Tuesday, 12/3/13 Math 471, Fall 2013, Section 001 Topic 8

1 Ordinary Differential Equations (ODEs) and Initial Value Problems (IVP)

1.1 First-order IVP

Text : sections 7.0 - 7.2

In this chapter we address the initial value problem for first-order differential equations of the form

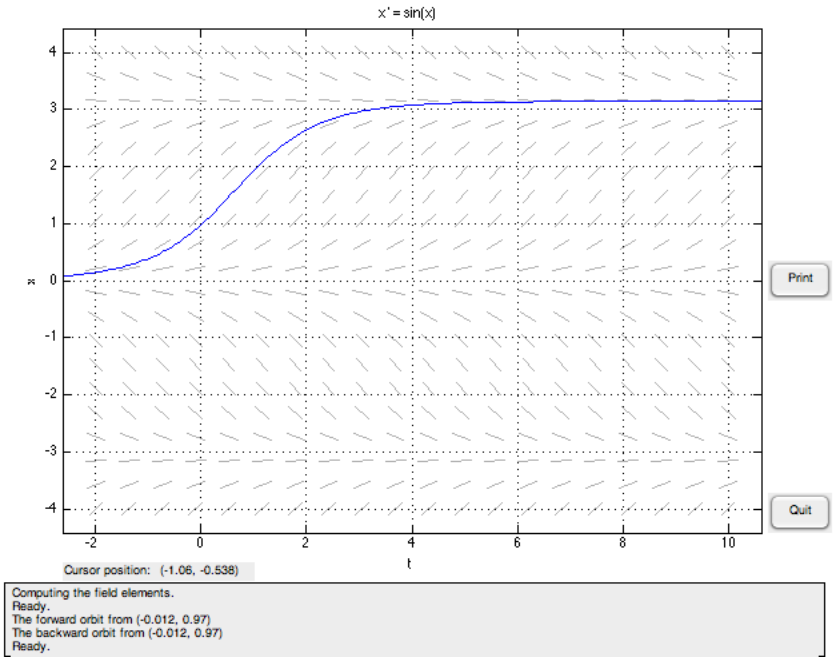
$$y'(t) = f(y(t), t), \qquad y(t_0) = \alpha.$$

The left term is the differential equation, which holds for $t > t_0$, and the right term $y(t_0) = \alpha$ represents the initial data. We will assume $t_0 = 0$ for simplicity in most of our work. ODEs whose right-hand sides do not depend explicitly on t (only on y) are called autonomous.

scalar equations

Let $x(t)$ represent the position of a particle moving on the x-axis at time t. Then $\frac{dx}{dt} = f(x)$ expresses velocity as a function of position, and $x(0) = \alpha$ is the initial datum. The goal is to find $x(t)$ for $t > 0$.

1. $\frac{dx}{dt} = x, \quad x(0) = 1 \Rightarrow x(t) = e^t$
2. $\frac{dx}{dt} = x^2, \quad x(0) = 1 \Rightarrow x(t) = \frac{1}{1-t}$ Note: solution does not exist for all time
3. $\frac{dx}{dt} = \sin x, \quad x(0) = 1 \Rightarrow x(t) =$? Examine with slope field



The first order equation is much more general than it appears, because we may write a higher order differential equation as a set of first order equations; in this case, $y = [y_1, y_2, \ldots, y_n]^T$ and $f(y(t), t) = [f_1(y, t), f_2(y, t), \ldots, f_n(y, t)]^T$ become vectors, and the differential equation becomes a system of differential equations.
Note that each component of f may be a nonlinear function of its arguments.

Example 1: Consider the third-order differential equation

$$u'''(t) = u'(t)u(t) - 2t \cdot u''(t)^2$$

with initial data $u(0) = \alpha_1$, $u'(0) = \alpha_2$, and $u''(0) = \alpha_3$. Rewrite the equation as a first-order system.
Let $y_1(t) = u(t)$, $y_2(t) = u'(t)$ and $y_3(t) = u''(t)$. Then

$$y = \begin{pmatrix} u(t) \\ u'(t) \\ u''(t) \end{pmatrix} \quad \text{and} \quad y'(t) = \begin{pmatrix} u'(t) \\ u''(t) \\ u'''(t) \end{pmatrix} = \begin{pmatrix} y_2(t) \\ y_3(t) \\ y_2(t)y_1(t) - 2ty_2^2(t) \end{pmatrix} = f(y(t), t)$$

and the initial data are $y(0) = \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{pmatrix}$.

△

The simplest method for finding numerical approximations of $x(t)$ is known as Euler's method.

choose Δt : time step

define w_n : numerical solution at time $t_n = n\Delta t$, approximate $y'(t)$ using forward difference

$$\frac{w_{n+1} - w_n}{\Delta t} = f(w_n) \;\Rightarrow\; w_{n+1} = w_n + \Delta t f(w_n)$$

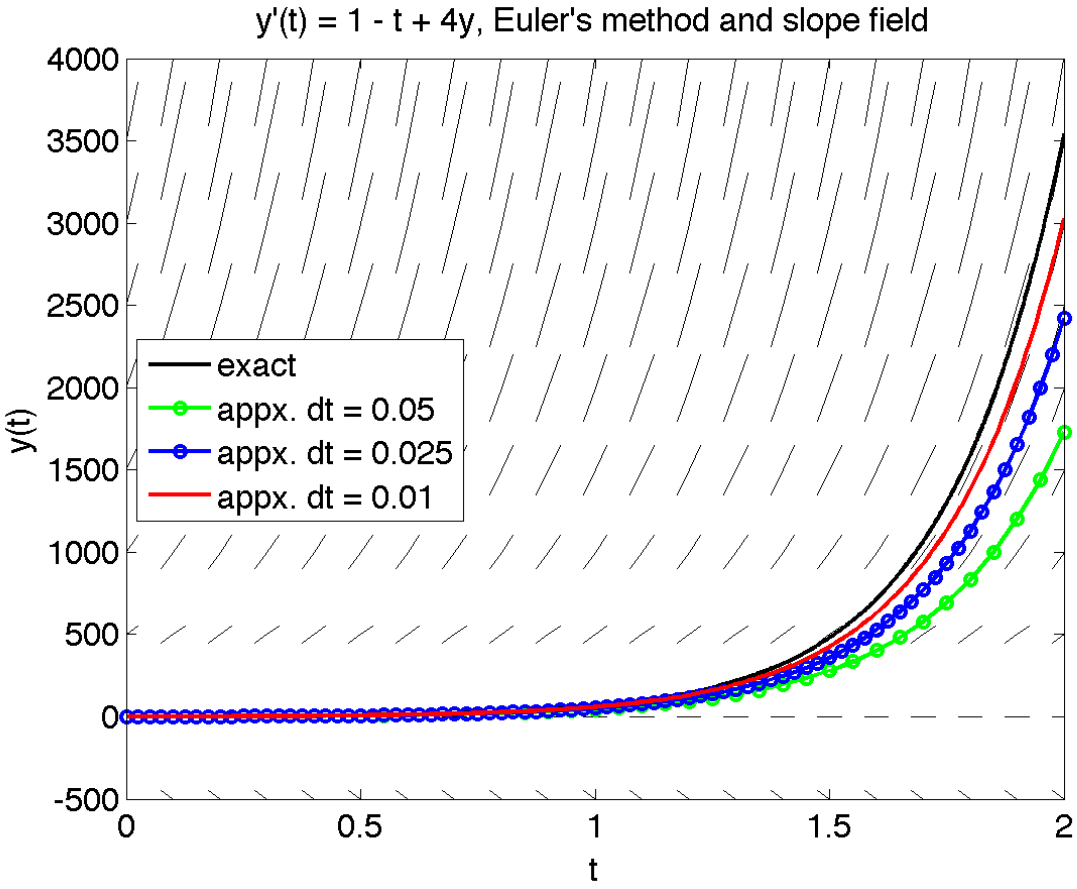
Given w_0 from the initial condition, compute $w_1, w_2, \ldots$

Example 2: $y'(t) = 1 - t + 4y, \quad y(0) = 1$

Algorithm 1: Euler's method

Input : $f(t, y)$, $y(0)$, T, Δt
Output: $w_n \approx y(T)$

```
n = T/Δt
w_0 = y(0)
for j = 1 : n do
    t_{j-1} = (j − 1) · Δt
    w_j = w_{j-1} + Δt · f(t_{j-1}, w_{j-1})
end
return w_j
```



△

Question 1:

1. Is this method accurate?
2. Is this method stable?
3. Is this method efficient?

Definition 1. A numerical method is said to be consistent if its local truncation error at each step (or grid point) goes to zero as the step size goes to zero, i.e., $\lim_{\Delta t \to 0} |\tau_i| = 0$.

Let $x(t)$ represent the exact solution, which we assume has at least 2 continuous derivatives. Then we can expand x in a Taylor series about some point in time t_i as

$$x(t) = x(t_i) + (t - t_i)x'(t_i) + \frac{1}{2}(t - t_i)^2 x''(\zeta_i) \text{ for some } \zeta_i \in [t, t_i]$$

Plugging in $t = t_{i+1}$, and taking $x'(t_i) = f(x(t_i))$ from the ODE, we find that

$$x(t_{i+1}) = x(t_i) + \Delta t f\big(x(t_i)\big) + \frac{1}{2}\Delta t^2 x''(\zeta_i)$$

rewriting the above, we have

$$\frac{x(t_{i+1}) - x(t_i)}{\Delta t} = f\big(x(t_i)\big) + \frac{1}{2}\Delta t x''(\zeta_i),$$

and it is clear that the truncation error τ_i at step i of Euler's method satisfies

$$|\tau_i| \leq \frac{\Delta t}{2} \max_{0 \leq t \leq t_f} \left|x''(t)\right|.$$

Thus, Euler's method for the solution of a first-order ODE has first-order truncation error and is consistent.

1.1.1 Distinguishing local vs. global error

- Special case : Numerical integration. Global error = sum of local error. We can think of numerical integrals as a special case of an ODE that does not depend on y.
- ODEs : Global error depends on the growth rate of local error. At each step, the computed solution depends on the current solution : error accumulates at each step.

Definition 2. A function $f(y,t)$ satisfies a Lipschitz condition on a domain

$$D = \{(y,t) : |y - \eta| \leq a,\ t_0 \leq t \leq t_1\}$$

if there exists some constant $L \geq 0$ such that

$$|f(y,t) - f(\hat{y},t)| \leq L\,|y - \hat{y}|$$

for all (y,t) and $(\hat{y},t)$ in D.

Note: The Lipschitz constant L may depend on the choice of the domain D. On some choices of D the function f may be Lipschitz, on another D, the same function may not; and the constant L may also be different for different choices of D.

Example 3: Lipschitz conditions.

1. $f(y,t) = y \sin t$ is Lipschitz on all of $\mathbb{R}^2$.
   $f(y,t) - f(\hat{y},t) = y \sin t - \hat{y} \sin t = (y - \hat{y}) \sin t$
   $\Rightarrow |f(y,t) - f(\hat{y},t)| \leq |y - \hat{y}|, \quad L = 1$

2. $f(y,t) = \dfrac{t^2 y^2}{1 + t^2}$ is Lipschitz on any finite subset of $\mathbb{R}^2$.
   Let $M > 0$ and set $D = \{(t,y) : t \in \mathbb{R},\ -M \leq y \leq M\}$.
   $f(y,t) - f(\hat{y},t) = \dfrac{t^2 y^2}{1 + t^2} - \dfrac{t^2 \hat{y}^2}{1 + t^2} = \dfrac{t^2}{1 + t^2}(y^2 - \hat{y}^2) = \dfrac{t^2}{1 + t^2}(y + \hat{y})(y - \hat{y})$
   $\Rightarrow |f(y,t) - f(\hat{y},t)| \leq |y + \hat{y}| \cdot |y - \hat{y}| \leq (|y| + |\hat{y}|)\,|y - \hat{y}| \leq 2M\,|y - \hat{y}|, \quad L = 2M$
   Note that this does not satisfy the Lipschitz condition in the limit $M \to \infty$.

△

Sometimes it's easier to use the following theorem (rather than the definition) to show that a function is Lipschitz in a domain.

Theorem 3. *Suppose a function f is defined on a (possibly infinite) domain D and that there exists a constant L such that everywhere in D*

$$\left|\frac{\partial f}{\partial y}(y,t)\right| \leq L,$$

then the function f is Lipschitz on D with Lipschitz constant L.

Proof. Mean value theorem (see section 7.1).

Notes:

- In differential equation theory, continuity and Lipschitz condition are used in the existence/uniqueness theorem
- In numerical methods, Lipschitz constants give indications to the stability of a problem, whether solution curves that start close together will stay close together.

Euler's method cont'd:

For the IVP $y'(t) = f(y(t))$, $0 \leq t \leq T$, discretize time by $t_i = i\Delta t$, $\Delta t = T/n$, $i = 0 : n$.

Euler's method yields

$w_{n+1} = w_n + \Delta t f(w_n)$: computed solution

$\tau_n = \frac{1}{2}\Delta t y''(t_n) + O(\Delta t^2)$: truncation error

$y(t_{n+1}) = y(t_n) + \Delta t f(y(t_n) + \Delta t \tau_n$: true solution

Error : subtract computed from exact

$y(t_{n+1}) - w_{n+1} = y(t_n) - w_n + \Delta t\left(f(y(t_n) - f(w_n)\right) + \Delta t \tau_n$

$E_{n+1} = E_n + \Delta t\left(f(y(t_n) - f(w_n)\right) + \Delta t \tau_n$

If $f(y)$ is Lipschitz, and we consider $y(t_n)$ and w_n as separate points in the Lipschitz domain of f, then

$$|f(y(t_n)) - f(w_n)| \leq L\,|y(t_n) - w_n| = L\,|E_n|\,.$$

Plugging this bound into the above,

$|E_{n+1}| \leq |E_n| + \Delta t L\,|E_n| + \Delta t\,|\tau_n| = (1 + \Delta t L)\,|E_n| + \Delta t\,|\tau_n|$

Looking at this recursion relation, we find

$|E_1| \leq (1 + \Delta t L)\,|E_0| + \Delta t\,|\tau_0|$

$|E_2| \leq (1 + \Delta t L)\,|E_1| + \Delta t \tau_1 = (1 + \Delta t L)^2\,|E_0| + (1 + \Delta t L)\Delta t\,|\tau_0| + \Delta t\,|\tau_1|$

$|E_3| \leq (1 + \Delta t L)\,|E_2| + \Delta t \tau_2 = (1 + \Delta t L)^3\,|E_0| + (1 + \Delta t L)^2\Delta t\,|\tau_0| + (1 + \Delta t L)\Delta t\,|\tau_1| + \Delta t\,|\tau_2|$

$\vdots$

$$|E_n| \le (1+\Delta tL)^n |E_0| + \sum_{j=1}^{n} (1+\Delta tL)^{n-j} \Delta t \, |\tau_{j-1}|$$

claim : $(1+x)^m \le e^{mx}$ proof (hw)

Set $x = \Delta tL$ and $m = n - j$; then $(1+\Delta tL)^{n-j} \le e^{(n-j)\Delta tL} \le e^{n\Delta tL} = e^{TL}$

$$\Rightarrow \sum_{j=1}^{n} (1+\Delta tL)^{n-j} \Delta t \, |\tau_{j-1}| \le \sum_{j=1}^{n} e^{TL} \Delta t \, \|\tau\|_\infty = T e^{TL} \|\tau\|_\infty$$

note: $|E_0| = 0$ why?

$\Rightarrow |E_n| \le T e^{TL} \|\tau\|_\infty = O(\Delta t)$ as $\Delta t \to 0$ for all n with $n\Delta t \le T$.

Thus the global error in Euler's method $\to 0$ as $\Delta t \to 0$, and the method is convergent.
In fact, since $\|\tau\| \sim O(h)$, we can use the above relation to state that Euler's method is first-order accurate in terms of global error.
Note : The text proceeds further, using the form of τ to define the constant even more precisely, in terms of the second derivative of the solution.

notes:

- This is related to an important concept : The Lax Equivalence Theorem. A numerical method that is consistent and stable is convergent.
- Convergence means that global error converges to zero as $\Delta t \to 0$.
- Euler's method is a one-step method; w_{n+1} depends only on one previous step (in this case w_n). Proof of convergence for higher order one-step methods follows the same outline as above.
- Multistep methods require a more sophisticated definition of stability that is covered in Math 572.

1.2 Higher order accuracy

Since Euler's method can be derived from the Taylor series expansion of the solution to $y'(t) = f(y,t)$, we might use more terms in the Taylor series to create more accurate methods. This leads to numerical methods for IVP known as Taylor Series methods, but they are harder to generalize to systems of first order equations.
Multistep methods use data from multiple time steps, say w_{n-1} and w_{n+1} in the Leapfrog Method, and these methods can be derived from piecewise interpolation using higher order polynomials.

1.2.1 Leapfrog method

The simplest multi-step method is the midpoint method, also known as the leapfrog method. It can be derived in the same way as Euler's method by replacing the D_+ approximation to $y'(t)$ with the centered D_0 approximation.

$$\frac{w_{n+1} - w_{n-1}}{2\Delta t} = f(w_n, t_n) \quad \Rightarrow \quad w_{n+1} = w_{n-1} + 2\Delta t f(w_n, t_n)$$

This is a second-order accurate two-step method: w_{n+1} depends on both w_n and w_{n-1}. Note that this requires additional care at the initial time, where you may only have one initial condition.

Algorithm 2: Leapfrog method

Input : $f(t,y),\quad y(0),\quad T,\quad \Delta t$
Output: $w_n \approx y(T)$

$n = T/\Delta t$
$w_0 = y(0)$
$w_1 = w_0 + \Delta t \cdot f(0, w_0)$ `% use Euler's method to appx. w_1`
for $j = 2 : n$ **do**
  $t_{j-1} = (j-1) \cdot \Delta t$
  $w_j = w_{j-2} + 2 \cdot \Delta t \cdot f(t_{j-1}, w_{j-1})$
end
return w_j

Since stability requires more care to prove for such methods, we will stick to one-step methods and to the important family of Runge-Kutta methods, due to the following fact:

Important result: If $f(y,t)$ satisfies a Lipschitz condition, than any one-step method that is consistent will also be stable, and therefore convergent. The proof follows the same form as our proof of convergence for Euler's method.

1.2.2 2-stage Runge-Kutta

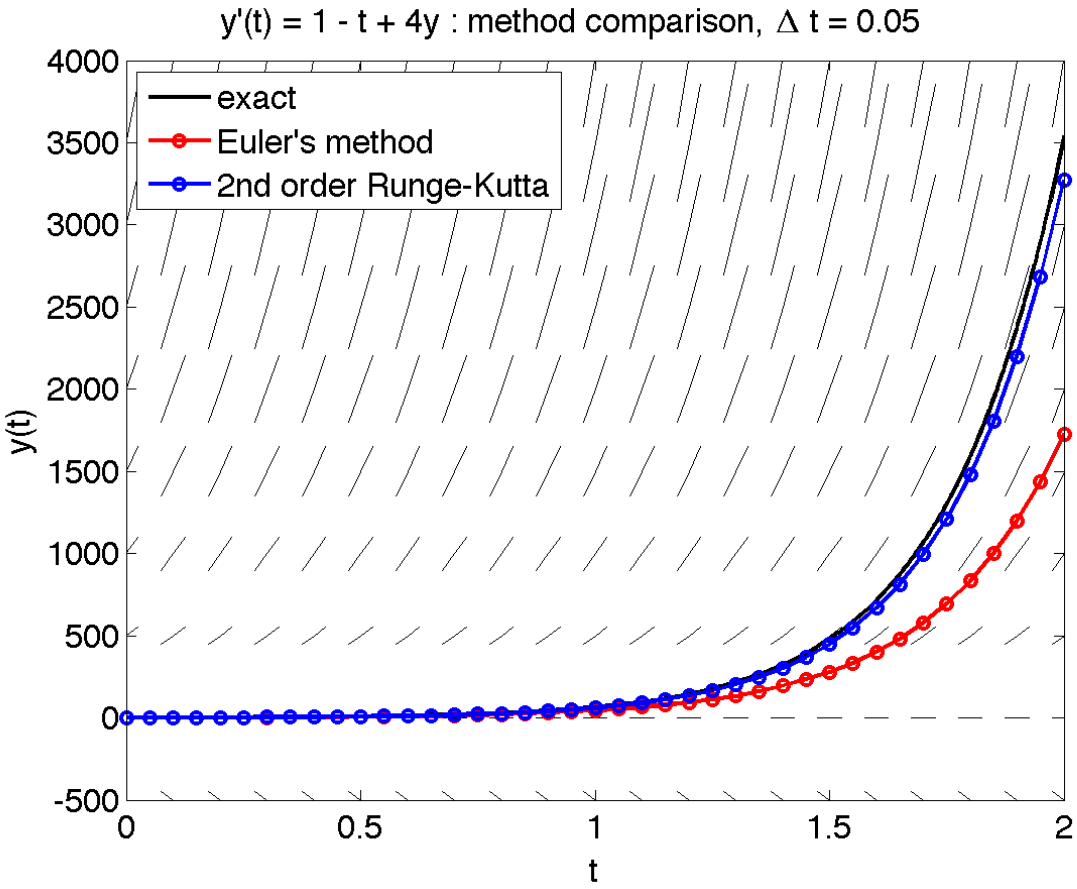
The two-stage Runge-Kutta method is given by

$$w^* = w_n + \frac{1}{2}\Delta t f(w_n, t_n)$$
$$w_{n+1} = w_n + \Delta t f\left(w^*, t_n + \frac{\Delta t}{2}\right)$$

Even though this method has two stages, it is still a one-step method, as can be seen by plugging the intermediate stage w^* into the w_{n+1} equation:

$$w_{n+1} = w_n + \frac{\Delta t}{2} f\left(w_n + \frac{\Delta t}{2} f(w_n, t_n), t_n + \frac{\Delta t}{2}\right)$$

which verifies that w_{n+1} is determined only from w_n.
This is a second-order accurate method.



Algorithm 3: 2-stage Runge-Kutta

Input : $f(t,y)$, $y(0)$, T, Δt
Output: $w_n \approx y(T)$

$n = T/\Delta t$
$w_0 = y(0)$
for $j = 1 : n$ **do**
  $t_{j-1} = (j-1) \cdot \Delta t$
  $w^* = w_{j-1} + \frac{1}{2} \cdot \Delta t \cdot f(t_{j-1}, w_{j-1})$
  $w_j = w_{j-1} + \Delta t \cdot f\left(t_{j-1} + \frac{\Delta t}{2}, w^*\right)$
end
return w_j

1.2.3 4-stage Runge-Kutta

Perhaps the most common Runge-Kutta scheme (type `help ode45` in Matlab) is the fourth order version, given by

$$
\begin{aligned}
k_1 &= \Delta t f(w_n, t_n) \\
k_2 &= \Delta t f\left(w_n + \frac{1}{2}k_1, t_n + \frac{\Delta t}{2}\right) \\
k_3 &= \Delta t f\left(w_n + \frac{1}{2}k_2, t_n + \frac{\Delta t}{2}\right) \\
k_4 &= \Delta t f\left(w_n + k_3, t_n + \Delta t\right) \\
w_{n+1} &= w_n + \frac{1}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right)
\end{aligned}
$$

where the $k_i,\ i = 1, 2, 3, 4$ are the intermediate stages. This method is 4th order accurate, i.e., $|\tau_n| \sim O(\Delta t^4)$ as $\Delta t \to 0$, but requires more memory and 4 function evaluations per step, which can be costly depending on your application.
Again, even though it has four stages, it's still a one-step method.