**МГТУ им. Н.Э. Баумана**

**ВЫЧИСЛИТЕЛЬНЫЕ АЛГОРИТМЫ**

**Лабораторный практикум №2**

**по теме: «Многомерная интерполяция »**

***Студент: Нгуен Фыок Санг***

***Группa: ИУ7И-46***

***Преподаватель: Градов В.М.***

2020

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

**1. Техническое задание**

**Исходные данные.**

1. Таблица функции с количеством узлов NxM.

|  |  |  |
| --- | --- | --- |
| x/y |  |  |
|  |  |  |
|  |  |  |

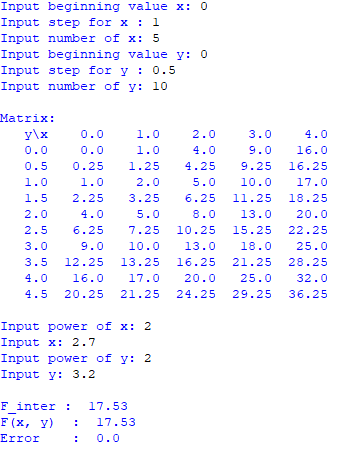
2. Степень аппроксимирующих полиномов - nx и ny.

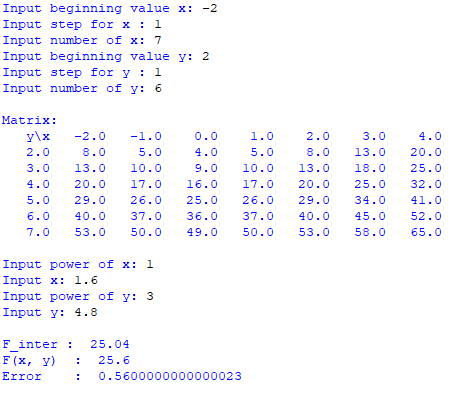
3. Значение аргументов x, y, для которого выполняется интерполяция.

**Результат работы программы.**

Значения z(x,y).

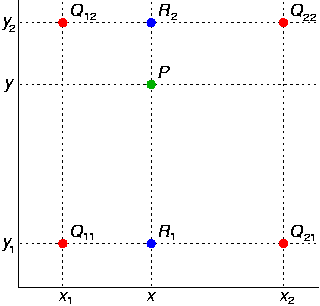
**Пример выполнения прграммы:**

****

****

**2. Описание алгоритма**

Билинейная интерполяция основывается на линейной интерполяции



Допустим, что необходимо интерполировать значение функции f(x, y) в точке {\displaystyle P=(x,y)}

P = (x, y). Значения функции в окружающих точку P известны точках

Q11 = (x1, y1),

Q12 = (x1, y2),

Q21 = (x2, y1),

Q22 = (x2, y2)

Первым шагом линейно интерполируется значение вспомогательных точек R1 и R2 вдоль [оси абсцисс](https://ru.wikipedia.org/wiki/Абсцисса), где

R1 = (x, y1)

R2 = (x, y2)

f(R1) = f(Q11) + f(Q21)

f(R2) = f(Q12) + f(Q22)

Теперь проводится линейная интерполяция между вспомогательными точками {\displaystyle R\_{1}}R1 и R2.

f(P) = = f(R1) + f(R1)

Это и есть интерполируемое значение функции f(x,y), причем значения интерполирующей функции F(x,y) равны значениям интерполируемой функции в исходных точках (x1,y1);(x2,y2);(x2,y1);(x1,y2):

f(x,y) =F(x,y) =

**3. Код программы**

def f(x, y):

return x\*\*2 + y\*\*2

def get\_matrix(x\_beg, x\_h, x\_n, y\_beg, y\_h, y\_n):

x = [x\_beg + i\*x\_h for i in range(x\_n)]

y = [y\_beg + i\*y\_h for i in range(y\_n)]

z = [[f(i, j) for i in x] for j in y]

return x, y, z

def print\_matrix(x, y, z):

print(" y\\x ", end = '')

for i in x:

print("{:6}".format(i), end = ' ')

for i in range(len(y)):

print("\n{:6}".format(y[i]), end = ' ')

for j in z[i]:

print("{:6}".format(j), end = ' ')

print('\n')

def choose\_dots(a, n, x):

a\_len = len(a)

i\_near = min(range(a\_len), key = lambda i: abs(a[i] - x)) # index of nearest value

space\_needed = ceil(n / 2)

if (i\_near + space\_needed + 1 > a\_len):

i\_end = a\_len

i\_start = a\_len - n

elif (i\_near < space\_needed):

i\_start = 0

i\_end = n

else:

i\_start = i\_near - space\_needed + 1

i\_end = i\_start + n

return i\_start, i\_end

# Matrix of differences

def get\_diff\_matr(tbl, n):

for i in range(n):

tmp = []

for j in range(n-i):

tmp.append((tbl[i+1][j] - tbl[i+1][j+1]) / (tbl[0][j] - tbl[0][i+j+1]))

tbl.append(tmp)

return tbl

# n - polynom's power

def newtons\_interpolation(tbl, n, x):

matr = get\_diff\_matr(tbl, n)

tmp = 1

res = 0

for i in range(n+1):

res += tmp \* matr[i+1][0]

tmp \*= (x - matr[0][i])

return res

def multi\_interpolation(x, y, z, x\_val, y\_val, x\_n, y\_n):

ix\_beg, ix\_end = choose\_dots(x, x\_n + 1, x\_val)

iy\_beg, iy\_end = choose\_dots(y, y\_n + 1, y\_val)

x = x[ix\_beg : ix\_end]

y = y[iy\_beg : iy\_end]

z = z[iy\_beg : iy\_end]

for i in range(y\_n + 1):

z[i] = z[i][ix\_beg : ix\_end]

res = [newtons\_interpolation([x, z[i]], x\_n, x\_val) for i in range(y\_n + 1)]

return newtons\_interpolation([y, res], y\_n, y\_val)

x\_beg = float(input("Input beginning value x: "))

x\_h = float(input("Input step for x : "))

x\_N = int(input("Input number of x: "))

y\_beg = float(input("Input beginning value y: "))

y\_h = float(input("Input step for y : "))

y\_N = int(input("Input number of y: "))

x, y, z = get\_matrix(x\_beg, x\_h, x\_N, y\_beg, y\_h, y\_N)

print("\nMatrix:")

print\_matrix(x, y, z)

x\_n = int(input("Input power of x: "))

x\_find = float(input("Input x: "))

y\_n = int(input("Input power of y: "))

y\_find = float(input("Input y: "))

# Results

found = multi\_interpolation(x, y, z, x\_find, y\_find, x\_n, y\_n)

print("\nF\_inter : ", found)

print("F(x, y) : ", f(x\_find, y\_find))

print("Error : ", abs(f(x\_find, y\_find) - found), "\n")