ODE Problems

SciMLBase.ODEProblem — Type

Defines an ordinary differential equation (ODE) problem. Documentation Page: https://docs.sciml.ai/DiffEqDocs/stable/types/ode_types/

Mathematical Specification of an ODE Problem

To define an ODE Problem, you simply need to give the function f and the initial condition u_0 which define an ODE:

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

There are two different ways of specifying `f`:

- `f(du,u,p,t)`: in-place. Memory-efficient when avoiding allocations. Best option for most cases unless mutation is not allowed.
- `f(u,p,t)`: returning `du`. Less memory-efficient way, particularly suitable when mutation is not allowed (e.g. with certain automatic differentiation packages such as Zygote).

`u₀` should be an AbstractArray (or number) whose geometry matches the desired geometry of `u`. Note that we are not limited to numbers or vectors for `u₀`; one is allowed to provide `u₀` as arbitrary matrices / higher dimension tensors as well.

For the mass matrix M, see the documentation of `ODEFunction`.

Problem Type

Constructors

`ODEProblem` can be constructed by first building an `ODEFunction` or by simply passing the ODE right-hand side to the constructor. The constructors are:

`ODEProblem(f::ODEFunction,u0,tspan,p=NullParameters();kwargs...)`

`ODEProblem{isinplace,specialize}(f,u0,tspan,p=NullParameters();kwargs...)` : Defines the ODE with the specified functions. `isinplace` optionally sets whether the function is inplace or not. This is determined automatically, but not inferred. `specialize` optionally controls the specialization level. See the specialization levels section of the SciMLBase documentation for more details. The default is `AutoSpecialize`.

For more details on the in-place and specialization controls, see the ODEFunction documentation.

Parameters are optional, and if not given, then a `NullParameters()` singleton will be used which will throw nice errors if you try to index non-existent parameters. Any extra keyword arguments are passed on to the solvers. For example, if you set a `callback` in the problem, then that `callback` will be added in every solve call.

For specifying Jacobians and mass matrices, see the `ODEFunction` documentation.

Fields

- `f`: The function in the ODE.
- `u0`: The initial condition.
- `tspan`: The timespan for the problem.
- `p`: The parameters.
- `kwargs`: The keyword arguments passed onto the solves.

Example Problem

```julia
using SciMLBase
function lorenz!(du,u,p,t)
 du[1] = 10.0(u[2]-u[1])
 du[2] = u[1]*(28.0-u[3]) - u[2]
 du[3] = u[1]*u[2] - (8/3)*u[3]
end
u0 = [1.0;0.0;0.0]
tspan = (0.0,100.0)
prob = ODEProblem(lorenz!,u0,tspan)

# Test that it worked
using OrdinaryDiffEq
sol = solve(prob,Tsit5())
using Plots; plot(sol,vars=(1,2,3))
```

More Example Problems

Example problems can be found in DiffEqProblemLibrary.jl.

To use a sample problem, such as `prob_ode_linear`, you can do something like:

```
#] add ODEProblemLibrary
using ODEProblemLibrary
prob = ODEProblemLibrary.prob_ode_linear
sol = solve(prob)
```

source

`SciMLBase.ODEFunction` — Type

```
ODEFunction{iip,F,TMM,Ta,Tt,TJ,JVP,VJP,JP,SP,TW,TWt,TPJ,S,S2,S3,O,TCV} <: Abstrac
```

A representation of an ODE function `f`, defined by:

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

and all of its related functions, such as the Jacobian of `f`, its gradient with respect to time, and more. For all cases, `u0` is the initial condition, `p` are the parameters, and `t` is the independent variable.

Constructor

```
ODEFunction{iip,specialize}(f;
                           mass_matrix = __has_mass_matrix(f) ? f.mass_matrix : I
                           analytic = __has_analytic(f) ? f.analytic : nothing,
                           tgrad= __has_tgrad(f) ? f.tgrad : nothing,
                           jac = __has_jac(f) ? f.jac : nothing,
                           jvp = __has_jvp(f) ? f.jvp : nothing,
                           vjp = __has_vjp(f) ? f.vjp : nothing,
                           jac_prototype = __has_jac_prototype(f) ? f.jac_prototy
                           sparsity = __has_sparsity(f) ? f.sparsity : jac_protot
                           paramjac = __has_paramjac(f) ? f.paramjac : nothing,
                           syms = __has_syms(f) ? f.syms : nothing,
                           indepsym= __has_indepsym(f) ? f.indepsym : nothing,
                           paramsyms = __has_paramsyms(f) ? f.paramsyms : nothing
                           colorvec = __has_colorvec(f) ? f.colorvec : nothing,
```

```
                    sys = __has_sys(f) ? f.sys : nothing)
```

Note that only the function `f` itself is required. This function should be given as `f!(du,u,p,t)` or `du = f(u,p,t)`. See the section on `iip` for more details on in-place vs out-of-place handling.

All of the remaining functions are optional for improving or accelerating the usage of `f`. These include:

- `mass_matrix`: the mass matrix `M` represented in the ODE function. Can be used to determine that the equation is actually a differential-algebraic equation (DAE) if `M` is singular. Note that in this case special solvers are required, see the DAE solver page for more details: https://docs.sciml.ai/DiffEqDocs/stable/solvers/dae_solve/. Must be an AbstractArray or an AbstractSciMLOperator.
- `analytic(u0,p,t)`: used to pass an analytical solution function for the analytical solution of the ODE. Generally only used for testing and development of the solvers.
- `tgrad(dT,u,p,t)` or dT=tgrad(u,p,t): returns $\frac{\partial f(u,p,t)}{\partial t}$
- `jac(J,u,p,t)` or `J=jac(u,p,t)`: returns $\frac{df}{du}$
- `jvp(Jv,v,u,p,t)` or `Jv=jvp(v,u,p,t)`: returns the directional derivative $\frac{df}{du} v$
- `vjp(Jv,v,u,p,t)` or `Jv=vjp(v,u,p,t)`: returns the adjoint derivative $\frac{df}{du}^* v$
- `jac_prototype`: a prototype matrix matching the type that matches the Jacobian. For example, if the Jacobian is tridiagonal, then an appropriately sized `Tridiagonal` matrix can be used as the prototype and integrators will specialize on this structure where possible. Non-structured sparsity patterns should use a `SparseMatrixCSC` with a correct sparsity pattern for the Jacobian. The default is `nothing`, which means a dense Jacobian.
- `paramjac(pJ,u,p,t)`: returns the parameter Jacobian $\frac{df}{dp}$.
- `syms`: the symbol names for the elements of the equation. This should match `u0` in size. For example, if `u0 = [0.0,1.0]` and `syms = [:x, :y]`, this will apply a canonical naming to the values, allowing `sol[:x]` in the solution and automatically naming values in plots.
- `indepsym`: the canonical naming for the independent variable. Defaults to nothing, which internally uses `t` as the representation in any plots.
- `paramsyms`: the symbol names for the parameters of the equation. This should match `p` in size. For example, if `p = [0.0, 1.0]` and `paramsyms = [:a, :b]`, this will apply a canonical naming to the values, allowing `sol[:a]` in the solution.
- `colorvec`: a color vector according to the SparseDiffTools.jl definition for the sparsity pattern of the `jac_prototype`. This specializes the Jacobian construction when using finite differences and automatic differentiation to be computed in an accelerated manner based on

the sparsity pattern. Defaults to `nothing`, which means a color vector will be internally computed on demand when required. The cost of this operation is highly dependent on the sparsity pattern.

iip: In-Place vs Out-Of-Place

`iip` is the optional boolean for determining whether a given function is written to be used in-place or out-of-place. In-place functions are `f!(du,u,p,t)` where the return is ignored, and the result is expected to be mutated into the value of `du`. Out-of-place functions are `du=f(u,p,t)`.

Normally, this is determined automatically by looking at the method table for `f` and seeing the maximum number of arguments in available dispatches. For this reason, the constructor `ODEFunction(f)` generally works (but is type-unstable). However, for type-stability or to enforce correctness, this option is passed via `ODEFunction{true}(f)`.

specialize: Controlling Compilation and Specialization

The `specialize` parameter controls the specialization level of the ODEFunction on the function `f`. This allows for a trade-off between compile and run time performance. The available specialization levels are:

- `SciMLBase.AutoSpecialize`: this form performs a lazy function wrapping on the functions of the ODE in order to stop recompilation of the ODE solver, but allow for the `prob.f` to stay unwrapped for normal usage. This is the default specialization level and strikes a balance in compile time vs runtime performance.
- `SciMLBase.FullSpecialize`: this form fully specializes the `ODEFunction` on the constituent functions that make its fields. As such, each `ODEFunction` in this form is uniquely typed, requiring re-specialization and compilation for each new ODE definition. This form has the highest compile-time at the cost of being the most optimal in runtime. This form should be preferred for long-running calculations (such as within optimization loops) and for benchmarking.
- `SciMLBase.NoSpecialize`: this form fully unspecializes the function types in the ODEFunction definition by using an `Any` type declaration. As a result, it can result in reduced runtime performance, but is the form that induces the least compile-time.
- `SciMLBase.FunctionWrapperSpecialize`: this is an eager function wrapping form. It is unsafe with many solvers, and thus is mostly used for development testing.

For more details, see the specialization levels section of the SciMLBase documentation.

Fields

The fields of the ODEFunction type directly match the names of the inputs.

More Details on Jacobians

The following example creates an inplace `ODEFunction` whose Jacobian is a `Diagonal`:

```
using LinearAlgebra
f = (du,u,p,t) -> du .= t .* u
jac = (J,u,p,t) -> (J[1,1] = t; J[2,2] = t; J)
jp = Diagonal(zeros(2))
fun = ODEFunction(f; jac=jac, jac_prototype=jp)
```

Note that the integrators will always make a deep copy of `fun.jac_prototype`, so there's no worry of aliasing.

In general, the Jacobian prototype can be anything that has `mul!` defined, in particular sparse matrices or custom lazy types that support `mul!`. A special case is when the `jac_prototype` is a `AbstractSciMLOperator`, in which case you do not need to supply `jac` as it is automatically set to `update_coefficients!`. Refer to the AbstractSciMLOperators documentation for more information on setting up time/parameter dependent operators.

Examples

Declaring Explicit Jacobians for ODEs

The most standard case, declaring a function for a Jacobian is done by overloading the function `f(du,u,p,t)` with an in-place updating function for the Jacobian: `f_jac(J,u,p,t)` where the value type is used for dispatch. For example, take the Lotka-Volterra model:

```
function f(du,u,p,t)
  du[1] = 2.0 * u[1] - 1.2 * u[1]*u[2]
  du[2] = -3 * u[2] + u[1]*u[2]
end
```

To declare the Jacobian, we simply add the dispatch:

```
function f_jac(J,u,p,t)
  J[1,1] = 2.0 - 1.2 * u[2]
  J[1,2] = -1.2 * u[1]
  J[2,1] = 1 * u[2]
  J[2,2] = -3 + u[1]
  nothing
```

```
end
```

Then we can supply the Jacobian with our ODE as:

```
ff = ODEFunction(f;jac=f_jac)
```

and use this in an `ODEProblem`:

```
prob = ODEProblem(ff,ones(2),(0.0,10.0))
```

Symbolically Generating the Functions

See the `modelingtoolkitize` function from ModelingToolkit.jl for automatically symbolically generating the Jacobian and more from the numerically-defined functions.

Solution Type

`SciMLBase.ODESolution` — Type

```
struct ODESolution{T, N, uType, uType2, DType, tType, rateType, P, A, IType, S, A
```

Representation of the solution to an ordinary differential equation defined by an ODEProblem.

DESolution Interface

For more information on interacting with `DESolution` types, check out the Solution Handling page of the DifferentialEquations.jl documentation.

https://docs.sciml.ai/DiffEqDocs/stable/basics/solution/

Fields

- `u`: the representation of the ODE solution. Given as an array of solutions, where `u[i]` corresponds to the solution at time `t[i]`. It is recommended in most cases one does not access `sol.u` directly and instead use the array interface described in the Solution Handling page of the DifferentialEquations.jl documentation.
- `t`: the time points corresponding to the saved values of the ODE solution.
- `prob`: the original ODEProblem that was solved.

`alg`: the algorithm type used by the solver.

`stats`: statistics of the solver, such as the number of function evaluations required, number of Jacobians computed, and more.

`retcode`: the return code from the solver. Used to determine whether the solver solved successfully, whether it terminated early due to a user-defined callback, or whether it exited due to an error. For more details, see the return code documentation.

Example Problems

Example problems can be found in DiffEqProblemLibrary.jl.

To use a sample problem, such as `prob_ode_linear`, you can do something like:

```
#] add DiffEqProblemLibrary
using DiffEqProblemLibrary.ODEProblemLibrary
# load problems
ODEProblemLibrary.importodeproblems()
prob = ODEProblemLibrary.prob_ode_linear
sol = solve(prob)
```

`ODEProblemLibrary.prob_ode_linear` — Constant

Linear ODE

$$\frac{du}{dt} = \alpha u$$

with initial condition $u_0 = \frac{1}{2}$, $\alpha = 1.01$, and solution

$$u(t) = u_0 e^{\alpha t}$$

with Float64s. The parameter is α

`ODEProblemLibrary.prob_ode_2Dlinear` — Constant

4x2 version of the Linear ODE

$$\frac{du}{dt} = \alpha u$$

with initial condition u_0 as all uniformly distributed random numbers, $\alpha = 1.01$, and solution

$$u(t) = u_0 e^{\alpha t}$$

with Float64s

`ODEProblemLibrary.prob_ode_bigfloatlinear` — Constant

Linear ODE

$$\frac{du}{dt} = \alpha u$$

with initial condition $u_0 = \frac{1}{2}$, $\alpha = 1.01$, and solution

$$u(t) = u_0 e^{\alpha t}$$

with BigFloats

`ODEProblemLibrary.prob_ode_bigfloat2Dlinear` — Constant

4x2 version of the Linear ODE

$$\frac{du}{dt} = \alpha u$$

with initial condition u_0 as all uniformly distributed random numbers, $\alpha = 1.01$, and solution

$$u(t) = u_0 e^{\alpha t}$$

with BigFloats

`ODEProblemLibrary.prob_ode_large2Dlinear` — Constant

100x100 version of the Linear ODE

$$\frac{du}{dt} = \alpha u$$

with initial condition u_0 as all uniformly distributed random numbers, $\alpha = 1.01$, and solution

$$u(t) = u_0 e^{\alpha t}$$

with Float64s

`ODEProblemLibrary.prob_ode_2Dlinear_notinplace` — Constant

4x2 version of the Linear ODE

$$\frac{du}{dt} = \alpha u$$

with initial condition u_0 as all uniformly distributed random numbers, $\alpha = 1.01$, and solution

$$u(t) = u_0 e^{\alpha t}$$

on Float64. Purposefully not in-place as a test.

`ODEProblemLibrary.prob_ode_lotkavolterra` — Constant

Lotka-Volterra Equations (Non-stiff)

$$\frac{dx}{dt} = ax - bxy$$

$$\frac{dy}{dt} = -cy + dxy$$

with initial condition $x = y = 1$

`ODEProblemLibrary.prob_ode_fitzhughnagumo` — Constant

Fitzhugh-Nagumo (Non-stiff)

$$\frac{dv}{dt} = v - \frac{v^3}{3} - w + I_{est}$$

$$\tau \frac{dw}{dt} = v + a - bw$$

with initial condition $v = w = 1$

`ODEProblemLibrary.prob_ode_threebody` — Constant

The ThreeBody problem as written by Hairer: (Non-stiff)

$$\frac{dy_1}{dt} = y_1 + 2\frac{dy_2}{dt} - \bar{\mu}\frac{y_1 + \mu}{D_1} - \mu\frac{y_1 - \bar{\mu}}{D_2}$$

$$\frac{dy_2}{dt} = y_2 - 2\frac{dy_1}{dt} - \bar{\mu}\frac{y_2}{D_1} - \mu\frac{y_2}{D_2}$$

$$D_1 = ((y_1 + \mu)^2 + y_2^2)^{3/2}$$

$$D_2 = ((y_1 - \bar{\mu})^2 + y_2^2)^{3/2}$$

$$\mu = 0.012277471$$

$$\bar{\mu} = 1 - \mu$$

From Hairer Norsett Wanner Solving Ordinary Differential Equations I - Nonstiff Problems Page 129

Usually solved on $t_0 = 0.0$ and $T = 17.0652165601579625588917206249$ Periodic with that setup.

`ODEProblemLibrary.prob_ode_pleiades` — Constant

Pleiades Problem (Non-stiff)

$$\frac{d^2x_i}{dt^2} = \sum_{j\neq i} m_j(x_j - x_i)/r_{ij}$$

$$\frac{d^2y_i}{dt^2} = \sum_{j\neq i} m_j(y_j - y_i)/r_{ij}$$

where

$$r_{ij} = ((x_i - x_j)^2 + (y_i - y_j)^2)^{3/2}$$

and initial conditions are

$$x_1(0) = 3$$

$$x_2(0) = 3$$

$$x_3(0) = -1$$

$$x_4(0) = -3$$

$$x_5(0) = 2$$

$$x_6(0) = -2$$

$$x_7(0) = 2$$

$$y_1(0) = 3$$

$$y_2(0) = -3$$

$$y_3(0) = 2$$

$$y_4(0) = 0$$

$$y_5(0) = 0$$

$$y_6(0) = -4$$

$$y_7(0) = 4$$

and with $\frac{dx_i(0)}{dt} = \frac{dy_i(0)}{dt} = 0$ except for

$$\frac{dx_6(0)}{dt} = 1.75$$

$$\frac{dx_7(0)}{dt} = -1.5$$

$$\frac{dy_4(0)}{dt} = -1.25$$

$$\frac{dy_5(0)}{dt} = 1$$

From Hairer Norsett Wanner Solving Ordinary Differential Equations I - Nonstiff Problems Page 244

Usually solved from 0 to 3.

`ODEProblemLibrary.prob_ode_vanderpol` — Constant

Van der Pol Equations

$$\frac{dx}{dt} = y$$

$$\frac{dy}{dt} = \mu((1 - x^2)y - x)$$

with $\mu = 1.0$ and $u_0 = [0, \quad 3]$

Non-stiff parameters.

`ODEProblemLibrary.prob_ode_vanderpol_stiff` — Constant

Van der Pol Equations

$$\frac{dx}{dt} = y$$

$$\frac{dy}{dt} = \mu((1 - x^2)y - x)$$

with $\mu = 10^6$ and $u_0 = [0, \quad 3]$

Stiff parameters.

`ODEProblemLibrary.prob_ode_rober` — Constant

The Robertson biochemical reactions: (Stiff)

$$\frac{dy_1}{dt} = -k_1 y_1 + k_3 y_2 y_3$$

$$\frac{dy_2}{dt} = k_1 y_1 - k_2 y_2^2 - k_3 y_2 y_3$$

$$\frac{dy_3}{dt} = k_2 y_2^2$$

where $k_1 = 0.04$, $k_2 = 3 \times 10^7$, $k_3 = 10^4$. For details, see:

Hairer Norsett Wanner Solving Ordinary Differential Equations I - Nonstiff Problems Page 129

Usually solved on $[0, 1e11]$

`ODEProblemLibrary.prob_ode_rigidbody` — Constant

Rigid Body Equations (Non-stiff)

$$\frac{dy_1}{dt} = I_1 y_2 y_3$$

$$\frac{dy_2}{dt} = I_2 y_1 y_3$$

$$\frac{dy_3}{dt} = I_3 y_1 y_2$$

with $I_1 = -2$, $I_2 = 1.25$, and $I_3 = -1/2$.

The initial condition is $y = [1.0; 0.0; 0.9]$.

From Solving Differential Equations in R by Karline Soetaert

or Hairer Norsett Wanner Solving Ordinary Differential Equations I - Nonstiff Problems Page 244

Usually solved from 0 to 20.

`ODEProblemLibrary.prob_ode_hires` — Constant

Hires Problem (Stiff)

It is in the form of

$$\frac{dy}{dt} = f(y)$$

with

$$y(0) = y_0, \quad y \in \mathbb{R}^8, \quad 0 \leq t \leq 321.8122$$

where f is defined by

$$f(y) = \begin{array}{lllll} -1.71y_1 & +0.43y_2 & +8.32y_3 & +0.0007y_4 & \\ 1.71y_1 & -8.75y_2 & & & \\ -10.03y_3 & +0.43y_4 & +0.035y_5 & & \\ 8.32y_2 & +1.71y_3 & -1.12y_4 & & \\ -1.745y_5 & +0.43y_6 & +0.43y_7 & & \\ -280y_6y_8 & +0.69y_4 & +1.71y_5 & -0.43y_6 & +0.69y_7 \\ 280y_6y_8 & -1.81y_7 & & & \\ -280y_6y_8 & +1.81y_7 & & & \end{array}$$

Reference: demohires.pdf Notebook: Hires.ipynb

`ODEProblemLibrary.prob_ode_orego` — Constant

Orego Problem (Stiff)

It is in the form of $\frac{dy}{dt} = f(y), \quad y(0) = y0$, with

$$y \in \mathbb{R}^3, \quad 0 \leq t \leq 360$$

where f is defined by

$$f(y) = \begin{array}{c} s(y_2 - y_1(1 - qy_1 - y_2)) \\ (y_3 - y_2(1 + y_1))/s \\ w(y_1 - y_3) \end{array}$$

where $s = 77.27$, $w = 0.161$ and $q = 8.375 \cdot 10^{-6}$.

Reference: demoorego.pdf Notebook: Orego.ipynb

`ODEProblemLibrary.prob_ode_pollution` — Constant

Pollution Problem (Stiff)

This IVP is a stiff system of 20 non-linear Ordinary Differential Equations. It is in the form of

$$\frac{dy}{dt} = f(y)$$

with

$$y(0) = y0, \quad y \in \mathbb{R}^20, \quad 0 \le t \le 60$$

where f is defined by

$$f(y) = \begin{pmatrix} -\sum_{j\in 1,10,14,23,24} r_j + \sum_{j\in 2,3,9,11,12,22,25} r_j \\ -r_2 - r_3 - r_9 - r_12 + r_1 + r_{21} \\ -r_{15} + r_1 + r_{17} + r_{19} + r_{22} \\ -r_2 - r_{16} - r_{17} - r_{23} + r_{15} \\ -r_3 + 2r_4 + r_6 + r_7 + r_{13} + r_{20} \\ -r_6 - r_8 - r_{14} - r_{20} + r_3 + 2r_{18} \\ -r_4 - r_5 - r_6 + r_{13} \\ r_4 + r_5 + r_6 + r_7 \\ -r_7 - r_8 \\ -r_{12} + r_7 + r_9 \\ -r_9 - r_{10} + r_8 + r_{11} \\ r_9 \\ -r_{11} + r_{10} \\ -r_{13} + r_{12} \\ r_{14} \\ -r_{18} - r_{19} + r_{16} \\ -r_{20} \\ r_{20} \\ -r21 - r_{22} - r_{24} + r_{23} + r_{25} \\ -r_{25} + r_{24} \end{pmatrix}$$

with the initial condition of

$$y0 = (0, 0.2, 0, 0.04, 0, 0, 0.1, 0.3, 0.01, 0, 0, 0, 0, 0, 0, 0, 0.007, 0, 0, 0)^T$$

Analytical Jacobian is included.

Reference: pollu.pdf Notebook: Pollution.ipynb

`ODEProblemLibrary.prob_ode_nonlinchem` — Constant

Nonlinear system of reactions with an analytical solution

$$\frac{dy_1}{dt} = -y_1$$

$$\frac{dy_2}{dt} = y_1 - y_2^2$$

$$\frac{dy_3}{dt} = y_2^2$$

with initial condition $y = [1; 0; 0]$ on a time span of $t \in (0, 20)$

From

Liu, L. C., Tian, B., Xue, Y. S., Wang, M., & Liu, W. J. (2012). Analytic solution for a nonlinear chemistry system of ordinary differential equations. Nonlinear Dynamics, 68(1-2), 17-21.

The analytical solution is implemented, allowing easy testing of ODE solvers.

`ODEProblemLibrary.prob_ode_brusselator_1d` — Constant

1D Brusselator

$$\frac{\partial u}{\partial t} = A + u^2 v - (B+1)u + \alpha \frac{\partial^2 u}{\partial x^2}$$

$$\frac{\partial v}{\partial t} = Bu - u^2 v + \alpha \frac{\partial^2 u}{\partial x^2}$$

and the initial conditions are

$$u(x, 0) = 1 + \sin(2\pi x)$$

$$v(x, 0) = 3$$

with the boundary condition

$$u(0, t) = u(1, t) = 1$$

$$v(0, t) = v(1, t) = 3$$

From Hairer Norsett Wanner Solving Ordinary Differential Equations II - Stiff and Differential-

Algebraic Problems Page 6

ODEProblemLibrary.prob_ode_brusselator_2d — Constant

2D Brusselator

$$\frac{\partial u}{\partial t} = 1 + u^2v - 4.4u + \alpha(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}) + f(x, y, t)$$

$$\frac{\partial v}{\partial t} = 3.4u - u^2v + \alpha(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2})$$

where

$$f(x, y, t) = \begin{cases} 5 & \text{if } (x - 0.3)^2 + (y - 0.6)^2 \le 0.1^2 \text{ and } t \ge 1.1 \\ 0 & \text{else} \end{cases}$$

and the initial conditions are

$$u(x, y, 0) = 22 \cdot y(1 - y)^{3/2}$$

$$v(x, y, 0) = 27 \cdot x(1 - x)^{3/2}$$

with the periodic boundary condition

$$u(x + 1, y, t) = u(x, y, t)$$

$$u(x, y + 1, t) = u(x, y, t)$$

From Hairer Norsett Wanner Solving Ordinary Differential Equations II - Stiff and Differential-Algebraic Problems Page 152

ODEProblemLibrary.prob_ode_filament — Constant

Filament PDE Discretization

Notebook: Filament.ipynb

In this problem is a real-world biological model from a paper entitled Magnetic dipole with a flexible tail as a self-propelling microdevice. It is a system of PDEs representing a Kirchhoff model of an elastic rod, where the equations of motion are given by the Rouse approximation with free boundary conditions.

`ODEProblemLibrary.prob_ode_thomas` — Constant

Thomas' cyclically symmetric attractor equations

$$\begin{aligned} \frac{\mathrm{d}x\left(t\right)}{\mathrm{d}t} &= \sin\left(y\left(t\right)\right) - b x\left(t\right) && (2) \\ \frac{\mathrm{d}y\left(t\right)}{\mathrm{d}t} &= \sin\left(z\left(t\right)\right) - b y\left(t\right) && (3) \\ \frac{\mathrm{d}z\left(t\right)}{\mathrm{d}t} &= \sin\left(x\left(t\right)\right) - b z\left(t\right) \end{aligned}$$

Reference

Wikipedia

`ODEProblemLibrary.prob_ode_lorenz` — Constant

Lorenz equations

$$\begin{aligned} \frac{\mathrm{d}x\left(t\right)}{\mathrm{d}t} &= \left(-x\left(t\right) + y\left(t\right)\right)\sigma && (5) \\ \frac{\mathrm{d}y\left(t\right)}{\mathrm{d}t} &= -y\left(t\right) + x\left(t\right)\left(-z\left(t\right) + \rho\right) && (6) \\ \frac{\mathrm{d}z\left(t\right)}{\mathrm{d}t} &= x\left(t\right)y\left(t\right) - z\left(t\right)\beta \end{aligned}$$

Reference

Wikipedia

`ODEProblemLibrary.prob_ode_aizawa` — Constant

Aizawa equations

$$\frac{\mathrm{d}x\left(t \right)}{\mathrm{d}t} = \left(-b + z\left(t \right) \right) x\left(t \right) - d y\left(t \right) \tag{8}$$

$$\frac{\mathrm{d}y\left(t \right)}{\mathrm{d}t} = \left(-b + z\left(t \right) \right) y\left(t \right) + d x\left(t \right) \tag{9}$$

$$\frac{\mathrm{d}z\left(t \right)}{\mathrm{d}t} = c + a z\left(t \right) - \frac{1}{3} \left(z\left(t \right) \right)^{3} + \left(-1 - e z\left(t \right) \right) \left(\left(x\left(t \right) \right)^{2} + \left(y\left(t \right) \right)^{2} \right) + \left(x\left(t \right) \right)^{3} f z\left(t \right)$$

Reference

`ODEProblemLibrary.prob_ode_dadras` — Constant

Dadras equations

$$\frac{\mathrm{d}x\left(t \right)}{\mathrm{d}t} = y\left(t \right) - a x\left(t \right) + b y\left(t \right) z\left(t \right) \tag{11}$$

$$\frac{\mathrm{d}y\left(t \right)}{\mathrm{d}t} = z\left(t \right) + c y\left(t \right) - x\left(t \right) z\left(t \right) \tag{12}$$

$$\frac{\mathrm{d}z\left(t \right)}{\mathrm{d}t} = - e z\left(t \right) + d x\left(t \right) y\left(t \right)$$

Reference

`ODEProblemLibrary.prob_ode_chen` — Constant

chen equations

$$\frac{\mathrm{d}x\left(t \right)}{\mathrm{d}t} = a \left(-x\left(t \right) + y\left(t \right) \right) \tag{14}$$

$$\frac{\mathrm{d}y\left(t \right)}{\mathrm{d}t} = \left(-a + c \right) x\left(t \right) + c y\left(t \right) - x\left(t \right) z\left(t \right) \tag{15}$$

$$\frac{\mathrm{d}z\left(t \right)}{\mathrm{d}t} = - b z\left(t \right) + x\left(t \right) y\left(t \right)$$

Reference

ODEProblemLibrary.prob_ode_rossler — Constant

rossler equations

$$\begin{aligned}
\frac{\mathrm{d}x\left(t \right)}{\mathrm{d}t} &= - y\left(t \right) - z\left(t \right) \\
\frac{\mathrm{d}y\left(t \right)}{\mathrm{d}t} &= x\left(t \right) + a y\left(t \right) \\
\frac{\mathrm{d}z\left(t \right)}{\mathrm{d}t} &= b + \left(- c + x\left(t \right) \right) z\left(t \right)
\end{aligned}$$

(17)
(18)

ReferenceWikipedia

ODEProblemLibrary.prob_ode_rabinovich_fabrikant — Constant

rabinovich_fabrikant equations

$$\begin{aligned}
\frac{\mathrm{d}x\left(t \right)}{\mathrm{d}t} &= b x\left(t \right) + \left(-1 + z\left(t \right) + \left(x\left(t \right) \right)^{2} \right) y\left(t \right) \\
\frac{\mathrm{d}y\left(t \right)}{\mathrm{d}t} &= b y\left(t \right) + x\left(t \right) \left(1 + 3 z\left(t \right) - \left(x\left(t \right) \right)^{2} \right) \\
\frac{\mathrm{d}z\left(t \right)}{\mathrm{d}t} &= - 2 \left(a + x\left(t \right) y\left(t \right) \right) z\left(t \right)
\end{aligned}$$

(20)
(21)

Reference

ODEProblemLibrary.prob_ode_sprott — Constant

sprott equations

$$\begin{aligned}
\frac{\mathrm{d}x\left(t \right)}{\mathrm{d}t} &= y\left(t \right) + x\left(t \right) z\left(t \right) + a x\left(t \right) y\left(t \right) \\
\frac{\mathrm{d}y\left(t \right)}{\mathrm{d}t} &= 1 + y\left(t \right) z\left(t \right) - \left(x\left(t \right) \right)^{2} b \\
\frac{\mathrm{d}z\left(t \right)}{\mathrm{d}t} &= x\left(t \right) - \left(x\left(t \right) \right)^{2} - \left(y\left(t \right) \right)^{2}
\end{aligned}$$

(23)
(24)

Reference

ODEProblemLibrary.prob_ode_hindmarsh_rose — Constant

hindmarsh_rose equations

$$\frac{\mathrm{d}x\left(t \right)}{\mathrm{d}t} = i + y\left(t \right) - z\left(t \right) + \left(x\left(t \right) \right)^{2} b - \left(x\left(t \right) \right)^{3} a \tag{26}$$

$$\frac{\mathrm{d}y\left(t \right)}{\mathrm{d}t} = c - y\left(t \right) - \left(x\left(t \right) \right)^{2} d \tag{27}$$

$$\frac{\mathrm{d}z\left(t \right)}{\mathrm{d}t} = r \left(- z\left(t \right) + s \left(- xr + x\left(t \right) \right) \right)$$

Reference