ICC Pràctica 2: Interpolació polinomial

Noah Márquez Vara

10 Desembre 2021

ÍNDEX

1 Algorisme de Horner

1.1 Avaluació d'un polinomi usant l'algorisme de Horner

L'algoritme de Horner s'utilitza per evaluar de forma eficient funcions polinòmiques de forma monomial.

Donat el polinomi

$$p(x) = a_0 + a_1 x + a_2 x^2 + a - 3x^3 + ... + a_n x^n \tag{1.1}$$

on $a_0, ..., a_n$ són nombres $\mathbb{R}$, volem avaluar el polinomi a un valor específic de x, diguem x_0.

Per dur a terme el procediment, definim una nova seqüència de constants com mostrem a continuació:

$$\begin{aligned} b_n &:= a_n \\ b_{n-1} &:= a_{n-1} + b_n x_0 \\ &\cdot \\ &\cdot \\ &\cdot \\ b_0 &:= a_o + b_1 x_0 \end{aligned} \tag{1.2}$$

Per veure com funciona això, ens podem fixar que el polinomi (1.1) pot escriure's de la forma:

$$p(x) = a_0 + x(a_1 + x(a_2 + ... + x(a_{n-1} + a_n x)...)) \tag{1.3}$$

Després, substituïnt iterativament la b_i en l'expressió:

$$\begin{aligned} p(x_0) &= a_0 + x_0(a_1 + x_0(a_2 + ... + x(a_{n-1} + b_n x_0)...)) \\ &= a_0 + x_0(a_1 + x_0(a_2 + ... + x(b_{n-1})...)) \\ &= a_0 + x_0(b_1) \\ &= b_0 \end{aligned} \tag{1.4}$$

També podem veure com calcula $p(z)$ el mètode de Horner de la següent manera:

$$p = c_n, \qquad \forall i = n-1, n-2, ..., 1, 0 \quad p \leftarrow p * (z - x_i) + c_i. \tag{1.5}$$

1.2 Codificació de la funció per evaluar l'algorisme de Horner

La funció que he codificat per tal d'avaluar un polinomi usant l'algorisme de Horner és la següent:

```
double horner(double z, double *x, double *c, int n){
    int i;
    double aval; /* Variable on s'acumula el resultat */

    /* Anem acumulant el resultat */
    aval = c[n];

    for(i = n-1; i >= 0; i--){
        /* Apliquem l'algorisme de Horner */
        aval = aval * (z - x[i]) + c[i];
    }
    return aval;
}
```

Aquesta funció rep com a paràmetres els següents termes:

- El valor on volem avaluar el polinomi **z**,
- El vector $\mathbf{x} = (x_0, x_1, ..., x_n)$ d'abcisses,
- El vector $\mathbf{c} = (c_0, c_1, ..., c_n)$ de coeficients,
- I el grau **n** del polinomi a evaluar.

La funció retornarà el valor

$$p(z) = \sum_{i=0}^{n} c_i \left(\prod_{j=0}^{i-1} (z - x_j) \right). \tag{1.6}$$

2 Diferències dividides de Newton

2.1 Càlcul de les diferències dividides de Newton

Partint de n punts (x, y), podem obtenir un polinomi de grau $n-1$ que passa pels anteriors punts. El mètode que utilitzarem és el de les diferències dividides de Newton per obtenir els coeficients c_j. Aquest mètode ens facilita la feina de resoldre un sistema d'equacions utilitzant el cocient de sumes i restes.

Donada una col·lecció de n punts de x i ls seves imatges $f(x)$, es poden calcular els coeficients del polinomi interpolador utilitzant les següents expressions:

$$\begin{aligned}
f[x_k] &= f(x_k) && k \in [0, n] \\
f[x_k, x_{k+1}] &= \frac{f[x_{k+1}] - f[x_k]}{x_{k+1} - x_k} && k \in [0, n-1] \\
f[x_k, x_{k+1}, x_{k+2}] &= \frac{f[x_{k+1}, x_{k+2}] - f[x_k, x_k+1]}{x_{k+2} - x_k} && k \in [0, n-2] \\
&\dots \\
f[x_k, x_{k+1}, ..., x_{k+i}] &= \frac{f[x_{k+1}, x_{k+2}, ..., x_{k+1}] - f[x_k, x_k+1, ..., x_{k+i-1}]}{x_{k+i} - x_k} && k \in [0, n-i]
\end{aligned} \tag{2.1}$$

Finalment, a partir dels valors obtinguts, es pot obtenir la següent forma de representar el polinomi:

$$\begin{aligned}
P_{n-1}(x) = f[x_0] + f[x_0, x_1] \cdot (x - x_0) + f[x_0, x_1, x_2] \cdot (x - x_0) \cdot (x - x_1) + \\
+ ... + f[x_0, x_1, ..., x_n] \cdot (x - x_0) \cdot (x - x_1) ... (x - x_{n-1})
\end{aligned} \tag{2.2}$$

Les diferències dividides, a més de com hem vist en (2.1), es poden obtenir de forma recursiva de la següent forma:

$$f[i] = (f[i] - f[i-1]) / (x[i] - x[i-k]) \quad \forall i = n, n-1, ..., k; \quad \forall k = 1, 2, ..., n. \tag{2.3}$$

2.2 Funció per calcular les diferències dividides de Newton

La funció que s'ha codificat per tal de calcular les diferències dividides és la següent:

```c
int difdiv(double *x, double *f, int n){
    int i, k;
    double tolerancia = 1.e-12; /* Tolerancia per la qual el proces no continua*/

    for(k = 1; k <= n; k++){
        for(i = n; i >= k; i--){
            /* Comprovem que el denominador es >= que la tolerancia per tal de poder continuar */
            if(fabs((x[i] - x[i-k])) >= tolerancia){
                /* Calcul de les diferencies dividides de forma recursiva */
                f[i] = (f[i] - f[i-1]) / (x[i] - x[i-k]);
            } else {
                return -1;
            }
        }
    }
    return 0;
}
```

Aquesta funció rep com a paràmetres els següents termes:

- El vector d'abcisses $\mathbf{x} = (x_0, x_1, ..., x_n)$,

- Els valors de la funció a interpolar en les abcisses $\mathbf{f} = (f_0, f_1, ..., f_n)$,
- I **n**, que ens indica el grau del polinomi interpolador.

A la sortida, el vector **f** contindrà les diferències dividides associades a la taula de valors (x_i, f_i), $i = 0, 1, ..., n$.

Si cap dels denominadors (en valor absolut) és $< 10^{-12}$ (tolerància), el procés continua i retorna 0. Altrament, retorna -1.

Q: Com es relaciona la forma recursiva anterior amb l'esquema triangular del càlcul de les diferències dividides de Newton (p.11 slides de teoria)?

A: Adjunto primerament l'esquema triangular del càlcul de les diferències dividides de Newton:

$$
\begin{array}{c|cccc}
x_0 & f[x_0] = y_0 = c_0 & & & \\
 & & f[x_0, x_1] = \frac{f[x_1]-f[x_0]}{x_1 - x_0} = c_1 & & \\
x_1 & f[x_1] = y_1 & & f[x_0, x_1, x_2] = \frac{f[x_1,x_2]-f[x_0,x_1]}{x_2 - x_0} = c_2 & \\
 & & f[x_1, x_2] = \frac{f[x_2]-f[x_1]}{x_2 - x_1} & & \\
x_2 & f[x_2] = y_2 & & & \\
\vdots & \vdots & \vdots & \ldots & \\
\vdots & \vdots & f[x_{n-1}, x_n] = \frac{f[x_n]-f[x_{n-1}]}{x_n - x_{n-1}} & & \\
x_n & f[x_n] = y_n & & &
\end{array}
$$

Figure 2.1: Esquema triangular de les diferències dividides

Com podem comprovar a la forma recursiva, per poder calcular qualsevol $f[i]$ o c_j com indiquen les slides de teoria, necessitarem el valor del propi $f(i)$ i l'anterior $f(i-1)$ i a més, la component de les abcisses x_i i la component x_{i-k}, que ens tancarà l'esquema triangular per tal de poder fer el càlcul de les diferències dividides de Newton de cada component c_j.

En l'equació (2.1) es mostra com es pot realitzar el càlcul dels coeficients del polinomi interpolador donada una colecció de n punts i les seves imatges.

3 MAIN_TAULA.C

En aquest apartat he implementat una funció principal (guardada en ***main_taula.c***) que llegeix les abcisses i les ordenades d'un fitxer (***taula.in***, que conté dues columnes amb x_k i $f(x_k)$ respectivament). Posteriorment llegeix per terminal el grau del polinomi n, llegeix els extrems d'un interval $[a, b]$ i escriu en un fitxer (***p3taula.out***, que contindrà dues columnes amb el punt x_k i la imatge del punt $f(x_k)$) el resultat d'avaluar el polinomi interpolador en una xarxa de 1000 punts equidistants en l'interval $[a, b]$.

La funció principal és la següent:

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "funs_interp.h"

int main (void){
    double *x, *o;
    int i, n;
    double a, b, distanciaPunts, polinomi, punt;
    FILE *entrada;

    /* Grau polinomi interpolador*/
    printf("#Grau del polinomi interpolador:\n");
    scanf ("%d", &n);

    /* Extrems d'un interval [a, b] */
    printf("#Extrem a de l'interval:\n");
    scanf ("%le", &a);

    printf("#Extrem b de l'interval:\n");
    scanf ("%le", &b);

    entrada = fopen("taula.in", "r");

    if(entrada == NULL){
        printf("Error en obrir el fitxer %s\n", "taula.in");
        return 1;
    }

    /* Reservar espai de mem ria per als vectors*/
    x = (double *) malloc ((n+1) * sizeof(double));
    o = (double *) malloc ((n+1) * sizeof(double));

    if (x == NULL) {
        printf ("No hi ha prou memoria\n");
        exit (1);
    }

    if (o == NULL) {
        printf("No hi ha prou memoria\n");
        exit(2);
    }

    /* Abcisses i Ordenades */
    printf("#Valors de les abcisses i les ordenades:\n");
    for (i = 0; i <= n; i++) {
        fscanf(entrada, "%le", &x[i]);
        fscanf(entrada, "%le", &o[i]);
    }

```

```c
    /* Cridem al m tode de les difer ncies dividides, si retorna 0 podem continuar */
    if(difdiv(x, o, n) == 0){

        /* Com que volem 1000 punts equidistants entre a i b ([a,b]), apliquem la
    seg ent f rmula */
        distanciaPunts = (b-a) / (999);

        for(i = 0; i < 1000; i++){
            punt = a + (i * distanciaPunts);
            polinomi = horner(punt, x, o, n);
            printf("%e %e\n", punt, polinomi);
        }
    } else{
        printf("No s'han pogut dur a terme les difer ncies dividides\n");
    }

    /* Alliberem mem ria */
    free(x);
    free(o);
    fclose(entrada);

    return 0;
}
```

Per tal de poder introduir bé les dades, el que he fet és demanar primer el grau del polinomi i els extrems de l'interval $[a, b]$ per posteriorment llegir el fitxer ***taula.in*** per tal d'omplir les dades correctament.

El primer cop que vaig intentar codificar aquesta funció no retornava la sortida esperada, i era perquè al calcular el punt dins de l'últim *for* em faltava sumar l'extrem de l'interval a.

He utilitzat aquesta funció per tal de representar el polinomi interpolador $p_3(x)$ corresponent a la taula:

x_k	1	2.7	3.2	4.8
$f(x_k)$	14.2	17.8	22.0	38.3

Per tal de facilitar l'execució de les comandes de gnuplot, he guardat la comanda que utilitzaré per representar la "corba" en un fitxer anomenat ***taula.gnu*** (el fitxer també estarà a la carpeta); el fitxer conté les següents dues linees:

- ***set yrange[10:40]*** per tal de fixar un rang de la y per la correcta visualització de la representació.
- ***plot 'p3taula.out' w d, 'taula.in' w p pt 7 lc 2 ps 1.5***, comanda amb què representem la corba.

La representació és la següent:

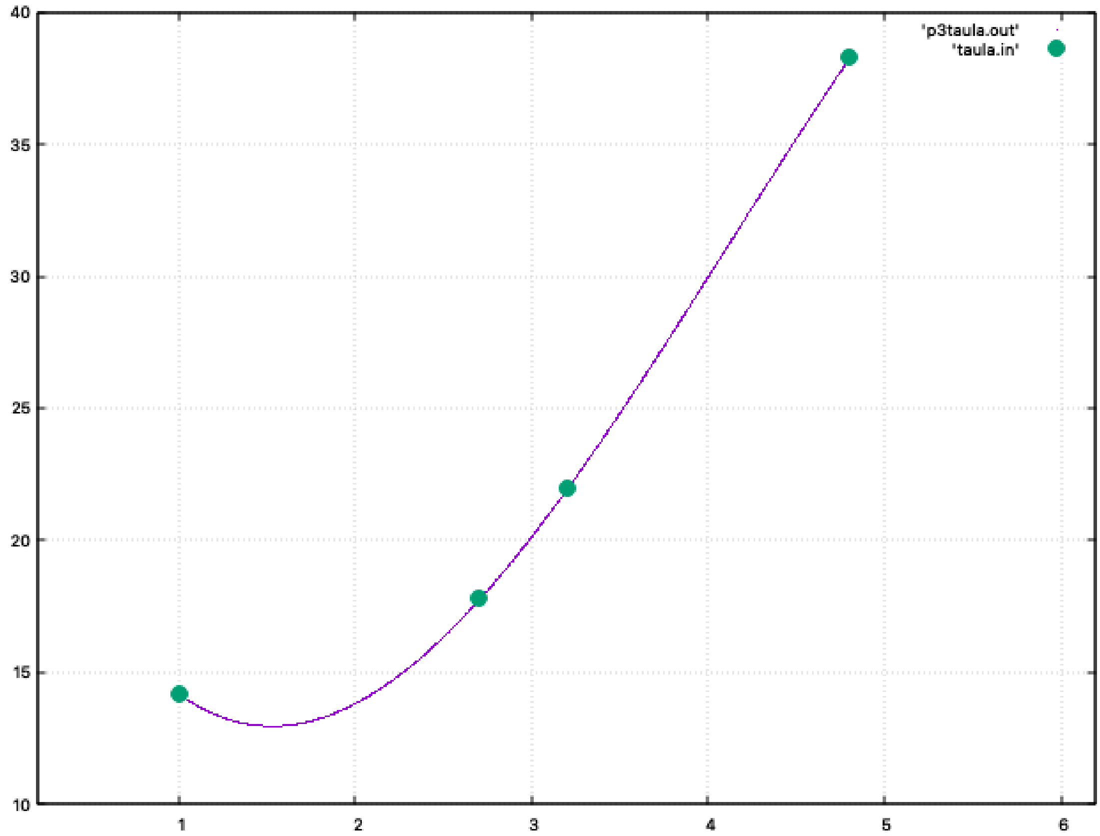


Figure 3.1: Representació de la "corba" corresponent a la taula.

Com podem veure, el polinomi interpolador passa per tots els punts de la taula com caldria esperar.

4 MAIN_ERRINTERP.C

L'objectiu d'aquest apartat és arribar a estudiar l'error en interpolar per un polinomi de grau n una determinada funció f en un interval $[a,b]$.

Per fer-ho he implementat una funció principal (***main_errinterp.c***) que llegeix el grau del polinomi d'interpolació n i els extrems de l'interval $[a,b]$ on s'interpolarà la funció usant nodes equidistants; també s'introdueix el nom del fitxer de sortida.

Aquesta funció considerarà una xarxa de 1000 punts equidistants z_j en $[a,b]$ i escriurà en el fitxer de sortida amb estructura:

$$z_j \qquad f(z_j) \qquad p_n(z_j)$$

L'última línia del fitxer de sortida comença amb el caràcter # (indicant que és un comentari de ***gnuplot***, per tant no es tindrà en compte a l'hora de representar), i en la mateixa línia s'escriu el màxim de $|f(z_j) - p_n(z_j)|$ en la xarxa de punts considerada.

Com per paràmetres només ens indiquen el grau del polinomi d'interpolació i els extrems de l'interval on s'interpolarà la funció, abans de procedir amb els càlculs del programa haurem de calcular la xarxa de 1000 punts equidistants z_j (abcisses) en $[a,b]$ i la seva imatge $f(z_j)$ (ordenades). Per fer això, primer calculem la distància entre els punts que ve donada per la següent fòrmula:

$$distanciaPunts = \frac{b-a}{n} \tag{4.1}$$

Posteriorment es procedeix al càlcul de les abcisses i de les ordenades (quedarà detallat en el programa adjuntat).

Ens demanen estudiar l'error per dues funcions:

- Logaritme neperìa (s'escollirà amb un 1 al programa)
- Funció de Runge (s'escollirà amb un 0 al programa)

Totes dues estan explicades en els següents apartats (4.1 i 4.2, respectivament).

Abans, però, detallo a continuació com s'ha realitzat la funció principal:

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "funs_interp.h"

int main (void){
    double *x, *o;
    int i, n, u;
    double a, b, distanciaPunts, polinomi, punt, funcio;
    double max = 0;
    char fname[20];
    FILE *sortida;

    /* Grau polinomi interpolacio */
    printf("#Grau del polinomi d'interpolacio:\n");
    scanf ("%d", &n);

    /* Reservar espai de memoria per als vectors*/
    x = (double *) malloc ((n+1) * sizeof(double));
    o = (double *) malloc ((n+1) * sizeof(double));
```

```c

    if (x == NULL) {
        printf ("No hi ha prou memoria\n");
        exit (1);
    }

    if (o == NULL) {
        printf("No hi ha prou memoria\n");
        exit(2);
    }

    /* Extrems d'un interval [a, b] */
    printf("#Extrem a de l'interval:\n");
    scanf ("%le", &a);

    printf("#Extrem b de l'interval:\n");
    scanf ("%le", &b);

    /* Nom del fitxer de sortida */
    printf("#Nom del fitxer de sortida:\n");
    scanf("%s", fname);

    /* Obrim el fitxer de sortida en mode 'write' */
    sortida = fopen(fname, "w");

    if(sortida == NULL){
        printf("Error en obrir el fitxer %s\n", "taula.in");
        return 1;
    }

    /* Eleccio de la funcio que es fara servir */
    printf("#Funcio ln [1] o funcio runge [0]?:\n");
    scanf ("%d", &u);

    /* Si no s'escull una funcio correcte */
    while(u != 0 && u != 1){
        printf("#Funcio ln [1] o funcio runge [0]?:\n");
        scanf ("%d", &u);
    }

    /* Com que volem n punts equidistants entre a i b, aplicare la seguent formula */
    distanciaPunts = (b-a) /(n);

    /* Funcio logaritme neperia*/
    if(u == 1){
        /* Omplim els vectors amb les dades necessaries (abscisses i ordenades) */
        for(i = 0; i < n+1; i++) {
            x[i] = a + (i * distanciaPunts);
            o[i] = fun_log(x[i]);
        }
    }
    /* Funcio Runge */
    else{
        /* Omplim els vectors amb les dades necessaries (abscisses i ordenades) */
        for(i = 0; i < n+1; i++) {
            x[i] = a + i * distanciaPunts;
            o[i] = fun_runge(x[i]);
        }
    }

    /* Cridem al metode de les diferencies dividides, si retorna 0 podem continuar */
    if(difdiv(x, o, n) == 0){
```

```
        /* Com que volem 1000 punts equidistants entre a i b ([a,b]), apliquem la seguent
    formula */
        distanciaPunts = (b-a) / (999);

        for(i = 0; i < 1000; i++){
            punt = a + (i * distanciaPunts);

            /* logaritme neperia*/
            if(u == 1){
                funcio = fun_log(punt);
            }
            /* Funcio de runge */
            else{
                funcio = fun_runge(punt);
            }
            polinomi = horner(punt, x, o, n);

            fprintf(sortida, "%e %e %e\n", punt, funcio, polinomi);

            /* Si es necessari actualitzem el maxim */
            if(max < fabs(funcio - polinomi)){
                max = fabs(funcio - polinomi);
            }
        }
    } else{
        printf("No s'han pogut dur a terme les diferencies dividides\n");
    }

    fprintf(sortida, "#%e", max);

    /* Alliberem memoria */
    free(x);
    free(o);
    fclose(sortida);

    return 0;
}
```

4.1 Logaritme neperià

Hem de considerar la següent funció:

$$f(x) = ln(x) \tag{4.2}$$

i abcisses equiespaiades en $[a, b] = [0.4, 0.8]$.

Per tal d'avaluar $f(x)$ he implementat la següent funció:

```
double fun_log(double z){
    return log(z);
}
```

La funció $log()$ de la llibreria ***math.h*** ens retorna el logaritme neperià.

A continuació se'ns demana representar els polinomis $p_n(x), 1 \leq n \leq 5$ i les gràfiques de l'error d'interpolació comés. Els noms dels fitxers de sortida seran: **log1.out**, ..., **log5.out**.

Per tal de representar més ràpidament els polinomis, he guardat en un fitxer (***log.gnu***, que estarà a la carpeta de l'entrega) les comandes a executar (entre cada comanda hi ha un *pause -1*, per tal de no mostrar la següent comanda fins que es pulsa *intro*). En aquest fitxer estan guardades les comandes tant per representar els polinomis (usant les columnes 1:3 del fitxer) com per representar les gràfiques de l'error comés (usant les columnes 1:2 i 1:3 del fitxer, per comparar l'error d'avaluar entre la funció i el polinomi interpolador).

Per tal de poder apreciar correctament la gràfica de l'error d'interpolació comés, m'he ajudat de l'eina *multiplot* per tal de graficar una taula amb *zoom*. Les instruccions que he utilitzat també estan en el fitxer ***log.gnu*** (on es podrà apreciar que he hagut d'utilitzar un *range* de y bastant petit per tal de poder apreciar correctament l'error), tot i que les he deixat comentades perquè entre cada *multiplot* havia de fer un *quit* abans de fer el següent, ja que no em funcionava correctament la instrucció *unset multiplot*.

A continuació la representació dels polinomis $p_n(x), 1 \leq n \leq 5$:

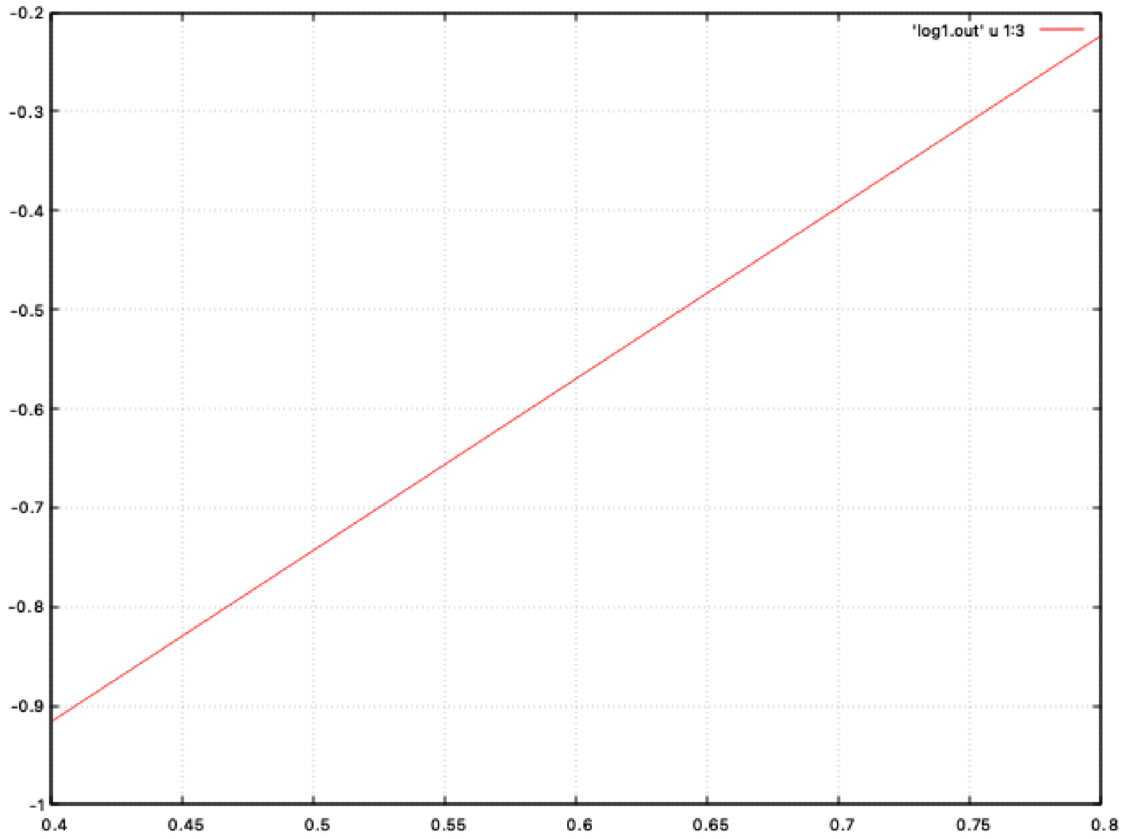


Figure 4.1: Representació del polinomi $p_1(x)$

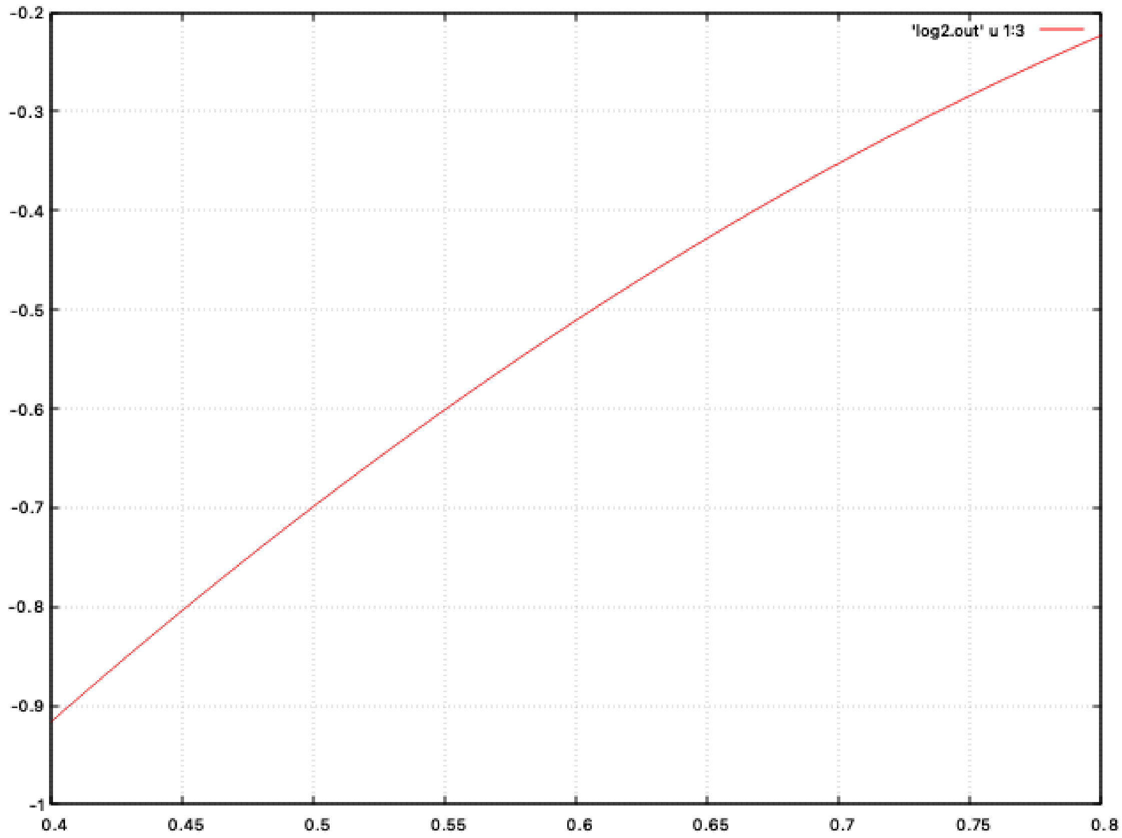


Figure 4.2: Representació del polinomi $p_2(x)$

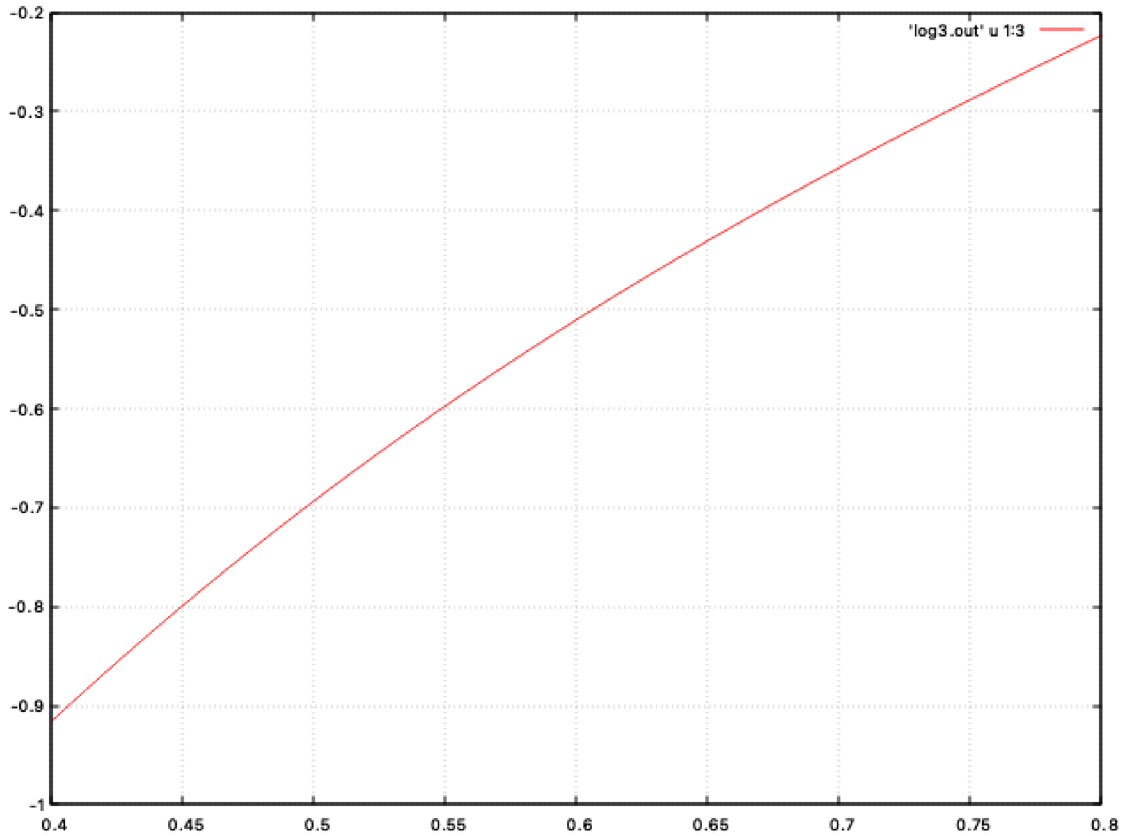


Figure 4.3: Representació del polinomi $p_3(x)$

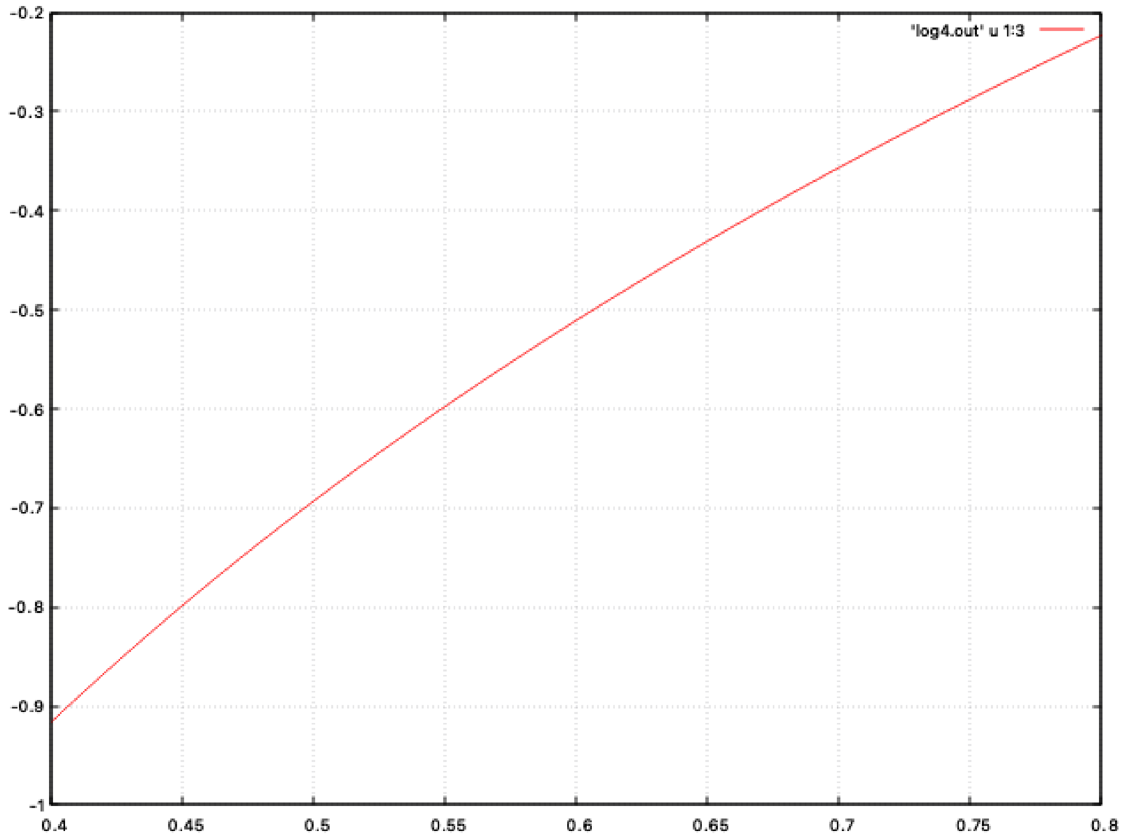


Figure 4.4: Representació del polinomi $p_4(x)$

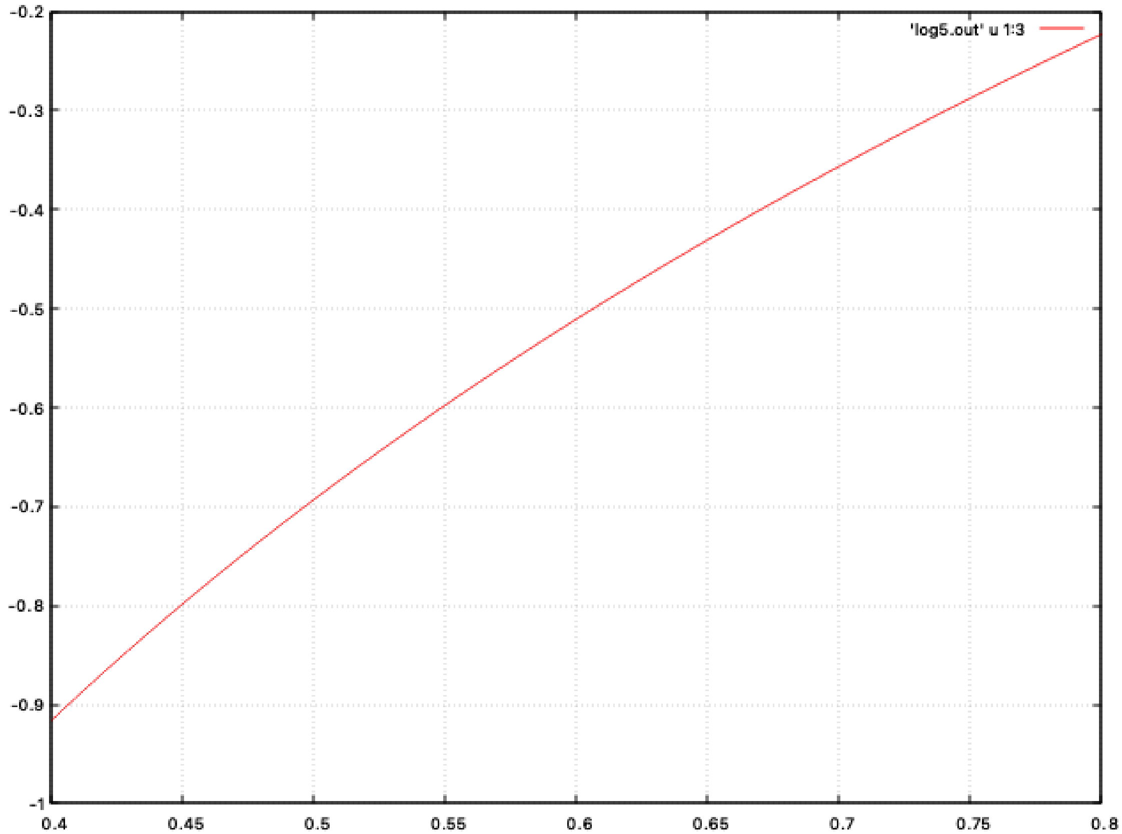


Figure 4.5: Representació del polinomi $p_5(x)$

A continuació es mostren les gràfiques de l'error d'interpolació per a cada polinomi (**línia verda** $f(x)$**, linia blava** $p_n(x)$):

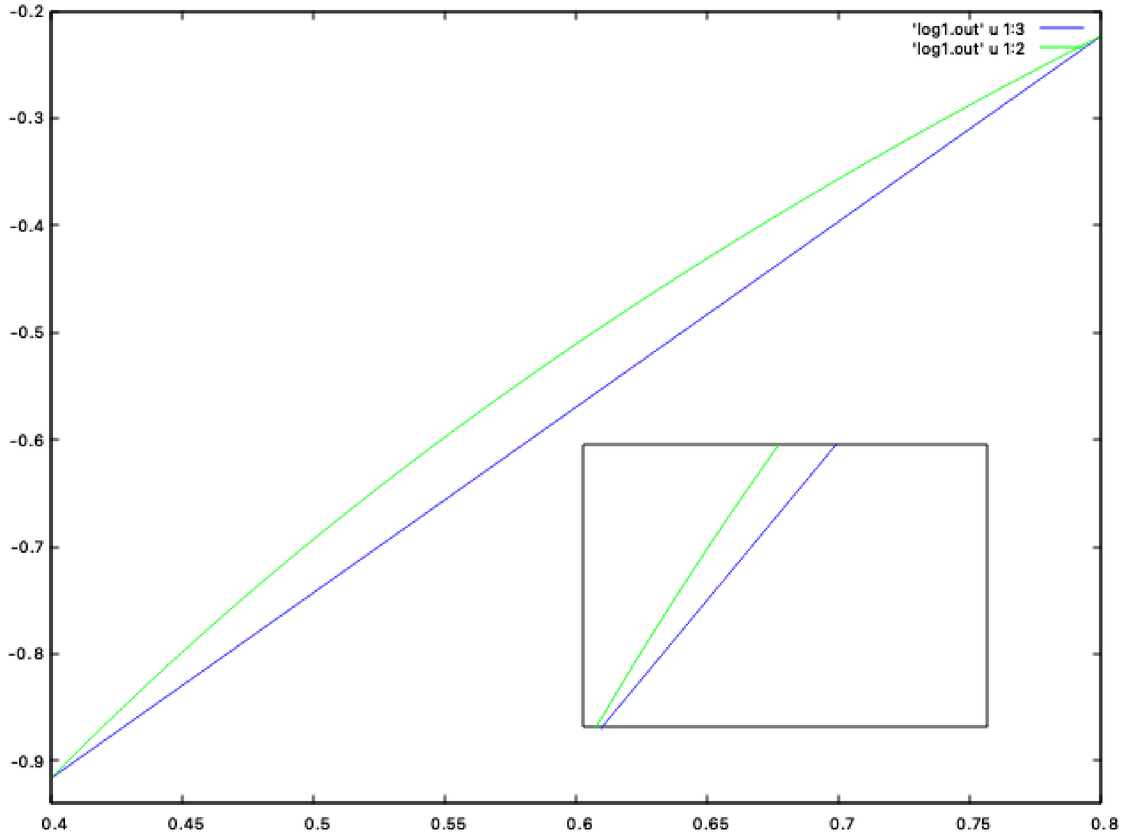


Figure 4.6: Gràfica de l'error d'interpolació del polinomi $p_1(x)$

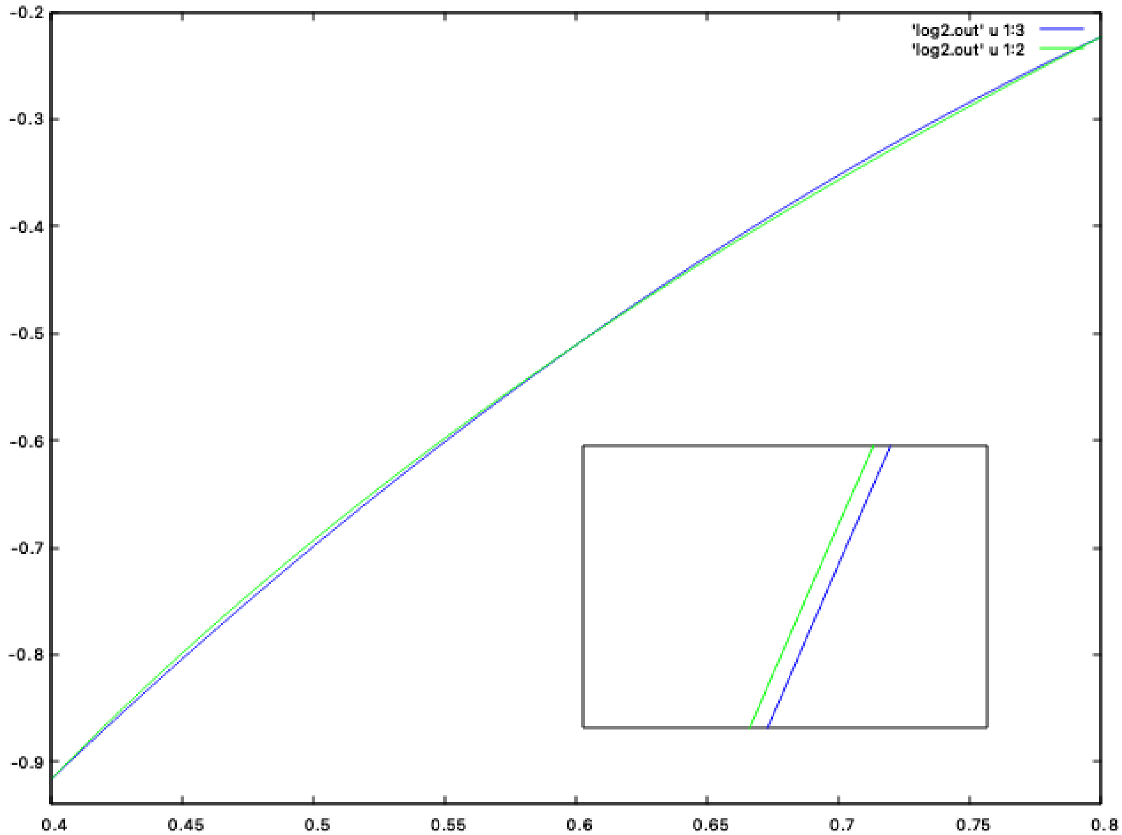


Figure 4.7: Gràfica de l'error d'interpolació del polinomi $p_2(x)$

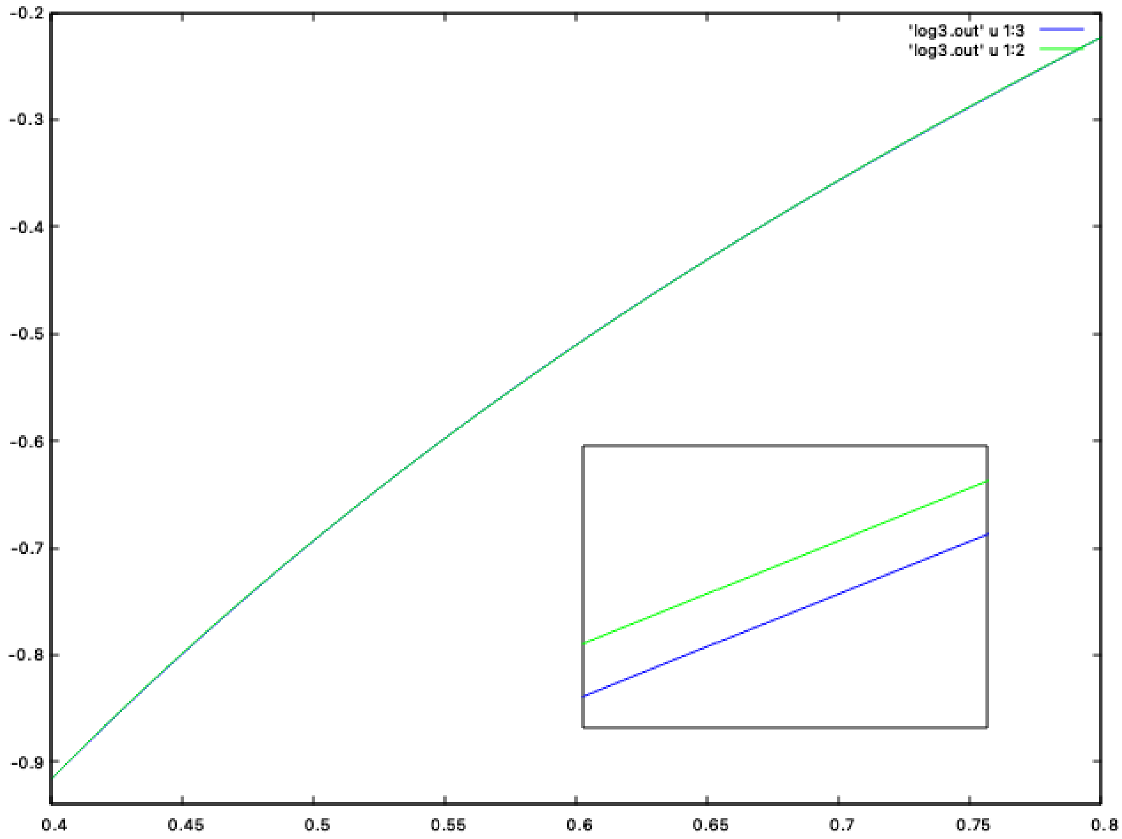


Figure 4.8: Gràfica de l'error d'interpolació del polinomi $p_3(x)$

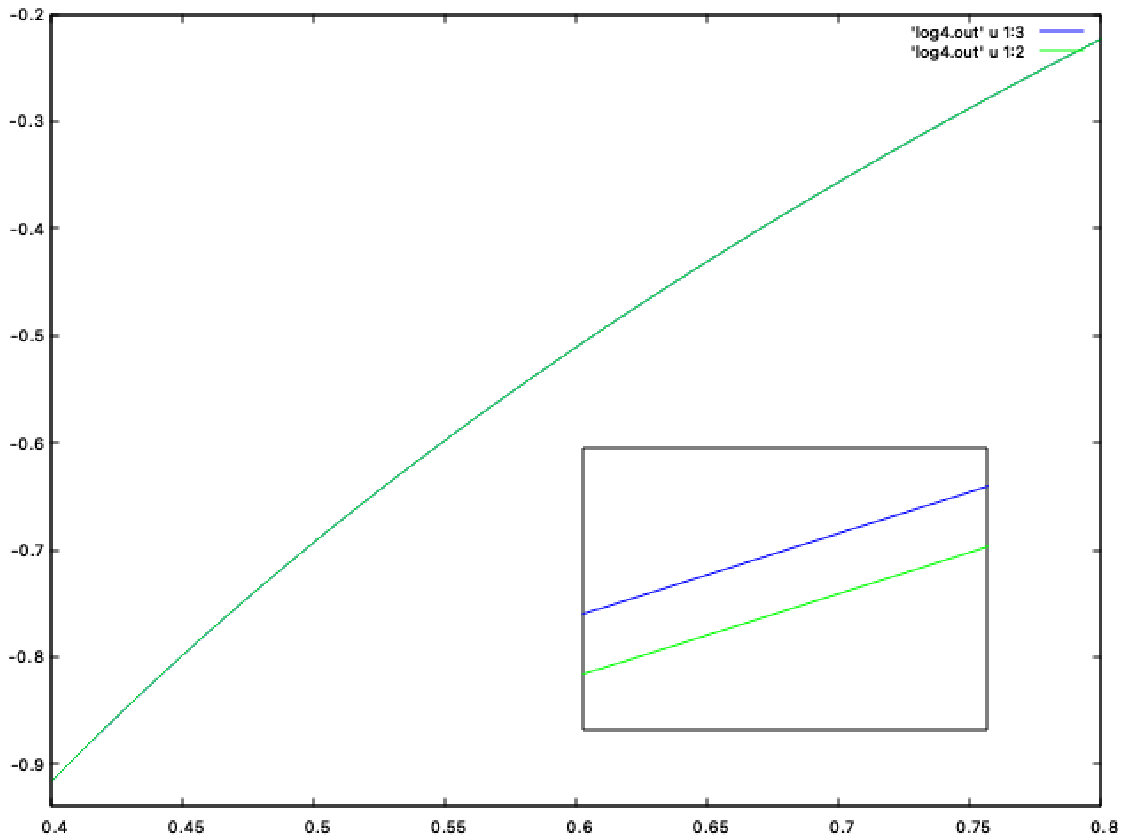


Figure 4.9: Gràfica de l'error d'interpolació del polinomi $p_4(x)$

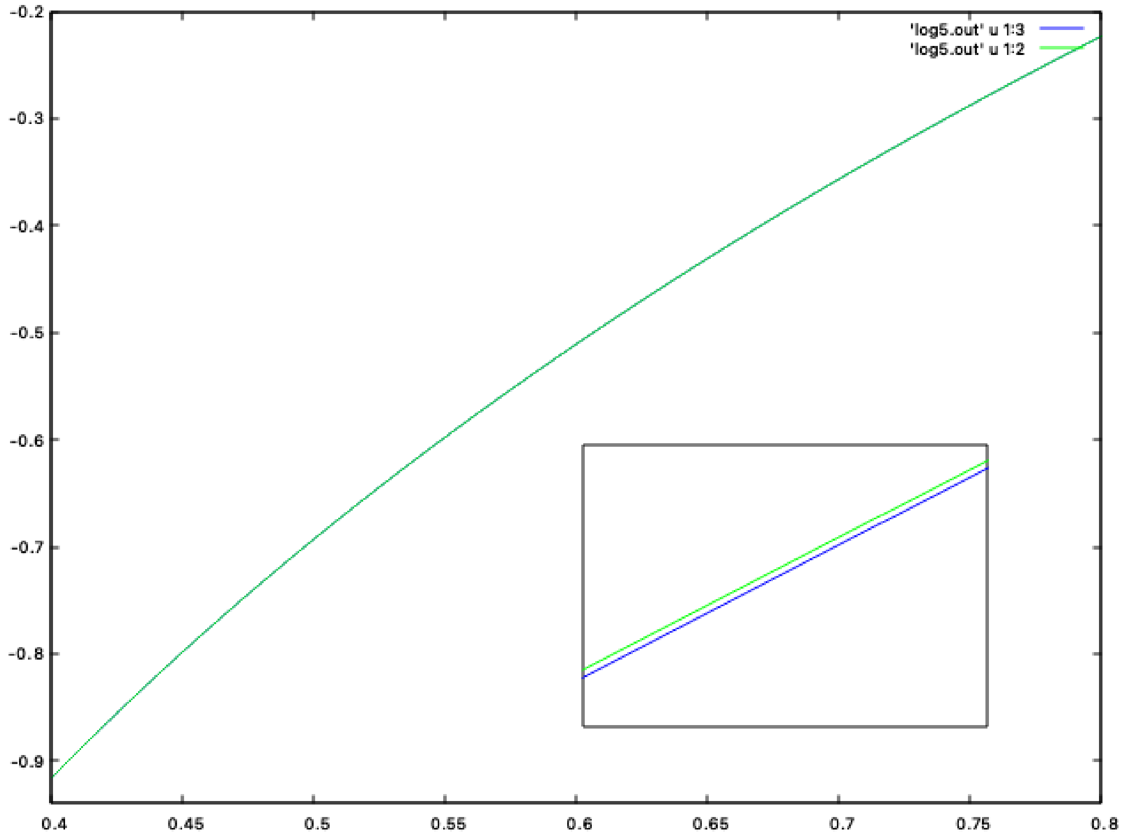


Figure 4.10: Gràfica de l'error d'interpolació del polinomi $p_5(x)$

Adjunto també taula amb el màxim de $|f(z_j) - p_n(z_j)|$ per a cada polinomi interpolador:

n	|f(z_j) - p_n(z_j)|
1	$5.966009e-02$
2	$6.004492e-03$
3	$8.347280e-04$
4	$1.349808e-04$
5	$2.385247e-05$

Això en una gràfica resulta en:

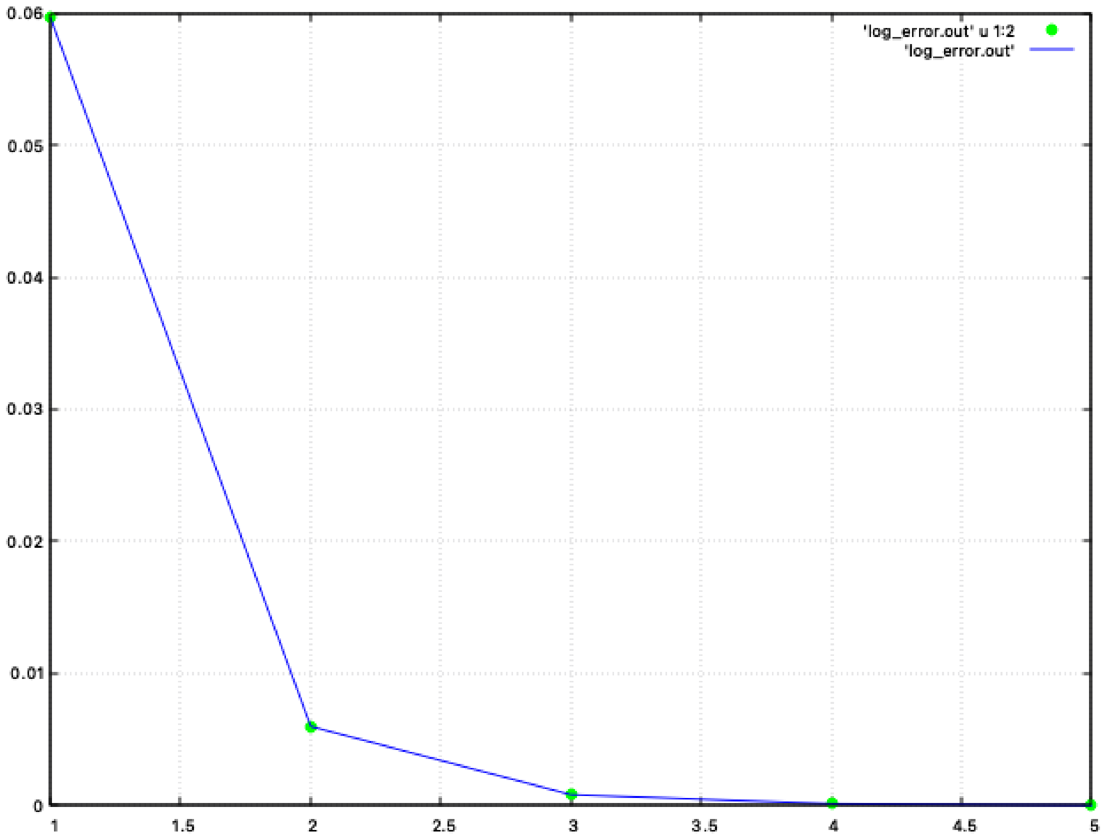


Figure 4.11: Gràfica del error segons la n

Q: És compatible l'error d'interpolació observat amb la fita de l'error en nodes equiespaiats (p.16 slides teoria)? Calculeu la fita i compareu.

A:Primerament calcularem la fita per tal de comparar:

Si $x_i = x_0 + i \cdot h$ per $i = 0, 1, ..., n$, amb $h = \frac{(b-a)}{n}$, i $|f^{(n+1)}(x)| \leq M_{n+1} \forall x \in [a, b]$, llavors:

$$|f(x) - p_n(x)| \leq \frac{M_{n+1}}{4(n+1)} \left[\frac{b-a}{n}\right]^{n+1}. \tag{4.3}$$

Si s'interpola la funció $f(x) = ln(x)$ per un polinomi de grau màxim 5 emprant nodes equiespaiats, l'error màxim comès serà:

$$\begin{aligned}
|f(x) - p_5(x)| &\leq \frac{M_{5+1}}{4(5+1)}\left[\frac{0.8-0.4}{5}\right]^{5+1}\\
|f(x) - p_5(x)| &\leq \frac{M_6}{4(6)}\left[\frac{0.8-0.4}{5}\right]^{6}\\
|f(x) - p_5(x)| &\leq \frac{M_6}{24}\left[\frac{0.4}{5}\right]^{6}\\
|f(x) - p_5(x)| &\leq \frac{M_6}{24}\left[\frac{2}{25}\right]^{6}
\end{aligned} \tag{4.4}$$

On M_6 és:

$$\max_{[0.4,0.8]} |f^{(6)}(\xi)| \tag{4.5}$$

i on $\xi \in [0.4, 0.8]$.

La derivada sisena de $ln(x)$ és la següent:

$$\frac{d^6}{dx^6}[ln(x)] = -\frac{120}{x^6} \tag{4.6}$$

M_6 (4.5) és igual a:

$$\begin{aligned}
\max_{[0.4,0.8]} |f^{(6)}(\xi)| &= \max_{[0.4,0.8]} |-\frac{120}{x^6}|\\
\max_{[0.4,0.8]} |f^{(6)}(\xi)| &= \frac{120}{0.4^6}
\end{aligned} \tag{4.7}$$

Llavors, per a qualsevol punt $x \in [0.4, 0.8]$, es pot fitar l'error comès per:

$$|f(x) - p_5(x)| \leq \frac{\frac{120}{0.4^6}}{24}\left[\frac{2}{25}\right]^{6} \leq 3.2\cdot 10^{-4} \tag{4.8}$$

Podem comprovar que l'error obtingut és compatible amb la fita de l'error en nodes equiespaiats.

A continuació indico les fites per la resta de polinomis interpoladors obviant els càlculs:

$$\begin{aligned}
|f(x) - p_4(x)| &\leq \frac{\frac{24}{0.4^5}}{20}\left[\frac{1}{10}\right]^{5} \leq 1.2\cdot 10^{-3}\\
|f(x) - p_3(x)| &\leq \frac{\frac{6}{0.4^4}}{16}\left[\frac{2}{15}\right]^{4} \leq 5\cdot 10^{-3}\\
|f(x) - p_2(x)| &\leq \frac{\frac{2}{0.4^3}}{12}\left[\frac{1}{15}\right]^{3} \leq 2.1\cdot 10^{-2}\\
|f(x) - p_1(x)| &\leq \frac{\frac{1}{0.4^2}}{8}\left[\frac{2}{5}\right]^{2} \leq 1.25\cdot 10^{-1}
\end{aligned} \tag{4.9}$$

Com podem comprovar, els errors d'interpolació observats són compatibles amb les fites de l'error en nodes equiespaiats.

4.2 Funció de Runge

Hem de considerar la següent funció:

$$f(x) = \frac{1}{1+25x^2} \tag{4.10}$$

amb abcisses equiespaidades a l'interval $[a, b] = [-1, 1]$.

Per avaluar $f(x)$ he implementat la següent funció:

```
double fun_runge(double z){
    return 1/(1+(25*z*z));
}
```

D'aquesta forma podem avaluar la funció (6.1) i a més sense fer ús de la funció ***pow***.

A continuació se'ns demana representar els polinomis $p_n(x), 1 \leq n \leq 10$ i les gràfiques de l'error comés. Els noms dels fitxers de sortida seran: **runge1.out**, ..., **runge10.out**.

Per tal de representar més ràpidament els polinomis, he guardat en un fitxer (***runge.gnu***, que estarà a la carpeta de l'entrega) les comandes a executar (entre cada comanda hi ha un *pause -1*, per tal de no mostrar la següent comanda fins que es pulsa *intro*). En aquest fitxer estan guardades les comandes tant per representar els polinomis (usant les columnes 1:3 del fitxer) com per a l'hora representar les gràfiques de l'error comés (usant les columnes 1:2 i 1:3 del fitxer, per comparar l'error d'avaluar entre la funció i el polinomi interpolador). Abans de cada instrucció, si és necessari, he fet un ***set yrange*** per tal de fitxar el rang de y.

A continuació la representació dels polinomis $p_n(x), 1 \leq n \leq 10$ (represento els polinomis i les gràfiques de l'error d'interpolació a l'hora per tal d'estalviar espai) (**línia verda** $f(x)$**, linia blava** $p_n(x)$):

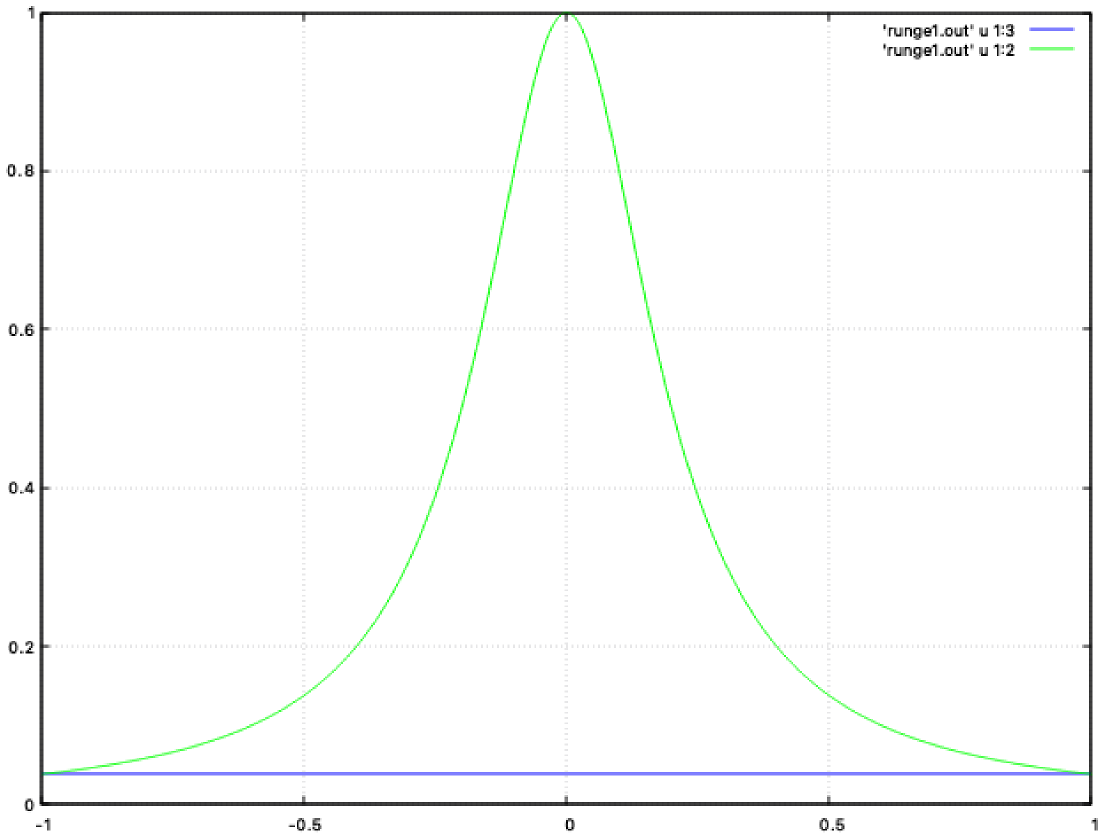


Figure 4.12: Representació del polinomi $p_1(x)$

Com podem veure, el polinomi interpolador $p_1(x)$ evaluat en 1000 punts equiespaiats en l'interval $[-1, 1]$ sempre té el mateix valor. Això és degut a que tenim la següent taula de valors:

x_k	$f(x_k)$
-1	$0.0\overline{384615}$
1	$0.0\overline{384615}$

I a l'hora de fer el càlcul del polinomi interpolador amb la funció de les diferències dividides, ens retorna el següent:

$$p_1(x) = 0.0\overline{384615} \tag{4.11}$$

Per tant, a l'hora de cridar a la funció de Horner en els 1000 punts, sempre retona el mateix valor (el del polinomi).

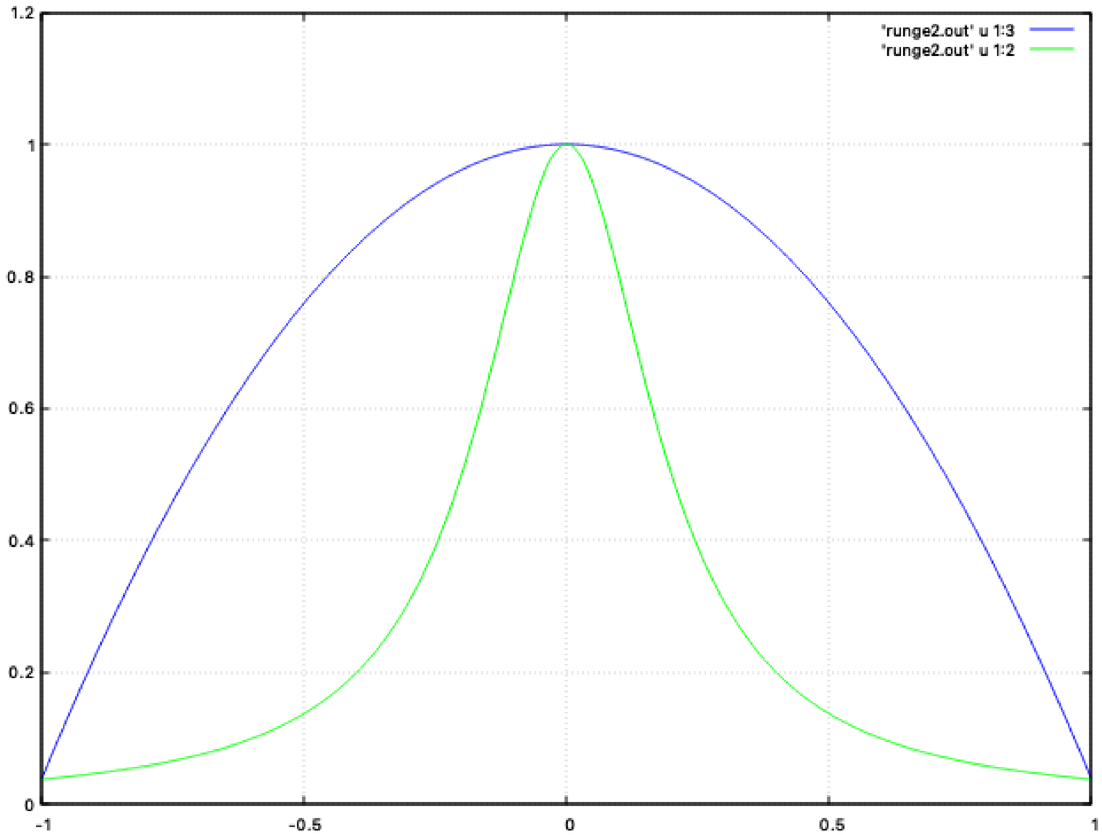


Figure 4.13: Representació del polinomi $p_2(x)$

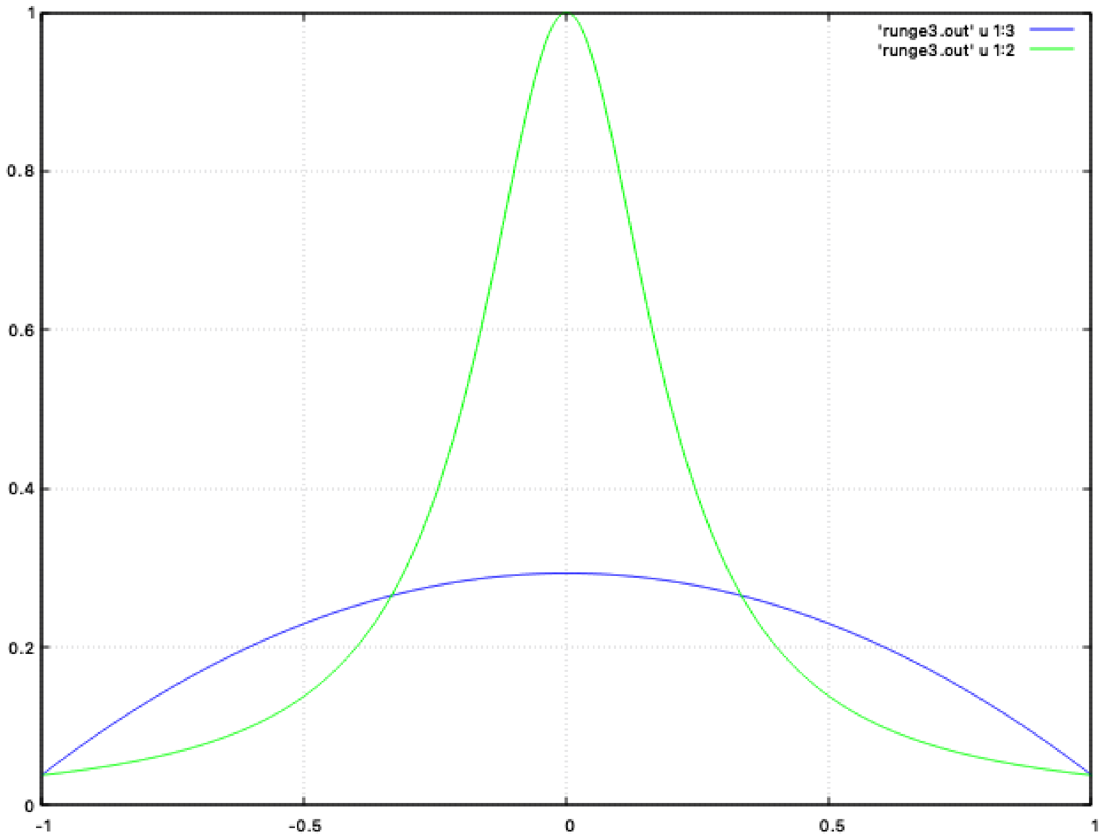


Figure 4.14: Representació del polinomi $p_3(x)$

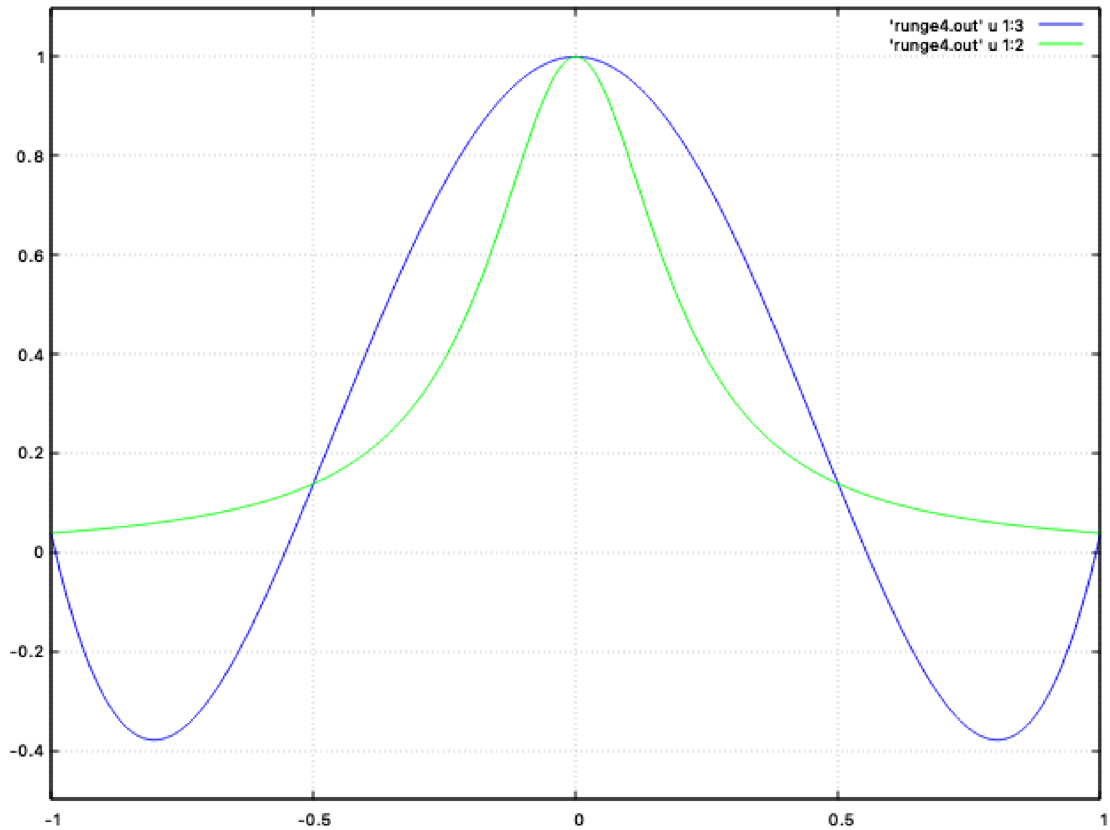


Figure 4.15: Representació del polinomi $p_4(x)$

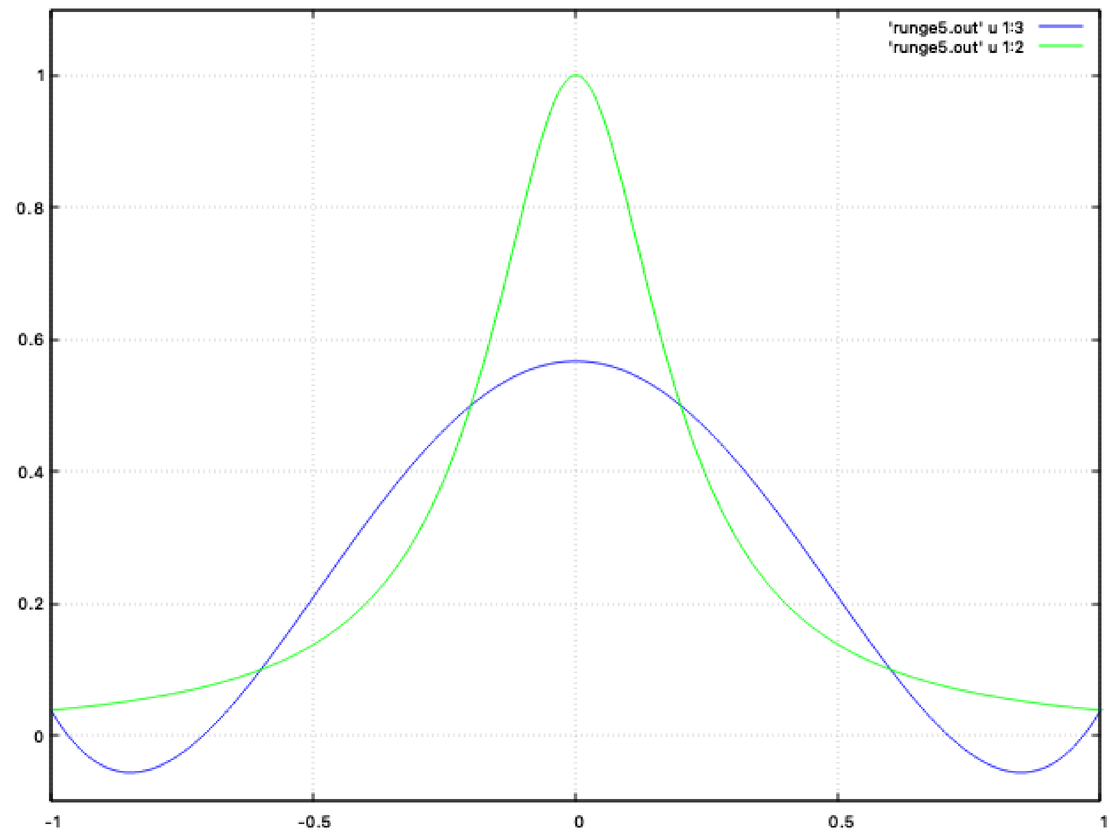


Figure 4.16: Representació del polinomi $p_5(x)$

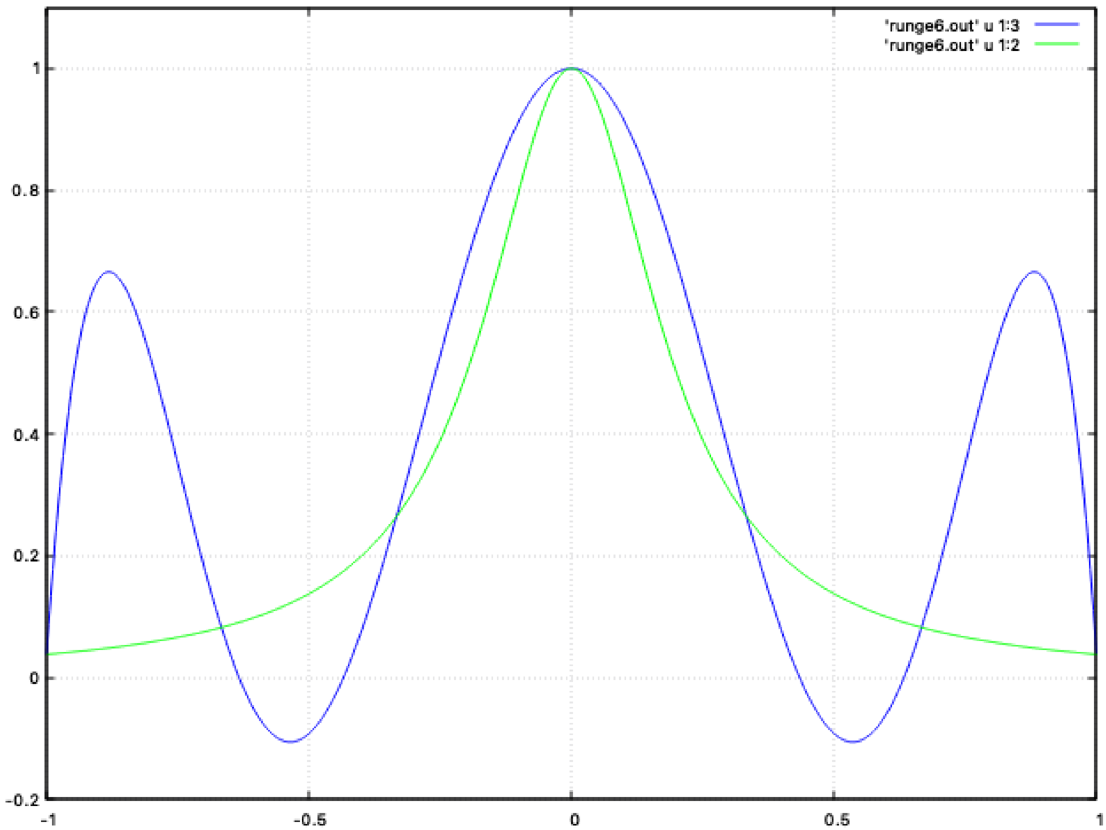


Figure 4.17: Representació del polinomi $p_6(x)$

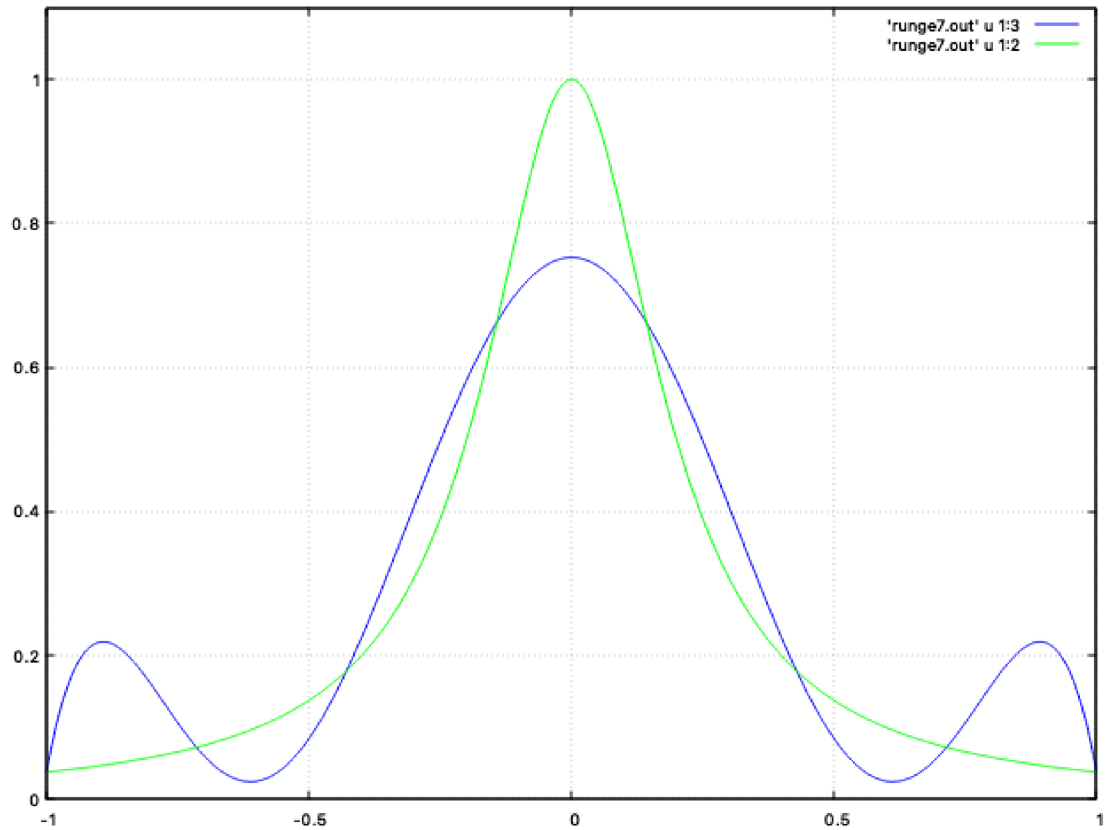


Figure 4.18: Representació del polinomi $p_7(x)$

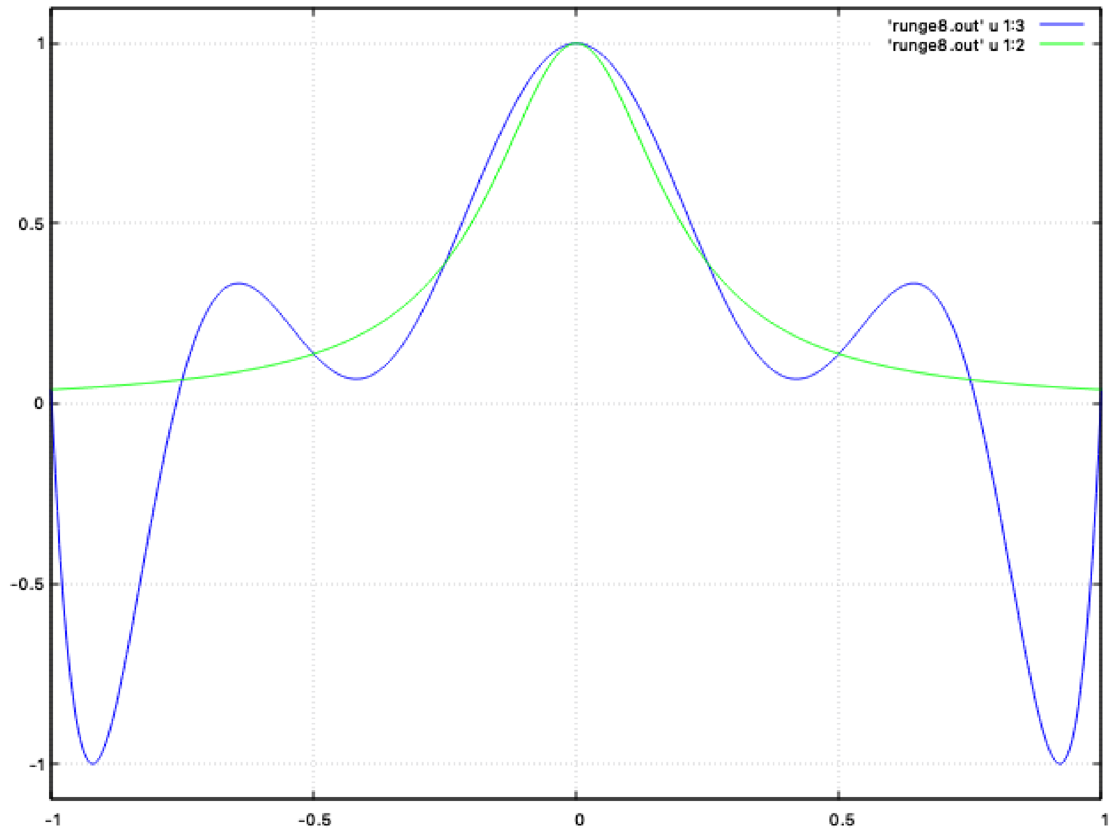


Figure 4.19: Representació del polinomi $p_8(x)$

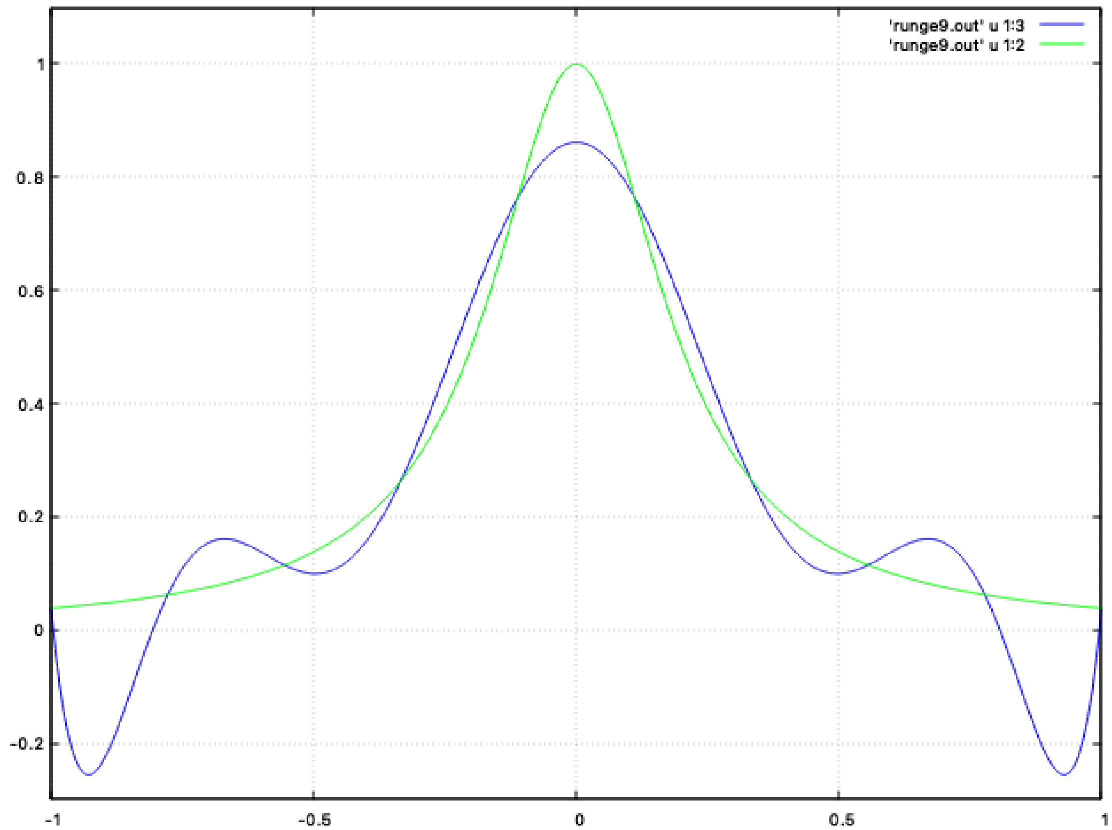


Figure 4.20: Representació del polinomi $p_9(x)$

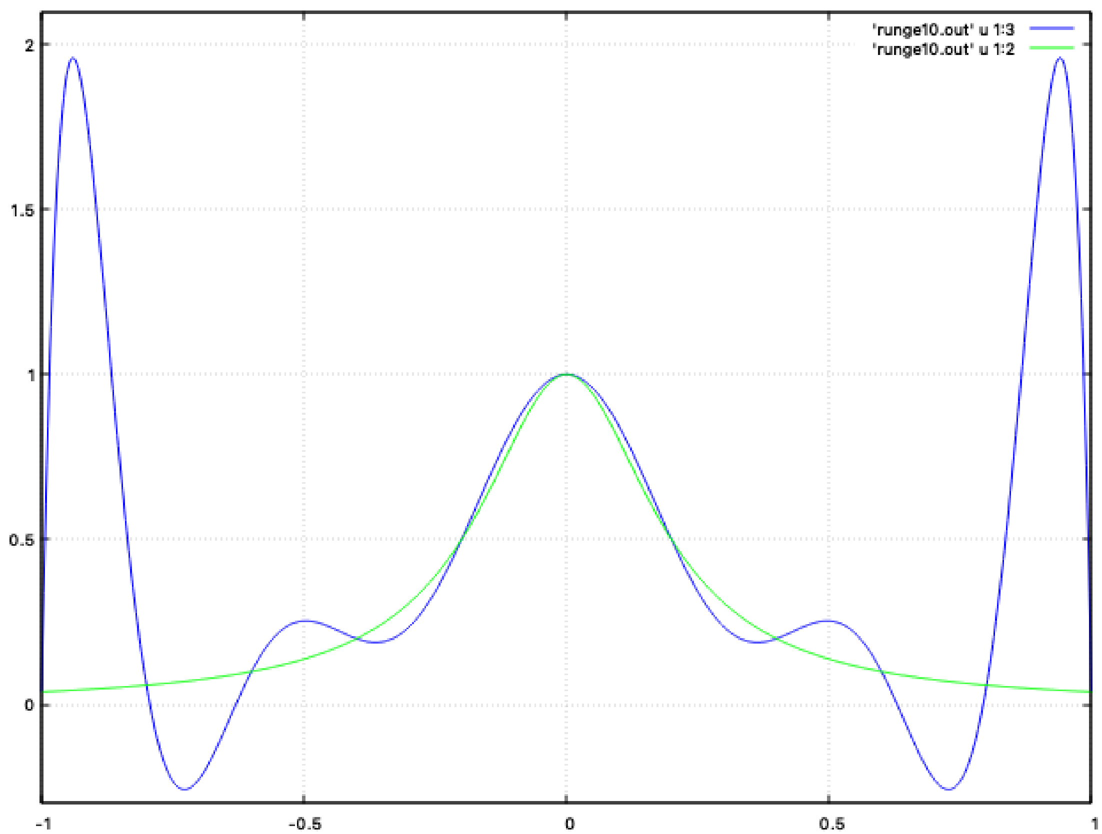


Figure 4.21: Representació del polinomi $p_{10}(x)$

Adjunto també taula amb el màxim de $|f(z_j) - p_n(z_j)|$ per a cada polinomi interpolador:

n	|f(z_j) - p_n(z_j)|
1	$9.615134e-01$
2	$6.462285e-01$
3	$7.069888e-01$
4	$4.383498e-01$
5	$4.326690e-01$
6	$6.169260e-01$
7	$2.473382e-01$
8	$1.045171e+00$
9	$3.002845e-01$
10	$1.915633e+00$

Això en una gràfica resulta en:

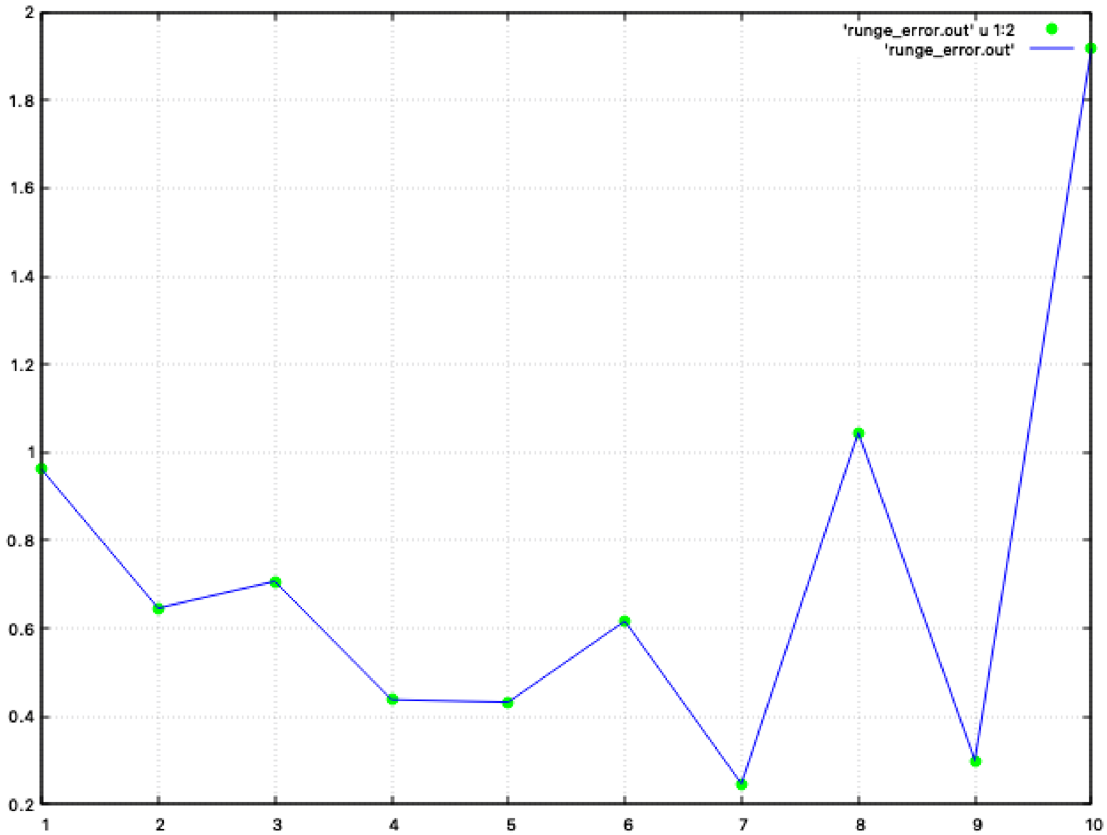
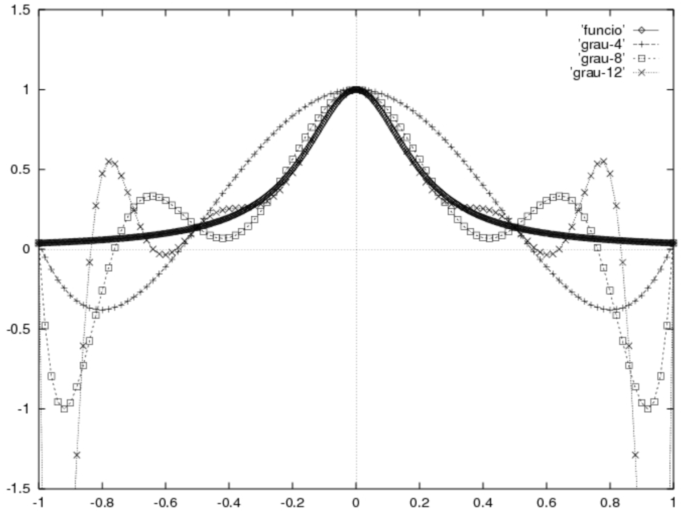
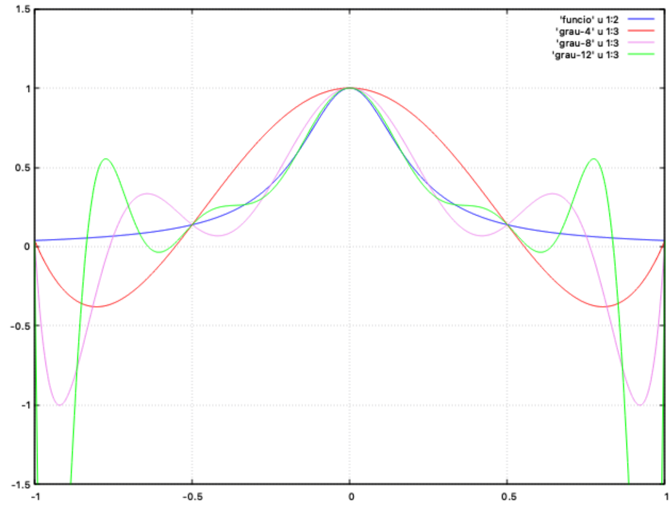


Figure 4.22: Gràfica del error segons la n

He volgut comparar la gràfica de la pàgina 19 de teoria amb el resultat del meu programa per comprovar que funcionava correctament. A continuació adjunto la gràfica de teoria i la obtinguda amb el programa realitzat:



[Teoria]



[Programa]

Figure 4.23: Comparació dels resultats

No he posat els mateixos tipus de punts a la representació, ja que a l'haver utilitzat 1000 nodes equiespaiats, no s'apreciava correctament.

Q: Llegiu les pàgines 18 i 19 de les slides de teoria i comenteu els resultats que obteniu en (4.2). És bona idea considerar molts nodes d'interpolació en un interval per tenir una millor aproximació?

> **A:** Com hem pogut comprovar amb les representacions anteriors i els comentaris sobre l'error d'interpolació generat per a cada $p_n(x)$, es compleix el que s'esmenta a les diapositives de teoria. L'error prop de l'origen és petit, però prop de -1 i 1 augmenta amb n.
>
> No és bona idea considerar molts nodes d'interpolació en un interval amb la intenció d'obtenir una bona aproximació.

Considerem la següent funció:

$$f(x) = \frac{1}{1+25x^2} \tag{4.12}$$

Carl Runge va trobar que si aquesta $f(x)$ s'interpola en punts equidistants x_i entre -1 i 1 tal que:

$$x_i = \frac{2i}{n} - 1, \quad i \in \{0, 1, ..., n\} \tag{4.13}$$

amb un polinomi interpolador $p_n(x)$ de grau $\leq n$, la interpolació resultant oscil·la cap al final de l'interval, és a dir, prop de -1 i 1. Fins i tot es pot demostrar que l'error d'interpolació augmenta (sense límit) quan augmenta el grau del polinomi:

$$\lim_{n\to\infty} \left[\max_{-1\leq x\leq 1} |f(x) - p_n(x)| \right] = +\infty \tag{4.14}$$

Això demostra que augmentar el nombre de nodes equidistants (i per tant el grau de p_n) pot ser problemàtic.

Això és degut a que l'error entre la funció $f(x)$ i el polinomi d'interpolació $p_n(x)$ d'ordre n ve donat per:

$$|f(x) - p_n(x)| = \frac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{i=1}^{n+1} (x - x_i) \tag{4.15}$$

per un $\xi \in \{-1, 1\}$. Per tant,

$$\max_{-1\leq x\leq 1} |f(x) - p_n(x)| \leq \max_{-1\leq x\leq 1} \frac{|f^{(n+1)}(x)|}{(n+1)!} \max_{-1\leq x\leq 1} \prod_{i=0}^{n} |x - x_i|. \tag{4.16}$$

Pel cas de la funció de Runge, interpolada en punts equidistants, cadascun dels dos multiplicadors en el límit superior de l'error d'aproximació creix fins l'infinit amb n. Tot i que s'acostuma a utilitzar per explicar el fenòmen de Runge, el fet de que el límit superior de l'error s'apropi a l'infinit, no implica necessariament que l'error en si mateix també divergeixi amb n.