Университет ИТМО

**Факультет: ПИиКТ**

**Дисциплина: Вычислительная математика**

**Лабораторная работа №4**

Вариант “4”

Выполнили: **Кудлаков Роман**

Группа: **P3231**

Преподаватель: **Перл Ольга Вячеславовна**

Санкт-Петербург, 2021

**Описание метода**

**Аппроксимация методом наименьших квадратов** – метод, основанный на замене экспериментально полученных данных аналитической функцией, которая вычисляется при помощи построения аппроксимирующего многочлена n-степени. Точки данного многочлена проходят максимально близко к экспериментальным точкам. Близость исходной и аппроксимирующей функции определяется числовой мерой: сумма квадратов отклонений экспериментальных данных от аппроксимирующей кривой должна быть наименьшей.

Аппроксимирующие функции могут быть представлены в виде:

1. Многочлена степени m

2. Логарифмической функции

3. Экспоненциальной функции

4. Степенной функции

Алгоритм реализации для многочлена степени m выглядит следующим образом:

1. Создается набор произвольных пар (x, y)

2. Задается степень многочлена

3. Определяются коэффициенты для построения системы уравнений размерностью (m+1)

4. Формируется системы линейных уравнений размерностью (m+1)

5. Решая систему, получаем коэффициенты аппроксимирующего многочлена степени m

6. Подставляем коэффициенты в формулу многочлена степени m

В алгоритмах для логарифмической, экспоненциальной и степенной функций вместо того, чтобы формировать систему уравнений и потом ее решать, можно сразу найти коэффициенты a и b подставив значения в следующие формулы:

1. Для логарифмической функции

2. Для экспоненциальной функции

3. Для степенной функции

**Листинг кода**

**def make\_linear\_system**(variables, sums, points):

funcs = []

for i in range(len(variables)):

values = [variables[j] \* sums[i + j]

for j in range(len(variables))]

funcs.append(sum(values) - sum(points[:, 0] \*\* i \* points[:, 1]))

return funcs

**def count\_coefs**(funcs, variables):

coefs = linsolve(funcs, variables).args[0]

coefs = rid\_of\_free\_variables(coefs, variables)

return coefs

**def index\_of\_the\_biggest\_difference**(differences):

max = differences[0]

index = 0

for i in range(1, len(differences)):

if max < differences[i]:

max = differences[i]

index = i

return index

**def count\_differences**(coefs, points):

differences = []

for i in range(len(points)):

total = 0

for j in range(len(coefs)):

total += coefs[j] \* points[i][0] \*\* j

differences.append(abs(total - points[i][1]))

return differences

**def count\_sums\_x\_in\_different\_pow**(points, degree):

sums = []

for i in range(2 \* degree + 1):

sums.append(sum(points[:, 0] \*\* i))

return sums

**def rid\_of\_free\_variables**(coefs, variables):

list\_of\_values = [(variables[i], 1) for i in range(len(variables))]

new\_coefs = []

for i in range(len(coefs)):

new\_coefs.append(coefs[i].subs(list\_of\_values).evalf())

return new\_coefs

**def** **do\_polynomial\_approximation**(points, variables, degree):

sums\_x\_in\_different\_pow = count\_sums\_x\_in\_different\_pow(points, degree)

functions = make\_linear\_system(variables, sums\_x\_in\_different\_pow, points)

first\_coefs = count\_coefs(functions, variables)

differences = count\_differences(first\_coefs, points)

index = index\_of\_the\_biggest\_difference(differences)

for i in range(len(sums\_x\_in\_different\_pow)):

sums\_x\_in\_different\_pow -= points[index][0] \*\* i

second\_points = np.delete(points, index, axis=0)

sums\_x\_in\_different\_pow = count\_sums\_x\_in\_different\_pow(second\_points, degree)

functions = make\_linear\_system(variables, sums\_x\_in\_different\_pow, second\_points)

second\_coefs = count\_coefs(functions, variables)

differences = count\_differences(second\_coefs, second\_points)

return [first\_coefs, second\_coefs, index]

**def** **do\_logarithmic\_approximation**(points, variables, degree):

for i in range(len(points)):

points[i][0] = log(points[i][0])

sums\_x\_in\_different\_pow = count\_sums\_x\_in\_different\_pow(points, degree)

functions = make\_linear\_system(variables, sums\_x\_in\_different\_pow, points)

first\_coefs = count\_coefs(functions, variables)

differences = count\_differences(first\_coefs, points)

index = index\_of\_the\_biggest\_difference(differences)

for i in range(len(sums\_x\_in\_different\_pow)):

sums\_x\_in\_different\_pow -= points[index][0] \*\* i

second\_points = np.delete(points, index, axis=0)

sums\_x\_in\_different\_pow = count\_sums\_x\_in\_different\_pow(second\_points, degree)

functions = make\_linear\_system(variables, sums\_x\_in\_different\_pow, second\_points)

second\_coefs = count\_coefs(functions, variables)

return [first\_coefs, second\_coefs, index]

**def** **do\_exponential\_approximation**(points, variables, degree):

for i in range(len(points)):

points[i][1] = log(points[i][1])

sums\_x\_in\_different\_pow = count\_sums\_x\_in\_different\_pow(points, degree)

functions = make\_linear\_system(variables, sums\_x\_in\_different\_pow, points)

first\_coefs = count\_coefs(functions, variables)

differences = count\_differences(first\_coefs, points)

index = index\_of\_the\_biggest\_difference(differences)

for i in range(len(sums\_x\_in\_different\_pow)):

sums\_x\_in\_different\_pow -= points[index][0] \*\* i

second\_points = np.delete(points, index, axis=0)

sums\_x\_in\_different\_pow = count\_sums\_x\_in\_different\_pow(second\_points, degree)

functions = make\_linear\_system(variables, sums\_x\_in\_different\_pow, second\_points)

second\_coefs = count\_coefs(functions, variables)

return [first\_coefs, second\_coefs, index]

**def** **do\_power\_approximation**(points, variables, degree):

for i in range(len(points)):

points[i][0] = log(points[i][0])

points[i][1] = log(points[i][1])

sums\_x\_in\_different\_pow = count\_sums\_x\_in\_different\_pow(points, degree)

functions = make\_linear\_system(variables, sums\_x\_in\_different\_pow, points)

first\_coefs = count\_coefs(functions, variables)

differences = count\_differences(first\_coefs, points)

index = index\_of\_the\_biggest\_difference(differences)

for i in range(len(sums\_x\_in\_different\_pow)):

sums\_x\_in\_different\_pow -= points[index][0] \*\* i

second\_points = np.delete(points, index, axis=0)

sums\_x\_in\_different\_pow = count\_sums\_x\_in\_different\_pow(second\_points, degree)

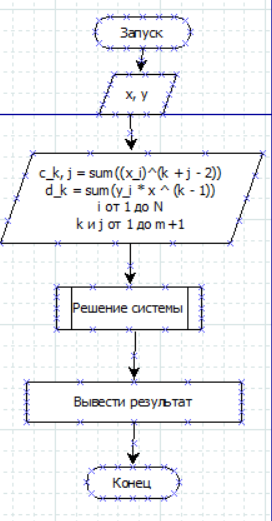
functions = make\_linear\_system(variables, sums\_x\_in\_different\_pow, second\_points)

second\_coefs = count\_coefs(functions, variables)

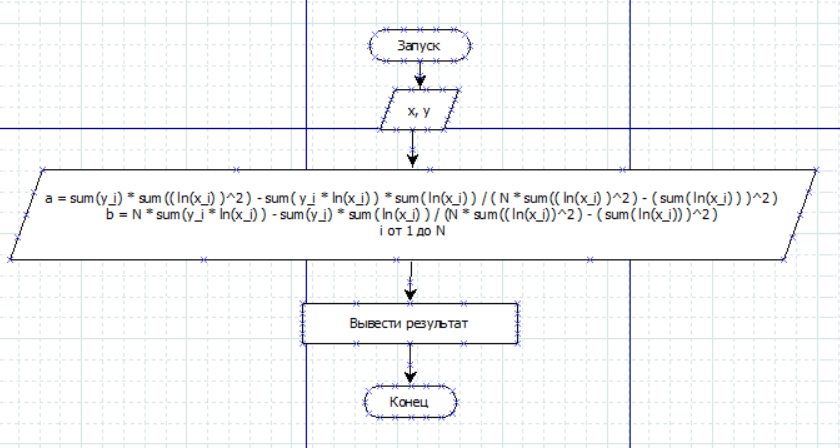
return [first\_coefs, second\_coefs, index]

**Блок-схема**

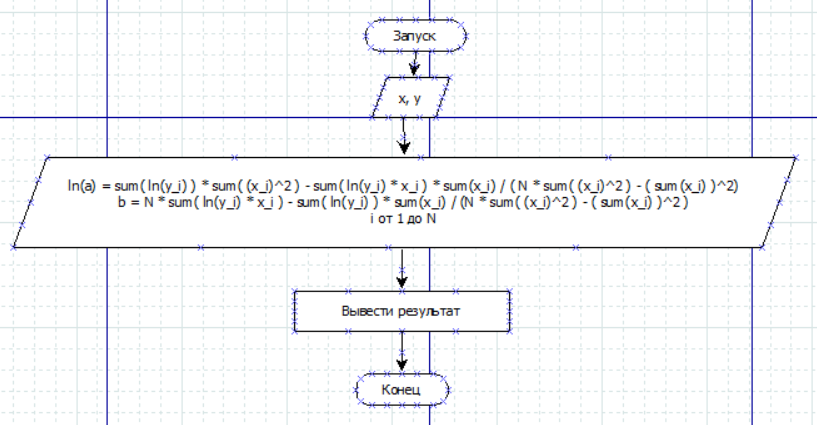
**Для многочлена степени m**

****

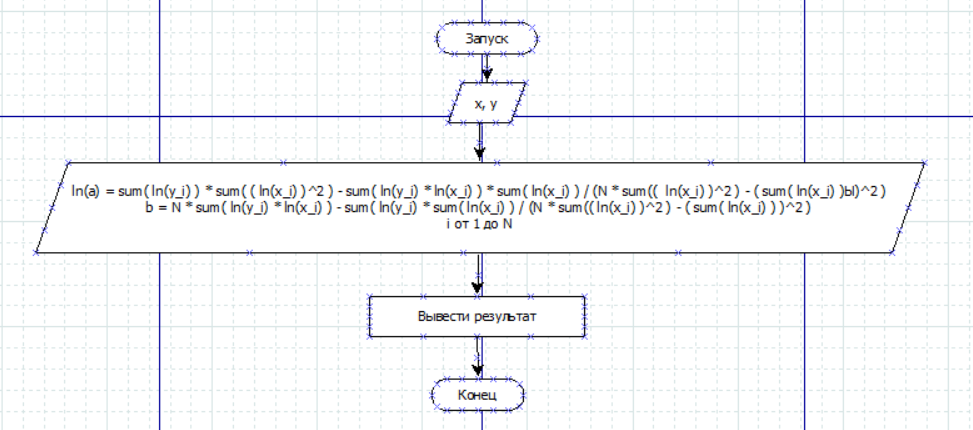
**Для логарифмической функции**

****

**Для экспоненциальной функции**

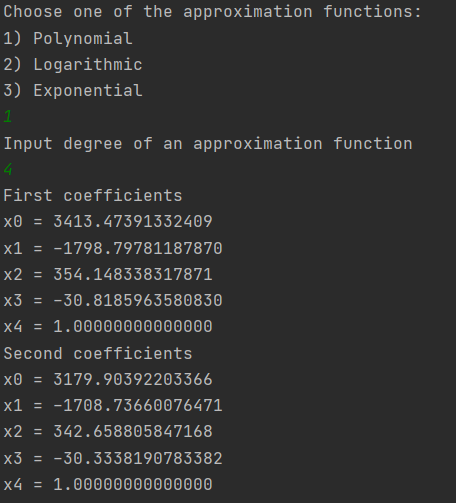
****

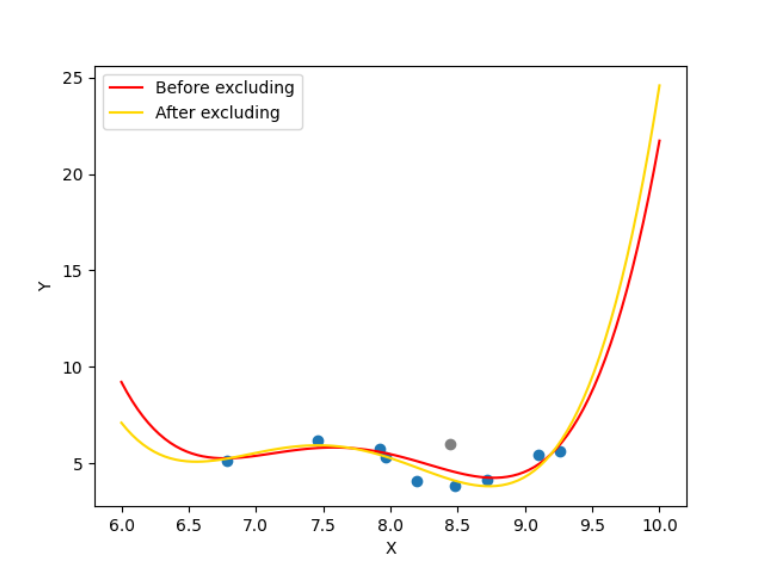
**Для степенной функции**

****

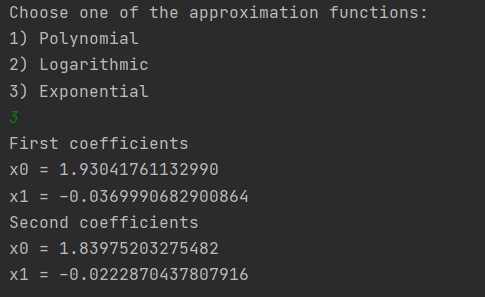
**Тесты и результаты**

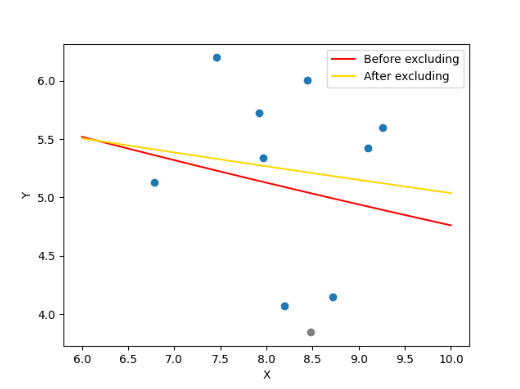
**Тест 1.**



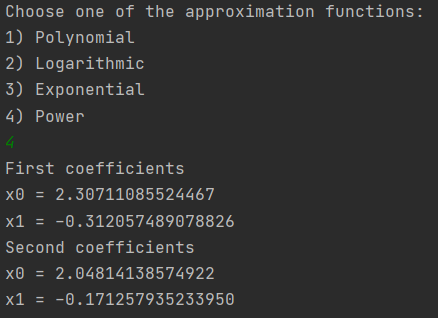


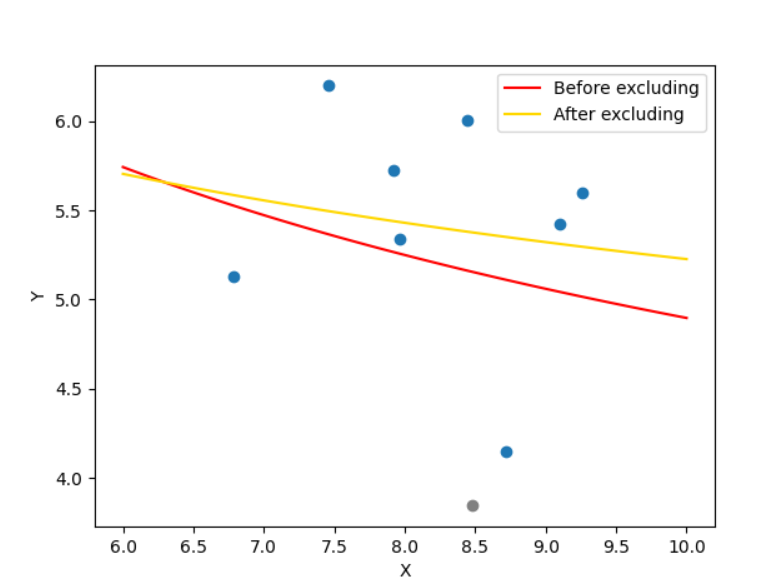
**Тест 2.**





**Тест 3**





**Вывод.**

Аппроксимация ­– метод, основанный на замене экспериментально полученных данных аналитической функцией наиболее близкой или совпадающей в узловых точках с исходными значениями. То есть задача состоит в том, чтобы данную функцию f(x) приближенно заменить некоторой функцией φ(x), такой что

Одним из способов аппроксимации функций является интерполяция. Данный способ используется, когда основная информация о приближаемой функции дается в виде таблицы значений. В итоге получается кривая, соответствующая интерполирующей функции, которая обязательно будет проходить через все точки исходных данных. Чаще всего в качестве интерполяционной функции выбирают алгебраический многочлен n-й степени, удовлетворяющий условию интерполяции:

Геометрический смысл интерполяции состоит в том, что графики функций должны проходить через все точки . В широком смысле это означает, что для заранее известных пар точек (x, y), для которых неизвестна связь в виде некоторой зависимости , ее требуется установить, так как зачастую на практике нам могут понадобиться не только значения найденные в данной области, а также и значения из любой другой, и способ нахождения таких значений называется экстраполяцией. Однако значения, полученные при помощи экстраполяции, могут рассматриваться только как вероятностные оценки, так как в действительности тенденция движения функции непостоянна.

Интерполяцией данные описываются точнее, чем при аппроксимации, но в некоторых случаях все-таки лучше применять именно аппроксимацию:

1. При значительном количестве табличных данных

2. Если интерполирующей функцией невозможно описать данные при повторении эксперимента в одних и тех же начальных условиях

3. Для сглаживания погрешностей эксперимента, так как обычно данные содержат выбросы, а это значит, что интерполяционная формула будет повторять эти значения, чего бы нам не хотелось

Аппроксимация методом наименьших квадратов дает такие математические свойства для полученной функции как:

1. Непрерывная дифференцируемость (может пригодится для минимизации значения квадратов)

2. Достаточная статистика для гауссовского распределения

3. Квадратная норма (может быть использована для доказательства сходимости функции)

Также аппроксимация может быть проведена при помощи абсолютных значений, однако функция уже не будет иметь свойства, указанные выше.