Rachunek Macierzowy i Statystyka Wielowymiarowa
Sprawozdanie 3
Normy Macierzowe i SVD

Adam Staniszewski
Przemysław Węglik

13 kwietnia 2024

Spis treści

1 Norma M_1

1.1 Obliczanie normy

```
matrix_1_norm = lambda x: np.max(np.sum(x, axis=0))
```

Każdą z norm obliczamy dla macierzy

$$\begin{matrix} 4 & 9 & 2 \\ 3 & 5 & 7 \\ 8 & 1 & 6 \end{matrix}$$

zwaną dalej macierzą M.

M_1 dla macierzy M wynosi 15.

1.2 Obliczanie współczynnika uwarunkowania

```
cond_matrix_1_norm = lambda x: matrix_1_norm(x) *
                               matrix_1_norm(np.linalg.inv(x))
```

Współczynnik uwarunkowania macierzy M wynosi w tym przypadku 1.0000000000000004.

2 Norma M_2

2.1 Obliczanie normy

```
matrix_2_norm = lambda x: np.linalg.svd(M, compute_uv=False)[0]
```

M_2 dla macierzy M wynosi 15.000000000000002.

2.2 Obliczanie współczynnika uwarunkowania

```
def cond_matrix_2_norm(matrix):
    singular_values = np.linalg.svd(matrix, compute_uv=False)
    return singular_values[0] / singular_values[matrix.shape[0] - 1]
```

Współczynnik uwarunkowania macierzy M wynosi w tym przypadku 4.330127018922198.

3 Norma M_p

Obliczenie normy M_p jest problemem NP-Trudnym. Odsyłamy do publikacji Approximating Matrix p-norms.

4 Norma M_∞

4.1 Obliczanie normy

```
matrix_inf_norm = lambda x: np.max(np.sum(x, axis=1))
```

M_∞ dla macierzy M wynosi 15.

4.2 Obliczanie współczynnika uwarunkowania

```python
cond_matrix_inf_norm = lambda x: matrix_inf_norm(x) *
                                matrix_inf_norm(np.linalg.inv(x))
```

Współczynnik uwarunkowania macierzy M wynosi w tym przypadku 1.0000000000000002.

5 SVD

Używamy algorytmy **Power Iteration**.

```python
def power_iteration(A, num_iterations: int):
    # Ideally choose a random vector
    # To decrease the chance that our vector
    # Is orthogonal to the eigenvector
    b_k = np.random.rand(A.shape[1])

    for _ in range(num_iterations):
        # calculate the matrix-by-vector product Ab
        b_k1 = np.dot(A, b_k)

        # calculate the norm
        b_k1_norm = np.linalg.norm(b_k1)

        # re normalize the vector
        b_k = b_k1 / b_k1_norm

    # Rayleigh quotient
    return b_k, b_k.T @ A @ b_k / (b_k.T @ b_k)
```