Szymon Winiecki Poznań, 18.06.2021

# Rozwiązywanie układu równań liniowych metodą Gaussa-Jordana z pełnym wyborem elementu podstawowego

## 1. Zastosowanie

Metoda *GaussJordan* w klasie *LinearEquationsSystem* rozwiązuje układ równań liniowych ***Ax = b*** metodą eliminacji Gaussa-Jordana z pełnym wyborem elementu podstawowego.

## 2. Matematyczny opis metody

Metoda eliminacji Gaussa-Jordana rozwiązuje układ *n* równań liniowych:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\ \cdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n \end{cases}$$

Przedstawiony w postaci macierzy uzupełnionej:

$$[A|b] = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\ & & \cdots & & \\ a_{n1} & a_{n2} & \cdots & a_{nn} & b_n \end{bmatrix}$$

Stosując elementarne operacje przekształca macierz A do postaci macierzy diagonalnej. W kroku *i* (*i* = 1, 2, …, *n*) szukamy elementu o największej wartości bezwzględnej w zakresie $a_{ii}$ do $a_{nn}$, następnie zamieniamy miejscami wiersz *i* z wierszem znalezionego elementu i kolumnę *i* z kolumną znalezionego elementu (należy przy tym pamiętać, że zamiana kolumn wiąże się za zmianą kolejności zmiennych w macierzy wynikowej). Następnie dzielimy wiersz *i* przez element $a_{ii,}$ a następnie od j-tego wiersza (*j* = 1, 2, …, *n* oraz *j* ≠ *i*) odejmujemy wiersz *i* pomnożony przez $a_{ji}$. Po n krokach algorytmu otrzymujemy macierz postaci:

$$\begin{bmatrix} x_1 & 0 & 0 & 0 & b_1 \\ 0 & x_2 & 0 & 0 & b_2 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & x_n & b_n \end{bmatrix}$$

Czyli gotowe rozwiązanie, gdzie:

$$x_i = b_i$$

**3. Wywołanie metody**

Przykład wywołania:

```
std::string userInput;
//wczytanie danych od użytkownika do zmiennej userInput
Matrix<Interval<long double>> mat(userInput, false);
LinearEquationsSystem<Interval<long double>> les(mat);
std::vector<Interval<long double>> result = les.GaussJordan();
//zmienna result przechowuje rozwiązanie układu równań
```

Wyjaśnienie:

Metoda *GaussJordan* może zostać wywołana na obiekcie klasy *LinearEquationsSystem*. Obiekt klasy *LinearEquationsSystem* jako parametr konstruktora przyjmuje obiekt klasy Matrix, reprezentujący układ równań **Ax = b** jako macierz uzupełnioną **[A|b]**.
Obiekt klasy Matrix jako parametr konstruktora przyjmuje:

a) Zmienną typu *string* z reprezentacją macierzy, w której wiersze oddzielone są znakiem nowej linii (*'\n'*) a kolumny białymi znakami (*spacją* lub *'\t'*) oraz zmienną typu *bool* gdzie wartość *true* oznacza że dane podane są w postaci przedziałów (lewy koniec przedziału, biały znak, prawy koniec przedziału) a wartość false oznacza ze dane podane są jako liczby (liczba staje się zarówno lewym jak i prawym końcem przedziału).
b) Lub zmienną typu *vector<vector< Interval<long double> >* która jest wektorem wierszy macierzy reprezentowanych jako wektory elementów wiersza.

**4. Wynik**

Metoda *GaussJordan* zwraca wektor ( *vector< Interval<long double> >* ) który zawiera rozwiązanie układu równań. Pod indeksem *i* tego wektora znajduje się wartość $x_i$ (numerowane od 0), ponadto wektor rozwiązania zostaje zapisany w polu *result* obiektu.

**5. Status wykonania i sygnalizacja błędów**

Pole *status* w obiektach klasy *LinearEquationsSystem* przechowuje aktualny status układu równań. Może przyjąć następujące wartości:

*empty* - jeżeli do układu nie jest przypisana żadna macierz
*unsolved* – jeżeli układ nie został jeszcze rozwiązany
*solved* – jeżeli układ został pomyślnie rozwiązany
*wrongEquationsNumber* – jeżeli macierz jest innych wymiarów niż *n* na *n+1,* gdzie *n* to liczba równań
*error* – jeżeli układ jest sprzeczny lub nieoznaczony

Przy konstruowaniu obiektu klasy Matrix, w przypadku gdy wiersze macierzy mają różną długość zgłaszany jest wyjątek *std::invalid_argument("rows of different lengths")* , a w przypadku wczytywania danych podanych jako przedziały w formie tekstowej zgłoszony zostanie wyjątek *std::invalid_argument("odd number of numbers - intervals could not be created")* jeżeli ilość liczb jest nieparzysta (brakuje któregoś końca przedziału).

## 6. Pola klas

Klasa *Matrix<T>:*

| | |
|---|---|
| - *vector<std::vector<T>> matrix;* | pole przechowujące wartości macierzy, pod indeksami *matrix[r][c]* znajduje się element w wierszu *r* i kolumnie *c* macierzy. Indeksowane od 0. |
| - *int rows;* | ilość rzędów w macierzy |
| - *int columns;* | ilość kolumn w macierzy |

Klasa *LinearEquationsSystem<T>:*

| | |
|---|---|
| - *Matrix<T> matrix;* | obiekt klasy *Matrix* reprezentujący macierz uzupełnioną [A\|b] układu równań liniowych Ax = b; |
| - *int equationsNumber;* | ilość równań w układzie |
| - *int constatntTermsColumn;* | indeks kolumny wyrazów wolnych |
| - *vector<int> columnsOrder;* | wektor pamiętający aktualną kolejność kolumn |
| - *vector<T> result;* | wektor przechowujący rozwiązanie układu po jego rozwiązaniu |
| - *Status status;* | pole typu wyliczeniowego, przechowuje status układu równań, przyjmuje jedną z wartości: *empty, solved, unsolved, wrongEquationsNumber, error.* |

## 7. Kod źródłowy

*Załączony tu kod nie zawiera implementacji funkcji i metod nieużywanych przez metodę GaussJordan dla arytmetyki przedziałowej.*

```
template <typename T>
class LinearEquationsSystem{
public:
    LinearEquationsSystem(Matrix<T> systemMatrix);
    LinearEquationsSystem(std::vector<std::vector<T>> systemMatrix) :
LinearEquationsSystem(Matrix<T>(systemMatrix)) { }
    LinearEquationsSystem(std::string systemMatrixString) :
LinearEquationsSystem(Matrix<T>(systemMatrixString)) { }
    LinearEquationsSystem();
    Matrix<T>* getMatrix() { return &matrix; }
    int getEquationsNumber() { return equationsNumber; }
    std::vector<T> GaussJordan();

    std::string getFormattedMatrix(){ return matrix.toString(); }
    std::string getFormattedResult(std::string variableName, std::string separator, int
precision, bool scientificNotation);
    std::string getFormattedResult();

    enum class Status{
        empty,
        solved,
        unsolved,
        wrongEquationsNumber,
        error
```

```
    };

private:
    Matrix<T> matrix;
    int equationsNumber;
    int constatntTermsColumn;
    std::vector<int> columnsOrder;
    std::vector<T> result;
    Status status = Status::empty;

    void replaceRows(int row1, int row2);
    void replaceColumns(int column1, int column2);
};

template <typename T>
std::vector<T> LinearEquationsSystem<T>::GaussJordan(){
    try{
        MatrixIndex maxElement;
        for(int i = 0; i < equationsNumber; ++i){
            maxElement = matrix.findMaxAbs({i, i}, {equationsNumber - 1, equationsNumber - 1});
            replaceRows(i, maxElement.row);
            replaceColumns(i, maxElement.column);
            matrix.divideRowByNumber(i, matrix.valueAt(i,i));
            for(int j = 0; j < equationsNumber; ++j){
                if(j == i){
                    continue;
                }
                matrix.addMultiplyOfRow(i, -1 * matrix.valueAt(j, i), j);
            }
        }

        for(int i = 0; i < columnsOrder.size(); ++i){
            result[columnsOrder[i]] = matrix.valueAt(i, constatntTermsColumn);
        }

        status = Status::solved;

        return result;
    }
    catch(...){
        status = Status::error;
        return {};
    }
}

template <typename T>
void LinearEquationsSystem<T>::replaceRows(int row1, int row2){
    matrix.replaceRows(row1, row2);
}

template <typename T>
void LinearEquationsSystem<T>::replaceColumns(int column1, int column2){
    matrix.replaceColumns(column1, column2);
    std::swap(columnsOrder[column1], columnsOrder[column2]);
}
```

```
struct MatrixIndex{
    int row;
    int column;
};
```

```
template <typename T>
class Matrix{
public:
    Matrix(std::vector<std::vector<T>> matrix);
    Matrix(int rows, int columns);
    Matrix() : Matrix(0, 0) { }
    Matrix(std::string matrixString, bool intervalInput);
    Matrix(std::string matrixString);

    T valueAt(MatrixIndex index);
    T valueAt(int r, int c);
    void setValue(MatrixIndex index, T value);
    void setValue(int r, int c, T value);

    int getRowsNumber() { return rows; }
    int getColumnsNumber() { return columns; }

    void replaceRows(int row1, int row2);
    void replaceColumns(int column1, int culumn2);
    void multiplyRowByNumber(int row, T number);
    void multiplyColumnByNumber(int column, T number);
    void divideRowByNumber(int row, T divisor);
    void divideColumnByNumber(int column, T divisor);
    void addMultiplyOfRow(int rowToAdd, T multiply, int addToRow);
    void addMultiplyOfColumn(int columnToAdd, T multiply, int addToColumn);
    void addDividedRow(int rowToAdd, T divisor, int addToRow);
    void addDividedColumn(int columnToAdd, T divisor, int addToColumn);

    bool indexValidation(MatrixIndex index);

    MatrixIndex findMaxAbs();
    MatrixIndex findMaxAbs(MatrixIndex upperLeftBoundsCorner, MatrixIndex
bottomRightBoundsCorner);

    void print();
    std::string toString(std::string columnSeparator, std::string rowSeparator, int precision,
bool scientificNotation);
    std::string toString();

private:
    int rows;
    int columns;
    std::vector<std::vector<T>> matrix;
};

template <typename T>
bool intervalContains(interval_arithmetic::Interval<T> interval, T number){
    if(interval.a <= number && interval.b >= number){
        return true;
    }
    else{
        return false;
    }
}

template <typename T>
bool Matrix<T>::indexValidation(MatrixIndex index){
    if(index.row < 0 || index.row >= rows || index.column < 0 || index.column >= columns){
        return false;
    }

    return true;
}
```

```
template <>
MatrixIndex Matrix<interval_arithmetic::Interval<long double>>::findMaxAbs(MatrixIndex
upperLeftBoundsCorner, MatrixIndex bottomRightBoundsCorner){
    if(!indexValidation(upperLeftBoundsCorner) || !indexValidation(bottomRightBoundsCorner)){
        throw std::out_of_range("try to access index outside the matrix");
    }

    MatrixIndex maximumIndex = upperLeftBoundsCorner;
    long double maximum = -1.0L;
    for(int r = upperLeftBoundsCorner.row; r <= bottomRightBoundsCorner.row; ++r){
        for(int c = upperLeftBoundsCorner.column; c <= bottomRightBoundsCorner.column; ++c){
            long double distanceFromZero = std::min(std::abs(matrix[r][c].a),
std::abs(matrix[r][c].b));
            if(intervalContains(matrix[r][c], 0.0L)){
                distanceFromZero *= -1;
            }
            if(distanceFromZero > maximum){
                maximum = distanceFromZero;
                maximumIndex = {r,c};
            }
        }
    }
    return maximumIndex;
}

template <>
void Matrix<interval_arithmetic::Interval<long double>>::divideRowByNumber(int row,
interval_arithmetic::Interval<long double> divisor){
    if(row < 0 || row >=  rows){
        throw std::out_of_range("try to access row outside the matrix");
    }
    if(intervalContains(divisor, 0.0L)){
        throw std::invalid_argument("divisor must not contain zero");
    }

    for(int c = 0; c < columns; ++c){
        matrix[row][c] = matrix[row][c] / divisor;
    }
}

template <typename T>
void Matrix<T>::addMultiplyOfRow(int rowToAdd, T multiply, int addToRow){
    if(rowToAdd < 0 || rowToAdd >= rows || addToRow < 0 || addToRow >= rows){
        throw std::out_of_range("try to access row outside the matrix");
    }

    for(int c =  0; c < columns; ++c){
        matrix[addToRow][c] = matrix[addToRow][c] + matrix[rowToAdd][c] * multiply;
    }
}

template <typename T>
void Matrix<T>::replaceRows(int row1, int row2){
    if(row1 == row2){
        return;
    }

    if(row1 < 0 || row1 >= rows || row2 < 0 || row2 >= rows){
        throw std::out_of_range("try to access row outside the matrix");
    }

    std::swap(matrix[row1], matrix[row2]);
```

```
}

template <typename T>
void Matrix<T>::replaceColumns(int column1, int column2){
    if(column1 == column2){
        return;
    }

    if(column1 < 0 || column1 >= columns || column2 < 0 || column2 >= columns){
        throw std::out_of_range("try to access column outside the matrix");
    }

    for(int r = 0; r < rows; ++r){
        std::swap(matrix[r][column1], matrix[r][column2]);
    }
}
```

## 8. Przykłady

Przykład 1 – dane w postaci liczb:

**Dane:**

```
-0.925     97.33      -40.309    65.328     -17.288    -15.97
-43.101    -11.599    26.24      57.927     -5.428     0.944
-14.977    27.711     24.177     46.522     -84.948    38.106
-37.666    73.299     37.683     -3.178     -7.637     -6.253
22.264     -13.217    10.038     79.965     14.086     88.115
```

**Wynik:**

```
x[0] = [1.4897955796435390E0, 1.4897955796435391E0],       width = 2.3e-18
x[1] = [2.6843465427160720E-2, 2.6843465427160725E-2],     width = 3.8e-18
x[2] = [1.3083663984641712E0, 1.3083663984641713E0],       width = 3.3e-18
x[3] = [5.3396417083268794E-1, 5.3396417083268795E-1],     width = 1.9e-18
x[4] = [-3.7686037984680719E-2, -3.7686037984680715E-2],   width = 2.6e-18

status = Status::solved
```

Przykład 2 – dane w postaci przedziałów:

**Dane:**

```
58.14 59.78       -30.24 -30.10     -4.44 -3.89

74.12 75.56       2.81 3.15         31.23 31.83
```

**Wynik:**

```
x[0] = [3.7490253775147151E-1, 3.8905355143076830E-1],      width = 1.4e-02

x[1] = [8.5172625194001922E-1, 9.0382598512949646E-1],      width = 5.2e-02

status = Status::solved
```

Przykład 3 – układ sprzeczny, dane w postaci liczb:

**Dane:**

```
12    18     24    90

6     9      12    11

1     56     7     12
```

**Wynik:**

```
status = Status::error
```

## 9. Literatura


I. B. Pańczyk, E. Łukasik, J. Sikora, T. Guziak, *Metody numeryczne w przykładach*, Politechnika Lubelska, 2012

II. Z. Fortuna, B. Macukow, J. Wąsowski, Metody numeryczne, WNT