Foundations of numerical methods

Material to the course:

http://www.staff.city.ac.uk/c.f.m.faria/numerical.htm

I. Preliminaries

1. Motivation: solve problems which have no explicit/analyt solution

Examples:

- Solve the partial differential equation

$$i\frac{\partial}{\partial t}\psi(x,t) = \left[a\,e^{-x^2/\sigma^2} + E_0 x \sin t - \frac{1}{2}\frac{\partial^2}{\partial x^2}\right]\psi(x,t)$$

with $\psi(0,t) = \psi(L,t) = 0$ and

$\psi(x,0) = c\,e^{-\alpha|x|}$ (σ, a, E_0, L, α, c given)

- Find the roots of

$$f(x) = e^{-bx} + \cos x - ax \arccos(x^2+1)$$

(a, b constants)

- Find a function to describe the data below

2. Goals

- Apply numerical approximations to problems of such a type
- Understand why & when an approximation works or goes wrong

(such problems are related to the generation / storage of data)

(*) Task : "tell" the computer what to do

Necessary tools :
- Programming language (Fortran, C++, Matlab, Mathematica, VBA, etc.)
- Program structure

3. Control statements

3.1 - Definition: Control statements are program structures which affect:
- The order in which statements are executed
- Whether statements are executed or not

3.1.1 : "If" statement :

Makes a program give different outputs depending on a specific condition

- (In mathematica): If [Condition, Value for true, Value for false]

Condition = expression 1 COMPARISON OPERATOR expression 2

COMPARISON OPERATORS : == (equal to)
!= (not equal to)
> (greater than)
>= (greater than or equal to)
< (less than)
<= (less than or equal to)

Example 1: a = 1
b = 3

If [a > b, Print [a], Print [b]]

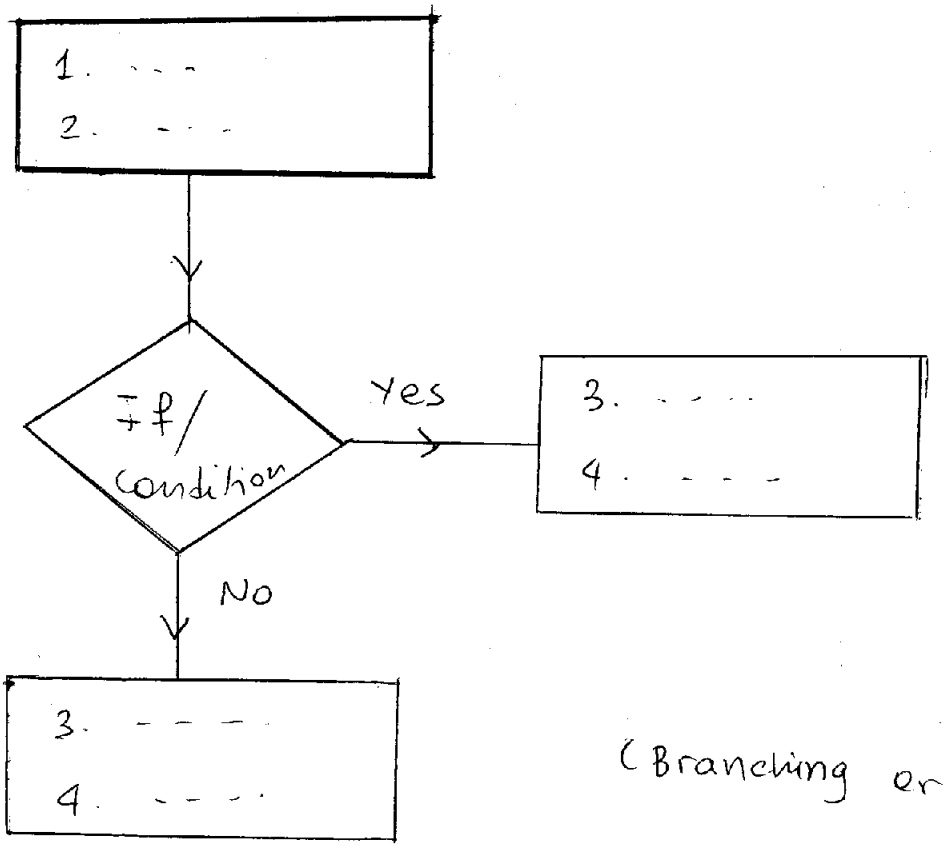
- => Expected result: 3
(One is defining a and b and "telling" the computer to write a or b, depending on a condition)

* Please note : Depending on the language used, the syntax and the comparison operators will be different, but the logic behind it is the same

- Example 1 (in fortran) :

```
a = 1
b = 3
If (a.gt.b) then
write (*,*) a
else
write (*,*) b
endif
```

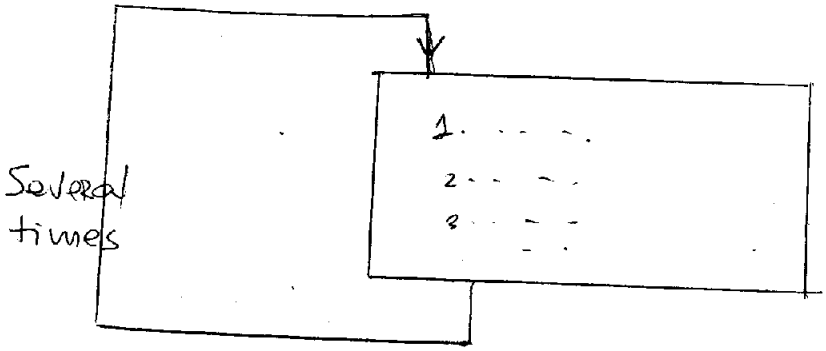
- More general notation: flow charts



(Branching or decision structures)

3.1.2 - Do loops : Makes the program perform the same Operation several times

. Flow chart :



(iteration structures)

④ Example 2: Write the integer numbers from 1 to 10

Do [Print[i] , {i, 1, 10}]

upper bound

lower bound

increment

index

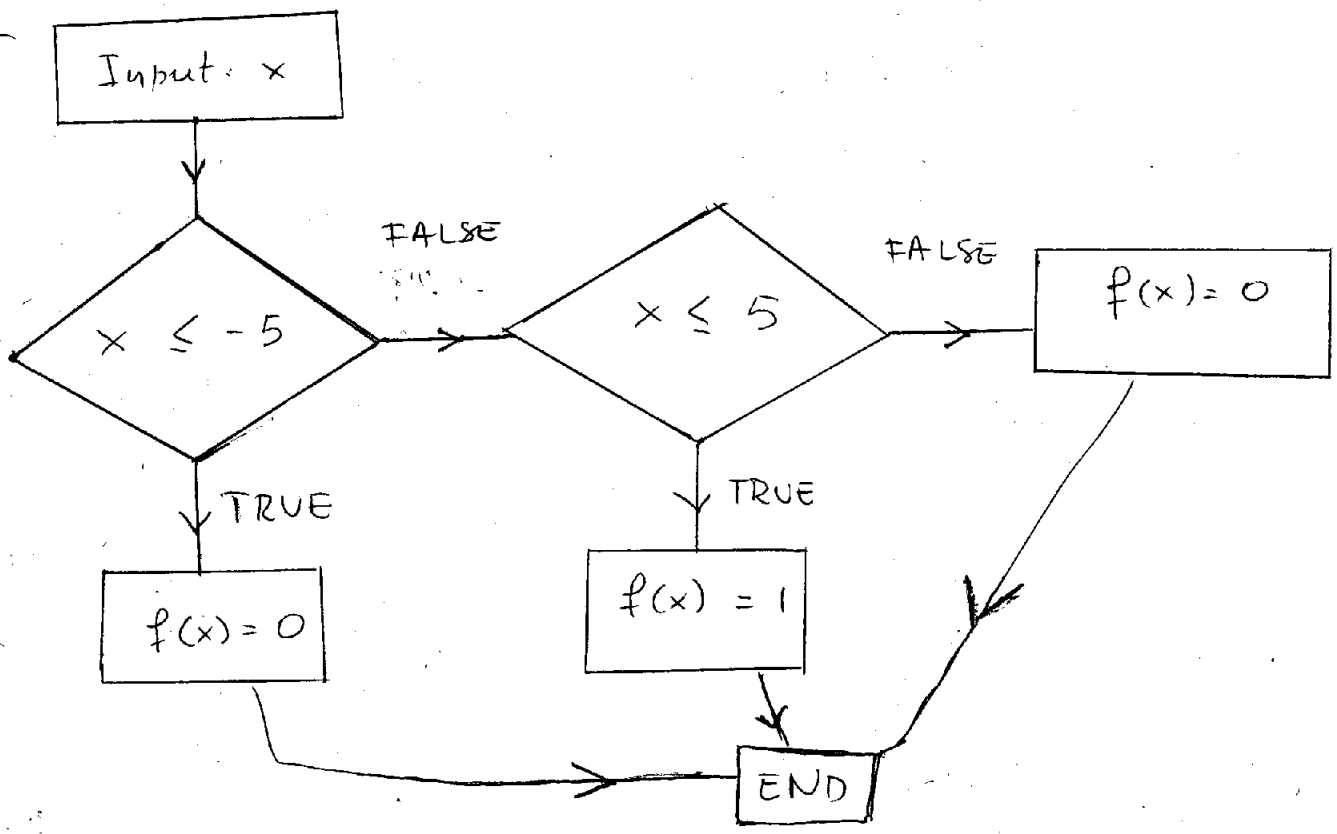
(*) Please note:

- If statements can in principle be nested

Example 3: produce the user-defined function

$$f(x) = \begin{cases} 0, & x \leq -5 \\ 1, & -5 \leq x \leq 5 \\ 0, & x > 5 \end{cases}$$

Flow chart:

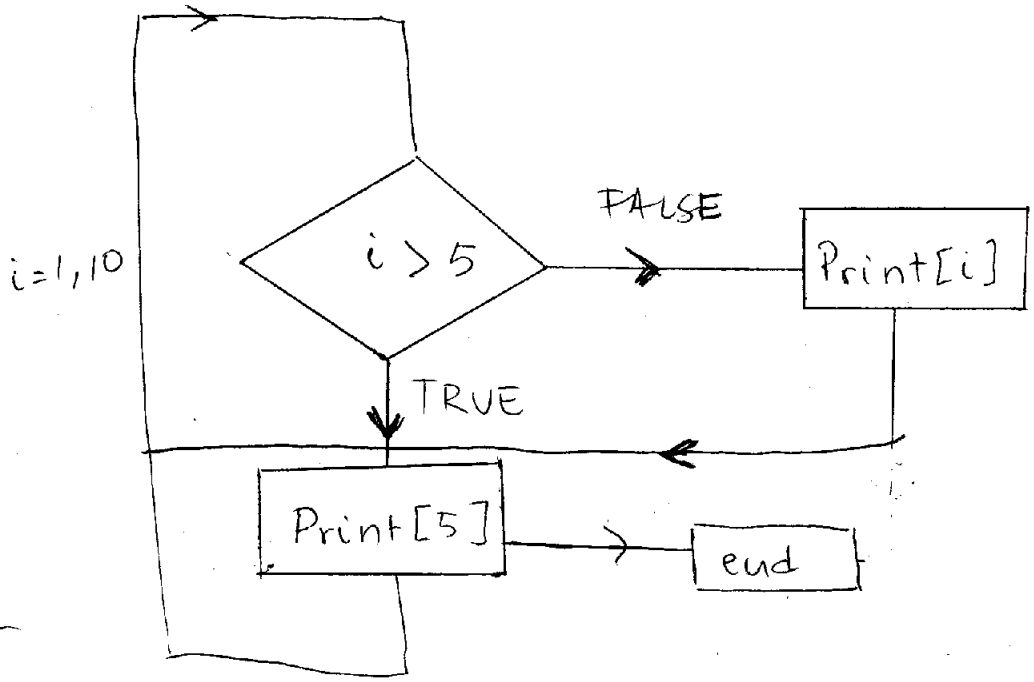



```
f[x_] := If[x <= 5, 0, If[x >= 5, 1, 0]]
```

- If and do statements can be combined

Example 4: For the set of numbers in Example 2, if $x \leq 5$, write the corresponding number. Otherwise, keep on writing 5.

• Flow chart



• In mathematica:

```
Do[ If[i>5, Print[5], Print[i], {i, 1, 10}]
```

4. Error, accuracy and stability

4.1 - Round-off error

Comes from the fact that a machine performs computations with a FINITE number of digits

~~A computer uses~~

⊛ Please note: it is not possible to represent all numbers using a finite number of digits

Examples. $\pi = 3.14159\ldots$

$\frac{1}{3} = 0.33333\ldots$

$\sqrt{3} = 1.73\ldots$

=> A computer uses ~~xxx~~ APPROXIMATE VALUES for the actual numbers

• Examples: single precision, double precision
($\sim 10^{-8}$) ($\sim 10^{-16}$)

Such estimates are performed by looking at how a number is represented in a machine

4.1.1 - Binary form

• We use a decimal representation (basis 10)

Example: $1492 = 1\times10^3 + 4\times10^2 + 9\times10^1 + 2\times10^0$

$2005 = 2\times10^3 + 0\times10^2 + 0\times10^1 + 2\times10^0$

• Computers use a binary representation (basis 2)

Example: $7 = 0\times2^3 + \underbrace{1\times2^2}_{4} + \underbrace{1\times2^1}_{2} + \underbrace{1\times2^0}_{1} = (0111)_2$

- The numbers are "packed" in bits (binary units)
&
bytes (sets of 8 bits)

- In general, a number is represented by

$$(-1)^S M\, 2^{C-E}$$

where $S \equiv$ sign bit
$M \equiv$ mantissa
$E \equiv$ bias (depends on the machine)
$C \equiv$ exponents

• For M & C the number of bits vary
• The precision is determined by the difference between two neighboring numbers

4.1.2 - Decimal floating-point form

A number is approximated by $\pm 0.d_1 d_2 \cdots d_k \times 10^n$, $1 \le d_i \le 9$
(note that k is finite)

Example: How to approximate $\pi = 3.14159\ldots$ up to 5 decimal digits?

- Chopping $= \pi \simeq 0.31415 \times 10^1$ (five-digit chopping)
- Rounding If $[d_{k+1} \geq 5, d_k = d_k + 1, d_k = d_k]$

$\pi \simeq 0.31416 \times 10^1$ (five-digit rounding)

"A round-off error is the error obtained from replacing a number by its floating-point form".

4.2 - Truncation error

Come from particular approximations used in the codes. The goal of the methods used in numerical analysis is to minimize such errors.

Example: The series expansion $\exp[x] = \sum_{n=0}^{\infty} \frac{x^n}{n!}$ can not be performed numerically until infinity (at some point it needs to be truncated)

Hence, in practice the sum $\sum_{n=0}^{N} \frac{x^n}{n!}$ (N finite) will be evaluated

4.3. Relative and absolute error

*If p^* is an approximation to p, the absolute error is $|p - p^*|$ and the relative error is $\frac{|p - p^*|}{|p|}$, $p \neq 0$

Examples: $p = 0.20 \times 10^{1}$ Absolute error : 0.1

$p^* = 0.21 \times 10^{1}$ Relative error : $\frac{0.1}{2} = 0.05$

• Significant digits:

The number p^* is approximate to p to t significant digits if t is the largest nonnegative integer for which $\frac{|p-p^*|}{|p|} < 5 \times 10^{-t}$

Example If $p = 0.1$ and $p^* = 0.102$

$$\frac{|p-p^*|}{|p|} = \frac{|0.02|}{0.1} = 0.2$$

$0.2 < 5 \times 10^{-t}$ satisfied for $t = 1$

$\Rightarrow p^*$ is approximate to p to one significant digit

• Relative error of the floating-point representation of a number y:

$y = 0.d_1 d_2 \cdots d_k d_{k+1} \cdots \times 10^{n}$ ("exact" number)

$fl(y) = 0.d_1 d_2 \cdots d_k \times 10^{n}$ (floating-point representation)

Relative error: $\text{err} = \frac{|y - fl(y)|}{|y|} = \frac{|(0.d_1 d_2 \cdots d_k d_{k+1} \cdots - 0.d_1 d_2 \cdots d_k) \times 10^{n}|}{|0.d_1 d_2 \cdots d_k d_{k+1} \cdots \times 10^{n}|}$

$$err = \left| \frac{0.d_{k+1}d_{k+2}\cdots}{0.d_1 d_2 \cdots} \right| \times 10^{-k}$$

Upper bound : • Maximum value for the numerator $\Rightarrow$ 1
• Minimum value for the denominator $\Rightarrow$ 0.1

$$\left| \frac{y - fl[y]}{y} \right| \leq \frac{1}{0.1} \times 10^{-k} = 10^{-k+1}$$

(*) Please note : The subtraction of nearly equal numbers is a very common error-producing computation.

Proof:
Let us consider two numbers x and y, $x > y$

The floating-point forms of both numbers are

$$fl(x) = 0.d_1 d_2 \cdots d_p \alpha_{p+1} \alpha_{p+2} \cdots \alpha_k \times 10^n \quad (1)$$

$$fl(y) = 0.d_1 d_2 \cdots d_p \beta_{p+1} \beta_{p+2} \cdots \beta_k \times 10^n \quad (2)$$

Floating-point form of $x - y$:

$$fl(fl(x) - fl(y)) = 0.\overbrace{0000\cdots}^{p \text{ times}} \sigma_{p+1} \sigma_{p+2} \cdots \sigma_k \times 10^{n-p}$$
$$= 0.\sigma_{p+1} \sigma_{p+2} \cdots \sigma_k \times 10^{n-p} \quad (3)$$

(3) has at most $k-p$ digits of significance.

Problem: it will be assigned k digits, since (1) and (2) have k digits of significance

- If a finite-digit calculation introduces an error, this error will be further enlarged by dividing by a number with small magnitude, or multiplying by a very large number

Proof: Consider a finite-digit approximation of a number z such that

$$fl(z) = z + \delta \quad \rightarrow \text{absolute error: } |z - (z+\delta)| = |\delta|$$

$$\text{relative error: } \left|\frac{\delta}{z}\right|$$

- Dividing by $\varepsilon = 10^{-n}$ or multiplying by 10^{n} $(n > 0)$ gives

$$\frac{z}{\varepsilon} \simeq fl\left(\frac{fl(z)}{fl(\varepsilon)}\right) = (z+\delta) \times 10^{n}$$

$$\text{Absolute error: } |z \times 10^{n} - (z+\delta) \times 10^{n}| = |\delta| \times 10^{n}$$

$\Rightarrow$ The absolute error has been enlarged!

Example 1: Consider the numbers

$$p = 0.54617$$
$$q = 0.54601$$

The difference $r = p - q = 0.00016$

Let us round p and q to four digits:

$$p^* = 0.5462 \quad \rightarrow \quad r^* = p^* - q^* = 0.0002$$
$$q^* = 0.5460$$

The relative error is:

$$\frac{|r - r^*|}{|r|} = \frac{|0.00016 - 0.0002|}{|0.00016|} = 0.25$$

$\Rightarrow$ The result has only one significant digit!

(*) **Please note**: One can reduce such problems by rearranging / reformulating computations

Example: Quadratic equation:

$$ax^2 + bx + c = 0 \quad , \quad a \neq 0$$

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \qquad (*)$$

$$x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a} \qquad (**)$$

If $b^2 >> 4ac$, in order to compute x_1 we will have to subtract nearly equal numbers

Consider this formula applied to $x^2 + 62.10x + 1 = 0$ using four-digit rounding arithmetic

We will take, as "exact" roots: (4-digit rounding)

$$x_1 = -0.01610723 \cong -0.01611$$

$$x_2 = -62.08390 \cong -62.08$$

• "Approximate" x_1: (*) in four-digit arithmetic...

$$\sqrt{b^2 - 4ac} \Rightarrow \sqrt{(0.6210 \times 10^2)^2 - 0.4000 \times 10^1} =$$

$$(62.10)^2 - 4.000$$

$$= \sqrt{3856. - 4.000} = \sqrt{3852.} = 62.06$$

$$fl(x_1) = \frac{-62.10 + 62.06}{2.000} = -0.02000$$

Relative error: $\dfrac{|-0.01611 + 0.02000|}{|0.01611|} \cong 2.4 \times 10^{-1}$

• "Approximate" x_2: $\dfrac{-62.10 + 62.06}{2.000} = \dfrac{-124.2}{2.000} = -62.10$

- Relative error: $\frac{|-62.08 + 62.10|}{|-62.08|} \approx 3.2 \times 10^{-4}$

⇒ A more accurate four-digit rounding approximation can be obtained by rationalizing the numerator

$$x_1 = \frac{-b + \sqrt{b^2-4ac}}{2a} \cdot \frac{(-b - \sqrt{b^2-4ac})}{(-b - \sqrt{b^2-4ac})}$$

(Remember: $(p-q)(p+q) = p^2 - q^2$)

$$\Rightarrow x_1 = \frac{b^2 - (b^2 - 4ac)}{2a(-b - \sqrt{b^2-4ac})} = \frac{-2c}{b + \sqrt{b^2-4ac}}$$

$$f_l(x_1) = \frac{-2.000}{62.10 + 62.06} = \frac{-2.000}{124.2} = -0.01610$$

$$\text{Relative error} \cdot \frac{|-0.01611 - 0.01610|}{|-0.01611|} \simeq 6.2 \times 10^{-4}$$

(4) Hence, the accuracy of a code may depend to a very large extent of how it is written.

5. Convergence

5.1. **Stability**: A code is stable if small changes in the initial data produce correspondingly small changes in the final results. Otherwise, it is unstable. If a code is stable only for specific choices of initial data, then it is conditionally stable

5.2. **Error growth**:

Suppose that E_0 denotes an initial error and E_n the error after n subsequent operations.

- If $E_n \approx C n E_0$ (C = const) then the error growth is linear

• If $E_n \approx C^n E_0$, $C > 1$, then the error growth is exponential

(*) Please note: a code where the error grows exponentially is UNSTABLE

5.3. Convergence

Definition: Suppose that $\{x_n\}_{n=1}^{\infty}$ is an infinite sequence. Then the sequence is said to have the limit L if, given any $\epsilon > 0$, there exists a positive integer $N = N(\epsilon)$ such that $n > N$ implies that $|x_n - L| < \epsilon$

Notation: $\lim_{n \to \infty} x_n = L$ or $\lim_{n \to \infty} (x_n - L) = 0$

In this case, this sequence is convergent

5.4 - Order of approximation (notation: $O(n^p)$)

Let $\{x_n\}_{n=1}^{\infty}$ and $\{y_n\}_{n=1}^{\infty}$ be two sequences. The sequence $\{x_n\}$ is "of the order of" $\{y_n\}$ ($x_n = O(y_n)$) if there exist constants C and N such that

$$|x_n| \leq C|y_n| \quad \text{whenever } n \geq N$$

Example: $\frac{n^2 - 1}{n^3} = O\left(\frac{1}{n}\right)$ since $\frac{n^2 - 1}{n^3} \leq \frac{n^2}{n^3} = \frac{1}{n}$ whenever $n \geq 1$

Order of convergence of a sequence

Suppose that $\{x_n\}_{n=1}^{\infty}$ is a sequence with $\lim_{n\to\infty} x_n = x$ and $\{r_n\}_{n=1}^{\infty}$ is a sequence with $\lim_{n\to\infty} r^n = 0$. We say that $\{x_n\}_{n=1}^{\infty}$ converges to x with the order of convergence $O(r_n)$ if there exists a constant $k > 0$ such that

$$|x_n - x| \leq k|r_n| \quad \text{for } n \text{ sufficiently large}$$

Notation: $x_n = x + O(r_n)$

Example: Let $x_n = \frac{\cos(n)}{n^2}$ and $r_n = \frac{1}{n^2}$

Then $\lim_{n\to\infty} x_n = 0$ with a rate of convergence $O(1/n^2)$

Why?

$$|\cos n / n^2| = \frac{|\cos n|}{n^2} \leq \frac{1}{n^2}$$ since $\cos n$ is bounded.

Algorithms

Definition: An algorithm is a procedure that describes, in an unambiguous manner, a finite sequence of steps to be performed in a specified order

(*) One uses a pseudocode to describe algorithms

- Do loops:

  For $i = 1, 2, \dots, n$

  While $i < N$ do steps 3-6

- If statements

  If...then

  If... then

  else

Example: Write an algorithm to compute $\sum_{i=1}^{N} x_1 + x_2 + \cdots + x_N$

Input $N, x_1, x_2, \dots, x_n$

Output $SUM = \sum_{i=1}^{N} x_i$

Step 1 Set sum = 0 (initialize accumulator)

Step 2 For $i = 1, 2, \dots, N$ do

Set $SUM = SUM + x_i$ (add the next term)

Step 3 Output (sum)

Stop