Кривая подгонки

Кривая подгонки или аппроксимации - это математическая модель или функция, которая приближенно описывает закономерности или зависимости в наборе данных. Процесс аппроксимации используется для нахождения наилучшего соответствия между математической моделью и реальными данными.

Основная цель кривой подгонки состоит в том, чтобы создать простую, но достаточно точную функцию, которая может быть использована для предсказания значений вне известных точек данных. Это особенно полезно, когда есть необходимость анализа или предсказания трендов, а также при работе с экспериментальными данными.

Процесс создания кривой подгонки включает в себя выбор подходящей математической формулы или модели, которая наилучшим образом соответствует данным. Обычно это может быть линейная или полиномиальная функция, экспоненциальная или логарифмическая кривая, или другие типы функций в зависимости от характера данных.

Методы подгонки, такие как метод наименьших квадратов, используются для определения параметров модели так, чтобы минимизировать разницу между предсказанными значениями и реальными данными. Полученная кривая подгонки может затем использоваться для анализа трендов, прогнозирования будущих значений или визуализации зависимостей в данных.

Конкретные примеры кривых подгонки могут включать следующие случаи:

1. **Линейная регрессия (частный случай многочлена):**
   - *Модель:* $y = mx + b$
   - *Применение:* Подгонка прямой линии к данным, где предполагается линейная зависимость между переменными.
2. **Полиномиальная аппроксимация:**
   - *Модель:* $y = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$
   - *Применение:* Аппроксимация кривой полиномом для более гибкого учета сложных зависимостей.
3. **Экспоненциальная кривая:**
   - *Модель:* $y = ae^{bx}$
   - *Применение:* Подгонка экспоненциальной кривой к данным с экспоненциальным ростом или затуханием.
4. **Логарифмическая кривая:**
   - *Модель:* $y = a\ln(x) + b$
   - *Применение:* Аппроксимация кривой логарифмической функцией, когда наблюдается логарифмическая зависимость.
5. **Парабола (частный случай многочлена):**
   - *Модель:* $y = ax^2 + bx + c$
   - *Применение:* Подгонка параболы к данным с квадратичной зависимостью.
6. **Кривая мощности (степенная функция, частный случай многочлена):**
   - *Модель:* $y = ax^b$
   - *Применение:* Подгонка степенной функции к данным с асимптотическими свойствами.

Эти примеры демонстрируют различные формы математических моделей, которые могут быть использованы для аппроксимации разнообразных данных. Выбор конкретной модели зависит от природы данных и требований конкретной задачи.

Ошибки

Обозначения

- n - количество наблюдений,
- y_i - фактическое значение,
- $\hat{y}_i$ - предсказанное значение для функции апроксимации $f(x_i)$.

Абсолютные ошибки

1. **Абсолютная ошибка (Absolute Error):**

   $$E_A(x_i) = |\hat{y}_i - y_i|$$

   Эта формула представляет собой абсолютное отклонение предсказанных значений от фактических значений в точке данных x_k.

2. **Максимальная ошибка (Infinity Norm):**

   $$E_\infty(f) = \max_{1<i<n} |\hat{y}_i - y_i|$$

   Эта формула представляет собой максимальное отклонение предсказанных значений от фактических значений на точках данных.

3. **Средняя ошибка (Mean Absolute Error):**

   $$E_1(f) = \mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n} |\hat{y}_i - y_i|$$

Здесь мы находим среднюю абсолютную разницу между предсказанными значениями и фактическими значениями.

4. **Среднеквадратичная ошибка (Mean Squared Error):**

$$E_2(f) = \text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$

Эта формула представляет собой среднеквадратичное отклонение предсказанных значений от фактических значений.

5. **Корень из среднеквадратичной ошибки (Root Mean Squared Error, RMSE):** Квадратный корень из MSE.

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}$$

Относительные ошибки

1. **Относительная ошибка (Relative Error):**

$$E_R(x_i) = \frac{|\hat{y}_i - y_i|}{|y_i|} \times 100\%$$

Эта формула измеряет относительное отклонение предсказанных значений от фактических значений в точке данных x_i, выраженное в процентах. Она учитывает масштаб фактических значений для более объективной оценки ошибки.

2. **Сложные относительные ошибки** Если мы хотим найти любую из вышеперечисленных относительных ошибок мы делим ее на модуль среднего значение от реальных значений:

$$|\bar{y}| = |\frac{1}{n}\sum_{i=1}^{n} y_i|$$

Например:

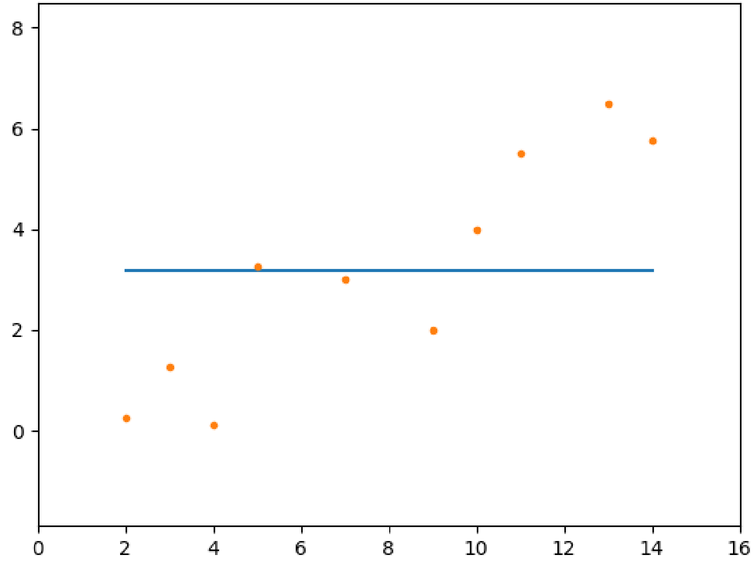
$$RealtiveMAE = \frac{|MAE|}{|\bar{y}|} = \frac{\frac{1}{n}\sum_{i=1}^{n}|\hat{y}_i - y_i|}{|\bar{y}|} = \frac{\frac{1}{n}\sum_{i=1}^{n}|\hat{y}_i - y_i|}{|\sum_{i=1}^{n} y_i|}$$

Примеры

In [80]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

def show_plot(x, y):
    plt.xlim(min(x) - 2, max(x) + 2)
    plt.ylim(min(y) - 2, max(y) + 2)
    plt.plot(x, y, '.')
    plt.show()
```

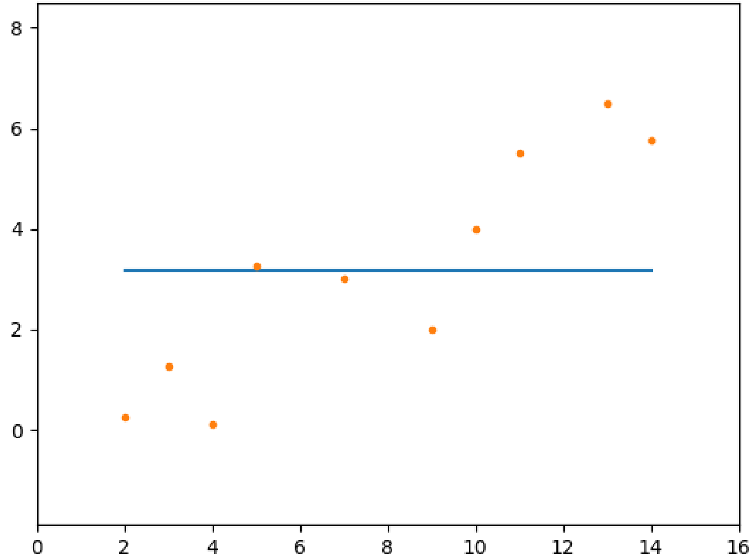
1. Апроксимирование прямой (линейное апроксимирование)

In [81]:
```
x = np.array([2, 3, 4, 5, 7, 9, 10, 11, 13, 14])
y = np.array([0.25, 1.25, 0.1, 3.25, 3, 2, 4, 5.5, 6.5, 5.75])

degree = 0
p = np.polyfit(x, y, deg=degree)
y_approx = np.polyval(p, x)
plt.plot(x, y_approx)
show_plot(x, y)
```



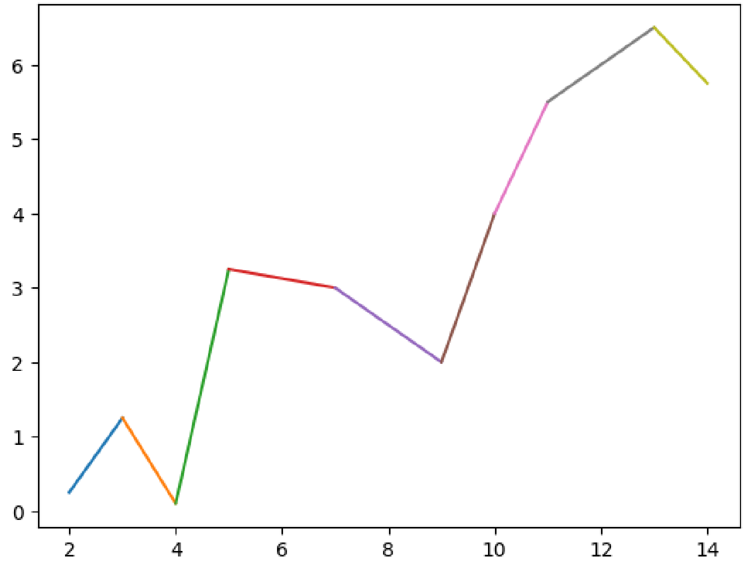
In [105…
```
## возвращает среднее всех аргументов y = p*mean(x)
## y = const + intercept
def mean_func(x, p):
    return np.mean(x) * np.ones_like(x) + p

x = np.array([2, 3, 4, 5, 7, 9, 10, 11, 13, 14])
y = np.array([0.25, 1.25, 0.1, 3.25, 3, 2, 4, 5.5, 6.5, 5.75])

p, covariance = curve_fit(mean_func, x, y)
y_approx = mean_func(x, p[0])
```

```
plt.plot(x, y_approx)
show_plot(x, y)

print(np.mean(x))
print(np.mean(y))
print(mean_func(x, p[0]))
```

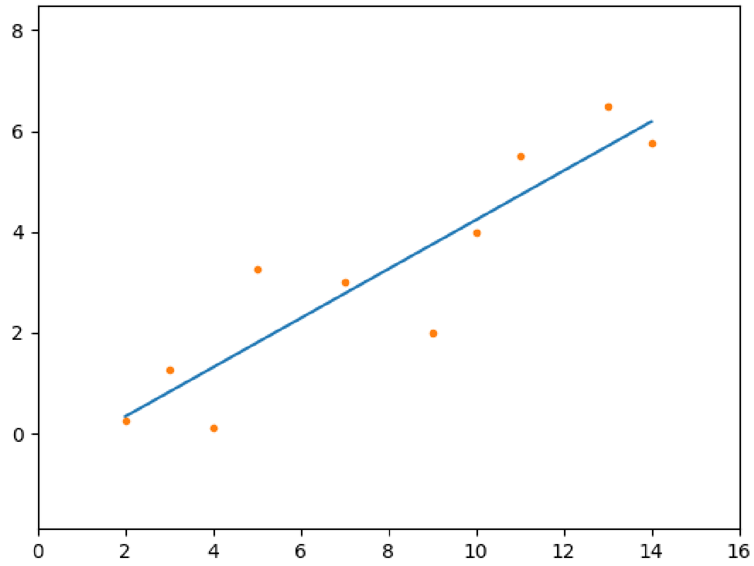


```
7.8
3.16
[3.16 3.16 3.16 3.16 3.16 3.16 3.16 3.16 3.16 3.16]
```

In [83]:
```
x = np.array([2, 3, 4, 5, 7, 9, 10, 11, 13, 14])
y = np.array([0.25, 1.25, 0.1, 3.25, 3, 2, 4, 5.5, 6.5, 5.75])

degree = 1
amount = 2
for i in range(0, len(x)-1, 1):
    p = np.polyfit(x[i:i+amount], y[i:i+amount], deg=degree)
    y_approx = np.polyval(p, x[i:i+amount])
    plt.plot(x[i:i+amount], y_approx)
```

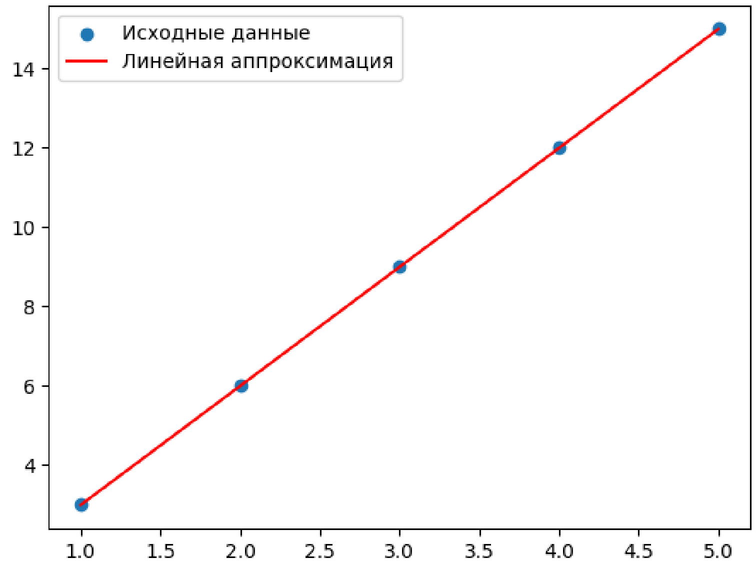


In [84]:
```
x = np.array([2, 3, 4, 5, 7, 9, 10, 11, 13, 14])
y = np.array([0.25, 1.25, 0.1, 3.25, 3, 2, 4, 5.5, 6.5, 5.75])

degree = 1
p = np.polyfit(x, y, deg=degree)
y_approx = np.polyval(p, x)
plt.plot(x, y_approx)
show_plot(x, y)
```



In [85]:

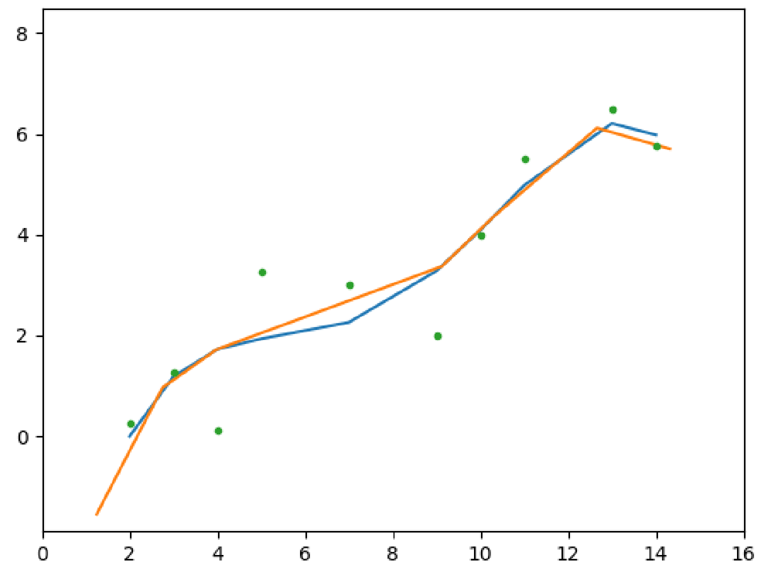
```python
# Исходные данные
x = np.array([1, 2, 3, 4, 5])
y = np.array([3,6,9,12,15])

# Аппроксимация линейной функцией
slope, intercept = np.polyfit(x, y, 1)

# Генерация новых данных для прямой
x_smooth = np.linspace(min(x), max(x), 100)
y_smooth = slope * x_smooth + intercept

# Визуализация
plt.scatter(x, y, label='Исходные данные')
plt.plot(x_smooth, y_smooth, label='Линейная аппроксимация', color='red')
plt.legend()
plt.show()
```



2. Апроксимирование полиномом

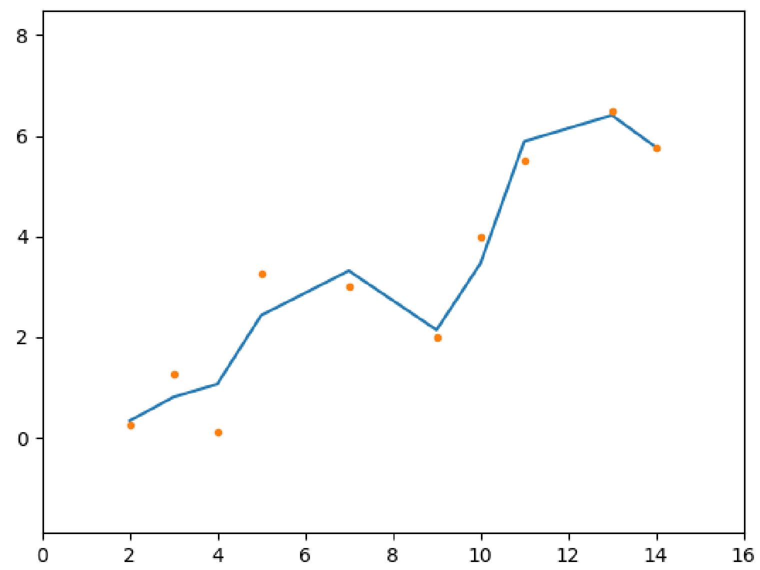
In [86]:

```python
x = np.array([2, 3, 4, 5, 7, 9, 10, 11, 13, 14])
y = np.array([0.25, 1.25, 0.1, 3.25, 3, 2, 4, 5.5, 6.5, 5.75])

degree = 4
p = np.polyfit(x, y, deg=degree)
y2_approx = np.polyval(p, x)
plt.plot(x, y2_approx)

new_x = np.sort(np.array([1.25, 2.75, 3.98, 10.31, 9.12,12.66, 14.32]))
y_approx = np.polyval(p, new_x)
plt.plot(new_x, y_approx)

show_plot(x, y)
```



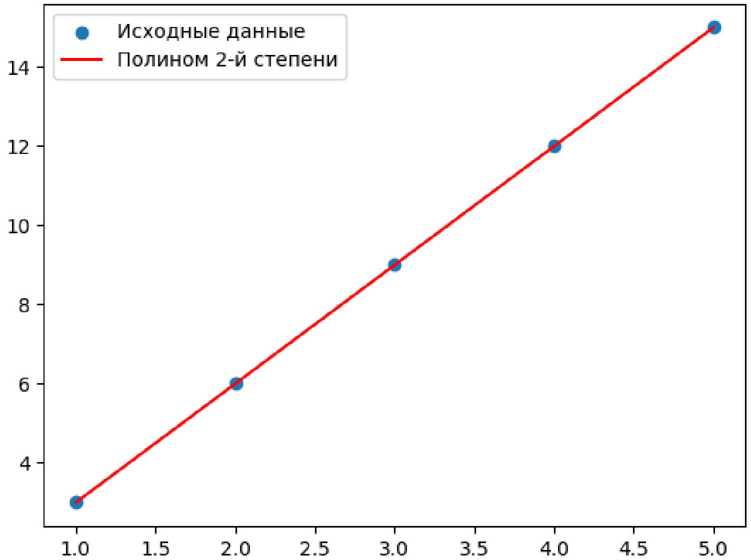
In [87]:

```python
x = np.array([2, 3, 4, 5, 7, 9, 10, 11, 13, 14])
y = np.array([0.25, 1.25, 0.1, 3.25, 3, 2, 4, 5.5, 6.5, 5.75])

degree = 7
p = np.polyfit(x, y, deg=degree)
y_approx = np.polyval(p, x)
plt.plot(x, y_approx)
show_plot(x, y)
```



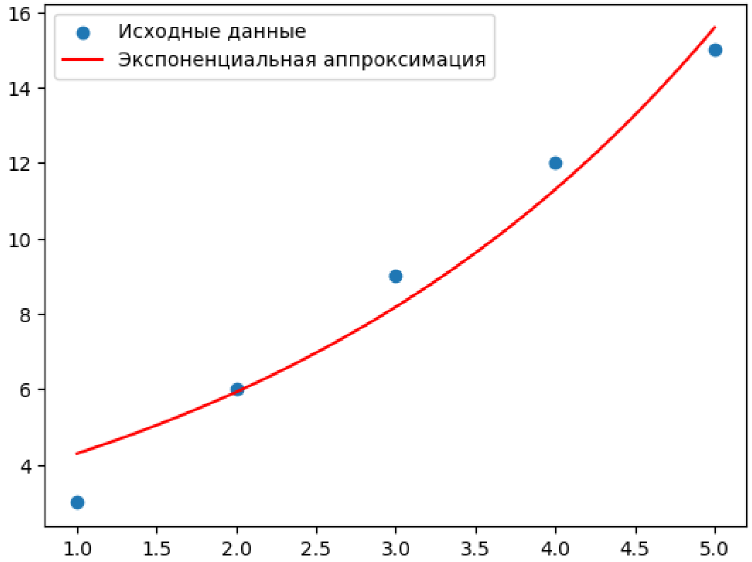
In [88]:
```python
# Исходные данные
x = np.array([1, 2, 3, 4, 5])
y = np.array([3,6,9,12,15])

# Аппроксимация полиномом
degree = 2
coefficients = np.polyfit(x, y, degree)
poly = np.poly1d(coefficients)

print(coefficients)

# Генерация новых данных для гладкой кривой
x_smooth = np.linspace(min(x), max(x), 100)
y_smooth = poly(x_smooth)

# Визуализация
plt.scatter(x, y, label='Исходные данные')
plt.plot(x_smooth, y_smooth, label=f'Полином {degree}-й степени', color='
plt.legend()
plt.show()
```

```
[1.04405949e-15 3.00000000e+00 1.27105749e-14]
```



3.Аппроксимация экспоненциальной кривой

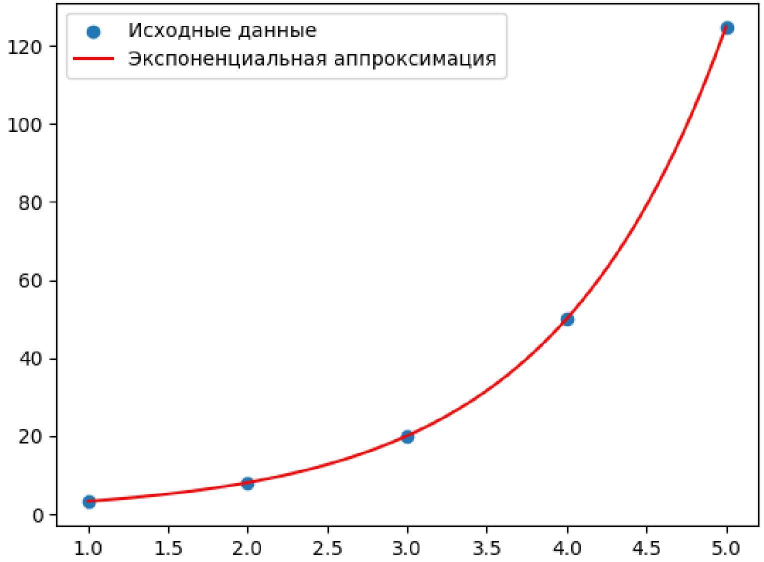
In [89]:
```python
# Исходные данные
x = np.array([1,2,3,4,5])
y = np.array([3,6,9,12,15])

# Определение экспоненциальной функции
def exponential_func(x, a, b):
    return a * np.exp(b * x)

# Аппроксимация с использованием curve_fit
params, covariance = curve_fit(exponential_func, x, y)

# Извлекаем параметры
a, b = params

# Генерация новых данных для гладкой кривой
x_smooth = np.linspace(min(x), max(x), 100)
y_smooth = exponential_func(x_smooth, a, b)

# Визуализация
plt.scatter(x, y, label='Исходные данные')
plt.plot(x_smooth, y_smooth, label='Экспоненциальная аппроксимация', colo
plt.legend()
plt.show()
```



4.Апроксимация логарифмической кривой

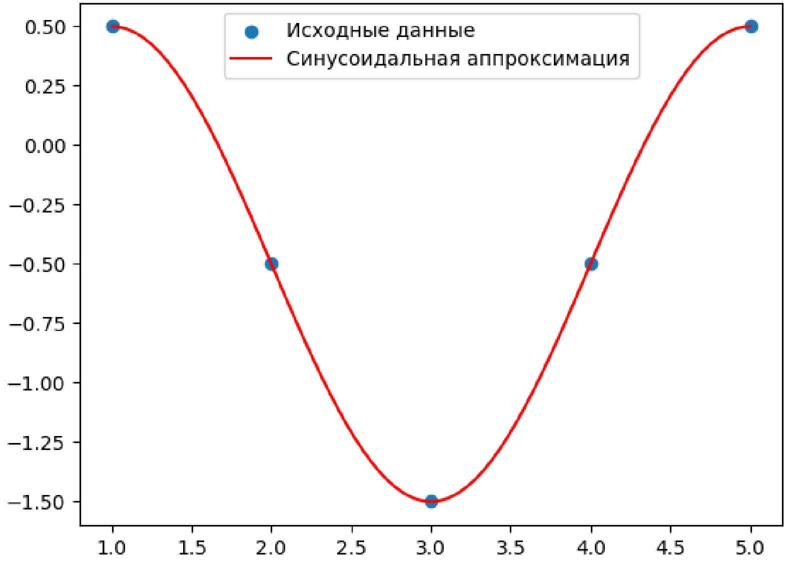
In [90]:
```python
# Исходные данные
x = np.array([1, 2, 3, 4, 5])
y = np.array([3, 8, 20, 50, 125])

# Определение экспоненциальной функции
def exponential_func(x, a, b):
    return a * np.exp(b * x)

# Аппроксимация с использованием curve_fit
params, covariance = curve_fit(exponential_func, x, y)

# Извлекаем параметры
a, b = params
```

```
# Генерация новых данных для гладкой кривой
x_smooth = np.linspace(min(x), max(x), 100)
y_smooth = exponential_func(x_smooth, a, b)

# Визуализация
plt.scatter(x, y, label='Исходные данные')
plt.plot(x_smooth, y_smooth, label='Экспоненциальная аппроксимация', colo
plt.legend()
plt.show()
```



5.Апроксимация синусоидальной кривой

In [91]:
```
# Исходные данные
x = np.array([1, 2, 3, 4, 5])
y = np.array([0.5, -0.5, -1.5, -0.5, 0.5])

# Определение синусоидальной функции
def sinusoidal_func(x, A, omega, phi, offset):
    return A * np.sin(omega * x + phi) + offset

# Аппроксимация с использованием curve_fit
params, covariance = curve_fit(sinusoidal_func, x, y)

# Извлекаем параметры
A, omega, phi, offset = params

# Генерация новых данных для гладкой кривой
x_smooth = np.linspace(min(x), max(x), 100)
y_smooth = sinusoidal_func(x_smooth, A, omega, phi, offset)

# Визуализация
plt.scatter(x, y, label='Исходные данные')
plt.plot(x_smooth, y_smooth, label='Синусоидальная аппроксимация', color=
plt.legend()
plt.show()
```



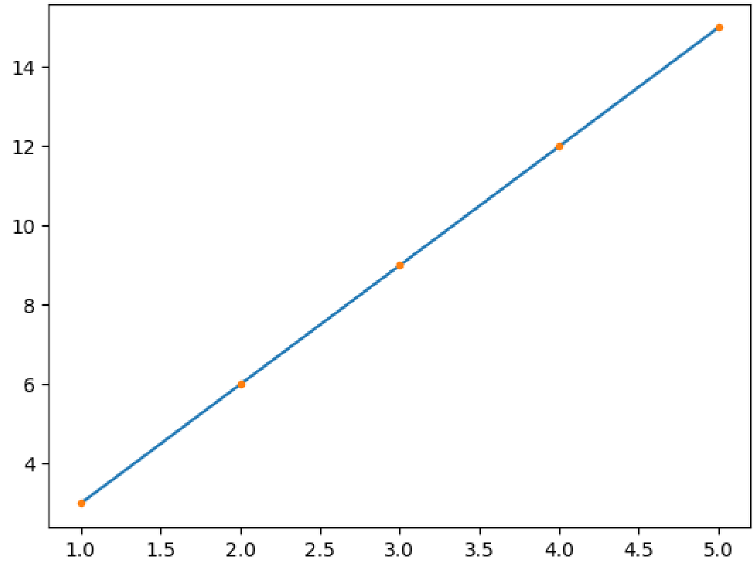
Пример 1.а

Проведен эксперимент для проверки закона Гука. Измерения позволили получить следующие данные:

- Сила (F): [1 2 3 4 5]
- Изменение длины (Δx): [3 6 9 12 15]

Составить график зависимости удлинения пружины от приложенной силы и определить модуль упругости Янга. На сколько пружина удлинится, если на нее действует сила в 7 Н?

In [106…
```
import numpy as np
import matplotlib.pyplot as plt

F = np.array([1, 2, 3, 4, 5])
delta_x = np.array([3, 6, 9, 12, 15])

p = np.polyfit(F, delta_x, 1)
y_aproks = np.polyval(p, F)
plt.plot(F, delta_x)
plt.plot(F, delta_x, '.')
plt.show()

sila = np.polyval(p, 7)
print(sila)
```




```
20.999999999999996
```

Пример 1.b

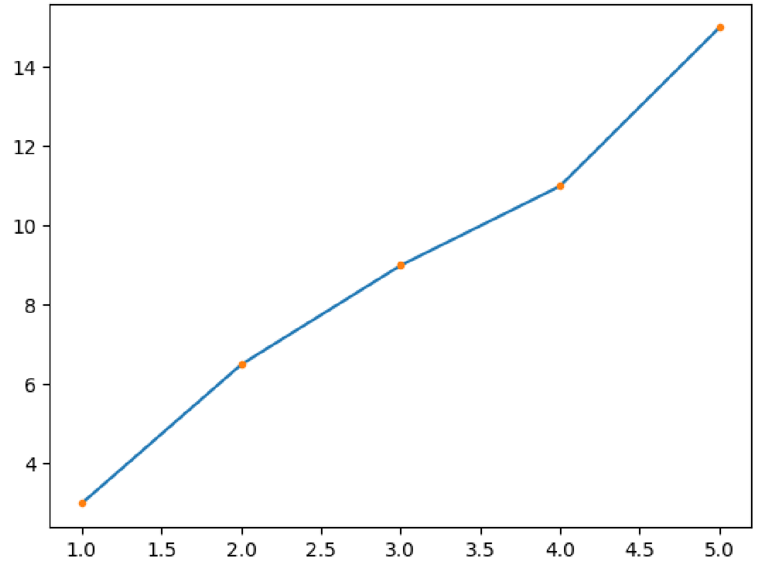
Вторая группа исследователей провела тот же эксперимент с той же пружиной, и получила следующие результаты:

- Сила (F): [1 2 3 4 5]
- Изменение длины (Δx): [3 6.5 9 11 15]

Какое значение модуля упругости Янга получила эта группа? На сколько пружина удлинится при применении силы в 7 Н?

In [107…

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

F = np.array([1,2,3,4,5])
delta_x = np.array([3,6.5,9,11,15])

p = np.polyfit(F, delta_x, 1)
y_aproks = np.polyval(p, F)
plt.plot(F,delta_x)
plt.plot(F,delta_x, '.')
plt.show()
```



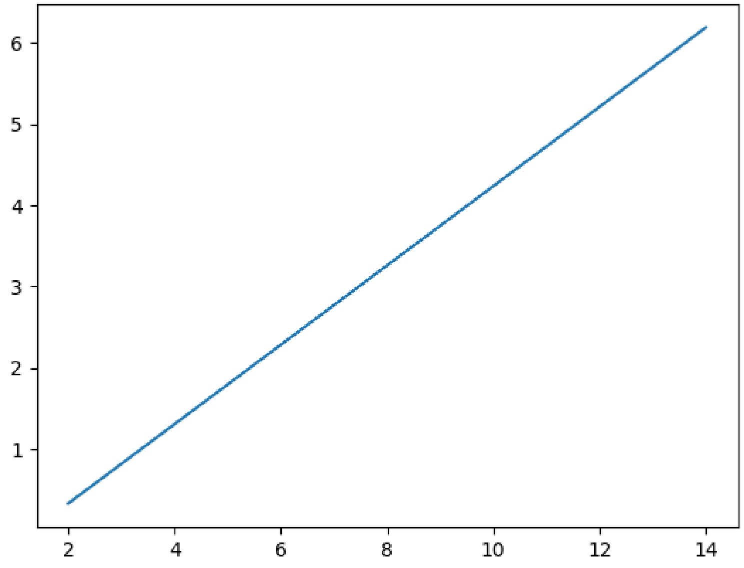
Задачи

ЗАДАЧА 1

Написать функцию, которая аппроксимирует заданный набор экспериментальных результатов полиномом первой степени. Входными параметрами функции являются векторы точек x, в которых производились измерения, и вектор, содержащий измеренные значения y. Выходным параметром функции является аппроксимационный полином P. Построить график экспериментальных данных и данных, полученных аппроксимацией.

In [108…

```
import numpy as np

def zadatak1(x,y):
    if len(x) != len(y):
        raise ValueError('Векторы не одинаковой длины')

    return np.polyfit(x,y,1)

x = np.array([2, 3, 4, 5, 7, 9, 10, 11, 13, 14])
y = np.array([0.25, 1.25, 0.1, 3.25, 3, 2, 4, 5.5, 6.5, 5.75])
p = zadatak1(x,y)

plt.plot(x, np.polyval(p, x))
plt.show()

print(p)
```



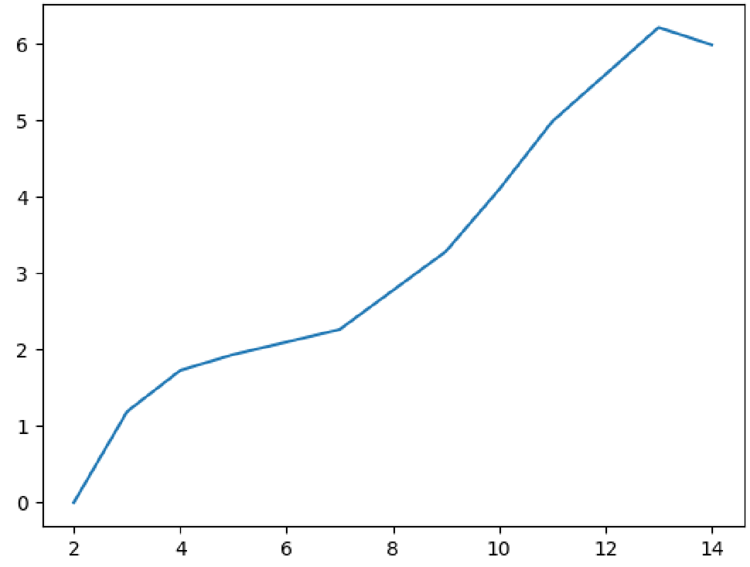
```
[ 0.48836634 -0.64925743]
```

ЗАДАЧА 2

Написать функцию, которая аппроксимирует заданный набор экспериментальных результатов полиномами третьей и четвертой степени. Входными параметрами функции являются векторы точек x, в которых производились измерения, и вектор, содержащий измеренные значения y. Выходным параметром функции является аппроксимационный полином с меньшей средней абсолютной ошибкой. В функции нужно проверить, имеют ли x и y одинаковую длину.

In [130…

```
import numpy as np
import statistics as stat

def zadatak2(x,y):
    p3 = np.polyfit(x,y,3)
    p4 = np.polyfit(x,y,4)

    yp3 = np.polyval(p3,x)
    yp4 = np.polyval(p4,x)

    aps_er3 = stat.mean(abs(y-yp3))
    aps_er4 = stat.mean(abs(y-yp4))

    return p3 if aps_er3 < aps_er4 else p4

x = np.array([2, 3, 4, 5, 7, 9, 10, 11, 13, 14])
y = np.array([0.25, 1.25, 0.1, 3.25, 3, 2, 4, 5.5, 6.5, 5.75])
p = zadatak2(x,y)

plt.plot(x, np.polyval(p, x))
plt.show()

print(p)
```



```
[-2.94860217e-03  9.40084211e-02 -1.00859227e+00  4.63847733e+00
 -5.95952012e+00]
```

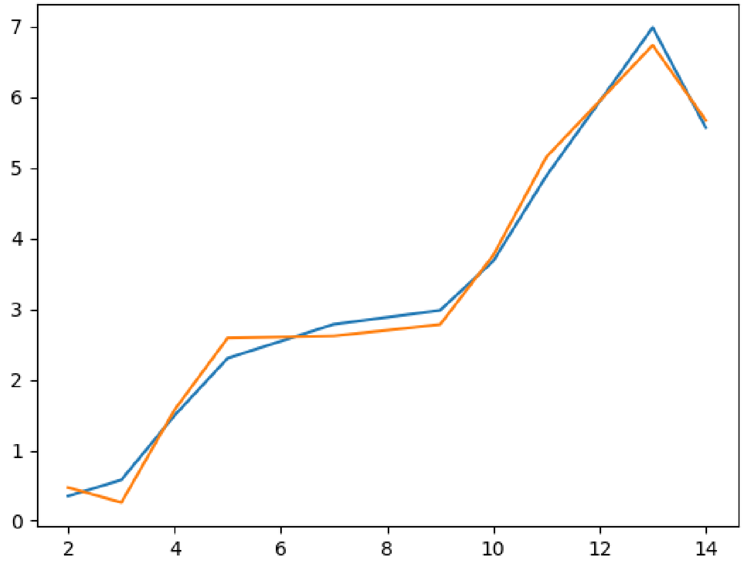
ЗАДАЧА 3

Написать функцию, которая аппроксимирует заданный набор экспериментальных результатов полиномами пятой и шестой степени. Входными параметрами функции являются векторы точек x, в которых производились измерения, и вектор, содержащий измеренные значения y. Выходными параметрами функции являются аппроксимационные полиномы P5 и P6. В функции нужно проверить, имеют ли x и y одинаковую длину.

In [112…

```
import numpy as np

def zadatak3(x,y):
    if len(x) != len(y):
        raise ValueError('Векторы не одинаковой длины')
    p5 = np.polyfit(x,y,5)
    p6 = np.polyfit(x,y,6)
    return p5, p6

x = np.array([2, 3, 4, 5, 7, 9, 10, 11, 13, 14])
y = np.array([0.25, 1.25, 0.1, 3.25, 3, 2, 4, 5.5, 6.5, 5.75])
[p5, p6] = zadatak3(x,y)

plt.plot(x, np.polyval(p5, x))
plt.plot(x, np.polyval(p6, x))
plt.show()
```

```
print(p5,p6, sep='\n')
```



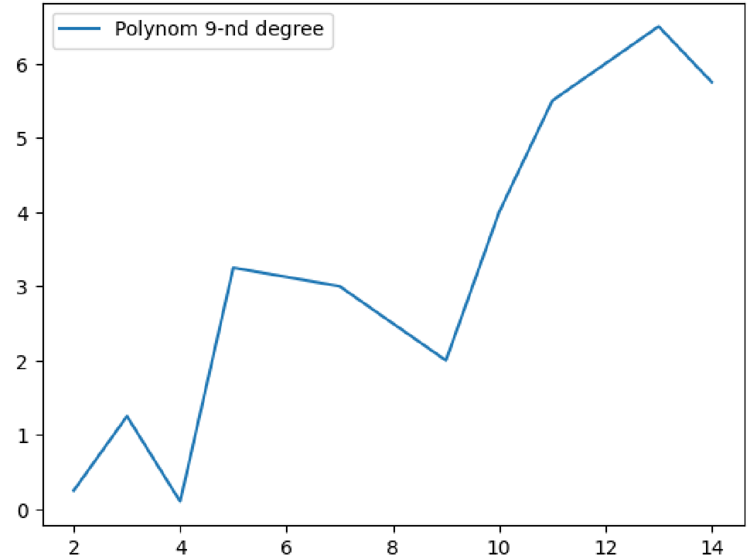
```
[-1.17854679e-03  4.39297255e-02 -5.98293341e-01  3.64521200e+00
 -9.23555727e+00  8.36339817e+00]
[ 1.86085443e-04 -1.01618386e-02  2.14241528e-01 -2.20211654e+00
  1.14429999e+01 -2.74928932e+01  2.41893237e+01]
```

ЗАДАЧА 4

Написать функцию, которая аппроксимирует заданный набор экспериментальных результатов полиномом произвольной степени так, чтобы максимальная относительная ошибка аппроксимации не превышала 1%. Входными параметрами функции являются векторы точек x, в которых производились измерения, и вектор, содержащий измеренные значения y. В функции нужно проверить, имеют ли векторы x и y одинаковую длину. Выходными параметрами функции являются коэффициенты и степень полинома.

In [129…

```
import numpy as np
import math

def solve(x, y):
    if (len(x) != len(y)):
        raise RuntimeError('len(x) != len(y)')

    p = []
    degree = 0
    err = math.inf

    while err >= 0.01 :
        degree += 1
        p = np.polyfit(x, y, degree)
        y_approx = np.polyval(p, x)
        err = np.max(abs( (y_approx - y) / y * 100 ))

    return p, degree

x = np.array([2, 3, 4, 5, 7, 9, 10, 11, 13, 14])
y = np.array([0.25, 1.25, 0.1, 3.25, 3, 2, 4, 5.5, 6.5, 5.75])
p, degree = solve(x, y)

plt.plot(x, np.polyval(p, x), label=f'Polynom {degree}-nd degree'.format(
plt.legend()
plt.show()
print(p)
```




```
[ 6.73400673e-06 -5.67898729e-04  2.04549663e-02 -4.11258072e-01
  5.05877844e+00 -3.92181131e+01  1.90148680e+02 -5.51496489e+02
  8.62253985e+02 -5.50008333e+02]
```