Министерство образования и науки 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 (ИУ)

КАФЕДРА Программное обеспечение ЭВМ и информационные технологии (ИУ7)

Лабораторная работа №1

Тема: Построение и программная реализация алгоритма полиномиальной интерполяции табличных функций

Студент: Хамзина Р.Р.

Группа: ИУ7-43Б

Оценка (баллы): _________

Преподаватель: Градов В.М.

Москва.
2020 г.

Цель работы: получение навыков построения алгоритма интерполяции таблично заданных функций полиномами Ньютона и Эрмита.

1 Исходные данные

1. Таблица функции и её производных.

x	y	y'
x_0	y_0	y'_0
x_1	y_1	y'_2
...	...	...

Для отладки:

x	y	y'
0.00	1.000000	-1.000000
0.15	0.838771	-1.14944
0.30	0.655336	-1.29552
0.45	0.450447	-1.43497
0.6	0.225336	-1.56464
0.75	-0.018310	-1.68164
0.9	-0.278390	-1.78333
1.05	-0.552430	-1.86742

2. Степень аппроксимирующего полинома.

Для полинома Ньютона $P_n(x) = y_0 + \sum_{k=0}^{n} (x - x_n) \dots (x - x_{k-1}) y(x_0, x_1, \dots, x_k)$ n — степень.

Для полинома Эрмита $H_n(x) = P_n(x, \underbrace{x_0, x_0, \dots, x_0}_{n_0}, x_1, x_1, \dots, x_1, x_2, x_2, \dots, x_2, \underbrace{x_k, x_k, \dots, x_k}_{n_k}),$ n — степень.

Для отладки: n = 1, 2, 3, 4.

3. Значение аргумента x, для которого выполняется интерполяция.

Для отладки: x = 0.525.

2 Код программы

Код программы представлен на листингах 1-3.

Листинг 1 — NewtonInterpolation.py

```
COUPLE = 2

def SortTable(Table, SizeTable):
    """
        Сортировка таблицы по возрастанию.
    """

    for i in range(SizeTable - 1):
        MinIndex = i
        for j in range(i + 1, SizeTable):
            if Table[j][0] < Table[MinIndex][0]:
                MinIndex = j

        Table[MinIndex], Table[i] = Table[i], Table[MinIndex]
    return Table

def CreateConfig(Table, SizeTable, power, argument):
    """
        Построение конфигурации узлов из таблицы Table
        размера SizeTable для построение полинома степени
        power при аргументе argument.
    """
    center = 0

    while center < SizeTable:
        if Table[center][0] >= argument:
            break
        center += 1

    if center == 0:
        return Table[:power + 1]

    if center == SizeTable:
        return Table[SizeTable - power - 1:]
```

```python
    if abs(Table[center][0] - argument) > abs(argument - Table[center - 1][0]):
        center -= 1

    low = center - power // 2 - 1
    top = center + power // 2 + 1

    if power % 2 == 0:
        return Table[center - power // 2:top]

    if abs(Table[top][0] - argument) > abs(argument - Table[low][0]):
        return Table[low:top]

    return Table[low: top + 1]

def CreateSplitDiff(Table, power):
    """
        Построение таблицы разделенных разностей.
        Параметры выбираются из таблицы Table,
        степень полинома power.
    """

    SplitDiff = []
    diffs = []

    for i in range(power + 1):
        diffs.append(Table[i][1])

    SplitDiff.append(diffs)

    for i in range(power):
        size = len(SplitDiff)
        diffs = []
        DiffX = Table[0][0] - Table[i + 1][0]

        for j in range(1, len(SplitDiff[size - 1])):
            DiffY = SplitDiff[size - 1][j - 1] - SplitDiff[size - 1][j]
            diffs.append(DiffY/DiffX)

        SplitDiff.append(diffs)
```

```
    return SplitDiff


def NewtonPolynomial(Config, power, argument, SplitDiff):
    """
        Получение значения интерполяционного полинома
        Ньютона степени power при аргументе argument.
        Начальная конфигурация Config, таблица разде-
        ленных разностей SplitDiff.
    """

    result = SplitDiff[0][0]
    factor = 1

    for i in range(power):
        factor *= argument - Config[i][0]
        result += SplitDiff[i + 1][0] * factor

    return result


def NewtonInterpolation(Table, SizeTable, power, argument):
    """
        Значение интерполяцинного полинома Ньютона
        при заданной степени power и аргументе argument.
        Параметры выбираются из таблицы Table размером
        SizeTable.
    """

    Table = SortTable(Table, SizeTable)
    Config = CreateConfig(Table, SizeTable, power, argument)
    SplitDiff = CreateSplitDiff(Config, power)
    result = NewtonPolynomial(Config, power, argument, SplitDiff)

    return result
```

Листинг 2 — HermitInterpolation.py

```
from NewtonInterpolation import SortTable

def CreateConfig(Table, SizeTable, power, argument):
    """
        Построение конфигурации узлов из таблицы Table
```

```python
        размера SizeTable для построение полинома степени
        power при аргументе argument.
    """
    center = 0

    while center < SizeTable:
        if Table[center][0] >= argument:
            break
        center += 1

    CountDerivs = (power + 1) // 2
    CountValues = power + 1 - CountDerivs

    if center == 0:
        return Table[:CountValues + 1]

    if center == SizeTable:
        return Table[SizeTable - CountValues - 1:]

    if abs(Table[center][0] - argument) > abs(argument - Table[center - 1][0]):
        center -= 1

    if (CountValues - 1) % 2 == 0:
        return Table[center - CountValues // 2:center + CountValues // 2 + 1]

    low = center - (CountValues - 1) // 2 - 1
    top = center + (CountValues - 1) // 2 + 1

    if abs(Table[top][0] - argument) > abs(argument - Table[low][0]):
        return Table[low:top]

    return Table[low: top + 1]

def CreateSplitDiff(Table, power):
    """
        Построение таблицы разделенных разностей.
        Параметры выбираются из таблицы Table,
        степень полинома power.
    """
```

```
    CountDerivs = (power + 1) // 2
    CountValues = power + 1 - CountDerivs

    SplitDiff = []
    diffs = []

    for i in range(CountValues):
        diffs.append(Table[i][1])

    SplitDiff.append(diffs)
    diffs = []

    j = 0
    for i in range(power):
        if i % 2 == 0:
            diffs.append(Table[j][2])
            j += 1
        else:
            DiffX = Table[j - 1][0] - Table[j][0]
            DiffY = SplitDiff[0][j - 1] - SplitDiff[0][j]
            diffs.append(DiffY/DiffX)
    SplitDiff.append(diffs)

    for i in range(power - 1):
        size = len(SplitDiff)
        diffs = []

        DiffX = Table[0][0] - Table[i // 2 + 1][0]

        for j in range(1, len(SplitDiff[size - 1])):
            DiffY = SplitDiff[size - 1][j - 1] - SplitDiff[size - 1][j]
            diffs.append(DiffY/DiffX)

        SplitDiff.append(diffs)

    return SplitDiff

def HermitPolynomial(Config, power, argument, SplitDiff):
    """
        Получение значения интерполяционного полинома
```

```
        Эрмита степени power при аргументе argument.
        Начальная конфигурация Config, таблица разде-
        ленных разностей SplitDiff.
    """
    result = SplitDiff[0][0]
    factor = 1

    j = 0
    for i in range(power):
        if i % 2 == 0:
            factor *= argument - Config[j][0]
        else:
            factor *= argument - Config[j][0]
            j += 1

        result += SplitDiff[i + 1][0] * factor

    return result

def HermitInterpolation(Table, SizeTable, power, argument):
    """
        Значение интерполяцинного полинома Эрмита
        при заданной степени power и аргументе argument.
        Параметры выбираются из таблицы Table размером
        SizeTable.
    """
    Table = SortTable(Table, SizeTable)
    Config = CreateConfig(Table, SizeTable, power, argument)
    SplitDiff = CreateSplitDiff(Config, power)
    result = HermitPolynomial(Config, power, argument, SplitDiff)

    return result
```

Листинг 3 — SearchRoot.py

```
from NewtonInterpolation import NewtonInterpolation

CHANGE = True
NOT_CHANGE = False

def ChangeColumns(Table, SizeTable):
```

```python
    """
        Смена местами значений x и y таблицы Table
        размера SizeTable.
    """
    for i in range(SizeTable):
        Table[i][0], Table[i][1] = Table[i][1], Table[i][0]

    return Table


def SignsChange(Table, SizeTable):
    """
        Проверка функции на смену знака.
    """
    Positives, Negatives, Zeros = 0, 0, 0

    for i in range(SizeTable):
        if Positives and Negatives or Zeros:
            return CHANGE

        if Table[i][1] < 0:
            Negatives = 1
        elif Table[i][1] > 0:
            Positives = 1
        else:
            Zeros = 1

    return NOT_CHANGE


def SearchRoot(Table, SizeTable, power, root):
    """
        Поиск корня заданной табличной функции
        с помощью обратной интерполяции, используя
        полином Ньютона.
    """
    if not SignsChange(Table, SizeTable):
        print("\nFunction has no root\n")
        return
```

```
    Table = ChangeColumns(Table, SizeTable)
    result = NewtonInterpolation(Table, SizeTable, power, root)

    return result
```

3 Результаты работы

Для полинома Ньютона:

1. Сортируем таблицу в порядке возрастания/убывания.

2. Из заданной таблицы Table строим конфигурацию узлов Config, которые примыкают к заданному значению аргумента x:

x	y
x_0	y_0
x_1	y_1
...	...

Число узлов: n + 1, n — степень полинома.

3. Из конфигурации узлов Config строим таблицу разделенных разностей ($y(x_i, x_j) = (y_i — y_j) / (x_i — x_j)$) SplitDiff:

$y(x_i, x_j) = (y_i — y_j) / (x_i — x_j)$

$y(x_i, x_j, x_k) = (y(x_i, x_j) — y(x_j, x_k)) / (x_i — x_k)$

…

<table>
<tr><td>y_0</td><td rowspan="2">$(y_0 — y_1) / (x_0 — x_1)$</td><td rowspan="3">$(y(x_0, x_1) — y(x_1, x_2)) / (x_0 — x_2)$</td></tr>
<tr><td>y_1</td></tr>
<tr><td>y_2</td><td>$(y_1 — y_2) / (x_1 — x_2)$</td></tr>
</table>

4. Выбираем значения из первой строки каждого столбца таблицы разделенных разностей и для значения аргумента x находим полином Ньютона степени n:

$$P_n(x) = y_0 + \sum_{k=0}^{n} (x - x_n) \dots (x - x_{k-1}) y(x_0, x_1, \dots, x_k)$$

Для нахождения корня заданной табличной функции:

1. Меняем местами последовательности аргумента функции и значения функции.

y	x
y_0	x_0
y_1	x_1
...	...

2. Применяем алгоритм построения интерполяционного полинома Ньютона, описанный выше.

Для полинома Эрмита:

1. Сортируем таблицу в порядке возрастания/убывания.

2. Из заданной таблицы Table строим конфигурацию узлов и производных Config, которые примыкают к заданному значению аргумента x:

x_i	y_i	y_i'
x_0	y_0	y_0'
x_1	y_1	y_1'
x_2	y_2	y_2'

Число производных — (n + 1) / 2, число узлов — n — число производных, n — степень полинома.

3. Из конфигурации узлов Config строим таблицу разделенных разностей SplitDiff следующим образом:

x_i	y_i	$y(x_k, x_m)$	$y(x_k, x_m, x_l)$
x_0	y_0	$y_0^{'}$	$(y_0^{'} - y(x_0, x_1))/(x_0 - x_1)$
x_0	y_0	$y(x_0, x_1)$	$(y(x_0, x_1) - y_1^{'})/(x_0 - x_1)$
x_1	y_1	$y_1^{'}$	и.т.д.
x_1	y_1	$y(x_1, x_2)$	
x_2	y_2	$y_2^{'}$	
x_2	y_2		

4. Выбираем значения из первой строки каждого столбца таблицы разделенных разностей и для значения аргумента x находим полином Эрмита степени n:

$$P_n(x, x_0, x_0, x_1, x_1) = y(x_0) + (x - x_0)\, y(x_0, x_0) + (x - x_0)^2\, y(x_0, x_0, x_1) +$$
$$+ (x - x_0)^2 (x - x_1)\, y(x_0, x_0, x_1, x_1) =$$
$$= y(x_0) + (x - x_0)\, y'(x_0) + (x - x_0)^2 \frac{y'(x_0) - y(x_0, x_1)}{x_0 - x_1} +$$
$$+ [y'(x_0) - 2y(x_0, x_1) + y'(x_1)] \frac{(x - x_0)^2 (x - x_1)}{(x_0 - x_1)^2}.$$

Степень n	Значение полинома Ньютона	Значение полинома Эрмита	Корень табличной функции
1	0.337891	0.342684	0.739440
2	0.340208	0.340358	0.739440
3	0.340314	0.340323	0.747619
4	0.340324	0.340334	0.747619

4 Вопросы при защите лабораторной работы

1. Будет ли работать программа при степени полинома n=0?

При степени полинома n = 0 программа будет работать:

Таблица разделенных разностей будет иметь вид:

y
y_0

Тогда полином будет равен:

$P_0(x) = y_0$

Для заданной для отладки таблицы, при аргументе x = 0.525 и степени n = 0, таблица разделенных разностей и полином:

y
0.225336

$P_0(0.525) = 0.225336$

2. Как практически оценить погрешность интерполяции? Почему сложно применить для этих целей теоретическую оценку?

Практически оценить погрешность интерполяции можно при помощи первого отброшенного члена: при маленьком шаге погрешность близка к первому отброшенному члену.

Погрешность интерполяции теоретически оценивается по формуле, в которой используются производные, например для полинома Ньютона:

$$\left| y(x) - P_n(x) \right| \le \frac{M_{n+1}}{(n+1)!} \left| \varpi_n(x) \right|,$$

где M_{n+1} — это максимальное значение производной функции, к которой применяется интерполяция на отрезке между минимальным и максимальным значением x. Производные функции не всегда известны, поэтому применить теоретическую оценку погрешности сложно.

3. Если в двух точках заданы значения функции и ее первых производных, то полином какой минимальной степени может быть построен на этих точках?

По условию дана таблица:

x	y	y'
x_0	y_0	y'_0

x_1	y_1	y'_1

Исходя из условия можно построить полином (как Ньютона, так и Эрмита) 0-ой степени $P_0(x) = y_0$, поэтому минимальная степень при заданных условиях — 0.

4. В каком месте алгоритма построения полинома существенна информация об упорядоченности аргумента функции (возрастает, убывает)?

Начальная конфигурация данных выбирается из примыкающих к аргументу значений аргументов таблицы, поэтому на этапе выбора конфигурации данных важно знать, упорядочены аргументы функции или нет. Если они не упорядочены, то необходимо отсортировать последовательность значений аргументов.

5. Что такое выравнивающие переменные и как их применить для повышения точности интерполяции?

Можно преобразовать переменные так, что в новых переменных ($n = n(x)$, $m = m(y)$) график функции $m(n)$ будет близок к прямой. Новые (преобразованные) переменные называют выравнивающими.

Выравнивающие переменные применяют так: пусть даны последовательности выравнивающих переменных ($n = n(x)$, $m = m(y)$).
1. Проводят интерполяцию в переменных (n, m).
2. Обратным интерполированием находят $y = y(m)$.