ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
НИЖЕГОРОДСКИЙ ГОССУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ им. Р. Е. АЛЕКСЕЕВА

Кафедра «Прикладная математика»

Отчет

Курсовая работа
по дисциплине «Численные Методы»

Студент

________ Шохов М.Е.
(Подпись) (Фамилия, И., О.)

17-ПМ 20.12.19
(Группа) (Дата сдачи)

Проверила

_________ Талалушкина Л.В.
(Подпись) (Фамилия, И., О.)

Отчет защищен «___»________2019г.

с оценкой__________________________

Нижний Новгород, 2019

Оглавление

1. Цель работы

Изучить и программно реализовать численные методы для кривой («ямы»), выданной преподавателем.

Численные методы – методы приближённого решения математических задач, сводящиеся к выполнению конечного числа элементарных операций над числами. В качестве элементарных операций фигурируют арифметические действия, выполняемые обычно приближённо, а также вспомогательные операции. Численные методы сводят решение математических задач к вычислениям, которые могут быть выполнены как вручную, так и с помощью вычислительных машин.

2. Оцифровка кривой

По заданию предполагалось брать каждые 2мм по оси X соответствующие им значения по оси Y. В результате было получено 95 точек. Все эти точки были записаны в файл формата «.csv»: сначала записывались значения X, ставился разделитель «;» и после него записывалось соответствующее значение Y. После этого на языке высокого уровня Python и при помощи библиотеки matplotlib была программно построена кривая. Результат видно на картинке:

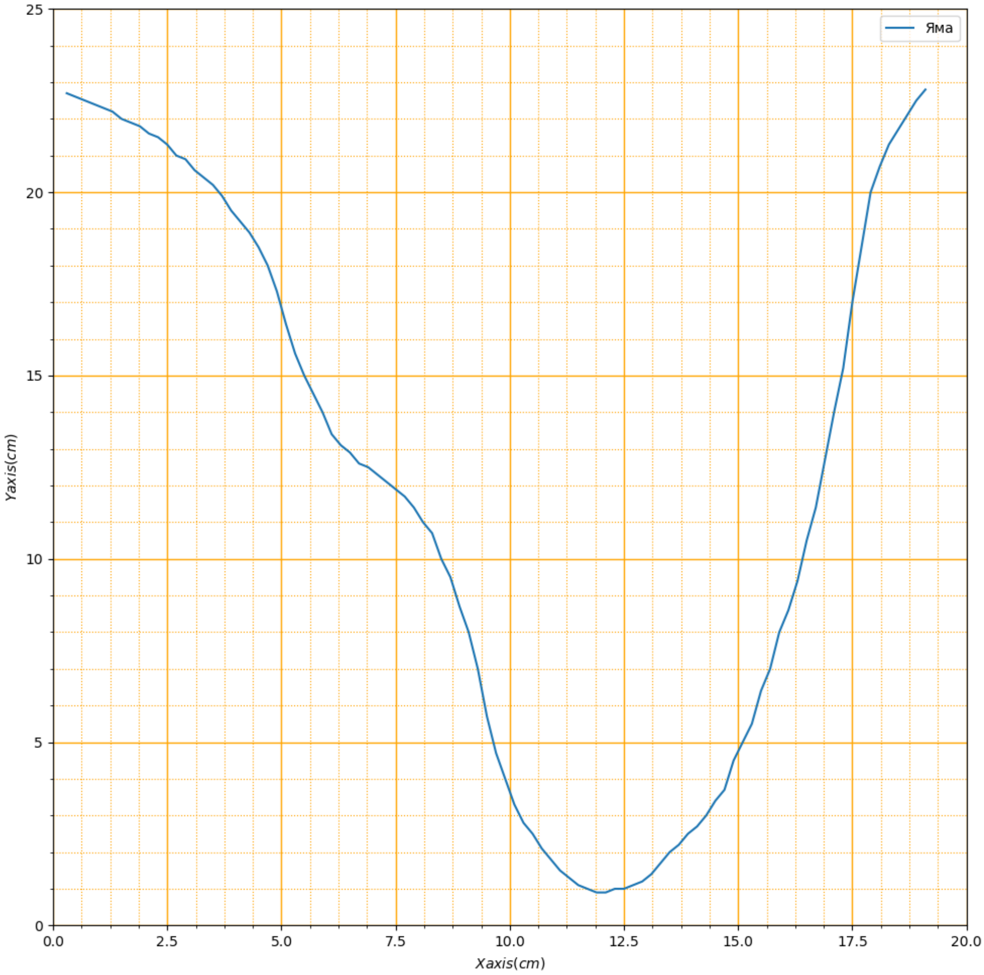

Рис.2 оцифрованная кривая

3. Польномиальная интерполяция (интерполяционные формулы Лагранжа и Ньютона)

Интерполирование — в вычислительной математике способ нахождения промежуточных значений величины по имеющемуся дискретному набору известных значений.

Интерполяционные формулы — в математике формулы, дающие приближённое выражение функции $y = f(x)$ при помощи интерполяции, то есть через интерполяционный многочлен $P_n(x)$ степени n, значения которого в заданных точках $x_0, x_1, \dots, x_n$ совпадают со значениями $y_0 \dots, y_n$ функции f в этих точках. Многочлен $P_n(x)$ определяется единственным образом, но в зависимости от задачи его удобно записывать различными по виду формулами.

Преподавателем было предложено рассмотреть два метода интерполяции функции: интерполяционный многочлен Лагранжа и интерполяционный многочлен Ньютона.

Интерполяционный 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 степели 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 X .

Интерполяционный многочлен Ньютона

Интерполяционный полином в форме Лагранжа не удобен для вычислений тем, что при увеличении числа узлов интерполяции приходится перестраивать весь полином заново.

Перепишем полином Лагранжа в другом виде:

$$P_n(x) = P_0(x) + \sum_{i=1}^{n} (P_i(x) - P_{i-1}(x))$$

где $P_i(x)$ - полиномы Лагранжа степени $i \leq n$.

Пусть $Q_i(x) = P_i(x) - P_{i-1}(x)$ (*).

Этот полином имеет степень i и обращается в нуль при $x = x_0, x = x_1, \dots, x = x_{i-1}$.

Поэтому он представим в виде:

$Q_i(x) = A_i(x - x_0)...(x - x_{i-1})$, где A_i - коэффициент при x^i. Так как x^i не входит в $P_{i-1}(x)$, то A_i совпадает с коэффициентом при x^i в полиноме $P_i(x)$. Таким образом из определения $P_i(x)$ получаем:

$$A_i = \sum_{k=0}^{i} \frac{f(x_k)}{w_{k,i}}$$

где $w_{k,i} = (x_k - x_0)...(x_k - x_{k-1})(x_k - x_{k+1})...(x_k - x_i)$.

Препишем формулу (*) в виде:

$$P_n(x) = P_{n-1} + A_n(x - x_0)...(x - x_{n-1})$$

Рекуррентно выражая $P_i(x)$ пролучам окончательную формулу для полинома:

$P_n(x) = A_0 + A_1(x\text{-}x_0) + ...+ A_n(x - x_0)...(x - x_{n-1})$

Такое представление полинома удобно для вычисления, потому что увеличение числа узлов на единицу требует добавления только одного слагаемого.

Задача

Было необходимо интерполировать заданную кривую методами Лагранжа и Ньютона:

- программно реализовать эти методы,
- применить их к 3,5 и 10 точкам кривой,
- вывести графики,
- при любом введённом X находить значение Y как на кривой, так и на интерполяционных кривых.

Поставленная задача была реализована на языке высокого уровня Python и библиотеки matplotlib.

1) 3 точки

X	0.3	10.1	19.1
Y	22.7	3.3	22.8

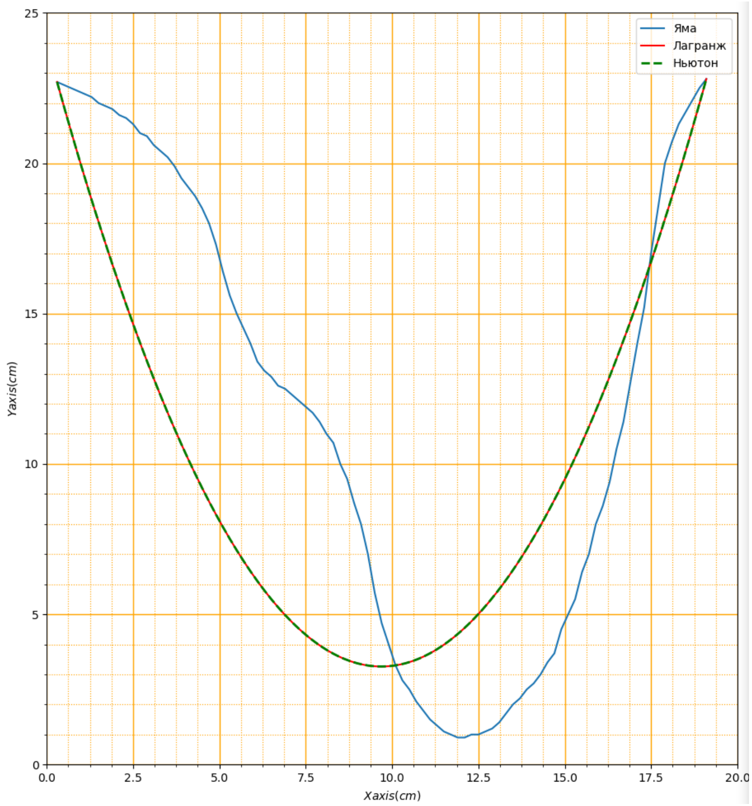

Рис.3 Интерполяция методом Лагранжа и Ньютона по 3 точкам

Рис.4 Поиск значения Y по введённому значению X

2) 5 точек

X	2.1	6.1	10.1	14.1	18.1
Y	21.6	13.4	3.3	2.7	20.7

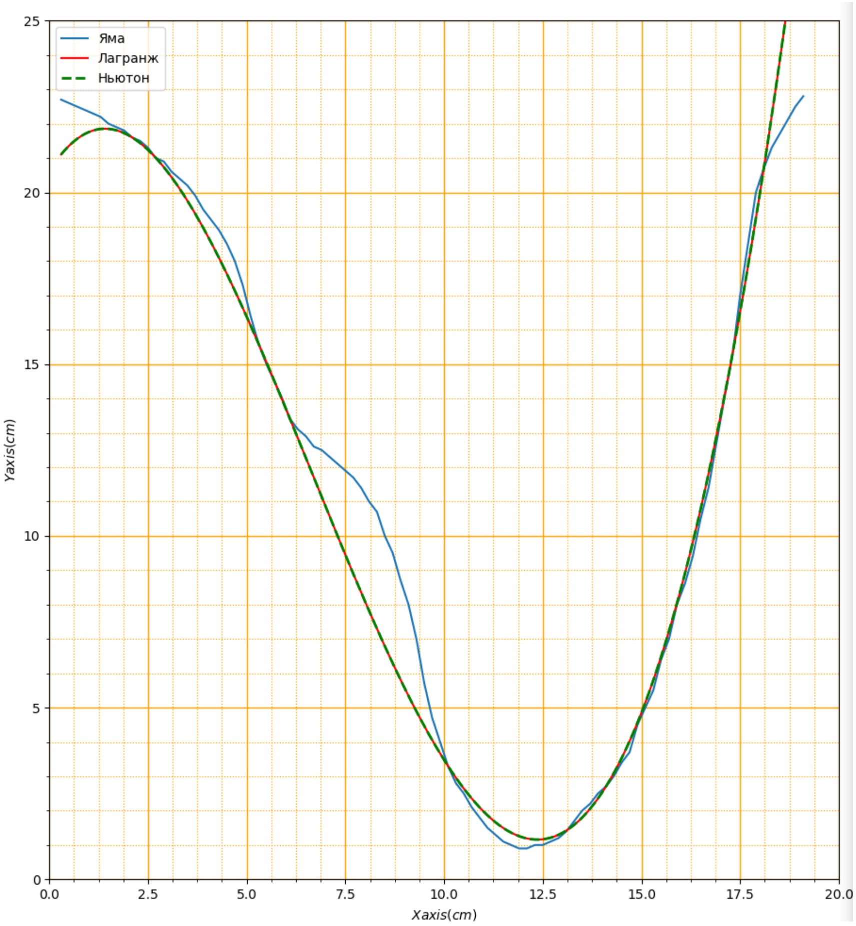

Рис.5 Интерполяция методом Лагранжа и Ньютона по 5 точкам

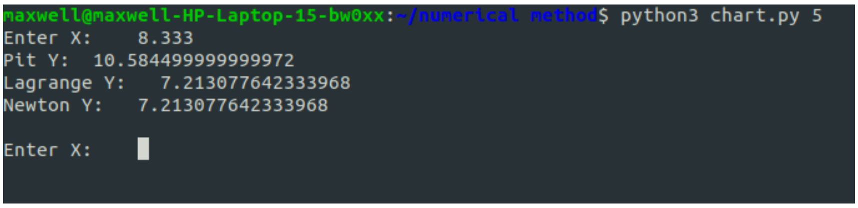

Рис.6 Поиск значения Y по введённому значению X

3) 10 точек

X	0.3	2.3	4.3	6.3	8.3	10.3	12.3	14.3	16.3	19.1
Y	22.7	21.5	18.9	13.1	10.7	2.8	1.0	3.0	9.4	22.8

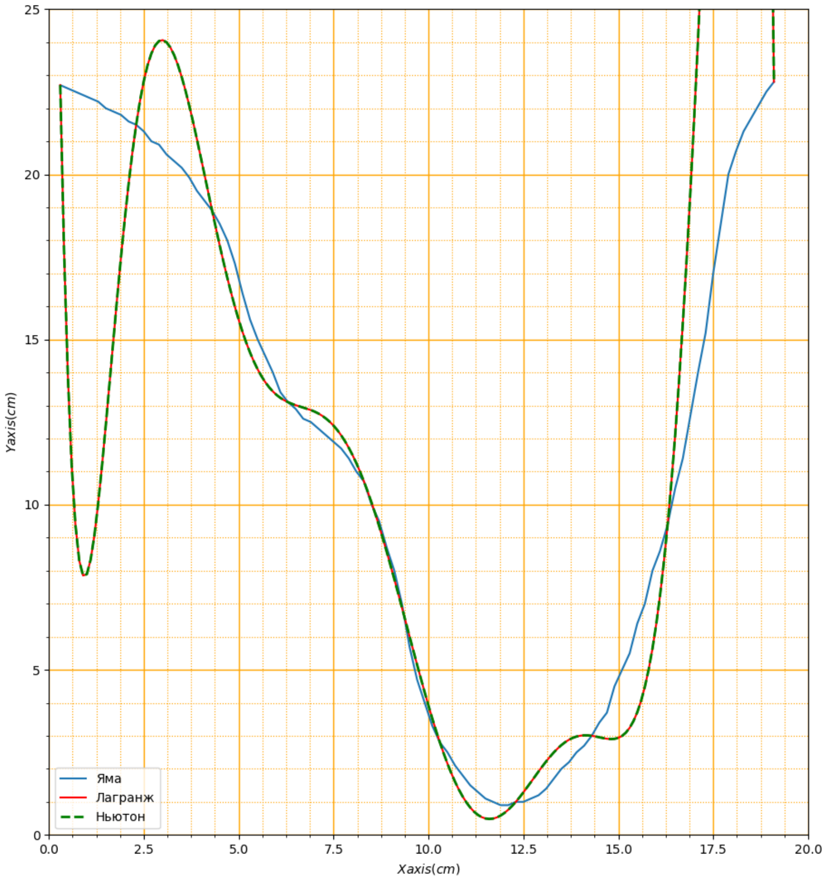

Рис.7 Интерполяция методом Лагранжа и Ньютона по 10 точкам

```
maxwell@maxwell-HP-Laptop-15-bw0xx:~/numerical method$ python3 chart.py 10
Enter X:    13
Pit Y:  1.3000000000000036
Lagrange Y:   2.0757197566090295
Newton Y:   2.0757197566090326

Enter X:
```

Рис.8 Поиск значения Y по введённому значению X

Из графиков видно, что то, как будут выглядеть кривые, построенные методами Лагранжа и Ньютона, зависит от того, какие будут выбраны точки. Если удалось подобрать точки «правильно», то интерполяционные кривые будут максимально близки к заданной кривой.

Также можно заметить, что интерполяции этих методов совпадают, максимально, приближаясь к исходной функции.

Можно сделать вывод, что чем больше будет выбрано точек и чем «удобнее» они окажутся, тем больше будет точность. Однако, следует заметить, что при совсем большом количестве точек графики интерполяций начинают «осцелировать» на краях, что делает интерполяцию методом Лагранжа и Ньютона не слишком точными.

4. Интерполяция сплайнами

Одной из основных задач численного анализа является задача об интерполяции функций. Пусть на отрезке $a \le \xi \le$ задана сетка $\omega = \{x_i : x_0 = a < x_1 < \cdots < x_i < \cdots < x_n = b\}$ и в её узлах заданы значения функции $y(x)$,

равные $y(x_0) = y_0, \cdots, y(x_i) = y_i, \cdots, y(x_n) = y_n$. Требуется

построить *интерполянту* — функцию $f(x)$, совпадающую с функцией $y(x)$ в узлах сетки:

(1)

$$f(x_i) = y_i, i = 0, 1, \cdots, n.$$

Основная цель интерполяции — получить быстрый (экономичный) алгоритм вычисления значений $f(x)$ для значений x, не содержащихся в таблице данных.

Интерполирующие функции $f(x)$, как правило строятся в виде линейных комбинаций некоторых элементарных функций:

$$f(x) = \sum_{k=0}^{N} c_k \Phi_k(x),$$

где $\{\Phi_k(x)\}$ — фиксированный линейно независимые функции, $c_0, c_1, \cdots, c_n$ — не определенные пока коэффициенты.

Из условия (1) получаем систему из $n+1$ уравнений относительно коэффициентов $\{c_k\}$:

$$\sum_{k=0}^{N} c_k \Phi_k(x_i) = y_i, i = 0, 1, \cdots, n.$$

Предположим, что система функций $\Phi_k(x)$ такова, что при любом выборе узлов $a < x_1 < \cdots < x_i < \cdots < x_n = b$ отличен от нуля определитель системы

Тогда по заданным $y_i (i = 1, \cdots, n)$ однозначно определяются коэффициенты $c_k (k = 1, \cdots, n)$.

Интерполяция кубическими сплайнами является частным случаем кусочно-полиномиальной интерполяцией. В этом специальном случае между любыми двумя соседними узлами функция интерполируется кубическим полиномом. его коэффициенты на каждом интервале определяются из условий сопряжения в узлах:

$$f_i = y_i, f'(x_i - 0) = f'(x_i + 0), f''(x_i - 0) = f''(x_i + 0), i = 1, 2, \cdots, n-1.$$

Кроме того, на границе при $x = x_0$ и $x = x_n$ ставятся условия

(2)

$$f''(x_0) = 0, f''(x_n) = 0.$$

Будем искать кубический полином в виде

(3)

$$f(x)=a_i+b_i(x-x_{i-1})+c_i(x-x_{i-1})^2+d_i(x-x_{i-1})^3, x_{i-1}\le\xi\le\xi_i.$$

Из условия $f_i=y_i$ имеем

(4)

$$f(x_{i-1})=a_i=y_{i-1}, f(x_i)=a_i+b_ih_i+c_ih_i^2+d_ih_i^3=y_i, h_i=x_i-x_{i-1}, i=1,2,\cdots,n-1.$$

Вычислим производные:

$$f'(x)=b_i+2c_i(x-x_{i-1})+3d_i(x-x_{i-1})^2, f''(x)=2c_i+6d_i(x-x_{i-1}), x_{i-1}\le\xi\le\xi_i,$$

и потребуем их непрерывности при $x=x_i$:

(5)

$$b_{i+1}=b_i+2c_ih_i+3d_ih_i^2, c_{i+1}=c_i+3d_ih_i, i=1,2,\cdots,n-1.$$

Общее число неизвестных коэффициентов, очевидно, равно $4n$, число уравнений (4) и (5) равно $4n-2$. Недостающие два уравнения получаем из условия (2) при $x=x_0$ и $x=x_n$:

$$c_1=0, c_n+3d_nh_n=0.$$

Выражение из (5) $d_i=\frac{c_{i+1}-c_i}{3h_i}$, подставляя это выражение в (4) и исключая $a_i=y_{i-1}$, получим

$$b_i=\left[\frac{y_i-y_{i-1}}{h}_i\right]-\frac{1}{3}h_i(c_{i+1}+2c_i), i=1,2,\cdots,n-1, b_n=\left[\frac{y_n-y_{n-1}}{h}_n\right]-\frac{2}{3}h_nc_n,.$$

Подставив теперь выражения для b_i, b_{i+1} и d_i в первую формулу (5), после несложных преобразований получаем для определения c_i разностное уравнение второго порядка

(6)

$$h_ic_i+2(h_i+h_{i+1})c_{i+1}+h_{i+1}c_{i+2}=3\left(\frac{y_{i+1}-y_i}{h_{i+1}}-\frac{y_i-y_{i-1}}{h_i}\right), i=1,2,\cdots,n-1.$$

С краевыми условиями

(7)

$$c_1=0, c_{n+1}=0.$$

Условие $c_{n+1} = 0$ эквивалентно условию $c_n + 3d_n h_n = 0$ и уравнению $c_{i+1} = c_i + d_i h_i$. Разностное уравнение (6) с условиями (7) можно решить методом прогонки, представив в виде системы линейных алгебраических уравнений вида $A^* x = F$, где вектор x соответствует вектору $\{c_i\}$, вектор F поэлементно равен правой части уравнения (6), а матрица A имеет следующий вид:

где $A_i = h_i, i = 2, \cdots, n, B_i = h_{i+1}, i = 1, \cdots, n-1$ и $C_i = 2(h_i + h_{i+1}), i = 1, \cdots, n$.

Задача

Было необходимо:

- программно реализовать построение кубических сплайнов исходной кривой по n, $n/3$, $n/5$ точкам, согласно алгоритму
- вывести графики

1) n точек

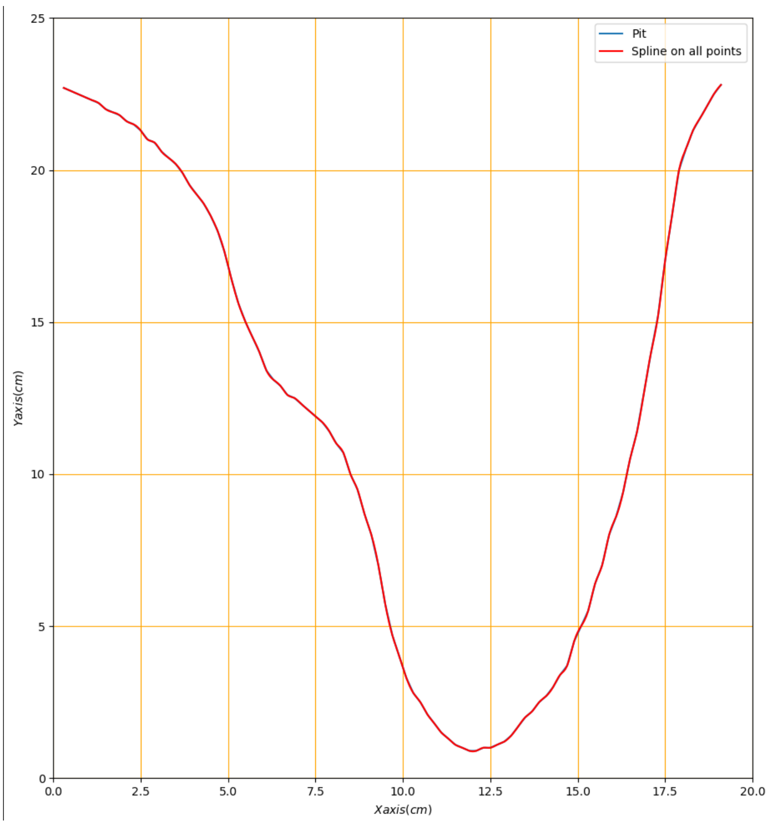

Рис.9 кубический сплайн по n точкам

2) $n/3$ точек

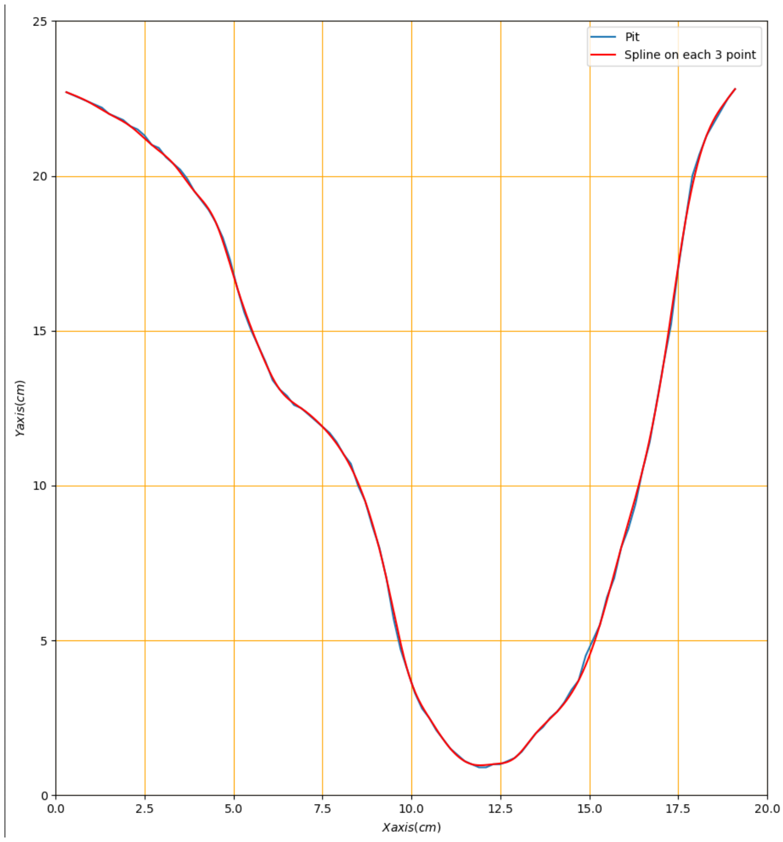

Рис.10 кубический сплайн по $n/_3$ точкам

3) $n/_5$ точек

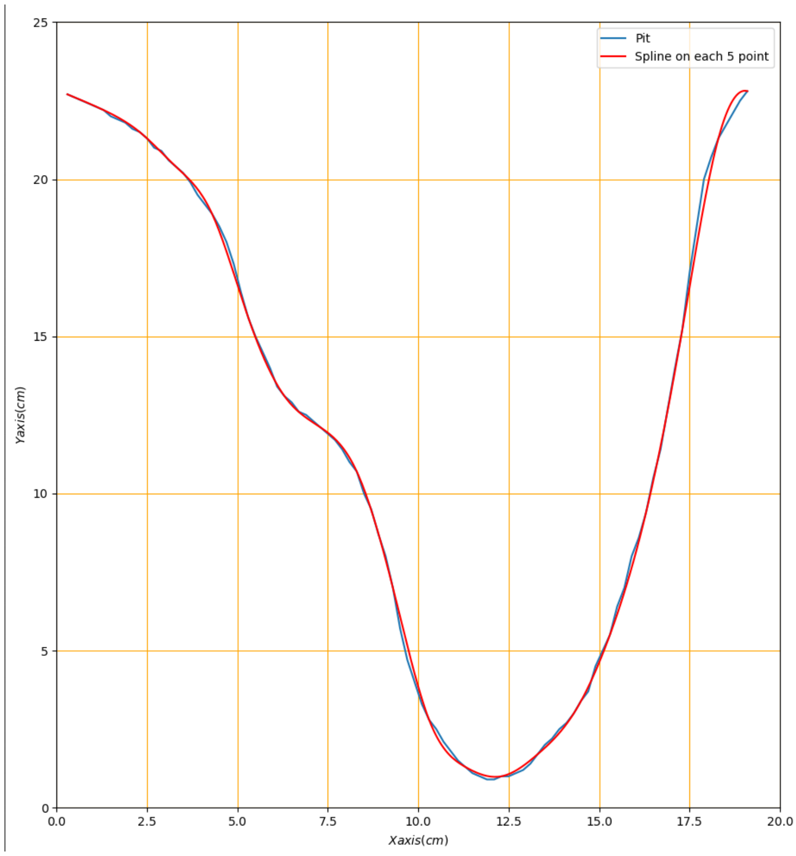

Рис.11 кубический сплайн по $n/_5$ точкам

На основании построенных графиков можно сделать вывод, что интерполяция кубическими сплайнами даёт большее приближение к исходной кривой, в отличие от методов Лагранжа и Ньютона. Однако здесь также, как и в них точность зависит напрямую от количества точек: так в случае n точек сложно разглядеть расхождения кубических сплайнов и исходной кривой, а в случае $n/_3$, $n/_5$ точек мы можем заметить лишь небольшие отклонения.

5. Численное интегрирование

1. **Метод прямоугольников** — метод численного интегрирования функции одной переменной, заключающийся в замене подынтегральной функции на многочлен нулевой степени, то есть константу, на каждом элементарном отрезке. Если рассмотреть график подынтегральной функции, то метод будет заключаться в приближённом вычислении площади под графиком суммированием площадей конечного числа прямоугольников, ширина которых будет определяться расстоянием между соответствующими соседними узлами интегрирования, а высота — значением подынтегральной функции в этих узлах. Алгебраический порядок точности равен 0. (Для формулы средних прямоугольников равен 1).

Если отрезок *[a,b]* является элементарным и не подвергается дальнейшему разбиению, значение интеграла можно найти по:

1. **Формуле левых прямоугольников**: $\int_a^b f(x)\,dx \approx f(a)(b-a).$
2. **Формуле правых прямоугольников**: $\int_a^b f(x)\,dx \approx f(b)(b-a).$
3. **Формуле прямоугольников** (средних): $\int_a^b f(x)\,dx \approx f\left(\frac{a+b}{2}\right)(b-a).$

В случае разбиения отрезка интегрирования на *n* элементарных отрезков приведённые выше формулы применяются на каждом из этих элементарных отрезков между двумя соседними узлами. В результате, получаются составные квадратурные формулы:

1. Для **левых прямоугольников**: $\int_a^b f(x)\,dx \approx \sum_{i=0}^{n-1} f(x_i)(x_{i+1}-x_i).$
2. Для **правых прямоугольников**: $\int_a^b f(x)\,dx \approx \sum_{i=1}^{n} f(x_i)(x_i-x_{i-1}).$
3. Для **средних прямоугольников**:

$$\int_a^b f(x)\,dx \approx \sum_{i=0}^{n-1} f\left(\frac{x_i+x_{i+1}}{2}\right)(x_{i+1}-x_i) = \sum_{i=1}^{n} f\left(\frac{x_{i-1}+x_i}{2}\right)(x_i-x_{i-1}).$$

2. **Метод трапеций** — метод численного интегрирования функции одной переменной, заключающийся в замене на каждом элементарном отрезке подынтегральной функции на многочлен первой степени, то есть линейную функцию. Площадь под графиком функции аппроксимируется прямоугольными трапециями. Алгебраический порядок точности равен 1.

Если отрезок *[a,b]* является элементарным и не подвергается дальнейшему разбиению, значение интеграла можно найти по формуле:

$$\int_a^b f(x)\,dx = \frac{f(a)+f(b)}{2}(b-a) + E(f), \qquad E(f) = -\frac{f''(\xi)}{12}(b-a)^3.$$

Это простое применение формулы для площади трапеции — произведение полусуммы оснований, которыми в данном случае являются значения функции в крайних точках отрезка, на высоту (длину отрезка интегрирования).

Если отрезок *[a,b]* разбивается узлами интегрирования и на каждом из элементарных отрезков применяется формула трапеций, то суммирование даст составную формулу трапеций:

$$\int_a^b f(x)\,dx \approx \sum_{i=0}^{n-1} \frac{f(x_i)+f(x_{i+1})}{2}(x_{i+1}-x_i)$$

$$x_j = a + jh, h = (b-a)/N, N\text{—четное}$$

3. **Формула Симпсона** относится к приёмам численного интегрирования. Получила название в честь британского математика Томаса Симпсона (1710—1761).

Суть метода заключается в приближении подынтегральной функции на отрезке *[a,b]* интерполяционным многочленом второй степени $p_2(x)$, то есть приближение графика функции на отрезке параболой. Метод Симпсона имеет порядок погрешности 4 и алгебраический порядок точности 3.

Формулой Симпсона называется интеграл от интерполяционного многочлена второй степени на отрезке *[a,b]*:

$$\int_a^b f(x)dx \approx \int_a^b p_2(x)dx = \frac{b-a}{6}\left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right)$$

Где *f(a), f((a+b)/2* и *f(b))* – значения функции в соответствующих точках (на концах отрезка и в его середине).

Задача

Было необходимо:

- программно реализовать методы: Симпсона, прямоугольников (левых, правых, средних), трапеций
- вычислить интеграл по введённому отрезку с заданной точностью
- сравнить методы.

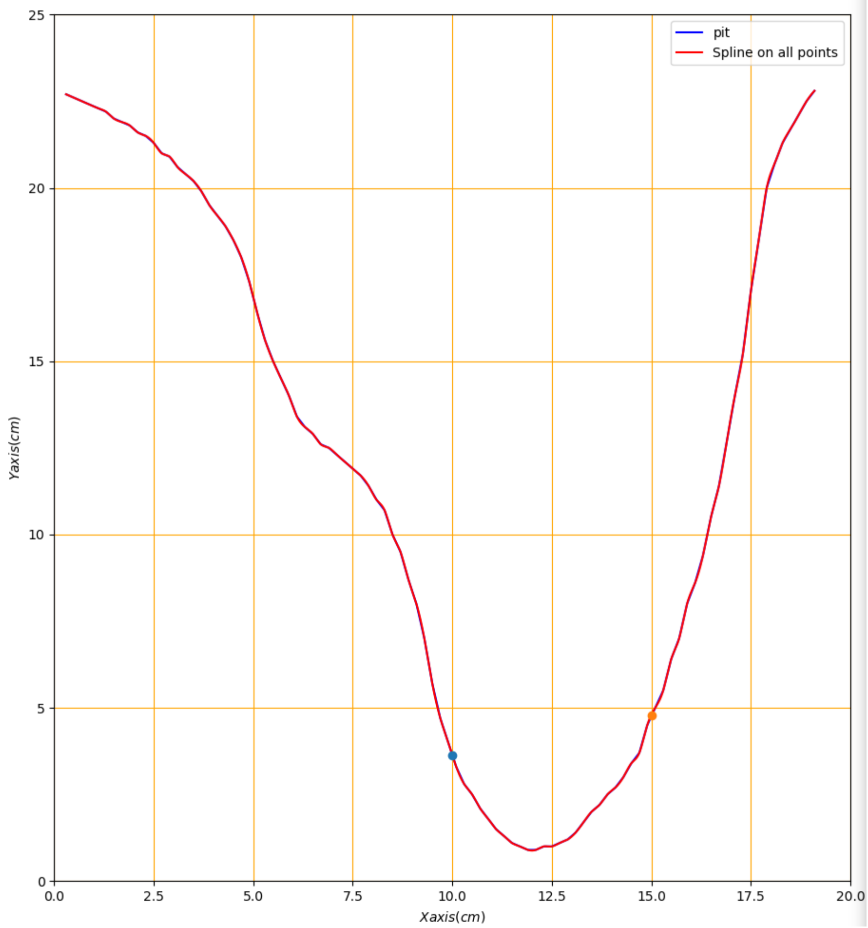

Рис.12 график на котором брался отрезок интегрирования

```
maxwell@maxwell-HP-Laptop-15-bw0xx:~/numerical method$ python3 integral.py
Enter left border in cm: 10
Enter right border in cm: 15
Enter accuracy: 0.0001

n = 22 Simpson method: 10.109187929517784
n = 35 Trapeze method: 10.139446729368538
n = 94 Left rectangular method: 10.10053776693708
n = 103 Right rectangular method: 10.15893011407134
n = 35 Central rectangular method: 10.139446729368538

Enter left border in cm: 
```

Рис.13 значения интеграла по отрезку с заданной точностью

В программе задаются: левая граница отрезка, правая и точность, с которой необходимо посчитать интеграл.

Можно сделать вывод, что алгоритм Симпсона является наиболее точным методом численного интегрирования, так как для вычисления интеграла в нём было задействовано меньшее число узлов.

6. Численное дифференцирование

Численное дифференцирование — совокупность методов вычисления значения производной дискретно заданной функции.

В основе численного дифференцирования лежит аппроксимация функции, от которой берется производная, интерполяционным многочленом. Все основные формулы численного дифференцирования могут быть получены при помощи первого интерполяционного многочлена Ньютона (формулы Ньютона для начала таблицы).

Основными задачами являются вычисление производной на краях таблицы и в её середине. Для равномерной сетки формулы численного дифференцирования «в начале таблицы» можно представить в общем виде следующим образом:

$$f_i' = \frac{1}{bh}\sum_j a_j f_{i+j} + \Delta(f),$$

Где $\Delta(f)$ — погрешность формулы. Здесь коэффициенты a_j и b зависят от степени n использовавшегося интерполяционного многочлена, то есть от необходимой точности (скорости сходимости к точному значению при уменьшении шага сетки) формулы.

Один из универсальных способов построения формул численного дифференцирования состоит в том, что по значениям функции $f(x)$ в некоторых узлах $x_0, x_1 \dots, x_N$ строят интерполяционный полином $P_N(x)$ (в форме Лагранжа или в форме Ньютона) и приближенно полагают:

$$f^{(r)}(x) \approx P_N^{(r)}(x), 0 \le r \le N$$

В ряде случаев, наряду с приближенным равенством удается (например, используя формулу Тейлора) получить точное равенство, содержащее остаточный член R (погрешность численного дифференцирования):

$$f^{(r)}(x) = P_N^{(r)}(x) + R, 0 \le r \le N$$

Степень, с которой входит величина $h=max\ h_i (h_i = x_i - x_{i-1})$ в остаточный член, называется порядком погрешности формулы численного дифференцирования. Формулы с отброшенными остаточными членами называются просто формулами численного дифференцирования.

Задача

Преподавателем было предложено программно реализовать и вывести график производную исходной кривой.

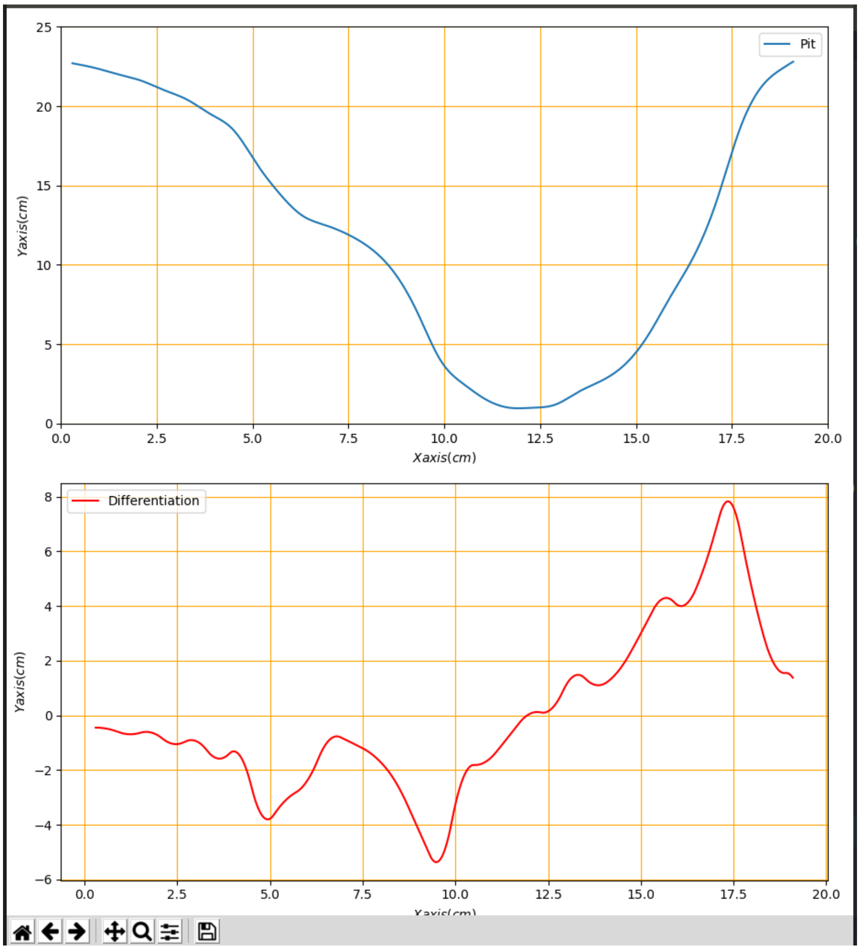

Рис.14 график кривой и её первая производная

7. Аппроксимация

Аппроксимация опытных данных – это метод, основанный на замене экспериментально полученных данных аналитической функцией наиболее близко проходящей или совпадающей в узловых точках с исходными значениями (данными полученными в ходе опыта или эксперимента). В настоящее время существует два способа определения аналитической функции:

- с помощью построения интерполяционного многочлена n-степени, который проходит непосредственно через все точки заданного массива данных. В данном случае аппроксимирующая функция представляется в виде: интерполяционного многочлена в форме Лагранжа или интерполяционного многочлена в форме Ньютона.

- с помощью построения аппроксимирующего многочлена n-степени, который проходит в ближайшей близости от точек из заданного массива данных. Таким образом, аппроксимирующая функция сглаживает все случайные помехи (или погрешности), которые могут возникать при выполнении эксперимента: измеряемые значения в ходе опыта зависят от случайных факторов, которые колеблются по своим собственным случайным законам (погрешности измерений или приборов, неточность или ошибки опыта). В данном случае аппроксимирующая функция определяется по методу наименьших квадратов.

Преподавателем было предложено рассматривать метод наименьших квадратов, для построения аппроксимации.

Метод наименьших квадратов - математический метод, основанный на определении аппроксимирующей функции, которая строится в ближайшей близости от точек из заданного массива экспериментальных данных. Близость исходной и аппроксимирующей функции F(x) определяется числовой мерой, а именно: сумма квадратов отклонений экспериментальных данных от аппроксимирующей кривой F(x) должна быть наименьшей.

Метод наименьших квадратов используется:

- для решения переопределенных систем уравнений, когда количество уравнений превышает количество неизвестных;

- для поиска решения в случае обычных (не переопределенных) нелинейных систем уравнений;

- для аппроксимации точечных значений некоторой аппроксимирующей функцией.

Аппроксимирующая функция по методу наименьших квадратов определяется из условия минимума суммы квадратов отклонений (ξ_i) расчетной аппроксимирующей функции от заданного массива экспериментальных данных. Данный критерий метода наименьших квадратов записывается в виде следующего выражения:

$$\sum_{i=1}^{N} \xi_i^{\,2} = \sum_{i=1}^{N} \left(F(x_i) - y_i\right)^2 \to \min$$

$F(x_i)$ - значения расчетной аппроксимирующей функции в узловых точках x_i,

y_i - заданный массив экспериментальных данных в узловых точках x_i.

Квадратичный критерий обладает рядом "хороших" свойств, таких, как дифференцируемость, обеспечение единственного решения задачи аппроксимации при полиномиальных аппроксимирующих функциях.

В зависимости от условий задачи аппроксимирующая функция представляет собой многочлен степени m:

$$F_m(x) = a_0 + a_1 \cdot x + \ldots + a_{m-1} \cdot x^{m-1} + a_m \cdot x^m$$

Степень аппроксимирующей функции (m) не зависит от числа узловых точек, но ее размерность должна быть всегда меньше размерности (количества точек) заданного массива экспериментальных данных.

$$1 \le m \le N - 1$$

· В случае если степень аппроксимирующей функции m=1, то мы аппроксимируем табличную функцию прямой линией (линейная регрессия).

· В случае если степень аппроксимирующей функции m=2, то мы аппроксимируем табличную функцию квадратичной параболой (квадратичная аппроксимация).

· В случае если степень аппроксимирующей функции m=3, то мы аппроксимируем табличную функцию кубической параболой (кубическая аппроксимация).

В общем случае, когда требуется построить аппроксимирующий многочлен степени m для заданных табличных значений, условие минимума суммы квадратов отклонений по всем узловым точкам переписывается в следующем виде:

$$S = \sum_{i=1}^{N} \xi_i^{\,2} = \sum_{i=1}^{N} \left(a_0 + a_1 \cdot x_i + \ldots + a_{m-1} \cdot x_i^{m-1} + a_m \cdot x_i^m - y_i\right)^2 \to \min$$

x_i, y_i - координаты узловых точек таблицы;

$a_j, (j = 0, \ldots, m)$ - неизвестные коэффициенты аппроксимирующего многочлена степени m;

N - количество заданных табличных значений.

Необходимым условием существования минимума функции является равенству нулю ее частных производных по неизвестным переменным $a_j, (j = 0, \ldots, m)$. В результате получим следующую систему уравнений:

$$\begin{cases} \dfrac{\partial S}{\partial a_0} = 2 \cdot \sum\limits_{i=1}^{N} \left(a_0 + a_1 \cdot x_i + \ldots + a_{m-1} \cdot x_i^{m-1} + a_m \cdot x_i^m - y_i \right) = 0 \\ \dfrac{\partial S}{\partial a_1} = 2 \cdot \sum\limits_{i=1}^{N} \left(a_0 + a_1 \cdot x_i + \ldots + a_{m-1} \cdot x_i^{m-1} + a_m \cdot x_i^m - y_i \right) \cdot x_i = 0 \\ \ldots \\ \dfrac{\partial S}{\partial a_m} = 2 \cdot \sum\limits_{i=1}^{N} \left(a_0 + a_1 \cdot x_i + \ldots + a_{m-1} \cdot x_i^{m-1} + a_m \cdot x_i^m - y_i \right) \cdot x_i^m = 0 \end{cases}$$

Преобразуем полученную линейную систему уравнений: раскроем скобки и перенесем свободные слагаемые в правую часть выражения. В результате полученная система линейных алгебраических выражений будет записываться в следующем виде:

$$\begin{cases} a_0 \cdot N + a_1 \cdot \sum\limits_{i=1}^{N} x_i + \ldots + a_{m-1} \cdot \sum\limits_{i=1}^{N} x_i^{m-1} + a_m \cdot \sum\limits_{i=1}^{N} x_i^m = \sum\limits_{i=1}^{N} y_i \\ a_0 \cdot \sum\limits_{i=1}^{N} x_i + a_1 \cdot \sum\limits_{i=1}^{N} x_i^2 + \ldots + a_{m-1} \cdot \sum\limits_{i=1}^{N} x_i^m + a_m \cdot \sum\limits_{i=1}^{N} x_i^{m+1} = \sum\limits_{i=1}^{N} y_i \cdot x_i \\ \ldots \\ a_0 \cdot \sum\limits_{i=1}^{N} x_i^m + a_1 \cdot \sum\limits_{i=1}^{N} x_i^{m+1} + \ldots + a_{m-1} \cdot \sum\limits_{i=1}^{N} x_i^{2\cdot m-1} + a_m \cdot \sum\limits_{i=1}^{N} x_i^{2\cdot m} = \sum\limits_{i=1}^{N} y_i \cdot x_i^m \end{cases}$$

Данная система линейных алгебраических выражений может быть переписана в матричном виде:

$$\left| \begin{matrix} N & \sum\limits_{i=1}^{N} x_i & \cdots & \sum\limits_{i=1}^{N} x_i^m \\ \sum\limits_{i=1}^{N} x_i & \sum\limits_{i=1}^{N} x_i^2 & \cdots & \sum\limits_{i=1}^{N} x_i^{m+1} \\ \vdots & \vdots & \vdots & \vdots \\ \sum\limits_{i=1}^{N} x_i^m & \sum\limits_{i=1}^{N} x_i^{m+1} & \cdots & \sum\limits_{i=1}^{N} x_i^{2\cdot m} \end{matrix} \right| \cdot \left| \begin{matrix} a_0 \\ a_1 \\ \vdots \\ a_m \end{matrix} \right| = \left| \begin{matrix} \sum\limits_{i=1}^{N} y_i \\ \sum\limits_{i=1}^{N} y_i \cdot x_i \\ \vdots \\ \sum\limits_{i=1}^{N} y_i \cdot x_i^m \end{matrix} \right|$$

В результате была получена система линейных уравнений размерностью m+1, которая состоит из m+1 неизвестных. Данная система может быть решена с помощью любого метода решения линейных алгебраических уравнений (например, методом Гаусса). В результате решения будут найдены неизвестные параметры аппроксимирующей функции, обеспечивающие минимальную сумму квадратов отклонений аппроксимирующей функции от исходных данных, т.е. наилучшее возможное квадратичное приближение. Следует помнить, что при изменении даже одного значения исходных данных все коэффициенты изменят свои значения, так как они полностью определяются исходными данными.

Задача

Программно реализовать аппроксимацию методом наименьших квадратов исходной кривой и вывести график.

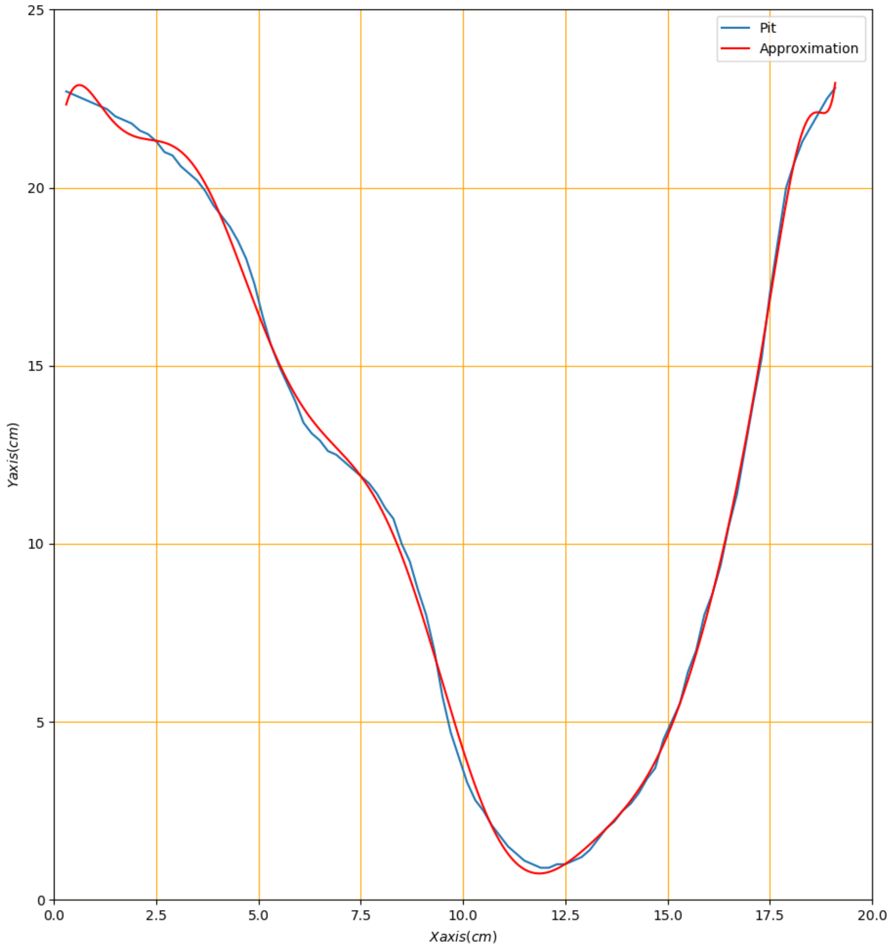

Рис.15 аппроксимация ямы многочленом 20-ой степени

На основании построенного графика можно сделать вывод, что аппроксимация приближается к исходной кривой, но не достигает той же точности, что кубический сплайн. Так же можно заметить, что с большей степенью многочлена кривая аппроксимации приближается ближе к исходной кривой.

8. Математическая модель

Основной целью курсовой работы являлось построение математической модели, такой, что при помощи полученных уравнений можно было бы воспроизвести движение мячика по исходной кривой («яме»). Движение шарика должно было быть «реалистичным», т.е. соответствовать законам физики, а так же происходить по кривой.

В качестве языка, на котором писалась бы программа был выбран язык высокого уровня Python, построение графиков – при помощи библиотеки matplotlib.

$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{x}}\right) - \frac{\partial L}{\partial x} = F_{\text{сопр.}}$ – уравнение Лагранжа 2-го рода (1).

$L = U - V$ – Лагранжиан (U – кинетическая энергия, V – потенциальная энергия).

$F_{\text{сопр.}} = -kv$ – сила сопротивления (k – коэффициент сопротивления, v - скорость).

$v = \dot{x}$ – скорость равна производной от координаты.

$L = \frac{mv^2}{2} - mgh = \frac{m\dot{x}^2}{2} - mgS(x)$, где S(x) – функция сплайна «ямы» (2).

m – масса шарика,

g – ускорение свободного падения,

k – коэффициент сопротивления.

Подставим найденное уравнение (2) в уравнение Лагранжа (1).

$\frac{\partial L}{\partial x} = -mg\frac{\partial S(x)}{\partial x}$ $\frac{\partial L}{\partial \dot{x}} = m\dot{x}$ $\frac{\partial}{\partial t}\left(\frac{\partial L}{\partial \dot{x}}\right) = m\ddot{x}$

Получаем:

$$m\ddot{x} + mg\frac{\partial S(x)}{\partial x} = -k\dot{x}$$

Поделив данное уравнение на m и перенеся слагаемое с функцией сплайна в правую сторону получим:

$$\ddot{x} = -\frac{k}{m}\dot{x} - g\frac{\partial S(x)}{\partial x}$$

Выведенное дифференциальное уравнение будет уравнением движения шарика по кривой (по «яме»).

В качестве начальных условий была выбрана точка X = 2, V = 0.

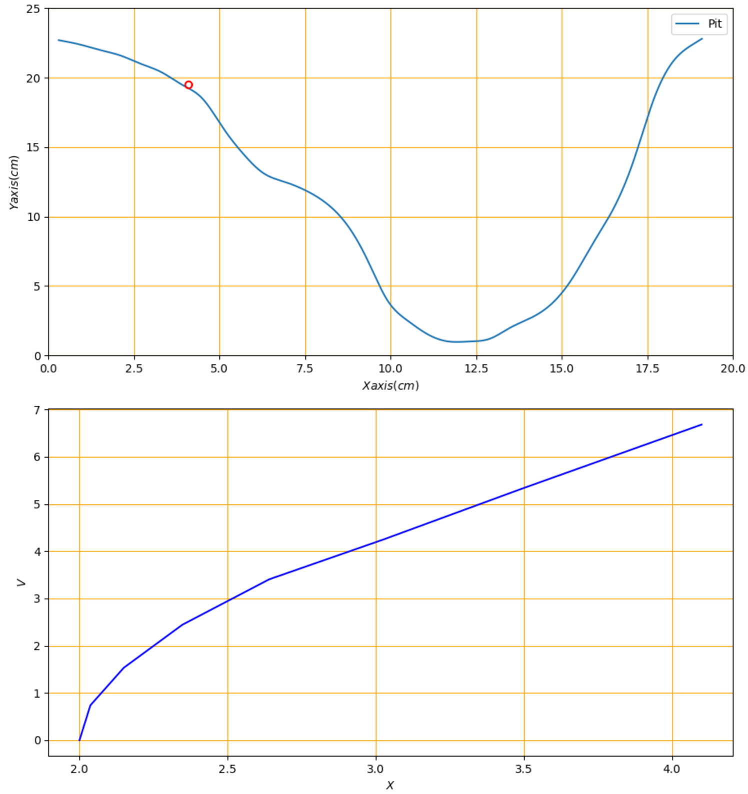

Рис.16 яма с двигающимся шариком и график скорости в момент времени t_1

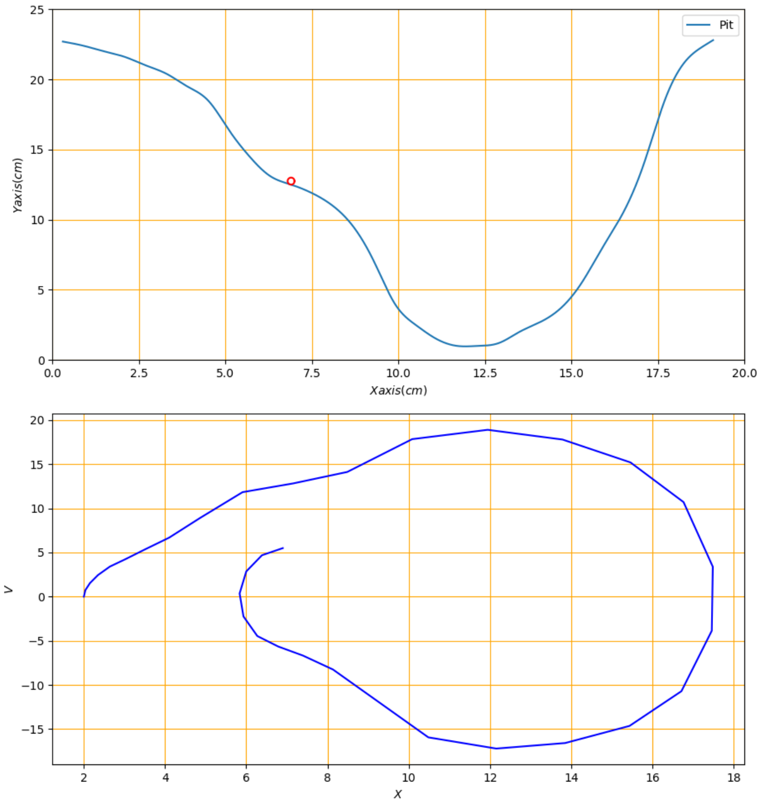

Рис.17 яма с двигающимся шариком и график скорости в момент времени t_2

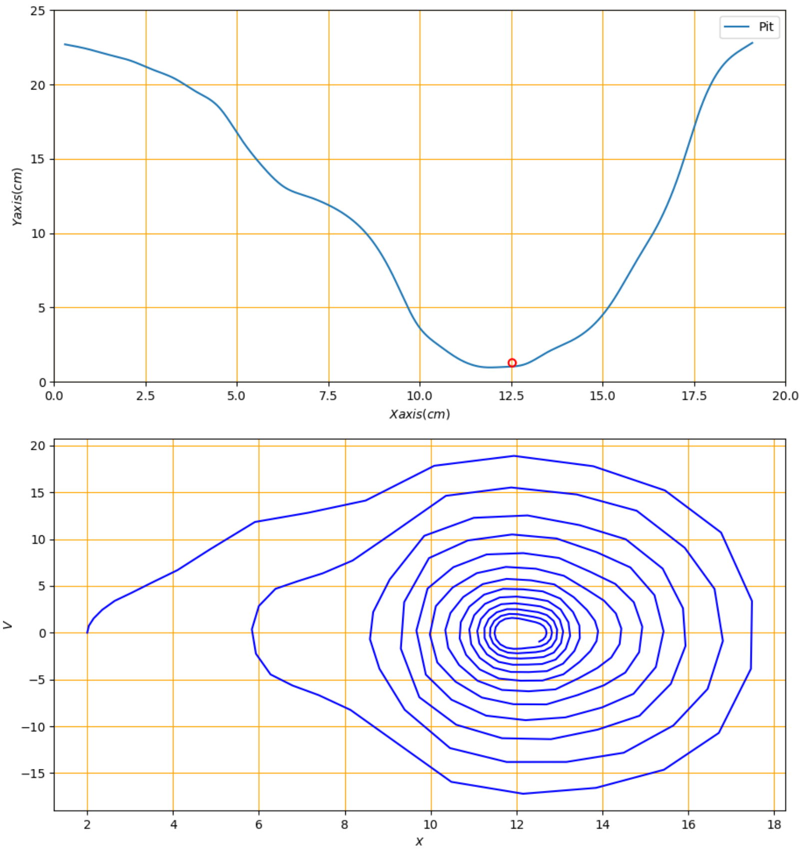

Рис.17 яма с двигающимся шариком и график скорости в момент времени t_3

9. Вывод

В течение длительного периода времени изучались численные методы, было освоено много нового материала, а также программно реализовано множество методов данной дисциплины. На основе вышенаписанного можно сделать вывод, что чем больше точек берётся при решении численными методами, тем большая точность достигается. Также было установлено опытным путём, что наилучшим приближением является сплайн интерполяция.

10. Приложение

Файл с координатами кривой coord.csv

X;Y

0.3;22.7

0.5;22.6

0.7;22.5

0.9;22.4

1.1;22.3

1.3;22.2

1.5;22.0

1.7;21.9

1.9;21.8

2.1;21.6

2.3;21.5

2.5;21.3

2.7;21.0

2.9;20.9

3.1;20.6

3.3;20.4

3.5;20.2

3.7;19.9

3.9;19.5

4.1;19.2

4.3;18.9

4.5;18.5

4.7;18.0

4.9;17.3

5.1;16.4

5.3;15.6

5.5;15.0

5.7;14.5

5.9;14.0

6.1;13.4

6.3;13.1

6.5;12.9

6.7;12.6

6.9;12.5

7.1;12.3

7.3;12.1

7.5;11.9

7.7;11.7

7.9;11.4

8.1;11.0

8.3;10.7

8.5;10.0

8.7;9.5

8.9;8.7

9.1;8.0

9.3;7.0

9.5;5.7

9.7;4.7

9.9;4.0

10.1;3.3

10.3;2.8

10.5;2.5

10.7;2.1

10.9;1.8

11.1;1.5

11.3;1.3

11.5;1.1

11.7;1.0

11.9;0.9

12.1;0.9

12.3;1.0

12.5;1.0

12.7;1.1

12.9;1.2

13.1;1.4

13.3;1.7

13.5;2.0

13.7;2.2

13.9;2.5

14.1;2.7

14.3;3.0

14.5;3.4

14.7;3.7

14.9;4.5

15.1;5.0

15.3;5.5

15.5;6.4

15.7;7.0

15.9;8.0

16.1;8.6

16.3;9.4

16.5;10.5

16.7;11.4

16.9;12.7

17.1;14.0

17.3;15.2

17.5;17.0

17.7;18.5

17.9;20.0

18.1;20.7

18.3;21.3

18.5;21.7

18.7;22.1

18.9;22.5

19.1;22.8

Код построения графика кривой

```
import numpy as np
import matplotlib.pyplot as plt
from sys import argv

def show_chart(*coords) :
    x, y = coords

    figure, axis = plt.subplots(figsize=(10, 10))

    axis.plot(x,y, label='Яма')

    axis.set_ylabel('$Y axis (cm)$')
    axis.set_xlabel('$X axis (cm)$')
    axis.set_xlim(left=0,right=20)
    axis.set_ylim(bottom=0,top=25)
    axis.minorticks_on()
    axis.grid(which = "major",color='orange',linewidth = 1)
    axis.grid(which = "minor",color='orange',linestyle = ":")
    figure.tight_layout()

    axis.legend()

    # plt.draw()
    plt.show()

if __name__ == "__main__" :
    scriptName = argv

    x = np.array([],dtype=float)
    y = np.array([],dtype=float)

    file = open("coord.csv")
```

```
    for line in file :
        sep = line.find(';')

        if line[0].isalpha() :
            continue

        x = np.append(x,float(line[0:sep]))
        y = np.append(y,float(line[sep+1:]))

    file.close()

    show_chart(x,y)
```

Код полиномиальной интерполяции (метод Ньютона и метод Лагранжа)

```
import numpy as np
import matplotlib.pyplot as plt
from sys import argv

def show_chart(*coords) :
    x, y, lx, ly, xnew, ynew, lxnew, lynew = coords

    figure, axis = plt.subplots(figsize=(10, 10))

    # plt.ion()
    axis.plot(x,y, label='Яма')

    axis.set_ylabel('$Y axis (cm)$')
    axis.set_xlabel('$X axis (cm)$')
    axis.set_xlim(left=0,right=20)
    axis.set_ylim(bottom=0,top=25)
    axis.minorticks_on()
    axis.grid(which = "major",color='orange',linewidth = 1)
    axis.grid(which = "minor",color='orange',linestyle = ":")
```

```
    figure.tight_layout()

    axis.plot(xnew,ynew,'r',label='Лагранж')
    axis.plot(lxnew,lynew,'g--',linewidth=2,label='Ньютон')

    axis.legend()

    plt.draw()
    plt.pause(0.1)

    while 1 :
        # tmp = input()
        curX = float(input("Enter X:    "))
        print(f"Pit Y:  {calculateY(x,y,curX)}")
        print(f"Lagrange Y:   {calculateY(xnew,ynew,curX)}")
        print(f"Newton Y:   {calculateY(lxnew,lynew,curX)}")
        print("")

        plt.plot(curX,calculateY(x,y,curX),'o')
        plt.plot(curX,calculateY(xnew,ynew,curX),'o')
        plt.plot(curX,calculateY(lxnew,lynew,curX),'o')
        plt.draw()

def L_polynomial(*polinomialArgs) :
    x, y, curX = polinomialArgs

    pn=0
    for k in range(len(y)):
        top=1; bottom=1
        for i in range(len(x)):
            if i == k:
                top = top*1; bottom = bottom*1
            else:
```

```
            top = top*(curX-x[i])
            bottom = bottom*(x[k]-x[i])
        pn = pn + y[k]*top/bottom
    return pn

def newtonCoefficient(x,y) :
    m = len(x)
    x = np.copy(x)
    a = np.copy(y)

    for k in range(1,m):
        a[k:m] = (a[k:m] - a[k-1])/(x[k:m] - x[k-1])

    return a

def calculateNewton(x, y, curX) :
    a = newtonCoefficient(x, y)
    n = len(x) - 1
    p = a[n]

    for k in range(1,n+1) :
        p = a[n-k] + (curX -x[n-k])*p

    return p

def calculateY(x, y, curX) :
    x1 = 0; x2 = 0
    y1 = 0; y2 = 0

    for i in range(len(x)) :
        if x[i] == curX :
            return y[i];
        if x[i] < curX and x[i+1] > curX :
```

```
            x1 = x[i]; x2 = x[i+1]
            y1 = y[i]; y2 = y[i+1]
            break;

    top1 = curX*y2 - curX*y1
    top2 = x2*y1-x1*y2
    bottom = x2 - x1
    y = (top1 + top2) / bottom
    return y

if __name__ == "__main__" :
    scriptName, amountOfPoints = argv
    amountOfPoints = int(amountOfPoints)

    x = np.array([],dtype=float)
    y = np.array([],dtype=float)

    lx = []
    ly = []

    if (amountOfPoints == 3) :
        lx.append(0.3); ly.append(22.7)
        lx.append(10.1); ly.append(3.3)
        lx.append(19.1); ly.append(22.8)
    elif (amountOfPoints == 5) :
        lx.append(2.1);  ly.append(21.6)
        lx.append(6.1);  ly.append(13.4)
        lx.append(10.1); ly.append(3.3)
        lx.append(14.1); ly.append(2.7)
        lx.append(18.1); ly.append(20.7)
    elif (amountOfPoints == 10) :
        lx.append(0.3); ly.append(22.7)
        lx.append(2.3);  ly.append(21.5)
```

```
    lx.append(4.3);  ly.append(18.9)
    lx.append(6.3);  ly.append(13.1)
    lx.append(8.3);  ly.append(10.7)
    lx.append(10.3); ly.append(2.8)
    lx.append(12.3); ly.append(1.0)
    lx.append(14.3); ly.append(3.0)
    lx.append(16.3); ly.append(9.4)
    lx.append(19.1); ly.append(22.8)

file = open("coord.csv")

for line in file :
    sep = line.find(';')

    if line[0].isalpha() :
        continue

    x = np.append(x,float(line[0:sep]))
    y = np.append(y,float(line[sep+1:]))


xnew = [i*0.1 for i in range(3,192)]
ynew = [L_polynomial(lx,ly,t) for t in xnew]

lxnew = xnew
lynew = [calculateNewton(lx,ly,t) for t in lxnew]

file.close()

show_chart(x,y, lx, ly, xnew,ynew, lxnew,lynew)
```

Код интерполяции сплайном

```
import numpy as np
import matplotlib.pyplot as plt
```

```
from sys import argv
from scipy import interpolate

class SplineCoefs :
    a = 0.0
    b = 0.0
    c = 0.0
    d = 0.0
    x = 0.0

def calculateCoefs(*args) :
    x, y, n, splineCoefs = args

    for i in range(0,n) :
        splineCoefs[i].x = x[i]
        splineCoefs[i].a = y[i]

    splineCoefs[0].c = 0

    alpha = [0.0 for i in range(0,n)]
    beta = [0.0 for i in range(0,n)]
    A = 0.0
    B = 0.0
    C = 0.0
    F = 0.0
    h_i = 0.0
    h_i1 = 0.0
    z = 0.0
    alpha[0] = 0.0
    beta[0] = 0.0
    for i in range(1,n-1) :
        h_i = x[i] - x[i - 1]
        h_i1 = x[i + 1] - x[i]
```

```
        A = h_i
        C = 2.0 * (h_i + h_i1)
        B = h_i1
        F = 6.0 * ((y[i + 1] - y[i]) / h_i1 - (y[i] - y[i - 1]) / h_i)
        z = (A * alpha[i - 1] + C)
        alpha[i] = -B / z
        beta[i] = (F - A * beta[i - 1]) / z

    splineCoefs[n-1].c = (F - A * beta[n - 2]) / (C + A * alpha[n - 2])


    for i in range(n-2,0,-1) :
        splineCoefs[i].c = alpha[i] * splineCoefs[i + 1].c + beta[i]


    for i in range(n-1,0,-1) :
        h_i = x[i] - x[i-1]
        splineCoefs[i].d = (splineCoefs[i].c - splineCoefs[i - 1].c) / h_i
        splineCoefs[i].b = h_i * (2.0 * splineCoefs[i].c + splineCoefs[i - 1].c) / 6.0 + (y[i] - y[i - 1])
/ h_i


def calculateSpline (*args) :
    curX, n, splineCoefs = args

    splineCoefsTmp = SplineCoefs()
    if curX <= splineCoefs[0].x :
        splineCoefsTmp = splineCoefs[1]
    elif curX >= splineCoefs[n-1].x :
        splineCoefsTmp = splineCoefs[n - 1]
    else :
        for j in range(0,n) :
            if curX <= splineCoefs[j].x :
                splineCoefsTmp = splineCoefs[j]
                break
```

```
    dx = (curX - splineCoefsTmp.x)
    return splineCoefsTmp.a + (splineCoefsTmp.b + (splineCoefsTmp.c / 2.0 + splineCoefsTmp.d * dx / 6.0) * dx) * dx


def show_chart(*coords) :
    x,y, xnew, ynew, lb = coords

    figure, axis = plt.subplots(figsize=(10, 10))

    plt.ion()
    axis.plot(x,y, label='Pit')

    axis.set_ylabel('$Y axis (cm)$')
    axis.set_xlabel('$X axis (cm)$')
    axis.set_xlim(left=0,right=20)
    axis.set_ylim(bottom=0,top=25)
    axis.grid(color='orange')
    figure.tight_layout()

    axis.plot(xnew,ynew,'r',label=lb)

    axis.legend()

    plt.draw()
    plt.pause(0.1)

    while 1 :
        tmp = input("Enter X:   ")
        curX = float(tmp)
        print(f"Pit Y    :  {calculateY(x,y,curX)}")
        print(f"Spline Y    :  {calculateY(xnew,ynew,curX)}")

        axis.plot(curX,calculateY(x,y,curX),'o')
```

```
        axis.plot(curX,calculateY(xnew,ynew,curX),'o')
    plt.draw()

def calculateY(x, y, curX) :
    x1 = 0; x2 = 0
    y1 = 0; y2 = 0

    for i in range(len(x)) :
        if x[i] == curX :
            return y[i];
        if x[i] < curX and x[i+1] > curX :
            x1 = x[i]; x2 = x[i+1]
            y1 = y[i]; y2 = y[i+1]
            break;

    top1 = curX*y2 - curX*y1
    top2 = x2*y1-x1*y2
    bottom = x2 - x1
    y = (top1 + top2) / bottom
    return y

if __name__ == "__main__" :
    scriptName, amountOfPoints = argv

    x = np.array([],dtype=float)
    y = np.array([],dtype=float)

    file = open("coord.csv")

    for line in file :
        sep = line.find(';')

        if line[0].isalpha() :
```

```
        continue

    x = np.append(x,float(line[0:sep]))
    y = np.append(y,float(line[sep+1:]))

file.close()

xnew = []
ynew = []
n1 = 95
n2 = int(95/3)
n3 = int(95/5)
splineCoefs = []
for i in range(0,95) :
    splineCoefs.append(SplineCoefs())
label=""

if amountOfPoints == 'all' :
    calculateCoefs(x,y,n1,splineCoefs)
    xnew = [i*0.05 for i in range(6,383)]
    ynew = [calculateSpline(t, n1, splineCoefs) for t in xnew]
    # ynew = [calc(t,x,y) for t in xnew]
    label='Spline on all points'
elif amountOfPoints == '3' :
    xTmp = x[::3]; xTmp = np.append(xTmp,x[-1])
    yTmp = y[::3]; yTmp = np.append(yTmp,y[-1])
    n2 = n2 + 2
    # print(xTmp)
    calculateCoefs(xTmp,yTmp,n2,splineCoefs)
    # xnew = [i*0.05 for i in range(6,383)]
    xnew = [i*0.01 for i in range(30,1911)]
    ynew = [calculateSpline(t, n2, splineCoefs) for t in xnew]
    label='Spline on each 3 point'
```

```
    elif amountOfPoints == '5' :
        xTmp = x[::5]; xTmp = np.append(xTmp,x[-1])
        yTmp = y[::5]; yTmp = np.append(yTmp,y[-1])
        n3 = n3 + 1
        calculateCoefs(xTmp,yTmp,n3,splineCoefs)
        xnew = [i*0.05 for i in range(6,383)]
        ynew = [calculateSpline(t, n3, splineCoefs) for t in xnew]
        label='Spline on each 5 point'

    show_chart(x,y, xnew, ynew, label)
```

Код интегрирования

```
import numpy as np
import matplotlib.pyplot as plt
from sys import argv

class SplineCoefs :
    a = 0.0
    b = 0.0
    c = 0.0
    d = 0.0
    x = 0.0

def calculateCoefs(*args) :
    x, y, n, splineCoefs = args

    for i in range(0,n) :
        splineCoefs[i].x = x[i]
        splineCoefs[i].a = y[i]

    splineCoefs[0].c = 0

    alpha = [0.0 for i in range(0,n)]
    beta = [0.0 for i in range(0,n)]
```

```
    A = 0.0
    B = 0.0
    C = 0.0
    F = 0.0
    h_i = 0.0
    h_i1 = 0.0
    z = 0.0
    alpha[0] = 0.0
    beta[0] = 0.0
    for i in range(1,n-1) :
        h_i = x[i] - x[i - 1]
        h_i1 = x[i + 1] - x[i]
        A = h_i
        C = 2.0 * (h_i + h_i1)
        B = h_i1
        F = 6.0 * ((y[i + 1] - y[i]) / h_i1 - (y[i] - y[i - 1]) / h_i)
        z = (A * alpha[i - 1] + C)
        alpha[i] = -B / z
        beta[i] = (F - A * beta[i - 1]) / z

    splineCoefs[n-1].c = (F - A * beta[n - 2]) / (C + A * alpha[n - 2])

    for i in range(n-2,0,-1) :
        splineCoefs[i].c = alpha[i] * splineCoefs[i + 1].c + beta[i]

    for i in range(n-1,0,-1) :
        h_i = x[i] - x[i-1]
        splineCoefs[i].d = (splineCoefs[i].c - splineCoefs[i - 1].c) / h_i
        splineCoefs[i].b = h_i * (2.0 * splineCoefs[i].c + splineCoefs[i - 1].c) / 6.0 + (y[i] - y[i - 1])
/ h_i

def calculateSpline (*args) :
    curX, splineCoefs = args
```

```
    n = len(splineCoefs)
    # print(len(splineCoefs))

    splineCoefsTmp = SplineCoefs()
    if curX <= splineCoefs[0].x :
        splineCoefsTmp = splineCoefs[1]
    elif curX >= splineCoefs[n-1].x :
        splineCoefsTmp = splineCoefs[n - 1]
    else :
        for j in range(0,n) :
            if curX <= splineCoefs[j].x :
                splineCoefsTmp = splineCoefs[j]
                break

    dx = (curX - splineCoefsTmp.x)
    return splineCoefsTmp.a + (splineCoefsTmp.b + (splineCoefsTmp.c / 2.0 + splineCoefsTmp.d
* dx / 6.0) * dx) * dx


def show_chart(*coords) :
    x,y, xnew, ynew, lb, splineCoefs = coords

    figure, axis = plt.subplots(figsize=(10, 10))

    plt.ion()
    axis.plot(x,y,'b',label='pit')
    axis.set_ylabel('$Y axis (cm)$')
    axis.set_xlabel('$X axis (cm)$')
    axis.set_xlim(left=0,right=20)
    axis.set_ylim(bottom=0,top=25)
    axis.grid(color='orange')
    figure.tight_layout()

    axis.plot(xnew,ynew,'r',label=lb)
```

```
    axis.legend()

    # plt.show()
    plt.draw()
    plt.pause(0.1)

    while 1 :
        a = float(input('Enter left border in cm: '))
        b = float(input('Enter right border in cm: '))
        eps = float(input('Enter accuracy: '))
        print("")

        first = xnew.index(a)
        second = xnew.index(b)
        amountOfPoints = int((b - a)/0.01)

        calculateIntegral(xnew,amountOfPoints,a,b,first,splineCoefs,eps)

        axis.plot(a,ynew[first],'o')
        axis.plot(b,ynew[second],'o')
    plt.draw()

def simpson(*args) :
    xnew,n,a,b,first,splineCoefs = args
    h = (b - a)/n
    f0 = calculateSpline(a, splineCoefs)
    fn = calculateSpline(b, splineCoefs)
    firstSum = 0
    secondSum = 0
    m = int(n/2)

    xTmp = []
```

```
    for i in range(0, n+1) :
        xTmp.append(a + h*i)

    for i in range(1,m) :
        firstSum += calculateSpline(xTmp[2*i], splineCoefs)

    for i in range(1,m+1) :
        secondSum += calculateSpline(xTmp[2*i-1], splineCoefs)

    return (h/3)*(f0 + 2*firstSum + 4*secondSum + fn)

def trapeze(*args) :
    xnew,n,a,b,first,splineCoefs = args
    h = (b - a)/n
    f0 = calculateSpline(a, splineCoefs)
    fn = calculateSpline(b, splineCoefs)
    sum = 0

    xTmp = []
    for i in range(0, n+1) :
        xTmp.append(a + h*i)

    for i in range(1,n) :
        sum += calculateSpline(xTmp[i], splineCoefs)

    return (h/2)*(f0 + 2*sum + fn)


def leftRect(*args) :
    xnew,n,a,b,first,splineCoefs = args
    h = (b - a)/n
    sum = 0
```

```
    xTmp = []
    for i in range(0, n+1) :
        xTmp.append(a + h*i)

    for i in range(0,n) :
        sum += calculateSpline(xTmp[i], splineCoefs)

    return h * sum

def rightRect(*args) :
    xnew,n,a,b,first,splineCoefs = args
    h = (b - a)/n
    sum = 0

    xTmp = []
    for i in range(0, n+1) :
        xTmp.append(a + h*i)

    for i in range(1,n+1) :
        sum += calculateSpline(xTmp[i], splineCoefs)

    return h * sum

def centralRect(*args) :
    xnew,n,a,b,first,splineCoefs = args
    f0 = calculateSpline(a, splineCoefs)/2
    fn = calculateSpline(b, splineCoefs)/2
    h = (b - a)/n
    sum = 0

    xTmp = []
    for i in range(0, n+1) :
        xTmp.append(a + h*i)
```

```
    for i in range(1,n) :
        sum += calculateSpline(xTmp[i], splineCoefs)

    return h * (f0 + sum + fn)

def checkForCondition(*args) :
    type,xnew,step,a,b,first,splineCoefs,eps = args

    n = 10
    res2 = 0
    res1 = 0
    while True :
        if type == "simpson" :
            res1 = simpson(xnew,n,a,b,first,splineCoefs)
        elif type == "trapeze" :
            res1 = trapeze(xnew,n,a,b,first,splineCoefs)
        elif type == "leftRect" :
            res1 = leftRect(xnew,n,a,b,first,splineCoefs)
        elif type == "rightRect" :
            res1 = rightRect(xnew,n,a,b,first,splineCoefs)
        elif type == "centralRect" :
            res1 = centralRect(xnew,n,a,b,first,splineCoefs)

        diff = abs(res1 - res2)

        res2 = res1
        n = n + 2 if type == "simpson" else n + 1

        if eps*15 >= diff and type == "simpson" :
            print(f"n = {n-2}",end=" ")
            break
        elif eps*3 >= diff and type != "simpson" :
```

```
            print(f"n = {n-1}",end=" ")
            break

    return res1

def calculateIntegral(*args) :
    xnew,step,a,b,first,splineCoefs,eps = args

    print(f"Simpson method: {checkForCondition('simpson',xnew,step,a,b,first,splineCoefs,eps)}")
    print(f"Trapeze method: {checkForCondition('trapeze',xnew,step,a,b,first,splineCoefs,eps)}")
    print(f"Left rectangular method: {checkForCondition('leftRect',xnew,step,a,b,first,splineCoefs,eps)}")
    print(f"Right rectangular method: {checkForCondition('rightRect',xnew,step,a,b,first,splineCoefs,eps)}")
    print(f"Central rectangular method: {checkForCondition('centralRect',xnew,step,a,b,first,splineCoefs,eps)}")
    print("")

if __name__ == "__main__" :
    x = np.array([],dtype=float)
    y = np.array([],dtype=float)

    file = open("coord.csv")

    for line in file :
        sep = line.find(';')

        if line[0].isalpha() :
            continue

        x = np.append(x,float(line[0:sep]))
        y = np.append(y,float(line[sep+1:]))
```

```
    file.close()

    n1 = 95
    splineCoefs = []
    for i in range(0,95) :
        splineCoefs.append(SplineCoefs())

    calculateCoefs(x,y,n1,splineCoefs)
    xnew = [round(i*0.001,2) for i in range(300,19101)]
    ynew = [calculateSpline(t, splineCoefs) for t in xnew]
    label='Spline on all points'

    show_chart(x,y, xnew, ynew, label, splineCoefs)
```

Код дифференцирования

```
import numpy as np
import matplotlib.pyplot as plt
import math
from scipy import interpolate

def show_chart(*coords) :
    x,y, xnew, ynew = coords

    # figure = plt.figure(figsize=(10, 10))
    # axis = plt.subplots(2,1,1)
    # ax = plt.subplots(2,1,2)
    figure, axis = plt.subplots(figsize=(10, 10))

    # plt.ion()
    axis.plot(x,y, label='Pit')

    axis.set_ylabel('$Y axis (cm)$')
    axis.set_xlim(left=0,right=20)
    axis.set_xlabel('$X axis (cm)$')
```

```
    axis.set_ylim(bottom=0,top=25)
    axis.grid(color='orange')
    axis.legend()

    figure.tight_layout()




    fig, ax = plt.subplots(figsize=(10, 10))
    ax.set_ylabel('$Y axis (cm)$')
    ax.set_xlabel('$X axis (cm)$')
    # ax.set_xlim(left=0,right=20)
    # ax.set_ylim(bottom=-10,top=10)
    ax.grid(color='orange')
    fig.tight_layout()

    ax.plot(xnew,ynew,'r',label="Differentiation")
    # ax.plot(xxnew,yynew,'b',label="Differentiation x")
    ax.legend()



    plt.show()
    # plt.pause(0.1)

def calculateY(x, y, curX) :
    x1 = 0; x2 = 0
    y1 = 0; y2 = 0

    for i in range(len(x)) :
        if x[i] == curX :
            return y[i];
        if x[i] < curX and x[i+1] > curX :
            x1 = x[i]; x2 = x[i+1]
```

```
            y1 = y[i]; y2 = y[i+1]
            break;

    top1 = curX*y2 - curX*y1
    top2 = x2*y1-x1*y2
    bottom = x2 - x1
    y = (top1 + top2) / bottom
    return y

def calculateDiff(x, y) :
    derivative = []
    first = (-3*y[0] + 4*y[1] - y[2])/(2 * 0.01)
    derivative.append(first);

    for i in range(1,len(x)-1) :
        value = (y[i+1] - y[i-1])/(2 * 0.01)
        derivative.append(value)

    last = (y[len(y)-1-2] - 4*y[len(y)-1-1] + 3*y[len(y)-1])/(2 * 0.01)
    derivative.append(last);

    return derivative

def spline(x,y,curX) :
    tck = interpolate.splrep(x,y)
    return interpolate.splev(curX,tck)

if __name__ == "__main__" :
    x = np.array([],dtype=float)
    y = np.array([],dtype=float)

    file = open("spline_coords.csv")
```

```
    for line in file :
        sep = line.find(';')

        if line[0].isalpha() :
            continue

        x = np.append(x,float(line[0:sep]))
        y = np.append(y,float(line[sep+1:]))

    file.close()

    # n = 95
    xSpl = [i*0.01 for i in range(30,1911)]
    # xSpl = [i*0.1 for i in range(3,191)]
    ySpl = [spline(x, y, t) for t in xSpl]

    xTmp, yTmp = xSpl, calculateDiff(xSpl,ySpl)
    # xTmp = np.delete(xTmp,[0,len(xTmp)-1])

    # xTmp2, yTmp2 = xSpl, calculateSecondDiff(xSpl,ySpl)
    # xTmp2, yTmp2 = xSpl, calculateDiffX(xSpl,ySpl)
    # xTmp2, yTmp2 = xSpl, calculateDiff(xTmp,yTmp)
    # xTmp2 = np.delete(xTmp2,[0,len(xTmp2)-1])

    show_chart(xSpl,ySpl, xTmp, yTmp)
```

Код аппроксимации

```
import numpy as np
import matplotlib.pyplot as plt

def show_chart(*coords) :
    x,y, xnew, ynew = coords

    figure, axis = plt.subplots(figsize=(10, 10))
```

```
    plt.ion()
    axis.plot(x,y, label='Pit')

    axis.set_ylabel('$Y axis (cm)$')
    axis.set_xlabel('$X axis (cm)$')
    axis.set_xlim(left=0,right=20)
    axis.set_ylim(bottom=0,top=25)
    axis.grid(color='orange')
    figure.tight_layout()

    axis.plot(xnew,ynew,'r',label="Approximation")

    axis.legend()

    plt.draw()
    plt.pause(0.1)

    while 1 :
        tmp = input("Enter X:   ")
        curX = float(tmp)
        print(f"Pit Y   :  {calculateY(x,y,curX)}")
        print(f"Approximation Y    :  {calculateY(xnew,ynew,curX)}")

        axis.plot(curX,calculateY(x,y,curX),'o')
        axis.plot(curX,calculateY(xnew,ynew,curX),'o')
    plt.draw()

def calculateY(x, y, curX) :
    x1 = 0; x2 = 0
    y1 = 0; y2 = 0

    for i in range(len(x)) :
```

```
        if x[i] == curX :
            return y[i];
        if x[i] < curX and x[i+1] > curX :
            x1 = x[i]; x2 = x[i+1]
            y1 = y[i]; y2 = y[i+1]
            break;

    top1 = curX*y2 - curX*y1
    top2 = x2*y1-x1*y2
    bottom = x2 - x1
    y = (top1 + top2) / bottom
    return y

def buildSystem(x,y,m,n) :
    a, b = [], []

    for i in range(0,m+1) :
        row = []
        for j in range(0,m+1) :
            if (i == 0 and j == 0) :
                row.append(n)
                continue


            xsum = 0
            for k in range(0,n) :
                xsum += x[k]**(j+i)
            row.append(xsum)

        a.append(row)

        ysum = 0
        for c in range(0,n) :
            ysum += y[c] * x[c] ** i
```

```
        b.append(ysum)

    aTmp = np.array(a)
    bTmp = np.array(b)
    return aTmp, bTmp

def calculatePolynomial(a, curX) :
    value = 0
    for i in range(0,m+1):
        value += a[i] * curX**i

    return value

if __name__ == "__main__" :
    x = np.array([],dtype=float)
    y = np.array([],dtype=float)

    file = open("coord.csv")

    for line in file :
        sep = line.find(';')

        if line[0].isalpha() :
            continue

        x = np.append(x,float(line[0:sep]))
        y = np.append(y,float(line[sep+1:]))

    file.close()

    n = 95; m = 20
    xTmp, yTmp = buildSystem(x,y,m,n)
    a = np.linalg.solve(xTmp,yTmp)
```

```
    xnew = [i*0.01 for i in range(30,1911)]
    ynew = [calculatePolynomial(a, t) for t in xnew]

    show_chart(x,y, xnew, ynew)
```

Код движения шарика по кривой

```
import numpy as np
import matplotlib.pyplot as plt
import math
from scipy import interpolate

def show_chart(*coords) :
    x,y, = coords

    figure, axis = plt.subplots(2,1,figsize=(10, 10))

    plt.ion()
    axis[0].plot(x,y, label='Pit')

    axis[0].set_ylabel('$Y axis (cm)$')
    axis[0].set_xlim(left=0,right=20)
    axis[0].set_xlabel('$X axis (cm)$')
    axis[0].set_ylim(bottom=0,top=25)
    axis[0].grid(color='orange')
    axis[0].legend()

    axis[1].set_ylabel('$V$')
    axis[1].set_xlabel('$X$')
    axis[1].grid(color='orange')

    figure.tight_layout()

    plt.draw()
```

```
plt.pause(0.1)

x0 = float(input("Enter initial X:    "))
v0 = float(input("Enter initial V:    "))
curX = x0
curV = v0
time = 0
v = []
coords = []
a,b = 0.1,0.25
c = np.linspace(0,2*math.pi,100)
while 1 :
    # circle = plt.Circle((curX, spline(x,y,curX)+0.2), 0.2, color='red')
    circle = axis[0].plot(curX + a*np.cos(c),spline(x,y,curX) + b*np.sin(c) + b,color="red")
    # circle = axis[0].plot(curX,spline(x,y,curX),'bo')
    coords.append(curX); v.append(curV)
    print(f"Current x: {curX}", end="   ")
    print(f"Current v: {curV}")
    line = axis[1].plot(coords,v,'b')

    # axis[0].add_artist(circle)
    time += 0.5
    tmp = currentPosition(x,y,curX,curV,time)
    curX, curV = tmp

    plt.pause(0.001)
    # circle.remove()
    axis[0].lines[1].remove()
    axis[1].lines[0].remove()

    if curX < x[0] or curX > x[-1] :
        print("ERROR: The ball passed the boundary")
        break
```

```
    esc = input("Enter something to exit: ")
    plt.draw()

def calculateDiffInPoint(x,y,curX) :
    value = ( spline(x,y,curX+0.01) - spline(x,y,curX-0.01) )/(2*0.01)
    return value

def currentPosition(xArr,yArr,curX,curZ,t) :
    dt = 0.1

    k1 = dt* f(t,curX,curZ)
    m1 = dt* g(t,curX,curZ,xArr,yArr)

    k2 = dt* f(t+dt,curX+k1/2,curZ+m1/2)
    m2 = dt* g(t+dt,curX+k1/2,curZ+m1/2,xArr,yArr)

    k3 = dt* f(t+dt,curX+k2/2,curZ+m2/2)
    m3 = dt* g(t+dt,curX+k2/2,curZ+m2/2,xArr,yArr)

    k4 = dt* f(t+dt,curX+k3,curZ+m3)
    m4 = dt* g(t+dt,curX+k3,curZ+m3,xArr,yArr)

    deltaX = (1/6)*(k1 + 2*k2 + 2*k3 + k4)
    deltaZ = (1/6)*(m1 + 2*m2 + 2*m3 + m4)

    newX = curX + deltaX
    newZ = curZ + deltaZ

    return newX, newZ

def g(t,x,z,xArr,yArr) :
    g = 9.8
```

```
    coeffOfFriction = 0.2
    m = 1

    currentValueOfSpline = calculateDiffInPoint(xArr,yArr,x)
    alpha = -g
    # beta = -g*coeffOfFriction
    beta = -(coeffOfFriction/m)

    # value = alpha*currentValueOfSpline +
(beta*currentValueOfSpline)/((1+currentValueOfSpline**2)**(1/2))
    value = alpha* currentValueOfSpline + beta*z
    return value

def f(t,x,z) :
    value = z
    return value

def spline(x,y,curX) :
    tck = interpolate.splrep(x,y)
    return interpolate.splev(curX,tck)

if __name__ == "__main__" :
    x = np.array([],dtype=float)
    y = np.array([],dtype=float)

    file = open("spline_coords.csv")

    for line in file :
        sep = line.find(';')

        if line[0].isalpha() :
            continue
```

```
    x = np.append(x,float(line[0:sep]))
    y = np.append(y,float(line[sep+1:]))

file.close()


xSpl = [i*0.01 for i in range(30,1911)]
ySpl = [spline(x, y, t) for t in xSpl]


show_chart(xSpl,ySpl)
```