Computational Physics

Part III: Differential Equations
Part IV: Computer Simulations

UCAS UNDERGRADUATE COURSE

Jure Dobnikar, May 2021

Literature:

R.W. Hamming: ***Numerical Methods for Scientists and Engineers***, Dover Publications, 1973

T. Pang: ***An Introduction to Computational Physics***, Cambridge University Press, 1997

D. Frenkel, B. Smit: ***Understanding Molecular Simulation,*** Academic Press, 2002

OUTLINE OF THE LECTURES

May

June

1. Comments on Computation and Cimputational methods; Ordinary **Differential Equations** (Part 1: Euler Methods)
2. **Ordinary Differential Equations** (Part 2: Runge-Kutta); Examples; Introduction of the project assignment
3. **Partial Differential Equations**
4. Partial Differential Equations: Examples
5. Random Numbers
6. **Computer Simulations**: Statistical Physiscs Background
7. **Basic Molecular Dynamics**
8. **Metropolis Monte Carlo**
9. Monte Carlo moves
10. Interactions & Boundary conditions
11. Computer Experiments
12. Free Energy estimation
13. Correlation functions and transport coefficients
14. Advanced MC topics
15. Discussion session

Project submission deadline: end of June (?)
Exam: end of July (?)
(dates will be determined and fixed soon)

INTRODUCTION TO COMPUTATION

In a modern society, we need to deal with a lot more computations daily. Almost every event in science or technology requires quantification of the data involved. For example, before a jet aircraft can be actually manufactured, extensive computer simulations of different flight conditions are now performed to check whether there is any design deficiency. This is not only necessary economically but may help avoid loss of life. A related use of the computer is in the reconstruction of an unexplained flight accident. This is extremely important in preventing the same accident from happening again. It is fair to say that computing has become part of our lives, permanently.

All these computationally intensive research projects accomplished in the 1950s showed that computation was no longer just a supporting tool for scientific research but rather an actual means of probing scientific problems and predicting new scientific phenomena. A new branch of science, *computational science*, was born. Since then, the field of scientific computing has developed and grown rapidly.

The concept of an all-purpose, automatic, programmable computing machine was introduced by British mathematician and astronomer Charles Babbage in the early nineteenth century. Babbage proposed the construction of a computing machine, called an *analytical engine*, which could be programmed to perform any type of computation. Unfortunately, the technology at that time was not advanced enough to provide Babbage with the necessary machinery to actually build the analytical engine. In the late nineteenth century, Spanish engineer Leonardo Torres y Quevedo showed that with the availability of electromechanical technology, which had just been developed, one might be able to construct the machine conceived earlier by Babbage. However, he could not actually build the whole machine either, due to lack of funds. American engineer and inventor Herman Hollerith built the very first electromechanical counting machine with punch cards, which was used to sort the populations in the 1890 American census. Hollerith used the profit obtained from selling this machine to set up a company, the Tabulating Machine Company, which was the predecessor of the International Business Machines Corporation (IBM). These developments continued in the early twentieth century. In the 1930s, scientists and engineers at IBM built the first difference tabulator and researchers at Bell Laboratories built the first relay calculator, which were among the very first electromechanical calculators built during that time.

The computational power of new computers has been increasing exponentially. To be specific, the computing power of a single computer system has doubled almost every two years in the last fifty years. Computers with transistors replaced those with vacuum tubes in the late 1950s and early 1960s, and computers with very-large-scale integrated circuits were built in the 1970s. Microprocessors also became available in the 1970s. In the mid-1970s, vector processor supercomputers were built to set the stage for supercomputing.

In the 1980s, microprocessor-based personal computers and workstations appeared. Now they have penetrated into all aspects of our lives, as well as all scientific disciplines, because of their affordability and low maintenance cost. With technological breakthroughs in the RISC (Reduced Instruction Set Computer) architecture, cache memory, and multiple instruction units, the speed of each microprocessor is now faster than that of the first generation of supercomputers. In the last few years, these fast microprocessors have been combined to form parallel or distributed computers, which can easily deliver a computing power of a few tens of gigaflops (10^9 floating-point operations per second).

Teraflop (10^{12} floating-point operations per second) computers are now emerging. With the availability of teraflop computers, scientists can start unfolding the mysteries of the grand challenges, such as the dynamics of the global environment; the mechanism of DNA sequencing; computer design of drugs deadly to viruses; and computer simulation of future electronic materials, structures, and devices.

The real beginning of the computer era started with the advent of electronic digital computers. John Vincent Atanasoff, a theoretical physicist at Iowa State University at Ames, invented the electronic digital computer between 1937 and 1939. The history regarding Atanasoff's accomplishment is described in Mackintosh (1987), Burks and Burks (1988), and Mollenhoff (1988). Atanasoff introduced vacuum tubes (instead of electromechanical devices used earlier by other people) as basic elements, a separated memory unit, and a scheme to keep the memory updated in his computer. With the assistance of Clifford E. Berry, a graduate assistant, Atanasoff built the first electronic computer in 1939. Most computer history books have cited ENIAC (Electronic Numerical Integrator and Computer), built by John W. Mauchly and J. Presper Eckert at the Moore School of the University of Pennsylvania with their colleagues in 1945, as the first electronic computer. ENIAC could complete about 5,000 additions or 400 multiplications in one second. Some very impressive scientific computations were performed on ENIAC as soon as it appeared, including the study of nuclear fission with the liquid drop model by Metropolis and Frankel (1947). In the early 1950s, scientists at Los Alamos built another electronic digital computer, called MANIAC I (Mathematical Analyzer, Numerator, Integrator, and Computer), which was very similar to ENIAC. Many important numerical studies, including Monte Carlo simulations of classical liquids (Metropolis et al., 1953), were completed on MANIAC I.

DESIGN PRINCIPLES FOR COMPUTATION

The purpose of computation is INSIGHT not just numbers!
Computing is intimately bound to the source of the physical problem and to the use that is going to be made of the answers. The choice of algorithms should be such that the answers are obtained in an optimal way.

Study families of problems & relate one family to another.
Avoid isolated formulas and algorithms.

Numerical analysis *vs* **Computational method → UNDERSTAND the problem**

ROUND-OFF ERROR

Finite nature of the computing machine,
e.g. $\frac{1}{3} \rightarrow 0.33333333$
Loss of precision when two similar numbers are subtracted
WE NEED TO CONTROL (PREDICT) AND, EVEN BETTER, AVOID THIS SITUATION!

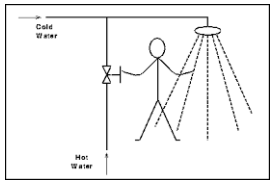
TRUNCATION ERROR

Finite speed of the computing machine,
Error expressions → predict the accuracy of the result

FEEDBACK

Loops: result of one stage are fed as in put to the next cycle (very powerful for solving problems!)

Stability of the feedback loop (not all methods work!)



Example: taking a shower (gentle or strong reactions…)

Stability of the feedback needs to be studied before computation!!

Computer (numerical) algorithm: a complete set of logical steps for a specific physical problem

Let us here use a very simple and familiar example in physics to illustrate how a typical numerical algorithm is constructed. Assume that a particle of mass m is confined to move along the x-axis under a force $f(x)$. If we describe its motion through Newton's equation, we have

$$f = ma, \tag{1.3}$$

where a is the acceleration of the particle. If we divide the time into small equal intervals τ, we know from introductory physics that the average velocity during the time interval $[t_n, t_{n+1}]$ is

$$v_n = \frac{x_{n+1} - x_n}{\tau}, \tag{1.4}$$

whereas the average acceleration in the same time interval is

$$a_n = \frac{v_{n+1} - v_n}{\tau}. \tag{1.5}$$

The simplest algorithm for finding the position and the velocity of the particle at time $t_{n+1} = (n+1)\tau$ from the corresponding quantities at time $t_n = n\tau$ is obtained after combining Eqs. (1.3), (1.4), and (1.5), and we have

$$x_{n+1} = x_n + \tau v_n, \tag{1.6}$$

$$v_{n+1} = v_n + \frac{\tau}{m} f_n, \tag{1.7}$$

where $f_n = f(x_n)$. If the initial position and velocity of the particle are given and the corresponding quantities at some later time are sought (the initial-value problem), we can obtain them recursively from the algorithm given in Eqs. (1.6) and (1.7). This algorithm is commonly known as the Euler method for the initial-value problem. This simple example illustrates how most algorithms are constructed. First, physical equations are transformed into discrete forms, that is, difference equations. Then the desired physical quantities or solutions of the equations at different variable points are expressed in a recursive manner; that is, the quantities at a later point are expressed in terms of the quantities at earlier points. In the above example, the position and velocity of the particle at t_{n+1} are given by the position and velocity at t_n, provided that the force at any position is explicitly given.

Later we will see that Euler method is not stable and it is not recommended to use it for the initial value problem. Here it only serves as a simple example of building up a numerical algorithm.

ESSENTIAL TOOLS FOR COMPUTING

- **FLOATING POINT REPRESENTATION** of numbers on computers

- **EVALUATING FUNCTIONS**

  Example: **finding the zeros of a function**, $f(x) = 0$ (VERY COMMON PROBLEM)

  - near zero, $f(x) \approx 0$, there is almost exact cancellation of positive and negative values
  - accurate function evaluation with control of round-off error is essential
  - *Relative error* = $\left|\frac{calculated\ value - true\ value}{true\ value}\right|$
  - number of zeros (known in case of polynomials…)

- **FINITE DIFFERENCE CALCULUS**

  - difference (derivatives)
  - summation (integration)
  - difference equations (differential equations)

COMPUTATION PRINCIPLES: SUMMARY

Exapmple: **Many-body gravitational problem**

We need computational methods to understand and predict planetary motion!

$$m_i \frac{d\boldsymbol{v}_i}{dt} = \boldsymbol{F}_i \quad \rightarrow \quad \frac{\Delta \boldsymbol{v}_i}{\Delta t} = \frac{\boldsymbol{F_i}}{m_i}$$

Evaluating forces

$$\boldsymbol{v}_i(t + \Delta t) = \boldsymbol{v}_i(t) + \frac{\boldsymbol{F_i}}{m_i}\Delta t$$
$$\boldsymbol{r}_i(t + \Delta t) = \boldsymbol{r}_i(t) + \boldsymbol{v}_i(t)\Delta t$$

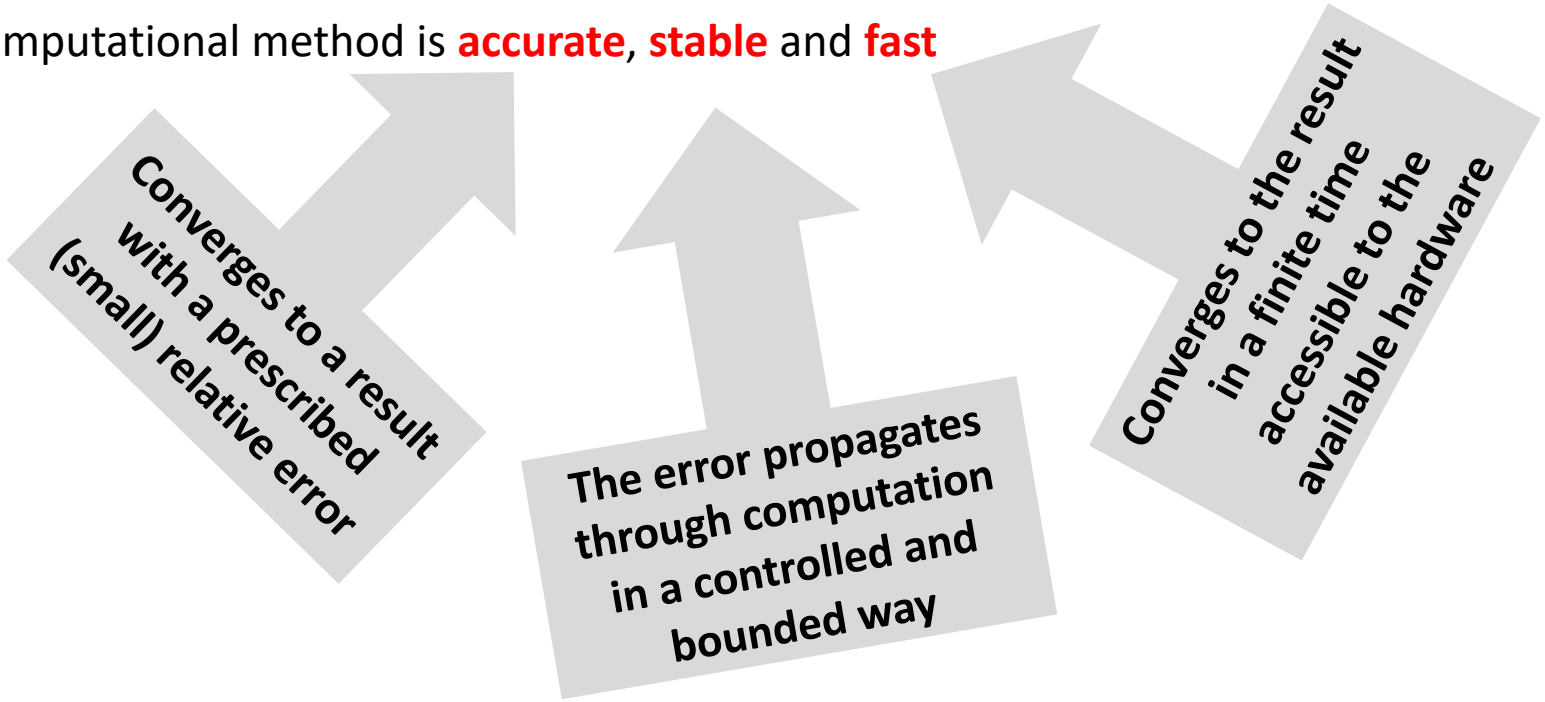
Propagating Equations of Motion (one way to do it…)

ESSENTIAL ALGEBRAIC TOOLS

- Evaluation of a function
- Numerical derivation
- Numerical integration
- Solving differential equations

COMPUTATION PRINCIPLES: SUMMARY

- We need to use computational methods to solve most physical or engineering problems
- In order to understand the problem, we need to analyze a family of problems and interdependencies of their solutions
- A good computational method is **accurate**, **stable** and **fast**



- Different *mathematically equivalent* expressions propagate the errors in a different way
- **Method design is important**: *time spent thinking about the method before computing is not wasted (up to a limit)*

COMPUTATIONAL APPROACHES TO SOLVING DIFFERENTIAL EQUATIONS

We often state the laws of nature in the form of differential equations. Two quick examples:

- **Newton's equation** of motion:

  $$\boldsymbol{F} = m\frac{d^2\boldsymbol{r}}{dt^2}$$ (second-rder differential equation)

- **Growth of bacterial colony** (y is the number of bacteria) with good food supply:

  $$\frac{dy}{dt} = \beta y(t)$$ (first-order differential equation)

COMPUTATIONAL APPROACHES TO SOLVING DIFFERENTIAL EQUATIONS

We often state the laws of nature in the form of differential equations. Two quick examples:

- Newton's equation of motion:

  $$\boldsymbol{F} = m\frac{d^2\boldsymbol{r}}{dt^2}$$ (second-rder differential equation)

- Growth of bacterial colony (y is the number of bacteria) with good food supply:

  $$\frac{dy}{dt} = \beta y(t)$$ (first-order differential equation)

Some simple models can be solved in a closed analytical form (textbooks often use those!),
***but** majority of ODE cannot be solved analytically and we need methods to find approximate solutions!*

Lecture plan:

Part 1: Methods for **Ordinary Differential Equations (ODE):** one independent variable

Part 2: Methods for **Partial Differential Equations (PDE)**

ORDINARY DIFFERENTIAL EQUATIONS

Consider equation $y' = x^2 - y^2$. What do we mean by **solution**? A curve $y(x)$ whose slope is locally given by $y'(x)$.
There is not a single curve, but a **family of curves**, *i.e.* (in principle) a curve from every point (x_0, y_0)!

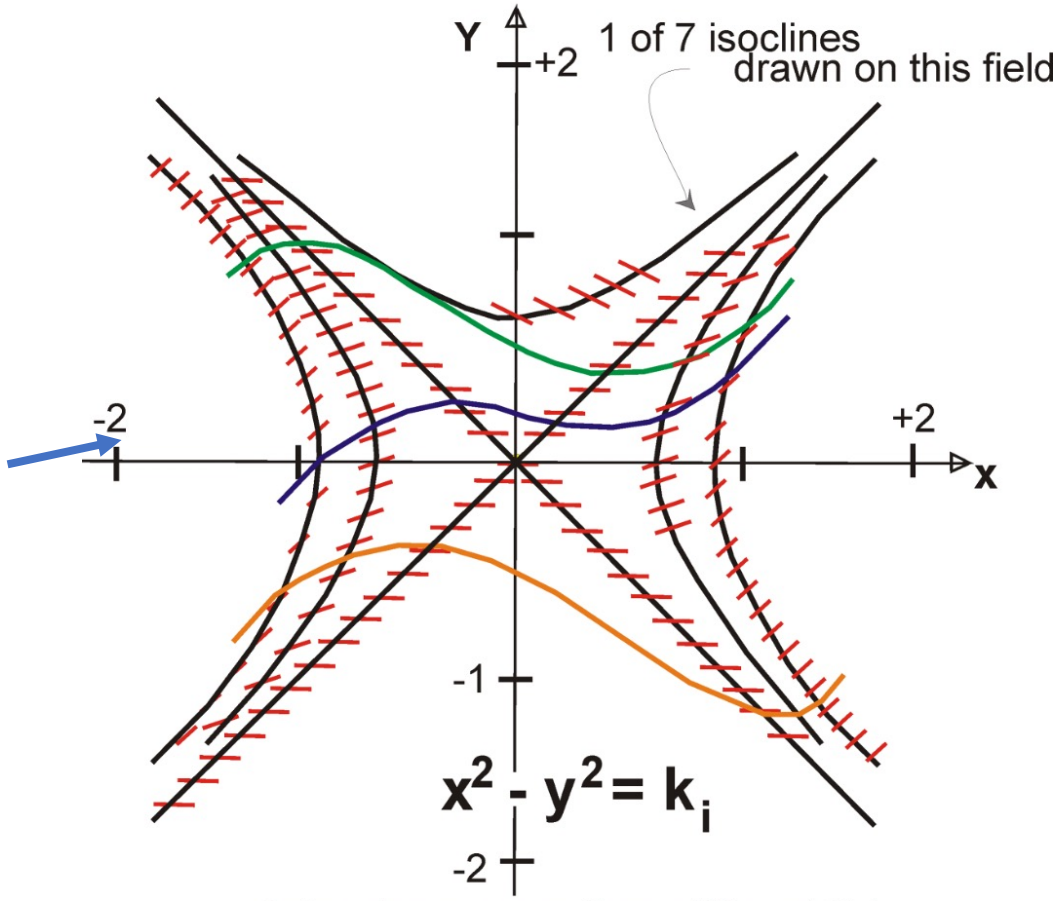
ORDINARY DIFFERENTIAL EQUATIONS

Consider equation $y' = x^2 - y^2$. What do we mean by **solution**? A curve $y(x)$ whose slope is locally given by $y'(x)$.
There is not a single curve, but a **family of curves**, *i.e.* (in principle) a curve from every point (x_0, y_0)!

Graphic presentation:

> The ideas of the above paragraph can be made graphic. In the x, y plane we choose various points and compute the slope $x^2 - y^2$. At each of these points we draw a short line with the computed slope. These lines indicate the local direction of the solution, and by the use of a little imagination we can easily sketch in various solutions, provided the set of points through which we draw lines is sufficiently dense.

isoclines: lines of the same slope (here hyperbolas $x^2 - y^2 = c$)



Colored curves are three different f(x)
solutions for the 3 different differential equations

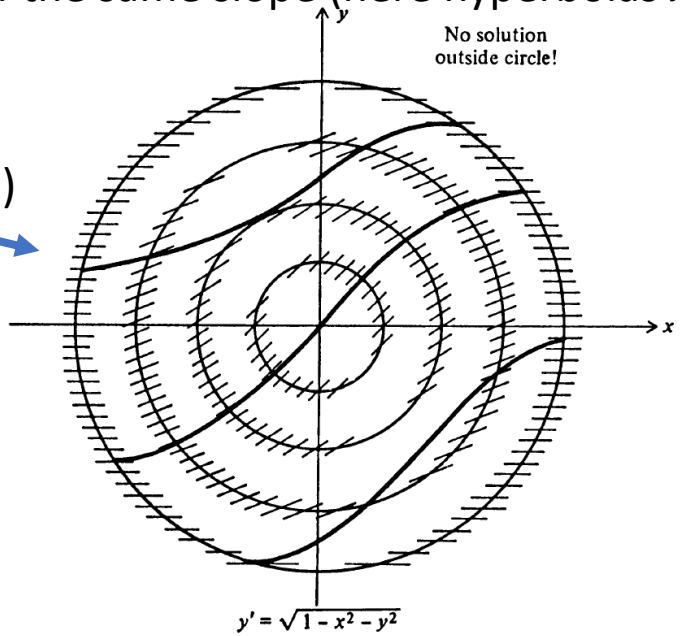
ORDINARY DIFFERENTIAL EQUATIONS

Consider equation $y' = x^2 - y^2$. What do we mean by **solution**? A curve $y(x)$ whose slope is locally given by $y'(x)$.
There is not a single curve, but a **family of curves**, *i.e.* (in principle) a curve from every point (x_0, y_0)!

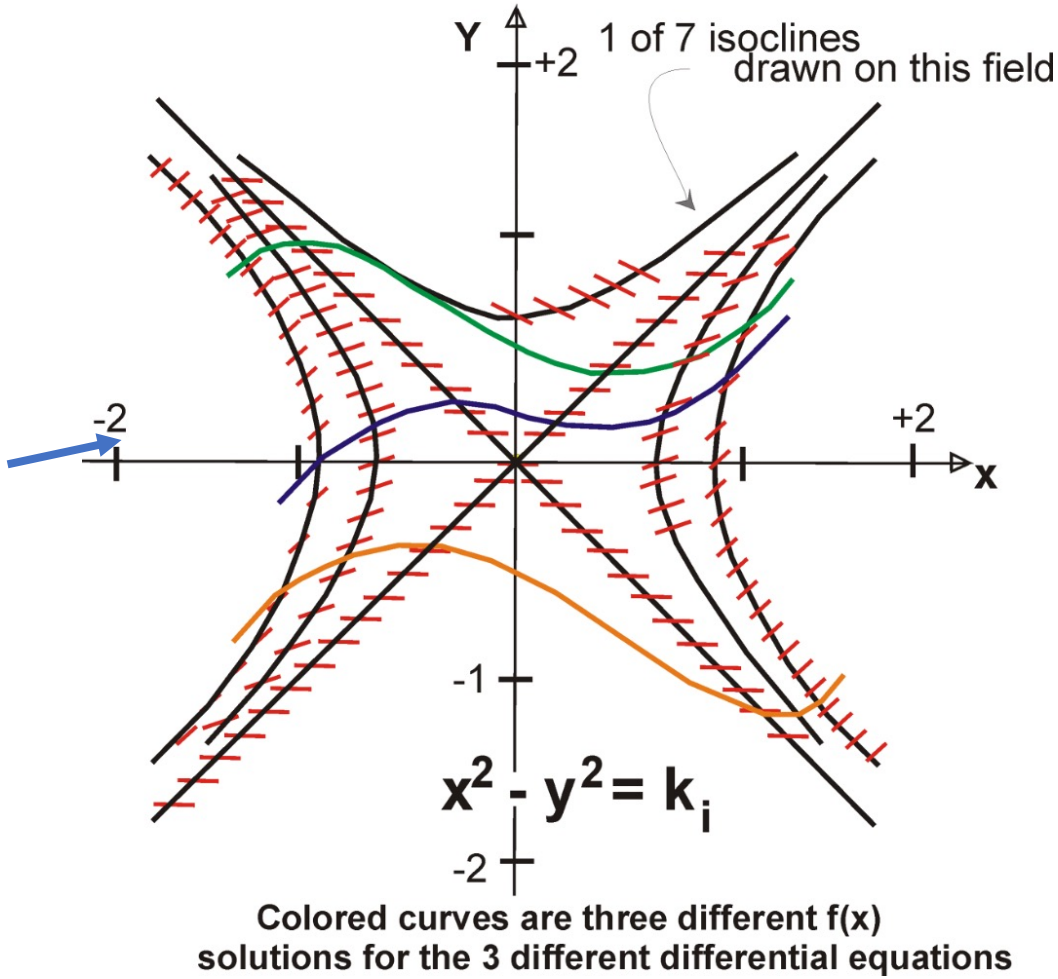
Graphic presentation:

> The ideas of the above paragraph can be made graphic. In the x, y plane we choose various points and compute the slope $x^2 - y^2$. At each of these points we draw a short line with the computed slope. These lines indicate the local direction of the solution, and by the use of a little imagination we can easily sketch in various solutions, provided the set of points through which we draw lines is sufficiently dense.

isoclines: lines of the same slope (here hyperbolas $x^2 - y^2 = c$)



(… here circles
$x^2 + y^2 = c \leq 1$)



**Colored curves are three different f(x)
solutions for the 3 different differential equations**

Consider equation $y' = x^2 - y^2$. What do we mean by **solution**? A curve $y(x)$ whose slope is locally given by $y'(x)$.
There is not a single curve, but a **family of curves**, *i.e.* (in principle) a curve from every point (x_0, y_0)!

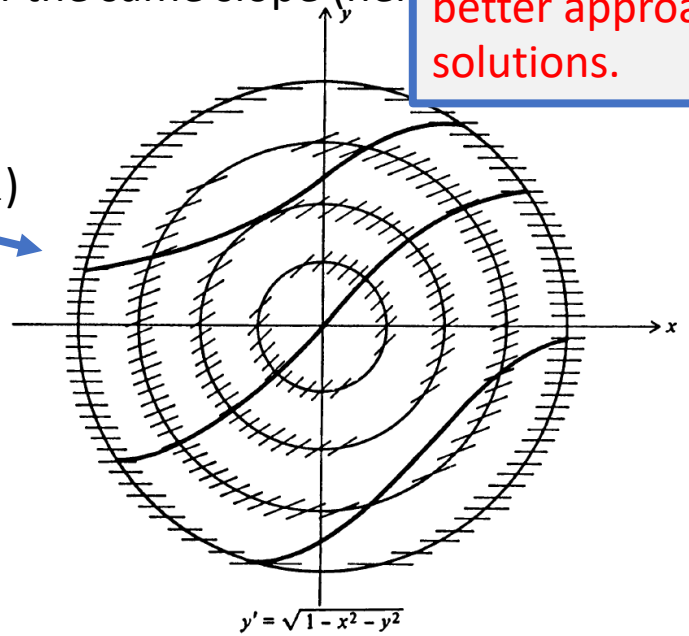
Graphic presentation:

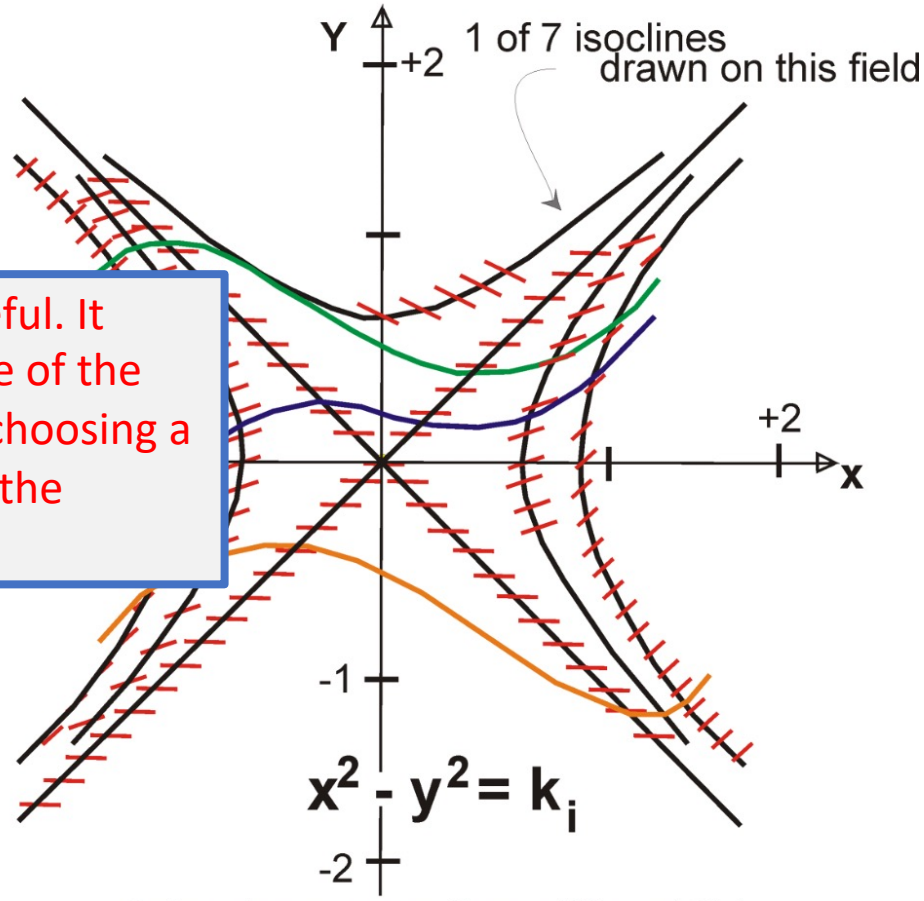
The ideas of the above paragraph can be made graphic. In the x, y plane we choose various points and compute the slope $x^2 - y^2$. At each of these points we draw a short line with the computed slope. These lines indicate the local direction of the solution, and by the use of a little imagin
various solutions, provided the set of points throu
ficiently dense.

Plotting **directional fields** is very useful. It provides us with a feel for the nature of the solution and this often assists us in choosing a better approach to numerically find the solutions.

isoclines: lines of the same slope (her

(… here circles
$x^2 + y^2 = c \leq 1$)



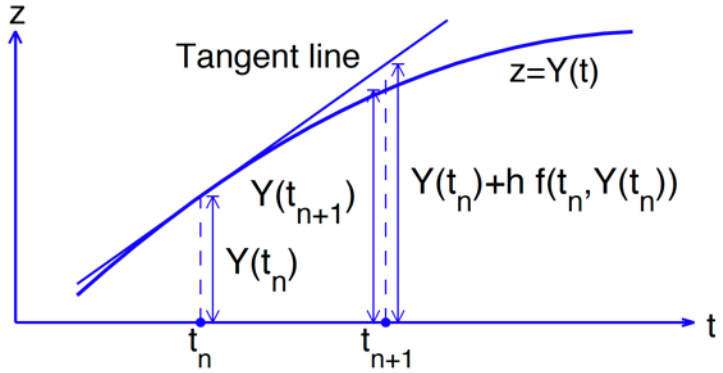


Colored curves are three different f(x)
solutions for the 3 different differential equations

ORDINARY DIFFERENTIAL EQUATIONS

General first-order ODE:

$$\frac{dy}{dt} \equiv \dot{y} = f(y,t)$$



The simplest idea:

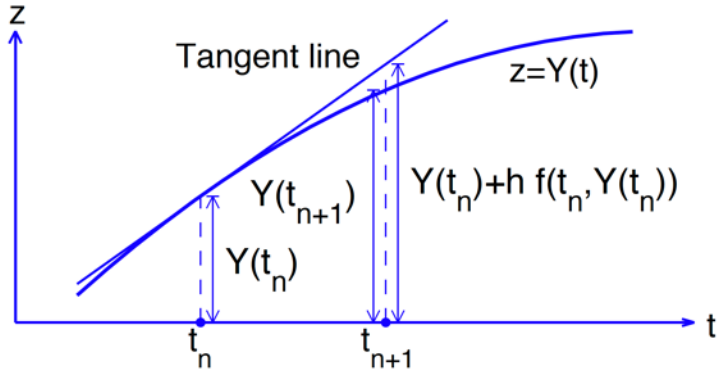
- start at the initial condition at $t = t_0$: (x_0, y_0)
- propagate the time in steps Δt
- calculate the value of y at the next step from the previous as

$$y_{k+1} = y_k + \Delta t\, \dot{y}_k = y_k + \Delta t\, f(y_k)$$

To get values $y_0 \to y_1 \to \cdots$ **DISCRETE TRAJECTORY** $\{y_k\}$ instead of continuous $y(t)$

General first-order ODE:

$$\frac{dy}{dt} \equiv \dot{y} = f(y,t)$$



The simplest idea:

- start at the initial condition at $t = t_0$: (x_0, y_0)
- propagate the time in steps Δt
- calculate the value of y at the next step from the previous as

$$y_{k+1} = y_k + \Delta t\, \dot{y}_k = y_k + \Delta t\, f(y_k)$$

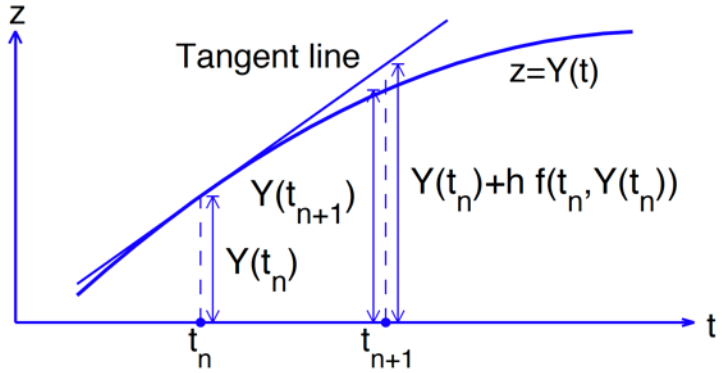
To get values $x_0 \to x_1 \to \cdots$ **DISCRETE TRAJECTORY** $\{x_k\}$ instead of continuous $x(t)$

This method is called **forward Euler method**. It is simple to implement.

ORDINARY DIFFERENTIAL EQUATIONS

General first-order ODE:

$$\frac{dy}{dt} \equiv \dot{y} = f(y,t)$$

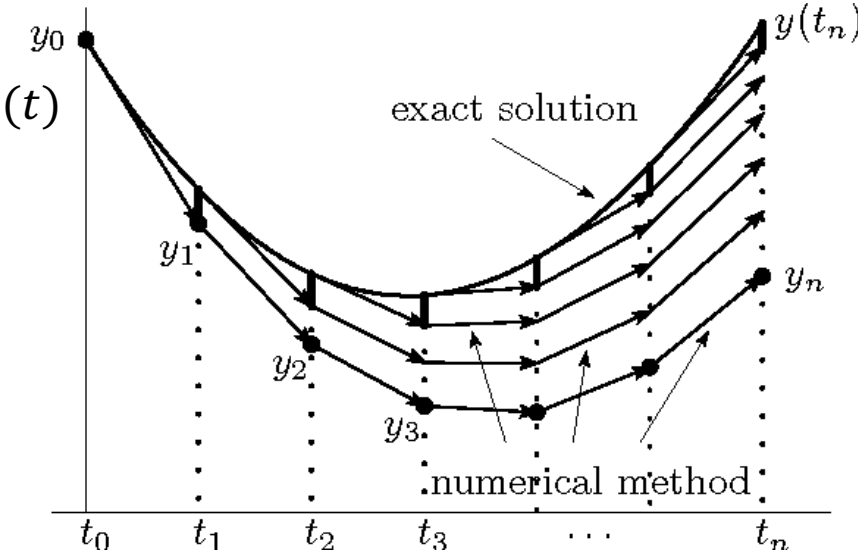


The simplest idea:

- start at the initial condition at $t = t_0$: (x_0, y_0)
- propagate the time in steps Δt
- calculate the value of y at the next step from the previous as

$$y_{k+1} = y_k + \Delta t\, \dot{y}_k = y_k + \Delta t\, f(y_k)$$

To get values $x_0 \to x_1 \to \cdots$ **DISCRETE TRAJECTORY** $\{x_k\}$ instead of continuous $x(t)$



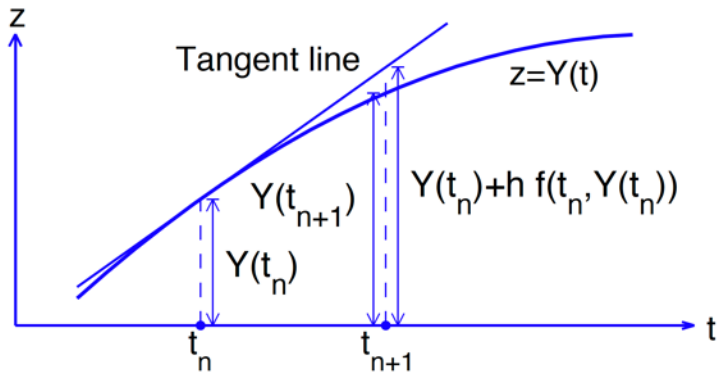
This method is called **forward Euler method**. It is simple to implement.
It is terribly **inacurrate** and **not stable**!
The error keeps accumulating, it scales globally with Δt
(we will later explain what this means and how to derive this…)

ORDINARY DIFFERENTIAL EQUATIONS

$$\frac{dy}{dt} \equiv \dot{y} = f(y,t)$$

Another possibility: calculate the value of y at the next step from the previous as

$$y_{k+1} = y_k + \Delta t\, f(y_{k+1})$$



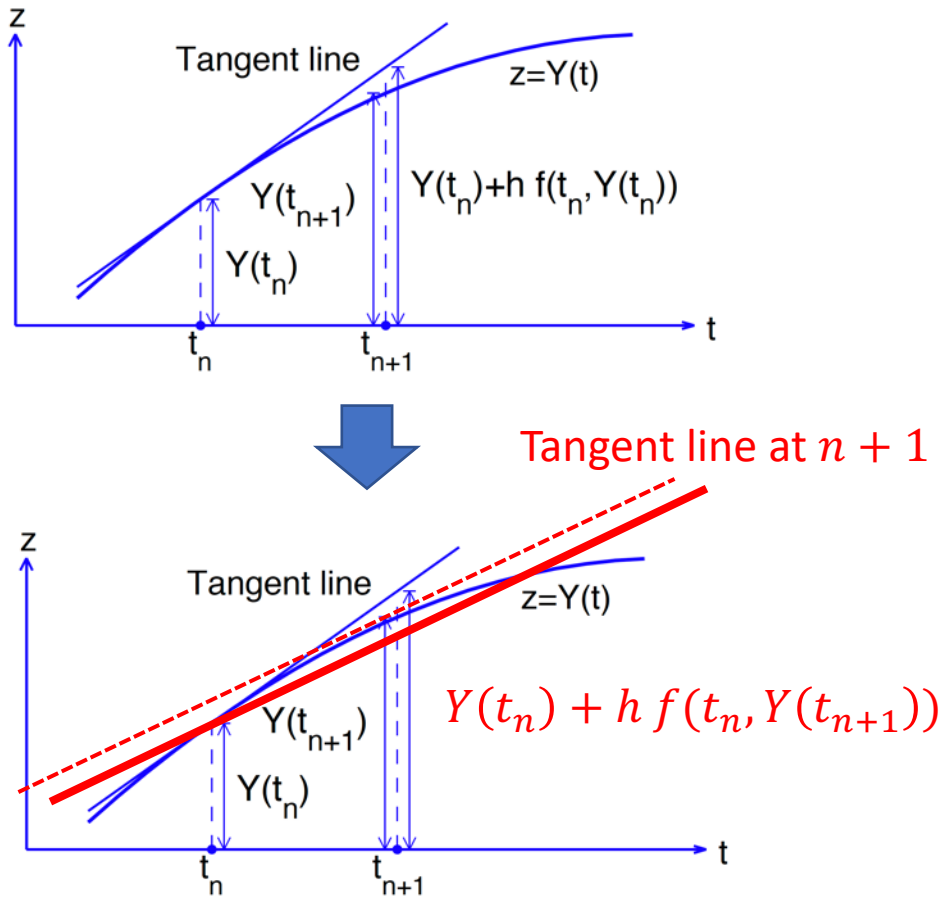
ORDINARY DIFFERENTIAL EQUATIONS

$$\frac{dy}{dt} \equiv \dot{y} = f(y,t)$$

Another possibility: calculate the value of y at the next step from the previous as

$$y_{k+1} = y_k + \Delta t\, f(y_{k+1})$$

We use the derivative at time $k+1$ to calculate value at $k+1$



ORDINARY DIFFERENTIAL EQUATIONS

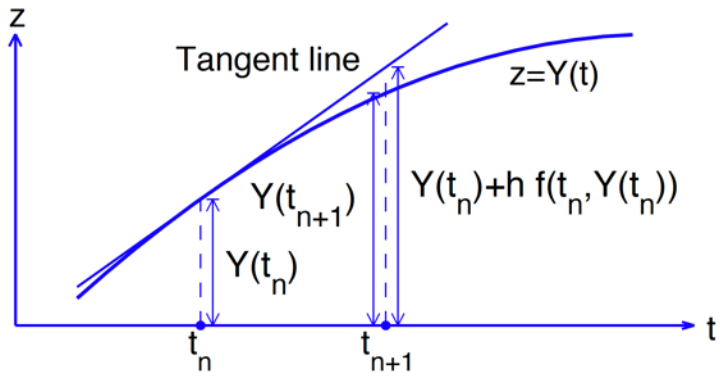
$$\frac{dy}{dt} \equiv \dot{y} = f(y,t)$$

Another possibility: calculate the value of y at the next step from the previous as

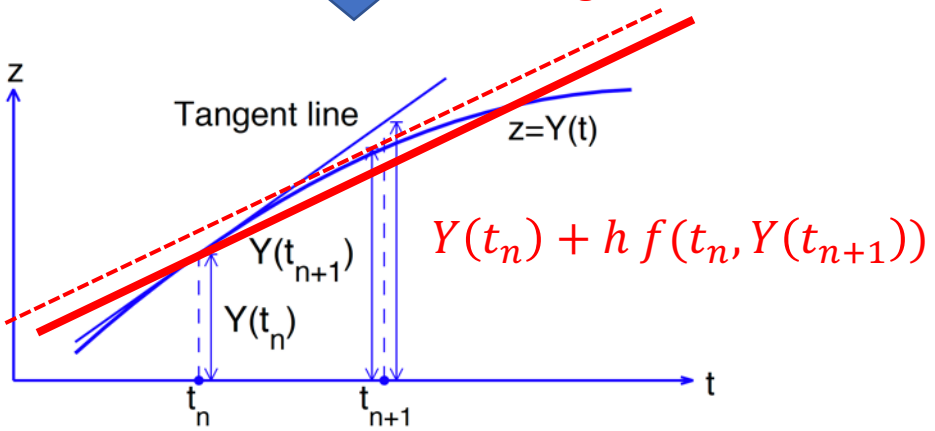
$$y_{k+1} = y_k + \Delta t\, f(y_{k+1})$$

We use the derivative at time $k+1$ to calculate value at $k+1$

- this method is called **backward Euler method**.
- it is a little bit more tricky to implement
- it is still quite **inacurrate** but much **more stable**!



Tangent line at $n+1$



$$Y(t_n) + h\, f(t_n, Y(t_{n+1}))$$

Taylor expansion: $y_{k+1} = y(t_k + \Delta t) = y_k + \Delta t\, \dot{y}_k + \frac{(\Delta t)^2}{2} \ddot{y}(\xi)$

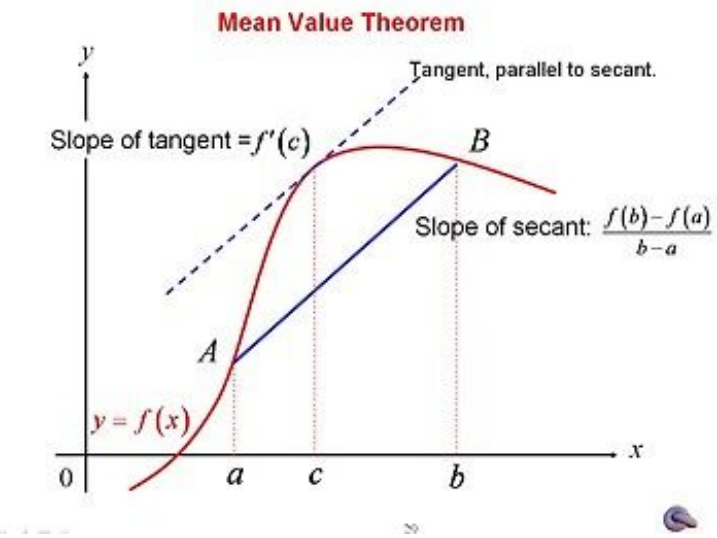
mean value theorem (remember?!)
ξ: some point on the interval…

EVALUATION OF FUNCTIONS

Mean value theorem

If a function f is continuous and has derivatives on the interval (a, b), then there exists a value θ on the interval, $a < \theta < b$, such that

$$f(b) - f(a) = (b - a)\, f'(\theta)$$



This can be used to speed up computations.

Example: if $f(x) = \ln x$, the mean value theorem says that $\ln b - \ln a = \frac{b-a}{\theta}$. The theorem does not specify the value of θ, only guaranties its existance. However, for practical purposes, since finite precision is required, it is often enough to assume that θ is the arithmetic mean $\theta = \frac{b-a}{2}$. This is especially useful, if $b \approx a$ and if the value $f(b)$ itself is large. In our example, if we assume a large a and $b = a + 1$, we finally have: $\ln a + 1 - \ln a \approx \frac{2}{2a+1}$. For $a = 100$, this would translate to $\mathbf{\ln 101 - \ln 100 \approx \frac{2}{201} \approx 0.0099502}$.

This is only different from the correct result on the 7th decimal digit, and **we did not need to evaluate the logarithm function at all!!** This "trick" can save us a lot of computational time.

Taylor expansion: $y_{k+1} = y(t_k + \Delta t) = y_k + \Delta t\, \dot{y}_k + \underbrace{\frac{(\Delta t)^2}{2}\ddot{y}(\xi)}$

mean value theorem (remember?!)
ξ: some point on the interval…

Local error: $\varepsilon_{k+1} = \frac{(\Delta t)^2}{2}\ddot{y}(\xi) \propto (\Delta t)^2$

Taylor expansion: $y_{k+1} = y(t_k + \Delta t) = y_k + \Delta t\, \dot{y}_k + \underbrace{\frac{(\Delta t)^2}{2}\ddot{y}(\xi)}$

mean value theorem (remember?!)
ξ: some point on the interval…

Local error: $\varepsilon_{k+1} = \frac{(\Delta t)^2}{2}\ddot{y}(\xi) \propto (\Delta t)^2$

At every time step we add this kind of error!
If we evaluate a trajectory of length T, we need to make $T/\Delta t$ steps

Global error: $\varepsilon_{k+1} \approx \frac{T\,\Delta t}{2}\ddot{y}(\xi) \propto \Delta t$

The same scaling for backward Euler.

Taylor expansion: $y_{k+1} = y(t_k + \Delta t) = y_k + \Delta t\, \dot{y}_k + \underbrace{\frac{(\Delta t)^2}{2}\ddot{y}(\xi)}$

mean value theorem (remember?!)
ξ: some point on the interval…

Local error: $\varepsilon_{k+1} = \frac{(\Delta t)^2}{2}\ddot{y}(\xi) \propto (\Delta t)^2$

At every time step we add this kind of error!
If we evaluate a trajectory of length T, we need to make $T/\Delta t$ steps

Global error: $\varepsilon_{k+1} \approx \frac{T\,\Delta t}{2}\ddot{y}(\xi) \propto \Delta t$

The same scaling for backward Euler.

The global error in both cases (FE and BE) scales linearly with the time step.

Taylor expansion: $y_{k+1} = y(t_k + \Delta t) = y_k + \Delta t\, \dot{y}_k + \underbrace{\frac{(\Delta t)^2}{2}\ddot{y}(\xi)}$

mean value theorem (remember?!)
ξ: some point on the interval…

Local error: $\varepsilon_{k+1} = \frac{(\Delta t)^2}{2}\ddot{y}(\xi) \propto (\Delta t)^2$

At every time step we add this kind of error!
If we evaluate a trajectory of length T, we need to make $T/\Delta t$ steps

Global error: $\varepsilon_{k+1} \approx \frac{T\,\Delta t}{2}\ddot{y}(\xi) \propto \Delta t$

The same scaling for backward Euler.

The global error in both cases (FE and BE) scales linearly with the time step.
We can construct much better algorithms (will do later).

STABILITY

Suppose we have a simpler equation where there is a **linear relation** $\dot{y} \equiv f(y) = \mathrm{A}y$

FORWARD EULER (FE): $y_{k+1} = y_k + \Delta t\, f(y_k) = (\mathrm{I} + \Delta t\, \mathrm{A})y_k$

BACKWARD EULER (BE): $y_{k+1} = y_k + \Delta t\, f(y_{k+1}) = y_k + \Delta t\, \mathrm{A}\, y_{k+1} = (\mathrm{I} - \Delta t\, \mathrm{A})^{-1} y_k$

GENERALLY: $y_{k+1} = \mathrm{B}\, y_k$ with $\mathrm{B} = \begin{cases} (\mathrm{I} + \Delta t\, \mathrm{A}) & \text{for FE} \\ (\mathrm{I} - \Delta t\, \mathrm{A})^{-1} & \text{for BE} \end{cases}$

A, B and I are matrices, I is identity matrix

We have matrices instead of scalars, if we are solving a **system of 1st order ODEs** instead of a single ODE. Fundamentally, there is no difference, so we can describe the method only once (*a scalar is a matrix of order 1*).

All ODEs Are (Systems of) First-Order ODEs

STABILITY

Suppose we have a simpler equation where there is a **linear relation** $\dot{y} \equiv f(y) = \mathrm{A}y$

FORWARD EULER (FE): $y_{k+1} = y_k + \Delta t\, f(y_k) = (\mathrm{I} + \Delta t\, \mathrm{A})y_k$

BACKWARD EULER (BE): $y_{k+1} = y_k + \Delta t\, f(y_{k+1}) = y_k + \Delta t\, \mathrm{A}\, y_{k+1} = (\mathrm{I} - \Delta t\, \mathrm{A})^{-1} y_k$

GENERALLY: $y_{k+1} = \mathrm{B}\, y_k$ with $\mathrm{B} = \begin{cases} (\mathrm{I} + \Delta t\, \mathrm{A}) & \text{for FE} \\ (\mathrm{I} - \Delta t\, \mathrm{A})^{-1} & \text{for BE} \end{cases}$

A, B and I are matrices, I is identity matrix

We have matrices instead of scalars, if we are solving a **system of 1st order ODEs** instead of a single ODE. Fundamentally, there is no difference, so we can describe the method only once (*a scalar is a matrix of order 1*).

All ODEs Are (Systems of) First-Order ODEs

If we iterate to go from y_0 to y_N, we get: $y_N = \mathrm{B}^N y_0$. This is **stable** (bounded) only when **no eigenvalue of** B, λ_i^B, is larger than 1:

$|\lambda_i^B| \leq 1$ for all eigenvalues of B → bounded solutions

STABILITY

Suppose we have a simpler equation where there is a **linear relation** $\dot{y} \equiv f(y) = \mathrm{A}y$

FORWARD EULER (FE): $y_{k+1} = y_k + \Delta t\, f(y_k) = (\mathrm{I} + \Delta t\, \mathrm{A})y_k$

BACKWARD EULER (BE): $y_{k+1} = y_k + \Delta t\, f(y_{k+1}) = y_k + \Delta t\, \mathrm{A}\, y_{k+1} = (\mathrm{I} - \Delta t\, \mathrm{A})^{-1} y_k$

GENERALLY: $y_{k+1} = \mathrm{B}\, y_k$ with $\mathrm{B} = \begin{cases} (\mathrm{I} + \Delta t\, \mathrm{A}) & \text{for FE} \\ (\mathrm{I} - \Delta t\, \mathrm{A})^{-1} & \text{for BE} \end{cases}$

A, B and I are matrices, I is identity matrix

We have matrices instead of scalars, if we are solving a **system of 1st order ODEs** instead of a single ODE. Fundamentally, there is no difference, so we can describe the method only once (*a scalar is a matrix of order 1*).

All ODEs Are (Systems of) First-Order ODEs

If we iterate to go from y_0 to y_N, we get: $y_N = \mathrm{B}^N y_0$. This is **stable** (bounded) only when **no eigenvalue of** B, λ_i^B, is larger than 1:

$|\lambda_i^B| \leq 1$ for all eigenvalues of B → bounded solutions

You can see that in a simple example: equation for population growth of bacteria (or a simple for epidemic dynamics): $\frac{dy}{dt} = \beta y(t)$

We know the solution: $\mathrm{y(t)} = y(0)e^{\beta t}$. **For positive $\boldsymbol{\beta}$ this is exponential growth, for negative $\boldsymbol{\beta}$, exponential decay.**

STABILITY

Suppose we have a simpler equation where there is a **linear relation** $\dot{y} \equiv f(y) = \mathrm{A}y$

FORWARD EULER (FE): $y_{k+1} = y_k + \Delta t\, f(y_k) = (\mathrm{I} + \Delta t\, \mathrm{A})y_k$

BACKWARD EULER (BE): $y_{k+1} = y_k + \Delta t\, f(y_{k+1}) = y_k + \Delta t\, \mathrm{A}\, y_{k+1} = (\mathrm{I} - \Delta t\, \mathrm{A})^{-1} y_k$

GENERALLY: $y_{k+1} = \mathrm{B}\, y_k$ with $\mathrm{B} = \begin{cases} (\mathrm{I} + \Delta t\, \mathrm{A}) & \text{for FE} \\ (\mathrm{I} - \Delta t\, \mathrm{A})^{-1} & \text{for BE} \end{cases}$

A, B and I are matrices, I is identity matrix

We have matrices instead of scalars, if we are solving a **system of 1st order ODEs** instead of a single ODE. Fundamentally, there is no difference, so we can describe the method only once (*a scalar is a matrix of order 1*).

All ODEs Are (Systems of) First-Order ODEs

If we iterate to go from y_0 to y_N, we get: $y_N = \mathrm{B}^N y_0$. This is **stable** (bounded) only when **no eigenvalue of** B, λ_i^B, is larger than 1:

$|\lambda_i^B| \leq 1$ for all eigenvalues of B → bounded solutions

You can see that in a simple example: equation for population growth of bacteria (or a simple for epidemic dynamics): $\frac{dy}{dt} = \beta y(t)$

We know the solution: $\mathrm{y(t)} = y(0)e^{\beta t}$. **For positive $\boldsymbol{\beta}$ this is exponential growth, for negative $\boldsymbol{\beta}$, exponential decay.**

Now what happens if we discretize the equation and use Euler formulae? We have: $y_N = b^N y_0$, where $\mathrm{b} = \begin{cases} 1 + \beta\Delta t & ;\text{for FE} \\ (1 - \beta\Delta t)^{-1} & ;\text{for BE} \end{cases}$

STABILITY

Suppose we have a simpler equation where there is a **linear relation** $\dot{y} \equiv f(y) = \mathrm{A}y$

FORWARD EULER (FE): $y_{k+1} = y_k + \Delta t\, f(y_k) = (\mathrm{I} + \Delta t\, \mathrm{A})y_k$

BACKWARD EULER (BE): $y_{k+1} = y_k + \Delta t\, f(y_{k+1}) = y_k + \Delta t\, \mathrm{A}\, y_{k+1} = (\mathrm{I} - \Delta t\, \mathrm{A})^{-1} y_k$

GENERALLY: $y_{k+1} = \mathrm{B}\, y_k$ with $\mathrm{B} = \begin{cases} (\mathrm{I} + \Delta t\, \mathrm{A}) & \text{for FE} \\ (\mathrm{I} - \Delta t\, \mathrm{A})^{-1} & \text{for BE} \end{cases}$

A, B and I are matrices, I is identity matrix

We have matrices instead of scalars, if we are solving a **system of 1st order ODEs** instead of a single ODE. Fundamentally, there is no difference, so we can describe the method only once (*a scalar is a matrix of order 1*).

All ODEs Are (Systems of) First-Order ODEs

If we iterate to go from y_0 to y_N, we get: $y_N = \mathrm{B}^N y_0$. This is **stable** (bounded) only when **no eigenvalue of** B, λ_i^B, is larger than 1:

$|\lambda_i^B| \leq 1$ for all eigenvalues of B → bounded solutions

You can see that in a simple example: equation for population growth of bacteria (or a simple for epidemic dynamics): $\frac{dy}{dt} = \beta y(t)$

We know the solution: $\mathrm{y(t)} = y(0)e^{\beta t}$. **For positive $\boldsymbol{\beta}$ this is exponential growth, for negative $\boldsymbol{\beta}$, exponential decay.**

Now what happens if we discretize the equation and use Euler formulae? We have: $y_N = b^N y_0$, where $\mathrm{b} = \begin{cases} 1 + \beta\Delta t & ;\text{for FE} \\ (1 - \beta\Delta t)^{-1} & ;\text{for BE} \end{cases}$

This is going to grow exponentially if $|b| > 1$. **For BE**, this means growth only if $\beta > 0$ and decay if $\beta < 0$, just like the analytical solution.

STABILITY

Suppose we have a simpler equation where there is a **linear relation** $\dot{y} \equiv f(y) = \mathrm{A}y$

FORWARD EULER (FE): $y_{k+1} = y_k + \Delta t\, f(y_k) = (\mathrm{I} + \Delta t\, \mathrm{A})y_k$

BACKWARD EULER (BE): $y_{k+1} = y_k + \Delta t\, f(y_{k+1}) = y_k + \Delta t\, \mathrm{A}\, y_{k+1} = (\mathrm{I} - \Delta t\, \mathrm{A})^{-1} y_k$

GENERALLY: $y_{k+1} = \mathrm{B}\, y_k$ with $\mathrm{B} = \begin{cases} (\mathrm{I} + \Delta t\, \mathrm{A}) & \text{for FE} \\ (\mathrm{I} - \Delta t\, \mathrm{A})^{-1} & \text{for BE} \end{cases}$

A, B and I are matrices, I is identity matrix

We have matrices instead of scalars, if we are solving a **system of 1st order ODEs** instead of a single ODE. Fundamentally, there is no difference, so we can describe the method only once (*a scalar is a matrix of order 1*).

All ODEs Are (Systems of) First-Order ODEs

If we iterate to go from y_0 to y_N, we get: $y_N = \mathrm{B}^N y_0$. This is **stable** (bounded) only when **no eigenvalue of** B, λ_i^B, is larger than 1:

$|\lambda_i^B| \leq 1$ for all eigenvalues of B → bounded solutions

You can see that in a simple example: equation for population growth of bacteria (or a simple for epidemic dynamics): $\frac{dy}{dt} = \beta y(t)$

We know the solution: $\mathrm{y(t)} = y(0)e^{\beta t}$. **For positive $\boldsymbol{\beta}$ this is exponential growth, for negative $\boldsymbol{\beta}$, exponential decay.**

Now what happens if we discretize the equation and use Euler formulae? We have: $y_N = b^N y_0$, where $\mathrm{b} = \begin{cases} 1 + \beta\Delta t & ;\text{for FE} \\ (1 - \beta\Delta t)^{-1} & ;\text{for BE} \end{cases}$

This is going to grow exponentially if $|b| > 1$. **For BE**, this means growth only if $\beta > 0$ and decay if $\beta < 0$, just like the analytical solution.

For FE, however, this means growth if $\begin{cases} \beta > 0 \\ \beta\Delta t < -1 \end{cases}$.

STABILITY

Suppose we have a simpler equation where there is a **linear relation** $\dot{y} \equiv f(y) = \mathrm{A}y$

FORWARD EULER (FE): $y_{k+1} = y_k + \Delta t\, f(y_k) = (\mathrm{I} + \Delta t\, \mathrm{A})y_k$

BACKWARD EULER (BE): $y_{k+1} = y_k + \Delta t\, f(y_{k+1}) = y_k + \Delta t\, \mathrm{A}\, y_{k+1} = (\mathrm{I} - \Delta t\, \mathrm{A})^{-1} y_k$

GENERALLY: $y_{k+1} = \mathrm{B}\, y_k$ with $\mathrm{B} = \begin{cases} (\mathrm{I} + \Delta t\, \mathrm{A}) & \text{for FE} \\ (\mathrm{I} - \Delta t\, \mathrm{A})^{-1} & \text{for BE} \end{cases}$

A, B and I are matrices, I is identity matrix

We have matrices instead of scalars, if we are solving a **system of 1st order ODEs** instead of a single ODE. Fundamentally, there is no difference, so we can describe the method only once (*a scalar is a matrix of order 1*).

All ODEs Are (Systems of) First-Order ODEs

If we iterate to go from y_0 to y_N, we get: $y_N = \mathrm{B}^N y_0$. This is **stable** (bounded) only when **no eigenvalue of** B, λ_i^B, is larger than 1:

$|\lambda_i^B| \leq 1$ for all eigenvalues of B → bounded solutions

You can see that in a simple example: equation for population growth of bacteria (or a simple for epidemic dynamics): $\frac{dy}{dt} = \beta y(t)$

We know the solution: $\mathrm{y(t)} = y(0)e^{\beta t}$. **For positive β this is exponential growth, for negative β, exponential decay.**

Now what happens if we discretize the equation and use Euler formulae? We have: $y_N = b^N y_0$, where $\mathrm{b} = \begin{cases} 1 + \beta\Delta t & ;\text{for FE} \\ (1 - \beta\Delta t)^{-1} & ;\text{for BE} \end{cases}$

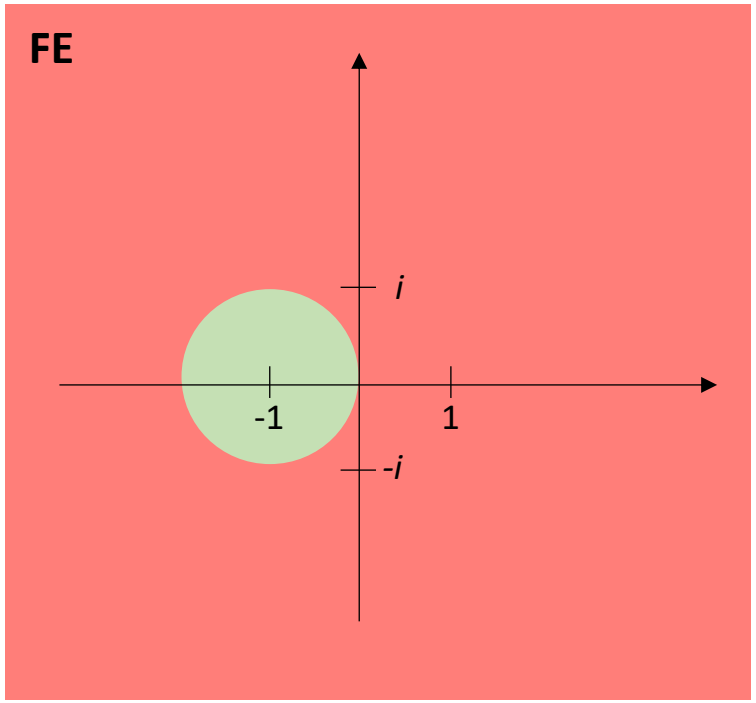
This is going to grow exponentially if $|b| > 1$. **For BE**, this means growth only if $\beta > 0$ and decay if $\beta < 0$, just like the analytical solution.

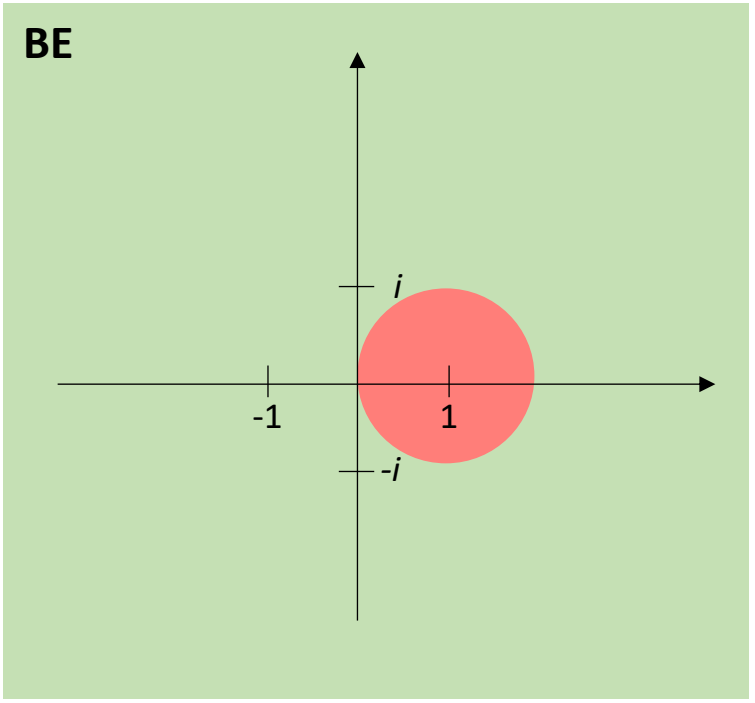
For FE, however, this means growth if $\begin{cases} \beta > 0 \\ \beta\Delta t < -1 \end{cases}$. **The FE is unstable even for large negative β, unless the time step Δt is really small!!**

ORDINARY DIFFERENTIAL EQUATIONS

In general, if $\beta \Delta t$ is a complex number, we can nicely visualize the regions of stability:

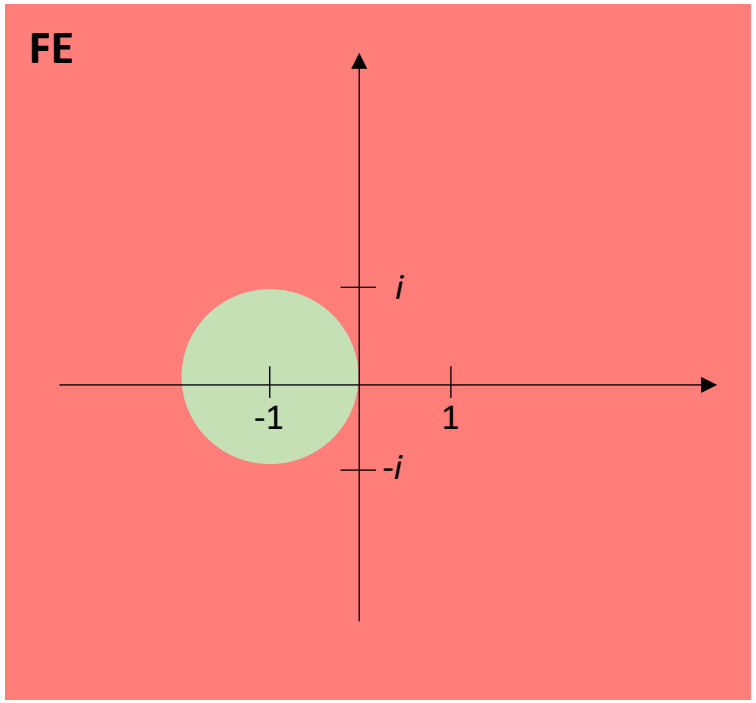
GREEN: stable
RED: unstable

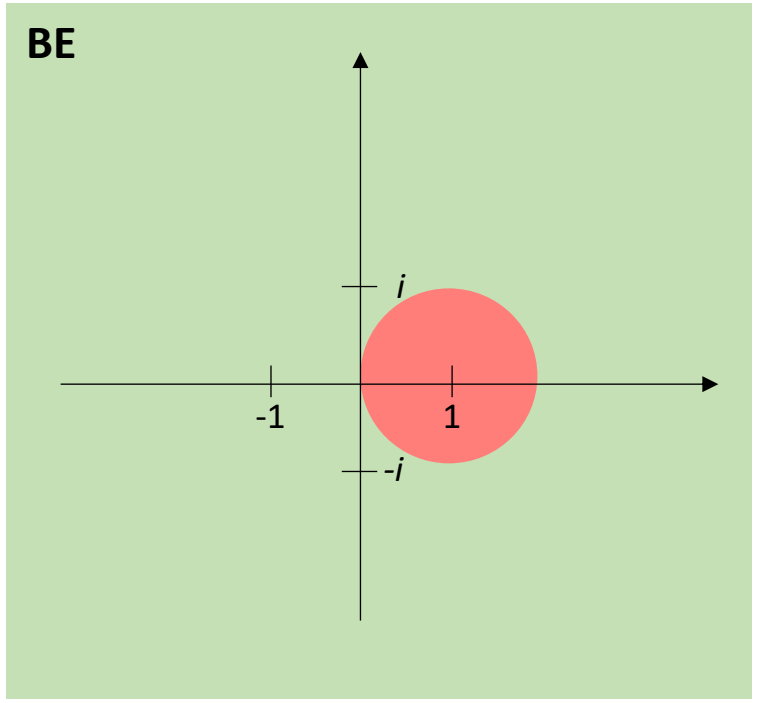




In general, if $\beta\Delta t$ is a complex number, we can nicely visualize the regions of stability:

GREEN: stable
RED: unstable



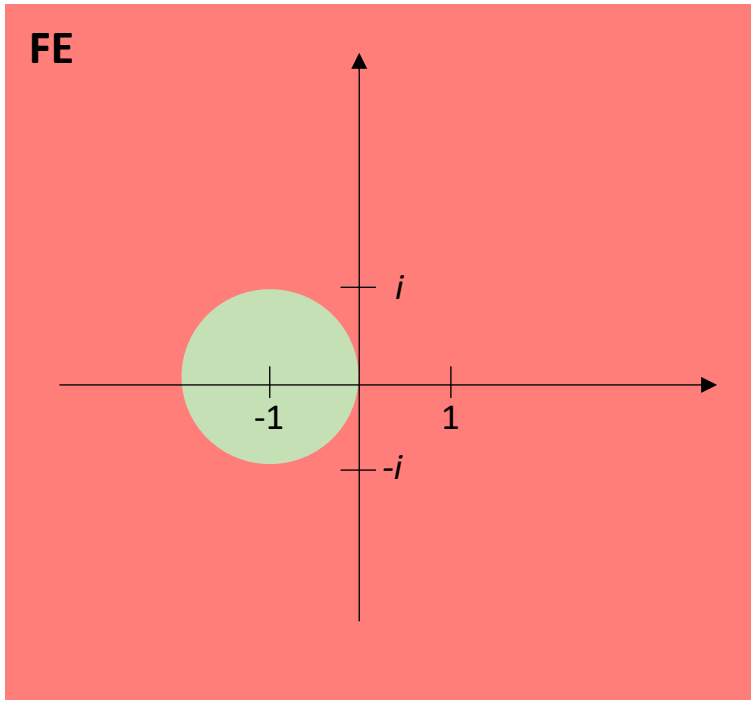


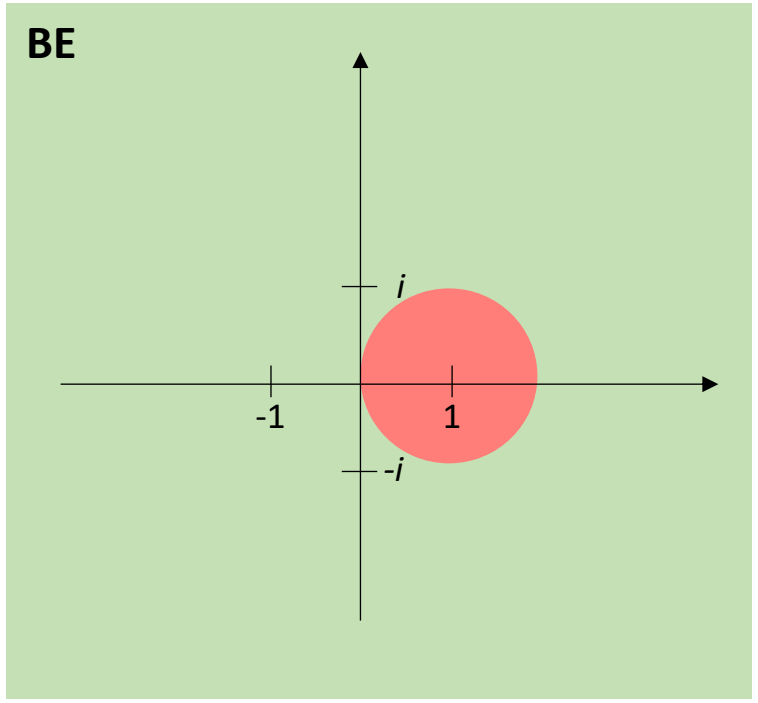
In general, if $\beta\Delta t$ is a complex number, we can nicely visualize the regions of stability:

GREEN: stable
RED: unstable

The forward formula is awful in stability: unstable everywhere in the complex plane except within the unit circle

Backward formula is much better in stability: stable everywhere except in the unit circle.





In general, if $\beta\Delta t$ is a complex number, we can nicely visualize the regions of stability:

GREEN: stable
RED: unstable

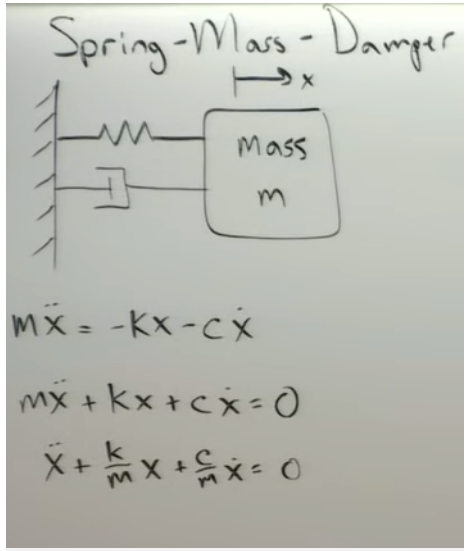
BE is slightly more difficult to evaluate, since we need to invert the matrix. In iterative algorithm, this means that we need to perform three operations instead of two at each time step:

- calculate the "trial" y_{k+1}^T with a forward method
- evaluate the derivative $\dot{y}_{k+1}^T$ at the trial point
- use that derivative to calculate real new value $y_{k+1} = y_k + \Delta t\, \dot{y}_{k+1}^T$

The forward formula is awful in stability: unstable everywhere in the complex plane except within the unit circle

Backward formula is much better in stability: stable everywhere except in the unit circle.

EXAMPLE: spring-mass-damper



$$\omega_n = \sqrt{\frac{k}{m}} \qquad \zeta = \frac{C}{2\sqrt{km}}$$

All ODEs Are (Systems of) First-Order ODEs

$\ddot{x} + 2\zeta\omega_n\dot{x} + {\omega_n}^2 x = 0$ Newton's 2nd law → 2nd order ODE → system of two first-order ODEs

$$\dot{x} = v$$
$$\dot{v} = -2\zeta\omega_n v - {\omega_n}^2 x$$

$$\frac{d}{dt}\begin{bmatrix} x \\ v \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -{\omega_n}^2 & -2\zeta\omega_n \end{bmatrix}\begin{bmatrix} x \\ v \end{bmatrix} \qquad \boldsymbol{x} = \begin{bmatrix} x \\ v \end{bmatrix} \rightarrow \dot{\boldsymbol{x}} = \mathrm{A}\,\boldsymbol{x}$$

Eigenvalues of A → behaviour of the system $\lambda_{1,2} = -\zeta\omega_n \pm \mathrm{i}\,\omega_n\sqrt{1-\zeta^2} = -\zeta\omega_n \pm \mathrm{i}\,\omega_d$

Case 1: underdamped ($\zeta < 1$): complex eigenvalues, system oscilates with $\omega_d = \omega_n\sqrt{1-\zeta^2}$

$$x = e^{-\zeta\omega_n t}\left[\frac{\dot{x}_o + \zeta\omega_n x_o}{\omega_d}\sin(\omega_d t) + x_o\cos(\omega_d t)\right]$$

Case 2: overdamped ($\zeta > 1$): eigenvalues both real and negative $\lambda_{2,1} = -\zeta\omega_n \pm \omega_d^*$

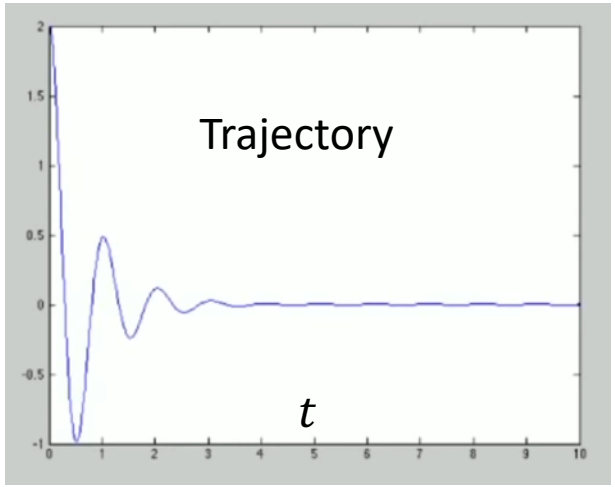
$$x = e^{-\zeta\omega_n t}\left(x_0\cosh\omega_d^* t + \frac{\dot{x}_0}{\omega_d^*}\sinh\omega_d^* t\right)$$

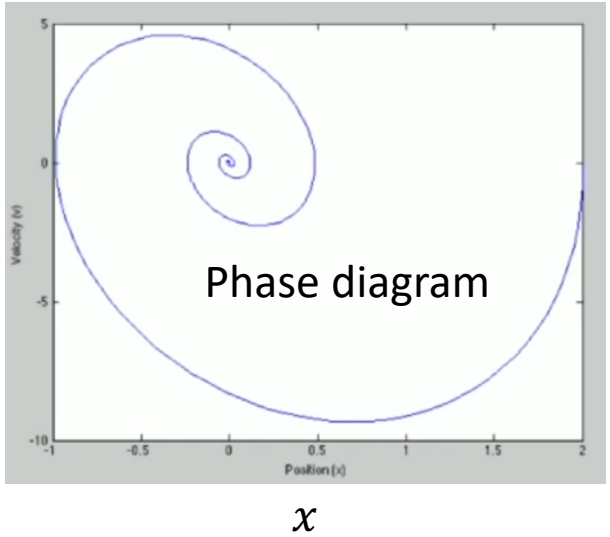
Case 3: critically damped ($\zeta = 1$): $\lambda_{1,2} = -\zeta\omega_n$

$$x = [x_0 + (x_0\zeta\omega_n + \dot{x}_0)t]\ e^{-\zeta\omega_n t}$$

Natural frequency: $\omega_n = 2\pi$
Damping: $\zeta = 0.25$
Initial condition: $x_0 = 2, v_0 = 0$
Time step: $\Delta t = 0.01$

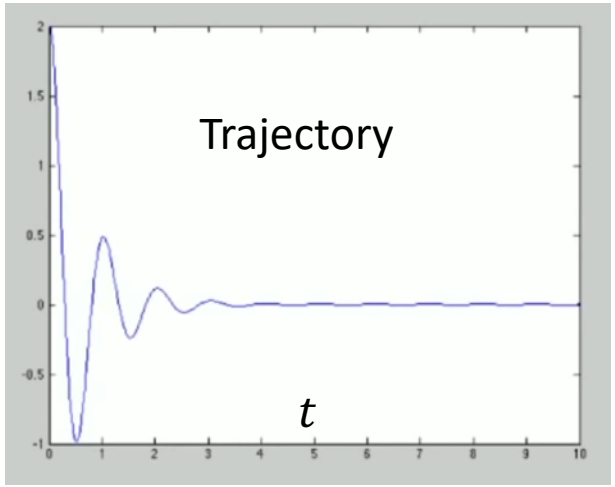
TEST: FORWARD EULER METHOD
FOR UNDERDAMPED CASE

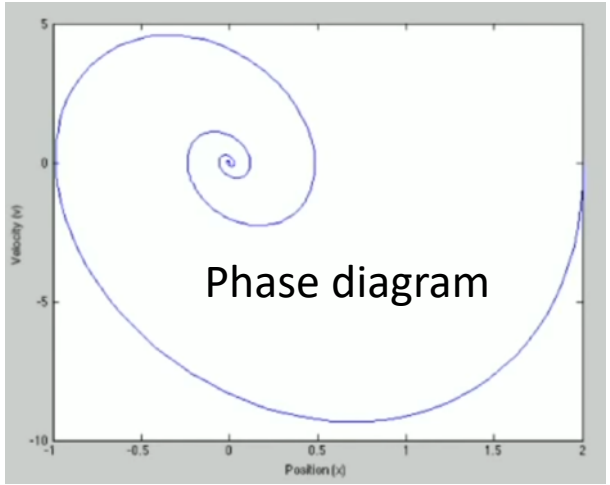




ORDINARY DIFFERENTIAL EQUATIONS

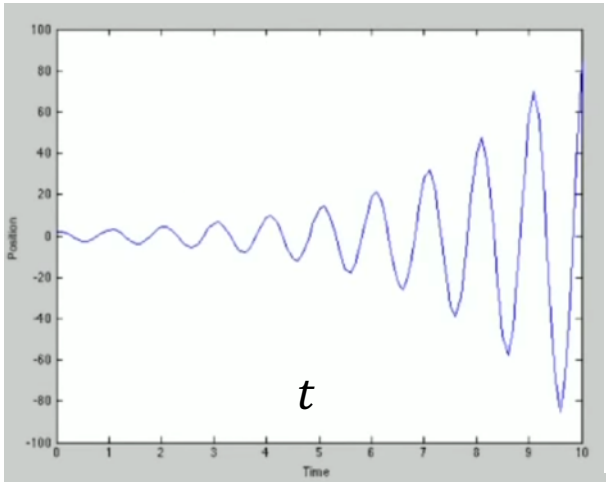
Natural frequency: $\omega_n = 2\pi$
Damping: $\zeta = 0.25$
Initial condition: $x_0 = 2, v_0 = 0$
Time step: $\Delta t = 0.01$

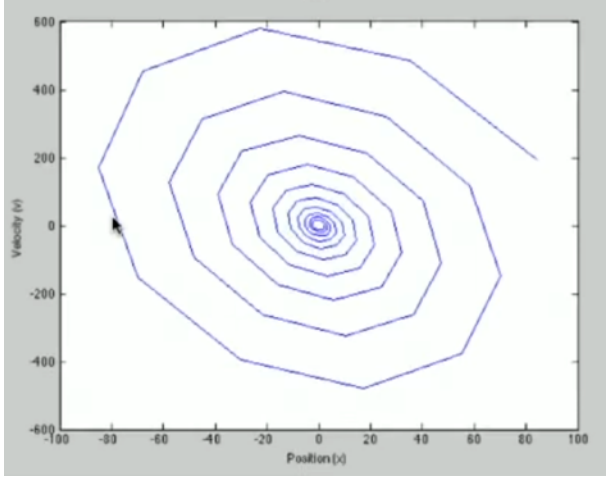




TEST: FORWARD EULER METHOD
FOR UNDERDAMPED CASE

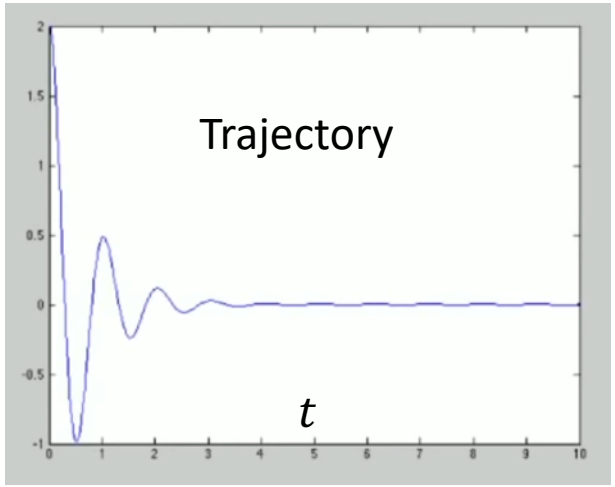
Natural frequency: $\omega_n = 2\pi$
Damping: $\zeta = 0.25$
Initial condition: $x_0 = 2, v_0 = 0$
Time step: $\Delta t = 0.1$

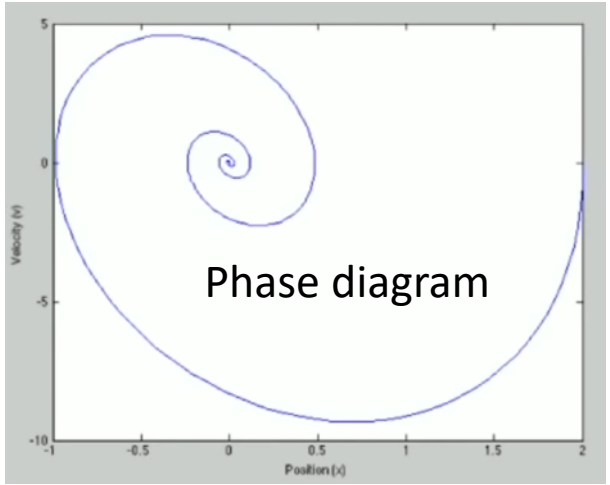




ORDINARY DIFFERENTIAL EQUATIONS

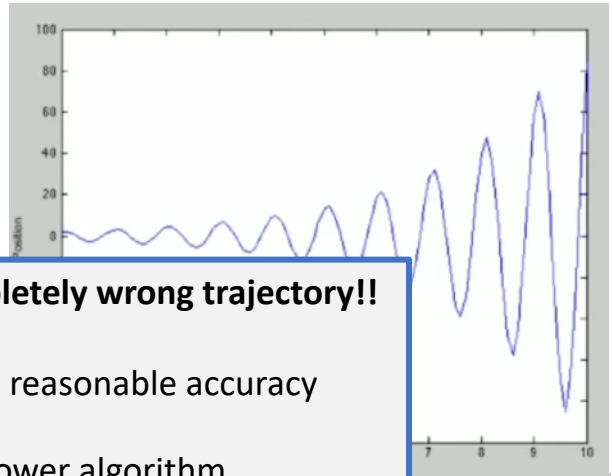
Natural frequency: $\omega_n = 2\pi$
Damping: $\zeta = 0.25$
Initial condition: $x_0 = 2, v_0 = 0$
Time step: $\Delta t = 0.01$





TEST: FORWARD EULER METHOD
FOR UNDERDAMPED CASE

Natural frequency: $\omega_n = 2\pi$
Damping: $\zeta = 0.25$
Initial condition: $x_0 = 2, v_0 = 0$
Time step: $\Delta t = 0.1$



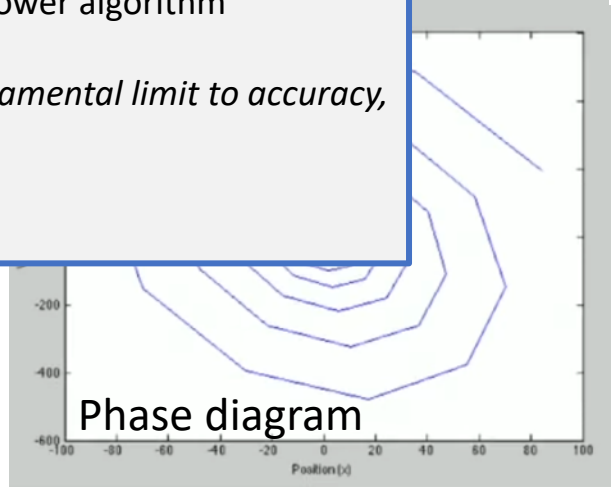
At larger time step the FE produces completely wrong trajectory!!

We need small-enough time step to reach reasonable accuracy

Small time step → more evaluations → slower algorithm

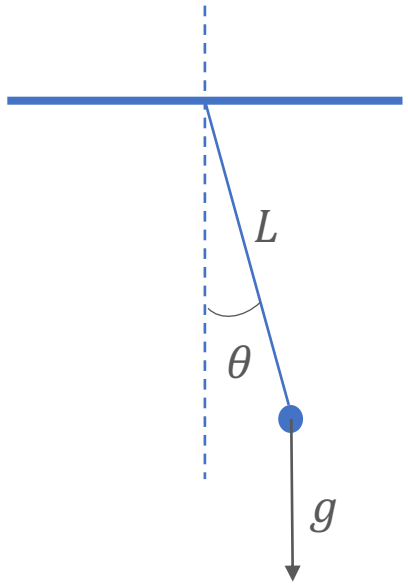
Due to rounding-off errors, there is a fundamental limit to accuracy, even with very small time steps

FE method should NOT be used!



ORDINARY DIFFERENTIAL EQUATIONS

EXAMPLE: single pendulum



Lagrangian formalism:

$$T = \frac{1}{2}mL^2\dot{\theta}^2$$
$$V = mLg(1-\cos\theta)$$
$$L = T - V$$

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{\theta}} - \frac{\partial L}{\partial \theta} = 0$$

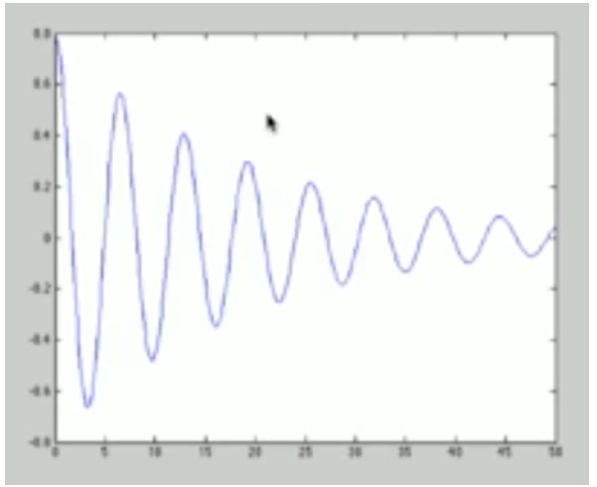
damping

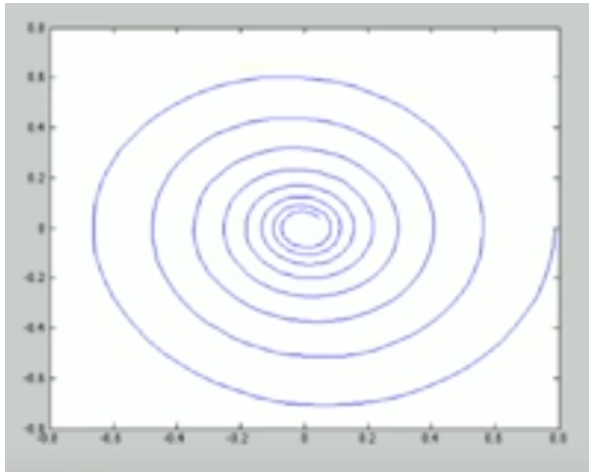
$$\ddot{\theta} = -\frac{g}{L}\sin\theta - \eta y$$

$$\dot{\theta} = y$$
$$\dot{y} = -\frac{g}{L}\sin\theta - \eta y$$

$$\begin{pmatrix}\dot{\theta}\\ \dot{y}\end{pmatrix} = \begin{pmatrix}f_1(\theta, y)\\ f_2(\theta, y)\end{pmatrix}$$

The solution is presented with plots:





Here a better integrator than Euler was used: Runge-Kutta, 4th order; we will introduce it later

TASK: TRY TO SIMULATE THIS WITH FE AND BE METHODS!

Let us look again at the equations:

FORWARD EULER (FE): $y_{k+1} = y_k + \Delta t\, f(t_k, y_k)$ **EXPLICIT**

BACKWARD EULER (BE): $y_{k+1} = y_k + \Delta t\, f(t_{k+1}, y_{k+1})$ **IMPLICIT**

We note that even if the equations look quite similar, they are in fact very different in terms of computational approach. The BE is **implicit**, which means that the next value y_{k+1} appears on both sides of the equation. The position of the next point can therefore not be simply calculated in one step as it is done in the explicit FE method. **We need to find such y_{k+1} that solves the equation!**

There are several approaches for solving implicit equations, *e.g.*, predictor-corrector methods (we will not study these methods, but you can look them up), systems of algebraic equations (write down equations for many time steps and solve the system of equations with suitable methods), or iterative method where **we iterate the solution at each time step** until the desired accuracy is reached:

$y_{k+1}^{[0]} = y_k;\;\; y_{k+1}^{[1]} = y_k + \Delta t f\left(t_{k+1}, y_{k+1}^{[0]}\right)$

... $y_{k+1}^{[i+1]} = y_k + \Delta t f\left(t_{k+1}, y_{k+1}^{[i]}\right)$ ... stop if $\left|y_{k+1}^{[i+1]} - y_{k+1}^{[i]}\right| <$ required accuracy

So, the implicit method is much harder to solve. The benefit we get for the effort is the stability. Unlike FE method, BE (as in general all implicit methods) is almost always stable.

In both methods (FE and BE), the error scales linearly with the time step Δt, which in practice means that we need a small time step for a reasonable accuracy, and that leads to more function evaluations and longer calculation time. Those who have done the exercises know what this means. Besides being slow, given the fundamental limit of accuracy due to the round-off errors on any computer, the linear scaling with Δt means that this limit accuracy is relatively low for FE and BE methods.

Can we do better? Design a method where error terms cancel (as we have done for numerivcal derivation & integration)? **Yes.**

Let us assume a general form of the iterative equation: $y_{k+1} = y_k + \Delta t\, \phi$

If we choose ϕ to be the slope at (t_k, y_k) →FE

If ϕ is the slope at $(t_{k+1}, y_{k+1}) \equiv (t_k + \Delta t\ ,\ y_k + \Delta t f(t_k, y_k))$ →BE

Let us relax this and assume that **ϕ is some combination of the slopes** at (t_k, y_k) AND at another point $\left(t_k + P\Delta t, y_k + Q\Delta t f(t_k, y_k)\right)$:

$$y_{k+1} = y_k + \Delta t \left[A f(t_k, y_k) + B f\left(t_k + P\Delta t, y_k + Q\Delta t f(t_k, y_k)\right)\right]$$

The expression has four parameters: A, B, Q, and P. **We will try to find such combination of these parameters, that the low-order error terms cancel.** We are free to choose constants A, B, Q, and P; we will try to choose them so that we increase the accuracy, i.e., scaling of the error with Δt. Let us look at the most commonly used version and assume that $P = Q$. Then, the second term looks like a "small FE step" of length $P\Delta t$ instead of Δt.

The slope at $t_k + P\Delta t$ can be written as a Taylor expansion:

$$f\left(t_k + P\Delta t, y_k + P\Delta t f(t_k, y_k)\right) = \mathrm{f}(t_k, y_k) + P\Delta t \frac{\partial f}{\partial t}(t_k, y_k) + P\Delta t \frac{\partial f}{\partial y}(t_k, y_k) \cdot f(t_k, y_k) + \mathcal{O}(\Delta t^2)$$

and the equation can be re-written as

$$y_{k+1} = y_k + \Delta t(A + B) f(t_k, y_k) + BP\Delta t^2 \left[\frac{\partial f}{\partial t}(t_k, y_k) + f(t_k, y_k) \frac{\partial f}{\partial y}(t_k, y_k)\right] + \mathcal{O}(\Delta t^3)$$

We can also Tayor-expand the true solution:

$$y_{k+1} = y(t_k + \Delta t) = y(t_k) + \Delta t \frac{dy}{dt}(t_k) + \frac{\Delta t^2 d^2 y}{2\ dt^2}(t_k) + \mathcal{O}(\Delta t^3) = y(t_k) + \Delta t\, f(t_k, y_k) + \frac{\Delta t^2}{2}\left[\frac{\partial f}{\partial t}(t_k, y_k) + f(t_k, y_k) \frac{\partial f}{\partial y}(t_k, y_k)\right] + \mathcal{O}(\Delta t^3)$$

Finally, comparing the terms will allow us to choose good values…

If we match the terms with Δt and Δt^2, we arrive at the following system of equations:

$A + B = 1$

$PB = \frac{1}{2}$

$P = Q$ (we have assumed this)

The standard choice is named **Runge-Kutta 2nd order method**: $A = 0, B = 1, P = Q = 1/2$:

$$y_{k+1} = y_k + \Delta t\, f\left(t_k + \frac{\Delta t}{2}, y_k + \frac{\Delta t}{2} f(t_k, y_k)\right)$$

So, Runge-Kutta 2 method calculates the derivatives at half a step forwards and then makes a full step with those values…

Since we have matched the Taylor expansion of the exact solution up to the second order terms, the **error of the method locally scales as $\mathcal{O}(\Delta t^3)$. This means that globally it scales with Δt^2**.

If we match the terms with Δt and Δt^2, we arrive at the following system of equations:

$A + B = 1$

$PB = \frac{1}{2}$

$P = Q$ (we have assumed this)

The standard choice is named **Runge-Kutta 2nd order method**: $A = 0, B = 1, P = Q = 1/2$:

$$y_{k+1} = y_k + \Delta t\, f\left(t_k + \frac{\Delta t}{2}, y_k + \frac{\Delta t}{2} f(t_k, y_k)\right)$$

So, Runge-Kutta 2 method calculates the derivatives at half a step forwards and then makes a full step with those values…

Since we have matched the Taylor expansion of the exact solution up to the second order terms, the **error of the method locally scales as $\mathcal{O}(\Delta t^3)$. This means that globally it scales with Δt^2.**

It is possible (although painful) to go further and design a **Runge-Kutta 4th order method**:

$$y_{k+1} = y_k + \frac{\Delta t}{6}[f_1 + 2f_2 + 2f_3 + f_4]$$
$$f_1 = f(t_k, y_k)$$
$$f_2 = f\left(t_k + \frac{\Delta t}{2}, y_k + \frac{\Delta t}{2} f_1\right)$$
$$f_3 = f\left(t_k + \frac{\Delta t}{2}, y_k + \frac{\Delta t}{2} f_2\right)$$
$$f_4 = f\left(t_k + \frac{\Delta t}{2}, y_k + \frac{\Delta t}{2} f_3\right)$$

The error scales locally with Δt^5, therefore globally with Δt^4.

Even if it takes more function evaluations at each step, the scaling with Δt makes up for this because much less time steps are needed for a given accuracy…

ORDINARY DIFFERENTIAL EQUATIONS

Runge-Kutta 4th order method is an excellent all-purpose integrator for ODEs and is a standard choice for scientist and engineers for many problems. It is fast, accurate and stable…

However, no ODE integrator is without problems and there is a vast number of different methods available. Some of those are (we are not going to study them in detail):

- Stiff integrators
- Energy conserving (symplectic) integrators
- Adams-Bashfort
- …

ORDINARY DIFFERENTIAL EQUATIONS

Runge-Kutta 4th order method is an excellent all-purpose integrator for ODEs and is a standard choice for scientist and engineers for many problems. It is fast, accurate and stable…

However, no ODE integrator is without problems and there is a vast number of different methods available. Some of those are (we are not going to study them in detail):

- Stiff integrators

  For solving equations with very different time-scales. Example: $\frac{dy}{dt} = -1000y + 3000 - 2000e^{-t}$.
  We know the analytical solution: $y = 3 - 0.998e^{-1000t} - 2.002e^{-t}$.
  You can imagine that this stiff equation is extremely difficult to integrate due to vastly different time-scales: e^{-1000t} and e^{-t}… Methods usually involve adaptive time steps, or/and solving for the Jacobian, which is computationally expensive. Fancy solvers switch between stiff and non-stiff methods according to the needs.

- Energy conserving (symplectic) integrators

  Instead of trying to match Taylor expansion terms, this family of integrators is constructed so that they conserve energy (or some other quantity…)

- Adams-Bashfort

  Specially optimized for fluid flow simulations, e.g. air flow around airplane wings…

- …

ORDINARY DIFFERENTIAL EQUATIONS

RECAP ODEs

- We have introduced ODEs and looked at some methods to solve them
- We have considered INITIAL VALUE PROBLEMS, this is problems where the initial conditions are specified and we are looking to predict trajectories for the evolution of the system following the underlying physical equations
- We analysed the accuracy and stability of different methods and mentioned some approaches to specific problems that we do not plan to discuss in detail (if interested, please find material or contact me to discuss further…)
- We have not looked at other important classes of ODE problems (**BOUNDARY VALUE** or **EIGENVALUE PROBLEMS**)

Another class of problems in physics requires the solving of differential equations with the values of physical quantities or their derivatives given at the boundaries of a specified region. This applies to the solution of the Poisson equation with a given charge distribution and known boundary values of the electrostatic potential or of the stationary Schrödinger equation with a given potential and boundary conditions.

The boundary-value problem is more difficult to solve than the similar initial-value problem with the differential equation. For example, if we want to solve an initial-value problem that is described by the differential equation $u'' = f(u, u'; x)$, with x replaced by time t and the initial conditions $u(0) = u_0$ and $u'(0) = v_0$, we can first transform the differential equation as a set of two first-order differential equations with a redefinition of the first-order derivative as a new variable. The solution will follow if we adopt one of the algorithms discussed earlier in this chapter. However, for the boundary-value problem, we know only $u(0)$ or $u'(0)$, which is not sufficient to start an algorithm for the initial-value problem without some further work.

A typical boundary value problem in physics is a second order differential equation of the form $u'' = f(u, u'; x),$

Instead of initial conditions $u(0), u'(0)$, we sometimes (quite often in fact) know the boundary conditions in pne of the following combinations:

$$(1)\ u(0) = u_0 \text{ and } u(1) = u_1;$$
$$(2)\ u(0) = u_0 \text{ and } u'(1) = v_1;$$
$$(3)\ u'(0) = v_0 \text{ and } u(1) = u_1;$$
$$(4)\ u'(0) = v_0 \text{ and } u'(1) = v_1.$$

We have used the fact that, without a loss of generality, we can assume the boundaries to be at $x = 0$ and $x = 1$. If they are not, we can always transform the coordinates with $x' = (x - x_1)/(x_2 - x_1)$.

One approach is the **shooting method**: make the problem an initial value problem by introducing a parameter and vary this parameter until the solution of the initial value problem satisfies the other boundary condition. Example: we have boundary value type (1) with $u(0) = u_0;\ u(1) = u_1$. The parameter we vary will be $\alpha = u'(0)$. The value of the solution $u_\alpha(1)$ will generally be different from the imposed condition u_1. The task is now to iterate and vary α as long as we achieve the desired accuracy δ: $|u_\alpha(1) - u_1| < \delta$.

ORDINARY DIFFERENTIAL EQUATIONS

Typical eigenvalue problems are even more complicated, because at least one more parameter, that is, the eigenvalue, is involved in the equation: for example,

$$u'' = f(u, u'; x; \lambda),$$

with a set of given boundary conditions, defines an eigenvalue problem. Here the eigenvalue λ can have only some selected values in order to yield acceptable solutions of the equation under the given boundary conditions.

Let us take the longitudinal vibrations along an elastic rod as an illustrative example here. The equation describing the stationary solution of elastic waves is

$$u''(x) = -k^2 u(x),$$

where $u(x)$ is the displacement from equilibrium at x and the allowed values of k^2 are the eigenvalues of the problem. The wavevector k in the equation is related to the phase speed c of the wave along the rod and the allowed angular frequency ω by the dispersion relation

$$\omega = ck.$$

If both ends ($x = 0$ and $x = 1$) of the rod are fixed, the boundary conditions are then $u(0) = u(1) = 0$. If one end ($x = 0$) is fixed and the other end ($x = 1$) is free, the boundary conditions are then $u(0) = 0$ and $u'(1) = 0$. For this problem, we can obtain an analytical solution. For example, if both ends of the rod are fixed, the eigenfunctions

$$u_l(x) = \sqrt{2} \sin k_l x$$

are the possible solutions of the differential equation. Here the eigenvalues are given by

$$k_l^2 = (l\pi)^2,$$

with $l = 1, 2, \ldots, \infty$. The complete solution of the longitudinal waves along the elastic rod is given by a linear combination of all the eigenfunctions with their associated initial-value solutions as

$$u(x, t) = \sum_{l=1}^{\infty} (a_l \sin \omega_l t + b_l \cos \omega_l t) u_l(x),$$

where $\omega_l = ck_l$, and a_l and b_l are the coefficients to be determined by the initial conditions.

We will come back to this problems when studying partial differential equations…

EXAMPLE: LORENZ SYSTEM

$$\frac{\mathrm{d}x}{\mathrm{d}t} = \sigma(y - x),$$

$$\frac{\mathrm{d}y}{\mathrm{d}t} = x(\rho - z) - y,$$

$$\frac{\mathrm{d}z}{\mathrm{d}t} = xy - \beta z.$$

Lorenz equations are a minimalist model of thermal convection in a box. The equations describe a two-dimensional fluid layer uniformly warmed from below and cooled from above: x is proportional to the rate of convection, y to the horizontal temperature variation, and z to the vertical temperature variation. The constants σ, ρ and β are system parameters proportional to the *Prandtl number*, *Rayleigh number*, and physical dimensions of the layer.

The Lorenz equations also arise in simplified models for lasers, dynamos, thermosyphons, brushless DC motors, electric circuits, chemical reactions and forward osmosis. They are the governing equations in Fourier space for the *Malkus waterwheel*. The Malkus waterwheel exhibits chaotic motion where instead of spinning in one direction at a constant speed, its rotation will speed up, slow down, stop, change directions, and oscillate back and forth in an unpredictable manner.

HOMEWORK: Solve the Lorenz system.

- Choose $(x_0, y_0, z_0) = (0,1,0)$ as the initial condition and set the parameter values to $\sigma = 10$, $\rho = 28$ and $\beta = 8/3$.
- Plot separately $x(t)$, $y(t)$ and $z(t)$ and also a 3D plot of the trajectory. Determine a suitable 2D crossection and visualize the solutions on that plane. *Hint: Lorenz found that plotting the solution in the x, z plane is particularly useful.*
- Can you determine the fixed points of the system? *Fixed point is the point in the phase space to which all trajectories or a subset of trajectories converge in time*. How many are there?
- Is the motion periodic? Is the energy (or energy-like constant of motion) of the trajectories conserved? *Hint: the system is dissipative.*
- Change the parameter values and observe how the solution changes.
- How many fixed points are there for $\rho < 1$, and how many for $28 > \rho > 1$? The value $\rho = \rho^{\mathrm{Hopf}} = 28$ is called a **subcritical Hopf bifurcation**, while $\rho = \rho^{\mathrm{BF}} = 1$ is a **pitchfork bifurcation** (try to find additional reference in case you are interested in further theoretical background on dynamic systems…).
- Once you implemented the equations, the solver, and the visualization of the results, enjoy the privilege and **play with the system**! For reference, see below some artwork people have made from the solutions of the Lorenz system (*and many more in the separate pdf uploaded for other chaotic systems exhibiting strange attractors in the phase space*).

Math::Rules

by Chaotic Atmospheres

The Lorenz Attractor

Equations :

$$\frac{dx}{dt} = \sigma(y-x),$$
$$\frac{dy}{dt} = x(\rho-z),$$
$$\frac{dz}{dt} = xy-\beta z.$$

Definitions :

σ, ρ, β = equation parameters

x, y, z = 3D coordinate

t = time

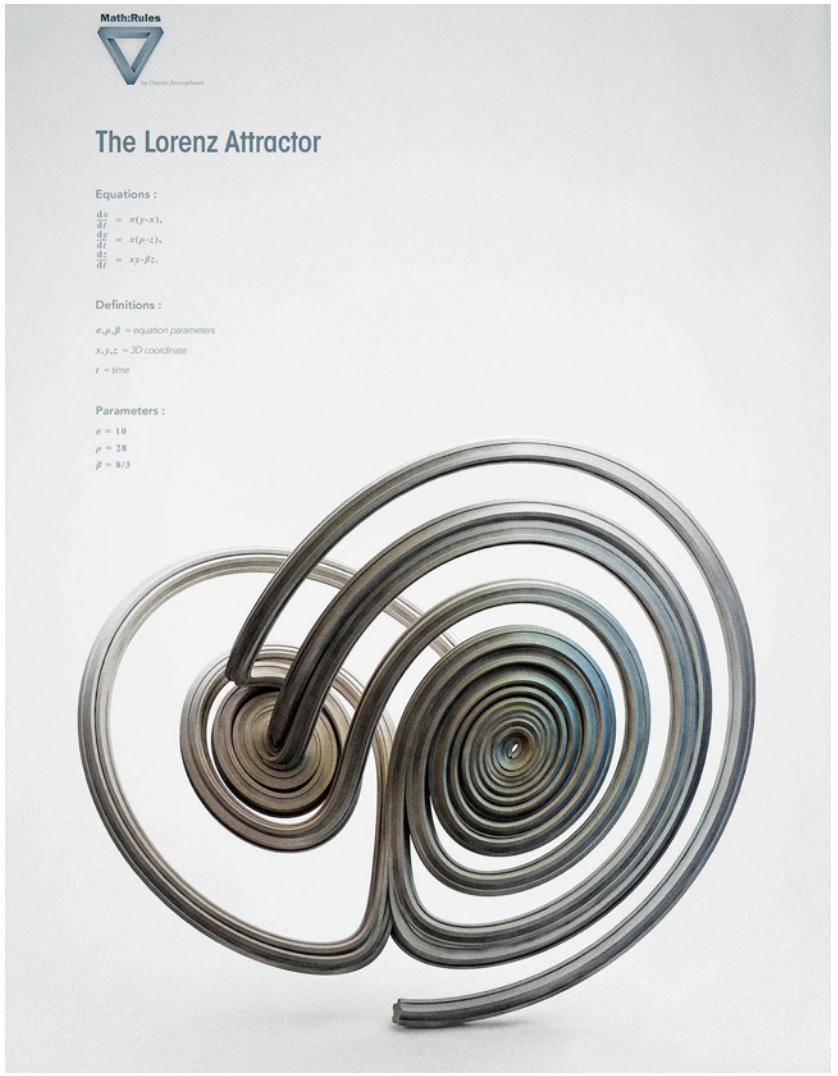
Parameters :

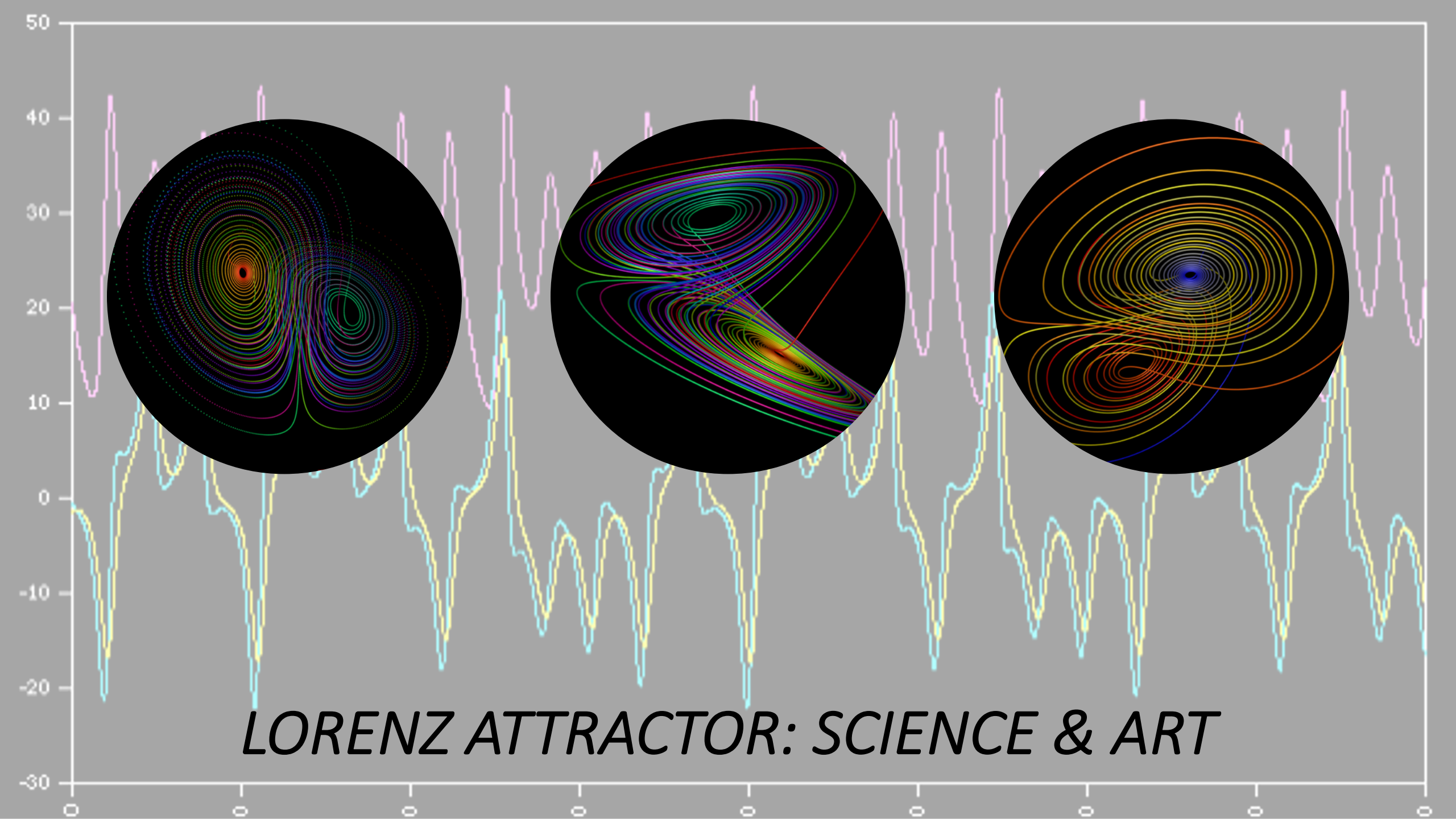
$\sigma = 10$

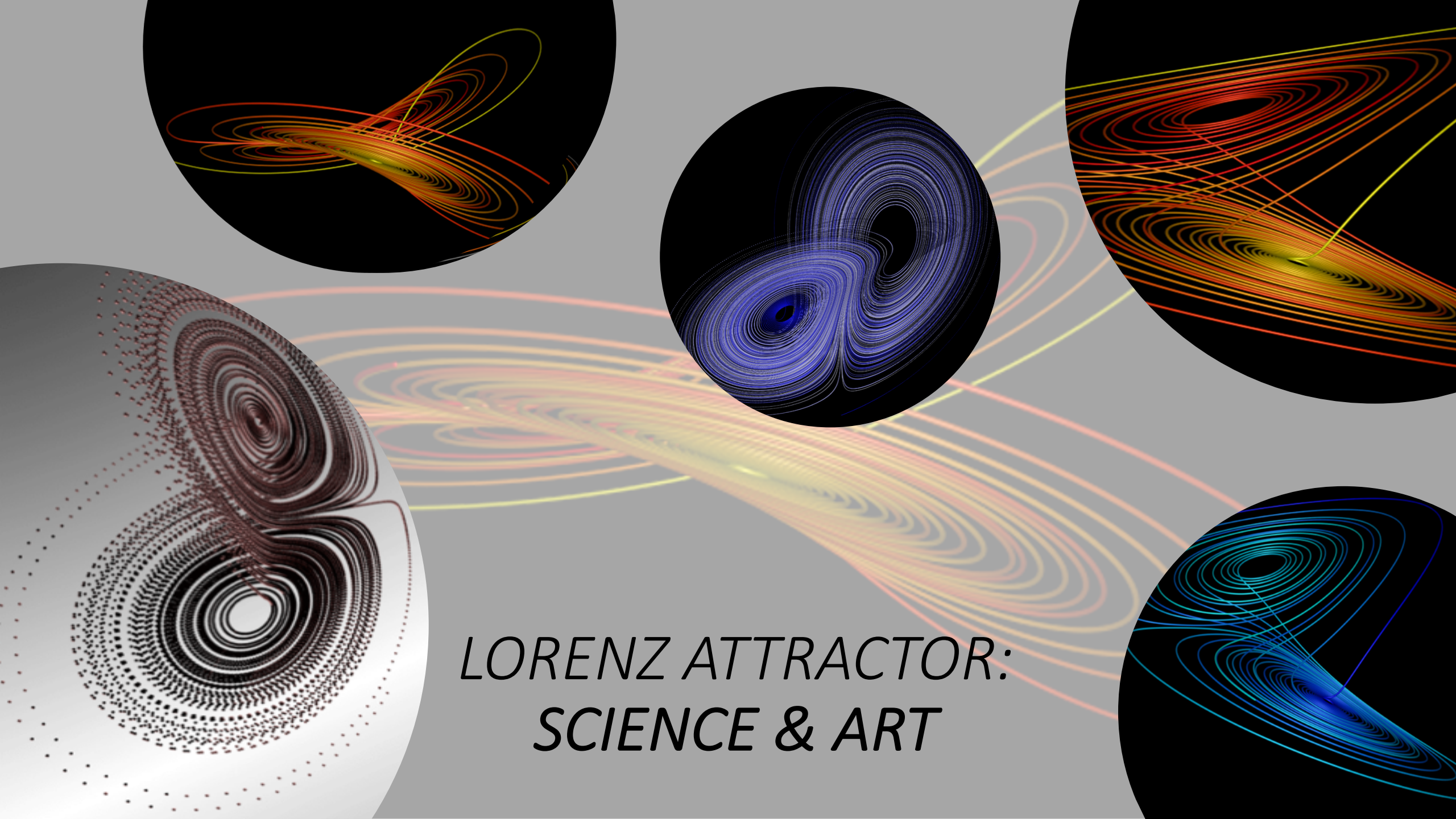
$\rho = 28$

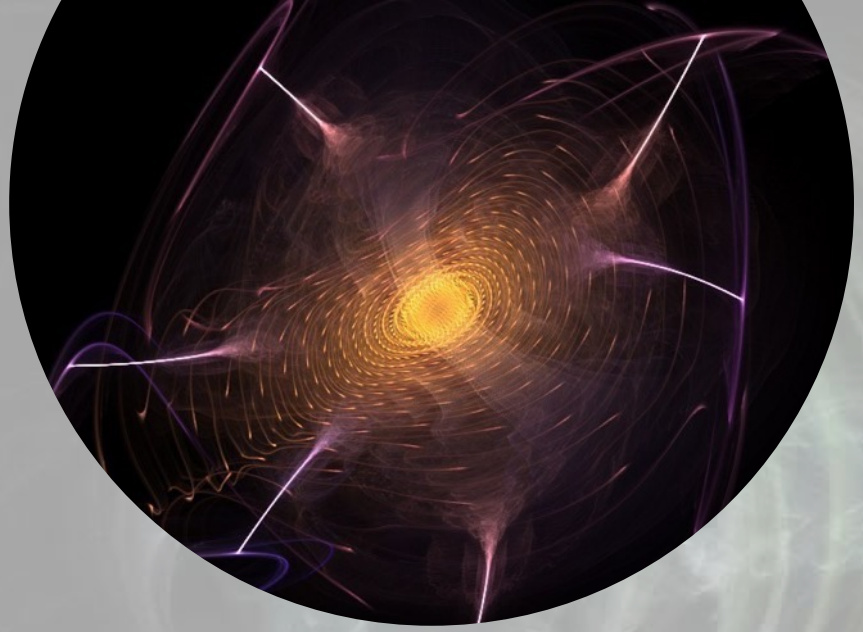
$\beta = 8/3$

LORENZ ATTRACTORS




50
40
30
20
10
0
-10
-20
-30
LORENZ ATTRACTOR: SCIENCE & ART

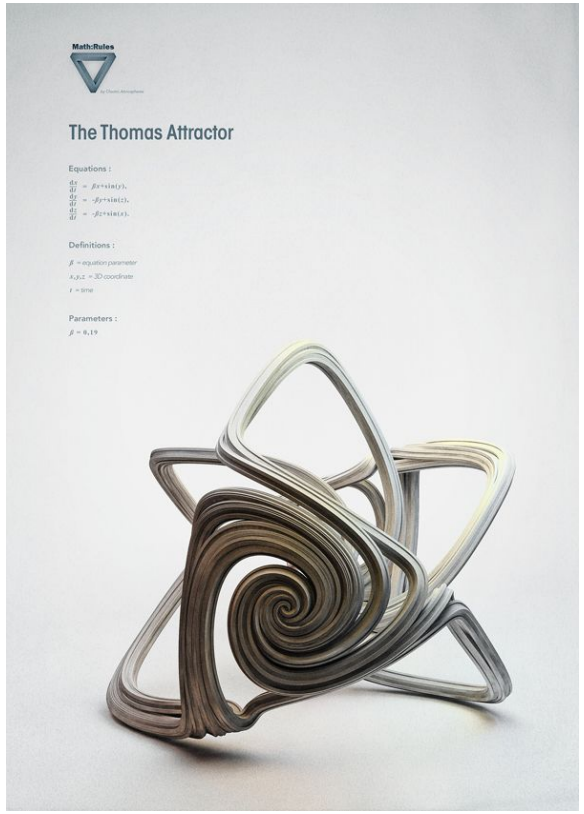

LORENZ ATTRACTOR:
SCIENCE & ART



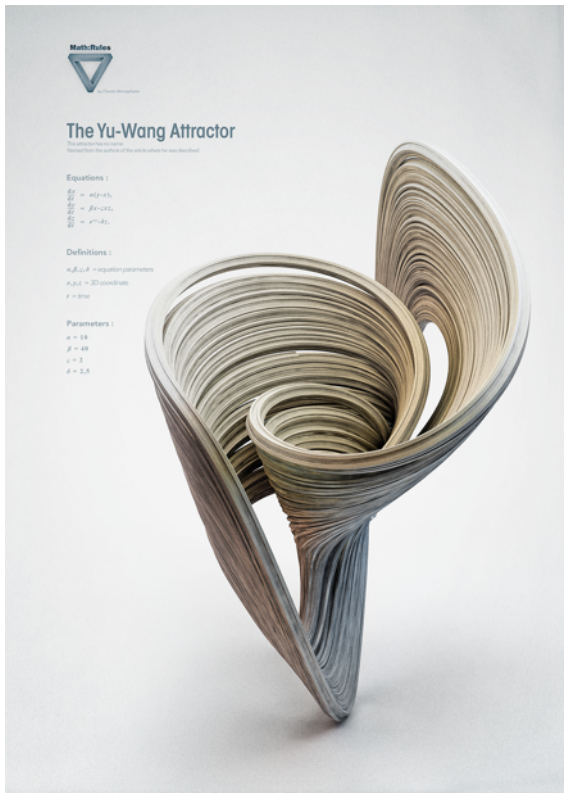
STRANGE ATTRACTORS:
SCIENCE & ART

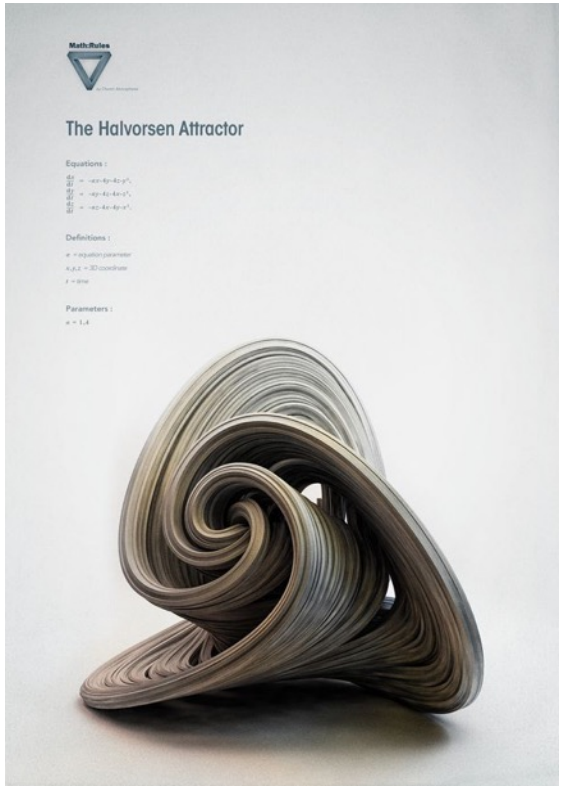
Pickover Algorithm

STRANGE ATTRACTORS

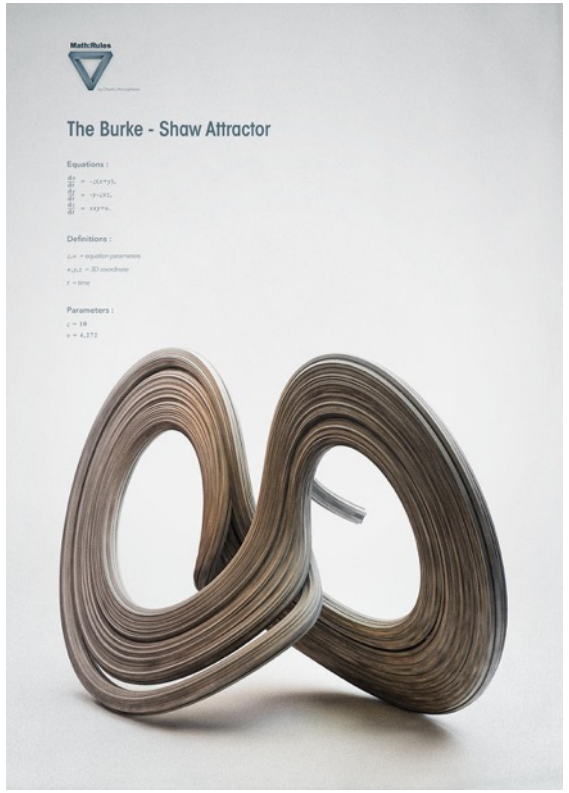


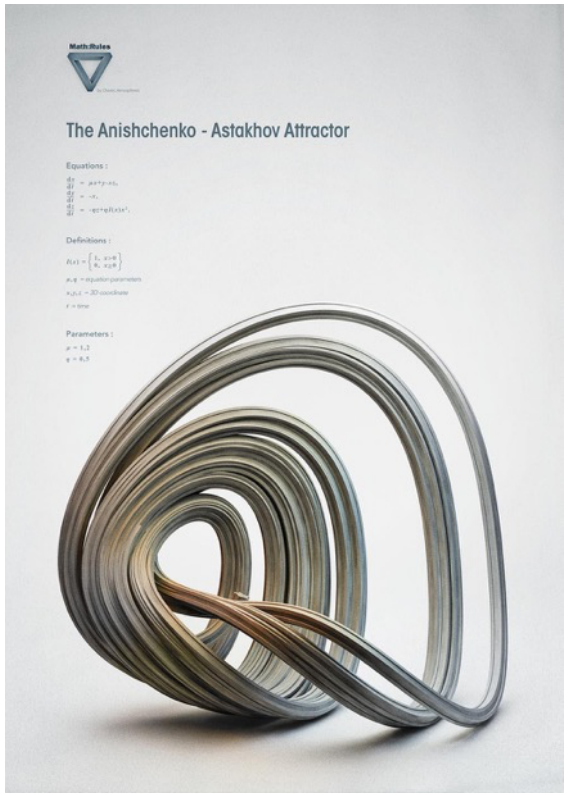
STRANGE ATTRACTORS



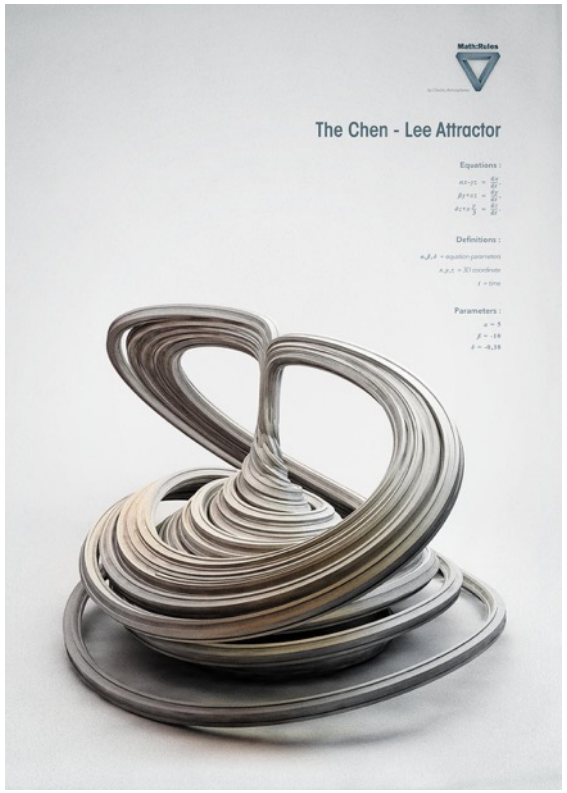


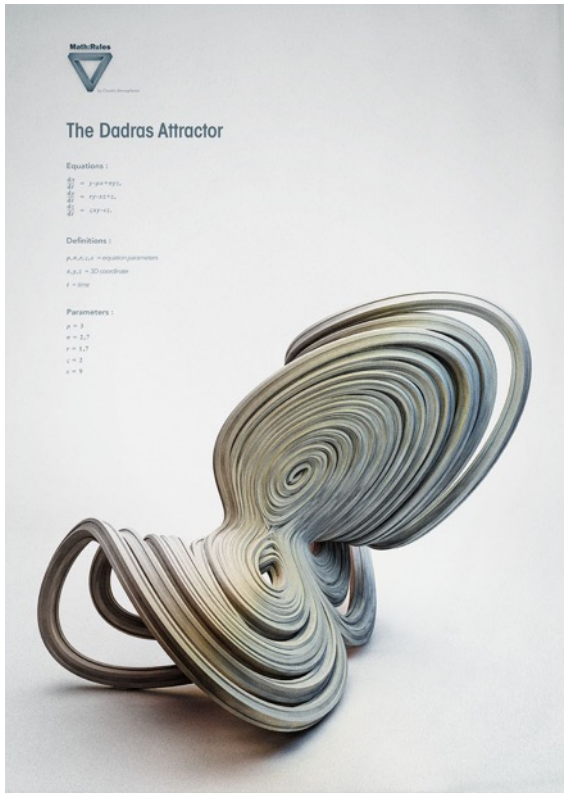
STRANGE ATTRACTORS

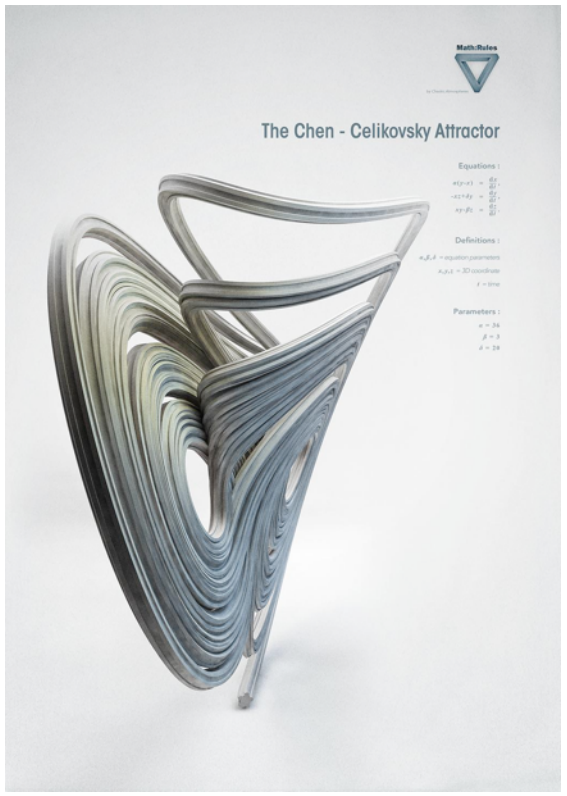




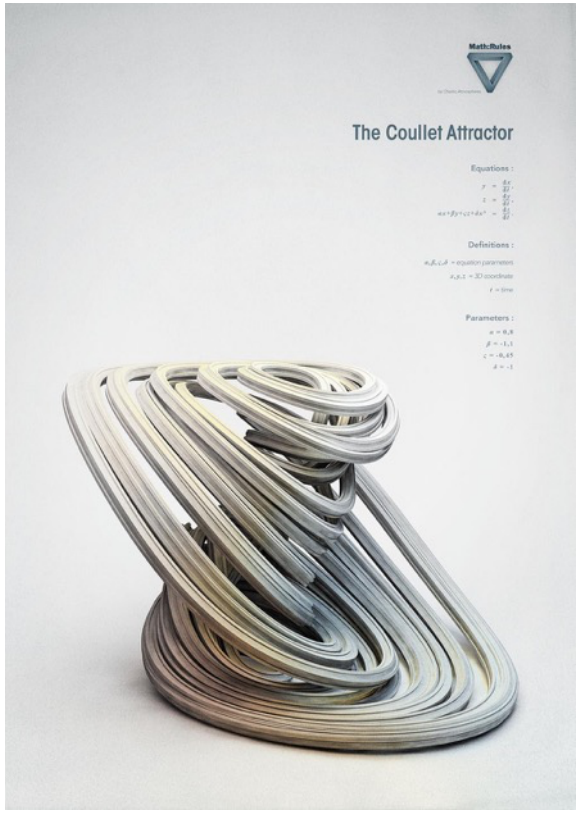
STRANGE ATTRACTORS

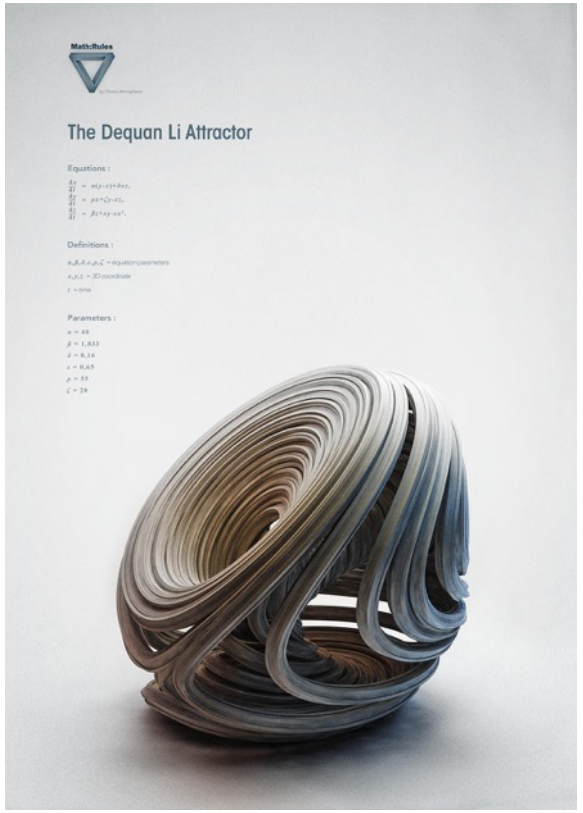




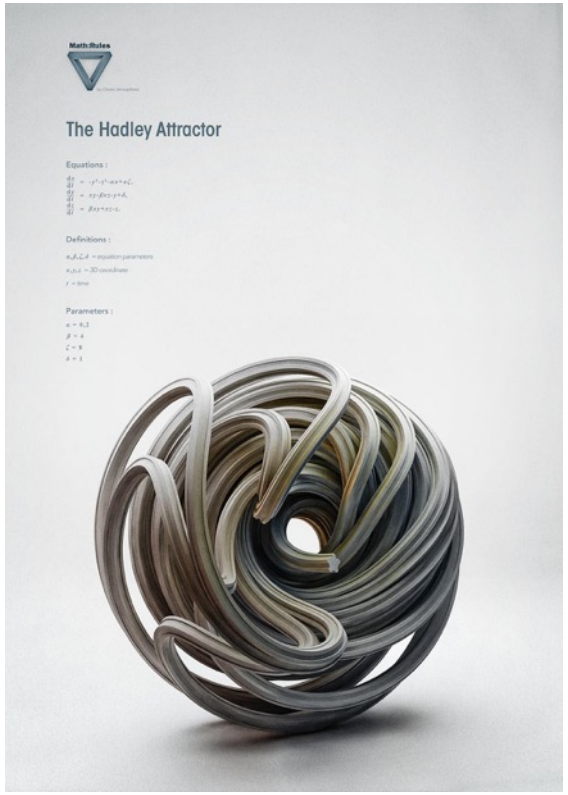


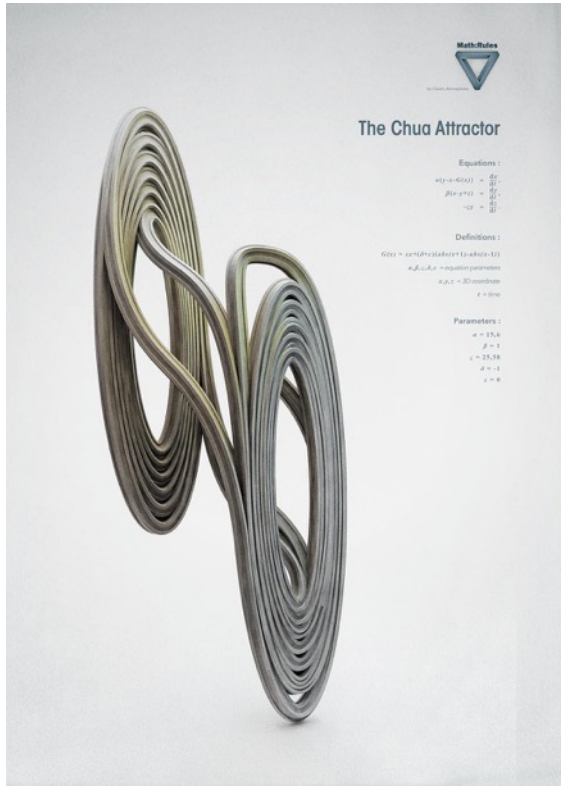
STRANGE ATTRACTORS

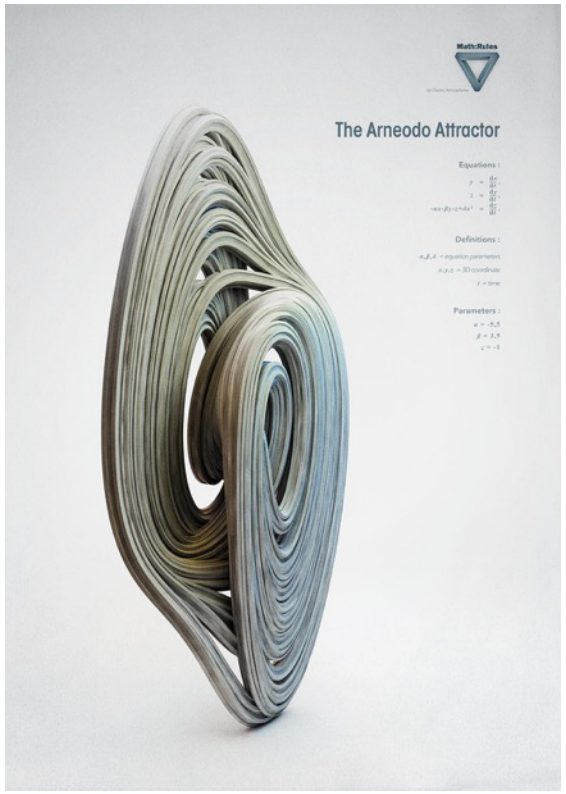




STRANGE ATTRACTORS







STRANGE ATTRACTORS

PARTIAL DIFFERENTIAL EQUATIONS

MANY physical problems are given in the form of **partial differential equations**. If an observable depends on several variables, for instance on space coordinates x, y, z and time t, the laws of physics predict how the observable will change if only one of the variables is slightly modified. For instnce, if u is a function of the coordinate in a 2D space (plane), and $u_0 = u(x_0, y_0)$, its value at a point $\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x_0 + \Delta x \\ y_0 + \Delta y \end{pmatrix}$ for small $\Delta x, \Delta y$ is given by

$$u(x,y) \approx u_0 + \Delta x \left.\frac{\partial u}{\partial x}\right|_{x_0,y_0} + \Delta y \left.\frac{\partial u}{\partial y}\right|_{x_0,y_0}$$

Where $\left.\frac{\partial u}{\partial x}\right|_{x_0,y_0}$ is a partial derivative of u at the point(x_0, y_0), in the direction x. In the following, we will often use the notation $u_{x_i} \equiv \frac{\partial u}{\partial x_i}$, which means for instance: $u_x = \frac{\partial u}{\partial x}$, $u_y = \frac{\partial u}{\partial y}$, $u_{xx} \equiv \frac{\partial^2 u}{\partial x^2}$, $u_{yy} \equiv \frac{\partial^2 u}{\partial y^2}$, and $u_{xy} \equiv \frac{\partial^2 u}{\partial x \partial y}$.

The **gradient** of the function is then $\nabla u \equiv (u_x, u_y)$, and the **Laplacian:** $\Delta u \equiv u_{xx} + u_{yy}$.

Of course, in 3D it is $\Delta u \equiv u_{xx} + u_{yy} + u_{zz}$ and in 4D space-time it is usually defined as $\Box u \equiv \frac{1}{c^2}\frac{\partial^2 u}{\partial t^2} - \Delta u = \frac{1}{c^2} u_{tt} - u_{xx} - u_{yy} - u_{zz}$.

Before we proceed, a revision of the math you know: if the coordinates implicitly depend on other functions, e.g. $x = x(r, p)$, $y = y(r, p)$ and $z = z(r, p)$, then the partial derivatives over r or p are calculated by applying the chain rule:

$$u_r \equiv \frac{\partial u}{\partial r} = \frac{\partial u}{\partial x}\frac{\partial x}{\partial r} + \frac{\partial u}{\partial y}\frac{\partial y}{\partial r} = u_x x_r + u_y y_r$$

$$u_p \equiv \frac{\partial u}{\partial p} = u_x x_p + u_y y_p$$

$$u_{rr} = u_{xx} {x_r}^2 + 2\, u_{xy} x_r y_r + u_{yy} {y_r}^2 + u_x\, x_{rr} + u_y\, y_{rr}$$

$$u_{rp} = u_{xx} x_r x_p + u_{xy}\left(x_r y_p + x_p y_r\right) + u_{yy} x_p y_r + u_x x_{rp} + u_y y_{rp}$$

$$u_{pp} = u_{xx} {x_p}^2 + 2\, u_{xy} x_p y_p + u_{yy} {y_p}^2 + u_x\, x_{pp} + u_y\, y_{pp}$$

Two examples of the Laplacian ($\Delta u \equiv u_{xx} + u_{yy} + u_{zz}$) in transformed coordinates.

2D POLAR COORDINATES: $r = \sqrt{x^2 + y^2}$; $x = r\cos\varphi$ $y = r\sin\varphi$

$u_r = u_x \cos\varphi + u_y \sin\varphi$ $u_\varphi = -u_x r \sin\varphi + u_y r \cos\varphi$

$u_{rr} = u_{xx}\cos^2\varphi + 2\,u_{xy}\sin\varphi\cos\varphi + u_{yy}\sin^2\varphi$

$u_{\varphi\varphi} = u_{xx}r^2\sin^2\varphi - 2\,u_{xy}r^2\sin\varphi\cos\varphi + u_{yy}r^2\cos^2\varphi - u_x\, r\cos\varphi - u_y\, r\sin\varphi$

$$\Delta u = u_{xx} + u_{yy} = u_{rr} + \frac{1}{r}u_r + \frac{1}{r^2}u_{\varphi\varphi} = \frac{1}{r}\frac{\partial}{\partial r}(r\,u_r) + \frac{1}{r^2}u_{\varphi\varphi}$$

3D SPHERICAL COORDINATES: $r = \sqrt{x^2 + y^2 + z^2}$; $x = r\cos\theta\cos\varphi$ $y = r\cos\theta\sin\varphi$ $z = r\sin\theta$

$$\Delta u = u_{xx} + u_{yy} + u_{zz} = \frac{1}{r^2}\frac{\partial}{\partial r}(r^2\,u_r) + \frac{1}{r^2\sin\theta}\frac{\partial}{\partial\theta}(\sin\theta\,u_\theta) + \frac{1}{r^2\sin^2\theta}u_{\varphi\varphi}$$

PARTIAL DIFFERENTIAL EQUATIONS

Before we proceed, we need to agree on what we mean by a solution of a partial differential equation. We will show that a solution of an equation is a broad family of functions that satisfy the symmetry or scaling requirements of the equation. Only after we include the boundary conditions, can we find out the actual functional dependence of the solution!

Let us illustrate this with some examples. Consider three functions:

$$u_1(x,y) = x^4 + 4(x^2y + y^2 + 1)$$
$$u_2(x,y) = \sin x^2 \cos 2y + \cos x^2 \sin 2y$$
$$u_3(x,y) = \frac{x^2 + 2y + 2}{3x^2 + 6y + 5}$$

On a first glance, these are complicated functions and it would be very difficult to conclude that they are all solutions of the same partial differential equation. However, we observe that each of these functions can be expressed in terms of a single variable $p = x^2 + 2y$ (with no other x or y involved):

$$u_1(x,y) = p^2 + 4 \equiv f_1(p)$$
$$u_2(x,y) = \sin p \equiv f_2(p)$$
$$u_3(x,y) = \frac{p+2}{3p+5} \equiv f_3(p)$$

We can calcuate the partial derivatives as $\frac{\partial u_i}{\partial x} = 2x f_i'$ and $\frac{\partial u_i}{\partial y} = 2f_i'$ for $i = 1, 2, 3$. Thus, for all three functions the following is true: $\frac{\partial u_i}{\partial x} = x\frac{\partial u_i}{\partial y}$. Therefore, all three functions are solutions of the following PDE:

$$\frac{\partial u}{\partial x} - x\frac{\partial u}{\partial y} = 0$$

Moreover, any arbitrary function $f(p)$ for which ther argument has the form $p = x^2 + 2y$ is a solution of this PDE!

Another example: find the solution of the first order differential equation $y\frac{\partial u}{\partial x} - x\frac{\partial u}{\partial y} = 3x$, which satisfies the boundary condition $u(x,0) = x^2$.

First, we notice that any function $f(x^2 + y^2)$ solves the homogeneous part of the equation since $y\ 2xf' - x\ 2yf' = 0$. We can also guess a particular integral that solves the original equation: $u_p(x,y) = -3y$. Therefore, the general solution of the PDE is $u(x,y) = f(x^2 + y^2) - 3y$ with arbitrary function f. Now, the boundary condition requires that $u(x,0) = x^2 = f(x^2)$, *i.e.*, $f(p) = p$.

Therefore, the solution of the PDE with the boundary condition is $u(x,y) = x^2 + y^2 - 3y$.

This was an example of a first-order PDE that (I hope) nicely demonstrates that only a combination of the PDE and the boundary condition completely define the solution. Here the boundary condition was given along a line in 2D. If it was given at a single point, the problem would be underdetermined and we would still have an arbitrary component in the solution. Imagine that the bouncondition would be $u(1,0) = 2$. That would imply $f(1) = 2$, which could be the case if we choose for instance $f(p) = 2$. The solution in this case would be $u(x,y) = 2 - 3y + g(x^2 + y^2)$ where $g$is an arbitrary function…dary

The general form we are going to discuss, is the second order PDE, which is by far the most common type of differential equations in physics: $A\frac{\partial^2 u}{\partial x^2} + B\frac{\partial^2 u}{\partial x \partial y} + C\frac{\partial^2 u}{\partial y^2} + f\left[u, x, y, \frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}\right] = 0$. For simplicity, let's first look at a homogeneous equation with constant coefficients:

$$Au_{xx} + Bu_{xy} + Cu_{yy} = 0$$

Function $u(x,y) = f(x + \lambda_1 y) + g(x + \lambda_2 y)$, with arbitrary f and g solves the equation, if $\lambda_{1,2} = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}$.

- For $B = 0, A = C = 1$, we recognize the **Laplace equation**: $\Delta u = 0$. The solution is $u(x,y) = f(x + iy) + g(x - iy)$.
- For $B = 0, A = \frac{1}{c^2}, C = -1$, we have a **wave equation:** $\frac{1}{c^2}u_{xx} - u_{yy} = 0$, and $u(x,y) = f(cx - y) + g(cx + \mathrm{y})$, i.e., waves traveling to the right and to the left (x is a time variable here).

As stressed many times above, the functional form of the solution must come from the boundary conditions.

PARTIAL DIFFERENTIAL EQUATIONS

The general form of linear second order PDE, which is by far the most common type of differential equations in physics:

$$A\frac{\partial^2 u}{\partial x^2} + B\frac{\partial^2 u}{\partial x \partial y} + C\frac{\partial^2 u}{\partial y^2} + f\left[u, x, y, \frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}\right] = 0$$

In order to classify the equations, we usually analyze the ***homogeneous part***, $A\frac{\partial^2 u}{\partial x^2} + B\frac{\partial^2 u}{\partial x \partial y} + C\frac{\partial^2 u}{\partial y^2} = 0$, separately from the ***particular (source) term*** f. Three major types of equastions (depending on the values of functions A, B, C) are:

$$B^2 - 4AC \begin{cases} > 0 & \textbf{HYPERBOLIC} \\ < 0 & \textbf{ELLIPTIC} \\ = 0 & \textbf{PARABOLIC} \end{cases}$$

Wave equation

$$\frac{1}{c^2}\frac{\partial^2 u(\boldsymbol{r},t)}{\partial t^2} - \nabla^2 u(\boldsymbol{r},t) = -R(\boldsymbol{r},t)$$

Laplace equation, Poisson equation

$$\frac{\partial^2 u(\boldsymbol{r},t)}{\partial x^2} + \frac{\partial^2 u(\boldsymbol{r},t)}{\partial y^2} + \frac{\partial^2 u(\boldsymbol{r},t)}{z^2} = -\frac{\rho(\boldsymbol{r})}{\varepsilon_0}$$

Heat / Diffusion equation

$$\frac{\partial u(\boldsymbol{r},t)}{\partial t} - \nabla \cdot D(\boldsymbol{r}) \nabla u(\boldsymbol{r},t) = S(\boldsymbol{r},t)$$

Also **Schrödinger equation**

$$i\hbar\frac{\partial \psi(\boldsymbol{r},t)}{\partial t} + \frac{\hbar^2}{2m}\nabla^2 \psi(\boldsymbol{r},t) = V\psi(\boldsymbol{r},t)$$

If we introduce imaginary time vasriable $t \to i\tau$, the equation becomes

$$\frac{\partial \psi(\boldsymbol{r},t)}{\partial \tau} + \frac{\hbar^2}{2m}\nabla^2 \psi(\boldsymbol{r},\tau) = \frac{V}{\hbar}\psi(\boldsymbol{r},\tau)$$

which is formally equivalent to the diffusion equation with $-\frac{\hbar}{2m} \leftrightarrow D$, $\frac{V}{\hbar} \leftrightarrow S$.

PARTIAL DIFFERENTIAL EQUATIONS

The general form of linear second order PDE, which is by far the most common type of differential equations in physics:

$$A\frac{\partial^2 u}{\partial x^2} + B\frac{\partial^2 u}{\partial x \partial y} + C\frac{\partial^2 u}{\partial y^2} + f\left[u, x, y, \frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}\right] = 0$$

In order to classify the equations, we usually analyze the ***homogeneous part***, $A\frac{\partial^2 u}{\partial x^2} + B\frac{\partial^2 u}{\partial x \partial y} + C\frac{\partial^2 u}{\partial y^2} = 0$, separately from the ***particular (source) term*** f. Three major types of equastions (depending on the values of functions A, B, C) are:

$$B^2 - 4AC \begin{cases} > 0 & \textbf{HYPERBOLIC} \\ < 0 & \textbf{ELLIPTIC} \\ < 0 & \textbf{PARABOLIC} \end{cases}$$

Wave equation

$$\frac{1}{c^2}\frac{\partial^2 u(\boldsymbol{r},t)}{\partial t^2} - \nabla^2 u(\boldsymbol{r},t) = -R(\boldsymbol{r},t)$$

Laplace equation, Poisson equation

$$\frac{\partial^2 u(\boldsymbol{r},t)}{\partial x^2} + \frac{\partial^2 u(\boldsymbol{r},t)}{\partial y^2} + \frac{\partial^2 u(\boldsymbol{r},t)}{z^2} = -\frac{\rho(\boldsymbol{r})}{\varepsilon_0}$$

Heat / Diffusion equation

$$\frac{\partial u(\boldsymbol{r},t)}{\partial t} - \nabla \cdot D(\boldsymbol{r}) \nabla u(\boldsymbol{r},t) = S(\boldsymbol{r},t)$$

Also **Schrödinger equation**

$$i\hbar\frac{\partial \psi(\boldsymbol{r},t)}{\partial t} + \frac{\hbar^2}{2m}\nabla^2 \psi(\boldsymbol{r},t) = V\psi(\boldsymbol{r},t)$$

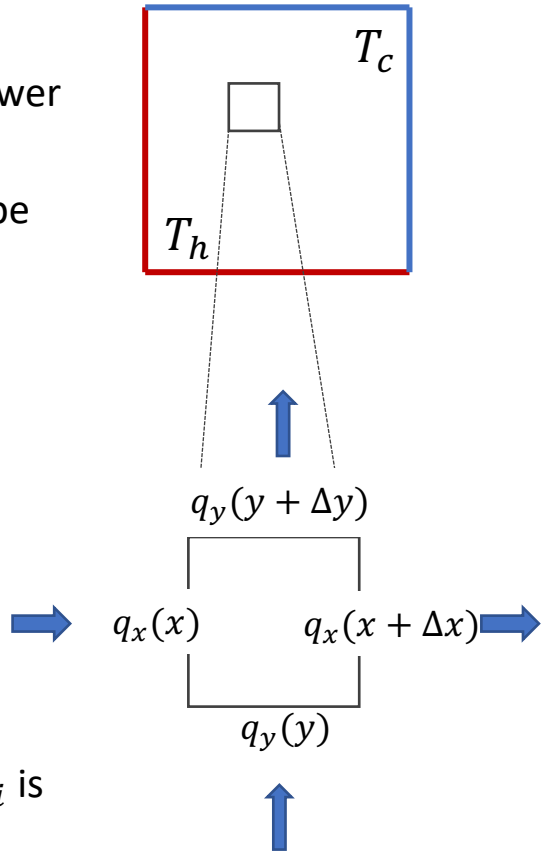
If we introduce imaginary time vasriable $t \to i\tau$, the equation becomes

$$\frac{\partial \psi(\boldsymbol{r},t)}{\partial \tau} + \frac{\hbar^2}{2m}\nabla^2 \psi(\boldsymbol{r},\tau) = \frac{V}{\hbar}\psi(\boldsymbol{r},\tau)$$

which is formally equivalent to the diffusion equation with $-\frac{\hbar}{2m} \leftrightarrow D$, $\frac{V}{\hbar} \leftrightarrow S$.

Heat flow in a slab

Imagine a slab of material of thickness Δz that is insulated at the top and bottom (in z direction). If we apply a temperature gradient, the heat will flow from hot to cold. Since the slab is insulated, the heat flow will be two-dimensional. Imagine that we are keeping two sides at a higher temperature T_h and the other two sides at a lower temperature T_c.



Let's look at the **steady state.** The balance of the heat flow q in and out of a small square of size $\Delta x \times \Delta y$ must be zero. The heat is coming *in* from left and from below and leaks *out* to the right and above: $q_x(x)\Delta y\Delta z\Delta t + q_y(y)\Delta x\Delta z\Delta t = q_x(x+\Delta x)\Delta y\Delta z\Delta t + q_y(y+\Delta y)\Delta x\Delta z\Delta t$, from which it follows that

$$\frac{q_x(x+\Delta x) - q_x(x)}{\Delta x} + \frac{q_y(y+\Delta y) - q_y(y)}{\Delta y} = 0$$

i.e.,

$$\frac{\partial q_x}{\partial x} + \frac{\partial q_y}{\partial y}$$

The fluxes are connected to the temperature gradients by the Fourier's law: $q_i = -k\rho C\frac{\partial u}{\partial x_i}$ (here u is the temperature, $\rho$is the density of the material, k the coefficient of thermal diffusivity and C the heat capacity; x_i is the direction x or y). So finally this results in the equation for the steady state temperature profile:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$$

This is the Laplace equation. It very often describes the steady state profiles in physics and engineering. We could of course have derived it from the heat equation $u_{xx} + u_{yy} = Ku_t$ by assuming that the time derivative $u_t = 0$.

PARTIAL DIFFERENTIAL EQUATIONS

DISCRETIZATION (FINITE DIFFERENCE METHOD)

Let us use the discretization approach we have learned for numerical derivation and replace the derivatives in the Laplace equation with differences:

$$\frac{\partial u(\boldsymbol{r},t)}{\partial t} = \frac{u(\boldsymbol{r},t_{k+1}) - u(\boldsymbol{r},t_{k-1})}{2\Delta t}$$

$$\frac{\partial^2 u(\boldsymbol{r},t)}{\partial t^2} = \frac{u(\boldsymbol{r},t_{k+1}) - 2u(\boldsymbol{r},t_k) + u(\boldsymbol{r},t_{k-1})}{\Delta t^2}$$

$$\frac{\partial u(\boldsymbol{r},t)}{\partial x} = \frac{u(x_{i+1},y,z,t) - u(x_{i-1},y,z,t)}{2\Delta x}$$

$$\frac{\partial^2 u(\boldsymbol{r},t)}{\partial x^2} = \frac{u(x_{i+1},y,z,t) - 2u(x_i,y,z,t) + u(x_{i-1},y,z,t)}{\Delta x^2}$$

$$\frac{\partial^2 u(\boldsymbol{r},t)}{\partial y^2} = \frac{u(x,y_{j+1},z,t) - 2u(x,y_j,z,t) + u(x,y_{j-1},z,t)}{\Delta y^2}$$

As we argued, there is no z dependence in the problem, so we can skip the z variable. We can also skip the time variable since we are now interested in the steady state. Let us decide that the lattice spacing will be uniform, therefore $\Delta x = \Delta y \equiv h$.

The equation then becomes

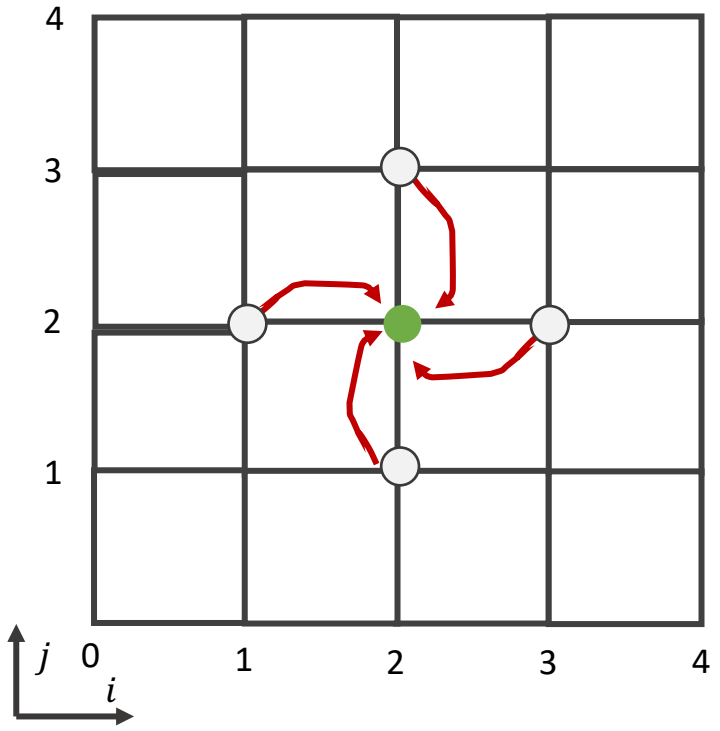
$$u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{i,j} = 0 \qquad \text{(for } i,j \text{ from } 0..N\text{)}$$

If we discretize the system into a $N \times N$ lattice, we can use the difference equation to calculate the values at each lattice point from the values at the neighbouring points.

Here is an example of a 4×4 lattice. The value at the point u_{22} can be obtained from the four neighbours as

$$u_{22} = \tfrac{1}{4}[u_{12}+u_{32}+u_{21}+u_{23}]$$

Clearly, this expression cannot be applied to any point at the outer boundary that does not have the four neighbours. For these **boundary points**, we need to specify the value through the **boundary conditions**. Two most common types of boundary conditions:

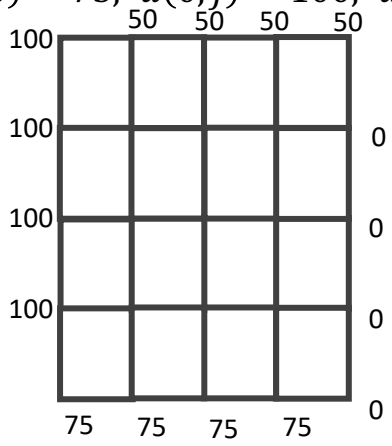


DIRICHLET: Fix the value of u at the boundary

VON NEUMANN: Fix the derivatives of u at the boundary

PARTIAL DIFFERENTIAL EQUATIONS

Let us assume Dirichlet boundary conditions, for instance: $u(i,0) = 75,\ u(0,j) = 100,\ u(i,4) = 50,$ $u(4,j) = 0.$



With these values fixed, we can write the equations for the inner points:

$u_{0,1}$ $u_{1,0}$

(1,1): $u_{2,1} + 100 + u_{1,2} + 75 - 4u_{1,1} = 0$

(2,1): $u_{3,1} + u_{1,1} + u_{2,2} + 75 - 4u_{2,1} = 0$

(3,1): $0 + u_{2,1} + u_{3,2} + 75 - 4u_{3,1} = 0$

(1,2): $u_{2,2} + 100 + u_{1,3} + u_{1,1} - 4u_{1,2} = 0$

(2,2): $u_{3,2} + u_{1,2} + u_{2,3} + u_{2,1} - 4u_{2,2} = 0$

(3,2): $0 + u_{2,2} + u_{3,3} + u_{3,1} - 4u_{3,2} = 0$

(1,3): $u_{2,3} + 100 + 50 + u_{1,2} - 4u_{1,3} = 0$

(2,3): $u_{3,3} + u_{1,3} + 50 + u_{2,2} - 4u_{2,3} = 0$

(3,3): $0 + u_{2,3} + 50 + u_{3,2} - 4u_{3,3} = 0$

We have a system of 9 linear equations for 9 variables. If we organize the values $u_{i,j}$ as a vector $\boldsymbol{U} = \begin{pmatrix} u_{1,1} \\ u_{2,1} \\ \vdots \\ u_{1,2} \\ \vdots \\ u_{3,3} \end{pmatrix}$ the equations can be written in the matrix form: $\mathrm{A}\,\boldsymbol{U} + \mathrm{B} = 0$

$$\begin{bmatrix} -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -4 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & -4 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & -4 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & -4 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & -4 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 \end{bmatrix} \begin{pmatrix} u_{1,1} \\ u_{2,1} \\ u_{3,1} \\ u_{1,2} \\ u_{2,2} \\ u_{3,2} \\ u_{1,3} \\ u_{2,3} \\ u_{3,3} \end{pmatrix} + \begin{pmatrix} 175 \\ 75 \\ 75 \\ 100 \\ 0 \\ 0 \\ 150 \\ 50 \\ 50 \end{pmatrix} = 0$$

In principle, we could invert the matrix and solve as $\boldsymbol{U} = -\mathrm{A}^{-1}\boldsymbol{B}$. However, we notice that the matrix A is **sparse**, this means that the elements A_{ij} are mostly equal to zero. In the serious calculation, we would have not only 9×9 but perhaps $10^4 \times 10^4$ or $10^4 \times 10^4 \times 10^4$ matrices and we would be adding up a lot of zeros!

The inversion of sparse matrices is a very inneficient operation. Therefore, the direct inversion is never attempted when solving PDEs computationally.

If not by inversion, how do we do it? One of the commonly used methods is the relaxation method, specifically the succsessive over-relaxation (SOR):

- instead of writing down all equations and trying to solve them in one go, we start by an initial guess, which could be any functional that satisfies the boundary conditions, so we assign some values u_{ij} to all the lattice points.
- **The initial guess is very important for efficiency of the method. We can usually apply physical knowledge/intuition to get a good guess.**
- Then we update values point-by-point using the equations above but with a following scheme:
  - calculate the quantity $s_{ij} = \frac{1}{4}(u_{i+1,j}+u_{i-1,j}+u_{i,j+1}+u_{i,j-1})$
  - in principle, s_{ij} should be equal u_{ij}; if this is achieved at every point, we have solved the original equation!
  - but since we have started with a guess that is not the true solution, s_{ij} will be different from u_{ij}
  - then we update the value at the lattice point i, j as

$$u_{ij}^{NEW} = (1-p)u_{ij}^{OLD} + p\, s_{ij}$$

  - we have introduced the over-relaxation parameter p.
  - the value of p is something that we need to optimize.
  - PDEs are not all stable for all values of p
  - the speed of the method depends on p and the optimal parameter varies from problem to problem.
  - for instance, for nonlinear 3D Poisson-Boltzmann equation the optimal choice is $p^{optimal} \approx 1.3$.

SOR is usually a pretty good method – and it is very easy to implement. However, nowadays there are other faster methods such as multigrid methods or finite elements methods, which are particularly useful in complex geometries…

Homework (not compulsary): Try solving the Poisson-Boltzmann equation (PBE) with SOR method.

PBE describes the distribution of charged microions around a bigger charged particle (colloid). Due to the large assymmetry in size, the distribution of the ions can be assumed to adiabatically follow the position of the colloid (the ensemble of microions is in a local thermodynamic equilibrium for any configuration of the colloids. We can describe the ions as a density (and charge) field and in the mean field approach, the distribution of the ions follows a Boltzmann distribution (remember statistical physics). If we assume the first-order approximation, where the microions are point particles not interacting with each other, their distribution is described by the Poisson equation combined with the Boltzmann distribution for the ideal gas of ions:

$$\Delta\varphi(\boldsymbol{r}) = \kappa^2 \sinh\varphi\,(\boldsymbol{r})$$

Here $\varphi = eV/kT$ is the dimensionless electrostatic potential and κ is a parameter depending on salt and counterions concentration. Besides the PBE, we need to satisfy the boundary condition on the surface of the particle, which comes from requirement that the entire system is electro-neutral: $\nabla V|_{colloid\ surface} = \frac{4\pi\sigma}{\varepsilon}$, with σ the charge density on the colloid surface, and ε the dielectric constant. In dimensionless units it becomes

$$\nabla\varphi\Big|_{colloid\ surface} = Z$$

Where Z is a measure for the charge on the colloid.

Practical hints. Assume the following parameters: κ=1, Z=100. Take a 3D system with the size equal 4. Colloid diameter is 1. Discretize the system with a mesh of $30\times30\times30$ points. Define the points around the colloids as boundary points (your colloid will not be really spherical!). Perform the SOR with the over-relaxation parameter $p = 1.3$. Assume the following function as a first guess: $\varphi(r) = Z\frac{e^{-\kappa r}}{r}$, where r is the radial distance of the point from the colloid center. If you manage to program this, try changing the parameters and see hoe the solution changes and also how the computational effort increases with increasing Z – making the equation more nonlinear…

In case the problem turns out too demanding, try a 2D problem.

RANDOM NUMBERS IN COMPUTATION

Why do we need random numbers?

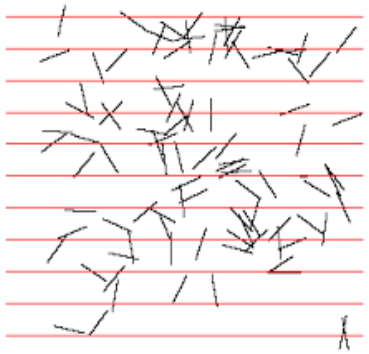
→ **Simulation of random processes** *(nuclear decay, Brownian motion, reactions, weather…, **noise**)*

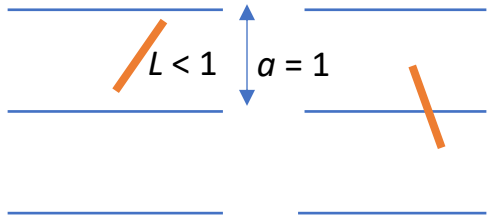
→ **Probability** *(using randomness to **test theories** or **estimate constants**…)*

EXAMPLE 1: **Buffon needle**

(G.L. leclerc, Comte de Buffon, 1773)

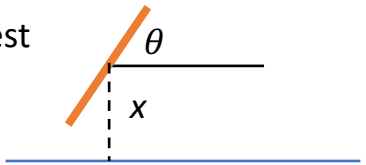
- We toss a needle of length *L* randomly to a horizontal surface ruled with equally spaced lines ($a = 1$).
- What is the probability of the needle crossing the line?



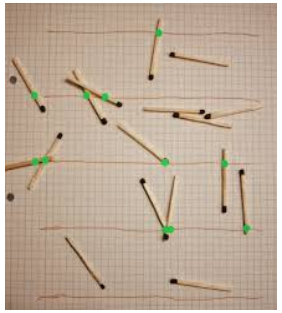


Solution:

- The needle tossed at random will land so that its center is a distance x to the closest line, so that $0 \leq x \leq \frac{1}{2}$.
- The angle of the needle against the lines is $0 \leq \theta \leq \pi$.
- Since the needle is "tossed at random", the probability distributions for x and θ are uniform.
- A crossing occurs if $x \leq \frac{L}{2}\sin\theta$.
- The probability for this to be tha case is given by:



$$p = \frac{\int_0^{\pi} \frac{L}{2}\sin\theta\, d\theta}{\int_0^{\pi} \frac{1}{2} d\theta} = \frac{2L}{\pi}$$



- → experimentally determine the value of π ($\pi \approx 3$ easy)
- Reaching accuracy $\pi \approx 3.14$ is extremely difficult. **Would you like to try at home?**
- Versions of method are used in *e.g.,* estimating the amount of "grain boundary" in micrographs

EXAMPLE 2: **Estimating *e***

(Roger Pinkham, ~ 1960)

- We draw $2k$ numbers $\{x_1, \cdots, x_{2k}\}$ from a random source
- $2k$ numbers x_i have $(2k)!$ possible orderings
- Only one of these orderings will be in ascending order
- The probability that the next number, x_{2k+1}, will not be in ascending order is

$$\frac{1}{(2k)!} - \frac{1}{(2k+1)!}$$

- The total probability that a sequence of an even number of random numbers will be ascending is

$$\sum_{k=1}^{\infty}\left[\frac{1}{(2k)!} - \frac{1}{(2k+1)!}\right] = \frac{1}{e}$$

RANDOM NUMBERS

Sources of random numbers

- **Natural processes** (cosmic rays, nuclear disintegration, random sampling of alternate frequency…)

Devices have been built and used based on similar

Problem: not possible to repeat events…

- **Published tables of random digits**

e.g.
Rand Table of One Million Random Digits and One Hundred Thousand Normal Derivatives,
Free Press Chicago, 1955

Problems:
- Slow to read from a tape or disk
- 1 million is not nearly enough for modern simulations…

- **Creating longer table by manipulating published tables**

Problems:
- not elegant
- still slow

- **Mathematical formulas** (many versions, more or less complex and successful)

- On a computer, we **cannot** generate a really random set of numbers! (calculations are deterministic)
- We **can** generate a **pseudo-random sequence**, which has a **long periodicity interval**

RANDOM NUMBERS

EXAMPLE: **Congruential generators**

Define a sequence (s_i) recurrently

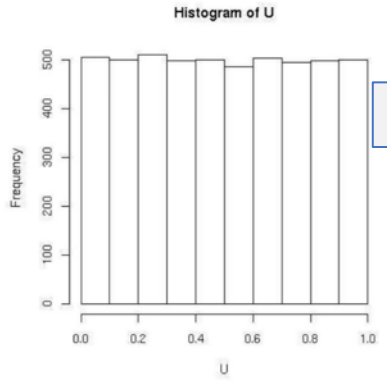
$$s_{i+1} = (as_i + b) \mod M.$$

Then

$$U_i = \frac{s_i}{M}$$

form an infinite sample from a uniform distribution on $[0, 1]$.

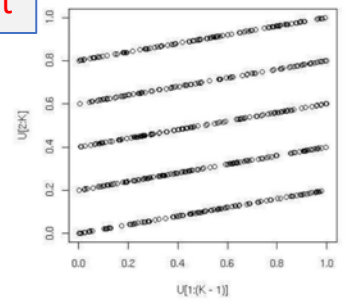
Are the numbers U_i uniformly distributed?



5000 uniform variates
$a = 1229$, $b = 1$, $M = 2048$, $s_0 = 1$

Are the numbers U_i independent?

The plot of pairs (U_i, U_{i+1}) for the linear congruential generator with $a = 1229$, $b = 1$, $M = 2048$.



Not very independent

The numbers s_i have the following properties:

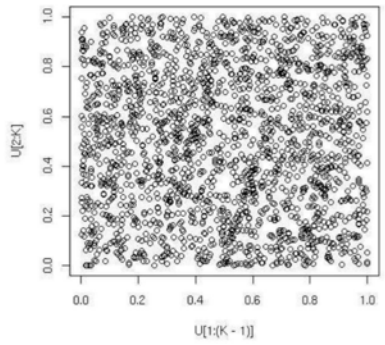
1. $s_i \in \{0, 1, 2, \ldots, M-1\}$
2. s_i are periodic with period $< M$. Indeed, since at least two values in $\{s_0, s_1, \ldots, s_M\}$ must be identical, therefore $s_i = s_{i+p}$ for some $p \leq M$.

In view of property 2 the number M should be as large as possible, because a small set of numbers make the outcome easier to predict.

But still p can be small even is M is large!

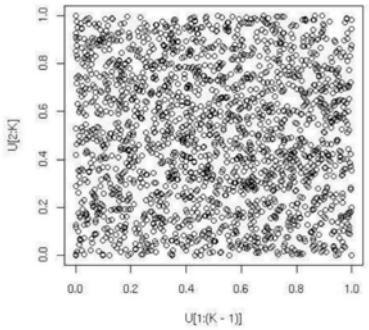
Good congruential generator with $a = 1597$, $b = 51749$ and $M = 244944$

Quite independent



RANDOM NUMBERS

Deceptively good congruential generator with $a = 2^{16} + 3$, $b = 0$, $M = 2^{31}$

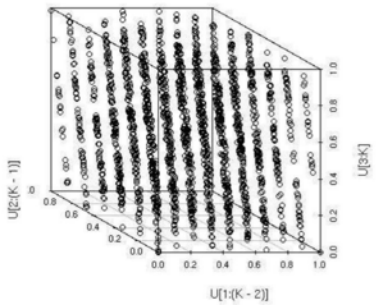


In general Marsaglia showed that the m-tuples $(U_i, \ldots, U_{i+m-1})$ generated with linear congruential generators lie on relatively low number of hyperplanes in $\mathbb{R}^m$.

This is a major disadvantage of linear congruential generators

An alternative way to generate uniform deviates is by using **Fibonacci generators**.

Apparently not so good congruential generator with $a = 2^{16} + 3$, $b = 0$, $M = 2^{31}$



Good generators are those generators that pass a large number of statistical tests, see, e.g.,

P. L'Ecuyer and R. Simard – TestU01: A C Library for Empirical Testing of Random Number Generators, *ACM Transactions on Mathematical Software*, Vol. 33, article 22, 2007

W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery – *Numerical Recipes in C*
offers a number of portable random number generators, which passed all new theoretical tests, and have been used successfully.

The simplest of these generators, called *ran0*, is a standard congruential generator

$$s_{i+1} = a s_i \mod M,$$

with $a = 7^5 = 16807$ and $M = 2^{31} - 1$, is a basis for more advanced generators *ran1* and *ran2*. There are also better generators like *ran3* and *ran4*.
Of these generators only *ran3* possesses sufficiently good properties to be used in financial calculations.
It can be difficult or impossible to implement these generators directly in a high-level language, since usually integer arithmetics is limited to 32 bits.

My research work from 1997:

J. Phys. A: Math. Gen. **30** (1997) L803–L813. Printed in the UK PII: S0305-4470(97)85389-9

LETTER TO THE EDITOR

New universal aspects of diffusion in strongly chaotic systems

Marko Robnik†¶, Jure Dobnikar†+, Andrea Rapisarda‡*, Tomaž Prosen§¤ and Marko Petkovšek‖††

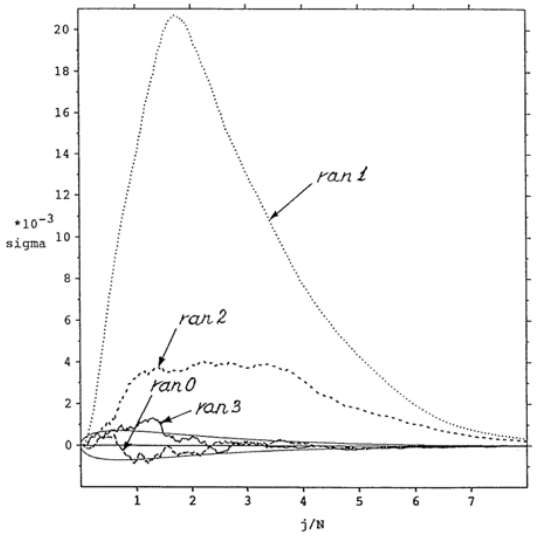


Figure 2. We show the results $\rho_{\text{numerical}}(j) - \rho_{\text{theory}}(j)$ versus j/N for the random number generators, compared with the theoretical $\pm\sigma(j)$ curves according to equation (13) (Press *et al* 1986): for ran0 and ran3 the agreement is excellent, while for ran1 and ran2 the deviations are so big and mainly positive, that they are actually not random: they seem to repel from the occupied cells.

Side product: **random number generator test**

- Discretize the plane $(x, y) \in ((0,1),(0,1))$
- Order (label) the cells from 1 to *N*
- Initially all cells are empty
- Choose random numbers with a generator and assign a cell to each number
- Once you visit the cell, it is full
- Follow how the fraction of full cells $\rho(j)$ grows with time (steps) *j*
- Analytically, if numbers are random, we expect

$$\rho(j) = 1 - (1-a)^j = 1 - \left(1 - \frac{1}{N}\right)^j \approx 1 - \exp\left(-\frac{j}{N}\right)$$

- And we can also calculate the standard deviation

$$\sigma^2(j) \approx N^{-1}[\exp(-j/N) - \exp(-2j/N)]$$

- From a good random generator we expect that it **fills up the plane expontnially** and that it stays within one σ to the analytical result

- We tested the random generators from Numerical Recepies (Press et al.) and as you can see, not all of them are good.

- Surprisingly, ran0, which is the simple congruential generator with $A = 7^5,\ B = 0,\ \ M = 2^{31} - 1$, seems to be the best (ran3 is also good).

OTHER DISTRIBUTIONS

Asume that we manage to get a good source of random numbers with a flat (uniform) distribution

In some (many) applications, we need numbers that are distributed according to a different function $f(y)$.

From cumulative: $\int_0^x dx = \int_0^y f(y)dy = F(y)$, applying the inverse operation: $F^{-1}\{F(y)\} = y = F^{-1}(x)$

For instance, if we need numbers to be distributed **exponentially**, $f(y) = e^{-y}$, we find

$$F(y) = \int_0^y e^{-y}dy = 1 - e^{-y} = x$$
$$e^{-y} = 1 - x$$
$$y = -\ln(1 - x)$$

If x_i are drawn from a flat distribution then $y_i = -\ln(1 - x_i)$ or $y_i = -\ln(x_i)$ will be distributed exponentially.

Exponential distribution comes about naturally when a system has distinct energy levels E_0 , E_1, … and we want to know the probability for the system to be a state i. This is given by the Boltzmann ratio: $p_i \propto e^{-(E_i - E_0)/k_B T}$.

OTHER DISTRIBUTIONS

Gaussian distribution, $g(y) = \frac{1}{\sqrt{2\pi\sigma}} e^{-y^2/2\sigma^2}$, ia another useful distribution for physics. σ is the variance, which we can set to 1. Other values of the variance can be modelled by rescaling $x \to x/\sigma$.

In order to construct a Gaussian-distributed set of random numbers, we consider two sets, one with a uniform distribution $f(\varphi) = 1$ for $\varphi \in [0,2\pi]$ and one with exponential distribution, $p(t) = e^{-t}$ for $t \in [0,\infty]$. We can construct two sets with a Gaussian distribution by first relating the product of the uniform and exponential distribution to a product of two Gaussians:

$$\frac{1}{2\pi} f(\phi)d\phi\, p(t)dt = g(x)dx\, g(y)dy$$

$$e^{-t}dtd\varphi = e^{-(x^2+y^2)/2}dxdy$$

This can be seen as a transformation from a polar to Cartesian coordinate system:

$$x = \sqrt{2t}\,\cos\varphi$$

$$y = \sqrt{2t}\,\sin\varphi$$

Again, if φ and t are chosen from uniform and exponential distributions, x and y will be Gaussian distributed.

In a similar way as above for exponential and Gaussian, we can in principle generate any distribution $f(y)$ numerically.

HOME EXERCISE: Testing random numbers

- Choose a few random number generators that you can think of
- Test how **uniform** and **independent** they are and discuss the **periodicity of the generated sequence**
- Perform the test of how they fill the plane that we discussed

Hints:

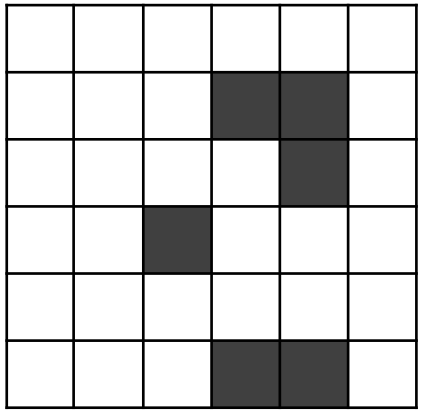
- Most programming languages have built-in random number generators.
- You can be creative in the way you create numbers, even if the "quality" is bad

- Having a good random number generator is crucial for most computer simulations of materials where we extract the physical properties by **averaging over fluctuations at the thermodynamic equilibrium**!!

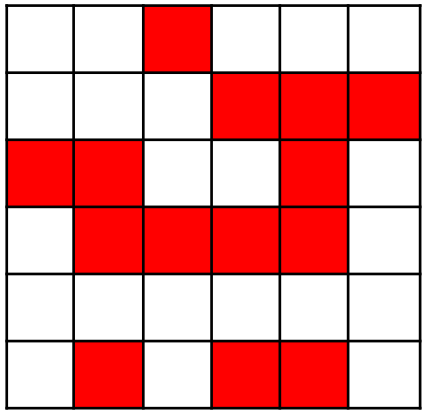
HOME EXERCISE: PERCOLATION TRESHOLD

- Imagine the following exercise: divide a square into $N \times N$ smaller squares
- Choose a probability p and colour each square with this probability
- Each time you colour the squares, you create a **configuration**
- A configuration is **percolating** if it spans the system from side to side

QUESTION: What is the probability of percolation for given p and N?



Non-percolating configuration



Horizontally percolating configuration

HOME EXERCISE: PERCOLATION TRESHOLD

- Imagine the following exercise: divide a square into $N \times N$ smaller squares
- Choose a probability p and colour each square with this probability
- Each time you colour the squares, you create a **configuration**
- A configuration is **percolating** if it spans the system from side to side

QUESTION:

What is the probability of percolation for given p and N?

APPROACH:

- Choose p and N
- Make a loop to go through all $N \times N$ small cells
- For each cell, use a random number generator to choose a number between 0 and 1
- Once you obtain a configuration, think about how to check whether it is percolating or not
- Repeat many times to obtain the probability for observing percolated configurations, p^{PER}
- Change p and N and study how does the percolation probability depend on both values: $p^{PER}(p, N)$
- Can you define a **percolation threshold p_c** (the critical value of p at which p^{PER} sharply increases from around 0 to almost 1)?
- How does this value depend on the number of cells: $p_c(N)$?

ADDITIONAL TASK: define all **clusters** in a given configuration *(areas that are connected via painted neighbours)* and calculate the **cluster size distribution** (histogram); Average this distribution over all created configurations at the given p and N and plot it.

REMARKS & ADDITIONAL CALLENGES:

- Percolation in 1D is clearly only possible if all the sites are painted, therefore $p_c^{1D} \equiv 1$
- In 2D, there is no general analytical answer
- In a 2×2 system, you can derive that the percolation probability p^{PER} is
- **Check this with your code!**
- **Can you calculate the analytically threshold for** 3×3 system?

Derivation: there are 4 percolated configurations with 2 painted cells, 4 with 3 painted cells and 1 with 4 painted cells; the probability to paint 2 cells is $p^2(1-p)^2$, to paint 3 cells it is $p^3(1-p)$, and to paint 4 cells it is p^4; from this:

$$p^{PER} = p^4 + 4p^3(1-p) + 4p^2(1-p)^2 = \\ = p^4 - 4p^3 + 4p^2$$

- Usually we are interested in what happens in the limit $N \to \infty$ in 2D or 3D.
- The answer depends on the type of lattice:

Lattice	# nn	Site percolation	Bond percolation
1d	2	**1**	**1**
2d Honeycomb	3	0.6962	$1-2\sin(\pi/18) \approx 0.65271$
2d Square	4	**0.592746**	**1/2**
2d Triangular	6	**1/2**	$2\sin(\pi/18) \approx 0.34729$
3d Diamond	4	0.43	0.388
3d Simple cubic	6	0.3116	0.2488
3d BCC	8	0.246	0.1803
3d FCC	12	0.198	0.119
4d Hypercubic	8	0.197	0.1601
5d Hypercubic	10	0.141	0.1182
6d Hypercubic	12	0.107	0.0942
7d Hypercubic	14	0.089	0.0787
Bethe lattice	z	**1/(z-1)**	**1/(z-1)**

- **Can you observe a difference between square and hexagonal lattice with your code?**

If we are only interested in either horizontal or vertical percolation, then the counting is different (only 2, not 4 configurations with 2 painted cells). The result is then $p^{PER} = p^2(2-p^2)$