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

**Тема:** Построение и программная реализация алгоритма полиномиальной интерполяции табличных функций.

**Студент:**Романов Семен

**Группа:** ИУ7-45Б

**Оценка (баллы):** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Преподаватель:**Градов Владимир Михайлович

*Москва*

*2022 г*

**Цель работы:**

Получение навыков построения алгоритма интерполяции таблично заданных функций полиномами Ньютона и Эрмита.

**Задачи:**

1. Значения y(x) при степенях полиномов Ньютона n= 1, 2, 3, 4 и 5 при фиксированном x, например, x=0.525 (середина интервала 0.45- 0.60).
2. Значения y(x) при интерполяции полиномом Эрмита по одному, 2-м и 3-м узлам при том же, что и в п.1 фиксированном x, например, x=0.525.
3. Результаты свести в таблицу для сравнения полиномов Ньютона и Эрмита одинаковых степеней.
4. Найти корень заданной выше табличной функции с помощью обратной интерполяции, используя полином Ньютона.

**Исходные данные**

1. Таблица функции и её производных

|  |  |  |
| --- | --- | --- |
| X | Y | Y` |
| 0 | 1 | -1 |
| 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 Ньютона **n**

3. Значение аргумента, для которого выполняется интерполяция.

**Код программы**

main.py

from modules.io import \*

from modules.newton import \*

from modules.hermite import \*

from modules.basic\_func import \*

import os

DIR\_NAME = os.path.dirname(\_\_file\_\_)

DATA\_PATH = 'data/table.txt'

DIR\_DATA = os.path.join(DIR\_NAME, DATA\_PATH)

MAX\_POWER = 5

MIN\_POWER = 0

class Table:

def \_\_init\_\_(self) -> None:

self.rows = 0

self.x = []

self.y = []

self.first\_derivative = []

def main():

table = Table()

fill\_table(DIR\_DATA, table)

results\_Newton = []

results\_Hermite = []

print\_first\_exercise()

x = input\_float()

print\_row('n', 'x', "y(x) [Newton's pol.]")

for power in range(MIN\_POWER, MAX\_POWER + 1):

polynom = NewtonPolynom(power, table, x)

results\_Newton.append([power, polynom])

print\_row(power, x, polynom)

print()

print\_second\_exercise()

print\_row\_b('n', "Nodes", 'x', "y(x) [Hermite's pol.]")

for power in range(MIN\_POWER, MAX\_POWER + 1):

polynom = HermitePolynom(power, table, x)

results\_Hermite.append([power, polynom])

print\_row\_b(power, (power)//2 + 1, x, polynom)

print()

print\_third\_exercise()

print\_row\_b('n', 'x', "y(x) [Newton's pol.]", "y(x) [Hermite's pol.]")

for power in range(len(results\_Hermite)):

print\_row\_b(results\_Newton[power][0], x, results\_Newton[power][1], results\_Hermite[power][1])

print()

print\_forth\_exercise()

y = input\_float()

reversed\_table = Table()

reversed\_table.x = table.y

reversed\_table.y = table.x

reversed\_table.rows = table.rows

reversed\_table.first\_derivative = None

print\_row('n', 'y', "x(y) [Newton's pol.]")

for power in range(MIN\_POWER, MAX\_POWER + 1):

polynom = NewtonPolynom(power, reversed\_table, y)

print\_row(power, y, polynom)

print("Success!")

return 0

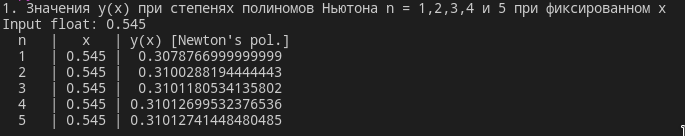
if \_\_name\_\_ == "\_\_main\_\_":

main()

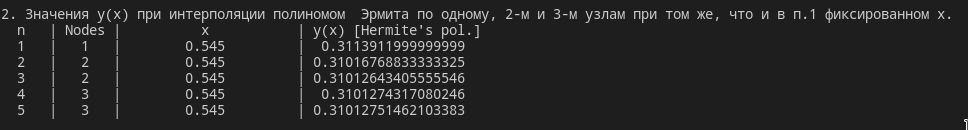
|  |
| --- |
| io.py |
| def input\_float():  '''  Ввод числа типа float  '''  flag = True  while flag:  x = input("Input float: ")  try:  x = float(x)  flag = False  except:  print("Error! Try again...")    return x  def fill\_table(filename, table):  '''  Считывание таблицы из текстового файла в объект Table  '''  f = open(filename, 'r')  for row in f:  row = list(map(float, row.split()))  table.rows += 1  table.x.append(row[0])  table.y.append(row[1])  table.first\_derivative.append(row[2])  f.close()  def print\_row(first\_col, second\_col, third\_col):  print("{:^5} | {:^5} | {:^20}".format(first\_col, second\_col, third\_col))  def print\_row\_b(first\_col, second\_col, third\_col, forth\_col):  print("{:^5} | {:^5} | {:^20} | {:^20}".format(first\_col, second\_col, third\_col, forth\_col))  def print\_first\_exercise():  print("1. Значения y(x) при степенях полиномов Ньютона n = 1,2,3,4 и 5 при фиксированном x")  def print\_second\_exercise():  print("2. Значения y(x) при интерполяции полиномом Эрмита по одному, 2-м и 3-м узлам при том же, что и в п.1 фиксированном x.")  def print\_third\_exercise():  print("3. Результаты свести в таблицу для сравнения полиномов Ньютона и Эрмита одинаковых степеней")  def print\_forth\_exercise():  print("4. Найти корень заданной выше табличной функции с помощью обратной интерполяции, используя полином Ньютона. ")  basic\_func.py  def find\_interval(table, power, arg):  '''  Возвращает начальный и конечный индексы, наиболее близких к аргументу  '''  idx = idx\_start = idx\_end = 1    while idx < table.rows and not table.x[idx] <= arg <= table.x[idx-1] and not table.x[idx] >= arg >= table.x[idx-1]:  idx += 1    if idx == table.rows:  if (arg >= table.x[0] >= table.x[1] or arg <= table.x[0] <= table.x[1]):  idx\_start = 0  idx\_end = power  else:  idx\_end = idx - 1  idx\_start = idx\_end - power  else:  idx\_end = min(table.rows - 1, idx + power // 2)  idx\_start = max(idx\_end - power, 0)  if idx\_end - idx\_start != power:  idx\_end = idx\_start + power  return (idx\_start, idx\_end)  hermite.py  from .basic\_func import \*  def HermitePolynom(power, table, x):  '''  Интерполяционный полином Эрмита  '''  idx\_start, idx\_end = find\_interval(table, (power) // 2, x)  result\_polynom = table.y[idx\_start]  idxs = [idx\_start]  new\_idx = idx\_start  cur = 1  for i in range(power):  idxs.append(new\_idx)  cur \*= (x - table.x[new\_idx - i % 2])  result\_polynom += NewtonForHermite(table, idxs)\*cur  new\_idx += (i + 1) % 2  return result\_polynom  def NewtonForHermite(table, idxs):  '''  Вычисление полинома Ньютона по узлам для интерполяционного полинома Эрмита  '''  if len(idxs) == 2:  if idxs[0] == idxs[1]:  return table.first\_derivative[idxs[0]]  else:  return (table.y[idxs[0]] - table.y[idxs[1]])/(table.x[idxs[0]] - table.x[idxs[1]])  else:  return (NewtonForHermite(table, idxs[:-1]) - NewtonForHermite(table, idxs[1:]))/(table.x[idxs[0]] - table.x[idxs[-1]])  newton.py  from .basic\_func import \*  def NewtonPolynom(power, table, arg):  '''  Интерполяционный полином Ньютона  '''  interval = find\_interval(table, power, arg)  polynom\_table = []  for i in range(1, power + 1):  fill\_polynom(polynom\_table, i, table, interval)  idx\_start, idx\_end = interval  diffs = []  cur = 1  for i in range(idx\_start, idx\_end + 1):  cur \*= arg - table.x[i]  diffs.append(cur)  polynom\_result = table.y[idx\_start]  for i in range(power):  cur = polynom\_table[i][0] \* diffs[i]  polynom\_result += cur  return polynom\_result  def fill\_polynom(polynoms, power, table, interval):  '''  Заполнение таблицы разделенных разностей  '''  if power < 1:  return  polynoms.append([])  idx\_start, idx\_end = interval  if power == 1:  for i in range(idx\_start, idx\_end):  polynoms[0].append((table.y[i + 1] - table.y[i])/(table.x[i+1] - table.x[i]))  else:  for i in range(len(polynoms[power - 2]) - 1):  polynoms[power - 1].append((polynoms[power - 2][i + 1] - polynoms[power - 2][i])/(table.x[idx\_start + i + power] - table.x[idx\_start + i])) |

**Результаты работы:**

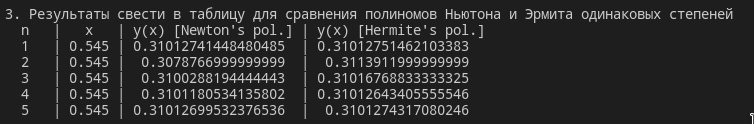
1. Значения y(x) при степенях полиномов Ньютона n= 1, 2, 3, 4 и 5 при фиксированном x, например, x=0.545.



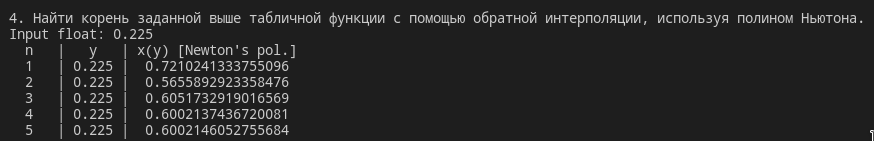
1. Значения y(x) при интерполяции полиномом Эрмита по одному, 2-м и 3-м узлам при том же, что и в п.1 фиксированном x, например, x=0.525.



1. Результаты свести в таблицу для сравнения полиномов Ньютона и Эрмита одинаковых степеней.



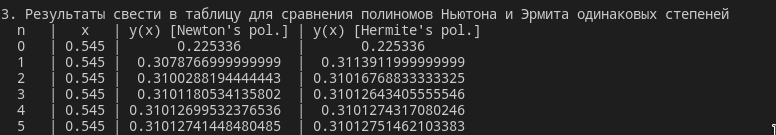
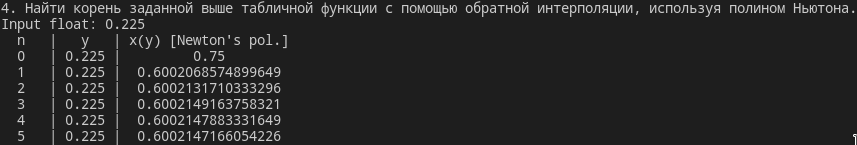
1. Найти корень заданной выше табличной функции с помощью обратной интерполяции, используя полином Ньютона.



**Вопросы при защите лабораторной работы**

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

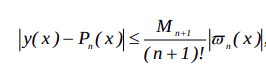
Да, работать программа с такой степенью будет, как расчет интерполяции с помощью полинома Ньютона, так и с помощью полинома Эрмита. Функция вернет значение из таблицы в точке, которая находится ближе к заданному аргументу. Но точность при такой конфигурации будет низкой.

  
Расчет корня с помощью обратной интерполяции тоже будет работать - это значение аргумента из данной таблицы, в которой функция принимает значение ближе к нулю.   


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

Практически оценить погрешность интерполяции можно при помощи оценки первого отброшенного члена в полиноме Ньютона. При этом в полиноме остаются члены, которые больше заданной погрешности расчетов.

Теоретическую погрешность многочлена Ньютона можно оценить с помощью формулы (где используются производные данной функции):

, где  - максимальное значение производной интерполируемой функции, а также 

Именно поэтому теоретическую погрешность сложно оценить.

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

При данном условии можно построить полиномы Эрмита 0, 1, 2 и 3 степени а полиномы Ньютона - 0 и 1 степени.  
Минимальная степень равна 0.

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

Информация об упорядоченности аргумента функции существенна при выборе приближенного интервала значений (из (n+1) узлов, которые по возможности расположены симметрично относительно заданного аргумента).

Если аргумент будет неупорядоченным, то значение функции получится с низкой точностью или совсем неверным.

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

Если функция, а точнее ее разделенные разности, значительно меняются на нескольких интервалах, то интерполяция обобщенным многочленом обычно не будет точной для дифференцирования данной функции. Поэтому для таких функций используется квазилинейная интерполяция, которая производится при помощи выравнивающих переменных. То есть выравнивающие переменные используются для того, чтобы повысить точность вычисления производной функции.

1. ***Будет ли работать ваша программа при произвольном неупорядоченном расположении узлов в исходной таблице?***

Да, однако точность вычислений будет снижена, поскольку не удастся найти интервал, элементы которого наиболее приближенны к аргументу на входе. В большинстве случаев, будут взяты **n** первых или последних элементов

1. ***Принципиально ли для корректной работы вашего алгоритма, чтобы узлы были расположены по возрастанию?***

Нет, поскольку интервал ищется исходя из того, находится ли аргумент между двумя соседними элементами или нет

1. ***Что будет происходить с точностью интерполяции по мере продвижения от центра к краям таблицы?***

Точность будет расти, поскольку будет расти выборка значений для интерполирования

1. ***Можно ли использовать для обратной интерполяции полином Эрмита?***

Можно, но поскольку в вычислениях используется производная функции, то необходимо найти обратную производную для данных значений.