

Практикум 10. Формула Тейлора для функций одной переменной.

Цель работы – изучить понятие формулы Тейлора для функций, познакомиться с разложением по формуле Тейлора основных элементарных функций, научиться многочлен Тейлора функции одной переменной; научиться в среде python производить запись в текстовый файл, работать с массивами ячеек, использовать python-функцию с переменным числом аргументом; использовать средства пакета matplotlib для иллюстрации формулы Тейлора.

Продолжительность работы – 4 часа.

Оборудование, приборы, инструментарий – работа выполняется в компьютерном классе с использованием языка программирования Python и интерактивного блокнота jupyter-notebook.

Порядок выполнения

1. Упражнения выполняются параллельно с изучением теоретического материала.
2. После выполнения каждого упражнения результаты заносятся в отчёт.
3. При выполнении упражнений в случае появления сообщения об ошибке рекомендуется сначала самостоятельно выяснить, чем оно вызвано, и исправить команду; если многократные попытки устранить ошибку не привели к успеху, то проконсультироваться с преподавателем.
4. Дома доделать упражнения из раздела «Краткие теоретические сведения и практические упражнения», которые Вы не успели выполнить во время аудиторного занятия.
5. После выполнения упражнений выполнить дополнительные упражнения для самостоятельной работы и ответить на контрольные вопросы и (см. ниже).
6. Подготовить отчёт, в который включить упражнения из раздела «Краткие теоретические сведения и практические упражнения» и упражнения для самостоятельной работы. Отчёт представить в виде документа Microsoft Word, имя файла (пример): mp_10_Ivanov_P_01_s_1 (факультет_группа_Фамилия студента_Инициал_номер лабораторной,

семестр). Отчет должен содержать по каждому выполненному упражнению: № упражнения, текст упражнения; команды, скопированные из командного окна, с комментариями к ним и результаты их выполнения, включая построенные графики; тексты М-сценариев и М-функций; выводы.

Краткие теоретические сведения и практические упражнения

1. Открытие текстового файла.

Работа с текстовыми файлами состоит из трёх этапов: открытие файла, считывание или запись информации, закрытие файла. В Python есть встроенная функция `open()`. С ее помощью можно открыть любой файл на компьютере. Технически Python создает на его основе объект.

Синтаксис следующий:

```
f = open(file_name, access_mode)
```

Где,

- `file_name` – имя открываемого файла
- `access_mode` – режим открытия файла. Он может быть: для чтения, записи и т. д. По умолчанию используется режим чтения (r), если другое не указано. Далее полный список режимов открытия файла

Режим	Описание
r	Только для чтения.
w	Только для записи. Создаст новый файл, если не найдет с указанным именем.
rb	Только для чтения (бинарный).
wb	Только для записи (бинарный). Создаст новый файл, если не найдет с указанным именем.
r+	Для чтения и записи.
rb+	Для чтения и записи (бинарный).
w+	Для чтения и записи. Создаст новый файл для записи, если не найдет с указанным именем.

Режим	Описание
wb+	Для чтения и записи (бинарный). Создаст новый файл для записи, если не найдет с указанным именем.
a	Откроет для добавления нового содержимого. Создаст новый файл для записи, если не найдет с указанным именем.
a+	Откроет для добавления нового содержимого. Создаст новый файл для чтения записи, если не найдет с указанным именем.
ab	Откроет для добавления нового содержимого (бинарный). Создаст новый файл для записи, если не найдет с указанным именем.
ab+	Откроет для добавления нового содержимого (бинарный). Создаст новый файл для чтения записи, если не найдет с указанным именем.

2. Заккрытие текстового файла.

После открытия файла в Python его нужно закрыть. Таким образом освобождаются ресурсы и убирается мусор. Python автоматически закрывает файл, когда объект присваивается другому файлу.

Существуют следующие способы:

Способ №1. Проще всего после открытия файла закрыть его, используя метод `close()`.

```
f = open('example.txt', 'r')
# работа с файлом
f.close()
```

После закрытия этот файл нельзя будет использовать до тех пор, пока заново его не открыть.

Способ №2. Также можно написать `try/finally`, которое гарантирует, что если после открытия файла операции с ним приводят к исключениям, он закроется автоматически. Без него программа завершается некорректно. Вот как сделать это исключение:

```
f = open('example.txt', 'r')
try:
    # работа с файлом
finally:
    f.close()
```

Файл нужно открыть до инструкции `try`, потому что если инструкция `open` сама по себе вызовет ошибку, то файл не будет открываться для последующего закрытия. Этот метод гарантирует, что если операции над файлом вызовут исключения, то он закроется до того, как программа остановится.

Способ №3. Инструкция `with`

Еще один подход — использовать инструкцию `with`, которая упрощает обработку исключений с помощью инкапсуляции начальных операций, а также задач по закрытию и очистке. В таком случае инструкция `close` не нужна, потому что `with` автоматически закроет файл.

Вот как это реализовать в коде.

```
with open('example.txt') as f:  
    # работа с файлом
```

3. Чтение и запись файлов в Python

Функция `read()` используется для чтения содержимого файла после открытия его в режиме чтения (`r`).

Синтаксис

```
file.read(size)
```

Где, `file` — объект файла, `size` — количество символов, которые нужно прочитать. Если не указать, то файл прочитается целиком.

Пример

```
>>> f = open('example.txt', 'r')  
>>> f.read(7)  # чтение 7 символов из example.txt  
'This is '
```

Интерпретатор прочитал 7 символов файла и если снова использовать функцию `read()`, то чтение начнется с 8-го символа.

```
>>> f.read(7)  # чтение следующих 7 символов  
' a text'
```

Функция `readline()` используется для построчного чтения содержимого файла. Она используется для крупных файлов. С ее помощью можно получать доступ к любой строке в любой момент.

Пример

Создадим файл `test.txt` с несколькими строками:

```
This is line1.  
This is line2.  
This is line3.
```

Посмотрим, как функция `readline()` работает в `test.txt`.

```
>>> x = open('test.txt', 'r')  
>>> x.readline() # прочитать первую строку  
This is line1.  
>>> x.readline(2) # прочитать вторую строку  
This is line2.  
>>> x.readlines() # прочитать все строки  
['This is line1.', 'This is line2.', 'This is line3.']
```

Обратите внимание, как в последнем случае строки отделены друг от друга.

Функция `write()` используется для записи в файлы Python, открытые в режиме записи. Если пытаться открыть файл, которого не существует, в этом режиме, тогда будет создан новый.

Синтаксис

```
file.write(string)
```

Пример

Предположим, файла `xyz.txt` не существует. Он будет создан при попытке открыть его в режиме чтения.

```
>>> f = open('xyz.txt', 'w') # открытие в режиме записи  
>>> f.write('Hello \n World') # запись Hello World в файл  
Hello  
World  
>>> f.close() # закрытие файла
```

Пример. Создание заголовка таблицы.

```
f = open('t1.txt', 'w');
# Печать в файл заголовка таблицы
f.write('ТАБЛИЦА ЗНАЧЕНИЙ ФУНКЦИИ sin x\n');
# Печать в файл шапки таблицы
f.write('_____\n')
f.write('|      x      |      sin x      |\n')
f.write('_____\n')
f.close();;
```

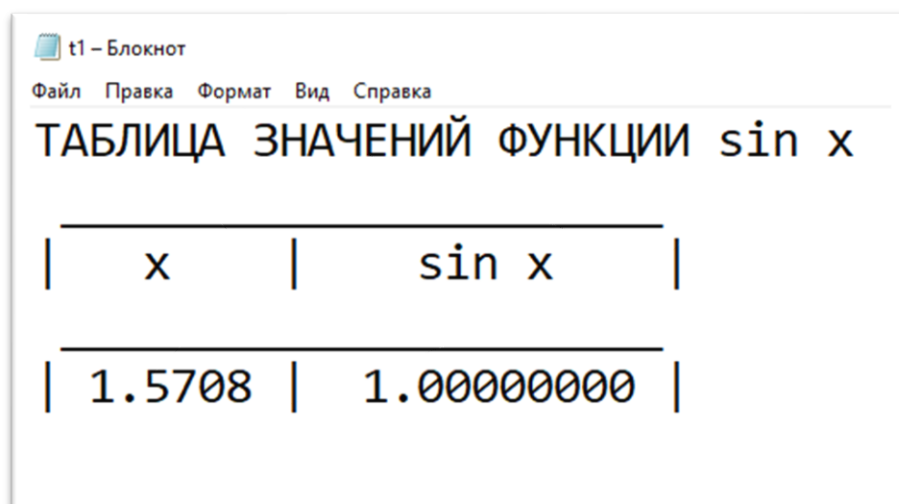
Занесение в файл **чисел или значений переменных** требуют форматного вывода. Схема использования `write` при работе с числовыми переменными такова:

```
f.write('формат'.format(список переменных))
```

Пример. После исполнения программы

```
import numpy as np
f = open('t1.txt', 'a');
x = np.pi/2;
y = np.sin(x);
f.write("| {x:7.4f} | {y:11.8f} |\n".format(x=x, y=y));
f.close();
```

в текстовом файле запишется



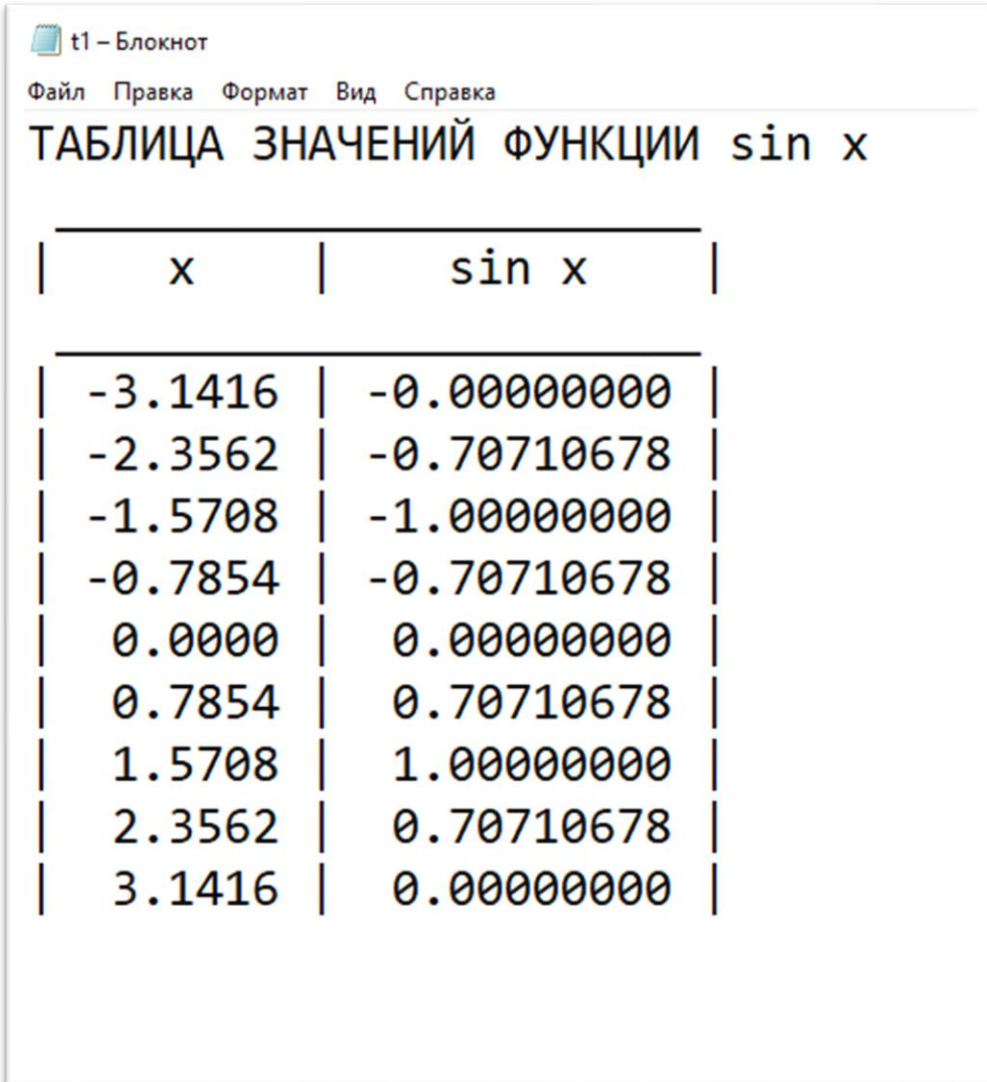
Последовательность `7.4f` задаёт формат вывода переменной `x`, которая расположена на первом месте в списке вывода. Цифра 6 обозначает, что всего на вывод переменной `x` отводится 7 позиций, цифра 4 после разделителя-точки

обеспечивает точность отображения результате – 4 цифры после десятичной точки, спецификатор `f` указывает на то, что следует вывести число в формате с плавающей точкой.

Пример. После исполнения программы

```
import numpy as np
f = open('t1.txt', 'a');
for x in np.linspace(-np.pi, np.pi, 9):
    y=np.sin(x);
    f.write("| {x:7.4f} | {y:11.8f} |\n".format(x=x, y=y));
f.close();
```

в текстовом файле запишется таблица



x	sin x
-3.1416	-0.00000000
-2.3562	-0.70710678
-1.5708	-1.00000000
-0.7854	-0.70710678
0.0000	0.00000000
0.7854	0.70710678
1.5708	1.00000000
2.3562	0.70710678
3.1416	0.00000000

Упражнение 1. Вычислить значения первых пяти производных функции $y = \cos x$ в точке 1, результат записать в текстовый файл в виде таблицы: первый

столбец – номер производной, второй – значение. Сделать заголовок и шапку таблицы.

4. Формула Тейлора для функций.

Если функция $f(x)$ имеет производную порядка n в точке x_0 , то для неё можно составить многочлен *Тейлора*

$$P_n(x) = b_n(x - x_0)^n + b_{n-1}(x - x_0)^{n-1} + \dots + b_0, \text{ где}$$

$$b_k = \frac{f^{(k)}(x_0)}{k!}.$$

Для выполнения следующего задания удобно использовать python-функции с переменным числом аргументов. Функция также может принимать переменное количество позиционных аргументов, тогда перед именем ставится *:

```
>>> def func(*args):
...     return args
...
>>> func(1, 2, 3, 'abc')
(1, 2, 3, 'abc')
>>> func()
()
>>> func(1)
(1,)
```

Как видно из примера, `args` - это кортеж из всех переданных аргументов функции, и с переменной можно работать также, как и с кортежем.

Функция может принимать и произвольное число именованных аргументов, тогда перед именем ставится **:

```
>>> def func(**kwargs):
...     return kwargs
...
>>> func(a=1, b=2, c=3)
{'a': 1, 'c': 3, 'b': 2}
>>> func()
{}
>>> func(a='python')
{'a': 'python'}
```

В переменной `kwargs` у нас хранится словарь, с которым мы, опять-таки, можем делать все, что нам заблагорассудится.

5. Формула Тейлора в Python

Синтаксис

```
from scipy.interpolate import approximate_taylor_polynomial
approximate_taylor_polynomial(f, x, degree, scale, order=None)
```

Входные параметры

`f` – вызываемая, функция, для которой ищется многочлен Тейлора. Должна принимать вектор значений `x`

`x` – скаляр, точка, в которой рассчитывается многочлен.

`degree` – целое, степень многочлена Тейлора

`scale` – скаляр, ширина интервала, используемая для расчета многочлена Тейлора.

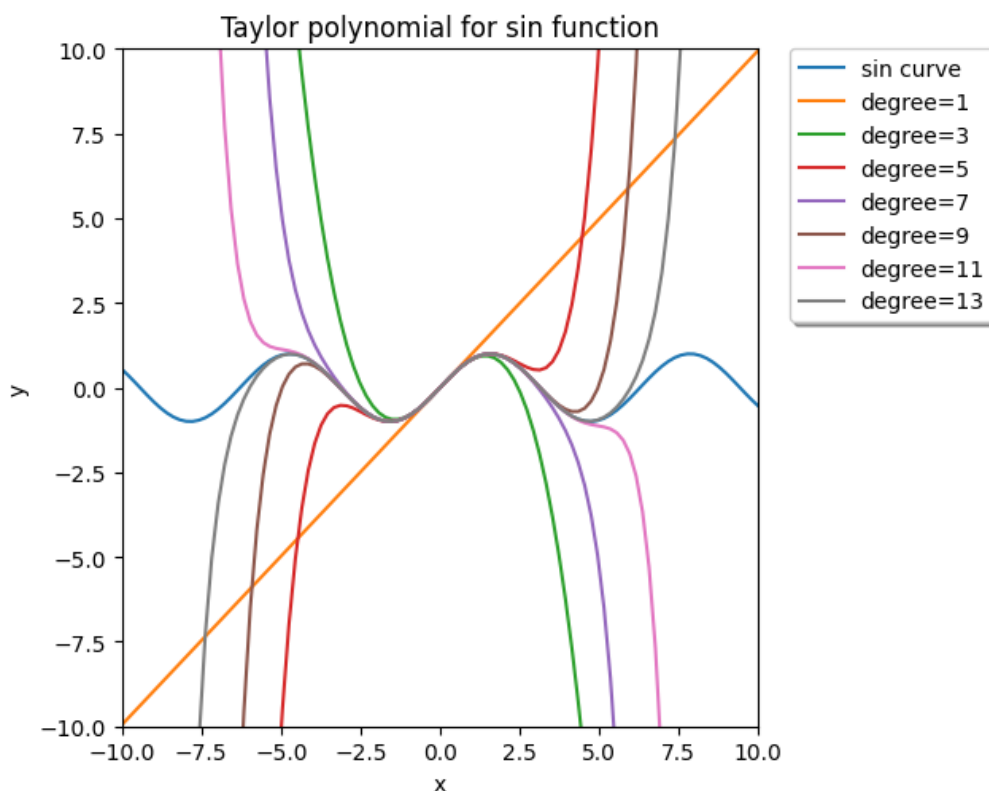
`order` – целое или `None`, опциональный, степень многочлена, используемого для подгонки. При значении `None` используется параметр `degree`

Возвращаемые параметры

`p` – объект класса `poly1d`, многочлен Тейлора.

Пример

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import approximate_taylor_polynomial
x = np.linspace(-10.0, 10.0, num=100)
plt.plot(x, np.sin(x), label="sin curve")
for degree in np.arange(1, 15, step=2):
    sin_taylor = approximate_taylor_polynomial(np.sin, 0, degree,
                                              1, order=degree + 2)
    plt.plot(x, sin_taylor(x), label=f"degree={degree}")
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left',
          borderaxespad=0.0, shadow=True)
plt.tight_layout()
plt.axis([-10, 10, -10, 10])
plt.title('Taylor polynomial for sin function')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```



Упражнение 2. Создать М-функцию, зависящую от функции, точки, и числа n - количества производных, выходным аргументом которой является вектор длины $n+1$, первый элемент которого – значение функции в точке, остальные – значения производных. Проверить работу М-функции для функций $\cos x$, $\sin x$, $\ln(1+x)$ в точке $x_0 = 0$.

Упражнение 3. Создать М-функцию, входным аргументом которой является массив, в первой ячейке которого записана x_0 - точка, в окрестности которой происходит разложение по формуле Тейлора, во второй число n - порядок, до которого происходит разложение, в третьей – вектор длины $(n+1)$, составленный из значений функции и производных в точке x_0 . Выходной аргумент – многочлен Тейлора.

Для следующих функций в указанной точке x_0 построить многочлены Тейлора порядка n_1 , n_2 , n_3 в одном графическом окне построить графики функции и многочленов Тейлора:

- а) $f(x) = \sin x$, $x_0 = 0$, $n_1 = 1$, $n_2 = 2$, $n_3 = 3$;
- б) $f(x) = \cos x$, $x_0 = 0$, $n_1 = 1$, $n_2 = 2$, $n_3 = 4$;
- в) $f(x) = \ln(4+x)$, $x_0 = 1$, $n_1 = 1$, $n_3 = 2$, $n_3 = 4$.

Задания для самостоятельной работы

1. Выполнить упражнения из раздела «Краткие теоретические сведения и практические упражнения», которые не успели сделать в аудитории.

2. Самостоятельно выполнить упражнения:

Упражнение С1. Для указанной функции в точке x_0 построить многочлены Тейлора порядка n_1 и n_2 , в одном графическом окне построить графики функции и многочленов Тейлора: $f(x) = \sqrt{4+x}$, $x_0 = 0$, $n_1 = 1$, $n_2 = 2$, $n_3 = 4$.

Упражнение С2. Используя функцию `approximate_taylor_polynomial`, найдите:

а) многочлены Тейлора 7-го порядка в точке $x_0 = 0$ для функций $\sin x$, $\cos x$, e^x , $\ln(1+x)$, $\operatorname{tg} x$;

б) многочлены Тейлора 5-го порядка в точке $x_0 = \pi/2$ для функций $\sin x$, $\cos x$.

3. Ответить на контрольные вопросы:

1. Как можно использовать многочлены Тейлора для приближенных оценок значений функции?
2. Как изменяется ошибка в оценках значений функции с помощью формулы Тейлора при увеличении порядка многочлена?
3. Как изменяется ошибка в оценках значений функции с помощью формулы Тейлора при удалении значений аргумента от точки разложения x_0 ?

Список рекомендуемой литературы

1. Официальная документация по языку программирования Python
<https://docs.python.org/3/>.

2. Официальная документация к библиотеке numpy
<https://numpy.org/doc/stable/index.html>.

3. Официальная документация к библиотеке matplotlib
<https://matplotlib.org/stable/api/index>.