Systems of First Order ODEs

eg $\dfrac{du}{dt} = u^2 + v^2 \qquad u(0) = u_0$

$\dfrac{dv}{dt} = \dfrac{uv}{t^2+1} \qquad v(0) = v_0$

i.e $\dfrac{d\vec{y}}{dt} = \vec{F}(\vec{y}(t), t)$, $\vec{y}(0) = \vec{y}_0$

where $\vec{y} = \begin{pmatrix} u \\ v \end{pmatrix}$ & $\vec{F} = \begin{bmatrix} f_1(u, v, t) \\ f_2(u, v, t) \end{bmatrix}$

Euler's method is

$$\vec{y}(t_{k+1}) = \vec{y}(t_k) + h\vec{F}(\vec{y}_k, t_k)$$

$$\begin{cases} u_1 = u_0 + h(u_0^2 + v_0^2) \\ v_1 = v_0 + h\left(\dfrac{u_0 v_0}{t_0^2+1}\right) \end{cases}$$

$$\begin{cases} u_2 = u_1 + h(u_1^2 + v_1^2) \\ v_2 = v_1 + h\left(\dfrac{u_1 v_1}{t_1^2+1}\right) \end{cases}$$

etc.

Higher ODEs

eg $\ddot{\theta} = -\frac{g}{L}\sin(\theta)$, $\theta(0) = A$, $\dot{\theta}(0) = B$

Let $\dot{\theta} = w$ Introduce a new variable

Then $\dot{w} = \ddot{\theta} = -\frac{g}{L}\sin(\theta)$

So we have a system of first order ODEs

$\dot{\theta} = w$ where $\theta(0) = A$

$\dot{w} = -\frac{g}{L}\sin\theta$ $w(0) = B$

$y''' = 2yy'' + t^2 y$ $y(0) = 1$, $y'(0) = 5$, $y''(0) = -6$

$y' = z$

$z' = w$ (so $y'' \equiv w$)

$w' = 2yw + t^2 y$ y'''

and $y(0) = 1$ $z(0) = 5$ $w(0) = -6$

Runge-Kutta Methods

A general single step method is characterized by a number of parameters: $\alpha_i, \beta_{ij}, \gamma_i$

There are s-stages each evaluating $f(y,t)$ for a particular value of t and a value of y obtained by linear combinations of the previous slopes

$$r_i = f\left(y_k + h\sum_{j=1}^{i-1} \beta_{ij} r_j,\ t_k + \alpha_i h\right)$$

for $i = 1, \ldots, s$.

$$y_{k+1} = y_k + h\sum_{i=1}^{s} \gamma_i r_i$$

2nd Order RK Methods

(*) $r_1 = f(y_k, t_k)$ $\quad \alpha_1 = 0$

$r_2 = f\left(y_k + \frac{1}{2}h \cdot r_1,\ t_k + \frac{1}{2}h\right)$ $\quad \alpha_2 = \frac{1}{2}$, $\beta_{2,1} = \frac{1}{2}$

$y_{k+1} = y_k + h r_2$ $\quad \gamma_1 = 0$, $\gamma_1 = 1$

What method is this?

(*) $r_1 = f(y_k, t_k)$

$r_2 = f(y_k + hr_1, t_k + h)$

$y_{k+1} = y_k + h\left(\frac{1}{2}r_1 + \frac{1}{2}r_2\right)$

What method is this?

(*) $r_1 = f(y_k, t_k)$

$r_2 = f\left(y_k + \frac{2}{3}hr_1, t_k + \frac{2}{3}h\right)$

$y_{k+1} = y_k + h\left(\frac{1}{4}r_1 + \frac{3}{4}r_2\right)$

This is called <u>HEUN'S METHOD.</u>

(*) Now, <u>4th order Runge Kutta Method</u>

$$\boxed{y_{k+1} = y_k + h\left(\frac{r_1}{6} + \frac{2r_2}{6} + \frac{2r_3}{6} + \frac{r_4}{6}\right)}$$

$r_1 = f(y_k, t_k)$

$r_2 = f\left(y_k + \frac{1}{2}hr_1, t_k + \frac{1}{2}h\right)$

$r_3 = f\left(y_k + \frac{1}{2}hr_2, t_k + \frac{1}{2}h\right)$

$r_4 = f(y_k + hr_3, t_k + h)$

IMPLICIT vs EXPLICIT
SYSTEM vs SCALAR
STIFF vs NON-STIFF

$$\frac{dy}{dt} = f(y(t), t) \qquad y(t_0) = y_0.$$

IMPLICIT vs EXPLICIT

Explicit Euler: $y_{k+1} = y_k + h \cdot f(y_k, t_k)$

Implicit Euler:

$$y_{k+1} = y_k + h \cdot f(y_{k+1}, t_{k+1})$$

One must solve this implicit equation to find y_{k+1}.

STIFFNESS

Consider $f(y(t), t) = \lambda y(t)$ so ODE is $\frac{dy}{dt} = \lambda y$

Exact Soln: $y(t) = y_0 e^{\lambda(t - t_0)}$

Explicit Euler: $y_{k+1} = y_k + h\lambda y_k = (1 + h\lambda) y_k$

$= (1 + h\lambda)^{k+1} y_0$

Implicit Euler $y_{k+1} = y_k + h\lambda y_{k+1}$

$\Rightarrow (1 - h\lambda) y_{k+1} = y_k$

$\Rightarrow y_{k+1} = (1 - h\lambda)^{-1} \cdot y_k$

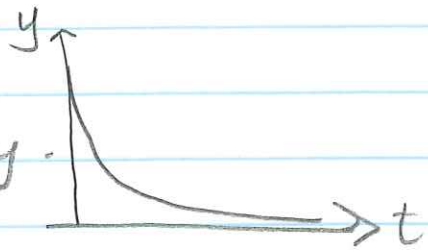
$\Rightarrow y_{k+1} = (1 - h\lambda)^{-(k+1)} y_0$

Suppose $y(t)$ changes quickly.

Consider $\frac{dy}{dt} = \lambda y$ when λ is negative

as a simple example

Exponential decay.



If λ is large & negative then $1+h\lambda$ may be negative and

$$y_{k+1} = (1+h\lambda)^{k+1} y_0$$

will produce oscillating y_k's.

STIFFNESS - numerical instability unless h is very small.

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,

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

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

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

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

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

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).