НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

**Лабораторна робота № 16.**

**Кусково-поліноміальна інтерполяція. Кубічні сплайни**

**З курсу Обчислювальна математика**

Вариант № 21

Виконтов студент

групи ДП-82

Мнацаканов Антон

Факультет Електроніки

Викладач: Домбругов М.Р.

Київ-2020

**Мета роботи:** застосування алгоритмів кусково-поліноміальної інтерполяції поліномами 1-го та 3-го степеня для побудови наближення функції.

**Що зробити:** побудувати поліноміальне наближення до функції f (x) за допомогою кусково-лінійної інтерполяції з вузлами, що розташовані на кривій f (x). Дослідити величину дефекту наближення в залежності від числа вузлів. Додатково – побудувати і дослідити сплайн-інтерполянт.

Хід роботи

Основний на С:

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

**double** a=-1.0;

**double** b=3.0;

**double** func(**double** x){

**return** x\*x\*10.0\*exp(-x)-x\*3.0;

// return sin(x);

}

**double** piecewise\_linear\_func(**int** n, **double**\*\* nodes, **double** x){

**for**(**int** i=1; i<=n; ++i){

**if**(x >= nodes[i-1][0] && x < nodes[i][0]){

**double** t = (x - nodes[i-1][0])/(nodes[i][0]-nodes[i-1][0]);

**return** (nodes[i-1][1]\*(1.0-t))+(nodes[i][1]\*t);

}

}

**return** -1.0;

}

**double**\*\* calc\_nodes(**int** nn){

**double**\*\* nodes = malloc((nn+1)\***sizeof**(**double**\*));

**for**(**int** i=0;i<=nn; ++i){

nodes[i] = malloc(3\***sizeof**(**double**));

nodes[i][0] = a + ((b-a)/(nn))\*i;

nodes[i][1] = func(nodes[i][0]);

nodes[i][2] = 0;

}

**return** nodes;

}

**void** calc\_spline\_u(**int** n, **double**\*\* nodes){

**double**\* aa = malloc(n\***sizeof**(**double**));

**double**\* bb = malloc(n\***sizeof**(**double**));

**double** h1 = nodes[1][0] - nodes[0][0];

**double** h2 = nodes[2][0] - nodes[1][0];

**double** d1 = (nodes[1][1] - nodes[0][1])/h1;

**double** d2 = (nodes[2][1] - nodes[1][1])/h2;

aa[1] = (h1+h2)\*2;

bb[1] = (d2-d1)\*6;

**for**(**int** i=2; i<=n-1; ++i){

h1 = nodes[i][0] - nodes[i-1][0];

h2 = nodes[i+1][0] - nodes[i][0];

d1 = (nodes[i][1] - nodes[i-1][1])/h1;

d2 = (nodes[i+1][1] - nodes[i][1])/h2;

aa[i] = (h1+h2)\*2 - h1\*h1/aa[i-1];

bb[i] = (d2-d1)\*6 - h1\*bb[i-1]/aa[i-1];

}

nodes[n][2] = 0;

nodes[n-1][2] = bb[n-1]/aa[n-1];

**for**(**int** i=n-2; i>0; --i){

h2 = nodes[i+1][0] - nodes[i][0];

nodes[i][2] = (bb[i]-h2\*nodes[i+1][2])/aa[i];

}

nodes[0][2] = 0;

}

**double** spline\_func(**int** n, **double**\*\* nodes, **double** x){

**for**(**int** i=1; i<=n; ++i){

**if**(x >= nodes[i-1][0] && x < nodes[i][0]){

**double** h = nodes[i][0]-nodes[i-1][0];

**double** t = (x - nodes[i-1][0])/h;

**double** nt = 1.0 - t;

**return** (nodes[i-1][1]\*nt+nodes[i][1]\*t) - t\*nt\*h\*h/6\*(nodes[i-1][2]\*(nt+1.0)+nodes[i][2]\*(t+1.0));

}

}

**return** -1.0;

}

**double** spline1\_func(**int** n, **double**\*\* nodes, **double** x){

**for**(**int** i=1; i<=n; ++i){

**if**(x >= nodes[i-1][0] && x < nodes[i][0]){

**double** h = nodes[i][0]-nodes[i-1][0];

**double** t = (x - nodes[i-1][0])/h;

**double** nt = 1.0 - t;

**return** (nodes[i][1]-nodes[i-1][1])/h - h/6\*(nodes[i-1][2]\*(3.0\*nt\*nt-1.0)-nodes[i][2]\*(3.0\*t\*t -1));

}

}

**return** 0.0;

}

**double** spline2\_func(**int** n, **double**\*\* nodes, **double** x){

**for**(**int** i=1; i<=n; ++i){

**if**(x >= nodes[i-1][0] && x < nodes[i][0]){

**double** h = nodes[i][0]-nodes[i-1][0];

**double** t = (x - nodes[i-1][0])/h;

**return** nodes[i-1][2]\*(1.0-t)+nodes[i][2]\*(t);

}

}

**return** 0.0;

}

**double** spline3\_func(**int** n, **double**\*\* nodes, **double** x){

**for**(**int** i=1; i<=n; ++i){

**if**(x >= nodes[i-1][0] && x < nodes[i][0]){

**double** h = nodes[i][0]-nodes[i-1][0];

**double** t = (x - nodes[i-1][0])/h;

**return** (nodes[i-1][2]-nodes[i][2])/h;

}

}

**return** -1.0;

}

**int** main(){

**double**\*\* nodes4 = calc\_nodes(4);

**double**\*\* nodes12 = calc\_nodes(12);

**double**\*\* nodes40 = calc\_nodes(40);

**double**\*\* nodes120 = calc\_nodes(120);

**double**\*\* nodes400 = calc\_nodes(400);

calc\_spline\_u(4, nodes4);

calc\_spline\_u(12, nodes12);

calc\_spline\_u(40, nodes40);

calc\_spline\_u(120, nodes120);

calc\_spline\_u(400, nodes400);

**double** d\_pl4 = 0;

**double** d\_pl12 = 0;

**double** d\_pl40 = 0;

**double** d\_pl120 = 0;

**double** d\_pl400 = 0;

**double** d\_s4 = 0;

**double** d\_s12 = 0;

**double** d\_s40 = 0;

**double** d\_s120 = 0;

**double** d\_s400 = 0;

FILE\* out = fopen("data.dat", "w");

**for**(**double** s=a; s<=b; s+=0.001){

**double** f = func(s);

**double** pl4 = piecewise\_linear\_func(4, nodes4, s);

**double** pl12 = piecewise\_linear\_func(12, nodes12, s);

**double** pl40 = piecewise\_linear\_func(40, nodes40, s);

**double** pl120 = piecewise\_linear\_func(120, nodes120, s);

**double** pl400 = piecewise\_linear\_func(400, nodes400, s);

**double** s4 = spline\_func(4, nodes4, s);

**double** s12 = spline\_func(12, nodes12, s);

**double** s40 = spline\_func(40, nodes40, s);

**double** s120 = spline\_func(120, nodes120, s);

**double** s400 = spline\_func(400, nodes400, s);

**if**(fabs(f - pl4) > d\_pl4){

d\_pl4 = fabs(f - pl4);

}

**if**(fabs(f - pl12) > d\_pl12){

d\_pl12 = fabs(f - pl12);

}

**if**(fabs(f - pl40) > d\_pl40){

d\_pl40 = fabs(f - pl40);

}

**if**(fabs(f - pl120) > d\_pl120){

d\_pl120 = fabs(f - pl120);

}

**if**(fabs(f - pl400) > d\_pl400){

d\_pl400 = fabs(f - pl400);

}

**if**(fabs(f - s4) > d\_s4){

d\_s4 = fabs(f - s4);

}

**if**(fabs(f - s12) > d\_s12){

d\_s12 = fabs(f - s12);

}

**if**(fabs(f - s40) > d\_s40){

d\_s40 = fabs(f - s40);

}

**if**(fabs(f - s120) > d\_s120){

d\_s120 = fabs(f - s120);

}

**if**(fabs(f - s400) > d\_s400){

d\_s400 = fabs(f - s400);

}

fprintf(out, "%e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e %e\n", s, f, pl4, pl12, pl40, pl120, pl400, f-pl4, f-pl12, f-pl40, f-pl120, f-pl400, s4, s12, s40, s120, s400, f-s4, f-s12, f-s40, f-s120, f-s400);

}

fclose(out);

printf("n\t\tpiecewise\_linear\t\tspline\n");

printf("4\t\t%e\t\t%e\n", d\_pl4, d\_s4);

printf("12\t\t%e\t\t%e\n", d\_pl12, d\_s12);

printf("40\t\t%e\t\t%e\n", d\_pl40, d\_s40);

printf("120\t\t%e\t\t%e\n", d\_pl120, d\_s120);

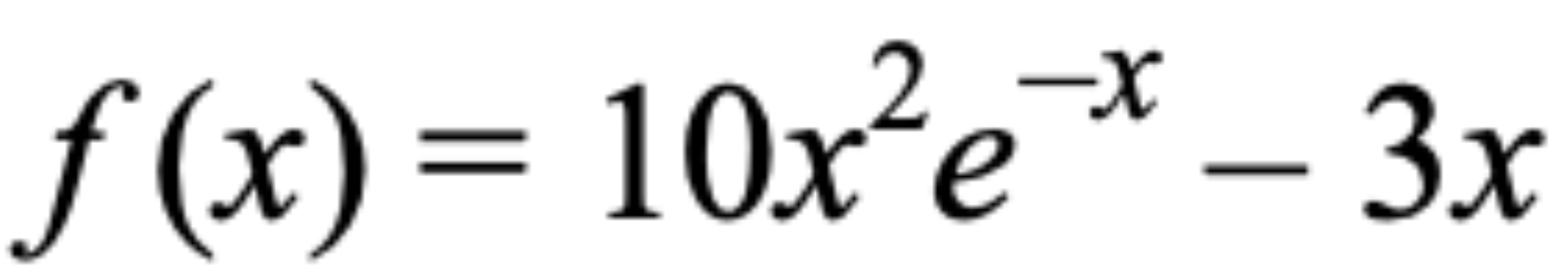
printf("400\t\t%e\t\t%e\n", d\_pl400, d\_s400);

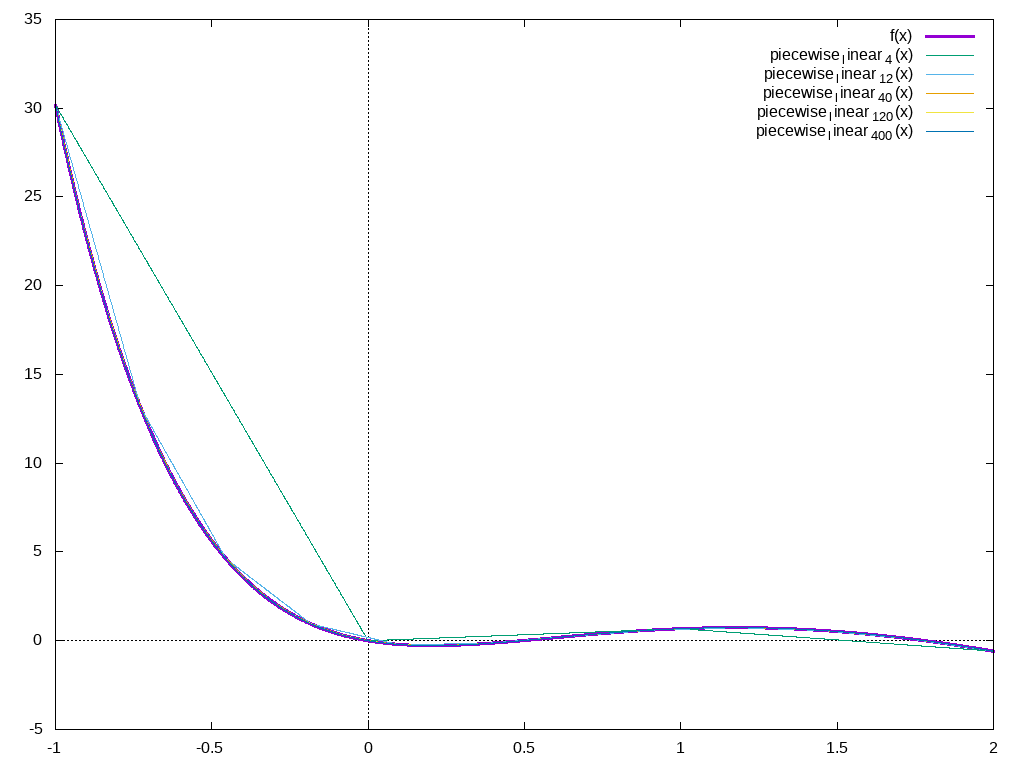
**return** 0;

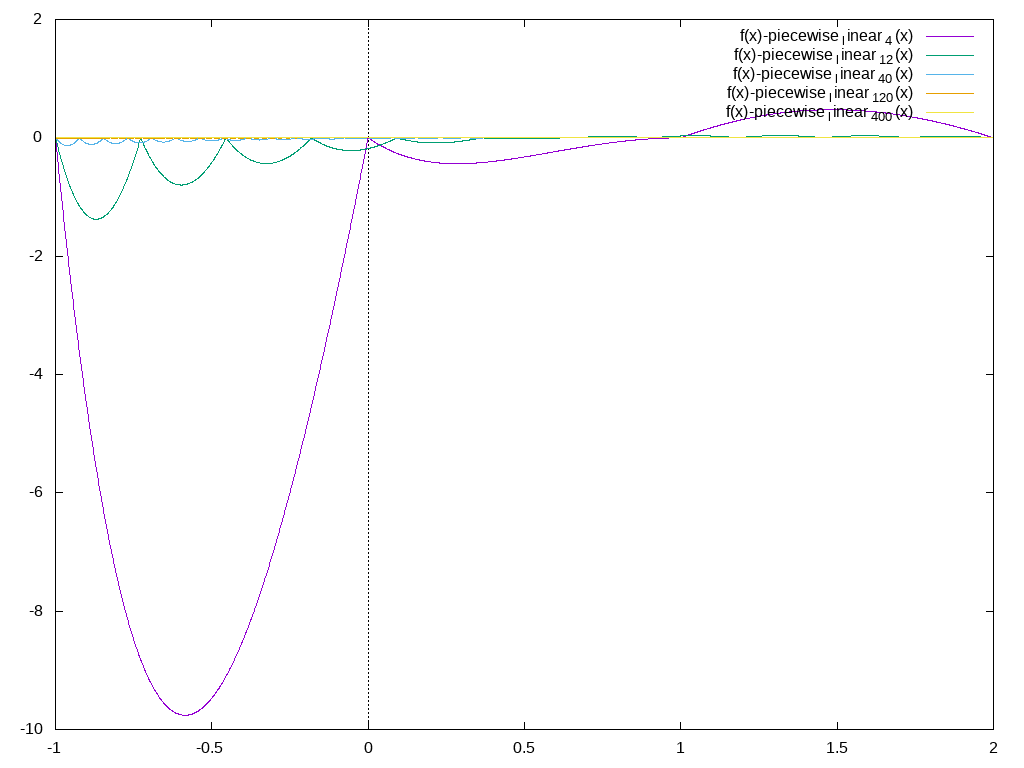
}

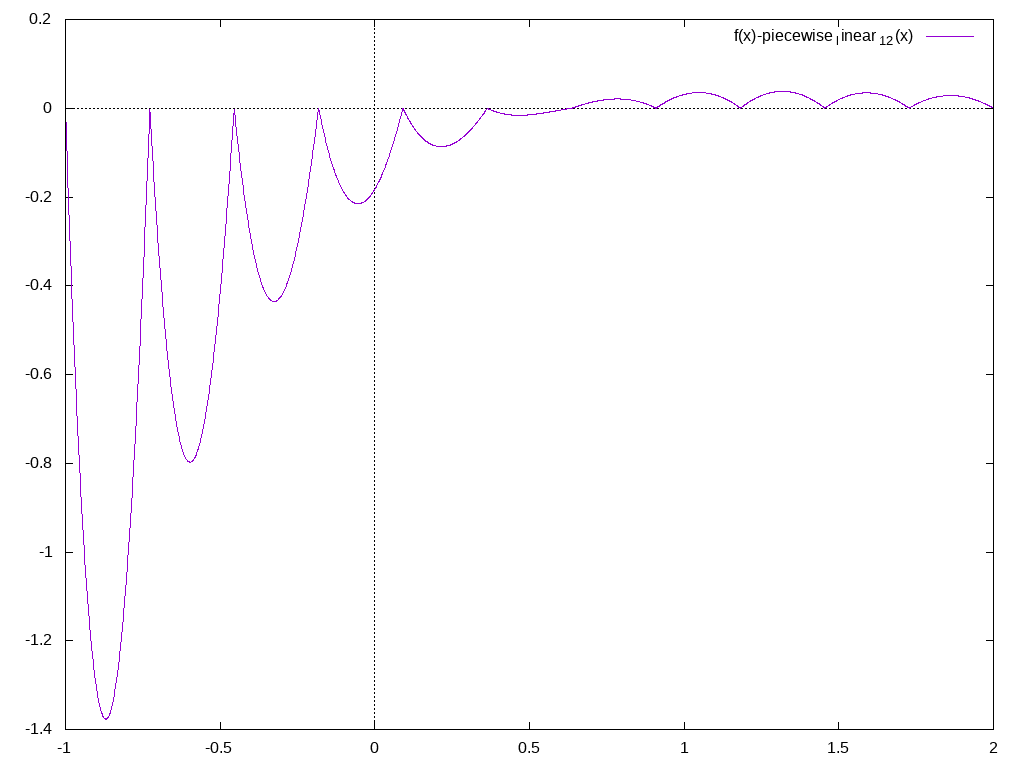
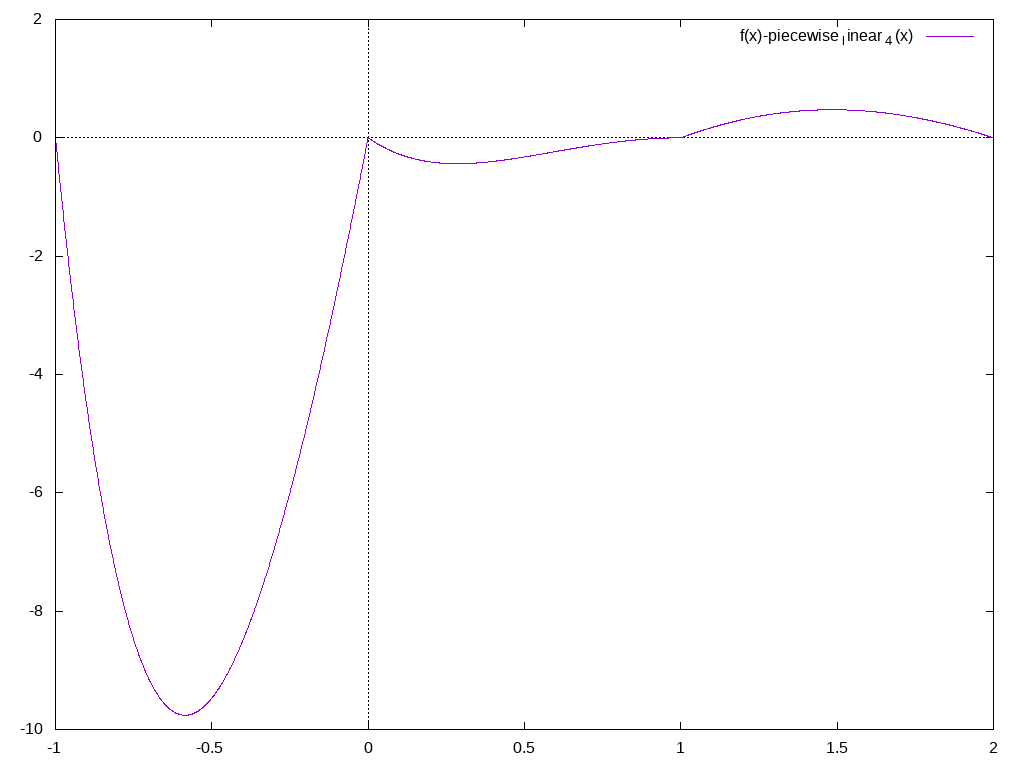
Результати

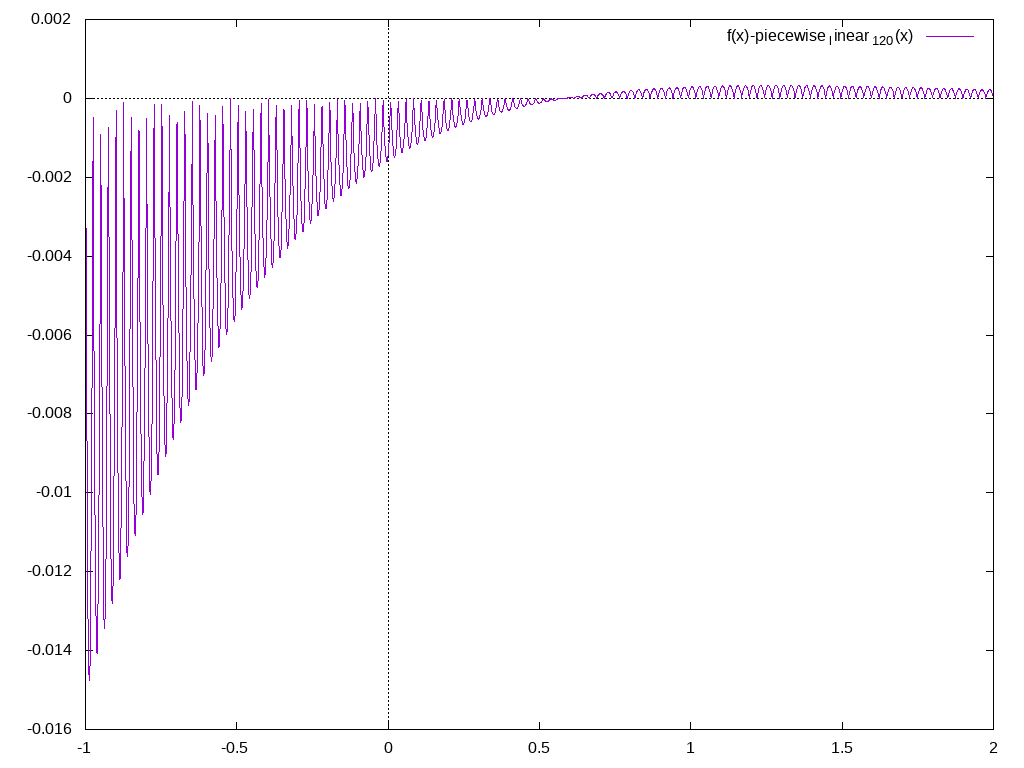
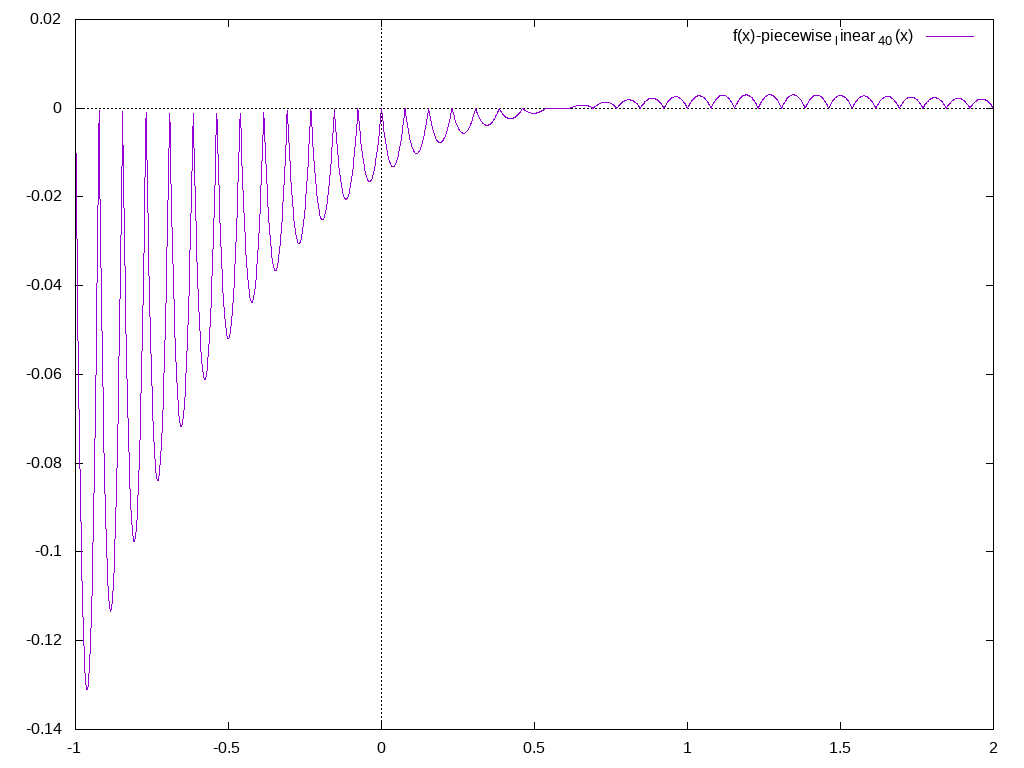
Це моя головна функція

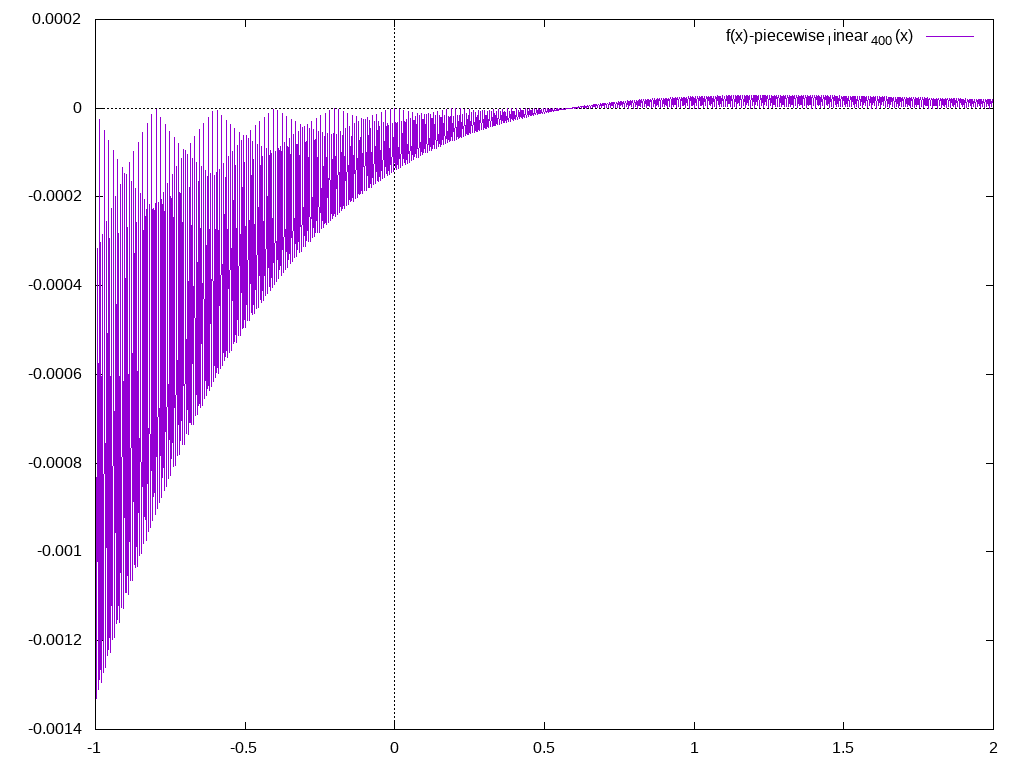


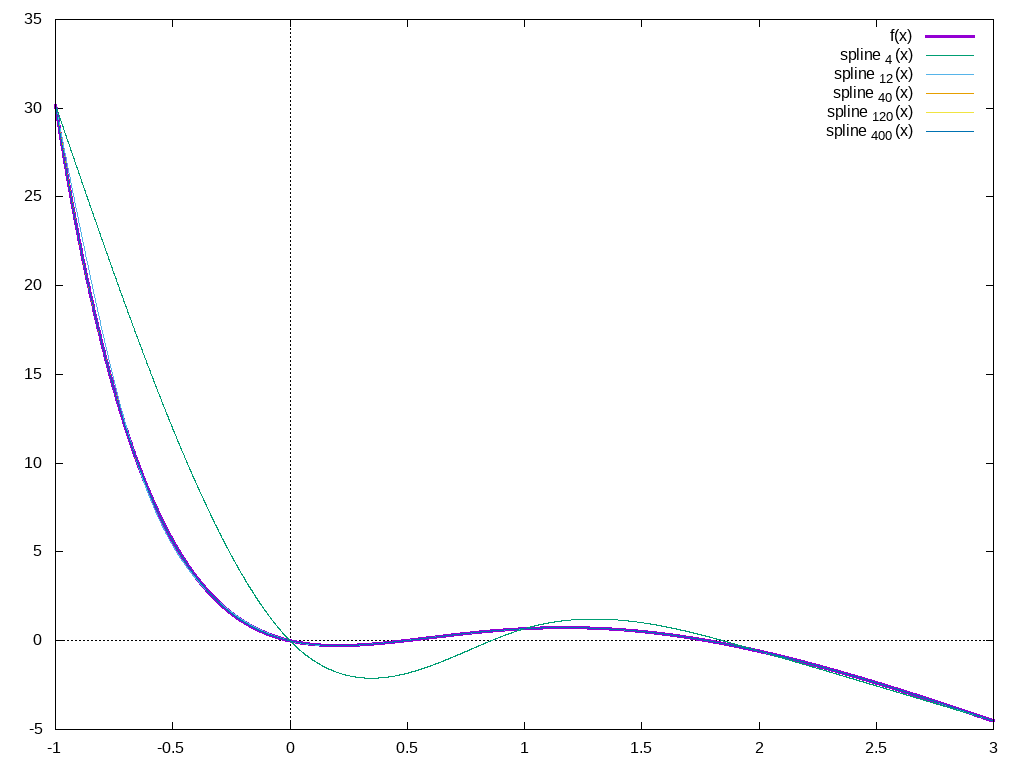
кусково-лінійного наближення

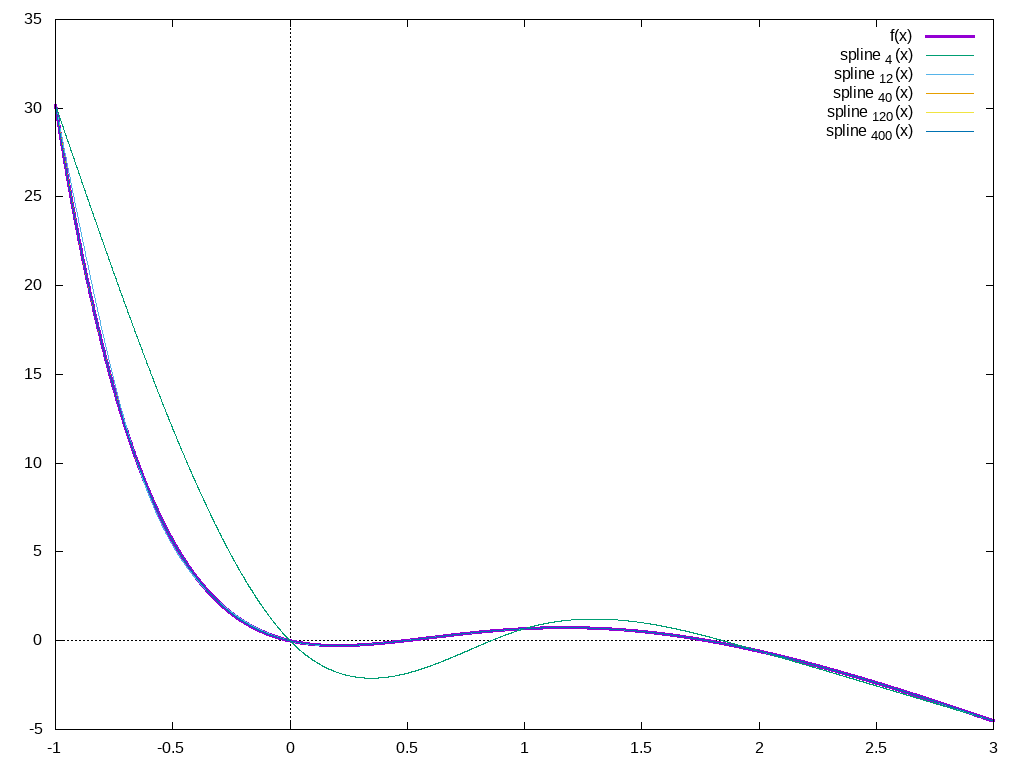
Різниця для 4,12,40,120,400 вузлів

Окремо для 4,12,40,120 та 400 вузлів

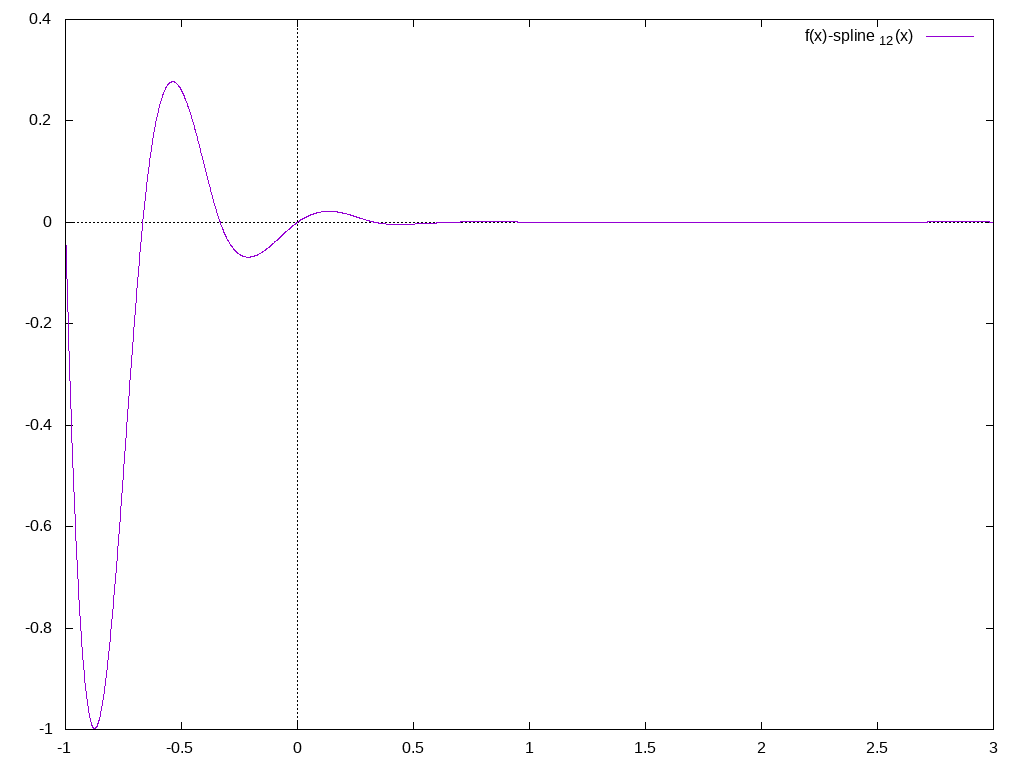


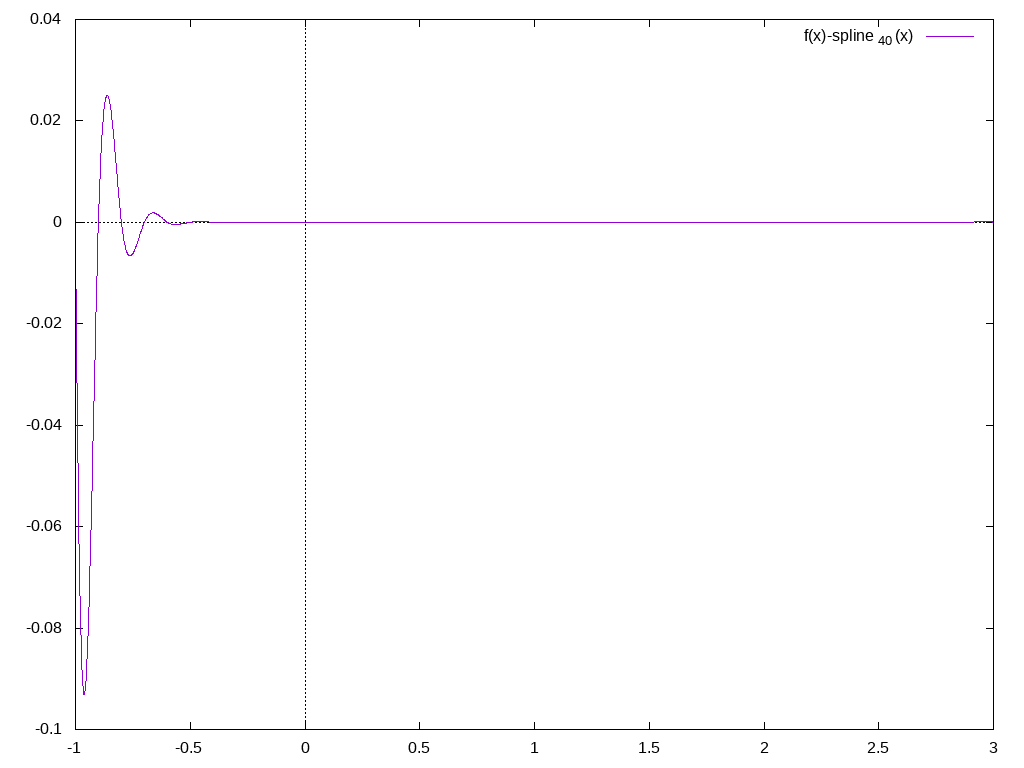
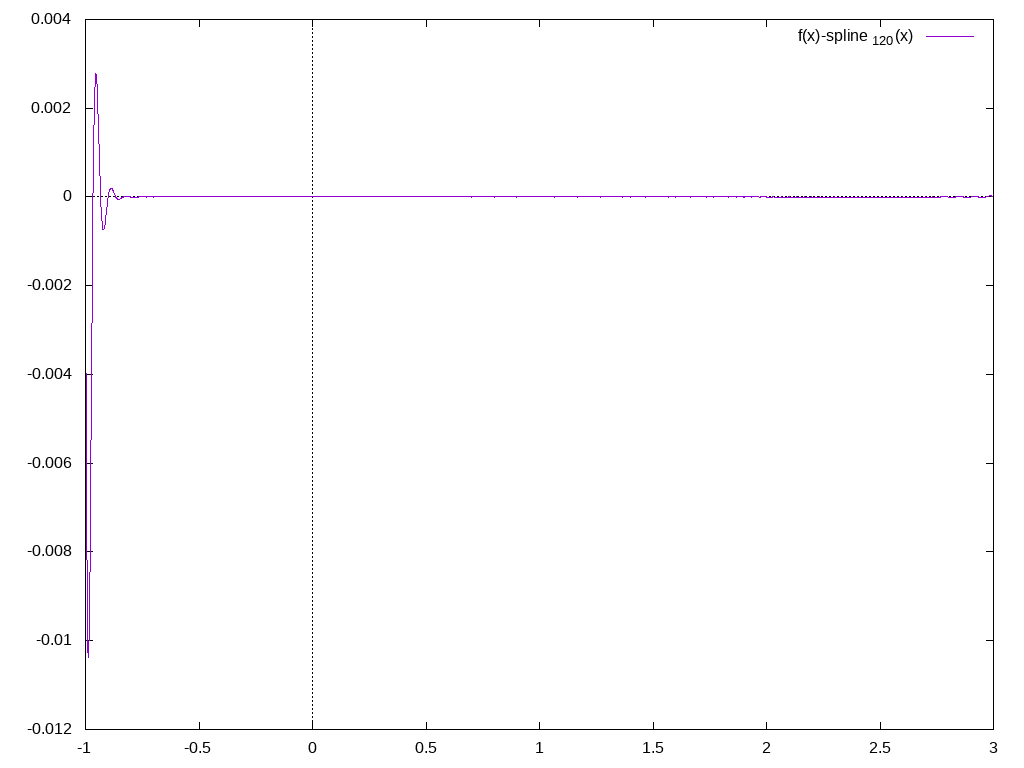


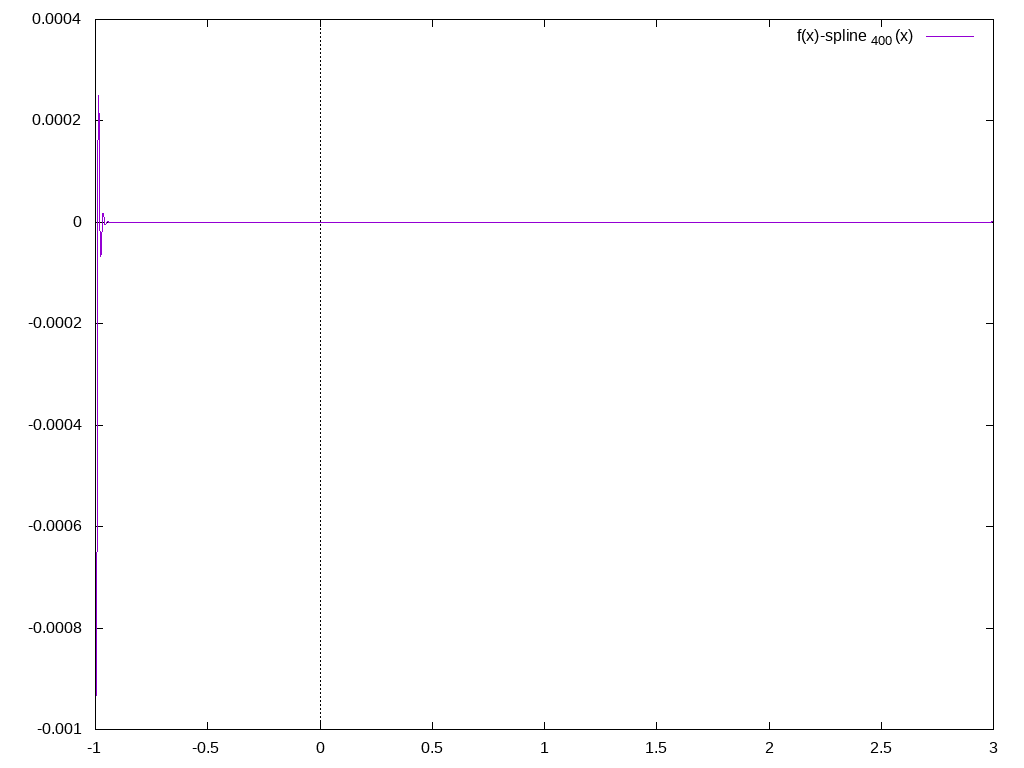
Сплайн-інтерполяція

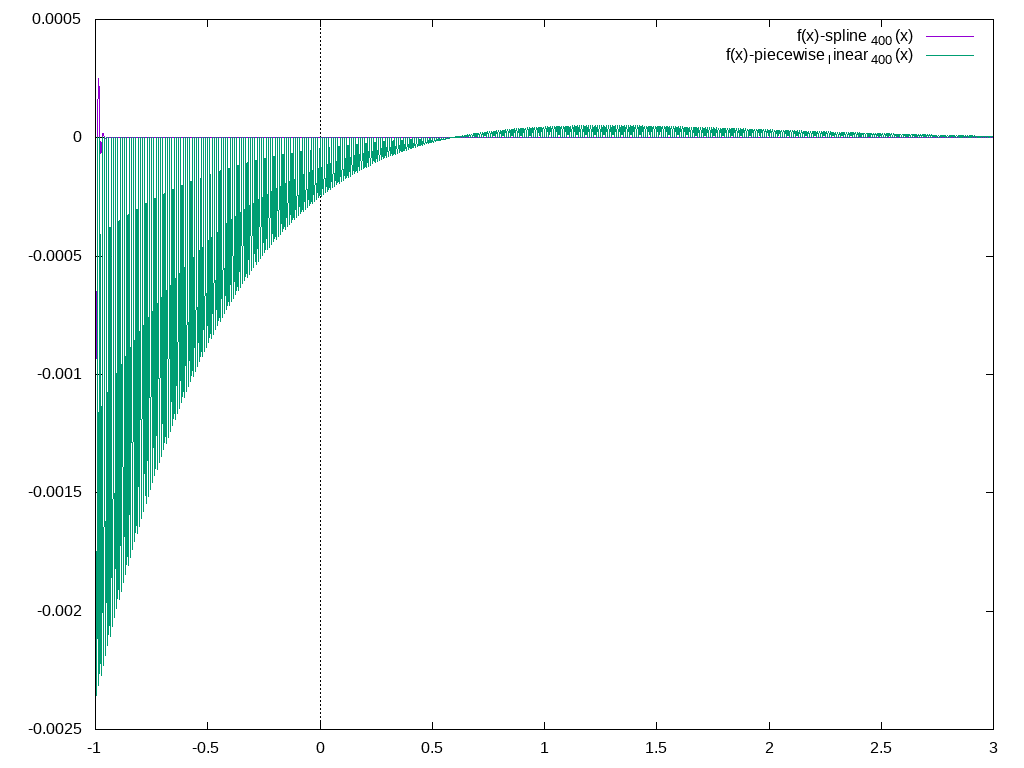
Різниця для 4,12,40,120 та 400 вузлів

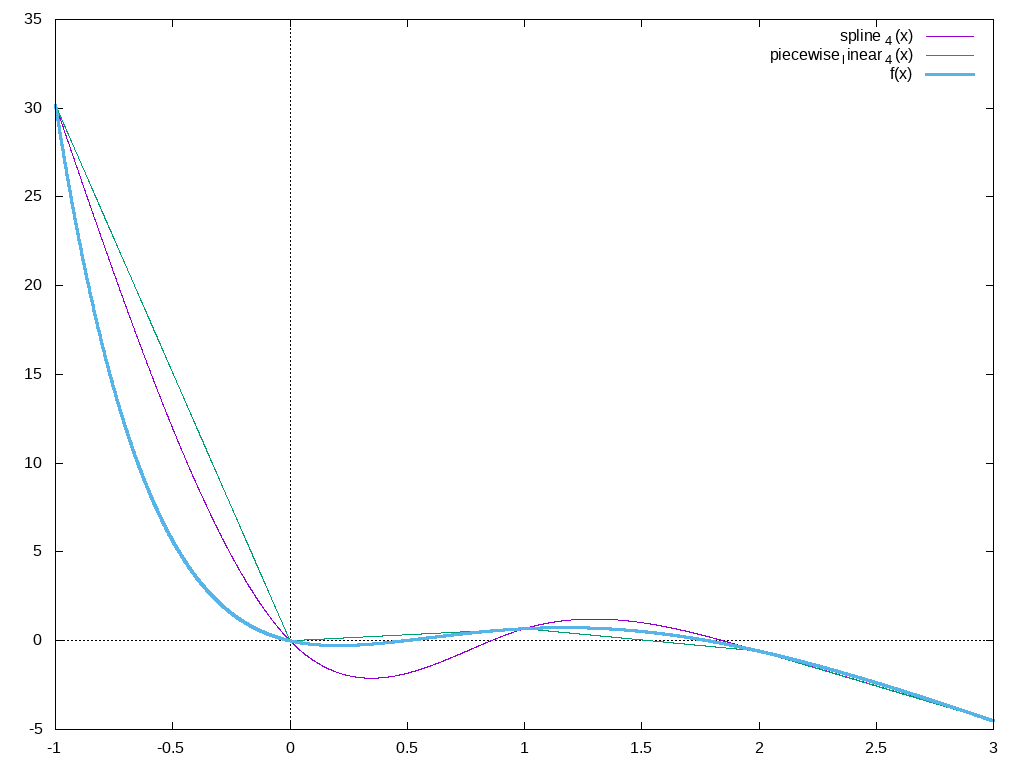
для кожної окремо







Доп граф.



**Висновок:**

Я застосовуючи алгоритми кусково-поліноміальної інтерполяції поліномами

1-го та 3-го степеня побудував наближення функції та вивіві їх на екран за допомогою программи Gnuplot, також склав окрему процедуру SPLINE для розрахунку масиву коефіцієнтів другої похідної {ui} для сплайн-інтерполяції «природним» сплайном, виконав пп. 3–5 для сплайн-інтерполяції, розраховуючи замість інтерполяційної ламаної p(x) сплайн-інтерполянт s(x).