Computational Tools

Andrea Latina
andrea.latina@cern.ch

CAS, Introduction to Accelerator Physics, 25 September 2021 - 08 October 2021

Table of contents

Purpose of this course

In these two lessons, we will outline the 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 the 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 the 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.

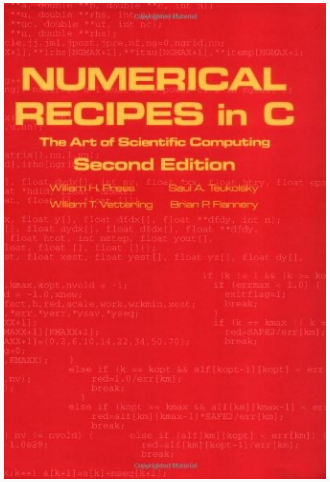
We will suggest reference readings and clarify important aspects of numerical stability to help avoid making bad but unfortunately common mistakes. Numerical stability should be basic knowledge of every scientist.

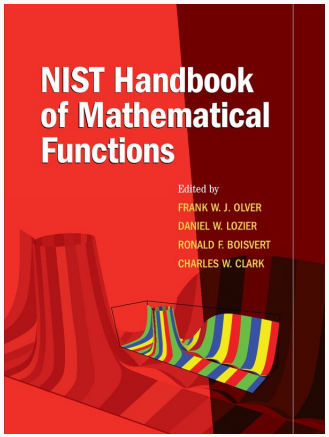
We will exclusively refer to free and open-source software running on Linux or other Unix-like operating systems. Also, we will unveil powerful shell commands that can speed up simulations, facilitate data processing, and in short, increase your scientific throughput.

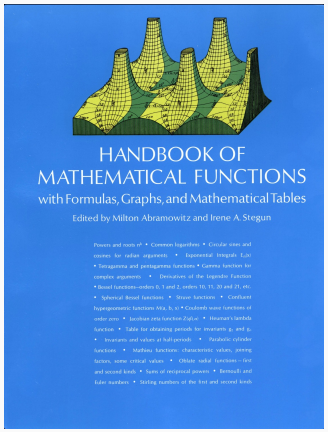
Some references

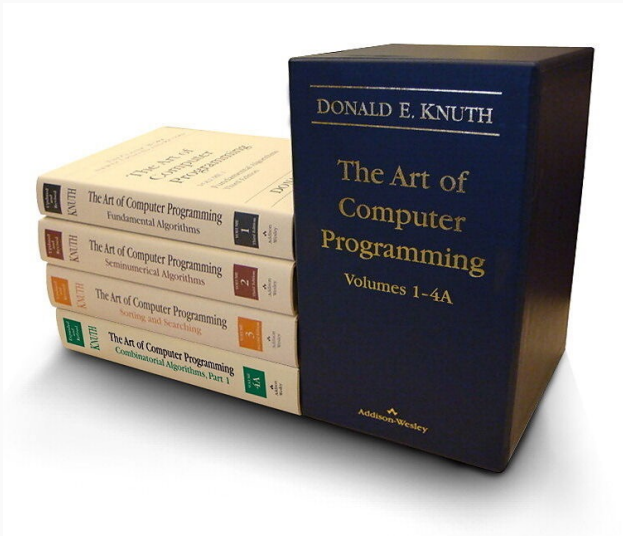
1. "*Numerical Recipes: The Art of Scientific Computing*", W. Press, S. Teukolsky, W. Vetterling, and B. Flannery, 1992 (2nd edition) – 2007 (3rd edition)
2. Donald Knuth, "*The Art of Computer programming*", 1968 – (the book is still incomplete)
3. Abramowitz and Stegun, "*Handbook of Mathematical Functions with Formulas*", 1964
4. Olver, F. , Lozier, D. , Boisvert, R. and Clark, C., "*The NIST Handbook of Mathematical Functions*", 2010
5. Zyla, P. A., et al., "*Review of Particle Physics*", Oxford University Press.

More in the proceedings...





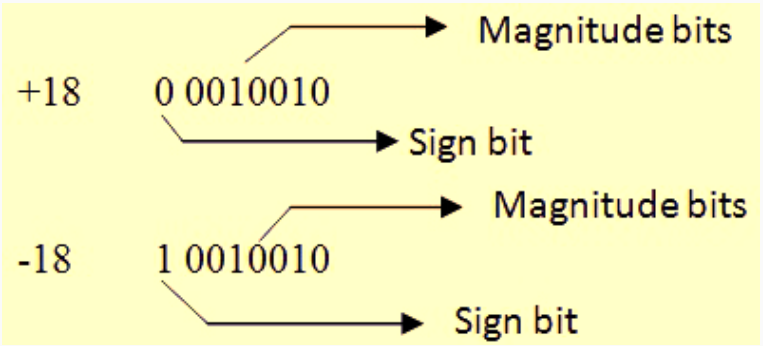




Internal representation of numbers

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

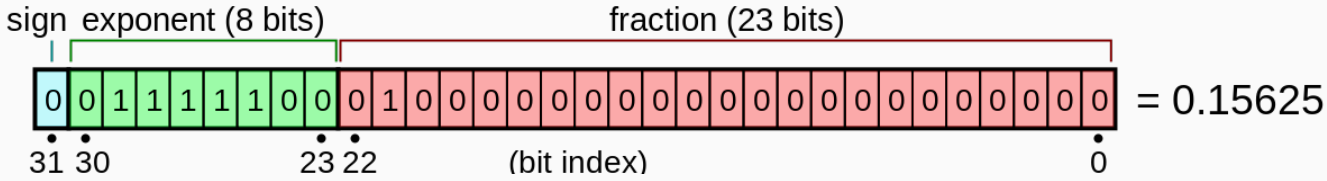
Real numbers

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

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

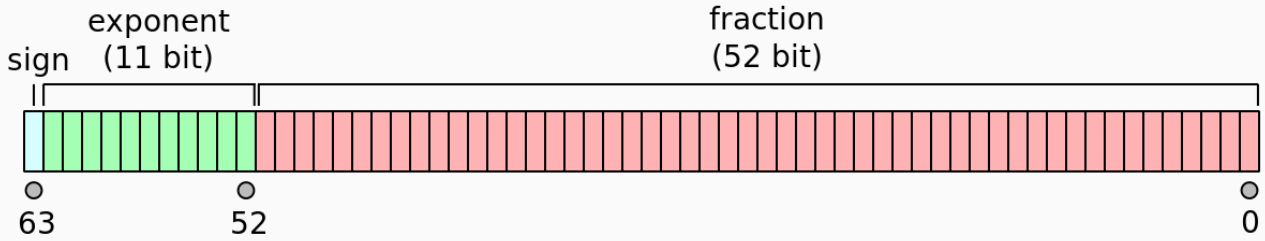
  - a sign bit (interpreted as plus or minus)
  - an exact integer exponent
  - en exact positive integer mantissa (or fraction)

- 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`.

Internal representation of numbers

Range of real numbers

The complete range of the positive normal floating-point numbers in *single-precision* is:

$$s_{\text{min}} = 2^{-126} \approx 1.17 \times 10^{-38},$$
$$s_{\text{max}} = 2^{127} \approx 3.4 \times 10^{38}.$$

In *double-precision* the range is:

$$d_{\text{min}} = 2^{-1022} \approx 2 \times 10^{-308},$$
$$d_{\text{max}} = 2^{1023} \approx 2 \times 10^{308}.$$

Single precision retains up to about 7 significant digits after the comma, double precision about 15.

Internal representation of numbers

Range of real numbers

The complete range of the positive normal floating-point numbers in *single-precision* is:

$$s_{\text{min}} = 2^{-126} \approx 1.17 \times 10^{-38},$$
$$s_{\text{max}} = 2^{127} \approx 3.4 \times 10^{38}.$$

In *double-precision* the range is:

$$d_{\text{min}} = 2^{-1022} \approx 2 \times 10^{-308},$$
$$d_{\text{max}} = 2^{1023} \approx 2 \times 10^{308}.$$

Single precision retains up to about 7 significant digits after the comma, double precision about 15.

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++ quadruple precision may be specified using the `long double` type, but this is not required by the language.

Internal representation of numbers

Special numbers

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

- **infinity** (positive and negative)
- the **negative zero**, -0.0. It compares equal to the positive zero, but is meaningful in some arithmetic operations, e.g. 1.0/0.0 == INFINITY, but 1.0/-0.0 == -INFINITY)
- **Not-a-number (NaN)**, which does not compare equal with anything (including itself)

Machine precision

- The *machine accuracy* ε_m is the smallest floating-point number which, added to 1.0, produces a floating-point result different from 1.0:

$$1.0 + \varepsilon_m \neq 1.0$$

  - For single precision

$$\varepsilon_m \approx 3 \cdot 10^{-8},$$

  - For double precision

$$\varepsilon_m \approx 2 \cdot 10^{-16}.$$

- It is important to understand that ε_m is not the smallest floating-point number that can be represented on a machine.

Machine precision

- The *machine accuracy* ε_m is the smallest floating-point number which, added to 1.0, produces a floating-point result different from 1.0:

$$1.0 + \varepsilon_m \neq 1.0$$

  - For single precision

$$\varepsilon_m \approx 3 \cdot 10^{-8},$$

  - For double precision

$$\varepsilon_m \approx 2 \cdot 10^{-16}.$$

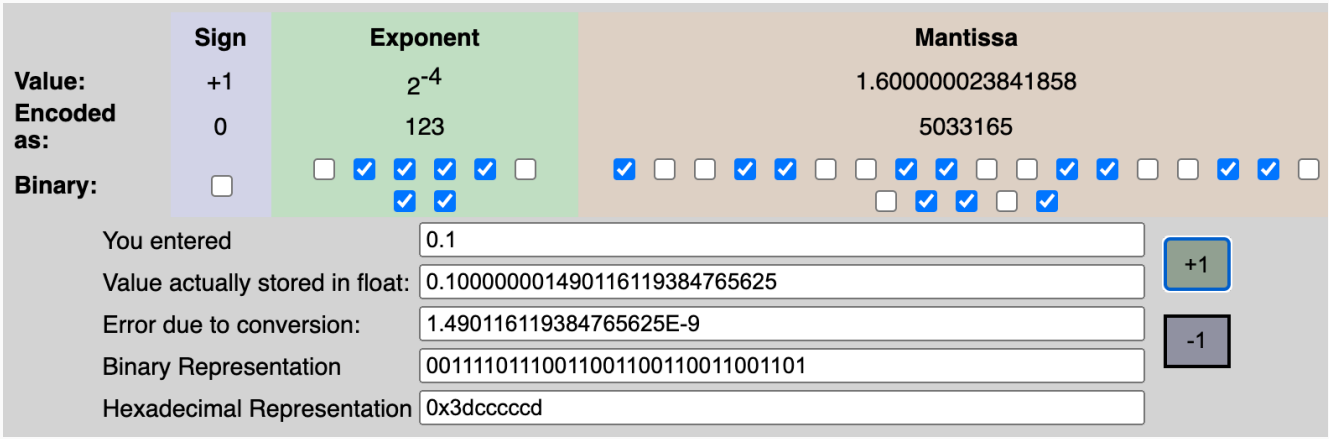
- It is important to understand that ε_m is not the smallest floating-point number that can be represented on a machine.

Note: 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



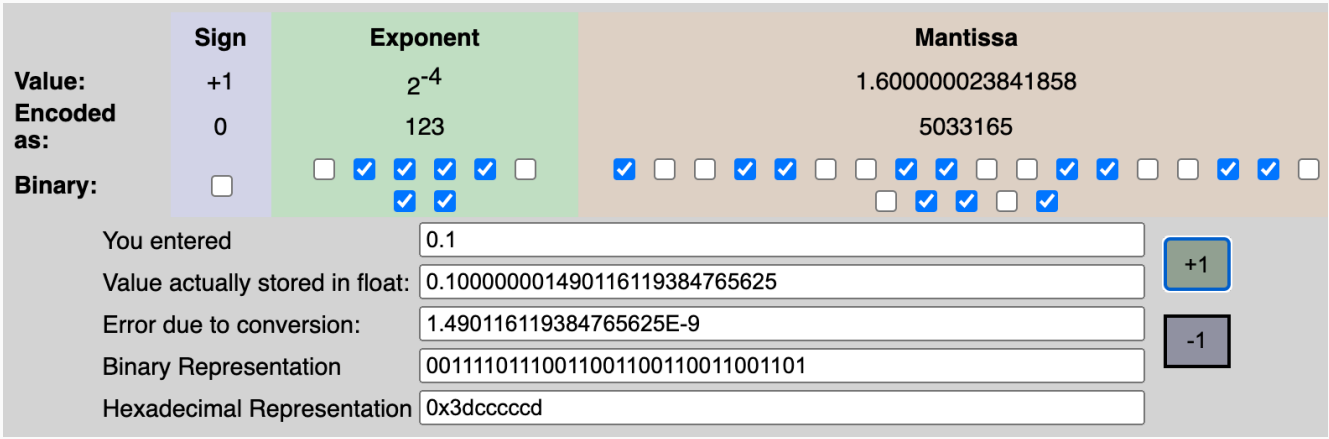
[link]

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.

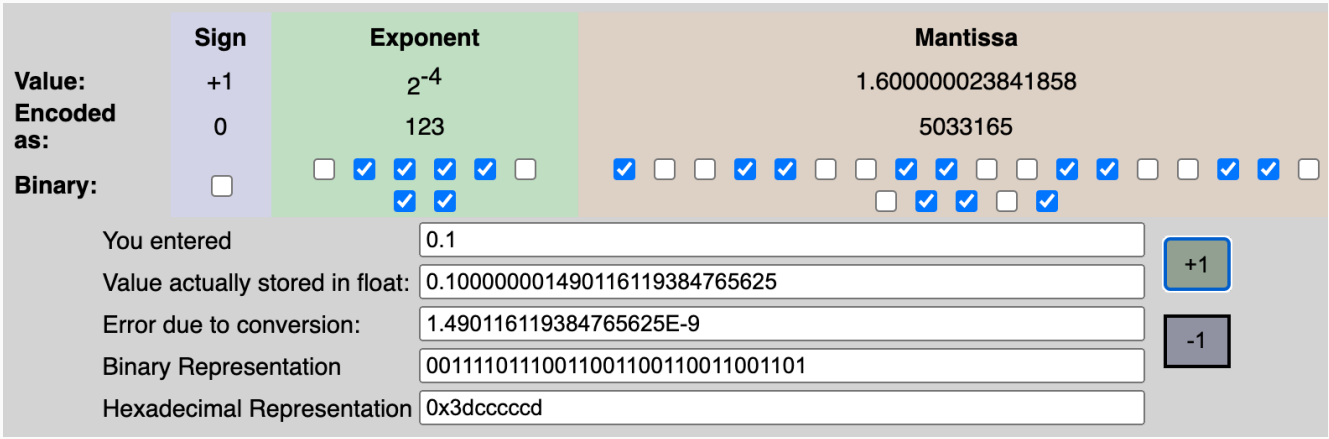
If, in the course of obtaining a calculated value, one performs N such arithmetic operations, one might end up having a total round-off error on the order of $\sqrt{N}\epsilon_m$ (when lucky)

(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.

Overflow and underflow, cancellation error

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, cancellation error

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.

Example: consider the difference between 123457.1467 and 123456.659 with 6-digit precision

```
  e=5; s=1.234571
- e=5; s=1.234567
----------------
  e=5; s=0.000004
  e=-1; s=4.000000 (after rounding and normalization)
```

which is 20% different from actual result is 0.4877. This is called cancellation error.

Catastrophic cancellations

Take for example

$$1e100 + 1 - 1e100 \; = \; ?$$

The result is zero... which is simply wrong. We call this a "catastrophic" cancellation.

Catastrophic cancellations

Take for example

$$1e100 + 1 - 1e100 \; = \; ?$$

The result is zero... which is simply wrong. We call this a "catastrophic" cancellation.

Catastrophic cancellation can occur in the evaluation of expressions like:

1. 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

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

Catastrophic cancellations

Take for example

$$1e100 + 1 - 1e100 \; = ?$$

The result is zero... which is simply wrong. We call this a "catastrophic" cancellation.

Catastrophic cancellation can occur in the evaluation of expressions like:

1. 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

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

Note: Although the expression $(x - y)\,(x + y)$ does not cause catastrophic cancellation, it is slightly less accurate then $x^2 - y^2$ if $x \gg y$ or $x \ll y$. In this case, $(x - y)\,(x + y)$ has three rounding errors, but $x^2 - y^2$ has only two, since the rounding error committed when computing the smaller of x^2 and y^2 does not affect the final subtraction.

Catastrophic cancellations /II

2. Quadratic formula:

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

cancellation occurs. A good solution is:

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

Catastrophic cancellations /II

2. Quadratic formula:

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

   cancellation occurs. A good solution is:

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

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 smallsr to the largest
   2. Kahan summation algorithm

Built-in mathematical functions

log1p(x)

Internally, all functions are implemented using Taylor expansions:

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

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

Built-in mathematical functions

log1p(x)

Internally, all functions are implemented using Taylor expansions:

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

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

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.

Built-in mathematical functions

hypot(a,b)

provides a numerically stable implementation of

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

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

Built-in mathematical functions

hypot(a,b)

provides a numerically stable implementation of

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

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

The C standard library provides `hypot(a,b)`:

$$c = m \cdot \sqrt{1 + (M/m)^2}$$

where $m = \mathbf{min}\left(|a|, |b|\right)$, $M = \mathbf{max}\left(|a|, |b|\right)$.

Implementation of functions

Sin cardinal

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

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

pdf

numerical instabilities appear due to the division between two nearly-zero numbers. A robust implementation comes from a careful consideration of this function.

Implementation of functions

Sin cardinal

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

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

pdf

numerical instabilities appear due to the division between two nearly-zero numbers. A robust implementation comes from a careful consideration of this function.

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

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

Implementation of functions

Sin cardinal

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

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

pdf

numerical instabilities appear due to the division between two nearly-zero numbers. A robust implementation comes from a careful consideration of this function.

Let's take the Taylor expansion $\mathrm{sinc}(x)$ 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:

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

Implementation of functions

Sin cardinal

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

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

pdf

numerical instabilities appear due to the division between two nearly-zero numbers. A robust implementation comes from a careful consideration of this function.

Let's take the Taylor expansion $\operatorname{sinc}(x)$ 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:

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

A similar approach might also be taken with functions like: $1 - \cos(x)$, $1 - \cosh(x)$, $\log(x)$, etc.

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

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

(Taylor expansion), therefore

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

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

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

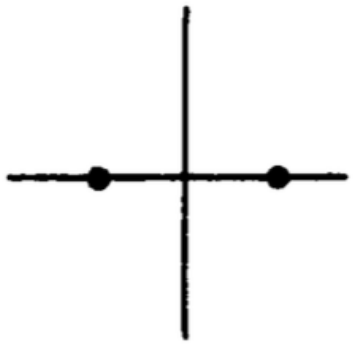
In this case, the truncation error is linearly proportional to h. Higher-order formulations of the first derivative give smaller error.

Finite difference formulae

Abramowitz and Stegun, page 883 and following

Partial Derivatives

25.3.21



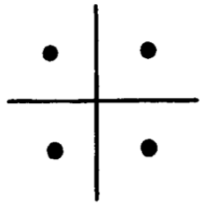
$$\frac{\partial f_{0,0}}{\partial x} = \frac{1}{2h}\left(f_{1,0} - f_{-1,0}\right) + O(h^2)$$

Finite difference formulae

Abramowitz and Stegun, page 883 and following

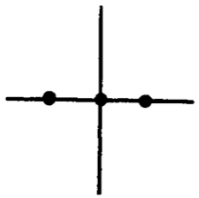
884 NUMERICAL ANALYSIS

25.3.22



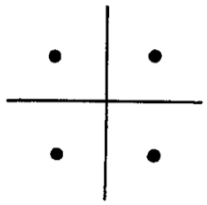
$$\frac{\partial f_{0,0}}{\partial x}=\frac{1}{4h}\,(f_{1,1}-f_{-1,1}+f_{1,-1}-f_{-1,-1})+O(h^2)$$

25.3.23



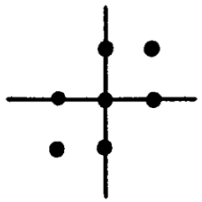
$$\frac{\partial^2 f_{0,0}}{\partial x^2}=\frac{1}{h^2}\,(f_{1,0}-2f_{0,0}+f_{-1,0})+O(h^2)$$

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



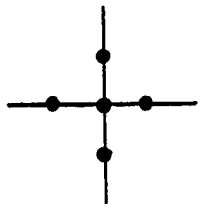
$$\frac{\partial^2 f_{0,0}}{\partial x\partial y}=\frac{-1}{2h^2}\,(f_{1,0}+f_{-1,0}+f_{0,1}+f_{0,-1}-2f_{0,0}-f_{1,1}-f_{-1,-1})+O(h^2)$$

Finite difference formulae

Abramowitz and Stegun, page 883 and following

Laplacian

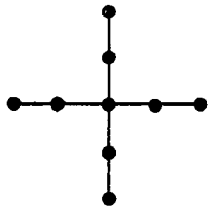
25.3.30



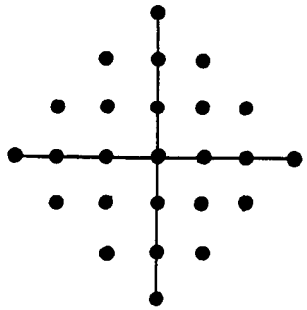
$$\nabla^2 u_{0,0} = \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right)_{0,0}$$

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

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

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

Numerical integration

Newton-Cotes formulas of the closed type.

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

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

Numerical integration

Newton-Cotes formulas of the closed type.

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:

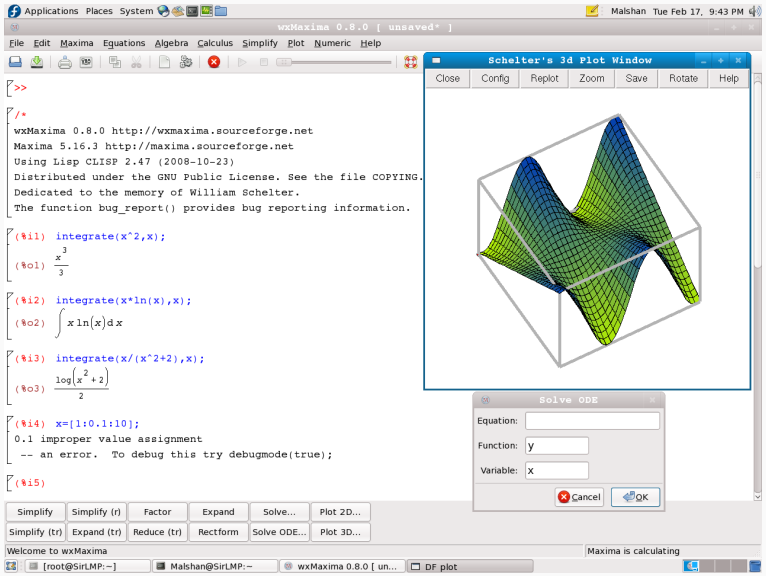
n	**Step size** h	**Common name**	**Formula**	**Error**
1	$b-a$	Trapezoidal rule	$\frac{h}{2}\,(f_0 + f_1)$	$-\frac{1}{12}h^3 f^{(2)}(\xi)$
2	$\frac{b-a}{2}$	Simpson's rule	$\frac{h}{3}\,(f_0 + 4f_1 + f_2)$	$-\frac{1}{90}h^5 f^{(4)}(\xi)$
3	$\frac{b-a}{3}$	Simpson's 3/8 rule	$\frac{3h}{8}\,(f_0 + 3f_1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

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©, 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



- **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»

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. Python is most praised for its elegant syntax and readable code, if you are just beginning your programming career python suits you best.

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. Python is most praised for its elegant syntax and readable code, if you are just beginning your programming career python suits you best.

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. Python is most praised for its elegant syntax and readable code, if you are just beginning your programming career python suits you best.

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

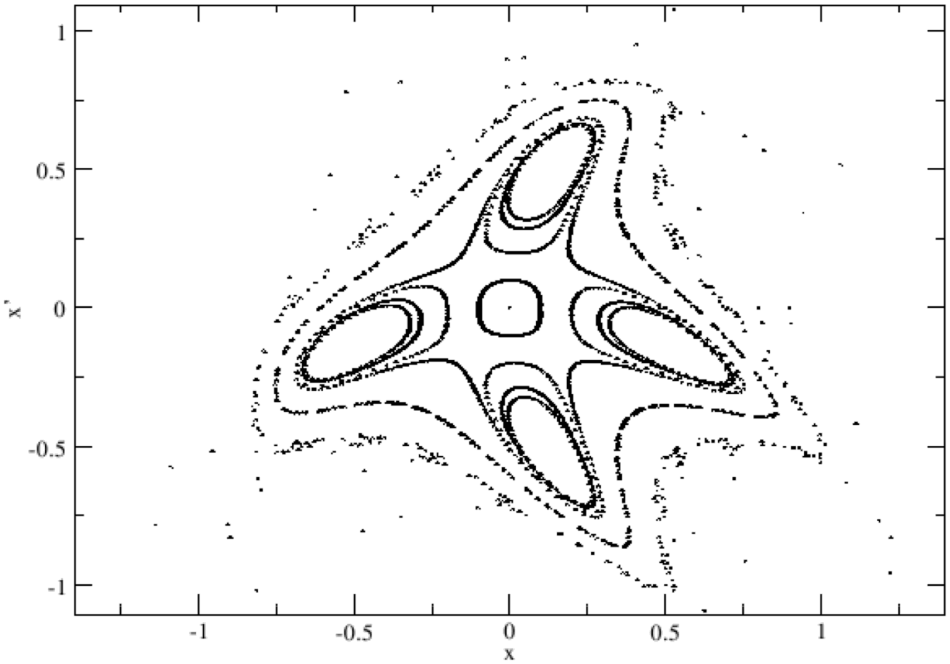
Octave is detailed as "A programming language for scientific computing". It is software featuring a high-level programming language, primarily intended for numerical computations. Octave helps in solving linear and nonlinear problems numerically, and for performing other numerical experiments using a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

`https://www.octave.org`

`https://octave.sourceforge.io`

Tools: 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)

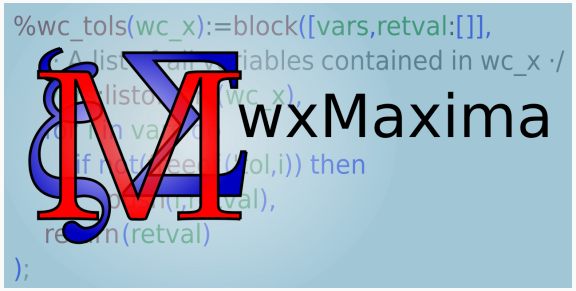
Tools: Symbolic computation

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.



An excellent front end for Maxima is wxMaxima.

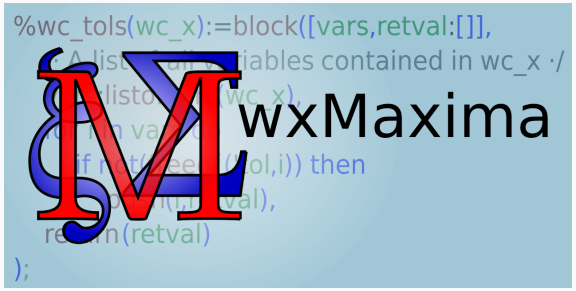
Tools: Symbolic computation

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.



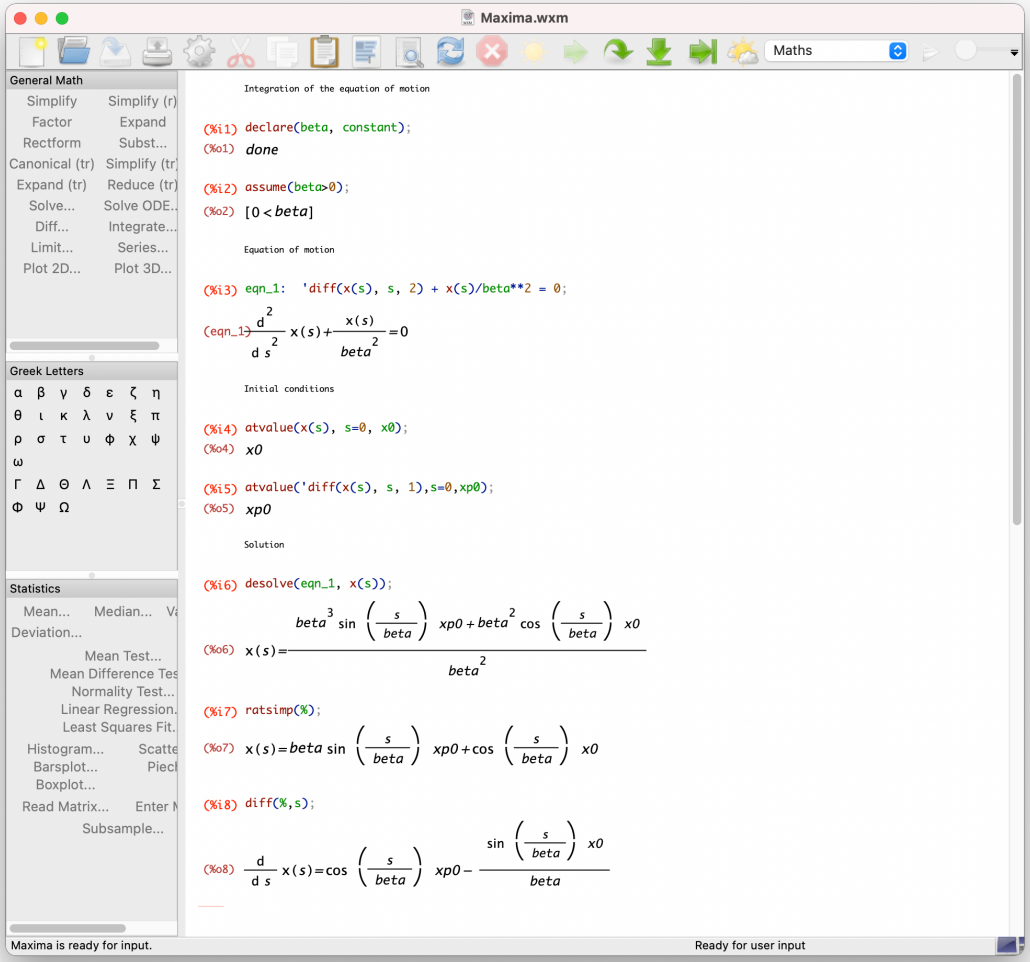
An excellent front end for Maxima is wxMaxima.

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

The Octave “symbolic” package

```
% Load the symbolic package
pkg load symbolic

% This is just a formula to start with, have fun and change it if you want to.
f = @(x) x.^2 + 3*x - 1 + 5*x.*sin(x);

% These next lines take the Anonymous function into a symbolic formula
syms x;
ff = f(x);

% Now we can calculate the derivative of the function
ffd = diff(ff, x);

% and convert it back to an Anonymous function
df = function_handle(ffd)
```

Tools: Symbolic computation

The Octave “symbolic” package

```
% Load the symbolic package
pkg load symbolic

% This is just a formula to start with, have fun and change it if you want to.
f = @(x) x.^2 + 3*x - 1 + 5*x.*sin(x);

% These next lines take the Anonymous function into a symbolic formula
syms x;
ff = f(x);

% Now we can calculate the derivative of the function
ffd = diff(ff, x);

% and convert it back to an Anonymous function
df = function_handle(ffd)
```

The Python “sympy” library

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

```
$ 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 an electron 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
```

Shell scientific tools

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

Use of named pipes for interprocess communication (of 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.

Use of named pipes for interprocess communication (of 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.

Use of named pipes for interprocess communication (of 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) is not suitable for accelerator physics. 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
```

A word about the choice of units...

The International System (SI) is not suitable for accelerator physics. 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...

A 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...

A 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("%.15g\n", x);
```

A word about data files...

A 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("%.15g\n", x);
```

In C++, use:

```
std::cout << std::setprecision(15) << x << std::endl;
```

You want to look for bit-wise preservation of the information.

Accelerator physics codes

MAD-X

MAD-X is a CERN code used world-wide, with a long history going back to the 80's in the field of high energy beam physics (i.e. MAD8, MAD9, MADX). It is an all-in-one application with its own scripting language used to design, simulate and optimise particle accelerators: lattice description, machine survey, single particles 6D tracking, optics modelling, beam simulation & analysis, machine optimisation, errors handling, orbit correction, aperture margin and emittance equilibrium.

SixTrack

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

PyHEADTAIL

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

Accelerator physics codes /II

PLACET

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. In includes also the emission of incoherent and coherent synchrotron radiation. Interfaced with Tcl, Octave, and Python.

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.

Accelerator physics codes /III

ASTRA

"A Space Charge Tracking Algorithm" is a tracking code developed at DESY (Hamburg, Germany), can simulate injectors and track in field maps. Uses its own description language.

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. Interfaced with Python and Octave.

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

The end.

Thank you for your attention.

Any questions?