

# Differential Equations with Julia

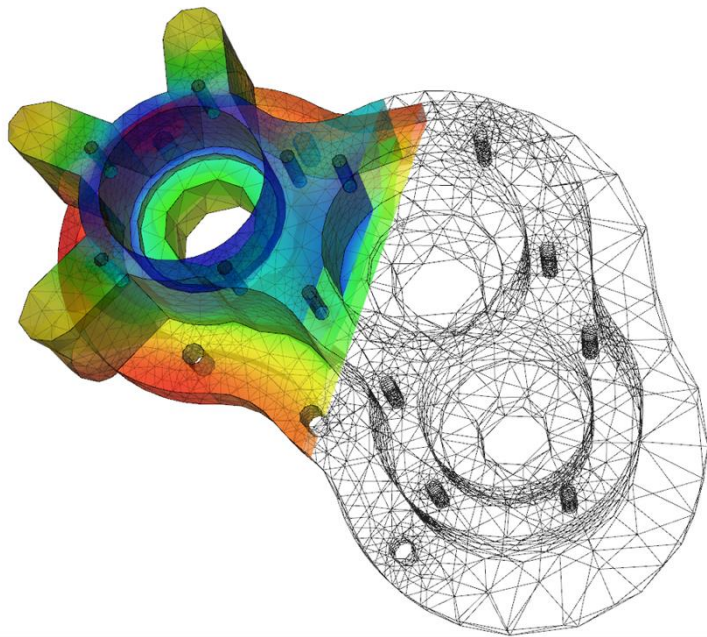
Michał Bernardelli

**SGH**

Warsaw School  
of Economics

Preparation of this workshop has been supported by the Polish National Agency for Academic Exchange under the Strategic Partnerships programme, grant number BPI/PST/2024/1/00129.





# Differential equations

# Application of differential equations\*

## 1. Physics:

- **Classical Mechanics:** Describing the motion of objects using equations like Newton's second law.
- **Electrodynamics:** Maxwell's equations describe the behavior of electric and magnetic fields.
- **Quantum Mechanics:** Schrödinger's equation is a fundamental differential equation in quantum mechanics.

## 2. Engineering:

- **Electrical Engineering:** Circuits and systems analysis involve differential equations, especially in transient and frequency domain analyses.
- **Mechanical Engineering:** Vibrations, fluid dynamics, and heat transfer are often modeled using differential equations.
- **Civil Engineering:** Structural analysis, fluid flow in pipes, and other phenomena involve differential equations.

## 3. Biology and Medicine:

- **Population Dynamics:** Modeling the growth or decline of populations.
- **Physiology:** Modeling the behavior of biological systems, such as the spread of diseases or drug absorption in the body.

## 4. Economics:

- **Macroeconomics:** Modeling economic growth, inflation, and unemployment.
- **Microeconomics:** Modeling supply and demand dynamics.

\* by ChatGPT

# Application of differential equations\*

## 5. Chemistry:

- **Chemical Kinetics:** Describing the rate of chemical reactions.
- **Transport Phenomena:** Diffusion and reaction processes in chemical systems.

## 6. Computer Science:

- **Computer Graphics:** Simulating physical phenomena like fluid flow, smoke, or fire.
- **Machine Learning:** Some models, such as neural networks, involve solving differential equations during training.

## 7. Environmental Science:

- **Ecology:** Modeling interactions between species in ecosystems.
- **Climate Modeling:** Describing the dynamics of the Earth's atmosphere.

## 8. Finance:

- **Option Pricing Models:** Black-Scholes equation and other models in finance involve differential equations.

## 9. Control Systems:

- **Control Theory:** Analyzing and designing control systems for engineering applications.

## 10. Telecommunications:

- **Signal Processing:** Analyzing and processing signals, often involving differential equations.

## 11. Mathematics:

- **Pure Mathematics:** Differential equations are studied as a field in their own right.

\* by ChatGDP

# Julia Packages

Problem type	Julia packages
Plotting	<a href="#">Plots</a>
Linear system / least squares	<a href="#">LinearSolve</a>
Sparse matrix	<a href="#">SparseArrays</a>
Interpolation	<a href="#">DataInterpolations</a> , <a href="#">ApproxFun</a>
Polynomial manipulations	<a href="#">Polynomials</a>
Rootfinding	<a href="#">NonlinearSolve</a>
Finite differences	<a href="#">FiniteDifferences</a> , <a href="#">FiniteDiff</a>
Integration	<a href="#">Quadgk</a> , <a href="#">HCubature</a>
Optimization	<a href="#">Optimization</a>
<b>Ordinary Differential Equations</b>	<b><a href="#">DifferentialEquations</a></b>
Finite Element Method	<a href="#">Gridap</a>
Automatic Differentiation	<a href="#">ForwardDiff</a> , <a href="#">Enzyme</a>
Fast Fourier Transform	<a href="#">FFTW</a>

Packages needed during this course:

- `DifferentialEquations`

- `Plots`

Optional:

- `DiffEqProblemLibrary`

# DifferentialEquations Package

- Discrete equations (function maps, discrete stochastic (Gillespie/Markov) simulations)
- Ordinary differential equations (ODEs)
- Split and Partitioned ODEs (Symplectic integrators, IMEX Methods)
- Stochastic ordinary differential equations (SODEs or SDEs)
- Stochastic differential-algebraic equations (SDAEs)
- Random differential equations (RODEs or RDEs)
- Differential algebraic equations (DAEs)

# DifferentialEquations Package

- Delay differential equations (DDEs)
- Neutral, retarded, and algebraic delay differential equations (NDDEs, RDDEs, and DDAEs)
- Stochastic delay differential equations (SDDEs)
- Experimental support for stochastic neutral, retarded, and algebraic delay differential equations (SNDDEs, SRDDEs, and SDDAEs)
- Mixed discrete and continuous equations (Hybrid Equations, Jump Diffusions)
- (Stochastic) partial differential equations ((S)PDEs) (with both finite difference and finite element methods)

# General workflow



Define a  
problem



Solve the  
problem



Analyze the  
output



# Ordinary Differential Equation (ODE)

## Defining a problem

Mathematical Specification of an ODE Problem:

$$M \frac{dt}{du} = f(u, p, t)$$

1. Definition of the function  $f$
2. Specification of the initial condition  $u_0$
3. The timespan  $tspan$  for the problem

# Ordinary Differential Equation (ODE)

## Solving a problem

[https://docs.sciml.ai/DiffEqDocs/stable/basics/common\\_solver\\_opts/#solver\\_options](https://docs.sciml.ai/DiffEqDocs/stable/basics/common_solver_opts/#solver_options)

Parameters:

- alg – algorithm; by default, alg = nothing (solve dispatches to the DifferentialEquations.jl automated algorithm selection)
- maxiters – maximum number of iterations before stopping. Defaults to 1e5.
- saveat – denotes specific times to save the solution at, during the solving phase

# Ordinary Differential Equation (ODE)

## Solving a problem

[https://docs.sciml.ai/DiffEqDocs/stable/basics/common\\_solver\\_opts/#solver\\_options](https://docs.sciml.ai/DiffEqDocs/stable/basics/common_solver_opts/#solver_options)

Parameters:

- `reltol` – Relative tolerance in adaptive timestepping. This is the tolerance on local error estimates, not necessarily the global error (though these quantities are related). Defaults to  $1e-3$  on deterministic equations (ODEs/DDEs/DAEs) and  $1e-2$  on stochastic equations (SDEs/RODEs).
- `abstol` – Absolute tolerance in adaptive timestepping. This is the tolerance on local error estimates, not necessarily the global error (though these quantities are related). Defaults to  $1e-6$  on deterministic equations (ODEs/DDEs/DAEs) and  $1e-2$  on stochastic equations (SDEs/RODEs).

# OrdinaryDiffEq algorithms

[https://docs.sciml.ai/DiffEqDocs/stable/solvers/ode\\_solve](https://docs.sciml.ai/DiffEqDocs/stable/solvers/ode_solve)

- Explicit Runge–Kutta Methods

Euler	OwrenZen4	RKO65	MSRK5
Midpoint	OwrenZen5	TanYam7	MSRK6
Heun	DP5	DP8	Stepanov5
Ralston	Tsit5	TsitPap8	SIR54
RK4	Anas5(w)	Feagin10	Alshina2
BS3	FRK65(w=0)	Feagin12	Alshina3
OwrenZen3	PFRK87(w=0)	Feagin14	Alshina6

- Parallel Explicit Runge–Kutta Methods
- Explicit Strong–Stability Preserving Runge–Kutta Methods for Hyperbolic PDEs

# OrdinaryDiffEq algorithms

- Low-Storage Methods
- Parallelized Explicit Extrapolation Methods
- Explicit Multistep Methods
- Adams-Bashforth Explicit Methods
- Adaptive step size Adams explicit Methods
- SDIRK Methods
- Fully-Implicit Runge-Kutta Methods
- Parallel Diagonally Implicit Runge-Kutta Methods
- Rosenbrock Methods

# OrdinaryDiffEq algorithms

- Rosenbrock–W Methods
- Stabilized Explicit Methods
- Parallelized Implicit Extrapolation Methods
- Parallelized DIRK Methods
- Exponential Runge–Kutta Methods
- Adaptive Exponential Rosenbrock Methods
- Exponential Propagation Iterative Runge–Kutta Methods
- Multistep Methods
- Implicit Strong–Stability Preserving Runge–Kutta Methods for Hyperbolic PDEs

# OrdinaryDiffEq algorithms – good “go-to” choices

- `AutoTsit5(Rosenbrock23())` handles both stiff and non-stiff equations. This is a good algorithm to use if you know nothing about the equation.
- `AutoVern7(Rodas5())` handles both stiff and non-stiff equations in a way that's efficient for high accuracy.
- `Tsit5()` for standard non-stiff. This is the first algorithm to try in most cases.
- `BS3()` for fast low accuracy non-stiff.
- `Vern7()` for high accuracy non-stiff.
- `Rodas4()` or `Rodas5()` for small stiff equations with Julia-defined types, events, etc.
- `KenCarp4()` or `TRBDF2()` for medium-sized (100–2000 ODEs) stiff equations.
- `RadauIIA5()` for really high accuracy stiff equations.
- `QNDF()` for large stiff equations.

# In-place functions

- *Instead of writing a function which outputs its solution  $f(u,p,t)$  you write a function which updates a vector that is designated to hold the solution  $f(du,u,p,t)$ . By doing this, DifferentialEquations.jl's solver packages are able to reduce the amount of array allocations and achieve better performance.*
- *Convention: name functions with ! at the end.*
- *Memory-efficient but not always possible (mutation sometimes not allowed).*



# Contact



dr hab. Michał Bernardelli, prof. SGH

[michal.bernardelli@sgh.waw.pl](mailto:michal.bernardelli@sgh.waw.pl)

Thank you for your attention!