

# Numerical Methods and Computational Tools

---

Andrea Latina

`andrea.latina@cern.ch`

CAS - Introduction to Accelerator Physics - Santa Susanna, Spain - 2023

# Table of contents

## 1. Introduction

- Purpose
- Some references

## 2. Internal representation of numbers

- Machine precision
- Numerical errors
  - Round-off
  - Cancellation
  - Truncation

## 3. Tools

- Octave and Python
- Maxima
- Shell tools
- C++ and libraries

## 4. Accelerator physics codes

## Purpose of this course

In these two lessons, we will outline some fundamental concepts in scientific computing and guide the novice through the multitude of tools available. We will describe the main tools and explain which tool should be used for a specific purpose, dispelling common misconceptions, and suggest good practices.

## Purpose of this course

In these two lessons, we will outline some fundamental concepts in scientific computing and guide the novice through the multitude of tools available. We will describe the main tools and explain which tool should be used for a specific purpose, dispelling common misconceptions, and suggest good practices.

We will suggest reference readings and clarify important aspects of numerical stability to help avoid making bad but unfortunately common mistakes. Numerical stability should be basic knowledge of every scientist.

## Purpose of this course

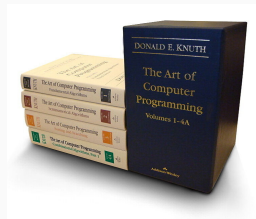
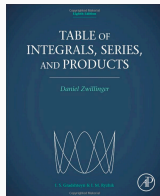
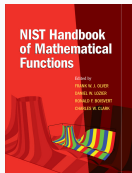
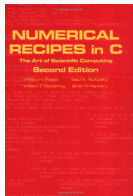
In these two lessons, we will outline some fundamental concepts in scientific computing and guide the novice through the multitude of tools available. We will describe the main tools and explain which tool should be used for a specific purpose, dispelling common misconceptions, and suggest good practices.

We will suggest reference readings and clarify important aspects of numerical stability to help avoid making bad but unfortunately common mistakes. Numerical stability should be basic knowledge of every scientist.

We will exclusively refer to free and open-source software running on Linux or other Unix-like operating systems. Also, we will unveil powerful shell commands that can speed up simulations, facilitate data processing, and in short, increase your scientific throughput.

# Some references

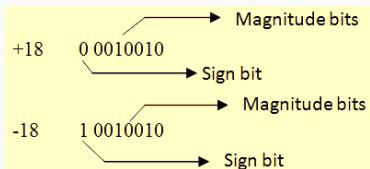
1. "Numerical Recipes: The Art of Scientific Computing", W. Press, S. Teukolsky, W. Vetterling, and B. Flannery, 1992 (2nd edition) – 2007 (3rd edition)
2. Abramowitz and Stegun, "Handbook of Mathematical Functions with Formulas", 1964
3. "The NIST Handbook of Mathematical Functions", Olver, Lozier, Boisvert, and Clark, 2010
4. Donald Knuth, "The Art of Computer programming", 1968 – (the book is still incomplete)
5. Gradshteyn and Ryzhik, "Table of Integrals, Series, and Products", Academic Press Inc; 8th edition (27 October 2014)
6. Particle Data Group (LBL), "Review of Particle Physics", Oxford University Press



# Internal representation of numbers

## Integers

- Int, or integer, is a whole number, positive or negative, without decimals. In binary format



- Typically, an integer occupies four bytes, or 32 bits.
- The possible range for 32-bit integers is

$$-2^{31} < X < 2^{31} - 1$$

(from -2,147,483,648 to 2,147,483,647).

# Internal representation of numbers

## Integer types

- In compiled languages such as C and C++, specific types exist for better control:

Data Type	Size	Size in bytes	Signed range
[un signed] char	8 bits	1	-128 to 127
[un signed] short int	16 bits	2	-32768 to 32767
[un signed] int	32 bits	4	-2147483648 to 2147483647
[un signed] long int	32 bits	4	-2147483648 to 2147483647
[un signed] long long int	64 bits	8	$-2^{63}$ to $2^{63} - 1$

- Arithmetic between numbers in integer representation is exact, if the answer is not outside the range of integers that can be represented.



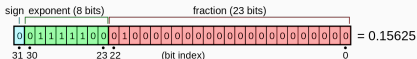
# Internal representation of numbers

## Real numbers

- Real numbers use a floating-point representation IEEE-754

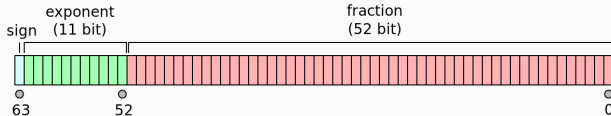
$$\text{value} = (-1)^{\text{sign}} \times 1.\text{fraction} \times 2^{\text{exponent}}$$

- Single-precision floating point representation (32 bits)



The C/C++ type is `float`.

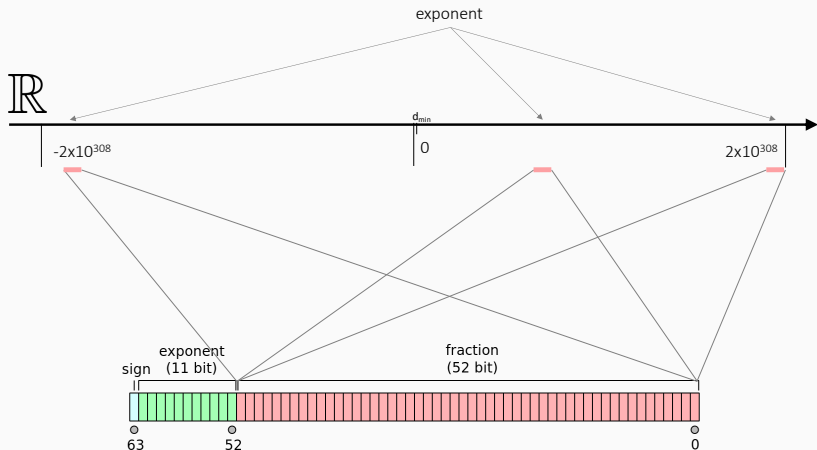
- Double-precision floating point representation (64 bits)



The C/C++ type is `double`.

Note: Some CPUs internally store floating point numbers in even higher precision: 80-bit in extended precision, and 128-bit in quadruple precision. In C++ quad. precision may be specified using `long double`.

# Range of double-precision numbers



- 52 bits of mantissa correspond to about 15-digit precision in base 10. In single precision, there are just 7-digit precision in base 10.
- Double-precision numbers as integers are **exact** within the range  $\pm 2^{53}$  ( $\approx \pm 10^{16}$ ).
- The smallest possible number that can be represented is  $d_{\min} = 2^{-1074} \approx 4.94 \times 10^{-324}$

# Special numbers

IEEE-754 floating-point types may support special values:

- **zero**
- the **negative zero**,  $-0.0$ . It compares equal to  $+0.0$ , but is meaningful in some arithmetic operations, e.g.  $1.0/-0.0 == -\text{INFINITY}$
- **infinity** (positive and negative)
- **Not-a-number (NaN)**, it's the result for example of  $0/0$ . A NaN does not compare equal with anything (including itself)

# Special numbers

IEEE-754 floating-point types may support special values:

- zero
- the **negative zero**,  $-0.0$ . It compares equal to  $+0.0$ , but is meaningful in some arithmetic operations, e.g.  $1.0/-0.0 == -\text{INFINITY}$
- **infinity** (positive and negative)
- **Not-a-number (NaN)**, it's the result for example of  $0/0$ . A NaN does not compare equal with anything (including itself)

C:

```
#include <math.h>
double f = INFINITY;
double nan = NAN;
```

C++:

```
#include <limits>
double f = std::numeric_limits<double>::infinity();
double qnan = std::numeric_limits<double>::quiet_NaN();
double snan = std::numeric_limits<double>::signaling_NaN();
```

Python:

```
f = numpy.inf
nan = numpy.nan
```

Octave:

```
f = inf;
n = nan;
```

# Machine epsilon, $\varepsilon_m$

The *machine epsilon*, or *accuracy*,  $\varepsilon_m$ , is the gap between 1 and the next representable double.  
The smallest double number such that:

$$1.0 + \varepsilon_m \neq 1.0$$

- For double precision

$$\varepsilon_m = 2^{-52} \approx 2 \cdot 10^{-16},$$

- For single precision

$$\varepsilon_m = 2^{-23} \approx 3 \cdot 10^{-8}.$$

**Exercise:** How much is

$$10^{20} + 1 - 10^{20} = ?$$

# Machine epsilon, $\varepsilon_m$

The *machine epsilon*, or *accuracy*,  $\varepsilon_m$ , is the gap between 1 and the next representable double. The smallest double number such that:

$$1.0 + \varepsilon_m \neq 1.0$$

- For double precision

$$\varepsilon_m = 2^{-52} \approx 2 \cdot 10^{-16},$$

- For single precision

$$\varepsilon_m = 2^{-23} \approx 3 \cdot 10^{-8}.$$

**Exercise:** How much is

$$10^{20} + 1 - 10^{20} = ?$$

**Note:** It is important to understand that  $\varepsilon_m$  is not the smallest floating-point number that can be represented on a machine.

The smallest number,  $d_{\min}$ , depends on how many bits there are in the exponent.  $\varepsilon_m$  depends on how many bits there are in the mantissa.

# Overflow and underflow

The **Overflow** occurs when an operation attempts to create a numeric value that is outside of the range that can be represented with a given number of digits – either higher than the maximum or lower than the minimum representable value.

The **Underflow** is a condition in a computer program where the result of a calculation is a number of smaller absolute value than the computer can actually represent in memory.

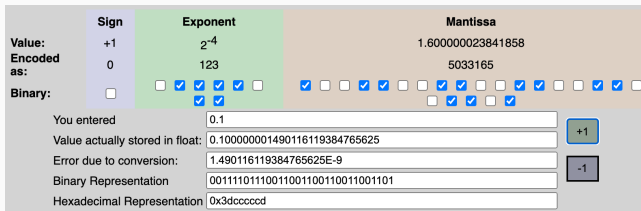




# Round-off error

The *round-off error*, also called *rounding error*, is the difference between the exact result and the result obtained using finite-precision, rounded arithmetic.

As an example of round-off error, see the representation of the number 0.1



[link]

Round-off errors accumulate with increasing amounts of calculation.

If, in the course of obtaining a calculated value, one performs  $N$  such arithmetic operations, one might end up having a total round-off error on the order of  $\sqrt{N}\epsilon_m$  (when lucky)

(Note: The square root comes from a random-walk, as the round-off errors come in randomly up or down.)

# Round-off error

The *round-off error*, also called *rounding error*, is the difference between the exact result and the result obtained using finite-precision, rounded arithmetic.

As an example of round-off error, see the representation of the number 0.1

The screenshot shows a web-based tool for visualizing floating-point numbers. It is divided into three main sections: Sign (light blue), Exponent (light green), and Mantissa (light brown). Below these are input fields for the user's value and the tool's output.

	Sign	Exponent	Mantissa
Value:	+1	$2^{-4}$	1.600000023841858
Encoded as:	0	123	5033165
Binary:	<input type="checkbox"/>	<input type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input type="checkbox"/>	<input checked="" type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input type="checkbox"/>

You entered	<input type="text" value="0.1"/>	<input type="button" value="+1"/>
Value actually stored in float:	<input type="text" value="0.100000001490116119384765625"/>	
Error due to conversion:	<input type="text" value="1.490116119384765625E-9"/>	<input type="button" value="-1"/>
Binary Representation	<input type="text" value="0011101110011001100110011001101"/>	
Hexadecimal Representation	<input type="text" value="0x3dccccd"/>	

[link]

Round-off errors accumulate with increasing amounts of calculation.

If, in the course of obtaining a calculated value, one performs  $N$  such arithmetic operations, one might end up having a total round-off error on the order of  $\sqrt{N}\epsilon_m$  (when lucky)

(Note: The square root comes from a random-walk, as the round-off errors come in randomly up or down.)

**The golden rule:** try to reduce the number of operations required to perform a calculation.

# Round-off error

## Example

```
a = 0.7;  
  
% how many iterations?  
  
while a < 0.8  
    a = a + 0.1;  
    print a;  
  
endwhile
```

Or even just : is  $0.2 + 0.1 == 0.3$  ? Try...

→ Most decimal numbers don't have exact binary representations.

## Then, how can we compare floating-point numbers?

If  $0.2 + 0.1 \neq 0.3$ , how can we assert if two numbers  $a$  and  $b$  are equal?

Floating point numbers can only be *approximately* equal.

We need to define the accepted relative,  $\epsilon_{\text{rel}}$ , and an absolute,  $\epsilon_{\text{abs}}$ , errors. Then,

$$a == b ? \left\{ \begin{array}{l} \text{if } a == b \\ \quad \text{return } \textit{true} \\ \\ \text{if } |a - b| < \max(\epsilon_{\text{abs}}, \epsilon_{\text{rel}} \cdot (|a| + |b|)) \\ \quad \text{return } \textit{true} \\ \\ \text{otherwise} \\ \quad \text{return } \textit{false} \end{array} \right.$$

where  $\epsilon_{\text{rel}} \geq \epsilon_m$ .

# Cancellation error

**Cancellation error:** when one adds two numbers with opposite sign but with similar absolute values. The result may be quite inexact and the situation is referred to as loss, or cancellation, of significant digits.

One encounters this type of error for example in polynomial evaluation.

**Example:**

$$\begin{aligned}f(x) &= (x - 1)^6 \\ &= x^6 - 6x^5 + 15x^4 - 20x^3 + 15x^2 - 6x + 1\end{aligned}$$

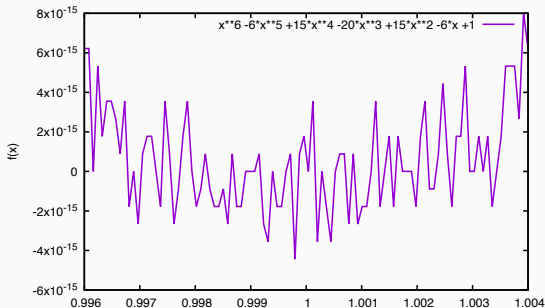
# Cancellation error

**Cancellation error:** when one adds two numbers with opposite sign but with similar absolute values. The result may be quite inexact and the situation is referred to as loss, or cancellation, of significant digits.

One encounters this type of error for example in polynomial evaluation.

**Example:**

$$\begin{aligned}f(x) &= (x - 1)^6 \\ &= x^6 - 6x^5 + 15x^4 - 20x^3 + 15x^2 - 6x + 1\end{aligned}$$



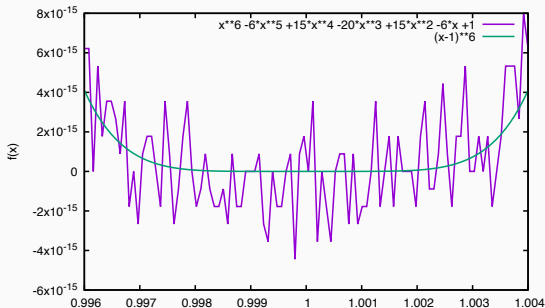
# Cancellation error (a.k.a. “Catastrophic” cancellation error)

**Cancellation error:** when one adds two numbers with opposite sign but with similar absolute values. The result may be quite inexact and the situation is referred to as loss, or cancellation, of significant digits.

One encounters this type of error for example in polynomial evaluation.

**Example:**

$$\begin{aligned} f(x) &= (x - 1)^6 \\ &= x^6 - 6x^5 + 15x^4 - 20x^3 + 15x^2 - 6x + 1 \end{aligned}$$



# Cancellation error: remedies...

Few cases are recurrent and dangerous...

1. Case of the quadratic formula:

$$ax^2 + bx + c = 0$$

Recall the solution from high school. If the solutions are written this way, cancellation can occur:

$$r_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

$$r_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$



# Cancellation error: remedies...

Few cases are recurrent and dangerous...

1. Case of the quadratic formula:

$$ax^2 + bx + c = 0$$

Recall the solution from high school. If the solutions are written this way, cancellation can occur:

$$r_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$
$$r_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

The remedy is to rewrite  $r_1$  and  $r_2$  to avoid cancellation:

- $r_1$ : if  $b^2 \gg ac$  and  $b > 0$ , use  $r_1 = \frac{2c}{-b - \sqrt{b^2 - 4ac}}$
- $r_2$ : if  $b^2 \gg ac$  and  $b < 0$ , use  $r_2 = \frac{2c}{-b + \sqrt{b^2 - 4ac}}$

# Cancellation error: remedies... [example]

1. Case of the quadratic formula:

$$ax^2 + bx + c = 0$$

Let's take an example:  $a = 10^{-20}$ ,  $b = 1$ ,  $c = 1$

# Cancellation error: remedies... [example]

1. Case of the quadratic formula:

$$ax^2 + bx + c = 0$$

Let's take an example:  $a = 10^{-20}$ ,  $b = 1$ ,  $c = 1$

- $r_1 = \frac{2c}{-b - \sqrt{b^2 - 4ac}} = \frac{2}{-1 - \sqrt{1 - 4 \cdot 10^{-20}}} = -1$  [ ALMOST CORRECT ]
- $r_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a} = \frac{-1 - \sqrt{1 - 4 \cdot 10^{-20}}}{10^{-20}} = -10^{20}$  [ ALMOST CORRECT ]

# Cancellation error: remedies... [example]

1. Case of the quadratic formula:

$$ax^2 + bx + c = 0$$

Let's take an example:  $a = 10^{-20}$ ,  $b = 1$ ,  $c = 1$

- $r_1 = \frac{2c}{-b - \sqrt{b^2 - 4ac}} = \frac{2}{-1 - \sqrt{1 - 4 \cdot 10^{-20}}} = -1$  [ ALMOST CORRECT ]
- $r_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a} = \frac{-1 - \sqrt{1 - 4 \cdot 10^{-20}}}{10^{-20}} = -10^{20}$  [ ALMOST CORRECT ]

Choosing the other formulation:

- $r_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} = \frac{-1 + \sqrt{1 - 4 \cdot 10^{-20}}}{10^{-20}} = 0$  [ WRONG! ]
- $r_2 = \frac{2c}{-b + \sqrt{b^2 - 4ac}} = \frac{2}{-1 + \sqrt{1 - 4 \cdot 10^{-20}}} = \infty$  [ WRONG! ]

# Cancellation error: remedies...

Catastrophic cancellation can occur in the evaluation of expressions like:

2. Algebraic binomials, e.g.

$$x^2 - y^2$$

can incur in underflow errors if  $y^2 \ll x^2$  (when  $y^2/x^2 < \epsilon_m$ ). This expression is more accurately evaluated as

$$(x + y)(x - y)$$

# Cancellation error: remedies...

Catastrophic cancellation can occur in the evaluation of expressions like:

2. Algebraic binomials, e.g.

$$x^2 - y^2$$

can incur in underflow errors if  $y^2 \ll x^2$  (when  $y^2/x^2 < \epsilon_m$ ). This expression is more accurately evaluated as

$$(x + y)(x - y)$$

3. Summations of many numbers of very large different magnitude. There are two solutions:

1. Sort the numbers by  $\text{abs}(\text{magnitude})$  and sum from the smallest to the largest
2. "Kahan summation" algorithm

## Another example of catastrophic cancellation

Suppose we want to compute the value of the function

$$f(x) = \frac{1 - \cos x}{x^2}$$

for  $x$  very close to 0, say  $x = 10^{-8}$ .

( Remember:  $\cos x \simeq 1 - \frac{x^2}{2} + \frac{x^4}{24} + \dots$  )

We can compute this value directly using a computer program as follows:

```
x = 1e-8
numerator = 1 - cos(x)
denominator = x**2
result = numerator/denominator
print(result)
```

This program should print out a value very close to 0.5. However, due to catastrophic cancellation error, the result may be significantly different (0, in fact).

# Special built-in mathematical functions

Transcendental functions are computed by the CPU (FPU) using Taylor expansions. E.g., the logarithm:

$$\log(x) = (x - 1) - \frac{(x - 1)^2}{2} + \frac{(x - 1)^3}{3} - \frac{(x - 1)^4}{4} + \dots$$

which makes the function incur in cancellation whenever  $x < \varepsilon_m$ .



# Special built-in mathematical functions

Transcendental functions are computed by the CPU (FPU) using Taylor expansions. E.g., the logarithm:

$$\log(x) = (x-1) - \frac{(x-1)^2}{2} + \frac{(x-1)^3}{3} - \frac{(x-1)^4}{4} + \dots$$

which makes the function incur in cancellation whenever  $x < \varepsilon_m$ .

## **log1p(x)**

To overcome this problem, the C standard library, as well as Octave and Python, provide the function `log1p`, which implements

$$\log1p(x) = \log(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \dots$$

this is numerically stable. So, whenever the argument of a logarithm is in the form  $1+x$ , use `log1p`.

# Special built-in mathematical functions

Transcendental functions are computed by the CPU (FPU) using Taylor expansions. E.g., the logarithm:

$$\log(x) = (x-1) - \frac{(x-1)^2}{2} + \frac{(x-1)^3}{3} - \frac{(x-1)^4}{4} + \dots$$

which makes the function incur in cancellation whenever  $x < \varepsilon_m$ .

## **log1p(x)**

To overcome this problem, the C standard library, as well as Octave and Python, provide the function `log1p`, which implements

$$\log1p(x) = \log(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \dots$$

this is numerically stable. So, whenever the argument of a logarithm is in the form  $1+x$ , use `log1p`.

**expm1(x)** =  $e^x - 1$ , is similar.

Functions like:  $1 - \cos(x)$ ,  $1 - \cosh(x)$ , require a similar approach, but there is no pre-defined solution.

Take for example the transfer matrix of a sector bend,

$$M_{\text{dipole}} = \begin{pmatrix} \cos \theta & \rho \sin \theta & 0 & 0 & 0 & \rho(1 - \cos \theta) \\ -\frac{1}{\rho} \sin \theta & \cos \theta & 0 & 0 & 0 & \sin \theta \\ 0 & 0 & 1 & L & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ -\sin \theta & -\rho(1 - \cos \theta) & 0 & 0 & 1 & \rho(\theta - \sin \theta) \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

where  $\theta = \frac{L}{\rho}$ ,  $L$  is the magnet's length, and  $\rho$  is the bending radius.

# Special built-in mathematical functions

`hypot(a, b)`

provides a numerically stable implementation of

$$c = \sqrt{a^2 + b^2}$$

which causes cancellation when  $|a| \ll |b|$  or  $|b| \ll |a|$ .

# Special built-in mathematical functions

`hypot(a, b)`

provides a numerically stable implementation of

$$c = \sqrt{a^2 + b^2}$$

which causes cancellation when  $|a| \ll |b|$  or  $|b| \ll |a|$ .

Example,  $E$ , total energy of a particle:

$$E = \text{hypot}(\text{mass}, P)$$

[ `hypot(a,b)`, which is part of the C (C++, Python, Octave) standard library, computes instead:  $c = m \cdot \sqrt{1 + (M/m)^2}$  where  $m = \min(|a|, |b|)$ ,  $M = \max(|a|, |b|)$ . ]

# Special built-in mathematical functions

**hypot(a, b)**

provides a numerically stable implementation of

$$c = \sqrt{a^2 + b^2}$$

which causes cancellation when  $|a| \ll |b|$  or  $|b| \ll |a|$ .

Example,  $E$ , total energy of a particle:

$$E = \text{hypot}(\text{mass}, P)$$

[ `hypot(a,b)`, which is part of the C (C++, Python, Octave) standard library, computes instead:  $c = m \cdot \sqrt{1 + (M/m)^2}$  where  $m = \min(|a|, |b|)$ ,  $M = \max(|a|, |b|)$ . ]

What about: **pow(x, y) =  $x^y$**  ?

# Special built-in mathematical functions

**hypot(a, b)**

provides a numerically stable implementation of

$$c = \sqrt{a^2 + b^2}$$

which causes cancellation when  $|a| \ll |b|$  or  $|b| \ll |a|$ .

Example,  $E$ , total energy of a particle:

$$E = \text{hypot}(\text{mass}, P)$$

[ `hypot(a,b)`, which is part of the C (C++, Python, Octave) standard library, computes instead:  $c = m \cdot \sqrt{1 + (M/m)^2}$  where  $m = \min(|a|, |b|)$ ,  $M = \max(|a|, |b|)$ . ]

What about: **pow(x, y) =  $x^y$**  ?

$$\text{pow}(x,y) = \exp(\log x^y) = \exp(y \log x)$$

It's computationally expensive and inaccurate. Avoid for  $y \in \text{integers}$ .

# Summary of special built-in mathematical functions

$$\text{log1p}(x) = \log(1 + x)$$

$$\text{expm1}(x) = e^x - 1$$

$$\text{hypot}(x,y) = \sqrt{x^2 + y^2}$$

$$\text{sqrt}(x) = \sqrt{x}, \text{ square root of } x.$$

$$\text{cbrt}(x) = \sqrt[3]{x}, \text{ cube root of } x.$$

**scalbn(x,n)** returns  $x \cdot 2^n$  computed by exponent manipulation.

**atan2(y,x)** =  $\arctan\left(\frac{y}{x}\right)$  with detection of the correct quadrant.

**fabs(x)** computes the absolute value of a floating-point number  $x$ .

**floor(x)** returns the largest integral value *less than or equal* to  $x$ .

**ceil(x)** returns the smallest integral value *greater than or equal* to  $x$ .

**round(x)** returns the integral value *nearest* to  $x$ .

**trunc(x)** returns the integral value *nearest to but no larger in magnitude* than  $x$ .

**pow(x,y)** =  $e^{y \log x}$ , computes floating-point number  $x$  raised to the power of floating-point number  $y$ .



# Implementation of functions: an example

## Sin cardinal

Also the implementation of functions requires attention. Take for example the function “sin cardinal”,

$$\text{sinc}(x) = \begin{cases} 1 & \text{for } x = 0 \\ \frac{\sin(x)}{x} & \text{otherwise.} \end{cases}$$

Numerical instabilities might appear due to the division between two nearly-zero numbers. For  $x = 0$ , the result is Nan. A robust implementation of the sinus cardinal comes from a careful consideration of this function.

# Implementation of functions: an example

## Sin cardinal

Also the implementation of functions requires attention. Take for example the function “sin cardinal”,

$$\text{sinc}(x) = \begin{cases} 1 & \text{for } x = 0 \\ \frac{\sin(x)}{x} & \text{otherwise.} \end{cases}$$

Numerical instabilities might appear due to the division between two nearly-zero numbers. For  $x = 0$ , the result is Nan. A robust implementation of the sinus cardinal comes from a careful consideration of this function.

Let's take the Taylor expansion  $\text{sinc}(x)$  to first order,

$$\frac{\sin x}{x} \approx 1 - \frac{x^2}{6} + \dots$$

# Implementation of functions: an example

## Sin cardinal

Also the implementation of functions requires attention. Take for example the function “sin cardinal”,

$$\text{sinc}(x) = \begin{cases} 1 & \text{for } x = 0 \\ \frac{\sin(x)}{x} & \text{otherwise.} \end{cases}$$

Numerical instabilities might appear due to the division between two nearly-zero numbers. For  $x = 0$ , the result is Nan. A robust implementation of the sinus cardinal comes from a careful consideration of this function.

Let's take the Taylor expansion  $\text{sinc}(x)$  to first order,

$$\frac{\sin x}{x} \approx 1 - \frac{x^2}{6} + \dots$$

If we look at the right-hand side, we can appreciate the fact that in this form, when  $x$  is small, the numerical instability simply disappears. The final result will differ from zero if and only if

$$\left| -\frac{x^2}{6} \right| < \varepsilon_m,$$

If  $x$  is made explicit, a robust implementation should return 1 when:

$$|x| < \sqrt{6\varepsilon_m}.$$

# Truncation error

## Finite differentiation

Imagine that you have a procedure which computes a function  $f(x)$ , and now you want to compute its derivative  $f'(x)$ . Easy, right? The definition of the derivative,

$$f'(x) = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$$

practically suggests the program: Pick a small value  $h$ ; evaluate  $f(x+h)$  and  $f(x)$ , finally apply the above equation.

Applied uncritically, the above procedure is almost guaranteed to produce inaccurate results. There are two sources of error in equation: the **truncation error** and the **round-off error**.

# Truncation error

## Finite differentiation

Imagine that you have a procedure which computes a function  $f(x)$ , and now you want to compute its derivative  $f'(x)$ . Easy, right? The definition of the derivative,

$$f'(x) = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$$

practically suggests the program: Pick a small value  $h$ ; evaluate  $f(x+h)$  and  $f(x)$ , finally apply the above equation.

Applied uncritically, the above procedure is almost guaranteed to produce inaccurate results. There are two sources of error in equation: the **truncation error** and the **round-off error**.

Let's focus on the **truncation error** now, we know that

$$f(x+h) = f(x) + hf'(x) + \frac{1}{2}h^2f''(x) + \dots$$

(Taylor expansion), therefore

$$\frac{f(x+h) - f(x)}{h} = f' + \frac{1}{2}hf'' + \dots$$

Then, when we approximate  $f'$  as in the above equation, we make a **truncation error**:

$$\varepsilon_t = \frac{1}{2}hf'' + \dots = O(h)$$

In this case, the truncation error is linearly proportional to  $h$ . Higher-order formulations of the **first derivative give smaller error**.

## Example of finite difference formulæ

- First derivative to first order (forward difference):

$$f'(x) = \frac{f(x+h) - f(x)}{h} + O(h)$$

- First derivative to second order (centred difference):

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} + O(h^2)$$

- First derivative to fourth order (centred difference):

$$f'(x) = \frac{-f_{x+2h} + 8f_{x+h} - 8f_{x-h} + f_{x-2h}}{12h} + O(h^4)$$

- Second derivative to second order (centred difference):

$$f''(x) = \frac{f_{x+h} - 2f_x + f_{x-h}}{h^2} + O(h^2)$$

- Second derivative to fourth order (centred difference):

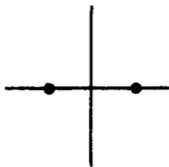
$$f''(x) = \frac{-f_{x+2h} + 16f_{x+h} - 30f_x + 16f_{x-h} - f_{x-2h}}{12h^2} + O(h^4)$$

# Finite difference formulæ

Abramowitz and Stegun, page 883 and following

## Partial Derivatives

25.3.21



$$\frac{\partial f_{0,0}}{\partial x} = \frac{1}{2h} (f_{1,0} - f_{-1,0}) + O(h^2)$$

# Finite difference formulæ

Abramowitz and Stegun, page 883 and following

884

NUMERICAL ANALYSIS

25.3.22



$$\frac{\partial^2 f_{0,0}}{\partial x^2} = \frac{1}{4h} (f_{1,1} - f_{-1,1} + f_{1,-1} - f_{-1,-1}) + O(h^2)$$

25.3.23



$$\frac{\partial^2 f_{0,0}}{\partial x^2} = \frac{1}{h^2} (f_{1,0} - 2f_{0,0} + f_{-1,0}) + O(h^2)$$

25.3.26



$$\frac{\partial^2 f_{0,0}}{\partial x \partial y} = \frac{1}{4h^2} (f_{1,1} - f_{-1,1} - f_{1,-1} + f_{-1,-1}) + O(h^2)$$

25.3.27



$$\frac{\partial^2 f_{0,0}}{\partial x \partial y} = \frac{-1}{2h^2} (f_{1,0} + f_{-1,0} + f_{0,1} + f_{0,-1} - 2f_{0,0} - f_{1,1} - f_{-1,-1}) + O(h^2)$$



Abramowitz and Stegun, page 883 and following

Laplacian

25.3.30

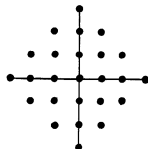


$$\begin{aligned}\nabla^2 u_{0,0} &= \left( \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right)_{0,0} \\ &= \frac{1}{h^2} (u_{1,0} + u_{0,1} + u_{-1,0} + u_{0,-1} - 4u_{0,0}) + O(h^2)\end{aligned}$$

25.3.31



25.3.33



$$\begin{aligned}\nabla^4 u_{0,0} &= \frac{1}{6h^4} [-(u_{0,3} + u_{0,-3} + u_{3,0} + u_{-3,0}) \\ &\quad + 14(u_{0,2} + u_{0,-2} + u_{2,0} + u_{-2,0}) \\ &\quad - 77(u_{0,1} + u_{0,-1} + u_{1,0} + u_{-1,0}) \\ &\quad + 184u_{0,0} + 20(u_{1,1} + u_{1,-1} + u_{-1,1} + u_{-1,-1}) \\ &\quad - (u_{1,2} + u_{2,1} + u_{1,-2} + u_{2,-1} + u_{-1,2} + u_{-2,1} \\ &\quad \quad + u_{-1,-2} + u_{-2,-1})] + O(h^6)\end{aligned}$$

25.4. Integration

Trapezoidal Rule

25.4.1

$$\int_{x_0}^{x_1} f(x) dx = \frac{h}{2} (f_0 + f_1) - \frac{1}{2} \int_{x_0}^{x_1} (t - x_0)(x_1 - t) f''(t) dt$$

# Numerical integration (or “quadrature”)

**Newton-Cotes formulas** of the closed type, for functions sampled at equidistant points.

For  $\{i \in \mathbb{N} \mid 0 \leq i \leq n\}$ , let  $x_i = a + i \frac{b-a}{n} = a + i h$ , and  $f_i = f(x_i)$ : then the integral can be approximated with a sum

$$\int_a^b f(x) dx \approx \sum_{i=0}^n w_i f(x_i)$$

# Numerical integration (or “quadrature”)

**Newton-Cotes formulas** of the closed type, for functions sampled at equidistant points.

For  $\{i \in \mathbb{N} \mid 0 \leq i \leq n\}$ , let  $x_i = a + i \frac{b-a}{n} = a + ih$ , and  $f_i = f(x_i)$ : then the integral can be approximated with a sum

$$\int_a^b f(x) dx \approx \sum_{i=0}^n w_i f(x_i)$$

where:

$n$	Step size $h$	Common name	Formula	Error
1	$b - a$	Trapezoidal rule	$\frac{h}{2} (f_0 + f_1)$	$-\frac{1}{12} h^3 f^{(2)}(\xi)$
2	$\frac{b-a}{2}$	Simpson's rule	$\frac{h}{3} (f_0 + 4f_1 + f_2)$	$-\frac{1}{90} h^5 f^{(4)}(\xi)$
3	$\frac{b-a}{3}$	Simpson's 3/8 rule	$\frac{3h}{8} (f_0 + 3f_1 + 3f_2 + f_3)$	$-\frac{3}{80} h^5 f^{(4)}(\xi)$
4	$\frac{b-a}{4}$	Boole's rule	$\frac{2h}{45} (7f_0 + 32f_1 + 12f_2 + 32f_3 + 7f_4)$	$-\frac{8}{945} h^7 f^{(6)}(\xi)$

# Numerical integration /II

For a function that is known analytically but cannot be integrated, one can use the Gauss-Legendre integration.

$$\int_a^b f(x) dx \approx \sum_{i=0}^n w_i f(x_i)$$

where:

- the function is defined in the interval  $[-1, 1]$
- $w_i = \frac{2}{(1 - x_i)^2 [P'_n(x_i)]^2}$
- $P_n(x)$  are the Legendre polynomials, normalised such that  $P_n(1) = 1$
- $x_i$  is the  $i$ -th root of  $P_n$

# Numerical integration /III

Abramowitz and Stegun suggest a generalisation to address different cases:

$$\int_a^b \omega(x) f(x) dx \approx \sum_{i=0}^n w_i f(x_i)$$

where:

- $\omega(x)$  is positive weight function
- $w_i$  depend on the method used
- $x_i$  is the  $i$ -th root of  $P_n$

Interval	$\omega(x)$	Orthogonal polynomials	A & S
$[-1, 1]$	1	<a href="#">Legendre polynomials</a>	25.4.29
$(-1, 1)$	$(1-x)^\alpha(1+x)^\beta$ , $\alpha, \beta > -1$	<a href="#">Jacobi polynomials</a>	25.4.33 ( $\beta = 0$ )
$(-1, 1)$	$\frac{1}{\sqrt{1-x^2}}$	<a href="#">Chebyshev polynomials (first kind)</a>	25.4.38
$[-1, 1]$	$\sqrt{1-x^2}$	<a href="#">Chebyshev polynomials (second kind)</a>	25.4.40
$[0, \infty)$	$e^{-x}$	<a href="#">Laguerre polynomials</a>	25.4.45
$[0, \infty)$	$x^\alpha e^{-x}$ , $\alpha > -1$	<a href="#">Generalized Laguerre polynomials</a>	
$(-\infty, \infty)$	$e^{-x^2}$	<a href="#">Hermite polynomials</a>	25.4.46

For more details see A & S.

# Random numbers

Random number generators are widely used in numerical physics. They are at the base of each Monte Carlo technique and are often the only practical way to evaluate difficult integrals or to sample random variables governed by complicated probability density functions.

It might seem impossible to produce random numbers through deterministic algorithms. Nevertheless, computer “random number generators” are in everyday use. Oftentimes, computer-generated sequences are called **Pseudo-random**, while the word **random** is reserved for the output of an intrinsically random physical process, like the elapsed time between clicks of a Geiger counter placed next to a sample of some radioactive element. Entire books have been dedicated to this topic, most notably Knuth’s “The Art of Computer Programming, Volume 2: Seminumerical Algorithms”. They are based on the idea generating a sequence from a “seed”.

There exist also **Quasi-random** sequences are sequences that progressively cover a  $N$ -dimensional space with a set of points that are uniformly distributed. Quasi-random sequences are also known as low-discrepancy sequences. Unlike pseudo-random sequences, quasi-random sequences fail many statistical tests for randomness. Approximating true randomness, however, is not their goal. Quasi-random sequences seek to fill space uniformly and do so so that initial segments approximate this behaviour up to a specified density.

# Pseudo-random numbers

An implementation of Donald E. Knuth's "Super-random" number generator given as an educational example of a (bad) pseudo random number generator in chapter 3.1 of his "The Art of Computer programming, Volume 2" book.

- K1.** [Choose number of iterations.] Set  $Y \leftarrow \lfloor X/10^9 \rfloor$ , the most significant digit of  $X$ . (We will execute steps K2 through K13 exactly  $Y + 1$  times; that is, we will apply randomizing transformations a *random* number of times.)
- K2.** [Choose random step.] Set  $Z \leftarrow \lfloor X/10^8 \rfloor \bmod 10$ , the second most significant digit of  $X$ . Go to step K( $3 + Z$ ). (That is, we now jump to a *random* step in the program.)
- K3.** [Ensure  $\geq 5 \times 10^9$ .] If  $X < 5000000000$ , set  $X \leftarrow X + 5000000000$ .
- K4.** [Middle square.] Replace  $X$  by  $\lfloor X^2/10^5 \rfloor \bmod 10^{10}$ , that is, by the middle of the square of  $X$ .
- K5.** [Multiply.] Replace  $X$  by  $(1001001001 X) \bmod 10^{10}$ .
- K6.** [Pseudo-complement.] If  $X < 1000000000$ , then set  $X \leftarrow X + 9814055677$ ; otherwise set  $X \leftarrow 10^{10} - X$ .
- K7.** [Interchange halves.] Interchange the low-order five digits of  $X$  with the high-order five digits; that is, set  $X \leftarrow 10^5(X \bmod 10^5) + \lfloor X/10^5 \rfloor$ , the middle 10 digits of  $(10^{10} + 1)X$ .
- K8.** [Multiply.] Same as step K5.
- K9.** [Decrease digits.] Decrease each nonzero digit of the decimal representation of  $X$  by one.
- K10.** [99999 modify.] If  $X < 10^5$ , set  $X \leftarrow X^2 + 99999$ ; otherwise set  $X \leftarrow X - 99999$ .
- K11.** [Normalize.] (At this point  $X$  cannot be zero.) If  $X < 10^9$ , set  $X \leftarrow 10X$  and repeat this step.
- K12.** [Modified middle square.] Replace  $X$  by  $\lfloor X(X - 1)/10^5 \rfloor \bmod 10^{10}$ , that is, by the middle 10 digits of  $X(X - 1)$ .
- K13.** [Repeat?] If  $Y > 0$ , decrease  $Y$  by 1 and return to step K2. If  $Y = 0$ , the algorithm terminates with  $X$  as the desired "random" value. ■

# Quasi-random numbers

There exist several algorithm to generate quasi-random numbers. For instance, the Halton and reverse Halton sequence, described in J.H. Halton, *Numerische Mathematik*, 2, 84-90 (1960) and B. Vandewoestyne and R. Cools, *Computational and Applied Mathematics*, 189, 1&2, 341-361 (2006), valid up to 1229 dimensions.

## Algorithm

```
algorithm Halton-Sequence is
```

```
  inputs: index  $i$ 
```

```
         base  $b$ 
```

```
  output: result  $r$ 
```

```
   $f \leftarrow 1$ 
```

```
   $r \leftarrow 0$ 
```

```
  while  $i > 0$  do
```

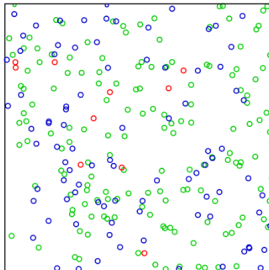
```
     $f \leftarrow f/b$ 
```

```
     $r \leftarrow r + f * (i \bmod b)$ 
```

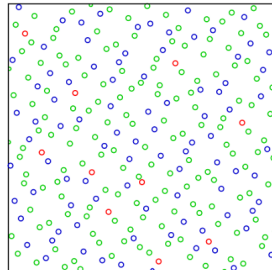
```
     $i \leftarrow \lfloor i/b \rfloor$ 
```

```
  return  $r$ 
```

## Pseudorandom

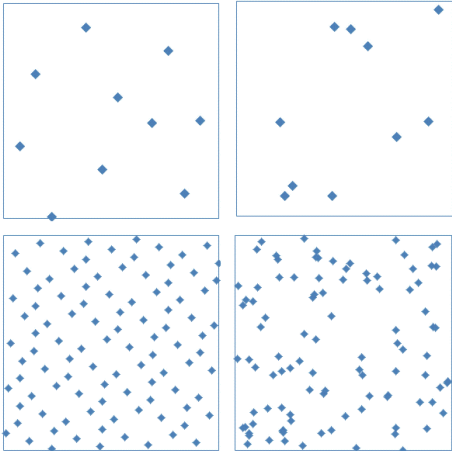


## Halton Sequence

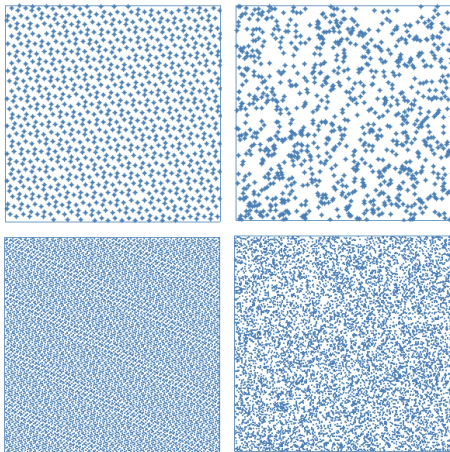




# Quasi-random vs Pseudo-random numbers



# Quasi-random vs Pseudo-random numbers



# Monte Carlo integration in many dimensions

We mentioned integration using quadrature formulae. For multi-dimensional functions, the most effective solution is to use a Monte Carlo integration:

$$I = \iiint\iiint f(x, y, z, t, u, v, w) dV$$

where  $dV$  is the multi-dimensional volume element.

$$I = V \times \langle f(x, y, z, t, u, v, w) \rangle$$

Here,  $V = \iiint\iiint dV$  is the volume of the entire domain, and

$$\langle f(x, y, z, t, u, v, w) \rangle$$

is the average function over such a domain, *sampled uniformly* over each dimension.

Quasi-random numbers help to sample the domain while introducing minimal numerical noise.

# Quasi-random generator

Example of Quasi-random generator for Octave, based on GSL (Gnu Scientific Library)

```
#include <octave/oct.h>
#include <gsl/gsl_qrng.h>

DEFUN_DLD (qrand, args, nargout, "Quasi-random generator")
{
  Matrix retval;

  if (args.length () < 2) {
    print_usage ();
  } else {
    const int nl = args(0).int_value();
    const int nc = args(1).int_value();
    gsl_qrng *qrng = gsl_qrng_alloc (gsl_qrng_halton, nc);
    if (qrng) {
      retval.resize (nl, nc);
      for (int i=0; i<nl; i++) {
        double tmp[nc];
        gsl_qrng_get (qrng, tmp);
        for (int j=0; j < nc; j++) {
          retval(i,j) = tmp[j];
        }
      }
      gsl_qrng_free (qrng);
    }
  }

  return octave_value (retval);
}
```

```
octave:1> qrand(10,4)
ans =

  0.500000  0.333333  0.200000  0.142857
  0.250000  0.666667  0.400000  0.285714
  0.750000  0.111111  0.600000  0.428571
  0.125000  0.444444  0.800000  0.571429
  0.625000  0.777778  0.040000  0.714286
  0.375000  0.222222  0.240000  0.857143
  0.875000  0.555556  0.440000  0.020408
  0.062500  0.888889  0.640000  0.163265
  0.562500  0.037037  0.840000  0.306122
  0.312500  0.370370  0.080000  0.448980

octave:2> |
```

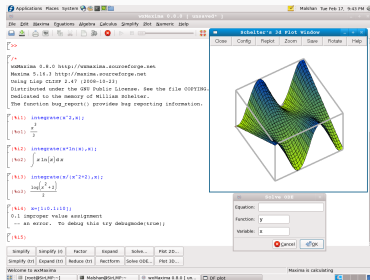
```
$ mkoctfile qrand.cc -o qrand.oct
```

# Exact and arbitrary-precision numbers

In cases where double-, extended- or even quadruple-precision are not enough, there exist a couple of solutions to achieve higher precision and in some cases even exact results.

- **Symbolic calculation** is the “holy grail” of exact calculations.

Programs such as Maxima, Mathematica<sup>®</sup>, or Maple<sup>®</sup>, know the rules of math and represents data as symbols rather rounded numbers. It is free software released under the terms of the GNU General Public License (GPL). An excellent front end for Maxima is wxMaxima



- **Arbitrary-precision arithmetic** can be achieved using dedicated libraries that can handle arbitrary, user-defined precision such as GMP, the GNU Multiple Precision Arithmetic Library for the C and C++ programming languages.



# Tools: Python vs Octave

**Python** is described as “A clear and powerful object-oriented programming language, comparable to Perl, Ruby, Scheme, or Java”. Python is a general purpose programming language created by Guido Van Rossum.

Libraries such as numpy, matplotlib, pandas offer many functionalities that make it similar to MATLAB and Octave.

# Tools: Python vs Octave

**Python** is described as “A clear and powerful object-oriented programming language, comparable to Perl, Ruby, Scheme, or Java”. Python is a general purpose programming language created by Guido Van Rossum.

Libraries such as numpy, matplotlib, pandas offer many functionalities that make it similar to MATLAB and Octave.

**Octave** is detailed as “A programming language for scientific computing”. It is software featuring a high-level programming language, primarily intended for numerical computations. Octave helps in solving linear and nonlinear problems numerically, and for performing other numerical experiments using a language that is mostly compatible with MATLAB.

# Tools: Python vs Octave

**Python** is described as “A clear and powerful object-oriented programming language, comparable to Perl, Ruby, Scheme, or Java”. Python is a general purpose programming language created by Guido Van Rossum.

Libraries such as numpy, matplotlib, pandas offer many functionalities that make it similar to MATLAB and Octave.

**Octave** is detailed as “A programming language for scientific computing”. It is software featuring a high-level programming language, primarily intended for numerical computations. Octave helps in solving linear and nonlinear problems numerically, and for performing other numerical experiments using a language that is mostly compatible with MATLAB.

<https://www.octave.org>

<https://octave.sourceforge.io>



# Tools: Python vs Octave, which one?

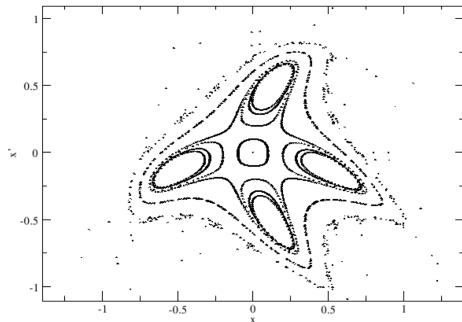
«Both **Python** and **Octave** are commonly used for scientific computing, but there are some differences between them that may make one a better choice depending on the specific needs of a project.

**Python** is a more general-purpose programming language with a wide range of libraries and frameworks that can be used for scientific computing. Some of the most popular libraries for scientific computing in **Python** include NumPy, SciPy, Pandas, and Matplotlib. **Python** also has a large and active community of developers, which means that it is easier to find support, resources, and tools.

**Octave**, on the other hand, is specifically designed for numerical and scientific computing, with a syntax that is similar to MATLAB. **Octave** has built-in support for matrix operations and linear algebra, making it well-suited for numerical computations. **Octave** is also open-source, which means that it is freely available and can be modified and distributed as needed.

In general, if your project requires a more general-purpose language or if you need access to a wide range of libraries and tools beyond just scientific computing, then **Python** may be the better choice. However, if your project is primarily focused on numerical computations, then **Octave** may be more appropriate. Ultimately, the choice between **Python** and **Octave** will depend on the specific needs and goals of your project.»

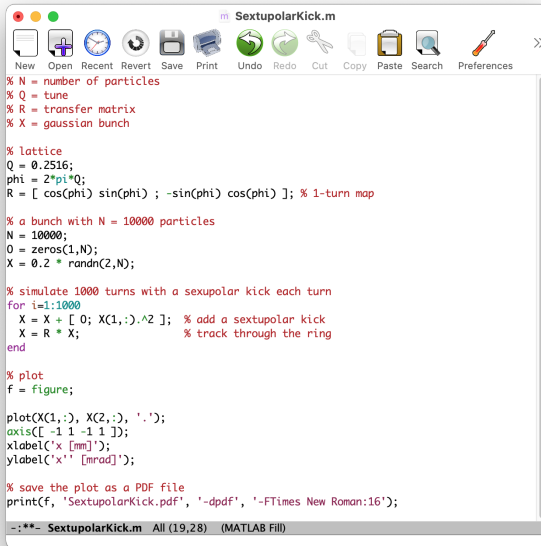
## Example: impact of nonlinear elements on linear optics



- $Q=0.2516$
- linear motion near center (circles)
- More and more square
- Non-linear tunes shift
- Islands
- Limit of stability
- Dynamic Aperture
- Crucial if strong quads and chromaticity correction in s.r. light sources
- many non-linearities in LHC due to s.c. magnet and finite manufacturing tolerances

$$\begin{pmatrix} x_{n+1} \\ x'_{n+1} \end{pmatrix} = \begin{pmatrix} \cos(2\pi Q) & \sin(2\pi Q) \\ -\sin(2\pi Q) & \cos(2\pi Q) \end{pmatrix} \begin{pmatrix} x_n \\ x'_n + x_n^2 \end{pmatrix}$$

# Tools: Octave simulation



```
% N = number of particles
% Q = tune
% R = transfer matrix
% X = gaussian bunch

% lattice
Q = 0.2516;
phi = 2*pi*Q;
R = [ cos(phi) sin(phi) ; -sin(phi) cos(phi) ]; % 1-turn map

% a bunch with N = 10000 particles
N = 10000;
O = zeros(1,N);
X = 0.2 * randn(2,N);

% simulate 1000 turns with a sexupolar kick each turn
for i=1:1000
    X = X + [ 0; X(1,:).^2 ]; % add a sextupolar kick
    X = R * X; % track through the ring
end

% plot
f = figure;

plot(X(1,:), X(2,:), '.');
axis([ -1 1 -1 1 ]);
xlabel('x [mm]');
ylabel('x'' [mrad]');

% save the plot as a PDF file
print(f, 'SextupolarKick.pdf', '-dpdf', '-FTimes New Roman:16');
```

--:\*\*- SextupolarKick.m All (19,28) (MATLAB Fill)

# Tools: Symbolic computation

## Maxima (and wxMaxima)

Maxima is a computer algebra system with a long history. It is based on a 1982 version of Macsyma.

It is written in Common Lisp and runs on all POSIX platforms such as macOS, Unix, BSD, and Linux, as well as under Microsoft Windows and Android.

It is free software released under the terms of the GNU General Public License (GPL). It is a valid alternative to commercial alternatives, and offers some advantage.

wxMaxima is an excellent front end for Maxima.



# Tools: Symbolic computation

## Maxima (and wxMaxima)

Maxima is a computer algebra system with a long history. It is based on a 1982 version of Macsyma.

It is written in Common Lisp and runs on all POSIX platforms such as macOS, Unix, BSD, and Linux, as well as under Microsoft Windows and Android.

It is free software released under the terms of the GNU General Public License (GPL). It is a valid alternative to commercial alternatives, and offers some advantage.

wxMaxima is an excellent front end for Maxima.

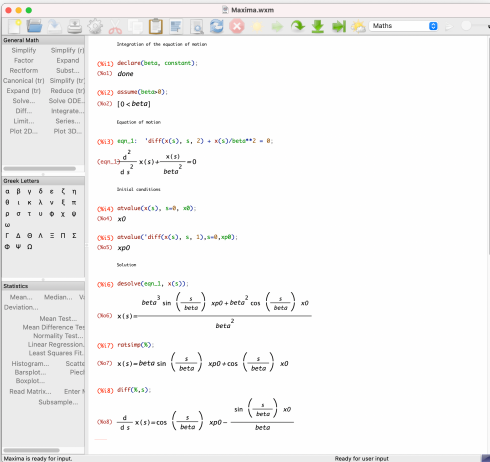


## Octave and Python

Symbolic computations can also be performed within Octave and Python. Dedicated packages add the possibility to perform basic symbolic computations, including common Computer Algebra System tools such as algebraic operations, calculus, equation solving, Fourier and Laplace transforms, variable precision arithmetic and other features, in scripts.

# Tools: Symbolic computation

## A 1D harmonic oscillator with wxMaxima



The screenshot shows the wxMaxima interface with the following content:

Integration of the equation of motion

```
(M1) declare(beta, constant);
(M2) done
(M3) assume(beta<0);
(M4) [0 < beta]
```

Equation of motion

```
(M5) eqn_1: 'diff(x(s), s, 2) + x(s)/beta**2 = 0;
(eqn_1)  $\frac{d^2}{ds^2} x(s) + \frac{x(s)}{\beta^2} = 0$ 
```

Initial conditions

```
(M6) atvalue(x(s), s=0, x0);
(M7) x0
(M8) atvalue('diff(x(s), s, 1), s=0, v0);
(M9) v0
```

Solution

```
(M10) desolve(eqn_1, x(s));
```

$$(M10) x(s) = \frac{\beta^2 \sin\left(\frac{s}{\beta}\right) x_0 + \beta^2 \cos\left(\frac{s}{\beta}\right) v_0}{\beta^2}$$

```
(M11) ratsimp(M10);
(M12) x(s) = beta sin(s/beta) x_0 + cos(s/beta) v_0
(M13) diff(M12, s);
(M14)  $\frac{d}{ds} x(s) = \cos\left(\frac{s}{\beta}\right) x_0 - \frac{\sin\left(\frac{s}{\beta}\right) v_0}{\beta}$ 
```

Maxima is ready for input. Ready for user input

# Tools: Symbolic computation

## FODO cell in wxMaxima

The screenshot shows the wxMaxima interface with the following content:

**General Math:**

- Simplify (S)
- Factor (F)
- Rectform (R)
- Canonical (C)
- Solve (S)
- Diff (D)
- Limit (L)
- Plot 2D (P)
- Simplify (S)
- Expand (E)
- Subst... (S)
- Simplify (S)
- Reduce (R)
- Solve ODE (S)
- Integrate (I)
- Series (S)
- Plot 3D (P)

**Great Letters:**

**Statistics:**

- Mean... (M)
- Median... (M)
- Deviation... (D)
- Mean Test... (M)
- Mean Difference Test... (M)
- Normality Test... (N)
- Linear Regression (L)
- Least Squares Fit (L)
- Histogram... (H)
- Scatter (S)
- Residuals... (R)
- Residuals... (R)
- Read Matrix... (R)
- Enter... (E)
- Subsample... (S)

**Transfer matrices**

(%i1)  $Q(F) := \text{matrix}([1, 0], [1/f, 1]);$

(%o1)  $Q(f) := \begin{pmatrix} 1 & 0 \\ 1/f & 1 \end{pmatrix}$

(%i2)  $D(L) := \text{matrix}([1, L], [0, 1]);$

(%o2)  $D(L) := \begin{pmatrix} 1 & L \\ 0 & 1 \end{pmatrix}$

**FODO Lattice**

(%i3)  $FODO := D(L) \cdot Q(F) \cdot D(L) \cdot Q(-F);$

(%i4)  $\text{ratsimp}(FODO);$

(%o4)  $\begin{pmatrix} f^2 - L f - L^2 & 2 L f + L^2 \\ -L & f + L \end{pmatrix}$

- + determinant(X);
- + ratsimp(X);

**Symplectic test (using the Jacobian matrix)**

- +  $J := \text{matrix}([0, 1], [-1, 0]);$
- +  $\text{transpose}(FODO) \cdot J \cdot FODO - J;$
- + ratsimp(X);

Maxima is ready for input. Ready for user input.

## The Octave “symbolic” package

```
1 % Load the symbolic package
2 pkg load symbolic
3
4 % This is just a formula to start with, have fun and change it if you want to.
5 f = @(x) x.^2 + 3*x - 1 + 5*x.*sin(x);
6
7 % These next lines take the Anonymous function into a symbolic formula
8 syms x;
9 ff = f(x);
10
11 % Now we can calculate the derivative of the function
12 ffd = diff(ff, x);
13
14 % and convert it back to an Anonymous function
15 df = function_handle(ffd)
```



# Tools: Symbolic computation

## The Octave “symbolic” package

```
1 % Load the symbolic package
2 pkg load symbolic
3
4 % This is just a formula to start with, have fun and change it if you want to.
5 f = @(x) x.^2 + 3*x - 1 + 5*x.*sin(x);
6
7 % These next lines take the Anonymous function into a symbolic formula
8 syms x;
9 ff = f(x);
10
11 % Now we can calculate the derivative of the function
12 ffd = diff(ff, x);
13
14 % and convert it back to an Anonymous function
15 df = function_handle(ffd)
```

## The Python “sympy” library

```
1 >>> from sympy import *
2 >>> x = symbols('x')
3 >>> simplify(sin(x)**2 + cos(x)**2)
4 1
```

# Shell scientific tools

## units

The ability to evaluate complex expressions involving units makes many computations easy to do, and the checking for compatibility of units guards against errors frequently made in scientific calculations. Units is a conversion program, but also calculator with units.

- Example 1: average beam power,  
bunch charge 300 pC, 15 GeV energy, 50 Hz repetition rate:

```
1 $ units -v
2 You have: 300 pC * 15 GV * 50 Hz
3 You want: W
4   300 pC * 15 GV * 50 Hz = 225 W
5   300 pC * 15 GV * 50 Hz = (1 / 0.004444444444444444) W
```

# Shell scientific tools

## units

The ability to evaluate complex expressions involving units makes many computations easy to do, and the checking for compatibility of units guards against errors frequently made in scientific calculations. Units is a conversion program, but also calculator with units.

- Example 1: average beam power,  
bunch charge 300 pC, 15 GeV energy, 50 Hz repetition rate:

```
1 $ units -v
2 You have: 300 pC * 15 GV * 50 Hz
3 You want: W
4 300 pC * 15 GV * 50 Hz = 225 W
5 300 pC * 15 GV * 50 Hz = (1 / 0.004444444444444444) W
```

- Example 2: beam size at the interaction point of an electron-positron collider,  
 $\sigma = \sqrt{\beta^* \cdot \epsilon_{\text{geometric}}}$ , with  $\beta^* = 1$  mm,  $\epsilon_{\text{normalized}} = 5$  nm,  $E = 1.5$  TeV:

```
1 You have: sqrt(0.001m * 5nm * electronmass c^2 / 1.5 TeV)
2 You want: nm
3 sqrt(0.001 m * 5 nm * electronmass c c / 1.5 TeV) = 1.305116 nm
4 sqrt(0.001 m * 5 nm * electronmass c c / 1.5 TeV) = (1 / 0.766214) nm
```

# An anecdote on units errors...

NEWS

## NASA's Failed Mars Missions That Cost Over \$200 Million



BY [GEORGINA TORBET](#) / NOV. 15, 2022 2:18 PM EST

On January 3, 1999, NASA launched what was going to be an exciting new mission to Mars: the [Mars Polar Lander](#). Designed to study the soil and climate of Mars' southern pole, the lander was accompanied by two smaller probes called Deep Space 2, which were intended to slam into the planet's surface at high speed and study the soil up close. The lander traveled through space as planned and arrived at Mars on December 3, 1999. The mission began the landing procedure and entered the atmosphere, but it never made contact again. The mission was declared lost and assumed to have crashed into the planet.

# An anecdote on units errors...

Another page gives more details:

On September 23, 1999, communication with the spacecraft was lost causing the spacecraft to go into orbital insertion. NASA discovered the failure was due to ground-based computer software that produced output in non-SI units of pound-seconds (lbf\*s) instead of the metric units of newton-seconds (N\*s) specified in the contract between NASA and Lockheed Martin. The spacecraft encountered Mars on a trajectory that brought it too close to the planet. It too passed through the upper atmosphere and disintegrated.

Easy, with units:

```
[$ units -v
Currency exchange rates from FloatRates (USD base) on 2022-09-05
3753 units, 113 prefixes, 120 nonlinear units

[You have: lbf * s
[You want: N * s
      lbf * s = 4.4482216152604996 N * s
      lbf * s = (1 / 0.2248089430997105) N * s
[You have:
$ █
```

# Shell scientific tools: arbitrary precision

## bc

It's a programmable shell calculator that supports arbitrary-precision numbers

```
1 $ bc
2 bc 1.06
3 Copyright 1991-1994, 1997, 1998, 2000 Free Software Foundation, Inc.
4 This is free software with ABSOLUTELY NO WARRANTY.
5 For details type 'warranty'.
6 scale=1
7 sqrt(2)
8 1.4
9 scale=40
10 sqrt(2)
11 1.4142135623730950488016887242096980785696
```

The variable “scale” allows one to select the total number of decimal digits after the decimal

# Shell scientific tools: arbitrary precision

## bc

It's a programmable shell calculator that supports arbitrary-precision numbers

```
1 $ bc
2 bc 1.06
3 Copyright 1991-1994, 1997, 1998, 2000 Free Software Foundation, Inc.
4 This is free software with ABSOLUTELY NO WARRANTY.
5 For details type 'warranty'.
6 scale=1
7 sqrt(2)
8 1.4
9 scale=40
10 sqrt(2)
11 1.4142135623730950488016887242096980785696
```

The variable “scale” allows one to select the total number of decimal digits after the decimal

## gnuplot

It's a portable command-line-driven graphing utility originally created to allow scientists and students to visualise mathematical functions and data interactively. It has grown to support many non-interactive uses. It implements excellent fitting routines.

# Useful linux tools

## Use of named pipes for interprocess communication (FIFOs)

Let's see how to create and use a named pipe:

```
1 $ mkfifo mypipe
2 $ ls -l mypipe
3 prw-r-----. 1 myself staff 0 Jan 31 13:59 mypipe
```

Notice the special file type designation of "p" and the file length of zero. You can write to a named pipe by redirecting output to it and the length will still be zero.

```
1 $ echo "Can you read this?" > mypipe
2 $ ls -l mypipe
3 prw-r-----. 1 myself staff 0 Jan 31 13:59 mypipe
```

So far, so good, but hit return and nothing much happens. While it might not be obvious, your text has entered into the pipe, but you're still peeking into the input end of it. You or someone else may be sitting at the output end and be ready to read the data that's being poured into the pipe, now waiting for it to be read.

```
1 $ cat mypipe
2 Can you read this?
```

Once read, the contents of the pipe are gone.



# A word about the choice of units...

The International System (SI) wasn't created for accelerator applications. The beam size isn't of the order of meters, the force shouldn't be expressed in Newtons.

Example:

Let's compute the force exerted by one of the LHC superconductive dipoles, in Newton:

```
1 $ units -v
2 You have: c * e * 8.5 T
3 You want: N
4 c * e * 8.5 T = 4.082724005684724e-10 N
5 c * e * 8.5 T = (1 / 2449345090.698306) N
```

Example of "practical" units:

quantity	units	quantity	units	quantity	units
position	mm	energy	MeV	momentum	MeV/c
angles	mrاد	time	mm/c	force	MeV/m

In fact,

$$c * e * 8.5 \text{ T} = 2548.235893 \text{ MeV/m}$$

# A word about data files...

An example of questionable choice:

```
$> tail ASTRA_distr_at_cathode.ini
 1.0700E-04  4.5476E-04  0.0000E+00  4.2084E+02 -7.2666E+02  5.8264E+02 -1.3444E-03 -6.1600E-06  1 -1
-9.7655E-04 -7.9998E-04  0.0000E+00  7.0706E+02 -8.1272E+01  7.2854E+02  1.0902E-03 -6.1600E-06  1 -1
-2.4085E-04  1.9553E-04  0.0000E+00 -4.0670E+02 -2.4789E+02  9.7767E+02 -6.3584E-04 -6.1600E-06  1 -1
-1.4163E-04 -3.7871E-04  0.0000E+00 -9.3474E+02 -4.2633E+02  3.4603E+02  2.2061E-04 -6.1600E-06  1 -1
-2.8669E-04  1.1817E-05  0.0000E+00 -9.4943E+02 -2.6439E+02  3.1839E+02 -1.8247E-03 -6.1600E-06  1 -1
 9.1701E-04 -3.8281E-04  0.0000E+00 -7.7430E+02  4.2284E+02  1.2272E+02 -1.5230E-03 -6.1600E-06  1 -1
 2.2139E-04  1.0007E-04  0.0000E+00  2.8189E+02  1.0234E+02  4.5108E+02 -1.7515E-03 -6.1600E-06  1 -1
 4.4429E-04 -8.8646E-05  0.0000E+00 -2.0888E+02  3.8810E+02  7.3747E+02 -9.6443E-05 -6.1600E-06  1 -1
-4.6858E-04  4.6416E-04  0.0000E+00  1.8117E+02  7.8001E+02  4.9974E+02 -1.5143E-03 -6.1600E-06  1 -1
 2.8663E-04  4.0295E-04  0.0000E+00 -4.4856E+02 -5.0962E+02  3.8406E+02  7.8835E-04 -6.1600E-06  1 -1
$> █
```

# A word about data files...

An example of questionable choice:

```
$> tail ASTRA_distr_at_cathode.ini
 1.0700E-04  4.5476E-04  0.0000E+00  4.2084E+02 -7.2666E+02  5.8264E+02 -1.3444E-03 -6.1600E-06  1 -1
-9.7655E-04 -7.9998E-04  0.0000E+00  7.0706E+02 -8.1272E+01  7.2854E+02  1.0902E-03 -6.1600E-06  1 -1
-2.4085E-04  1.9553E-04  0.0000E+00 -4.0670E+02 -2.4789E+02  9.7767E+02 -6.3584E-04 -6.1600E-06  1 -1
-1.4163E-04 -3.7871E-04  0.0000E+00 -9.3474E+02 -4.2633E+02  3.4603E+02  2.2061E-04 -6.1600E-06  1 -1
-2.8669E-04  1.1817E-05  0.0000E+00 -9.4943E+02 -2.6439E+02  3.1839E+02 -1.8247E-03 -6.1600E-06  1 -1
 9.1701E-04 -3.8281E-04  0.0000E+00 -7.7430E+02  4.2284E+02  1.2272E+02 -1.5230E-03 -6.1600E-06  1 -1
 2.2139E-04  1.0007E-04  0.0000E+00  2.8189E+02  1.0234E+02  4.5108E+02 -1.7515E-03 -6.1600E-06  1 -1
 4.4429E-04 -8.8646E-05  0.0000E+00 -2.0888E+02  3.8810E+02  7.3747E+02 -9.6443E-05 -6.1600E-06  1 -1
-4.6858E-04  4.6416E-04  0.0000E+00  1.8117E+02  7.8001E+02  4.9974E+02 -1.5143E-03 -6.1600E-06  1 -1
 2.8663E-04  4.0295E-04  0.0000E+00 -4.4856E+02 -5.0962E+02  3.8406E+02  7.8835E-04 -6.1600E-06  1 -1
$>
```

In C, use:

```
printf("%.17g\n", x);
```

In C++, use:

```
std::cout << std::setprecision(17) << x << std::endl;
```

...to preserve information bit by bit.

# Accelerator physics codes: storage rings

## MAD-X

MAD-X is a CERN code used world-wide, started in the 80's in the field of high energy beam physics (i.e. MAD8, MAD9, MADX). All-in-one application with its own scripting language used to design, simulate and optimise particle accelerators: optics modelling, single particles 6D tracking, machine survey, synchrotron radiation, aperture margin and emittance equilibrium. [rings, optics, tracking]

## MAD-NG

MAD-NG is a recent CERN code aiming to replace MAD-X. Symplectic integration of differential maps, single-particle tracking, optics calculations. Uses Lua as a scripting language. [rings, optics, tracking]

## SixTrack

CERN's single-particle 6D symplectic tracking code optimised for long term tracking in high energy rings. Uses its own description language. [rings, tracking]

## PyHEADTAIL

Python macro-particle simulation code library developed at CERN for modelling collective effects beam dynamics in circular accelerators. Interfaced with Python. [rings, tracking, collective effects]

## Xsuite

Modern integrated suite of accelerator tools. Runs on GPUs, integrates all of the above [rings, tracking, collective effects]

# Accelerator physics codes: linacs (also recirculating ones)

## **PLACET / PLACET2**

The “Program for Linear Accelerator Correction and Efficiency Tests”, is a code developed at CERN that simulates the dynamics of a beam in the main accelerating or decelerating part of a linac (CLIC) in the presence of wakefields. It allows for recirculating layouts. Recently adapted for muon tracking. It includes the emission of incoherent and coherent synchrotron radiation. Interfaced with Tcl, Octave, and Python. [linacs, tracking, collective effects, imperfections]

## **ELEGANT**

The “ELEctron Generation ANd Tracking”, it’s a code developed at the Argonne National Laboratory (ANL, USA) that can generate particle distributions, track them, and perform optics calculations. Uses its own description language. [linacs & rings, tracking, optics]

# Accelerator physics codes: injectors, and more exotic scenarios...

## **ASTRA**

“A Space Charge Tracking Algorithm” is a tracking code developed at DESY (Hamburg, Germany), can simulate injectors and track in field maps. Simulated photocathodes. Uses its own description language. [injectors, tracking, space charge]

## **RF-Track**

RF-Track was developed at CERN, to simulate beams of particles with arbitrary energy, mass, and charge, even mixed, in field maps and conventional elements. It can simulate space-charge, short- and long-range wakefields, electron cooling, inverse Compton scattering, beam loading, multiple Coulomb scattering. Simulates photocathodes. Interfaced with Python and Octave. [injectors & linacs, tracking, collective effects, design, imperfections]

And others, but some aren't maintained or they are not open-source and free...

# High-performance computing

## Parallelism

Parallelism can be achieved in different ways, depending on the problem:

1. **“Embarrassingly parallel” problems.** “Embarrassingly parallel” problems are those where a large number of tasks need to be performed, with each single task being completely independent of the others. Examples: imperfections studies; tracking of single-particles.
2. **MPI - “massively parallel” problems**

MPI, the Message Passing Interface, is just a protocol, to design codes that run on clusters of computers.

There exist several open-source implementations of MPI, which fostered the development of a parallel software industry, and encouraged development of portable and scalable large-scale parallel applications. Two well-established MPI implementations are “Open MPI” and “MPICH”.

```
1 #include <mpi.h>
2 #include <stdio.h>
3
4 int main()
5 {
6     // Initialize the MPI environment
7     MPI_Init(NULL, NULL);
8
9     // Get the number of processes
10    int world_size;
11    MPI_Comm_size(MPI_COMM_WORLD, &world_size);
12
13    // Get the rank of the process
14    int world_rank;
15    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
16
17    // Get the name of the processor
18    char processor_name[MPI_MAX_PROCESSOR_NAME];
19    int name_len;
20    MPI_Get_processor_name(processor_name, &name_len);
21
22    // Print off a hello world message
23    printf("Hello world from processor %s, rank %d out of %d processors\n",
24           processor_name, world_rank, world_size);
25
26    // Finalize the MPI environment.
27    MPI_Finalize();
28 }
```

# High-performance computing

## Parallelism

### 3. OpenMP. Multi-core parallelism. Hacking an existing code to make it parallel.

OpenMP is a programming interface that supports multi-platform shared-memory multiprocessing programming in C, C++, and Fortran. In simpler words, it makes programs run in parallel on multi-cores computers, exploiting the multi-threaded architecture of modern CPUs.

```
1 int main()
2 {
3     int a[100000];
4
5     #pragma omp parallel for
6     for (int i = 0; i < 100000; i++) {
7         a[i] = 2 * i;
8     }
9
10    return 0;
11 }
```

### 4. C++ threads

Since version C++11, the C++ language offers a set of classes to handle parallelism, synchronisation, and data exchange between threads. These functionalities are accessible using the class `std::thread`, defined in `<thread>`.

```
1 // thread example
2 #include <iostream>           // std::cout
3 #include <thread>            // std::thread
4
5 void foo()
6 {
7     // do stuff...
8 }
9
10 void bar(int x)
11 {
12     // do stuff...
13 }
14
15 int main()
16 {
17     std::thread first (foo);   // spawn new thread that calls foo()
18     std::thread second (bar,0); // spawn new thread that calls bar()
19
20     std::cout << "main, foo and bar now execute concurrently...\n";
21
22     // synchronize threads:
23     first.join();             // pauses until first finishes
24     second.join();           // pauses until second finishes
25
26     std::cout << "foo and bar completed.\n";
27
28     return 0;
29 }
```



# Scientific computing in C/C++

## C scientific library

- The GNU Scientific Library. The GNU Scientific Library (GSL) is an excellent numerical library written in C. It provides more than 1000 mathematical routines such as random number generators, special functions, least-squares fitting, etc. Uses BLAS and LAPACK for linear algebra functionality

## C++ template libraries

- Standard template library (STD), provides useful container classes, eg. `std::valarray<T>`. Dictionaries and associations are provided using `std::set<K>`, `std::map<K,T>`. Efficient algorithms, like `std::qsort`, are provided in `<algorithm>`.
- BOOST, a set of C++ template libraries that provides support for tasks and structures for numerical calculations and much more. More experimental and less mature than STD
- Armadillo, a high quality linear algebra template library providing a good balance between speed and ease of use. Syntax and functionality similar to Matlab and Octave
- Eigen, another C++ template library for linear algebra. Includes numerical solvers and related algorithms

# High-performance computing in C/C++

## Advanced programming

- Intel Intrinsics. By explicit vectorisation one can access specific instruction sets like MMX, SSE, SSE2, SSE3, AVX, AVX2. Most compilers will analyse your code and use these functions wherever possible.
- SWIG, is a software development tool that connects programs in C and C++ with a variety of high-level programming languages: Python, Octave, Tcl, Lua, ... [ Extremely useful ]

## GPU Programming

- OpenCL (Open Computing Language) is a framework for writing programs that execute across heterogeneous platforms consisting of central processing units (CPUs), graphics processing units (GPUs), digital signal processors (DSPs), field-programmable gate arrays (FPGAs) and other processors or hardware accelerators. OpenCL is an open standard maintained by the non-profit technology consortium Khronos Group. Conformant implementations are available from Altera, AMD, Apple, ARM, Creative, IBM, Imagination, Intel, Nvidia, Qualcomm, Samsung, Vivante, Xilinx, and ZiiLABS.
- CUDA (an acronym for Compute Unified Device Architecture) is a proprietary model created by Nvidia to program Nvidia GPUs for general purpose processing. The CUDA platform is a software layer that gives direct access to the GPU's virtual instruction set and parallel computational elements, for the execution of compute kernels.

The end.

Thank you for your attention!

Any questions?

**Acknowledgment:** Laurent Deniau (CERN / BE-ABP-LNO) for his suggestions and ideas.