Introduction to the concept of Algorithm

F. Zanlungo

January 13, 2016

We may informally define an algorithm as a list of operations that leads to the solution of a problem.

1 A rough sorting algorithm

Let us for example assume that we want to sort the array (vector)

$$\mathbf{x} = \{1, 3, -7, 5, 2\} \tag{1}$$

in growing order, i.e. rearrange it in such a way that $x_i \leq x_j$ if $i < j$.

Without worrying about the efficiency of the method, we may first look for a minimum over the vector's components, and put it in the first component (by swapping two components), and then proceed to find the minimum between the components x_i with $i > 1$, swapping again to have this minimum in the second component, and so on. For such a short array our brain finds almost automatically the solution

$$\mathbf{x}' = \{-7, 1, 2, 3, 5\}. \tag{2}$$

But if we want to implement such an algorithm on a computer, we have to define properly its operation list[1].

We have first to look for a minimum over $\mathbf{x}$. We compare the first two components, and we define a temporary minimum as the lowest value of the two (in our case, MIN $= x_1 = 1$). We also keep track of the position at which we found the minimum, in our case $i_{\min} = 1$. We then compare the value of such a minimum with x_3, and set MIN to the minimum value between x_3 and the current value of MIN (MIN $= x_1 = -7$; $i_{\min} = 3$). If the two values are the same, we may keep the current value of i. We proceed this way until we arrive at the end of the vector.

[1] Although some programming languages may have predefined operators able to perform complex operations on whole arrays, we will specify the algorithms in such a way that they may be implemented also in languages that define only basic operations on array components.

This operation of "finding the minimum" needs 4 comparisons between numbers (one less then the size of the array), and possibly up to 5 assignments to MIN of the value of one of the components, and to $i_{\min}$ of its position.

When we find the minimum, we have to swap the first component with the one corresponding to the minimum, and obtain

$$\mathbf{x} = \{-7, 3, 1, 5, 2\}. \tag{3}$$

The second step is to find the minimum on the components 2-5, or the minimum of the sub-array

$$\mathbf{x} = \{3, 1, 5, 2\}. \tag{4}$$

This involves the same operations we have done before, but on a smaller array. At each step we reduce the size of the array we operate on of one unit, until eventually we deal with a single component, and our algorithm terminates. Due to the construction of our algorithm, we have that $x_i \leq x_j$ if $i < j$, as required.

We may formalise our algorithm as:

- *for* $i = 1, n-1$
  - *find* j *such that* $x_j = \min_{i,\ldots,n} x_i$
  - *swap the positions of* x_i *and* x_j

Furthermore, the "sub-algorithm" for finding a minimum between x_i and x_n is

- *define* $MIN = x_i$ *and* $i_{min} = i$
- *for* $j = i + 1, n$
  - *if*$(x_j < MIN)$ $\{MIN = x_j\}$, $i_{min} = j$

1.1 Algorithm analysis

Each search for a minimum in a array of size n involves $n-1$ comparisons and a maximum of $2n$ assignments. The whole algorithm involves thus

$$\sum_{i=1}^{n-1}(n-i) \tag{5}$$

comparisons, a maximum of

$$2\left(\sum_{i=1}^{n-1}(n-i+1)\right) \tag{6}$$

assignments and a maximum of $n-1$ swaps (each swap involves in general 3 assignments). To evaluate eq. (5) (and similarly for eq. (6)) we may set $k = n - i$ to change it to

$$\sum_{k=1}^{n-1} k \tag{7}$$

and use the following equation

$$\sum_{i=1}^{n} i = (n+1)(n)/2. \tag{8}$$

The equation is true if $n = 1$. By induction, if it is true for $n - 1$ we have

$$\sum_{i=1}^{n} i = n + \sum_{i=1}^{n-1} i = n + (n)(n-1)/2 = (2n + n^2 - n)/2 = n(n+1)/2. \tag{9}$$

1.2 Exercise

Write a fortran program that implements this simple algorithm.

2 Algorithms for computing the determinant

In this course we will focus on the analysis of algorithm complexity, which, roughly, means the analysis of the number of operations needed to perform the algorithm. Given the power of modern computers, it may seem surprising that we still have to worry about the number of operations that our algorithm performs, but an example from linear algebra may convince us of the importance of such an analysis.

2.1 Review of the determinant

Let us consider two vectors $\mathbf{a} = (a_1, a_2)$, $\mathbf{b} = (b_1, b_2)$. We say that they are linearly dependent if $\mathbf{b} = \alpha\mathbf{a}$ or

$$\frac{a_1}{b_1} = \alpha = \frac{a_2}{b_2} \Rightarrow a_1 b_2 = a_2 b_1. \tag{10}$$

If we arrange our vectors as the row of a matrix A, $a_1 = a_{1,1}$, $a_2 = a_{1,2}$, $b_1 = a_{2,1}$, $b_2 = a_{2,2}$, we have that linear dependence corresponds to

$$\det(A) = a_{1,1}a_{2,2} - a_{2,1}a_{1,2} = 0. \tag{11}$$

In order to define the determinant of a n by n matrix we may use the completely asymmetric Levi-Civita symbol

$$\varepsilon_{i_1,\ldots,i_n}. \tag{12}$$

The symbol is defined by the following properties

$$\varepsilon_{1,2,3,\ldots,n-1,n} = 1, \tag{13}$$

$$\varepsilon_{i_1,\ldots,i_j,\ldots,i_k,\ldots,i_n} = -\varepsilon_{i_1,\ldots,i_k,\ldots,i_j,\ldots,i_n}. \tag{14}$$

As a result of its asymmetry, the symbol assumes value 0 if any of its indexes repeat, and for all the other permutations of the indexes it gives the sign of the permutations.

It is easier to understand if we consider explicit values of n. For $n = 2$ we have

$$\varepsilon_{1,1} = \varepsilon_{2,2} = 0; \quad \varepsilon_{2,1} = -\varepsilon_{1,2} = -1. \tag{15}$$

For $n = 3$

$$\varepsilon_{1,1,1} = \varepsilon_{1,1,2} = \varepsilon_{1,1,3} = \varepsilon_{1,2,1} = \varepsilon_{1,3,1} = \varepsilon_{2,1,1} = \varepsilon_{3,1,1} = 0, \tag{16}$$

$$\varepsilon_{2,2,2} = \varepsilon_{2,2,1} = \varepsilon_{2,2,3} = \varepsilon_{2,1,2} = \varepsilon_{2,3,2} = \varepsilon_{1,2,2} = \varepsilon_{3,2,2} = 0, \tag{17}$$

$$\varepsilon_{3,3,3} = \varepsilon_{3,3,1} = \varepsilon_{3,3,2} = \varepsilon_{3,1,3} = \varepsilon_{3,2,3} = \varepsilon_{1,3,3} = \varepsilon_{2,3,3} = 0, \tag{18}$$

$$\varepsilon_{1,2,3} = \varepsilon_{3,2,1} = \varepsilon_{2,3,1} = 1, \tag{19}$$

$$\varepsilon_{1,3,2} = \varepsilon_{3,1,2} = \varepsilon_{2,1,3} = -1. \tag{20}$$

We may now define

$$\det(A) = \sum_{\substack{i_1=1,\ldots,n;\ldots;\\ i_j=1,\ldots,n;\ldots;\\ i_n=1,\ldots,n}} \varepsilon_{i_1,\ldots,i_j,\ldots,i_n} a_{1,i_1} \ldots a_{j,i_j} \ldots a_{n,i_n}. \tag{21}$$

It is straightforward to show that in the $n = 2$ case we obtain the determinant as defined in eq. (11). With this definition we can show easily the main determinant properties.

The determinant of a matrix C whose jth row is given by the sum of vectors $\mathbf{u}$ and $\mathbf{v}$ is the sum of the determinants of two matrices A and B whose jth row is given respectively by $\mathbf{u}$ and $\mathbf{v}$

$$\det(C) = \sum_{\substack{i_1=1,\ldots,n;\ldots;\\ i_j=1,\ldots,n;\ldots;\\ i_n=1,\ldots,n}} \varepsilon_{i_1,\ldots,i_j,\ldots,i_n} a_{1,i_1} \ldots (u_{i_j} + v_{i_j}) \ldots a_{n,i_n} = \det(A) + \det(B). \tag{22}$$

If we multiply a row by α, also the determinant is multiplied by α

$$\det(A') = \sum_{\substack{i_1=1,\ldots,n;\ldots;\\ i_j=1,\ldots,n;\ldots;\\ i_n=1,\ldots,n}} \varepsilon_{i_1,\ldots,i_j,\ldots,i_n} a_{1,i_1} \ldots (\alpha) a_{j,i_j} \ldots a_{n,i_n} = \alpha \det(A). \tag{23}$$

If we swap two rows, the determinant changes sign

$$\det(A') = \sum_{\substack{i_1=1,\dots,n;\dots;\\ i_j=1,\dots,n;\dots;\\ i_k=1,\dots,n;\dots;\\ i_n=1,\dots,n}} \varepsilon_{i_1,\dots,i_j,\dots,i_n} a_{1,i_1}\dots u_{i_j}\dots v_{i_k}\dots a_{n,i_n} =$$
$$- \sum_{\substack{i_1=1,\dots,n;\dots;\\ i_j=1,\dots,n;\dots;\\ \imath_k=1,\dots,n;\dots;\\ i_n=1,\dots,n}} \varepsilon_{i_1,\dots,i_j,\dots,i_n} a_{1,i_1}\dots u_{i_j}\dots v_{i_k}\dots a_{n,i_n} = -\det(A). \tag{24}$$

This implies that if a matrix has two equal rows, its determinant is zero (since we should change the sign of the determinant by swapping the rows, but the matrix is unchanged by the swap and thus its determinant has to be zero).

We may now see that, naming the rows of the matrix the vectors $\mathbf{a}^i$, if one of the rows (for instance the nth) is a linear combinations of the others,

$$A = (\mathbf{a}^1, \dots, \mathbf{a}^{n-1}, \sum_{i=1}^{n-1} \alpha_i \mathbf{a}^i)^T, \tag{25}$$

we have

$$\det(A) = \sum_{i=1}^{n-1} \alpha_i (\mathbf{a}^1, \dots, \mathbf{a}^{n-1}, \mathbf{a}^i)^T = 0, \tag{26}$$

since each term has a repetition of rows.

2.2 Determinant by minors

It is possible to show that[2] that the definition given in 21 of the determinant is equivalent to the one that uses minors, that we may now revise. Assuming A to be a $n \times n$ square matrix, we have

$$\det(A) = \sum_j (-)^{j+1} \det(C_{j,1}). \tag{27}$$

here, $C_{j,1}$ is the $n-1 \times n-1$ square matrix obtained by deleting the jth row and 1st column, for example if

$$A = \begin{pmatrix} a_{1,1} & a_{2,1} & \dots & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & \dots & a_{2,n} \\ a_{3,1} & a_{3,2} & \dots & \dots & a_{3,n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n,1} & a_{n,2} & \dots & \dots & a_{n,n} \end{pmatrix}, \tag{28}$$

[2]See for example chapter 6 in "Linear Algebra" by S. Lang. For a sketched proof, we may note that by using expressions such as $\varepsilon_{1,i,j} = \varepsilon_{i-1,j-1}$, $\varepsilon_{2,i,j} = -\varepsilon_{i,2,j}$, and $\varepsilon_{i,2,j} = \varepsilon_{i,j-1}$ if $i < 2$, $j > 2$; $\varepsilon_{i,2,j} = \varepsilon_{i-1,j}$ if $i > 2$, $j < 2$, etc., we may reduce the n by n determinant to the sum of n products of $n-1$ by $n-1$ determinants given by eq. (27).

we have

$$C_{1,1} = \begin{pmatrix} a_{2,2} & \dots & \dots & a_{2,n} \\ a_{3,2} & \dots & \dots & a_{3,n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{n,2} & \dots & \dots & a_{n,n} \end{pmatrix}, \qquad C_{21} = \begin{pmatrix} a_{1,2} & \dots & \dots & a_{1,n} \\ a_{3,2} & \dots & \dots & a_{3,n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{n,2} & \dots & \dots & a_{n,n} \end{pmatrix}. \tag{29}$$

For a 2×2 matrix

$$A = \begin{pmatrix} a_{1,1} & a_{2,1} \\ a_{2,1} & a_{2,2} \end{pmatrix} \tag{30}$$

we have $C_{1,1} = a_{2,2}$, $C_{2,1} = a_{1,2}$ and thus

$$\det(A) = (-)^2 a_{1,1} a_{2,2} + (-)^3 a_{2,1} a_{1,2} = a_{1,1} a_{2,2} - a_{2,1} a_{1,2} \tag{31}$$

as we expected.

The computation of a 2×2 determinant implies 2 multiplications. For a 3×3 determinant, we will compute 3 multiplications with 2×2 determinants, and thus we overall have $3! = 6$ multiplications. For a 4×4 determinant we will have 4! multiplications, and by induction $n!$ multiplications for a n square matrix.

As we will study in detail later, the factorial grows extremely fast, and even the world's most powerful computers will not be able to compute the determinant of a 30 by 30 matrix using algorithm that follows eq. (27) (see Table 1).

2.2.1 Example

Let us compute

$$\det \begin{pmatrix} 1 & 2 & 0 \\ 3 & 2 & 1 \\ 2 & 1 & 0 \end{pmatrix}$$

using eq. (27). We find

$$\det = (-)^2 (1) \det \begin{pmatrix} 2 & 1 \\ 1 & 0 \end{pmatrix} + (-)^3 (2) \det \begin{pmatrix} 3 & 1 \\ 2 & 0 \end{pmatrix} = -1 + 4 = 3.$$

2.2.2 Exercise

Write a fortran program that computes the factorial. Try to do it both by using a recursive function and by not using it. See also what happens if you use a integer or a real function to return the factorial.

2.2.3 Exercise

Write a fortran program that implements eq. (27).

2.3 Determinant by Gaussian Elimination

Let us first consider

$$\det \begin{pmatrix} a_{1,1} & a_{2,1} \\ 0 & a_{2,2} \end{pmatrix} = a_{1,1}a_{2,2}. \tag{32}$$

We may now use eq. (27) and obtain

$$\det \begin{pmatrix} a_{1,1} & a_{2,1} & a_{3,1} \\ 0 & a_{2,2} & a_{2,3} \\ 0 & 0 & a_{3,3} \end{pmatrix} = +a_{1,1}\left(a_{2,2}a_{3,3}\right) - 0 + 0 = a_{1,1}a_{2,2}a_{3,3}. \tag{33}$$

We have thus seen that for 2 and 3 upper triangular matrices (and obviously for unit matrices) the determinant is given by the product of the diagonal terms. This is true for any n, i.e. if U is an upper triangular matrix, we have

$$\det(U) = \prod_{i=1}^{n} u_{j,j}. \tag{34}$$

The proof by induction is straightforward. If we know that (34) is true for $n-1$, then

$$\det \begin{pmatrix} u_{1,1} & u_{2,1} & \dots & \dots & u_{1,n} \\ 0 & u_{2,2} & \dots & \dots & \vdots \\ \vdots & 0 & \ddots & \dots & \vdots \\ \vdots & \vdots & 0 & \ddots & \vdots \\ 0 & \dots & \dots & 0 & u_{n,n} \end{pmatrix} = u_{1,1}\det\left(C_{1,1}\right) + 0 = u_{1,1}\prod_{i=2}^{n} u_{j,j}. \tag{35}$$

But we know a method to reduce a matrix to an upper triangular form, Gaussian Elimination. Let us revise it briefly. The main idea is to use the fact that subtracting to a row of our matrix another row multiplied by a coefficient does not change the determinant (it may be shown by using eq. (22) and noticing that we obtain two terms, one equal to the original determinant and one equal to zero due to eq. (25)). We may thus use this subtraction to reduce in all rows but one (the "pivot row") an element of our choice to zero. By reducing whole columns to 0, we finally arrive to an upper triangular form without changing the value of the determinant.

Formally,
for $j = 1, \dots, n$

- *for* $l = j+1, \dots, n$
  - *compute the multiplying factor (or multiplicator)* $m_{l,j} = a_{l,j}/a_{j,j}$ *(where* $a_{j,j}$ *is called the* j*th pivot)*
  - *substitute* $a_{l,j} = a_{l,j} - m_{l,j}a_{j,j} = 0$

- *for* $i = j+1, \ldots, n$
  - *substitute* $a_{l,i} = a_{l,i} - m_{l,j} a_{j,i}$

There is only a possible problem, namely the fact that the pivot $a_{j,j}$ may be zero. In that case, we have to look for a non-zero $a_{l,j}$. If we find it, we may swap the rows j and l, remember that this operation changes the sign of the determinant, and proceed to compute the multiplicators. If no $a_{l,j} \neq 0$, it means that our column is already full of zeros, and we could proceed with the next j. But there is no need, since a triangular matrix with a 0 on the diagonal has $\det = 0$, and thus we have found our solution.

We may thus re-write the algorithm as
Initialise $n_{swaps} = 0$
for $j = 1, \ldots, n$

- *if*$(a_{j,j} = 0)$
  - *Look for* $a_{l,j} \neq 0$, $l > j$
    - *if*$(a_{l,j} \neq 0)$ *swap the* j *and* l *rows,* $n_{swaps} = n_{swaps} + 1$
    - *else* $\det = 0$, *EXIT*
  - *for* $l = j+1, \ldots, n$
    - *compute the multiplying factor* $m_{l,j} = a_{l,j}/a_{j,j}$
    - *substitute* $a_{l,j} = a_{l,j} - m_{l,j} a_{j,j} = 0$
    - *for* $i = j+1, \ldots, n$
      - *substitute* $a_{l,i} = a_{l,i} - m_{l,j} a_{j,i}$

compute $\det = (-)^{n_{swaps}} \prod_{i=1}^{n} a_{jj}$

2.3.1 Example

Let us compute

$$\det \begin{pmatrix} 1 & -1 & 4 & 3 \\ 0 & 0 & 1 & 4 \\ 3 & 4 & -1 & 2 \\ 4 & 3 & -2 & 0 \end{pmatrix}.$$

After computing

$$m_{2,1} = a_{2,1}/a_{1,1} = 0/1 = 0,$$
$$m_{3,1} = a_{3,1}/a_{1,1} = 3/1 = 3,$$
$$m_{4,1} = a_{4,1}/a_{1,1} = 4/1 = 4,$$

and $a'_{j,i} = a_{j,i} - m_{j,1}a_{1,i}$, we obtain

$$\begin{pmatrix} 1 & -1 & 4 & 3 \\ 0 & 0 & 1 & 4 \\ 0 & 7 & -13 & -7 \\ 0 & 7 & -18 & -12 \end{pmatrix}.$$

We have $a_{2,2} = 0$. We thus swap the 2nd and 3rd column, obtaining $n_{swaps} = 1$ and

$$\begin{pmatrix} 1 & -1 & 4 & 3 \\ 0 & 7 & -13 & -7 \\ 0 & 0 & 1 & 4 \\ 0 & 7 & -18 & -12 \end{pmatrix}.$$

We now compute the multiplicators

$$m_{3,2} = a_{3,2}/a_{2,2} = 0/7 = 0,$$

$$m_{4,2} = a_{4,2}/a_{2,2} = 7/7 = 1,$$

and substitute, for each row from $j = 3, \ldots, 4$, $a'_{j,i} = a_{j,i} - m_{j,2}a_{2,i}$, obtaining

$$\begin{pmatrix} 1 & -1 & 4 & 3 \\ 0 & 7 & -13 & -7 \\ 0 & 0 & 1 & 4 \\ 0 & 0 & -5 & -5 \end{pmatrix}.$$

Finally, we compute $m_{4,3} = a_{4,3}/a_{33} = -5/1 = -5$, and by $-5-(-5)(4) = 15$ we obtain

$$\begin{pmatrix} 1 & -1 & 4 & 3 \\ 0 & 7 & -13 & -7 \\ 0 & 0 & 1 & 4 \\ 0 & 0 & 0 & 15 \end{pmatrix},$$

and finally

$$\det = (-)^1(1)(7)(1)(15) = -105.$$

2.3.2 Exercise

Write a fortran program that computes the determinant using Gaussian Elimination. Proceed step by step by writing subroutines for multiplicator computation, row swap, subtraction, etc. Check that each subroutine works before proceeding to the next step.

n	$n^3/3$	$n^3/3 + n^2/2 + n/6$	$n!$
3	9	13	6
5	41.667	53	120
10	333.33	384	3628800
30	9000	9455	$2.65 \cdot 10^{32}$

Table 1: Comparison between n^3, $n^3/3 + n^2/2 + n/6$ and $n!$

2.4 Algorithm analysis

Many different operations are performed during the algorithm. By following the same logic of section 1.1, we may see that we have up to $n(n-1)/2$ comparisons to look for a non-zero pivot, which may lead to an equal number of swaps. Also the number of divisions to compute the multiplicators follows the same logic.

Let us compute the number of sums and multiplications used to compute the row subtractions. We operate on all rows, $j = 1, \ldots, n$, and for each j we subtract on all rows $l = j + 1, \ldots, n$ for all components $i = j + 1, \ldots, n$. The number of operations is thus

$$\sum_{j=1}^{n-1} (n-j)^2. \tag{36}$$

Let us call $k = n - j$. We have

$$\sum_{k=1}^{n-1} k^2. \tag{37}$$

We may then use

$$\sum_{i=1}^{n} i^2 = n^3/3 + n^2/2 + n/6, \tag{38}$$

a result that we prove below. We may thus say that the number of operations grows like n^3. We will formalise this concept in the following lectures, but you may understand what we mean by noticing that

$$\lim_{n\to\infty} (\alpha n^3 + \beta n^2 + \gamma n + \delta)/n^3 = \alpha, \tag{39}$$

so that when n is big enough we may forget about the contribution of lower terms (see Table 1). The improvement, for large n, on the algorithm using minors is obvious (again Table 1).

2.4.1 Proof of eq. (38)

The sum

$$S_n = \sum_{j=1}^{n} j^2, \tag{40}$$

involves n terms of order j^2 with $j \leq n$, so we may assume that it will behave as

$$S_n = \alpha n^3 + \beta n^2 + \gamma n + \delta. \tag{41}$$

We may compute explicitly $S_0 = 0$, $S_1 = 1$, $S_2 = 5$ and $S_3 = 14$, and then solve the system

$$\begin{cases} \delta = 0 \\ \alpha + \beta + \gamma + \delta = 1 \\ 8\alpha + 4\beta + 2\gamma + \delta = 5 \\ 27\alpha + 9\beta + 3\gamma + \delta = 14 \end{cases}, \tag{42}$$

whose solution is

$$\begin{cases} \delta = 0 \\ \gamma = 1/6 \\ \beta = 1/2 \\ \alpha = 1/3 \end{cases}. \tag{43}$$

We still have to show that eq. (38) holds for $n > 3$. We do it by induction showing that if $S_n = n^3/3 + n^2/2 + n/6$ then

$$S_{n+1} = S_n + (n+1)^2 = \frac{2n^3 + 3n^2 + n}{6} + \frac{6n^2 + 12n + 6}{6} = \frac{2n^3 + 9n^2 + 13n + 6}{6} = \frac{2(n+1)^3 + 3(n+1)^2 + 6(n+1)}{6} = (n+1)^3/3 + (n+1)^2/2 + (n+1)/6. \tag{44}$$

2.4.2 Exercise

Write a fortran program that solves the system (42) using Gaussian Elimination. (Hint: write the system in matrix form $A\mathbf{x} = \mathbf{b}$. Modify the program used for the determinant, but perform the subtraction also on the known terms $\mathbf{b}$. The solution is found using back substitution, i.e. starting from x_n (where $\mathbf{x}$ is the solution vector) you compute

$$x_i = \frac{1}{a_{i,i}} \left(b_i - \sum_{j=i+1}^{n} a_{i,j} x_j \right). \tag{45}$$

)