Choose an ODE Solver

Ordinary Differential Equations

An *ordinary differential equation* (ODE) contains one or more derivatives of a dependent variable, *y*, with respect to a single independent variable, *t*, usually referred to as time. The notation used here for representing derivatives of *y* with respect to *t* is y' for a first derivative, y'' for a second derivative, and so on. The *order* of the ODE is equal to the highest-order derivative of *y* that appears in the equation.

For example, this is a second order ODE:

$$y'' = 9y$$

In an *initial value problem*, the ODE is solved by starting from an initial state. Using the initial condition, y_0, as well as a period of time over which the answer is to be obtained, (t_0, t_f), the solution is obtained iteratively. At each step the solver applies a particular algorithm to the results of previous steps. At the first such step, the initial condition provides the necessary information that allows the integration to proceed. The final result is that the ODE solver returns a vector of time steps $t = [t_0, t_1, t_2, ..., t_f]$ as well as the corresponding solution at each step $y = [y_0, y_1, y_2, ..., y_f]$.

Types of ODEs

The ODE solvers in MATLAB® solve these types of first-order ODEs:

- Explicit ODEs of the form $y' = f(t, y)$.
- Linearly implicit ODEs of the form $M(t, y)y' = f(t, y)$, where $M(t, y)$ is a nonsingular mass matrix. The mass matrix can be time- or state-dependent, or it can be a constant matrix. Linearly implicit ODEs involve linear combinations of the first derivative of *y*, which are encoded in the mass matrix.

  Linearly implicit ODEs can always be transformed to an explicit form, $y' = M^{-1}(t, y)f(t, y)$. However, specifying the mass matrix directly to the ODE solver avoids this transformation, which is inconvenient and can be computationally expensive.
- If some components of y' are missing, then the equations are called *differential algebraic equations*, or DAEs, and the system of DAEs contains some *algebraic variables*. Algebraic variables are dependent variables whose derivatives do not appear in the equations. A system of DAEs can be rewritten as an equivalent system of first-order ODEs by taking derivatives of the equations to eliminate the algebraic variables. The number of derivatives needed to rewrite a DAE as an ODE is called the differential index. The `ode15s` and `ode23t` solvers can solve index-1 DAEs.
- Fully implicit ODEs of the form $f(t, y, y') = 0$. Fully implicit ODEs cannot be rewritten in an explicit form, and might also contain some algebraic variables. The `ode15i` solver is designed for fully implicit problems, including index-1 DAEs.

You can supply additional information to the solver for some types of problems by using the `odeset` function to create an options structure.

Systems of ODEs

You can specify any number of coupled ODE equations to solve, and in principle the number of equations is only limited by available computer memory. If the system of equations has *n* equations,

$$\begin{pmatrix} y'_1 \\ y'_2 \\ \vdots \\ y'_n \end{pmatrix} = \begin{pmatrix} f_1(t, y_1, y_2, ..., y_n) \\ f_2(t, y_1, y_2, ..., y_n) \\ \vdots \\ f_n(t, y_1, y_2, ..., y_n) \end{pmatrix},$$

then the function that encodes the equations returns a vector with *n* elements, corresponding to the values for $y'_1,\ y'_2,\ \ldots,\ y'_n$. For example, consider the system of two equations

$$\begin{cases} y'_1 = y_2 \\ y'_2 = y_1 y_2 - 2 \end{cases}.$$

A function that encodes these equations is

```
function dy = myODE(t,y)
dy(1) = y(2);
dy(2) = y(1)*y(2)-2;
```

Higher-Order ODEs

The MATLAB ODE solvers only solve first-order equations. You must rewrite higher-order ODEs as an equivalent system of first-order equations using the generic substitutions

$$\begin{aligned} y_1 &= y \\ y_2 &= y' \\ y_3 &= y'' \\ &\vdots \\ y_n &= y^{(n-1)}. \end{aligned}$$

The result of these substitutions is a system of *n* first-order equations

$$\begin{cases} y'_1 = y_2 \\ y'_2 = y_3 \\ \quad\vdots \\ y'_n = f(t, y_1, y_2, ..., y_n). \end{cases}$$

For example, consider the third-order ODE

$$y''' - y''\, y + 1 = 0.$$

Using the substitutions

$$\begin{aligned} y_1 &= y \\ y_2 &= y' \\ y_3 &= y'' \end{aligned}$$

results in the equivalent first-order system

$$\begin{cases} y'_1 = y_2 \\ y'_2 = y_3 \\ y'_3 = y_1 y_3 - 1. \end{cases}$$

The code for this system of equations is then

```
function dydt = f(t,y)
dydt(1) = y(2);
dydt(2) = y(3);
dydt(3) = y(1)*y(3)-1;
```

Complex ODEs

Consider the complex ODE equation

$$y' = f(t, y),$$

where $y = y_1 + iy_2$. To solve it, separate the real and imaginary parts into different solution components, then recombine the results at the end. Conceptually, this looks like

$$yv = [\text{Real}(y) \quad \text{Imag}(y)]$$
$$fv = [\text{Real}(f(t, y)) \quad \text{Imag}(f(t, y))].$$

For example, if the ODE is $y' = yt + 2i$, then you can represent the equation using a function file.

```
function f = complexf(t,y)
% Define function that takes and returns complex values
f = y.*t + 2*i;
```

Then, the code to separate the real and imaginary parts is

```
function fv = imaginaryODE(t,yv)
% Construct y from the real and imaginary components
y = yv(1) + i*yv(2);

% Evaluate the function
yp = complexf(t,y);

% Return real and imaginary in separate components
fv = [real(yp); imag(yp)];
```

When you run a solver to obtain the solution, the initial condition `y0` is also separated into real and imaginary parts to provide an initial condition for each solution component.

```
y0 = 1+i;
yv0 = [real(y0); imag(y0)];
tspan = [0 2];
[t,yv] = ode45(@imaginaryODE, tspan, yv0);
```

Once you obtain the solution, combine the real and imaginary components together to obtain the final result.

```
y = yv(:,1) + i*yv(:,2);
```

Basic Solver Selection

`ode45` performs well with most ODE problems and should generally be your first choice of solver. However, `ode23` and `ode113` can be more efficient than `ode45` for problems with looser or tighter accuracy requirements.

Some ODE problems exhibit *stiffness*, or difficulty in evaluation. Stiffness is a term that defies a precise definition, but in general, stiffness occurs when there is a difference in scaling somewhere in the problem. For example, if an ODE has two solution components that vary on drastically different time scales, then the equation might be stiff. You can identify a problem as stiff if nonstiff solvers (such as `ode45`) are unable to solve the problem or are extremely slow. If you observe that a nonstiff solver is very slow, try using a stiff solver such as `ode15s` instead. When using a stiff solver, you can improve reliability and efficiency by supplying the Jacobian matrix or its sparsity pattern.

This table provides general guidelines on when to use each of the different solvers.

Solver	Problem Type	Accuracy	When to Use
`ode45`	Nonstiff	Medium	Most of the time. `ode45` should be the first solver you try.
`ode23`		Low	`ode23` can be more efficient than `ode45` at problems with crude tolerances, or in the presence of moderate stiffness.
`ode113`		Low to High	`ode113` can be more efficient than `ode45` at problems with stringent error tolerances, or when the ODE function is expensive to evaluate.
`ode15s`	Stiff	Low to Medium	Try `ode15s` when `ode45` fails or is inefficient and you suspect that the problem is stiff. Also use `ode15s` when solving differential algebraic equations (DAEs).
`ode23s`		Low	`ode23s` can be more efficient than `ode15s` at problems with crude error tolerances. It can solve some stiff problems for which `ode15s` is not effective. `ode23s` computes the Jacobian in each step, so it is beneficial to provide the Jacobian via `odeset` to maximize efficiency and accuracy. If there is a mass matrix, it must be constant.
`ode23t`		Low	Use `ode23t` if the problem is

			only moderately stiff and you need a solution without numerical damping. `ode23t` can solve differential algebraic equations (DAEs).
`ode23tb`		Low	Like `ode23s`, the `ode23tb` solver might be more efficient than `ode15s` at problems with crude error tolerances.
`ode15i`	Fully implicit	Low	Use `ode15i` for fully implicit problems *f(t,y,y') = 0* and for differential algebraic equations (DAEs) of index 1.

For details and further recommendations about when to use each solver, see [5].

Summary of ODE Examples and Files

There are several example files available that serve as excellent starting points for most ODE problems. To run the **Differential Equations Examples** app, which lets you easily explore and run examples, type

```
odeexamples
```

To open an individual example file for editing, type

```
edit exampleFileName.m
```

To run an example, type

```
exampleFileName
```

This table contains a list of the available ODE and DAE example files as well as the solvers and options they use. Links are included for the subset of examples that are also published directly in the documentation.

Example File	Solver Used	Options Specified	Description	Documentation Link
`amp1dae`	`ode23t`	• `'Mass'`	Stiff DAE — electrical circuit with constant, singular mass matrix	Solve Stiff Differential Algebraic Equation
`ballode`	`ode23`	• `'Events'` • `'OutputFcn'` • `'OutputSel'` • `'Refine'` • `'InitialStep'` • `'MaxStep'`	Simple event location — bouncing ball	ODE Event Location
`batonode`	`ode45`	• `'Mass'`		—

			ODE with time- and state-dependent mass matrix — motion of a baton	
`brussode`	`ode15s`	• `'JPattern'` • `'Vectorized'`	Stiff large problem — diffusion in a chemical reaction (the Brusselator)	Solve Stiff ODEs
`burgersode`	`ode15s`	• `'Mass'` • `'MStateDependence'` • `'JPattern'` • `'MvPattern'` • `'RelTol'` • `'AbsTol'`	ODE with strongly state-dependent mass matrix — Burgers' equation solved using a moving mesh technique	—
`fem1ode`	`ode15s`	• `'Mass'` • `'MStateDependence'` • `'Jacobian'`	Stiff problem with a time-dependent mass matrix — finite element method	—
`fem2ode`	`ode23s`	• `'Mass'`	Stiff problem with a constant mass matrix — finite element method	—
`hb1ode`	`ode15s`	—	Stiff ODE problem solved on a very long interval — Robertson chemical reaction	—
`hb1dae`	`ode15s`	• `'Mass'` • `'RelTol'` • `'AbsTol'` • `'Vectorized'`	Stiff, linearly implicit DAE from a conservation law — Robertson chemical reaction	Solve Robertson Problem as Semi-Explicit Differential Algebraic Equations (DAEs)
`ihb1dae`	`ode15i`	• `'RelTol'` • `'AbsTol'` • `'Jacobian'`	Stiff, fully implicit DAE — Robertson chemical reaction	Solve Robertson Problem as Implicit Differential Algebraic Equations (DAEs)
`iburgersode`	`ode15i`	• `'RelTol'` • `'AbsTol'` • `'Jacobian'` • `'JPattern'`	Implicit ODE system — Burgers' equation	—
`kneeode`	`ode15s`	• `'NonNegative'`	The "knee problem" with nonnegativity constraints	Nonnegative ODE Solution
`orbitode`	`ode45`	• `'RelTol'` • `'AbsTol'` • `'Events'` • `'OutputFcn'`	Advanced event location — restricted three body problem	ODE Event Location

`rigidode`	`ode45`	—	Nonstiff problem — Euler equations of a rigid body without external forces	Solve Nonstiff ODEs
`vdpode`	`ode15s`	• `'Jacobian'`	Parameterizable van der Pol equation (stiff for large μ)	Solve Stiff ODEs

References

[1] Shampine, L. F. and M. K. Gordon, *Computer Solution of Ordinary Differential Equations: the Initial Value Problem*, W. H. Freeman, San Francisco, 1975.

[2] Forsythe, G., M. Malcolm, and C. Moler, *Computer Methods for Mathematical Computations*, Prentice-Hall, New Jersey, 1977.

[3] Kahaner, D., C. Moler, and S. Nash, *Numerical Methods and Software*, Prentice-Hall, New Jersey, 1989.

[4] Shampine, L. F., *Numerical Solution of Ordinary Differential Equations*, Chapman & Hall, New York, 1994.

[5] Shampine, L. F. and M. W. Reichelt, "The MATLAB ODE Suite," *SIAM Journal on Scientific Computing*, Vol. 18, 1997, pp. 1–22.

[6] Shampine, L. F., Gladwell, I. and S. Thompson, *Solving ODEs with MATLAB*, Cambridge University Press, Cambridge UK, 2003.

See Also

`odeset` | `odextend`

More About

- Solve Nonstiff ODEs
- Solve Stiff ODEs
- Solve Differential Algebraic Equations (DAEs)

External Websites

- Ordinary Differential Equations