Lecture 13

Computational Cost

Recall Definition 11.1: An algorithm for a map f is a decomposition

$$f = f^{(l)} \circ \cdots \circ f^{(1)}, \quad l \in \mathbb{N}$$

into maps $f^{(i)}$ involving at most one elementary executable operation.

Definition 13.1. [4.1] *The cost of an algorithm is*

$$C(n) = \text{ number of elementary executable operations } [= l]$$

where n is a representative number for the size of the input data.

Example: Consider the following algorithm for the standard matrix-vector product.

Algorithm 5 MatVecStd (standard matrix-vector product)

input: $A = (a_{i,j})_{i,j=1}^{m,n} \in \mathbb{C}^{m\times n}$, $b = (b_i)_{i=1}^{n} \in \mathbb{C}^n$.
output: $x = Ab \in \mathbb{C}^m$.

```
1: for i = 1 to m do
2:     x_i := 0
3:     for j = 1 to n do
4:         x_i := x_i + a_{i,j} b_j
5:     end for
6: end for
```

Line 4 involves 1 addition and 1 multiplication $\rightsquigarrow$ 2op. With the loop in line 3 we obtain $n \times 2\text{op} = 2n\text{op}$. The assignment in line 2 does not count as an operation but from the loop in line 1 we get $m \times 2n\text{op}$, whence Algorithm 5 has computational cost

$$C(m, n) = 2mn\text{op}.$$

Any algorithm for the matrix-vector product will have at least $\Theta(m)$ operations as $m \to \infty$ which can be seen from the fact that m values have to be computed.

Some remarks:

- Estimating the computation time is **not** the aim as this depends too much on the computer architecture. We rather want to get an idea of the complexity of the algorithm for which the number of operations is a good measure.

- In older books there is a distinction between addition/subtraction (cheapest), multiplication, division, and the square root (most expensive) motivated by the way how the operations were carried out by the processors. Nowadays, the processors contain look-up tables so that the time to execute each of those operations basically is the same and this distinction is obsolete.
- High performance computers are parallel computers containing multiple processing units. Issues here are not only the number of operations but also the data exchange and the balancing of the work load. An algorithm is said to scale optimally if doubling the number of processors halves the computation time.

Let us turn our attention to the computational cost of solving systems of linear equations with Gaussian elimination, [5.1], [5.2].

Lemma 13.2. [5.2] **LU** *has cost* $C_{\mathbf{LU}}(n) = \frac{2}{3}n^3 + O(n^2)$ *as* $n \to \infty$.

Proof. Recall Algorithm 2.
Line 7: 1 addition and 1 multiplication, 2op.
Line 6: loop over i, $\sum_{i=k+1}^{n}(2\text{op}) = 2(n-k)\text{op}$.
Line 4: 1 division, $1 + 2(n-k)\text{op}$.
Line 3: loop over j, $\sum_{j=k+1}^{n}(2(n-k)+1)\text{op} = (n-k)(2(n-k)+1)\text{op}$.
Line 2: loop over k, $\sum_{k=1}^{n-1}(n-k)(2(n-k)+1)$. As $n \to \infty$, this can be simplified by considering the following identities,

$$\sum_{k=1}^{n} 1 = n \tag{13.1}$$

$$\sum_{k=1}^{n} k = \frac{1}{2}n(n+1) \tag{13.2}$$

$$\sum_{k=1}^{n} k^2 = \frac{1}{6}n(2n+1)(n+1) \tag{13.3}$$

Thus,

$$\sum_{k=1}^{n-1}(n-k)(2(n-k)+1) = 2\sum_{k=1}^{n-1}(n-k)^2 + \sum_{k=1}^{n-1}(n-k) \tag{13.4}$$

Let,

$$\begin{aligned} & l = n-k \\ & \implies \text{when } k = 1 \longrightarrow l = n-1 \\ & \text{when } k = n-1 \longrightarrow l = 1 \end{aligned}$$

Hence (13.4) can be written as,

$$\begin{aligned} = 2\sum_{l=1}^{n-1} l^2 + \sum_{l=1}^{n-1} l &= \frac{2}{6}(n-1)(2(n-1)+1)n + \frac{1}{2}(n-1)n \\ &= \frac{2}{3}n^3 + O(n^2). \end{aligned}$$

□

It is relatively straightforward to show that the cost of **FS** or **BS** are $O(n^2)$ which leads to the following result:

Theorem 13.1. *The computational cost of* **GE** *is*

$$C_{\mathbf{GE}}(n) \sim \frac{2}{3}n^3 \quad \text{as } n \to \infty.$$

The cost of **GEPP** is the same since the pivoting does not affect the computational cost as exchanging rows does not contribute to the cost according to our definition. But it should be remarked that exchanging data is a big issue on parallel computers and has a huge influence on the overall performance.
The algorithms seen so far have been relatively easy to analyse. Things are getting more interesting when considering divide & conquer algorithms.

Exercise: Formulate the algorithm of the dot product of two vectors, hence evaluate its computational complexity.

Exercise: Given a matrix $A \in \mathbb{C}^{n\times m}$, and a matrix $B \in \mathbb{C}^{m\times q}$ formulate the algorithm that can compute $C = AB$, hence compute its computational cost.

Algorithm 6 MM (Matrix multiplication)

Lecture 14

Divide & Conquer Algorithms

Divide & Conquer is an important design paradigm for algorithms. The idea is to break down a problem into sub-problems which are recursively solved until the problem become small enough to be solved directly. After, the sub-solutions are combined in merging steps to yield the solution to the originating problem.

As an example, we will look at a method to compute the matrix-matrix product which goes back to Strassen (1969). But before that let us recall the standard matrix multiplication.

$$\underbrace{\begin{pmatrix} 2 & 2 & 5 \\ 3 & 8 & 2 \\ 2 & 6 & 8 \end{pmatrix}}_{A} \times \underbrace{\begin{pmatrix} 6 & 4 & 2 \\ 4 & 1 & 2 \\ 5 & 3 & 3 \end{pmatrix}}_{B} = \underbrace{\begin{pmatrix} 45 & 25 & 23 \\ 60 & 26 & 28 \\ 76 & 38 & 40 \end{pmatrix}}_{C}$$

From this it can be seen that the standard matrix-matrix product $C = AB$ of two matrices $A, B \in \mathbb{C}^{n \times n}$ is computed via

$$c_{i,j} = \sum_{k=1}^{n} a_{i,k} b_{k,j}, \quad i, j = 1, \ldots, n$$

In the above example,

$$c_{11} = a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} = 45$$
$$c_{12} = a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32} = 25$$

This standard multiplication of two matrices can be easily translated to a code,

Algorithm 7 MM (Matrix multiplication)

input: $A = (A_{i,j})_{i,j=1}^{n,m} \in \mathbb{C}^{n\times m}$ and $B = (B_{i,j})_{i,j=1}^{m,q} \in \mathbb{C}^{m\times q}$.
output: $C \in \mathbb{C}^{n\times q}$ is a new matrix such that $AB = C$.

```
1: for i = 1 to n do
2:   for j = 1 to q do
3:     sum := 0
4:     for k = 1 to m do
5:       sum := sum + A_{i,k} × B_{k,j}
6:     end for
7:     C_{i,j} := sum
8:   end for
9:   C := C
10: end for
```

This method has a computational cost of $C_{\mathbf{MMStd}}(n) = \Theta(n^3)$ as $n \to \infty$. Since n^2 entries are to be computed any algorithm has cost $\Omega(n^2)$ as $n \to \infty$.

But there are other algorithms for matrix multiplication, the Strassen algorithm. It is based on standard matrix-matrix multiplication. To clarify, consider the multiplication of the following 2×2 matrices,

$$\underbrace{\begin{pmatrix} a & b \\ c & d \end{pmatrix}}_{A_{2\times 2}} \times \underbrace{\begin{pmatrix} e & f \\ g & h \end{pmatrix}}_{B_{2\times 2}} = \underbrace{\begin{pmatrix} ae+bg & af+bh \\ ce+dg & cf+dh \end{pmatrix}}_{C_{2\times 2}}$$

Assuming, just for convenience, that $n = 2^k$ with some $k \in \mathbb{N}$ the same logic can be applied if a, b, c, d, e, f, g, h were matrices,

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

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

where, $A, B, C \in \mathbb{C}^{2^k\times 2^k}$, $\quad A_{ij}, B_{ij}, C_{ij} \in \mathbb{C}^{2^{k-1}\times 2^{k-1}}$.

Example1:

$$\underbrace{\begin{pmatrix} 3 & 3 & 7 & 1 & 6 & 7 & 6 & 7 \\ 1 & 1 & 7 & 2 & 2 & 6 & 6 & 4 \\ 7 & 6 & 5 & 5 & 5 & 4 & 5 & 5 \\ 7 & 3 & 1 & 6 & 2 & 4 & 3 & 5 \\ 4 & 2 & 2 & 5 & 3 & 4 & 6 & 2 \\ 4 & 3 & 3 & 4 & 5 & 3 & 4 & 3 \\ 3 & 1 & 6 & 4 & 6 & 4 & 3 & 4 \\ 7 & 1 & 1 & 3 & 1 & 4 & 7 & 2 \end{pmatrix}}_{A} \times \underbrace{\begin{pmatrix} 5 & 3 & 3 & 3 & 3 & 3 & 5 & 1 \\ 1 & 1 & 2 & 3 & 3 & 4 & 4 & 4 \\ 2 & 5 & 4 & 1 & 2 & 4 & 1 & 3 \\ 1 & 5 & 4 & 2 & 3 & 2 & 2 & 4 \\ 2 & 3 & 2 & 5 & 5 & 2 & 2 & 4 \\ 3 & 1 & 1 & 1 & 3 & 5 & 4 & 5 \\ 2 & 2 & 2 & 5 & 3 & 1 & 1 & 5 \\ 5 & 3 & 2 & 4 & 2 & 5 & 4 & 2 \end{pmatrix}}_{B}$$

$$= \begin{pmatrix} \underbrace{\begin{pmatrix} 3 & 3 & 7 & 1 \\ 1 & 1 & 7 & 2 \\ 7 & 6 & 5 & 5 \\ 7 & 3 & 1 & 6 \end{pmatrix}}_{A_{11}} & \underbrace{\begin{pmatrix} 6 & 7 & 6 & 7 \\ 2 & 6 & 6 & 4 \\ 5 & 4 & 5 & 5 \\ 2 & 4 & 3 & 5 \end{pmatrix}}_{A_{12}} \\ \underbrace{\begin{pmatrix} 4 & 2 & 2 & 5 \\ 4 & 3 & 3 & 4 \\ 3 & 1 & 6 & 4 \\ 7 & 1 & 1 & 3 \end{pmatrix}}_{A_{21}} & \underbrace{\begin{pmatrix} 3 & 4 & 6 & 2 \\ 5 & 3 & 4 & 3 \\ 6 & 4 & 3 & 4 \\ 1 & 4 & 7 & 2 \end{pmatrix}}_{A_{22}} \end{pmatrix} \times \begin{pmatrix} \underbrace{\begin{pmatrix} 5 & 3 & 3 & 3 \\ 1 & 1 & 2 & 3 \\ 2 & 5 & 4 & 1 \\ 1 & 5 & 4 & 2 \end{pmatrix}}_{B_{11}} & \underbrace{\begin{pmatrix} 3 & 3 & 5 & 1 \\ 3 & 4 & 4 & 4 \\ 2 & 4 & 1 & 3 \\ 3 & 2 & 2 & 4 \end{pmatrix}}_{B_{12}} \\ \underbrace{\begin{pmatrix} 2 & 3 & 2 & 5 \\ 3 & 1 & 1 & 1 \\ 2 & 2 & 2 & 5 \\ 5 & 3 & 2 & 4 \end{pmatrix}}_{B_{21}} & \underbrace{\begin{pmatrix} 5 & 2 & 2 & 4 \\ 3 & 5 & 4 & 5 \\ 3 & 1 & 1 & 5 \\ 2 & 5 & 4 & 2 \end{pmatrix}}_{B_{22}} \end{pmatrix}$$

$$= \underbrace{\begin{pmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{pmatrix}}_{C}$$

But $A_{11}, A_{12}, A_{21}, A_{22}, B_{11}, B_{12}, B_{21}, B_{22}$ are still matrices that can also be partitioned. For example to calculate $A_{11}B_{11}$ we split A_{11} and B_{11} into 4 parts each,

$$\underbrace{\begin{pmatrix} 3 & 3 & 7 & 1 \\ 1 & 1 & 7 & 2 \\ 7 & 6 & 5 & 5 \\ 7 & 3 & 1 & 6 \end{pmatrix}}_{A_{11}} \times \underbrace{\begin{pmatrix} 5 & 3 & 3 & 3 \\ 1 & 1 & 2 & 3 \\ 2 & 5 & 4 & 1 \\ 1 & 5 & 4 & 2 \end{pmatrix}}_{B_{11}} = \begin{pmatrix} \underbrace{\begin{pmatrix} 3 & 3 \\ 1 & 1 \end{pmatrix}}_{a_{11}} & \underbrace{\begin{pmatrix} 7 & 1 \\ 7 & 2 \end{pmatrix}}_{a_{12}} \\ \underbrace{\begin{pmatrix} 7 & 6 \\ 7 & 3 \end{pmatrix}}_{a_{21}} & \underbrace{\begin{pmatrix} 5 & 5 \\ 1 & 6 \end{pmatrix}}_{a_{22}} \end{pmatrix} \times \begin{pmatrix} \underbrace{\begin{pmatrix} 5 & 3 \\ 1 & 1 \end{pmatrix}}_{b_{11}} & \underbrace{\begin{pmatrix} 3 & 3 \\ 2 & 3 \end{pmatrix}}_{b_{12}} \\ \underbrace{\begin{pmatrix} 2 & 5 \\ 1 & 5 \end{pmatrix}}_{b_{21}} & \underbrace{\begin{pmatrix} 4 & 1 \\ 4 & 2 \end{pmatrix}}_{b_{22}} \end{pmatrix}$$

$$= \begin{pmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{pmatrix}$$

Similarly for the other matrices. This is called divide and conquer algorithm. In this algorithm, the multiplication of two matrices of size $n \times n$ can be obtained by splitting the problem into smaller sub-problems consisting of the multiplication of matrices of size $\frac{n}{2} \times \frac{n}{2}$. And to multiply the $\frac{n}{2} \times \frac{n}{2}$ matrices, we split them into $\frac{n}{4} \times \frac{n}{4}$. The splitting process continues until the size of the matrices become 2×2 at which stage we perform standard matrix multiplication.

For the time complexity, the computation of C this way involves eight multiplications of $2^{k-1} \times 2^{k-1} = \frac{n}{2} \times \frac{n}{2}$-matrices. Using recursion, the cost of an algorithms based on this splitting satisfies

$$C_{\textbf{MMSplit}}(n) = 8C_{\textbf{MMSplit}}(\tfrac{n}{2}) + n^2$$

where the n^2 contribution comes from the 4 additions of $\frac{n}{2} \times \frac{n}{2}$-matrices. One can show that this leads to a cost of $\Theta(n^3)$ as $n \to \infty$, and the algorithm is not better than the standard one!

However, referring to the 8×8 example above, after splitting the matrices, the following computations were required,

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

For this, we had to calculate the 8 products as well as the sums. This computation can instead be modified in order to reduce the required matrix multiplication,

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

Thus, defining the **seven** $2^{k-1} \times 2^{k-1}$-matrices

$$\begin{aligned} P_1 &:= (A_{11} + A_{22})(B_{11} + B_{22}), \\ P_2 &:= (A_{21} + A_{22})B_{11}, \\ P_3 &:= A_{11}(B_{12} - B_{22}), \\ P_4 &:= A_{22}(B_{21} - B_{11}), \\ P_5 &:= (A_{11} + A_{12})B_{22}, \\ P_6 &:= (A_{21} - A_{11})(B_{11} + B_{12}), \\ P_7 &:= (A_{12} - A_{22})(B_{21} + B_{22}) \end{aligned} \tag{14.1}$$

we get,

$$\begin{aligned} C_{11} &= P_1 + P_4 - P_5 + P_7, \\ C_{12} &= P_3 + P_5, \\ C_{21} &= P_2 + P_4, \\ C_{22} &= P_1 + P_3 - P_2 + P_6. \end{aligned} \tag{14.2}$$

To re-emphasise, with this modification 7 products instead of 8 are required. Using this recursively as in Algorithm 8 we obtain that the cost satisfies

$$C_{\mathbf{MMStrassen}}(n) = 7^{k+1} - 6 \cdot 4^k, \quad n = 2^k,\ k \in \mathbb{N}, \tag{14.3}$$

and since $\log_2(7) \approx 2.807 < 3$ we have got an algorithm that asymptotically is of lower cost $\Theta(n^{\log_2(7)})$ as $n \to \infty$ than the previously presented algorithms.

Noting that $2^k = n$. Taking log of both sides $\longrightarrow \log_2 n = k$ means that

$$7^{k+1} = 7^{(\log_2 n)+1} = 7.7^{(\log_2 n)} = 7.n^{\log_2(7)}$$

Similarly,

$$6.4^k = 6.2^{2k} = 6.2^{2\log_2 n} = 6.2^{\log_2 n^2} = 6.n^2$$

This means that (14.3) can be equivalently written as,

$$C_{\mathbf{MMStrassen}}(n) = 7n^{\log_2(7)} - 6n^2 \tag{14.4}$$

Let us briefly prove (14.3).

Proof. The recursive definition of the Strassen multiplication invites for an induction proof. For $k = 0$ there is only one multiplication of scalars, and formula (14.3) indeed yields one.
Assume now that (14.3) is true for $k-1$. As there are 18 additions of $\frac{n}{2} \times \frac{n}{2}$-matrices in (14.1) and (14.2) we obtain the formula

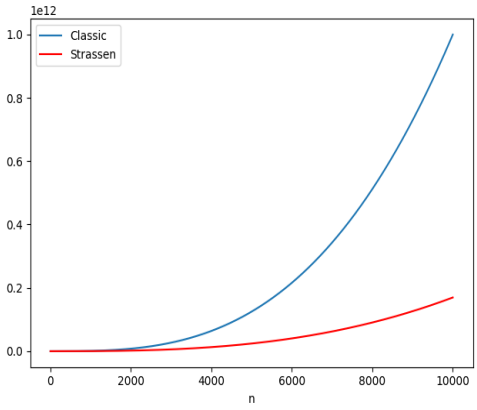
$$C_{\mathbf{MMStrassen}}(2^k) = 7C_{\mathbf{MMStrassen}}(2^{k-1}) + 18(2^{k-1})^2,$$

which yields, using the induction hypothesiss

$$\begin{aligned} &= 7(7^k - 6 \times 4^{k-1}) + 18 \times 4^{k-1} \\ &= 7^{k+1} - 24 \times 4^{k-1} \\ &= 7^{k+1} - 6 \times 4^k. \end{aligned}$$

□

Although Strassen method is asymptotically better than the standard multiplication method, the size of the matrix, n has to be quite large to get considerable reduction in computational complexity. It is not a huge improvement! Also in Strassen algorithm, standard multiplication is used when $n = 2$ which is not necessarily a good choice. In practice, you have to find the size of the matrix from when Strassen algorithm starts to be faster than the standard method. This is because, even if standard method is $O(n^3)$, it is faster for small values of n. Also note that the critical n value depends on the implementation and the hardware.



To deal with the case that n is no power of 2 one may pick $k \in \mathbb{N}$ such that $2^k \geq n > 2^{k-1}$ and then define

$$\tilde{A} = \begin{pmatrix} A & 0 \\ 0 & 0 \end{pmatrix} \in \mathbb{C}^{2^k \times 2^k}$$

by adding some blocks containing zeros, similarly with $\tilde{B}$. It turns out that the upper left $n \times n$ block of $\tilde{C} := \tilde{A}\tilde{B}$ then contains the desired product $C = AB$. Applying the Strassen multiplication to $\tilde{A}$ and $\tilde{B}$ involves a cost that is $\Theta((2^k)^{\log_2(7)})$ as $k \to \infty$, but since $2^k \leq 2n$ by the choice of k in dependence of n this is $\Theta(n^{\log_2(7)})$ as $n \to \infty$. We may summarise the findings in

Theorem 14.1. *Using the Strassen multiplication one can construct an algorithm for computing the product of $n \times n$-matrices, $n \in \mathbb{N}$, with computational cost*

$$C(n) = \Theta(n^{\log_2(7)}) \quad \text{as } n \to \infty.$$

An algorithm developed by Coppersmith and Winograd (1990) scales even better as it has a cost of about $\Theta(n^{2.376})$ but it is not (yet?) practical as its advantage only becomes perceptible for such big n that the corresponding matrices are just too big for even the most modern supercomputers. Any algorithm will have a cost $\Omega(n^2)$ as n^2 is the number of elements to be computed.

What is this studying of the exponent useful for? Well, if one can multiply $n \times n$-matrices with an asymptotic cost of $O(n^\alpha)$ as $n \to \infty$, $\alpha \geq 2$, then it is also possible to invert regular $n \times n$ matrices with cost $O(n^\alpha)$. For the proof one may use the Schur complement $S = D_{22} - D_{12}^* D_{11}^{-1} D_{12}$ of a Hermitian matrix

$$D = \begin{pmatrix} D_{11} & D_{12} \\ D_{12}^* & D_{22} \end{pmatrix}$$

and proceed recursively. The assertion on the cost then follows similarly as for the Strassen multiplication which is why the proof is omitted.

Algorithm 8 MMStrassen (Strassen matrix-matrix multiplication)

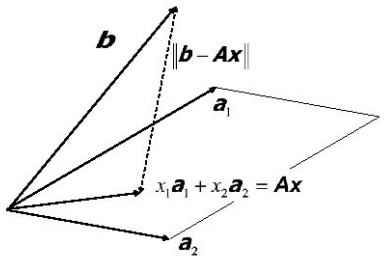
input: $A, B \in \mathbb{C}^{n \times n}$ with $n = 2^k$ for some $k \in \mathbb{N}$.
output: $C = AB \in \mathbb{C}^{n \times n}$.

1: **if** $n = 1$ **then**
2: return $C := AB$
3: **else**
4: compute $P_1, \ldots, P_7$ as defined in (14.1) using recursion for the matrix-matrix products
5: compute $C_{11}, C_{12}, C_{21}, C_{22}$ according to (14.2)
6: return C
7: **end if**

Lecture 15

Least Squares Problems

A SLE $Ax = b$ can be solved if and only if $\in b \in \text{range}(A)$ (which is also know as the column space of A). Although there is no exact solution when $b \notin \text{range}(A)$, an estimate x that makes Ax as close as possible to b can be found. This is called the least square solution.



Column and row spaces A column space (or range) of matrix A is the space that is spanned by As columns. Similarly, a row space is the space spanned by As rows. In particular, if $A \in \mathbb{C}^{m\times n}$, with column vectors $v_1, v_2, \ldots, v_n$, then a linear combination of these vectors is any vector of the form,

$$c_1v_1 + c_2v_2 + \cdots + c_nv_n$$

where $c_1, c_2, \ldots, c_n$ are scalars. The set of all possible linear combinations of $v_1, v_2, \ldots, v_n$ is called the range(A). This means that the range(A) is the span of the vectors $v_1, v_2, \ldots, v_n$.
Any linear combination of the column vectors of a matrix A can be written as the product of A with a column vector,

$$A\begin{pmatrix} c_1 \\ c_2 \\ \cdots \\ c_n \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}\begin{pmatrix} c_1 \\ c_2 \\ \cdots \\ c_n \end{pmatrix} = c_1\underbrace{\begin{pmatrix} a_{11} \\ a_{21} \\ \cdots \\ a_{m1} \end{pmatrix}}_{v_1} + c_2\underbrace{\begin{pmatrix} a_{12} \\ a_{22} \\ \cdots \\ a_{m2} \end{pmatrix}}_{v_2} + \cdots + c_n\underbrace{\begin{pmatrix} a_{1n} \\ a_{2n} \\ \cdots \\ a_{mn} \end{pmatrix}}_{v_n}$$

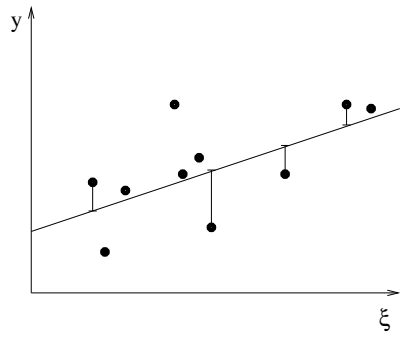
Definition 15.1. *Given a matrix $A \in \mathbb{R}^{m\times n}$ and a vector $b \in \mathbb{R}^m$, the least squares problem*

LSQ *consists of minimising the function*

$$g : \mathbb{R}^n \to \mathbb{R}, \quad g(x) = \frac{1}{2}\|Ax - b\|_2^2.$$

Example: Recall the linear regression problem. Given points $(\xi_i, y_i)_{i=1}^m$ find a linear function $\xi \to x_1 + x_2\xi$ such that

$$g(x) = \frac{1}{2}\sum_{i=1}^{m} \left(x_1 + x_2\xi_i - y_i\right)^2 \quad \text{is minimal.}$$



In this case,

$$A = \begin{pmatrix} 1 & \xi_1 \\ \vdots & \vdots \\ 1 & \xi_m \end{pmatrix}, \quad b = \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix}.$$

Example: Solve,

$$\begin{aligned} x_1 + 2x_2 &= 3 \\ x_1 + 3x_2 &= 5 \\ x_1 + 4x_2 &= 5 \\ x_1 + 5x_2 &= 6 \end{aligned} \quad \Longrightarrow \quad \underbrace{\begin{pmatrix} 1 & 2 \\ 1 & 3 \\ 1 & 4 \\ 1 & 5 \end{pmatrix}}_{A} \underbrace{\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}}_{x} = \underbrace{\begin{pmatrix} 3 \\ 2 \\ 6 \\ 5 \end{pmatrix}}_{b}$$

which can be written as,

$$x_1 \underbrace{\begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}}_{a_1} + x_2 \underbrace{\begin{pmatrix} 2 \\ 3 \\ 4 \\ 5 \end{pmatrix}}_{a_2} = \underbrace{\begin{pmatrix} 3 \\ 2 \\ 6 \\ 5 \end{pmatrix}}_{y}$$

Theorem 15.1. ([7.1], *$x \in \mathbb{R}^n$ solves the least squares problem if and only if $Ax - b \perp \text{range}(A)$, which is the case if and only if the normal equation*

$$A^T Ax = A^T b \tag{7.1}$$

is satisfied.

Proof. If x minimises g then for all $y \in \mathbb{R}^n$

$$\begin{aligned}
0 &= \frac{d}{d\varepsilon} g(x+\varepsilon y)\big|_{\varepsilon=0} \\
&= \frac{d}{d\varepsilon}\Big(\frac{1}{2}\langle Ax+\varepsilon Ay-b, Ax+\varepsilon Ay-b\rangle\Big)\Big|_{\varepsilon=0} \\
&= \frac{d}{d\varepsilon}\Big(\frac{1}{2}\langle Ax-b, Ax-b\rangle + \frac{1}{2}\varepsilon\big(\langle Ay, Ax-b\rangle + \langle Ax-b, Ay\rangle\big) + \frac{1}{2}\varepsilon^2\langle Ay, Ay\rangle\Big)\Big|_{\varepsilon=0} \\
&= \frac{d}{d\varepsilon}\Big(\frac{1}{2}\|Ax-b\|_2 + \varepsilon\langle Ay, Ax-b\rangle + \frac{1}{2}\varepsilon^2\|Ay\|_2\Big)\Big|_{\varepsilon=0} \\
&= \langle Ay, Ax-b\rangle
\end{aligned}$$

which means that $Ax-b \perp \text{range}(A)$. The other way round, if $Ax-b \perp \text{range}(A)$ then $\langle Ax-b, Ay-Ax\rangle = 0$ for all $y \in \mathbb{R}^n$, hence with Pythagoras

$$2g(y) = \|Ay-b\|_2^2 = \|Ay-Ax\|_2^2 + \|Ax-b\|_2^2 \geq \|Ax-b\|_2^2 = 2g(x).$$

For the second assertion we use that $Ax-b \perp \text{range}(A) \Leftrightarrow Ax-b \perp a_i$ where the a_i, $i=1,\dots,m$, are the column vectors of A. But this is equivalent to

$$\big(\langle a_i, Ax\rangle\big)_{i=1}^m = \big(\langle a_i, b\rangle\big)_{i=1}^m \quad \Leftrightarrow \quad (7.1)$$

□

Example: In the linear regression problem the normal equation is a 2×2 system where

$$A^TA = \begin{pmatrix} m & \sum_{i=1}^m \xi_i \\ \sum_{i=1}^m \xi_i & \sum_{i=1}^m \xi_i^2 \end{pmatrix}, \quad A^Tb = \begin{pmatrix} \sum_{i=1}^m y_i \\ \sum_{i=1}^m \xi_i y_i \end{pmatrix}.$$

Example:

$$A = \begin{pmatrix} 1 & 2 \\ 1 & 3 \\ 1 & 4 \\ 1 & 5 \end{pmatrix}, \qquad b = \begin{pmatrix} 3 \\ 2 \\ 6 \\ 5 \end{pmatrix}$$

$$A^T\begin{pmatrix} 1 & 1 & 1 & 1 \\ 2 & 3 & 4 & 5 \end{pmatrix} \implies A^TA = \begin{pmatrix} 4 & 14 \\ 14 & 54 \end{pmatrix}, \quad A^Tb = \begin{pmatrix} 16 \\ 61 \end{pmatrix}$$

Thus,

$$A^TAx = A^Tb$$
$$\begin{pmatrix} 4 & 14 \\ 14 & 54 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 16 \\ 61 \end{pmatrix}$$

Which can be solved using LU factorisation as follows,

$$A^TA = LU$$
$$\begin{pmatrix} 4 & 14 \\ 14 & 54 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 3.5 & 1 \end{pmatrix}\begin{pmatrix} 4 & 14 \\ 0 & 5 \end{pmatrix}$$

Then solve $Ly = A^T b$ using forward substitution,

$$\begin{pmatrix} 1 & 0 \\ 3.5 & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} 16 \\ 61 \end{pmatrix} \implies y_1 = 16,\ y_2 = 5$$

Finally solve $Ux = y$,

$$\begin{pmatrix} 4 & 14 \\ 0 & 5 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 16 \\ 5 \end{pmatrix} \implies x_1 = 0.5,\ x_2 = 1$$

which yields,

$$Ax = \begin{pmatrix} 2.5 \\ 3.5 \\ 4.5 \\ 5.5 \end{pmatrix}, \quad b - Ax = b_\perp = \begin{pmatrix} -0.5 \\ 1.5 \\ -1.5 \\ 0.5 \end{pmatrix}$$

It is certainly possible to use (7.1) to solve LSQ, for example one could use Cholesky since $A^T A$ is positive definite provided that A has full rank. But, as we will see later on, we have

$$\kappa_2(A^T A) = (\kappa_2(A))^2$$

for the condition number, and even the condition number $\kappa_2(A)$ (which is not yet defined for $m \neq n$) can be big in practical applications. There are better approaches, based on the QR factorisation (later) or on the singular value decomposition that we consider next.

Singular Value Decomposition (SVD) [2.3]

SVD gives insight into the least square problems. The least square is,

$$\underbrace{\min}_{x} \frac{1}{2}\|Ax - b\|_2^2$$

We have seen that the solution in this case is given by,

$$\begin{aligned} & A^T Ax = A^T b \\ \implies\ & x = (A^T A)^{-1} A^T b \end{aligned} \tag{15.1}$$

Which means that the estimated measured values are,

$$\begin{aligned} \hat{b} &= Ax \\ &= A(A^T A)^{-1} A^T b \\ &= P_{A_\parallel} b \end{aligned}$$

which is the projection of b into the column space of A. The error is then,

$$\begin{aligned} \text{error} &= b - \hat{b} \\ &= b - A(A^T A)^{-1} A^T b \\ &= P_{A_\perp} b \end{aligned}$$

where $P_{A_\perp}$ is the orthogonal projection of b onto the column space of A.

SVD can be used to factorise the matrix A. Any arbitrary matrix $A \in \mathbb{R}^{m\times n}$ factors into,

$$A = U\Sigma V^T$$

where $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{n\times n}$ are orthogonal matrices and $\Sigma \in \mathbb{R}^{m\times n}$ is a diagonal matrix with entries $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_p \geq 0$ where $p = \min(m, n)$. Those values σ_i are called singular values. Physically this means a rotation, a stretch and a second rotation. Note that in this decomposition we have two different matrices. But these matrices are in fact orthogonal matrices. Also this decomposition can be done for rectangular matrices, however eigenvalues really worked for square matrices. However, now we have input matrix and an output matrix. In those spaces m and n can have different dimension. So by allowing two separate basis we can factorise rectangular matrices using orthogonal factors and a diagonal matrix consists of singular values instead of eigenvalues. U is called the left singular vector while V is called the right singular vector.
Denoting by u_i, $i = 1 = 1, \ldots, m$, the column vectors of U and by v_i, $i = 1, \ldots, n$, the column vectors of V the SVD says that $Av_i = \sigma_i u_i$, $i = 1, \ldots, p$.

SVD: Here we outline the procedure for obtaining the U and V matrices, for an arbitrary matrix $\mathbf{X} \in \mathbb{R}^{m \times n}$. Let us look at $\mathbf{X}^T\mathbf{X}$,

$$\mathbf{X}^T\mathbf{X} = V\Sigma^T(U^TU)\Sigma V^T \tag{15.2}$$
$$= V(\Sigma^T\Sigma)V^T. \tag{15.3}$$

This is the usual eigenvector decomposition. But here the matrix is $\mathbf{X}^T\mathbf{X}$. Note that although $\mathbf{X}$ is rectangular and completely general, its diagonalisation is not possible. However the transition to $\mathbf{X}^T\mathbf{X}$ provided a positive semidefinite and symmetric matrix ($\mathbf{X}^T\mathbf{X}$) which has orthogonal eigenvectors and positive eigenvalues. Also note that ($\Sigma^T\Sigma$) are the square of the singular values, i.e λ for $\mathbf{X}^T\mathbf{X}$ are the σ^2 for $\mathbf{X}$. This gives us V and Σ. However, U disappears here because $U^TU = I$ where I is the identity. How can we get hold of U?

Let us look at $\mathbf{X}\mathbf{X}^T$,

$$\mathbf{X}\mathbf{X}^T = U\Sigma(V^TV)\Sigma U^T \tag{15.4}$$
$$= U(\Sigma^T\Sigma)U^T. \tag{15.5}$$

This clearly shows that U is the eigenvector matrix for $\mathbf{X}\mathbf{X}^T$. So $\mathbf{X}^T\mathbf{X}$ and $\mathbf{X}\mathbf{X}^T$ have the same eigenvalues ((AB) has the same eigenvalues as (BA)). However $\mathbf{X}^T\mathbf{X}$ has eigenvectors V while $\mathbf{X}\mathbf{X}^T$ has eigenvectors U and those are the U and the V in the singular value decomposition.

Example:

$$X = \begin{bmatrix} 3 & 1 & 1 \\ 1 & 3 & -1 \end{bmatrix}$$

$$XX^T = \frac{1}{\sqrt{5}}\begin{bmatrix} 11 & 5 \\ 5 & 11 \end{bmatrix} \qquad X^TX = \begin{bmatrix} 10 & 6 & 2 \\ 6 & 10 & -2 \\ 2 & -2 & 2 \end{bmatrix}$$

Eigenvalues and eigenvectors of XX^T,

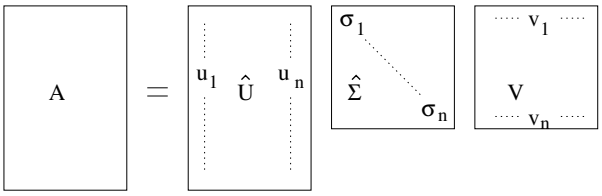
$$u_1 = \begin{bmatrix} -0.7071 \\ 0.7071 \end{bmatrix} \quad u_2 = \begin{bmatrix} 0.7071 \\ 0.7071 \end{bmatrix} \quad \lambda_1 = 6,\ \lambda_2 = 16$$

Eigenvalues and eigenvectors of X^TX,

$$v_1 = \begin{bmatrix} 0.4082 \\ -0.4082 \\ -0.8165 \end{bmatrix} \quad v_2 = \begin{bmatrix} -0.5774 \\ 0.5774 \\ -0.5774 \end{bmatrix} \quad v_3 = \begin{bmatrix} 0.7071 \\ 0.7071 \\ 0 \end{bmatrix} \quad \lambda_1 = 0,\ \lambda_2 = 6,\ \lambda_3 = 16.$$

$$X = U\Sigma V$$
$$= \begin{bmatrix} 0.7071 & -0.7071 \\ 0.7071 & 0.7071 \end{bmatrix}\begin{bmatrix} 4 & 0 & 0 \\ 0 & 2.45 & 0 \end{bmatrix}\begin{bmatrix} 0.7071 & 0.7071 & 0 \\ -0.5774 & 0.5774 & -0.5774 \\ 0.4082 & -0.4082 & -0.8165 \end{bmatrix}$$

In the case $m \geq n(= p)$ which is of most interest to us the reduced singular value decomposition is defined by dropping the last $m - n$ columns of U and the last $m - n$ rows of Σ. Defining $\hat{U} := (u_1, \ldots, u_n) \in \mathbb{R}^{m \times n}$ and $\hat{\Sigma} := \mathrm{diag}(\sigma_1, \ldots, \sigma_n) \in \mathbb{R}^{n \times n}$



the matrix A can be expressed as

$$A = \hat{U}\hat{\Sigma}V^T. \tag{15.6}$$

Least square solution can then be obtained by substituting (15.6) into (15.1) which yields,

$$\begin{aligned} x &= (A^TA)^{-1}A^Tb \\ &= (V\hat{\Sigma}^T\hat{U}^T\hat{U}\hat{\Sigma}V^T)^{-1}V\hat{\Sigma}^T\hat{U}^Tb \\ &= (V\hat{\Sigma}^2V^T)^{-1}V\hat{\Sigma}^T\hat{U}^Tb \\ &= V\hat{\Sigma}^{-2}V^TV\hat{\Sigma}^T\hat{U}^Tb \\ &= V\hat{\Sigma}^{-1}\hat{U}^Tb \\ &= \sum_{i=1}^{p} \frac{1}{\sigma_i} v_i(\hat{u}_i^T)b \end{aligned}$$

Note that $(A^TA)^{-1}A^T = A^\dagger$ is referred to as the pseudo inverse of A.

Note that multiplying $A^\dagger$ by A on the right yields ,

$$\{(A^TA)^{-1}A^T\}A = (A^TA)^{-1}(A^TA) = I$$

Similarly, For the SVD, we have,

$$V\hat{\Sigma}^{-1}\hat{U}^T\hat{U}\hat{\Sigma}V^T = I$$

Exercise: Using the SVD for the matrix A, evaluate the estimated measured values, $\hat{b}$.

Exercise: Using the SVD for the matrix A, evaluate the error in the estimated measured values.

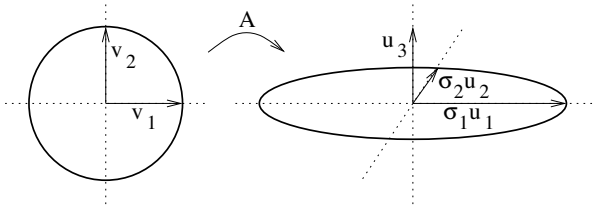
The SVD is a very powerful decomposition for analytical purposes as it provides much insight into the properties of the linear map associated with the matrix. The rank is the number of non-vanishing singular values, i.e., the biggest integer $r \leq p$ such that $\sigma_r > 0$ but $\sigma_{r+1} = 0$ (if $r + 1 \leq p$). Moreover, the image of A is spanned by the first r left singular vectors,

$$\mathrm{range}(A) = \mathrm{span}\{u_1, \ldots, u_r\}$$

and the kernel is

$$\mathrm{kernel}(A) = \mathrm{span}\{v_{r+1}, \ldots, v_p\}.$$

In addition to this algebraic information the SVD also reveals some geometrical information. Recalling the identities $Av_i = u_i\sigma_i$, $i = 1, \ldots, p$, we see that the unit sphere in $\mathbb{R}^p$ containing the vectors v_i is mapped to an ellipsoid which has semi-axes in the directions of the u_i that have the extensions σ_i.



Example: Images can be compressed using the SVD. This will be demonstrated through an example on Python.