# ODEs: Initial Value Problems

**Review 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.

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 $y_1, y_2, \ldots, y_n$ to approximate values of the true solution $y(t)$ 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

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

**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 $\phi_n$ to denote the approximate solution of the IVP at $t_n$, then the Euler's method has formula

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

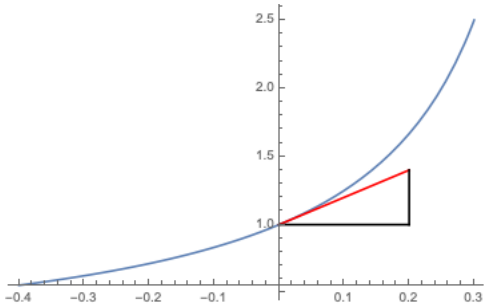
We note this is one step method, and it is *explicit in time* method.

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.

**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 $\phi_n$ to denote the approximate solution of the IVP at $t_n$, then the backwards Euler's method has formula is

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

We note this is one step method, and it is *implicit in time* method. The implicit nature of the method makes it more stable than the explicit method. But we may need to solve a nonlinear equation to get the value $\phi_{n+1}$.

For example, if we have $y' = (1 - 2t)y$, $y(0) = 1$, then the backward Euler's method is (we get linear equation for $\phi_{n+1}$ in this case)

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

**Error Analysis**

Given the true solution $y(t)$ of the IVP, we can define the **global truncation error** at time $t_n$, between the true solution and the approximate solution, as

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

We can also 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 $\phi_{n+1}$, as we know the true solution $y(t_n)$ to compute

the one step approximation $\phi^*_{n+1}$,

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

For example, if we consider the forward Euler's method in (3), then the $\phi^*_{n+1}$ is

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

The local truncation error is then

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

**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 is of order $O(h^2)$, as

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

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

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

From this, we can see that the forward Euler's method is a first order method in terms of global error: $E_n = O(h)$. We can see this from the error analysis, as for each $n$, we have error local error $\tau_{k+1} = y''(\xi_k)h^2/2$. When 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.

For ODE algorithm, we usually analyze the error for a simple ODE, for example, $y' = \lambda y + g(t)$, $y(0) = y_0$. We can apply the forward Euler's method to this simple ODE, and then analyze the error.

**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 $\phi_n$ to denote the approximate solution of the IVP at $t_n$, then the **Heun's method** has formula

$$\phi_{n+1} = \phi_n + \frac{h}{2}(f(t_n, \phi_n) + f(t_{n+1}, \phi_n + hf(t_n, \phi_n))). \tag{5}$$

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.

**Runge-Kutta Methods**

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 to approximate the integral in (2), and it has the general form

$$y(t_{n+1}) \approx y(t_n) + h \sum_{i=1}^{s} b_i k_i, \tag{6}$$

where $k_i$ are the slopes 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. 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 (6) is to approximate the integral in (2).

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

efficients:

| | | | | |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1/2 | 1/2 | 0 | 0 | 0 |
| 1/2 | 0 | 1/2 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| | 1/6 | 1/3 | 1/3 | 1/6 |

The fourth order Runge-Kutta method is

$$k_1 = f(t_n, y_n),$$

$$k_2 = f(t_n + h/2, y_n + hk_1/2),$$

$$y_{n+1} = y_n + h(k_1 + 2k_2 + 2k_3 + k_4)/6.$$

$$k_3 = f(t_n + h/2, y_n + hk_2/2),$$

$$k_4 = f(t_n + h, y_n + hk_3),$$

Here the integral $\int_{t_n}^{t_{n+1}} f(t, y(t))dt$ is approximated by the sum $h(k_1 + 2k_2 +$

$2k_3+k_4)/6$, 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.

**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.

**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 $\phi^*_{n+1}$, and then use the implicit Adams-Moulton method to correct the value of $y_{n+1}$, and denote it by $\phi_{n+1}$. 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 $\phi^*_{n+1}$ 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 $h_{\text{new}}$ such that the error estimate is less than the tolerance tol, i.e., $Ch^{p+1} < \text{tol}$. We can solve for $h_{\text{new}}$ to get $h_{\text{new}} = (C/\text{tol})^{1/(p+1)}h$. We can use the new time step size $h_{\text{new}}$ 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**.