|  |  |
| --- | --- |
|  | **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**

**Отчет**

|  |  |
| --- | --- |
| **по домашнему заданию №** | 2 |

**Дисциплина:**  Вычислительная математика

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Студент | ИУ6-62Б |  |  | С.В. Астахов | |
|  | (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 2x-0.1 – 1

= 0, x ∈ [0,1] и (x – 0.2)3 = 0, x ∈ [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 ɛ.

**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 ∈ [a, b]. Для решения уравнения методом простой итерации приведем его к равносильному виду:

x=φ(x)

Это всегда можно сделать, причем многими способами. Например: x=g(x)\*f(x) + x ≡ φ(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) = φ(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 x0 мы можем построить линейную аппроксимацию F(x) в окрестности x0: F(x0 + h) ≈ F(x0) + F’(x0) \* h и решить получающееся линейное уравнение F(x0) + F’(x0) \* h = 0. Так мы приходим к итеративному методу:

x(k+1) = xk – F’(xk)-1 \* F(xk),

k = 0, 1, ...

**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)

%Берѐм 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.

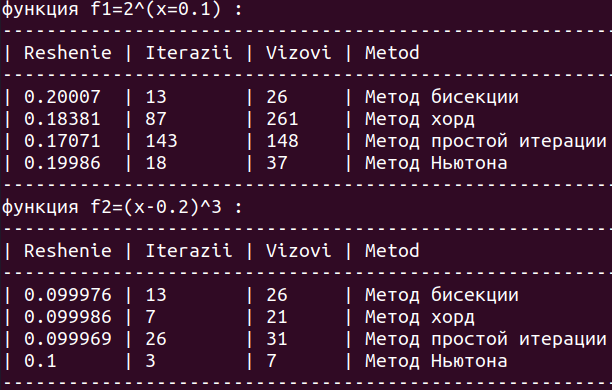


Рисунок 1 – приближенные решения уравнений

**Часть 2**

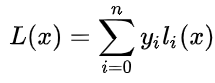
**Задание**

* Реализовать алгоритмы построения интерполяционного полинома Лагранжа и системы кубических сплайнов;
* отладить алгоритмы на тестовых примерах, построив интерполянты для 2x на [0, 4], (1 + 25x2) -1 на x ∈ [−2,2], используя равномерную сетку;
* в программе предусмотреть возможность вывода графиков функции f и двух интерполирующих функций, построенных разным цветов в общих осях.

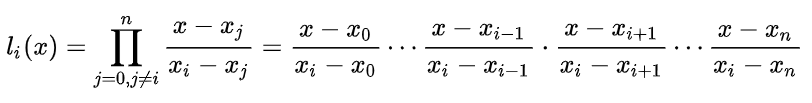
**Краткое описание интерполяционного полинома и его построения в форме Лагранжа**

Интерполяция алгебраическими многочленами функции f(x) действительного аргумента на отрезке [a,b] — нахождение коэффициентов многочлена Pn(x) степени меньшей или равной n, принимающего при значениях аргумента x0, x1, .., xn значения f(xi).

Многочлен Лагранжа:



Базисные полиномы:



**Краткое описание сплайн-интерполяции и алгоритма построения кубических сплайнов**

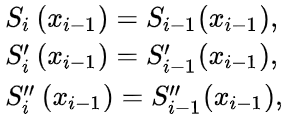
Понятие интерполяции описано выше.

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

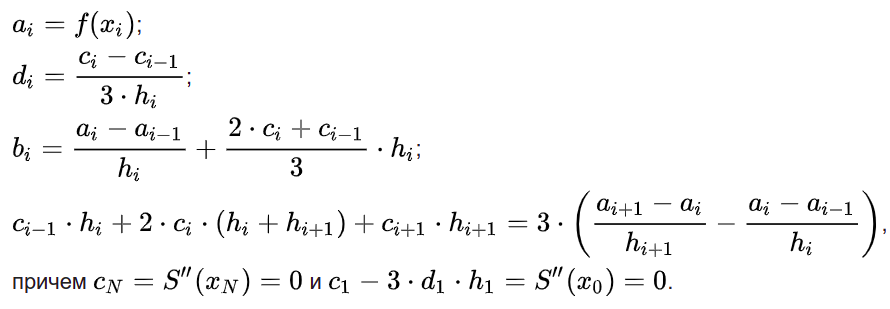
Пусть кубический сплайн на каждом отрезке [xi-1, xi] задается функцией:



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



Исходя из этого выводятся формулы для коэффициентов сплайна:



Если учесть, что c0 = cN = 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.

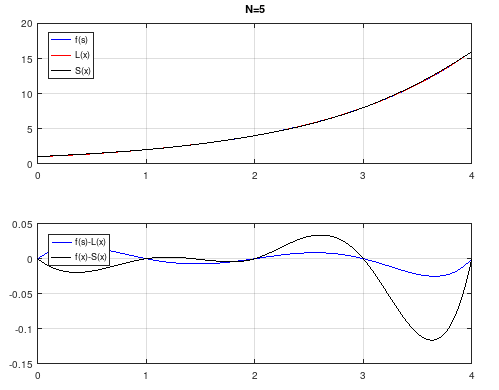


Рисунок 2 – интерполяция f1 (N=5)

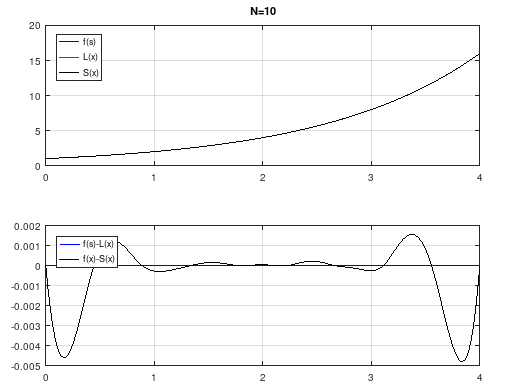


Рисунок 3 – интерполяция f1 (N=10)

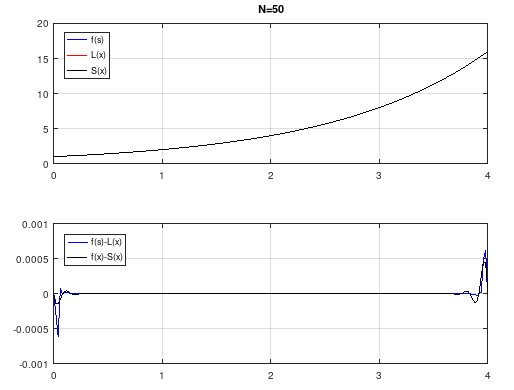


Рисунок 4 – интерполяция f1 (N=50)

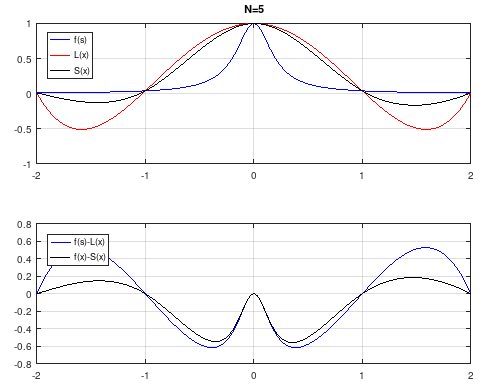


Рисунок 5 – интерполяция f2 (N=5)

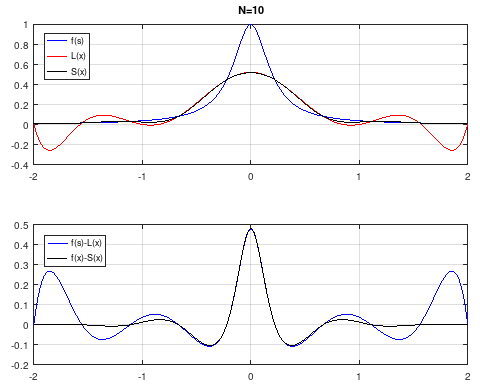


Рисунок 6 – интерполяция f2 (N=10)

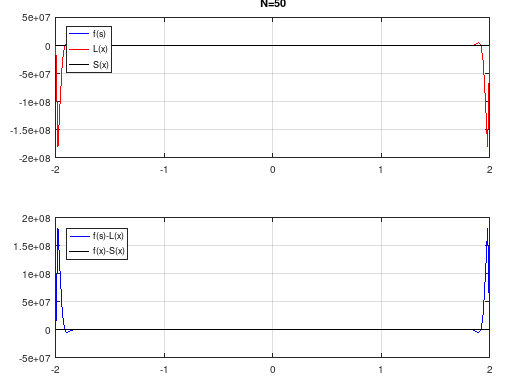


Рисунок 6 – интерполяция f2 (N=50)

**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.