Лабораторная работа по курсу

«Методы Численного Анализа»

Интерполирование функции

Вариант 10

Выполнил: Прудникович

Владислав Вячеславович

2 курс 6 группа

Преподаватель: Будник Анатолий Михайлович

**Постановка задачи**

Рассмотрим набор различных точек на отрезке [a, b], где

[a, b] = = 0.1 + 0.05 = 0.1 + 0.5 = 0.6.

[a, b] = .

= + где = , = 10, = 1/

В данных точках известны значения функции:

F( = (1- )sin = 0.6 + 0.4cos.

Требуется восстановить значения F( в других точках отрезка [0.6, 1.6], а именно в:

; ; .

Для восстановления значений использовать:

**А) Метод наименьших квадратов**

**Б) Многочлен Лагранжа**

**В) Многочлен Ньютона**

**Г) Многочлен Чебышева**

**Д) Интерполирование по равностоящим узлам**

(только в точке )

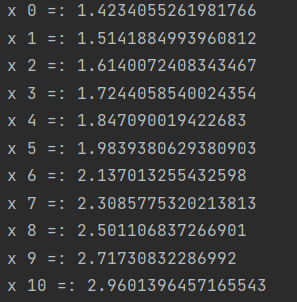
Оценить точноть полученного результата для каждого из методов.

**Пункт 0. Подготовка к методам**

Перед тем, как перейти к интерполированию, найдём значения известных нам узлов и составим таблицу значений и F().  
Значения найдём с точностью до

|  |  |  |
| --- | --- | --- |
|  | F() |  |
| 0.6 | 1.4234055261981766 | 0 |
| 0.7 | 1.5141884993960812 | 1 |
| 0.8 | 1.6140072408343467 | 2 |
| 0.9 | 1.7244058540024354 | 3 |
| 1.0 | 1.847090019422683 | 4 |
| 1.1 | 1.9839380629380903 | 5 |
| 1.2 | 2.137013255432598 | 6 |
| 1.3 | 2.3085775320213813 | 7 |
| 1.4 | 2.501106837266901 | 8 |
| 1.5 | 2.71730832286992 | 9 |
| 1.6 | 2.9601396457165543 | 10 |

Код программы : Результат работы программы:

def function(xj):   
 rez = 0.6\*np.exp(xj)+0.4\*np.cos(xj)  
 return rez  
  
xvector = [0]\*11  
x0 = 0.6  
i = 0  
while i < 11:  
 xvector[i] = function(x0)  
 print("x", i, "=:", xvector[i])  
 i += 1  
 x0 += 0.1

**Пункт А. Интерполирование методом наименьших квадратов**

Элемент наилучшего приближения определим, как линейную комбинацию вида :

Ф() =, где =

Коэффициенты найдём из решения СЛАУ, которое записывается в следующем виде:

Заметим, что матрица данной СЛАУ является матрицей Гильберта и является плохо обусловленной уже при относительно небольших , поэтому для получения ЭНП ограничимся многочленом 5-ой степени, => = 5.

Итак , ЭНП будет иметь вид . Полученная система:

Погрешность приближения определим среднеквадратичным отклонением, которое будет вычисляться по след.формуле:

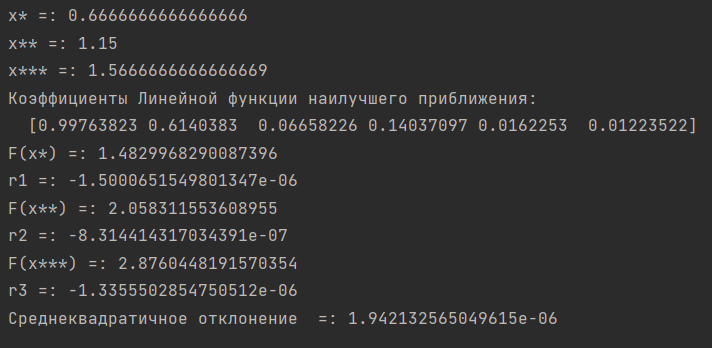
, где - исходная функция.

Для определения элементов расширенной матрицы системы будем использовать следующие формулы:

**Код программы:**

import numpy as np  
def f(xj):  
 rez = 0.6\*np.exp(xj)+0.4\*np.cos(xj)  
 return rez  
def ANP(x):  
 F = 0  
 i = 0  
 while i < 6:  
 F += c[i]\*x\*\*i  
 i += 1  
 return F  
def s(i):  
 return np.sum([xvector[0]\*\*i, xvector[1]\*\*i, xvector[2]\*\*i,  
 xvector[3]\*\*i, xvector[4]\*\*i, xvector[5]\*\*i,  
 xvector[6]\*\*i, xvector[7]\*\*i, xvector[8]\*\*i,  
 xvector[9]\*\*i, xvector[10]\*\*i])  
def m(i):  
 return np.sum([f(xvector[0])\*xvector[0]\*\*i, f(xvector[1])\*xvector[1]\*\*i, f(xvector[2])\*xvector[2]\*\*i,  
 f(xvector[3])\*xvector[3]\*\*i, f(xvector[4])\*xvector[4]\*\*i, f(xvector[5])\*xvector[5]\*\*i,  
 f(xvector[6])\*xvector[6]\*\*i, f(xvector[7])\*xvector[7]\*\*i, f(xvector[8])\*xvector[8]\*\*i,  
 f(xvector[9])\*xvector[9]\*\*i, f(xvector[10])\*xvector[10]\*\*i])  
def delta():  
 delt = 0  
 i = 0  
 while i < 6:  
 delt += (ANP(xvector[i])-f(xvector[i]))\*\*2  
 i += 1  
 delt = np.sqrt(delt)  
 return delt  
xvector = [0]\*11  
x0 = 0.6  
i1 = 0  
while i1 < 11:  
 xvector[i1] = x0  
 i1 += 1  
 x0 += 0.1  
A = np.array([[s(0), s(1), s(2), s(3), s(4), s(5)], [s(1), s(2), s(3), s(4), s(5), s(6)],  
 [s(2), s(3), s(4), s(5), s(6), s(7)], [s(3), s(4), s(5), s(6), s(7), s(8)],  
 [s(4), s(5), s(6), s(7), s(8), s(9)], [s(5), s(6), s(7), s(8), s(9), s(10)]])  
B = np.array(([m(0), m(1), m(2), m(3), m(4), m(5)]))  
c = np.linalg.solve(A, B)  
x1 = xvector[0] + 2/3\*0.1  
print("\nx\* =:", x1)  
x2 = xvector[5] + 0.5\*0.1  
print("x\*\* =:", x2)  
x3 = xvector[10] - 1/3\*0.1  
print("x\*\*\* =:", x3)  
print("Коэффициенты Линейной функции наилучшего приближения:\n ", c)  
print("F(x\*) =:", ANP(x1))  
print("r1 =:", f(x1) - ANP(x1))  
print("F(x\*\*) =:", ANP(x2))  
print("r2 =:", f(x2) - ANP(x2))  
print("F(x\*\*\*) =:", ANP(x3))  
print("r3 =:", f(x3) - ANP(x3))  
print("Cреднеквадратичное отклонение =:", delta())

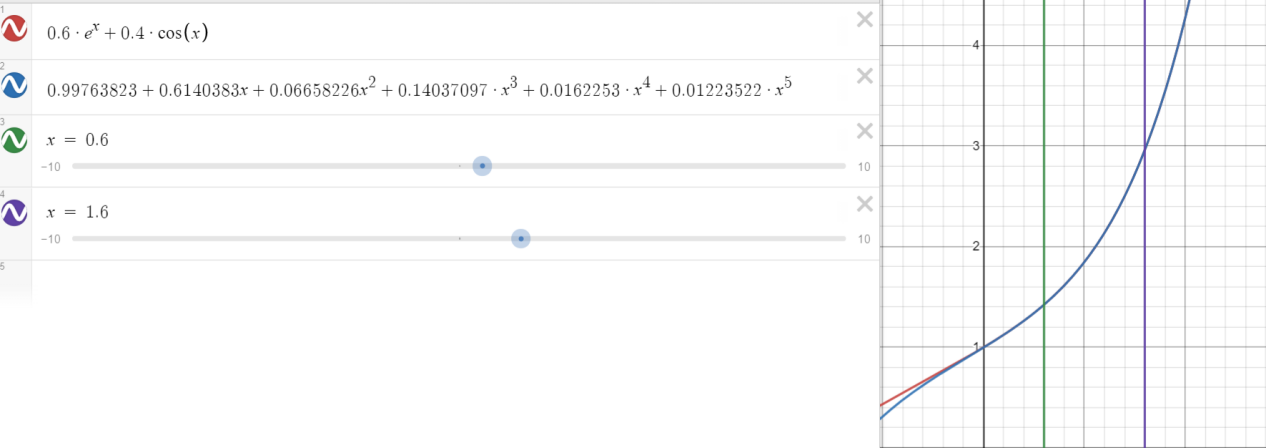
**Результат работы программы.**



**Анализ полученных значений.**

Вычислив погрешность приближения путём разности значений f(x) и Ф(x), мы получили r для всех полученных х. Данный результат совпадает со среднеквадратичным отклонением .

Зная коэффициенты полученного многочлена, можно построить его график и сравнить его с графиком исходной функции на заданном промежутке.



Можно заметить, что график полученной функции совпадает с графиком исходной на заданном отрезке [0.6, 1.6].  
Также заметим, что исходная функция монотонно возрастрает и её скорость роста увеличивается. Это в будущем поможет нам найти

M = max|f’(x)| на [a, b].

**Пункт Б. Интерполирование с помощью многочлена Лагранжа**

Интерполяционный многочлен P(x) можно представить в виде:

, где = , = .

= (, =>

= ( -

Данное представление интерполяционного многочлена называется формой Лагранжа.

В данном случае выбор степени многочлена не влияет на разрешимость уравнения, однако увеличивает точность приближения, => возьмём максимально возможную степень, а именно n = 10.

, где = , = .

= (, =>

= ( -

Для оценки погрешности используем верхнюю оценку остатка интерполирования:

Заметим, что производные четных порядков (не кратных 4) у нашей функции 0.6 + 0.4cos отличаются только знаком перед 0.4cos.

Тогда

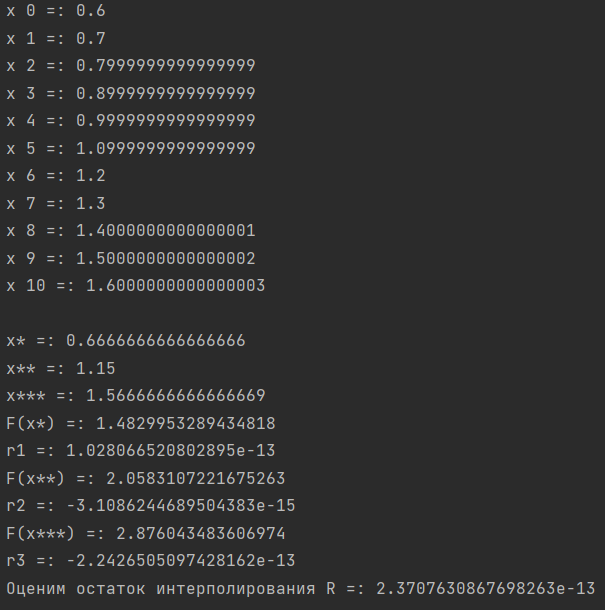
= 0.6 + 0.4sin

Теперь нам известно всё, чтобы найти ЭНП для заданной функции ф форме Лагранжа и оценить точность приближения.

**Код программы:**

import numpy as np  
import math  
def f(xj):  
 rez = 0.6\*np.exp(xj)+0.4\*np.cos(xj)  
 return rez  
def df(xj):  
 rez = 0.6\*np.exp(xj)+0.4\*np.cos(xj)  
 return rez  
def w11(x):  
 w = 1  
 i = 0  
 while i < 11:  
 w = w\*(x-xvector[i])  
 i += 1;  
 return w  
def dw(i):  
 dw = 1  
 j = 0  
 while j < 11:  
 if i != j:  
 dw = dw \* (xvector[i] - xvector[j])  
 j += 1  
 return dw  
def L(x) :  
 p = 0;  
 i = 0  
 while i < 11:  
 pn = w11(x)/((x - xvector[i]) \* dw(i))  
 p += pn\*f(xvector[i])  
 i+=1  
 return p  
xvector = [0]\*11  
x0 = 0.6  
i1 = 0  
while i1 < 11:  
 xvector[i1] = x0  
 print("x", i1, "=:", xvector[i1])  
 i1 += 1  
 x0 += 0.1  
R = w11(0.65)\*df(1.6)/math.factorial(11)  
x1 = xvector[0] + 2/3\*0.1  
print("\nx\* =:", x1)  
x2 = xvector[5] + 0.5\*0.1  
print("x\*\* =:", x2)  
x3 = xvector[10] - 1/3\*0.1  
print("x\*\*\* =:", x3)  
print("F(x\*) =:", L(x1))  
print("r1 =:", f(x1) - L(x1))  
print("F(x\*\*) =:", L(x2))  
print("r2 =:", f(x2) - L(x2))  
print("F(x\*\*\*) =:", L(x3))  
print("r3 =:", f(x3) - L(x3))  
print("Оценим остаток интерполирования R =:", R)

**Результат работы программы:**



**Анализ полученных значений.**

**F(x\*) =: 1.4829953289434818**

**r1 =: 1.028066520802895e-13**

**F(x\*\*) =: 2.0583107221675263**

**r2 =: -3.1086244689504383e-15**

**F(x\*\*\*) =: 2.876043483606974**

**r3 =: -2.2426505097428162e-13**

Значения невязок полученных значений довольно малы и соответствуют верхней оценке по остатку интерполирования. Заметим, что интерполирование с помощью интерполяционного многочлена в форме Лагранжа является не только более точным, чем метод наименьших квадратов, но и более быстрым, т.к. нам не нужно решать СЛАУ, появляющуюся в ходе решения МНК.

**Пункт В. Интерполирование с помощью многочлена Ньютона**

Запишем интерполяционный многочлен в следующем виде:

*,*

Где - разделенная разность и = .

= , а =

Для удобства в обращении к разделенным разностям во время программирования метода, составим следующую таблицую Разделенный разностей:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  | ….………….. |  |
|  |  | ….……………. |  |  |
| ….……………. | ….…………….. |  |  |  |
| ….…………… |  |  |  |  |
|  |  |  |  |  |

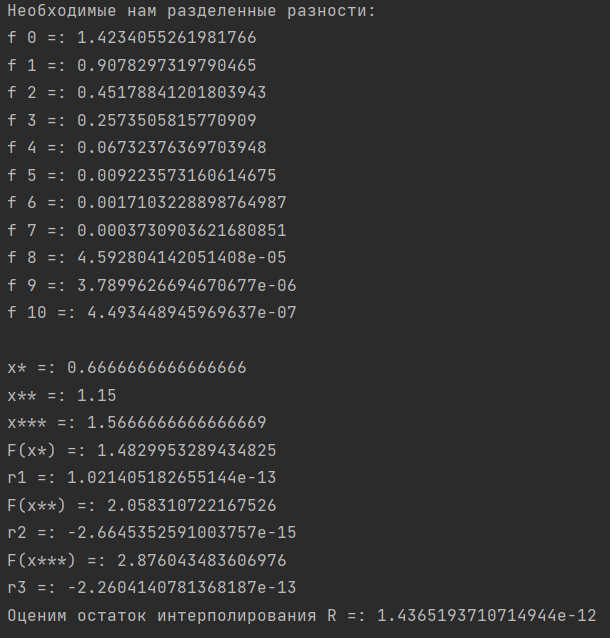
Остаточный член многочлена в форме Ньютона оценивается следующим образом:

Где , как и ранее, равно (

**Код программы:**

import numpy  
import numpy as np  
import math  
def f(xj):  
 rez = 0.6\*np.exp(xj)+0.4\*np.cos(xj)  
 return rez  
def df(xj):  
 rez = 0.6\*np.exp(xj)+0.4\*np.cos(xj)  
 return rez  
def w11(x):  
 w = 1  
 i = 0  
 while i < 11:  
 w = w\*(x-xvector[i])  
 i += 1;  
 return w  
def createFmatrix():  
 for i in range(11):  
 F[i][0] = f(xvector[i])  
 for i in range(1, 11):  
 for j in range(11-i):  
 F[j][i] = (F[j+1][i-1]-F[j][i-1])/(xvector[i+j]-xvector[j])  
 return F[0]  
def N(x):  
 p = f1[0]  
 w = 1;  
 for i in range(10):  
 w \*= (x - xvector[i])  
 p += f1[i + 1] \* w  
 return p  
xvector = [0]\*11  
x0 = 0.6  
i1 = 0  
while i1 < 11:  
 xvector[i1] = x0  
 print("x", i1, "=:", xvector[i1])  
 i1 += 1  
 x0 += 0.1  
F = numpy.zeros((11, 11))  
f1 = createFmatrix()  
print("Необходимые нам разделенные разности:")  
for i in range(11):  
 print("f", i, "=:", f1[i])  
R = w11(0.65)\*f1[10]  
x1 = xvector[0] + 2/3\*0.1  
print("\nx\* =:", x1)  
x2 = xvector[5] + 0.5\*0.1  
print("x\*\* =:", x2)  
x3 = xvector[10] - 1/3\*0.1  
print("x\*\*\* =:", x3)  
print("F(x\*) =:", N(x1))  
print("r1 =:", f(x1) - N(x1))  
print("F(x\*\*) =:", N(x2))  
print("r2 =:", f(x2) - N(x2))  
print("F(x\*\*\*) =:", N(x3))  
print("r3 =:", f(x3) - N(x3))  
print("Оценим остаток интерполирования R =:", R)

**Результат работы программы:**



**Анализ полученных значений.**

F(x\*) =: 1.4829953289434825 r1 =: 1.021405182655144e-13

F(x\*\*) =: 2.058310722167526 r2 =: -2.6645352591003757e-15

F(x\*\*\*) =: 2.876043483606976 r3 =: -2.2604140781368187e-13

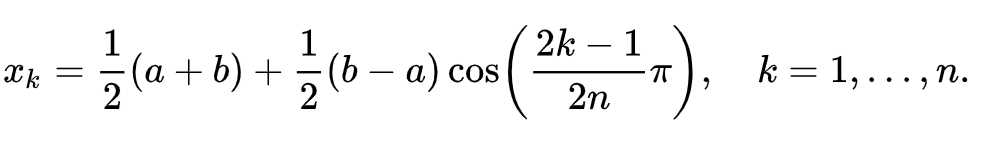
Оценим остаток интерполирования R =: 1.4365193710714944e-12

Порядок полученных невязок совпадает с порядком значений при интерполировании с помощью многочлена Лагранжа. Для запуска алгоритма необходимо высчитать N значений разделенных разностей, для чего в программе составлялась таблица РР.

Остаток интерполирования действительно является верхней границей для полученных значений невязок.

**Пункт Г. Интерполирование с помощью узлов многочлена Чебышева**

Корни многочлена Чебышёва определяются по формуле:



В нашем случае a = 0.6, b = 1.6, n = 11.

Перепишем исходную формулу для k = 0,…,n-1

Данный подход к поиску начальных узлов интерполяции позволяет нам снизить влияние феномена Рунге при интерполировании многочленами высокой степени.

Для дальнейшего интерполирования мы можем использовать любой из уже описанных ранее методов, подставляя в качестве уже известные нам узлы.

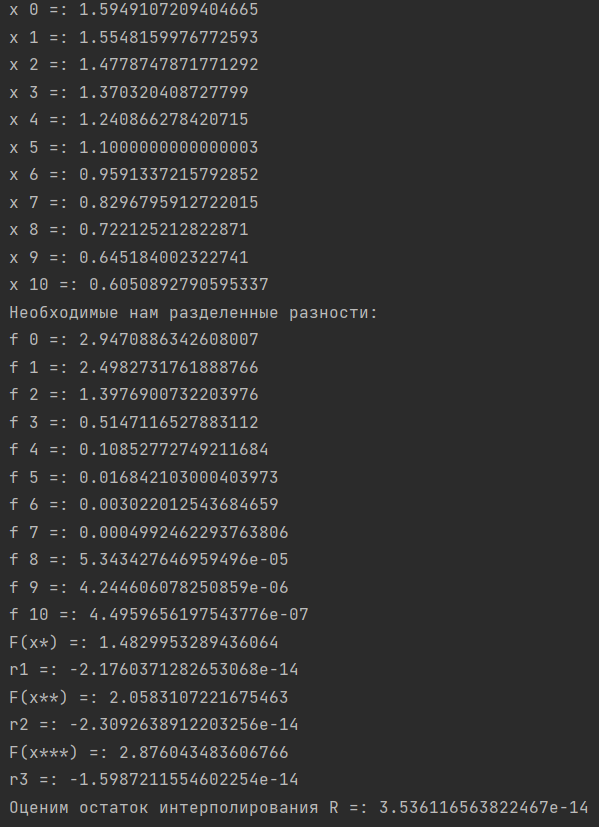
Остаток интерполирования оценивается следующим образом:

Воспользуемся методом интерполяции Ньютона.

**Код программы:**

import numpy as np  
import math  
def f(xj):  
 rez = 0.6\*np.exp(xj)+0.4\*np.cos(xj)  
 return rez  
def df(xj):  
 rez = 0.6\*np.exp(xj)+0.4\*np.cos(xj)  
 return rez  
def w11(x):  
 w = 1  
 i = 0  
 while i < 11:  
 w = w\*(x-xvector[i])  
 i += 1;  
 return w  
def createFmatrix():  
 for i in range(11):  
 F[i][0] = f(xvector[i])  
 for i in range(1, 11):  
 for j in range(11-i):  
 F[j][i] = (F[j+1][i-1]-F[j][i-1])/(xvector[i+j]-xvector[j])  
 return F[0]  
def N(x):  
 p = f1[0]  
 w = 1;  
 for i in range(10):  
 w \*= (x - xvector[i])  
 p += f1[i + 1] \* w  
 return p  
xvector = [0]\*11  
x0 = 0.6  
i1 = 0  
while i1 < 11:  
 x0 = 1.1+0.5\*(np.cos(math.pi\*(2\*i1+1)/22))  
 xvector[i1] = x0  
 print("x", i1, "=:", xvector[i1])  
 i1 += 1  
F = np.zeros((11, 11))  
f1 = createFmatrix()  
print("Необходимые нам разделенные разности:")  
for i in range(11):  
 print("f", i, "=:", f1[i])  
R = df(1.6)/(math.factorial(11)\*2\*\*21)  
x1 = xvector[0] + 2/3\*0.1  
x2 = xvector[5] + 0.5\*0.1  
x3 = xvector[10] - 1/3\*0.1  
print("F(x\*) =:", N(x1))  
print("r1 =:", f(x1) - N(x1))  
print("F(x\*\*) =:", N(x2))  
print("r2 =:", f(x2) - N(x2))  
print("F(x\*\*\*) =:", N(x3))  
print("r3 =:", f(x3) - N(x3))  
print("Оценим остаток интерполирования R =:", R)

**Результат работы программы:**



Значения невязки и остатка интерполирования имеют меньший порядок, чем при работе с методом Ньютона. Это говорит нам о том, что выбор узлов по формулам Чебышёва увеличивает точность вычислений по крайней мере для метода Ньютона.