MATH 4513 Numerical Analysis

Chapter 5. Initial-Value Problems for Ordinary Differential Equations

Xu Zhang

Department of Mathematics
Oklahoma State University

Text Book: Numerical Analysis (10th edition)
R. L. Burden, D. J. Faires, A. M. Burden

Table of Contents

Chapter 5. Initial-Value Problems for Differential Equations

- 5.1 Elementary Theory of Initial-Value Problems
- 5.2 Euler's Method
- 5.3 Higher-Order Taylor Methods
- 5.4 Runge-Kutta Methods
- 5.5 Multistep Methods
- 5.6 Higher-Order Equations and Systems of Differential Equations

5.1 Elementary Theory of Initial-Value Problems

- In this chapter, we study numerical approximation of the initial-value problem (IVP) of ordinary differential equations.
- We start from the IVP of a first-order ODE

$$\begin{cases} \dfrac{dy}{dt} = f(t, y), & a \le t \le b \\ y(a) = \alpha \end{cases} \tag{5.1}$$

where α is a given constant.

Definition 1 (Lipschitz Condition).

A function $f(t, y)$ is said to satisfy a **Lipschitz condition** in the variable y on a set $D \subset \mathbb{R}^2$ if a constant $L > 0$ exists with

$$|f(t, y_1) - f(t, y_2)| \le L|y_1 - y_2| \tag{5.2}$$

whenever (t, y_1) and (t, y_2) are in D. The constant L is called a **Lipschitz constant** for f.

Example 2.

Show that $f(t, y) = 3ty$ satisfies the Lipschitz condition on the region

$$D = \{(t, y) | 1 \leq t \leq 2, \ -3 \leq y \leq 4\}.$$

Solution

For each pair of points (t, y_1) and (t, y_2) in D, we have

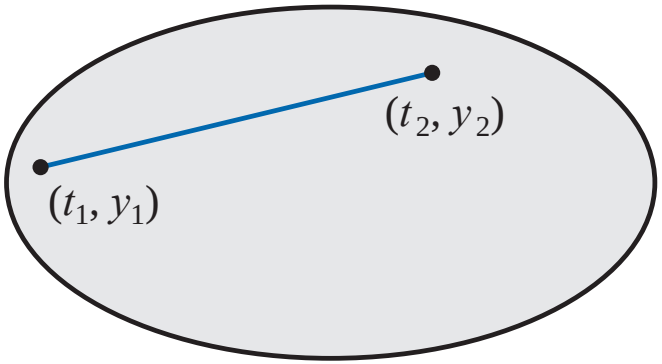
$$|f(t, y_1) - f(t, y_2)| = |3ty_1 - 3ty_2| = 3t|y_1 - y_2| \leq 6|y_1 - y_2|.$$

Hence, the function satisfies Lipschitz condition with the Lipschitz constant $L = 6$.

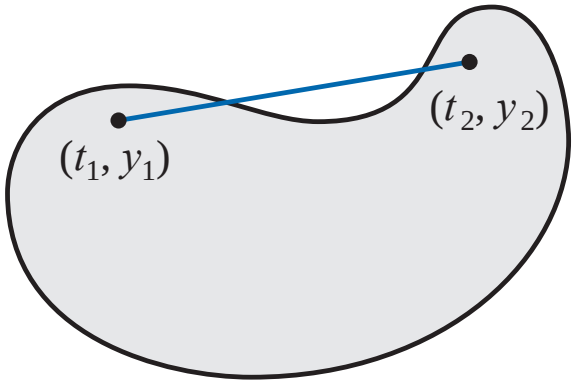
Definition 3 (Convex Set).

A set $D \subset \mathbb{R}^2$ is said to be **convex** if whenever (t_1, y_1) and (t_2, y_2) belong to D, then the entire straight line segment between these two points also belong to the set. That is

$$\Big((1-\lambda)t_1 + \lambda t_2, (1-\lambda)y_1 + \lambda y_2\Big) \in D, \quad \text{for all } \lambda \in [0,1]. \tag{5.3}$$



Convex



Not convex

Theorem 4 (A Sufficient Condition for Lipschitz Condition).

Suppose $f(t, y)$ is defined on a convex set $D \subset \mathbb{R}^2$. If there exists a constant L such that

$$\left|\frac{\partial f}{\partial y}(t, y)\right| \leq L, \quad \text{for all } (t, y) \in D, \tag{5.4}$$

then f satisfies a Lipschitz condition D in the variable y with Lipschitz constant L.

Theorem 5 (Existence and Uniqueness of First-Order Equations).

Suppose $D = \{(t, y)|a \leq t \leq b, \ -\infty < y < \infty\}$ and that $f(t, y)$ is continuous on D. If f satisfies the Lipschitz condition on D in the variable y, then the initial value problem

$$y'(t) = f(t, y), \quad y(a) = \alpha$$

has a unique solution $y(t)$ for $a \leq t \leq b$.

Example 6.

Show that there is a unique solution to the IVP

$$\begin{cases} y' = 1 + t\sin(ty), & 0 \le t \le 2, \\ y(0) = 2. \end{cases}$$

Solution

- In this example, $f(t, y) = 1 + t\sin(ty)$. Then

$$\left|\frac{\partial}{\partial y} f(t, y)\right| = |t^2 \cos(ty)| = |t^2||\cos(ty)| \le 2^2 \times 1 = 4.$$

- Hence, f satisfies Lipschitz condition with $L = 4$. By theorem 5, the IVP has a unique solution.

Well-Posed Problems

well-posedness

A **well-posed** mathematical problem should have the properties that

- a solution exists,
- the solution is unique,
- the solution's behavior changes continuously with the initial conditions. This means small changes in input will introduce small changes in the solution.

- Now that we have know the questions of when an IVP has solutions and when it is unique, we can move to the third question.
- The reason why the well-posedness is important is because IVPs obtained by observing physical phenomena generally only approximate the true situation, so we need to know if small errors in the statement of the problem introduce correspondingly small changes in the solution.
- Another reason is because of the round-off error when numerical methods are used.

Theorem 7 (Well-posedness Theorem).

Suppose $D \subset \mathbb{R}^2$. If f is continuous and satisfies a Lipschitz condition in the variable of y on the set D, then the initial-value problem

$$\frac{dy}{dt} = f(t, y), \quad a \le t \le b, \quad y(a) = \alpha$$

is well posed.

Example 8.

Show that the initial-value problem

$$y' = y - t^2 + 1, \ \ 0 \le t \le 2, \qquad y(0) = 0.5.$$

is well posed on $D = \{(t, y) | 0 \le t \le 2, \ -\infty < y < \infty\}$.

Solution

Because $f(t, y) = y - t^2 + 1$ is continuous and

$$\left| \frac{\partial f}{\partial y} \right| = \left| \frac{\partial (y - t^2 + 1)}{\partial y} \right| = 1.$$

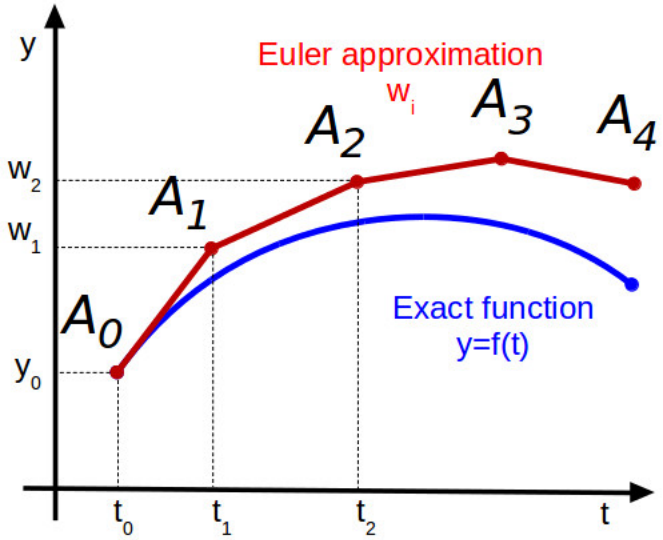
By Theorem 4, f satisfies Lipschitz condition. Thus the IVP is well posed.

5.2 Euler's Method

- In this section, we introduce the Euler's method, which is the most elementary approximation technique for solving the initial-value problem

$$\begin{cases} y' = f(t, y) & a \le t \le b, \\ y(a) = y_0. \end{cases}$$

- A continuous approximation to the solution $y(t)$ will not be obtained; instead, approximations to y will be generated at various values, called **grid points**, on the interval $[a, b]$.



- We divide the interval $[a, b]$ into N equal subintervals, with the **step size** $h = (b - a)/N$:

$$t_i = a + ih, \quad i = 0, 1, \cdots, N.$$

- By Taylor expansion and using $y'(t) = f(t, y)$, we have

$$\begin{aligned} y(t_{i+1}) &= y(t_i) + (t_{i+1} - t_i)y'(t_i) + \frac{(t_{i+1} - t_i)^2}{2}y''(\xi_i) \\ &= y(t_i) + hf(t_i, y(t_i)) + \frac{h^2}{2}y''(\xi_i) \end{aligned}$$

- Dropping the quadratic term, we have

Euler's Method

Given the initial data t_0 and y_0, and the step size h. For $i = 0, 1, \cdots, N - 1$,

$$\begin{aligned} t_{i+1} &= t_i + h, \\ y_{i+1} &= y_i + hf(t_i, y_i). \end{aligned} \tag{5.5}$$

Example 9.

Use Euler's method with $h = 0.5$ to solve the initial-value problem

$$y' = y - t^2 + 1, \quad 0 \le t \le 2, \quad y(0) = 0.5.$$

Solution

We have $t_0 = 0$, $y_0 = 0.5$, $h = 0.5$, and $f(t, y) = y - t^2 + 1$. Then

$$\begin{aligned}
t_1 &= t_0 + h = 0.5, & y_1 &= y_0 + hf(t_0, y_0) = 0.5 + 0.5(0.5 - 0^2 + 1) = 1.25 \\
t_2 &= t_1 + h = 1.0, & y_2 &= y_1 + hf(t_1, y_1) = 1.25 + 0.5(1.25 - 0.5^2 + 1) = 2.25 \\
t_3 &= t_2 + h = 1.5, & y_3 &= y_2 + hf(t_2, y_2) = 2.25 + 0.5(2.25 + 1^2 + 1) = 3.375 \\
t_4 &= t_3 + h = 2.0, & y_4 &= y_3 + hf(t_3, y_3) = 3.375 + 0.5(3.375 + 1.5^2 + 1) = 4.4375
\end{aligned}$$

The exact solution of this IVP is $y(t) = (t + 1)^2 - 0.5e^t$. Thus

	i	0	1	2	3	4
	t_i	0	0.5	1.0	1.5	2.0
$Exact$	$y(t_i)$	0.5	1.4265	2.6409	4.0092	5.3055
$Euler$	y_i	0.5	1.25	2.25	3.375	4.4375
$Error$	$\lvert y_i - y(t_i)\rvert$	0	0.1765	0.3909	0.6342	0.8675

Now we write a code for the Euler's method. The inputs should contain
(1) function f,
(2) endpoints of interval a and b,
(3) initial point y_0,
(4) the number of partition N (or the step size $h = \frac{b-a}{N}$).

MATLAB Code for Euler Method

```
function [t,y] = eul(fun,a,b,y0,N)
% Euler's method for solving the IVP
%   y' = f(t,y) in [a,b],  y(a) = y0

h = (b-a)/N;
t = zeros(N+1,1);
y = zeros(N+1,1);
t(1) = a;
y(1) = y0;

for i = 1:N
    t(i+1) = t(i) + h;
    y(i+1) = y(i) + h*fun(t(i),y(i));
end
```

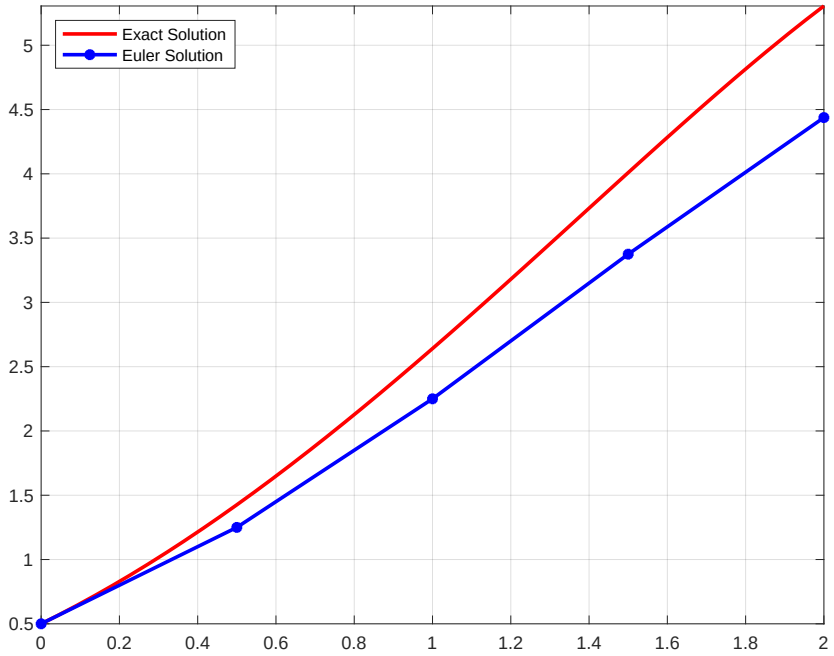
Driver File of Example 5.1

```
%% ex_5_1_1
clc
fun = @(t,y) y-t^2+1; % function f(t,y)
y0 = 0.5; % initial conditino
a = 0; % starting time
b = 2; % ending time
N = 4; % number of subintervals, h = (b-a)/N
[t,y] = eul(fun,a,b,y0,N);

%% Display Solution
exactY = @(t) (t+1).^2-0.5*exp(t);
Y = exactY(t);
disp('-----------------------------------------------');
disp(['          Euler Method   h = ', num2str((b-a)/N)])
disp('-----------------------------------------------');
disp('t_i        y_i        y(t_i)     |y_i-y(t_i)|')
disp('-----------------------------------------------')
formatSpec = '%.4f    % .4f    % .4f    % .4f\n';
fprintf(formatSpec,[t';y';Y';(abs(y-Y))'])

%% Plot Solution
fplot(exactY, [a,b],'r-','linewidth',2)
hold on
plot(t,y,'b*-','linewidth',2)
hold off
legend('Exact Solution','Euler Solution','Location','northwest')
grid on
```

- The output of Matlab code is

```
-------------------------------------------
          Euler Method   h = 0.5
-------------------------------------------
t_i         y_i         y(t_i)      |y_i-y(t_i)|
-------------------------------------------
0.0000      0.5000      0.5000      0.0000
0.5000      1.2500      1.4256      0.1756
1.0000      2.2500      2.6409      0.3909
1.5000      3.3750      4.0092      0.6342
2.0000      4.4375      5.3055      0.8680
```

- The plot of the numerical solution and exact solution are



- Note that the error grows as t increases.

Example 10.

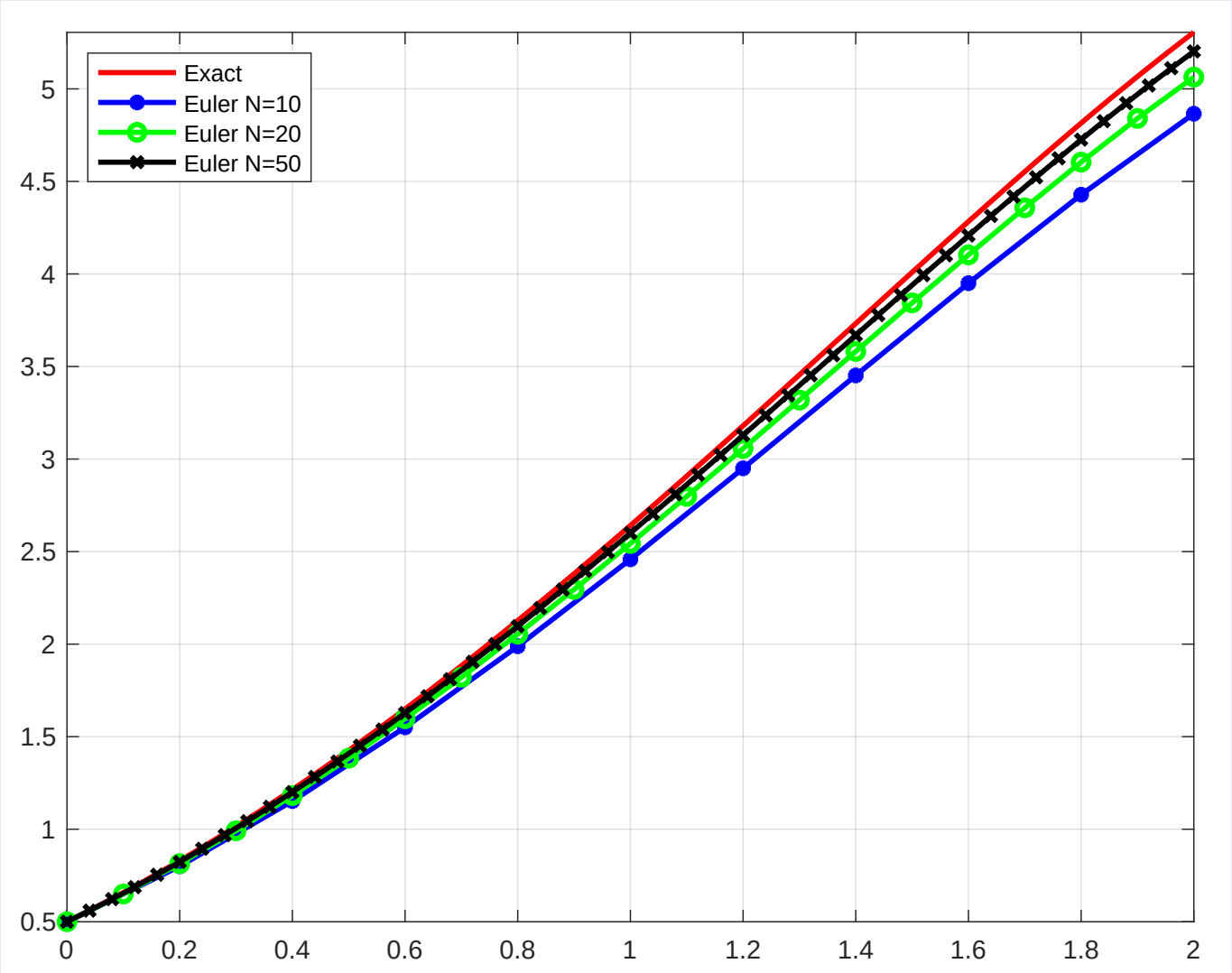
Re-do Example 9 using Euler's method with different step sizes:

$$h = \frac{1}{10}, \quad h = \frac{1}{20}, \quad h = \frac{1}{50}$$

Solution (1/2) MATLAB Driver File

```
%% ex_5_1_1 different step sizes
clc
fun = @(t,y) y-t^2+1;
y0 = 0.5;
a = 0; % starting time
b = 2; % ending time
N1 = 10;
[t1,y1] = eul(fun,a,b,y0,N1);
N2 = 20;
[t2,y2] = eul(fun,a,b,y0,N2);
N3 = 50;
[t3,y3] = eul(fun,a,b,y0,N3);

%% Plot Solution
exactY = @(t) (t+1).^2-0.5*exp(t);
fplot(exactY, [a,b],'r-','linewidth',2)
hold on
plot(t1,y1,'b*-',t2,y2,'go-',t3,y3,'kx-','linewidth',2)
hold off
legend('Exact','Euler N=10','Euler N=20','Euler N=50','Location','northwest')
grid on
```

Solution (2/2) Plots of Euler's Approximations with different N



The approximations become more accurate as the step size h gets smaller.

Theorem 11 (Error Bound for Euler's Method).

Suppose f is continuous and satisfies the Lipschitz condition with constant L on $[a, b] \times (-\infty, \infty)$. We also assume that $|y''(t)| \leq M$ for all $t \in [a, b]$, where y is the solution to the initial value problem

$$y' = f(t, y), \quad a \leq t \leq b, \quad y(a) = \alpha$$

Let $y_0, y_1, \cdots, y_N$ be the approximations generated by Euler's methods. Then for each $i = 0, 1, 2, \cdots, N$,

$$|y(t_i) - y_i| \leq \frac{Mh}{2L}\left[e^{L(t_i - a)} - 1\right]. \tag{5.6}$$

Remark

- From the error bound,

$$e_i \triangleq y(t_i) - y_i \leq \frac{Mh}{2L}\left[e^{L(t_i-a)} - 1\right]$$

  we note that the error e_i depends linearly on h, i.e. $e_i \approx \mathcal{O}(h)$.
- For a fixed h, the error e_i grows as t_i increases because of the exponential term.
- e_i is called the **global truncation error**. Euler's method is a first-order method.
- The **local truncation error** denotes the error made in a single step. Since,

$$\begin{aligned} y(t_1) &= y(t_0) + hy'(t_0) + \frac{1}{2}h^2y''(\xi) \\ &= y(t_0) + hf(t_0, y_0) + \frac{1}{2}h^2y''(\xi) \\ &= y_1 + \frac{1}{2}h^2y''(\xi) \end{aligned}$$

  Thus, $|y(t_1) - y_1| = \frac{1}{2}h^2|y''(\xi)|$. The local truncation error is $\mathcal{O}(h^2)$.

5.3 Higher-Order Taylor Methods

- Recall that the Euler's method is a first-order method. This means the global truncation error $|y_i - y(t_i)| \approx \mathcal{O}(h)$. The local truncation error $|y_1 - y(t_1)| \approx \mathcal{O}(h^2)$.
- In this section, we introduce some higher-order numerical methods for solving the initial-value problem

$$\begin{cases} y' = f(t, y) & a \le t \le b, \\ y(a) = y_0. \end{cases}$$

- Note that in deriving the Euler method, we used the Taylor expansion,

$$y(t_{i+1}) = y(t_i) + hy'(t_i) + \frac{h^2}{2} y''(\xi),$$

then dropped the last term involving h^2. (That is why the LTE $\approx \mathcal{O}(h^2)$)

- To improve the accuracy, we will keep more terms in the Taylor expansion,

$$\begin{aligned} y(t_{i+1}) &= y(t_i) + hy'(t_i) + \frac{h^2}{2}y''(t_i) + \cdots + \frac{h^n}{n!}y^{(n)}(t_i) + \frac{h^{n+1}}{(n+1)!}y^{(n+1)}(\xi_i) \\ &= y(t_i) + hf(t_i, y(t_i)) + \frac{h^2}{2}f'(t_i, y(t_i)) + \cdots + \frac{h^n}{n!}f^{(n-1)}(t_i, y(t_i)) \\ &\quad + \frac{h^{n+1}}{(n+1)!}f^{(n)}(\xi_i, y(\xi_i)) \end{aligned}$$

Taylor's Method of order n

Given t_0, y_0, and h. For $i = 0, 1, \cdots, N-1$,

$$\begin{aligned} t_{i+1} &= t_i + h, \\ y_{i+1} &= y_i + hf(t_i, y_i) + \frac{h^2}{2}f'(t_i, y_i) + \cdots + \frac{h^n}{n!}f^{(n-1)}(t_i, y_i). \end{aligned} \tag{5.7}$$

Remark

- The local truncation error is $O(h^{n+1})$.
- Euler's method is Taylor's method of order 1.
- The method requires the evaluation of the derivatives of $f(t, y)$.

Example 12.

Apply Taylor's method of order two and four with $N = 10$ to the initial value problem

$$y' = y - t^2 + 1, \quad 0 \le t \le 2, \quad y(0) = 0.5.$$

Solution (1/3)

- For Taylor's method of order two, we have

$$\begin{aligned} f(t, y(t)) &= y(t) - t^2 + 1. \\ f'(t, y(t)) &= y'(t) - 2t = y(t) - t^2 + 1 - 2t \end{aligned}$$

- Therefore, the 2nd-order Taylor method is

$$\begin{aligned} y_{i+1} &= y_i + hf(t_i, y_i) + \frac{h^2}{2} f'(t_i, y_i) \\ &= y_i + h(y_i - t_i^2 + 1) + \frac{h^2}{2}(y_i - t_i^2 + 1 - 2t_i). \end{aligned}$$

Solution (2/3)

- Given $t_0 = 0$, $y_0 = 0.5$, $N = 10$, so $h = \dfrac{2-0}{10} = 0.2$, and

$$\begin{aligned} t_1 &= t_0 + h = 0.2. \\ y_1 &= y_0 + h(y_0 - t_0^2 + 1) + \frac{h^2}{2}(y_0 - t_0^2 + 1 - 2t_0) = 0.83. \end{aligned}$$

$$\begin{aligned} t_2 &= t_1 + h = 0.4. \\ y_2 &= y_1 + h(y_1 - t_1^2 + 1) + \frac{h^2}{2}(y_1 - t_1^2 + 1 - 2t_1) = 1.2158 \\ &\vdots \end{aligned}$$

- We will write codes to finish the rest of computation.

Solution (3/3)

- For 4th-order Taylor's method, we have

$$\begin{aligned} f(t,y(t)) &= y - t^2 + 1. \\ f'(t,y(t)) &= y' - 2t = y - t^2 + 1 - 2t \\ f''(t,y(t)) &= y' - 2t - 2 = y - t^2 - 2t - 1 \\ f'''(t,y(t)) &= y' - 2t - 2 = y - t^2 - 2t - 1 \end{aligned}$$

- Therefore, the 4th-order Taylor method is

$$\begin{aligned} y_{i+1} = {} & y_i + h(y_i + t_i^2 + 1) + \frac{h^2}{2}(y_i - t_i^2 - 2t_i + 1) \\ & + \frac{h^3}{6}(y_i - t_i^2 - 2t_i + 1) + \frac{h^4}{24}(y_i - t_i^2 - 2t_i + 1) \end{aligned}$$

- We will finish the computation with programming.

MATLAB Code for 2nd-order Taylor Method

```
function [t,y] = taylor2(fun,Dfun,a,b,y0,N)

h = (b-a)/N;
t = zeros(N+1,1);
y = zeros(N+1,1);
t(1) = a;
y(1) = y0;

for i = 1:N
    t(i+1) = t(i) + h;
    y(i+1) = y(i) + h*fun(t(i),y(i)) + h^2/2*Dfun(t(i),y(i));
end
```

MATLAB Code for 4th-order Taylor Method

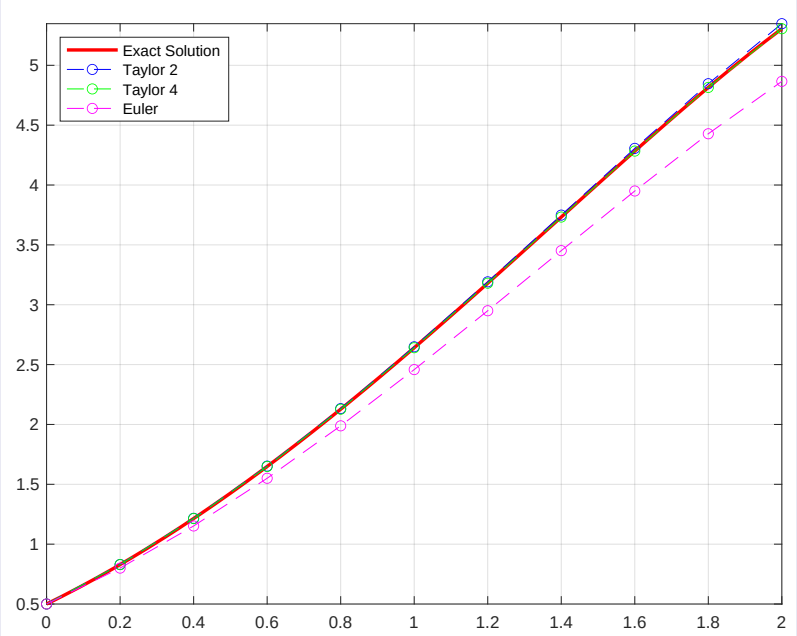
```
function [t,y] = taylor4(fun,Dfun,DDfun,DDDfun,a,b,y0,N)

h = (b-a)/N;
t = zeros(N+1,1);
y = zeros(N+1,1);
t(1) = a;
y(1) = y0;

for i = 1:N
    t(i+1) = t(i) + h;
    y(i+1) = y(i) + h*fun(t(i),y(i)) + h^2/2*Dfun(t(i),y(i)) ...
        + h^3/6*DDfun(t(i),y(i)) + h^4/24*DDDfun(t(i),y(i));
end
```

MATLAB Driver File

```
%% ex_5_3_1
clc
fun = @(t,y) y-t.^2+1;
Dfun = @(t,y) y-t.^2+1 -2*t;
DDfun = @(t,y) y-t.^2 -2*t - 1;
DDDfun = @(t,y) y-t.^2 -2*t -1;
a = 0; b = 2; y0 = 0.5; N = 10;
[t2,y2] = taylor2(fun,Dfun,a,b,y0,N);
[t4,y4] = taylor4(fun,Dfun,DDfun,DDDfun,a,b,y0,N);
[t1,y1] = eul(fun,a,b,y0,N);

%% Display Solution
exactY = @(t) (t+1).^2-0.5*exp(t); Y = exactY(t1);
disp('-------------------------------------------------------------------------');
disp('      Exact          Euler            Taylor(2)           Taylor(4) ')
disp('-------------------------------------------------------------------------');
disp('t_i   y(t_i)    y_i       |e_i|     y_i       |e_i|     y_i       |e_i|')
disp('-------------------------------------------------------------------------')
formatSpec = '%.1f  % .5f  % .5f  % .5f   % .5f  % .5f   % .5f  % .5f\n';
fprintf(formatSpec,[t1';Y';y1';(abs(y1-Y))';y2';(abs(y2-Y))';y4';(abs(y4-Y))'])

%% Plot Solution
fplot(exactY, [a,b],'r-','linewidth',2)
hold on
plot(t2,y2,'bo--')
plot(t4,y4,'go--')
plot(t1,y1,'mo--')
hold off; grid on
legend('Exact Solution','Taylor 2','Taylor 4','Euler','Location','northwest')
```

MATLAB Outputs

	Exact	Euler		Taylor(2)		Taylor(4)	
t_i	y(t_i)	y_i	|e_i|	y_i	|e_i|	y_i	|e_i|
0.0	0.50000	0.50000	0.00000	0.50000	0.00000	0.50000	0.00000
0.2	0.82930	0.80000	0.02930	0.83000	0.00070	0.82930	0.00000
0.4	1.21409	1.15200	0.06209	1.21580	0.00171	1.21409	0.00000
0.6	1.64894	1.55040	0.09854	1.65208	0.00314	1.64895	0.00001
0.8	2.12723	1.98848	0.13875	2.13233	0.00510	2.12724	0.00001
1.0	2.64086	2.45818	0.18268	2.64865	0.00779	2.64087	0.00002
1.2	3.17994	2.94981	0.23013	3.19135	0.01141	3.17996	0.00002
1.4	3.73240	3.45177	0.28063	3.74864	0.01624	3.73243	0.00003
1.6	4.28348	3.95013	0.33336	4.30615	0.02266	4.28353	0.00004
1.8	4.81518	4.42815	0.38702	4.84630	0.03112	4.81524	0.00006
2.0	5.30547	4.86578	0.43969	5.34768	0.04221	5.30556	0.00008



Approximation at Intermediate Points

- As we can see in the last example, the 4th-order Taylor method generates very accurate approximation at the nodes $x = 0.2, 0.4, 0.6, \cdots, 1.8, 2.0$.
- What if we want to determine the value of an intermediate point, say $x = 1.527$?
- Let's try first using the Lagrange linear interpolating polynomial at $x = 1.4$ and $x = 1.6$, we have

$$\begin{aligned} P_1(1.527) &= y(1.4)\frac{1.527 - 1.6}{1.4 - 1.6} + y(1.6)\frac{1.527 - 1.4}{1.6 - 1.4} \\ &\approx 3.73243\frac{1.527 - 1.6}{1.4 - 1.6} + 4.28353\frac{1.527 - 1.4}{1.6 - 1.4} \\ &= 4.08238 \end{aligned}$$

- The true solution $y(t) = (t+1)^2 - 0.5e^t$ implies $y(1.527) = 4.08356$.
- The absolute error is 0.00118, which is over 30 times the errors at $x = 1.4$ and $x = 1.6$. (error between 0.00003 and 0.00004)

- We can significantly improve the result by using high-order interpolation. For example, cubic Hermite interpolation.
- However, Hermite interpolation requires the derivative $y'(1.4)$ and $y'(1.6)$.
- By evaluating the differential equation, $y'(t) = f(t, y) = y - t^2 + 1$

$$y'(1.4) = 3.73243 - 1.4^2 + 1 = 2.77243, \quad y'(1.6) = 4.28353 - 1.6^2 + 1 = 2.72353$$

- By Newton's Divided Difference table (for Hermite)

1.4	3.73243	2.77243	-0.08465	-0.37600
1.4	3.73243	2.75550	-0.15985	
1.6	4.28353	2.72353		
1.6	4.28353			

- The cubic Hermite interpolation is

$$H_3(x) = 3.73243 + 2.77243(x - 1.4) - 0.08465(x - 1.4)^2 - 0.37600(x - 1.4)^2(x - 1.6).$$

$$H_3(1.527) = 4.08361$$

- The error now becomes 0.00005, which is comparable to the errors at $x = 1.4$ and $x = 1.6$. (error between 0.00003 and 0.00004)

5.4 Runge-Kutta Methods

- The Taylor methods introduced in the last section have desirable property of high-order local truncation error.
- However, the drawback of Taylor methods is that it requires the computation and evaluation of derivatives of $f(t, y)$. This is a complicated and time-consuming procedure for most problems, so the Taylor methods are seldom used in practice.
- In this section, we introduce **Runge-Kutta methods** that have the high-order local truncation error like the Taylor methods, but eliminate the need to compute and evaluate derivatives of $f(t, y)$.
- We consider the model initial-value problem

$$\begin{cases} y' = f(t, y), \\ y(t_0) = y_0. \end{cases}$$

- We will first derive the 2nd-order Runge-Kutta method. Taking the derivative with respect to t on both side of the differential equation

$$y' = f(t, y(t))$$

and using the chain rule, we have

$$y''(t) = f_t(t, y) + f_y(t, y)y'(t) = f_t(t, y) + f_y(t, y)f(t, y) \tag{5.8}$$

- The Taylor series for $y(t + h)$ around t is

$$\begin{aligned} y(t+h) &= y(t) + hy'(t) + \frac{h^2}{2!}y''(t) + \mathcal{O}(h^3) \\ &= y(t) + hf(t, y) + \frac{h^2}{2!}\Big(f_t(t, y) + f_y(t, y)f(t, y)\Big) + \mathcal{O}(h^3) \end{aligned} \tag{5.9}$$

- Now let us see how to compute the term $f_t(t, y) + f_y(t, y)f(t, y)$ without knowing the derivative of f.

- Note that

$$y(t+h) = y(t) + hf(t,y) + \frac{h^2}{2!}\Big(f_t(t,y) + f_y(t,y)f(t,y)\Big) + \mathcal{O}(h^3)$$

- To approximate the red terms, we use the **Taylor series in two variables**

$$f(t+ah, y+bh) = f(t,y) + ahf_t(t,y) + bhf_y(t,y) + \mathcal{O}(h^2).$$

- Choosing $a = 1$, $b = f$, we have

$$f(t+h, y+hf) = f(t,y) + hf_t(t,y) + hf(t,y)f_y(t,y) + \mathcal{O}(h^2).$$

- Thus, multiplying by $h/2$ on both sides we have

$$\frac{h}{2}f(t+h, y+hf) = \frac{h}{2}f(t,y) + \frac{h^2}{2}\Big(f_t(t,y) + f(t,y)f_y(t,y)\Big) + \frac{h}{2}\mathcal{O}(h^2).$$

- Plug in to (5.9)

$$\begin{aligned} y(t+h) &= y + \frac{h}{2}f(t,y) + \frac{h}{2}\Big(f(t+h, y+hf(t,y)) + \mathcal{O}(h^2)\Big) + \mathcal{O}(h^3) \\ &= y + \frac{h}{2}f(t,y) + \frac{h}{2}f(t+h, y+hf(t,y)) + \mathcal{O}(h^3). \end{aligned}$$

- Dropping the higher order term $\mathcal{O}(h^3)$, we have the formula

$$y(t+h) \approx y(t) + \frac{h}{2} f(t,y) + \frac{h}{2} f(t+h, y+hf(t,y)).$$

which has the local truncation error $\mathcal{O}(h^3)$ and does not require evaluation of the derivative of f.

- The method can be written as: Given y_0,

$$y_{i+1} = y_i + \frac{h}{2} f(t_i, y_i) + \frac{h}{2} f(t_{i+1}, y_i + hf(t_i, y_i)), \qquad i = 0, 1, \cdots, N-1$$

- Equivalently, we can write it as a two-stage method

Modified Euler Method

$$\text{Given } y_0, \qquad y_{i+1} = y_i + \frac{1}{2}(k_1 + k_2), \quad i = 0, 1, \cdots, N-1 \tag{5.10}$$

$$\text{where} \quad k_1 = hf(t_i, y_i),$$
$$k_2 = hf(t_i + h, y_i + k_1).$$

- The LTE of the modified Euler method is $\mathcal{O}(h^3)$, and the GTE is $\mathcal{O}(h^2)$.

- As from the aformentioned procedure, we basically need to select coefficients a_1, α, β so that the red term in

$$y(t+h) = y(t) + hf(t,y) + \frac{h^2}{2!}\Big(f_t(t,y) + f_y(t,y)f(t,y)\Big) + \mathcal{O}(h^3)$$

and blue terms in

$$\begin{aligned} y(t+h) &= y(t) + a_1 f(t+\alpha, y+\beta) \\ &= y(t) + a_1 f(t,y) + a_1\alpha f_t(t,y) + a_1\beta f_y(t,y) + H.O.T. \end{aligned}$$

match. Here, $H.O.T$ stands for high-order terms.

- Comparing the coefficients, we have

$$a_1 = h, \quad a_1\alpha = \frac{h^2}{2}, \quad a_1\beta = \frac{h^2}{2}f(t,y)$$

- That means,

$$a_1 = h, \quad \alpha = \frac{h}{2}, \quad \beta = \frac{h}{2}f(t,y)$$

- This gives us another second-order method: Given y_0

$$y_{i+1} = y_i + hf\left(t_i + \frac{h}{2},\ y_i + \frac{h}{2}f(t_i, y_i)\right), \quad i = 0, 1, \cdots, N-1$$

- Or equivalently we write it as a two-stage method

Midpoint method

$$\text{Given } y_0, \qquad y_{i+1} = y_i + k_2, \quad i = 0, 1, \cdots, N-1 \tag{5.11}$$

$$\text{where} \quad k_1 = hf(t_i, y_i),$$

$$k_2 = hf(t_i + \frac{1}{2}h, y_i + \frac{1}{2}k_1).$$

- The LTE of the midpoint method is $\mathcal{O}(h^3)$, and the GTE is $\mathcal{O}(h^2)$.

Example 13.

Use the Midpoint's method and the Modified Euler method with $N = 10$, $h = 0.2$ to approximate the solution to our usual example

$$y' = y - t^2 + 1, \quad 0 \le t \le 2, \quad y(0) = 0.5.$$

Solution (1/2)

- Use Midpoint's method, $f(t, y) = y - t^2 + 1$, $h = 0.2$,
- For $i = 0$, $t_0 = 0$, $y_0 = 0.5$,

$$\begin{aligned} k_1 &= hf(t_0, y_0) = 0.2(0.5 - 0 + 1) = 0.3. \\ k_2 &= hf(t_0 + \frac{1}{2}h, y_0 + \frac{1}{2}k_1) = 0.2f(0.1, 0.65) = 0.328 \\ y_1 &= y_0 + k_2 = 0.828. \end{aligned}$$

- For $i = 1$, $t_1 = 0.2$, $y_1 = 0.828$,

$$\begin{aligned} k_1 &= hf(t_1, y_1) = 0.2(0.828 - 0.04 + 1) = 0.3576. \\ k_2 &= hf(t_1 + \frac{1}{2}h, y_1 + \frac{1}{2}k_1) = 0.2f(0.3, 1.0068) = 0.38336 \\ y_1 &= y_1 + k_2 = 1.21136. \quad \cdots \text{(Program to finish)} \end{aligned}$$

Solution (2/2)

- Use Modified Euler's method, $f(t, y) = y - t^2 + 1$, $h = 0.2$,
- For $i = 0$, $t_0 = 0$, $y_0 = 0.5$,

$$\begin{aligned} k_1 &= hf(t_0, y_0) = 0.2(0.5 - 0 + 1) = 0.3. \\ k_2 &= hf(t_0 + h, y_0 + k_1) = 0.2f(0.2, 0.8) = 0.352 \\ y_1 &= y_0 + \frac{1}{2}(k_1 + k_2) = 0.826. \end{aligned}$$

- For $i = 1$, $t_1 = 0.2$, $y_1 = 0.826$,

$$\begin{aligned} k_1 &= hf(t_1, y_1) = 0.2(0.5 - 0 + 1) = 0.3572. \\ k_2 &= hf(t_1 + h, y_1 + k_1) = 0.2f(0.2, 0.8) = 0.48082 \\ y_2 &= y_1 + \frac{1}{2}(k_1 + k_2) = 1.20692. \end{aligned}$$

- We will finish the rest of the computation by programming

MATLAB Code Modified Euler's Method

```
function [t,y] = eulmod(fun,a,b,y0,N)

h = (b-a)/N;
t = zeros(N+1,1);
y = zeros(N+1,1);
t(1) = a;
y(1) = y0;

for i = 1:N
    t(i+1) = t(i) + h;
    k1 = h*fun(t(i),y(i));
    k2 = h*fun(t(i)+h,y(i)+k1);
    y(i+1) = y(i) + 1/2*(k1 + k2);
end
```

MATLAB Code for Midpoint Method

```
function [t,y] = midpoint(fun,a,b,y0,N)

h = (b-a)/N;
t = zeros(N+1,1);
y = zeros(N+1,1);
t(1) = a;
y(1) = y0;

for i = 1:N
    t(i+1) = t(i) + h;
    k1 = h*fun(t(i),y(i));
    k2 = h*fun(t(i)+h/2,y(i)+k1/2);
    y(i+1) = y(i) + k2;
end
```

MATLAB Driver File

```
%% ex_5_4_2
clc
fun = @(t,y) y-t.^2+1;
y0 = 0.5;
a = 0;
b = 2;
N = 10;
[t1,y1] = eul(fun,a,b,y0,N);
[t2,y2] = eulmod(fun,a,b,y0,N);
[t3,y3] = midpoint(fun,a,b,y0,N);

%% Display Solution
exactY = @(t) (t+1).^2-0.5*exp(t); Y = exactY(t1);
disp('------------------------------------------------------------------');
disp('t_i   y(t_i)    1.Eul     1.Error   2.ModEul  2. Error  3.Midpt   3.Error')
disp('------------------------------------------------------------ ---------------------')
formatSpec = '%.1f  % .5f  % .5f  % .5f  % .5f  % .5f  % .5f  % .5f\n';
fprintf(formatSpec,[t1';Y';y1';(abs(y1-Y))';y2';(abs(y2-Y))';y3';(abs(y3-Y))'])
```

MATLAB Outputs

t_i	y(t_i)	1.Eul	1.Error	2.ModEul	2. Error	3.Midpt	3.Error
0.0	0.50000	0.50000	0.00000	0.50000	0.00000	0.50000	0.00000
0.2	0.82930	0.80000	0.02930	0.82600	0.00330	0.82800	0.00130
0.4	1.21409	1.15200	0.06209	1.20692	0.00717	1.21136	0.00273
0.6	1.64894	1.55040	0.09854	1.63724	0.01170	1.64466	0.00428
0.8	2.12723	1.98848	0.13875	2.11024	0.01699	2.12128	0.00595
1.0	2.64086	2.45818	0.18268	2.61769	0.02317	2.63317	0.00769
1.2	3.17994	2.94981	0.23013	3.14958	0.03036	3.17046	0.00948
1.4	3.73240	3.45177	0.28063	3.69369	0.03871	3.72117	0.01123
1.6	4.28348	3.95013	0.33336	4.23510	0.04839	4.27062	0.01286
1.8	4.81518	4.42815	0.38702	4.75562	0.05956	4.80096	0.01422
2.0	5.30547	4.86578	0.43969	5.23305	0.07242	5.29037	0.01510

- Both the Midpoint method and the Modified Euler method outperform Classical Euler method. The former are second-order methods, and the latter is a first-order method.
- For this problem, the result of midpoint method is superior to the modified Euler method.

The most common Runge-Kutta method in use is

Runge-Kutta Method (Order Four)

Given y_0,

$$y_{i+1} = y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4), \quad i = 0, 1, \cdots, N-1 \tag{5.12}$$

where
$$\begin{aligned} k_1 &= hf\left(t_i, y_i\right), \\ k_2 &= hf\left(t_i + \frac{1}{2}h, y_i + \frac{1}{2}k_1\right), \\ k_3 &= hf\left(t_i + \frac{1}{2}h, y_i + \frac{1}{2}k_2\right), \\ k_4 &= hf\left(t_i + h, y_i + k_3\right). \end{aligned}$$

- The method RK4 is a four-stage method (requiring four evaluations of function f at each step).
- The LTE for RK4 is $\mathcal{O}(h^5)$. The GTE is $\mathcal{O}(h^4)$

Example 14.

Use the 4th-order Runge-Kutta method with $N = 10$ to approximate the solution to our usual example

$$y' = y - t^2 + 1, \quad 0 \le t \le 2, \quad y(0) = 0.5.$$

Solution (1/3)

- Note that $f(t, y) = y - t^2 + 1$, $h = 0.2$
- For $i = 0$, $t_0 = 0$, $y_0 = 0.5$,

$$\begin{aligned}
k_1 &= hf(t_0, y_0) = 0.2f(0, 0.5) = 0.2(0.5 - 0 + 1) = 0.3. \\
k_2 &= hf(t_0 + \frac{h}{2}, y_0 + \frac{1}{2}k_1) = 0.2f(0.1, 0.65) = 0.328 \\
k_3 &= hf(t_0 + \frac{h}{2}, y_0 + \frac{1}{2}k_2) = 0.2f(0.1, 0.664) = 0.3308 \\
k_4 &= hf(t_0 + h, y_0 + k_3) = 0.2f(0.2, 0.8308) = 0.35816 \\
y_1 &= y_0 + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \\
&= 0.5 + \frac{1}{6}(0.3 + 2(0.328) + 2(0.3308) + 0.35816) = 0.8292933.
\end{aligned}$$

Solution (2/3) MATLAB File of the Runge-Kutta Method

```
function [t,y] = rk4(fun,a,b,y0,N)

h = (b-a)/N;
t = zeros(N+1,1);
y = zeros(N+1,1);
t(1) = a;
y(1) = y0;

for i = 1:N
    t(i+1) = t(i) + h;
    k1 = h*fun(t(i),y(i));
    k2 = h*fun(t(i)+h/2,y(i)+k1/2);
    k3 = h*fun(t(i)+h/2,y(i)+k2/2);
    k4 = h*fun(t(i)+h,y(i)+k3);
    y(i+1) = y(i) + 1/6*(k1 + 2*k2 + 2*k3 + k4);
end
```

Solution (2/3) MATLAB Driver File

```
%% ex_5_4_4
clc
fun = @(t,y) y-t.^2+1;
y0 = 0.5;
a = 0;
b = 2;
N = 10;
[t,y] = rk4(fun,a,b,y0,N);

%% Display Solution
exactY = @(t) (t+1).^2-0.5*exp(t); Y = exactY(t);
disp('-------------------------------------');
disp('t_i   y(t_i)     RK4        Error')
disp('-------------------------------------')
formatSpec = '%.1f  % .6f  % .6f  % .6f\n';
fprintf(formatSpec,[t';Y';y';(abs(y-Y))'])
```

Solution (3/3)

The numerical result is

t_i	y(t_i)	RK4	Error
0.0	0.500000	0.500000	0.000000
0.2	0.829299	0.829293	0.000005
0.4	1.214088	1.214076	0.000011
0.6	1.648941	1.648922	0.000019
0.8	2.127230	2.127203	0.000027
1.0	2.640859	2.640823	0.000036
1.2	3.179942	3.179894	0.000047
1.4	3.732400	3.732340	0.000060
1.6	4.283484	4.283409	0.000074
1.8	4.815176	4.815086	0.000091
2.0	5.305472	5.305363	0.000109

Computational Comparison

- The main computational effort in applying the Runge-Kutta methods is the evaluation of $f(t, y)$.
- In Euler's method (1st-order), we require only one evaluation per step.

$$y_{i+1} = y_i + hf(t_i, y_i).$$

- In Modified Euler method

$$y_{i+1} = y_i + \frac{1}{2}(k_1 + k_2), \quad \text{where } k_1 = hf(t_i, y_i), \quad k_2 = hf(t_i + h, y_i + k_1).$$

and the Midpoint method (both 2nd-order),

$$y_{i+1} = y_i + k_2, \quad \text{where } k_1 = hf(t_i, y_i), \quad k_2 = hf(t_i + \frac{1}{2}h, y_i + \frac{1}{2}k_1).$$

we require two evaluations per step.

- The 4th-order Runge-Kutta method requires four evaluations per step. evaluations per step.

- The table below summarizes the best possible order of the global truncation error.

Evaluation	2	3	4	$5 \leq n \leq 7$	$8 \leq n \leq 9$	$10 \leq n$
Best Order	$\mathcal{O}(h^2)$	$\mathcal{O}(h^3)$	$\mathcal{O}(h^4)$	$\mathcal{O}(h^{n-1})$	$\mathcal{O}(h^{n-2})$	$\mathcal{O}(h^{n-3})$

- To fairly compare the performance of different Runge-Kutta methods, we should set the total number of the function evaluations to be the same.
- That means, for a test problem, we should choose different step sizes for different methods so that their total number of the function evaluation is about the same.
- For example, consider the usual example

$$y' = y - t^2 + 1, \quad 0 \leq t \leq 2, \quad y(0) = 0.5.$$

we let $h = 0.05$ for Euler, and $h = 0.1$ for Modified Euler, and $h = 0.2$ for 4th-order Runge-Kutta. So to get from $t = 0$ to $t = 2$, each method requires 40-time evaluations.

- The numerical result is

t_i	Exact	Euler h = 0.05	ModEul h = 0.1	RK4 h = 0.2
0.0	0.5000000	0.5000000	0.5000000	0.5000000
0.2	0.8292986	0.8214716	0.8284350	0.8292933
0.4	1.2140877	1.1973995	1.2122113	1.2140762
0.6	1.6489406	1.6222790	1.6458789	1.6489220
0.8	2.1272295	2.0894190	2.1227827	2.1272027
1.0	2.6408591	2.5906863	2.6347973	2.6408227
1.2	3.1799415	3.1161950	3.1720009	3.1798942
1.4	3.7324000	3.6539290	3.7222788	3.7323401
1.6	4.2834838	4.1892822	4.2708390	4.2834095
1.8	4.8151763	4.7045011	4.7996197	4.8150857
2.0	5.3054720	5.1780062	5.2865672	5.3053630

- We can see from this example that the 4th-order method is clearly superior.

5.5 Multistep Methods

- The methods discussed in previous sections are called **one-step methods** because the approximation for the mesh point t_{i+1} involves information from only one of the previous mesh points, t_i.
- Note that the approximate solution is available at each of the mesh points $t_0, t_1, \cdots, t_i$ before the approximation at t_{i+1} is obtained, and because the error $|y_j - y(t_j)|$ tends to increase with j, so it seems reasonable to develop methods that use these more accurate previous data when approximating the solution at t_{i+1}.
- Methods using the approximation at more than one previous mesh point to determine the approximation at the next point are called **multistep methods**.

Definition 15 (multistep method).

An **m-step multistep method** for solving

$$\begin{cases} y' = f(t, y) & a \le t \le b \\ y(a) = y_0. \end{cases}$$

is

$$\begin{aligned} y_{i+1} = a_{m-1}y_i + a_{m-2}y_{i-1} + \cdots a_0 y_{i-m+1} \\ + h[b_m f_{i+1} + b_{m-1}f_i + \cdots b_0 f_{i-m+1}], \end{aligned} \tag{5.13}$$

for $i = m-1, m, \cdots, N-1$, where $h = (b-a)/N$, the $a_0, \cdots, a_{m-1}$, $b_0, \cdots, b_m$ are constants, and $f_j = f(t_j, y_j)$.

Remark

- When $b_m = 0$, the method is called **explicit**.
- When $b_m \neq 0$, the method is called **implicit**.
- To use the m-step multistep method, the starting values $y_0, y_1, \cdots, y_{m-1}$ must be specified.

- For example,

$$\text{Given } y_0, y_1, y_2, y_3, \text{ for each } i = 3, 4, \cdots, N-1,$$
$$y_{i+1} = y_i + \frac{h}{24}[55f(t_i, y_i) - 59f(t_{i-1}, y_{i-1}) + 37f(t_{i-2}, y_{i-2}) - 9f(t_{i-3}, y_{i-3})]$$

This is an explicit four-step method (known as **fourth-order Adams-Bashforth technique**).

- Another example,

$$\text{Given } y_0, y_1, y_2, \text{ for each } i = 2, 3, \cdots, N-1,$$
$$y_{i+1} = y_i + \frac{h}{24}[9f(t_{i+1}, y_{i+1}) + 19f(t_i, y_i) - 5f(t_{i-1}, y_{i-1}) + f(t_{i-2}, y_{i-2})]$$

This is an implicit three-step method (known as **fourth-order Adams-Moulton technique**).

- Now let us see how to derive the multistep methods.
- Note the differential equation is

$$y' = f(t, y) \quad a \leq t \leq b$$

- Taking integral on both sides over $[t_i, t_{i+1}]$, we have

$$y(t_{i+1}) = y(t_i) + \int_{t_i}^{t_{i+1}} f(t, y(t))dt.$$

- We cannot integrate $f(t, y(t))$ without knowing $y(t)$, so we integrate an interpolating polynomial $P(t)$ to $f(t, y(t))$, one that is determined by some previous obtained data points $(t_0, y_0), \cdots, (t_i, y_i)$. Then

$$y(t_{i+1}) \approx y_i + \int_{t_i}^{t_{i+1}} P(t)dt.$$

- For example, we let $P_1(t)$ interpolates $f(t, y(t))$ at two points (t_{i-1}, f_{i-1}), and (t_i, f_i), where $f_{i-1} = f(t_{i-1}, y_{i-1})$, and $f_i = f(t_i, y_i)$. That is

$$P_1(t) = f_{i-1}\frac{t - t_i}{t_{i-1} - t_i} + f_i\frac{t - t_{i-1}}{t_i - t_{i-1}} = \frac{1}{h}\Big(- f_{i-1}(t - t_i) + f_i(t - t_{i-1})\Big).$$

- Then

$$\begin{aligned}
\int_{t_i}^{t_{i+1}} P_1(t)dt &= \frac{1}{h}\int_{t_i}^{t_{i+1}} -f_{i-1}(t - t_i) + f_i(t - t_{i-1})dt \\
&= \frac{1}{h}\left(\frac{f_i}{2}(t - t_{i-1})^2 - \frac{f_{i-1}}{2}(t - t_i)^2\right)\Big|_{t_i}^{t_{i+1}} \\
&= \frac{1}{h}\left(\frac{3f_i}{2}h^2 - \frac{f_{i-1}}{2}h^2\right) \\
&= h\left(\frac{3}{2}f_i - \frac{1}{2}f_{i-1}\right).
\end{aligned}$$

- This is **Adam-Bashforth Two-Step Method**, an explicit method

$$y_{i+1} = y_i + h\left(\frac{3}{2}f_i - \frac{1}{2}f_{i-1}\right).$$

- Let's consider another example. Let $P_2(t)$ interpolates $f(t, y(t))$ at three points (t_{i-1}, f_{i-1}), (t_i, f_i), and (t_{i+1}, f_{i+1}).

$$P_2(t) = f_{i-1}\frac{(t-t_i)(t-t_{i+1})}{(t_{i-1}-t_i)(t_{i-1}-t_{i+1})} + f_i\frac{(t-t_{i-1})(t-t_{i+1})}{(t_i-t_{i-1})(t_i-t_{i+1})} + f_{i+1}\frac{(t-t_{i-1})(t-t_i)}{(t_{i+1}-t_{i-1})(t_{i+1}-t_i)}$$

- Taking the integral of $P_2(t)$ over $[t_i, t_{i+1}]$, we obtain

$$\int_{t_i}^{t_{i+1}} P_2(t)dt = h\left(\frac{1}{12}f_{i-1} - \frac{2}{3}f_i + \frac{5}{12}f_{i+1}\right)$$

- This is **Adam-Moulton Two-Step Method**, an implicit method

$$y_{i+1} = y_i + h\left(\frac{1}{12}f_{i-1} - \frac{2}{3}f_i + \frac{5}{12}f_{i+1}\right).$$

Adam-Bashforth Explicit Methods

- **AB two-step explicit method** (LTE = $O(h^3)$)

$$y_{i+1} = y_i + \frac{h}{2}(3f_i - f_{i-1}).$$

- **AB three-step explicit method** (LTE = $O(h^4)$)

$$y_{i+1} = y_i + \frac{h}{12}(23f_i - 16f_{i-1} + 5f_{i-2}).$$

- **AB four-step explicit method** (LTE = $O(h^5)$)

$$y_{i+1} = y_i + \frac{h}{24}(55f_i - 59f_{i-1} + 37f_{i-2} - 9f_{i-3}).$$

- **AB five-step explicit method** (LTE = $O(h^6)$)

$$y_{i+1} = y_i + \frac{h}{720}(1901f_i - 2774f_{i-1} + 2616f_{i-2} - 1274f_{i-3} + 251f_{i-4}).$$

Adam-Moulton Implicit Methods

- **AM two-step implicit method** (LTE = $O(h^3)$)

$$y_{i+1} = y_i + \frac{h}{12}(5f_{i+1} + 8f_i - f_{i-1}).$$

- **AM three-step implicit method** (LTE = $O(h^4)$)

$$y_{i+1} = y_i + \frac{h}{24}(9f_{i+1} + 19f_i - 5f_{i-1} + f_{i-2}).$$

- **AM four-step implicit method** (LTE = $O(h^5)$)

$$y_{i+1} = y_i + \frac{h}{720}(251f_{i+1} + 646f_i - 264f_{i-1} + 106f_{i-2} - 19f_{i-3}).$$

Example 16.

Consider the initial value problem

$$y' = y - t^2 + 1, \quad 0 \le t \le 2, \quad y(0) = 0.5.$$

Use the exact values given from $y(t) = (t+1)^2 - 0.5e^t$ as starting values. Apply the fourth-order Adam-Bashforth methods with $N = 10$.

Solution

- The Adam-Bashforth four-step method is

$$y_{i+1} = y_i + \frac{h}{24}(55f_i - 59f_{i-1} + 37f_{i-2} - 9f_{i-3}).$$

- For this problem, $h = 0.2$, $t_0 = 0$, $y_0 = 0.5$. $f(t, y) = y - t^2 + 1$.
- Use exact solution $y(t) = (t+1)^2 - 0.5e^t$ as starting value

$$\begin{aligned} t_1 = 0.2, &\quad y_1 = y(t_1) = 0.5, &\quad f_1 = f(t_1, y_1) = 1.7893 \\ t_2 = 0.4, &\quad y_2 = y(t_2) = 0.8293, &\quad f_2 = f(t_2, y_2) = 2.0541 \\ t_3 = 0.6, &\quad y_3 = y(t_3) = 1.2141, &\quad f_3 = f(t_3, y_3) = 2.2889 \end{aligned}$$

- Use the Adam-Bashforth four-step method

$$y_4 = y_3 + \frac{h}{24}(55f_3 - 59f_2 + 37f_1 - 9f_0) = 2.1273.$$

Matlab Code for four-step Adam-Bashforth method

```
function [t,y] = adamBash4(fun,tspan,y0,y1,y2,y3,N)

a = tspan(1);
b = tspan(2);
h = (b-a)/N;
t = zeros(N+1,1);
y = zeros(N+1,1);
t(1) = a; y(1) = y0;
t(2) = a+h; y(2) = y1;
t(3) = a+2*h; y(3) = y2;
t(4) = a+3*h; y(4) = y3;

for i = 4:N
    t(i+1) = t(i) + h;
    Fi = fun(t(i),y(i));
    Fi1 = fun(t(i-1),y(i-1));
    Fi2 = fun(t(i-2),y(i-2));
    Fi3 = fun(t(i-3),y(i-3));
    y(i+1) = y(i) + h/24*(55*Fi - 59*Fi1 + 37*Fi2 - 9*Fi3);
end
```

Matlab Driver File

```
%% ex_5_6_1
clc
fun = @(t,y) y-t.^2+1;
y0 = 0.5;
tspan = [0,2];
N = 10;

%% Initial Values
h = (tspan(2) - tspan(1))/N;
exactY = @(t) (t+1).^2-0.5*exp(t);
t1 = tspan(1) + h; y1 = exactY(t1);
t2 = tspan(1) + 2*h; y2 = exactY(t2);
t3 = tspan(1) + 3*h; y3 = exactY(t3);
[t2,Y2] = adamBash2(fun,tspan,y0,y1,N);
[t3,Y3] = adamBash3(fun,tspan,y0,y1,y2,N);
[t4,Y4] = adamBash4(fun,tspan,y0,y1,y2,y3,N);

%% Display Solution
Y = exactY(t2);
disp↙
('-------------------------------------------------------------------------------');
disp('t_i     y(t_i)     AB2        Error      AB3        Error      AB4↙
Error')
disp↙
('-------------------------------------------------------------------------------')
formatSpec = '%.2f   % .5f   % .5f   % .5f   % .5f   % .5f   % .5f   % .5f\n';
fprintf(formatSpec,[t2';Y';Y2';abs(Y'-Y2');Y3';abs(Y'-Y3');Y4';abs(Y'-Y4')])
```

Matlab Outputs

```
------------------------------------------------------------------------------
t_i     y(t_i)     AD2        Error      AD3        Error      AD4        Error
------------------------------------------------------------------------------
0.00    0.50000    0.50000    0.00000    0.50000    0.00000    0.50000    0.00000
0.20    0.82930    0.82930    0.00000    0.82930    0.00000    0.82930    0.00000
0.40    1.21409    1.21609    0.00200    1.21409    0.00000    1.21409    0.00000
0.60    1.64894    1.65398    0.00504    1.64934    0.00040    1.64894    0.00000
0.80    2.12723    2.13657    0.00934    2.12827    0.00104    2.12731    0.00008
1.00    2.64086    2.65614    0.01529    2.64280    0.00194    2.64108    0.00022
1.20    3.17994    3.20333    0.02339    3.18311    0.00316    3.18035    0.00041
1.40    3.73240    3.76672    0.03432    3.73724    0.00484    3.73306    0.00066
1.60    4.28348    4.33240    0.04891    4.29059    0.00710    4.28449    0.00101
1.80    4.81518    4.88344    0.06827    4.82531    0.01013    4.81666    0.00148
2.00    5.30547    5.39924    0.09377    5.31962    0.01415    5.30758    0.00211
>>
```

Remarks on Multistep Methods

- To use an m-step multistep, we need to know $y_1, y_2, \cdots, y_{m-1}$. We can use a Runge-Kutta method of the same order to obtain these values.

- The Adam-Moulton methods are implicit methods, they usually involves solving an algebraic equation for y_{i+1}. For example

  $$y' = f(t, y) = e^y,$$

  the two-step Adams-Moulton method is

  $$y_{i+1} = y_i + \frac{h}{5}(5e^{y_{i+1}} + 8e^{y_i} - e^{y_{i-1}}).$$

- This is an equation for y_{i+1}, and the analytical solution is impossible to find. Therefore, we need to use Newton's method or Secant method.

Predictor-Corrector Methods

- In practice, the implicit Adam-Moulton methods are rarely used by themselves. They usually combined with explicit Adam-Bashforth methods to enhance the precision.
- The combination of an explicit method to predict and an implicit to improve the prediction is called a **predictor-corrector method**.

Fourth-Order Predictor-Corrector Method

- **Step 1** Use the fourth-order Runge Kutta method to calculate

$$y_1, \quad y_2, \quad \text{and} \quad y_3$$

- **Step 2** Use the explicit Adam Bashforth method to calculate a "predictor":

$$y_{4p} = y_3 + \frac{h}{24}\Big(55f(t_3, y_3) - 59f(t_2, y_2) + 37f(t_1, y_1) - 9f(t_0, y_0)\Big).$$

- **Step 3** Improve the result by the implicit Adam Moulton method to calculate a "corrector":

$$y_4 = y_4 + \frac{h}{24}(9f(t_4, y_{4p}) + 19f(t_3, y_3) - 5f(t_2, y_2) + f(t_1, y_1)).$$

Same process is used to find y_{5p} and y_5, and so on, until we obtain an approximation to y_N.

Example 17.

Apply the Adams fourth-order predictor-corrector method with $h = 0.2$ and starting values from the Runge-Kutta fourth order method to the initial-value problem

$$y' = y - t^2 + 1, \quad 0 \le t \le 2, \quad y(0) = 0.5.$$

Solution

Write a Matlab code to solve this problem. (**Homework**)

5.6 Higher-Order Equations and System of Equations

- In this section, we introduce numerical methods for higher-order equations.
- Many higher-order equations can be written as a system of first-order equations, so we first introduce numerical solutions of a system of first-order differential equations.
- For example, the following is an initial-value problem of a system of two 1st-order equations.

$$\begin{cases} x'(t) = x + 4y - e^t, & x(0) = 4 \\ y'(t) = x + y + 2e^t, & y(0) = \dfrac{5}{4} \end{cases}$$

- The true solution to this IVP is

$$\begin{cases} x(t) = 4e^{3t} + 2e^{-t} - 2e^t, \\ y(t) = 2e^{3t} - e^{-t} + \dfrac{1}{4}e^t. \end{cases}$$

- In general, a system of m differential equations has the form

$$\begin{cases} y_1'(t) = f_1(t, y_1, y_2, \cdots, y_m), \\ y_2'(t) = f_2(t, y_1, y_2, \cdots, y_m), \\ \qquad \vdots \\ y_m'(t) = f_m(t, y_1, y_2, \cdots, y_m), \end{cases} \tag{5.14}$$

  for $a \le t \le b$. Here, f_1, f_2, $\cdots$, f_m are given functions.
- The initial conditions of this system are

$$\begin{cases} y_1(a) = \alpha_1, \\ y_2(a) = \alpha_2, \\ \qquad \vdots \\ y_m(a) = \alpha_m, \end{cases} \tag{5.15}$$

  where α_1, α_2, $\cdots$, α_m are given constants.
- The objective is to find m functions y_1, y_2, $\cdots$, y_m that satisfy each of the equations in (5.14) and all the initial conditions (5.15).

- We introduce the **Vector Notation** for this differential-equation system.
- Let Y_0 be a vector in $\mathbb{R}^m$ and $Y : \mathbb{R} \to \mathbb{R}^m$ be a vector-valued function defined by

$$Y(t) = \begin{bmatrix} y_1(t) \\ y_2(t) \\ \vdots \\ y_m(t) \end{bmatrix} \qquad Y_0 = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_m \end{bmatrix}$$

- Let $F : \mathbb{R}^{m+1} \to \mathbb{R}^m$ such that f_i is the i-th row of F.

$$F(t, y_1, y_2, \cdots, y_m) = \begin{bmatrix} f_1(t, y_1, y_2, \cdots, y_m) \\ f_2(t, y_1, y_2, \cdots, y_m) \\ \vdots \\ f_m(t, y_1, y_2, \cdots, y_m) \end{bmatrix}$$

- Then, the IVP (5.14)-(5.15) is to find a vector-valued function $Y(t)$ such that

$$Y'(t) = F(t, Y), \quad Y(0) = Y_0.$$

- We partition the interval $[a, b]$ into N equal subintervals.
- The step size $h = \frac{b-a}{N}$. The nodes are $t_i = a + ih$, for $i = 0, 1, \cdots, N$.

Euler's Method for ODE System

Given the initial vector Y_0,

$$Y_{i+1} = Y_i + hF(t_i, Y_i), \quad i = 0, 1, \cdots, N-1.$$

Fourth-Order Runge-Kutta Method for ODE System

Given the initial vector Y_0,

$$Y_{i+1} = Y_i + \frac{1}{6}(K_1 + 2K_2 + 2K_3 + K_4), \quad i = 0, 1, \cdots, N-1.$$

where

$$\begin{aligned} K_1 &= hF\left(t_i, Y_i\right), \\ K_2 &= hF\left(t_i + \frac{1}{2}h, Y_i + \frac{1}{2}K_1\right), \\ K_3 &= hF\left(t_i + \frac{1}{2}h, Y_i + \frac{1}{2}K_2\right), \\ K_4 &= hF\left(t_i + h, Y_i + K_3\right). \end{aligned}$$

Example 18.

Use the Runge-Kutta method with $N = 10$ for solve the initial value problem of the differential equation system

$$\begin{cases} x'(t) = x + 4y - e^t, \\ y'(t) = x + y + 2e^t. \end{cases} \quad 0 \le t \le 1$$

subject to the initial conditions

$$x(0) = 4, \qquad y(0) = \frac{5}{4}.$$

Solution (1/2)

- Write the system in vector form:

$$Y(t) = \begin{bmatrix} x(t) \\ y(t) \end{bmatrix}, \quad F(t, Y) = F(t, x, y) = \begin{bmatrix} x + 4y - e^t \\ x + y + 2e^t \end{bmatrix}$$

Solution (2/2)

- Then the initial condition is

$$Y_0 = \begin{bmatrix} 4 \\ 5/4 \end{bmatrix}.$$

- The step size $h = 1/10$. Then

$$K_1 = hF(t_0, Y_0) = \frac{1}{10}\begin{bmatrix} 4+5-e^0 \\ 4+5+2e^0 \end{bmatrix} = \begin{bmatrix} 4/5 \\ 1 \end{bmatrix}$$

$$K_2 = hF\left(t_0 + \frac{1}{2}h, Y_0 + \frac{1}{2}K_1\right) = \ldots$$

$$K_3 = hF\left(t_0 + \frac{1}{2}h, Y_0 + \frac{1}{2}K_2\right) = \ldots$$

$$K_4 = hF(t_0 + h, Y_0 + K_3) = \ldots$$

- We will finish this problem by MATLAB programing.

Euler's Method

```
function [t,Y] = eulsys(Fun,a,b,Y0,N)

h = (b-a)/N;
m = length(Y0); % number of equations
t = zeros(1,N+1); % initialize t
Y = zeros(m,N+1); % initialize Y
t(1) = a;
Y(:,1) = Y0; % the first column of Y is Y0

for i = 1:N
    t(i+1) = t(i) + h;
    Y(:,i+1) = Y(:,i) + h*Fun(t(i),Y(:,i));
end
```

Runge-Kutta Method

```
function [t,Y] = rk4sys(Fun,a,b,Y0,N)

h = (b-a)/N;
m = length(Y0); % number of equations
t = zeros(1,N+1); % initialize t
Y = zeros(m,N+1); % initialize Y
t(1) = a;
Y(:,1) = Y0; % the first row of Y is Y0

for i = 1:N
    t(i+1) = t(i) + h;
    K1 = h*Fun(t(i),Y(:,i));
    K2 = h*Fun(t(i)+h/2,Y(:,i)+K1/2);
    K3 = h*Fun(t(i)+h/2,Y(:,i)+K2/2);
    K4 = h*Fun(t(i)+h,Y(:,i)+K3);
    Y(:,i+1) = Y(:,i) + 1/6*(K1 + 2*K2 + 2*K3 + K4);
end
```

MATLAB Driver File

```
%% ex_5_9_0
clc
f1 = @(t,Y) Y(1)+4*Y(2)-exp(t);
f2 = @(t,Y) Y(1)+Y(2)+2*exp(t);
Fun = @(t,Y) [f1(t,Y); f2(t,Y)];
Y0 = [4;5/4];
a = 0;
b = 1;
N = 10;
[t1,Y1] = eulsys(Fun,a,b,Y0,N);
[t2,Y2] = rk4sys(Fun,a,b,Y0,N);

%% Display Solution
exactY = @(t) [4*exp(3*t)+2*exp(-t)-2*exp(t); 2*exp(3*t)-exp(-t)+(1/4)*exp(t)];
trueY = exactY(t1);
E1 = abs(Y1-trueY);
E2 = abs(Y2-trueY);

disp('--------- Euler --------');
disp('t_i      y1(t_i)      y1        Error      y2(t_i)    y2         Error')
disp('-------------------------------------------------------------------------')
formatSpec = '%.2f  % 10.5f % 10.5f % 10.5f  % 10.5f % 10.5f % 10.5f\n';
fprintf(formatSpec,[t1; trueY(1,:);Y1(1,:);E1(1,:); trueY(2,:);Y1(2,:);E1(2,:)])

disp(' ')
disp('--------- RK4 ---------');
disp('t_i      y1(t_i)      y1        Error      y2(t_i)    y2         Error')
disp('-------------------------------------------------------------------------')
formatSpec = '%.2f  % 10.5f % 10.5f % 10.5f  % 10.5f % 10.5f % 10.5f\n';
fprintf(formatSpec,[t2;trueY(1,:);Y2(1,:);E2(1,:); trueY(2,:);Y2(2,:);E2(2,:)])
```

MATLAB Result

```
--------- Euler --------
t_i      y1(t_i)      y1          Error      y2(t_i)    y2          Error
-------------------------------------------------------------------------
0.00     4.00000    4.00000     0.00000     1.25000    1.25000    0.00000
0.10     4.99877    4.80000     0.19877     2.07117    1.97500    0.09617
0.20     6.48313    5.95948     0.52365     3.13086    2.87353    0.25732
0.30     8.62033    7.58270     1.03763     4.51585    4.00112    0.51474
0.40    11.63746    9.80644     1.83102     6.34287    5.42947    0.91340
0.50    15.84238   12.80968     3.03269     8.76903    7.25143    1.51760
0.60    21.65198   16.82635     4.82562    12.00601    9.58728    2.41873
0.70    29.63034   22.16169     7.46866    16.33919   12.59307    3.74612
0.80    40.54028   29.21371    11.32657    22.15341   16.47129    5.68211
0.90    55.41286   38.50104    16.91182    29.96779   21.48490    8.48289
1.00    75.64134   50.69915    24.94220    40.48276   27.97542   12.50735

--------- RK4 ---------
t_i      y1(t_i)      y1          Error      y2(t_i)    y2          Error
-------------------------------------------------------------------------
0.00     4.00000    4.00000     0.00000     1.25000    1.25000    0.00000
0.10     4.99877    4.99868     0.00008     2.07117    2.07113    0.00004
0.20     6.48313    6.48291     0.00023     3.13086    3.13074    0.00011
0.30     8.62033    8.61987     0.00046     4.51585    4.51562    0.00023
0.40    11.63746   11.63663     0.00082     6.34287    6.34246    0.00041
0.50    15.84238   15.84098     0.00139     8.76903    8.76833    0.00070
0.60    21.65198   21.64972     0.00226    12.00601   12.00488    0.00113
0.70    29.63034   29.62678     0.00356    16.33919   16.33741    0.00178
0.80    40.54028   40.53478     0.00550    22.15341   22.15066    0.00275
0.90    55.41286   55.40450     0.00836    29.96779   29.96362    0.00418
1.00    75.64134   75.62880     0.01255    40.48276   40.47649    0.00627
```

Higher-Order Equations

- Consider the initial-value problem of m-th order differential equation

$$y^{(m)} = f(t, y, y', y'', \cdots, y^{(m-1)}), \quad a \le t \le b. \tag{5.16}$$

$$y(a) = \alpha_1, \;\; y'(a) = \alpha_2, \;\; \cdots \;\; , y^{(m-1)}(a) = \alpha_m. \tag{5.17}$$

- We substitute $y_1 = y, \;\; y_2 = y', \;\; y_3 = y'' \cdots , \quad y_m = y^{(m-1)}$, then

$$\begin{cases} y_1'(t) = y_2, \\ y_2'(t) = y_3, \\ \qquad \vdots \\ y_{m-1}'(t) = y_m, \\ y_m'(t) = f(t, y_1, y_2, \cdots, y_m). \end{cases}$$

with the initial conditions

$$y_1(a) = \alpha_1, \;\; y_2(a) = \alpha_2, \;\; \cdots \;\; , y_m(a) = \alpha_m.$$

Example 19.

Convert the initial-value problem

$$y''(t) + 4y'(t) + 3y(t) = 0; \quad y(0) = 1.5, \quad y'(0) = -2.5$$

into a system of first-order equations with initial values. Use Euler's method and Runge-Kutta method for approximate its solution $y(t) = e^{-t} + \frac{1}{2}e^{-3t}$.

Solution

- The equation can be written as

$$y'' = -3y(t) - 4y'(t).$$

- Let $y_1 = y$, $y_2 = y'$. Then

$$\begin{cases} y_1'(t) = y_2, & y_1(0) = 1.5, \\ y_2'(t) = -3y_1 - 4y_2, & y_2(0) = -2.5. \end{cases}$$

- Finish this problem by yourself.

Example 20 (Higher-Order Equation).

Convert the initial-value problem

$$\begin{cases} \sin(t)y''' + \cos(ty) + \sin(t^2 + y'') + (y')^3 = \log(t), & 2 \le t \le 3, \\ y(2) = 7, \ \ y'(2) = 3, \ \ y''(2) = -4 \end{cases}$$

into a system of first-order equations with initial values.

Solution (1/4)

- The equation can be written as

$$y''' = \frac{1}{\sin(t)}\Big[\log(t) - \cos(ty) - (y')^3 - \sin(t^2 + y'')\Big]$$

- Let $y_1 = y$, $y_2 = y'$, $y_3 = y''$. Then

$$\begin{cases} y_1'(t) = y_2, \\ y_2'(t) = y_3, \\ y_3'(t) = [\log(t) - y_2^3 - \sin(t^2 + y_3) - \cos(ty_1)]/\sin(t) \end{cases}$$

- The initial condition is $y_1(2) = 7, \ \ y_2(2) = 3, \ \ y_3(2) = -4$.

Solution (2/4)

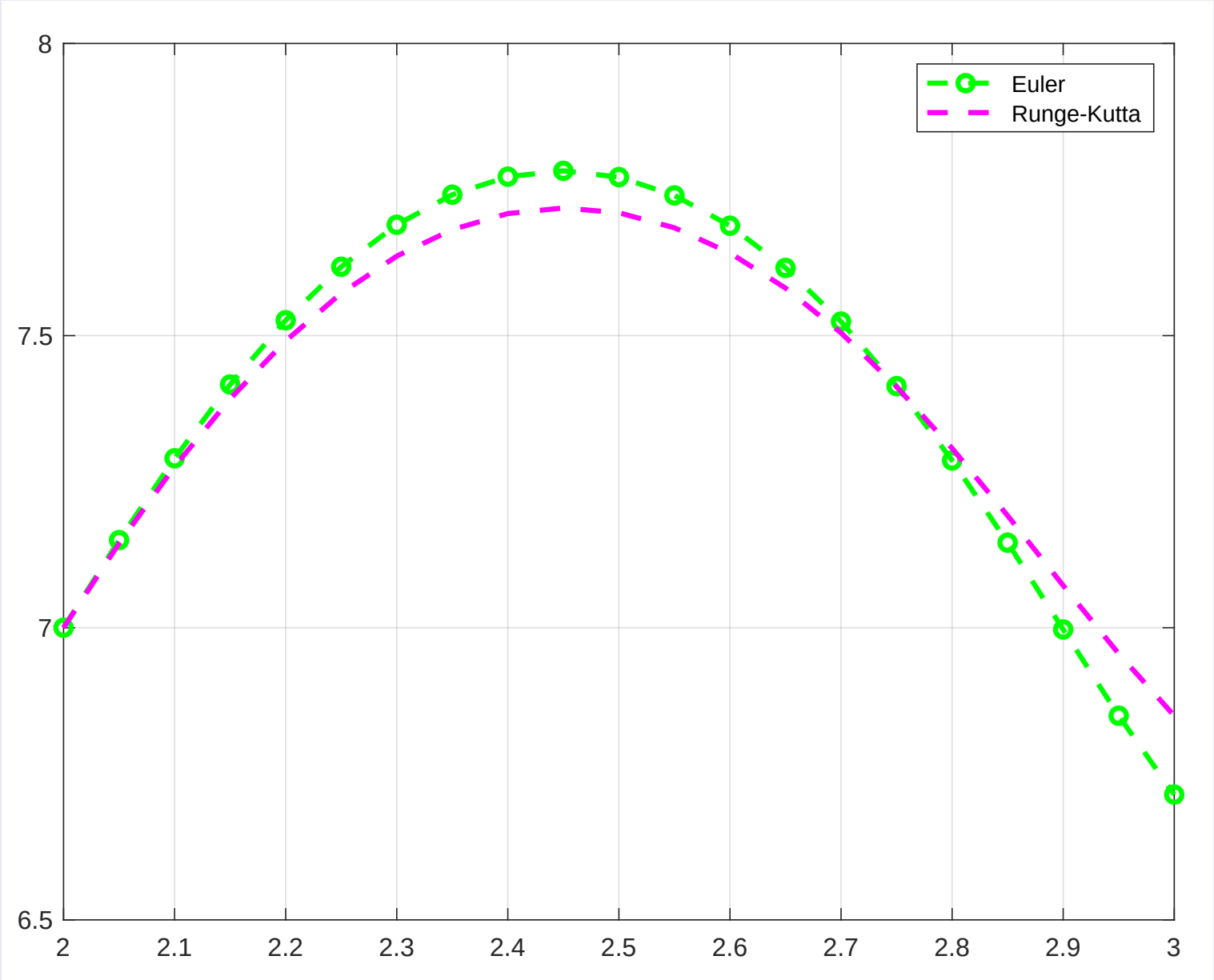
```
%% ex_5_8_3
clc
clear
f1 = @(t,Y) Y(2);
f2 = @(t,Y) Y(3);
f3 = @(t,Y) (log(t) - Y(2).^3 - sin(t.^2+Y(3)) -cos(t.*Y(1))).\sin(t);
Fun = @(t,Y) [f1(t,Y); f2(t,Y); f3(t,Y)];
Y0 = [7;3;-4];
a = 2;
b = 3;
N = 20;
[t1,Y1] = eulsys(Fun,a,b,Y0,N);
[t2,Y2] = rk4sys(Fun,a,b,Y0,N);

%% Display Solution
disp('------------------Euler-------------------------- RK4↙
---------------');
disp('t_i     y1         y2          y3          y1         y2         y3')
disp('-------------------------------------------------------------------------------')
formatSpec = '%.2f   % 7.5f   % 7.5f   % 9.5f    % 7.5f   % 7.5f   % 7.5f\n';
fprintf(formatSpec,[t1; Y1(1,:);Y1(2,:);Y1(3,:); Y2(1,:);Y2(2,:);Y2(3,:)])

%% Plot Solution
figure(1)
yeu = Y1(1,:);
yrk = Y2(1,:);
clf
plot(t1,yeu,'go--','linewidth',2)
hold on
plot(t2,yrk,'m--','linewidth',2)
hold off
legend('Euler','Runge-Kutta','Location','northeast')
axis([2,3,6.5,8])
grid on
```

Solution (3/4)

	Euler			RK4		
t_i	y1	y2	y3	y1	y2	y3
2.00	7.00000	3.00000	−4.00000	7.00000	3.00000	−4.00000
2.05	7.15000	2.80000	−5.45407	7.14443	2.76681	−5.26489
2.10	7.29000	2.52730	−6.56903	7.27578	2.47965	−6.16401
2.15	7.41636	2.19885	−7.35952	7.39178	2.15572	−6.74543
2.20	7.52631	1.83087	−7.86735	7.49098	1.80935	−7.07332
2.25	7.61785	1.43750	−8.15027	7.57253	1.45145	−7.21772
2.30	7.68973	1.02999	−8.27671	7.63605	1.08952	−7.24466
2.35	7.74123	0.61615	−8.31036	7.68149	0.72803	−7.20834
2.40	7.77203	0.20064	−8.29803	7.70890	0.36909	−7.14750
2.45	7.78206	−0.21427	−8.26440	7.71845	0.01330	−7.08433
2.50	7.77135	−0.62749	−8.20954	7.71028	−0.33940	−7.02376
2.55	7.73998	−1.03796	−8.10497	7.68456	−0.68891	−6.95232
2.60	7.68808	−1.44321	−7.88778	7.64147	−1.03390	−6.83581
2.65	7.61592	−1.83760	−7.45647	7.58130	−1.37078	−6.61531
2.70	7.52404	−2.21042	−6.67640	7.50465	−1.69219	−6.19764
2.75	7.41352	−2.54425	−5.37636	7.41257	−1.98467	−5.42215
2.80	7.28630	−2.81306	−3.19756	7.30707	−2.22332	−3.98235
2.85	7.14565	−2.97294	0.42483	7.19181	−2.36813	−1.71488
2.90	6.99700	−2.95170	5.03369	7.07234	−2.38924	0.96523
2.95	6.84942	−2.70001	10.44307	6.95547	−2.25694	4.34266
3.00	6.71442	−2.17786	15.76294	6.84937	−1.96263	7.44563

Remark: Only y_1 approximates the true solution $y(t)$.

Solution (4/4)



Example 21 (System of Higher-Order Equation).

Solve the following system of differential equations on $(1, 2)$

$$\begin{cases} x'' + te^y + y' = x' - x, & x(1) = 1, \quad x'(1) = 2 \\ yy'' - \cos(xy') + \sin(tx'y) = x, & y(1) = 3, \quad y'(1) = 4 \end{cases}$$

Solution (1/3)

Both $x(t)$ and $y(t)$ are unknown functions.
Let $z_1 = x$, $z_2 = x'$, $z_3 = y$, $z_4 = y'$. Then

$$\begin{cases} z_1'(t) = z_2, \\ z_2'(t) = z_2 - z_1 - z_4 - te^{z_3}, \\ z_3'(t) = z_4, \\ z_4'(t) = [z_1 - \sin(tz_2z_3) + \cos(z_1z_4)]/z_3. \end{cases}$$

The initial condition is $z_1(1) = 1, \quad z_2(1) = 2, \quad z_3(1) = 3, \quad z_4(1) = 4.$

Solution (2/3)

```
%% ex_5_8_4
clc
clear
f1 = @(t,Z) Z(2);
f2 = @(t,Z) Z(2) - Z(1) - Z(4) - t.*exp(Z(3));
f3 = @(t,Z) Z(4);
f4 = @(t,Z) (Z(1) - sin(t.*Z(2).*Z(3)) + cos(Z(1).*Z(4)))./Z(3);

Fun = @(t,Z) [f1(t,Z); f2(t,Z); f3(t,Z); f4(t,Z)];
Z0 = [1;2;3;4];
a = 1;
b = 2;
N = 20;
[t,Zeu] = eulsys(Fun,a,b,Z0,N);
[t,Zrk] = rk4sys(Fun,a,b,Z0,N);

%% Display Solution
disp(['N = ', int2str(N)])
disp('t_i       Eul x        RK4 x       Eul y         RK4 y   ')
disp('-------------------------------------------------------------')
formatSpec = '%.2f   % 10.5f   % 10.5f  % 10.5f   % 10.5f   \n';
fprintf(formatSpec,[t; Zeu(1,:); Zrk(1,:); Zeu(3,:); Zrk(3,:)])

%% Plot Solution
xeu = Zeu(1,:); yeu = Zeu(3,:);
xrk = Zrk(1,:); yrk = Zeu(3,:);

figure(1); clf
plot(t,xeu,'go--','linewidth',2)
hold on
plot(t,xrk,'m--','linewidth',2)
hold off
legend('Euler ','Runge-Kutta','Location','southwest')
axis([1,2,-120,20])
grid on

figure(2); clf
plot(t,yeu,'go--','linewidth',2)
hold on
plot(t,yrk,'m--','linewidth',2)
hold off
legend('Euler ','Runge-Kutta','Location','northwest')
axis([1,2,3,7])
grid on
```

Solution (3/3)

The solution curves for $x(t)$ (left) and $y(t)$ (right) are