

Metoda najmniejszych kwadratów

(Dzmitry, Nikitsin, Gr6)

Krótkie wprowadzenie

Naszym celem w metodzie najmniejszych kwadratów jest znalezienie takiego wektora parametrów \mathbf{x} , który minimalizuje błąd dopasowania modelu do danych. Błąd ten definiujemy jako **sumę kwadratów różnic** między wartościami rzeczywistymi \mathbf{b} a przewidywanymi $A\mathbf{x}$ czyli mówiąc prościej szukamy minimalny błąd średniokwadratowy:

$$L(\mathbf{x}) = \sum_{i=1}^m (y_i - (A\mathbf{x})_i)^2 \rightarrow \min$$

Powyzsza suma to nic innego jak kwadrat normy euklidesowej (długości) wektora różnic. Możemy więc zapisać to używając wektora wyników \mathbf{b} (zawierającego y_i) oraz macierzy danych A :

$$L(\mathbf{x}) = \|\mathbf{b} - A\mathbf{x}\|^2$$

Korzystając z własności iloczynu skalarnego wektorów ($\mathbf{v} \cdot \mathbf{v} = \mathbf{v}^T \mathbf{v}$), zapisujemy funkcję w postaci algebraicznej:

$$(\mathbf{b} - A\mathbf{x})^T (\mathbf{b} - A\mathbf{x})$$

Po otwarciu nawiasów ostatecznie dostajemy funkcję, która jest łatwa do zminimalizowania:

$$L(\mathbf{x}) = \mathbf{b}^T \mathbf{b} - 2\mathbf{x}^T A^T \mathbf{b} + \mathbf{x}^T A^T A \mathbf{x}$$

Aby znaleźć minimum tej funkcji, musimy obliczyć jej **gradient** względem wektora \mathbf{x} i przyrównać go do zera. Gradient $\nabla_{\mathbf{x}} L$ to w rzeczywistości wektor składający się ze wszystkich **pochodnych cząstkowych** funkcji po poszczególnych zmiennych x_1, x_2, \dots, x_n :

$$\nabla_{\mathbf{x}} L(\mathbf{x}) = \begin{bmatrix} \frac{\partial L}{\partial x_1} \\ \frac{\partial L}{\partial x_2} \\ \vdots \\ \frac{\partial L}{\partial x_n} \end{bmatrix}$$

Każda pochodna cząstkowa $\frac{\partial L}{\partial x_k}$ mówi nam, jak zmienia się błąd, gdy zmieniamy tylko jeden parametr x_k . W minimum wszystkie te zmiany muszą wynosić zero jednocześnie. Obliczając gradient (czyli wykonując operacje pochodnych cząstkowych na całym wektorze jednocześnie), otrzymujemy:

1. Pochodna wyrazu wolnego $\mathbf{b}^T \mathbf{b}$ wynosi 0.

2. Pochodna wyrazu liniowego $-2\mathbf{x}^T A^T \mathbf{b}$ wynosi $-2A^T \mathbf{b}$.
3. Pochodna wyrazu kwadratowego $\mathbf{x}^T A^T A \mathbf{x}$ wynosi $2A^T A \mathbf{x}$.

Stawiamy warunek konieczny istnienia minimum ($\nabla_{\mathbf{x}} L = 0$):

$$-2A^T \mathbf{b} + 2A^T A \mathbf{x} = 0$$

Po przekształceniu otrzymujemy **Układ Równań Normalnych**:

$$A^T A \mathbf{x} = A^T \mathbf{b}$$

Mnożąc lewostronnie przez odwrotność macierzy $(A^T A)^{-1}$, otrzymujemy szukany wzór na \mathbf{x} :

$$\mathbf{x} = \underbrace{(A^T A)^{-1}}_{A^+} \underbrace{A^T \mathbf{b}}$$

Wyrażenie $(A^T A)^{-1} A^T$ nazywamy **macierzą pseudoodwrotną Moore'a-Penrose'a** (A^+). Pozwala ona rozwiązać problem minimalizacji sumy kwadratów błędów jednym działaniem macierzowym.

In [50]: #IMPORTS

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

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from numpy.polynomial import Chebyshev
```

```
In [51]: def pinvMP(A):
    """
        Oblicza macierz pseudoodwrotną Moore'a-Penrose'a dla macierzy A,
        korzystając z powyżej wyprowadzonego wzoru

    Zastosowanie:
    Używana do rozwiązywania nadokreślonych układów równań Ax=b
    w sensie najmniejszych kwadratów: x = A^+ b.

    :param A -- macierz numpy o wymiarach (m, n), gdzie m >= n
    :return A_pinv -- macierz pseudoodwrotna o wymiarach (n, m)
    :raise ValueError -- w przypadku, jeżeli macierz jest nieodwracalna
    """

    A_T = A.T
    ATA = A_T @ A

    # Odwrócenie macierzy (A^T * A)
    try:
        ATA_inv = np.linalg.inv(ATA)
    except np.linalg.LinAlgError:
        raise ValueError("Macierz A^T A jest nieodwracalna")

    A_pinv = ATA_inv @ A_T

    return A_pinv
```

Komentarz dotyczący stabilności numerycznej:

Zaimplementowane powyżej rozwiązanie korzystające z jawnego wyznaczenia macierzy $(A^T A)^{-1}$ jest poprawne teoretycznie (algebraicznie), jednak w praktyce inżynierskiej jest uznawane za mniej stabilne numerycznie.

Głównym powodem jest operacja obliczania iloczynu $A^T A$, która powoduje podniesienie do kwadratu wskaźnika uwarunkowania macierzy ($\kappa(A^T A) \approx \kappa(A)^2$). W przypadku macierzy źle uwarunkowanych (ill-conditioned), może to prowadzić do drastycznej utraty precyzji obliczeń zmiennoprzecinkowych lub błędów typu `Singular Matrix Error`, nawet jeśli wyjściowy problem jest rozwiązywalny.

W bibliotekach numerycznych (jak LAPACK/NumPy) preferowane są metody omijające bezpośrednie tworzenie macierzy $A^T A$, takie jak rozkład QR (patrz Zadanie 2) lub SVD (używane domyślnie w `np.linalg.pinv`), które zachowują pierwotny wskaźnik uwarunkowania $\kappa(A)$.

ZADANIE 1

Zapiszmy nadokreślony układ równań za pomocą macierzy $A : Ax = b$. Zaimplementuj rozwiązanie problemu aproksymacji liniowej za pomocą metody najmniejszych

kwadratów. Do wyznaczenia rozwiązania wykorzystaj macierz pseudoodwrotną Moore'a-Penrose'a

```
In [52]: def solve_LSQ_with_MoorePenrose_matrix(A, b):
    """
        Rozwiązuje problem najmniejszych kwadratów przy użyciu macierzy pseudoodwrotnej
        Ax = b -> x = A^+ b, gdzie A^+ jest macierzą pseudoodwrotną Moore'a-Penrose'a

        Funkcja oddelegowuje obliczenie A^+ do funkcji pomocniczej pinvMP.

        :param A: tablica 2D (array-like), macierz współczynników układu.
        :param b: tablica 1D lub 2D (array-like), wektor(y) zmiennych zależnych.
        :return: tablica 1D lub 2D reprezentująca rozwiązanie x układu Ax = b.
    """

    #A^+
    A_pseudo_inv = pinvMP(A)

    x = A_pseudo_inv @ b
    return x
```

ZADANIE 2

Zaimplementuj rozwiązanie problemu OLS przy pomocy macierzy pseudoodwrotnej Moore'a-Penrose'a zapisanej za pomocą macierzy Q i R uzyskanych z rozkładu oryginalnej macierzy A . Wykorzystaj w tym celu właściwości macierzy oraz rozkładu QR.

Wyprowadzenie macierzy pseudoodwrotnej za pomocą macierzy Q i R

Z wyżej napisanego definicja macierzy pseudoodwrotnej to:

$$A^+ = (A^T A)^{-1} A^T$$

Wiemy, że $A = QR$. Dla macierzy prostokątnej A ($m \times n, m \geq n$), stosujemy tzw. **zredukowany rozkład QR**, gdzie:

- Q jest macierzą ortonormalną o wymiarach $m \times n$ (ma własność $Q^T Q = I$).
- R jest macierzą górnopróbką kwadratową $n \times n$ (jest odwracalna, jeśli A ma pełny rząd).

Podstawiamy $A = QR$ do wzoru na A^+ :

$$\begin{aligned} A^+ &= ((QR)^T (QR))^{-1} (QR)^T = (R^T Q^T QR)^{-1} R^T Q^T = (R^T I R)^{-1} R^T Q^T = \\ &= (R^T R)^{-1} R^T Q^T = R^{-1} (R^T)^{-1} R^T Q^T = R^{-1} I Q^T \end{aligned}$$

Wynik końcowy:

$$A^+ = R^{-1} Q^T$$

Zatem nasze rozwiązanie $x = A^+ b$ przyjmuje postać:

$$x = R^{-1} Q^T b$$

```
In [53]: def solve_LSQ_with_QR(A, b):
    """
        Rozwiązuje problem OLS przy pomocy rozkładu QR i wyznaczenia
        macierzy pseudoodwrotnej z wzoru  $A^+ = R^{-1} * Q^T$ .
    :param A: tablica 2D (array-like), macierz współczynników układu.
    :param b: tablica 1D lub 2D (array-like), wektor(y) zmiennych zależnych.
    :return: tablica 1D lub 2D reprezentująca rozwiązanie x układu  $Ax = b \Leftrightarrow (R^+Q)x = b$ 
    """
    # Q,R (mode='reduced' -> R kwadratowe (n x n))
    Q, R = np.linalg.qr(A, mode='reduced')

    #R^-1
    R_inv = np.linalg.inv(R)

    #A^+
    A_pinv = R_inv @ Q.T

    x = A_pinv @ b
    return x
```

ZADANIE 4

Wygeneruj przykładowe zasumione próbki funkcji wielomianowej jednej zmiennej. Liczba próbek powinna wynosić ok. 100 dla przedziału o długości 10. Niech będzie to wielomian co najmniej siódmego stopnia. Za pomocą implementacji metod najmniejszych kwadratów z 1. i 2. punktu znajdź funkcje aproksymujące te dane. Dobierz odpowiedni stopień wielomianu i porównaj otrzymane wyniki. Do porównania, oprócz sumy kwadratu błędów wykorzystaj trzy inne metryki (np. błąd średniokwadratowy).

eeAby dopasować wielomian stopnia d postaci:

$$y = w_0 + w_1x + w_2x^2 + \cdots + w_dx^d$$

za pomocą metod zaimplementowanych z zad.1 oraz zad.2, przekształcono wektor wejściowy x w **macierz Vandermonde'a**. Dla m próbek wygląda ona następująco:

$$A = \begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^d \\ 1 & x_2 & x_2^2 & \dots & x_2^d \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_m & x_m^2 & \dots & x_m^d \end{bmatrix}$$

Wtedy wektor $w = [w_0, w_1, \dots, w_d]^T$ zawiera szukane współczynniki wielomianu.

Metryki oceny modelu

Do porównania jakości dopasowania dla różnych stopni wielomianu wykorzystano następujące metryki, gdzie y_i to wartość rzeczywista, a \hat{y}_i to wartość przewidziana przez model:

1. **SSE (Sum of Squared Errors):** Suma kwadratów błędów.

$$SSE = \sum_{i=1}^m (y_i - \hat{y}_i)^2$$

2. **MSE (Mean Squared Error):** Błąd średniokwadratowy.

$$MSE = \frac{1}{m} SSE$$

3. **RMSE (Root Mean Squared Error):** Pierwiastek błędu średniokwadratowego.

$$RMSE = \sqrt{MSE}$$

4. **MAE (Mean Absolute Error):** Średni błąd bezwzględny.

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

W celu weryfikacji poprawności zaimplementowanych metod aproksymacji, jako funkcję bazową do generowania danych wybrano **Wielomian Czebyszewa pierwszego rodzaju stopnia 8**, oznaczany jako $T_8(x)$.

Dlaczego ten wybór?

Wielomiany Czebyszewa odgrywają fundamentalną rolę w teorii aproksymacji ze względu na swoje unikalne własności:

1. **Silna oscylacja (Stress-test):** Wielomian $T_n(x)$ w przedziale $[-1, 1]$ posiada n miejsc zerowych oraz $n + 1$ ekstremów lokalnych, przyjmujących na przemian wartości -1 i 1 . Dla stopnia 8 oznacza to bardzo dużą zmienność funkcji (gęste "falowanie"). Jest to doskonały test dla algorytmu regresji – sprawdza, czy model potrafi uchwycić skomplikowaną strukturę danych, czy też ulegnie zjawisku *underfittingu* (wygładzenia).
2. **Własność minimaksowa:** Użycie funkcji o charakterystyce Czebyszewa pozwala na analizę zachowania modelu na krańcach przedziału, gdzie standardowe wielomiany (w bazie potęgowej) często wykazują niestabilność (efekt Rungego).

Definicja matematyczna

Wielomiany Czebyszewa definiowane są rekurencyjnie:

$$T_0(x) = 1$$

$$T_1(x) = x$$

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)$$

W naszym eksperymencie użyjemy wielomianu $T_8(x)$, który po rozwinięciu do bazy potęgowej przyjmuje postać:

$$T_8(x) = 128x^8 - 256x^6 + 160x^4 - 32x^2 + 1$$

Eksperyment zostanie przeprowadzony na znormalizowanym przedziale $x \in [-1, 1]$, z dodanym szumem losowym o rozkładzie normalnym, aby zasymulować rzeczywiste

warunki pomiarowe.

```
In [54]: # funkcja Generująca Dane (Wielomian Czebyszewa)
def generate_chebyshev_data(n_samples=100, noise_std=0.3):
    """
    Generuje zaszumione próbki na podstawie wielomianu Czebyszewa 8. stopnia (T8
    :return: x (próbki), y_true (idealne), y_noisy (zaszumione)
    """
    #definicja wielomianu T8: [0,0,0,0,0,0,0,1] -> 1*T8
    #P(x) = 0 * x^T_0 + 0 * x^T_1 + ... + 0 * x^T_7 + 1 * x^T_8(x)
    cheb_poly = Chebyshev([0] * 8 + [1])

    # 128x^8 - 256x^6 + 160x^4 - 32x^2 + 1
    p_true = cheb_poly.convert(kind=np.polynomial.Polynomial)

    # generowanie próbek w naturalnym przedziale Czebyszewa [-1, 1]
    x = np.linspace(-1, 1, n_samples)
    y_ideal = p_true(x)

    # szum
    np.random.seed(13)
    noise = np.random.normal(0, noise_std, size=len(x))
    y_noisy = y_ideal + noise

    return x, y_ideal, y_noisy

# funkcja Tworząca Macierz Vandermonde

def create_vandermonde_matrix(x, degree):
    """
    Tworzy macierz Vandermonde'a dla wektora x i zadanego stopnia wielomianu.
    Kolejność potęg: rosnąca (1, x, x^2, ..., x^d)
    Z w_0 * 1 + w_1 x + w_2 x^2 dla [x1, x2, ..., xd]
    tworzymy macierz postaci
    1 x1^1 x1^2 ... x1^d
    1 x2^1 x2^2 ... x2^d
    itd
    degree + 1 aby uwzględnić wyraz wolny
    """
    return np.vander(x, degree + 1, increasing=True)

# funkcja Obliczająca Metryki

def calculate_metrics(y_true, y_pred):
    """
    Oblicza zestaw metryk porównujących predykcję z danymi rzeczywistymi (lub za
    :return słownik z wynikami: SSE, MSE, RMSE, MAE.
    """
    residuals = y_true - y_pred
    sse = np.sum(residuals ** 2)
    mse = np.mean(residuals ** 2)
    rmse = np.sqrt(mse)
    mae = np.mean(np.abs(residuals))

    return {
        "SSE": sse,
```

```

        "MSE": mse,
        "RMSE": rmse,
        "MAE": mae
    }

def run_polynomial_experiment(degrees, x, y_noisy, y_ideal):
    """
    Przeprowadza serię aproksymacji dla różnych stopni wielomianu.
    Rysuje wykres i wypisuje tabelę metryk.
    """
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(18, 7))

    colors = ['salmon', 'turquoise', 'orange', 'purple']

    for i, deg in enumerate(degrees):
        #tworzenie macierzy Vandermonde -> A(vander)
        A = create_vandermonde_matrix(x, deg)

        #rozwiążanie układy Ax = b -> x
        coffs = solve_LSQ_with_MoorePenrose_matrix(A, y_noisy)

        # obliczenie wartości przewidywanych (y = Aw) z celu narysowania wykresu
        y_pred = A @ coffs

        # obliczenie metryk
        m = calculate_metrics(y_noisy, y_pred)
        print(
            f"{i + 1}. stopień: {deg}, SSE: {m['SSE']:.4f}, MSE: {m['MSE']:.4f}, RMSE: {m['RMSE']:.4f}, MAE: {m['MAE']:.4f}"
        )

        ax1.plot(x, y_pred, color=colors[i], linewidth=2, label=f'Aproksymacja stopnia {deg}')
        residuals = y_noisy - y_pred
        ax2.scatter(x, residuals, color=colors[i], label=f'Erro st. {deg}')

    # plot1
    ax1.scatter(x, y_noisy, color='gray', alpha=0.5, label='Pomiary zaszumione')
    ax1.plot(x, y_ideal, 'k--', linewidth=2, label='Oryginał (T8)')
    ax1.set_title("Dopasowanie modeli wielomianowych")
    ax1.set_xlabel("x")
    ax1.set_ylabel("y")
    ax1.legend()
    ax1.grid(True, alpha=0.3)

    # plot2
    ax2.axhline(0, color='black', linestyle='--', linewidth=1, label='Błąd zerowy')
    ax2.set_title("Wykres Rezyduów (y_noisy - y_pred)")
    ax2.set_xlabel("x")
    ax2.set_ylabel("Wartość błędu")
    ax2.legend()
    ax2.grid(True, alpha=0.3)

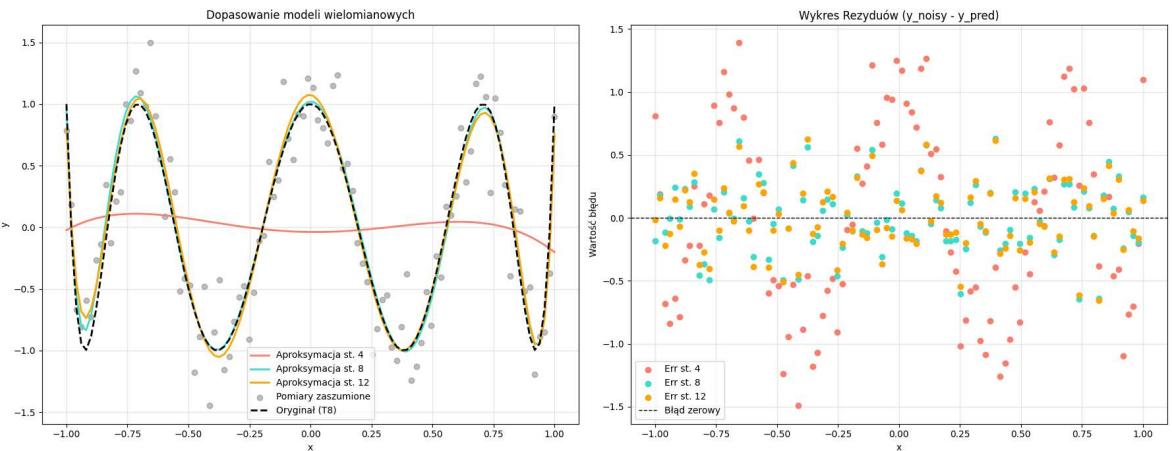
    fig.tight_layout() # tądnie układ odstęp między wykresami

```

In [55]:

```
x_samples, y_true, y_data = generate_chebyshev_data(n_samples=100, noise_std=0.3)
degrees_to_test = [4, 8, 12]
run_polynomial_experiment(degrees_to_test, x_samples, y_data, y_true)
```

1. stopień: 4, SSE: 57.1466, MSE: 0.5715, RMSE: 0.7560, MAE: 0.6628
2. stopień: 8, SSE: 7.4177, MSE: 0.0742, RMSE: 0.2724, MAE: 0.2187
3. stopień: 12, SSE: 7.1671, MSE: 0.0717, RMSE: 0.2677, MAE: 0.2189



Wnioski z analizy aproksymacji wielomianowej

Na podstawie przeprowadzonego eksperymentu, analizy wykresu dopasowania (po lewej), wykresu rezyduów (po prawej) oraz zestawienia metryk błędów, sformułowano następujące wnioski:

1. Niedopasowanie modelu (Underfitting) – Stopień 4

- **Analiza wizualna:** Czerwona krzywa jest zbyt gładka i nie potrafi odwzorować częstotliwości oscylacji oryginalnej funkcji T_8 .
- **Analiza Rezyduów (Kluczowe):** Spójrzmy na czerwone punkty na prawym wykresie. **Nie są one rozrzucone losowo**, lecz układają się w wyraźny wzór (kształt fali/sinusoidy). Taka struktura błędu jest dowodem na to, że model jest za prosty i systematycznie "gubi" istotną część informacji zawartej w danych.
- **Metryki:** Wysokie wartości błędów ($\text{MSE} \approx 0.57$) potwierdzają słabą jakość modelu.

2. Optymalne dopasowanie – Stopień 8

- **Analiza wizualna:** Błękitna krzywa idealnie podąża za linią przerywaną (oryginałem).
- **Analiza Rezyduów:** Błękitne punkty na prawym wykresie są **losowo rozrzucone wokół zera** (osi przerywanej). Brak widocznego wzorca w rezyduach oznacza, że model skutecznie "wyciągnął" z danych całą strukturę (sygnał), pozostawiając jedynie losowy szum pomiarowy (czyli to, co dodaliśmy funkcją np. `random.normal`). Jest to pożądany stan w regresji.
- **Metryki:** Drastyczny spadek błędu do poziomu wariancji szumu ($\text{MSE} \approx 0.074$).

3. Ryzyko przeuczenia (Overfitting) – Stopień 12

- **Analiza metryk:** Wielomian 12. stopnia osiągnął minimalnie niższe błędy niż stopień 8., jednak różnica jest marginalna.
- **Analiza Rezyduów:** Pomarańczowe punkty nakładają się na błękitne i również wykazują losowy charakter. Fakt, że bardziej skomplikowany model (st. 12) nie zmienia struktury rezyduów względem modelu prostszego (st. 8), potwierdza, że dodatkowe stopnie swobody są zbędne i model zaczyna dopasowywać się do pojedynczych odchyleń szumu.

Podsumowanie: Najlepszym modelem jest wielomian **8. stopnia**. Decydującym argumentem jest analiza rezyduów: dla stopnia 4. błędy mają strukturę systematyczną (model źle dobrany), natomiast dla stopnia 8. błędy stają się szumem białym (model poprawny). Stopień 12 nie wnosi nowej jakości, zwiększając jedynie koszt obliczeniowy.

ZADANIE 5

W samodzielnie wybranym zbiorze danych znajdź liniową zależność opisującą wartość jednego z atrybutów w funkcji kombinacji liniowej pozostałych. Podziel znaleziony zbiór na dwie części: zbiór treningowy (uczący) i zbiór testowy. Do danych treningowych dopasuj funkcję aproksymującą i sprawdź dokładność modelu w zbiorze testowym.

W tym zadaniu analizujemy zbiór danych opisujący cechy fizyczne ryb. Naszym celem jest znalezienie liniowej zależności pozwalającej przewidzieć **wagę ryby (Weight)** na podstawie jej wymiarów geometrycznych.

Zbiór danych zawiera następujące atrybuty:

- **Zmienna celu (y):** Weight (Waga w gramach)
- **Zmienne objaśniające (x_1, \dots, x_5):**
 - Length1 (Długość standardowa)
 - Length2 (Długość w widelcu)
 - Length3 (Długość całkowita)
 - Height (Wysokość)
 - Width (Szerokość)

Atrybut Species (Gatunek) jest wartością kategoryczną (tekstową) i zostanie pominięty w prostym modelu regresji liniowej, który opiera się na zależnościach liczbowych.

Model matematyczny

Szukamy wektora współczynników $w = [w_0, w_1, w_2, w_3, w_4, w_5]^T$, który minimalizuje błąd dopasowania w równaniu:

$$\text{Weight} \approx w_0 + w_1 \cdot \text{L1} + w_2 \cdot \text{L2} + w_3 \cdot \text{L3} + w_4 \cdot \text{H} + w_5 \cdot \text{W}$$

Macierz planu A dla m próbek, po uwzględnieniu wyrazu wolnego (bias, w_0), przyjmuje postać:

$$A = \begin{bmatrix} 1 & \text{Length1}_1 & \text{Length2}_1 & \text{Length3}_1 & \text{Height}_1 & \text{Width}_1 \\ 1 & \text{Length1}_2 & \text{Length2}_2 & \text{Length3}_2 & \text{Height}_2 & \text{Width}_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \text{Length1}_m & \text{Length2}_m & \text{Length3}_m & \text{Height}_m & \text{Width}_m \end{bmatrix}$$

Zbiór zostanie podzielony na część treningową (do wyznaczenia w) i testową (do weryfikacji).

In [56]: df = pd.read_csv("data/Fish.csv")

```
df
```

```
Out[56]:
```

	Species	Weight	Length1	Length2	Length3	Height	Width
0	Bream	242.0	23.2	25.4	30.0	11.5200	4.0200
1	Bream	290.0	24.0	26.3	31.2	12.4800	4.3056
2	Bream	340.0	23.9	26.5	31.1	12.3778	4.6961
3	Bream	363.0	26.3	29.0	33.5	12.7300	4.4555
4	Bream	430.0	26.5	29.0	34.0	12.4440	5.1340
...
154	Smelt	12.2	11.5	12.2	13.4	2.0904	1.3936
155	Smelt	13.4	11.7	12.4	13.5	2.4300	1.2690
156	Smelt	12.2	12.1	13.0	13.8	2.2770	1.2558
157	Smelt	19.7	13.2	14.3	15.2	2.8728	2.0672
158	Smelt	19.9	13.8	15.0	16.2	2.9322	1.8792

159 rows × 7 columns

```
In [57]:
```

```
#przygotowanie danych
X = df[df.columns[2:]]
Y = df[df.columns[1]]

#podziął
X_train, X_test, Y_train, Y_test = train_test_split(x, y, test_size=0.3, random_

#ręczne dodanie biasu
A_train = np.c_[np.ones(len(x_train)), x_train]
A_test = np.c_[np.ones(len(x_test)), x_test]
```

```
In [58]:
```

```
def print_metrics(dict_of_metrics, list_of_metrix):

    """Pomocnicza funkcja do wypisywania metryk"""
    for metric in list_of_metrix:
        print(f'{metric}: {dict_of_metrics[metric]:.4f}')
metrix_list = ['SSE', 'MSE', 'RMSE', 'MAE']
```

```
In [59]:
```

```
w_lin = solve_LSQ_with_MoorePenrose_matrix(A_train, y_train)

y_pred_train = A_train @ w_lin
y_pred_test = A_test @ w_lin

residuals_train = y_train - y_pred_train
residuals_test = y_test - y_pred_test

# metryki
metrics_lin_train = calculate_metrics(y_train, y_pred_train)
metrics_lin_test = calculate_metrics(y_test, y_pred_test)

print("\nWyniki Modelu Liniowego(TRAIN):")
print_metrics(metrics_lin_train, metrix_poly)
print("\nWyniki Modelu Liniowego(TEST):")
```

```

print_metrics(metrics_lin_test, metrics_poly)

#plot1(real vs pred)
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 6))

ax1.scatter(y_train, y_pred_train, color='turquoise', label='Zbiór Treningowy')
ax1.scatter(y_test, y_pred_test, color='salmon', label='Zbiór Testowy')

all_vals = np.concatenate([y_train, y_test, y_pred_train, y_pred_test])
min_val, max_val = all_vals.min(), all_vals.max()
ax1.plot([min_val, max_val], [min_val, max_val], '--k', alpha=0.8, linewidth=1,
         label='Idealne dopasowanie')

ax1.set_title("Model Liniowy: Rzeczywista vs Przewidywana")
ax1.set_xlabel("Rzeczywista Waga [g]")
ax1.set_ylabel("Przewidywana Waga [g]")
ax1.legend()
ax1.grid(True, alpha=0.3)

#plot2(residuum)
ax2.scatter(y_pred_train, residuals_train, color='turquoise', label='Zbiór Treningowy')
ax2.scatter(y_pred_test, residuals_test, color='salmon', label='Zbiór Testowy')

ax2.axhline(y=0, linestyle='--', color='black', alpha=0.8, linewidth=1, label='Błąd zerowy')

ax2.set_title("Model Liniowy: Wykres Rezyduów")
ax2.set_xlabel("Przewidywana Waga [g]")
ax2.set_ylabel("Rezyduum (Błąd) [g]")
ax2.legend()
ax2.grid(True, alpha=0.3)

fig.tight_layout() # ładnie układają odstępy między wykresami

```

Wyniki Modelu Liniowego(TRAIN):

SSE: 1378993.3215

MSE: 12423.3633

RMSE: 111.4601

MAE: 85.1257

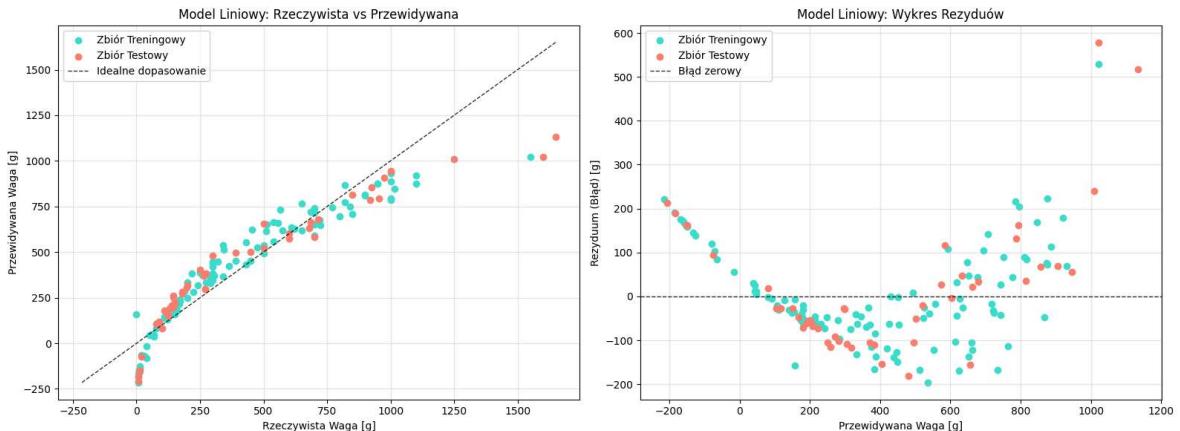
Wyniki Modelu Liniowego(TEST):

SSE: 1099574.4493

MSE: 22907.8010

RMSE: 151.3532

MAE: 106.2385



Wnioski z analizy regresji liniowej (Fish Market)

W zadaniu przeprowadzono predykcję wagi ryb na podstawie ich wymiarów geometrycznych (Długości, Szerokości, Wysokości) przy użyciu modelu liniowego. Model został nauczony na 70% danych (zbiór treningowy) i zweryfikowany na pozostałych 30% (zbiór testowy).

1. Ocena ilościowa (Metryki):

- **RMSE (Pierwiastek błędu średniokwadratowego):** Wynosi **111.46 g**. Oznacza to, że model myli się przeciętnie o ponad 100 gramów przy szacowaniu wagi ryby.
- **MAE (Średni błąd bezwzględny):** Wynosi **85.13 g**.
- Biorąc pod uwagę, że wagi ryb w zbiorze wahają się od kilku gramów do ponad kilograma, błąd rzędu 100g jest znaczący, szczególnie dla mniejszych osobników.

2. Analiza wykresu "Rzeczywistość vs Predykcja" (po lewej):

Wykres ujawnia systematyczne odchylenia od idealnego dopasowania (linia przerywana):

- **Problem ujemnych wag:** Dla najmniejszych ryb (blisko zera na osi X), model przewiduje wagi ujemne (punkty turkusowe i łososiowe spadające poniżej zera na osi Y). Jest to fizycznie niemożliwe i dyskwalifikuje model liniowy w tym zakresie.
- **Niedoszacowanie dla dużych ryb:** Dla największych okazów (powyżej 1000g), punkty leżą wyraźnie pod linią idealną, co oznacza, że model zaniża ich wagę.

3. Analiza Wykresu Rezyduów (po prawej) – Diagnoza:

Rozkład błędów (rezyduów) ma charakterystyczny kształt **paraboli** (U-shape).

- Model systematycznie przeszacowuje wagę ryb średnich (błędy ujemne) i niedoszacowuje ryb małych oraz dużych (błędy dodatnie).
- Taka struktura rezyduów jest dowodem na to, że **zależność między wymiarami a wagą nie jest liniowa**.

ZADANIE 6

Przeprowadź analizę zależności szukanej wartości od każdego z atrybutów z osobna. Czy założenie liniowej zależności przewidywanej zmiennej od każdej z cech jest odpowiednie? Zaproponuj alternatywną formułę zależności pomiędzy szukaną wartością, a cechami. Znajdź optymalne współczynniki dla tego rozwijania.

Poprzednia analiza (Zadanie 5) wykazała, że model liniowy systematycznie myli się w sposób przewidywalny (rezydua w kształcie paraboli). Aby to naprawić, przeprowadzimy szczegółową analizę zależności zmiennej celu (`Weight`) od poszczególnych cech.

Propozycja alternatywnej formuły (Regresja Wielomianowa)

Zamiast prostej kombinacji liniowej:

$$y = w_0 + w_1x_1 + \dots + w_5x_5$$

Zastosujemy transformację cech do **wielomianu stopnia 2**. Oznacza to, że nasz model będzie uwzględniał nie tylko same wymiary (x_i), ale też ich kwadraty (x_i^2) oraz interakcje

między nimi ($x_i \cdot x_j$). Pozwoli to modelowi na tworzenie zakrzywionych powierzchni decyzyjnych, lepiej dopasowanych do naturalnej budowy ryb.

Nowa macierz danych A będzie zawierać kolumny typu:

$$1, \quad L1, \quad L1^2, \quad L1 \cdot H, \quad H^2, \quad \dots$$

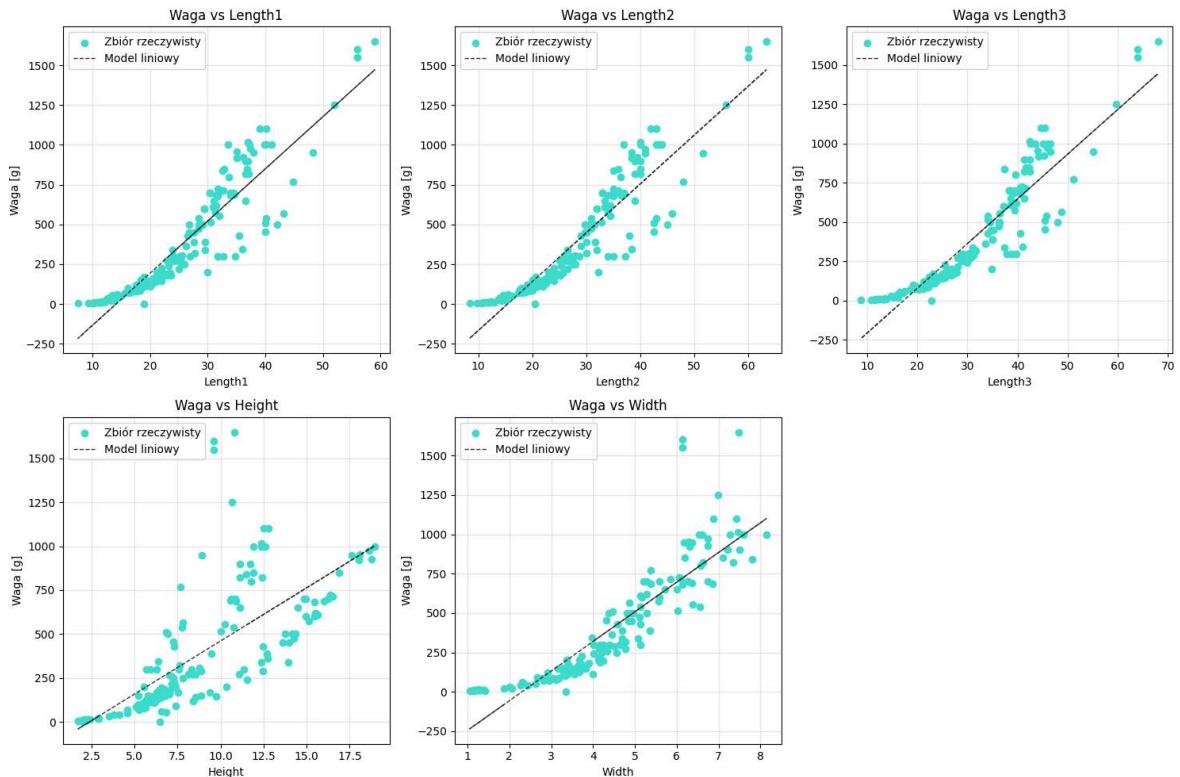
```
In [60]: feature_names = df.columns[2:]

# 5 wykresów - waga vs wszysko
fig, axes = plt.subplots(2, 3, figsize=(15, 10))
axes = axes.flatten()

for i, feature in enumerate(feature_names):
    ax = axes[i]
    x_feat = df[feature]
    ax.scatter(x_feat, y, color='turquoise', label='Zbiór rzeczywisty')
    ax.set_title(f"Waga vs {feature}")
    ax.set_xlabel(feature)
    ax.set_ylabel("Waga [g]")
    ax.grid(True, alpha=0.3)

    # aproksymacja liniowa chmury punktów
    # x, y, degree
    a, b = np.polyfit(x_feat, y, 1)
    ax.plot(x_feat, a * x_feat + b, 'k--', alpha=0.8, linewidth=1, label='Model')
    ax.legend(loc='upper left')

fig.delaxes(axes[5])
plt.tight_layout()
```



Analiza zależności cech (Wnioski z wykresów)

Przeprowadzona analiza wizualna zależności zmiennej celu (`Weight`) od poszczególnych cech geometrycznych (`Length1` , `Length2` , `Length3` , `Height` , `Width`) ujawnia, że:

1. **Nieliniowość:** Na wszystkich wykresach widoczna jest wyraźna nieliniowa tendencja wzrostowa. Punkty nie układają się wzdłuż linii prostej (zaznaczonej przerywaną kreską), lecz tworzą krzywą przypominającą funkcję potową lub wykładniczą. Jest to szczególnie widoczne dla cech związanych z długością (`Length1-3`), gdzie przyrost wagi przyspiesza wraz ze wzrostem wymiaru ryby.
2. **Niedopasowanie modelu liniowego:** Prosta regresji liniowej (fit liniowy) systematycznie błędnie szacuje wartości w skrajnych przedziałach:
 - Dla małych wymiarów model przewiduje wartości ujemne (poniżej zera na osi Y), co jest fizycznie niemożliwe.
 - Dla dużych wymiarów model zaniża rzeczywistą wagę.

```
In [61]: # degree=2 oznacza: x1^2, x2^2 oraz x1*x2. bias: dodanie 1
poly = PolynomialFeatures(degree=2, include_bias=True)
#fit transform dodaje do x wszystkie kwadratowe interakcje elementow x oraz kolu
#dla n cech stopnia d: symbol_newtona z n+d nad n (7 nad 2) -> 21 | bias, 5 cech
#pozwala wykryc nieliniowe zaleznosci
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.transform(X_test)

w_poly = solve_LSQ_with_MoorePenrose_matrix(X_train_poly, y_train)

y_pred_train_poly = X_train_poly @ w_poly
y_pred_test_poly = X_test_poly @ w_poly

residuals_train_poly = y_train - y_pred_train_poly
residuals_test_poly = y_test - y_pred_test_poly

# metryki
metrics_poly_train = calculate_metrics(y_train, y_pred_train_poly)
metrics_poly_test = calculate_metrics(y_test, y_pred_test_poly)

print("\nWyniki Modelu Nieliniowego(TRAIN):")
print_metrics(metrics_poly_train, metrix_list)
print("\nWyniki Modelu Nieliniowego(TEST):")
print_metrics(metrics_poly_test, metrix_list)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 6))

#plot1
ax1.scatter(y_train, y_pred_train_poly, color='turquoise', label='Zbiór Treningowy')
ax1.scatter(y_test, y_pred_test_poly, color='salmon', label='Zbiór Testowy')

# linia idealna
min_val, max_val = y.min(), y.max()
ax1.plot([min_val, max_val], [min_val, max_val], '--k', alpha=0.8, linewidth=1)

ax1.set_title("Model Wielomianowy: Rzeczywista vs Przewidywana")
ax1.set_xlabel("Rzeczywista Waga [g]")
ax1.set_ylabel("Przewidywana Waga [g]")
ax1.legend()
ax1.grid(True, alpha=0.3)
```

```

#plot2
ax2.scatter(y_pred_train_poly, residuals_train_poly, color='turquoise', label='Zbiór Treningowy')
ax2.scatter(y_pred_test_poly, residuals_test_poly, color='salmon', label='Zbiór Testowy')
ax2.axhline(y=0, linestyle='--', color='black', alpha=0.8, linewidth=1, label='Błąd zerowy')

ax2.set_title("Model Wielomianowy: Wykres Rezyduów")
ax2.set_xlabel("Przewidywana Waga [g]")
ax2.set_ylabel("Rezyduum (Błąd) [g]")
ax2.legend()
ax2.grid(True, alpha=0.3)

fig.tight_layout()

```

Wyniki Modelu Nieliniowego(TRAIN):

SSE: 189536.7882

MSE: 1707.5386

RMSE: 41.3224

MAE: 27.3399

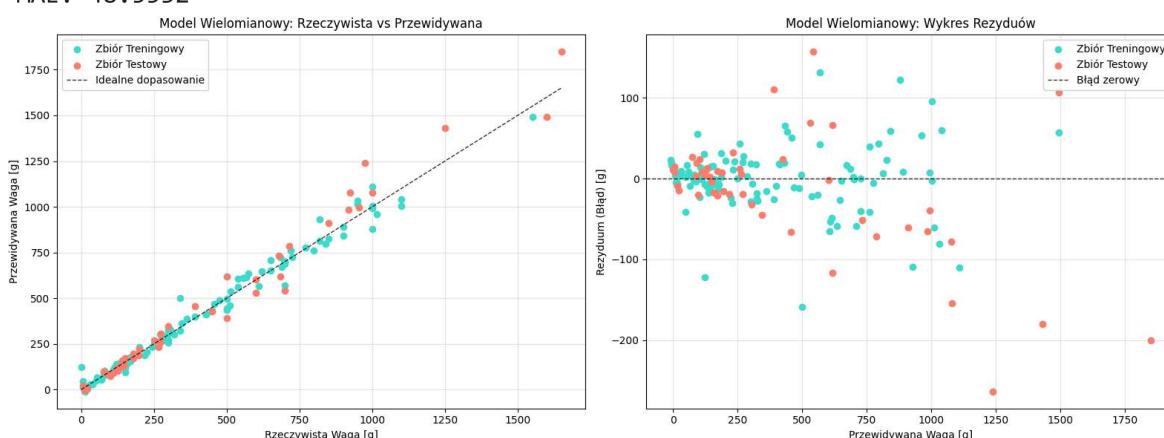
Wyniki Modelu Nieliniowego(TEST):

SSE: 274999.6682

MSE: 5729.1598

RMSE: 75.6912

MAE: 48.5532



Wniosek z analizy regresji wielomianowej

1. Skuteczność transformacji wielomianowej:

Zastosowanie `PolynomialFeatures(degree=2)` pozwoliło na wprowadzenie do modelu interakcji między cechami (np. $L_n \cdot H$) oraz ich kwadratów. Dzięki temu model regresji liniowej (działający teraz w przestrzeni 21 cech) mógł dopasować się do krzywoliniowej natury danych.

2. Wyniki ilościowe:

Model wielomianowy zredukował błąd RMSE na zbiorze testowym o połowę (**z ~151g do ~75g**) względem modelu liniowego. Wykres rezyduów (błędów) stracił wyraźną strukturę paraboli widoczną w modelu liniowym i stał się bardziej losowy (przypominający szum biały), co świadczy o poprawnym przechwyceniu sygnału przez model.

3. Zjawisko Przeuczenia (Overfitting):

Załobserwowano różnicę między błędem treningowym (RMSE ~41) a testowym (RMSE ~75). Wskazuje to na lekkie przeuczenie. Model wielomianowy przy małej liczbie danych (zbior Fish jest mały) i

dużej liczbie cech (21 po transformacji) zaczyna "uczyć się szumu". Mimo to, jego zdolność generalizacji jest nadal znacznie lepsza niż modelu liniowego.

ZADANIE 7

Celem tego zadania jest zestawienie wyników uzyskanych w punktach 5. (Regresja Liniowa) i 6. (Regresja Wielomianowa) oraz ocena jakości modeli na podstawie wyznaczonych metryk.

Zestawienie Wyników (Tabele Metryk)

Poniżej przedstawiono wartości czterech metryk błędu dla obu modeli, z podziałem na zbiór treningowy (na którym model się uczył) i testowy (na którym był sprawdzany).

Tabela 1: Wyniki Modelu Liniowego (Underfitting)

Metryka	Opis	Zbiór Treningowy	Zbiór Testowy
SSE	Suma Kwadratów Błędów	1 378 993.32	1 099 574.45
MSE	Błąd Średniokwadratowy	12 423.36	22 907.80
RMSE	Pierwiastek Błędu Średniokw. 111.46		151.35
MAE	Średni Błąd Bezwzględny	85.13	106.24

Tabela 2: Wyniki Modelu Wielomianowego st. 2 (Lepsze Dopasowanie)

Metryka	Opis	Zbiór Treningowy	Zbiór Testowy
SSE	Suma Kwadratów Błędów	189 536.79	274 999.67
MSE	Błąd Średniokwadratowy	1 707.54	5 729.16
RMSE	Pierwiastek Błędu Średniokw. 41.32		75.69
MAE	Średni Błąd Bezwzględny	27.34	48.55

Szczegółowa Analiza Metryk

Porównanie Model Liniowy vs. Wielomianowy (Skok Jakościowy) Najważniejszym wnioskiem jest drastyczna poprawa wyników po zastosowaniu inżynierii cech (`PolynomialFeatures`).

- **Spadek Błędu:** Błąd RMSE na zbiorze testowym spadł z **151.35 g** (model liniowy) do **75.69 g** (model wielomianowy). Oznacza to **dwukrotną redukcję błędu**.
- **Interpretacja:** Model liniowy był zbyt prosty, by opisać zależność wagi od wymiarów (zjawisko **Underfittingu**). Waga ryby zależy od jej objętości (zależność sześcienna lub kwadratowa od wymiarów), a nie liniowo od długości. Dodanie interakcji cech (np. $L_n \cdot H$) pozwoliło modelowi "zrozumieć" geometrię problemu.

Diagnostyka Przeuczenia (Overfitting) Analizując różnice między wynikami na zbiorze treningowym a testowym, możemy ocenić zdolność generalizacji modeli:

1. **Model Liniowy:** Różnica między RMSE treningowym (111) a testowym (151) jest widoczna, ale oba błędy są bardzo wysokie. Głównym problemem jest tutaj wysokie obciążenie (bias) modelu, a nie wariancja.
2. **Model Wielomianowy:** Tutaj obserwujemy ciekawszą zależność.
 - RMSE Trening: **41.32**
 - RMSE Test: **75.69** Błąd na zbiorze testowym jest prawie **dwukrotnie wyższy** niż na treningowym. Jest to sygnał **lekkiego przeuczenia (overfittingu)**. Model dopasował się bardzo precyzyjnie do danych, które widział (Train), ale na nowych danych (Test) radzi sobie nieco gorzej. Mimo to, wynik 75.69 jest nadal znacznie lepszy niż 151.35 w modelu liniowym, więc akceptujemy to ryzyko.

Analiza Rezyduów (Struktura Błędu)

- W modelu liniowym (Zadanie 5) rezydua układały się w wyraźną parabolę (systematyczne zaniżanie wag dla ryb małych i bardzo dużych).
- W modelu wielomianowym (Zadanie 6) wykres rezyduów stał się bardziej chaotyczny i skupiony wokół zera. Oznacza to, że model usunął systematyczny błąd strukturalny i pozostał głównie nieskorelowany szum (błąd losowy), co jest cechą dobrego modelu regresji.