ 1 & 4 & 16 \\ 1 & 5 & 25 \\ 1 & 5 & 25 \end{pmatrix} \begin{pmatrix} x \\ v \\ a/2 \end{pmatrix} = \begin{pmatrix} 2.9 \\ 3.1 \\ 6.9 \\ 7.1 \\ 12.9 \\ 13.1 \\ 20.9 \\ 21.1 \\ 30.9 \\ 31.1 \end{pmatrix}$$

Il problema è ben posto ed esiste un'unica soluzione in quanot abbiamo misurazioni in $n + 1$ ascisse.
La matrice A è una matrice di Vandermonde con rango massimo 3.
É possibile calcolare una soluzione per il sistema dato utilizzando le function per la risoluzione e fattorizzazione dei sistemi QR sviluppati negli eserci precedenti:

$$\begin{pmatrix} x \\ v \\ a/2 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 0.99999 \end{pmatrix}$$