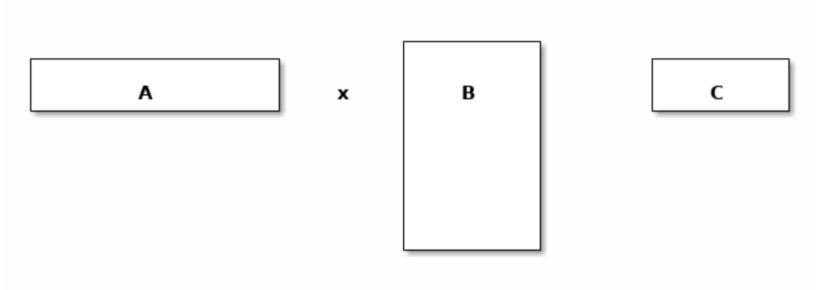
Matrix multiplication

Serge Kruk

October 5, 2021

Matrix multiplication

$$A_{p \times q} \times B_{q \times r} = C_{p \times r}$$



Design an algorithm to perform matrix multiplication

By the method you learned in Linear Algebra.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \cdot \begin{bmatrix} 9 & 8 \\ 7 & 6 \\ 5 & 4 \end{bmatrix} = \begin{bmatrix} 9 + 14 + 15 & 8 + 12 + 12 \\ 36 + 35 + 30 & 32 + 30 + 24 \end{bmatrix}$$

$$= \begin{bmatrix} 38 & 32 \\ 101 & 86 \end{bmatrix}$$

Design an algorithm to perform matrix multiplication

- Loop over every row of the first matrix.
    - Loop over every column of the second matrix.
        - Do an dot product of the selected row/column pair.

Design an algorithm to perform matrix multiplication

```
def size(A):
  return len(A),len(A[0])

def mm(A,B):
  (p,q) = size(A)
  (q,r) = size(B)
  C = [[0 for _ in range(r)] for _ in range(p)]
  for i in range(p):
    for j in range(q):
      for k in range(r):
        C[i][k] += A[i][j] * B[j][k]
  return C
```

Little test

```
A=[[1,2,3],[4,5,6]]
B=[[9,8],[7,6],[5,4]]
mm(A,B)
```

$$\begin{matrix} 38 & 32 \\ 101 & 86 \end{matrix}$$

Runtime

Let us count multiplications and additions

$$T(p, q, r) = \sum_p \sum_r \sum_q 1$$

- $\Theta(pqr)$
- $\Theta(n^3)$ if all matrices are size $n \times n$
    - n^3 Multiplications
    - n^3 Additions

We could reduce the number of additions a bit

- The innermost operation is an inner product

```
def mmf(A,B):
  (p,q) = size(A)
  (q,r) = size(B)
  C = [[0 for _ in range(r)] for _ in range(p)]
  for i in range(p):
    for j in range(r):
      C[i][j] = sum(A[i][k] * B[k][j] for k in range(q))
  return C
```

```
A=[[1,2,3],[4,5,6]]
B=[[9,8],[7,6],[5,4]]
mmf(A,B)
```

$$\begin{matrix} 38 & 32 \\ 101 & 86 \end{matrix}$$

We could reduce the number of additions a bit

- We avoid one addition per inner product.
- There are n^2 inner products on an $n \times n$ matrix.
- Hence the counts are now
  - n^3 multiplications
  - $n^3 - n^2$ additions
- Don't lose sleep over this. It is the kind of micro-optimization that makes or breaks numerical code but is not important for asymptotic analysis.

Note also, in passing . . .

```
for i in range(p):
  for j in range(q):
    for k in range(r):
      C[i][k] += A[i][j] * B[j][k]
```

or

```
for k in range(r):
  for i in range(p):
    for j in range(q):
      C[i][k] += A[i][j] * B[j][k]
```

or

```
for i in range(p):
  for k in range(r):
    for j in range(q):
      C[i][k] += A[i][j] * B[j][k]
```

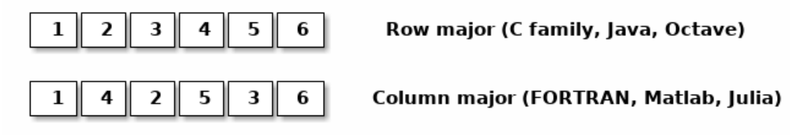
or . . .

How are matrices stored in computer memory?

- Say

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

Then, in memory it could be either of these



Data access pattern

Loop order	Data access
ijk	A by row, B by column
jik	A by row, B by column
ikj	B by row, C by row
jki	A by column, C by column
kij	B by row, C by row
kji	A by column, C by column

- Again, do not lose sleep over this. I mention it to highlight the kinds of nightmares numerical analysts face daily.
- For more on this, the bible is Matrix computations by Golub and Van Loan.

Back to our regular programme

Now let's go back to our problem: how to multiply matrices.

Challenge: do better than n^3

-Do you think it is possible?
-We will need to look at cost in more detail.

Assumption

For a while, let us assume that n is a power of 2.

Demo of block multiplication

```
octave:2> A11=round(10*rand(2,2))
A11 =
    5    3
    0   10
octave:3> A12=round(10*rand(2,2))
A12 =
   7   8
   0   0
octave:4> A21=round(10*rand(2,2))
A21 =
   0   3
   2   6
octave:5> A22=round(10*rand(2,2))
A22 =
   2   8
   2   8
```

Demo of block multiplication

```
octave:6> B11=round(10*rand(2,2))
B11 =
    9   10
    3    6
octave:7> B12=round(10*rand(2,2))
B12 =
   7   9
   5   9
octave:8> B21=round(10*rand(2,2))
B21 =
   8   9
   3   8
octave:9> B22=round(10*rand(2,2))
B22 =
   3   7
   6   5
```

Demo of block multiplication

```
octave:10> A=[A11,A12;A21,A22]
A =

    5    3    7    8
    0   10    0    0
    0    3    2    8
    2    6    2    8

octave:11> B=[B11,B12;B21,B22]
B =

    9   10    7    9
    3    6    5    9
    8    9    3    7
    3    8    6    5
```

Demo of block multiplication

```
octave:12> A*B
ans =
   134   195   119   161
    30    60    50    90
    49   100    69    81
    76   138    98   126

octave:16> [A11*B11+A12*B21, A11*B12+A12*B22;
             A21*B11+A22*B21, A21*B12+A22*B22]
ans =

   134   195   119   161
    30    60    50    90
    49   100    69    81
    76   138    98   126
```

Consider a $2m \times 2m$ block matrix

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

Where

$$\begin{aligned} C_{11} &= A_{11}B_{11} + A_{12}B_{21} \\ C_{12} &= A_{11}B_{12} + A_{12}B_{22} \\ C_{21} &= A_{21}B_{11} + A_{22}B_{21} \\ C_{22} &= A_{21}B_{12} + A_{22}B_{22} \end{aligned}$$

- 8 multiplications (of $m \times m$ matrices)
- 4 additions (of $m \times m$ matrices)

Assuming we do the submatrix operations in standard way

- $A_{ij}B_{kl}$
  - $(m-1)m^2 = m^3 - m^2$ additions
  - m^3 multiplications
- $A_{ij}B_{kl} + A_{op}B_{qp}$
  - $2(m^3 - m^2) + m^2$ additions
  - $2m^3$ multiplications
- For the four blocks
  - $4(2m^3 - m^2) = 8m^3 - 4m^2$ additions
  - $8m^3$ multiplications

Strassen's first brilliant idea

He claims that

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

Strassen's first brilliant idea

Does it work?

```
octave:26> P1=(A11+A22)*(B11+B22)
P1 =
   183   240
   186   232
octave:20> P4=A22*(B21-B11)
P4 =
   -2   14
   -2   14
octave:21> P5=(A11+A12)*B22
P5 =
   102   139
    60    50
octave:23> P7=(A12-A22)*(B21+B22)
P7 =
    55    80
   -94  -136
```

Strassen's first brilliant idea

```
octave:27> P1+P4-P5+P7
ans =
   134   195
    30    60
octave:28> A*B
ans =
   134   195   119   161
    30    60    50    90
    49   100    69    81
    76   138    98   126
```

Strassen's first brilliant idea

- How would you prove that it works?
- By showing that the specified sums yield the product.

Strassen's first brilliant idea (runtime)

Count multiplications and additions.

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

Strassen's first brilliant idea (runtime)

- We have 7 matrix multiplications and 18 additions.
- Doing this, using standard multiplication at the block level.
  - $7(m^3 - m^2) + 18m^2 = 7m^3 + 11m^2$ scalar additions
  - $7m^3$ scalar multiplications

Second surprise

By simple algebra, we shave some operations

- By block view
  - $8m^3 - 4m^2$ additions
  - $8m^3$ multiplications
- By Strassen's first idea and block view
  - $7m^3 + 11m^2$ additions
  - $7m^3$ multiplications

If m is large enough Strassen's wins!

- Assuming multiplication is at least as costly as addition.
- And that the data access pattern doesn't kill cache access.
- Assuming the pipeline is filled properly
- Etc...

Strassen's second brilliant idea

Why stop at one level, recurse! Down to a certain small size.

Strassen's runtime

Assumption (made to simplify analysis)

- $n = 2^q$
- $n_0 = 2^d$

What is the proper recurrence?

Strassen's runtime

Assumption (made to simplify analysis)

- $n = 2^q$
- $n_0 = 2^d$

$$T(n) = \begin{cases} n_0^3 & n \le n_0 \\ 7T(n/2) + c_0 n^2 & \text{otherwise} \end{cases}$$

Strassen's runtime

$$
\begin{aligned}
T(n) &= 7T(n/2) + c_0 n^2 \\
&= 7[7T(n/4) + c_{11}(n/2)^2] + c_0 n^2 && = 7^2 T(n/4) + c_1 n^2 \\
&= 7^2[7T(n/8) + c_{22}(n/4)^2] + c_1 n^2 && = 7^3 T(n/8) + c_2 n^2 \\
&= \dots \\
&= 7^{\log_2 n} n_0^3 + c_{\log_2 n} n^2 \\
&= \Theta(7^{\log_2 n})
\end{aligned}
$$

Strassen's runtime

By the master theorem, you get $\Theta(n^{\log_2 7})$

- Equivalent to our bound (maybe you want to prove that)
- $\Theta(7^{\log_2 n}) = \Theta(n^{\log_2 7}) \approx \Theta(n^{2.8})$

Practical matter

- Assume the matrix is $m \times m$
  - If m is even then we simply divide in half
  - If m is odd, we pad with a row/column of zeros
- We stop the recursion when m is smallish
  - Nowadays that may mean in the low thousands depending on hardware/software/cache

Strassen's algorithm

WARNING: This is pseudo-code. It does not compile.

```
def strassen(A,B,n,n0=1024):
    if n <= n0:
        return mm(A,B)
    else:
        m,u,v = n//2,range(m),range(m+1,n)
        P1 = strassen(A(u,u)+A(v,v),B(u,u)+B(v,v),m,n0)
        P2 = strassen(A(v,u)+A(v,v),B(u,u),m,n0)
        P3 = strassen(A(u,u), B(u,v)-B(v,v),m,n0)
        ...
        C(u,u) = P1 + P4 - P5 + P7
        C(u,v) = P3 + P5
        C(v,u) = P2 + P4
        C(v,v) = P1 - P2 + P3 + P6
    return C
```

Strassen in Octave (This is executable, if sloooooow)

```
function [C] = strassen(A, B, n0=3)
  [row,common] = size(A);
  [common0, column] = size(B);
  assert(common==common0);
  if row <= n0 || column <= n0,
    C = A*B;
  else
    if rem(row, 2) == 1,
        A = [A; zeros(1, common)];
    endif;
    if rem(column, 2) == 1,
        B = [B zeros(common, 1)];
    endif;
    if rem(common, 2) == 1,
        A = [A zeros(size(A, 1), 1)];
        B = [B; zeros(1, size(B, 2))];
    endif;
```

Strassen in Octave

```
[m,o] = size(A);
[o,n] = size(B);
A11 = A(1:m/2, 1:n/2 );
A12 = A(1:m/2, n/2+1: n);
A21 = A(m/2+1 :m, 1:n/2);
A22 = A(m/2+1 :m ,n/2+1: n);
B11 = B(1:n/2, 1:o/2 );
B12 = B(1:n/2, o/2+1: o);
B21 = B(n/2+1 :n, 1:o/2);
B22 = B(n/2+1 :n, o/2+1: o);
P1 = strassen(A11 + A22, B11 + B22, n0);
P2 = strassen(A21 + A22, B11, n0);
P3 = strassen(A11, B12 - B22, n0);
P4 = strassen(A22, B21 - B11, n0);
P5 = strassen(A11 + A12, B22, n0);
P6 = strassen(A21 - A11, B11 + B12, n0);
P7 = strassen(A12 - A22, B21 + B22, n0);
```

Strassen in Octave

```
C = [(P1 + P4 - P5 + P7) (P3 + P5);
     (P2 + P4) (P1 + P3 - P2 + P6)](1:row, 1:column);
  endif;
endfunction;
```

256	259	256	329	212	293
114	117	172	166	128	141
143	102	139	220	115	191
156	175	109	174	105	175
115	145	80	140	92	160
149	127	197	224	161	200

Theoretical conclusions

- Can we do better than $\Theta(n^3)$?

Theoretical conclusions

- Can we do better than $\Theta(n^3)$?
  - Yes! The first was Strassen $\Theta(n^{2.8})$, but many followed.

Theoretical conclusions

- Can we do better than $\Theta(n^3)$?
  - Yes! The first was Strassen $\Theta(n^{2.8})$, but many followed.
- Can we do better than $\Theta(n^2)$?

Theoretical conclusions

- Can we do better than $\Theta(n^3)$?
  - Yes! The first was Strassen $\Theta(n^{2.8})$, but many followed.
- Can we do better than $\Theta(n^2)$?
  - No! We need to output n^2 elements!

Theoretical conclusions

- Can we do better than $\Theta(n^3)$?
  - Yes! The first was Strassen $\Theta(n^{2.8})$, but many followed.
- Can we do better than $\Theta(n^2)$?
  - No! We need to output n^2 elements!
- What is the optimal runtime?

Theoretical conclusions

- Can we do better than $\Theta(n^3)$?
  - Yes! The first was Strassen $\Theta(n^{2.8})$, but many followed.
- Can we do better than $\Theta(n^2)$?
  - No! We need to output n^2 elements!
- What is the optimal runtime?
  - Nobody knows.

Theoretical conclusions

- Can we do better than $\Theta(n^3)$?
  - Yes! The first was Strassen $\Theta(n^{2.8})$, but many followed.
- Can we do better than $\Theta(n^2)$?
  - No! We need to output n^2 elements!
- What is the optimal runtime?
  - Nobody knows.
  - Currently theoretical best is $\Theta(n^{2.3728639})$ (2014, François Le Gall)

Theoretical conclusions

- Can we do better than $\Theta(n^3)$?
  - Yes! The first was Strassen $\Theta(n^{2.8})$, but many followed.
- Can we do better than $\Theta(n^2)$?
  - No! We need to output n^2 elements!
- What is the optimal runtime?
  - Nobody knows.
  - Currently theoretical best is $\Theta(n^{2.3728639})$ (2014, François Le Gall)
  - Currently Strassen's is better in practice.

Theoretical conclusions

- Can we do better than $\Theta(n^3)$?
  - Yes! The first was Strassen $\Theta(n^{2.8})$, but many followed.
- Can we do better than $\Theta(n^2)$?
  - No! We need to output n^2 elements!
- What is the optimal runtime?
  - Nobody knows.
  - Currently theoretical best is $\Theta(n^{2.3728639})$ (2014, François Le Gall)
  - Currently Strassen's is better in practice.
  - There is a group theoretic conjecture implying $\Theta(n^2)$.

Class project

- Implement Strassen's algorithm.