Rozwiązywanie układów równań NxM

DO ZROBIENIA PODPUNKT 3.4 ZAPYTAC SIE NA ZAJECIACH

Instrukcja: Na zajęciach należy wykonać poniższe zadania, a następnie sporządzić sprawozdanie zawierające odpowiedzi (w postaci kodu) z komentarzami w środowisku Jupyter Notebook i umieścić je na platformie e-learningowej.

Ponizsze funckje będą niezbędne to dzisiejszego laboratorium:

In [40]:
```
import scipy as sp
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from scipy import linalg
from  datetime import datetime
import pickle
from typing import Union, List, Tuple
import time
import psutil
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

def get_memory_usage():
    process = psutil.Process()
    return process.memory_info().rss  # Zwraca zużycie pamięci w bajtach

def square_from_rectan(A: np.ndarray, b: np.ndarray):
    if not isinstance(A, np.ndarray) or not isinstance(b, np.ndarray):
        print("Błąd: A i b muszą być macierzami numpy.")
        return None

    m, n = A.shape

    if m != len(b):
        print("Błąd: Niezgodne wymiary macierzy A i wektora b.")
        return None

    if n > m:
        print("Błąd: Macierz A musi mieć mniej kolumn niż wierszy.")
        return None

    # Zbuduj nową macierz kwadratową
    A_square = np.dot(A.T, A)

    # Zbuduj nowy wektor
    b_square = np.dot(A.T, b)

    return A_square, b_square


def residual_norm(A:np.ndarray,x:np.ndarray, b:np.ndarray):
    if A.shape[0] != A.shape[1] or A.shape[0] != x.shape[0] or A.shape[0] !=
        print("Niepoprawne wymiary")
        return None

    # Obliczenie residuum: r = b - Ax
    residuum = b - np.dot(A, x)
    # Obliczenie normy residuum
    norma_residuum = np.linalg.norm(residuum)
    return norma_residuum
```

Cel zajęć: Celem zajęć jest zapoznanie się z numerycznymi metodami rozwiązywania układów równań liniowych w postaci macierzowej, z rzadkimi macierzami prostokątnymi. Czyli dana jest macierz $\mathbf{A}$ prostokątna o wymiarach $(m \times n)$ i rzadka

(https://pl.wikipedia.org/wiki/Macierz_rzadka) oraz wektor $\mathbf{b}$ $(m \times 1)$, należy rozwiązać układ równań postaci:

$$\mathbf{Ax} = \mathbf{b}$$

gdzie $\mathbf{A}$ to macierz współczynników z lewej strony równania, wektor $\mathbf{x}$ jest wektorem zmiennych a wektor $\mathbf{b}$ wyników prawej strony równania.

Zadanie 1

Dane jest $m = 50$ oraz $n = 12$.

Rozwiąż układ równań $\mathbf{Ax} = \mathbf{b}$ postaci:

$$\begin{bmatrix} 1 & t_0 & t_0^2 & \cdots & t_0^{n-1} \\ 1 & t_1 & t_1^2 & \cdots & t_1^{n-1} \\ 1 & t_2 & t_2^2 & \cdots & t_2^{n-1} \\ \vdots & \cdots & \ddots & \cdots & \vdots \\ 1 & t_{m-1} & t_{m-1}^2 & \cdots & t_{m-1}^{n-1} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{n-1} \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_{m-1} \end{bmatrix}$$

za pomocą następujacych metod:

1. Przekształcenia układu równań do postaci: $\mathbf{A}^{\mathrm{T}}\mathbf{Ax} = \mathbf{A}^{\mathrm{T}}\mathbf{b}$ (zdefinuj funkcję *square_from_rectan* która przekształci odpowiednio macierz A i wektor b) i stosując funkcję solve, z poprzednich zajęć.
2. Domyślnej metody Pythona rozwiązywania układów równań z macierzą prostokątną lstsq (https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.linalg.lstsq.html#numpy-linalg-lstsq).
3. Rozkładu QR rozwiązywania układów równań podanego na wykładzie. Do dokonania rozkładu QR w Pythonie używa się funkcji qr (https://docs.scipy.org/doc/numpy-1.15.1/reference/generated/numpy.linalg.qr.html), natomiast do rozwiązywania układu równań z macierzą trójkątną służy funkcja solve_triangular (https://docs.scipy.org/doc/scipy/reference/generated/scipy.linalg.solve_triangular.html)
4. Metody rozkładu SVD rozwiązywania układów równań podanego na wykładzie.
5. Porównaj czasy wykonania wszystkich metod oraz zużycie pamięci.
6. Porównaj jakość otrzymanych wyników

In [41]:
```
#Dane
m = 50
n = 12
t = np.linspace(0, 1, m)  # Przykładowe wartości t
A = np.column_stack([t**i for i in range(n)]) #Tworzy macierz A, gdzie każdo
x_true = np.random.rand(n) #Generuje losowy wektor x_true o długości n któro
b = np.dot(A, x_true) #Oblicza wektor wyników b mnożąc macierz A przez wekto
```

In [42]:
```
#Zadanie 1.1

memory_before_square_from_rectan = get_memory_usage()
start_time = time.time()
# Przekształcenie układu równań
A_square, b_square = square_from_rectan(A, b)

# Rozwiązanie układu równań
x_solution = np.linalg.solve(A_square, b_square)

# Wypisz rozwiązanie
print("Rozwiązanie układu równań:", x_solution)
time_square_from_rectan = time.time() - start_time
memory_after_square_from_rectan = get_memory_usage()
print(memory_before_square_from_rectan)
print(memory_after_square_from_rectan)
```

```
Rozwiązanie układu równań: [0.26434305 0.10451972 0.16552354 0.32453277 0.
4999778  0.94745091
 0.71942037 0.00242259 0.18810783 0.59720796 0.2772499  0.90439171]
195432448
195457024
```

In [43]:
```
#Zadanie 1.2

memory_before_lstsq = get_memory_usage()
start_time = time.time()
# Przykładowe dane
m = 50
n = 12
t = np.linspace(0, 1, m)
A = np.column_stack([t**i for i in range(n)])
x_true = np.random.rand(n)
b = np.dot(A, x_true)

# Użyto lstsq do rozwiązania układu równań
x_solution, residuals, rank, singular_values = np.linalg.lstsq(A, b, rcond=N

# Wypisz rozwiązanie
print("Rozwiązanie układu równań:", x_solution)
time_lstsq = time.time() - start_time
memory_after_lstsq = get_memory_usage()
```

```
Rozwiązanie układu równań: [0.43753435 0.68968937 0.62723151 0.40387962 0.
07733701 0.75563576
 0.41333426 0.75211868 0.63442146 0.50351221 0.55665892 0.41096743]
```

In [44]:
```
#Zadanie 1.3
from scipy.linalg import qr, solve_triangular

memory_before_qr = get_memory_usage()
start_time = time.time()
# Rozkład QR
Q, R = qr(A, mode='economic')

# Rozwiązanie układu równań
y = np.dot(Q.T, b)
x_solution = solve_triangular(R, y)

# Wypisz rozwiązanie
print("Rozwiązanie układu równań:", x_solution)
time_qr = time.time() - start_time
memory_after_qr = get_memory_usage()
```

```
Rozwiązanie układu równań: [0.43753435 0.68968937 0.62723151 0.40387962 0.
07733701 0.75563576
 0.41333425 0.75211869 0.63442145 0.50351221 0.55665892 0.41096743]
```

In [45]:
```
#Zadanie 1.4

memory_before_svd = get_memory_usage()
start_time = time.time()
# Rozkład SVD
U, Sigma, VT = np.linalg.svd(A, full_matrices=False)

# Obliczenie wektora wag
w = np.dot(U.T, b)

# Obliczenie rozwiązania
x_solution = np.dot(VT.T, w / Sigma)

# Wypisz rozwiązanie
print("Rozwiązanie układu równań:", x_solution)
time_svd = time.time() - start_time
memory_after_svd = get_memory_usage()
```

```
Rozwiązanie układu równań: [0.43753435 0.68968937 0.62723151 0.40387962 0.
07733701 0.75563576
 0.41333425 0.75211869 0.63442145 0.50351222 0.55665892 0.41096743]
```

In [46]:
```
print("Czas wykonania square_from_rectan:", time_square_from_rectan * 1000,
print(f"Zużycie pamięci przed: {memory_before_square_from_rectan} B, po: {me
print("Czas wykonania lstsq:", time_lstsq * 1000, "ms.")
print(f"Zużycie pamięci przed: {memory_before_lstsq} B, po: {memory_after_ls
print("Czas wykonania qr:", time_qr * 1000, "ms.")
print(f"Zużycie pamięci przed: {memory_before_qr} B, po: {memory_after_qr} B
print("Czas wykonania svd:", time_svd * 1000, "ms.")
print(f"Zużycie pamięci przed: {memory_before_svd} B, po: {memory_after_svd}
```

```
Czas wykonania square_from_rectan: 0.9992122650146484 ms.
Zużycie pamięci przed: 195432448 B, po: 195457024 B

Czas wykonania lstsq: 0.9996891021728516 ms.
Zużycie pamięci przed: 195457024 B, po: 195457024 B

Czas wykonania qr: 0.4706382751464844 ms.
Zużycie pamięci przed: 195457024 B, po: 195457024 B

Czas wykonania svd: 0.2853870391845703 ms.
Zużycie pamięci przed: 195461120 B, po: 195461120 B
```

Zadanie 2

Utwórz dwa wektory x_1 oraz x_2 opiujące dochód i wydatki pewnego gospodarstwa.

Dochody = [210, 270, 290, 310, 370, 400, 450, 480, 510, 520]

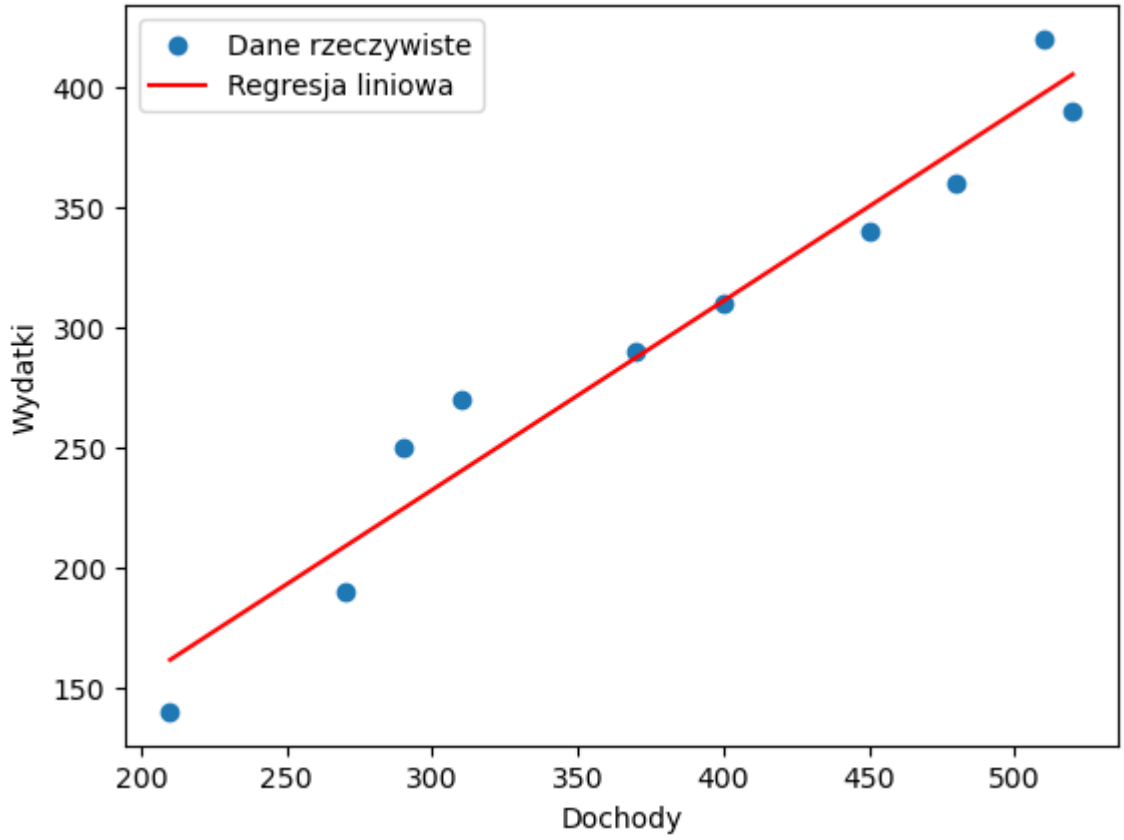
Wydatki = [140, 190, 250, 270, 290, 310, 340, 360, 420, 390]

Utwórz regresję liniową zależności wydatków od dochodów.

Jaki jest błąd uzyskanej prostej względem danych? Czy jest możliwość uzyskania lepszego wyniku?

W celu wyznaczenia współczynników wykorzystaj niniejszą informację
https://en.wikipedia.org/wiki/Simple_linear_regression
(https://en.wikipedia.org/wiki/Simple_linear_regression)

In [47]:
```
dochody = np.array([210, 270, 290, 310, 370, 400, 450, 480, 510, 520])
wydatki = np.array([140, 190, 250, 270, 290, 310, 340, 360, 420, 390])

# Przekształcenie wektorów do postaci kolumnowej
dochody = dochody.reshape(-1, 1)
wydatki = wydatki.reshape(-1, 1)

# Utworzenie modelu regresji liniowej
model = LinearRegression()

# Dopasowanie modelu do danych
model.fit(dochody, wydatki)

# Przewidywanie wydatków na podstawie dochodów
wydatki_pred = model.predict(dochody)

# Wizualizacja
plt.scatter(dochody, wydatki, label='Dane rzeczywiste')
plt.plot(dochody, wydatki_pred, color='red', label='Regresja liniowa')
plt.xlabel('Dochody')
plt.ylabel('Wydatki')
plt.legend()
plt.show()

# Obliczenie błędu
mse = mean_squared_error(wydatki, wydatki_pred)
print("Błąd średniokwadratowy (MSE):", mse)
```




```
Błąd średniokwadratowy (MSE): 340.51374067428463
```

Odpowiedź: W celu uzyskania lepszego wyniku, można rozważyć użycie bardziej złożonych modeli regresji, na przykład wielomianowej, lub skorzystać z innych technik dostępnych w dziedzinie uczenia maszynowego. Jednak w tym konkretnym przypadku regresja liniowa

może być wystarczająco adekwatna, biorąc pod uwagę naturę zależności między dochodami a wydatkami.

Zadanie 3

1. Przy użyciu funkcji random.normal (https://numpy.org/doc/stable/reference/random/generated/numpy.random.normal.html) wygeneruj trzy wektory x_1, x_2 oraz ϵ o wymiarze (100,1) o wartości średniej równej 0 oraz odchyleniu standardowemu równego 1.
2. Przy pomocy funkcji concatenate (https://numpy.org/doc/stable/reference/generated/numpy.concatenate.html) połącz wektor samych jedynek o wymiarze (100,1) z:

- x_1,
- x_2,
- x_1 i x_2
- x_1, x_2 i $x_1 * x_2$

z parametrem *axis = 1*. To zadanie jest przygotowaniem struktury danych potrzebnych przy regresji liniowej w dalszej części zadania.

3. Na podstawie wektoró z punktu 1 wylicz wartości wektora y danego wzorem:

$$y = 2 + x_1 - 0.5x_2 + \epsilon$$

Dla lepszego zobrazowania problemu należy spojrzeć na niego w sposób następujący:

- y - proces który chcemy w dalszej części zadania zamodelować, a którego dokładnego opisu nie znamy
- x_1, x_2 - zmienne niezależne które jesteśmy w stanie mierzyć i wiemy że wpływają na proces
- ϵ - zakłócenie procesu

4. W zależności od możliwości pomiaru zmiennych niezależnych można podjąć próbę zamodelowania procesu, na potrzeby ćwiczenia wykorzystana zostanie regresja liniowa (https://pl.wikipedia.org/wiki/Regresja_liniowa). Zakładając że dostępne dane to wektor wartości y i odpowiednio wektor x_1 lub x_2 do obliczenia jej współczynników można użyć rozkładu QR. Aby to zrobić należy przyjąć hipotetyczny model procesu (wzory 1-4) i potraktować "zmierzone" i podstawione do wzoru dane jako układ równań

Korzystając z macierzy z punktu 3 oblicz współczynniki regresji liniowej z wykorzystaniem rozkładu QR, dla modeli procesu opisanych w następujacy sposób:

$$\hat{y} \sim a + z_1 + z_2 + \ldots + z_n$$

Gdzie:

- $\hat{y}$ - przybliżenie modelowanego procesu
- a - wyraz wolny
- $z_1 + z_2 + \ldots + z_n$ - zmienne niezależne, których ilość dobiera się na podstawie dostępnych danych, tak aby otrzymać jak najlepszy model

Celem regresji jest dobranie takich współczynników zmiennych niezależnych i wyrazu wolnego, aby zaproponowany model jak najbliżej odwzorowywał pierwotny procecs.

Modele do przeprowadzenia eksperymentów:

1. $\hat{y} \sim a + x_1$

2. $\hat{y} \sim a + x_2$
3. $\hat{y} \sim a + x_1 + x_2$
4. $\hat{y} \sim a + x_1 + x_2 + x_1 * x_2$
5. Przeanalizuj (znanymi metrykami) i przedstaw otrzymane wyniki na odpowiednich subplotach.

In [48]:
```
#Zadanie 3.1

# Wygenerowanie wektorów x1, x2, epsilon
x1 = np.random.normal(loc=0, scale=1, size=(100, 1))
x2 = np.random.normal(loc=0, scale=1, size=(100, 1))
epsilon = np.random.normal(loc=0, scale=1, size=(100, 1))
```

In [49]:
```
#Zadanie 3.2
ones = np.ones((100,1))
x1_times_x2 = x1 * x2
result_a = np.concatenate((ones, x1, x2, x1, x2), axis=1)
result_b = np.concatenate((ones, x1, x2, x1 * x2), axis=1)
```

In [50]:
```
#Zadanie 3.3
y = 2 + x1 - 0.5*x2 + epsilon
```

In [51]:
```
#Zadanie 3.4
from sklearn.linear_model import LinearRegression
# Modele do przeprowadzenia eksperymentów
models = [
    ("ŷ ~ a + x1", result_a[:, [0, 1]]),
    ("ŷ ~ a + x2", result_a[:, [0, 2]]),
    ("ŷ ~ a + x1 + x2", result_a[:, [0, 1, 2]]),
    ("ŷ ~ a + x1 + x2 + x1 * x2", result_a)
]


for i, (model_desc, result) in enumerate(models):
    reg = LinearRegression().fit(result, y)
    r_squared = reg.score(result, y)
    print(f"Model {i + 1}: {model_desc}")
    print(f"Współczynnik determinacji (R-kwadrat): {r_squared}")
```

```
Model 1: ŷ ~ a + x1
Współczynnik determinacji (R-kwadrat): 0.5475865628380758
Model 2: ŷ ~ a + x2
Współczynnik determinacji (R-kwadrat): 0.18739335103404187
Model 3: ŷ ~ a + x1 + x2
Współczynnik determinacji (R-kwadrat): 0.7120117528909633
Model 4: ŷ ~ a + x1 + x2 + x1 * x2
Współczynnik determinacji (R-kwadrat): 0.7120117528909633
```

Materiały uzupełniające:

- Scipy Lecture Notes (http://www.scipy-lectures.org/index.html)
- NumPy for Matlab users (https://docs.scipy.org/doc/numpy/user/numpy-for-matlab-users.html#numpy-for-matlab-users)
- Python Tutorial - W3Schools (https://www.w3schools.com/python/default.asp)
- NumPy (https://www.numpy.org)

- Matplotlib (https://matplotlib.org/)
- Anaconda (https://www.anaconda.com/)
- Learn Python for Data Science (https://www.datacamp.com/learn-python-with-anaconda?utm_source=Anaconda_download&utm_campaign=datacamp_training&utm_medium=bar
- Learn Python (https://www.learnpython.org/)
- Wujek Google (https://google.pl) i Ciocia Wikipedia (https://pl.wikipedia.org/wiki/Wikipedia:Strona_g%C5%82%C3%B3wna)