

# DifferentialEquations.jl

Chris Rackauckas\*

DifferentialEquations.jl is a metapackage for solving differential equations in the Julia programming language. It utilizes a novel confederated software architecture in order to encapsulate the over 70 packages of the JuliaDiffEq ecosystem into a single extensible API. Because of this, DifferentialEquations.jl consists of over 200 methods for solving ordinary differential equations (ODEs), stochastic differential equations (SDEs), delay differential equations (DDEs), differential-algebraic equations (DAEs), random differential equations (RDEs), continuous Markov chains (Gillespie SSA), mixed discrete and continuous evolution equations (hybrid equations, jump diffusions), and (stochastic) partial differential equations ((S)PDEs). Wrappers over existing C++, Fortran, and MATLAB solvers are combined with a wide array of native Julia implementations to give rise to the largest collected set of algorithms, giving users a flexible way to solve difficult differential equations and methods researchers a test suite which covers the discipline. In addition, the native Julia solvers are consistently among the most efficient according to the extensive open testing and benchmarking suites hosted in the JuliaDiffEq organization (DiffEqBenchmarks.jl). What follows is a quick overview of the features that are documented in more detail at [docs.juliadiffeq.org](https://docs.juliadiffeq.org).

## Installation Instructions and Manual

DifferentialEquations.jl is a metapackage of the JuliaDiffEq organization, hosted at <http://juliadiffeq.org/> and its Github organization <https://github.com/JuliaDiffEq>. Its over 70 packages, plus the external extension libraries such as LSODA.jl, comprise the features of DifferentialEquations.jl. Its canonical installation is via the Julia (v0.6+) package manager command:

```
Pkg.add("DifferentialEquations")
```

which installs the basic functionality (with additional addons mentioned in the documentation). Its documentation, tutorials, benchmarks, etc. are all hosted in separate repositories due to their size and span. For detailed instructions for installing and using DifferentialEquations.jl, please see the documentation at [docs.juliadiffeq.org](https://docs.juliadiffeq.org).

## General Structure

DifferentialEquations.jl is structured around its confederated architecture via dispatch on problem types. For example, ODEs are equations of the form  $u' = f(t, p, u)$  with  $u(t_0) = u_0$ . The ODEProblem type contains the values  $f$ ,  $u_0$ , and  $tspan$  (the time span to solve on), along with other problem descriptions like a function for the Jacobian  $f'$ , a type for its Jacobian (sparse, banded, etc.). In DifferentialEquations.jl, the state variable  $u$  can be any abstract scalar or “abstract array” quantity, which is any type which has the field operations  $+$ ,  $-$ ,  $*$ ,  $/$  and be broadcastable (element-wise) if an array. Additionally, the existence of a norm is required for adaptive time stepping. Examples of possible state variables are vectors of Float64 numbers, GPU arrays of Float32 numbers, multiscale biological models generated via MultiScaleArrays.jl, arrays of arbitrary precision floats using the Arb library via ArbFloats.jl, or numbers with linear uncertainty propagation via Measurements.jl. This list cannot be enumerated since any types which define these actions are applicable and a specialized code will be compiled on-demand when encountering new types. All aspects of the DifferentialEquations.jl native solvers are similarly generic, for example the Jacobian type can be a lazy matrix-free operator with direct dispatch definitions on  $*$  or any appropriate user-defined special matrix such as a BlockBandedMatrix from BlockBandedMatrices.jl, and will compile on-demand to optimize and specialize the code for the new encountered types.

---

\*PhD Candidate, University of California, Irvine, Department of Mathematics

The solvers then dispatch on the command `solve(problem,algorithm)` via the algorithm and problem types. Using Julia’s dispatch, this then calls into the appropriate solver algorithm’s code which returns an appropriate solution type. This solution type has an interface which includes call-overloading such that by default the solutions are high order continuous solutions (defined by an interpolation scheme given by the algorithm). This means that the solution API is package-agnostic and any external developers can plug into the API. Many solvers, such as `LSODA.jl` and `GeometricIntegrators.jl`, extend the possible user choices from outside the `JuliaDiffEq` ecosystem in this manner. User algorithms can thus be written agnostic to the time evolution problem and solution method. For example, the parameter estimation algorithms in `DiffEqParamEstim.jl` perform single shooting maximum-likelihood estimation, multiple shooting, maximum a priori (MAP) estimation, etc. with local and global optimizers (allowing the user to choose from any optimizer in the Julia ecosystem). Additionally, each of the solvers have an alternative iterator form (called the integrator interface) which allows for step-by-step control of the integration, along with event handling that allows for arbitrary control over the internals, for example allowing events to change the size of the system or integrator hyperparameters such as the gain in PI-adaptive time stepping. The algorithms allow for full control over many of the internals, allowing the users to choose methods for internal linear solvers, choose the time stepping adaptivity algorithms, and more.

Given the genericness of the code and the abstract composibility, it is difficult (if not impossible) to fully enumerate a features list. The documentation hosted at [docs.juliadiffeq.org](https://docs.juliadiffeq.org) goes through many traditional and non-traditional use cases, such as solving Monte Carlo simulations of stochastic differential equations with distributed parallelism and developing multiscale biological models of changing size (with cellular birth/death) without a canonical array structure. Thus the rest of this summary will focus on the implemented solvers which can be used by `DifferentialEquations.jl` models.

## Available Differential Equation Solvers

The core packages for solving ordinary differential equations in `DifferentialEquations.jl` consist of `OrdinaryDiffEq.jl`, `Sundials.jl`, and `ODEInterface.jl`. Together these include the CVODE and IDA methods of Sundials, the Fortran methods of Earnst Hairer (`dopri5`, `dop853`, `radau`, etc.), and the core native Julia algorithms. `OrdinaryDiffEq.jl` has over 200 solvers itself, with many unique implementations for first-order ODEs, including but not limited to explicit Runge-Kutta methods, the many Strong-Stability Preserving (SSP) Runge-Kutta methods for Hyperbolic conservation laws, low storage Runge-Kutta methods, high and low order Singly-Diagonally Implicit Runge-Kutta methods (SDIRK), high order Rosenbrock methods, adaptive multistep Adams and BDF methods. Many of these implementations, such as Verner’s high efficiency order 7-9 Runge-Kutta methods, Feagin’s 10th-14th order Runge-Kutta methods, the 5th order Rosenbrock method, benchmark as some of the most efficient methods yet have no alternative open source implementations. These methods can be combined using the `CompositeAlgorithm` to generate automated switching algorithms. Examples include `AutoTsit5(Rosenbrock23())` which does automated stiffness detection and switching between a 5th order explicit Runge-Kutta method and a second order L-stable Rosenbrock-W method (with a default switching scheme using eigenvalue estimates to be submitted for publication). The appropriate implicit and semi-implicit methods, such as the Rosenbrock and SDIRK methods, are capable of solving differential-algebraic equations in mass matrix form by specifying the appropriate mass matrix in the problem type.

The ordinary differential equation problem architecture also has extra forms for structured ODE problems. The `DynamicalODEProblem` (and `SecondOrderODEProblem/HamiltonianODEProblem` helpers) solve partitioned ODEs with a dynamical structure using Runge-Kutta-Nystrom and Symplectic integrators of orders 1-12, to the author’s knowledge the most extensive set gathered to date. The `SplitODEProblem` can be used to specify implicit-explicit (IMEX) splitting for methods like the orders 3-5 SDIRK methods of Kennedy and Carpenter to take advantage of partial implicitness, that is implicitness in only part of the ODE. If the `SplitODEProblem` is given a linear operator, it is recognized as a semilinear ODE ( $u' = Au + f(t, u)$ ) and can be solved with a wide variety of exponential integrators such as `ETDRK4` with both dense matrix exponential calculations or using adaptive Krylov `expmv` (and `phimv`). These tools can be used for efficient time stepping of partial differential equations and the authors do not know of other packages which include this array of methods. Additionally, these solvers for ODEs can be utilized via `DelayDiffEq.jl` to solve delay differential equations. This package takes the `OrdinaryDiffEq.jl` solvers and wraps them within the integrator of itself to build the interpolating history function and adds discontinuity tracking (for constant

and state-dependent delays through rootfinding) along with implicit step handling (for when the time step exceeds the size of the smallest delay).

For stochastic differential equations, the core of `DifferentialEquations.jl` is `StochasticDiffEq.jl`. This package contains the first open-source high order adaptive time stepping algorithms for SDEs utilizing an algorithm developed by the author. The package includes many recent methods, such as split-stepping, stability-optimized Strong Order 1.5 Runge-Kutta methods, Strong Order 2.0 SDIRK methods for additive noise, which have no alternative open-source implementation but have been shown to be hundreds of times more efficient than commonly used methods like Euler-Maruyama. The stochastic integration engine is also the first open source SDE software to support features like event handling and stochastic DAEs in mass matrix form.

## Citations

Citations for the current list of algorithms can be found at the JuliaDiffEq citations page <http://juliadiffeq.org/citing.html>

## Further Information

The following information gives additional details about `DifferentialEquations.jl`. The attached paper is a publication in JORS which details some of the early functionality and usage. It was written for `DifferentialEquations.jl` v1.0 and does not include a lot of the latest features like the efficient native methods for stiff differential equations and the exponential integrators. For information on the latest algorithms, please see [docs.juliadiffeq.org](https://docs.juliadiffeq.org). Additional pages of note to view are:

- <https://github.com/JuliaDiffEq/DiffEqTutorials.jl>: Extended tutorials for using `DifferentialEquations.jl`
- <https://github.com/JuliaDiffEq/DiffEqBenchmarks.jl>: Benchmarks of the `DifferentialEquations.jl` solvers. Includes comparisons against MATLAB, Hairer's suite (dopri5, dop853, radau), Sundials, etc.
- <https://github.com/JuliaDiffEq/diffeqr>: R bindings for `DifferentialEquations.jl`
- <https://github.com/JuliaDiffEq/diffeqpy>: Python bindings for `DifferentialEquations.jl`
- <http://juliadiffeq.org/news/index.html>: The `DifferentialEquations.jl` release blog. One post of note is the post which introduces the R and Python bindings which includes speed comparison tests to `deSolve` and `SciPy` (<http://juliadiffeq.org/2018/04/30/Jupyter.html>).
- Video Introduction Tutorial: Intro to solving differential equations in Julia, <https://www.youtube.com/watch?v=KPEqYtEd-zY>
- Overview Video: JuliaCon 2017 | The Unique Features and Performance of `DifferentialEquations.jl` | Chris Rackauckas, <https://www.youtube.com/watch?v=75SCMIR1NXM>
- Online Differential Equation Solver: <http://app.juliadiffeq.org/> (ODEs and diagonal noise SDEs)