|  |  |
| --- | --- |
| 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 № 2***

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

**Студент:** Прянишников А.Н.

**Группа:** ИУ7-45Б

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

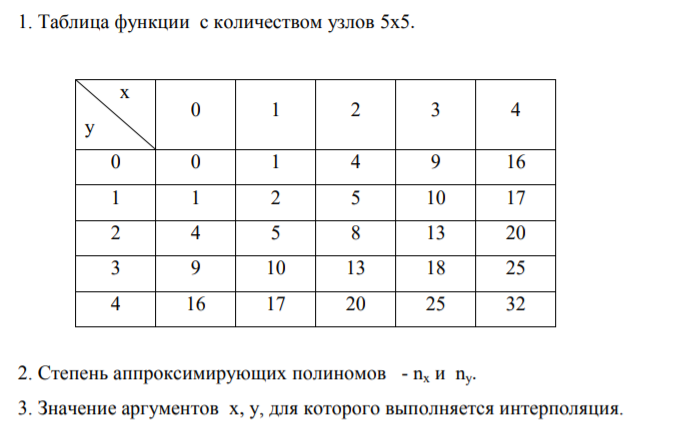
**Преподаватель:** Градов В.М.

*Москва*

*2021 г*

**Цель работы:** Получение навыков построения алгоритма интерполяции таблично заданных функций двух переменных.

1. **Исходные данные**



1. **Идея реализации**

Для выполнения задания я применил алгоритм линейной интерполяции полинома Ньютона из прошлой лабораторной работы. Сначала этот алгоритм был выполнен для z(x) и x, а результат был сохранён в массив размером ny. Затем программа снова интерполировала y и полученный массив, результат этого действия и есть искомое приближение.

1. **Код программы**
2. **import** numpy as np
4. X\_TABLE = [0, 1, 2, 3, 4]
5. Y\_TABLE = [0, 1, 2, 3, 4]
6. Z\_TABLE = [[0, 1, 4, 9, 16], [1, 2, 5, 10, 17], [4, 5, 8, 13, 20], [9, 10, 13, 18, 25], [16, 17, 20, 25, 32]]
8. # Функция подсчёта разделённой разности по заданным индексам
9. # x\_table, y\_table - таблицы значений
10. # list\_indexes - индексы элементов, с которыми работаем
11. # Возвращает численный результат
12. **def** difference(x\_table, y\_table, list\_indexes):
13. result = 0
14. length = len(list\_indexes)
16. # Если всего два элемента
17. **if** (length == 2):
19. # Элементы для подсчёта разделённой разности различны
20. **if** (list\_indexes[0] != list\_indexes[1]):
21. delta\_y = y\_table[list\_indexes[0]] - y\_table[list\_indexes[1]]
22. delta\_x = x\_table[list\_indexes[0]] - x\_table[list\_indexes[1]]
23. result = delta\_y / delta\_x
25. # Если элементов больше, то подсчитываем через рекурсию
26. **elif** (length > 2):
27. dif\_first = difference(x\_table, y\_table, list\_indexes[:length - 1])
28. dif\_second = difference(x\_table, y\_table, list\_indexes[1:])
29. result = (dif\_first - dif\_second) / (x\_table[list\_indexes[0]] - x\_table[list\_indexes[length - 1]])
30. **return** result
32. # Функция подсчёта коэффициентов для построения полиномов
33. # x\_table, y\_table - таблицы значений
34. # list\_indexes - индексы элементов, с которыми работаем
35. # Возвращает массив коэффициентов для полинома
36. **def** count\_polinom\_diff\_coefs(x\_table, y\_table, list\_indexes):
37. result\_coefs = []
38. **for** i **in** range(2, len(list\_indexes) + 1):
39. result\_coefs.append(difference(x\_table, y\_table, list\_indexes[:i]))
40. **return** result\_coefs
42. # Функция подсчёта значения полинома
43. # x\_table, y\_table - таблицы значений
44. # list\_indexes - индексы элементов, с которыми работаем
45. # x\_var - переменная, для которой ищем значение у апроксимирующей функции
46. # Возвращает результат по заданному x
47. **def** count\_polinom\_value(x\_table, y\_table, x\_var, list\_indexes):
48. coefs = count\_polinom\_diff\_coefs(x\_table, y\_table, list\_indexes)
49. x\_multiply = 1
50. result = y\_table[list\_indexes[0]]
51. **for** i **in** range(len(list\_indexes) - 1):
52. x\_multiply \*= (x\_var - x\_table[list\_indexes[i]])
53. result += (x\_multiply \* coefs[i])
54. **return** result
56. # Функция нахождения индекса элемента, который первый больше, чем заданный x
57. # x\_var - заданная переменная
58. # x\_table - табличные значения
59. **def** find\_near\_index(x\_var, x\_table):
60. length\_table = len(x\_table)
61. index = 0
62. **while** ((index < length\_table) **and** (x\_table[index] < x\_var)):
63. index += 1
64. **return** index
66. # Функция расчёта индексов для составления интерполяционного полинома Ньютона
67. # x\_var - заданная переменная
68. # n - степень полинома
69. # x\_table - табличные значения
70. **def** create\_list\_indexes\_newton(x\_var, n, x\_table):
71. index = find\_near\_index(x\_var, x\_table)
72. list\_indexes = np.array(list(range(index - ((n // 2) + 1), index - ((n // 2) + 1) + n + 1)))
73. **while** (list\_indexes[0] < 0):
74. list\_indexes += 1
75. **while** (list\_indexes[list\_indexes.shape[0] - 1] >= len(x\_table)):
76. list\_indexes -= 1
77. **return** list(list\_indexes)
79. **def** mul\_polinom(x\_table, y\_table, z\_table, nx, ny, x\_var, y\_var):
81. index\_x = create\_list\_indexes\_newton(x\_var, nx, x\_table)
82. index\_y = create\_list\_indexes\_newton(y\_var, ny, y\_table)
84. x\_table = x\_table[index\_x[0] : index\_x[len(index\_x) - 1] + 1]
85. y\_table = y\_table[index\_y[0] : index\_y[len(index\_y) - 1] + 1]
86. z\_table = z\_table[index\_y[0] : index\_y[len(index\_y) - 1] + 1]
88. **for** i **in** range(ny + 1):
89. z\_table[i] = z\_table[i][index\_x[0] : index\_x[len(index\_x) - 1] + 1]
91. x\_first = []
92. **for** i **in** range(ny + 1):
93. x\_first.append(count\_polinom\_value(x\_table, z\_table[i], x\_var, list(range(nx + 1))))
94. y\_res = count\_polinom\_value(y\_table, x\_first, y\_var, list(range(len(x\_first))))
96. **return** y\_res
98. x = float(input("Введите x: "))
99. y = float(input("Введите y: "))
100. nx = int(input("Введите nx: "))
101. ny = int(input("Введите ny: "))
103. mul\_polinom(X\_TABLE, Y\_TABLE, Z\_TABLE, nx, ny, x, y)

# Результаты

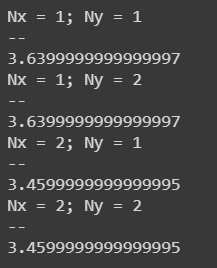
**Этап 1: тестирование алгоритма**

# Сначала надо убедиться, что программа выдаёт правильные результаты на простейших функциях и обычных примерах. Для этого проверим работу функции на известных узлах и заданной функции, где легко предсказать результат.

# Заданная функция: z = 2\*x^2 – y

# 

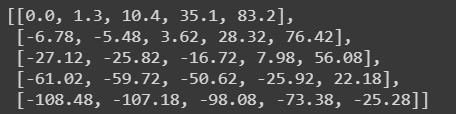
# Теперь зададим поищем значения для значений x = 1.9, y = 3.76



Сведём результаты в отдельную таблицу (истинное значение z = 3.46):

|  |  |  |
| --- | --- | --- |
|  | Результат | Погрешность |
| Nx = 1; Ny = 1 | 3.64 | 0.18 |
| Nx = 1; Ny = 2 | 3.64 | 0.18 |
| Nx = 2; Ny = 1 | 3.46 | 0 |
| Nx = 2; Ny = 2 | 3.46 | 0 |

* Кубическая функция z = 1.3 \* x^3 – 6.78 \* y^2

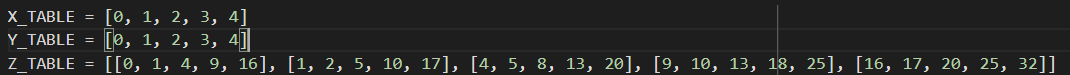


Опять составим таблицу для x = 3.73, y = 0.33 (истинное значение z = 66.7253101)

|  |  |  |
| --- | --- | --- |
|  | Результат | Погрешность |
| Nx = 1; Ny = 1 | 67.9756 | -1,2502899 |
| Nx = 1; Ny = 2 | 69.474658 | 2,7493479 |
| Nx = 2; Ny = 1 | 65.669530 | 1,0557801 |
| Nx = 2; Ny = 2 | 67.168588 | 0,4432779 |
| Nx = 3; Ny = 2 | 66.7253101 | 0 |
| Nx = 2; Ny = 3 | 67.168588 | 0,4432779 |
| Nx = 3; Ny = 3 | 66.7253101 | 0 |

Видно, что наш алгоритм работает с отличной точностью, и как только полином достигает нужных степеней, погрешность и вовсе стремится к нулю.

* Подсчёт в узлах заданной таблицы



Составим таблицу для x = 1; y = 3:

|  |  |  |
| --- | --- | --- |
|  | Результат | Значение в таблице |
| Nx = 1; Ny = 1 | 10 | 10 |
| Nx = 2; Ny = 1 | 10 | 10 |
| Nx = 2; Ny = 2 | 10 | 10 |
| Nx = 3; Ny = 3 | 10 | 10 |

Теперь можно приступить непосредственно к решению задачи, так как видно, что алгоритм считает правильно.

**Этап 2: Подсчёт значений для задания**

Рассчитаем значения для точки (x = 1.5; y = 1.5) и (x = 2.7; y = 0.2)

**X = 1.5; Y = 1.5**

|  |  |  |  |
| --- | --- | --- | --- |
| Nx / Ny | Ny = 1 | Ny = 2 | Ny = 3 |
| Nx = 1 | 5 | 4.75 | 4.75 |
| Nx = 2 | 4.75 | 4.5 | 4.5 |
| Nx = 3 | 4.75 | 4.5 | 4.5 |

**X = 2.7; Y = 0.2**

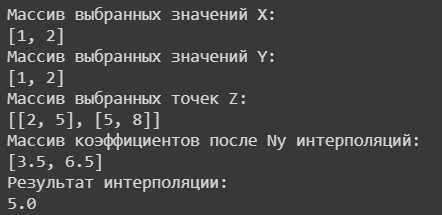
|  |  |  |  |
| --- | --- | --- | --- |
| Nx / Ny | Ny = 1 | Ny = 2 | Ny = 3 |
| Nx = 1 | 7.7 | 7.54 | 7.54 |
| Nx = 2 | 7.49 | 7.33 | 7.33 |
| Nx = 3 | 7.49 | 7.33 | 7.33 |

Таблица исходных данных повторяет значения функции z = x^2 + y^2, и получившиеся значения начиная с Ny >= 2, Nx >= 2 также с нулевой погрешностью удовлетворяют такой функции.

1. **Выводы**

Метод многомерной интерполяции позволяет приближённо вычислять значения функции по заданной таблице данных. Моя реализация отлично справилась с задачей практически без погрешности на тестовых данных, а на задании дала значения, которые соответствуют теоретическим значениям.

1. **Вопросы к защите лабораторной работы**
2. **Пусть производящая функция таблицы суть z(x,y)=x^2 +y^2 . Область определения по x и y 0-5 и 0-5. Шаги по переменным равны 1. Степени nx = ny =1, x=y=1.5. Приведите по шагам те значения функции, которые получаются в ходе последовательных интерполяций по строкам и столбцу?**

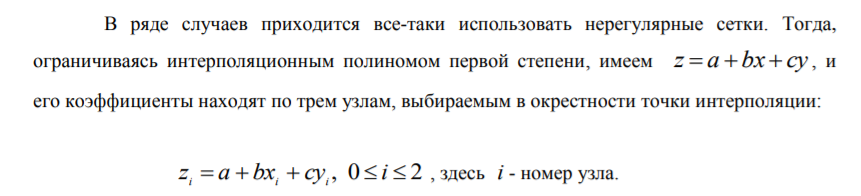


Этот результат в точности соответствует теоретическим подсчётам: при n = 1 между соседними точками проводится прямая, и значение берётся в соответствии с ней.

1. **Какова минимальная степень двумерного полинома, построенного на четырех узлах? На шести узлах?**

На четырёх узлах можно построить полином третьей степени или меньше, на шести узлах – пятой степени или меньше.

1. **Предложите алгоритм двумерной интерполяции при хаотичном расположении узлов, т.е. когда таблицы функции на регулярной сетке нет, и метод последовательной интерполяции не работает. Какие имеются ограничения на расположение узлов при разных степенях полинома**



То есть по сути для каждой точки записывается уравнение, находятся коэффициенты по соседним точкам, для каждой последующей степени нужно больше точек вокруг. При этом для полинома первой степени точки не должны лежать на одной прямой на плоскости, а для полинома второй степени точки не должны лежать в одной плоскости.

1. **Пусть на каком-либо языке программирования написана функция, выполняющая интерполяцию по двум переменным. Опишите алгоритм использования этой функции для интерполяции по трем переменным.**

Сначала выполним интерполяцию (x, F(x, y, z)) Ny раз, занесём результат в массив M, затем интерполяцию (y, M) Nz раз, занесём результат в массив T, а затем выполним интерполяцию (z, T) и найдём искомое значение.

1. **Можно ли при последовательной интерполяции по разным направлениям использовать полиномы несовпадающих степеней или даже разные методы одномерной интерполяции, например, полином Ньютона и сплайн?**

Результаты итоговой интерполяции не зависят от способа вычисления, поэтому итоги не изменятся. Если я правильно понял вопрос, то у меня это и получилось в лабораторной работе, когда Nx != Ny

1. **Опишите алгоритм двумерной интерполяции на треугольной конфигурации узлов?**

При треугольной конфигурации нужно лишь делать проверку на то, что используются все узлы, а только затем запускать алгоритм, который был описан выше.