Numerical Methods and Computational Tools in Accelerator Physics

Andrea Latina
andrea.latina@cern.ch

CAS - Introduction to Accelerator Physics - Santa Susanna, Spain - 2024

Table of contents

1. Introduction
    - Purpose
    - Some references
2. Computational methods
    - Machine precision
    - Numerical errors
        - Round-off
        - Cancellation
        - Truncation
3. Computational tools
    - Free and open-source tools for scientific computing
    - Overview of accelerator physics codes
    - Machine learning in accelerator physics

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 for scientific and accelerator physics computing, and explain which tool should be used for a specific purpose, dispelling common misconceptions, and suggesting 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 for scientific and accelerator physics computing, and explain which tool should be used for a specific purpose, dispelling common misconceptions, and suggesting 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

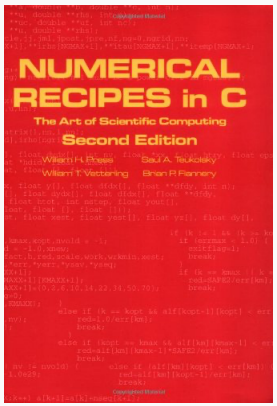
In these two lessons, we will outline some fundamental concepts in scientific computing and guide the novice through the multitude of tools available. We will describe the main tools for scientific and accelerator physics computing, and explain which tool should be used for a specific purpose, dispelling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

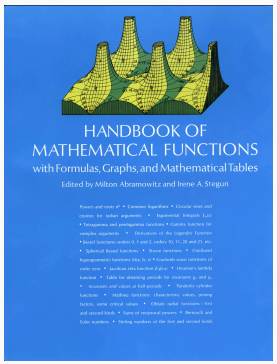
We will suggest reference readings and clarify important aspects of numerical stability to help avoid making bad but unfortunately common mistakes. Numerical stability should be basic knowledge of every scientist.

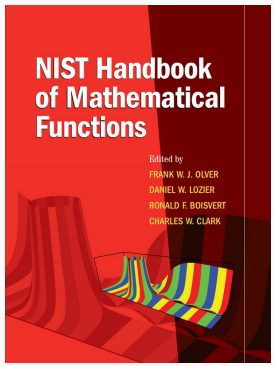
We will exclusively refer to free and open-source software running on Linux or other Unix-like operating systems. Also, we will unveil powerful shell commands that can speed up simulations, facilitate data processing, and in short, increase your scientific throughput.

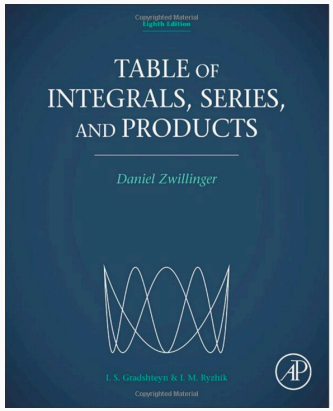
Some references

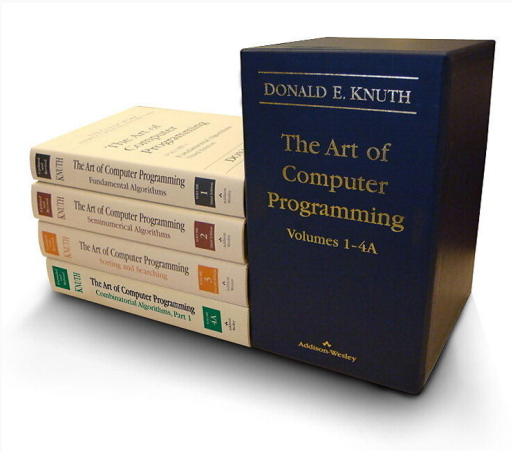
1. "*Numerical Recipes: The Art of Scientific Computing*", W. Press, S. Teukolsky, W. Vetterling, and B. Flannery, 1992 (2nd edition) – 2007 (3rd edition)
2. Abramowitz and Stegun, "*Handbook of Mathematical Functions with Formulas*", 1964
3. "*The NIST Handbook of Mathematical Functions*", Olver, Lozier, 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











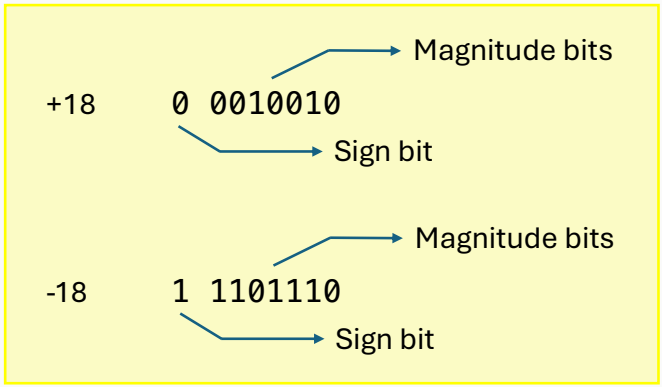
Part 1.

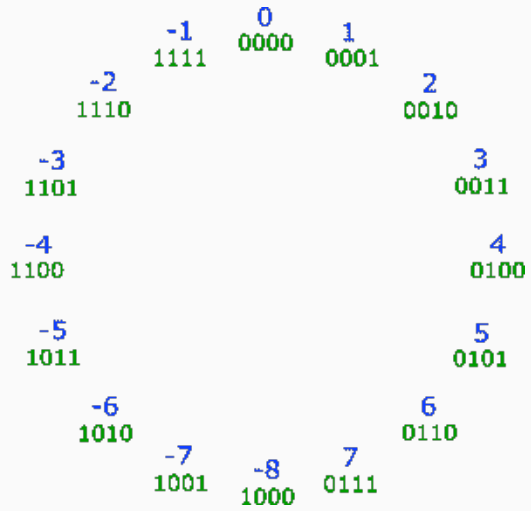
Primer on Computational Methods

Internal representation of numbers

Integers

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





Two's complement representation

- 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.

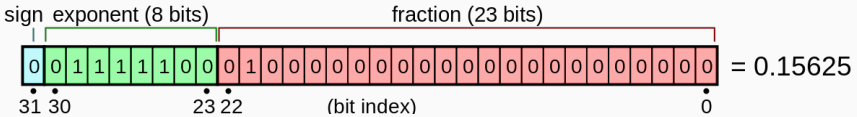
Internal representation of numbers

Real numbers

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

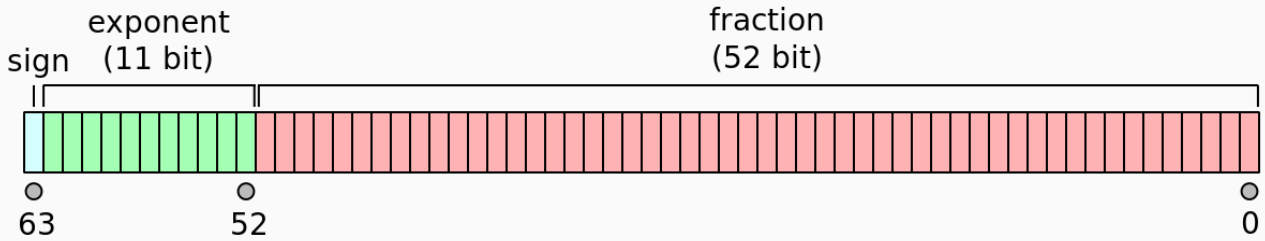
$$\text{value} = (-1)^{\text{sign}} \times 1.\text{fraction} \times 2^{\text{exponent}}$$

- Single-precision floating point representation (32 bits)



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

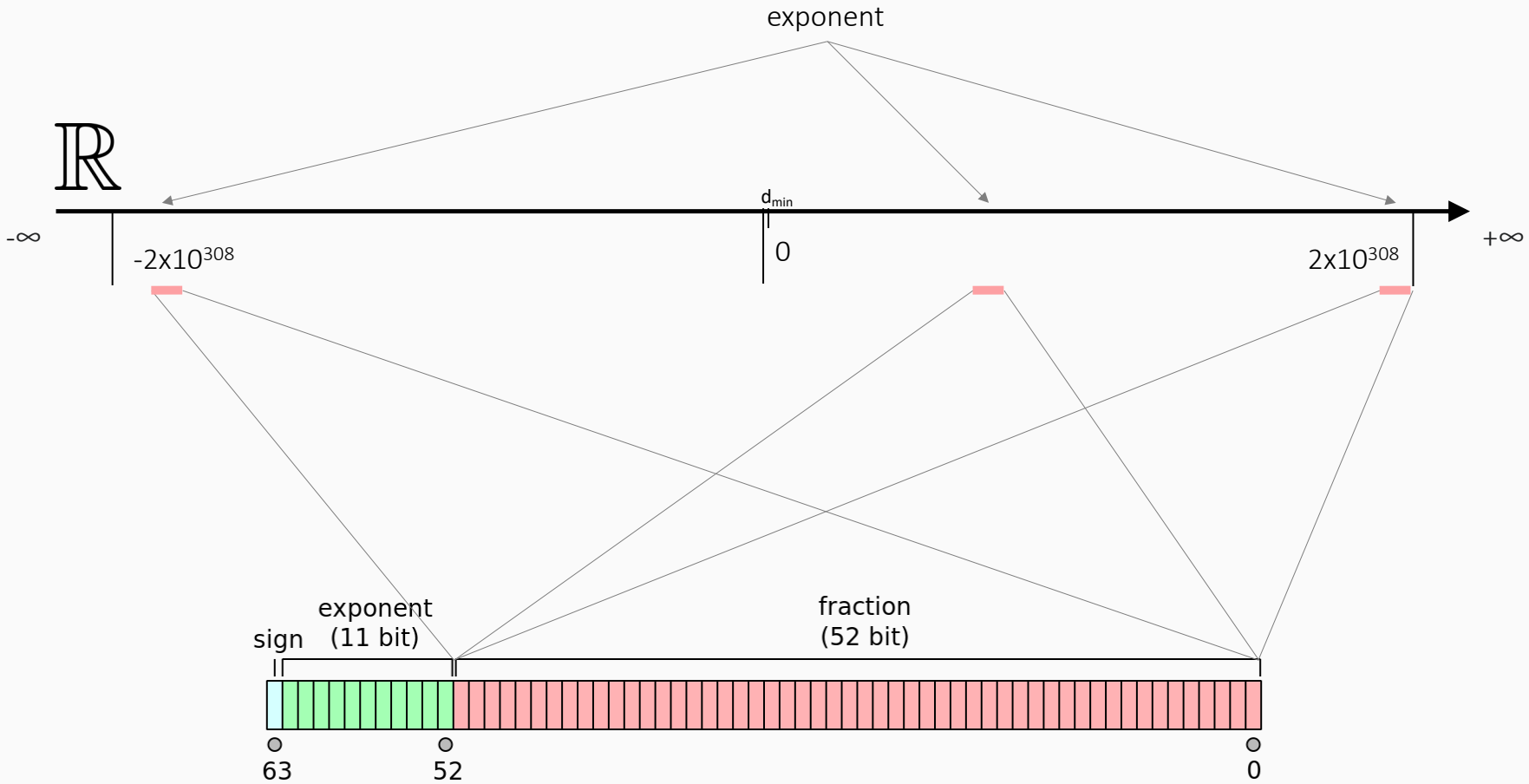
- Double-precision floating point representation (64 bits)



  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 $d_{\text{min}} = 2^{-1074} \approx 4.94 \times 10^{-324}$

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 == -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. That is, `a==a` is false if `a` is NaN.

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 == -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. That is, `a==a` is false if `a` is NaN.

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

C++:
```cpp
#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:
```python
f = numpy.inf
nan = numpy.nan
```

Octave:
```
f = inf;
n = nan;
```

Machine epsilon, ε_m

The *machine epsilon,* or *accuracy*, ε_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, ε_m

The *machine epsilon,* or *accuracy*, ε_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

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

In this case the problem occurs because:

$$10^{20} + 1 \quad \Rightarrow \quad 10^{20} \left(1 + \underbrace{\frac{1}{10^{20}}}_{\text{disappears as } \ll \varepsilon_m} \right) = 10^{20}$$

Machine epsilon, ε_m

The *machine epsilon,* or *accuracy*, ε_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

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

In this case the problem occurs because:

$$10^{20} + 1 \quad \Rightarrow \quad 10^{20} \left(1 + \underbrace{\frac{1}{10^{20}}}_{\text{disappears as } \ll \varepsilon_m} \right) = 10^{20}$$

Note: However, it is important to understand that ε_m is not the smallest floating-point number that can be represented on a machine.

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.

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.

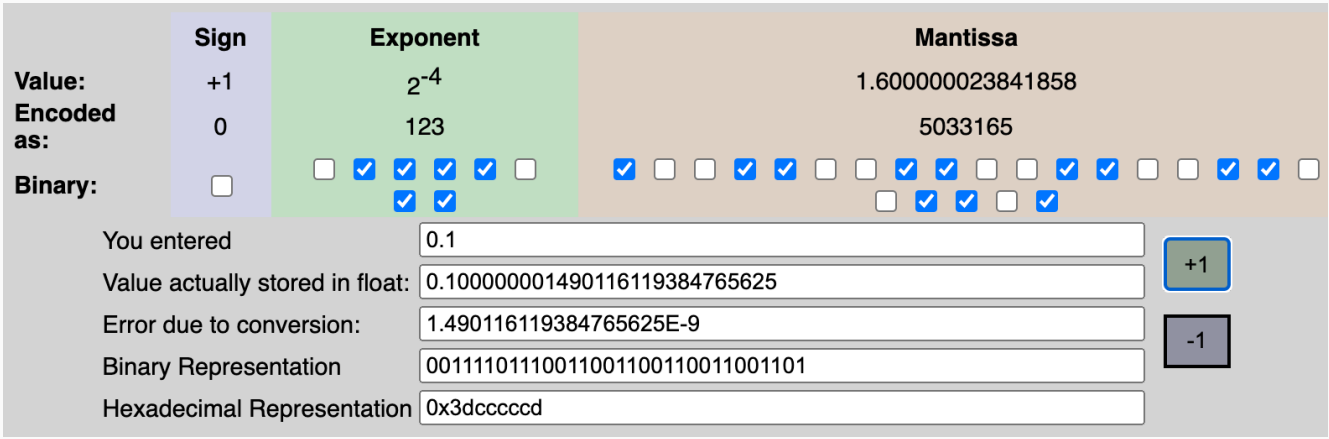
The **Underflow** is a condition in a computer program where the result of a calculation is a number of smaller absolute value than the computer can actually represent in memory.

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

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



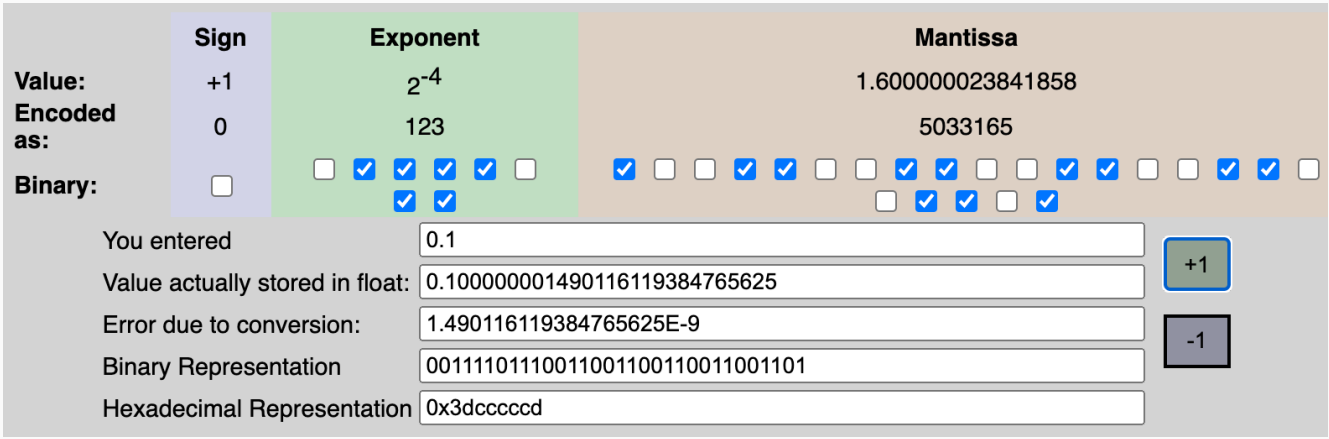
[link]

Round-off errors accumulate with increasing amounts of calculation.

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.

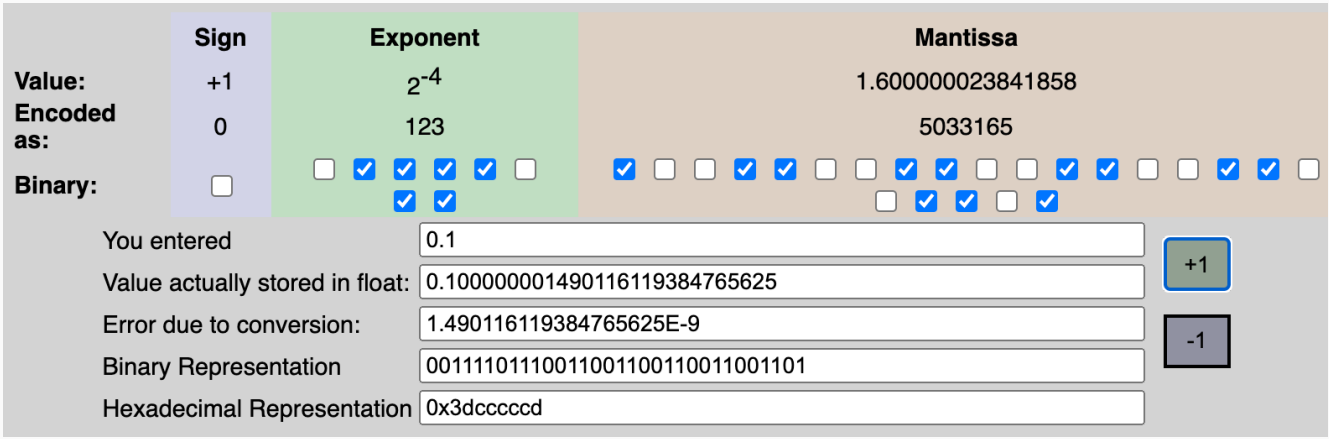
If, in the course of obtaining a calculated value, one performs N such arithmetic operations, one might end up having a total round-off error on the order of $\sqrt{N}\epsilon_m$ (when lucky)

(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



[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 != 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, ϵ_{rel}, and an absolute, ϵ_{abs}, errors. Then,

$$
a == b \; ? \quad \begin{cases} \text{if } a == b \\ \qquad \text{return } \textbf{true} \\ \\ \text{if } |a-b| < \max\left(\epsilon_{\text{abs}},\ \epsilon_{\text{rel}} \cdot (|a|+|b|)\right) \\ \qquad \text{return } \textbf{true} \\ \\ \text{otherwise} \\ \qquad \text{return } \textbf{false} \end{cases}
$$

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:

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

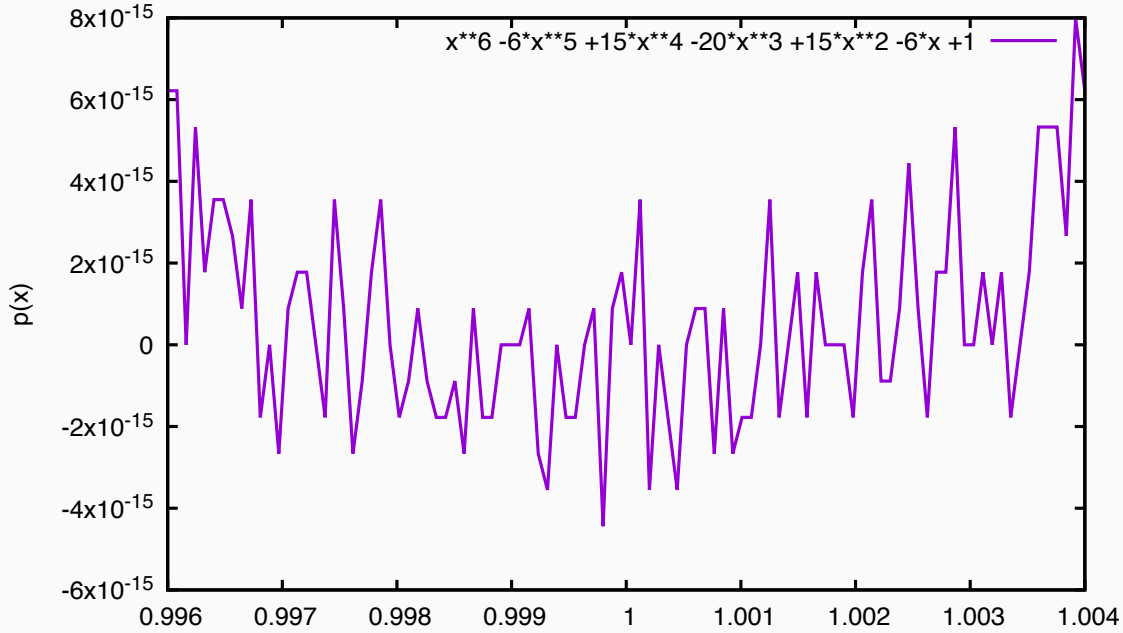
Cancellation error

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:

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



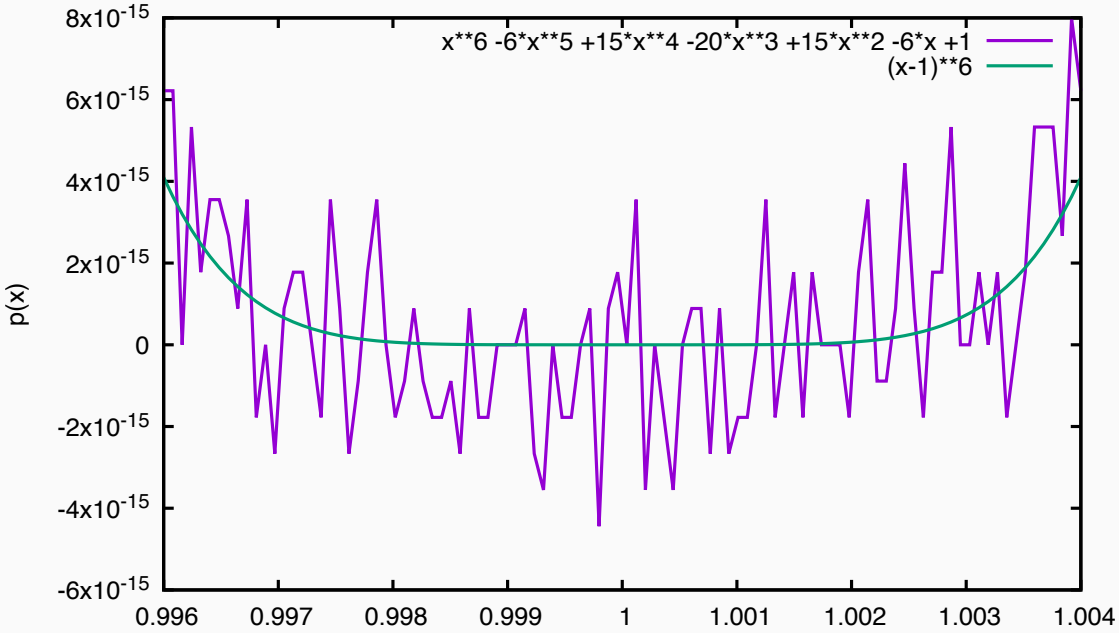
Cancellation error (a.k.a. "Catastrophic" cancellation error)

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:

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



Avoiding cancellation error: example 1

Few cases are recurrent and dangerous...

1. Case of the quadratic formula:

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

it the solutions are written like this, cancellation can occur:

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

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

Avoiding cancellation error: example 1

Few cases are recurrent and dangerous...

1. Case of the quadratic formula:

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

it the solutions are written like this, 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 in a way that avoids cancellation:

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

Avoiding cancellation error: example 1

1. Case of the **quadratic** formula:

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

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

Avoiding cancellation error: example 1

1. Case of the **quadratic** formula:

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

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

Choosing the right formulae:

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

Avoiding cancellation error: example 1

1. Case of the **quadratic** formula:

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

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

Choosing the right formulae:

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

Imagine we used the other formulae:

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

Avoiding cancellation error: examples 2 and 3

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 $y^2/x^2 < \varepsilon_m$). This expression is more accurately evaluated as

$$(x + y)\,(x - y)$$

Avoiding cancellation error: examples 2 and 3

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 $y^2/x^2 < \varepsilon_m$). 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 abs(magnitude) and sum from the smallest to the largest
2. "Kahan summation" algorithm

Evaluation of functions

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 $x = 10^{-8}$.

This function should return a value very close to $\frac{1}{2}$. But, due to cancellation error, the result is different (wrong). We can compute this value using a program as follows:

```
>>> x = 1e-8
>>> numerator = 1 - cos(x)
>>> denominator = x**2
>>> result = numerator/denominator
>>> print('result = ', result)
result = 0
```

What does this happen: because $\cos x \simeq 1 - \frac{x^2}{2} + \frac{x^4}{24} + \ldots$

Special built-in functions

Even built-in functions can incur in cancellation error. E.g., the logarithm, which the CPU computes using a Taylor expansion:

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

which makes the function incur in **cancellation** whenever $x = 1 + \varepsilon$, with $\varepsilon < \varepsilon_m$.

Special built-in functions

Even built-in functions can incur in cancellation error. E.g., the logarithm, which the CPU computes using a Taylor expansion:

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

which makes the function incur in **cancellation** whenever $x = 1 + \varepsilon$, with $\varepsilon < \varepsilon_m$.

To help with these cases, the CPU provides **special** built-in mathematical functions:

log1p(x)

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

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

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

Special built-in functions

Even built-in functions can incur in cancellation error. E.g., the logarithm, which the CPU computes using a Taylor expansion:

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

which makes the function incur in **cancellation** whenever $x = 1 + \varepsilon$, with $\varepsilon < \varepsilon_m$.

To help with these cases, the CPU provides **special** built-in mathematical functions:

log1p(x)

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

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

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.

Other 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 $|b| \ll |a|$.

Other 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 $|b| \ll |a|$.

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

Other 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 $|b| \ll |a|$.

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

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

When $y$is an integer number, this function is computationally expensive and inaccurate.

Advice: use `x**y` or `x^y` whenever y is an integer number.

Summary of special built-in mathematical functions

log1p(x) $= \log(1+x)$

expm1(x) $= e^x - 1$

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

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

cbrt(x) $= \sqrt[1/3]{x}$, cube root of x.

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

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.

Robust implementation of functions

Let's take the example we have already seen:

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

Numerical **instabilities** appear when x is small due to **cancellation** error: we get 0 instead of $\frac{1}{2}$. Even worst, for $x = 0$, the result is NaN.

Robust implementation of functions

Let's take the example we have already seen:

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

Numerical **instabilities** appear when x is small due to **cancellation** error: we get 0 instead of $\frac{1}{2}$. Even worst, for $x = 0$, the result is NaN.

A **robust implementation** comes from a careful consideration of this function.

Let's take the Taylor expansion of $f(x)$ to second order:

$$\frac{1 - \cos x}{x^2} \approx \frac{1}{2} - \frac{x^2}{24} + \dots$$

Robust implementation of functions

Let's take the example we have already seen:

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

Numerical **instabilities** appear when x is small due to **cancellation** error: we get 0 instead of $\frac{1}{2}$. Even worst, for $x = 0$, the result is NaN.

A **robust implementation** comes from a careful consideration of this function.

Let's take the Taylor expansion of $f(x)$ to second order:

$$\frac{1 - \cos x}{x^2} \approx \frac{1}{2} - \frac{x^2}{24} + \dots$$

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 $\rightarrow$ no problem occurs. The final result will differ from $\frac{1}{2}$ if and only if

$$\frac{1}{2} - \frac{x^2}{24} \quad \Rightarrow \quad 2\left(\frac{1}{2} - \frac{x^2}{24}\right) \quad \Rightarrow \quad 1 - \frac{x^2}{12} \quad \Rightarrow \quad \left|-\frac{x^2}{12}\right| \geq \varepsilon_m,$$

In other words, we should compute the function only when this condition is fulfilled.

[CONT. ON THE NEXT SLIDE]

Robust implementation of functions

A robust implementation of

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

is

$$f(x) = \begin{cases} \dfrac{1}{2} & \text{when } |x| < 2\sqrt{3\,\varepsilon_m}. \\ \dfrac{1 - \cos x}{x^2} & \text{otherwise} \end{cases}$$

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,

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

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,

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

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

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

(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):

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

- First derivative to fourth order (centred difference):

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

- Second derivative to second order (centred difference):

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

- Second derivative to fourth order (centred difference):

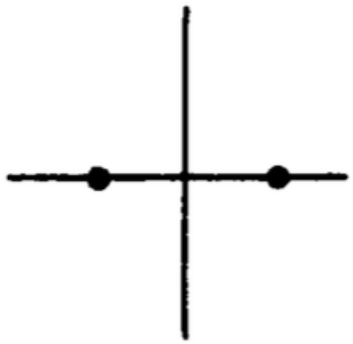
$$f''(x) = \frac{-f_{x+2h} + 16f_{x+h} - 30f_x + 16f_{x-h} - f_{x-2h}}{12h^2} + O\left(h^4\right)$$

Finite difference formulæ

Abramowitz and Stegun, page 883 and following

Partial Derivatives

25.3.21

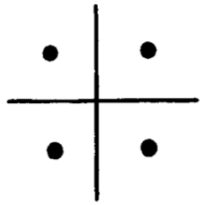


$$\frac{\partial f_{0,0}}{\partial x} = \frac{1}{2h}\,(f_{1,0} - f_{-1,0}) + O(h^2)$$

Finite difference formulæ

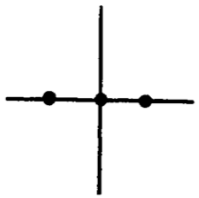
Abramowitz and Stegun, page 883 and following

25.3.22



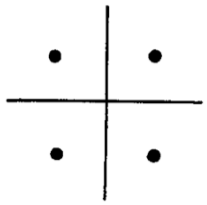
$$\frac{\partial f_{0,0}}{\partial x}=\frac{1}{4h}\,(f_{1,1}-f_{-1,1}+f_{1,-1}-f_{-1,-1})+O(h^2)$$

25.3.23



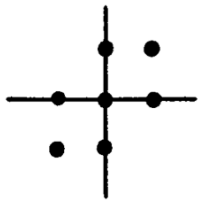
$$\frac{\partial^2 f_{0,0}}{\partial x^2}=\frac{1}{h^2}\,(f_{1,0}-2f_{0,0}+f_{-1,0})+O(h^2)$$

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



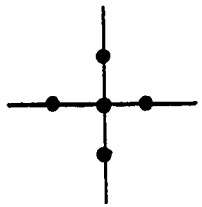
$$\frac{\partial^2 f_{0,0}}{\partial x\partial y}=\frac{-1}{2h^2}\,(f_{1,0}+f_{-1,0}+f_{0,1}+f_{0,-1}-2f_{0,0}-f_{1,1}-f_{-1,-1})+O(h^2)$$

Finite difference formulæ

Abramowitz and Stegun, page 883 and following

Laplacian

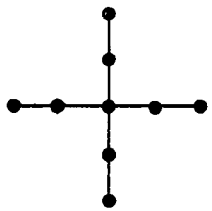
25.3.30



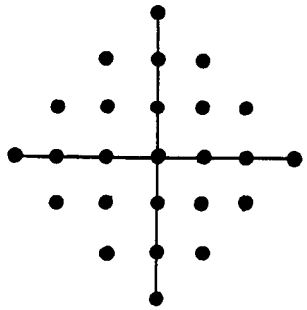
$$\nabla^2 u_{0,0} = \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right)_{0,0}$$

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

25.3.31



25.3.33



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

25.4. Integration

Trapezoidal Rule

25.4.1

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

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.

Psuedo-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 $X < 100000000$, 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. ∎

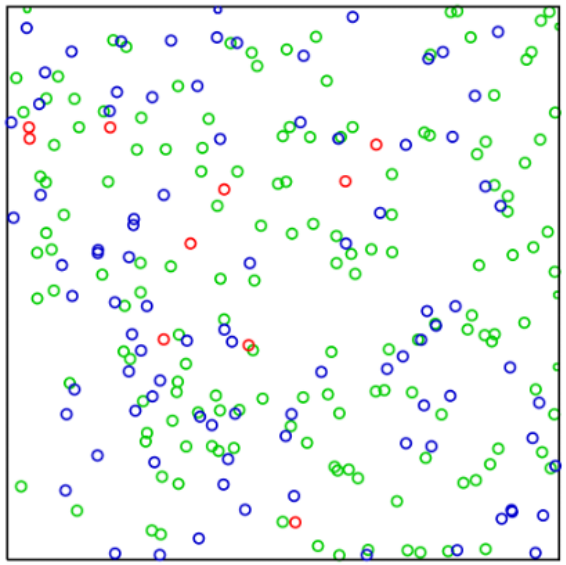
Quasi-random numbers

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.

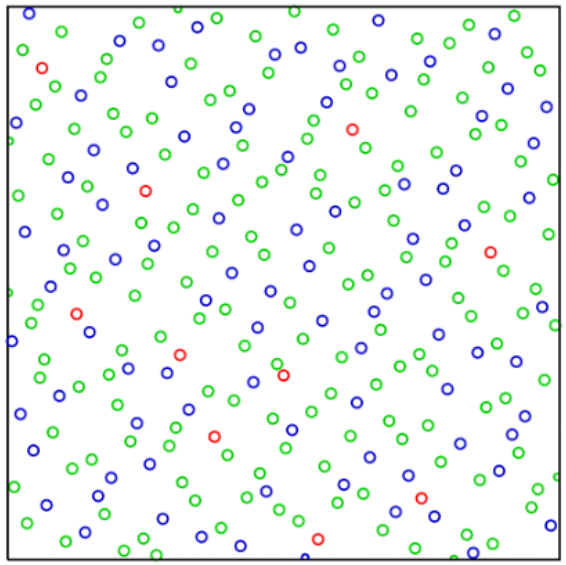
Algorithm

```
algorithm Halton-Sequence is
    inputs: index i
            base b
    output: result r

    f ← 1
    r ← 0

    while i > 0 do
        f ← f/b
        r ← r + f * (i mod b)
        i ← ⌊i/b⌋

    return r
```

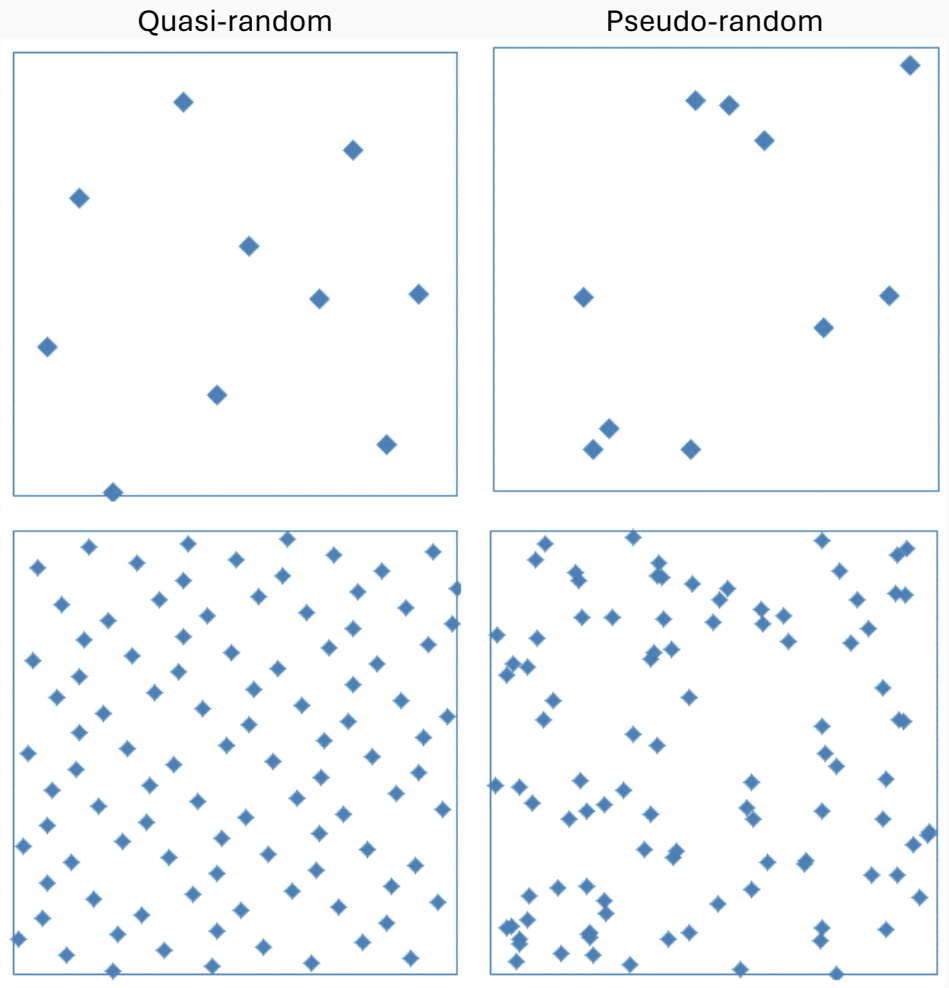
Pseudorandom



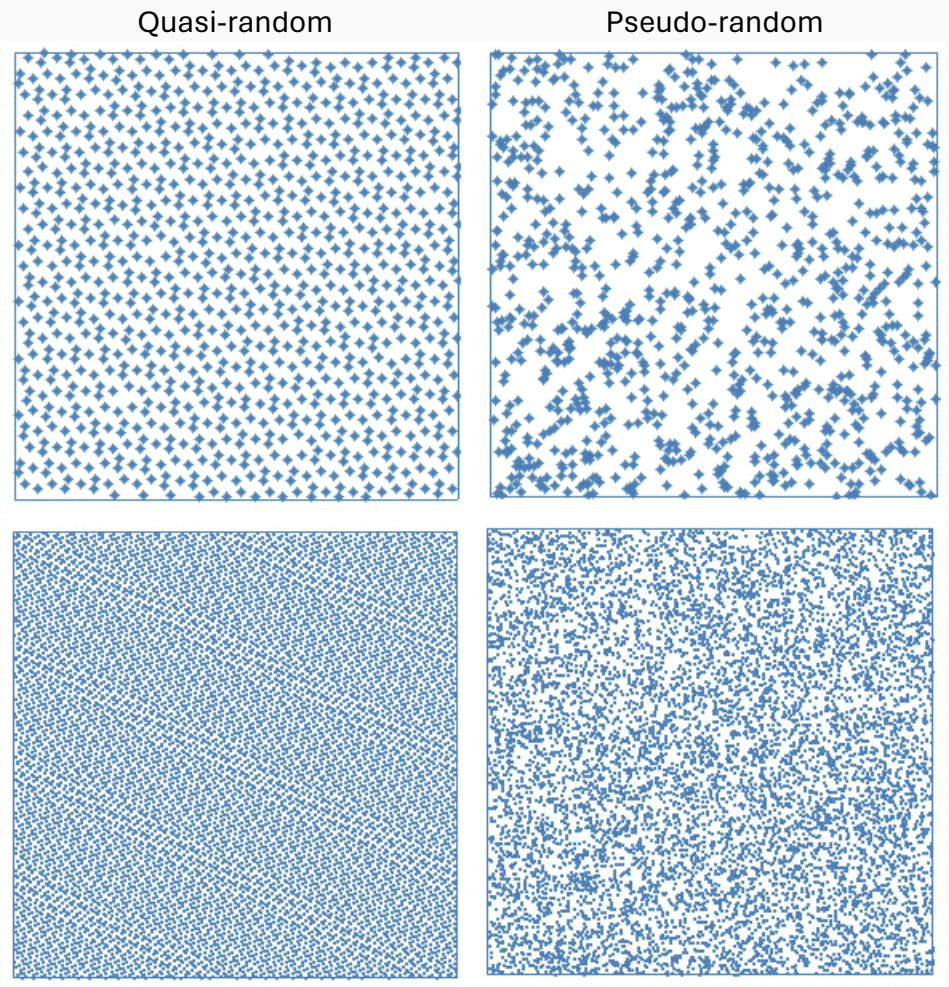
Halton Sequence



Quasi-random vs Pseudo-random numbers



Quasi-random vs Pseudo-random numbers



Exact and arbitrary-precision numbers

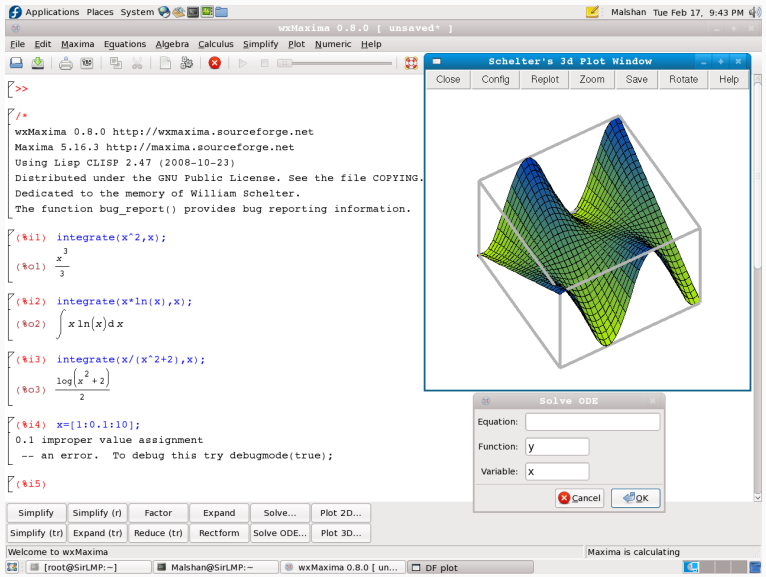
In cases where double-, extended- or even quadruple-precision are not enough, there exist a couple of solutions to achieve higher precision and in some cases even exact results.

- **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.

GMP
«Arithmetic without limitations»

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

Programs such as Maxima, Mathematica©, or Maple©, 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



Part 2.

Tools for Scientific and Accelerator Physics Computing

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.

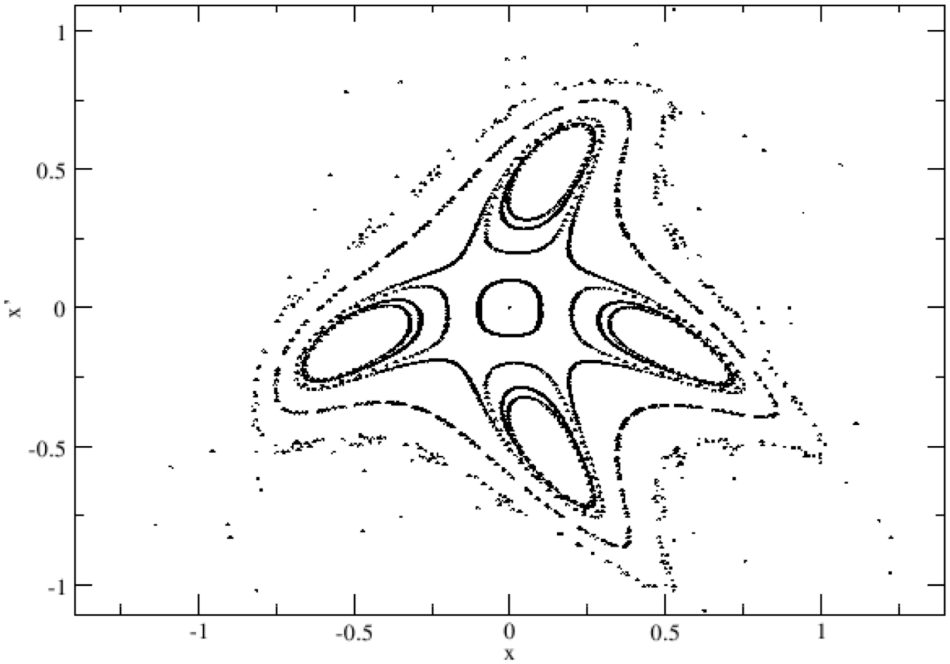
Python is a more general-purpose programming language with a wide range of libraries and frameworks that can be used for scientific computing. Some of the most popular libraries for scientific computing in **Python** include NumPy, SciPy, Pandas, and Matplotlib. **Python** also has a large and active 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.»

Tools: Octave

Example: impact of nonlinear elements on linear optics



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

- Q=0.2516
- linear motion near center (circles)
- More and more square
- Non-linear tuneshift
- 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

Tools: Octave simulation

SextupolarKick.m

New Open Recent Revert Save Print Undo Redo Cut Copy Paste Search Preferences

```
% 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 + [ O; 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)

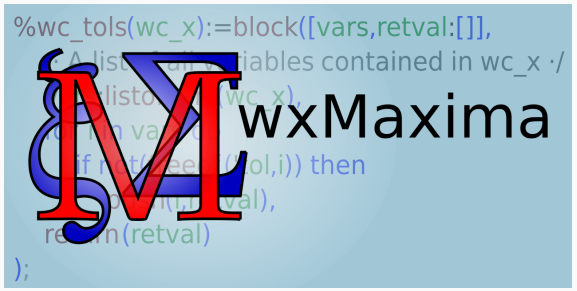
Tools: Symbolic computation

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.

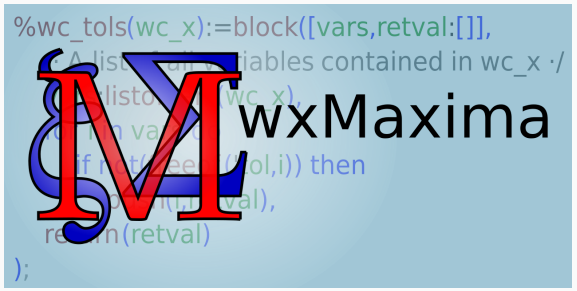
Tools: Symbolic computation

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.



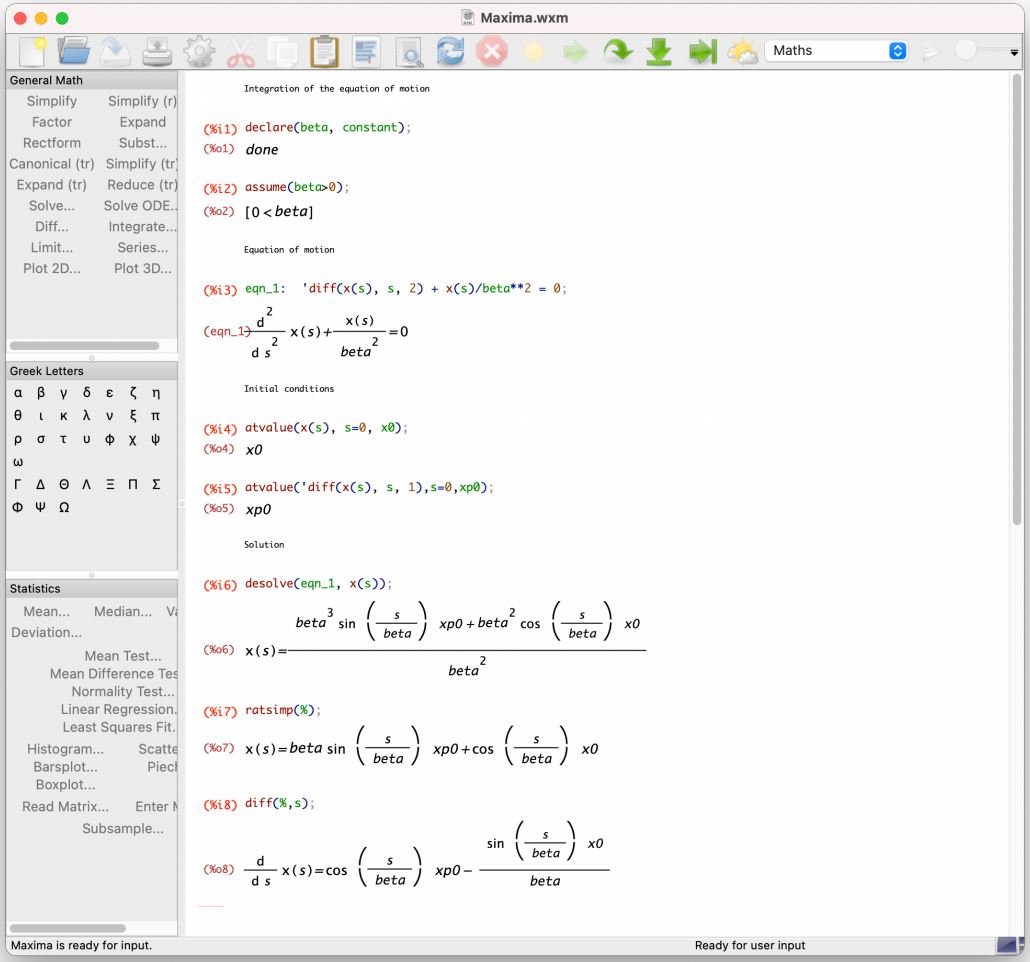
wxMaxima is an excellent front end for Maxima.

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.

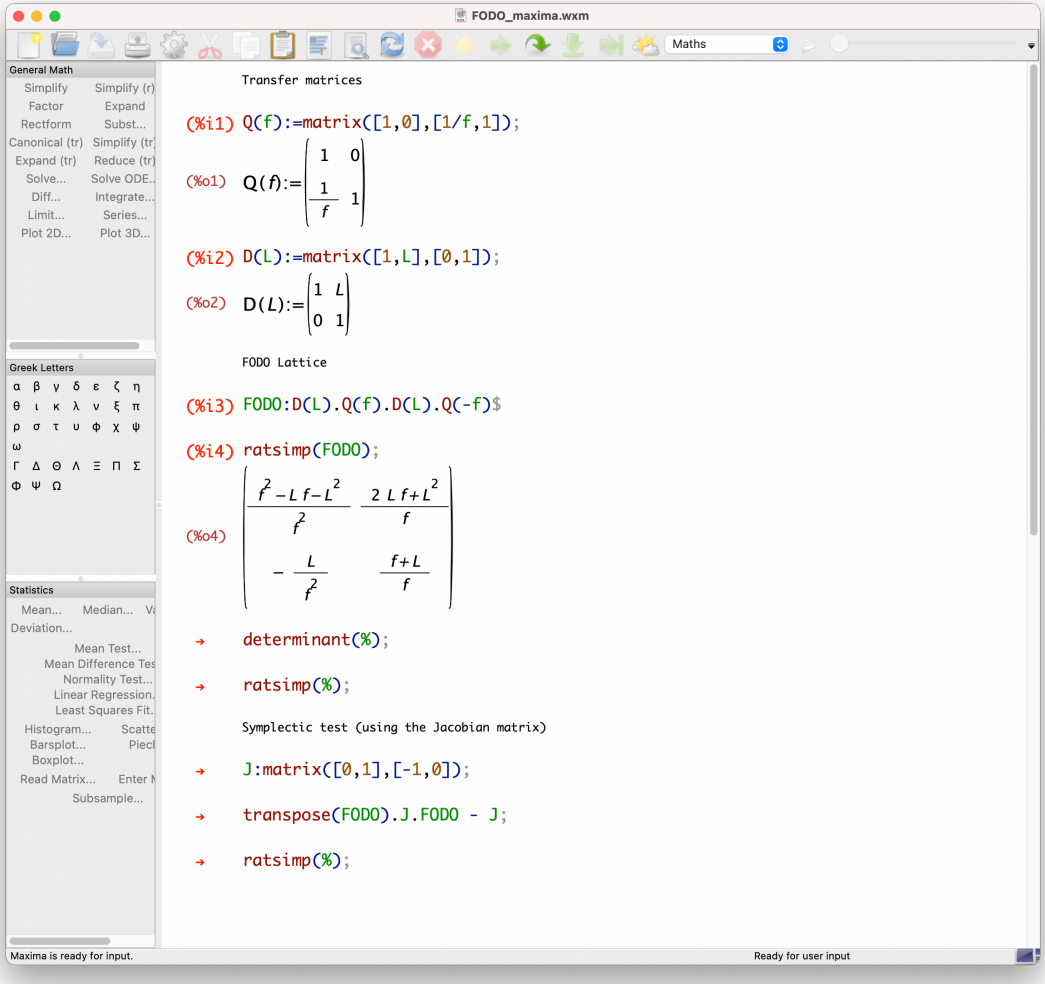
Tools: Symbolic computation

A 1D harmonic oscillator with wxMaxima



Tools: Symbolic computation

FODO cell in wxMaima



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:

```
$ units -v
You have: 300 pC * 15 GV * 50 Hz
You want: W
    300 pC * 15 GV * 50 Hz = 225 W
    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:

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

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

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

It likely crashed. There was a units mistake: the ground-based control system used «pounds×seconds», Imperial units, whereas the spacecraft used SI units, $N \times s$.

Shell scientific tools: arbitrary precision

bc

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

```
$ bc
bc 1.06
Copyright 1991-1994, 1997, 1998, 2000 Free Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type ‘warranty’.
scale=1
sqrt(2)
1.4
scale=40
sqrt(2)
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

```
$ bc
bc 1.06
Copyright 1991-1994, 1997, 1998, 2000 Free Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type `warranty'.
scale=1
sqrt(2)
1.4
scale=40
sqrt(2)
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.

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".

```c
#include <mpi.h>
#include <stdio.h>

int main()
{
    // Initialize the MPI environment
    MPI_Init(NULL, NULL);

    // Get the number of processes
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    // Get the rank of the process
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

    // Get the name of the processor
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    int name_len;
    MPI_Get_processor_name(processor_name, &name_len);

    // Print off a hello world message
    printf("Hello world from processor %s, rank %d out of %d processors\n",
           processor_name, world_rank, world_size);

    // Finalize the MPI environment.
    MPI_Finalize();
}
```

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.

```cpp
int main()
{
    int a[100000];

    #pragma omp parallel for
    for (int i = 0; i < 100000; i++) {
        a[i] = 2 * i;
    }

    return 0;
}
```

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>`.

```cpp
// thread example
#include <iostream>       // std::cout
#include <thread>         // std::thread

void foo()
{
  // do stuff...
}

void bar(int x)
{
  // do stuff...
}

int main()
{
  std::thread first (foo);     // spawn new thread that calls foo()
  std::thread second (bar,0);  // spawn new thread that calls bar(0)

  std::cout << "main, foo and bar now execute concurrently...\n";

  // synchronize threads:
  first.join();                // pauses until first finishes
  second.join();               // pauses until second finishes

  std::cout << "foo and bar completed.\n";

  return 0;
}
```

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.

Accelerator physics codes

There are many.

Specialised in storage rings, linacs, injectors, ...

- Optics codes: e.g., MAD8, MAD-X, MAD-NG
  - Focus on optics design
- Tracking codes: e.g., X-suite, RF-Track, ELEGANT
  - Particle tracking
  - Dynamic aperture (rings)
  - Assessing beam performance under various imperfections
- Codes focused on collective effects: e.g., PyHEADTAIL, GUINEA-PIG
  - Impedances
  - Specific effects: e.g. beam beam, space charge, ...

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]

Accelerator physics codes: storage rings

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

BLonD
Beam **Lon**gitudinal **D**ynamics is a CERN simulation suite for longitudinal particle tracking in synchrotrons. Single and multi-bunch Acceleration, multiple RF systems, multiple RF stations Collective effects in frequency and time domain Low-power level RF options (phase noise, phase loop, feedbacks...) Monitoring, plotting, data analysis Documentation [tracking, collective effects, imperfections]

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]

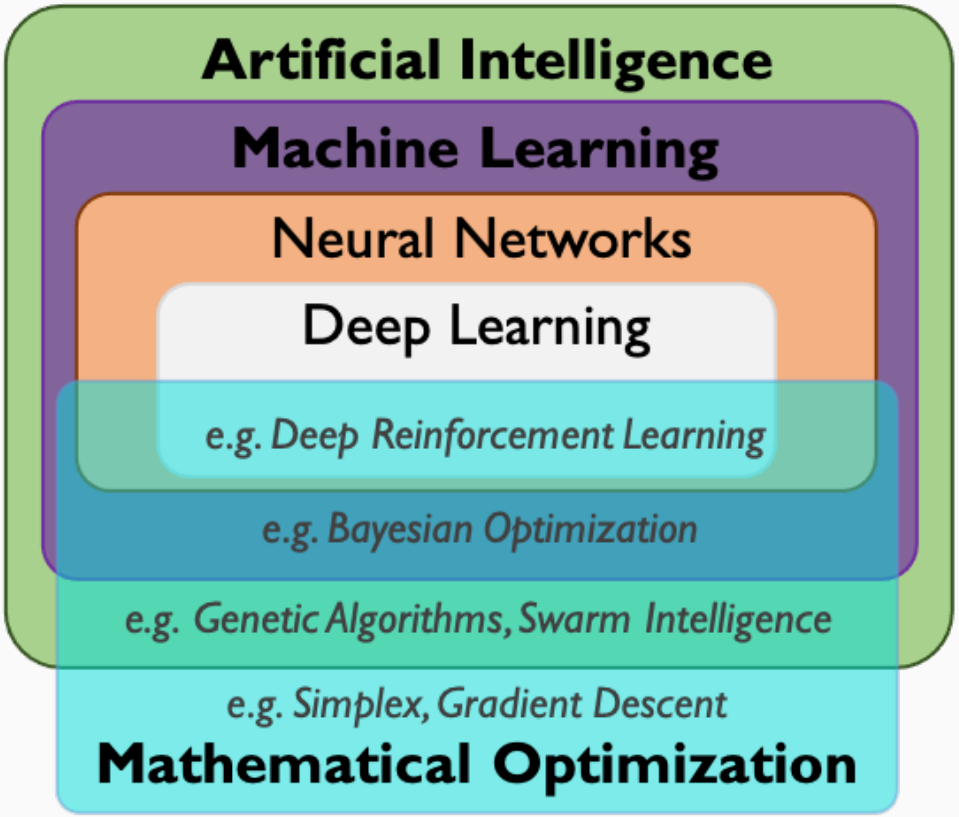
And others, but some aren't maintained or they are not open-source and free...

Part 3.

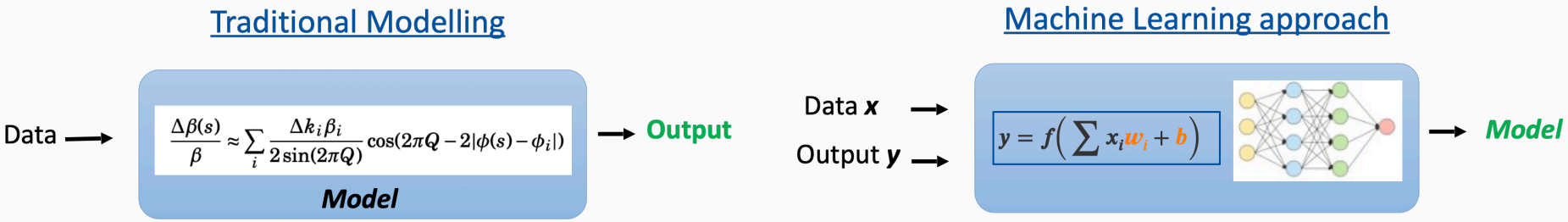
Machine Learning for

Accelerator Physics Applications

Taxonomy of AI and optimisation-related topics



Machine learning vs traditional methods



- Creating **manually** a set of commands / equations and rules
- Example: comparing simulations and measurements

- **Learn from data automatically**
- Model is developed by adjusting model's parameters **to explain the relation between given data and output**

Machine learning concepts

Supervised Learning

- **Input/output pairs** available
- Learn a mapping function, **generalizing for all provided data**
- Predict from **unseen data**

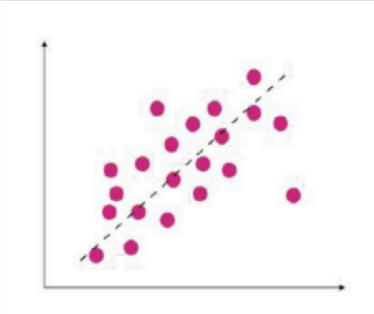
Unsupervised Learning

- **Only input** data is given
- Discover **structures and patterns**

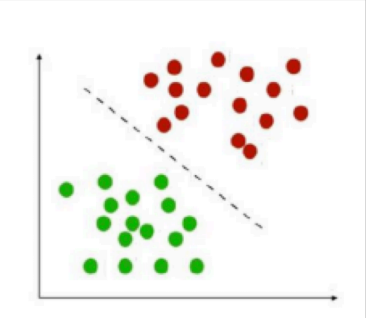
Reinforcement Learning

- No labeled dataset for training
- Interact with an environment
- Trying to learn **optimal sequences of decisions**

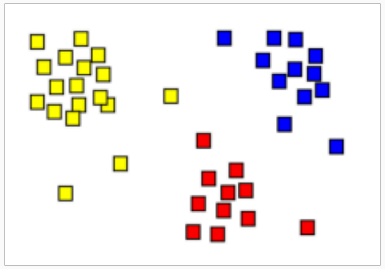
Regression

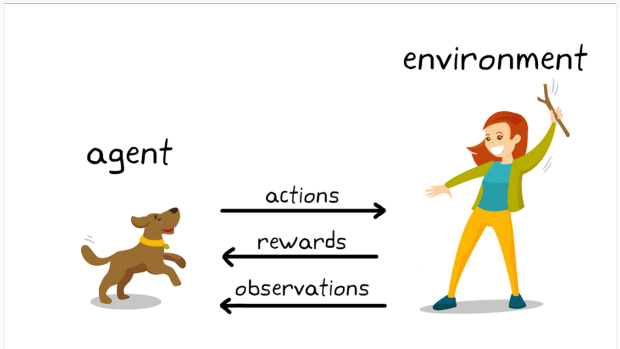


Classification

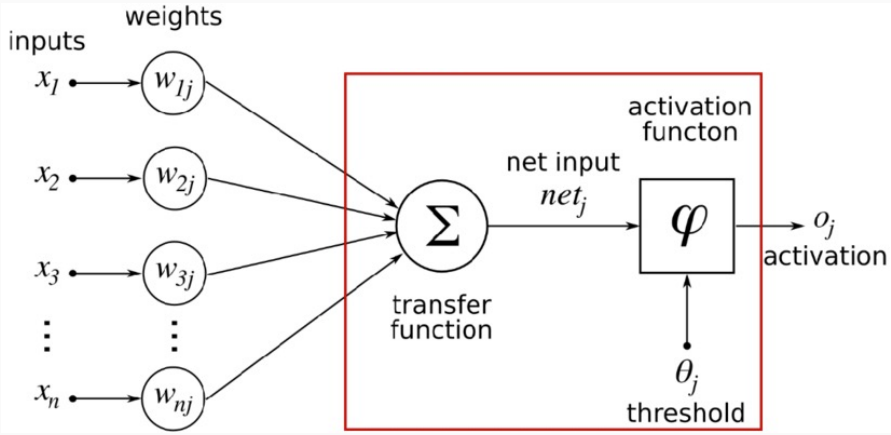


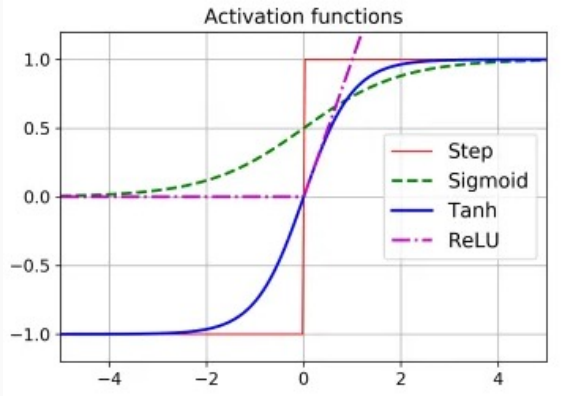
Clustering





The simplest neural netowrk: the perceptron



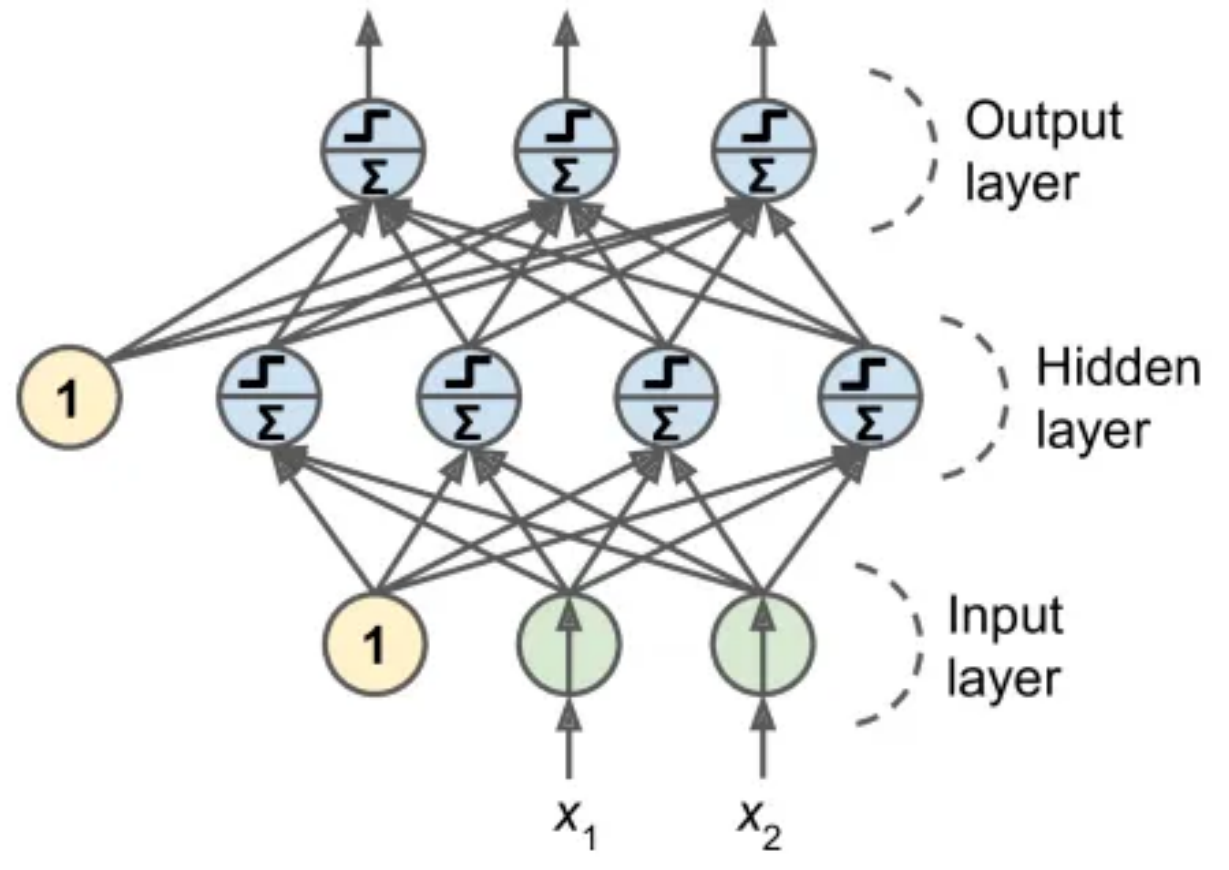


Cost function (or error function):

$$\chi^2 = \frac{1}{2} \sum_{\text{samples}} \sum_{\text{neurons}} \left(o_{j,\text{expected}} - o_{j,\text{computed}} \right)^2$$

Example of learning algorithm: backpropagation

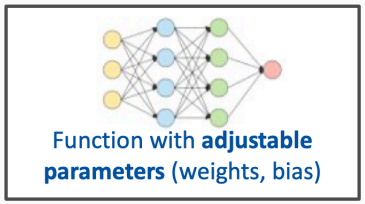
Deep learning: the multi-layer perceptron



Supervised learning: building models from data

example 1
example 2
example 3
.
.
.

Training input data



Function with **adjustable parameters** (weights, bias)

Model output

$$y = f\left(\sum x_i wi + b\right)$$

Compute the **loss** (approximation error): e.g. Mean Squared Error

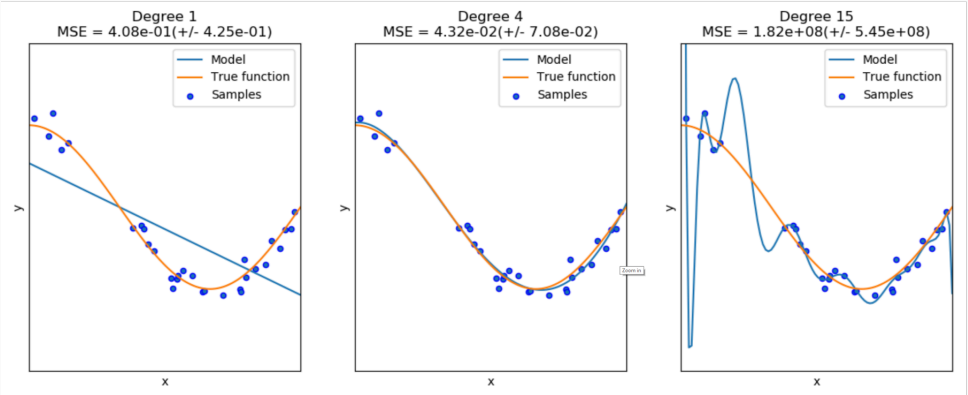
Training output data

Adjust parameters

Minimizing the loss

e.g. Gradient Descent

Generalized model **explaining relationship between input and output** variables in **all training samples**.



Generalisation vs. Overfitting

Simple models underfit

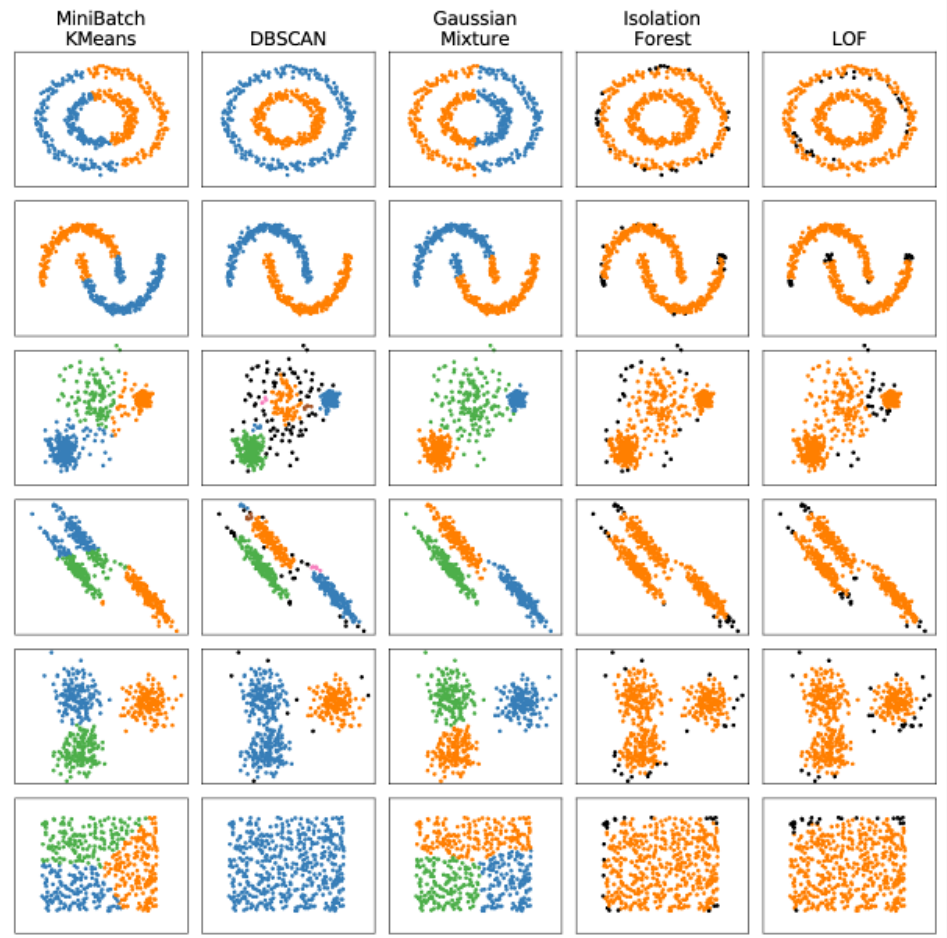
- Derivate from data (high bias)
- Do not correspond to data structure (low variance)

Complex models overfit

- Very low systematical deviation (low bias)
- Very sensitive to data (high variance)

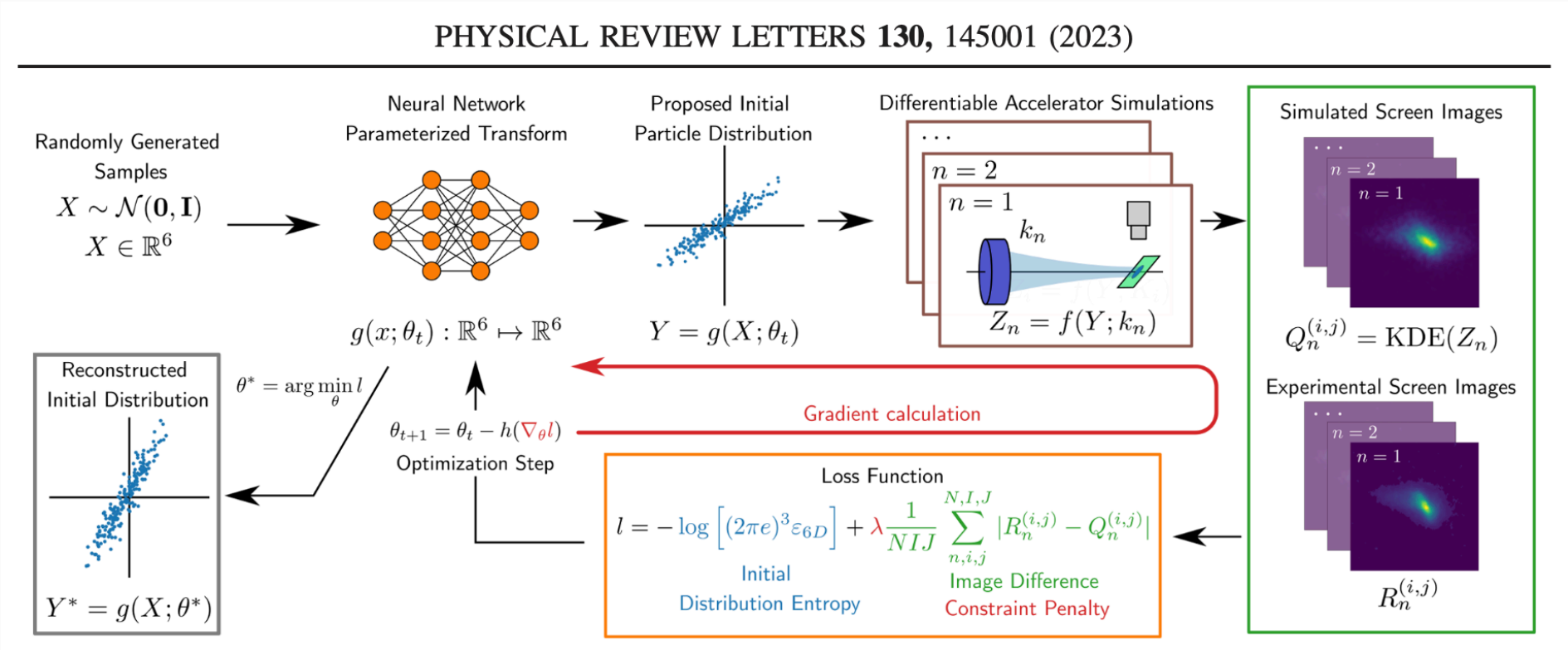
➡ Bias-Variance tradeoff

Unsupervised learning: clustering and anomaly detection



] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12:2825–2830.]

Supervised learning: 6D phase space reconstruction



R. Roussel, A. Edelen, C. Mayes, D. Ratner, J. P. Gonzalez-Aguilera, S. Kim, E. Wisniewski, and J. Power, "Phase Space Reconstruction from Accelerator Beam Measurements Using Neural Networks and Differentiable Simulations", Phys. Rev. Lett. 130, 145001

Machine learning in accelerator modelling: examples

- Speeding-up computationally costly **simulations**:

  Methods: **Clustering techniques**, **Gaussian Processes**, Supervised Learning (inverse) models

  Applications: Sample-efficient dynamic aperture estimation [1], electron beam size optimisation[2]

- **Operation** automation and online tuning:

  Methods: Bayesian optimization (using **Gaussian Processes**), Reinforcement Learning, **physics-informed NN** for modelling, **Clustering techniques**

  Applications: Tuning optics models in storage rings [3], beam trajectory steering [4], faulty BPMs detection [5]

- Virtual **Diagnostics**:

  Methods: Image-based analysis using **Convolutional NN** trained on simulations

  Applications: 6D phase space reconstruction [6]

Machine Learning:
✓ ***Learn arbitrary models***
✓ ***Directly from provided data***

[1] F.F. Van der Veken et al., "Using Machine Learning to Improve Dynamic Aperture Estimates", IPAC'21
[2] A. Edelen et al., "Machine learning for orders of magnitude speedup in multiobjective optimization of particle accelerator systems", Phys. Rev. Accel. Beams **23**, 044601 (2020)
[3] A. Ivanov, I. Agapov, "Physics-Based Deep Neural Networks for Beam Dynamics in Charged Particle Accelerators", Phys. Rev. Accel. Beams **23**, 074601 (2020)
[4] V. Kain et al., "Sample-efficient reinforcement learning for CERN accelerator control", Phys. Rev. Accel. Beams, 23.124801 (2020)
[5] E. Fol et al., "Detection of faulty beam position monitors using unsupervised learning", Phys. Rev. Accel. Beams **23**, 102805 (2020)
[6] R. Roussel et al., "Phase Space Reconstruction from Accelerator Beam Measurements Using Neural Networks and Differentiable Simulations", Phys. Rev. Lett. **130**, 145001 (2023)

The end.

Thank you for your attention!

Any questions?

Many thanks to: Laurent Deniau (CERN / BE-ABP-LNO), for his suggestions and ideas; and Elena Fol (CERN / BE-ABP-LAF), for inputs and references on machine learning.

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 $x_i = a + i\frac{b-a}{n} = a + i\,h$, and $f_i = f(x_i)$: then the integral can be approximated with a sum

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

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 $[-1, 1]$
- $w_i = \dfrac{2}{(1-x_i)^2 \left[P_n'(x_i)\right]^2}$
- $P_n(x)$ 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	Legendre polynomials	25.4.29
$(-1, 1)$	$(1-x)^\alpha (1+x)^\beta, \quad \alpha, \beta > -1$	Jacobi polynomials	25.4.33 ($\beta = 0$)
$(-1, 1)$	$\frac{1}{\sqrt{1-x^2}}$	Chebyshev polynomials (first kind)	25.4.38
$[-1, 1]$	$\sqrt{1-x^2}$	Chebyshev polynomials (second kind)	25.4.40
$[0, \infty)$	e^{-x}	Laguerre polynomials	25.4.45
$[0, \infty)$	$x^\alpha e^{-x}, \quad \alpha > -1$	Generalized Laguerre polynomials	
$(-\infty, \infty)$	e^{-x^2}	Hermite polynomials	25.4.46

For more details see A & S.

Integration in many dimensions

Monte Carlo method

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

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

where dV is the multi-dimensional volume element.

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

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

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

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
```

Useful linux tools

Use of named pipes for interprocess communication (FIFOs)

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

```
$ mkfifo mypipe
$ ls -l mypipe
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.

```
$ echo "Can you read this?" > mypipe
$ ls -l mypipe
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.

```
$ cat mypipe
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:

```
$ units -v
You have: c * e * 8.5 T
You want: N
  c * e * 8.5 T = 4.082724005684724e-10 N
  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	mrad	time	mm/c	force	MeV/m

In fact,

```
c * e * 8.5 T = 2548.235893 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.