**Министерство образования Республики Беларусь**

**Белорусский государственный университет**

**Факультет Прикладной математики и инфоРМАТИКИ**

**Кафедра информационных систем управления**

**Отчет о выполнении лабораторной работы №1 по «МВ»**

**Приближение функций**

**Вариант 2**

Подготовил студент  
3 курса 12 группы  
Полывяный Глеб Андреевич

Преподаватель  
Будник А.М.

Минск, 2022

# Генерация начальных данные

**Постановка задачи**

Рассмотрим на набор различных значений:

Для функции:

(12)

Требуется восстановить значения в точках восстановления

j = 12, т.к. такой номер в списке

= 0,7

Интервал:

[0,7; 1,7];

Точки восстановления:

x\* = 0.7666666666666667332800481442

x\*\* = 1.250000000000000097144514655

x\*\*\* = 1.666666666666666788791199376

Таблица:

|  |  |
| --- | --- |
|  |  |
| 0,7 | 0.5770623688393331273178554657 |
| 0,8 | 0.5235422876862046938721110115 |
| 0,9 | 0.4710817522996186413859270738 |
| 1,0 | 0.4196063005804515518344689687 |
| 1,1 | 0.3690885950163288717412709964 |
| 1,2 | 0.3195432746815434336464661304 |
| 1,3 | 0.2710219037111850196608318793 |
| 1,4 | 0.2236080176291967531707497035 |
| 1,5 | 0.1774122726042116686624189424 |
| 1,6 | 0.1325677059058721391529705100 |
| 1,7 | 0.0892251185482567889991413211 |

**Листинг**

Получение и объявление основных констант:

VARIANT\_COEF = Decimal(0.1 + 0.05 \* 12)

N = 10

m = N // 2

H = Decimal(1 / N)

Код для получения интервала:

x = [(VARIANT\_COEF + Decimal(i \* H)) for i in range(N + 1)]

Код для получения значения функции из задания:

def f(x) -> Decimal:

return VARIANT\_COEF \* Decimal.exp(-x) + (1 - VARIANT\_COEF) \* Decimal(math.cos(x))

fx = [f(x[i]) for i in range(N + 1)]

Код для получения точек восстановления:

x\_1 = x[0] + Decimal(2/3) \* H

x\_2 = x[round((N) / 2)] + Decimal(1 / 2) \* H

x\_3 = x[N] - Decimal(1 / 3) \* H

Интерполяционный многочлен Ньютона

**Постановка задачи**

* По заданным начальным данным построить интерполяционный многочлен Ньютона.
* Вычислить значения в точках x\*, x\*\*, x\*\*\*.
* Оценить погрешность по формуле Лагранжа в точках x\*, x\*\*, x\*\*\*.
* Сравнить погрешность с истинной погрешностью

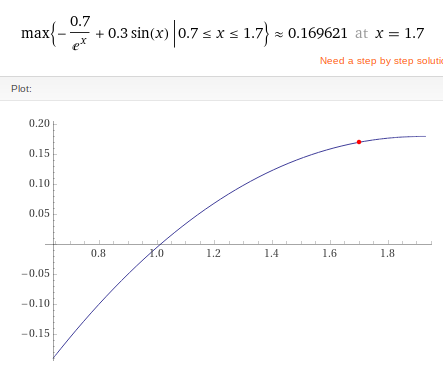
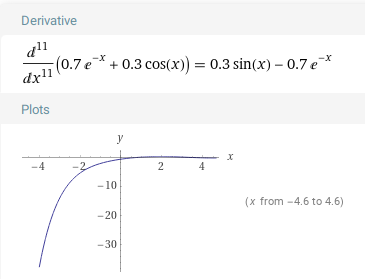
Построение таблицы разделённых разностей:

, j = , i =

Построение многочлена Ньютона

Для нахождения погрешности по формуле Лагранжа нам нужно найти максимальное значение производной 11-го порядка.

=



Видим, что максимальное значение на промежутке функция примет в последней точке промежутка, значит

берём для x = 1.7

**Результаты:**

= 0.7666666666666667332800481442

= 0.5412594525687322947342314343

= 2.6551948337466E-15

= 1.250000000000000097144514655

= 0.2951500665207136928775535075

= -4.19036656193E-17

= 1.666666666666666788791199376

= 0.1034958575819885839494726657

= -7.9919050601644E-15

**Вывод:**

Заметим, что истинная погрешность согласуется с оценкой теоретической погрешности во всех точках.

**Листинг:**

Создание таблицы разделённых разностей:

def createDDTable(x: list, f: list) -> list:

dd = [[Decimal(0.0) for \_ in range(N + 1)] for \_ in range(N + 1)]

for i in range(0, N + 1):

dd[i][0] = f[i]

for i in range(1, N + 1):

for j in range(N - i + 1):

dd[j][i] = (dd[j + 1][i - 1] - dd[j][i - 1]) / (x[i + j] - x[j])

return dd

Построение многочлена Ньютона:

def newton(x, nodes: list, dd\_table: list) -> Decimal:

result = Decimal(0.0)

temp = Decimal(0.0)

for i in range(N + 1):

temp = dd\_table[0][i]

for j in range(i):

temp \*= (x - nodes[j])

result += temp

return result

Функция производной 11-го порядка:

def f\_11\_derivative(x) -> Decimal:

return (-VARIANT\_COEF) \* Decimal.exp(-x) + (1 - VARIANT\_COEF) \* Decimal(math.sin(x))

Погрешность методом Лагранжа:

def theoreticalErrorByLagrange(x: Decimal, nodes: list, M: Decimal) -> Decimal:

w = Decimal(1)

for i in range(N + 1):

w \*= x - nodes[i]

return abs(w \* M) / Decimal(math.factorial(N + 1))

Реальная погрешность:

def realError(x: Decimal, px: Decimal) -> Decimal:

return f(x) - px

Вызов функций:

x\_1 = x[0] + Decimal(2/3) \* H

x\_2 = x[round((N) / 2)] + Decimal(1 / 2) \* H

x\_3 = x[N] - Decimal(1 / 3) \* H

dd\_table = createDDTable(x, fx)

y\_1 = newton(x\_1, x, dd\_table)

y\_2 = newton(x\_2, x, dd\_table)

y\_3 = newton(x\_3, x, dd\_table)

Таблица разделённых разностей:

5.771e-1 -5.352e-1 5.298e-2 -1.241e-2 1.964e-2 -4.290e-3 1.340e-4 2.738e-6 9.162e-6 -1.365e-6 2.852e-8

5.235e-1 -5.246e-1 4.925e-2 -4.556e-3 1.749e-2 -4.210e-3 1.359e-4 1.007e-5 7.934e-6 -1.336e-6

4.711e-1 -5.148e-1 4.789e-2 2.440e-3 1.539e-2 -4.128e-3 1.430e-4 1.641e-5 6.731e-6

4.196e-1 -5.052e-1 4.862e-2 8.594e-3 1.332e-2 -4.042e-3 1.544e-4 2.180e-5

3.691e-1 -4.955e-1 5.120e-2 1.392e-2 1.130e-2 -3.950e-3 1.697e-4

3.195e-1 -4.852e-1 5.537e-2 1.844e-2 9.325e-3 -3.848e-3

2.710e-1 -4.741e-1 6.091e-2 2.217e-2 7.402e-3

2.236e-1 -4.620e-1 6.756e-2 2.513e-2

1.774e-1 -4.484e-1 7.510e-2

1.326e-1 -4.334e-1

8.923e-2

Интерполирование на равномерной сетке узлов

**Постановка задачи:**

Провести интерполирование в конце таблицы полиномом третьей степени. Оценить погрешность.

Узлы были заданы на равномерной сетке узлов, поэтому корректировать их не нужно.

Введем формулу для конечной разности:

Положим

Для нахождения конечных разностей, построим таблицу конечных разностей.

Будем использовать общую формулу . Таким образом, начиная с значения функций в узлах, мы будем строить разделенные разности по порядку, вплоть до n-порядка. В конечном итоге таблица конечных разностей будет иметь вид:

Преобразуем полином Ньютона с учетом равномерной сетки, т.е. , с использованием конечных разностей:

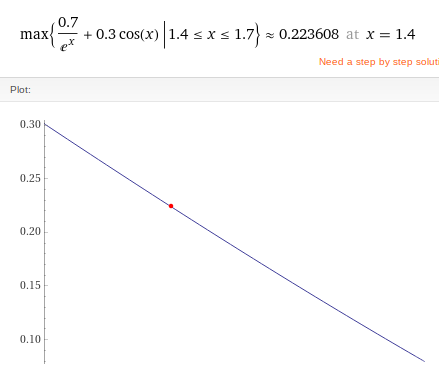
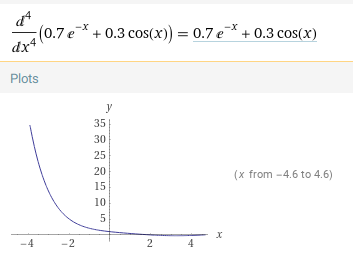
Произведем замену переменной, введем :

Погрешность:

Для оценки сверху необходимо оценить четвертую производную:

Четвертая производная эквивалентна самой восстанавливаемой функции, т.е. . Таким образом, нам нужно оценить значения восстанавливаемой функции на

Функция монотонно убывает, поэтому достигает максимума на промежутке в точке .



Соответственно оценкабудет выглядеть следующим образом:

В нашем случае будет использоваться полином степени 3, следовательно

Тогда остаток интерполирования принимает вид:

**Результаты:**

x\*\*\* = 1.666666666666666788791199376

f(x\*\*\*) = 0.1034964523694753580746685101

= -5.947874947660302560088E-7

**Вывод:**

Реальная погрешность согласуется с теоретической оценкой, а так же явно видно, насколько сильно упала точность из-за уменьшения степени полинома.

**Листинг**

Создание таблицы конечных разностей:

def createFDTable(f: list) -> list:

kr = [[Decimal(0.0)for \_ in range(N + 1)] for \_ in range(N + 1)]

for i in range(N + 1):

kr[i][0] = f[i]

for i in range(1, N + 1):

for j in range(N - i + 1):

kr[j][i] = kr[j + 1][i - 1] - kr[j][i - 1]

return kr

Построение интерполяционного многочлена:

def interpolateOnUniformGrid(x: Decimal, x0: Decimal, fd: list, dots\_count: int) -> Decimal:

t = (x - x0) / H

result = fd[N][0]

temp = t

for i in range(1, dots\_count):

result += (fd[N - i][i] \* temp)

temp \*= ((t + i) / (i + 1))

return result

Оценка сверху для погрешности:

def theoreticalErrorOnGrid(x: Decimal, xn: Decimal, M: Decimal, dots\_count: int):

t = (x - xn) / H

result = H \*\* dots\_count

for i in range(1, dots\_count):

result \*= (t + i)

return (result \* M) / math.factorial(dots\_count)

Таблица конечных разностей:

5.771e-1 -5.352e-2 1.060e-3 -7.446e-5 4.712e-5 -5.148e-6 9.647e-8 1.380e-9 3.694e-9 -4.953e-10 1.035e-11

5.235e-1 -5.246e-2 9.851e-4 -2.734e-5 4.198e-5 -5.052e-6 9.785e-8 5.074e-9 3.199e-9 -4.849e-10

4.711e-1 -5.148e-2 9.577e-4 1.464e-5 3.693e-5 -4.954e-6 1.029e-7 8.273e-9 2.714e-9

4.196e-1 -5.052e-2 9.724e-4 5.156e-5 3.197e-5 -4.851e-6 1.112e-7 1.099e-8

3.691e-1 -4.955e-2 1.024e-3 8.354e-5 2.712e-5 -4.740e-6 1.222e-7

3.195e-1 -4.852e-2 1.107e-3 1.107e-4 2.238e-5 -4.617e-6

2.710e-1 -4.741e-2 1.218e-3 1.330e-4 1.776e-5

2.236e-1 -4.620e-2 1.351e-3 1.508e-4

1.774e-1 -4.484e-2 1.502e-3

1.326e-1 -4.334e-2

8.923e-2

Минимизация остатка интерполирования.

**Постановка задачи:**

Минимизировать остаток интерполирования за счет минимизации , что достигается использованием корней полинома Чебышева в качестве узлов интерполяции для восстанавливаемой функции.

* Найти корни многочлена Чебышева
* Задать значения функции в заданных точках
* Построить интерполяционный многочлен Лагранжа

Рекуррентная формула для построения полинома Чебышева:

В тригонометрической форме полином Чебышева имеет следующий вид:

Свойства многочлена Чебышева:

1. Многочлен Чебышева имеет на отрезке [-1,1] ровно m различных действительных корней.
2. Для приведенного полинома Чебышева справедлива теорема:  
   Среди всех многочленов фиксированной степени со старшим коэффициентом , коэффициентов , равным единице, наименьшее уклонение от нуля, равное , имеет приведенный полином Чебышева.

Чтобы найти корни полинома, надо решить уравнение:

Чтобы получить нужное количество узлов, а именно n + 1 узел, нужно чтобы полином Чебышева был степени

Корни вычисляем по формуле:

Но для того, чтобы получить узлы интерполяции, нам нужно еще перенести узлы полинома Чебышева на нужный нам интервал, т.е. с . Для этого используем формулу:

После нахождения всех найдем значения и получим таблицу узлов-значений.

Результаты:

Таблица:

|  |  |
| --- | --- |
|  |  |
| 0.7050892790595337247854956259 | 0.5743112317363268814481249569 |
| 0.7451840023227409595776295112 | 0.5527418114985215062751376410 |
| 0.8221252128228709944515857808 | 0.5118473943880285320971441687 |
| 0.9296795912722014681816901797 | 0.4557033436415577096704877386 |
| 1.059133721579285251031750191 | 0.3896175276080953280596430474 |
| 1.2 | 0.3195432746815434336464661304 |
| 1.340866278420714993217315226 | 0.2515060789134962304897894815 |
| 1.470320408727798942600828931 | 0.1909883625334831861149940840 |
| 1.577874787177129249797479636 | 0.1423657928390003462247618864 |
| 1.654815997677259340182587137 | 0.1086127985199025872163729029 |
| 1.694910720940466463952418560 | 0.09139213836430955217282233992 |

= 0.7666666666666667332800481442

= 0.5412594525687354989850143414

= -5.490559491605E-16

= 1.250000000000000097144514655

= 0.2951500665207143538443573297

= -7.028704694415E-16

= 1.666666666666666788791199376

= 0.1034958575819811507967930501

= -5.587523805488E-16

Оценка погрешности для узлов Чебышева:

Формула:

**Вывод:**

Результаты остатка интерполирования на узлах Чебышева соответствуют теоретической оценке.

Результаты из первого задания для сравнения:

= 0.7666666666666667332800481442

= 0.5412594525687910809826672981

= 2.7242523250027E-15

= 1.250000000000000097144514655

= 0.2951500665207106359871162374

= -1.47773500326E-17

= 1.666666666666666788791199376

= 0.1034958575784758451236556845

= -7.6821882246314E-15

Можно заметить, что реальная погрешность уменьшилась для всех узлов кроме x\*\*\*,

**Листинг**

Построение узлов Чебышева:

def calculateChebyshevNodes(a: Decimal, b: Decimal) -> list:

a\_b\_sum = (a + b) / 2

a\_b\_diff = (b - a) / 2

x = [Decimal(0.0) for \_ in range(N + 1)]

for i in range(N + 1):

x[i] = a\_b\_sum + (a\_b\_diff \* Decimal(math.cos(math.pi \*

(i + 1 / 2) / (N + 1))))

x.reverse()

return x

Создание таблицы разделённых разностей(код такой же как в первом пункте):

def createDDTable(x: list, f: list) -> list:

dd = [[Decimal(0.0) for \_ in range(N + 1)] for \_ in range(N + 1)]

for i in range(0, N + 1):

dd[i][0] = f[i]

for i in range(1, N + 1):

for j in range(N - i + 1):

dd[j][i] = (dd[j + 1][i - 1] - dd[j][i - 1]) / (x[i + j] - x[j])

return dd

Функция для построения интерполяционного многочлена Ньютона(код из первого пункта):

def newton(x, nodes: list, dd\_table: list) -> Decimal:

result = Decimal(0.0)

temp = Decimal(0.0)

for i in range(N + 1):

temp = dd\_table[0][i]

for j in range(i):

temp \*= (x - nodes[j])

result += temp

return result

Получение точных значений функции в Чебышевских узлах:

chebyshev\_values = [f(chebyshev\_nodes[i]) for i in range(N + 1)]

Функция производной 11го порядка:

def f\_11\_derivative(x) -> Decimal:

return (-VARIANT\_COEF) \* Decimal.exp(-x) + (1 - VARIANT\_COEF) \* Decimal(math.sin(x))

Подсчёт погрешности для Чебышевского набора узлов:

def chebyshevTheoreticalError(a: Decimal, b: Decimal) -> Decimal:

a\_b\_diff = (b - a) / 2

for i in range(1, N + 1):

a\_b\_diff \*= ((b - a) / (4 \* (i + 1)))

return f\_11\_derivative(b) \* a\_b\_diff

Вызов функций для узлов Чебышева:

chebyshev\_nodes = calculateChebyshevNodes(x[0], x[N])

chebyshev\_values = [f(chebyshev\_nodes[i]) for i in range(N + 1)]

dd\_chebishev = createDDTable(chebyshev\_nodes, chebyshev\_values)

y\_1 = newton(x\_1, chebyshev\_nodes, dd\_chebishev)

y\_2 = newton(x\_2, chebyshev\_nodes, dd\_chebishev)

y\_3 = newton(x\_3, chebyshev\_nodes, dd\_chebishev)

Таблица разделённых разностей:

5.743e-1 -5.380e-1 5.519e-2 -1.657e-2 2.067e-2 -4.322e-3 1.348e-4 1.379e-6 9.234e-6 -1.365e-6 2.860e-8

5.527e-1 -5.315e-1 5.147e-2 -9.254e-3 1.853e-2 -4.237e-3 1.359e-4 9.438e-6 7.938e-6 -1.336e-6

5.118e-1 -5.220e-1 4.856e-2 -8.247e-4 1.601e-2 -4.138e-3 1.437e-4 1.666e-5 6.669e-6

4.557e-1 -5.105e-1 4.825e-2 7.480e-3 1.333e-2 -4.030e-3 1.576e-4 2.248e-5

3.896e-1 -4.975e-1 5.133e-2 1.469e-2 1.072e-2 -3.915e-3 1.748e-4

3.195e-1 -4.830e-1 5.737e-2 2.024e-2 8.383e-3 -3.804e-3

2.515e-1 -4.675e-1 6.502e-2 2.406e-2 6.501e-3

1.910e-1 -4.521e-1 7.257e-2 2.636e-2

1.424e-1 -4.387e-1 7.849e-2

1.086e-1 -4.295e-1

9.139e-2

Среднеквадратичное приближение.

**Постановка задачи:**

На заданной таблице из первой части построить полином наилучшего среднеквадратичного приближения 5 степени. Найти приближенные значения в . Оценить погрешность. Сравнить значения с расчетом по методу Лагранжа.

В рамках этой части

Для приближения используем алгебраический полином:

Коэффициенты находим, решив систему:

Найти оценку сверху для среднеквадратичного метода можно по формуле:

**Результаты**

= 0.7666666666666667332800481442

= 0.5412595043862207235217609310

= -5.18174857735926957501E-8

= 1.250000000000000097144514655

= 0.2951500718629586774458438639

= -5.3422450264719559757E-9

= 1.666666666666666788791199376

= 0.1034954220335559624293869544

= 4.355484246296150255469E-7

= 9.211962905877545357296454468E-7

Можно увидеть, что реальная погрешность соответствует оценке сверху для всех точек восстановления .

**Листинг:**

Генерация СЛАУ:

def makeSLU(x: list, f: list) -> tuple:

A = [[float(0.0) for \_ in range(m + 1)] for \_ in range(m + 1)]

b = [float(0.0) for \_ in range(m + 1)]

for i in range(m + 1):

for j in range(m + 1):

for k in range(N):

A[i][j] += float(np.power(x[k], i + j))

for j in range(N):

b[i] += float(f[j]) \* float(np.power(x[j], i))

return A, b

Нахождение коэффицентов для построения многочлена:

def findCoef(A: list, b: list) -> list:

return list(np.linalg.solve(A, b))

Построение многочлена:

def lsm(x: Decimal, coefs: list) -> Decimal:

result = Decimal(0.0)

for i in range(m + 1):

result += Decimal(coefs[i]) \* Decimal(x \*\* i)

return result

Оценка погрешности:

def lsmError(x: list, coefs: list) -> Decimal:

sum = Decimal(0.0)

for i in range(N + 1):

sum += (lsm(x[i], coefs) - f(x[i])) \*\* 2

return Decimal.sqrt(sum)

Вызов функций:

A, b = makeSLU(x, fx)

coefs = findCoef(A, b)

y\_1 = lsm(x\_1, coefs)

y\_2 = lsm(x\_2, coefs)

y\_3 = lsm(x\_3, coefs)