ODEs: Initial Value Problems

1 Review of ODEs

Now we focus on first-order differential equations, in particular to initial value problems (IVPs) of the form

$$\begin{cases} y'(t) = f(t, y(t)), \quad t \in [a, b], \\ y(a) = y_0. \end{cases} \tag{1}$$

Examples of first-order ODEs include

- Linear ODEs: $y' + p(t)y = g(t)$,
- Separable ODEs: $y' = g(t)h(y)$,
- Exact ODEs: $M(t, y) + N(t, y)y' = 0$.
- Bernoulli ODEs: $y' + p(t)y = q(t)y^n$.

Example 1: For the equation $y'(t) = \cos(t)$, we have general solution $y(t) = \sin(t) + C$.

Example 2: Solve the IVP $y' = 2ty^2$, $y(0) = y_0$. We can use the method of separation of variables to obtain

$$\frac{dy}{y^2} = 2tdt \Rightarrow -\frac{1}{y} = t^2 + C \Rightarrow y(t) = \frac{1}{-t^2 - C}.$$

With the initial condition $y(0) = y_0$, we have $y(t) = \frac{1}{-t^2+1/y_0}$.

Existence and Uniqueness The existence and uniqueness of solutions to the IVP (1) are guaranteed under certain conditions, and we need to check these conditions before solving the IVP, either analytically or numerically.

Theorem 1. *Let $f(t, y)$ and $f_y(t, y)$ be continuous functions of t and y at all points (t, y) in some neighborhood S of the point (t_0, y_0). Then, there is a unique function $y = \phi(t)$ defined on some interval $t \in [t_0 - \delta, t_0 + \delta]$ satisfying* (1).

We can check the example above to see if the conditions are satisfied.

2 Numerical Methods for ODEs

2.1 Notations:

Suppose we want to approximate the solution $y(t)$ of (1) in some interval $t \in [t_0, T]$. Obviously, we can not solve for all t in this interval, so we consider discrete *time* points $t_0 < t_1 < \ldots < t_{n-1} < t_n \leq T$. Our goal is to obtain $y_1 = y(t_1), y_2 = y(t_2), \ldots, y_n = y(t_n)$ as the exact solution (true value) $y(t)$, and we use $\tilde{y}_i, i = 1, \ldots, n$ (or ϕ_i in some cases) to approximate values of the true solution y_i at these $t_1, t_2, \ldots, t_n$ time points. For simplicity, we can assume the times points t_i are equally spaced, so the timestep $t_{i+1} - t_i = h$ for all i indices. We note, the methods we will discuss may not require equally space timesteps. This even spaced time points simplification lead to simpler discussions of analysis in convergence and stability.

In general, if we integrate the equation in (1) from t_n to t_{n+1}, we have

$$\int_{t_n}^{t_{n+1}} y'(t)dt = \int_{t_n}^{t_{n+1}} f(t, y(t))dt \Longrightarrow y(t_{n+1}) = y(t_n) + \int_{t_n}^{t_{n+1}} f(t, y(t))dt. \tag{2}$$

2.2 (Forward) Euler's Method

The simplest method to the integral in (2) is to use a constant for the $f(t, y(t))$, for example we let the constant to be $f(t_n, y(t_n))$.

$$y(t_{n+1}) \approx y(t_n) + (t_{n+1} - t_n) f(t_n, y(t_n)).$$

And this is the (forward or explicit) Euler's method.

If we use ϕ_n to denote the approximate solution of the IVP at t_n, then the Euler's method has formula

$$\phi_{n+1} = \phi_n + (t_{n+1} - t_n) f(t_n, \phi_n). \tag{3}$$

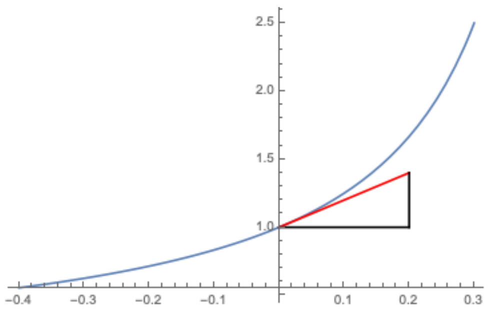
We note this is one step method, and it is *explicit in time* method.

Example: Solve the IVP $y' = (-2t + 1)y$, $y(0) = 1$ using forward Euler's method with $h = 0.2$. (fixed h means equally spaced time: $t_{n+1} - t_n = h$)

Answer: We have $f(t_n, \phi_n) = (1 - 2t_n)\phi_n$, so

$$\phi_{n+1} = \phi_n + h(1 - 2t_n)\phi_n = \phi_n(1 + h(1 - 2t_n)).$$

$t_0 = 0$, $t_1 = 0.2$, $t_2 = 0.4$, and $t_3 = 0.6$



n	t_n	$f(t_n, \phi_n) = (1 - 2t_n)\phi_n$	$\phi_{n+1} = \phi_n + (t_{n+1} - t_n)f(t_n, \phi_n)$
0	0	1	–
1	0.2	$(1 - 2 \cdot 0.2) \cdot 1.2 = 0.72$	$1 + 0.2(1) = 1.2$
2	0.4	$(1 - 2 \cdot 0.4) \cdot 1.344 = 0.2688$	$1.2 + 0.2(0.72) = 1.344$

We should avoid using Euler's method for stiff problems, where the solution changes rapidly.

2.3 Backward Euler's Method

Recall the integral in (2), $y(t_{n+1}) = y(t_n) + \int_{t_n}^{t_{n+1}} f(t, y(t))dt$, we can approximate the integral by using the value of $f(t_{n+1}, y(t_{n+1}))$ instead of $f(t_n, y(t_n))$, and this leads to the (backward or implicit) Euler's method,

$$y(t_{n+1}) \approx y(t_n) + (t_{n+1} - t_n)f(t_{n+1}, y(t_{n+1})). \tag{4}$$

If we use ϕ_n to denote the approximate solution of the IVP at t_n, then the backwards Euler's method has formula is

$$\phi_{n+1} = \phi_n + (t_{n+1} - t_n) f(t_{n+1}, \phi_{n+1}). \tag{5}$$

We note this is one step method (the method only involves values at steps n and $n+1$), and it is *implicit in time* method. The implicit nature of the method makes it more stable than the explicit methods. But we **may need to solve a nonlinear equation** to get the value ϕ_{n+1}. If (5) is a nonlinear equation, we can use Newton's method to solve it: we introduce a function $F(\phi_{n+1}) = \phi_n + hf(t_{n+1}, \phi_{n+1}) - \phi_{n+1}$, and we solve $F(\phi_{n+1}) = 0$ using Newton's method. The iteration is $\phi_{n+1}^{(0)} = \phi_n$ and

$$\phi_{n+1}^{(k+1)} = \phi_{n+1}^{(k)} - \frac{F(\phi_{n+1}^{(k)})}{F'(\phi_{n+1}^{(k)})},$$

where $F'(z)$ is obtained by taking the derivative of $F(z) = \phi_n + hf(t_{n+1}, z) - z$ with respect to z. This gives $F'(z) = h\frac{\partial f(t_{n+1}, z)}{\partial z} - 1$.

For simple example (this does not need to solve nonlinear equation to get ϕ_{n+1}), if we have $y' = (1-2t)y$, $y(0) = 1$, then the backward Euler's method is (we get linear equation for ϕ_{n+1} in this case)

$$\phi_{n+1} = \phi_n + h(1 - 2t_{n+1})\phi_{n+1} \Rightarrow \phi_{n+1} = \frac{\phi_n}{1 - h(1 - 2t_{n+1})}.$$

2.4 Error Analysis

We can define the **local truncation error** or the **one step error** at time t_n, as the error made in one step of the method, i.e., the error made in approximating $y(t_{n+1})$ by ϕ_{n+1}, as we know the true solution $y(t_n)$ to compute the one step approximation ϕ^*_{n+1},

$$\tau_{n+1} = y(t_{n+1}) - \phi^*_{n+1}.$$

For example, if we consider the forward Euler's method in (3), then the ϕ^*_{n+1} is (note here $y(t_n)$ is the true solution value at t_n)

$$\phi^*_{n+1} = y(t_n) + hf(t_n, y(t_n)).$$

The local truncation error is then ($y(t_{n+1})$ is the true solution value at t_{n+1})

$$\tau_{n+1} = y(t_{n+1}) - \phi^*_{n+1} = y(t_{n+1}) - y(t_n) - hf(t_n, y(t_n)).$$

Local Truncation Error Analysis for Forward Euler's Method

We can expand the true solution $y(t_{n+1})$ in a Taylor series about t_n,

$$y(t_{n+1}) = y(t_n) + hy'(t_n) + \frac{h^2}{2}y''(\xi).$$

We can subtract the two expansions to get the local truncation error,

$$\tau_{n+1} = y(t_{n+1}) - \phi^*_{n+1} = \frac{h^2}{2} y''(\xi)$$

This shows that the local truncation error of the forward Euler's method has the property

$$|\tau_{n+1}| \leq \frac{1}{2} M_n h^2 \sim O(h^2).$$

We say the forward Euler's method is a **locally second order accurate** method, as the **local truncation error** (LTE) is of order $O(h^2)$.

We also want to introduce the **global error** for numerical solutions to ODEs. Suppose we have a sequence of approximations ϕ_n at t_n to the true solution $y(t_n)$ of the IVP (1), for time interval $[a, b]$, with $t_0 = a, t_1, \ldots, t_n = b$. Then the global discretization error E_n at time t_n is defined as

$$E_n = y(t_n) - \phi_n.$$

We note the error E_n at the end of the interval is usually called the **final global error**.

And particularly the global error is the maximum of the errors at all time points,

$$E = \max_{0 \leq n \leq N} |E_n|.$$

This error is a measure of how accurate ϕ_i agree with the true solution over the whole interval.

While one can derive the global error for a method, the local truncation error tends to be relatively easy to get. In general, the order of error for the global truncation error is one order lower than the local truncation error.

From the concept of global error, we can derive that the forward Euler's method is a first order method in terms of global error: $E_n = O(h)$. We can see this conclusion from the *error analysis.*

As for each n, we have the local error $\tau_{k+1} = y''(\xi_k)h^2/2$ derived above. **Roughly speaking**, we sum up the local errors for $k = 1, \ldots, n$, we get the accumulated error upto time t_n as

$$\sum_{k=1}^{n} |\tau_k| \leq \sum_{k=1}^{n} y''(\xi_k)h^2 \leq ny''(c)h^2 \leq nMh^2$$

where M is the maximum value of $y''(t)$ on $[a, b]$. We also note that $n \approx (b-a)/h$, so the **global error** is $O(h)$, which is first order accurate.

2.5 Convergence, consistency and stability

Definition: a numerical method is called **consistent** if the LTE at each step is at least $o(h)$ as $h \to 0$. This means we need

$$\lim_{h\to 0} \max_{0\leq n\leq N} \frac{\tau_n}{h} = 0.$$

Definition: A numerical method is said to be **convergent** if the global error E goes to zero as the stepsize h goes to zero. (This means all E_n go to zero as $h \to 0$.)

Example: for IVP $y' = f(t, y), y(0) = y_0$, what is the local order of error of the following explicit two-step method, and is it consistent?

$$\phi_{n+1} + 9\phi_n - 10\phi_{n-1} = \frac{h}{2}(13f(t_n, \phi_n) + 9f(t_{n-1}, \phi_{n-1})).$$

You can derive it, and the error is 2 and it is consistent method.

Stability: A consistent method might not be convergent because of the way numerical errors accumulate over time steps. For example, consider the method above multi-step method. It is consistent with order 2. However, it is unstable. An easy way to see it is to apply it to solving the IVP $y' = 0, y(0) = a$. To initiate the method, we need two initial values $u(0) = u_0$ and $u(h) = u_1$. Since the right-hand side is zero, the method becomes the following linear recurrent relationship:

$$\phi_{n+1} + 9\phi_n - 10\phi_{n-1} = 0$$

The general solution to above is $\phi_n = Ar_1^n + Br_2^n$ where r_1 and r_2 are the roots to the characteristic equation $r^2 + 9r - 10 = 0$, i.e., $r_1 = 1, r_2 = -10$. In order to obtain the constant solution $\phi_n = a$, we need $\phi_1 = \phi_2 = a$. Then $A = a$ and $B = 0$. If either of these values ϕ_1 or ϕ_2, will be slightly perturbed, the coefficient B will be nonzero and hence the solution will blow up. Note that the smaller the time step h will be, the more the solution will blow up over a fixed interval of time.

This example shows that besides requiring that the errors committed at each time step be small, they also need to accumulate stably.

Indeed, we will need the so called zero-stability condition to ensure the stability of the method: for IVP $y' = f(t, y), y(t_0) = y_0$, if $\{\phi_n\}$ and $\{\psi_n\}$ are the numerical solutions to the IVP with the initial conditions ϕ_0 and ψ_0, if it holds that

$$\max_{0 \le n \le N} |\phi_n - \psi_n| \le C \max_{0 \le n \le N} |\phi_0 - \psi_0|,$$

where C is a constant independent of h (so independent of n), then the method is said to be zero-stable.

Convergence Theorem: A typical numerical method is convergent if it is zero-stable and consistent.

Heun's Method

Heun's method is a two-stage method, and it is also known as the improved Euler's method. We can still use the integral in (2) $y(t_{n+1}) = y(t_n) + \int_{t_n}^{t_{n+1}} f(t, y(t))dt$ to start. But now we want to use the trapezoidal rule to approximate the integral, and this leads to $y(t_{n+1}) \approx y(t_n) + \frac{h}{2}(f(t_n, y(t_n)) + f(t_{n+1}, y(t_{n+1}))$. The term $f(t_{n+1}, y(t_{n+1}))$ is unknown, but we can approximate it using the forward Euler's method. If we use ϕ_n to denote the approximate solution of the IVP at t_n, then the **Heun's method** has formula

$$\phi_{n+1} = \phi_n + \frac{h}{2}(f(t_n, \phi_n) + f(t_{n+1}, \phi_n + hf(t_n, \phi_n))). \tag{6}$$

The Heun method is two stage method, and it is *explicit in time* method. It is also called midpoint method. We note it can be broken down to two steps: a **predictor** step and a **corrector** step. The predictor step is

$$\phi^*_{n+1} = \phi_n + hf(t_n, \phi_n),$$

and the corrector step is

$$\phi_{n+1} = \phi_n + \frac{h}{2}(f(t_n, \phi_n) + f(t_{n+1}, \phi^*_{n+1})).$$

The Heun method has locally 3rd order accurate, as the trapezoidal rule is used in approximating the integral. So the method is 2nd order accurate in terms of global error. This is also called improved Euler's method.

3 Runge-Kutta Methods

3.1 Introduction

The Runge-Kutta methods are a family of methods that are based on the idea of using several stages to approximate the integral in (2). Indeed both Euler's method and Heun's method are special cases of the Runge-Kutta methods. The Runge-Kutta method for advancing from t_n to t_{n+1} is to use multiple stages (f at intermiediate time points between t_n and t_{n+1}) to approximate the integral in (2), and it has the general form

$$y(t_{n+1}) \approx y(t_n) + h \sum_{i=1}^{s} b_i k_i, \tag{7}$$

where k_i are the slopes (f) at the stages $t_n + c_i h$, and b_i are the weights. The slopes k_i are computed as

$$k_i = f(t_n + c_i h, y(t_n) + h \sum_{j=1}^{i-1} a_{ij} k_j),$$

where a_{ij} are the coefficients (note to make sure k_i are at the right estimates, we need to ensure $c_i = \sum_{j=1}^{i-1} a_{ij}$). The Runge-Kutta method is explicit if the coefficients a_{ij} are zero for $i \geq j$, and it is implicit if the coefficients a_{ij} are non-zero for $i \geq j$. It is important to note that the sumation in (7) is to approximate the integral in (2): $y_{n+1} = y_n + \int_{t_n}^{t_{n+1}} f(t, y(t))$.

To approximate the integral, we have the goal (the Runge-Kutta method) to use function evaluations

$$f(t_h a, y + hb)$$

to replace the derivatives $y''_n, y'''n, \cdots$ and get a method with the desired order of accuracy. We note the number of stages as explained above is the number of function evaluations in the formula. Why? Let's see.

Recall that Euler's method was derived by expanding y_{n+1} in a Taylor series:

$$y_{n+1} = y_n + hy'(t_n) + O(h^2) \Longrightarrow y_{n+1} = y_n + hf(t_n, y_n) + O(h^2).$$

Since $y'(t) = f(t, y(t))$, we can derive a one step method (involving function evaluations only at t_N) by using

$$y_{n+1} = y_n + hy'(t_n) + \cdots \frac{h^p}{p!} y^{(p)}(t_n) + O(h^{p+1}).$$

We want to point out

$$y'' = \frac{d}{dt}(f(t, y)) = \frac{\partial f}{\partial t} + \frac{\partial f}{\partial y} y' = f_t + f_y f.$$

We can see if we want to compute higher derivatives for y, there will be some ugly formula to compute. And if you do, then you can derive the **Taylor series method** for the ODE.

3.2 Simple Runge-Kutta Methods

Now we see that $y' = f(t_n, y_n)$ are reasonable to compute, but $y'' = f_t + f_y f$ are not. The Runge-Kutta methods are designed to approximate the integral in (2) by using interpolant of $f(t, y(t))$ at several stages between t_n and t_{n+1}, and the interpolant is constructed by using the slopes k_i at these stages. The simplest Runge-Kutta method is the second order Runge-Kutta method, which is also known as the midpoint method. The method can be illustrated as the following

$$f_1 = f(t_n, y_n),\ \ f_2 = f(t_n + ch, y_n + haf_1),$$

$$y_{n+1} = y_n + h(b_1 f_1 + b_2 f_2).$$

with constants b_1, b_2, a, c to be determined. We note the method is second order accurate and the term $h(b_1 f_1 + b_2 f_2)$ is the approximation of the integral in $y_{n+1} = y_n + \int_{t_n}^{t_{n+1}} f(t, y(t))dt$.

The goal is to choose the constants so that

$$LHS = \frac{1}{h}(y_{n+1} - y_n) = b_1 f_1 + b_2 f_2 + \tau_{n+1} = RHS,\ \ \tau_{n+1} \sim O(h^2). \qquad (8)$$

The strategy is to expan in a Taylor series around t_n and y_n to get the terms of f and its derivatives and **match terms**!

For LHS
$$\begin{aligned}\frac{1}{h}(y_{n+1} - y_n) =& y'_n + \frac{h}{2}y''_n + O(h^2)\\ =& f + \frac{h}{2}(f_t + f_y f) + O(h^2)\end{aligned}$$

For the RHS, first expand f_2 using a Taylor series around (t_n, y_n):

$$\begin{aligned}f_2 =& f_n + (ch)\frac{\partial f(t_n, y_n)}{\partial t} + (haf_1)\frac{\partial f(t_n, y_n)}{\partial y} + O(h^)\\ =& f + chf_t + haff_y + O(h^2)\end{aligned}$$

Then we can get the RHS:

$$RHS = b_1 f + b_2(f + chf_t + hbff_y) + \tau_{n+1} + O(h^2)$$

Both LHS and RHS of equation (8) are now available and we can match the terms to get the coefficients b_1, b_2:

$$\begin{aligned}b_1 + b_2 =& 1,\\ cb_2 = ab_2 =& \frac{1}{2}\end{aligned}$$

If we choose $b_2 = \theta$, then $b_1 = 1 - \theta$, $a = c = 1/(2\theta)$, so for any $\theta \neq 0$, we have

$$\begin{aligned}
f_1 =& f(t_n, y_n),\\
f_2 =& f(t_n + h, y_n + h\theta f_1),\\
y_{n+1} =& y_n + h((1-\theta)f_1 + \theta f_2).
\end{aligned}$$

The most popular choice is $\theta = 1/2$, and we can get the coefficients $b_1 = \frac{1}{2}$, $b_2 = \frac{1}{2}$, $a = 1$, $c = \frac{1}{2}$.

3.3 Typical Runge-Kutta Methods

We note the general form of the Runge-Kutta method in (7), and we can see that the method is determined by the coefficients a_{ij}, b_i, c_i. The most popular Runge-Kutta methods are the second order Runge-Kutta method (midpoint method), the fourth order Runge-Kutta method (RK4), and the fifth order Runge-Kutta method (RK5).

$$\begin{aligned}
f_1 =& f(t_n, \phi_n),\\
f_2 =& f(t_n + c_2 h, \phi_n + h a_{21} f_1),\\
f_3 =& f(t_n + c_3 h, \phi_n + h a_{31} f_1 + h a_{32} f_2),\\
&\vdots\\
f_m =& f(t_n + c_m h, \phi_n + h a_{m1} f_1 + h a_{m2} f_2 + \cdots + h a_{m,m-1} f_{m-1}),\\
\phi_{n+1} =& \phi_n + h(b_1 f_1 + b_2 f_2 + \cdots + b_m f_m).
\end{aligned}$$

And this can be summarized in a tableau as

$$\begin{array}{c|cccc}
c_1 & & & & \\
c_2 & a_{21} & & & \\
c_3 & a_{31} & a_{32} & & \\
\vdots & \vdots & \vdots & \ddots & \\
c_m & a_{m1} & a_{m2} & \cdots & a_{m,m-1} \\
\hline
 & b_1 & b_2 & \cdots & b_m
\end{array}$$

The second order Runge-Kutta method is the midpoint method, and it has the following coefficients:

$$\begin{array}{c|cc}
0 & 0 & 0 \\
1/2 & 1/2 & 0 \\
\hline
 & 0 & 1
\end{array}$$

The second order Runge-Kutta method is

$$f_1 = f(t_n, y_n),$$

$$f_2 = f(t_n + h/2, y_n + hf_1/2),$$

$$y_{n+1} = y_n + hf_2.$$

The second order Runge-Kutta method is 2nd order accurate in terms of global error.

The most popular Runge-Kutta method is the fourth order Runge-Kutta method (RK4), which is explicit one step method and has the following coefficients:

$$\begin{array}{c|cccc}
0 & 0 & 0 & 0 & 0 \\
1/2 & 1/2 & 0 & 0 & 0 \\
1/2 & 0 & 1/2 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 \\
\hline
 & 1/6 & 1/3 & 1/3 & 1/6
\end{array}$$

The fourth order Runge-Kutta method is

$$f_1 = f(t_n, y_n),$$

$$f_2 = f(t_n + h/2, y_n + hf_1/2),$$

$$f_3 = f(t_n + h/2, y_n + hf_2/2),$$

$$f_4 = f(t_n + h, y_n + hf_3),$$

$$y_{n+1} = y_n + h(f_1 + 2f_2 + 2f_3 + f_4)/6.$$

Here the integral $\int_{t_n}^{t_{n+1}} f(t, y(t))dt$ is approximated by the sum $h(f_1 + 2f_2 + 2f_3 + f_4)/6$, which is the approximated Simpson's rule and has local truncation error of order $O(h^5)$. So the RK4 method is 4th order accurate in terms of global error.

3.4 Implicit Runge-Kutta Methods

The implicit Runge-Kutta methods are constructed by using the implicit trapezoidal rule to approximate the integral in (2). The implicit Runge-Kutta methods are more stable than the explicit Runge-Kutta methods, but they are more expensive to compute as they require the solution of nonlinear equations at each stage. The implicit Runge-Kutta methods are usually used for stiff ODEs.

Multistep Methods: Adams-Bashforth Methods If we check again (2),

$$y(t_{n+1}) = y(t_n) + \int_{t_n}^{t_{n+1}} f(t, y(t))dt,$$

we can see that the integral is indeed $\int_{t_n}^{t_{n+1}} f(t, y(t))dt = \int_{t_n}^{t_{n+1}} y'(t)dt$ due to the definition of the ODE $y'(t) = f(t, y(t))$. We can approximate $y'(t), t_n \leq t \leq t_{n+1}$ in the integral by constucting a polynomial $P_r(t), t_n \leq t \leq t_{n+1}$ of degree r that interpolates the pairs $(t_i, y(t_i))$ for $i = n, n-1, \ldots, n-r$. This will lead to the multistep methods.

Consider the simple case of $r = 1$. We set up $P_1(t) = At + B$ to interpolate the points (t_n, y_n) and (t_{n-1}, y_{n-1}). We have

$$P_1(t_n) = y'_n = f(t_n, y_n) = At_n + B,\ P_1(t_{n-1}) = f(t_{n-1}, y_{tn-1}) = y'_{n-1} = At_{n-1} + B.$$

Let $f_n = f(t_n, y_n)$ and $f_{n-1} = f(t_{n-1}, y_{n-1})$. Solving the above equations, we get

$$A = \frac{f_n - f_{n-1}}{t_n - t_{n-1}},\ B = \frac{f_{n-1}t_n - f_n t_{n-1}}{t_n - t_{n-1}}.$$

We should note given $P_1(t) = At + B$, the integral

$$\int_{t_n}^{t_{n+1}} y'(t)dt \approx \int_{t_n}^{t_{n+1}} P_1(t)dt = \frac{A}{2}(t_{n+1}^2 - t_n^2) + B(t_{n+1} - t_n)$$

If even spaced time points are used, $t_n - t_{n-1} = t_{n+1} - t_n = h$, and the integral above is reduced to

$$\int_{t_n}^{t_{n+1}} P_1(t)dt = h\left(\frac{3}{2}f_n - \frac{1}{2}f_{n-1}\right).$$

This leads to the two-step Adams-Bashforth method, which is

$$y_{n+1} = y_n + \frac{h}{2}(3f(t_n, y_n) - f(t_{n-1}, y_{n-1})).$$

We note that the Adams-Bashforth method is explicit in time, and it is a two-step method (uses two previous time points). The method is 2nd order accurate in terms of global error.

Multistep Methods: Adams-Moulton Methods The Adams-Moulton methods are implicit in time, and they are constructed by using the polynomial $P_1(t)$ to interpolate the points (t_{n+1}, y_{n+1}) and (t_n, y_n). The second order Adams-Moulton method is

$$y_{n+1} = y_n + \frac{h}{2}(f(t_{n+1}, y_{n+1}) + f(t_n, y_n)).$$

The method is 2nd order accurate in terms of global error, and it is implicit in time. This is also called the trapezoidal method. We note in general the implicit Moulton method has $r + 1$ order of accuracy, with r being the steps involved.

Predictor Corrector Methods

A strategy of combining the explicit and implicit methods is to use a predictor-corrector pair. For example, we can use the explicit Adams-Bashforth method to predict the value of at t_{n+1}, and denote it by ϕ^*_{n+1}, and then use the implicit Adams-Moulton method to correct the value of y_{n+1}, and denote it by ϕ_{n+1}. The predictor-corrector pair is

$$\phi^*_{n+1} = \phi_n + \tfrac{h}{2}(3f(t_n, \phi_n) - f(t_{n-1}, \phi_{n-1})),$$

$$\phi_{n+1} = \phi_n + \tfrac{h}{2}(f(t_{n+1}, \phi^*_{n+1}) + f(t_n, \phi_n)).$$

This predictor-corrector pair is 2nd order accurate in terms of global error, and it is *explicit* in time!

A more involved predictor-corrector pair is the Adams-Bashforth-Moulton method, which is a combination of the Adams-Bashforth and Adams-Moulton methods. We illustrate the two-step as follows. First we use the Adams-Bashforth method to predict the value at t_{n+1}:

$$\phi^*_{n+1} = \phi_n + \frac{h}{24}(55f(t_n, \phi_n) - 59f(t_{n-1}, \phi_{n-1}) + 37f(t_{n-2}, \phi_{n-2}) - 9f(t_{n-3}, \phi_{n-3})),$$

and then we use the Adams-Moulton method to correct the value at t_{n+1}:

$$\phi_{n+1} = \phi_n + \frac{h}{24}(9f(t_{n+1}, \phi_{n+1}) + 19f(t_n, \phi_n) - 5f(t_{n-1}, \phi_{n-1}) + f(t_{n-2}, \phi_{n-2})).$$

This predictor-corrector pair is 4th order accurate in terms of global error, and it is *implicit* in time! The corrector step solved by a fixed number of fixed point iterations, using the solution ϕ^*_{n+1} by the explicit method as starting values for the iterations.

The standard procedure of using the Adams-Bashforth-Moulton method with given order r is to use several steps of RK methods to get the initial values, and then do the predictor and corrector update. The explicit predictor use the Adams-Bashforth method, and the implicit corrector use the Adams-Moulton method.

Systems of ODEs For systems of ODEs, we can use the same methods as for scalar ODEs. For example, for the system

$$y_1' = f_1(t, y_1, y_2),$$

$$y_2' = f_2(t, y_1, y_2),$$

we can use the RK4 method to advance the solution from t_n to t_{n+1} as follows:

$$k_{11} = f_1(t_n, y_{1n}, y_{2n}),\ k_{21} = f_2(t_n, y_{1n}, y_{2n}),$$

$$k_{12} = f_1(t_n + \tfrac{h}{2}, y_{1n} + \tfrac{h}{2}k_{11}, y_{2n} + \tfrac{h}{2}k_{21}),\ k_{22} = f_2(t_n + \tfrac{h}{2}, y_{1n} + \tfrac{h}{2}k_{11}, y_{2n} + \tfrac{h}{2}k_{21}),$$

$$k_{13} = f_1(t_n + \tfrac{h}{2}, y_{1n} + \tfrac{h}{2}k_{12}, y_{2n} + \tfrac{h}{2}k_{22}),\ k_{23} = f_2(t_n + \tfrac{h}{2}, y_{1n} + \tfrac{h}{2}k_{12}, y_{2n} + \tfrac{h}{2}k_{22}),$$

$$k_{14} = f_1(t_n + h, y_{1n} + hk_{13}, y_{2n} + hk_{23}),\ k_{24} = f_2(t_n + h, y_{1n} + hk_{13}, y_{2n} + hk_{23}),$$

$$y_{1n+1} = y_{1n} + \tfrac{h}{6}(k_{11} + 2k_{12} + 2k_{13} + k_{14}),\ y_{2n+1} = y_{2n} + \tfrac{h}{6}(k_{21} + 2k_{22} + 2k_{23} + k_{24}).$$

We note that the RK4 method is 4th order accurate in terms of global error for systems of ODEs. And the method is explicit in time, which makes the implementation much easier. For *harder* problems (stiff problems, whose solutions change rapidly), we can use the implicit methods, which are more stable than the explicit methods.

Converting higher order ODEs to system of 1st order eqns

For higher order ODEs, we can convert them to systems of first order ODEs. For example, for the second order ODE

$$u''(t) = f(t, u(t), u'(t)), \quad u(a) = u_0, \quad u'(a) = u_1,$$

we can introduce a new variable $y_1(t) = u(t)$, $y_2(t) = u'(t)$, and then we have the system

$$y_1'(t) = y_2(t),$$

$$y_2'(t) = f(t, y_1(t), y_2(t)).$$

This is a system of first order ODEs, and we can use the RK4 method to solve it. We can also use the Adams-Bashforth-Moulton method to solve the system.

Example 1, for the second order ODE (Van der Pol's equation)

$$u'' + \mu(u^2 - 1)u' + u = 0$$

we can convert it to the system

$$y_1'(t) = y_2(t),$$

$$y_2'(t) = -\mu(y_1^2 - 1)y_2 - y_1.$$

Example 2, for the given system of equations

$$y_1' = -2y_1 + y_2, y_2' = 998y_1 - 999y_2,\ y_1(0) = 1,\ y_2(0) = 1,$$

We can try to use Euler's method and other methods to solve the system, and compare the results (check $h = 0.00210$ and $h = 0.00200$) We note the eigenvalues of the matrix $A = \begin{bmatrix} -2 & 1 \\ 998 & -999 \end{bmatrix}$ are $\lambda_1 = -1000$ and $\lambda_2 = -1$, and the system is **stiff**.

Using Matlab's built-in function for solving system of ODEs

Matlab has a built-in function `ode45` for solving systems of ODEs. The function `ode45` uses the Runge-Kutta method to solve the system of ODEs. The function is called as

$$[\mathtt{t},\mathtt{y}] = \mathtt{ode45}(@\mathtt{f},[\mathtt{a},\mathtt{b}],\mathtt{y_0}),$$

where `@f` is the function that defines the system of ODEs, $[a, b]$ is the interval of the solution, and y_0 is the initial value of the system. The function `@f` should be defined as

$$\mathtt{functiondydt} = \mathtt{f}(\mathtt{t},\mathtt{y}),$$

where `dydt` is the derivative of the system of ODEs. For example, for the system

$$y_1' = -2y_1 + y_2,$$

$$y_2' = 998y_1 - 999y_2,$$

$$y_1(0) = 1,\ y_2(0) = 1,$$

the function `@f` should be defined as

$$\mathtt{function\ dydt} = \mathtt{f}(\mathtt{t},\mathtt{y}), \mathtt{dydt} = [-2 * \mathtt{y}(1) + \mathtt{y}(2); 998 * \mathtt{y}(1) - 999 * \mathtt{y}(2)];$$

The function `ode45` returns the solution y at the time points t. The solution y is a matrix with the first column being the solution of the first ODE, and the second column being the solution of the second ODE. The time points t are stored in the vector t. We can plot the solution using the command `plot(t,y)`.

For stiff problems, we can use the function `ode15s`, which is a stiff solver. The function `ode15s` uses the implicit methods to solve the system of ODEs. The function `ode15s` is called as

$$[\mathtt{t}, \mathtt{y}] = \mathtt{ode15s}(@\mathtt{f}, [\mathtt{a}, \mathtt{b}], \mathtt{y}_0).$$

The function `@f` is defined as before.

We note that the function `ode45` is a general purpose solver. The function implements a Runge-Kutta method, and it is explicit in time. The function use adaptive time stepping to solve the system of ODEs. The method is called embedded method, as it uses two Runge-Kutta methods of order 4 and 5 to solve the system of ODEs. The method is 4th order accurate in terms of global error.

If we have two method of different order of accuracy, we can use the difference between the two methods to estimate the error of the solution. The difference between the two methods is called the **error estimate**. The error estimate is used to adjust the time step size in the adaptive time stepping method.

The error estimate is used to control the error of the solution. For example,

$$y \approx y^{(p)} + \mathcal{O}(h^{p+1}), y \approx y^{(p+1)} + \mathcal{O}(h^{p+2}),$$

then the error estimate is

$$\text{error estimate} = |y^{(p+1)} - y^{(p)}| = \mathcal{O}(h^{p+1}) = Ch^{p+1}.$$

We want to calculate the new time step size h_{new} such that the error estimate is less than the tolerance tol, i.e., $Ch^{p+1} < \text{tol}$. We can solve for h_{new} to get $h_{\text{new}} = (C/\text{tol})^{1/(p+1)}h$. We can use the new time step size h_{new} to solve the system of ODEs. The adaptive time stepping method is used to control the error of the solution. This is called the **adaptive time stepping method**.