CMPSC/Math 451, Numerical Computation

Wen Shen

Department of Mathematics, Penn State University

Numerical solution of ordinary differential equations (ODE)

Definition of ODE:
an equation which contains one or more ordinary derivatives of an unknown function.

Example 1. Let $x = x(t)$ be the unknown function of t, ODE examples can be

$$x' = x^2, \qquad x'' + x \cdot x' + 4 = 0, \qquad \text{etc.}$$

We consider the initial-value problem for first-order ODE

$$\begin{cases} x' = f(t, x), & \text{differential equation} \\ x(t_0) = x_0 & \text{given initial condition (IC)} \end{cases}$$

Some examples:

$$x'(t) = 2, \quad x(0) = 0. \qquad \text{solution:} \qquad x(t) = 2t.$$

$$x'(t) = 2t, \quad x(0) = 0. \qquad \text{solution:} \qquad x(t) = t^2.$$

$$x'(t) = x + 1, \quad x(0) = 0. \qquad \text{solution:} \qquad x(t) = e^t - 1.$$

In many situations, exact solutions can be very difficult/impossible to obtain.

Numerical solutions

We seek approximate values of the solution at discrete sampling points.

Uniform grid for time variable. Let h be the time step length

$$t_{k+1} - t_k = h, \qquad t_k = t_0 + kh, \qquad t_0 < t_1 < \cdots < t_N.$$

Given an ODE, and a final computing time t_N.
We seek values $x_n \approx x(t_n)$, $n = 1, 2, \cdots, N$, and $t_0 < t_1 < \cdots < t_N$.

Overview of the Chapter:

- Taylor series method, and error estimates
- Runge-Kutta methods
- Multi-step methods
- System of ODE
- High order equations and systems
- Stiff systems
- Matlab solvers

Taylor series methods for ODEs

Given

$$x'(t) = f(t, x(t)), \qquad x(t_0) = x_0.$$

Let $t_1 = t_0 + h$. Let's find the value $x_1 \approx x(t_1) = x(t_0 + h)$.
Taylor expansion of $x(t_0 + h)$ expand at t_0 gives

$$x(t_0 + h) = x(t_0) + hx'(t_0) + \frac{1}{2}h^2 x''(t_0) + \cdots = \sum_{m=0}^{\infty} \frac{1}{m!} h^m x^{(m)}(t_0)$$

Taylor series method of order m

We take the first $(m+1)$ terms in Taylor expansion.

$$x(t_0+h) \approx x(t_0) + hx'(t_0) + \frac{1}{2}h^2x''(t_0) + \cdots + \frac{1}{m!}h^m x^{(m)}(t_0).$$

Error in each step:

$$x(t_0+h) - x_1 = \sum_{k=m+1}^{\infty} \frac{1}{k!}h^k x^{(k)}(t_0) = \frac{1}{(m+1)!}h^{m+1}x^{(m+1)}(\xi)$$

for some $\xi \in (t_0, t_1)$.

For $m = 1$:

$$x_1 = x_0 + hx'(t_0) = x_0 + h \cdot f(t_0, x_0)$$

This is called **forward Euler step**.

General formula for step number k:

$$x_{k+1} = x_k + h \cdot f(t_k, x_k), \qquad k = 0, 1, 2, \cdots N - 1$$

For $m = 2$:

$$x_1 \quad = \quad x_0 + hx'(t_0) + \frac{1}{2}h^2 x''(t_0)$$

Computing x'':

$$x'' = \frac{d}{dt}x'(t) = \frac{d}{dt}f(t, x(t)) = f_t + f_x \cdot x' = f_t + f_x \cdot f$$

we get

$$x_1 = x_0 + hf(t_0, x_0) + \frac{1}{2}h^2 \left[f_t(t_0, x_0) + f_x(t_0, x_0) \cdot f(t_0, x_0)\right]$$

For general step k, $k = 0, 1, 2, \cdots N - 1$, we have

$$x_{k+1} = x_k + hf(t_k, x_k) + \frac{1}{2}h^2 \left[f_t(t_k, x_k) + f_x(t_k, x_k) \cdot f(t_k, x_k)\right]$$

Examples of Taylor Series Method

Example 1. Set up Taylor series methods with $m = 1$ and $m = 2$ for

$$x' = -x + e^{-t}, \qquad x(0) = 0.$$

The exact solution is $x(t) = te^{-t}$.

Answer. The initial data gives $t_0 = 0, x_0 = 0$.

For $m = 1$, we have

$$x_{k+1} = x_k + h\left(-x_k + e^{-t_k}\right) = (1-h)x_k + he^{-t_k}$$

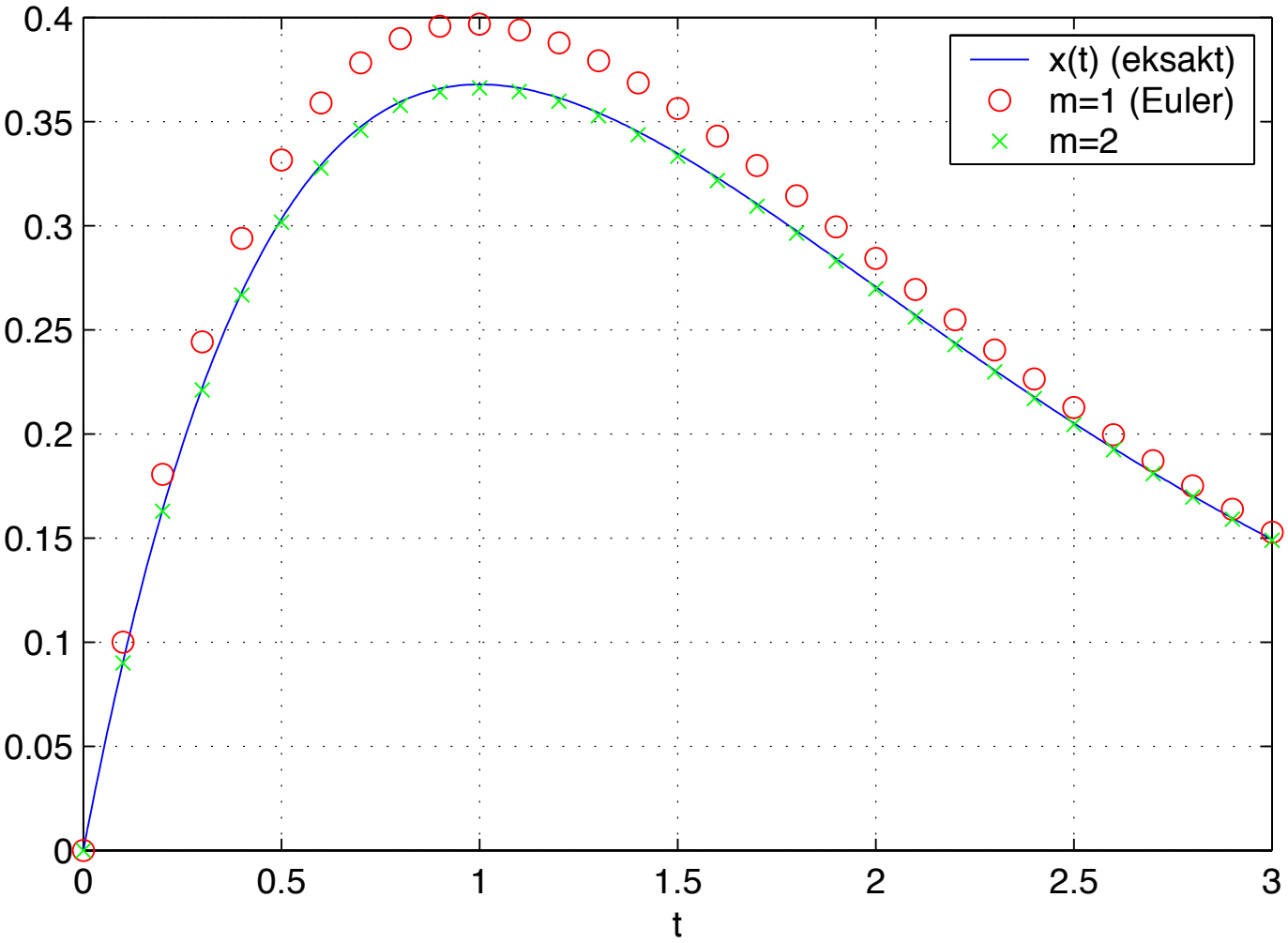
For $m = 2$, we have

$$x'' = \left(-x + e^{-t}\right)' = -x' - e^{-t} = x - e^{-t} - e^{-t} = x - 2e^{-t}$$

so

$$\begin{aligned} x_{k+1} &= x_k + hx_k' + 0.5h^2 x_k'' \\ &= x_k + h\left(-x_k + e^{-t_k}\right) + 0.5h^2\left(x_k - 2e^{-t_k}\right) \\ &= \left(1 - h + 0.5h^2\right)x_k + \left(h - h^2\right)e^{-t_k} \end{aligned}$$

Simulation result



Example 2. Set up Taylor series methods with $m = 1, 2, 3, 4$ for

$$x' = x, \qquad x(0) = 1.$$

The exact solution is $x(t) = e^t$.

Answer. We set $t_0 = 0, x_0 = 1$. Note that

$$x'' = x' = x, \quad x''' = x'' = x, \quad \cdots \quad x^{(m)} = x$$

Taylor series method of order m:

$$x_{k+1} = x_k + hx_k + h^2 x_k/2 + \cdots + h^m x_k/(m!)$$

So

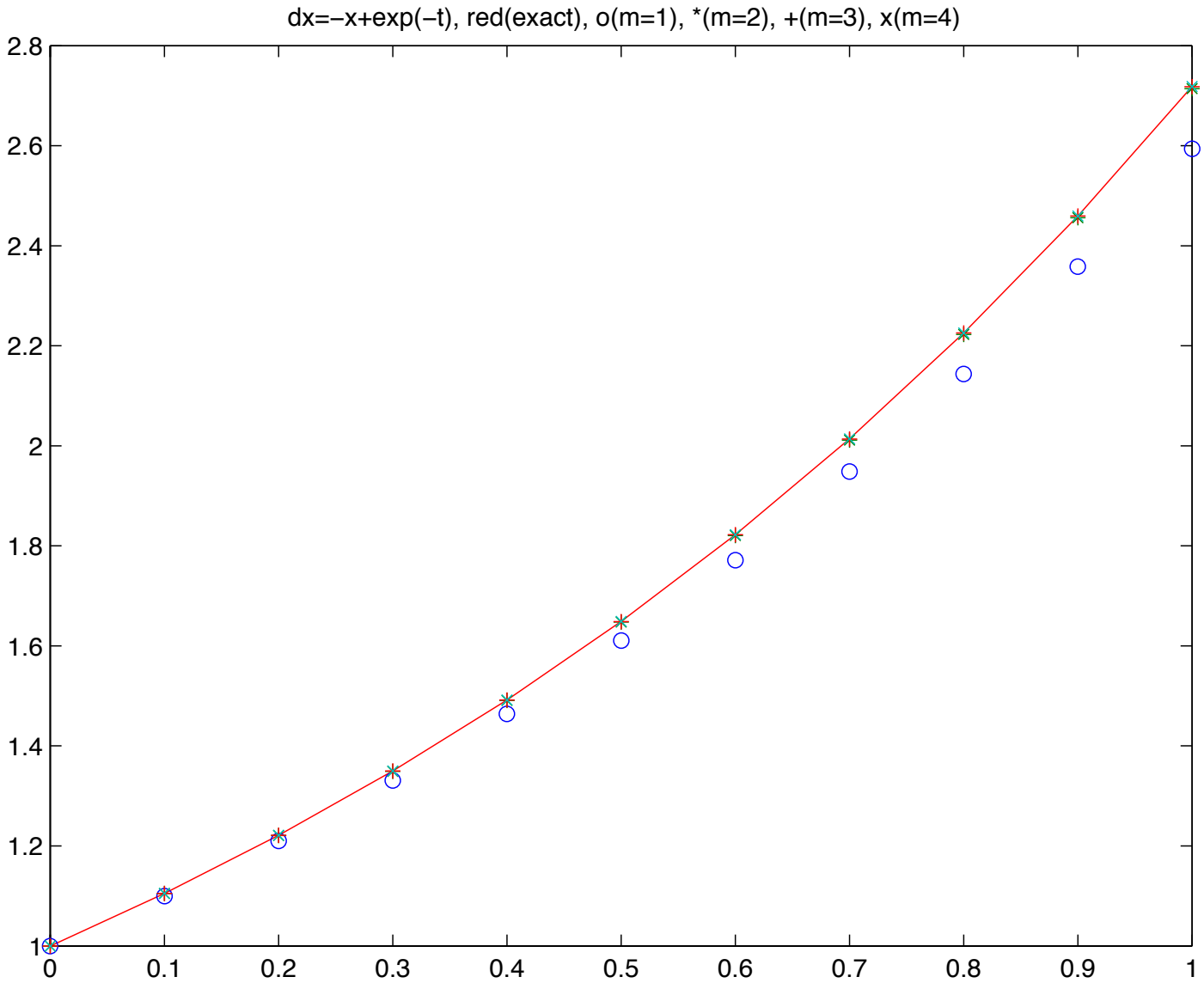
$$m = 1: \qquad x_{k+1} = x_k + hx_k = (1 + h)x_k$$

$$m = 2: \qquad x_{k+1} = x_k + hx_k + h^2 x_k/2 = \left(1 + h + h^2/2\right) x_k$$

$$m = 3: \qquad x_{k+1} = x_k + hx_k + h^2 x_k/2 + h^3 x_k/6 = \left(1 + h + h^2/2 + h^3/6\right) x_k$$

$$m = 4: \qquad x_{k+1} = \cdots = \left(1 + h + h^2/2 + h^3/6 + h^4/24\right) x_k$$

Simulation result



Error analysis for Taylor Series Methods

Given ODE

$$x' = f(t, x), \qquad x(t_0) = x_0. \tag{$*$}$$

Local error (error in each time step) for Taylor series method of order m.
Let t_k, x_k be given,
let x_{k+1} be the numerical solution after one iteration,
and let $x(t_k + h)$ be the exact solution for the IVP

$$x' = f(t, x), \qquad x(t_k) = x_k.$$

Then, the local truncation error is define as

$$e_k \doteq |x_{k+1} - x(t_k + h)|.$$

Theorem *For Taylor series method of order m at step k, the local error is of order $m+1$, i.e., $e_k \leq Mh^{m+1}$ for some bounded constant M.*

Proof. We have

$$e_k = |x_{k+1} - x(t_k + h)| = \frac{h^{m+1}}{(m+1)!} \left| x^{(m+1)}(\xi) \right| = \frac{h^{m+1}}{(m+1)!} \left| \frac{d^m f}{dt^m}(\xi, x(\xi)) \right|,$$

for some $\xi \in (t_k, t_{k+1})$.
We assume

$$\left| \frac{d^m f}{dt^m} \right| \leq M,$$

Now we have

$$e_k \leq \frac{M}{(m+1)!} h^{m+1} = \mathcal{O}(h^{m+1}).$$

Definition. The ODE $x' = f(t, x)$ is called **well-posed** if it is stable w.r.t. perturbations in initial data. This means, let $x(t)$ and $\tilde{x}(t)$ be the solutions with two different initial conditions $x(t_0) = x_0$ and $\tilde{x}(t_0) = \tilde{x}_0$. Fix a final time T. Then, there exists a constant C, independent of t, such that

$$|x(t) - \tilde{x}(t)| \leq C\,|x_0 - \tilde{x}_0|\,.$$

Total error is the error at the final computing time T.
Let

$$N = \frac{T}{h}, \qquad i.e., \quad T = Nh.$$

The total error is defined as

$$E \doteq |x(T) - x_N|.$$

Theorem. *Assume that the ODE is well-posed. If the local error of a numerical iteration satisfies*

$$e_k \leq Mh^{m+1}$$

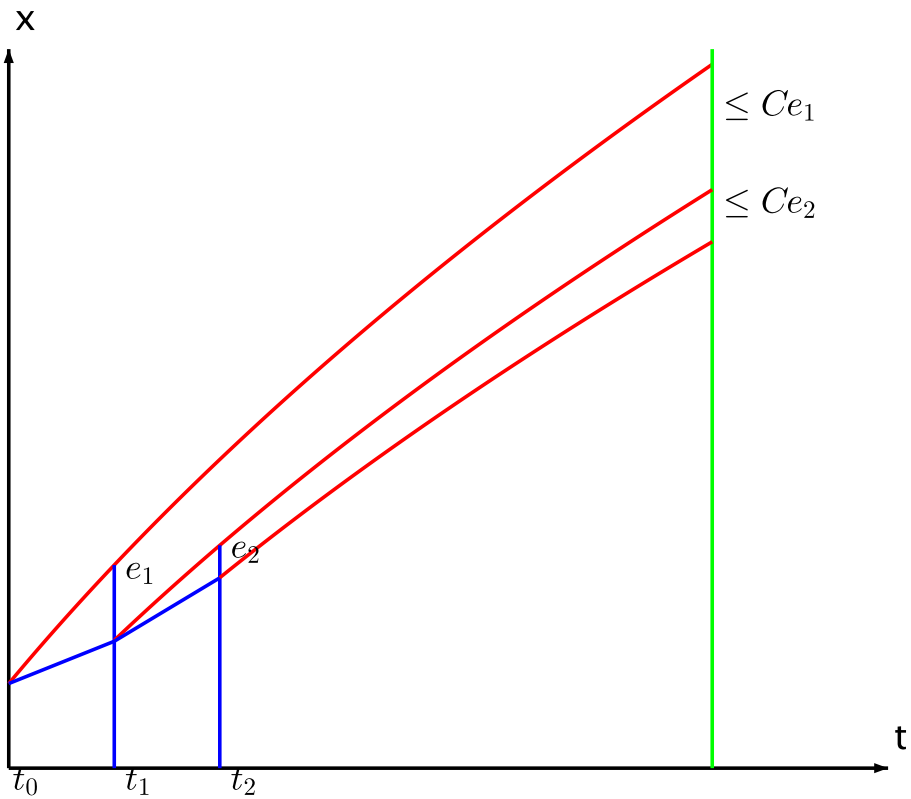
then the total error satisfies

$$E \leq Ch^m$$

for some bounded constant C, where C is uniform in t.

This means that the order of the method is always 1 less than the order of the local truncation error.

Proof. We observe two facts about the errors. First, at every step k, the local error is being carried on through the rest of the simulation. Second, the local errors accumulate through time iteration steps.
By the well-posedness assumption, at each time step k, the local error e_k is amplified at most by a factor of C in the answer at the final time T.



We can add up all the accumulated errors at T caused by all the local errors

$$\begin{aligned} E &= C\sum_{k=1}^{N}\left|e_L^{(k)}\right| \le C\sum_{k=1}^{N}\frac{M}{(m+1)!}h^{m+1} \\ &= CN\frac{M}{(m+1)!}h^{m+1} = C(Nh)\frac{M}{(m+1)!}h^m = \frac{CMT}{(m+1)!}h^m = \mathcal{O}(h^m) \end{aligned}$$

Therefore, the method is of order m.

Runge-Kutta Methods

One difficulty in high order Taylor series methods lies in the fact that it uses the higher order derivatives $x'', x''', \cdots$, which might be very difficult to get.
A better method should only use $f(t, x)$, not its derivatives.
These methods are called Runge-Kutta methods.

1st order method: The same as Euler's method.

2nd order method: Let $h = t_{k+1} - t_k$. Given x_k, the next value x_{k+1} is computed as

$$x_{k+1} = x_k + \frac{1}{2}(K_1 + K_2)$$

where

$$\begin{cases} K_1 &= h \cdot f(t_k, x_k) \\ K_2 &= h \cdot f(t_k + h, x_k + K_1) \end{cases}$$

This is called Heun's method.

Theorem. *Heun's method is of second order.*

Proof. It suffices to show that the local truncation error is of order 3.
Taylor expansion in two variables gives

$$f(t_k + h, x_k + K_1) = f(t_k, x_k) + hf_t(t_k, x_k) + K_1 f_x(t_k, x_k) + \mathcal{O}(h^2, K_1^2).$$

We have $K_1 = hf(t_k, x_k)$, so the last term above is actually $\mathcal{O}(h^2)$. We also have

$$K_2 = h\Big[f(t_k, x_k) + hf_t(t_k, x_k) + hf(t_k, x_k)f_x(t_k, x_k) + \mathcal{O}(h^2)\Big]$$

Then, our method is:

$$\begin{aligned} x_{k+1} &= x_k + \frac{1}{2}\Big[hf + hf + h^2 f_t + h^2 f f_x + \mathcal{O}(h^3)\Big] \\ &= x_k + hf + \frac{1}{2}h^2[f_t + f f_x] + \mathcal{O}(h^3) \end{aligned}$$

$$x_{k+1} = x_k + hf + \frac{1}{2}h^2[f_t + ff_x] + \mathcal{O}(h^3)$$

Compare this with Taylor expansion for $x(t_{k+1}) = x(t_k + h)$

$$\begin{aligned} x(t_k + h) &= x(t_k) + hx'(t_k) + \frac{1}{2}h^2x''(t_k) + \mathcal{O}(h^3) \\ &= x(t_k) + hf(t_k, x_k) + \frac{1}{2}h^2[f_t + f_x x'] + \mathcal{O}(h^3) \\ &= x(t_k) + hf + \frac{1}{2}h^2[f_t + f_x f] + \mathcal{O}(h^3). \end{aligned}$$

We see the first 3 terms are identical, this gives the local truncation error:

$$e_L = |x_{k+1} - x(t_k + h)| = \mathcal{O}(h^3)$$

Viewing Heun's Method as trapezoid method.

Integrating the ODE $x' = f(t, x)$ over the integral $t \in [t_k, t_k + h]$, we get

$$x(t_k + h) = x(t_k) + \int_{t_k}^{t_k+h} x'(t)\, dt = x(t_k) + \int_{t_k}^{t_k+h} f(t, x(t))\, dt. \quad (1)$$

Once $x(t_k) \approx x_k$ is given, then $x_{k+1} \approx x(t_k + h)$ can be computed by suitably approximating the integral.
For the Heun's method, we see that

$$K_1 \approx h x'(t_k), \qquad K_2 \approx h x'(t_k + h)\,.$$

Then, the trapezoid rule

$$\int_{t_k}^{t_k+h} x'(t)\, dt \approx \frac{h}{2}\left[x'(t_k) + x'(t_k + h)\right] = \frac{1}{2}(K_1 + K_2)$$

exactly gives the Heun's iteration.

General Rung-Kutta methods of order m

These methods take the form

$$x_{k+1} = x_k + w_1 K_1 + w_2 K_2 + \cdots + w_m K_m$$

where

$$\begin{cases} K_1 &= h \cdot f(t_k, x_k) \\ K_2 &= h \cdot f(t_k + a_2 h, x + b_2 K_1) \\ K_3 &= h \cdot f(t_k + a_3 h, x + b_3 K_1 + c_3 K_2) \\ &\vdots \\ K_m &= h \cdot f(t_k + a_m h, x + \sum_{i=1}^{m-1} \phi_i K_i) \end{cases}$$

The parameters w_i, a_i, b_i, ϕ_i are carefully chosen to guarantee the order m.
NB! The choice is NOT unique!

The classical RK4

This elegant 4th order method takes the form

$$x_{k+1} = x_k + \frac{1}{6}\Big[K_1 + 2K_2 + 2K_3 + K_4\Big]$$

where

$$\begin{aligned}
K_1 &= h \cdot f(t_k,\ x_k), \\
K_2 &= h \cdot f(t_k + \frac{1}{2}h,\ x_k + \frac{1}{2}K_1), \\
K_3 &= h \cdot f(t_k + \frac{1}{2}h,\ x_k + \frac{1}{2}K_2), \\
K_4 &= h \cdot f(t_k + h,\ x_k + K_3).
\end{aligned}$$

Viewing RK4 Method as Simpsons rule.

The integral form of the ODE $x' = f(t, x)$ gives

$$x(t_k + h) = x(t_k) + \int_{t_k}^{t_k+h} x'(t)\, dt = x(t_k) + \int_{t_k}^{t_k+h} f(t, x(t))\, dt.$$

For the RK4 method, we see that

$$K_1 \approx hx'(t_k), \quad K_2 \approx hx'(t_k + h/2), \quad K_3 \approx hx'(t_k + h/2), \quad K_4 \approx hx'(t_k + h)$$

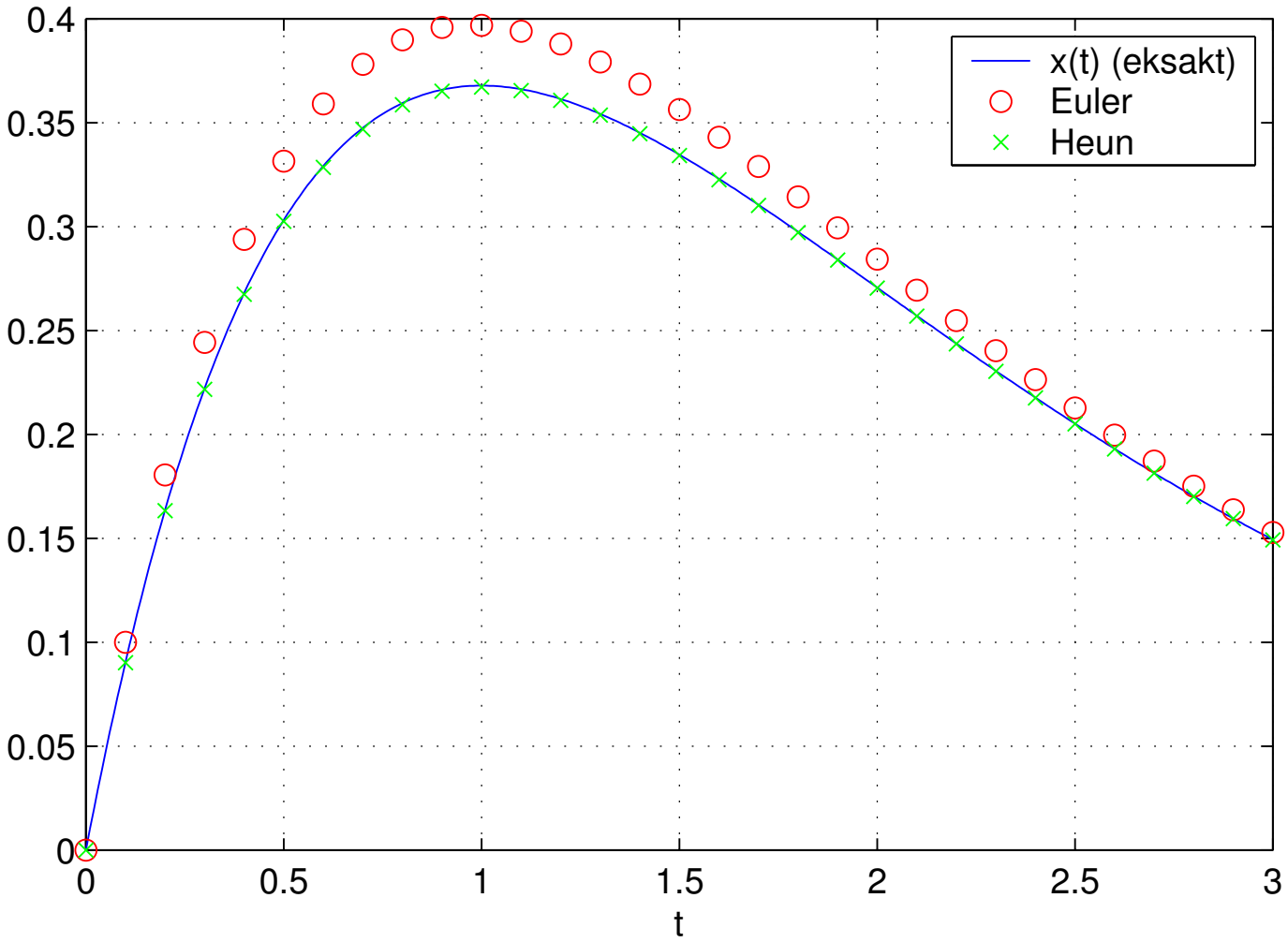
Then, the Simpson's rule

$$\int_{t_k}^{t_k+h} x'(t)\, dt \approx \frac{h}{6}\left[x'(t_k) + 4x'(t_k + h/2) + x'(t_k + h)\right] = \frac{1}{2}(K_1 + 2K_2 + 2K_3 + K_4)$$

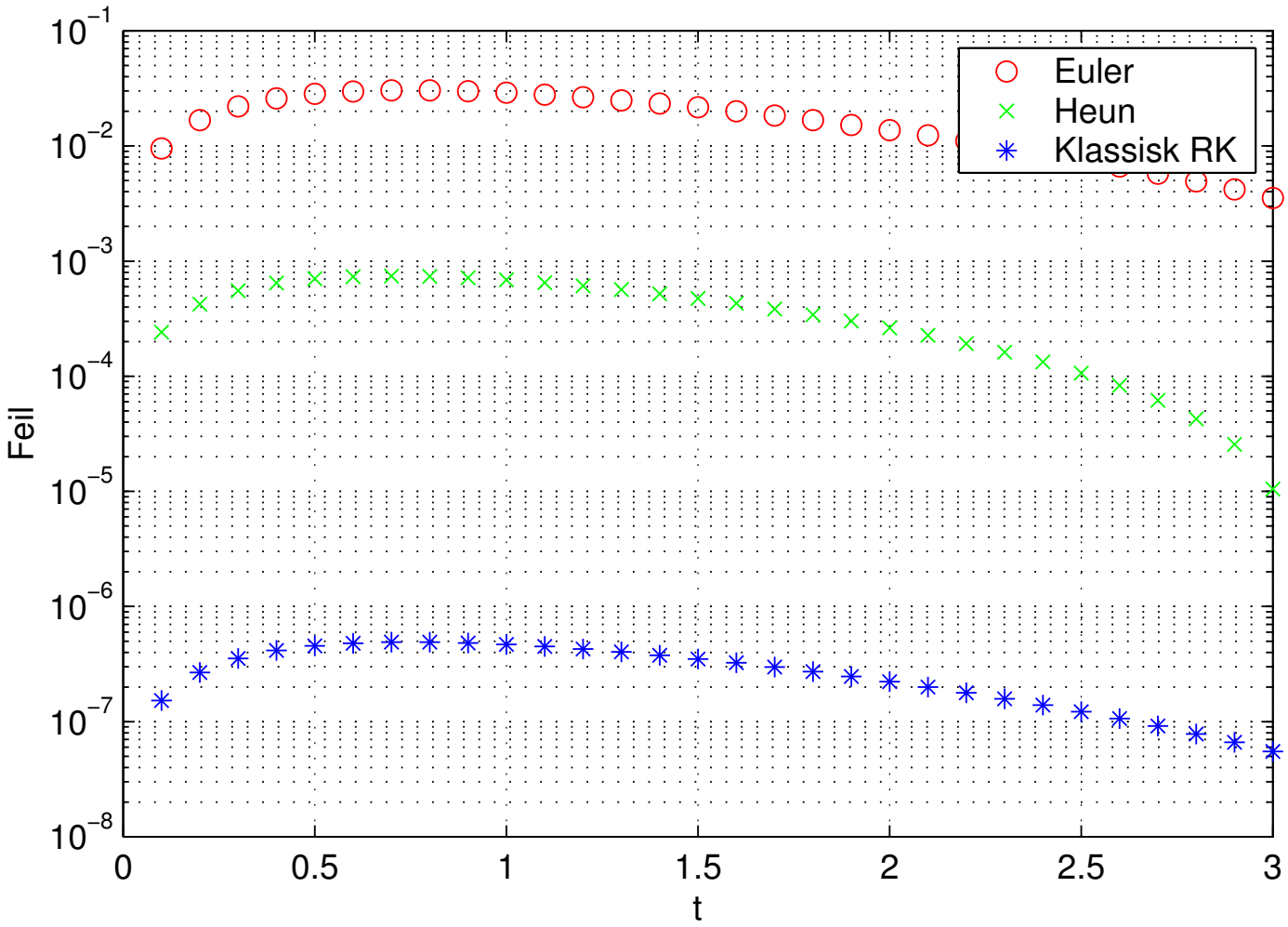
exactly gives the classical RK4 iteration.

Numerical Simulations of Rung-Kutta methods

The ODE: $x' = -x + e^{-t}, \qquad x(0) = 0$



Plot for the errors with various Runge-Kutta methods.



One sees that the higher order methods perform significantly better than the lower order ones.

An adaptive Runge-Kutta-Fehlberg method

In general, we have

1. Smaller time step h gives smaller error;
2. Higher order methods give better approximations;
3. With uniform grid, the local error varies at each step, depending on properties of f and its derivatives.

Optimal situation: h varies each step to get uniform error at each step. This leads to *adaptive methods.*

Key point:
How to get an error estimate at each time step?

Main observation:
If we use two different methods, one is more accurate than the other, then we can assume the better solution is very close to the exact solution.
Thus, the difference between these two solutions becomes a measurement for the local error.

The algorithm reads:

- Compute $x(t+h)$ from $x(t)$ with Method 1, call it $x(t+h)$;
- Compute $x(t+h)$ from $x(t)$ with Method 2, a better method that generates a more accurate approximation. Call this solution $\bar{x}(t+h)$;
- Then, $|x(t+h) - \bar{x}(t+h)|$ gives a measure to error;
- if error $>>$ tol, half the step size;
- if error $<<$ tol, double the step size;
- if error $\approx$ tol, keep the step size;

Now we need to find two methods with different accuracy.
By our observation earlier we may propose the following:

- Method 1: Compute $x(t+h)$ from $x(t)$ with step h, using a specific method, say RK4 method;
- Method 2: Compute $x(t+\frac{1}{2}h)$ from $x(t)$ with step $\frac{1}{2}h$, then compute $x(t+h)$ from $x(t+\frac{1}{2}h)$ with step $\frac{1}{2}h$, with the same methods in Method 1. This approximation is better.

But this is rather wasteful of computing time.

A better approach due to Fehlberg, is built upon some higher order RK methods. He has a 4th order method:

$$x(t+h) = x(t) + \frac{25}{216}K_1 + \frac{1408}{2565}K_3 + \frac{2197}{4104}K_4 - \frac{1}{5}K_5$$

where

$$\begin{aligned}
K_1 &= h \cdot f(t, x), \\
K_2 &= h \cdot f\left(t + \frac{1}{4}h,\ x + \frac{1}{4}K_1\right), \\
K_3 &= h \cdot f\left(t + \frac{3}{8}h,\ x + \frac{3}{32}K_1 + \frac{9}{32}K_2\right), \\
K_4 &= h \cdot f\left(t + \frac{12}{13}h,\ x + \frac{1932}{2197}K_1 - \frac{7200}{2197}K_2 + \frac{7296}{2197}K_3\right), \\
K_5 &= h \cdot f\left(t + h,\ x + \frac{439}{216}K_1 - 8K_2 + \frac{3680}{513}K_3 - \frac{845}{4104}K_4\right).
\end{aligned}$$

Adding an additional term:

$$K_6 = h \cdot f\left(t + \frac{1}{2}h,\ x - \frac{8}{27}K_1 + 2K_2 - \frac{3544}{2565}K_3 + \frac{1859}{4104}K_4 - \frac{11}{40}K_5\right)$$

one obtains a 5th order method:

$$\bar{x}(t+h) = x(t) + \frac{16}{135}K_1 + \frac{6656}{12825}K_3 + \frac{28561}{56430}K_4 - \frac{9}{50}K_5 + \frac{2}{55}K_6.$$

Assuming that the 5th order approximation is almost the exact solution, we can use the difference $|x(t+h) - \bar{x}(t+h)|$ as an estimate for the error.

Pseudo code for adaptive RK45, with time step controller, is provided below.

Given $t_0, t_f, x_0, h_0, n_{max}, e_{min}, e_{max}, h_{min}, h_{max}$

set $h = h_0, t = t_0, x_0 = x_0, k = 0,$

while $(k < n_{max}$ and $t < t_f)$ do

if $h < h_{min}$ then $h = h_{min}$,

else if $h > h_{max}$ then $h = h_{max}$,

end

Compute RKF4, RKF5, and $e = |\text{RKF4} - \text{RKF5}|$

if $(e > e_{max}$ and $h > h_{min})$, then $h = h/2$; (reject the step)

else –(accept the step)

$k = k + 1; t = t + h; x_k = \text{RKF5};$

If $e < e_{min}$, then $h = 2 * h$; end

end

end (while)

Explicit Adam-Bashforth methods

Given

$$x' = f(t, x), \quad x(t_0) = x_0,$$

Let $t_n = t_0 + nh$. If $x(t_n)$ is given, the exact value for $x(t_{n+1})$ would be

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

Idea: Find a good approximation to the integral

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

How? One possibility: approximate f by polynomials, i.e., polynomial interpolations.

Given

$$t_n, t_{n-1}, \cdots, t_{n-k}, \qquad x_n, x_{n-1}, \cdots, x_{n-k},$$

we can compute

$$f_n = f(t_n, x_n), \quad f_{n-1} = f(t_{n-1}, x_{n-1}), \quad \cdots, \quad f_{n-k} = f(t_{n-k}, x_{n-k}),$$

to obtain the data set

$$(t_n, f_n), \quad (t_{n-1}, f_{n-1}), \quad \cdots, \quad (t_{n-k}, f_{n-k}).$$

We find interpolating polynomial $P_k(t)$ that interpolates the above data set $(t_i, f_i)_{i=n-k}^{n}$.

The Lagrange form for $P_k(t)$ with the cardinal functions $l_i(t)$

$$P_k(t) = f_n l_n(t) + f_{n-1} l_{n-1}(t) + \cdots + f_{n-k} l_{n-k}(t).$$

We then use $P_k(t)$ as an approximation to $f(t, x(t))$, and obtain the time iteration step:

$$x_{n+1} = x_n + \int_{t_n}^{t_{n+1}} P_k(s)\, ds,$$

which gives

$$\begin{aligned}
x_{n+1} &= x_n + \int_{t_n}^{t_n+h} \left(f_n l_n(t) + f_{n-1} l_{n-1}(t) + \cdots + f_{n-k} l_{n-k}(t)\right)\, dt \\
&= x_n + \int_{t_n}^{t_n+h} f_n l_n(t)\, dt + \cdots + \int_{t_n}^{t_n+h} f_{n-k} l_{n-k}(t)\, dt \\
&= x_n + f_n \int_{t_n}^{t_n+h} l_n(t)\, dt + \cdots + f_{n-k} \int_{t_n}^{t_n+h} l_{n-k}(t)\, dt \\
&= x_n + h \cdot (b_0 f_n + b_1 f_{n-1} + b_2 f_{n-2} + \cdots + b_k f_{n-k}),
\end{aligned}$$

where $b_0, b_1, \cdots, b_k$ are constants,

$$b_0 = \frac{1}{h} \int_{t_n}^{t_n+h} l_n(t)\, dt, \quad b_1 = \frac{1}{h} \int_{t_n}^{t_n+h} l_{n-1}(t)\, dt, \quad \cdots \quad b_k = \frac{1}{h} \int_{t_n}^{t_n+h} l_{n-k}(t)\, dt.$$

Examples of some explicit Adam-Bashforth (AB) methods

Example 1. If $k = 0$, then we use only one point, i.e., (t_n, f_n), and $P_0(t) = f_n$.

This gives

$$x_{n+1} = x_n + h \cdot f(t_n, x_n)$$

We recognize this as the forward explicit Euler's method.

Example 2. Consider $k = 1$, and we will use two points.
Given x_n, x_{n-1}, we can compute f_n, f_{n-1} as

$$f_n = f(t_n, x_n), \qquad f_{n-1} = f(t_{n-1}, x_{n-1})$$

Use now linear interpolation, we get

$$x'(s) \approx P_1(s) = f_{n-1} + \frac{f_n - f_{n-1}}{h}(s - t_{n-1}).$$

Then

$$x_{n+1} = x_n + \int_{t_n}^{t_{n+1}} P_1(s)\, ds = x_n + \frac{h}{2}(3f_n - f_{n-1}).$$

This is the famous 2nd order Adam-Bashforth method.
One needs two initial values to start the iteration, where $x_0 = x(t_0)$ is given, and x_1 could be computed by for either Euler, Heun's or some RK4 method.

Example 3. For $k = 2$, we get a third order method:

$$x_{n+1} = x_n + h\left(\frac{23}{12}f_n - \frac{4}{3}f_{n-1} + \frac{5}{12}f_{n-2}\right).$$

For $k = 3$, we have a 4th order method:

$$x_{n+1} = x_n + h\left(\frac{55}{24}f_n - \frac{59}{24}f_{n-1} + \frac{37}{24}f_{n-2} - \frac{3}{8}f_{n-3}\right).$$

Critics on the method:

- Good sides: Simple, minimum number of $f(\cdot)$ evaluations. Fast.
- Disadvantage: Here we use interpolating polynomial to approximate a function outside the interval of interpolating points. This is called extrapolation, and it gives a bigger error.

Implicit Adam-Bashforth-Moulton (ABM) methods

The main idea here is to avoid using extrapolation, to reduce the interpolation error.

We will now find an interpolating polynomial $P_{k+1}(t)$ that interpolates

$$(f_{n+1}, t_{n+1}), \quad (f_n, t_n), \quad \cdots, \quad (f_{n-k}, t_{n-k}),$$

and use it to approximate $f(t, x(t))$, to generate an iteration step

$$x_{n+1} = x_n + \int_{t_n}^{t_{n+1}} P_{k+1}(s)\, ds.$$

From previous discussion, using the Lagrange form for P_{k+1}, we end up with the general form for each time step:

$$x_{n+1} = x_n + h \cdot (b_{-1} f_{n+1} + b_0 f_n + b_1 f_{n-1} + b_2 f_{n-2} + \cdots + b_k f_{n-k})$$

for some suitable constants $b_{-1}, b_0, \cdots, b_k$.

Important observation: The function $f_{n+1} = f(t_{n+1}, x_{n+1})$ is unknown! Therefore, we get a non-linear equation to solve for x_{n+1}. We called this kind of method an **implicit method**.

One can solve the nonlinear equation by Newton or secant method. An excellent choice for initial guess would be the solution with 2nd order AB (explicit) method.Then, Newton iteration will converge in 1-2 iterations.

Examples of ABM methods

Consider $k = -1, 0, 1, 2, 3$ (which correspond to using 1, 2, 3, 4, 5 points respectively). Straight computation gives

$$k = -1: \quad x_{n+1} = x_n + h \cdot f_{n+1}, \qquad \text{(implicit backward Euler's method)}$$

$$k = 0: \quad x_{n+1} = x_n + \frac{h}{2}\left(f_n + f_{n+1}\right), \qquad \text{(trapezoid rule)}$$

$$k = 1: \quad x_{n+1} = x_n + h \cdot \left(\frac{5}{12} f_{n+1} + \frac{2}{3} f_n - \frac{1}{12} f_{n-1}\right)$$

$$k = 2: \quad x_{n+1} = x_n + h \cdot \left(\frac{3}{8} f_{n+1} + \frac{19}{24} f_n - \frac{5}{24} f_{n-1} + \frac{1}{24} f_{n-2}\right)$$

$$k = 3: \quad x_{n+1} = x_n + h \cdot \left(\frac{251}{720} f_{n+1} + \frac{646}{720} f_n - \frac{264}{720} f_{n-1} + \frac{106}{720} f_{n-2} - \frac{19}{720} f_{n-3}\right)$$

Multi-step ABM method.

Next is the famous 2nd order ABM (Adam-Bashforth-Moulton) method, which combines the explicit and implicit methods in a smart way.

1. Given $x_n, x_{n-1}, f_n, f_{n-1}$, compute explicit AB solution with $k = 1$:

$$(P) \qquad \begin{cases} x^*_{n+1} &= x_n + h\left(\frac{3}{2}f_n - \frac{1}{2}f_{n-1}\right), \\ f^*_{n+1} &= f\left(t_{n+1}, x^*_{n+1}\right). \end{cases}$$

2. Take one step of the implicit AB solution with $k = 0$, using the value in step 1 as the value for t_{n+1}:

$$(C) \qquad \begin{cases} x_{n+1} &= x_n + \frac{h}{2}\left(f^*_{n+1} + f_n\right), \\ f_{n+1} &= f(t_{n+1}, x_{n+1}). \end{cases}$$

Here step (P) is called the *predictor*, and step (C) is the *corrector*.
This is called a *predictor-corrector's method.*

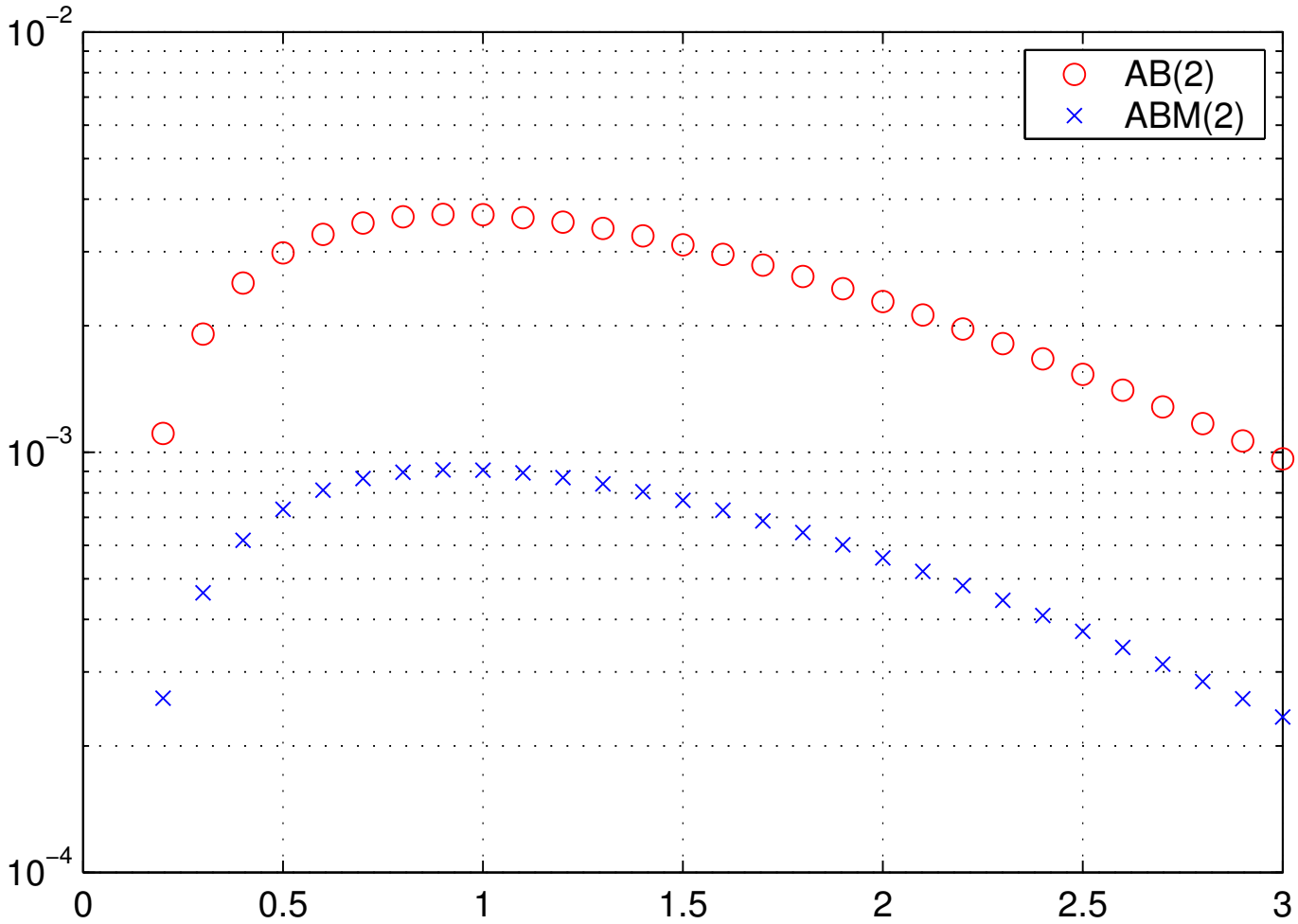
Numerical Simulations.

We consider again the ODE

$$x' = -x + \mathrm{e}^{-t}, \qquad x(0) = 0.$$

In order to convince ourselves that the corrector step really reduces the error, we solve the equation by both the explicit Adam-Bashforth method and the multi-step Adam-Bashforth-Moulton method.
Both methods are formally second order.

The errors are plotted in the Figure below.



The merit of the corrector step is apparent in the result.

Methods for first order systems of ODE

We consider

$$\vec{x}' = F(t, \vec{x}), \qquad \vec{x}(t_0) = \vec{x}_0$$

Here $\vec{x} = (x_1, x_2, \cdots, x_n)^t$ is a vector, and $F = (f_1, f_2, \cdots, f_n)^t$ is a vector-valued function.

Write it out

$$\begin{cases} x_1' &= f_1(t, x_1, x_2, \cdots, x_n) \\ x_2' &= f_2(t, x_1, x_2, \cdots, x_n) \\ & \cdots \\ x_n' &= f_n(t, x_1, x_2, \cdots, x_n) \end{cases}$$

Good news: All methods for scalar equation can be adapted for systems!

Taylor series methods:

$$\vec{x}(t+h) = \vec{x} + h\vec{x}' + \frac{1}{2}h^2\vec{x}'' + \cdots + \frac{1}{m!}h^m\vec{x}^{(m)}$$

Example

Consider

$$\begin{cases} x_1' &= x_1 - x_2 + 2t - t^2 - t^3 \\ x_2' &= x_1 + x_2 - 4t^2 + t^3 \end{cases}$$

We will need the high order derivatives:

$$\begin{cases} x_1'' &= x_1' - x_2' + 2 - 2t - 3t^2 \\ x_2'' &= x_1' + x_2' - 8t + 3t^2 \end{cases}$$

and

$$\begin{cases} x_1''' &= x_1'' - x_2'' - 2 - 6t \\ x_2''' &= x_1'' + x_2'' - 8 + 6t \end{cases}$$

and so on...

Runge-Kutta methods also take the same form for systems. For example, the classical RK4 becomes:

$$\vec{x}_{k+1} = \vec{x}_k + \frac{1}{6}\left[\vec{K}_1 + 2\vec{K}_2 + 2\vec{K}_3 + \vec{K}_4\right]$$

where

$$\begin{aligned}
\vec{K}_1 &= h \cdot F(t_k,\ \vec{x}_k) \\
\vec{K}_2 &= h \cdot F(t_k + \frac{1}{2}h,\ \vec{x}_k + \frac{1}{2}\vec{K}_1) \\
\vec{K}_3 &= h \cdot F(t_k + \frac{1}{2}h,\ \vec{x}_k + \frac{1}{2}\vec{K}_2) \\
\vec{K}_4 &= h \cdot F(t_k + h,\ \vec{x}_k + \vec{K}_3)
\end{aligned}$$

Here everything is a vector instead of a scalar value.

Matlab Codes: The classical RK4 method .

```
function [t,x] = rk4(f,t0,x0,tend,N)
% f  : Differential equation xp = f(t,x)
% x0 : initial condition
% t0,tend : initial and final time
% N : number of time steps

h = (tend-t0)/N;
t = [t0:h:tend];
s = length(x0);  % x0 can be a vector
x = zeros(s,N+1);
x(:,1) = x0;

for n = 1:N
  k1 = feval(f,t(n),x(:,n));
  k2 = feval(f,t(n)+0.5*h,x(:,n)+0.5*h*k1);
  k3 = feval(f,t(n)+0.5*h,x(:,n)+0.5*h*k2);
  k4 = feval(f,t(n)+h,x(:,n)+h*k3);
  x(:,n+1) = x(:,n) + h/6*(k1+2*(k2+k3)+k4);
end
```

Matlab code: The multi-step second order Adams-Bashforth-Moulton method.

```
function [t,x] = abm2(f,t0,x0,x1,tend,N)
% f  : Differential equation xp = f(t,x)
% x0,x1 : Starting data
% t0,tend : initial time and final time
% N : number of time steps

h = (tend-t0)/N;  t = [t0:h:tend];  s = length(x0);
x = zeros(s,N+1);
x(:,1) = x0;  x(:,2) = x1;  % starting data for time iterations
fnm1 = feval(f,t(1),x0);  fn = feval(f,t(2),x1);

for n = 2:N
  xs = x(:,n) + 0.5*h*(3*fn-fnm1);  % predictor
  fnp1 = feval(f,t(n+1),xs);  % predictor

  x(:,n+1) = x(:,n)+0.5*h*(fnp1+fn); % corrector
  fnm1 = fn;
  fn = feval(f,t(n),x(:,n)); % corrector
end
```

Higher order equations and systems

Consider the higher order ODE

$$u^{(n)} = f(t, u, u', u'', \cdots, u^{(n-1)}), \qquad \text{ICs:} \quad u(t_0), u'(t_0), u''(t_0), \cdots, u^{(n-1)}(t_0).$$

Introduce a systematic change of variables

$$x_1 = u, \quad x_2 = u', \quad x_3 = u'', \quad \cdots \quad x_n = u^{(n-1)}.$$

We then have

$$\begin{cases} x_1' &= u' = x_2 \\ x_2' &= u'' = x_3 \\ x_3' &= u''' = x_4 \\ & \vdots \\ x_{n-1}' &= u^{(n-1)} = x_n \\ x_n' &= u^{(n)} = f(t, x_1, x_2, \cdots, x_n) \end{cases}$$

This is a system of 1st order ODEs, with initial data given at

$$x_1(t_0) = u(t_0), \quad x_2(t_0) = u'(t)0), \quad \cdots \quad , x_n(t)0) = u^{(n-1)}(t_0).$$

Example

Consider the ODE

$$u'' + 4uu' + t^2u + t = 0, \qquad u(0) = 1, \quad u'(0) = 2.$$

By the variable change

$$x_1 = u(t), \quad x_2 = u'(t)$$

we have

$$\begin{cases} x_1' &= x_2 \\ x_2' &= -4x_1x_2 - t^2x_1 - t \end{cases}$$

with initial conditions

$$\begin{cases} x_1(0) &= u(0) = 1 \\ x_2(0) &= u'(0) = 2 \end{cases}$$

Systems of high-order equations are treated in the same way. We can write each higher order ODE into a system of first order ODEs, and we go through all the equations.

Example

Consider the system of 2nd order ODEs

$$\begin{aligned} u'' &= u(1-v') + t, \\ v'' &= v^2 - u'v' + tu^2, \end{aligned}$$

with the initial conditions

$$u(1) = 1, \quad u'(1) = 2, \quad v(1) = 3, \quad v'(1) = 4.$$

Introduce the variable change

$$x_1 = u, \quad x_2 = u', \quad x_3 = v, \quad x_4 = v',$$

we have the 4×4 system of first order equations

$$\begin{cases} x_1' &= x_2 \\ x_2' &= x_1(1-x_4) + t \\ x_3' &= x_4 \\ x_4' &= x_3^2 - x_2 x_4 + t x_1^2 \end{cases} \qquad \text{IC.} \qquad \begin{cases} x_1(1) &= u(1) = 1 \\ x_2(1) &= u'(1) = 2 \\ x_3(1) &= v(1) = 3 \\ x_4(1) &= v'(1) = 4 \end{cases}$$

Stiff systems

A scalar equation. We first consider a simple scalar equation

$$x' = -ax, \qquad x(0) = 1$$

where $a > 0$ is a constant, possibly very large.The exact solution is

$$x(t) = e^{-at}.$$

This is an exponential decay. We see that

$$x \to 0 \quad \text{as} \quad t \to +\infty. \tag{1}$$

Furthermore, the larger the value a, the faster the decay.

We now solve it by forward Euler's method:

$$x_0 = 1, \qquad x_{n+1} = x_n - ahx_n = (1 - ah)x_n, \quad n \geq 1.$$

Simple induction argument shows that

$$x_n = (1 - ah)^n x_0 = (1 - ah)^n.$$

We expect that the numerical solution should preserve the important property (1), i.e.,

$$x_n \to 0, \quad \text{as} \quad n \to +\infty.$$

We must require

$$|1 - ah| < 1, \quad \Rightarrow \quad h < \frac{2}{a}.$$

This gives a restriction to the time step size h, i.e., h must be sufficiently small. The larger the value of a, the smaller h must be, even though the solution is almost 0 after a very short time!

To improve the stability, we now use the implicit Euler step:

$$x_0 = 1, \qquad x_{n+1} = x_n - ahx_{n+1}, \quad n \geq 1.$$

This implies

$$x_{n+1} = \frac{1}{1+ah} x_n.$$

Simple induction argument shows that for all $n \geq 0$, we have

$$x_n = \left(\frac{1}{1+ah}\right)^n.$$

Since $ah > 0$, we have

$$0 < \frac{1}{1+ah} < 1$$

leading to

$$\lim_{n \to +\infty} x_n = 0$$

for any values of h. This is called **unconditionally stable**.

Remark. Stability condition occurs also for nonlinear equations. But if one applies an implicit method, it becomes unconditionally stable, but at a price.
Consider the general equation

$$x' = f(t, x), \quad x(t_0) = x_0.$$

where $f(t, x)$ is nonlinear in x. The implicit Euler step becomes

$$x_{n+1} = x_n + h \cdot f(t_{n+1}, x_{n+1}).$$

We see that this becomes a non-linear equation for x_{n+1}, which may or may not have solutions, or multiple solutions. An approximate solution could be obtained using a possible Newton iteration or some varieties of it. It can be very time consuming.

System of ODEs.

The problem is more annoying for systems. We now consider a system

$$\begin{cases} x' &= -20x - 19y \\ y' &= -19x - 20y \end{cases} \qquad \begin{cases} x(0) &= 2 \\ y(0) &= 0 \end{cases}$$

We can re-write the system in vector and matrix form as

$$\vec{x}'(t) = A\vec{x}, \qquad A = \begin{pmatrix} -20 & -19 \\ -19 & -20 \end{pmatrix}, \qquad \vec{x} = \begin{pmatrix} x \\ y \end{pmatrix}$$

For the coefficient matrix A, we have the following eigenvalues and condition number

$$\lambda_1(A) = -1, \qquad \lambda_2(A) = -39, \qquad \text{cond}(A) = 39.$$

Note that the condition number is rather large.

The exact solution is

$$\begin{cases} x(t) &= e^{-39t} + e^{-t} \\ y(t) &= e^{-39t} - e^{-t} \end{cases}$$

One can easily verify it by plugging it in the ODEs and the ICs.
The exact solution has the following decay property

$$x \to 0, \quad y \to 0 \qquad \text{as} \qquad t \to +\infty. \tag{1}$$

We have the following further observations:

- Two components in the solution, e^{-39t} and e^{-t};
- The two eigenvalues give exact the two decay rates in the solution.
- The condition number of A is rather large, indicating two very different rates of decay in the solution. The term e^{-39t} tends to 0 much faster than the term e^{-t};
- For large values of t, the term e^{-t} dominate.
- Therefore, e^{-39t} is called the *transient term*.

We solve the system with forward Euler's method:

$$\begin{cases} x_{n+1} &= x_n + h \cdot (-20x_n - 19y_n), \\ y_{n+1} &= y_n + h \cdot (-19x_n - 20y_n), \end{cases} \qquad \begin{cases} x_0 &= 2, \\ y_0 &= 0. \end{cases}$$

One can show by induction that

$$\begin{cases} x_n &= (1-39h)^n + (1-h)^n, \\ y_n &= (1-39h)^n - (1-h)^n. \end{cases}$$

We must require that the numerical approximation preserves the property (1), i.e.,

$$x_n \to 0, \qquad y_n \to 0 \qquad \text{as} \quad n \to +\infty.$$

This gives the conditions

$$|1 - 39h| < 1 \qquad \text{and} \qquad |1 - h| < 1$$

which implies

$$(1): \ h < \frac{2}{39} \qquad \text{and} \qquad (2): \ h < 2$$

We see that condition (1) is much stronger than condition (2), therefore it must be satisfied.
Condition (1) corresponds to the term e^{-39t}, which is the transient term and it tends to 0 very quickly as t grows. Unfortunately, time step size is restricted by this transient term.

Why is this system stiff? Because the condition number is very large, so the system has two components with very different varying rates.

Stiff System: Implicit method

We now propose a more stable method, the implicit Backward Euler method:

$$\begin{cases} x_{n+1} &= x_n + h\cdot(-20x_{n+1} - 19y_{n+1}), \\ y_{n+1} &= y_n + h\cdot(-19x_{n+1} - 20y_{n+1}), \end{cases} \qquad \begin{cases} x_0 &= 2, \\ y_0 &= 0. \end{cases}$$

Let

$$A = \begin{pmatrix} -20 & -19 \\ -19 & -20 \end{pmatrix}, \qquad \vec{x} = \begin{pmatrix} x \\ y \end{pmatrix}, \qquad \vec{x}_n = \begin{pmatrix} x_n \\ y_n \end{pmatrix}.$$

We can write

$$\vec{x}_{n+1} = \vec{x}_n + hA\cdot\vec{x}_{n+1} \quad \Rightarrow \quad (I - hA)\vec{x}_{n+1} = \vec{x}_n \quad \Rightarrow \quad \vec{x}_{n+1} = (I - hA)^{-1}\vec{x}_n.$$

$$\vec{x}_{n+1} = (I - hA)^{-1}\vec{x}_n.$$

Take some vector norm $\|\cdot\|$ on both sides, we get

$$\|\vec{x}_{n+1}\| = \left\|(I - hA)^{-1}\vec{x}_n\right\| \leq \left\|(I - hA)^{-1}\right\| \cdot \|\vec{x}_n\|.$$

We see that if $\left\|(I - hA)^{-1}\right\| < 1$, then $\vec{x}_n \to 0$ as $n \to +\infty$.
Let's check this condition using the l_2 norm:

$$\left\|(I - hA)^{-1}\right\|_2 = \max_i \left|\lambda_i (I - hA)^{-1}\right| = \max_i \frac{1}{|(1 - h \cdot \lambda_i(A))|}.$$

We have

$$\lambda_1(A) = -1, \qquad \lambda_2(A) = -39$$

They are both negative, therefore $1 - h\lambda_i > 1$ for $i = 1, 2$, implying

$$\left\|(I - hA)^{-1}\right\|_2 < 1$$

independent of the value of h.
Therefore, this implicit method is called *unconditionally stable*.

Some critics on the implicit method:

- Advantage: On can choose large h, and the method is always stable. This is particularly suitable for stiff systems.
- Disadvantage: One must solve a system of linear equations at each time step

$$(I - hA)\vec{x}_{n+1} = \vec{x}_n$$

  Longer computing time for each step.
  In general, for nonlinear systems, implicit method leads to a system of nonlinear equations to solve at every time step. One must then use Newton/secant method. Very expensive to compute.

Therefore the implicit method is NOT recommended if the system is not stiff.