Iteracyjne równania liniowe i macierze rzadkie

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 w formie PDF. Ponisze funckje będą niezbędne do wykonania laboratorium:

In [3]:
```
import scipy as sp
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from pympler import asizeof
from scipy.sparse import diags
from scipy import linalg
from  datetime import datetime
import pickle

from typing import Union, List, Tuple

def residual_norm(A, x, b):
    """Oblicza normę residuum dla równania Ax = b."""
    return np.linalg.norm(b - A.dot(x))

def is_diagonally_dominant(A):
    """Sprawdza, czy macierz A jest diagonalnie zdominowana."""
    diagonal = np.abs(A.diagonal())
    row_sums = np.sum(np.abs(A), axis=1) - diagonal
    return np.all(diagonal >= row_sums)
```

Cel zajęć: Celem zajęć jest zapoznanie się z macierzami rzadkimi, oraz iteracyjnymi metodami rozwiązywania układów równań liniowych w postaci macierzowej. Czyli dana jest macierz rzadka (https://pl.wikipedia.org/wiki/Macierz_rzadka) $\mathbf{A}$ o wymiarach $(m \times m)$ 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.

Do oceny jakości rozwiązania będziemy wykorzystywać residuum (ang. *residual*)

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

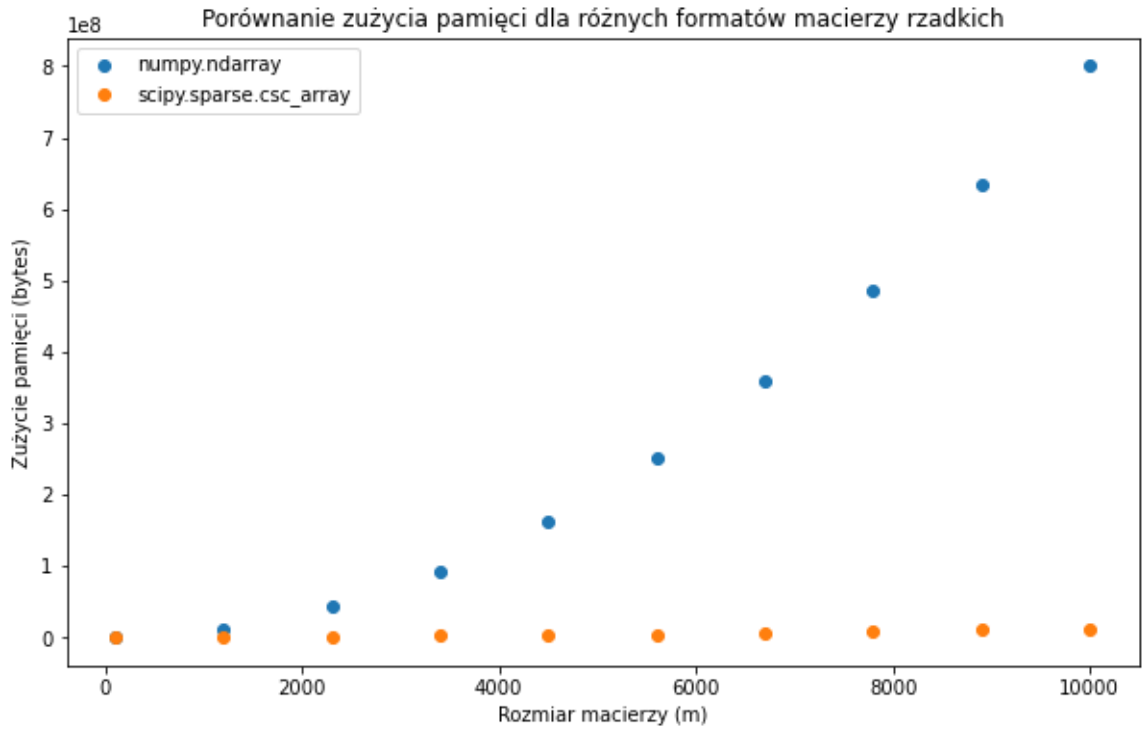
Zadanie 1

W macierzy rzadkiej większość elementów wynosi 0, w związku z tym przechowywanie wprost takiej macierzy w pamięci jest niepraktyczne. Do przechowywania i wykonywania operacji na macierzach rzadkich służy moduł scipy.sparse (https://docs.scipy.org/doc/scipy/reference/sparse.html).

1. Zapoznaj się z różnymi formatami przechowywania macierzy rzadkich. Na protrzeby ninejszego laboratorium wykorzystany zostanie format Compressed Sparse Column (https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.csc_array.html#scipy.sp
2. Przedstaw na jednym wykresie (scatter plot) dla 10 równo rozmieszczonych wartości m z zakresu od 100 do 10000 ile pamięci zajmuje ta sama macierz o wymiarach $(m \times m)$ w zapisana w formacie $numpy.ndarray$ oraz w formacie $scipy.sparse.csc\_array$

Do obliczenia rozmiaru zmiennych monżna wykorzystać funkcję asizeof.asizeof() (https://pympler.readthedocs.io/en/latest/library/asizeof.html#)

In [2]:
```
def generate_sparse_matrix(m):
    """Generuje macierz rzadką w formacie csc."""
    return sp.sparse.random(m, m, density=0.01, format='csc')

def compare_memory_usage(m_values):
    """Porównuje zużycie pamięci dla macierzy w różnych formatach."""
    memory_usage_np = []
    memory_usage_csc = []

    for m in m_values:
        # Generowanie macierzy w formacie scipy.sparse.csc_array
        A_sparse = generate_sparse_matrix(m)
        # Uzyskanie reprezentacji tej samej macierzy w formacie numpy.ndarra
        A_dense = A_sparse.toarray()

        # Pomiar zużycia pamięci
        memory_usage_np.append(asizeof.asizeof(A_dense))
        memory_usage_csc.append(asizeof.asizeof(A_sparse))

    # Wykres porównujący zużycie pamięci
    plt.figure(figsize=(10, 6))
    plt.scatter(m_values, memory_usage_np, label='numpy.ndarray')
    plt.scatter(m_values, memory_usage_csc, label='scipy.sparse.csc_array')
    plt.xlabel('Rozmiar macierzy (m)')
    plt.ylabel('Zużycie pamięci (bytes)')
    plt.legend()
    plt.title('Porównanie zużycia pamięci dla różnych formatów macierzy rzad
    plt.show()

# Przykładowe wartości m
m_values = np.linspace(100, 10000, 10, dtype=int)

# Porównanie zużycia pamięci
compare_memory_usage(m_values)
```



3. Wygląd wykresów wynika z faktu, że macierze rzadkie przechowują jedynie niezerowe elementy, co znacznie redukuje zużycie pamięci w porównaniu do macierzy gęstych, które przechowują wszystkie elementy. W miarę wzrostu rozmiaru macierzy, oszczędności pamięci dla macierzy rzadkich stają się bardziej zauważalne, co widać na wykresie.

In [6]:
```
# Generowanie macierzy w formacie scipy.sparse.csc_array
m = 100
A = sp.sparse.random(m, m, density=0.01, format='csc')

# Uzyskanie reprezentacji tej samej macierzy w formacie numpy.ndarray
np_A = A.toarray()
```

Zadanie 2

Moduł [scipy.sparse (https://docs.scipy.org/doc/scipy/reference/sparse.html)](https://docs.scipy.org/doc/scipy/reference/sparse.html) implementuje operacje na macierzach rzadkich w dowolnym formacie.

1. Wygeneruj dwie macierze $\mathbf{A}$ i $\mathbf{B}$ o wymiarach 1000×1000
2. Porównaj przy pomocy funkcji $\%timeit$ czas potrzebny na wykonanie mnożenia macierzowe $\mathbf{A} * \mathbf{B}$ zapisanych w formacie $numpy.ndarray$ oraz $scipy.sparse.csc\_array$
3. Z czego wynika różnica?

In [4]:
```
# Generowanie dwóch macierzy rzadkich
A_sparse = sp.sparse.random(1000, 1000, density=0.01, format='csc')
B_sparse = sp.sparse.random(1000, 1000, density=0.01, format='csc')

# Uzyskanie reprezentacji tych samych macierzy w formacie numpy.ndarray
A_dense = A_sparse.toarray()
B_dense = B_sparse.toarray()

# Porównanie czasu mnożenia macierzowego dla numpy.ndarray
time_np = %timeit -o np.dot(A_dense, B_dense)

# Porównanie czasu mnożenia macierzowego dla scipy.sparse.csc_array
time_sparse = %timeit -o A_sparse.dot(B_sparse)

# Wyświetlenie wyników
print(f"Czas mnożenia macierzowego dla numpy.ndarray: {time_np.best} seconds
print(f"Czas mnożenia macierzowego dla scipy.sparse.csc_array: {time_sparse.
```

```
11 ms ± 196 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
549 µs ± 1.24 µs per loop (mean ± std. dev. of 7 runs, 1,000 loops each)
Czas mnożenia macierzowego dla numpy.ndarray: 0.010690258999966317 seconds
Czas mnożenia macierzowego dla scipy.sparse.csc_array: 0.00054780870000104
18 seconds
```

Zadanie 3

Aby metody iteracyjne znalazły rozwiązanie układu należy zadbać o to by macierz $\mathbf{A}$ w układzie $\mathbf{Ax} = \mathbf{b}$ była [diagonalnie zdominowana (https://en.wikipedia.org/wiki/Diagonally_dominant_matrix)](https://en.wikipedia.org/wiki/Diagonally_dominant_matrix). Przekształcanie dowolnej

macierzy do tej postaci jest skomplikowanym zagadnieniem, w związku z czym na potrzeby tego zadania należy sprawdzić czy wygenerowaliśmy macierz o odpowiedniej własności, aby pominąć krok przekształcania.

1. Uzupełnij funkcję $is\_diagonaly\_dominant()$ w pliku $main.py$ zgodnie z opisem (podpowiedź: korzystanie z pętli for nie jest dobrym rozwiązaniem, ponieważ często mamy do czynienia z bardzo dużymi wymiarami, przy których pętle zajmują bardzo duożo czasu. Skoprzystaj z funkcji np.sum() (https://numpy.org/doc/stable/reference/generated/numpy.sum.html), oraz np.diagonal() (https://numpy.org/doc/stable/reference/generated/numpy.diagonal.html))
2. Wygeneruj przy pomocy funkcji $generate\_matrix()$ macierz $\mathbf{A}$ o wymiarach 1000×1000. Przy pomocy funkcji z poprzedniego punktu zweryfikuj, czy wygenerowana macierz jest diagonalnie zdominowana. Wygeneruj również wektor $\mathbf{b}$ 1000×1

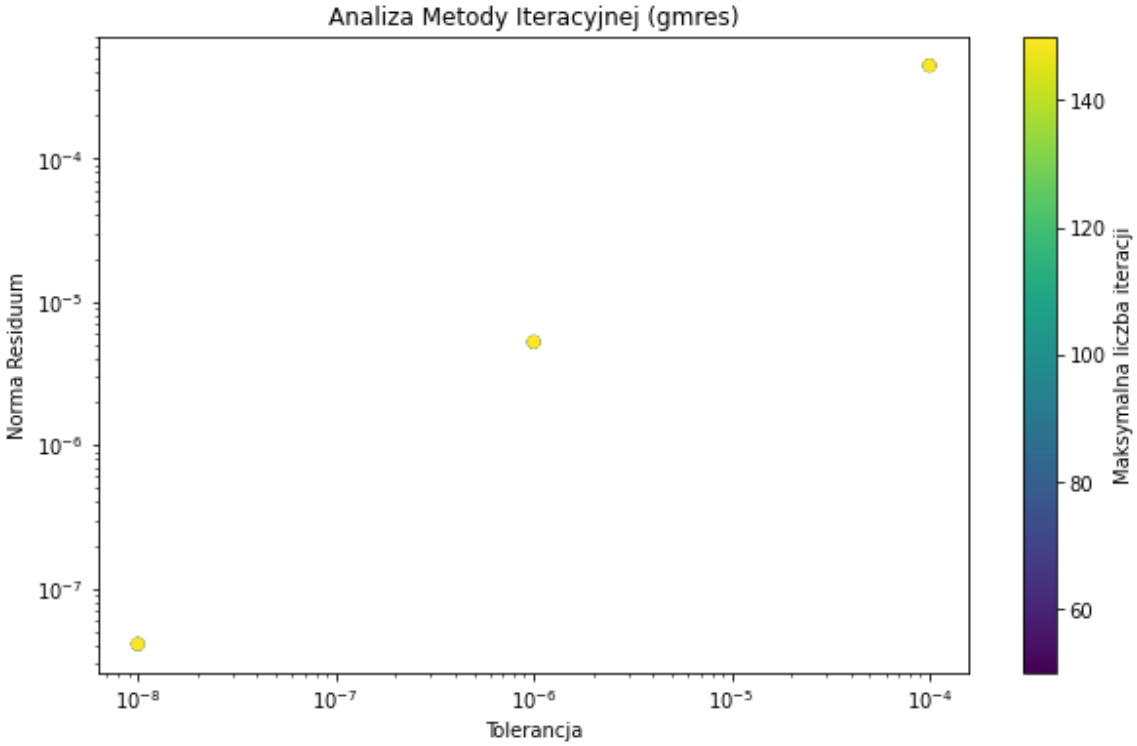
Istnieje wiele metod iteracyjnego rozwiązywania równań, nie różnią się one znacząco od siebie w kwesti wywołania metody, dlatego w dalszej części zadania należy wybrać jedną z metod: GMRES (https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.linalg.gmres.html) lub Conjugate Gradient (https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.linalg.cg.html) i dla niej przeprowadzić dalszą analizę. W przypadku wyboru drugiej opcji należy zadbać o to aby macierz była dodatnio określona, poprzez podanie argumentu $is\_simetrical = True$

3. Korzystając z funkcji $residual\_norm()$ z pliku $main.py$ zbadaj wpływ argumentów tol i $maxiter$ na działanie wybranej metody. Sporządź wykresy przedstawiające wartość

In [20]:
```
def generate_matrix(m=1000, is_simetrical=False):
    if is_simetrical:
        num_u = np.random.randint(2, m//2)
        num_l = np.random.randint(2, m//2)
        diagonal_values = np.random.rand(num_u)
        matrix = diags(diagonal_values, list(range(0, num_u)), shape=(m, m))
        matrix = matrix + matrix.T
        a = matrix.toarray()
        np.fill_diagonal(a, np.sum(np.abs(a), axis=1) - np.abs(np.diagonal(a
        return sp.sparse.csc_array(a)
    else:
        num_u = np.random.randint(2, m//2)
        num_l = np.random.randint(2, m//2)
        diagonal_values = np.random.rand(num_u + num_l)
        matrix = diags(diagonal_values, list(range(-num_l, num_u)), shape=(m
        a = matrix.toarray()
        np.fill_diagonal(a, np.sum(np.abs(a), axis=1) - np.abs(np.diagonal(a
        return sp.sparse.csc_array(a)
```

In [6]:
```
def analyze_iterative_method(A, b, method='gmres', tol_values=[1e-8, 1e-6, 1
    """Analizuje wybraną metodę iteracyjną dla różnych wartości tol i maxite
    results = []

    for tol in tol_values:
        for maxiter in maxiter_values:
            if method == 'gmres':
                x, _ = gmres(A, b, tol=tol, maxiter=maxiter)
            else:
                # Zakładając metodę 'cg'
                x, _ = sp.sparse.linalg.cg(A, b, tol=tol, maxiter=maxiter)

            norm_residual = residual_norm(A, x, b)
            results.append((tol, maxiter, norm_residual))

    return results

# Generowanie macierzy A i wektora b
A = generate_matrix(1000, is_symmetrical=True)
b = np.random.rand(1000)

# Sprawdzenie, czy A jest diagonalnie zdominowana
is_diagonally_dominant_A = is_diagonally_dominant(A)
print(f"Czy macierz A jest diagonalnie zdominowana? {is_diagonally_dominant_

# Wybór metody iteracyjnej (np. 'gmres' lub 'cg')
method = 'gmres'

# Analiza metody iteracyjnej dla różnych wartości tol i maxiter
results = analyze_iterative_method(A, b, method=method)

# Wykres przedstawiający zależność normy residuum od wartości tol dla różnyc
tol_values, maxiter_values, norm_residual_values = zip(*results)

plt.figure(figsize=(10, 6))
plt.scatter(tol_values, norm_residual_values, c=maxiter_values, cmap='viridi
plt.colorbar(label='Maksymalna liczba iteracji')
plt.xscale('log')
plt.yscale('log')
plt.xlabel('Tolerancja')
plt.ylabel('Norma Residuum')
plt.title(f'Analiza Metody Iteracyjnej ({method})')
plt.show()
```

```
Czy macierz A jest diagonalnie zdominowana? True
```


Analiza Metody Iteracyjnej (gmres)
Norma Residuum
10^{-4}
10^{-5}
10^{-6}
10^{-7}
10^{-8}
10^{-7}
10^{-6}
10^{-5}
10^{-4}
Tolerancja
140
120
100
80
60
Maksymalna liczba iteracji