Белорусский государственный университет

Факультет прикладной математики и информатики

Кафедра технологий программирования

Стефанович Константин Андреевич

**Лабораторная работа №4**

**Приближение функций**

студента 2 курса 6 группы

**Преподаватель**

***Радкевич Елена Владимировна***

Ассистент кафедры вычислительной математики ФПМИ

Работа сдана \_\_\_\_ мая 2017г.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Подпись преподавателя)

Минск, 2017

Оглавление

[1. Техническое задание 3](#_Toc483253605)

[2. Многочлен Лагранжа 3](#_Toc483253606)

[3. Многочлен Ньютона 4](#_Toc483253607)

[4. Многочлен Ньютона по равностоящим в начале и конце таблицы 8](#_Toc483253608)

[5. Многочлен Эрмита 10](#_Toc483253609)

[6. Приложение: Листинг 12](#_Toc483253610)

# 1. Техническое задание

* Построить многочлен Лагранжа и оценить погрешность, также найти значения в точках
* Построить многочлен Ньютона
* Вычислить значение, используя интерполяционный многочлен Ньютона по равностоящим узлам в начале и в конце таблицы
* Построить многочлен Эрмита
* Вычисления выполнять со следующими входными данными:

,

Примечание: для каждого метода будет приводиться только код, соответствующий данному методу, весь код полностью можно посмотреть в разделе «Листинг»

# 2. Многочлен Лагранжа

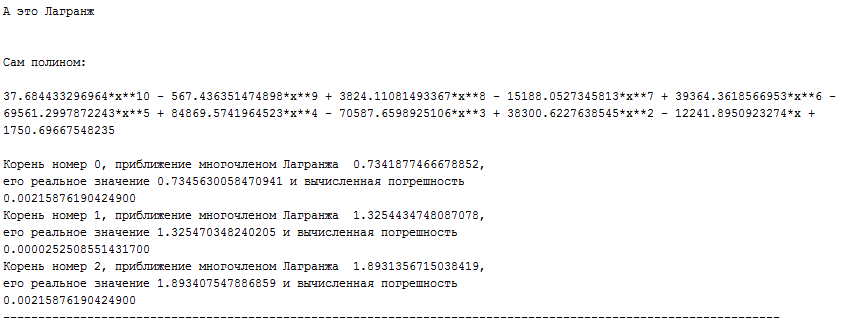
Представим интерполяционную функцию в виде полинома  
P_n(x) = \sum_{i=0}^n y_i Q_{n,i}(x)  
где Q_{n,i}(x) - полиномы степени n вида:  
Q_{n,i}(x) = \prod_{j=0, j\neq i}^{n} \frac{(x-x_j)}{(x_i-x_j)}  
Очевидно, что Q_{n,i}(x) принимает значение 1 в точке x_i и 0 в остальных узлах интерполяции. Следовательно, в точке x_i исходный полином принимает значение y_i  
Таким образом, построенный полином P_n(x) является интерполяционным полиномом для функции  y = f(x)  на сетке  \bf{X} .

Поставим вопрос о том, насколько хорошо интерполяционный полином P_n(x) приближает функцию f(x) на отрезке [a,b].  
Рассмотрим остаточный член:  
R_n(x) = f(x) - P_n(x), x ∈ [a, b].  
По определению интерполяционного полинома, <br>R_n(x_i) = 0, x_i \in \bf X   
поэтому речь идет об оценке R_n(x) при значениях x \not= x_i.  
Пусть f(x) имеет непрерывную (n+1) производную на отрезке [a, b].  
Тогда погрешность определяется формулой:  
|R_n(x)| = \frac{f^{n+1}(\varepsilon)}{(n+1)!}w_{n+1}(x),  
где w_{n+1} = (x - x_0)...(x - x_n),  
\varepsilon - точка из [a, b].  
Так как точка \varepsilon  неизвестна, то эта формула позволяет только оценить погрешность:  
|R_n(x)| \le \frac{M_{n+1}}{(n+1)!}|w_{n+1}(x)|  
где M_{n+1} = \max_{x\in[\alpha, \beta]}|f^{n+1}(x)|

Код программы для данного метода:

print(**'-----------------------------------------------------------------------------------'**)  
print(**'А это Лагранж\n\n'**)  
f = 1  
**for** e **in** params:  
 f\*=(x-e)  
lPl=0  
**for** i **in** range(len(params)):  
 q, r = div(f, x-params[i], x)  
 g = lambdify(x, q)  
 lPl+=q\*values[i]/g(params[i])  
print(**'Сам полином:\n'**)  
print(fill(str(simplify(lPl)), 120)+**'\n'**)  
l = lambdify(x, lPl)  
**for** i **in** range(len(v)):  
 dif=diff(1.7\*sp.exp(-x)-1.7\*sp.cos(x), x, 11)  
 func = lambdify(x, abs(dif)\*-1)  
 fr=fminbound(func, 1,2)  
 func = lambdify(x, abs(dif))  
 w = lambdify(x, f)  
 mistake=(func(fr)/factorial(11))\*abs(w(v[i]))  
 print(**"Корень номер {0}, приближение многочленом Лагранжа {1}, его реальное значение {2} и вычисленная погрешность{3}"**.format(i,l(v[i]),myf(v[i]),mistake))

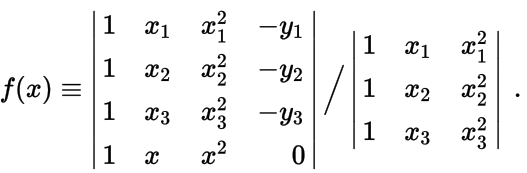
Вывод для данного метода:



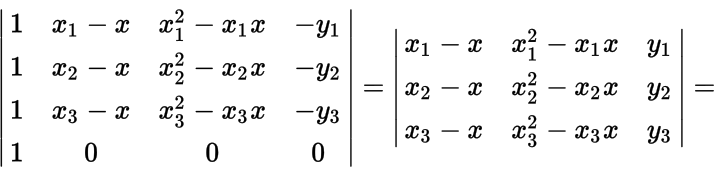
# 3. Многочлен Ньютона

Основной недостаток построения интерполяционного полинома по методу (в форме) Лагранжа заключается в том, что при добавлении в таблицу нового узла (новых результатов измерений), в формуле приходится пересчитывать все слагаемые. От этого недостатка свободен метод Ньютона, в котором добавление нового узла ведет к добавлению лишь одного слагаемого к построенному ранее полиному.

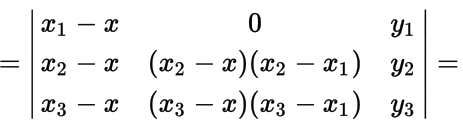
Для пояснения идеи вновь обратимся к случаю n_{}=3 и в качестве стартового представления интерполяционного полинома воспользуемся его детерминантной формой



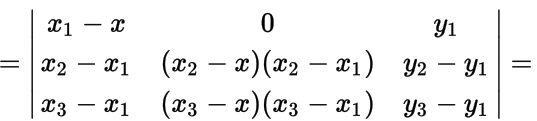
Преобразуем определитель, стоящий в числителе; с этой целью вычтем из третьего столбца второй, домноженный на x_{}, а потом из второго столбца — первый, домноженный на x_{}, получим



Далее, вычтем из второго столбца первый, домноженный на x_{1}:

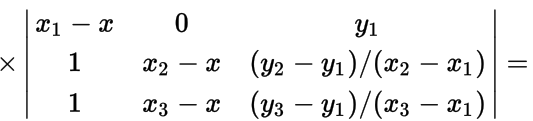


Теперь вычтем первую строчку из второй и третьей:



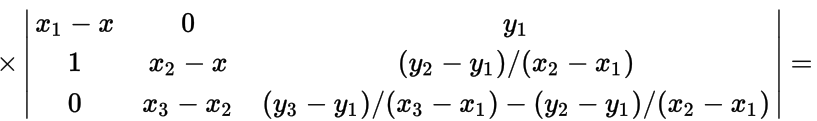
и вынесем множители из строк:

=(x_2-x_1)(x_3-x_1) \times



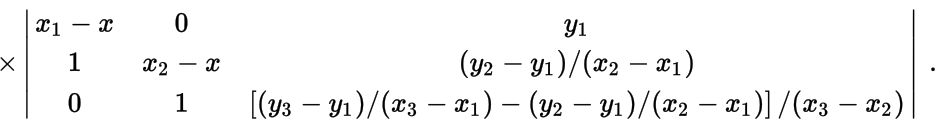
Из третьей строчки вычтем вторую:

=(x_2-x_1)(x_3-x_1)\times

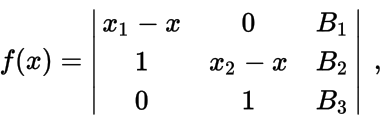


и снова вынесем множитель:

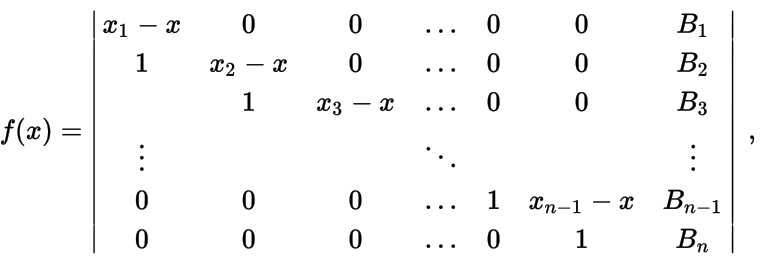
=(x_2-x_1)(x_3-x_1)(x_3-x_2) \times



В результате интерполяционный полином получается в виде определителя, имеющего структуру



а в случае произвольного n_{}:



который напоминает характеристический полином матрицы Фробениуса. Раскладывая его рекурсивно по строкам (см. ☞ вычисление определителя по методу рекуррентных соотношений ) , получаем для n_{}=3 интерполяционный полином в форме

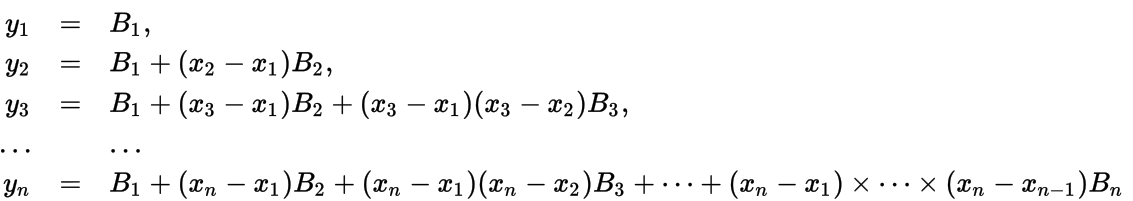
f(x)=B_1+(x-x_1)B_2+(x-x_1)(x-x_2)B_3 \ ,

а в случае произвольного n_{} — в форме

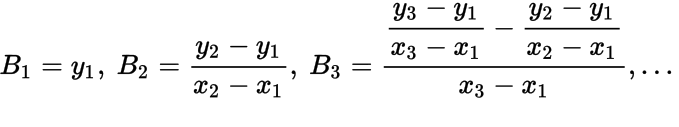
f(x)=B_1+(x-x_1)B_2+(x-x_1)(x-x_2)B_3 + \dots + (x-x_1)\times \dots \times (x-x_{n-1})B_n \ ,

которая и называется **формой Ньютона**.

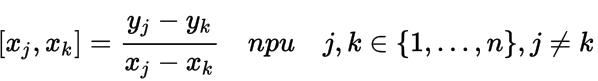
Коэффициенты B_{1},B_2,\dots,B_n в этой форме определяются с помощью интерполяционной таблицы:



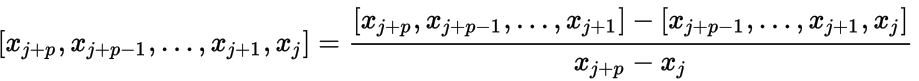
Получаем последовательно:



Схему вычисления коэффициентов можно оформить в виде таблицы, если ввести следующее обозначение. Выражение



называется **разделенной разностью первого порядка**. Выражение



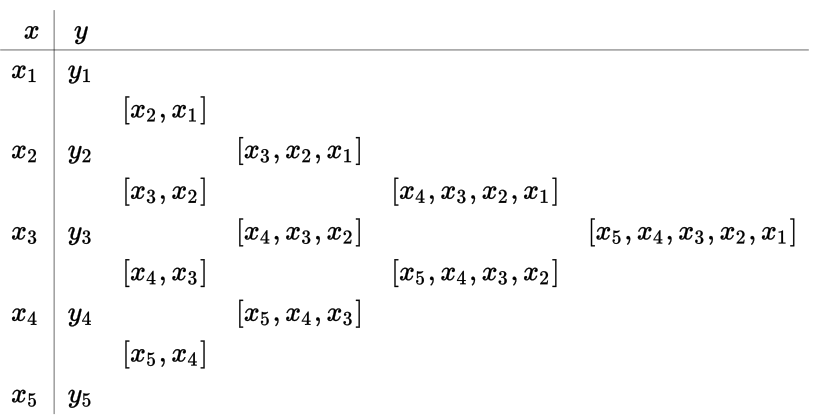
называется **разделенной разностью порядка** \mathbf p.

**Теорема.** Интерполяционный полином в форме Ньютона записывается в виде:

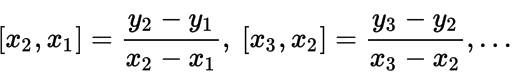
![f(x)=y_1+[x_2,x_1](x-x_1)+[x_3,x_2,x_1](x-x_1)(x-x_2)+\dots+](data:image/png;base64,)

+[x_n,x_{n-1},\dots,x_1] (x-x_1)\times \dots \times (x-x_{n-1})

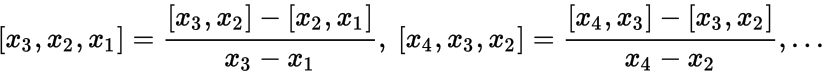
Поясним схему вычисления разделенных разностей для случая n_{}=5. В первых двух столбцах стоят данные интерполяционной таблицы (со вставленными пустыми строками между соседними):



Для вычисления третьего столбца мы вычитаем соседние значения y_{} и делим на соответствующую разность x_{}:



и ставим получившиеся числа между соответствующими значениями x_{}. Четвертый столбец заполняется аналогично: вычитаются соседние числа третьего столбца и делятся на разность крайних задействованных значений x_{}:

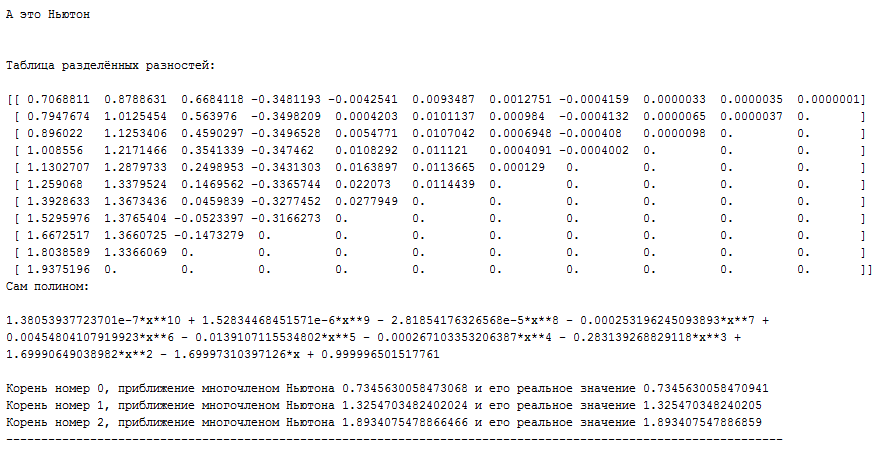


И так далее. После получения треугольной таблицы выбираются числа из верхней стороны треугольника: именно они и являются искомыми коэффициентами интерполяционного полинома в форме Ньютона.

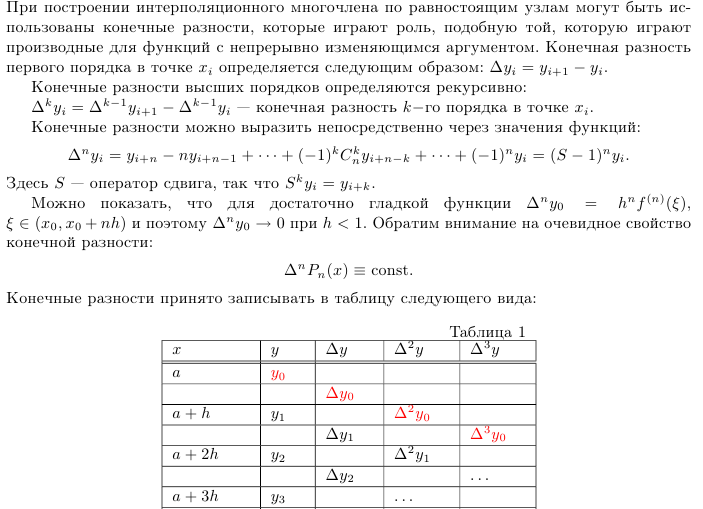
Код программы для данного метода:

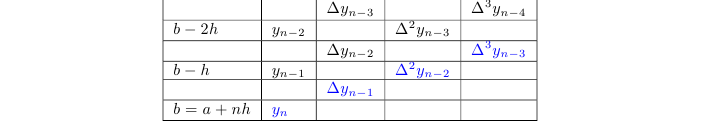
print(**'-------------------------------------------------------------------------------------'**)  
print(**'А это Ньютон\n\n'**)  
interpolant = rr(params,values)  
f = 1  
g = 0  
**for** i **in** range(len(params)):  
 g+=interpolant[i]\*f  
 f\*=(x-params[i])  
print(**'Сам полином:\n'**)  
print(fill(str(simplify(g)), 120)+**'\n'**)  
l = lambdify(x, g)  
**for** i **in** range(len(v)):  
 print(**"Корень номер {0}, приближение многочленом Ньютона {1} и его реальное значение {2}"**.format(i,l(v[i]),myf(v[i])))

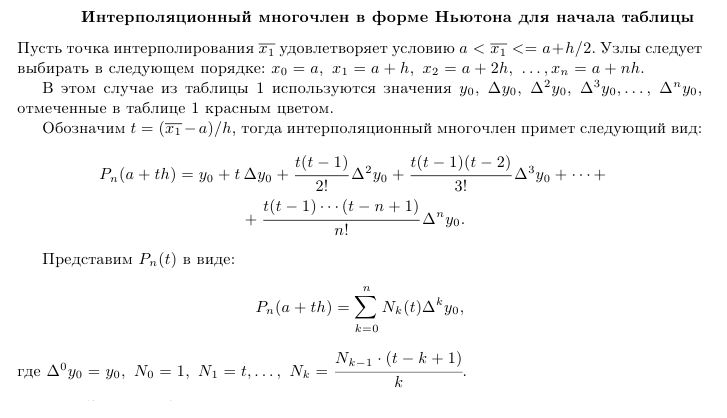
Вывод для данного метода:

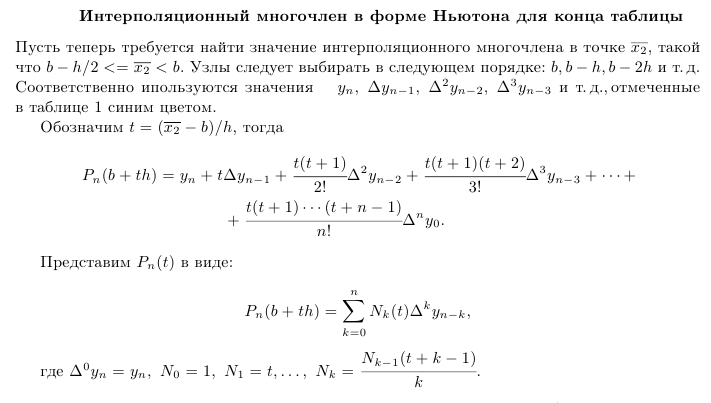


# 4. Многочлен Ньютона по равностоящим в начале и конце таблицы





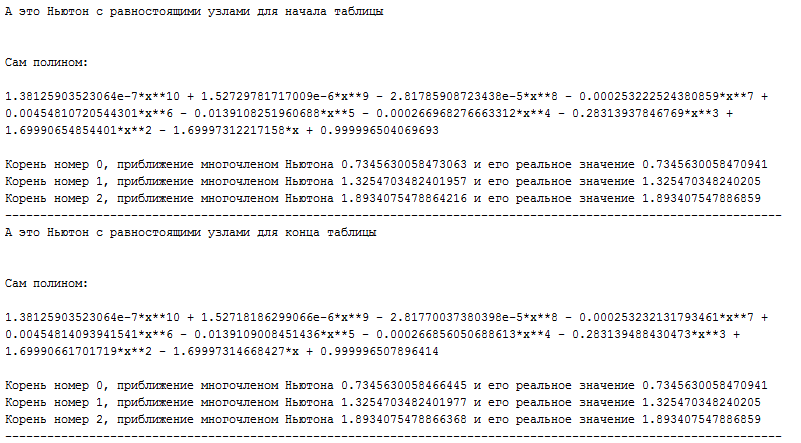




Код программы для данного метода:

print(**'-------------------------------------------------------------------------------------'**)  
print(**'А это Ньютон с равностоящими узлами для начала таблицы\n\n'**)  
f = 1  
g = 0  
**for** i **in** range(len(params)):  
 g+=delta(i,values,0)\*f  
 f\*=((x-params[0])/h-i)/(i+1)  
l = lambdify(x, g)  
print(**'Сам полином:\n'**)  
print(fill(str(simplify(g)), 120)+**'\n'**)  
**for** i **in** range(len(v)):  
 print(**"Корень номер {0}, приближение многочленом Ньютона {1} и его реальное значение {2}"**.format(i,l(v[i]),myf(v[i])))  
  
print(**'-------------------------------------------------------------------------------------'**)  
print(**'А это Ньютон с равностоящими узлами для конца таблицы\n\n'**)  
f = 1  
g = 0  
**for** i **in** range(len(params)):  
 g+=delta(i,values,len(params)-i-1)\*f  
 f\*=((x-params[len(params)-1])/h+i)/(i+1)  
l = lambdify(x, g)  
print(**'Сам полином:\n'**)  
print(fill(str(simplify(g)), 120)+**'\n'**)  
**for** i **in** range(len(v)):  
 print(**"Корень номер {0}, приближение многочленом Ньютона {1} и его реальное значение {2}"**.format(i,l(v[i]),myf(v[i])))

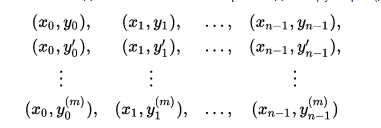
Вывод для данного метода:



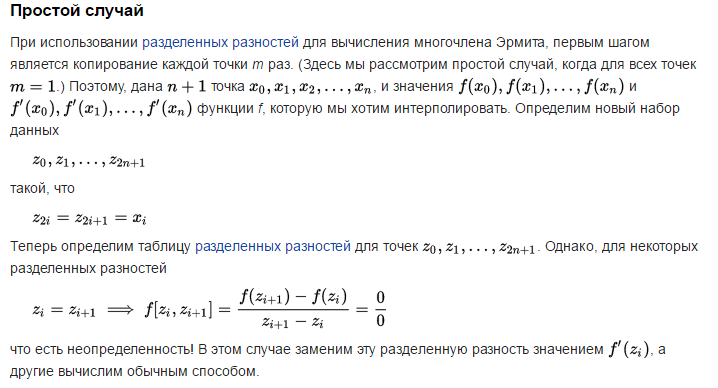
# 5. Многочлен Эрмита

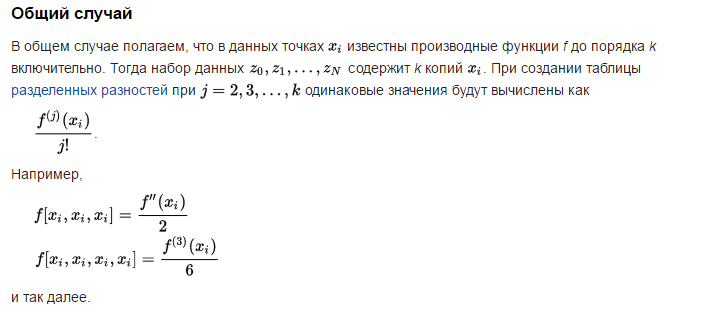
**Эрмитова интерполяция** - метод полиномиальной интерполяции, названный в честь французского математика Шарля Эрмита. Многочлены Эрмита тесно связаны с многочленами Ньютона.

В отличие от интерполяции Ньютона, эрмитова интерполяция строит многочлен, значения которого в выбранных точках совпадают со значениями исходной функции в этих точках, и производные многочлена в данных точках совпадают со значениями производных функции (до некоторого порядка m). Это означает, что *n*(*m* + 1) величин

****

{\displaystyle {\begin{matrix}(x\_{0},y\_{0}),&(x\_{1},y\_{1}),&\ldots ,&(x\_{n-1},y\_{n-1}),\\(x\_{0},y\_{0}'),&(x\_{1},y\_{1}'),&\ldots ,&(x\_{n-1},y\_{n-1}'),\\\vdots &\vdots &&\vdots \\(x\_{0},y\_{0}^{(m)}),&(x\_{1},y\_{1}^{(m)}),&\ldots ,&(x\_{n-1},y\_{n-1}^{(m)})\end{matrix}}}должны быть известны, тогда как для ньютоновской интерполяции необходимы только первые *n* значений. Полученный многочлен может иметь степень не более, чем *n*(*m* + 1) − 1, максимальная степень многочлена Ньютона же равна *n* − 1. (В общем случае *m* не обязательно должно быть фиксировано, то есть в одних точках может быть известно значение большего количества производных, чем в других. В этом случае многочлен будет иметь степень *N* − 1, где *N* - число известных значений.)

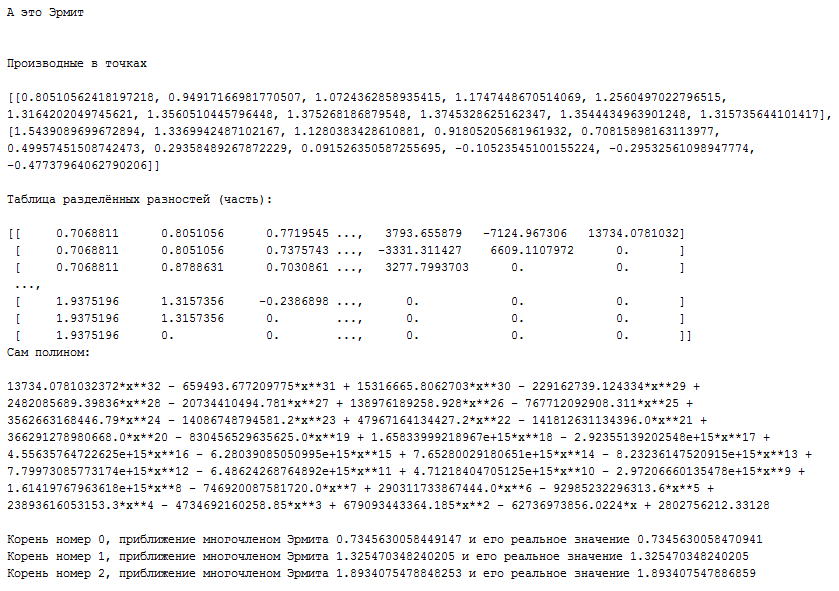




Код программы для данного метода:

print(**'-------------------------------------------------------------------------------------'**)  
print(**'А это Эрмит\n\n'**)  
k = 2  
fg=[0]\*(len(params)\*(k+1))  
fr=[0]\*(len(values)\*(k+1))  
**for** i **in** range(len(fg)) :  
 fg[i]=params[i//(k+1)]  
 fr[i]=values[i//(k+1)]  
a = []  
**for** r **in** range(k):  
 a.append([])  
 **for** c **in** range(len(params)):  
 phi = lambdify((x), diff(eval(**'1.7\*exp(-x)+1-1.7\*cos(x)'**), x, r+1), **'numpy'**)  
 a[r].append((phi(params[c])))  
print(**'Производные в точках\n'**)  
print(fill(str(a), 120)+**'\n'**)  
interpolant = rr\_ermit(fg,fr,a,(k+1))  
f = 1  
g = 0  
**for** i **in** range(len(fg)):  
 g+=interpolant[i]\*f  
 f\*=(x-fg[i])  
print(**'Сам полином:\n'**)  
print(fill(str(simplify(g)), 120)+**'\n'**)  
l = lambdify(x, g)  
**for** i **in** range(len(v)):  
 print(**"Корень номер {0}, приближение многочленом Эрмита {1} и его реальное значение {2}"**.format(i,l(v[i]),myf(v[i])))

Вывод для данного метода:



# 6. Приложение: Листинг

//functions.py

**from** math **import** \*  
**from** sympy **import** \*  
**import** numpy **as** np  
**import** sympy **as** sp  
**def** binom(n,r):  
 **return** factorial(n) / factorial(r) / factorial(n-r)  
**def** rr(x,y):  
 n = len(x)  
 v = np.zeros((n,n))  
 **for** j **in** range(n):  
 v[j,0] = y[j]  
 **for** i **in** range(1,n):  
 **for** j **in** range(n-i):  
 v[j,i] = (v[j+1,i-1]-v[j,i-1])/(x[j+i]-x[j])  
 c = v[0,:].copy()  
 print(**"Таблица разделённых разностей:\n"**)  
 print(v)  
 **return** c  
**def** delta(k,y,how):  
 **if** k==0 :**return** y[how]  
 sum=0  
 **for** i **in** range(k+1):  
 sum+=y[k-i+how]\*binom(k,i)\*pow(-1,i)  
 **return** sum  
**def** rr\_ermit(x,y,der,k):  
 n = len(x)  
 v = np.zeros((n,n))  
 **for** j **in** range(n):  
 v[j,0] = y[j]  
 **for** i **in** range(1,n):  
 **for** j **in** range(n-i):  
 **if**((j+i)//k==j//k):  
 v[j, i]=(der[i-1][j//k])/factorial(i)  
 **else**:  
 v[j,i] = (v[j+1,i-1]-v[j,i-1])/(x[j+i]-x[j])  
 c = v[0,:].copy()  
 print(**"Таблица разделённых разностей (часть):\n"**)  
 print(v)  
 **return** c

//script.py

**from** functions **import** \*  
**from** textwrap **import** \*  
**from** scipy.optimize **import** fminbound  
np.set\_printoptions(precision=7)  
np.set\_printoptions(linewidth=500)  
np.set\_printoptions(suppress=**True**)  
*#np.set\_printoptions(threshold=np.inf)*v = [1+0.1/3, 1.5+0.1/2, 2-0.1/3]  
x = Symbol(**'x'**)  
myf=lambdify(x, 1.7\*e\*\*(-x)+1-1.7\*sp.cos(x))  
params = [0]\*11  
values = [0]\*11  
start = 1.0  
h = 0.1  
**for** i **in** range (11):  
 params[i]=start  
 values[i]=myf(start)  
 start+=h  
print(**"Точки:"**)  
print(params)  
print(**"Значения в точках:"**)  
print(values)  
print(**'---------------------------------------------------------------------------------------------------------------'**)  
print(**'А это Лагранж\n\n'**)  
f = 1  
**for** e **in** params:  
 f\*=(x-e)  
lPl=0  
**for** i **in** range(len(params)):  
 q, r = div(f, x-params[i], x)  
 g = lambdify(x, q)  
 lPl+=q\*values[i]/g(params[i])  
print(**'Сам полином:\n'**)  
print(fill(str(simplify(lPl)), 120)+**'\n'**)  
l = lambdify(x, lPl)  
**for** i **in** range(len(v)):  
 dif=diff(1.7\*sp.exp(-x)-1.7\*sp.cos(x), x, 11)  
 func = lambdify(x, abs(dif)\*-1)  
 fr=fminbound(func, 1,2)  
 func = lambdify(x, abs(dif))  
 w = lambdify(x, f)  
 mistake=(func(fr)/factorial(11))\*abs(w(v[i]))  
 print(**"Корень номер {0}, приближение многочленом Лагранжа {1}, его реальное значение {2} и вычисленная погрешность{3}"**.format(i,l(v[i]),myf(v[i]),mistake))  
  
print(**'---------------------------------------------------------------------------------------------------------------'**)  
print(**'А это Ньютон\n\n'**)  
interpolant = rr(params,values)  
f = 1  
g = 0  
**for** i **in** range(len(params)):  
 g+=interpolant[i]\*f  
 f\*=(x-params[i])  
print(**'Сам полином:\n'**)  
print(fill(str(simplify(g)), 120)+**'\n'**)  
l = lambdify(x, g)  
**for** i **in** range(len(v)):  
 print(**"Корень номер {0}, приближение многочленом Ньютона {1} и его реальное значение {2}"**.format(i,l(v[i]),myf(v[i])))  
  
print(**'---------------------------------------------------------------------------------------------------------------'**)  
print(**'А это Ньютон с равностоящими узлами для начала таблицы\n\n'**)  
f = 1  
g = 0  
**for** i **in** range(len(params)):  
 g+=delta(i,values,0)\*f  
 f\*=((x-params[0])/h-i)/(i+1)  
l = lambdify(x, g)  
print(**'Сам полином:\n'**)  
print(fill(str(simplify(g)), 120)+**'\n'**)  
**for** i **in** range(len(v)):  
 print(**"Корень номер {0}, приближение многочленом Ньютона {1} и его реальное значение {2}"**.format(i,l(v[i]),myf(v[i])))  
  
print(**'---------------------------------------------------------------------------------------------------------------'**)  
print(**'А это Ньютон с равностоящими узлами для конца таблицы\n\n'**)  
f = 1  
g = 0  
**for** i **in** range(len(params)):  
 g+=delta(i,values,len(params)-i-1)\*f  
 f\*=((x-params[len(params)-1])/h+i)/(i+1)  
l = lambdify(x, g)  
print(**'Сам полином:\n'**)  
print(fill(str(simplify(g)), 120)+**'\n'**)  
**for** i **in** range(len(v)):  
 print(**"Корень номер {0}, приближение многочленом Ньютона {1} и его реальное значение {2}"**.format(i,l(v[i]),myf(v[i])))  
  
print(**'---------------------------------------------------------------------------------------------------------------'**)  
print(**'А это Эрмит\n\n'**)  
k = 2  
fg=[0]\*(len(params)\*(k+1))  
fr=[0]\*(len(values)\*(k+1))  
**for** i **in** range(len(fg)) :  
 fg[i]=params[i//(k+1)]  
 fr[i]=values[i//(k+1)]  
a = []  
**for** r **in** range(k):  
 a.append([])  
 **for** c **in** range(len(params)):  
 phi = lambdify((x), diff(eval(**'1.7\*exp(-x)+1-1.7\*cos(x)'**), x, r+1), **'numpy'**)  
 a[r].append((phi(params[c])))  
print(**'Производные в точках\n'**)  
print(fill(str(a), 120)+**'\n'**)  
interpolant = rr\_ermit(fg,fr,a,(k+1))  
f = 1  
g = 0  
**for** i **in** range(len(fg)):  
 g+=interpolant[i]\*f  
 f\*=(x-fg[i])  
print(**'Сам полином:\n'**)  
print(fill(str(simplify(g)), 120)+**'\n'**)  
l = lambdify(x, g)  
**for** i **in** range(len(v)):  
 print(**"Корень номер {0}, приближение многочленом Эрмита {1} и его реальное значение {2}"**.format(i,l(v[i]),myf(v[i])))