Algorytmy macierzowe - laboratorium 2

Marta Bukowińska
Jakub Karczewski

5.11.2024

1 Treść zadania

Proszę wybrać ulubiony język programowania, wygenerować macierze losowe o wartościach z przedziału otwartego $(0.00000001, 1.0)$ i zaimplementować:

1. Rekurencyjne odwracanie macierzy (10 punktów)
2. Rekurencyjna eliminacja Gaussa (10 punktów)
3. Rekurencyjna LU faktoryzacja (10 punktów)
4. Rekurencyjne liczenie wyznacznika (10 punktów)

Proszę zliczać liczbę operacji zmienno-przecinkowych wykonywanych podczas mnożenia macierzy.

2 Rekurencyjne odwracanie macierzy

2.1 Pseudokod

Algorithm 1 Recursive Function to Compute the Inverse of a Matrix

function InverseRecursive(A)
  $n \leftarrow \text{dimension}(A)$
  if $n = 1$ **then**
    return $1/A$
  end if
  $mid \leftarrow \lfloor n/2 \rfloor$
  Partition A as follows:
  $A_{11} \leftarrow A[0 : mid, 0 : mid]$
  $A_{12} \leftarrow A[0 : mid, mid : n]$
  $A_{21} \leftarrow A[mid : n, 0 : mid]$
  $A_{22} \leftarrow A[mid : n, mid : n]$
  $A_{11}^{-1} \leftarrow \text{InverseRecursive}(A_{11})$
  $S_{22} \leftarrow A_{22} - A_{21} \cdot A_{11}^{-1} \cdot A_{12}$
  $S_{22}^{-1} \leftarrow \text{InverseRecursive}(S_{22})$
  $B_{11} \leftarrow A_{11}^{-1} \cdot \left(I + A_{12} \cdot S_{22}^{-1} \cdot A_{21} \cdot A_{11}^{-1}\right)$
  $B_{12} \leftarrow -A_{11}^{-1} \cdot A_{12} \cdot S_{22}^{-1}$
  $B_{21} \leftarrow -S_{22}^{-1} \cdot A_{21} \cdot A_{11}^{-1}$
  $B_{22} \leftarrow S_{22}^{-1}$
  Combine $B_{11}, B_{12}, B_{21}, B_{22}$ into the result matrix
  return result
end function

2.2 Fragmenty kodu

```
A11_inv = inverse_recursive(A11, mult_alg)

S22 = subtract_matrices(A22, mult_alg(mult_alg(A21, A11_inv), A12))

S22_inv = inverse_recursive(S22, mult_alg)

B11 = mult_alg(A11_inv, add_matrices(np.identity(mid), mult_alg(mult_alg(A12, S22_inv), mult_alg(A21, A11_inv))))
B12 = (-1) * mult_alg(mult_alg(A11_inv, A12), S22_inv)
B21 = (-1) * mult_alg(mult_alg(S22_inv, A21), A11_inv)
B22 = S22_inv
```

```
if(n == 1):
    return 1/A1
mid = n//2
```

2.3 Sprawdzenie poprawności

Wynik dla przykładowej macierzy porównano z wynikiem zwracanym przez `np.linalg.inv`. Różnica mieściła się w zadanej tolerancji

2.4 Złożoność obliczeniowa

rekurencyjny algorytm odwrotności macierzy przy użyciu mnożenia macierzy algorytmem Strassena ma złożoność:

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n^{2.81})$$

Rozwiązując to równanie rekurencyjne, otrzymujemy złożoność całkowitą:

$$T(n) = O(n^{2.81})$$

3 Rekurencyjna eliminacja Gaussa

1. Pseudokod

Algorithm 2 Recursive Gaussian Elimination

function GaussRecursive(A, B)
    $n \leftarrow \text{dimension}(A)$
    if $n = 1$ **then**
        return A, B
    end if
    $mid \leftarrow \lfloor n/2 \rfloor$
    Partition A and B as follows:
    $A_{11} \leftarrow A[0 : mid, 0 : mid]$
    $A_{12} \leftarrow A[0 : mid, mid : n]$
    $A_{21} \leftarrow A[mid : n, 0 : mid]$
    $A_{22} \leftarrow A[mid : n, mid : n]$
    $b_1 \leftarrow B[0 : mid]$
    $b_2 \leftarrow B[mid : n]$
    $L_{11}, U_{11} \leftarrow \text{LU\_Factor}(A_{11})$
    $L_{11}^{-1} \leftarrow \text{InverseRecursive}(L_{11})$
    $U_{11}^{-1} \leftarrow \text{InverseRecursive}(U_{11})$
    $S \leftarrow A_{22} - A_{21} \cdot U_{11}^{-1} \cdot L_{11}^{-1} \cdot A_{12}$
    $L_s, U_s \leftarrow \text{LU\_Factor}(S)$
    $L_s^{-1} \leftarrow \text{InverseRecursive}(L_s)$
    $C_{11} \leftarrow U_{11}$
    $C_{12} \leftarrow L_{11}^{-1} \cdot A_{12}$
    $C_{22} \leftarrow U_s$
    $b'_1 \leftarrow L_{11}^{-1} \cdot b_1$
    $b'_2 \leftarrow L_s^{-1} \cdot b_2 - L_s^{-1} \cdot A_{21} \cdot U_{11}^{-1} \cdot L_{11}^{-1} \cdot b_1$
    Combine C_{11}, C_{12}, C_{22} into the result matrix
    Combine b'_1, b'_2 into the result vector
    return C, b
end function

3.1 Fragmenty kodu

```python
n = len(A)
if(n == 1):
    return A, B
mid = n//2

A11 = A[:mid, :mid]
A12 = A[:mid, mid:]
A21 = A[mid:, :mid]
A22 = A[mid:, mid:]

b1 = B[:mid, 0]
b2 = B[mid:, 0]

L11, U11 = LU_factor(A11, mult_alg)
L11_inv = inverse_recursive(L11, mult_alg)
U11_inv = inverse_recursive(U11, mult_alg)
S = subtract_matrices(A22, mult_alg(mult_alg(mult_alg(A21, U11_inv), L11_inv), A12))
Ls, Us = LU_factor(S, mult_alg)
Ls_inv = inverse_recursive(Ls, mult_alg)

C11 = U11
C12 = mult_alg(L11_inv, A12)
C22 = Us

b1_new = matrix_naive(L11_inv, b1)
b2_new = subtract_matrices(matrix_naive(Ls_inv, b2), matrix_naive(mult_alg(mult_alg(mult_alg(Ls_inv, A21), U11_inv), L11_inv ), b1 ))
```

3.2 Sprawdzenie poprawności

Zostało sprawdzone, że macierz po eliminacji Gaussa jest w postaci schodkowej. Rozwiaznie układu równań dla przykładowej macierzy z użyciem `np.linalg.solve` porównano z rozwiazniem uzyskanym przez uzycie tej metody na macierzy po eliminacji. Różnica mieściła się w zadanej tolerancji

3.3 Złożoność obliczeniowa

4 Rekurencyjna LU faktoryzacja

1. Pseudokod

Algorithm 3 Recursive LU factorization

function LUFactor(A)
  $n \leftarrow \text{dimension}(A)$
  if $n = 1$ **then**
    return $A, \begin{bmatrix} 1 \end{bmatrix}$
  end if
  $mid \leftarrow \lfloor n/2 \rfloor$
  Partition A as:
  $A_{11} \leftarrow A[0 : mid, 0 : mid]$
  $A_{12} \leftarrow A[0 : mid, mid : n]$
  $A_{21} \leftarrow A[mid : n, 0 : mid]$
  $A_{22} \leftarrow A[mid : n, mid : n]$
  $L_{11}, U_{11} \leftarrow \text{LUFactor}(A_{11})$
  $U_{11}^{-1} \leftarrow \text{InverseRecursive}(U_{11})$
  $L_{21} \leftarrow A_{21} \cdot U_{11}^{-1}$
  $L_{11}^{-1} \leftarrow \text{InverseRecursive}(L_{11})$
  $U_{12} \leftarrow L_{11}^{-1} \cdot A_{12}$
  $S \leftarrow A_{22} - (A_{21} \cdot U_{11}^{-1} \cdot L_{11}^{-1} \cdot A_{12})$
  $L_s, U_s \leftarrow \text{LUFactor}(S)$
  $U_{22} \leftarrow U_s$
  $L_{22} \leftarrow L_s$
  Combine L_{11}, L_{21}, L_{22} into L_{res}
  Combine U_{11}, U_{12}, U_{22} into U_{res}
  return L_{res}, U_{res}
end function

4.1 Fragmenty kodu

```
L11, U11 = LU_factor(A11, mult_alg)
U11_inv = inverse_recursive(U11, mult_alg)
L21 = mult_alg(A21, U11_inv)
L11_inv = inverse_recursive(L11, mult_alg)
U12 = mult_alg(L11_inv, A12)
S = subtract_matrices(A22, mult_alg(mult_alg(mult_alg(A21, U11_inv), L11_inv), A12))
Ls, Us = LU_factor(S, mult_alg)
U22 = Us
L22 = Ls
```

```
n = A.shape[0]
if(n == 1):
    return A, np.array([[1]])
```

4.2 Sprawdzenie poprawnosci

```python
delta_LU = abs( x - np.matmul(L1.astype(np.double), U1.astype(np.double)))
for j in range(U1.shape[1]):
    for i in range(j+1, U1.shape[0]):
        assert (U1[i, j] <= 1e-8)

for j in range(L1.shape[1]):
    for i in range(j-1, -1, -1):
        assert (L1[i, j] <= 1e-8)
print("delta_LU_max_error; ", delta_LU.max())
```

Sprawdzenie poprawności odbyło się poprzez sprawdzenie funkcją assert, czy nad/pod przekątną znajdują się elementy bardzo bliskie zeru. Dodatkowo wymnożyliśmy macierze L i U, odjęliśmy od wyniki macierz wejściową, a dalej bierzemy z pozostałej macierzy maximum co do wartości bezwzględnej. Wynik mieści się w zadanej tolerancji

4.3 Złożoność obliczeniowa

Rekurencyjny algorytm dekompozycji LU przy użyciu mnożenia macierzy algorytmem Strassena ma złożoność

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n^{2.81})$$

Rozwiązując to równanie rekurencyjne, otrzymujemy złożoność całkowitą:

$$T(n) = O(n^{2.81})$$

5 Rekurencyjne liczenie wyznacznika

1. Pseudokod

Algorithm 4 Recursive Function for Determinant Calculation

```
function DeterminantRecursive(A)
    n ← dimension(A)
    res ← 1
    L, U ← LUFactor(A)
    for i = 0 to n − 1 do
        res ← res × L[i, i] × U[i, i]
    end for
    return res
end function
```

5.1 Fragmenty kodu

```python
def determinant_recursive(A, mult_alg):
    global flops
    res = 1.0
    L, U = LU_factor(A, mult_alg)
    for i in range(A.shape[0]):
        res *= L[i, i] * U[i, i]
        flops += 2
    return res
```

5.2 Sprawdzenie poprawnosci

```python
print("det difference: ", determinant_recursive(x, strassen_alg) - np.linalg.det(x.astype(np.double)))
```

Wynik dla przykładowej macierzy porównano z wynikiem zwracanym przez `np.linalg.det`. Różnica mieściła się w zadanej tolerancji

5.3 Zlozonosc obliczeniowa

Złożoność obliczeniowa wyznacznika macierzy obliczanego za pomocą dekompozycji LU wynosi:

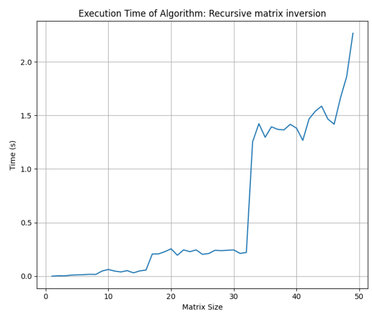
$$T(n) = O(n^{2.81}) + O(n)$$

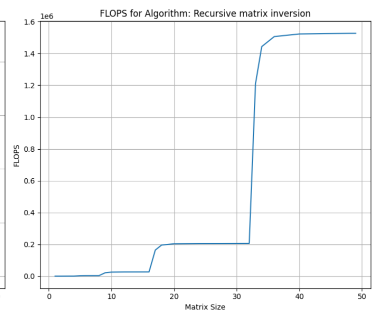
Ponieważ $O(n^{2.81})$ dominuje nad $O(n)$, całkowita złożoność obliczeniowa wyznacznika wynosi:

$$T(n) = O(n^{2.81})$$

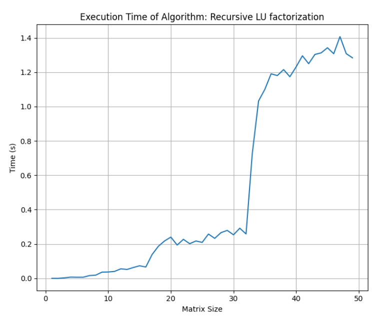
6 Wykresy

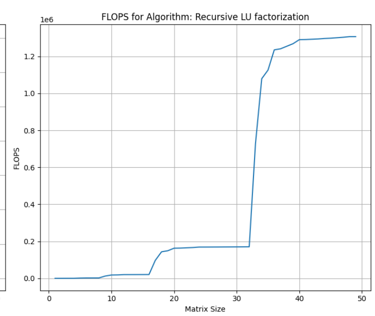
6.1 Rekurencyjne odwracanie macierzy



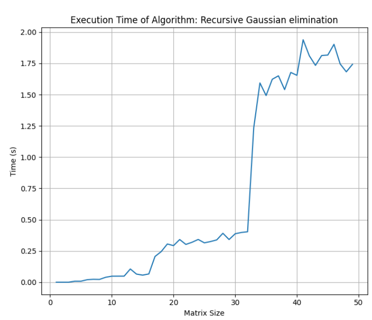


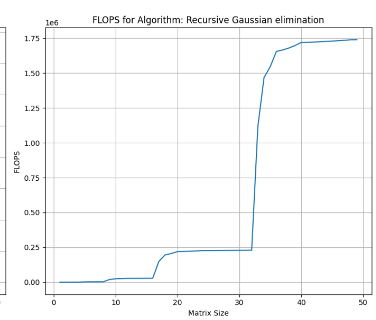
6.2 Rekurencyjna LU faktoryzacja



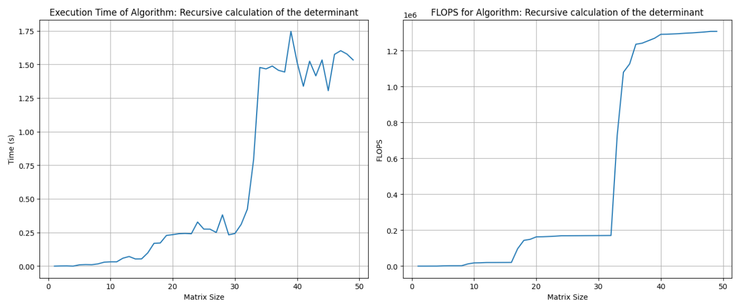


6.3 Rekurencyjny algorytm Gaussa





6.4 Rekurencyjne wyznaczanie wyznacznika macierzy



6.5 Porownanie wszystkich algorytmow na 1 wykresie