Rachunek Macierzowy
Program 3

Paulina Dziwak, Karol Markowicz

Norma 1 (MatrixNorm1):

Norma 1 macierzy jest określana jako maksymalna suma wartości absolutnych kolumn macierzy. W tej funkcji najpierw inicjujemy zmienną maxColSum na wartość 0, która będzie przechowywać maksymalną sumę kolumn. Następnie iterujemy po kolumnach macierzy, obliczając sumę wartości absolutnych każdej kolumny. Jeśli obliczona suma jest większa niż aktualna wartość maxColSum, aktualizujemy maxColSum. Po zakończeniu iteracji zwracamy maxColSum jako wynik normy 1 macierzy.

```
public static double matrixNorm1(double[][] matrix){
    double maxColSum = 0;
    for (int c = 0; c < matrix[0].length; c++){
        double colSum = 0;
        for (int r = 0; r < matrix.length; r++){
            colSum += Math.abs(matrix[r][c]);
        }
        if (colSum > maxColSum){
            maxColSum = colSum;
        }
    }
    return maxColSum;
}
```

Norma 2 (MatrixNorm2):

Norma 2 macierzy została zaimplementowana jako pierwiastek kwadratowy z największej wartości własnej macierzy M^T * M, gdzie M^T to transpozycja macierzy M. W tej funkcji najpierw obliczamy macierz M^T * M, a następnie obliczamy jej wartości własne. Zwracamy pierwiastek kwadratowy z największej wartości własnej jako wynik normy 2 macierzy.

```
public static double matrixNorm2(double[][] matrix) {
    // Obliczenie macierzy M^T * M
    int cols = matrix[0].length;
    double[][] mtm = new double[cols][cols];

    for (int i = 0; i < cols; i++) {
        for (int j = 0; j < cols; j++) {
```

```
            for (int r = 0; r < matrix.length; r++) {
                mtm[i][j] += matrix[r][i] * matrix[r][j];
            }
        }
    }
    // Obliczenie wartości własnych macierzy M^T * M
    RealMatrix mtmMatrix = new Array2DRowRealMatrix(mtm);
    EigenDecomposition eig = new EigenDecomposition(mtmMatrix);
    double maxEigenvalue = eig.getRealEigenvalue(0);
    // Pierwsza największa wartość własna

    // Zwrócenie pierwiastka kwadratowego z największej wartości własnej
    return Math.sqrt(maxEigenvalue);
}
```

Norma 3 (MatrixNorm3):

Jako przykład innej normy należącej do norm p wybraliśmy normę 3 macierzy. Jest ona zaimplementowana jako maksymalna z norm 3, wektorów kolumn macierzy. W tej funkcji najpierw inicjujemy zmienną maxColumnNorm na wartość 0, która będzie przechowywać maksymalną normę 3 kolumn. Następnie iterujemy po kolumnach macierzy, obliczając normę 3 każdej kolumny. Norma 3 kolumny jest obliczana jako suma sześcianów wartości bezwzględnych elementów tej kolumny podniesionych do potęgi 1/3. Jeśli obliczona norma jest większa niż aktualna wartość maxColumnNorm, aktualizujemy maxColumnNorm. Po zakończeniu iteracji zwracamy maxColumnNorm jako wynik normy 3 macierzy.

```
public static double matrixNorm3(double[][] matrix) {
    double maxColumnNorm = 0;
    for (int c = 0; c < matrix[0].length; c++){
        double columnNorm = 0;
        for (double[] row : matrix) {
            columnNorm += Math.pow(Math.abs(row[c]), 3);
        }
        maxColumnNorm = Math.max(maxColumnNorm, Math.pow(columnNorm, 1.0 /
3.0));
    }
    return maxColumnNorm;
}
```

Norma ∞ (MatrixNormInfinity):

Norma ∞ macierzy została zaimplementowana jako maksymalna suma wartości absolutnych wierszy macierzy. W tej funkcji najpierw inicjujemy zmienną maxRowSum na wartość 0, która będzie przechowywać maksymalną sumę wartości absolutnych wierszy. Następnie iterujemy po wierszach macierzy, obliczając sumę wartości absolutnych każdego wiersza. Jeśli obliczona suma jest większa niż aktualna wartość maxRowSum, aktualizujemy maxRowSum. Po zakończeniu iteracji zwracamy maxRowSum jako wynik normy ∞ macierzy.

```java
public static double matrixNormInfinity(double[][] matrix){
    double maxRowSum = 0;
    for (int r = 0; r < matrix.length; r++){
        double rowSum = 0;
        for (int c = 0; c < matrix[0].length; c++){
            rowSum += Math.abs(matrix[r][c]);
        }
        if (rowSum > maxRowSum){
            maxRowSum = rowSum;
        }
    }
    return maxRowSum;
}
```

Współczynniki uwarunkowania macierzy (conditionNumber):

Ta funkcja oblicza współczynnik uwarunkowania macierzy na podstawie danej macierzy i funkcji normy. Funkcja ta umożliwia obliczenie uwarunkowania macierzy dla różnych norm i ocenienie jak bardzo operacje numeryczne na macierzach mogą być podatne na błędy w zależności od normy użytej do analizy.

Algorytm polega na utorzeniu macierzy odwrotnej inverseMatrix dla danej macierzy matrix za pomocą metody MatrixUtils.inverse() z biblioteki Apache Commons Math.

Następnie funkcja oblicza normę macierzy matrix za pomocą funkcji normy przekazanej jako parametr normFunction i zapisuje ją jako norm. Następnie oblicza normę odwrotnej macierzy i zapisuje ją jako inverseNorm.

Na koniec funkcja zwraca iloczyn normy macierzy matrix i normy odwrotnej macierzy jako współczynnik uwarunkowania.

```java
public static double conditionNumber(double[][] matrix,
ToDoubleFunction<double[][]> normFunction) {
    double[][] inverse;

    RealMatrix matrix1 = new Array2DRowRealMatrix(matrix);
    RealMatrix inverseMatrix = MatrixUtils.inverse(matrix1);
    inverse = inverseMatrix.getData();


    double norm = normFunction.applyAsDouble(matrix);
    double inverseNorm = normFunction.applyAsDouble(inverse);

    return norm * inverseNorm;
}
```

Wyniki działania opisanych funkcji dla macierzy Tybetańskiej

```
Norma macierzowa ||M||1 wynosi: 15.0
Norma macierzowa ||M||2 wynosi: 14.999999999999996
Norma macierzowa ||M||3 wynosi: 9.49121995802933
Norma macierzowa ||M||Inf wynosi: 15.0
Współczynnik uwarunkowania macierzowego ||M||1 wynosi: 5.333333333333333
Współczynnik uwarunkowania macierzowego ||M||2 wynosi: 4.330127018922192
Współczynnik uwarunkowania macierzowego ||M||3 wynosi: 2.028611376513223
Współczynnik uwarunkowania macierzowego ||M||inf wynosi: 5.333333333333333
```

Rozkład według wartości osobliwych (SVD)

Rozkład według wartości osobliwych jest jednym z najważniejszych narzędzi w analizie danych i przetwarzaniu sygnałów. Stosowany jest w analizie składowych głównych (PCA), filtracji i kompresji obrazów, ale też może być pomocny w rozwiązywaniu układów liniowych.

Na implementację SVD składają się następujące kroki:

1. Wyliczenie transpozycji macierzy A na której dokonujemy rozkładu.
2. Wyliczenie iloczynów AA^T oraz A^TA.
3. Wyliczenie wartości własnych iloczynu AA^T i posortowanie ich malejąco:

```
RealMatrix matrix = new Array2DRowRealMatrix(product1);
EigenDecomposition eigenDecomposition =  new EigenDecomposition(matrix);
double[] eigenvalues = eigenDecomposition.getRealEigenvalues();
Double[] eigenvaluesSorted = sortArray(eigenvalues);
```

4. Wstawienie pierwiastków wartości własnych jako elementy głównej przekątnej macierzy sigma:

```
private double[][] getSigmaMatrix(Double[] eigenValues){
    double[][] result = new double[rows][columns];
    for(int i=0; i<rows; i++){
        for(int j=0; j<columns; j++){
            if(i == j)
                result[i][j] = Math.sqrt(eigenValues[i]);
        }
    }
    return result;
}
```

5. Wyliczenie macierzy U oraz V korzystając z poniższej funkcji poprzez podanie odpowiedniego iloczynu macierzy jako jej parametr:

```
private double[][] getMatrixFromEigenvectors(double[][] productMatrix){
    RealMatrix matrix = new Array2DRowRealMatrix(productMatrix);
    EigenDecomposition eigenDecomposition =  new
EigenDecomposition(matrix);
    RealMatrix eigenvectors = eigenDecomposition.getV();
    return transpose(eigenvectors.getData());
}
```

W celu obliczenia wartości i wektorów własnych macierzy, wykorzystaliśmy bibliotekę Apache Commons Math.

Wyniki rozkładu SVD dla macierzy Tybetańskiej prezentują się następująco:

```
U matrix:
0.5773502691896255 0.577350269189626 0.577350269189626
0.7071067811865475 1.813870234568249E-16 -0.7071067811865476
-0.4082482904638631 0.816496580927726 -0.40824829046386274

S matrix:
15.000000000000002 0.0 0.0
0.0 6.92820323027551 0.0
0.0 0.0 3.4641016151377526

V matrix:
0.5773502691896257 0.5773502691896256 0.5773502691896258
0.40824829046386285 -0.8164965809277258 0.4082482904638629
-0.7071067811865476 2.083104027468 59E-17 0.7071067811865474
```

Porównując nasze wyniki z rozkładem dokonanym za pomocą macierzowego kalkulatora, zauważamy, że rezultaty są generalnie niemal identyczne, jednakże występują pewne różnice. Wartość znajdująca się pod indeksem [1][1] w macierzy U oraz wartość pod indeksem [1][2] w macierzy V przyjmują kolejno wartości: 1.81E-16 oraz 2.083E-17. Prawdopodobnie wynika to z błędów numerycznych związanych z reprezentacją liczb zmiennoprzecinkowych oraz ograniczoną precyzją reprezentacji liczb w pamięci komputera. Podobna sytuacja miała miejsce również w poprzednim programie.

Inną różnicą jest obecność wierszy, w których wszystkie wartości mają przeciwny znak w porównaniu do innych. Prawdopodobną przyczyną tego zjawiska jest sparametryzowana natura obliczania wektorów własnych macierzy. Dla jednej wartości własnej może istnieć nieskończenie wiele odpowiadających jej wektorów własnych. W wyniku tego, zastosowanie nieco odmiennych kroków podczas obliczeń może prowadzić do różnic w rezultatach. Na

przykład, różne podejścia mogą zwrócić wektor własny z przeciwnym znakiem, co jednak nie jest błędem, lecz jedynie odmienną parametryzacją.

Drugą z różnic jest fakt, że niektóre wiersze mają przeciwny znak przed wszystkimi swoimi wartościami. Prawdopodobną przyczyną tego zjawiska jest to, że w przypadku wyliczania wektorów własnych macierzy otrzymujemy jego sparametryzowaną wersję. Dla jednej wartości własnej może być nieskończenie wiele wektorów własnych i w wyniku nieco odmiennych kroków.