Rachunek Macierzowy i Statystyka Wielowymiarowa

Raport z zadania 2. - Implementacja eliminacji gaussa oraz rozkładu LU.

Wojciech Jasiński, Michał Stefanik

Wydział Informatyki
Akademia Górniczo Hutnicza
Kraków
26 marca 2024

Spis treści

1 Wstęp

Należało zaimplementować eliminację Gaussa oraz rozkład LU w wersji bez pivotingu oraz z pivotingiem. Następnie opisać pseudokod algorytmów oraz zaimplementować je w wybranym języku programowania. Aby przetestować poprawność implementacji, należało przetestować je na macierzy gęstej i porównać wyniki z wynikami uzyskanymi za pomocą Matlab/Octave.

2 Postać macierzy

Macierze zostały zapisane w postaci listy list, gdzie każda lista wewnętrzna reprezentuje wiersz macierzy. Dla przykładu macierz 3x3: byłaby zapisana jako:

```
A = [[1, 2, 3],
     [4, 5, 6],
     [7, 8, 9]]
```

a wektor b:

```
b = [[1],
     [2],
     [3]]
```

Używany przez nas typ to:

```
matrixType = List[List[float]]
```

3 Eliminacja Gaussa

3.1 Eliminacja Gaussa bez pivotingu - pseudokod

Funkcja rozwiązująca układ równań (solve_matrix):
Wejście: macierz A, wektor b, wartość epsilon

1. Skopiuj macierz A i wektor b
2. Przejdź przez każdy wiersz i:
   (a) Jeśli wartość na przekątnej jest bliska zero (mniejsza od epsilon), zamień ten wiersz z następnym, który ma wartość na przekątnej większą od epsilon
   (b) Dla każdego wiersza poniżej obecnego, oblicz współczynnik jako wartość elementu w kolumnie obecnego wiersza podzielona przez wartość na przekątnej obecnego wiersza. Następnie odejmij od tego wiersza obecny wiersz pomnożony przez współczynnik.
3. Przejdź przez każdy wiersz od końca do początku i:
   (a) Dla każdego wiersza powyżej obecnego, oblicz współczynnik jako wartość elementu w kolumnie obecnego wiersza podzielona przez wartość na przekątnej obecnego wiersza. Następnie odejmij od tego wiersza obecny wiersz pomnożony przez współczynnik.
4. Przejdź przez każdy wiersz i:
   (a) Jeśli wartość na przekątnej jest większa od epsilon, podziel wartość w wektorze b przez wartość na przekątnej i ustaw wartość na przekątnej na 1.0. W przeciwnym razie ustaw wartość w wektorze b na 0.0.
5. Zwróć wektor b jako rozwiązanie układu równań.

3.2 Eliminacja Gaussa z pivotingiem - pseudokod

Funkcja rozwiązująca układ równań (solve_matrix):
Wejście: macierz A, wektor b, wartość epsilon

1. Skopiuj macierz A i wektor b
2. Przejdź przez każdy wiersz i:

   (a) Dla każdego wiersza poniżej obecnego, jeśli wartość absolutna elementu w kolumnie obecnego wiersza jest większa od wartości absolutnej elementu na przekątnej obecnego wiersza, zamień te dwa wiersze
   (b) Dla każdego wiersza poniżej obecnego, oblicz współczynnik jako wartość elementu w kolumnie obecnego wiersza podzielona przez wartość na przekątnej obecnego wiersza. Następnie odejmij od tego wiersza obecny wiersz pomnożony przez współczynnik.

3. Przejdź przez każdy wiersz od końca do początku i:
   (a) Dla każdego wiersza powyżej obecnego, oblicz współczynnik jako wartość elementu w kolumnie obecnego wiersza podzielona przez wartość na przekątnej obecnego wiersza. Następnie odejmij od tego wiersza obecny wiersz pomnożony przez współczynnik.

4. Przejdź przez każdy wiersz i:
   (a) Jeśli wartość na przekątnej jest większa od epsilon, podziel wartość w wektorze b przez wartość na przekątnej i ustaw wartość na przekątnej na 1.0. W przeciwnym razie ustaw wartość w wektorze b na 0.0.

5. Zwróć wektor b jako rozwiązanie układu równań.

3.3 Funkcja rozwiązująca układ równań (solve_matrix) dla obu wersji eliminacji Gaussa

```python
def solve_matrix(A: matrixType, b: matrixType, pivot=False, eps=1.0e-10):
    n = len(A)
    A = copy_matrix(A)
    b = copy_matrix(b)

    # forward pass
    for i in range(n):
        if pivot:
            for j in range(i + 1, n):
                if abs(A[j][i]) > abs(A[i][i]):
                    A[i], A[j] = A[j], A[i]
                    b[i], b[j] = b[j], b[i]
        # check if A[i][i] is zero, if yes, swap with the next non-zero row
        if abs(A[i][i]) < eps:
            for j in range(i + 1, n):
                if abs(A[j][i]) > eps:
                    A[i], A[j] = A[j], A[i]
                    b[i], b[j] = b[j], b[i]
                    break
        for j in range(i + 1, n):
            factor = A[j][i] / A[i][i]
            for k in range(i, n):
                A[j][k] -= factor * A[i][k]
            b[j][0] -= factor * b[i][0]

    # back substitution
    for j in range(n - 1, -1, -1):
        for i in range(j - 1, -1, -1):
            if abs(A[j][j]) > eps:
                factor = A[i][j] / A[j][j]
            else:
                factor = 0.0
            for k in range(j, n):
                A[i][k] -= factor * A[j][k]
            b[i][0] -= factor * b[j][0]

    # normalize
    for i in range(n):
        if abs(A[i][i]) > eps:
```

```
            b[i][0] /= A[i][i]
            A[i][i] = 1.0
        else:
            b[i][0] = 0.0

    return b
```

3.4 Testy

Poprawność rozwiązań sprawdzano przez generacje dużych losowych macierz oraz wektorów i porównanie wyników z wynikami uzyskanymi za pomocą funkcji linsolve z Octave/Matlab. Dla każdego testu wyniki okazały się zgodne z wynikami uzyskanymi za pomocą funkcji linsolve z Octave/Matlab. Poniżej kod generujący testy oraz kod testujący poprawność rozwiązań.

3.4.1 Generowanie rozwiązań

```
found = False
while not found:
    dim = 200
    A = np.random.rand(dim, dim).astype(float)
    b = np.random.rand(dim, 1).astype(float)

    try:
        solution = np.linalg.solve(A, b)
    except np.linalg.LinAlgError:
        continue
    found = True

    A_lst = A.tolist()
    b_lst = b.tolist()

    dirpath = "02/"

    gf_pivot = np.array(gauss_factorization(A_lst, b_lst, pivot=True))
    gf_nopivot = np.array(gauss_factorization(A_lst, b_lst))

    np.savetxt(dirpath + "A.csv", A, delimiter=",")
    np.savetxt(dirpath + "b.csv", b, delimiter=",")
    np.savetxt(dirpath + "gf_pivot.csv", gf_pivot, delimiter=",")
    np.savetxt(dirpath + "gf_nopivot.csv", gf_nopivot, delimiter=",")
    np.savetxt(dirpath + "np_solution.csv", solution, delimiter=",")
```

3.4.2 Testowanie poprawności rozwiązań

```
filePath_A = '02/A.csv';
filePath_b = '02/b.csv';
filePath_p = '02/gf_pivot.csv';
filePath_np = '02/gf_nopivot.csv';
filePath_solution = '02/np_solution.csv';


A = csvread(filePath_A);
b = csvread(filePath_b);
p = csvread(filePath_p);
np = csvread(filePath_np);
solution = csvread(filePath_solution);

x = linsolve(A,b);

tolerance = 1e-10;
```

```
    disp(isequal(abs(x-p) < tolerance, ones(size(x))));
    disp(isequal(abs(x-np) < tolerance, ones(size(x))));
    disp(isequal(abs(x-solution) < tolerance, ones(size(x))));
```

4 Rozkład LU

Poniżej przedstawiamy implementację rozkładu LU w wersji bez pivotingu oraz z pivotingiem.

4.1 Rozkład LU bez pivotingu - pseudokod

4.2 Rozkład LU z pivotingiem - pseudokod

4.3 Rozkład LU bez pivotingu - kod

```
def lu_factorization(A: matrixType) -> Tuple[matrixType, matrixType]:
    n = len(A)
    L = [[0.0] * n for _ in range(n)]
    U = [[0.0] * n for _ in range(n)]

    for j in range(n):
        L[j][j] = 1.0
        for i in range(j + 1):
            sum_u = sum(U[k][j] * L[i][k] for k in range(i))
            U[i][j] = A[i][j] - sum_u
        for i in range(j + 1, n):
            sum_l = sum(L[i][k] * U[k][j] for k in range(j))
            L[i][j] = (A[i][j] - sum_l) / U[j][j]
    return L, U
```

4.4 Rozkład LU z pivotingiem - kod

```
def lu_factorization_with_pivoting(A: matrixType) \
    -> Tuple[matrixType, matrixType, matrixType]:
    n = len(A)
    L = [[0.0] * n for _ in range(n)]
    U = [[0.0] * n for _ in range(n)]

    # Initialize the permutation matrix as an identity matrix
    P = [[float(i == j) for i in range(n)] for j in range(n)]

    for j in range(n):
        # Partial pivoting
        pivot_value = abs(A[j][j])
        pivot_row = j
        for i in range(j+1, n):
            if abs(A[i][j]) > pivot_value:
                pivot_value = abs(A[i][j])
                pivot_row = i
        if pivot_row != j:
            # Swap rows in A, P
            A[j], A[pivot_row] = A[pivot_row], A[j]
            P[j], P[pivot_row] = P[pivot_row], P[j]

        # Proceed with LU decomposition
        L[j][j] = 1.0
        for i in range(j + 1):
            sum_u = sum(U[k][j] * L[i][k] for k in range(i))
            U[i][j] = A[i][j] - sum_u
        for i in range(j + 1, n):
```

```
        sum_l = sum(L[i][k] * U[k][j] for k in range(j))
        L[i][j] = (A[i][j] - sum_l) / U[j][j]

    return L, U, P
```

5 Wnioski

Testowane algorytmy różniły się od siebie stabilnością. Warianty bez pivotingu były mniej stabilne, co skutkowało większą liczbą błędów numerycznych. W szczególności częściej pojawiał się błąd dzielenia przez zero. Warianty z pivotingiem były bardziej stabilne, co skutkowało mniejszą liczbą błędów numerycznych. Warto zauważyć, że warianty z pivotingiem były nieco bardziej złożone obliczeniowo, co skutkowało dłuższym czasem wykonania.