# Numerical Methods in Engineering and Applied Science

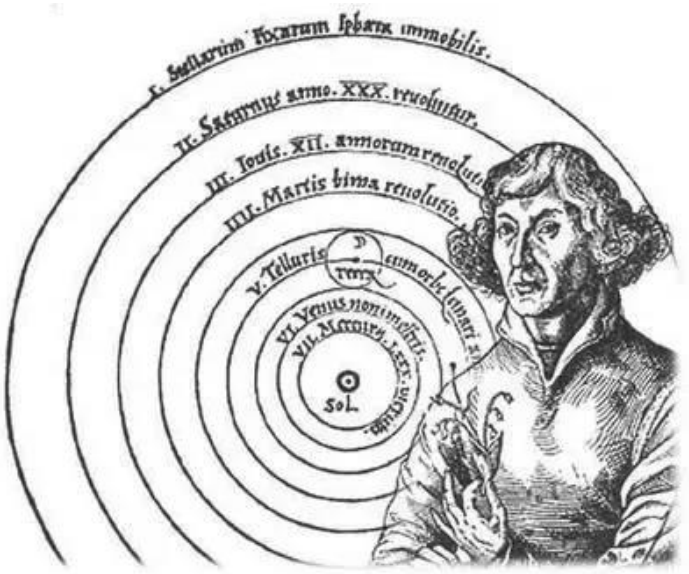
**Dmitry Kolomenskiy**
D.Kolomenskiy@skoltech.ru
Online & E-R3-2002

**TA: Oleg Volgin**

Skoltech, Term 3, 2022

# Why mathematical modelling?

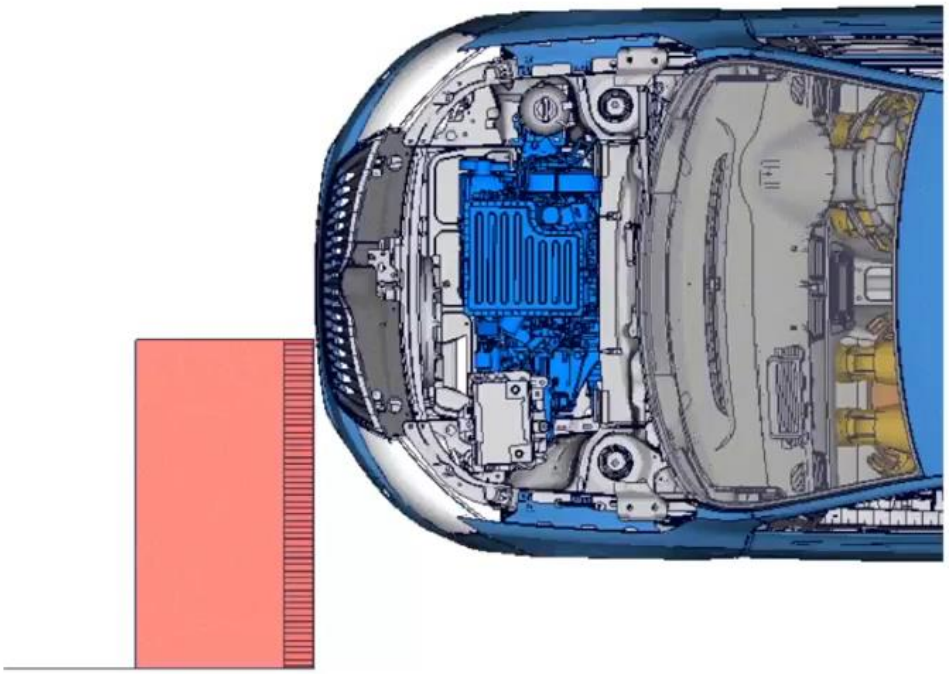
- Because it may be impossible to set up a repeatable experiment

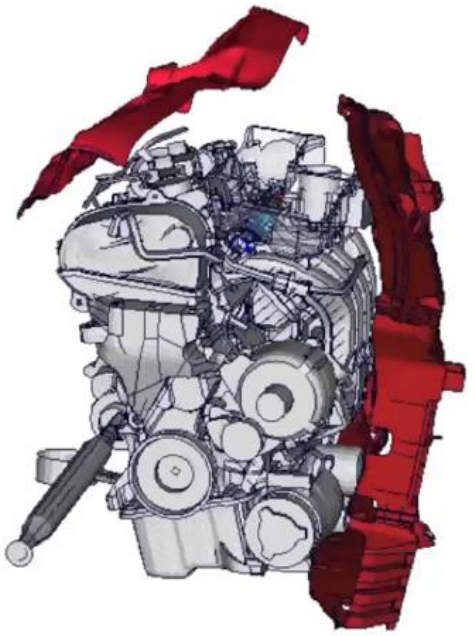



https://en.wikipedia.org/wiki/Orrery#/media/File:Orrery_small.jpg

# Why mathematical modelling?

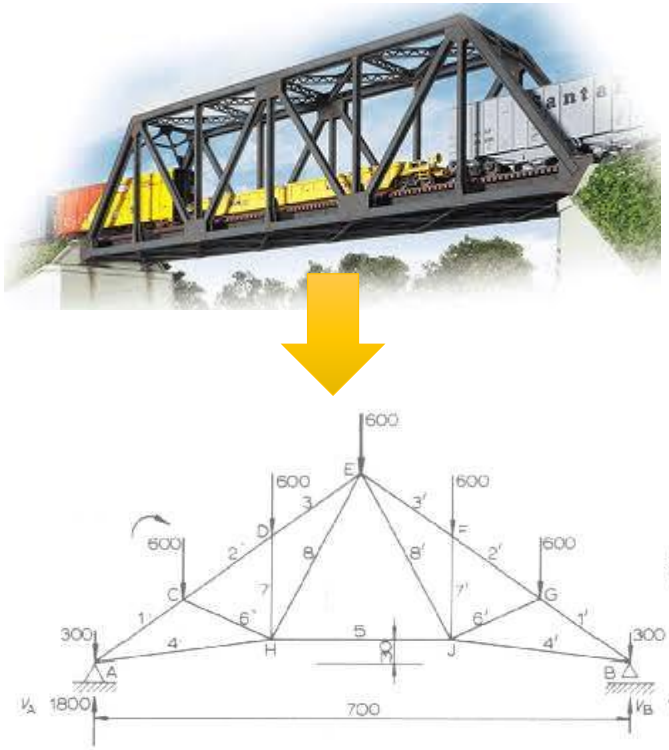
- because repeatable experiments may be too expensive

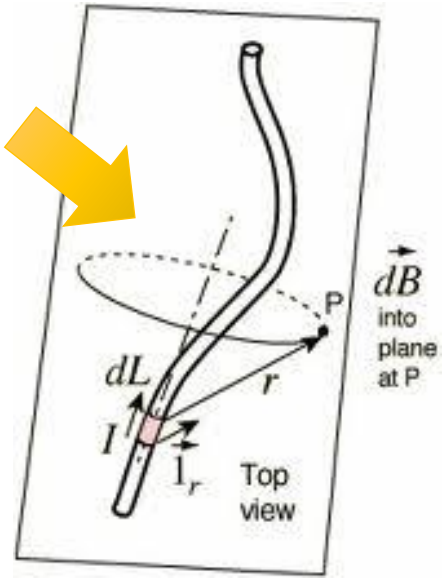




https://youtu.be/SyfwsaPSSQE

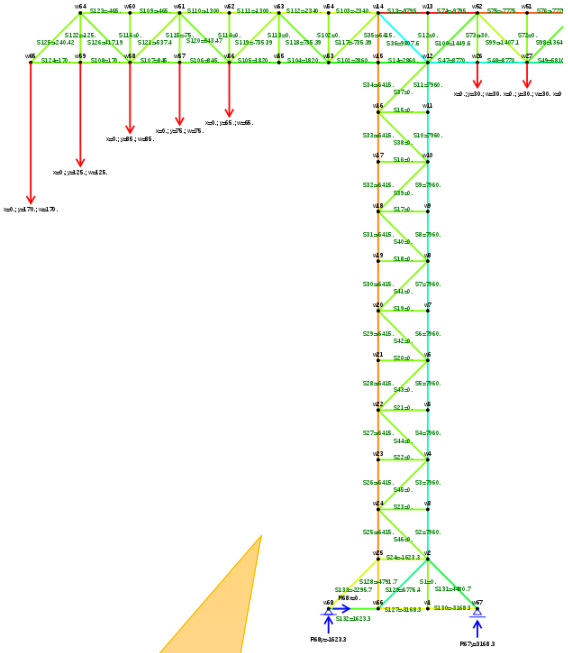
# What is a mathematical model?



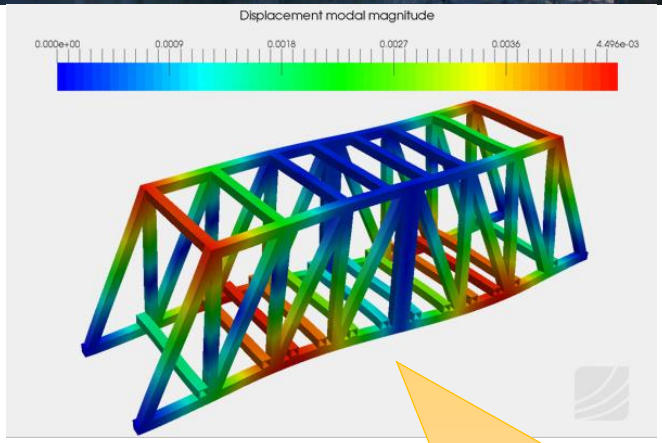




**A mathematical model is a description of a system using mathematical concepts and language.**

# What is a mathematical model?



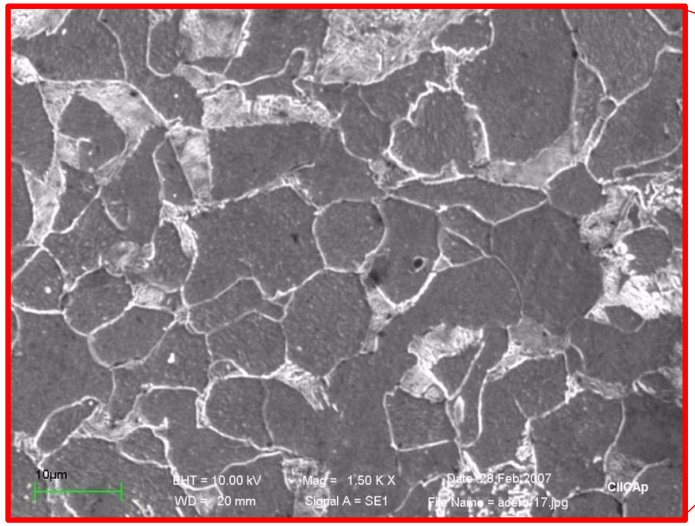
A simple model



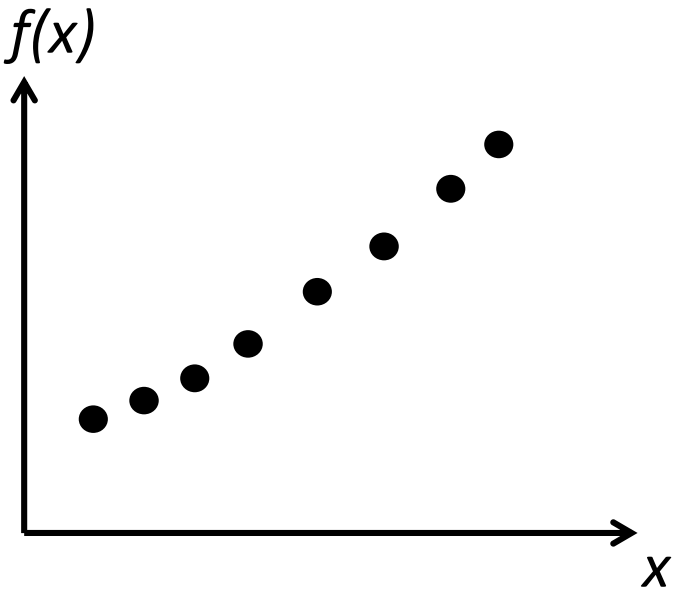

A less simple model

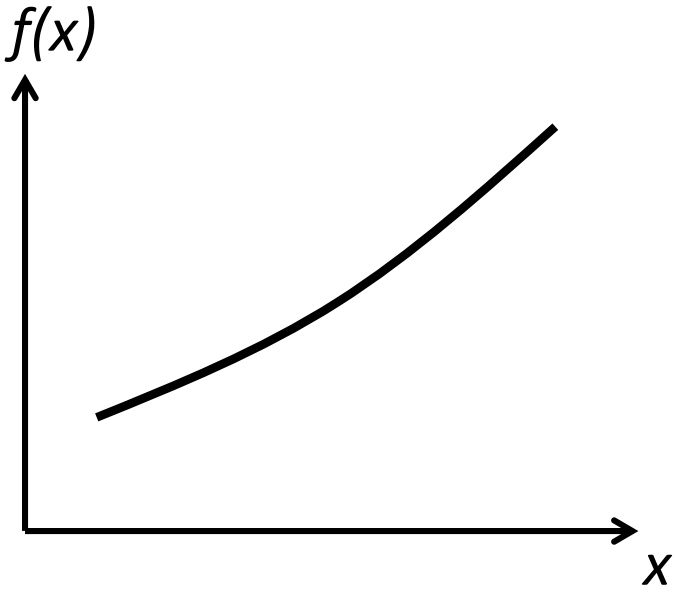
**A mathematical model of a real-world object is always an incomplete representation of it.**

# Continuum

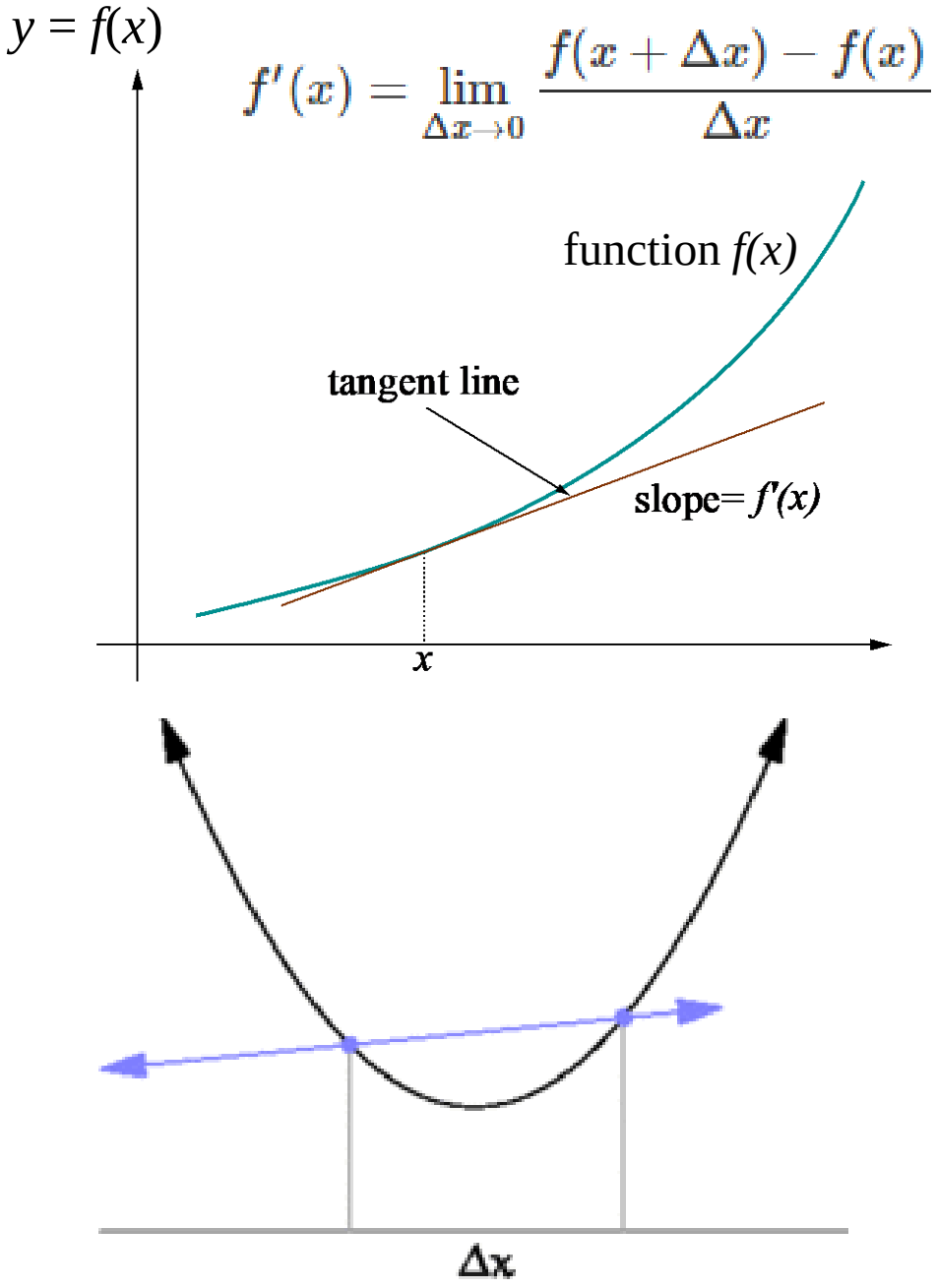



C. Cuevas-Arteaga, *International Journal of Electrochemical Science* 7(1):445-470, 2012



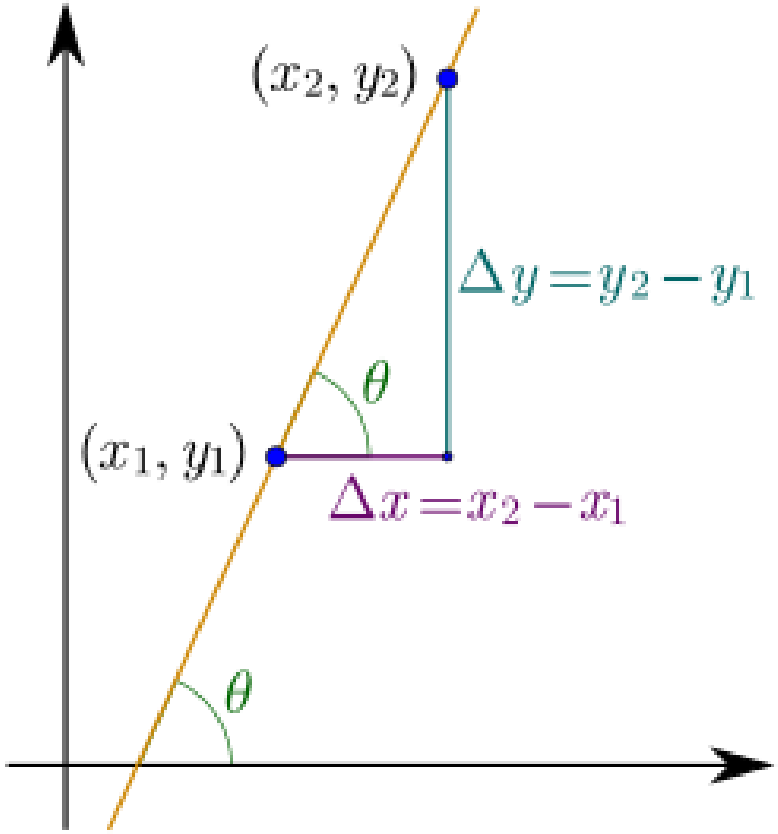
# Derivatives




**Notations:**

$$f'(x), \dot{f}(x), \frac{df}{dx}$$

**Slope of a straight line** $m = \frac{\Delta y}{\Delta x}$



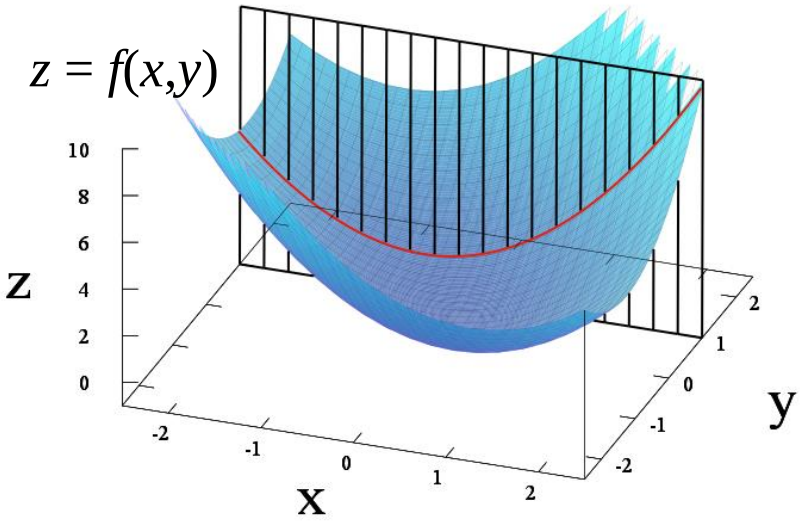
# Partial derivatives

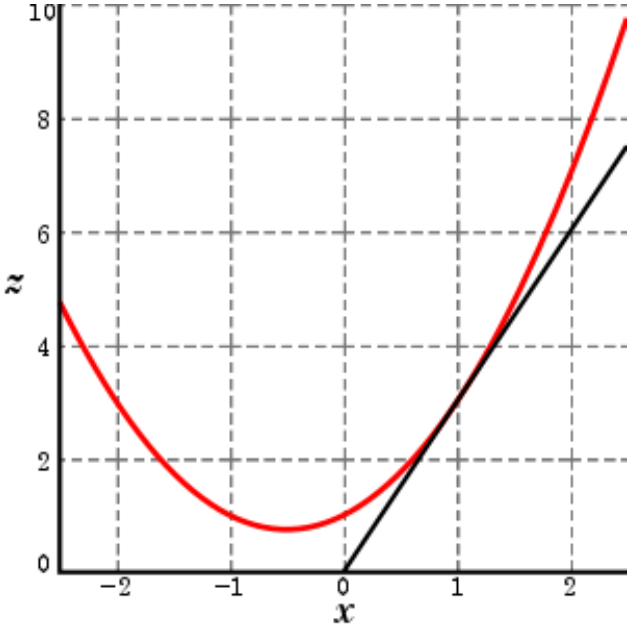
**A partial derivative of a function of several variables is its derivative with respect to one of those variables, with the others held constant**

**Notations:** $f'_x,\ f_x,\ \partial_x f,\ \dfrac{\partial}{\partial x} f,$ or $\dfrac{\partial f}{\partial x}.$

Partial derivative of function $f(x,y)$ with respect to $x$ at the point $x=a$, $y=b$

$$\left.\frac{\partial f}{\partial x}\right|_{(a,b)} = \lim_{\Delta x \to 0} \frac{f(a+\Delta x, b) - f(a,b)}{\Delta x}$$



# Differential equations

**A differential equation is an equation that relates some function with its derivatives**

**Example:**

$$\frac{dy}{dx} = \cos x \qquad y(x) = ?$$

$$y(x) = \int \cos x \, dx + C \qquad y(x) = \sin x + C$$

$$C = ? \qquad C = const$$

$$\text{Let } x = 0. \quad \text{Then } y(0) = \sin 0 + C \qquad C = y(0)$$

$y(0)$ is the initial condition

**An initial value problem is an ordinary differential equation together with a specified initial condition**

# Differential equations

A differential equation is an equation that relates some function with its derivatives.

An initial value problem is an ordinary differential equation together with a specified initial condition.

**Example:**

$$\begin{cases} \dfrac{dy}{dx} = y\cos x & y(x) = ? \\ y(0) = y_0 \end{cases}$$

if $y \neq 0$

$$\int \frac{dy}{y} = \int \cos x dx + C \qquad \ln y = \sin x + C$$

$$C = \ln y_0$$

$$y(x) = y_0 e^{\sin x}$$

# Differential equations

**One more example:**

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + \frac{\partial^2 u}{\partial x^2} + \nu\frac{\partial^4 u}{\partial x^4} = 0 \qquad u(x,t) = ?$$

with initial condition $u(x,0)=u_0(x)$ and suitable boundary conditions at $x$=-1 and $x$=1.

**One-dimensional Kuramoto-Sivashinsky (KS) equation**

**No closed-form general solution**

# Finite-difference numerical approximation

**Definition of the derivative**

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

**Approximation**

$$f'(x) = \frac{f(x+\Delta x) - f(x)}{\Delta x} + error$$

$$error \to 0 \quad \text{as} \quad \Delta x \to 0$$

**Example:**

$$f(x) = \sin(x)$$

**Exact value of the derivative at $x=\pi/3$**

$$f'(\pi/3) = 1/2$$

**Approximate value of the derivative at $x=\pi/3$**

| $\Delta x$ | $\frac{f(\pi/3+\Delta x) - f(\pi/3)}{\Delta x}$ | |
|---|---|---|
| 0.1 | 0.455901885410761 | 9% error |
| 0.01 | 0.495661575773687 | |
| 0.001 | 0.499566904000770 | |
| 0.0001 | 0.499956697895820 | 0.009% |

# Numerical approximation of differential equations

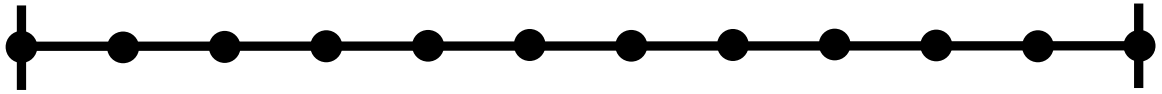
**Example:**

$$\begin{cases} \dfrac{dy}{dx} = y\cos x \\ y(0) = y_0 \end{cases}$$

**Discretization grid**

$$\text{Let} \quad x \in [0, 6\pi] \quad \text{and} \quad x_j = j \cdot \Delta x,$$

$$\text{where} \quad \Delta x = 6\pi/N, \quad j = 0, 1, \ldots, N$$




**Approximation**

$$\frac{y(x_{j+1}) - y(x_j)}{\Delta x} = y(x_j)\cos x_j$$

**Recursive formula**

$$y(x_{j+1}) = y(x_j) + \Delta x \cdot y(x_j)\cos x_j$$

$$y(0) = y_0$$

$$y(\Delta x) = y_0 + \Delta x \cdot y_0$$

$$y(2\Delta x) = y(\Delta x) + \Delta x \cdot y(\Delta x)\cos \Delta x$$

$$\ldots$$

# Numerical approximation of differential equations

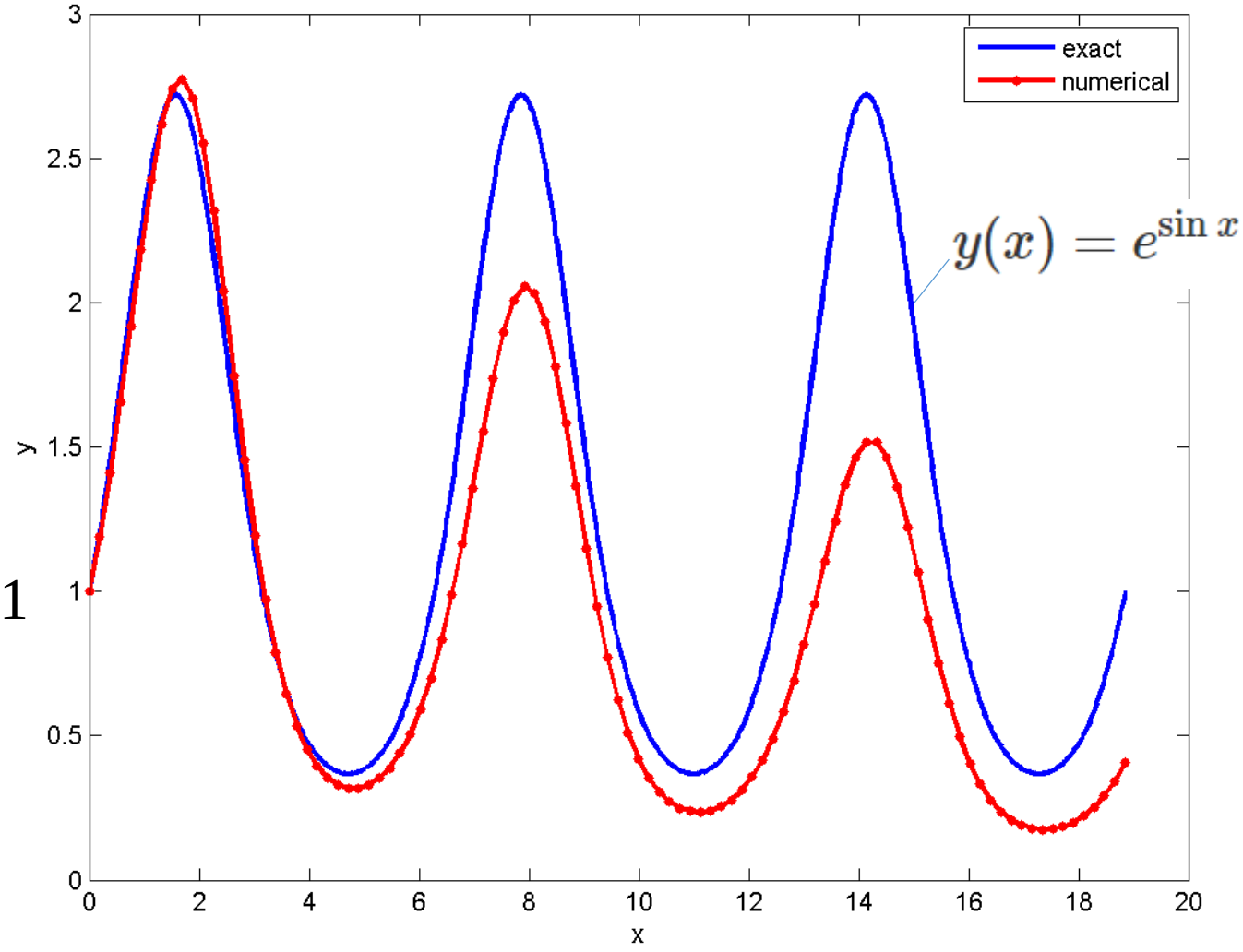
## Computer program in Matlab

```
% Number of grid points
n = 10;
% Domain size
lx = 6*pi;
% Initial condition
y0 = 1;
% Discretization step
dx = lx/n;
% Independent variable
x = (0:n).'*dx;
% Initialize the approximate solution
y = zeros(n+1,1);
y(1) = y0;
% Compute the approximate solution
for j=1:n
    y(j+1) = y(j) + dx*y(j)*cos(x(j));
end;
```

# Numerical approximation of differential equations

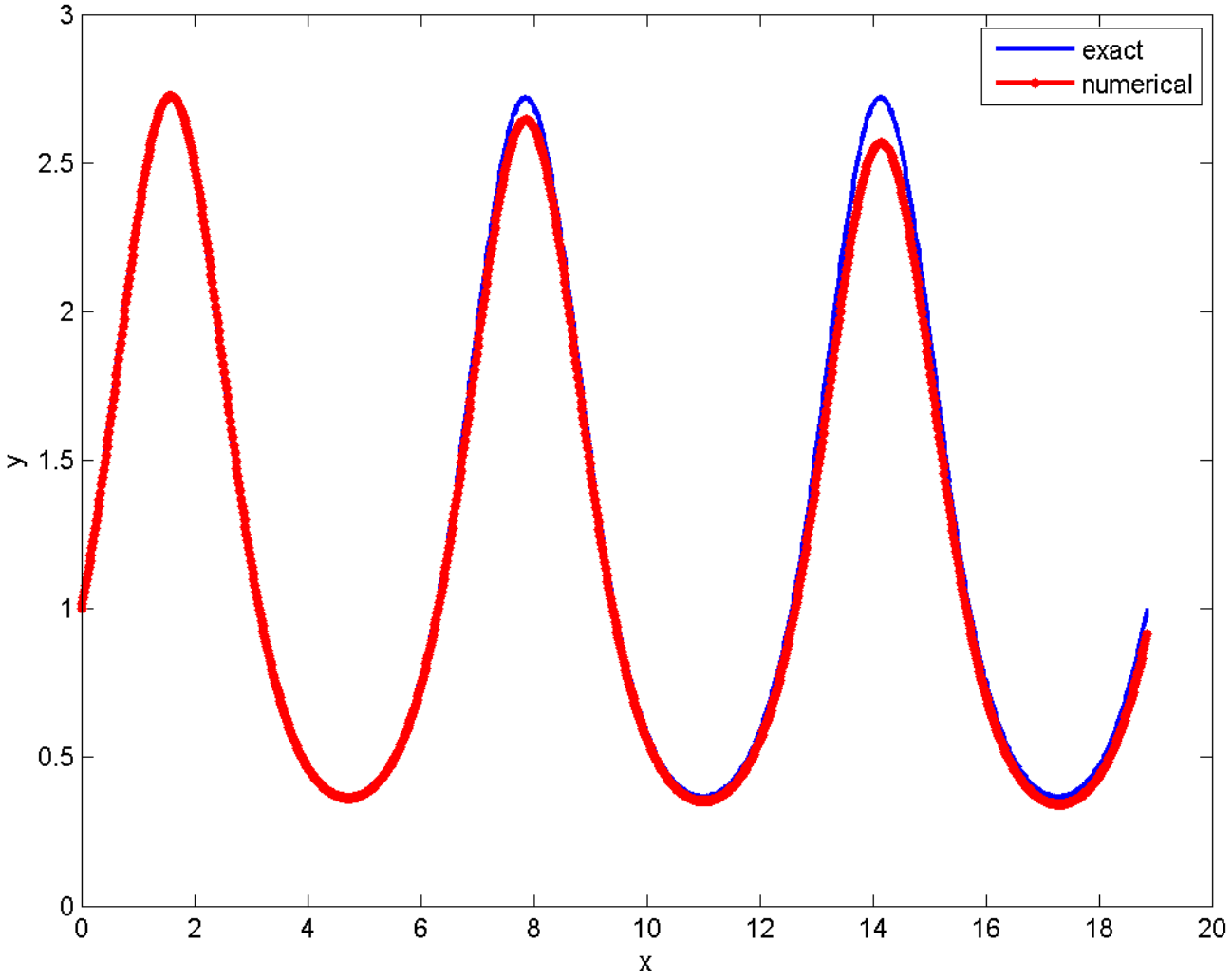
N=100 discretization grid points, $\Delta x=1/100$

Let $y(0)=1$



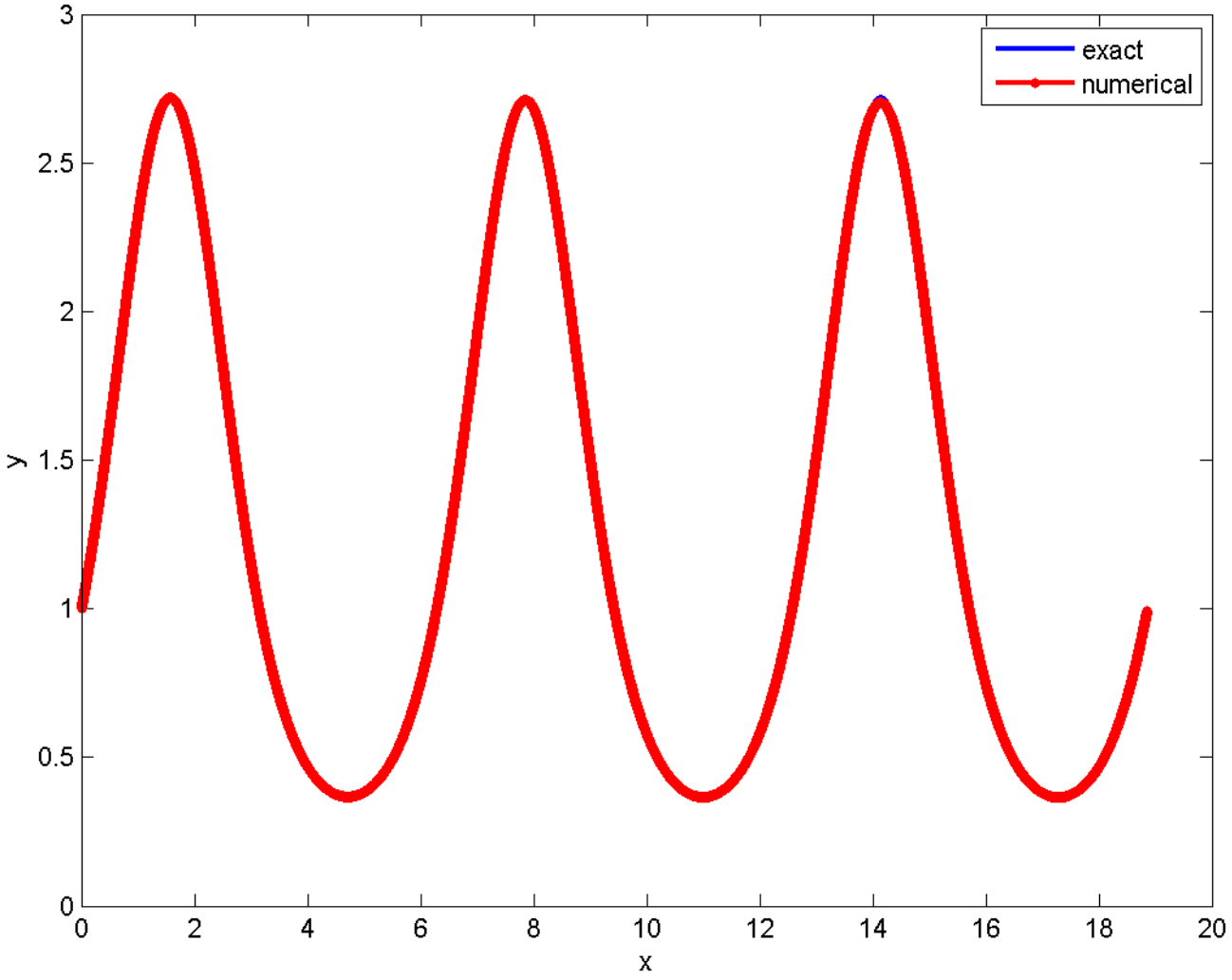
# Numerical approximation of differential equations

N=1000 discretization grid points, $\Delta x = 1/1000$



# Numerical approximation of differential equations

N=10000 discretization grid points, $\Delta x = 1/10000$



# Numerical solution of partial differential equations

## Navier-Stokes equations

$$\rho\left(\frac{\partial v_x}{\partial t}+v_x\frac{\partial v_x}{\partial x}+v_y\frac{\partial v_x}{\partial y}+v_z\frac{\partial v_x}{\partial z}\right) = -\frac{\partial P}{\partial x}+\mu\left(\frac{\partial^2 v_x}{\partial x^2}+\frac{\partial^2 v_x}{\partial y^2}+\frac{\partial^2 v_x}{\partial z^2}\right)+\rho g_x \qquad \frac{\partial v_x}{\partial x}+\frac{\partial v_y}{\partial y}+\frac{\partial v_z}{\partial z} = 0$$

$$\rho\left(\frac{\partial v_y}{\partial t}+v_x\frac{\partial v_y}{\partial x}+v_y\frac{\partial v_y}{\partial y}+v_z\frac{\partial v_y}{\partial z}\right) = -\frac{\partial P}{\partial y}+\mu\left(\frac{\partial^2 v_y}{\partial x^2}+\frac{\partial^2 v_y}{\partial y^2}+\frac{\partial^2 v_y}{\partial z^2}\right)+\rho g_y$$

$$\rho\left(\frac{\partial v_z}{\partial t}+v_x\frac{\partial v_z}{\partial x}+v_y\frac{\partial v_z}{\partial y}+v_z\frac{\partial v_z}{\partial z}\right) = -\frac{\partial P}{\partial z}+\mu\left(\frac{\partial^2 v_z}{\partial x^2}+\frac{\partial^2 v_z}{\partial y^2}+\frac{\partial^2 v_z}{\partial z^2}\right)+\rho g_z$$

$x, y, z, t$ : **3**+1 independent variables

Distretization $x_j=\{j\Delta x\}$, $y_j=\{j\Delta y\}$, $z_j=\{j\Delta z\}$, $j=1,\ldots,N$ Let $N$=**10000**

$t_k=\{k\Delta t\}$, $k=1,\ldots,M$

$v_x(x,y,z,t)$, $v_y(x,y,z,t)$, $v_z(x,y,z,t)$, $P(x,y,z,t)$ : **4** scalar fields

**At every time step $t_k$, we need to store $4\cdot 10000^3$ real numbers**

**We need 29 Tb volatile memory (RAM)!**

**Accurate and efficient numerical methods**

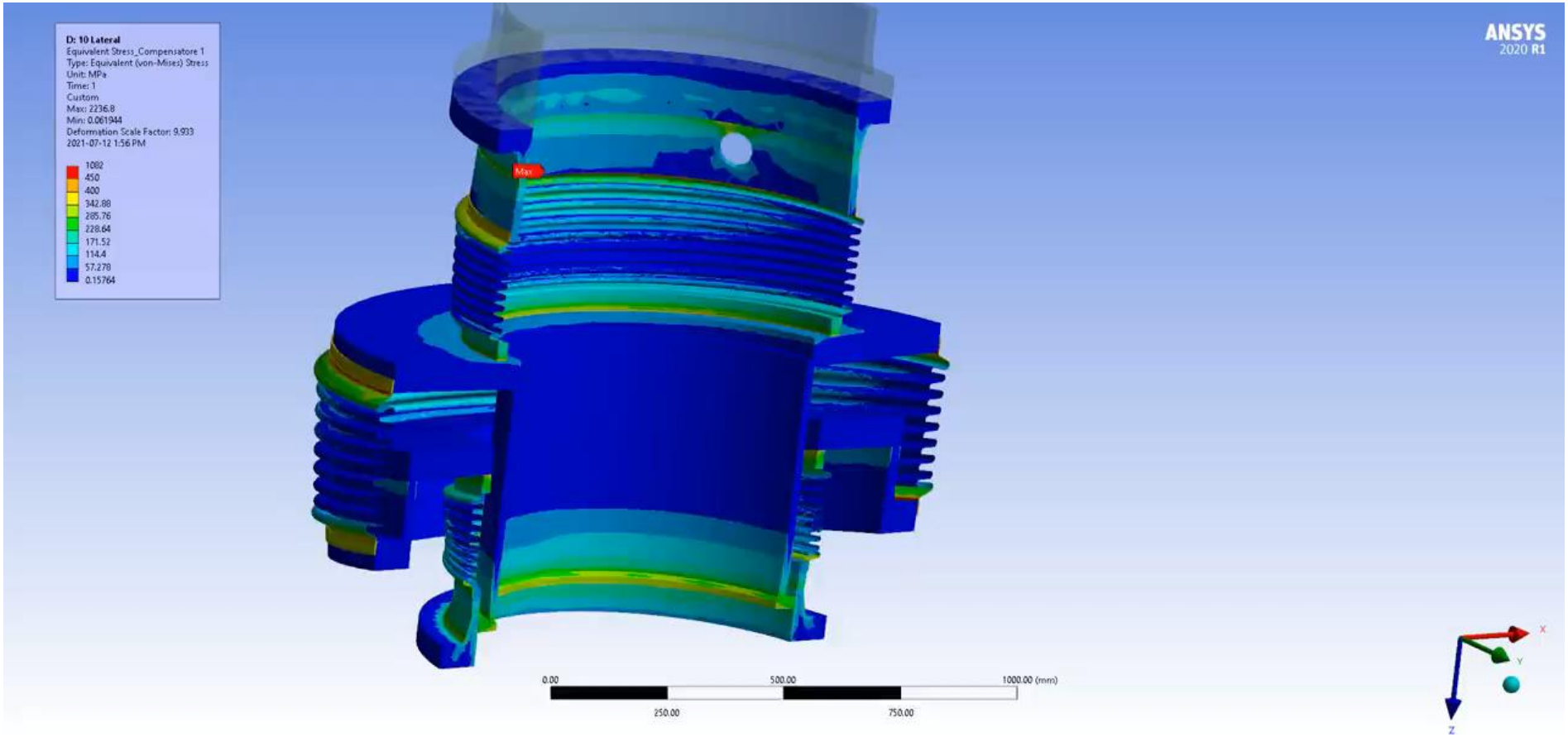
**Powerful computers**

# Numerical simulation

**A numerical simulation is a calculation that is run on a computer following a program that implements a mathematical model.**

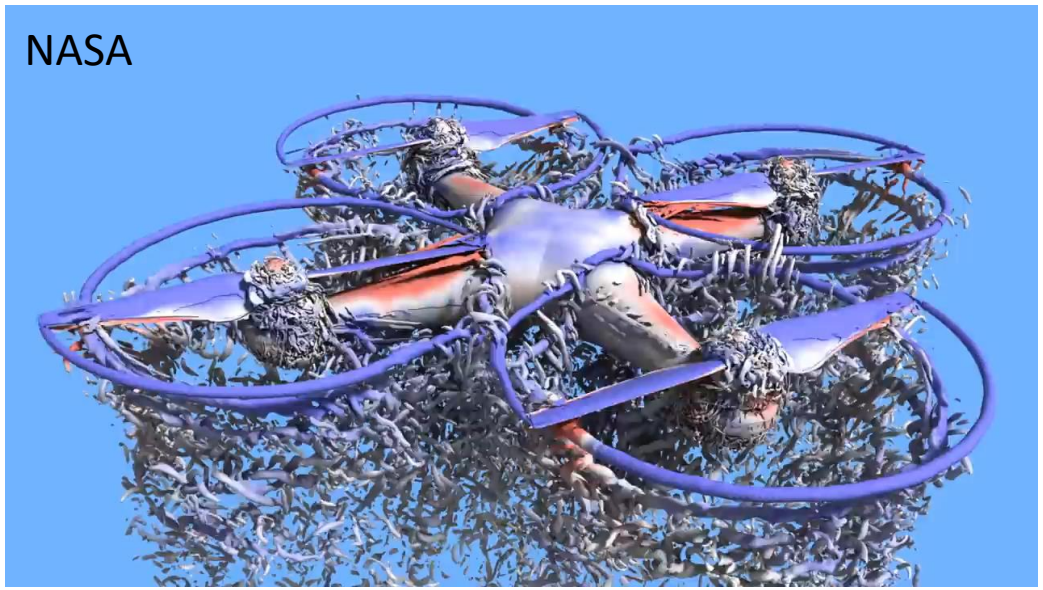
**The mathematical models of many problems in science and engineering are systems of differential equations. Therefore, the main focus of this course will be on numerical methods for differential equations.**

# Solid mechanics

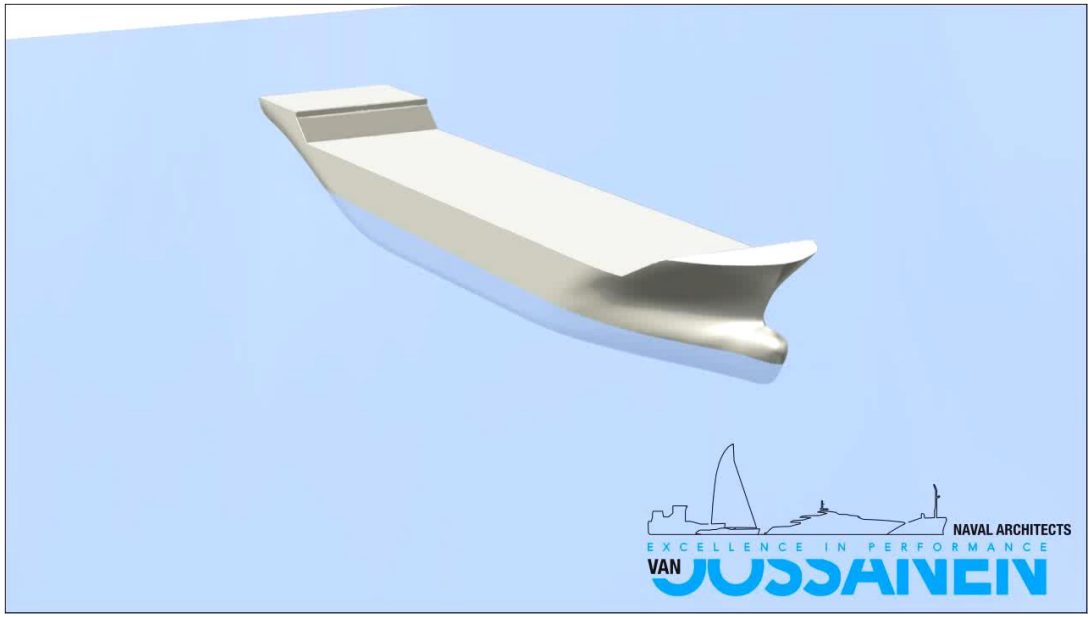


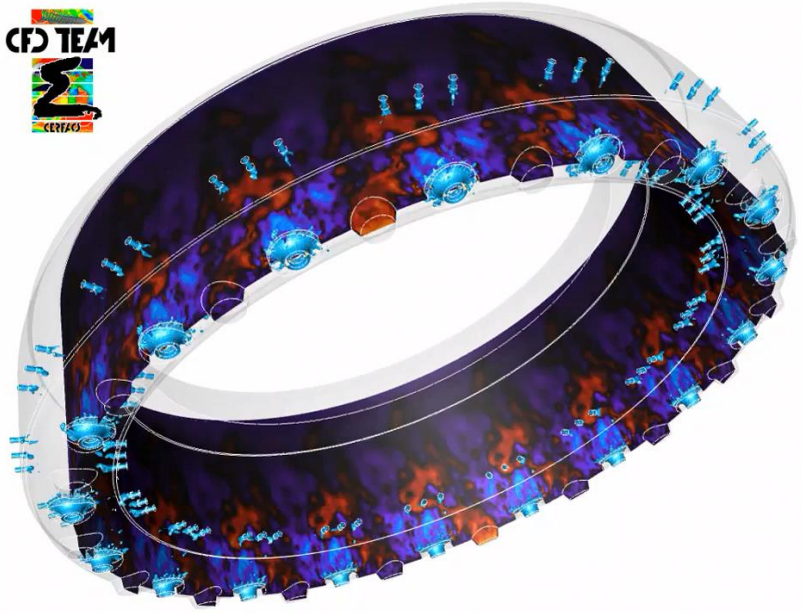

https://youtu.be/jIUalovX0rI

# Computational fluid dynamics

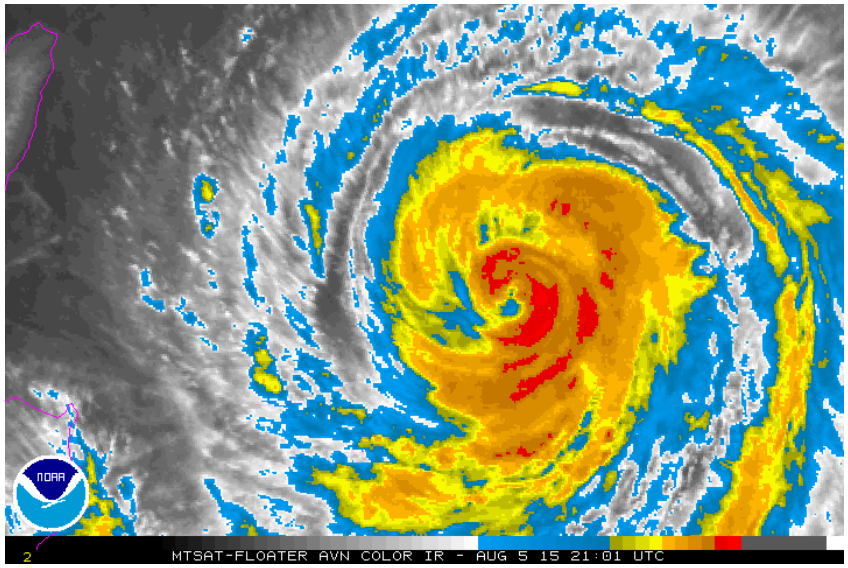

www.nasa.gov/image-feature/ames/exploring-drone-aerodynamics-with-computers

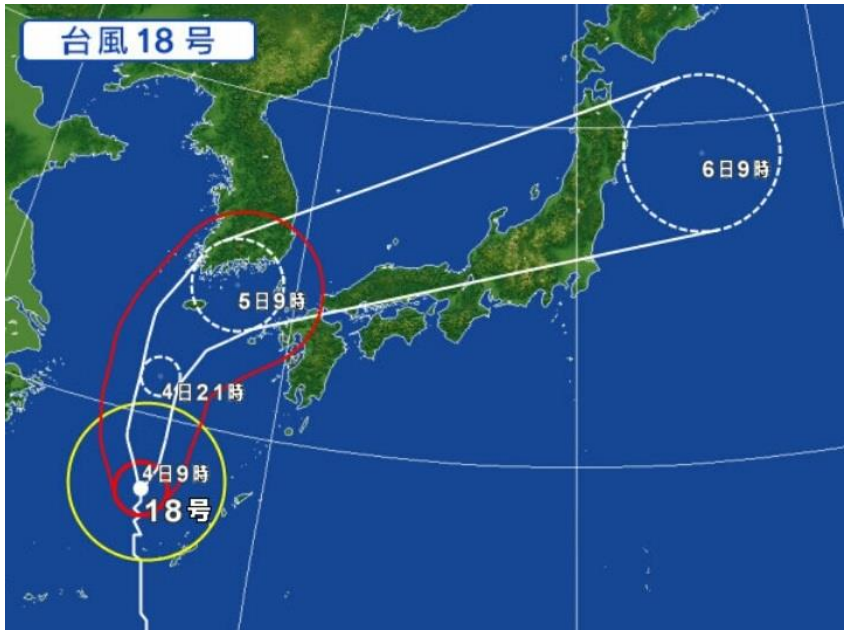


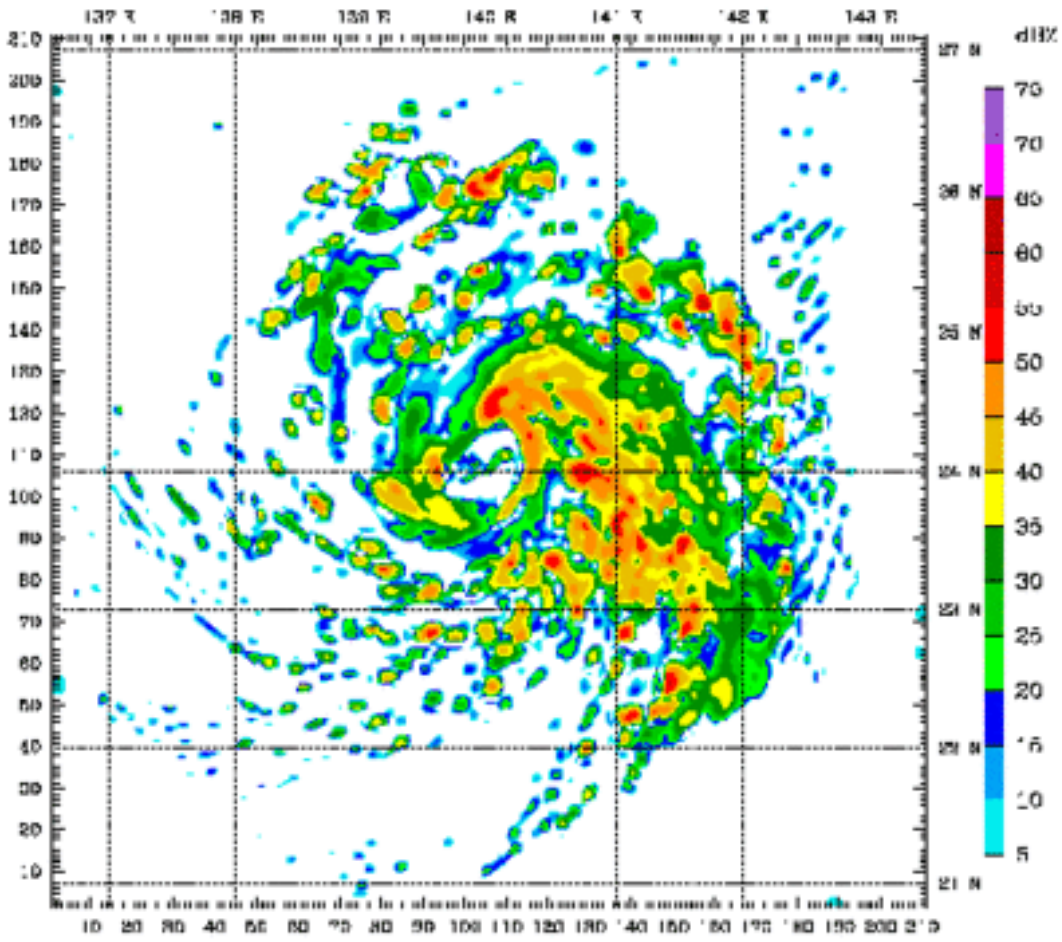




https://cerfacs.fr/combustion-reacting-flows/

# Meteorology

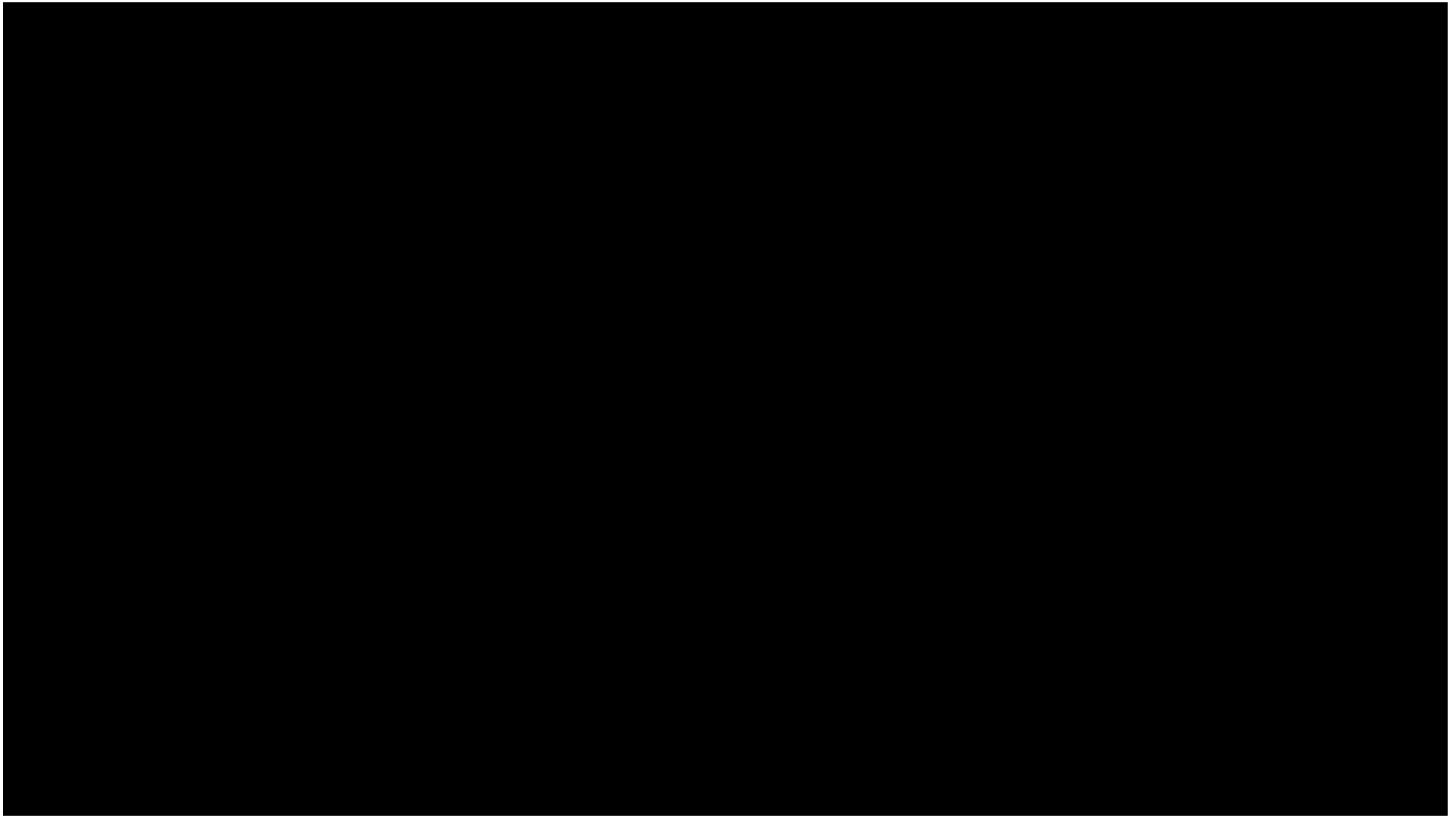


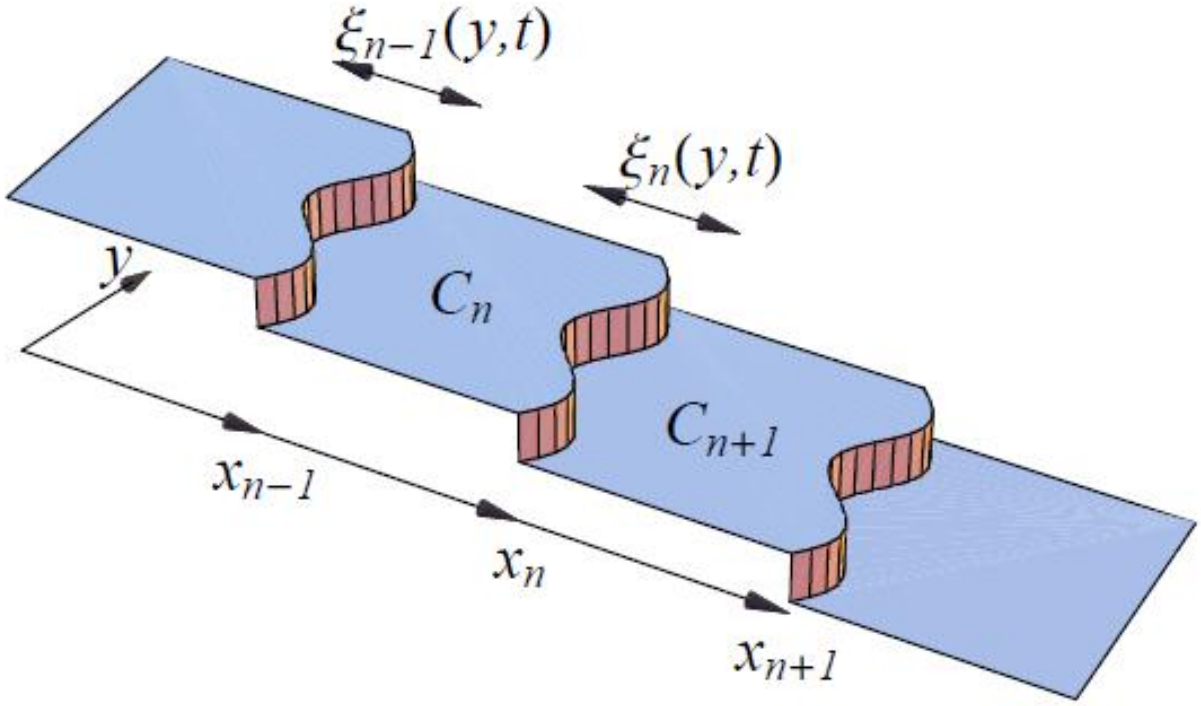
A 48-hour computer simulation of Typhoon Mawar using the Weather Research and Forecasting model. https://en.wikipedia.org/wiki/Computer_simulation

# Astrophysics



First 150 milliseconds of a supernova explosion. Colors represent temperatures of spewing gases.
https://svs.gsfc.nasa.gov/11735

# Crystal growth




$$\partial_t h = -\partial_y \left[ \frac{1}{1+(\partial_y h)^2} \left( \partial_y h + \partial_y \left( \frac{\partial_y^2 h}{(1+(\partial_y h)^2)^{3/2}} \right) \right) \right]$$

T. Frisch & A. Verga https://arxiv.org/abs/cond-mat/0607553v1

≈ 600 BC: A

before 179

1620-27: **Sl**

1642: **Pasca**

1759-61: I

1834: **Babb**

1883: **Ada**

1946: **ENIA**

1947: **Cra**

1960: **Lax**

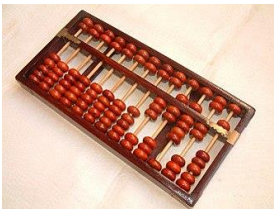
2020: **Fuga**

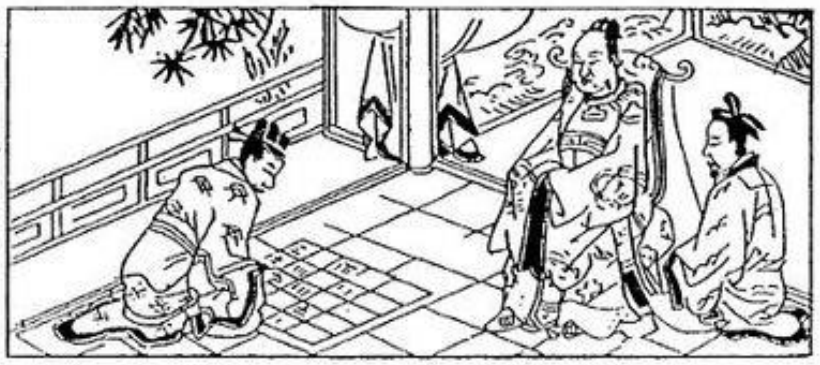
ose **computer**.

nal

Science, Japan: #1 in Top 500 since 06/2020.

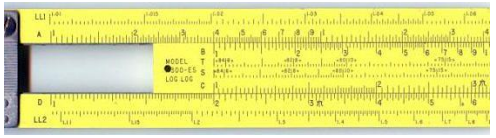
≈ 600 BC: **Abacus**.





before 179: **Gaussian elimination**.

1620-27: **Slide rule**.



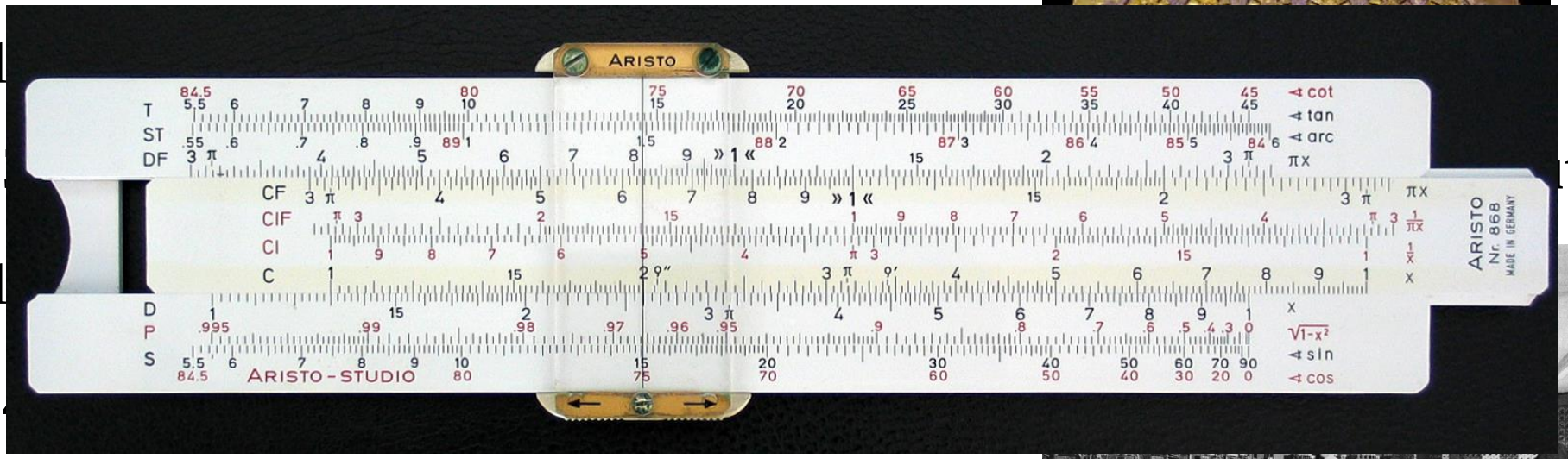
1642: **Pascal's mechanical calculator**.

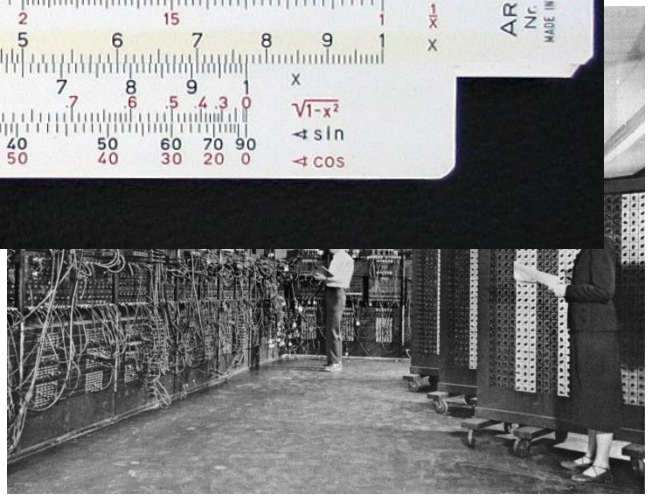
1

18 ter.

1

19



1947: **Crank–Nicolson** method.

1960: **Lax–Wendroff** method.

2020: **Fugaku -** Fujitsu RIKEN Center for Computational Science, Japan: #1 in Top 500 since 06/2020.

≈ 600 BC: **Abacus**.

be

162

164

17

183

**uter**.

18

194

19

19




2020: **Fugaku -** Fujitsu RIKEN Center for Computational Science, Japan:  #1 in Top 500 since 06/2020.

≈ 6

b

16

16

1

18

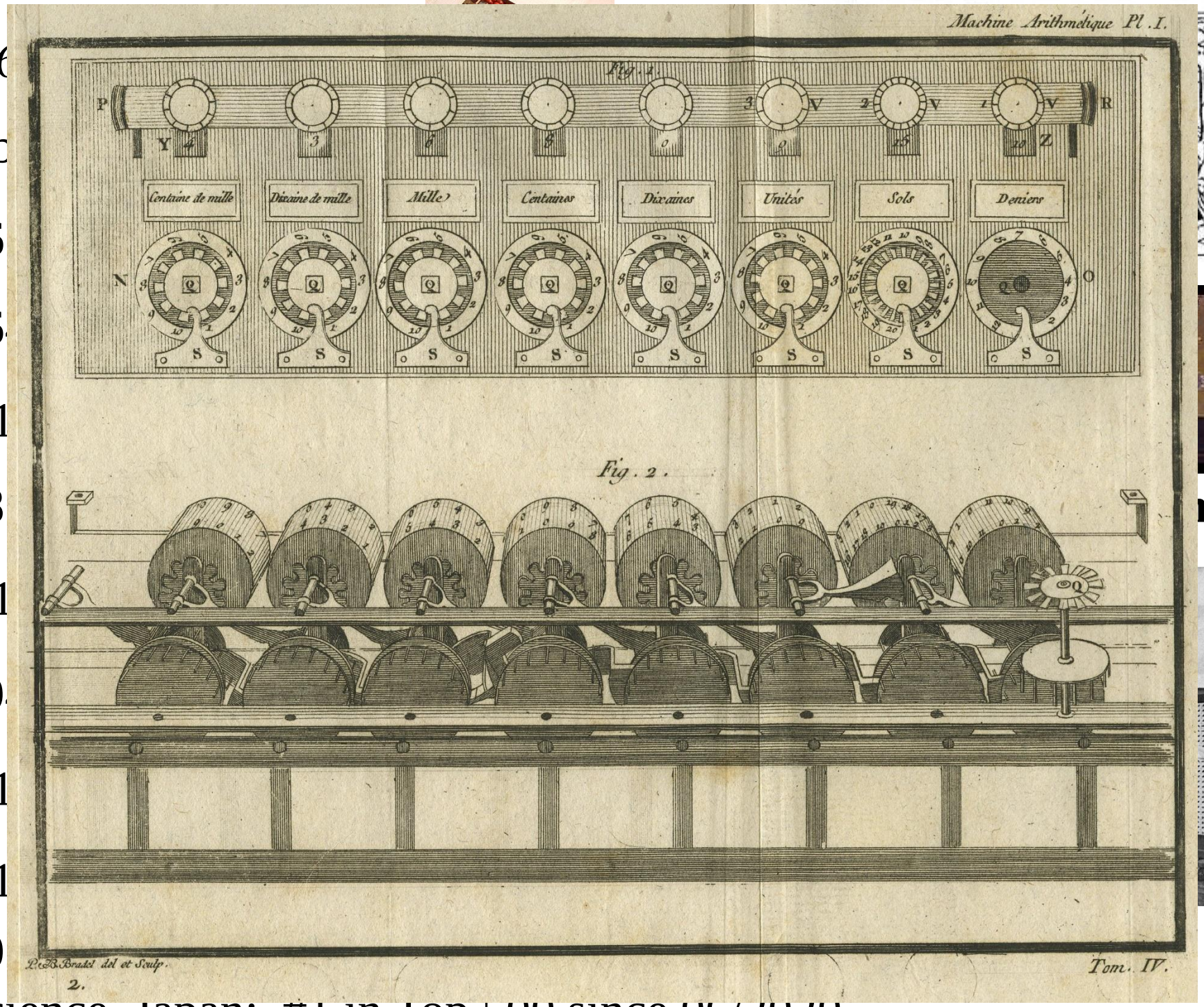
1

19

1

1

20




**nputer.**

Science, Japan: #1 in Top 500 since 06/2020.

≈ 600

befo

1620-2

1642:

1759

1834:

1883

1946:

1947

1960




**mputer.**

2020: **Fugaku -** Fujitsu RIKEN Center for Computational Science, Japan: #1 in Top 500 since 06/2020.

≈ 60

bef

1620

1642

17

1834

188

1946

19

19

2020


**mputer.**

Scien

≈ 6

b

16

16

1

18

ter.

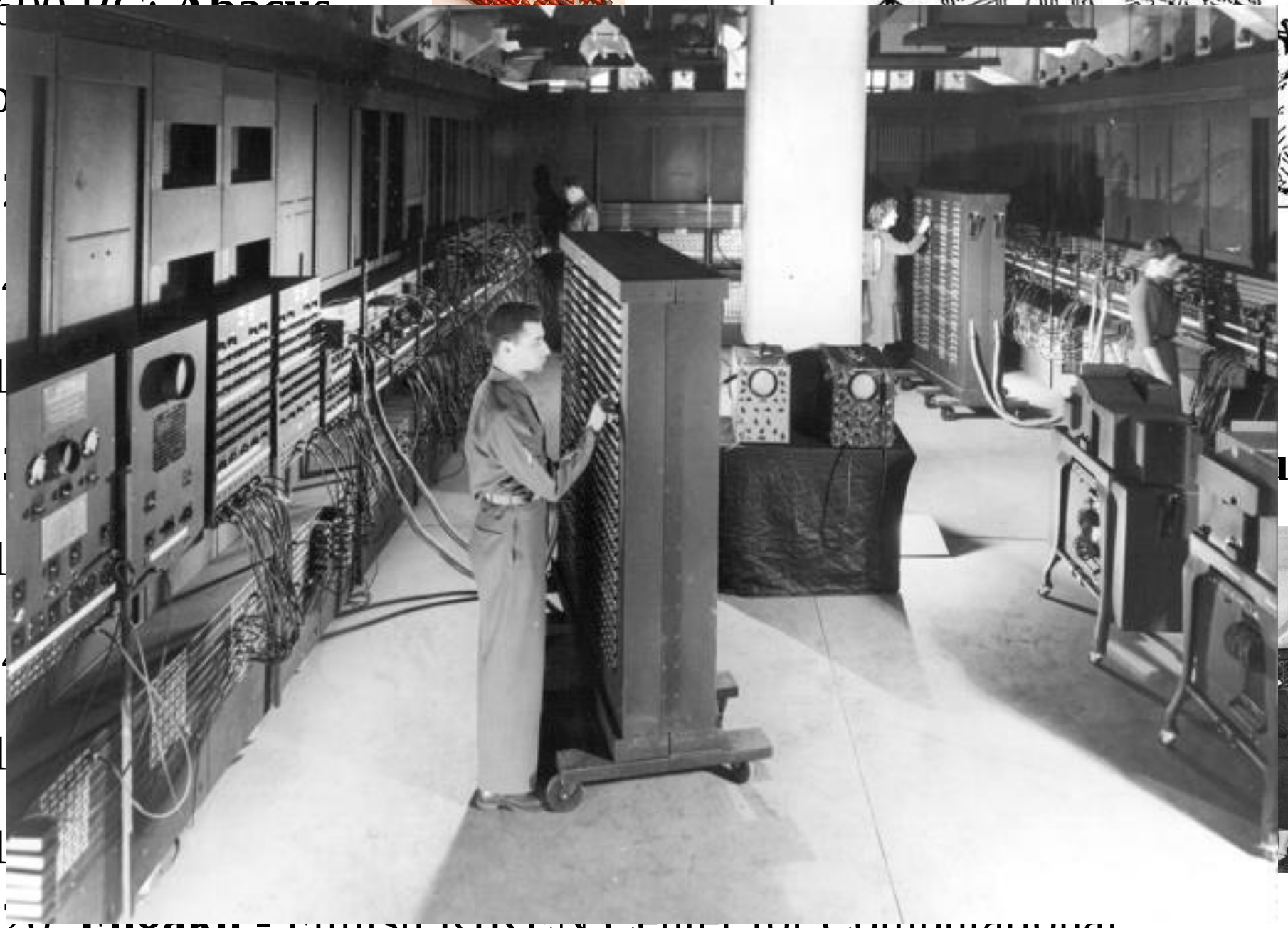
1

19

1

1

20



20: **Fugaku** - Fujitsu RIKEN Center for Computational Science, Japan: #1 in Top 500 since 06/2020.

≈ 600 BC: **Abacus**.

bef

1620

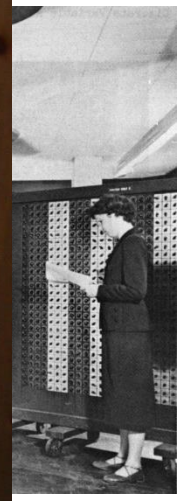
1642

175

1834 **mputer**.

188

1946

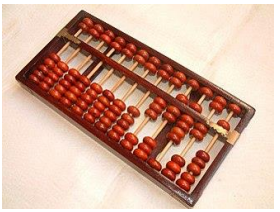


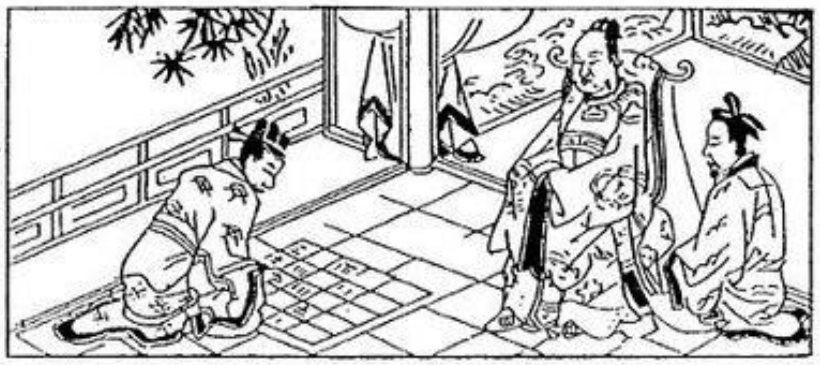
19

19


2020: **Fugaku -** Fujitsu RIKEN Center for Computational Science, Japan: #1 in Top 500 since 06/2020.

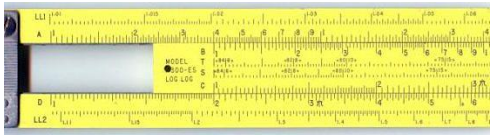
≈ 600 BC: **Abacus**.





before 179: **Gaussian elimination**.

1620-27: **Slide rule**.

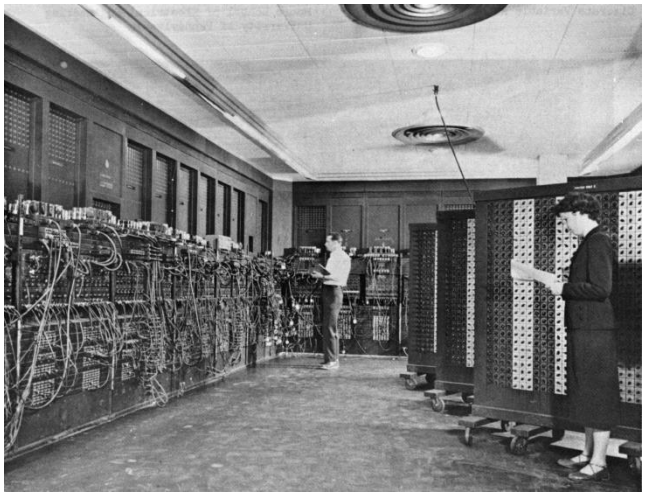


1642: **Pascal's mechanical calculator**.

1759-61: **Euler method** for ODEs.

1834: **Babbage**'s project of a **mechanical** general-purpose **computer**.

1883: **Adams-Bashforth method**.



1946: **ENIAC** – 1st electronic computer.

1947: **Crank–Nicolson** method.

1960: **Lax–Wendroff** method.

2020: **Fugaku -** Fujitsu RIKEN Center for Computational Science, Japan: #1 in Top 500 since 06/2020.

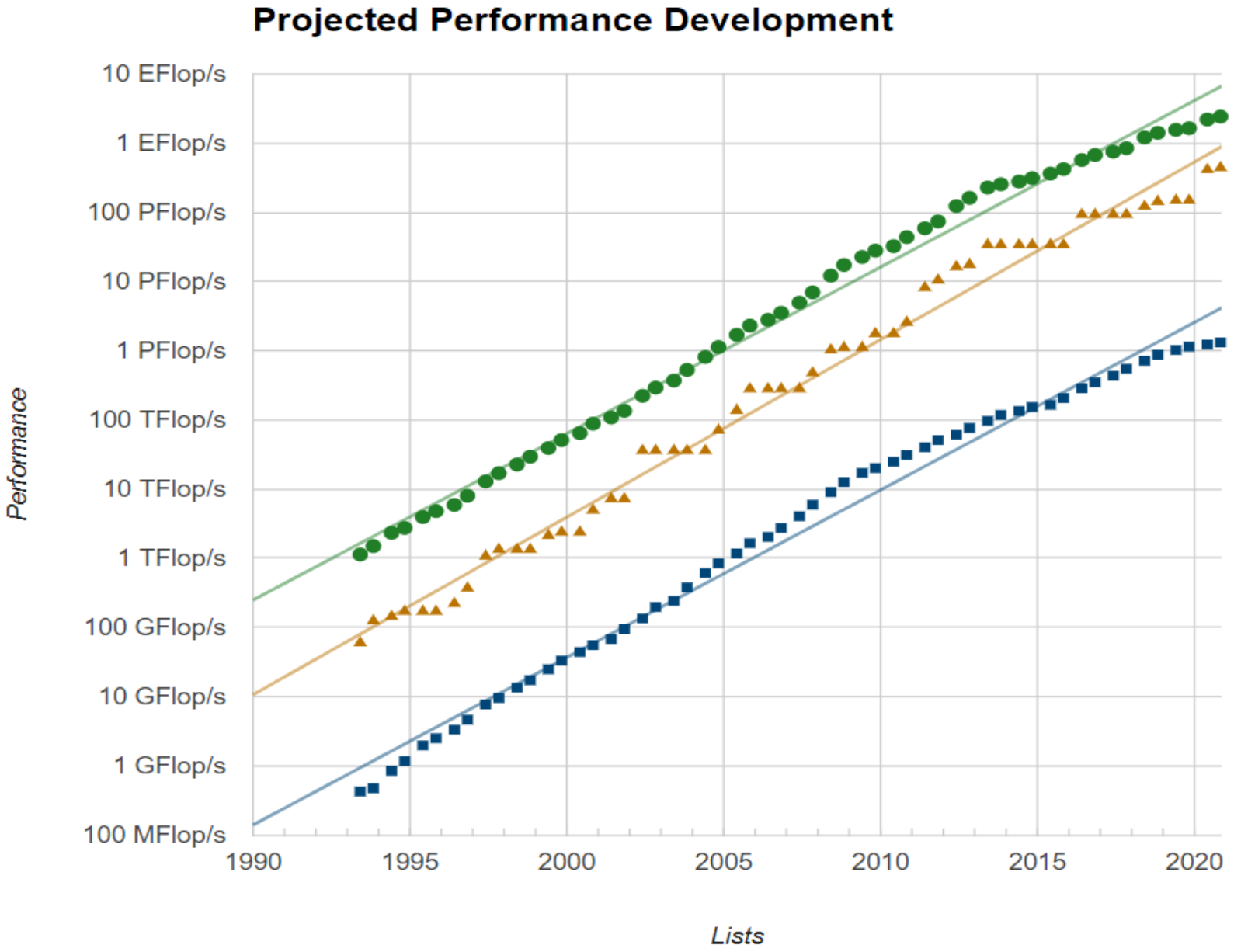
# Supercomputer Fugaku – #1 on Top 500

https://www.arm.com/blogs/blueprint/fugaku-supercomputer




https://www.r-ccs.riken.jp/en/fugaku/3d-models/

# Supercomputer performance development




https://www.top500.org/statistics/perfdevel/

# Supercomputer performance development

## TOP500 LIST - JUNE 2021

$R_{max}$ and $R_{peak}$ values are in TFlops. For more details about other fields, check the TOP500 description.

$R_{peak}$ values are calculated using the advertised clock rate of the CPU. For the efficiency of the systems you should take into account the Turbo CPU clock rate where it applies.

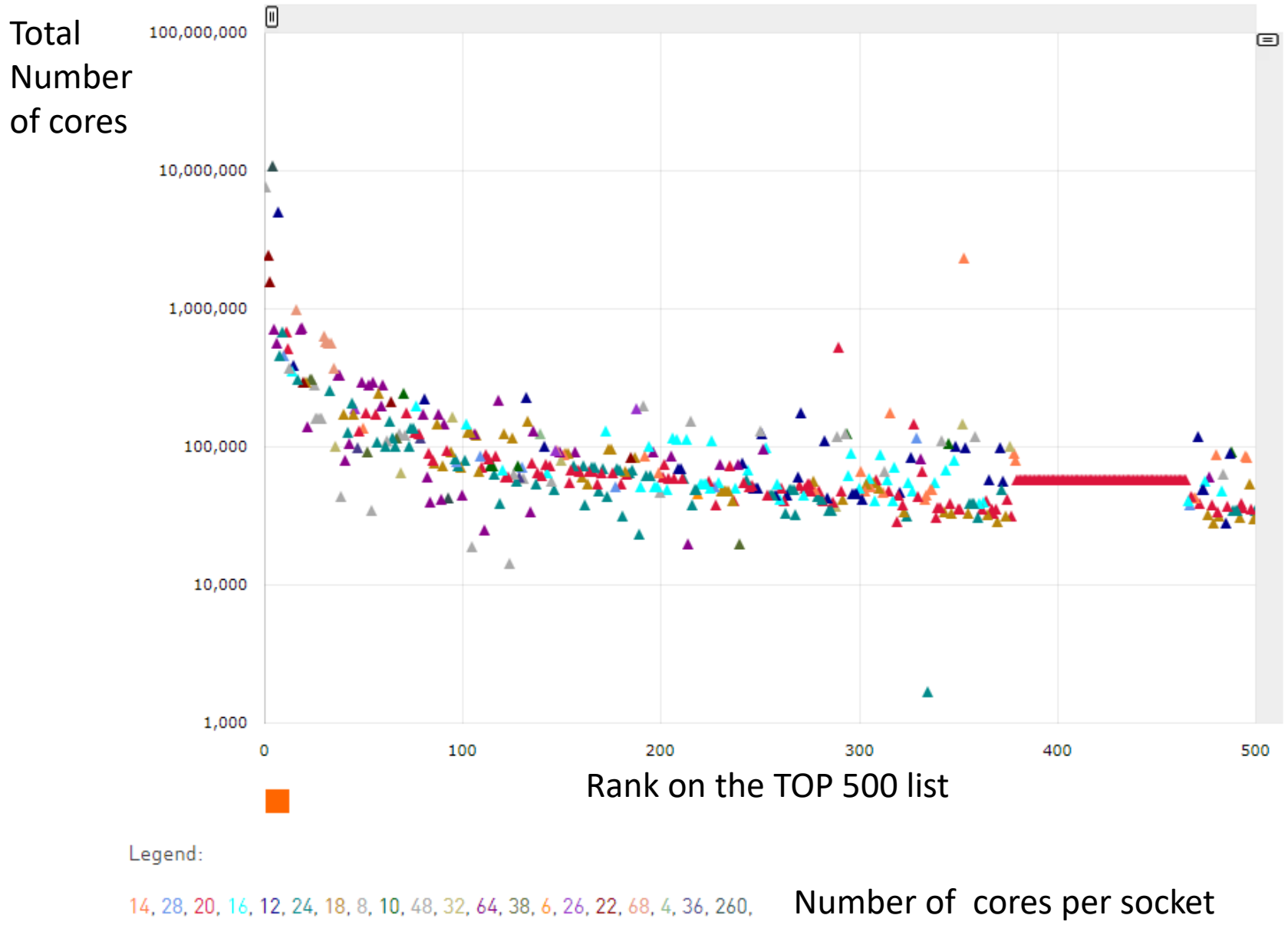
← | 1-100 | 101-200 | 201-300 | 301-400 | 401-500 | →

| Rank | System | Cores | Rmax (TFlop/s) | Rpeak (TFlop/s) | Power (kW) |
|---|---|---|---|---|---|
| 1 | **Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu<br>RIKEN Center for Computational Science<br>Japan | 7,630,848 | 442,010.0 | 537,212.0 | 29,899 |
| 2 | **Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM<br>DOE/SC/Oak Ridge National Laboratory<br>United States | 2,414,592 | 148,600.0 | 200,794.9 | 10,096 |
| 3 | **Sierra** - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM / NVIDIA / Mellanox<br>DOE/NNSA/LLNL<br>United States | 1,572,480 | 94,640.0 | 125,712.0 | 7,438 |
| 4 | **Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway, NRCPC<br>National Supercomputing Center in Wuxi<br>China | 10,649,600 | 93,014.6 | 125,435.9 | 15,371 |



https://www.top500.org/lists/top500/list/2021/06/?page=1

# Supercomputer performance development

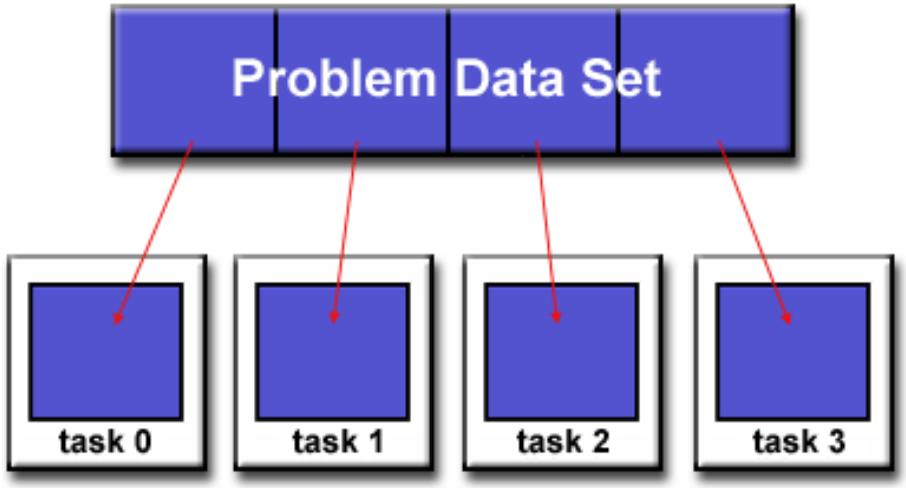


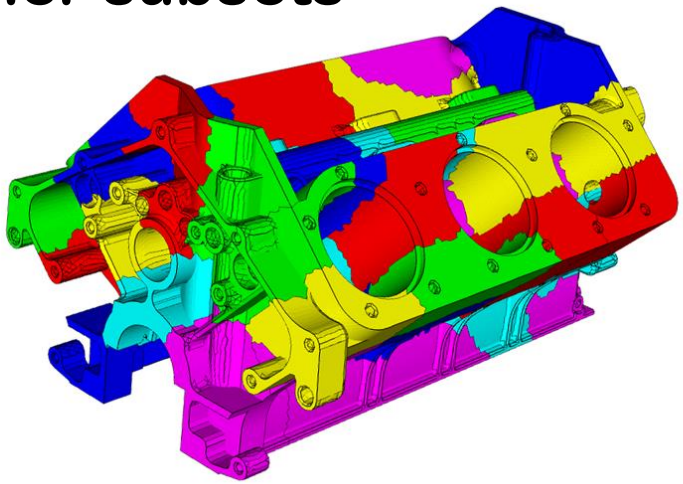



https://www.top500.org/statistics/efficiency-power-cores/

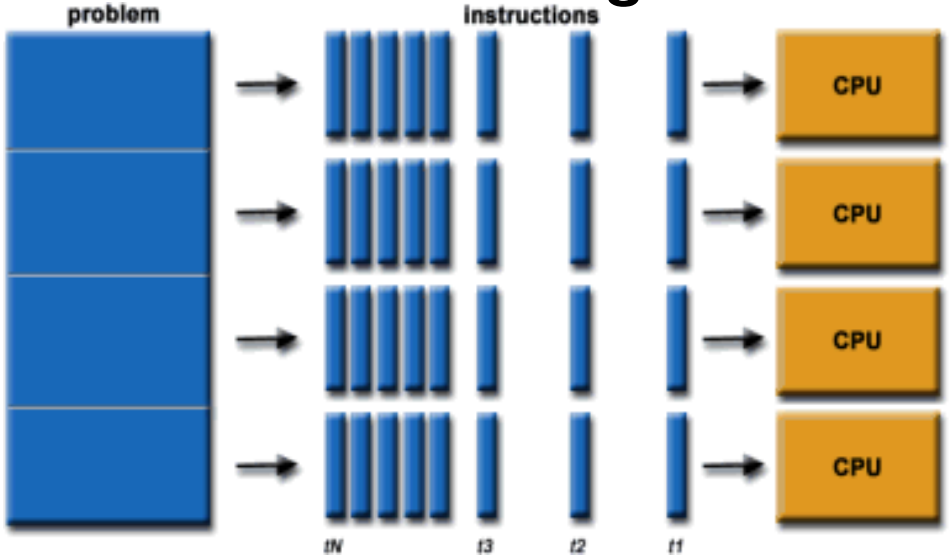
# Domain decomposition and parallel computing

## The big data set is divided into smaller subsets



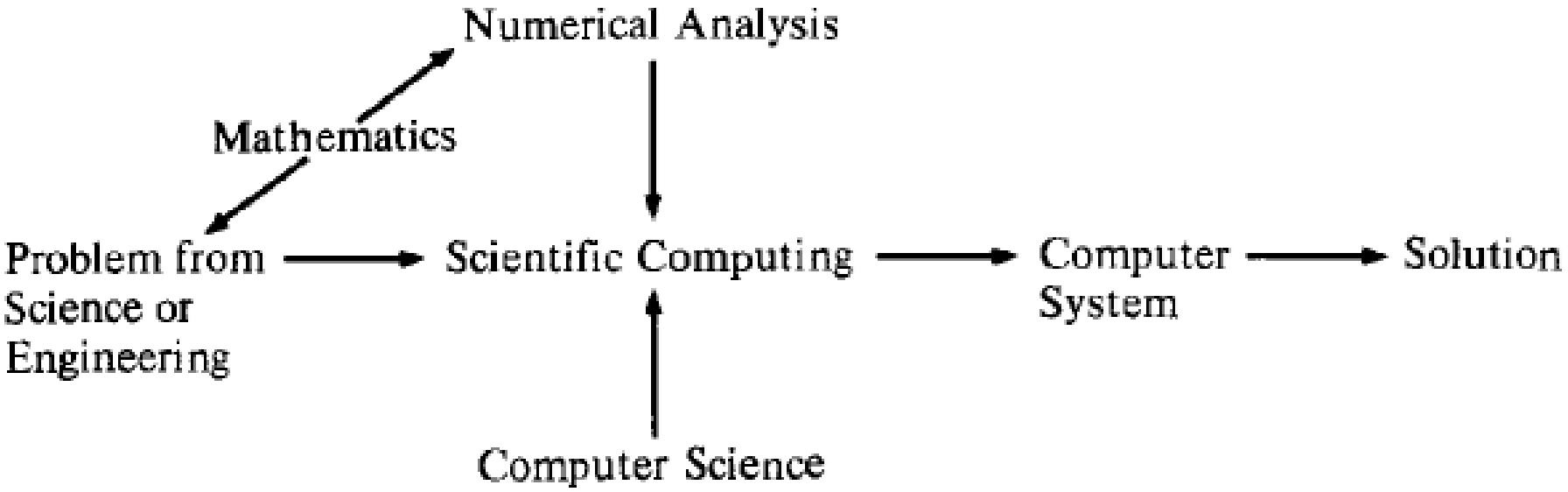
## Each subset is assigned to its "own" processor




https://computing.llnl.gov/tutorials/parallel_comp/
http://industry.it4i.cz/en/research/basic-research/feti-based-domain-decomposition-methods/

http://www.aics.riken.jp/en/k-computer/establishment-of-the-k-computer

# Scientific computing and related areas




[G. H. Golub & J. M. Ortega, Scientific Computing and Differential Equations]

# Types of approximation error

$e_m$ : **modelling error**. It comes from the fact that the **physical problem** (PP) is substituted with a **mathematical model** (MM).

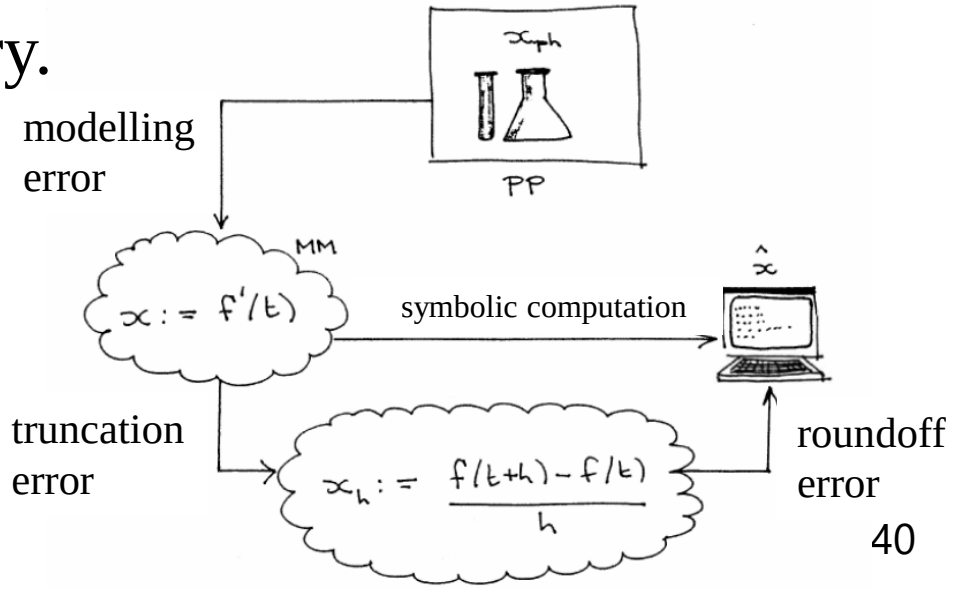
$e_t$ : **truncation error**. It is related with the fact that, in general, a computer can only perform calculations using a finite number of arithmetic operations.
Example: truncation error of the Taylor series expansion.

$e_r$ : **roundoff error**. It is present because *real numbers* are approximated by a finite set of *rational numbers* when stored in the computer memory.

Other sources of error include

errors in the input data;

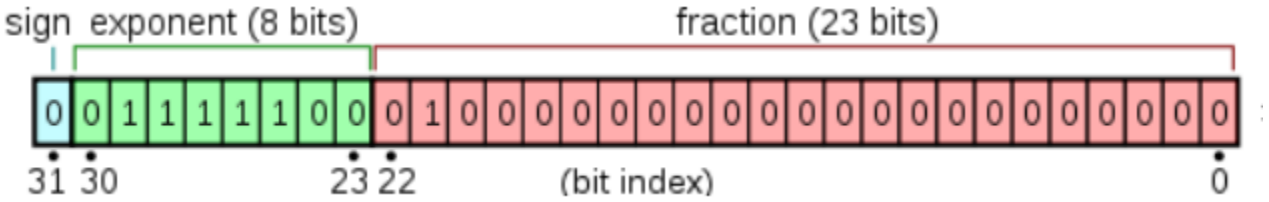
human errors.



# Round-off error (or rounding error)

Round-off error is the difference between an approximation of a *number* used in computation and its exact value. Those approximate values are typically represented on a computer using a standardized floating-point format. For example, IEEE 754 single-precision binary floating-point format has 32 explicitly stored bits:

- Sign: 1 bit
- Exponent: 8 bits
- Significand: 23 bits




$$\text{value} = (-1)^{\text{sign}} \times 2^{(E-127)} \times \left(1 + \sum_{i=1}^{23} b_{23-i} 2^{-i}\right)$$

Real numbers that cannot be exactly represented in this format are rounded.

# Truncation error

Truncation error results from ignoring all but a finite number of terms of an infinite series expansion of a *function*.

For example, the exponential function $e^x$ may be expressed as the sum of the infinite series

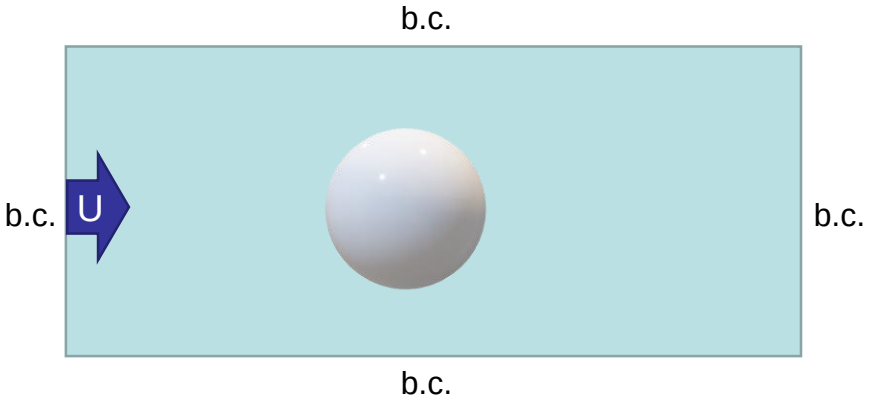
$$e^x = 1 + x + x^2/2 + x^3/6 + \cdots + x^n/n! + \textbf{truncation error}$$

Stopping the calculation after any finite value of $n$ will give an approximation to the value of $e^x$ that will be in error, but this error can be made as small as desired by making $n$ large enough.

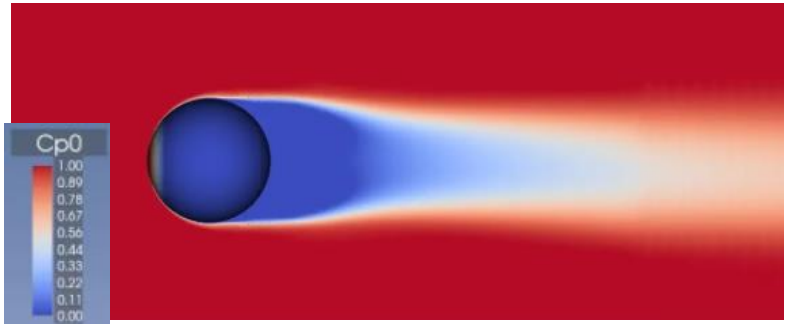
**Discretization error** is the error that occurs from the representation of the governing equations as algebraic expressions in a discrete domain of space.
By basis expansion, one can show that discretization error is also a truncation error, and the number of points or cells in the discrete domain relates with the number of terms retained in the basis expansion.

# Modelling error

Modelling errors are those due to **deliberate simplifications** or **uncertainty** in the formulation of the mathematical model.

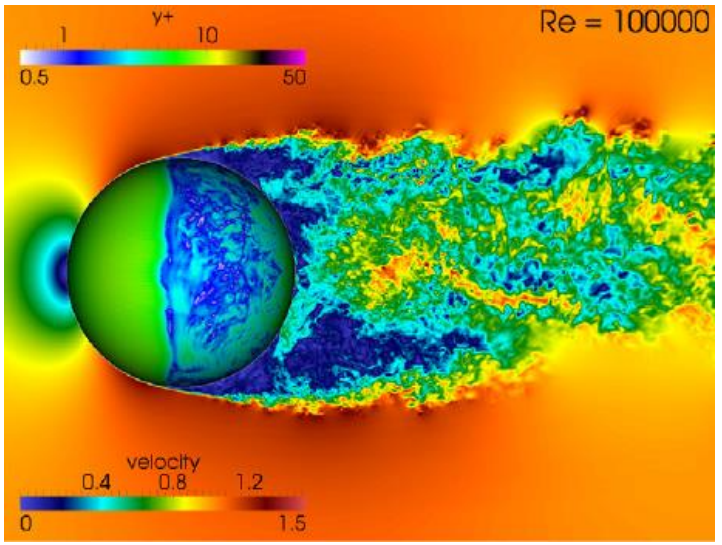



Sphere, steady RANS solver



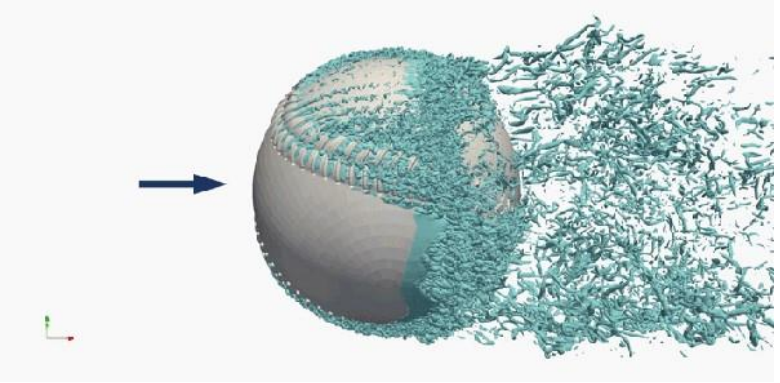

https://www.idealsimulations.com/simworks-tutorials/drag-coefficient-of-a-sphere-cfd-simulation/

Sphere, unsteady lattice Boltzmann solver



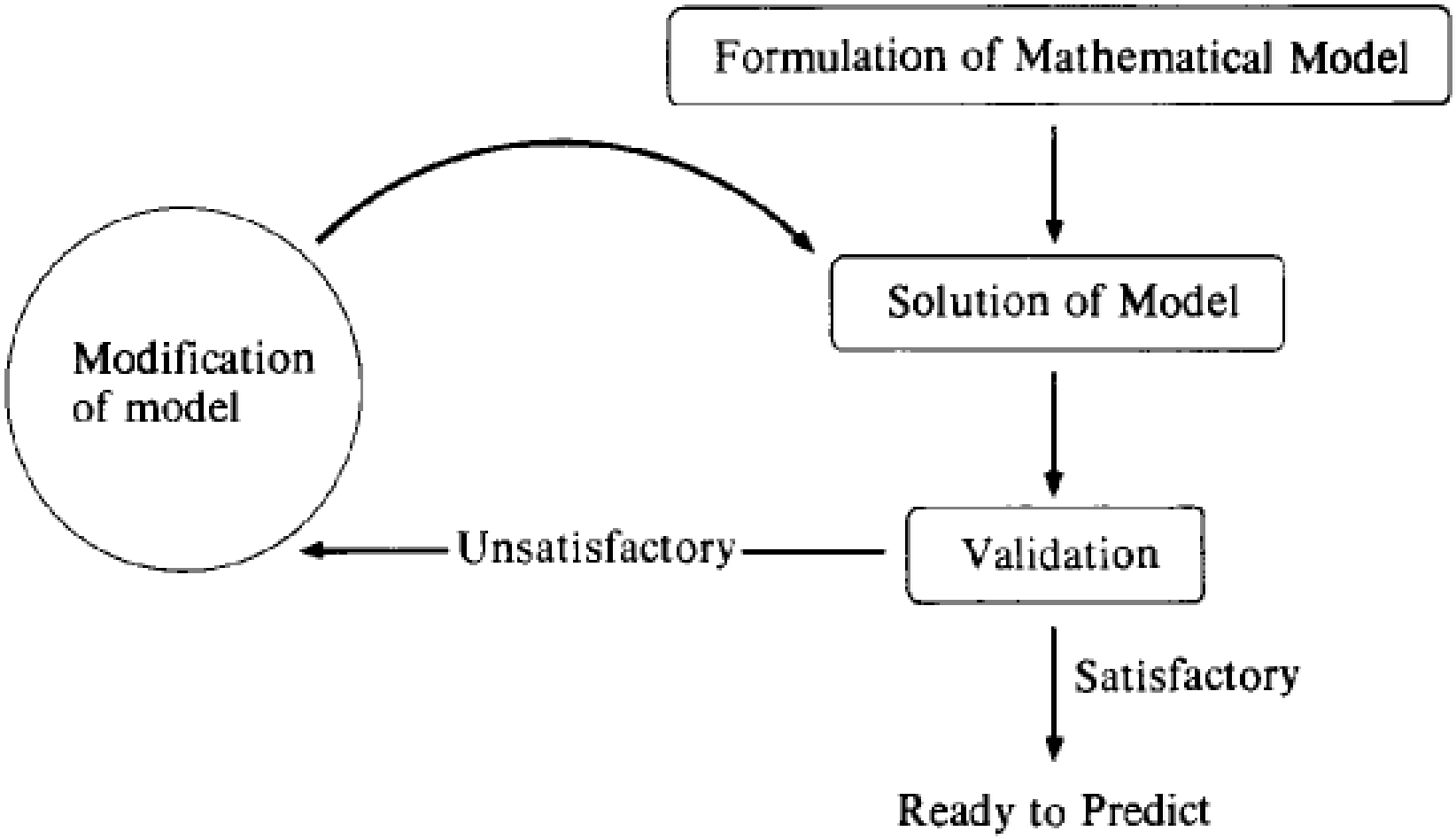

M. Geier et al. / Journal of Computational Physics 348 (2017) 889–898

Ball with seams, unsteady lattice Boltzmann solver



The negative Magnus effect on a two seam ball.
(Takayuki Aoki, Tokyo Institute of Technology)
https://sj.jst.go.jp/news/202106/n0609-02k.html

# Validation




[G. H. Golub & J. M. Ortega, Scientific Computing and Differential Equations]

# Relative and absolute error

The *absolute* error $e^{abs}$ is the absolute value (more generally, the norm) of the difference between the **approximate value**, $\hat{x}$, and the **reference value** (exact value) of a variable $x$.

$$e^{abs} := \left|\hat{x} - x\right|$$

The *relative* error $e^{rel}$ is defined as

$$e^{rel} := \frac{\left|\hat{x} - x\right|}{\left|x\right|}$$

# Convergence

Numerical calculations generally involve a discretization parameter (which will be designated by $h$).

If $\hat{x} \to x$ as $h \to 0$, the calculation *converges* and we can write $e^{abs} \to 0$.

In addition if, for any sufficiently small $h$, we have

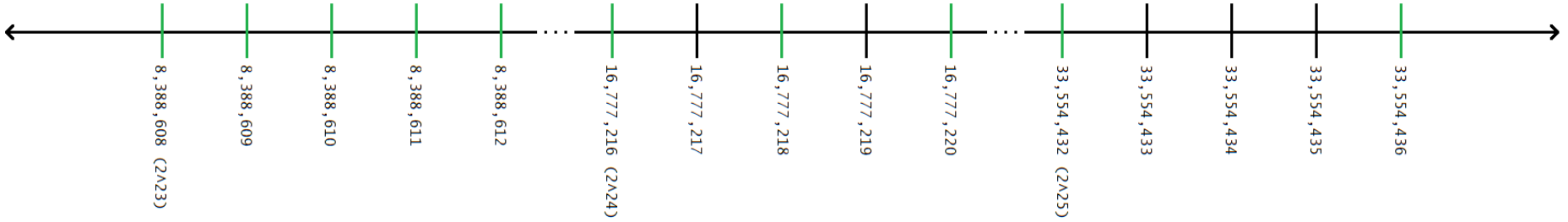
$$e^{abs} \leq Ch^p$$

where $C$ is a constant that does not depend on $h$ and where $p$ is a positive number, we say that the convergence is of order $p$.

# Computer representation of real numbers

Computers have only a finite number of digits to represent all real numbers. As a result,

- All representable numbers are rational and there are gaps between them.




- In general, a representation error is introduced: rounding error.
- There exist a minimum and a maximum representable positive numbers.

# Computer representation of real numbers

| | | |
|---|---|---|
| Level 1 | $\{-\infty \ldots 0 \ldots +\infty\}$ | Extended real numbers. |
| many-to-one ↓ | *rounding* | ↑ projection (except for NaN) |
| Level 2 | $\{-\infty \ldots -0\} \cup \{+0 \ldots +\infty\} \cup$ **NaN** | Floating-point data—an algebraically closed system. |
| one-to-many ↓ | *representation specification* | ↑ many-to-one |
| Level 3 | (***sign***, ***exponent***, ***significand***) $\cup \{-\infty, +\infty\} \cup$ **qNaN** $\cup$ **sNaN** | Representations of floating-point data. |
| one-to-many ↓ | *encoding for representations of floating-point data* | ↑ many-to-one |
| Level 4 | **0111000…** | Bit strings. |

# Floating-point numbers

Let $F$ be the set of machine-representable numbers.

Under the IEEE 754 convention, $x \in F$ is represented in double precision using 64 binary digits, i.e. 8 bytes, as *normal* numbers

$$x := (-1)^{\sigma}\, 2^{e-1023}\,(1+f)$$
$$= (-1)^{d_1}\, 2^{(d_2 \ldots d_{12})-1023}\,(1+(0.d_{13}\ldots d_{64})_2)$$

where $d_i=0$ or $d_i=1$, $i=1,\ldots,64$

- $\sigma = d_1$: the sign of the number ("+" if $\sigma=0$, "-" if $\sigma=1$).
- $e = (d_2\ldots d_{12})_2$: the exponent
- $1+f = (1,d_{13}\ldots d_{64})_2$

# Special values

• The largest machine-representable real number (called *realmax* in Matlab) is

$$realmax = 2^{2046-1023} \times (2 - 2^{-52}) \approx 1.7977 \times 10^{308}$$

• Values $x \notin [-realmax, realmax]$ correspond to *Inf* or *-Inf*

*Inf* is represented as

$$\sigma = 0,\ e = (11111111111)_2,\ 1 + f = (1.00...0)$$

*-Inf* is represented as

$$\sigma = 1,\ e = (11111111111)_2,\ 1 + f = (1.00...0)$$

• Not a number: *NaN*

$$\sigma = 0 \text{ or } \sigma = 1,\ e = (11111111111)_2,\ f \neq 0$$

# Subnormal numbers

- The smallest normal number is

$$realmin = 2^{1-1023} \times 1 \approx 2.2251 \times 10^{-308}$$

- The denormalized representation of

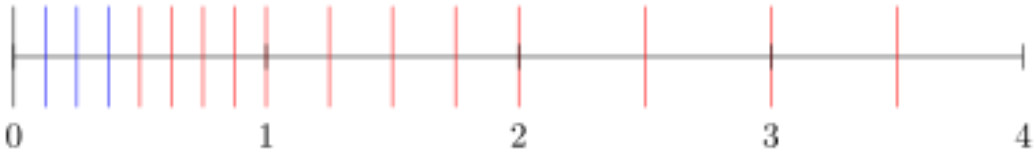
$x \in ]-realmin, realmin[$ is

$x := (-1)^{\sigma}\, 2^{-1022}\, (0+f)$

$\sigma = d_1$: the sign of the number ("+" if $\sigma$=0, "-" if $\sigma$=1).

$e = (0\ldots0)_2$: the exponent

$0+f = (0, d_{13}\ldots d_{64})_2$




- There are two zeros: +0 and -0

# Rounding

• Let $x = \pm(1.b_1 b_2 \ldots b_{52} b_{53} b_{54} \ldots)_2 2^E$

• Different round-off rules exist. For example, *round to zero* rule results in the following floating-point number

$$fl(x) = \pm(1.b_1 b_2 \ldots b_{n-1} b_n)_2 2^E$$

• *Round half to even* is the default rounding mode used in IEEE 754:

- rounds to the nearest value;
- if the number falls midway, it is rounded to the nearest value with an even least significant digit.

# Rounding

• Any deterministic rule for rounding have non-zero statistical bias. For example, if we implement the algorithm

$$y_0=1,\ y_k=(y_{k-1}/k)k, \quad k=1,2,\ldots$$

the residual error $\varepsilon_k=|y_k-1|$ will increase with the number of iteration $k$.

• The *round half to even* rule has the advantage that repeated rounded addition or subtraction of independent numbers will give a result with an error that tends to grow in proportion to the square root of the number of operations rather than linearly.

# Relative error of rounding

If the significand of the normalized floating point representation is in base $\beta$ ($\beta$ even, e.g., $\beta$=2) and contains $n$ decimal places, then any machine-representable number can be represented with a relative error that does not exceed $\beta^{-n}$ if we round to zero or $\beta^{-n}/2$ if we round to half to even.

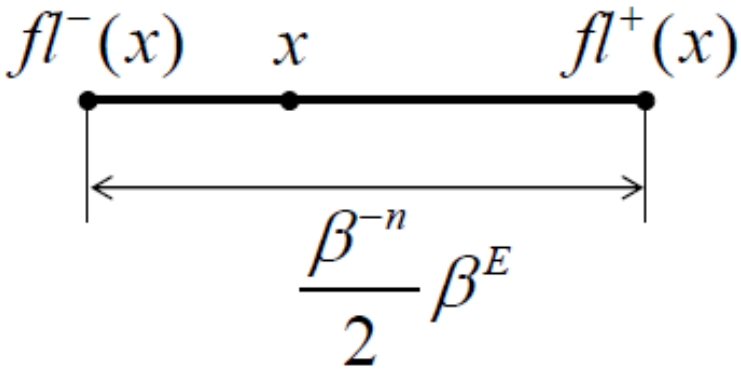
# Relative error of rounding

*Proof for the case of rounding to nearest.*

$$x = \pm(b_0.b_1b_2\ldots b_{n-1}b_nb_{n+1}\ldots)_\beta \times \beta^E$$

$$fl^+(x) = \pm(b_0.b_1b_2\ldots b_n)_\beta \times \beta^E$$

$$fl^-(x) = \pm(b_0.b_1b_2\ldots b_n + 1)_\beta \times \beta^E$$



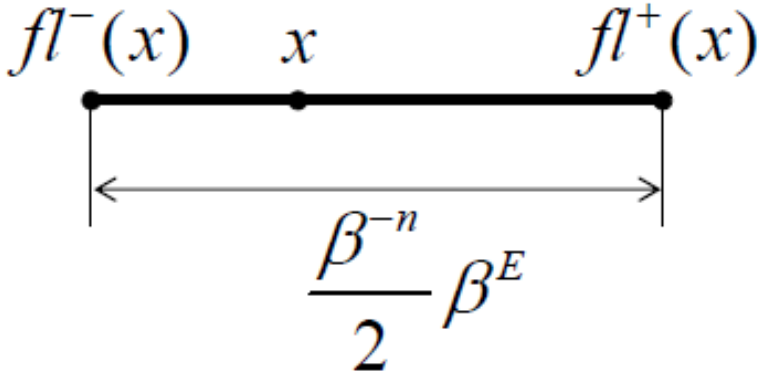

If $b_{n+1} < \frac{\beta}{2}$ : $|x - fl^-(x)| < \frac{\beta^{-n}}{2}\beta^E < |fl^+(x) - x|$, $\quad fl(x) = fl^-(x)$

If $b_{n+1} \geq \frac{\beta}{2}$ : $|fl^+(x) - x| \leq \frac{\beta^{-n}}{2}\beta^E \leq |x - fl^-(x)|$, $\quad fl(x) = fl^+(x)$

# Relative error of rounding

Then,

$$|\, fl(x) - x \,| \leq \frac{\beta^{-n}}{2}\beta^{E}$$




We know that $|\, x \,| \geq \beta^{E}$ , because $b_0 \geq 1$

$$\frac{|\, fl(x) - x \,|}{|\, x \,|} \leq \frac{\beta^{-n}}{2}$$

If $\beta = 2$ , $n = 52$ (double precision)

the relative error is no greater than $2^{-53} \approx 1.1102 \times 10^{-16}$

# Machine precision

*Machine precision* (*machine epsilon*) is the largest relative error that can be introduced by representing a real number on a computer using rounding. The machine precision is therefore bounded by $\beta^{-n}/2$.

$$|fl(x) - x| \leq \varepsilon_{machine} |x|$$

$$x \in [-realmax, -realmin] \cup [realmin, realmax]$$

$\varepsilon_{machine} = 2^{-53}$ if $\beta = 2$, $n = 52$ (binary, double precision)

# Machine precision

Warning: the *eps* command in Matlab gives the distance from 1.0 to the next real number greater than 1. It is equal to $2\varepsilon_{machine}$.

Let $x$ be a floating-point number,

$$x = 2^E(1+f)$$

Its nearest larger floating-point number is

$$y = x + 2^E \times 2^{-52} = x + 2^{E+1}\varepsilon_{machine}$$

i.e., neighboring floating point numbers between 1 and 2 are separated by $2\varepsilon_{machine}$, between 2 and 4 – by $4\varepsilon_{machine}$, …, between $2^{1023}$ and *realmax* – by $2^{1024}\varepsilon_{machine}$

# Matlab code to determine machine precision

```
>> macheps = 1;
while (1 + macheps) - 1 > 0
    macheps = macheps / 2;
end;
macheps

macheps =

   1.1102e-016

>>
```

# Floating-point arithmetics

$$x \oplus y = fl(fl(x) + fl(y)),$$

$$x \ominus y = fl(fl(x) - fl(y)),$$

$$x \otimes y = fl(fl(x) \times fl(y)),$$

$$x \odot y = fl(fl(x) / fl(y)).$$

# Catastrophic cancellation

Suppose that two numbers $a$ and $b$ are equal to within their last digit. Then the difference $c=a-b$ will have only one significant digit of accuracy. Future calculations with $c$ will then usually limit the final result to one correct digit.

Catastrophic cancellation may happen even if the difference is computed exactly, as in the example above—it is not a property of any particular kind of arithmetic like floating-point arithmetic; rather, it is inherent to subtraction, when the inputs are approximations themselves.

# Computational efficiency

Algorithmic complexity: number of arithmetic operations

Example:

*Cramer's rule* for solving linear systems requires calculating determinants. This takes about $n!$ operations.

$$x_1 + 3x_2 = 5$$
$$2x_1 + 2x_2 = 6$$

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 2 \end{bmatrix} \quad A_1 = \begin{bmatrix} 5 & 3 \\ 6 & 2 \end{bmatrix} \quad A_2 = \begin{bmatrix} 1 & 5 \\ 2 & 6 \end{bmatrix}$$

$$|A| = -4 \qquad |A_1| = -8$$

$$x_1 = 2 \qquad x_2 = \frac{|A_2|}{|A|}$$

$$n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

The number of operations in *Gauss elimination* is ~ $n^3$

Parallel efficiency

Amdahl's law:

$$S_{\text{latency}}(s) = \frac{1}{(1-p) + \frac{p}{s}}$$