Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ **ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ**

КАФЕДРА **КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.01 Информатика и вычислительная техника**

О Т Ч Е Т

по домашнему заданию № 2

Дисциплина: Вычислительная математика

Студент ИУ6-62Б ______ И.С. Марчук
(Группа) (Подпись, дата) (И.О. Фамилия)

Преподаватель ______ ______
(Подпись, дата) (И.О. Фамилия)

Москва, 2022

Введение

Цель работы:

- изучение методов решения нелинейного уравнения f(x) = 0, сравнение скорости их работы и точности;
- изучение методов построения интерполяционной формулы Лагранжа и интерполяции кубическими сплайнами.

Часть 1

Задание

– Реализовать методы бисекции, хорд, простой итерации и Ньютона;

– отладить алгоритмы на тестовых примерах, решив уравнения $2^{x-0.1} - 1 = 0$, $x \in [0,1]$ и $(x - 0.2)^3 = 0$, $x \in [0,1]$;

– в программе предусмотреть возможность вывода результатов в виде таблицы.

Примечание: установим eps = 0.0001

Ход работы

Краткое описание метода бисекции

Метод бисекции один из методов решения нелинейных уравнений и основан на последовательном сужении интервала (за счет деления интервала пополам), содержащего единственный корень уравнения F(x)=0 до того времени, пока не будет достигнута заданная точность ε.

Краткое описание метода хорд

Этот итерационный метод, подобно описанному выше методу, заключается в повторяющемся делении интервала на две части с выбором из них той, которая содержит корень уравнения. Однако в методе хорд точка, с помощью которой исходный отрезок [a, b] делится на две части, выбирается не как средняя, а вычисляется с помощью линейной интерполяции функции f(x) на [a, b].

Краткое описание метода простых итераций

Рассмотрим уравнение:

$$f(x)=0$$

с отделенным корнем $X \in [a, b]$. Для решения уравнения методом простой итерации приведем его к равносильному виду:

$$x=\varphi(x)$$

Это всегда можно сделать, причем многими способами. Например: $x=g(x)*f(x) + x \equiv \varphi(x)$, где g(x) - произвольная непрерывная функция, не имеющая корней на отрезке [a,b]. Пусть x(0) – полученное каким-либо способом приближение к корню x (в простейшем случае $x^{(0)}$ = (a+b)/2). Метод простой итерации заключается в последовательном вычислении членов итерационной последовательности: $x^{(k+1)} = \varphi(x^{(k)})$, k=0, 1, 2, ...начиная с приближения $x^{(0)}$.

Краткое описание метода Ньютона

Основная идея метода Ньютона — это идея линеаризации. Предположим что F(x) дифференцируемая функция и мы решаем уравнение F(x) = 0. Начав с точки x_0 мы можем построить линейную аппроксимацию F(x) в окрестности x_0: $F(x_0 + h) \approx F(x_0) + F'(x_0) * h$ и решить получающееся линейное уравнение $F(x_0) + F'(x_0) * h = 0$. Так мы приходим к итеративному методу:

$$x_{(k+1)} = x_k - F'(x_k)^{-1} * F(x_k),$$
$$k = 0, 1, \dots$$

Исходный код программы на языке Octave:

```
pkg load symbolic
pkg load tablicious

warning('off','all');

format shortG

% ==============================================

% Вывод ошибки, если значения на концах отрезка не принимают разные знаки
function cond(f, low_f, high_f)
	if f(low_f) * f(high_f) >= 0
		error('Значения на концах отрезка не противоположных знаков')
	end
end

% ==============================================

function [middle, iter, obr] = bisection(f, low, high, tol)
	iter = 0; %Число итераций
	obr = 0; %Число обращений к f(x)
	%Изменяем high и low, пока не достигнем максимального приближения к нулю
	while (abs(high - low) >= 2*tol)
		middle = (high + low)/2; %Новое значение в качестве корня
		y3 = f(middle);
		%Сужаем границы
		if f(low) * y3 > 0
			low = middle;
		else
			high = middle;
		end
		iter = iter + 1;
```

```
            obr = obr + 2;
        end
end

% ==============================================

function [bn, iter, obr] = horda(f, low, high, tol)
        %вторая производная
        syms x
        double_diff(x) = diff(diff(f(x),'x'),'x');
        %Выбор подвижного и неподвижного конца
        if f(low)*double_diff(low)>0
            b = high;
            a = low;
        else
            b = low;
            a = high;
        end
            bn = b-f(b)*(a-b)/(f(a)-f(b)); %1-я итерация
            iter = 1; %Число итераций
            obr = 3; %Число обращений к f(x)
        while(abs(b-bn)>=tol)
            b=bn;
            bn=b-f(b)*(a-b)/(f(a)-f(b));
            iter = iter + 1;
            obr = obr + 3;
        end
end

% ==============================================

function [xm, iiter, obr] = iter(f, low, high, tol)
        %Берём производную
        syms x
        func(x) = diff(f(x),'x');
        %Поиск максимума производной функции
        M = fminbnd(@(x) - double(subs(func(x),x)), low, high);
        max = f(M);
        %Поиск минимума производной функции
        m = fminbnd(@(x) double(subs(func(x),x)), low, high);
        min = f(m);

        t=2/(min+max);
        q=(max-min)/(max+min);

        fi=@(x) x-t*f(x);
        xn = low;
        xm = fi(low);
        iiter = 1; %Число итераций
        obr = 6; %Число обращений к f(x)
```

```
	while(abs(xn-xm)>=tol)
		xn = xm;
		xm = fi(xn);
		iiter = iiter + 1;
		obr = obr + 1;
	end
end

% ==============================================

function [xm, iter, obr] = newton(f, low, high, tol)
	%Берём производную
	syms x
	func(x) = diff(f(x),'x');
	xn = low;
	xm = xn - f(xn)/double(subs(func(xn),xn));
	iter = 1; %Число итераций
	obr = 3; %Число обращений к f(x)

	while abs(xn-xm)>=tol
		xn = xm;
		xm = xn - f(xn)/double(subs(func(xn),xn));
		iter = iter + 1;
		obr = obr + 2;
	end
end

% ==============================================

% Условия:
f1 = @(x) (x-0.2)^3; % x* = 0.2
a1 = 0; b1 = 1;
eps1 = 0.0001;

f2 = @(x) 2.^(x-0.1)-1; % x* = 0.1
a2 = 0; b2 = 1;
eps2= 0.0001;

%Проверка знака на концах отрезка
cond(f1, a1, b1);
cond(f2, a2, b2);
%Метод бисекции:
[bis1_appr, bis1_it, bis1_calls] = bisection(f1, a1,b1, eps1);
%A[]=bisection(f1, a1,b1, eps);
[bis2_appr, bis2_it, bis2_calls] = bisection(f2, a2,b2, eps2);
%Метод хорд
[hord1_appr, hord1_it, hord1_calls] = horda(f1, a1,b1, eps1);
[hord2_appr, hord2_it, hord2_calls] = horda(f2, a2,b2, eps2);
%Метод простой итерации
[i1_appr, i1_it, i1_calls] = iter(f1, a1, b1,eps1);
```

```
[i2_appr, i2_it, i2_calls] = iter(f2, a2, b2,eps2);
%Метод Ньютона
[newt1_appr, newt1_it, newt1_calls] = newton (f1, a1,b1, eps1);
[newt2_appr, newt2_it, newt2_calls] = newton (f2, a2,b2, eps2);

%Таблицы:
disp('функция f1=2^(x=0.1) :');
Metod = {'Метод бисекции';'Метод хорд';'Метод простой итерации';'Метод Ньютона'};
Reshenie = [bis1_appr;hord1_appr;i1_appr;newt1_appr];
Iterazii = [bis1_it;hord1_it;i1_it;newt1_it];
Vizovi = [bis1_calls;hord1_calls;i1_calls;newt1_calls];
T1 = ...
table(
        Reshenie,
        Iterazii,
        Vizovi,
        Metod
        );
prettyprint(T1);

disp('функция f2=(x-0.2)^3 :');
Metod = {'Метод бисекции';'Метод хорд';'Метод простой итерации';'Метод Ньютона'};
Reshenie = [bis2_appr;hord2_appr;i2_appr;newt2_appr];
Iterazii = [bis2_it;hord2_it;i2_it;newt2_it];
Vizovi = [bis2_calls;hord2_calls;i2_calls;newt2_calls];
T2 = ...
table(
        Reshenie,
        Iterazii,
        Vizovi,
        Metod
        );

prettyprint(T2);
```

Результат работы программы представлен на рисунке 1.

```
функция f1=2^(x=0.1) :
-------------------------------------------------------
| Reshenie | Iterazii | Vizovi | Metod
-------------------------------------------------------
| 0.20007  | 13       | 26     | Метод бисекции
| 0.18381  | 87       | 261    | Метод хорд
| 0.17071  | 143      | 148    | Метод простой итерации
| 0.19986  | 18       | 37     | Метод Ньютона
-------------------------------------------------------
функция f2=(x-0.2)^3 :
-------------------------------------------------------
| Reshenie | Iterazii | Vizovi | Metod
-------------------------------------------------------
| 0.099976 | 13       | 26     | Метод бисекции
| 0.099986 | 7        | 21     | Метод хорд
| 0.099969 | 26       | 31     | Метод простой итерации
| 0.1      | 3        | 7      | Метод Ньютона
-------------------------------------------------------
```

Рисунок 1 – приближенные решения уравнений

Часть 2

Задание

– Реализовать алгоритмы построения интерполяционного полинома Лагранжа и системы кубических сплайнов;
– отладить алгоритмы на тестовых примерах, построив интерполянты для 2^x на [0, 4], $(1 + 25x^2)^{-1}$ на $x \in [-2,2]$, используя равномерную сетку;
– в программе предусмотреть возможность вывода графиков функции f и двух интерполирующих функций, построенных разным цветов в общих осях.

Краткое описание интерполяционного полинома и его построения в форме Лагранжа

Интерполяция алгебраическими многочленами функции f(x) действительного аргумента на отрезке [a,b] — нахождение коэффициентов многочлена $P_n(x)$ степени меньшей или равной n, принимающего при значениях аргумента $x_0, x_1, .., x_n$ значения $f(x_i)$.

Многочлен Лагранжа:

$$L(x) = \sum_{i=0}^{n} y_i l_i(x)$$

Базисные полиномы:

$$l_i(x) = \prod_{j=0, j\neq i}^{n} \frac{x - x_j}{x_i - x_j} = \frac{x - x_0}{x_i - x_0} \cdots \frac{x - x_{i-1}}{x_i - x_{i-1}} \cdot \frac{x - x_{i+1}}{x_i - x_{i+1}} \cdots \frac{x - x_n}{x_i - x_n}$$

Краткое описание сплайн-интерполяции и алгоритма построения кубических сплайнов

Понятие интерполяции описано выше.

Сплайн – функция, которая вместе с несколькими производными непрерывна на всем заданном отрезке [a,b], а на каждом частичном отрезке [x_i, x_{i+1}] в отдельности является некоторым алгебраическим многочленом.

Пусть кубический сплайн на каждом отрезке [x_{i-1}, x_i] задается функцией:

$$S_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3$$

Условия непрерывности всех производных до второго порядка включительно записываются в виде:

$$\begin{aligned} S_i\,(x_{i-1}) &= S_{i-1}(x_{i-1}), \\ S_i'\,(x_{i-1}) &= S_{i-1}'(x_{i-1}), \\ S_i''\,(x_{i-1}) &= S_{i-1}''(x_{i-1}), \end{aligned}$$

Исходя из этого выводятся формулы для коэффициентов сплайна:

$$a_i = f(x_i);$$

$$d_i = \frac{c_i - c_{i-1}}{3 \cdot h_i};$$

$$b_i = \frac{a_i - a_{i-1}}{h_i} + \frac{2 \cdot c_i + c_{i-1}}{3} \cdot h_i;$$

$$c_{i-1} \cdot h_i + 2 \cdot c_i \cdot (h_i + h_{i+1}) + c_{i+1} \cdot h_{i+1} = 3 \cdot \left(\frac{a_{i+1} - a_i}{h_{i+1}} - \frac{a_i - a_{i-1}}{h_i} \right),$$

причем $c_N = S''(x_N) = 0$ и $c_1 - 3 \cdot d_1 \cdot h_1 = S''(x_0) = 0$.

Если учесть, что $c_0 = c_N = 0$, то вычисление c можно провести с помощью метода прогонки для трёхдиагональной матрицы.

Исходный код программы на языке Octave:

```
warning('off','all');

function [v] = Lanrange(X, Y, t)
% вычисление полинома Лагранжа в точке
	v=0;
	n=size(X,1);
	for i=1:n
		bp=Y(i);
		for j =1:n
			if i ~= j
				bp=bp*(t-X(j))/(X(i)-X(j));
			end
		end
		v=v+bp;
	end
end

function [A,B,C,D] = GetCoeff(X,Y)
% Инициализация массива сплайнов
	N = size(X,1);
	A = Y;
	C(1) = 0; C(N) = 0;
	alfa = zeros(N);
	beta = zeros(N);
	% Решение СЛАУ относительно коэффициентов сплайнов c[i]
	% методом прогонки для трехдиагональных матриц
	alfa(1) = 0; beta(1) = 0;
	for i = 2:N-1
		h_i = X(i)-X(i-1);
		h_i1 = X(i+1)-X(i);
		wA = h_i;
		wC = 2*(h_i+h_i1);
		wB = h_i1;
		wF = 6*((Y(i+1)-Y(i))/h_i1 - (Y(i)-Y(i-1))/h_i);
		wz = wA * alfa(i-1) + wC;
		alfa(i) = -wB/wz;
		beta(i) = (wF-wA*beta(i-1))/wz;
	end
	C(N) = (wF-wA*beta(N-1))/(wC+wA*alfa(N-1));
	% Обратный ход метода прогонки
	for i = N-1:-1:2
		C(i) = alfa(i)*C(i+1)+ beta(i);
		for i = N:-1:2
			h_i = X(i)-X(i-1);
			D(i) = (C(i)-C(i-1))/h_i;
			B(i) = h_i*(2*C(i)+C(i-1))/6 + (Y(i)-Y(i-1))/h_i;
		end
	end
```

```
end

function [v] = SplineCube(X,A,B,C,D, u)
	N=size(X,1);
	if u<=X(1)
		j = 1;
	elseif u>=X(N)
		j = N;
	else
		i = 1; j = N;
	while (i+1<j)
		k = round((i+j)/2);
		if u<=X(k)
			j = k;
		else
			i = k;
		end
		end
	end
	dx = u - X(j);
	v = A(j)+(B(j)+(C(j)/2+D(j)*dx/6)*dx)*dx;
end

function [] = Draw(f,L,R,N)
	X = linspace(L,R,N)'; % сетка интерполяции
	Y = f(X); % значение функции в узлах
	% полином Лагранжа
	% построение сплайна b полинома
	[A,B,C,D] = GetCoeff(X,Y);
	K=200;
	XG = linspace(L,R,K)';
	YG = f(XG);
	for i=1:K
		YL(i) = Lanrange(X,Y,XG(i));
		YS(i) = SplineCube(X,A,B,C,D, XG(i));
	end

	% графики функции и интерполянтов
	figure
	subplot(2,1,1);
	plot(XG,YG,'-b')
	hold on
	plot(XG,YL','-r')
	plot(XG,YS','-k')
	grid on
	legend('f(s)','L(x)','S(x)','Location','NorthWest')
	str=sprintf('N=%d',N);
	title(str);
	% графики отклонений
```

```
        subplot(2,1,2);
        plot(XG,YG-YL','-b')
        hold on
        plot(XG,YG-YS','-k')
        grid on
        legend('f(s)-L(x)','f(x)-S(x)','Location','NorthWest')
        str=sprintf('N=%d',N)
end

% =============================================

% Условия:
f1 = @(x) 2.^x;
f2 = @(x) (1+25*x.^2).^(-1);

a1=0; b1=4;
a2=-2; b2=2;
%Вывод графиков:
disp('Для функции f1=2^x :');
Draw(f1, a1, b1, 5);
Draw(f1, a1, b1, 10);
Draw(f1, a1, b1, 50);
disp('Для функции f2=(1+25*x^2)^(-1) :');
Draw(f2, a2, b2, 5);
Draw(f2, a2, b2, 10);
Draw(f2, a2, b2, 50);
```

Результаты работы программы представлены на рисунках 2-7.

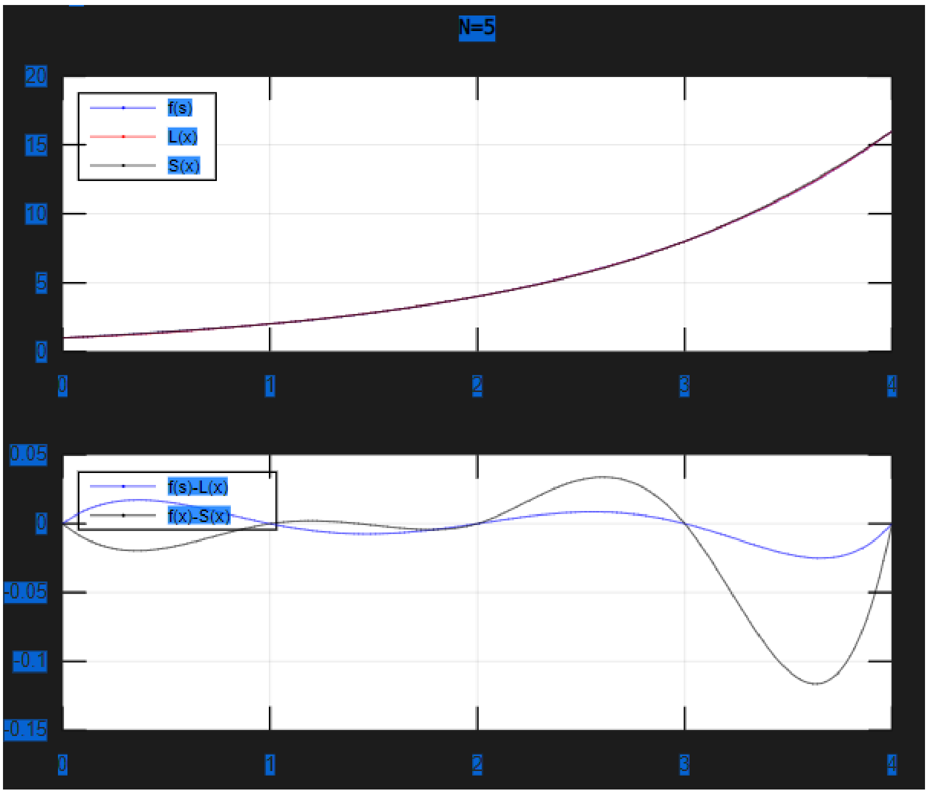

Рисунок 2 – интерполяция f_1 (N=5)

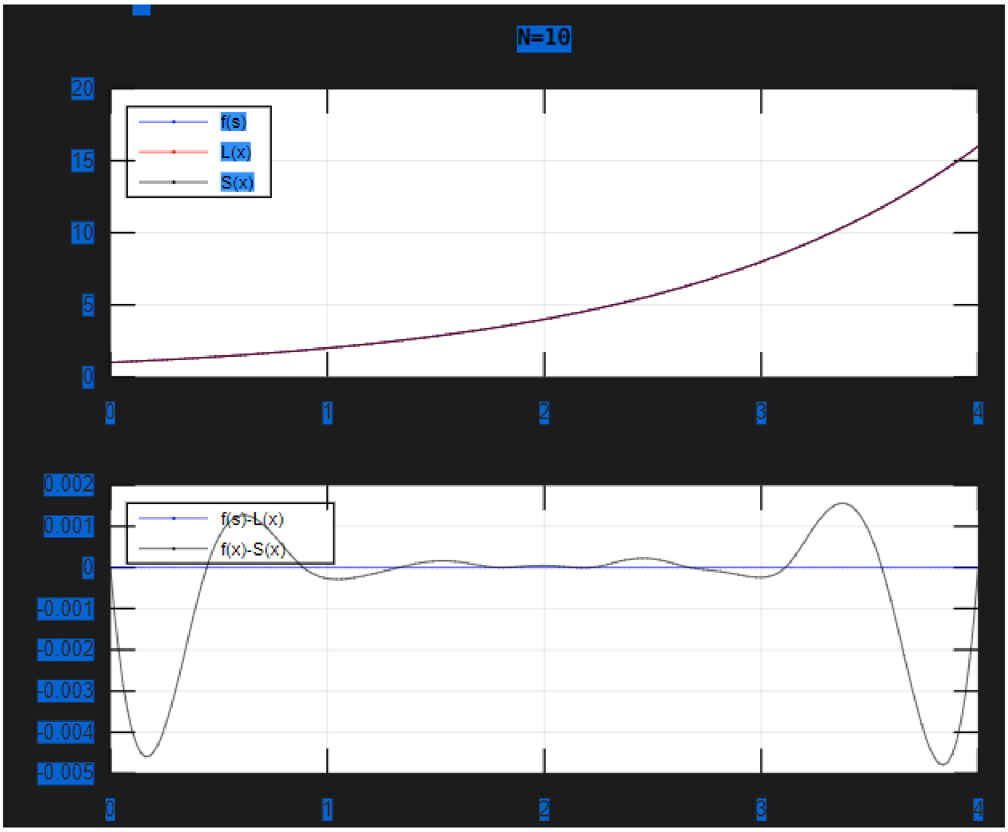

Рисунок 3 – интерполяция f_1 (N=10)

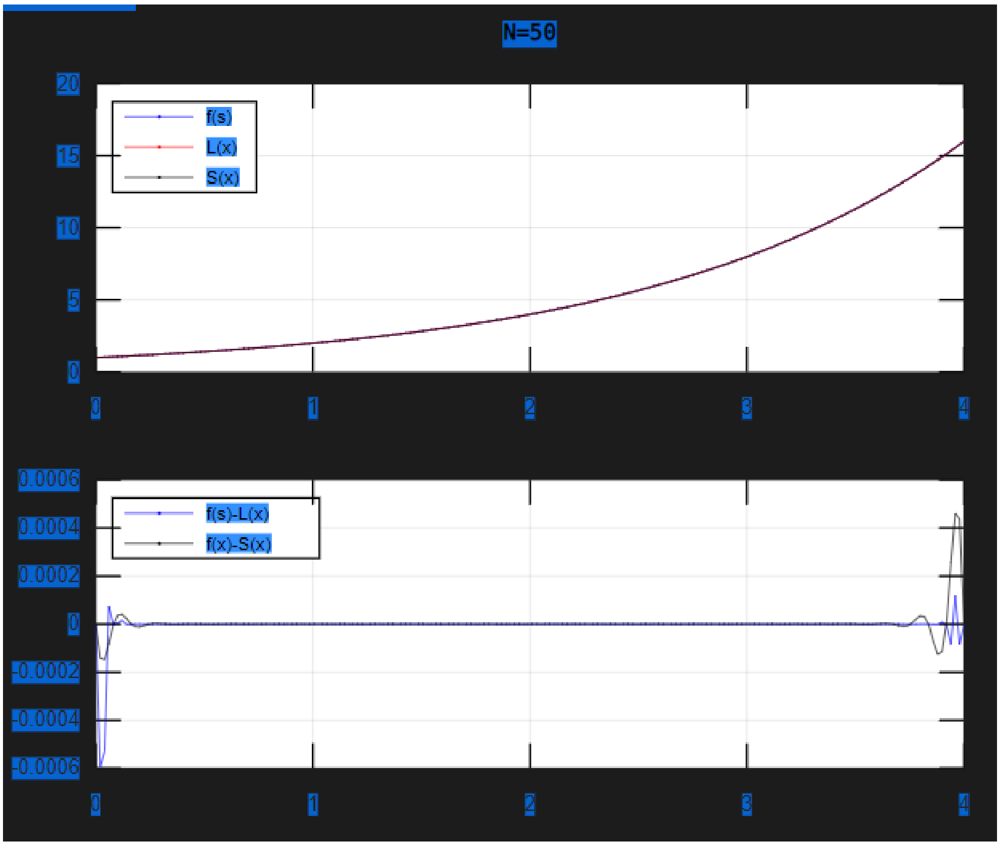

Рисунок 4 – интерполяция f_1 (N=50)

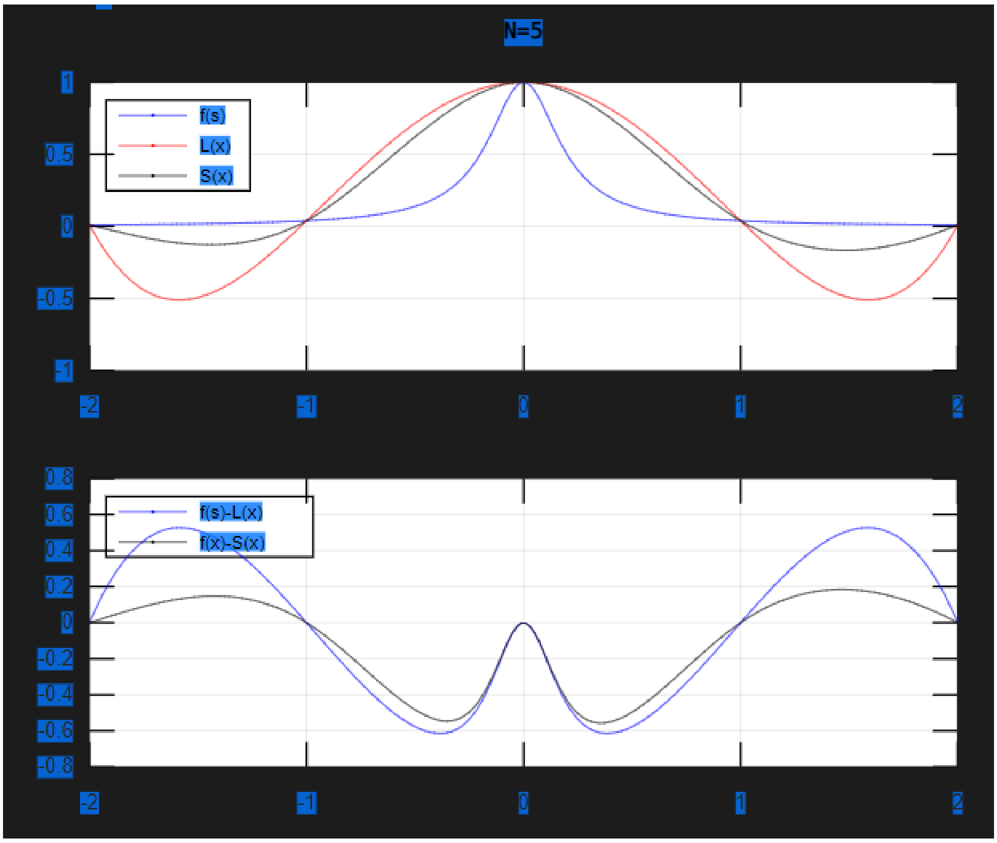

Рисунок 5 – интерполяция f_2 (N=5)

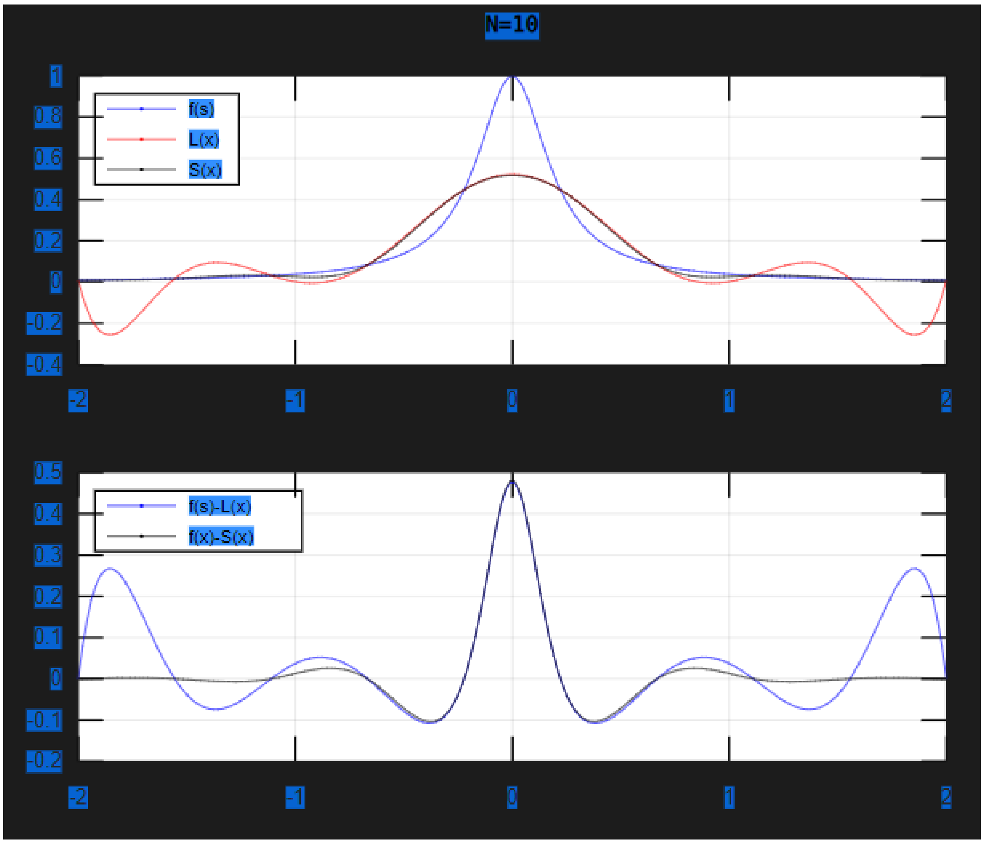

Рисунок 6 – интерполяция f_2 (N=10)

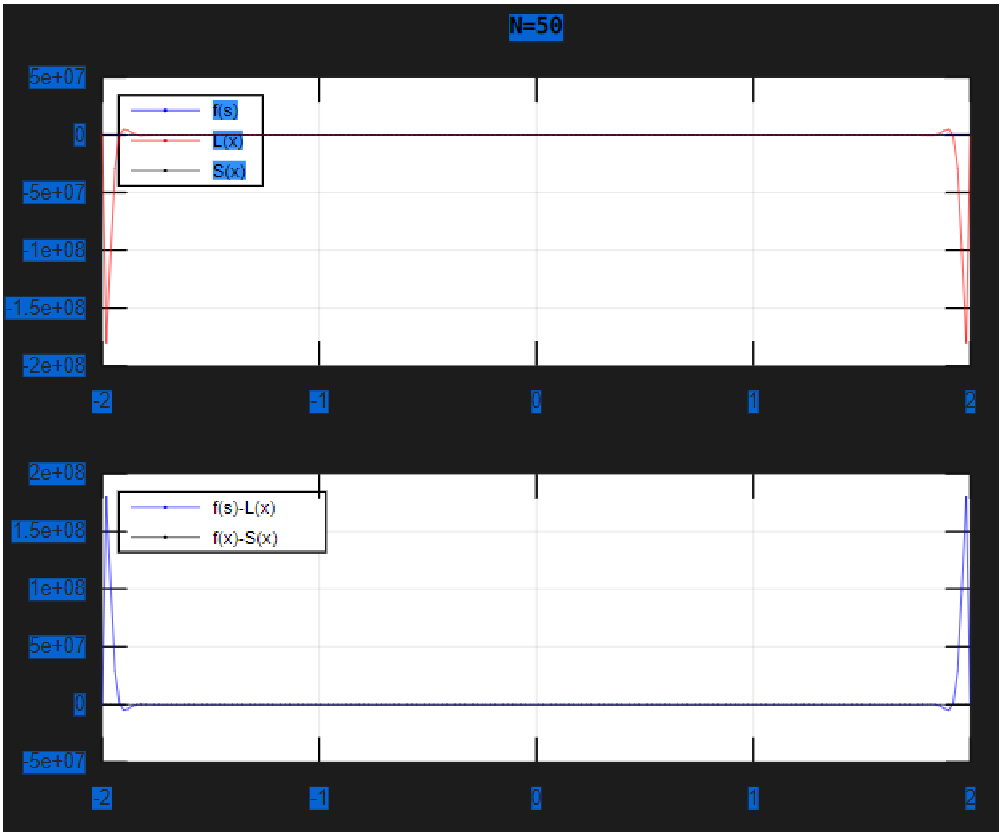

Рисунок 6 – интерполяция f_2 (N=50)

Вывод:

1) Метод бисекции показал себя универсальным и более эффективным в случае степенной функции. Остальные методы оказались значительно более эффективны в случае показательной функции, в особенности, метод Ньютона.

2) Интерполяция Лагранжа оказывается сопоставимой по точности с кубическими сплайнами при малом числе узлов и эффективной, например, для интерполяции монотонных функций. Однако, при росте числа узлов, метод интерполяции Лагранжа может давать огромные погрешности на границах интервала интерполяции.