|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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 **Программное обеспечение ЭВМ и информационные технологии**

**Лабораторная работа №1.**

**«Построение и программная реализация алгоритма полиномиальной интерполяции табличных функций»**

Студент **Леонов Владислав Вячеславович**

Группа **ИУ7-46Б**

Студент **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** Леонов В.В.

*подпись, дата фамилия, и.о.*

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Градов В.М.

*подпись, дата фамилия, и.о.*

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Москва, 2021 г.*

**Оглавление**

[Цель работы 3](#_Toc64074185)

[Исходные данные 3](#_Toc64074186)

[Описание алгоритма 3](#_Toc64074187)

[Код программы 4](#_Toc64074188)

[Результаты работы 7](#_Toc64074189)

[Ответы на контрольные вопросы 8](#_Toc64074190)

# Цель работы

Построение и программная реализация полиномиальной интерполяции табличных функций.

# Исходные данные

1. Таблица функции и ее производных

|  |  |  |
| --- | --- | --- |
| x | y | y‘ |
| 0.00 | 1.000000 | -1.00000 |
| 0.15 | 0.838771 | -1.14944 |
| 0.30 | 0.655336 | -1.29552 |
| 0.45 | 0.450447 | -1.43497 |
| 0.60 | 0.225336 | -1.56464 |
| 0.75 | -0.018310 | -1.68164 |
| 0.90 | -0.278390 | -1.78333 |
| 1.05 | -0.552430 | -1.86742 |

2. Степень аппроксимирующего полинома – n.

3. Значение аргумента, для которого вычисляется интерполяция.

# Описание алгоритма

Для решения поставленной задачи используется построение **полинома Ньютона** и **полинома Эрмита**. Удобно ввести понятие разделенной разности:

*…*

Правило образования таких конструкций понятно из приведенной выше записи. Тогда в результате получаем так называемый полином Ньютона, который имеет следующий вид:

Построение полинома Эрмита происходит аналогичных образом, за исключением того, что в таблице исходных данных происходит дублирование точек и когда производится вычисление разделенной разности вида , то используется известное значение производной функции.

В свою очередь для отыскания приближенного значения корня для монотонной функции используется понятие **обратной интерполяции**, суть которой состоит в том, что в таблице функции меняются местами столбцы и выполняется обычная интерполяция при задании аргумента, равного 0. Факт наличия корня у функции устанавливается по наличию смены знака функции.

# Код программы

Код программы представлен ниже.

|  |
| --- |
| Файл ***main.py*** |
| **import** math  **from** newtone **import** newtone\_perform  **from** hermite **import** hermite\_perform  **def** main():  mode = int(input("1)Newtone\n2)Hermite\nMode:"))  **if** (mode == 1):  newtone\_perform()  **if** (mode == 2):  hermite\_perform()  **if** (\_\_name\_\_ == "\_\_main\_\_"):  main() |

|  |
| --- |
| Файл ***newtone.py*** |
| **def** newtone\_data\_read(filename):  data = []  f = open(filename, "r")  **for** line **in** f:  line = (line.replace("\n", "")).split(" ")  line[0] = float(line[0])  line[1] = float(line[1])  data.append(line)  f.close()  **print**("Data was read.")  **return** data  **def** newtone\_data\_print(data):  **print**("|{0:^10}|{1:^10}|".format("x", "y"))  **for** row **in** data:  **print**("|{0:^10}|{1:^10}|".format(row[0], row[1]))  **def** newtone\_find\_dots(data, polinom\_degree, func\_arg):  **for** i **in** range(len(data)-1):  **if** (data[i][0] < func\_arg **and** data[i+1][0] > func\_arg):  index = i  **break**  curr\_dots = 0  dots = []  step = 0  **while** curr\_dots <= polinom\_degree:  **if** (index - step >= 0):  dots.insert(0, data[index-step])  curr\_dots += 1  **if** (index + 1 + step < len(data) **and** curr\_dots <= polinom\_degree):  dots.append(data[index+1+step])  curr\_dots += 1  step += 1  **return** dots  **def** newtone\_func\_value(data, polinom\_degree, func\_arg):  data = newtone\_find\_dots(data, polinom\_degree, func\_arg)  coefficients = []  curr\_col = []  curr\_col\_i = 1  **for** row **in** data:  curr\_col.append(row[1])  coefficients.append(curr\_col[0])  new\_col = []  **while** (len(curr\_col) > 1):  **for** i **in** range(len(curr\_col)-1):  new\_col.append((curr\_col[i]-curr\_col[i+1]) /  (data[i][0]-data[i+curr\_col\_i][0]))  **if** (i == 0):  coefficients.append(new\_col[0])  curr\_col\_i += 1  curr\_col = new\_col.copy()  new\_col.clear()  **print**("Coefficients list: ", \*coefficients)  func\_value = coefficients[0]  curr\_x = 1  **for** i **in** range(len(data)-1):  curr\_x \*= (func\_arg - data[i][0])  func\_value += coefficients[i+1]\*curr\_x  **return** func\_value  **def** newtone\_func\_root(data, polinom\_degree):  **for** row **in** data:  row[0], row[1] = row[1], row[0]  data.sort()  func\_root = newtone\_func\_value(data, polinom\_degree, 0.0)  **return** func\_root  **def** newtone\_perform():  filename = input("Data file:")  data = newtone\_data\_read(filename)  newtone\_data\_print(data)  polinom\_degree = int(input("Polinom degree:"))  func\_arg = float(input("Function argument:"))  func\_value = newtone\_func\_value(data, polinom\_degree, func\_arg)  **print**("Newtone polinom value = ", func\_value)  func\_root = newtone\_func\_root(data, polinom\_degree)  **print**("Func root = ", func\_root) |

|  |
| --- |
| Файл ***hermite.py*** |
| **def** hermite\_data\_read(filename):  data = []  f = open(filename, "r")  **for** line **in** f:  line = (line.replace("\n", "")).split(" ")  line[0] = float(line[0])  line[1] = float(line[1])  line[2] = float(line[2])  data.append(line)  f.close()  **print**("Data was read.")  **return** data  **def** hermite\_data\_print(data):  **print**("|{0:^10}|{1:^10}|{2:^10}|".format("x", "y", "y\'"))  **for** row **in** data:  **print**("|{0:^10}|{1:^10}|{2:^10}|".format(row[0], row[1], row[2]))  **def** hermite\_find\_dots(data, polinom\_degree, func\_arg):  **for** i **in** range(len(data)-1):  **if** (data[i][0] < func\_arg **and** data[i+1][0] > func\_arg):  index = i  **break**  curr\_dots = 0  dots = []  step = 0  **while** curr\_dots <= polinom\_degree:  **if** (index - step >= 0):  dots.insert(0, data[index-step])  curr\_dots += 1  **if** (curr\_dots <= polinom\_degree):  dots.insert(0, data[index-step])  curr\_dots += 1  **if** (index + 1 + step < len(data) **and** curr\_dots <= polinom\_degree):  dots.append(data[index+1+step])  curr\_dots += 1  **if** (curr\_dots <= polinom\_degree):  dots.append(data[index+1+step])  curr\_dots += 1  step += 1  **return** dots  **def** hermite\_func\_value(data, polinom\_degree, func\_arg):  data = hermite\_find\_dots(data, polinom\_degree, func\_arg)  coefficients = []  curr\_col = []  curr\_col\_i = 1  **for** row **in** data:  curr\_col.append(row[1])  coefficients.append(curr\_col[0])  new\_col = []  **while** (len(curr\_col) > 1):  **for** i **in** range(len(curr\_col)-1):  **if** (data[i][0] - data[i][0] < 1e-6):  new\_col.append(data[i//2][2])  **else**:  new\_col.append((curr\_col[i]-curr\_col[i+1]) /  (data[i][0]-data[i+curr\_col\_i][0]))  **if** (i == 0):  coefficients.append(new\_col[0])  curr\_col\_i += 1  curr\_col = new\_col.copy()  new\_col.clear()  **print**("Coefficients list: ", \*coefficients)  func\_value = coefficients[0]  curr\_x = 1  **for** i **in** range(len(data)-1):  curr\_x \*= (func\_arg - data[i][0])  func\_value += coefficients[i+1]\*curr\_x  **return** func\_value  **def** hermite\_perform():  filename = input("Data file:")  data = hermite\_data\_read(filename)  hermite\_data\_print(data)  polinom\_degree = int(input("Polinom degree:"))  func\_arg = float(input("Function argument:"))  func\_value = hermite\_func\_value(data, polinom\_degree, func\_arg)  **print**("Hermite polinom value = ", func\_value) |

# Результаты работы

1. Значения функции при степенях полиномов Ньютона и Эрмита при n = 1, 2, 3, 4 при фиксированном x = 0.525. Результаты свести в таблицу.

|  |  |  |
| --- | --- | --- |
| Полином Ньютона | Полином Эрмита | Степень полинома |
| 0.337891 | 0.342824 | 1 |
| 0.340419 | 0.334752 | 2 |
| 0.340313 | 0.335357 | 3 |
| 0.340324 | 0.340465 | 4 |

2. Найти корень заданной табличной функции с помощью обратной интерполяции, используя полином Ньютона.

# Ответы на контрольные вопросы

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

Да, будет. При вычислении значения функции в точке программа выведет значение функции в меньшей ближайшей точке (значение, которое задано в таблице данных для интерполяции).

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

На практике точность расчетов удобно оценивать, наблюдая за тем, насколько быстро убывают члены ряда, составляющего полином. Если это происходит достаточно быстро, можно оставлять только те члены, которые больше заданной погрешности расчётов.

Теоретическую погрешность можно оценить по формуле:

где – максимальное значение производной интерполируемой функции на отрезке, .

Трудность использования указанных теоретических оценок на практике состоит в том, что производные обычно неизвестны, а также производные высших порядков могут быть трудно представимы в виде, который использует компьютер для работы с действительными числами (выход за границы разрядной сетки).

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

При данных условиях можно построить максимально полином 3 степени (кубический).

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

При поиске точек для интерполяции и построения полинома Ньютона или полинома Эрмита удобно, если данные будут отсортированы по возрастанию или убыванию, что позволит сократить время, затрачиваемое на данную операцию.

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

При использовании выравнивающих переменных можно добиться того, чтобы график в новых переменных был близок к прямой на отдельных участках, что упрощает работу с функцией. Как правило преобразования для выравнивающих стараются выбрать довольно простыми и позаботиться о том, чтобы и обратное преобразование было довольно простым.