Solving Ordinary Differential Equations

Sanzheng Qiao

Department of Computing and Software
McMaster University

March, 2014
Outline

1 Initial Value Problem
   - Euler’s Method
   - Runge-Kutta Methods
   - Multistep Methods
   - Implicit Methods
   - Hybrid Method

2 Software Packages
Outline

1 Initial Value Problem
   - Euler’s Method
   - Runge-Kutta Methods
   - Multistep Methods
   - Implicit Methods
   - Hybrid Method

2 Software Packages
Problem setting

Initial Value Problem (first order)

find $y(t)$ such that

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

given initial value $y(t_0)$. Usually we assume $t_0 = 0$
Problem setting

Initial Value Problem (first order)

find $y(t)$ such that

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

given initial value $y(t_0)$. Usually we assume $t_0 = 0$

Generalization 1: system of first order ODEs: $y$ is a vector and $f$ a vector-valued function.

Example

$$\begin{cases} y_1' = f_1(y_1, y_2, t) \\ y_2' = f_2(y_1, y_2, t) \end{cases}$$

or in vector notations:

$$\mathbf{y}' = \mathbf{f}(\mathbf{y}, t)$$
Generalization 2: high order equation

\[ u'' = g(u, u', t). \]

Let

\[
\begin{align*}
y_1 & = u \\
y_2 & = u'
\end{align*}
\]

and transform the above into the following system of first order ODEs:

\[
\begin{cases}
y_1' = y_2 \\
y_2' = g(y_1, y_2, t)
\end{cases}
\]

Initial conditions: \( u(0) \) and \( u'(0) \)
A differential equation has a family of solutions, each corresponds to an initial value.

\[ y' = -y, \text{ solution family } y(t) = Ce^{-t}, \; y(0) = C. \]
Euler’s method

We consider the initial value problem:

\[ y' = f(y, t), \quad y(t_0) = y_0 \]

Numerical solution: find approximations

\[ y_n \approx y(t_n), \quad \text{for} \quad n = 1, 2, \ldots \]

Note: \( y_0 = y(t_0) \) (initial value)
Euler’s method

We consider the initial value problem:

\[ y' = f(y, t), \quad y(t_0) = y_0 \]

Numerical solution: find approximations

\[ y_n \approx y(t_n), \quad \text{for} \quad n = 1, 2, ... \]

Note: \( y_0 = y(t_0) \) (initial value)

A \( k \)-step method: Compute \( y_{n+1} \) using

\[ y_{n-k+1}, \ldots, y_{n-1}, y_n. \]
Euler’s method (cont.)

A single-step method: Euler’s method.

\[ f(y_0, t_0) = y'(t_0) \approx \frac{y(t_1) - y(t_0)}{h_0}, \]

where \( h_0 = t_1 - t_0 \). The first step:

\[ y_1 = y_0 + h_0 f(y_0, t_0) \]
Euler’s method (cont.)

A single-step method: Euler’s method.

\[ f(y_0, t_0) = y'(t_0) \approx \frac{y(t_1) - y(t_0)}{h_0}, \]

where \( h_0 = t_1 - t_0 \). The first step:

\[ y_1 = y_0 + h_0 f(y_0, t_0) \]

Euler’s method

\[ y_{n+1} = y_n + h_n f(y_n, t_n) \]

Produces: \( y_0 = y(t_0), y_1 \approx y(t_1), y_2 \approx y(t_2), \ldots \)
Example

\[ y' = -y, \quad y(0) = 1.0. \quad (\text{Solution } y = e^{-t}) \]
Example

\[ y' = -y, \ y(0) = 1.0. \ (\text{Solution} \ y = e^{-t}) \]

\[ h = 0.4 \]

Step 1:

\[ y_1 = y_0 - h y_0 = 1.0 - 0.4 \times 1.0 = 0.6 \]

\( \approx y(0.4) = e^{-0.4} \approx 0.6703 \)
Example

\[ u_1(t) = 0.6e^{-t} + 0.4 \approx 0.8951e^{-t} \text{ in the solution family.} \]

\[ u_1'(t) = -u_1(t), \quad u_1(0) \approx 0.8951 \quad (u_1(0.4) = 0.6) \]
Step 2:
\[ y_2 = y_1 - hy_1 = 0.6 - 0.4 \times 0.6 = 0.36 \]
Step 2:
\[ y_2 = y_1 - hy_1 = 0.6 - 0.4 \times 0.6 = 0.36 \]
Example

\[ u_2(t) = 0.36e^{-t+0.8} \approx 0.8012e^{-t} \] in the solution family.

\[ u'_2 = -u_2, \; u_2(0) \approx 0.8012 \; (u_2(0.8) = 0.36) \]
Euler’s method

In general

\[ u'_n(t) = f(u_n(t), t), \text{ in the solution family} \]
\[ u_n(t_n) = y_n, \text{ passing } (t_n, y_n) \]
\[ u_n(t_{n+1}) \approx u_n(t_n) + h_n u'_n(t_n) = y_n + h_n f(u_n(t_n), t_n) = y_n + h_n f(y_n, t_n) = y_{n+1} \]
Euler’s method

Starting with $t_0$ and $y_0 = y(t_0)$, as we proceed, we jump from one solution in the family to another.
Two sources of errors: discretization error and roundoff error.

- *Discretization error*: caused by the method used, independent of the computer used and the program implementing the method.
Two sources of errors: discretization error and roundoff error.

- **Discretization error**: caused by the method used, independent of the computer used and the program implementing the method.

- Two types of discretization error:
  - Global error: \( e_n = y_n - y(t_n) \)
  - Local error: the error in one step
Consider $t_n$ as the starting point and the approximation $y_n$ at $t_n$ as the initial value, if $u_n(t)$ is the solution of

$$u'_n = f(u_n, t), \quad u_n(t_n) = y_n$$

then the local error is

$$d_n = y_{n+1} - u_n(t_{n+1})$$
Local error $d_0 = y_1 - y(t_1) = 0.6 - e^{-0.4} \approx -0.0703$.

Global error $e_1$ is the same as the local error $d_0$. 
Example

Step 2

Local error $d_1 = y_2 - u_1(t_2) = 0.36 - u_1(0.8) \approx -0.0422$.
Global error $e_2 = y_2 - y(t_2) = 0.36 - e^{-0.8} \approx -0.0893$. 
Example

\[ d_0 = y_1 - y(t_1) = 0.6 - e^{-0.4} \approx -0.0703 \]

\[ d_1 = y_2 - u_1(t_2) = 0.36 - u_1(0.8) \approx -0.0422 \]

\[ e_2 = y_2 - y(t_2) = 0.36 - e^{-0.8} \approx -0.0893 \]
Stability

Relation between global error $e_n$ and local error $d_n$

If the differential equation is unstable,

$$|e_N| > \sum_{n=0}^{N-1} |d_n|$$

If the differential equation is stable,

$$|e_N| \leq \sum_{n=0}^{N-1} |d_n|$$

In this case, $\sum_{n=0}^{N-1} |d_n|$ is an upper bound for the global error $|e_N|$. 
In the previous example:

Local errors $|d_0| = 0.0703$ and $|d_1| = 0.0422$

Global error $|e_2| = 0.0893$

$|e_2| < |d_0| + |d_1|$
In the previous example:

Local errors $|d_0| = 0.0703$ and $|d_1| = 0.0422$

Global error $|e_2| = 0.0893$

$|e_2| < |d_0| + |d_1|$

More generally,

$y' = \alpha y$, solution family $y = Ce^{\alpha t}$.

Stable when $\alpha < 0$. 
Accuracy

A measurement for the accuracy of a method
An order \( p \) method:

\[ |d_n| \leq C h_n^{p+1} \quad \text{(or } O(h_n^{p+1})) \]

\( C \): independent of \( n \) and \( h_n \).
Example: Euler’s method $y_{n+1} = y_n + h_n f(y_n, t_n)$

Local solution $u_n(t)$

$$u'_n(t) = f(u_n(t), t), \quad u_n(t_n) = y_n$$

Taylor expansion at $t_n$:

$$u_n(t) = u_n(t_n) + (t - t_n)u'(t_n) + O((t - t_n)^2)$$

Since $y_n = u_n(t_n)$ and $u'(t_n) = f(y_n, t_n)$, we get

$$u_n(t_{n+1}) = y_n + h_n f(y_n, t_n) + O(h_n^2)$$

Local error

$$d_n = y_{n+1} - u_n(t_{n+1}) = O(h_n^2)$$

Euler’s method is a first order method ($p = 1$)
Consider the interval \([t_0, t_N]\) and partition \(t_0, t_1, \ldots, t_N\). Roughly, the global error

\[
|e_N| \approx \sum_{n=0}^{N-1} |d_n| \approx N \cdot O(h^{p+1}) \approx (t_N - t_0) \cdot O(h^p)
\]

at the final point \(t_N\) is roughly \(O(h^p)\) for a method of order \(p\).
Consider the interval $[t_0, t_N]$ and partition $t_0, t_1, ..., t_N$. Roughly, the global error

$$|e_N| \approx \sum_{n=0}^{N-1} |d_n| \approx N \cdot O(h^{p+1}) \approx (t_N - t_0) \cdot O(h^p)$$

at the final point $t_N$ is roughly $O(h^p)$ for a method of order $p$.

For a $p$th order method, if the subintervals $h_n$ are cut in half, then the average local error is reduced by a factor of $2^{p+1}$, the global error is reduced by a factor of $2^p$. (But double the number of steps, i.e., more work.)
Roundoff Error

Each step of the Euler’s method

\[ y_{n+1} = y_n + h_n f(y_n, t_n) + \epsilon \quad |\epsilon| = O(u). \]

Total rounding error: \( N\epsilon = b \epsilon / h \) \( (b = t_N - t_0, \text{fixed step size } h) \)

\[ \text{total error} \approx b \left( Ch + \frac{\epsilon}{h} \right) \]
Roundoff Error

Each step of the Euler’s method

\[ y_{n+1} = y_n + h_n f(y_n, t_n) + \epsilon \quad |\epsilon| = O(u). \]

Total rounding error: \( N\epsilon = b \epsilon / h \) \( (b = t_N - t_0, \text{ fixed step size } h) \)

\[ \text{total error} \approx b \left( Ch + \frac{\epsilon}{h} \right) \]

Remarks
- If \( h \) is too small, the roundoff error is large
- If \( h \) is too large, the discretization error is large
Roundoff Error

Each step of the Euler’s method

\[ y_{n+1} = y_n + hn f(y_n, t_n) + \epsilon \quad |\epsilon| = O(u). \]

Total rounding error: \( N\epsilon = b \epsilon / h \) \((b = t_N - t_0, \text{ fixed step size } h)\)

\[
\text{total error} \approx b \left( Ch + \frac{\epsilon}{h} \right)
\]

Remarks

- If \( h \) is too small, the roundoff error is large
- If \( h \) is too large, the discretization error is large

The total error is minimized by

\[ h_{\text{opt}} \approx \sqrt{\frac{u}{C}} \]

recalling that \( u \) is the unit of roundoff.
Runge-Kutta methods

Idea: Sample $f$ at several spots to achieve high order.
Cost: More function evaluations
Runge-Kutta methods

Idea: Sample $f$ at several spots to achieve high order.
Cost: More function evaluations

Example. A second-order Runge-Kutta method
Suppose

$$y_{n+1} = y_n + \gamma_1 k_0 + \gamma_2 k_1$$

where $\gamma_1$ and $\gamma_2$ to be determined and

$k_0 = h_n f(y_n, t_n)$

$k_1 = h_n f(y_n + \beta k_0, t_n + \alpha h_n)$

$\alpha$ and $\beta$ to be determined.
Runge-Kutta methods (cont.)

Taylor series (two variables $y$ and $t$):

\[ k_1 = h_n(f_n + \beta k_0 f'_y(y_n, t_n) f_n + \alpha h_n f'_t(y_n, t_n) + \cdots) \]

Thus

\[ y_{n+1} = y_n + (\gamma_1 + \gamma_2) h_n f_n + \gamma_2 \beta h^2_n f_n f'_y(y_n, t_n) + \gamma_2 \alpha h^2_n f'_t(y_n, t_n) + \cdots \]

The Taylor expansion of the true local solution:

\[ u_n(t_{n+1}) = u_n(t_n) + h_n u'_n(t_n) + \frac{h^2_n}{2} u''_n(t_n) + \cdots \]

\[ = y_n + h_n f_n + \frac{h^2_n}{2} (f'_y(y_n, t_n) f_n + f'_t(y_n, t_n)) + \cdots \]
Second order RK method

Comparing the two expressions, we set

\[
\begin{aligned}
\gamma_1 + \gamma_2 &= 1 \\
\gamma_2 \beta &= 1/2 \\
\gamma_2 \alpha &= 1/2
\end{aligned}
\]

Then the local error

\[
d_n = y_{n+1} - u_n(t_{n+1}) = O(h_n^3)
\]

The global error \(O(h^2)\)
Second order RK method

Let

\[ \gamma_1 = 1 - \frac{1}{2\alpha}, \quad \gamma_2 = \frac{1}{2\alpha}, \quad \beta = \alpha \]

Second-order RK method

\[ y_{n+1} = y_n + \left( 1 - \frac{1}{2\alpha} \right) k_0 + \frac{1}{2\alpha} k_1 \]

where

\[ k_0 = h_n f(y_n, t_n) \]
\[ k_1 = h_n f(y_n + \alpha k_0, t_n + \alpha h_n) \]
Second order RK method

Let

\[ \gamma_1 = 1 - \frac{1}{2\alpha}, \quad \gamma_2 = \frac{1}{2\alpha}, \quad \beta = \alpha \]

Second-order RK method

\[ y_{n+1} = y_n + \left( 1 - \frac{1}{2\alpha} \right) k_0 + \frac{1}{2\alpha} k_1 \]

where

\[ k_0 = h_n f(y_n, t_n) \]
\[ k_1 = h_n f(y_n + \alpha k_0, t_n + \alpha h_n) \]

When \( \alpha = 1/2 \), related to the rectangle rule
When \( \alpha = 1 \), related to the trapezoid rule
Classical fourth-order Runge-Kutta method

\[ y_{n+1} = y_n + \frac{1}{6}(k_0 + 2k_1 + 2k_2 + k_3) \]

where

\[ k_0 = hf(y_n, t_n) \]
\[ k_1 = hf \left( y_n + \frac{1}{2}k_0, t_n + \frac{h}{2} \right) \]
\[ k_2 = hf \left( y_n + \frac{1}{2}k_1, t_n + \frac{h}{2} \right) \]
\[ k_3 = hf( y_n + k_2, t_n + h) \]
Multistep Methods

Compute $y_{n+1}$ using $y_n, y_{n-1}, ...$ and $f_n, f_{n-1}, ...$ possibly $f_{n+1}$ ($f_i = f(y_i, t_i)$).

General linear $k$-step method:

$$y_{n+1} = \sum_{i=1}^{k} \alpha_i y_{n-i+1} + h \sum_{i=0}^{k} \beta_i f_{n-i+1}$$

- $\beta_0 = 0$ (no $f_{n+1}$), explicit method
- $\beta_0 \neq 0$, implicit method
Examples

Adams-Bashforth methods (explicit).

\[ y_{n+1} = y_n + \int_{t_n}^{t_{n+1}} p_{k-1}(t) \, dt \]

where \( p_{k-1}(t) \) is a polynomial of degree \( k - 1 \) which interpolates \( f(y, t) \) at \( (y_{n-j}, t_{n-j}), \) \( j = 0, ..., k - 1. \)
Adams-Bashforth methods (explicit).

\[ y_{n+1} = y_n + \int_{t_n}^{t_{n+1}} p_{k-1}(t) \, dt \]

where \( p_{k-1}(t) \) is a polynomial of degree \( k - 1 \) which interpolates \( f(y, t) \) at \( (y_{n-j}, t_{n-j}) \), \( j = 0, \ldots, k - 1 \).

For example.
\( p_0(t) = f_n \), Euler’s method
\( p_1(t) = f_{n-1} + \frac{f_n - f_{n-1}}{h_{n-1}}(t - t_{n-1}) \)
Adams-Bashforth family

\[ y_{n+1} = y_n + hf_n \]
local error \( \frac{h^2}{2} y^{(2)}(\eta) \), order 1
\[ y_{n+1} = y_n + \frac{h}{2} (3f_n - f_{n-1}) \]
local error \( \frac{5h^3}{12} y^{(3)}(\eta) \), order 2
\[ y_{n+1} = y_n + \frac{h}{12} (23f_n - 16f_{n-1} + 5f_{n-2}) \]
local error \( \frac{3h^4}{8} y^{(4)}(\eta) \), order 3
\[ y_{n+1} = y_n + \frac{h}{24} (55f_n - 59f_{n-1} + 37f_{n-2} - 9f_{n-3}) \]
local error \( \frac{251h^5}{720} y^{(5)}(\eta) \), order 4
Multistep methods (cont.)

The “start-up” issue in multistep methods:
How to get the $k - 1$ start values $f_j = f(y_j, t_j)$, $j = 1, ..., k - 1$?
The “start-up” issue in multistep methods:
How to get the $k - 1$ start values $f_j = f(y_j, t_j)$, $j = 1, ..., k - 1$?
Use a single step method to get start values, then switch to multistep method.
The “start-up” issue in multistep methods:
How to get the $k - 1$ start values $f_j = f(y_j, t_j), j = 1, \ldots, k - 1$?
Use a single step method to get start values, then switch to multistep method.

Note. Careful about accuracy consistency.
Example

The motion of two bodies under mutual gravitational attraction.

A coordination system:
origin: position of one body
\( x(t), y(t) \): position of the other body

Differential equations derived from Newton’s laws of motion:

\[
\begin{align*}
x''(t) &= -\frac{\alpha^2 x(t)}{r(t)} \\
y''(t) &= -\frac{\alpha^2 y(t)}{r(t)}
\end{align*}
\]

where \( r(t) = [x(t)^2 + y(t)^2]^{3/2} \) and \( \alpha \) is a constant involving the gravitational constant, the masses of the two bodies, and the units of measurement.
If the initial conditions are chosen as

\[ x(0) = 1 - e, \quad x'(0) = 0, \]
\[ y(0) = 0, \quad y'(0) = \alpha \left( \frac{1+e}{1-e} \right)^{1/2} \]

for some \( e \) with \( 0 \leq e < 1 \), then the solution is periodic with period \( 2\pi/\alpha \). The orbit is an ellipse with eccentricity \( e \) and with one focus at the origin.
If the initial conditions are chosen as

\[
x(0) = 1 - e, \quad x'(0) = 0, \\
y(0) = 0, \quad y'(0) = \alpha \left(\frac{1+e}{1-e}\right)^{1/2}
\]

for some \(e\) with \(0 \leq e < 1\), then the solution is periodic with period \(2\pi/\alpha\). The orbit is an ellipse with eccentricity \(e\) and with one focus at the origin.

To write the two second-order differential equations as four first-order differential equations, we introduce

\[
y_1 = x, \quad y_2 = y, \quad y_3 = x', \quad y_4 = y'
\]
We have a system of first-order equations

\[ s = \frac{(y_1^2 + y_2^2)^{3/2}}{\alpha^2}, \]

\[
\begin{bmatrix}
  y_1 \\
  y_2 \\
  y_3 \\
  y_4
\end{bmatrix}' = \begin{bmatrix}
  y_3 \\
  y_4 \\
  -\frac{y_1}{s} \\
  -\frac{y_2}{s}
\end{bmatrix} = \begin{bmatrix}
  0 & 0 & 1 & 0 \\
  0 & 0 & 0 & 1 \\
  -1/s & 0 & 0 & 0 \\
  0 & -1/s & 0 & 0
\end{bmatrix} \begin{bmatrix}
  y_1 \\
  y_2 \\
  y_3 \\
  y_4
\end{bmatrix}
\]

with the initial condition

\[
\begin{bmatrix}
  y_1(0) \\
  y_2(0) \\
  y_3(0) \\
  y_4(0)
\end{bmatrix} = \begin{bmatrix}
  1 - e \\
  0 \\
  0 \\
  \alpha \left(\frac{1+e}{1-e}\right)^{1/2}
\end{bmatrix}
\]
Example

The function defining the system of equations:

```matlab
function yp = orbit(y, t)

global a
global e

yp = zeros(size(y));
r = y(1)*y(1) + y(2)*y(2);
r = r*sqrt(r)/(a*a);
yp(1) = y(3);
yp(2) = y(4);
yp(3) = -y(1)/r;
yp(4) = -y(2)/r;

endfunction
```
A script file `TwoBody.m` (Octave)

```octave
global a
global e

a = input('a = ');
e = input('e = ');

# initial value
x0 = [1-e; 0.0; 0.0; a*sqrt((1+e)/(1-e))];
# time span
t = [0.0:0.1:(2*pi/a)];

# solve ode
[x, state, msg] = lsode('orbit', x0, t);
```

Matlab

```matlab
[t, x] = ode45('orbit', [0.0 2*pi/a], x0);
```
Example

Output $x$: matrix of four columns

$x(:, 1): x(t)$
$x(:, 2): y(t)$
$x(:, 3): x'(t)$
$x(:, 4): y'(t)$

plot($x(:, 1), x(:, 2)$)

$e = 1/4, \alpha = \pi / 4$
Implicit methods

Example

\[ y' = -20y, \ y(0) = 1 \text{ (Solution } e^{-20t}) \]

Euler’s method, \( h = 0.1 \)

\[ y_{n+1} = y_n - 20 \times 0.1 y_n = y_n - 2y_n \]
Implicit methods

Example

\[ y' = -20y, \quad y(0) = 1 \]  \hspace{1cm} \text{(Solution } y^{-20t})

Euler's method, \( h = 0.1 \)

\[ y_{n+1} = y_n - 20 \times 0.1 y_n = y_n - 2y_n \]
Backward Euler’s method

Example

\[ y' = -20y, \quad y(0) = 1 \] (Solution \( y^{-20t} \))

Backward Euler’s method, \( h = 0.1 \)

\[
y_{n+1} = y_n - 20 \times 0.1 y_{n+1} = y_n - 2y_{n+1}
\]
Backward Euler’s method

Taylor expansion of the solution $y(t)$ about $t = t_{n+1}$ (instead of $t = t_n$)

$$y(t_{n+1} + h) \approx y(t_{n+1}) + y'(t_{n+1})h$$
$$= y(t_{n+1}) + f(y(t_{n+1}), t_{n+1})h$$

and set $h = -h_n = (t_n - t_{n+1})$, then we get

$$y(t_n) \approx y(t_{n+1}) - h_n f(y(t_{n+1}), t_{n+1})$$
Backward Euler’s method (cont.)

\[ y(t_n) \approx y(t_{n+1}) - h_n f(y(t_{n+1}), t_{n+1}) \]

Substituting \( y_n \) for \( y(t_n) \) and \( y_{n+1} \) for \( y(t_{n+1}) \), we have

**Backward Euler’s method**

\[ y_{n+1} = y_n + h_n f(y_{n+1}, t_{n+1}) \]
Backward Euler’s method (cont.)

\[ y(t_n) \approx y(t_{n+1}) - h_n f(y(t_{n+1}), t_{n+1}) \]

Substituting \( y_n \) for \( y(t_n) \) and \( y_{n+1} \) for \( y(t_{n+1}) \), we have

**Backward Euler’s method**

\[ y_{n+1} = y_n + h_n f(y_{n+1}, t_{n+1}) \]

Implicit methods tend to be more stable than their explicit counterparts. But there is a price, \( y_{n+1} \) is a zero of a usually nonlinear function.
Example

A system of two differential equations.

\[
\begin{align*}
    u' &= 998u + 1998v \\
    v' &= -999u - 1999v
\end{align*}
\]

If the initial values \( u(0) = v(0) = 1 \), then the exact solution is

\[
\begin{align*}
    u &= 4e^{-t} - 3e^{-1000t} \\
    v &= -2e^{-t} + 3e^{-1000t}
\end{align*}
\]

![Graph](image_url)
### Example (cont.)

Suppose we use (forward) Euler’s method

\[
\begin{align*}
  u_{n+1} &= u_n + h(998u_n + 1998v_n) \\
  v_{n+1} &= v_n + h(-999u_n - 1999v_n)
\end{align*}
\]

with \( u_0 = v_0 = 1.0 \), then we get

<table>
<thead>
<tr>
<th></th>
<th>( h = 0.01 )</th>
<th>( h = 0.001 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>u0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>u1</td>
<td>30.96</td>
<td>3.996</td>
</tr>
<tr>
<td>u2</td>
<td>-239.1</td>
<td>3.992</td>
</tr>
<tr>
<td>u3</td>
<td>-9785</td>
<td>-7.988</td>
</tr>
<tr>
<td>u4</td>
<td>52420</td>
<td>-31.92</td>
</tr>
</tbody>
</table>
Example (cont.)
If we use the backward Euler method

\[ u_{n+1} = u_n + h(998u_{n+1} + 1998v_{n+1}) \]
\[ v_{n+1} = v_n + h(-999u_{n+1} - 1999v_{n+1}) \]
If we use the backward Euler method

\[ u_{n+1} = u_n + h(998u_{n+1} + 1998v_{n+1}) \]
\[ v_{n+1} = v_n + h(-999u_{n+1} - 1999v_{n+1}) \]

Solving the linear system

\[
\begin{bmatrix}
1 - 998h & -1998h \\
999h & 1 + 1999h
\end{bmatrix}
\begin{bmatrix}
u_{n+1} \\
v_{n+1}
\end{bmatrix}
= 
\begin{bmatrix}
u_n \\
v_n
\end{bmatrix}
\]
With initial values \( u_0 = v_0 = 1.0 \),

\[
\begin{array}{c|c|c}
\ & h = 0.01 & h = 0.001 \\
 u_0 & 1.0 & 1.0 \\
u_1 & 3.688 & 2.496 \\
u_2 & 3.896 & 3.242 \\
u_3 & 3.880 & 3.613 \\
u_4 & 3.844 & 3.797 \\
\end{array}
\]
Example (cont.)

For comparison, the solution values

\[
\begin{array}{c|c|c}
\text{h} & 0.01 & 0.001 \\
\hline
\text{u(0)} & 1.0 & 1.0 \\
\text{u(1)} & 3.960 & 2.892 \\
\text{u(2)} & 3.921 & 3.586 \\
\text{u(3)} & 3.882 & 3.839 \\
\text{u(4)} & 3.843 & 3.929 \\
\end{array}
\]
Hybrid methods

One of the difficulties in implicit methods is that they involve solving usually nonlinear systems.
Hybrid methods

One of the difficulties in implicit methods is that they involve solving usually nonlinear systems.
Combining both explicit and implicit:
Use an explicit method for predictor and an implicit method for corrector
Example. Fourth-order Adam’s method
Predictor (fourth-order, explicit):

\[ y_{n+1} = y_n + \frac{h}{24}(55f_n - 59f_{n-1} + 37f_{n-2} - 9f_{n-3}) \]

Corrector (fourth-order, implicit):

\[ y_{n+1} = y_n + \frac{h}{24}(9f_{n+1} + 19f_n - 5f_{n-1} + f_{n-2}) \]
Algorithm:

1. **Prediction**: use the predictor to calculate $y_{n+1}^{(0)}$
2. **Evaluation**: $f_{n+1}^{(0)} = f(y_{n+1}^{(0)}, t_{n+1})$
3. **Correction**: use the corrector to calculate $y_{n+1}^{(1)}$

Steps 2 and 3 are repeated until $|y_{n+1}^{(i+1)} - y_{n+1}^{(i)}| \leq tol$. Then set $y_{n+1} = y_{n+1}^{(i+1)}$. 
PECE methods

Algorithm:

1. **Prediction**: use the predictor to calculate $y_{n+1}^{(0)}$
2. **Evaluation**: $f_{n+1}^{(0)} = f(y_{n+1}^{(0)}, t_{n+1})$
3. **Correction**: use the corrector to calculate $y_{n+1}^{(1)}$

Steps 2 and 3 are repeated until $|y_{n+1}^{(i+1)} - y_{n+1}^{(i)}| \leq tol$. Then set $y_{n+1} = y_{n+1}^{(i+1)}$.

Usually, PECE, one iteration and a final evaluation for the next step.
Outline

1. Initial Value Problem
   - Euler’s Method
   - Runge-Kutta Methods
   - Multistep Methods
   - Implicit Methods
   - Hybrid Method

2. Software Packages
Software packages

**IMSL**  ivprk (RK), ivpag (AB)

**MATLAB**  ode23, ode45 (RK),
            ode113 (AB), ode15s, ode23s (stiff), bvp4c (BVP)

**NAG**  d02baf (RK), d02caf (AB),
         d02eaf (stiff)

**Netlib**  dverk (RK), ode (AB),
          vode, vodpk (stiff)

**Octave**  lsode
Family of solutions of a differential equation, initial value problem

Transform a high order ODE into a system of first order ODEs

Errors: Discretization errors (global, local), stability of a differential equation (mathematical stability), roundoff error, total error

Accuracy: Order of a method
Euler’s method: Explicit, single step, first order
Runge-Kutta methods: Explicit, single step
Multistep methods: Adams-Bashforth family
Implicit methods: Backward Euler’s method
Prediction-Correction scheme: Combination of explicit and implicit methods