Исследование зависимости скорости сходимости от различных параметров

Теория, постановка вопроса

Что вообще такое скорость сходимости? Во-первых, речь идёт о градиентном спуске, так что сходимости, лучше линейной говорить не приходится, но у нас всё ещё есть некий показатель в экспоненте, который может быть разным. Теорема 3.4 утверждает, коэффициент этот — $\frac{\kappa-1}{\kappa+1}$, где κ — число обусловленности второго дифференциала.

Если говорить о кумулятивной быстроте сходимости абстрактного алгоритма с практической точки зрения, имеет смысл смотреть на количество итераций до достижения фиксированного значения функции ошибки (ε). Далее будет показано, что, как следует из теоремы, это количество итераций линейно зависит от числа обусловленности.

Как измеряем?

Проверим это. Понятно, что интересные эффекты есть только от второго дифференциала. Кроме того, для чистоты эксперимента будем испытывать алгоритм на эталонной системе функций, имеющих вблизи функции приспособленности второй дифференциал с фиксированным числом обусловленности, генерацию которых мы умеем контролировать, подгоняя под нужные параметры — квадратичные формы (см. функцию приспособленности).

Независимость от выбора базиса

Покажем, что скорость сходимости не зависит от выбора базиса в генераторе: на распределении виден один чёткий пик, а отношение $\frac{\sigma}{\mu}$ весьма и весьма мало́ (≈ 2 в данном случае — это в рамках погрешности, случайных флюктуаций).

Поэтому можно дальше исследовать поведение в случайном базисе «как одно целое» (но раз процесс ± случайный, не забываем усреднять, благо мы теперь имеем на это право).

Характер зависимости от числа обусловленности и размерности задачи

Замерим по s раз количество итераций для достижения ошибки ε для каждого значения из сетки $(n, k) \in \left[2..10^3\right] \times \left[1..10^3\right]$ и усредним результат по s для каждой

пары.

Исследуем эту зависимость:

- Если для каждого рассматриваемого n построить сечение и посмотреть на зависимость $\text{iterations}_n(k)$, то для каждого n получится зависимость, очень близкая к линейной, почти не меняющая своих параметров при изменении n. Делаем гипотезу, что $T(n,k) \approx an + bk + c$, где c не велико, а $a \approx 0$
- Чтобы не быть голословными, построим линейную регрессию на точках, в которых считали значения $T(n,k)$. Как видно из результатов ниже, именно такими коэффициенты и являются, в то время как оценка качества аппроксимации R^2 весьма и весьма велика, так что линейная модель действительно соответствует реальности.

Давайте теперь это докажем:

Теоретическое обоснование линейности

Из построения эксперимента,

$$\Delta_0 = \|f(x_0)\| \approx 1 \tag{1}$$
$$\Delta_n = \|f(x_n)\| \approx \varepsilon \tag{2}$$

Из теоремы 3.4,

$$\Delta_n = r^2\Delta_{n-1} = \ldots = (r^2)^n\Delta_0$$
$$\Rightarrow n = \log_{r^2}\frac{1}{\varepsilon} = \frac{\log\varepsilon}{2}\frac{1}{-\log(r)} = \left[r = \frac{\lambda_n-\lambda_1}{\lambda_n+\lambda_1} = \frac{\kappa-1}{\kappa+1}\right] = \frac{\log\varepsilon}{2}\frac{1}{\log(\kappa+1)-\log(\kappa}$$
$$\cdot\,\kappa$$

Ссылка на г-на Вольфрама Тейлора

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt

plt.rcParams["figure.figsize"] = (10, 10)
%load_ext autoreload
%autoreload 2

from core.gradient_descent import *
from core.visualizer import *
from core.optimizer_evaluator import *
```

Экспериментально показываем, что количество итераций до сходимости не зависит от выбора базиса

In [3]:
```
generator = lambda: generate_positive_definite_quadratic_form(10, 10, random_orthon
```

```
computations = [iteration_count_computer(generator, gradient_descent, fibonacci_sea
computations
```

Out[3]: [46,
52,
47,
51,
51,
50,
46,
51,
50,
52,
52,
51,
52,
50,
50,
50,
52,
51,
51,
50,
50,
47,
44,
46,
50,
49,
38,
48,
52,
52,
46,
42,
51,
51,
51,
51,
51,
50,
50,
50,
50,
44,
48,
44,
48,
50,
48,
50,
52,
51,
50,
50,
50,
52,
44,
52,

```
 50,
 49,
 50,
 52,
 52,
 50,
 50,
 44,
 48,
 50,
 53,
 46,
 52,
 51,
 48,
 50,
 44,
 51,
 51,
 51,
 48,
 51,
 52,
 51,
 52,
 50,
 46,
 50,
 48,
 52,
 51,
 52,
 50,
 52,
 50,
 53,
 51,
 50,
 51,
 49,
 51,
 52,
 52,
 50]
```

In [4]:
```
mu = np.mean(computations)
mu
```

Out[4]: 49.63

In [5]:
```
sigma = np.std(computations)
sigma
```

Out[5]: 2.6063576116872373

In [ ]:
```

```

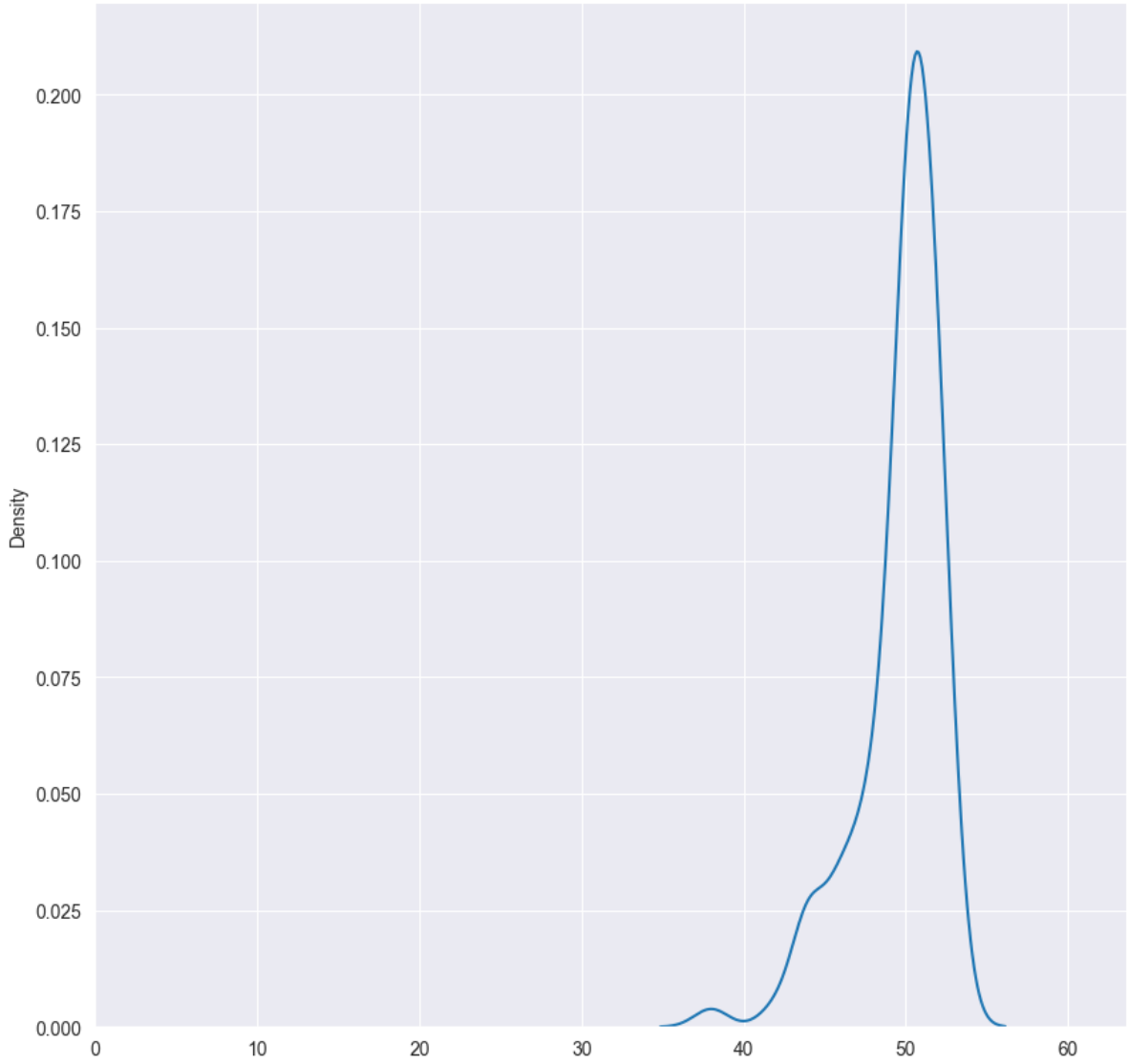
In [6]:
```
sigma / mu
```

Out[6]:
```
0.05251576892378072
```

In [7]:
```
import seaborn as sns

fig, ax = plt.subplots()
sns.kdeplot(computations, ax=ax)
ax.set_xlim(0, max(computations) * 1.2)
```

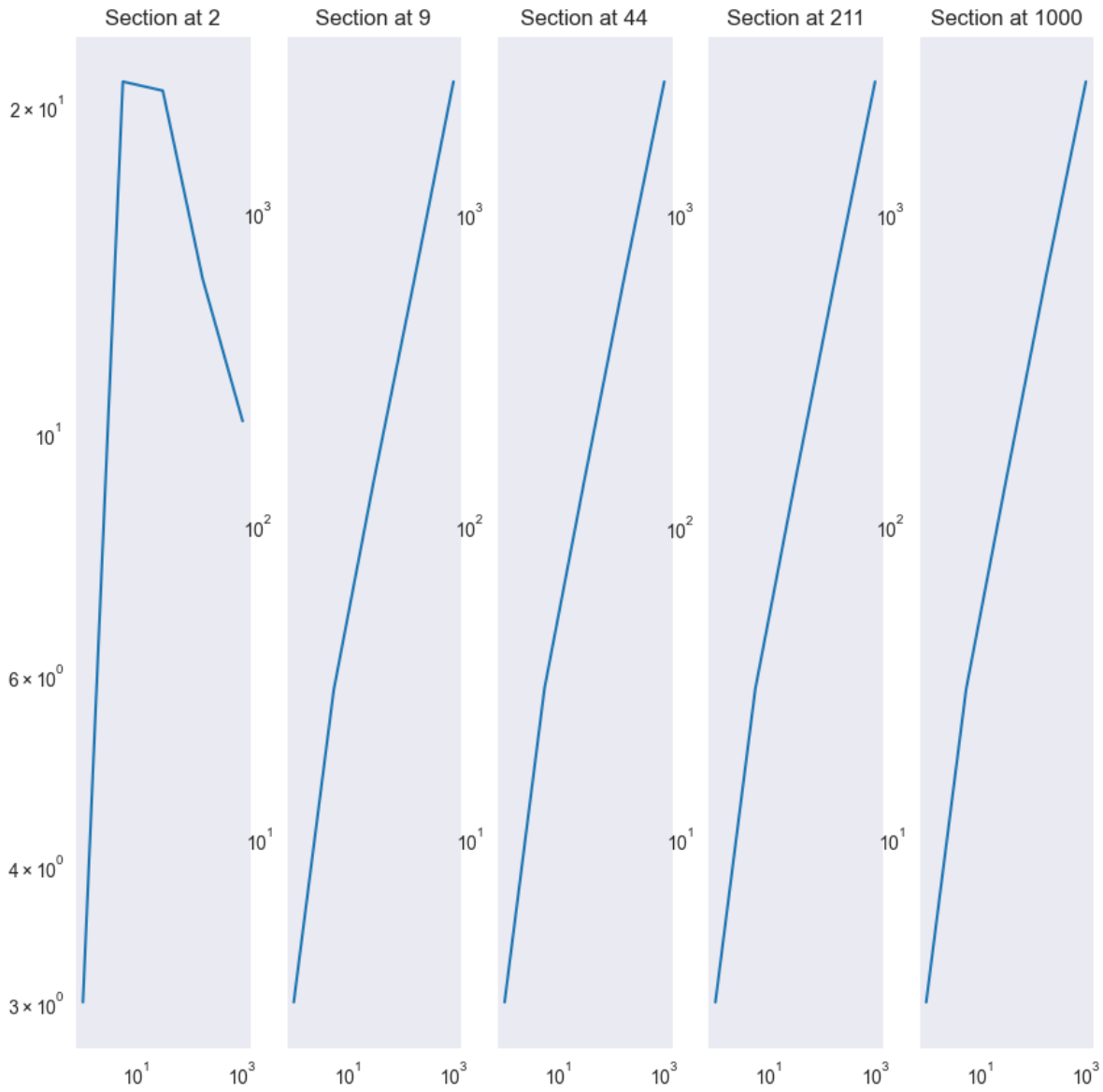
Out[7]:
```
(0.0, 63.599999999999994)
```



Изучаем зависимость количества итераций до *сходимости* от размерности и числа обусловленности задачи (спойлер: она, как и в теории, линейна)

In [31]:
```
plot_section_graphs(
    lambda n, k: average_iterations_until_convergence(
```

```
    lambda: generate_positive_definite_quadratic_form(n, k, random_orthonormal_basi
    gradient_descent, 15, fibonacci_search(40)
    ),
    logspace_range(2, 1000, 5, dtype=int),
    logspace_range(1, 1000, 5))
```



Заметим, однако, что объём вычислений растёт не линейно и даже зависит от размерности задачи: число итераций внутреннего поиска тоже требуется бОльшее с ростом числа обусловленности, а работа с векторами $n \times 1$ (и тем более — с матрицами $n \times n$) тоже занимает растущее время. Но измеряем мы конкретный показатель — число итераций до сходимости.

Например, поиск Фибоначчи с 10-ю итерациями фейлится где-то начиная с $k \geqslant 80$ в то время, как для 30-и итераций всё работает нормально даже при $k = 1$

```
RuntimeWarning: overflow encountered in matmul
```

In [55]:
```
# iteration_count_computer(lambda: generate_positive_definite_quadratic_form(1000,
```

```
C:\dev\Education\MethOpt\GradientDescent\core\optimizer_evaluator.py:21: RuntimeWa
rning: overflow encountered in matmul
  return (x @ self.matrix @ x[:, newaxis])[0]
C:\dev\Education\MethOpt\GradientDescent\core\gradient_descent.py:22: RuntimeWarni
ng: invalid value encountered in multiply
  next_point = last_point - g * linear_search(lambda l: target_function(last_point
- g * l),
C:\dev\Education\MethOpt\GradientDescent\core\optimizer_evaluator.py:21: RuntimeWa
rning: invalid value encountered in matmul
  return (x @ self.matrix @ x[:, newaxis])[0]
C:\dev\Education\MethOpt\GradientDescent\core\optimizer_evaluator.py:24: RuntimeWa
rning: invalid value encountered in matmul
  return lambda x: (2 * self.matrix @ x[:, newaxis])[:, 0]
C:\dev\Education\MethOpt\GradientDescent\core\gradient_descent.py:22: RuntimeWarni
ng: invalid value encountered in subtract
  next_point = last_point - g * linear_search(lambda l: target_function(last_point
- g * l),
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[55], line 1
----> 1 iteration_count_computer(lambda: generate_positive_definite_quadratic_form
(1000, 80, canonical_basis), gradient_descent, fibonacci_search(10))()

File C:\dev\Education\MethOpt\GradientDescent\core\optimizer_evaluator.py:83, in i
teration_count_computer.<locals>.computation()
     81 def computation():
     82     f = form_generator()
---> 83     return len(optimizer(f, f.gradient_function(), random_normalized_vecto
r(f.n), line_searcher, precision_termination_condition))

File C:\dev\Education\MethOpt\GradientDescent\core\gradient_descent.py:22, in grad
ient_descent(target_function, gradient_function, x0, linear_search, terminate_cond
ition)
     20     if np.linalg.norm(g) == 0:
     21         return points
---> 22     next_point = last_point - g * linear_search(lambda l: target_function(
last_point - g * l),
     23                                                 lambda l: -np.dot(g, gradi
ent_function(last_point - g * l)))
     24     points.append(next_point)
     25 return points

File C:\dev\Education\MethOpt\GradientDescent\core\gradient_descent.py:89, in fibo
nacci_search.<locals>.search(f, _derivative)
     87         x2 = x1
     88         x1 = l + (r - l) * fibs[-k - 4] / fibs[-k - 2]
---> 89         y1, y2 = f(x1), y1
     91 return r

File C:\dev\Education\MethOpt\GradientDescent\core\gradient_descent.py:22, in grad
ient_descent.<locals>.<lambda>(l)
     20     if np.linalg.norm(g) == 0:
     21         return points
---> 22     next_point = last_point - g * linear_search(lambda l: target_function(
last_point - g * l),
     23                                                 lambda l: -np.dot(g, gradi
ent_function(last_point - g * l)))
     24     points.append(next_point)
     25 return points

KeyboardInterrupt:
```

Демонстрируем, что бинпоиск с остановкой по затуханию производной и поиск Фибоначчи с 30-ю итерациями отлично справляются даже для $k = 1000$

In [54]:
```
iteration_count_computer(
    lambda: generate_positive_definite_quadratic_form(1000, 500, canonical_basis),
    gradient_descent,
    fibonacci_search(30)
)()
```

Out[54]: 602

In [56]:
```
iteration_count_computer(
    lambda: generate_positive_definite_quadratic_form(1000, 1000, canonical_basis),
    gradient_descent,
    bin_search
)()
```

Out[56]:
```
1309
```

In [43]:
```
n_values = logspace_range(10, 100, 10, dtype=int)
k_values = logspace_range(1, 150, 30)

values = [[(n, k) for k in k_values] for n in n_values]

evaluation = [[average_iterations_until_convergence(
    lambda: generate_positive_definite_quadratic_form(n, k, random_orthonormal_basi
    gradient_descent,
    15,
) for k in k_values] for n in n_values]
```

In [44]:
```
np.array(evaluation)
```

Out[44]:
```
array([[  3.        ,   6.8       ,   8.53333333,  10.        ,
         12.06666667,  14.26666667,  16.73333333,  19.13333333,
         22.26666667,  25.86666667,  29.06666667,  35.        ,
         40.6       ,  46.8       ,  55.33333333,  64.33333333,
         75.        ,  87.46666667, 101.53333333, 119.06666667,
        138.46666667, 158.46666667, 189.66666667, 217.26666667,
        248.4       , 289.53333333, 339.93333333, 394.86666667,
        453.73333333, 537.33333333],
       [  3.        ,   7.        ,   8.46666667,  10.06666667,
         12.13333333,  14.06666667,  16.66666667,  19.2       ,
         22.46666667,  26.06666667,  30.4       ,  34.26666667,
         39.46666667,  47.13333333,  55.13333333,  65.4       ,
         75.73333333,  88.93333333, 102.        , 117.93333333,
        139.        , 160.73333333, 189.13333333, 219.6       ,
        245.4       , 288.86666667, 338.53333333, 393.33333333,
        460.33333333, 529.26666667],
       [  3.        ,   7.        ,   8.6       ,  10.06666667,
         12.06666667,  14.33333333,  16.46666667,  19.66666667,
         22.13333333,  26.06666667,  29.6       ,  34.2       ,
         40.8       ,  47.6       ,  56.53333333,  65.13333333,
         75.93333333,  87.73333333, 102.6       , 119.2       ,
        139.13333333, 162.06666667, 187.6       , 220.66666667,
        250.8       , 297.73333333, 342.8       , 392.06666667,
        458.4       , 531.2       ],
       [  3.        ,   6.93333333,   8.4       ,  10.06666667,
         11.93333333,  14.2       ,  16.6       ,  19.4       ,
         22.2       ,  26.33333333,  30.2       ,  34.86666667,
         40.8       ,  47.53333333,  56.13333333,  65.46666667,
         75.66666667,  86.93333333, 103.66666667, 120.4       ,
        139.86666667, 162.73333333, 188.26666667, 220.4       ,
        255.33333333, 294.        , 346.66666667, 401.        ,
        462.53333333, 535.53333333],
       [  3.        ,   7.        ,   8.53333333,  10.13333333,
         11.93333333,  14.26666667,  16.86666667,  19.26666667,
         22.4       ,  25.93333333,  29.4       ,  34.93333333,
         40.73333333,  48.33333333,  55.2       ,  65.        ,
         75.4       ,  88.46666667, 102.6       , 120.6       ,
        138.6       , 161.33333333, 189.46666667, 220.33333333,
        253.8       , 293.4       , 345.6       , 399.53333333,
        461.6       , 541.13333333],
       [  3.        ,   7.        ,   8.4       ,  10.06666667,
         12.        ,  14.13333333,  16.66666667,  19.46666667,
         22.53333333,  25.73333333,  30.06666667,  35.2       ,
         41.6       ,  47.8       ,  56.06666667,  65.13333333,
         76.33333333,  88.86666667, 103.4       , 120.8       ,
        139.66666667, 161.66666667, 188.8       , 219.46666667,
        254.66666667, 299.93333333, 346.        , 399.66666667,
        464.93333333, 542.4       ],
       [  3.        ,   7.        ,   8.66666667,  10.13333333,
         12.06666667,  14.13333333,  16.73333333,  19.46666667,
         22.6       ,  26.26666667,  30.2       ,  35.6       ,
         41.13333333,  48.33333333,  56.46666667,  65.33333333,
         76.4       ,  89.06666667, 103.73333333, 118.66666667,
        140.06666667, 161.33333333, 188.4       , 219.86666667,
        256.        , 298.        , 344.93333333, 398.86666667,
        466.2       , 536.73333333],
```

```
[  3.        ,   7.        ,   8.53333333,  10.06666667,
  12.06666667,  14.13333333,  16.66666667,  19.66666667,
  22.33333333,  25.86666667,  29.93333333,  35.2       ,
  41.2       ,  47.66666667,  56.06666667,  65.4       ,
  75.73333333,  89.26666667, 103.66666667, 120.46666667,
 140.26666667, 163.53333333, 190.33333333, 219.8       ,
 253.66666667, 298.73333333, 345.93333333, 401.6       ,
 466.93333333, 538.8       ],
[  3.        ,   7.        ,   8.53333333,  10.        ,
  12.        ,  14.13333333,  16.66666667,  19.66666667,
  22.53333333,  25.6       ,  30.13333333,  35.46666667,
  41.        ,  48.06666667,  55.93333333,  65.53333333,
  76.4       ,  88.53333333, 103.2       , 120.26666667,
 140.8       , 163.33333333, 189.53333333, 220.26666667,
 256.86666667, 296.8       , 346.2       , 399.33333333,
 466.06666667, 540.93333333],
[  3.        ,   7.        ,   8.53333333,  10.13333333,
  12.        ,  14.06666667,  16.8       ,  19.33333333,
  22.4       ,  25.8       ,  30.4       ,  35.73333333,
  40.8       ,  48.13333333,  56.06666667,  65.4       ,
  76.33333333,  89.46666667, 103.93333333, 120.8       ,
 140.2       , 163.2       , 190.46666667, 219.8       ,
 256.2       , 294.33333333, 345.6       , 400.26666667,
 467.8       , 537.26666667]])
```

In [45]:
```
from sklearn.linear_model import LinearRegression

x = np.array(values).reshape(-1, 2)
y = np.array(evaluation).reshape(-1, 1)[:, 0]
```

In [46]:
```
x
```

Out[46]:
```
array([[ 10.        ,   1.        ],
       [ 10.        ,   1.18860521],
       [ 10.        ,   1.41278235],
       [ 10.        ,   1.67924046],
       [ 10.        ,   1.99595396],
       [ 10.        ,   2.37240127],
       [ 10.        ,   2.81984851],
       [ 10.        ,   3.35168663],
       [ 10.        ,   3.98383219],
       [ 10.        ,   4.7352037 ],
       [ 10.        ,   5.62828779],
       [ 10.        ,   6.68981219],
       [ 10.        ,   7.95154563],
       [ 10.        ,   9.45124856],
       [ 10.        ,  11.23380329],
       [ 10.        ,  13.35255712],
       [ 10.        ,  15.87091896],
       [ 10.        ,  18.86425696],
       [ 10.        ,  22.42215411],
       [ 10.        ,  26.6510892 ],
       [ 10.        ,  31.67762348],
       [ 10.        ,  37.65218831],
       [ 10.        ,  44.7535872 ],
       [ 10.        ,  53.19434692],
       [ 10.        ,  63.2270779 ],
       [ 10.        ,  75.15203422],
       [ 10.        ,  89.32609942],
       [ 10.        , 106.17346718],
       [ 10.        , 126.19833627],
       [ 10.        , 150.        ],
       [ 12.        ,   1.        ],
       [ 12.        ,   1.18860521],
       [ 12.        ,   1.41278235],
       [ 12.        ,   1.67924046],
       [ 12.        ,   1.99595396],
       [ 12.        ,   2.37240127],
       [ 12.        ,   2.81984851],
       [ 12.        ,   3.35168663],
       [ 12.        ,   3.98383219],
       [ 12.        ,   4.7352037 ],
       [ 12.        ,   5.62828779],
       [ 12.        ,   6.68981219],
       [ 12.        ,   7.95154563],
       [ 12.        ,   9.45124856],
       [ 12.        ,  11.23380329],
       [ 12.        ,  13.35255712],
       [ 12.        ,  15.87091896],
       [ 12.        ,  18.86425696],
       [ 12.        ,  22.42215411],
       [ 12.        ,  26.6510892 ],
       [ 12.        ,  31.67762348],
       [ 12.        ,  37.65218831],
       [ 12.        ,  44.7535872 ],
       [ 12.        ,  53.19434692],
       [ 12.        ,  63.2270779 ],
       [ 12.        ,  75.15203422],
```

```
[ 12.        ,  89.32609942],
[ 12.        , 106.17346718],
[ 12.        , 126.19833627],
[ 12.        , 150.        ],
[ 16.        ,   1.        ],
[ 16.        ,   1.18860521],
[ 16.        ,   1.41278235],
[ 16.        ,   1.67924046],
[ 16.        ,   1.99595396],
[ 16.        ,   2.37240127],
[ 16.        ,   2.81984851],
[ 16.        ,   3.35168663],
[ 16.        ,   3.98383219],
[ 16.        ,   4.7352037 ],
[ 16.        ,   5.62828779],
[ 16.        ,   6.68981219],
[ 16.        ,   7.95154563],
[ 16.        ,   9.45124856],
[ 16.        ,  11.23380329],
[ 16.        ,  13.35255712],
[ 16.        ,  15.87091896],
[ 16.        ,  18.86425696],
[ 16.        ,  22.42215411],
[ 16.        ,  26.6510892 ],
[ 16.        ,  31.67762348],
[ 16.        ,  37.65218831],
[ 16.        ,  44.7535872 ],
[ 16.        ,  53.19434692],
[ 16.        ,  63.2270779 ],
[ 16.        ,  75.15203422],
[ 16.        ,  89.32609942],
[ 16.        , 106.17346718],
[ 16.        , 126.19833627],
[ 16.        , 150.        ],
[ 21.        ,   1.        ],
[ 21.        ,   1.18860521],
[ 21.        ,   1.41278235],
[ 21.        ,   1.67924046],
[ 21.        ,   1.99595396],
[ 21.        ,   2.37240127],
[ 21.        ,   2.81984851],
[ 21.        ,   3.35168663],
[ 21.        ,   3.98383219],
[ 21.        ,   4.7352037 ],
[ 21.        ,   5.62828779],
[ 21.        ,   6.68981219],
[ 21.        ,   7.95154563],
[ 21.        ,   9.45124856],
[ 21.        ,  11.23380329],
[ 21.        ,  13.35255712],
[ 21.        ,  15.87091896],
[ 21.        ,  18.86425696],
[ 21.        ,  22.42215411],
[ 21.        ,  26.6510892 ],
[ 21.        ,  31.67762348],
[ 21.        ,  37.65218831],
```

```
[ 21.        ,  44.7535872 ],
[ 21.        ,  53.19434692],
[ 21.        ,  63.2270779 ],
[ 21.        ,  75.15203422],
[ 21.        ,  89.32609942],
[ 21.        , 106.17346718],
[ 21.        , 126.19833627],
[ 21.        , 150.        ],
[ 27.        ,   1.        ],
[ 27.        ,   1.18860521],
[ 27.        ,   1.41278235],
[ 27.        ,   1.67924046],
[ 27.        ,   1.99595396],
[ 27.        ,   2.37240127],
[ 27.        ,   2.81984851],
[ 27.        ,   3.35168663],
[ 27.        ,   3.98383219],
[ 27.        ,   4.7352037 ],
[ 27.        ,   5.62828779],
[ 27.        ,   6.68981219],
[ 27.        ,   7.95154563],
[ 27.        ,   9.45124856],
[ 27.        ,  11.23380329],
[ 27.        ,  13.35255712],
[ 27.        ,  15.87091896],
[ 27.        ,  18.86425696],
[ 27.        ,  22.42215411],
[ 27.        ,  26.6510892 ],
[ 27.        ,  31.67762348],
[ 27.        ,  37.65218831],
[ 27.        ,  44.7535872 ],
[ 27.        ,  53.19434692],
[ 27.        ,  63.2270779 ],
[ 27.        ,  75.15203422],
[ 27.        ,  89.32609942],
[ 27.        , 106.17346718],
[ 27.        , 126.19833627],
[ 27.        , 150.        ],
[ 35.        ,   1.        ],
[ 35.        ,   1.18860521],
[ 35.        ,   1.41278235],
[ 35.        ,   1.67924046],
[ 35.        ,   1.99595396],
[ 35.        ,   2.37240127],
[ 35.        ,   2.81984851],
[ 35.        ,   3.35168663],
[ 35.        ,   3.98383219],
[ 35.        ,   4.7352037 ],
[ 35.        ,   5.62828779],
[ 35.        ,   6.68981219],
[ 35.        ,   7.95154563],
[ 35.        ,   9.45124856],
[ 35.        ,  11.23380329],
[ 35.        ,  13.35255712],
[ 35.        ,  15.87091896],
[ 35.        ,  18.86425696],
```

```
[ 35.        ,  22.42215411],
[ 35.        ,  26.6510892 ],
[ 35.        ,  31.67762348],
[ 35.        ,  37.65218831],
[ 35.        ,  44.7535872 ],
[ 35.        ,  53.19434692],
[ 35.        ,  63.2270779 ],
[ 35.        ,  75.15203422],
[ 35.        ,  89.32609942],
[ 35.        , 106.17346718],
[ 35.        , 126.19833627],
[ 35.        , 150.        ],
[ 46.        ,   1.        ],
[ 46.        ,   1.18860521],
[ 46.        ,   1.41278235],
[ 46.        ,   1.67924046],
[ 46.        ,   1.99595396],
[ 46.        ,   2.37240127],
[ 46.        ,   2.81984851],
[ 46.        ,   3.35168663],
[ 46.        ,   3.98383219],
[ 46.        ,   4.7352037 ],
[ 46.        ,   5.62828779],
[ 46.        ,   6.68981219],
[ 46.        ,   7.95154563],
[ 46.        ,   9.45124856],
[ 46.        ,  11.23380329],
[ 46.        ,  13.35255712],
[ 46.        ,  15.87091896],
[ 46.        ,  18.86425696],
[ 46.        ,  22.42215411],
[ 46.        ,  26.6510892 ],
[ 46.        ,  31.67762348],
[ 46.        ,  37.65218831],
[ 46.        ,  44.7535872 ],
[ 46.        ,  53.19434692],
[ 46.        ,  63.2270779 ],
[ 46.        ,  75.15203422],
[ 46.        ,  89.32609942],
[ 46.        , 106.17346718],
[ 46.        , 126.19833627],
[ 46.        , 150.        ],
[ 59.        ,   1.        ],
[ 59.        ,   1.18860521],
[ 59.        ,   1.41278235],
[ 59.        ,   1.67924046],
[ 59.        ,   1.99595396],
[ 59.        ,   2.37240127],
[ 59.        ,   2.81984851],
[ 59.        ,   3.35168663],
[ 59.        ,   3.98383219],
[ 59.        ,   4.7352037 ],
[ 59.        ,   5.62828779],
[ 59.        ,   6.68981219],
[ 59.        ,   7.95154563],
[ 59.        ,   9.45124856],
```

```
[ 59.        ,  11.23380329],
[ 59.        ,  13.35255712],
[ 59.        ,  15.87091896],
[ 59.        ,  18.86425696],
[ 59.        ,  22.42215411],
[ 59.        ,  26.6510892 ],
[ 59.        ,  31.67762348],
[ 59.        ,  37.65218831],
[ 59.        ,  44.7535872 ],
[ 59.        ,  53.19434692],
[ 59.        ,  63.2270779 ],
[ 59.        ,  75.15203422],
[ 59.        ,  89.32609942],
[ 59.        , 106.17346718],
[ 59.        , 126.19833627],
[ 59.        , 150.        ],
[ 77.        ,   1.        ],
[ 77.        ,   1.18860521],
[ 77.        ,   1.41278235],
[ 77.        ,   1.67924046],
[ 77.        ,   1.99595396],
[ 77.        ,   2.37240127],
[ 77.        ,   2.81984851],
[ 77.        ,   3.35168663],
[ 77.        ,   3.98383219],
[ 77.        ,   4.7352037 ],
[ 77.        ,   5.62828779],
[ 77.        ,   6.68981219],
[ 77.        ,   7.95154563],
[ 77.        ,   9.45124856],
[ 77.        ,  11.23380329],
[ 77.        ,  13.35255712],
[ 77.        ,  15.87091896],
[ 77.        ,  18.86425696],
[ 77.        ,  22.42215411],
[ 77.        ,  26.6510892 ],
[ 77.        ,  31.67762348],
[ 77.        ,  37.65218831],
[ 77.        ,  44.7535872 ],
[ 77.        ,  53.19434692],
[ 77.        ,  63.2270779 ],
[ 77.        ,  75.15203422],
[ 77.        ,  89.32609942],
[ 77.        , 106.17346718],
[ 77.        , 126.19833627],
[ 77.        , 150.        ],
[100.        ,   1.        ],
[100.        ,   1.18860521],
[100.        ,   1.41278235],
[100.        ,   1.67924046],
[100.        ,   1.99595396],
[100.        ,   2.37240127],
[100.        ,   2.81984851],
[100.        ,   3.35168663],
[100.        ,   3.98383219],
[100.        ,   4.7352037 ],
```

```
[100.        ,   5.62828779],
[100.        ,   6.68981219],
[100.        ,   7.95154563],
[100.        ,   9.45124856],
[100.        ,  11.23380329],
[100.        ,  13.35255712],
[100.        ,  15.87091896],
[100.        ,  18.86425696],
[100.        ,  22.42215411],
[100.        ,  26.6510892 ],
[100.        ,  31.67762348],
[100.        ,  37.65218831],
[100.        ,  44.7535872 ],
[100.        ,  53.19434692],
[100.        ,  63.2270779 ],
[100.        ,  75.15203422],
[100.        ,  89.32609942],
[100.        , 106.17346718],
[100.        , 126.19833627],
[100.        , 150.        ]])
```

In [47]:
```
y
```

Out[47]:
```
array([  3.        ,   6.8       ,   8.53333333,  10.        ,
        12.06666667,  14.26666667,  16.73333333,  19.13333333,
        22.26666667,  25.86666667,  29.06666667,  35.        ,
        40.6       ,  46.8       ,  55.33333333,  64.33333333,
        75.        ,  87.46666667, 101.53333333, 119.06666667,
       138.46666667, 158.46666667, 189.66666667, 217.26666667,
       248.4       , 289.53333333, 339.93333333, 394.86666667,
       453.73333333, 537.33333333,   3.        ,   7.        ,
         8.46666667,  10.06666667,  12.13333333,  14.06666667,
        16.66666667,  19.2       ,  22.46666667,  26.06666667,
        30.4       ,  34.26666667,  39.46666667,  47.13333333,
        55.13333333,  65.4       ,  75.73333333,  88.93333333,
       102.        , 117.93333333, 139.        , 160.73333333,
       189.13333333, 219.6       , 245.4       , 288.86666667,
       338.53333333, 393.33333333, 460.33333333, 529.26666667,
         3.        ,   7.        ,   8.6       ,  10.06666667,
        12.06666667,  14.33333333,  16.46666667,  19.66666667,
        22.13333333,  26.06666667,  29.6       ,  34.2       ,
        40.8       ,  47.6       ,  56.53333333,  65.13333333,
        75.93333333,  87.73333333, 102.6       , 119.2       ,
       139.13333333, 162.06666667, 187.6       , 220.66666667,
       250.8       , 297.73333333, 342.8       , 392.06666667,
       458.4       , 531.2       ,   3.        ,   6.93333333,
         8.4       ,  10.06666667,  11.93333333,  14.2       ,
        16.6       ,  19.4       ,  22.2       ,  26.33333333,
        30.2       ,  34.86666667,  40.8       ,  47.53333333,
        56.13333333,  65.46666667,  75.66666667,  86.93333333,
       103.66666667, 120.4       , 139.86666667, 162.73333333,
       188.26666667, 220.4       , 255.33333333, 294.        ,
       346.66666667, 401.        , 462.53333333, 535.53333333,
         3.        ,   7.        ,   8.53333333,  10.13333333,
        11.93333333,  14.26666667,  16.86666667,  19.26666667,
        22.4       ,  25.93333333,  29.4       ,  34.93333333,
        40.73333333,  48.33333333,  55.2       ,  65.        ,
        75.4       ,  88.46666667, 102.6       , 120.6       ,
       138.6       , 161.33333333, 189.46666667, 220.33333333,
       253.8       , 293.4       , 345.6       , 399.53333333,
       461.6       , 541.13333333,   3.        ,   7.        ,
         8.4       ,  10.06666667,  12.        ,  14.13333333,
        16.66666667,  19.46666667,  22.53333333,  25.73333333,
        30.06666667,  35.2       ,  41.6       ,  47.8       ,
        56.06666667,  65.13333333,  76.33333333,  88.86666667,
       103.4       , 120.8       , 139.66666667, 161.66666667,
       188.8       , 219.46666667, 254.66666667, 299.93333333,
       346.        , 399.66666667, 464.93333333, 542.4       ,
         3.        ,   7.        ,   8.66666667,  10.13333333,
        12.06666667,  14.13333333,  16.73333333,  19.46666667,
        22.6       ,  26.26666667,  30.2       ,  35.6       ,
        41.13333333,  48.33333333,  56.46666667,  65.33333333,
        76.4       ,  89.06666667, 103.73333333, 118.66666667,
       140.06666667, 161.33333333, 188.4       , 219.86666667,
       256.        , 298.        , 344.93333333, 398.86666667,
       466.2       , 536.73333333,   3.        ,   7.        ,
         8.53333333,  10.06666667,  12.06666667,  14.13333333,
        16.66666667,  19.66666667,  22.33333333,  25.86666667,
        29.93333333,  35.2       ,  41.2       ,  47.66666667,
```

```
 56.06666667,  65.4       ,  75.73333333,  89.26666667,
103.66666667, 120.46666667, 140.26666667, 163.53333333,
190.33333333, 219.8       , 253.66666667, 298.73333333,
345.93333333, 401.6       , 466.93333333, 538.8       ,
  3.        ,   7.        ,   8.53333333,  10.        ,
 12.        ,  14.13333333,  16.66666667,  19.66666667,
 22.53333333,  25.6       ,  30.13333333,  35.46666667,
 41.        ,  48.06666667,  55.93333333,  65.53333333,
 76.4       ,  88.53333333, 103.2       , 120.26666667,
140.8       , 163.33333333, 189.53333333, 220.26666667,
256.86666667, 296.8       , 346.2       , 399.33333333,
466.06666667, 540.93333333,   3.        ,   7.        ,
  8.53333333,  10.13333333,  12.        ,  14.06666667,
 16.8       ,  19.33333333,  22.4       ,  25.8       ,
 30.4       ,  35.73333333,  40.8       ,  48.13333333,
 56.06666667,  65.4       ,  76.33333333,  89.46666667,
103.93333333, 120.8       , 140.2       , 163.2       ,
190.46666667, 219.8       , 256.2       , 294.33333333,
345.6       , 400.26666667, 467.8       , 537.26666667])
```

In [48]:
```
model = LinearRegression().fit(x, y)
```

In [49]:
```
print(f"R^2: {model.score(x, y)} ≈ 1")
```

```
R^2: 0.9961194694942691
```

In [50]:
```
model.coef_
```

Out[50]:
```
array([0.02098247, 3.64203576])
```

In [51]:
```
model.intercept_
```

Out[51]:
```
11.76540676227873
```

In [52]:
```
print(f"{model.coef_[0]} ≈ 0")
```

```
0.020982465577030398 ≈ 0
```