Wydział WFiIS	Imię i nazwisko 1. Ihnatsi Yermakovich		Rok II	Grupa 03	
METODY **NUMERYCZNE**	Temat Metoda Jacobiego				Nr ćwiczenia 02
Data wykonania 26.03.2022	Data oddania 27.03.2022	Zwrot do popr.	Data oddania	Data zaliczenia	OCENA

Metoda Jacobiego

Ćwiczenie nr 02

Ihnatsi Yermakovich

1 Wprowadzenie

Jak wcześniej sformułujemy problem jako wyznaczenie wektoru $\mathbf{x}$ przy znanej macierzy współczynników A i wektorze wyrazów wolnych $\mathbf{b}$:

$$A\mathbf{x} = \mathbf{b} \tag{1}$$

1.1 Metody iteracyjne

Prawie zawsze macierze wielkich układów liniowych nie są macierzami pełnymi, ale rzadkimi, tzn. mają niewiele elementów niezerowych. Jednym ze sposobów rozwiązywania wielkich układów równań jest stosowanie metod iteracyjnych.

Jedną z najprostszych metod iteracyjnych jest metoda iteracji prostej. Polega ona na przejściu od danego układu równań liniowych (1) do równoważnego (tzn. mającego te same rozwiązania) układu:

$$\mathbf{x} = B\mathbf{x} + \mathbf{c} \tag{2}$$

1.2 Zbieżność metod iteracyjnych

Dla macierzy: $A \in R^{n \times n}$ definiujemy liczbę:

$$\rho(A) = \max_{i=1,2,\ldots,n} \geqslant |\lambda_i| \tag{3}$$

gdzie:

λ_i jest i-tą wartością własną macierzy A.

którą nazywamy promieniem spektralnym macierzy.

Istotną rzeczą jest znajomość warunku wystarczającego zbieżności metody iteracyjnej. Okazuje się, że wystarczającym warunkiem na to, aby ciąg zdefiniowany wzorem $\left\{\mathbf{x}^{(i)}\right\}, i = 1, 2, ...$ był zbieżny do rozwiązania układu (1) jest, aby była spełniona poniższa nierówność:

$$\rho(B) < 1 \tag{4}$$

1.3 Metoda Jacobiego

Jako wstęp do metod iteracyjnych rozpatrzyliśmy metodę Jacobiego. Dla metody Jacobiego wystarczającym warunkiem zbieżności jest spełnienie przez macierz A warunku dominującej przekątnej:

$$|a_{ii}| > \sum_{j=1, j\neq i}^{n} |a_{ij}|, \;\; dla\ i\ = 1, 2, ..., n \tag{5}$$

Jeżeli macierz $A = [a_{ij}]$ spełnia warunek *warunek dominującej przekątnej*, to istnieje iteracyjnie zbieżne rozwiązanie układu równań (1).

Metoda Jacobiego powstaje poprzez wyliczenie z i-tego równania układu (1) zmiennej x_i. Co dla kolejnych iteracji możemy sformułować jako:

$$x_i^{(k+1)} = \frac{b_i - \sum\limits_{j\neq i} a_{ij} x_j^{(k)}}{a_{ii}} \tag{6}$$

Aby rozpącząć iteracje musimy wybrać wektor początkowy $\mathbf{x}^{(0)} = \left[x_1^{(0)}, ..., x_n^{(0)}\right]^T$. Ale jeżeli macierz A spełnia warunek (5), to ciąg przybliżeń $\left(\mathbf{x}^{(k)}\right)_{k=0}^{\infty}$ jest zawsze zbieżny do rozwiązania układu (1) niezależnie od wyboru wektora startowego.

1.4 Macierz odwrotna za pomocą metody Jacobiego

Aby znaleźć macierz odwrotną do macierzy A rozwiążamy n razy następujący układ:

$$A\mathbf{x} = \mathbf{e}_i \tag{7}$$

gdzie $\mathbf{e}_i$ jest równe:

$$\mathbf{e}_i = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{8}$$

Rozwiązania dla $i = 1, 2, ..., n$ będą kolejnymi kolumnami macierzy odwrotnej A^{-1}.

2 Implementacja i wyniki

2.1 Rozwiązanie układu równań liniowych za pomocą metody Jacobiego

Wczytamy macierze A i b z pliku następująco:

```c
#define N 4

gsl_matrix *A = read_matrix_from_file("src/pd/pd_2/assets/A_matrix.txt", N, N);
gsl_matrix *b_matrix = read_matrix_from_file("src/pd/pd_2/assets/b_vector.txt", N, 1);
gsl_vector *b = gsl_vector_alloc_col_from_matrix(b_matrix, 0);
```

Listing 1: Wczytanie macierzy A i b z plików na dysku

Następnie sprawdzimy, czy macierz jest jest dominująca przekątniowo:

```c
int is_diagonally_dominant(gsl_matrix *A)
{
    for (size_t i = 0; i < A->size1; i++)
    {
        double Aii = m_get(A, i, i);
        double sum = 0;

        for (size_t j = 0; j < A->size2; j++)
        {
            if (i == j)
                continue;

            sum += abs(m_get(A, i, j));
        }

        if (abs(Aii) < sum)
            return 0;
    }
    return 1;
}

.............

if (!is_diagonally_dominant(A))
{
    printf("Matrix should be diagonally dominant!\n\n");
    exit(EXIT_FAILURE);
}
```

Listing 2: Sprawdzanie czy macierz A jest dominująca przekątniowo

Zdefiniujemy wektor rozwiązań **x** i rozwiążemy układ (1) wypisując dane z każdej 10-ej iteracji:

```
gsl_vector *x = gsl_vector_calloc(N);


for (size_t i = 0; i < I; i++)
{
    for (size_t k = 0; k < N; k++)
    {
        double sum = 0;
        for (size_t j = 0; j < N; j++)
        {
            if (j == k)
                continue;

            sum += m_get(A, k, j) * v_get(x, j);
        }

        double newX = (v_get(b, k) - sum) / m_get(A, k, k);

        v_set(x, k, newX);
    }

    if (i % 10 == 0)
        print_iteration(i, x);
}
```

Listing 3: Sprawdzanie czy macierz A jest dominująca przekątniowo

2.2 Testowanie zapropowanego rozwiązania

Pryzkład rozwiązania dla n = 5:

```
Macierz współczynnikow A:
11.000 2.000 4.000 0.000 1.000
0.000 5.000 -1.000 2.000 0.000
1.000 2.000 10.000 4.000 2.000
0.000 9.000 1.000 -15.000 1.000
3.000 5.000 6.000 1.000 29.000


Macierz wyrazów wolnych b:
4.000
9.000
2.000
5.000
6.000


Iteration i = 0.   x: (0.364   1.800   -0.196   0.734   -0.126)
Iteration i = 10.   x: (0.214   1.512   -0.339   0.550   -0.025)
Iteration i = 20.   x: (0.214   1.512   -0.339   0.550   -0.025)
Iteration i = 30.   x: (0.214   1.512   -0.339   0.550   -0.025)
Iteration i = 40.   x: (0.214   1.512   -0.339   0.550   -0.025)
Iteration i = 50.   x: (0.214   1.512   -0.339   0.550   -0.025)
Iteration i = 60.   x: (0.214   1.512   -0.339   0.550   -0.025)
```

```
Iteration i = 70.   x: (0.214   1.512   -0.339   0.550   -0.025)
Iteration i = 80.   x: (0.214   1.512   -0.339   0.550   -0.025)
Iteration i = 90.   x: (0.214   1.512   -0.339   0.550   -0.025)
Iteration i = 100.   x: (0.214   1.512   -0.339   0.550   -0.025)

Rozwiązanie:
0.214
1.512
-0.339
0.550
-0.025
```

Listing 4: Sprawdzenie metody Jacobiego dla n = 5

Pryzkład rozwiązania dla n = 6:

```
Macierz współczynnikow A:
19 2 4 0 1 3
0 5 -1 2 0 2
1 2 10 4 0 1
0 9 1 -15 1 1
3 5 6 1 29 1
5 1 9 8 1 39


Macierz wyrazów wolnych b:
4
9
2
5
6
9


Iteration i = 0.   x: (0.211   1.800   -0.181   0.735   -0.113   0.052)
Iteration i = 10.   x: (0.103   1.462   -0.330   0.531   -0.011   0.148)
Iteration i = 20.   x: (0.103   1.462   -0.330   0.531   -0.011   0.148)
Iteration i = 30.   x: (0.103   1.462   -0.330   0.531   -0.011   0.148)
Iteration i = 40.   x: (0.103   1.462   -0.330   0.531   -0.011   0.148)
Iteration i = 50.   x: (0.103   1.462   -0.330   0.531   -0.011   0.148)
Iteration i = 60.   x: (0.103   1.462   -0.330   0.531   -0.011   0.148)
Iteration i = 70.   x: (0.103   1.462   -0.330   0.531   -0.011   0.148)
Iteration i = 80.   x: (0.103   1.462   -0.330   0.531   -0.011   0.148)
Iteration i = 90.   x: (0.103   1.462   -0.330   0.531   -0.011   0.148)
Iteration i = 100.   x: (0.103   1.462   -0.330   0.531   -0.011   0.148)

Rozwiązanie:
0.1034
1.462
-0.3301
0.5312
-0.01106
0.1475
```

Listing 5: Sprawdzenie metody Jacobiego dla n = 6

2.3 Zależność rozwiązania od wyboru x

Wpisując duże losowe wartości do początkowego wektoru rozwiązań $\mathbf{x}$ nie udało się zepsuć końcowego rozwiązania, natomiast udało się sprawić, aby rozwiązanie było otrzymywane przy większej ilości niezbędnych iteracji. Jeżeli przy $\mathbf{x} = 0$ poprawna odpowiedź (do 3 znaków po przecinku) jest otrzymywana w mniej niż 10 krokach, to przy losowych wartościach odpowidź jest otrzymywana przy liczbie niezbędnych kroków mniejszej od 20.

2.4 Znajdowanie macierzy odwrotnej za pomocą metody Jacobiego

Znajdziemy macierz odwrotną i sprawdzimy rozwiązanie za pomocą następującego równania:

$$A \times A^{-1} = I \tag{9}$$

```c
gsl_matrix *A = read_matrix_from_file("src/pd/pd_2/2/assets/A_matrix.txt", N, N);
gsl_matrix *A_1 = gsl_matrix_calloc(N, N);

printf("Macierz A:\n");
print_matrix(A);

for (size_t i = 0; i < N; i++)
{
    gsl_vector *e_i = gsl_vector_calloc(N);
    v_set(e_i, i, 1);

    gsl_vector *x = linalg_solve_jacobi(A, e_i, I);
    gsl_matrix_set_col(A_1, i, x);

    gsl_vector_free(e_i);
    gsl_vector_free(x);
}

printf("Odworocona macierz A_1:\n");
print_matrix(A_1);

gsl_matrix *I_ = gsl_matrix_calloc(N, N);
gsl_blas_dgemm(CblasNoTrans, CblasNoTrans, 1.0, A, A_1, 0.0, I_);

printf("Wynik mnozenia macierzy A i A_1:\n");
print_matrix(I_);

save_matrix_to_file("src/pd/pd_2/2/assets/result.txt", I_);

gsl_matrix_free(A);
gsl_matrix_free(A_1);
gsl_matrix_free(I_);
```

Listing 6: Wyznaczenie macierzy odwrotnej, spradzenie warunku (9) i zapisanie wyniku do pliku

W wyniku otrzmymy nastepujący output:

```
Macierz A:
4.0000 0.2857 1.0000 0.3333 0.1666 0.2000
0.3333 -5.0000 0.2000 1.5000 0.2857 0.1250
0.1111 0.1666 -3.0000 -1.0000 0.1666 0.2857
0.2000 0.5000 -1.0000 8.0000 2.0000 0.3333
0.1250 0.3330 0.3000 0.1250 3.0000 0.2500
0.2857 2.0000 0.1666 0.2000 -1.0000 6.0000


Odworocona macierz A_1:
0.2484 0.0107 0.0812 -0.0016 -0.0221 -0.0114
0.0150 -0.1955 -0.0193 0.0336 -0.0026 0.0027
0.0085 -0.0047 -0.3119 -0.0396 0.0486 0.0148
-0.0025 0.0046 -0.0482 0.1195 -0.0776 -0.0011
-0.0112 0.0159 0.0305 -0.0035 0.3281 -0.0149
-0.0189 0.0673 0.0179 -0.0146 0.0578 0.1634


Wynik mnozenia macierzy A i A_1:
1.0000 0.0000 0.0000 -0.0000 -0.0000 -0.0000
-0.0000 1.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 1.0000 0.0000 -0.0000 0.0000
-0.0000 0.0000 -0.0000 1.0000 0.0000 0.0000
-0.0000 0.0000 0.0000 0.0000 1.0000 0.0000
-0.0000 0.0000 0.0000 0.0000 0.0000 1.0000
```

Listing 7: Ouput dla listingu (6)

Analizując powyższy output możemy wnioskować o poprawności zastosowanej metody.

3 Wnioski

- Dla macierzy rzadkich stosowanie metod iteracyjnych może dawać szybsze rozwiązanie niż metody skończone.
- Chociaż metoda Jacobiego nie jest optymalna, jest ona jedną z najprostszych i wprowadzających w świat metod iteracyjnych.
- Dla macierzy dominujących przekątniowo metoda Jacobiego daje poprawne wyniki nezależnie od wartości początkowej wektora $\mathbf{x}$.
- Za pomocą metody Jacobiego potrafimy znaleźć macierz odwrotną do macierzy A.