Algorytmy macierzowe - Mnożenie macierzy

Jakub Frączek Kacper Garus

13 października 2024

Spis treści

1 Wstęp

Tematem zadania było wygenerowalnie losowych macierzy o wartościach z przedziału otwartego $(0.00000001, 1.0)$, a następnie zaimplementowanie algorytmów:

1. Rekurencyjnego mnożenia macierzy metodą Binét'a
2. Rekurencyjnego mnożenia macierzy metodą Strassena
3. Mnożenia macierzy metodą AI na podstawie artykułu w Nature*

* - https://deepmind.google/discover/blog/discovering-novel-algorithms-with-alphatensor/#: :text=In%20our%20paper,%20published%20today%20in%20Nature,%20we

Następnie zliczyć liczbę operacji zmienno-przecinkowych dokonaną podczas mnożenia macierzy. Algorytmy miały zostać zaprojektowane tak, aby przyjmować macierze o dowolnych wymiarach.

2 Metoda Binét'a

2.1 Opis teoretyczny

Algrorytm Binét'a jest rekurencyjny i można go przedstawić dla przykładowych macierzy A i B w taki sposób:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \quad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$C = \begin{bmatrix} (A_{11}B_{11} + A_{12}B_{21}) & (A_{11}B_{12} + A_{12}B_{22}) \\ (A_{21}B_{11} + A_{22}B_{21}) & (A_{21}B_{12} + A_{22}B_{22}) \end{bmatrix}$$

Gdzie A_{ij}, B_{ij} dla $i = 1, 2, ..., n$ i $j = 1, 2, ..., n$, to macierze

2.2 Pseudokod

```
Funkcja Binet(A, B)
    Jeżeli rozmiar(A) == 1
        Zwróć A * B

    środek = dzielenie_całkowite(rozmiar(A[0]), 2)

    a11 = Wiersze od 0 do środek, Kolumny od 0 do środek z macierzy A
    a12 = Wiersze od 0 do środek, Kolumny od środek do n z macierzy A
    a21 = Wiersze od środek do n, Kolumny od 0 do środek z macierzy A
    a22 = Wiersze od środek do n, Kolumny od środek do n z macierzy A
```

```
b11 = Wiersze od 0 do środek, Kolumny od 0 do środek z macierzy B
b12 = Wiersze od 0 do środek, Kolumny od środek do n z macierzy B
b21 = Wiersze od środek do n, Kolumny od 0 do środek z macierzy B
b22 = Wiersze od środek do n, Kolumny od środek do n z macierzy B

c11 = Binet(a11, b11) + Binet(a12, b21)
c12 = Binet(a11, b12) + Binet(a12, b22)
c21 = Binet(a21, b11) + Binet(a22, b21)
c22 = Binet(a21, b12) + Binet(a22, b22)

Zwróć macierz C złożóną z c11, c12, c21, c22
```

2.3 Implementacja

Algorytm postanowiliśmy zaimplementować w języku Python:

```python
def binet(a,b):

    if np.size(a)==1:
        return a*b
    n=np.size(a[0])
    mid=n//2

    a11=a[:mid,:mid]
    a12=a[:mid,mid:]
    a21=a[mid:,:mid]
    a22=a[mid:,mid:]
    b11=b[:mid,:mid]
    b12=b[:mid,mid:]
    b21=b[mid:,:mid]
    b22=b[mid:,mid:]

    c11=binet(a11, b11)+binet(a12, b21)
    c12=binet(a11, b12)+binet(a12, b22)
    c21=binet(a21, b11)+binet(a22, b21)
    c22=binet(a21, b12)+binet(a22, b22)

    return np.vstack((np.hstack((c11,c12)),np.hstack((c21,c22))))
```

3 Metoda Strassena

3.1 Opis teoretyczny

Algorytm Strassena jest rekurencyjny i można go przedstawić dla przykładowych macierzy A i B w następujący sposób:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \quad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$C = \begin{bmatrix} P_1 + P_4 - P_5 + P_7 & P_2 + P_4 \\ P_3 + P_5 & P_1 - P_2 + P_3 + P_6 \end{bmatrix}$$

$$\begin{aligned} P_1 &= (A_{11} + A_{22})(B_{11} + B_{22}) & P_2 &= (A_{21} + A_{22})B_{11} \\ P_3 &= A_{11}(B_{12} - B_{22}) & P_4 &= A_{22}(B_{21} - B_{11}) \\ P_5 &= (A_{11} + A_{12})B_{22} & P_6 &= (A_{21} - A_{11})(B_{11} + B_{12}) \\ P_7 &= (A_{12} - A_{22})(B_{21} + B_{22}) \end{aligned}$$

Gdzie A_{ij}, B_{ij} dla $i = 1, 2, ..., n$ i $j = 1, 2, ..., n$, to macierze

3.2 Pseudokod

```
Funkcja Binet(A, B)
    Jeżeli rozmiar(A) == 1
        Zwróć A * B

    środek = dzielenie_całkowite(rozmiar(A[0]), 2)

    a11 = Wiersze od 0 do środek, Kolumny od 0 do środek z macierzy A
    a12 = Wiersze od 0 do środek, Kolumny od środek do n z macierzy A
    a21 = Wiersze od środek do n, Kolumny od 0 do środek z macierzy A
    a22 = Wiersze od środek do n, Kolumny od środek do n z macierzy A

    b11 = Wiersze od 0 do środek, Kolumny od 0 do środek z macierzy B
    b12 = Wiersze od 0 do środek, Kolumny od środek do n z macierzy B
    b21 = Wiersze od środek do n, Kolumny od 0 do środek z macierzy B
    b22 = Wiersze od środek do n, Kolumny od środek do n z macierzy B

    p1 = strassen(a11+a22, b11+b22)
    p2 = strassen(a21+a22, b11)
    p3 = strassen(a11, b12-b22)
    p4 = strassen(a22, b21-b11)
    p5 = strassen(a11+a12, b22)
    p6 = strassen(a21-a11, b11+b12)
    p7 = strassen(a12-a22, b21+b22)
```

```
c11=p1+p4-p5+p7
c12=p3+p5
c21=p2+p4
c22=p1-p2+p3+p6

Zwróć macierz C złożóną z c11, c12, c21, c22
```

3.3 Implementacja

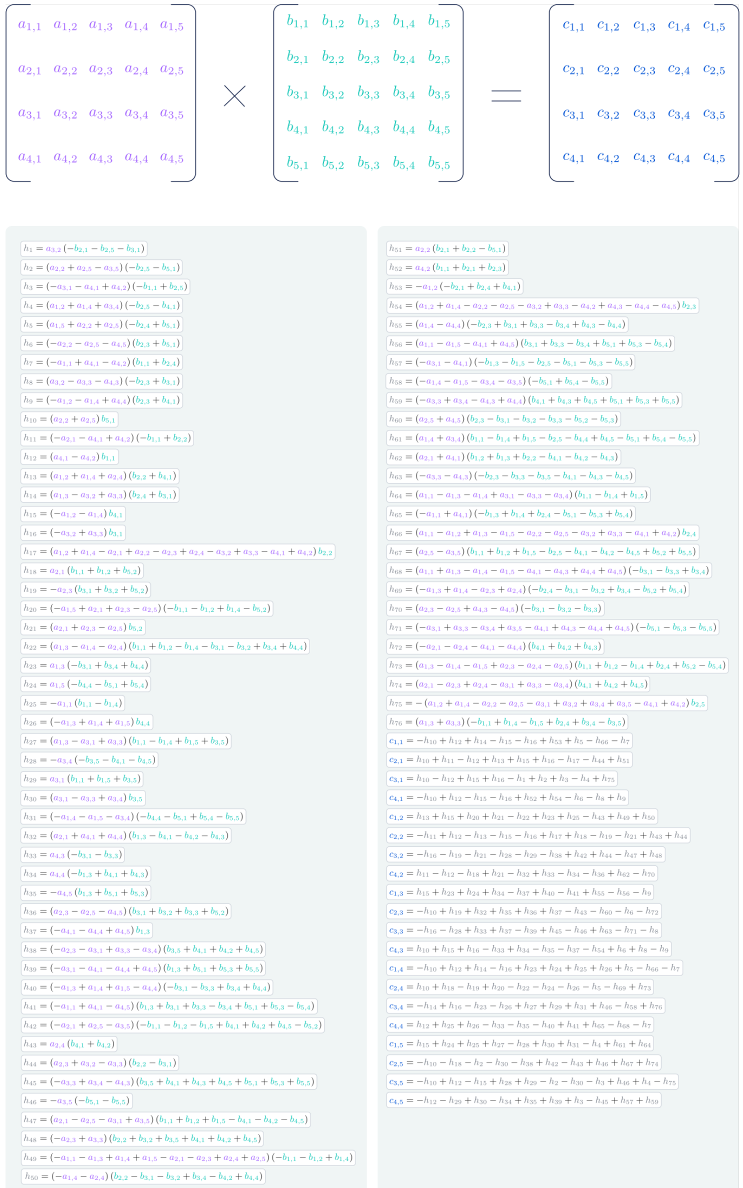
Algorytm Strassena również został zaimplementowany w języku Python:

```python
def strassen(a,b):
    n=np.size(a[0])

    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:]
    b11=b[:mid,:mid]
    b12=b[:mid,mid:]
    b21=b[mid:,:mid]
    b22=b[mid:,mid:]

    p1 = strassen(a11+a22, b11+b22)
    p2 = strassen(a21+a22, b11)
    p3 = strassen(a11, b12-b22)
    p4 = strassen(a22, b21-b11)
    p5 = strassen(a11+a12, b22)
    p6 = strassen(a21-a11, b11+b12)
    p7 = strassen(a12-a22, b21+b22)

    c11=p1+p4-p5+p7
    c12=p3+p5
    c21=p2+p4
    c22=p1-p2+p3+p6

    return np.vstack((np.hstack((c11,c12)),np.hstack((c21,c22))))
```

4 Metoda AI

4.1 Opis teoretyczny

Autorzy artykułu "Discovering novel algorithms with AlphaTensor"w Nature postanowili spojrzeć na problem mnożenia macierzy w inny sposób przekształcając go w grę z bardzo dużą liczbą możliwych ruchów, której ukończenie jest równoważne znalezieniu szukanej macierzy. A następnie za pomocą uczenia maszynowego nauczyli model AlphaTensor, jak w nią graćm a ten metodą prób i błędóœ zaczął odkrywać najpierw już znane algorytmy, takie jak metoda Binét'a oraz metoda Strassena, aż dokonał przełomu odkrywając sposób wymnożenia macierzy 4x5 przez macierz 5x5 szybciej niż to było dotychczas możliwe. Najprostszy znany algorytm wykonuje obliczenia przy użyciu 100 mnożeń, algorytm Strassena przy 80, a algorytm AI przy 76.



Rysunek 1: Algorytm wymyślony przez AI

4.2 Implementacja

Algorytm wymyśloney przez sztuczną inteligencję również został zaimplementowany w Pythonie:

```python
def ai_matrix_mult(a,b):
    a11=a[0,0]
    a12=a[0,1]
    a13=a[0,2]
    a14=a[0,3]
    a15=a[0,4]
    a21=a[1,0]
    a22=a[1,1]
    a23=a[1,2]
    a24=a[1,3]
    a25=a[1,4]
    a31=a[2,0]
    a32=a[2,1]
    a33=a[2,2]
    a34=a[2,3]
    a35=a[2,4]
    a41=a[3,0]
    a42=a[3,1]
    a43=a[3,2]
    a44=a[3,3]
    a45=a[3,4]
    b11=b[0,0]
    b12=b[0,1]
    b13=b[0,2]
    b14=b[0,3]
    b15=b[0,4]
    b21=b[1,0]
    b22=b[1,1]
    b23=b[1,2]
    b24=b[1,3]
    b25=b[1,4]
    b31=b[2,0]
    b32=b[2,1]
    b33=b[2,2]
    b34=b[2,3]
    b35=b[2,4]
    b41=b[3,0]
    b42=b[3,1]
    b43=b[3,2]
    b44=b[3,3]
    b45=b[3,4]
    b51=b[4,0]
    b52=b[4,1]
    b53=b[4,2]
    b54=b[4,3]
```

```
b55=b[4,4]

h1=a32*(-b21-b25-b31)
h2=(a22+a25-a35)*(-b25-b51)
h3=(-a31-a41+a42)*(-b11+b25)
h4=(a12+a14+a34)*(-b25-b41)
h5=(a15+a22+a25)*(-b24+b51)
h6=(-a22-a25-a45)*(b23+b51)
h7=(-a11+a41-a42)*(b11+b24)
h8=(a32-a33-a43)*(-b23+b31)
h9=(-a12-a14+a44)*(b23+b41)
h10=(a22+a25)*(b51)
h11=(-a21-a41+a42)*(-b11+b22)
h12=(a41-a42)*(b11)
h13=(a12+a14+a24)*(b22+b41)
h14=(a13-a32+a33)*(b24+b31)
h15=(-a12-a14)*(b41)
h16=(-a32+a33)*(b31)
h17=(a12+a14-a21+a22-a23+a24-a32+a33-a41+a42)*(b22)
h18=(a21)*(b11+b12+b52)
h19=(-a23)*(b31+b32+b52)
h20=(-a15+a21+a23-a25)*(-b11-b12+b14-b52)
h21=(a21+a23-a25)*(b52)
h22=(a13-a14-a24)*(b11+b12-b14-b31-b32+b34+b44)
h23=(a13)*(-b31+b34+b44)
h24=(a15)*(-b44-b51+b54)
h25=(-a11)*(b11-b14)
h26=(-a13+a14+a15)*(b44)
h27=(a13-a31+a33)*(b11-b14+b15+b35)
h28=(-a34)*(-b35-b41-b45)
h29=(a31)*(b11+b15+b35)
h30=(a31-a33+a34)*(b35)
h31=(-a14-a15-a34)*(-b44-b51+b54-b55)
h32=(a21+a41+a44)*(b13-b41-b42-b43)
h33=(a43)*(-b31-b33)
h34=(a44)*(-b13+b41+b43)
h35=(-a45)*(b13+b51+b53)
h36=(a23-a25-a45)*(b31+b32+b33+b52)
h37=(-a41-a44+a45)*(b13)
h38=(-a23-a31+a33-a34)*(b35+b41+b42+b45)
h39=(-a31-a41-a44+a45)*(b13+b51+b53+b55)
h40=(-a13+a14+a15-a44)*(-b31-b33+b34+b44)
h41=(-a11+a41-a45)*(b13+b31+b33-b34+b51+b53-b54)
h42=(-a21+a25-a35)*(-b11-b12-b15+b41+b42+b45-b52)
h43=(a24)*(b41+b42)
```

```
h44=(a23+a32-a33)*(b22-b31)
h45=(-a33+a34-a43)*(b35+b41+b43+b45+b51+b53+b55)
h46=(-a35)*(-b51-b55)
h47=(a21-a25-a31+a35)*(b11+b12+b15-b41-b42-b45)
h48=(-a23+a33)*(b22+b32+b35+b41+b42+b45)
h49=(-a11-a13+a14+a15-a21-a23+a24+a25)*(-b11-b12+b14)
h50=(-a14-a24)*(b22-b31-b32+b34-b42+b44)
h51=(a22)*(b21+b22-b51)
h52=(a42)*(b11+b21+b23)
h53=(-a12)*(-b21+b24+b41)
h54=(a12+a14-a22-a25-a32+a33-a42+a43-a44-a45)*(b23)
h55=(a14-a44)*(-b23+b31+b33-b34+b43-b44)
h56=(a11-a15-a41+a45)*(b31+b33-b34+b51+b53-b54)
h57=(-a31-a41)*(-b13-b15-b25-b51-b53-b55)
h58=(-a14-a15-a34-a35)*(-b51+b54-b55)
h59=(-a33+a34-a43+a44)*(b41+b43+b45+b51+b53+b55)
h60=(a25+a45)*(b23-b31-b32-b33-b52-b53)
h61=(a14+a34)*(b11-b14+b15-b25-b44+b45-b51+b54-b55)
h62=(a21+a41)*(b12+b13+b22-b41-b42-b43)
h63=(-a33-a43)*(-b23-b33-b35-b41-b43-b45)
h64=(a11-a13-a14+a31-a33-a34)*(b11-b14+b15)
h65=(-a11+a41)*(-b13+b14+b24-b51-b53+b54)
h66=(a11-a12+a13-a15-a22-a25-a32+a33-a41+a42)*(b24)
h67=(a25-a35)*(b11+b12+b15-b25-b41-b42-b45+b52+b55)
h68=(a11+a13-a14-a15-a41-a43+a44+a45)*(-b31-b33+b34)
h69=(-a13+a14-a23+a24)*(-b24-b31-b32+b34-b52+b54)
h70=(a23-a25+a43-a45)*(-b31-b32-b33)
h71=(-a31+a33-a34+a35-a41+a43-a44+a45)*(-b51-b53-b55)
h72=(-a21-a24-a41-a44)*(b41+b42+b43)
h73=(a13-a14-a15+a23-a24-a25)*(b11+b12-b14+b24+b52-b54)
h74=(a21-a23+a24-a31+a33-a34)*(b41+b42+b45)
h75=-(a12+a14-a22-a25-a31+a32+a34+a35-a41+a42)*(b25)
h76=(a13+a33)*(-b11+b14-b15+b24+b34-b35)

c11=-h10+h12+h14-h15-h16+h53+h5-h66-h7
c21=h10+h11-h12+h13+h15+h16-h17-h44+h51
c31=h10-h12+h15+h16-h1+h2+h3-h4+h75
c41=-h10+h12-h15-h16+h52+h54-h6-h8+h9
c12=h13+h15+h20+h21-h22+h23+h25-h43+h49+h50
c22=-h11+h12-h13-h15-h16+h17+h18-h19-h21+h43+h44
c32=-h16-h19-h21-h28-h29-h38+h42+h44-h47+h48
c42=h11-h12-h18+h21-h32+h33-h34-h36+h62-h70
c13=h15+h23+h24+h34-h37+h40-h41+h55-h56-h9
c23=-h10+h19+h32+h35+h36+h37-h43-h60-h6-h72
c33=-h16-h28+h33+h37-h39+h45-h46+h63-h71-h8
```

```
c43=h10+h15+h16-h33+h34-h35-h37-h54+h6+h8-h9
c14=-h10+h12+h14-h16+h23+h24+h25+h26+h5-h66-h7
c24=h10+h18-h19+h20-h22-h24-h26-h5-h69+h73
c34=-h14+h16-h23-h26+h27+h29+h31+h46-h58+h76
c44=h12+h25+h26-h33-h35-h40+h41+h65-h68-h7
c15=h15+h24+h25+h27-h28+h30+h31-h4+h61+h64
c25=-h10-h18-h2-h30-h38+h42-h43+h46+h67+h74
c35=-h10+h12-h15+h28+h29-h2-h30-h3+h46+h4-h75
c45=-h12-h29+h30-h34+h35+h39+h3-h45+h57+h59

c=np.array([[c11,c12,c13,c14,c15],[c21,c22,c23,c24,c25],
           [c31,c32,c33,c34,c35],[c41,c42,c43,c44,c45]])

return c
```

5 Porównanie wydajności algorytmów

5.1 Zlliczanie liczby operacji zmiennoprzecinkowych

5.2 Porównanie czasów działania

6 Oszacowanie złożoności obliczeniowej

Złożoność obliczeniową postanowiliśmy oszacować teoretycznie. Dla algorytmu Binét'a algorytm wykonuje 8 mnożeń na każdym etapie, a liczba podziałów wynosi $log2[n]$, zatem liczba operacji wynosi $8^{log2n} = n^3$. Jeśli chodzi o algorytm Strassena wykonuje on 7 mnożeń na każdym etapie, zatem złożoność wynosi $7^{log2n} = 2^{2.81}$.

7 Porównanie wyników z Octave

7.1 Octave

7.2 Wyniki

8 Wnioski