Rachunek macierzowy i statystyka wielowymiarowa

Laboratorium 2. Eliminacja Gaussa i LU-faktoryzacja

Ivan Smaliakou, Jakub Karbowski

26 marca 2024

1 Eliminacja Gaussa bez pivotyngu

1.1 Pseudokod

```
begin
    for i=0 to n_columny
        for j=i to n_wierszy
            if i == j {
                A[j] /= A[j][j];
            } else {
                A[j] -= A[i] * A[j][i];
            }
        end
    end
end
```

1.2 Kod

```
def gaussian_elimination_no_pivoting(A: np.array, B: np.array):
    A = A.copy()
    B = B.copy()
    assert len(A.shape) == 2
    assert len(B.shape) == 1

    for i in range(A.shape[1]):
        # always start from the diag element, so as a result it'll yeild low-ech
        for j in range(i, A.shape[0]):
            el = A[j][i]
            # 1's elemination part
            if i == j:
                if el != 1:
```

```
                A, B = row_divide(A, B, j, i)
            else:
                if el != 0:
                    A, B = rows_eliminate(A, B, j, i)
    return (np.around(A), np.around(B))


# would return the new matrix with row divided by the element A_iijj
def row_divide(A: np.array, B: np.array, ii, jj):
    n = A[ii][jj]
    for j in range(len(A[ii])):
        A[ii][j] /= n
    B[ii] /= n
    return (A, B)


def rows_eliminate(A: np.array, B: np.array, ii, jj):
    A_copy = A.copy()
    B_copy = B.copy()
    support_row_index = ii + 1 if ii < A.shape[0] - 1 and A_copy[ii + 1][jj] !=
    if A_copy[support_row_index][jj] == 0:
        raise ValueError('support value must be non-zero')
    support_row_multiplier = A_copy[ii][jj] / A_copy[support_row_index][jj]

    for j in range(len(A[support_row_index])):
        A_copy[support_row_index][j] *= support_row_multiplier
    B_copy[support_row_index] *= support_row_multiplier
    for j in range(len(A[support_row_index])):
        A[ii][j] -= A_copy[support_row_index][j]
    B[ii] -= B_copy[support_row_index]
    return (A, B)
```

1.3 Porównanie z MATLAB

W środowisku MATLAB nie znalazłem odpowiedniej funkcji dla eliminacji Gaussa bez pivotingu. Poza tym każda eliminacja dzieje w swój sposób.

2 Eliminacja Gaussa z pivotyngiem

2.1 Pseudokod

```
begin
    for i=0 to n_wierszy{
        if A[i][i] == 0{
            for j = 0 to n_wierszy{
```

```
                if i == j {continue};
                A[i],A[j] = A[j], A[i];
            }
        }
    }
    for i=0 to n_columny
        for j=i to n_wierszy
            if i == j {
                A[j] /= A[j][j];
            } else {
                A[j] -= A[i] * A[j][i];
            }
        end
    end
end
```

2.2 Kod

```
def row_divide(A: np.array, B: np.array, ii, jj):
    n = A[ii][jj]
    for j in range(len(A[ii])):
        A[ii][j] /= n
    B[ii] /= n
    return (A, B)


def rows_eliminate(A: np.array, B: np.array, ii, jj):
    A_copy = A.copy()
    B_copy = B.copy()
    support_row_index = ii + 1 if ii < A.shape[0] - 1 and A_copy[ii + 1][jj] !=
    if A_copy[support_row_index][jj] == 0:
        raise ValueError('support value must be non-zero')
    support_row_multiplier = A_copy[ii][jj] / A_copy[support_row_index][jj]

    for j in range(len(A[support_row_index])):
        A_copy[support_row_index][j] *= support_row_multiplier
    B_copy[support_row_index] *= support_row_multiplier
    for j in range(len(A[support_row_index])):
        A[ii][j] -= A_copy[support_row_index][j]
    B[ii] -= B_copy[support_row_index]
    return (A, B)

def swap(A, b, i):
    A_copy = A.copy()
    b_copy = b.copy()
```

```
    for j in range(A_copy.shape[0]):
        if j == i:
            continue
        if A_copy[j][i] != 0 and A_copy[i][j] != 0:
            swap_buff = A_copy[j].copy()
            swap_buff_b = b_copy[j]
            A_copy[j] = A_copy[i].copy()
            A_copy[i] = swap_buff
            b_copy[j] = b_copy[i]
            b_copy[i] = swap_buff_b
            return (A_copy, b_copy)
    raise ValueError('row for pivoting is not found')


def gaussian_elimination_pivoting(A: np.array, b: np.array):
    assert len(A.shape) == 2
    assert len(b.shape) == 1
    for i in range(A.shape[0]):
        if A[i][i] == 0:
            A, b = swap(A, b, i)
    display(pd.DataFrame(A))
    return gaussian_elimination_no_pivoting(A, b)
```

2.3 Porównanie z MATLAB

W środowisku MATLAB nie znalazłem odpowiedniej wbudowanej funkcji dla eliminacji Gaussa z pivotingiem. Poza tym każda eliminacja dzieje w swój sposób.

3 LU faktoryzacja bez pivotyngu

3.1 Pseudokod

```
begin
    U = A.copy();
    L = identity_matrix(A.size);
    for i=0 to n_columny
        for j=i to n_wierszy
            L[j][i] = U[j][i];
            U[j] -= U[i] * U[j][i];
        end
    end
    return L, U
end
```

3.2 Kod

```
def LU_no_pivoting(A, B):
A = A.copy()
B = B.copy()
C = np.identity(A.shape[0])
assert len(A.shape) == 2
assert len(B.shape) == 1

for i in range(A.shape[1]):
    # always start from the diag element, so as a result it'll yeild low-ech
    for j in range(i+1, A.shape[0]):
        el = A[j][i]
        if el != 0:
            A, B, mult = rows_eliminate_lu(A, B, j, i)
            C[j][i] = mult
return (A, B, C) # C is an L matrix
```

3.3 Porównanie z MATLAB

W MATLAB istnieją tylko opcja LU faktoryzacji z pivoryngiem (częstkowym lub pełnym - do wyboru). Więc możemy sprawdzić w taki sposób:
$P_{matlab} \times L_{kod} \times U_{kod} = L_{matlab} \times U_{matlab}$, gdzie index X_{matlab} oznacza zmienną z MATLABU, a X_{kod} - zmienną z kodu.
Załóżmy macierz zródłowa A ma postać:

Tabela 1: Macierz A

0.8147	0.6324	0.9575	0.9572
0.9058	0.0975	0.9649	0.4854
0.1270	0.2785	0.1576	0.8003
0.9134	0.5469	0.9706	0.1419

Wówczas lewa strona przyjmi postać L*:

Tabela 2: Lewa strona L*

0.9134	0.5469	0.9706	0.1419
0.9058	0.0975	0.9649	0.4854
0.8147	0.6324	0.9575	0.9572
0.127	0.2785	0.1576	0.8003

Oraz prawa strona przyjmi postać R*:

Tabela 3: Prawa strona R*

0.9134	0.5469	0.9706	0.1419
0.9058	0.0975	0.9649	0.4854
0.8147	0.6324	0.9575	0.9572
0.127	0.2785	0.1576	0.8003

L*=R* ■

4 LU factorizacja z pivotingiem

4.1 Pseudokod

```
begin
    U = A.copy();
    L = identity_matrix(A.size);
    P = identity_matrix(A.size);
    for i=0 to n_columny
        for j=n_wierszy to i (inversed)
            if abs(A[j][i]) > abs(A[i][i]){
                swap(A, j, i);
                break;
            }
        end
    end
    for i=0 to n_columny
        for j=i to n_wierszy
            L[j][i] = U[j][i];
            U[j] -= U[i] * U[j][i];
        end
    end
    return L, U
end
```

4.2 Kod

```
def swap(A, b, P, i, j):
    A_copy = A.copy()
    b_copy = b.copy()
    P_copy = P.copy()
    temp = A_copy[i].copy()
    A_copy[i] = A_copy[j].copy()
    A_copy[j] = temp
    temp_b = b_copy[i]
```

```
    b[i] = b[j]
    b[j] = temp_b
    temp_P = P_copy[i].copy()
    P_copy[i] = P_copy[j].copy()
    P_copy[j] = temp_P
    return (A_copy, b_copy, P_copy)


def LU_pivoting(A, b):
    # sorting
    P = np.identity(A.shape[0])
    for i in range(A.shape[1]):
        for j in range(A.shape[0]-1, i, -1):
            if np.abs(A[j][i]) > np.abs(A[i][i]):
                A, b, P = swap(A, b, P, i, j)
                break
    return LU_no_pivoting(A, b), P.astype(int)
```

4.3 Porównanie z MATLAB

LU faktoryzacja z piovtyngiem w MATLAB oraz własna faktoryzacja mogą być oparte na różnych elementach macierzy A. Zatem dla weryfikacji zastosowano formuły:
$A = P_{matlab}^{-1} \times L_{matlab} \times U_{matlab} = P_{kod}^{-1} \times L_{kod} \times U_{kod}$;
Niech A=

Tabela 4: Macierz A

0	0.6324	0.9575	0.9572
0.9058	0	0.9649	0.4854
0.1270	0.2785	0	0.8003
0.9134	0.5469	0.9706	0

4.3.1 Część MATLAB

Wówczas z MATLABu otzymujemy L_{matlab}:

Tabela 5: Macierz L_{matlab}

1	0	0	0
0.9058	1	0	0
0.9917	-0.8576	1	0
0.1390	0.3201	-0.5481	1

oraz U_{matlab}:

Tabela 6: Macierz U_{matlab}

0.9134	0.5469	0.9706	0
0	0.6324	0.9575	0.9572
0	0	0.8055	1.3063
0	0	0	1.2098

oraz P_{matlab}:

Tabela 7: Macierz P_{matlab}

0	0	0	1
1	0	0	0
0	1	0	0
0	0	1	0

wtedy lewa strona L* = $P^{-1}_{matlab} \times L_{matlab} \times U_{matlab} = A$

Tabela 8: Macierz $P^{-1}_{matlab} \times L_{matlab} \times U_{matlab}$

0	0.6324	0.9575	0.9572
0.9058	0	0.9649	0.4854
0.1270	0.2785	0	0.8003
0.9134	0.5469	0.9706	0

oraz L_{kod}:

Tabela 9: Macierz L_{kod}

1	0	0	0
0	1	0	0
0.99167944	-0.85760513	1	0
0.13904095	0.32014312	-0.53609298	1

oraz U_{kod}:

Tabela 10: Macierz U_{kod}

0.9134	0.5469	0.9706	0
0	0.6324	0.9575	0.9572
0	0	0.82353	1.30629963
0	0	0	1.19415707

oraz P_{kod}:

Tabela 11: Macierz P_{kod}

0	0	0	1
1	0	0	0
0	1	0	0
0	0	1	0

Wówczas prawa strona P* = $P_{kod}^{-1} \times L_{kod} \times U_{kod} = A$

Tabela 12: Macierz $P_{kod}^{-1} \times L_{kod} \times U_{kod}$

0	0.6324	0.9575	0.9572
0.9058	0	0.9649	0.4854
0.1270	0.2785	0	0.8003
0.9134	0.5469	0.9706	0

L*=P* ■