1 Biochemical Reactions

The following reactions are those of an autocatalytic reaction between 3 chemical species $A$, $B$, and $C$:

\[
\begin{align*}
A & \xrightarrow{k_1} B, \\
B + B & \xrightarrow{k_3} C + B, \\
B + C & \xrightarrow{k_3} A + C.
\end{align*}
\]

Write this as a system of first order ODEs in standard form.

**Solution:**

\[
\begin{align*}
A'(t) &= -k_1 A(t) + k_3 B(t)C(t), \\
B'(t) &= -k_2 B^2(t) - k_3 B(t)C(t) + k_1 A(t) + k_2 B^2(t), \\
C'(t) &= -k_3 B(t)C(t) + k_2 B^2(t) + k_3 B(t)C(t).
\end{align*}
\]

(1.1)

\[
\begin{align*}
A'(t) &= -k_1 A(t) + k_3 B(t)C(t), \\
B'(t) &= -k_2 B(t)C(t) + k_1 A(t), \\
C'(t) &= k_2 B^2(t).
\end{align*}
\]

(1.2)

Note that here $A$ is an enzyme, $B$ is a complex and $C$ is a product.
2 From high order equation to first order system

Write the IVP
\[
\begin{cases}
x'''(t) - ax''(t) - bx'(t) - cx(t) = f(t), \\
x(0) = \xi, x'(0) = \eta, x''(0) = \zeta,
\end{cases}
\] (2.1)

as a first-order system \( \vec{x}'(t) = A \vec{x}(t) + \vec{g}(t) \). What is the characteristic polynomial of \( A \)?

Solution:
Let \( x_1(t) = x(t), x_2(t) = x'(t), x_3(t) = x''(t) \). By Eq.(2.1), we can get
\[
\begin{align*}
x'_1(t) &= x_2(t), \\
x'_2(t) &= x_3(t), \\
x'_3(t) &= ax_3(t) + bx_2(t) + cx_1(t) + f(t), \\
x_1(0) &= \xi, x_2(0) = \eta, x_3(0) = \zeta.
\end{align*}
\] (2.2)

Let
\[
\vec{x}(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \end{bmatrix}, \quad A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ c & b & a \end{bmatrix}, \quad \vec{g}(t) = \begin{bmatrix} 0 \\ 0 \\ f(t) \end{bmatrix}.
\]

Then Eqs.(2.2) can be rewritten as
\[
\begin{align*}
\frac{d\vec{x}}{dt} &= A\vec{x}(t) + \vec{g}(t), \\
\vec{x}(0) &= \begin{bmatrix} \xi \\ \eta \\ \zeta \end{bmatrix}.
\end{align*}
\] (2.3)

The characteristic polynomial of \( A \) is
\[
p(\lambda) = \det(\lambda I - A) = \lambda^3 - a\lambda^2 - b\lambda - c.
\] (2.4)
3 Nonlinear ODE

The solution of $y' = y(1 - y)$, $y(0) = c$ is

$$y(t) = \frac{c}{\exp(-t) + c(1 - \exp(-t))}. \tag{3.1}$$

Determine the linearized ODE for a solution with a slightly different initial value.

Solution:
To be specific, let $y(t; \varepsilon)$ satisfy the given ODE with $y(0; \varepsilon) = c + \varepsilon$.

Let

$$y(t; \varepsilon) = y(t) + \varepsilon z(t) + O(\varepsilon^2). \tag{3.2}$$

We want to determine an ODE that is satisfied by the linear term $z(t)$ in Eq.(3.2).

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

$$\Rightarrow y' + \varepsilon z' + O(\varepsilon^2) = f(t, y) + \varepsilon \frac{\partial f}{\partial y}(t, y) z + O(\varepsilon^2),$$

$$\Rightarrow z' = \frac{\partial f}{\partial y}(t, y) z. \tag{3.3}$$

Here,

$$f(t, y) = y(1 - y),$$

$$\Rightarrow \frac{\partial f}{\partial y}(t, y) = 1 - 2y. \tag{3.4}$$

By Eqs.(3.1),(3.3),(3.4), we can get

$$z'(t) = (1 - 2y(t)) z,$$

$$\Rightarrow z'(t) = \frac{\exp(-t) - c(1 + \exp(-t))}{\exp(-t) + c(1 - \exp(-t))} z, \tag{3.5}$$

with initial condition

$$z(0) = 1. \tag{3.6}$$
4 Long-term dynamics

Suppose the separable Hamiltonian problem

\begin{align}
p'(t) &= -V'(q), \\
q'(t) &= T'(p),
\end{align}

is such that $V'(0) = T'(0) = 0$, $V''(0) > 0$ and $T''(0) > 0$. Show that the Jacobian at the equilibrium point $p = q = 0$ has purely imaginary eigenvalues.

**Proof.** Let

\[
\vec{y}(t) = \begin{pmatrix} p(t) \\ q(t) \end{pmatrix}, \quad \vec{f}(\vec{y}) = \begin{pmatrix} -V'(q) \\ T'(p) \end{pmatrix}.
\]

First, $V'(0) = T'(0) = 0$ implies that $p = q = 0$ is the equilibrium point. Second, the Jacobian matrix is

\[
J = \begin{pmatrix} 0 & T''(p) \\ -V''(q) & 0 \end{pmatrix}, \quad J_0 = \begin{pmatrix} 0 & T''(0) \\ -V''(0) & 0 \end{pmatrix},
\]

\[
\Rightarrow \quad \det(\lambda I - J_0) = \lambda^2 + T''(0)V''(0),
\]

\[
\Rightarrow \quad \lambda = \pm i \sqrt{T''(0)V''(0)},
\]

since $V''(0) > 0$ and $T''(0) > 0$.

It follows that Jacobian at the equilibrium point $p = q = 0$ has purely imaginary eigenvalues.

\[\square\]
5 Euler method

The Euler method with step size $h$ applied to $y' = y$, $y(0) = 1$, produces a numerical solution $\{y_n\}$, which is identical (at grid points) to the analytical solution for $y' = c(h)y$ where $c(h)$ is to be determined.

1. Determine $c(h)$.

2. Let $h = t/n$ for some fixed $t > 0$. Obtain an expression for the global error $y_n - y(t_n)$ in terms of $h$ and $t$ only.

3. Use your answer to part (b) to obtain the first 3 terms in an expansion for the global error in powers of $h$.

5.1 Determine $c(h)$

On one hand, suppose that we use forward Euler, then

\[
\frac{y_{n+1} - y_n}{h} = y_n, \\
y_{n+1} = (1 + h)y_n, \\
y_n = (1 + h)^n, \tag{5.1}
\]

since $y(0) = y_0 = 1$.

On the other hand,

\[
y' = c(h)y, \quad y(0) = 1, \\
y(t) = e^{c(h)t}. \tag{5.2}
\]

By Eqs.(5.1)-(5.2), we can get

\[
c(h) = \frac{\ln(1 + h)}{h}. \tag{5.3}
\]

Check that

\[
\lim_{h \to 0} c(h) = \lim_{h \to 0} \frac{\ln(1 + h)}{h} = 1, \\
\Rightarrow \lim_{h \to 0} y(t) = \lim_{h \to 0} e^{c(h)t} = e^t,
\]

which is reasonable.
5.2 Global error

It is easy to know that \( y_n = (1 + h)^n, y(t_n) = e^{t_n} \). Then the global error at time \( t = t_n \) should be

\[
\begin{align*}
y_n - y(t_n) &= (1 + h)^n - e^{t_n}, \\
&= (1 + h)^{t_n/h} - e^{t_n},
\end{align*}
\]

(5.4)

since \( t_n = nh \).

5.3 First 3 terms

It is easy to know that the Taylor series of \((1 + x)^{t/x}\) at \( x = 0 \) is

\[
(1 + x)^{t/x} = e^t - \frac{(te^t)x}{2} + \frac{t(3t + 8)e^t x^2}{24} - \frac{t(t^2 + 8t + 12)e^t x^3}{48} + o(x^4).
\]

(5.5)

Therefore, the global error is

\[
y_n - y(t_n) = (1 + h)^{t_n/h} - e^{t_n}
\]

\[
= -\frac{(t_n e^{t_n})}{2} + \frac{t_n (3t_n + 8) e^{t_n}}{24} h^2 - \frac{t_n (t_n^2 + 8t_n + 12) e^{t_n}}{48} h^3 + o(h^4).
\]

(5.6)

It implies that if \( t_n \) is fixed and \( h = t_n/n \to 0 \), then the rate of the convergence of \( |y_n - y(t_n)| \) is just first order \( o(h) \).
1 Van del Pol Oscillator

For the second-order IVP
\[ \ddot{x}(t) - 10(1 - x^2(t))x'(t) + x(t) = \sin(\pi t), \quad (1.1) \]
\[ x(t_0) = \eta_0, \quad x'(t_0) = \eta_1. \quad (1.2) \]

We define \( u = x, v = x' \) so that
\[ u'(t) = v(t), \quad (1.3) \]
\[ v'(t) = 10(1 - u^2(t))v(t) - u(t) + \sin(\pi t), \quad (1.4) \]

with initial conditions
\[ u(t_0) = \eta_0, \quad v(t_0) = \eta_1. \quad (1.5) \]

We can rewrite this problem in vector form
\[ \vec{x}'(t) = \vec{f}(t, \vec{x}(t)), \quad (1.6) \]

where
\[ \vec{x}(t) = \begin{pmatrix} u(t) \\ v(t) \end{pmatrix}, \quad \vec{f}(t, \vec{x}(t)) = \begin{pmatrix} v(t) \\ 10(1 - u^2(t))v(t) - u(t) + \sin(\pi t) \end{pmatrix}. \]

In the following numerical experiments, we drop the term \( \sin(\pi t) \) and use the initial condition \( u(0) = 1, v(0) = 5. \)

*E-mail address: wywshtj@gmail.com; Tel: 765 237 7149
1.1 Explicit Adams methods

Implement the explicit Adams (fixed-stepszie) methods of stepnumber 1, 2, and 3. For the 2-step method, take the 1st step with the 1-step method. For the 3-step method, take the 1st step with the 1-step method and the 2nd with the 2-step method. Normally, when the order is varied, one uses a smaller step size when the order is lower.

1. 1-step explicit Adams:

$$x_{n+1} = x_n + hf_n.$$  \hspace{1cm} (1.7)

The Matlab code is “adams_one”:

```matlab
function [x,t] = adams_one(t0,h,N,x0)
    t = [t0:h:(t0+N*h)];
    x = zeros(2,N+1);
    x(:,1) = x0;
    for j=1:N
        x(:,j+1) = x(:,j) + h*f(t(j),x(:,j));
    end
    plot(x(1,:),x(2,:))
end
```

2. 2-step explicit Adams:

- first step: \( x_1 = x_0 + hf_0 \), \hspace{1cm} (1.8)
- after first step: \( x_{n+2} = x_{n+1} + \frac{h}{2}(3f_{n+1} - f_n) \). \hspace{1cm} (1.9)

The Matlab code is “adams_two”: 

```matlab
function [x,t] = adams_two(t0,h,N,x0)
    t = [t0:h:(t0+N*h)];
    x = zeros(2,N+1);
    x(:,1) = x0;
    x(:,2) = x0 + h*f(t(1),x(:,1));
    for j=2:N
        x(:,j+1) = x(:,j) + 0.5*h*(3*f(t(j),x(:,j)) - f(t(j-1),x(:,j-1)));
    end
    plot(x(1,:),x(2,:))
end
```
function [x,t] = adams_two(t0,h,N,x0)
%%% solve the problem \( x'(t) = f(t,x) \) using 2-step explicit Adams
%%% the first step is 1-step explicit Adams with step size \( h/10 \)
%%% \( t0 \): initial time
%%% \( h \): step size
%%% \( N \): the number of time steps
%%% \( x0 \): initial value (column vector)
%%% \( [x,t] \): numerical solution (vector)

%%% set up
\[
t = [t0:h:(t0+N*h)];
\]
\[
x = zeros(2,N+1);
x(:,1) = x0;
\]

%%% use 1-step method to compute the first step
\[
xx = adams_one(t0,h/10,10,x0);
x(:,2) = xx(:,end);
\]

%%% main loop
\[
for j=1:N-1
\]
\[
x(:,j+2) = x(:,j+1) + 0.5*h*(3*f(t(j+1),x(:,j+1)) - f(t(j),x(:,j)));
\]
\[
end
\]

3. 3-step explicit Adams:

\[
\text{first step: } \quad x_1 = x_0 + hf_0, \quad (1.10)
\]
\[
\text{second step: } \quad x_2 = x_1 + \frac{h}{2}(3f_1 - f_0), \quad (1.11)
\]
\[
\text{after first two steps: } \quad x_{n+3} = x_{n+2} + \frac{h}{12}(23f_{n+2} - 16f_{n+1} + 5f_n). \quad (1.12)
\]

The Matlab code is “adams_three”:

function [x,t] = adams_three(t0,h,N,x0)
%%% solve the problem \( x'(t) = f(t,x) \) using 3-step explicit Adams
%%% the first step is 1-step explicit Adams with step size \( h/100 \)
%%% the second step is 2-step explicit Adams with step size \( h/10 \)
%%% t0: initial time
%%% h: step size
%%% N: the number of time steps
%%% x0: initial value (column vector)
%%% [x,t]: numerical solution (vector)

%% set up
\begin{verbatim}
t = [t0:h:(t0+N*h)];
x = zeros(2,N+1);
x(:,1) = x0;
\end{verbatim}

%% use 1-step method to compute the first step
[xx,tx] = adams_one(t0,h/100,100,x0);
x(:,2) = xx(:,end);

%% use 2-step method to compute the second step
[yy,ty] = adams_two(t0+h,h/10,10,x(:,2));
x(:,3) = yy(:,end);

%% main loop
\begin{verbatim}
for j=1:N-2
    x(:,j+3) = x(:,j+2) + 1/12*h*(23*f(t(j+2),x(:,j+2))
    ...- 16*f(t(j+1),x(:,j+1)) + 5*f(t(j),x(:,j))));
end
\end{verbatim}

Remark 1.1. In the Matlab codes for 2-step and 3-step method, I use smaller step sizes to start.

1.2 Largest stepsize

Let $\mathcal{H} = \{1,0.5,0.2,0.1,0.05,0.02,\ldots\}$. Determine the largest stepsize $h \in \mathcal{H}$ for the 1-step method is stable.

Solution: The results of several numerical tests are shown in Fig.1, which indicates that in order to get stable numerical solutions, the largest possible stepsize is $h = 0.02$. 
1.3 1-step method

Compute numerical solutions using the 1-step method with step size $h$ and with step size $h/2$. Also compute the numerical solution using the 3-step method with stepsize 0.002. We will call this the exact solution.

The numerical results are shown in Fig.2. It indicates that the numerical solution obtained by $h/2$ is better than the one obtained by $h$, which agrees with the theory.
1.4 2-step method

Compute numerical solutions using the 1-step method and using the 2-step method, each with step size $h$. The numerical results are shown in Fig.3. It indicates that the numerical solution obtained by 2-step method is better that the one obtained by 1-step method with the same step size $h$, which agrees with the theory.

Compare the solution of the 2-step method with step size $h$ to that of the 1-step method with step size $h/2$. The numerical results are shown in Fig.4. It indicates that the numerical solution obtained by 2-step method with stepsize $h$ is better that the one obtained by 1-step method with the stepsize $h/2$, which is reasonable since the global error for the 2-step method with $h$ is $o(h^2)$ while the one for 1-step method with $h/2$ is $o(h/2)$. We know that $h^2 \ll h/2$ if $h = 0.02$. 

Figure 2: 1-step explicit Adams
1.5 3-step method

Using the 3-step method with step size $h$. Compare the solution of the 3-step method to that of the 2-step method.

The numerical results are shown in Fig. 5. It indicates that the numerical solution obtained by 3-step method is more accurate than that the one obtained by 2-step method with the same step size $h$ over most of the region.

However, there is a little bit oscillation in the numerical solution obtained by 3-step method with step size $h = 0.02$. (Notice that there is no oscillation in the so called “exact solution” which is obtained by 3-step method with step size $\tilde{h} = 0.002$). Hence, in my mind, the 2-step method with step size $h$ is qualitatively superior than 3-step method with the same step size $h$. 

Figure 3: 1-step and 2-step explicit Adams
2 Taylor expansion

Let \( f(x) \) be a vector of functions of many variables \( x \).

2.1 Difference

Use the fundamental theorem of calculus to get an expression for the difference \( f(x + w) - f(x) \) solely in terms of values of the Jacobian \( \partial f / \partial x \).

Solution:
Define
\[
\varphi(s) = f(x + sw),
\]  
(2.1)
Figure 5: 2-step and 3-step explicit Adams

then

\[ \varphi'(s) = \frac{\partial f}{\partial x}(x + sw)w, \]

where \( \frac{\partial f}{\partial x} \) is the Jacobian and \( \frac{\partial f}{\partial x}(x + sw) \) means the Jacobian evaluated at \( x + sw \).

According to the fundamental theorem of calculus, we can get

\[ \varphi(1) - \varphi(0) = \int_{0}^{1} \varphi'(s)ds = \varphi'(t)(1 - 0), \quad t \in [0, 1], \]

\[ \Rightarrow \quad f(x + w) - f(x) = \frac{\partial f}{\partial x}(x + tw)w, \quad \text{for some } t \in [0, 1]. \]
2.2 Multivariate Taylor expansion

Obtain the first two terms in a multivariate Taylor expansion for \( f(x + w) \) in “powers” of \( w \). Do not resort to indexing—stay with the vector notation.

Solution:

\[
    f(x + w) = f(x) + \left( \frac{\partial f}{\partial x}(x) \right) w + o(w^2).
\]

3 System of ODEs

Consider the initial value problem

\[
    \begin{cases}
        u'(t) = v(t), & v'(t) = -u(t), \quad t > 0, \\
        u(0) = 1, & v(0) = 0.
    \end{cases} \tag{3.1}
\]

3.1 Taylor series methods

We calculate

\[
    \frac{d}{dt}(u^2(t) + v^2(t)) = 2u(t)u'(t) + 2v(t)v'(t) = 0,
\]

\[\Rightarrow\]

\[
    u^2(t) + v^2(t) = \text{constant},
\]

\[\Rightarrow\]

\[
    u^2(t) + v^2(t) = u^2(0) + v^2(0) = 1.
\]

Recall that the TS(2) method is

\[
    x_{n+1} = x_n + hx'_n + \frac{1}{2}h^2x''_n. \tag{3.2}
\]

Using the TS(2) method, we find a means of computing \( u_{n+1} \) and \( v_{n+1} \):

\[
    \begin{cases}
        u'(t) = v(t), \\
        v'(t) = -u(t),
    \end{cases}
\]

\[\Rightarrow\]

\[
    \begin{cases}
        u''(t) = v'(t) = -u(t), \\
        v''(t) = -u'(t) = -v(t),
    \end{cases}
\]

\[\Rightarrow\]

\[
    \begin{cases}
        u_{n+1} = u_n + hv_n - \frac{1}{2}h^2u_n, \\
        v_{n+1} = v_n - hu_n - \frac{1}{2}h^2v_n.
    \end{cases} \tag{3.3}
\]
By Eq.(3.3), we can get
\[ u_{n+1}^2 + v_{n+1}^2 = \left( u_n + hv_n - \frac{1}{2} h^2 u_n \right)^2 + \left( v_n - hu_n - \frac{1}{2} h^2 v_n \right)^2, \]
\[ = (u_n^2 + v_n^2) \left( 1 + \frac{1}{4} h^4 \right). \]
Considering the initial conditions in Eq.(3.1) and by induction, we can get
\[ u_n^2 + v_n^2 = \left( 1 + \frac{1}{4} h^4 \right)^n. \quad (3.4) \]

3.2 Preservation of the invariant

Obtain the first two nonzero coefficients \( c_k(t) \) in an asymptotic expansion
\[ u_n^2 + v_n^2 = c_0(t_n) + hc_1(t_n) + h^2 c_2(t_n) + \cdots \quad (3.5) \]
valid on a fixed finite interval \( 0 \leq t_n \leq b \). The first nonzero coefficient is \( c_0(t) \). Compare the order of accuracy of the preservation of the invariant with that of the numerical solution.

Solution:
Let \( t_n = nh \). By Eq.(3.4), we can get
\[ u_n^2 + v_n^2 = \left( 1 + \frac{1}{4} h^4 \right)^n = \left( 1 + \frac{1}{4} h^4 \right)^{t_n/h}. \]
It follows that
\[ \ln(u_n^2 + v_n^2) = \frac{t_n}{h} \ln \left( 1 + \frac{1}{4} h^4 \right), \]
\[ = \frac{t_n}{h} \left( \frac{1}{4} h^4 - \frac{1}{2} \left( \frac{1}{16} h^8 \right) + \cdots \right), \]
\[ = t_n \left( \frac{1}{4} h^3 - \frac{1}{32} h^8 + \cdots \right). \]
It follows that
\[
    u_n^2 + v_n^2 = e^{t_n\left(\frac{1}{4}h^3 - \frac{1}{32}h^8 + \cdots\right)},
\]
\[
    = 1 + \frac{t_n}{4}h^3 - \frac{1}{32}h^8 + \cdots,
\]

which means the first nonzero coefficient in Eq.(3.5) is \(c_0(t_n) = 1\) and the second one is \(c_3(t_n) = \frac{1}{4}t_n\).

Furthermore, the numerical solutions also enjoy the preservation of the invariant in some sense, i.e. \(u_n^2 + v_n^2 = 1\). And this kind of preservation has order \(o(h^3)\).

### 4 Quadratic nonlinear term

Consider the initial value problem

\[
\begin{align*}
    x'(t) &= 1 + x^2(t), \\
    x(0) &= 0.
\end{align*}
\]

(4.1)

Using the backward Euler method, we can get

\[
    x_1 - x_0h = (1 + x_1^2), \quad x_0 = 0,
\]

\[
    \Rightarrow \quad hx_1^2 - x_1 + h = 0,
\]

\[
    \Rightarrow \quad x_1 = \frac{1 \pm \sqrt{1 - 4h^2}}{2h}.
\]

(4.2)

If we choose the “+” sign in Eq.(4.2), then

\[
    x_1 = \frac{1 + \sqrt{1 - 4h^2}}{2h},
\]

\[
    \Rightarrow \quad \lim_{h \to 0} x_1 = \infty,
\]

which contradicts with fact that

\[
    \lim_{h \to 0} x_1 = x_0 = 0.
\]

Thus, we should choose the “−” sign in Eq.(4.2). Besides, in order to avoid the round-off error, it is better to rewrite \(x_1\) as

\[
    x_1 = \frac{1 - \sqrt{1 - 4h^2}}{2h} = \frac{2h}{1 + \sqrt{1 - 4h^2}}.
\]

(4.3)
5 Two step LLM

Consider the initial value problem

\[
\begin{aligned}
  \frac{dx}{dt} &= -x(t), \\
  x(0) &= 1.
\end{aligned}
\]  

(5.1)

The exact solution to this problem is

\[ x(t) = e^{-t}. \]

Employing the two step LLM

\[ x_{n+2} - 2x_{n+1} + x_n = h(f_{n+1} - f_n), \]  

(5.2)

to this problem, we can get

\[
\begin{aligned}
  x_{n+2} - 2x_{n+1} + x_n &= h(-x_{n+1} + x_n), \\
  \Rightarrow x_{n+2} - (2 - h)x_{n+1} + (1 - h)x_n &= 0.
\end{aligned}
\]  

(5.3)

The auxiliary equation is

\[
\rho(r) = r^2 - (2 - h)r + (1 - h) = (r - 1)(r - (1 - h)).
\]

It implies that the general solution to the Eq.(5.3) is

\[ x_n = A + B(1 - h)^n. \]  

(5.4)

1. If we use this starting values

\[ x_0 = x_1 = 1, \]

then the numerical solution is

\[ x_n = 1. \]

However, the exact solution is

\[ x(t_n) = e^{-t_n}. \]

Obviously, the numerical solution does not converge to the exact solution as \( n \) gets bigger and bigger.
2. If we use this starting values

\[ x_0 = 1, x_1 = 1 + ah, \]

then

\[ x_n = 1 + a - a(1 - h)^n, \]
\[ = 1 + a - a(1 - h)^{tn/h}, \]
\[ \rightarrow 1 + a - ae^{-tn}, \text{ as } n \rightarrow \infty. \]

Besides, if we start with an Euler step, then

\[ x_1 - x_0 = hx_0' = -x_0, \]
\[ \Rightarrow x_1 = x_0 - hx_0 = 1 - h, \]

which implies that \( a = -1. \)

In order to know the zero-stability of the LMM (5.2), we need the “first characteristic polynomial”,

\[ \rho(r) = r^2 - 2r + 1 = (r - 1)^2, \]

which does not satisfy the “root condition”. It follows that the LMM (5.2) is not zero-stable.
1 Extrapolation formula

For the BDFs, it is typical to obtain an initial guess for the solution of the implicit equations by extrapolating from past values $y_{n-i}$ only. Such a predictor can be expressed as a linear multistep formula in which the $\beta_i$ are all zero. Though such a $k$-step extrapolation formula has local error $O(h^k)$, it does not use the RHS function $f(t, y)$, so it cannot be consistent. Determine the $k$-step extrapolation formula for $k = 1, 2, 3$ (for constant step size).

Solution: Recall that the Newton interpolation polynomial is

$$f(t) = f(t_n) + (t - t_n)f[t_n, t_{n-1}] + (t - t_n)(t - t_{n-1})f[t_n, t_{n-1}, t_{n-2}] + \cdots. \quad (1.1)$$

Then the extrapolation formula are

1. 1-step extrapolation:

$$f_{n+1} = f_n. \quad (1.2)$$

2. 2-step extrapolation:

$$f_{n+1} = f_n + hf_n - f_{n-1} = 2f_n - f_{n-1}. \quad (1.3)$$

3. 3-step extrapolation:

$$f_{n+1} = f_n + hf_n - f_{n-1} + 2h^2(f_n - f_{n-1})/h - (f_{n-1} - f_{n-2})/2h = 3f_n - 3f_{n-1} + f_{n-2}. \quad (1.4)$$

*E-mail address: wywshtj@gmail.com; Tel: 765 237 7149
2 Backward differentiation method

Consider the Van del Pol Oscillator

\[ u'(t) = v(t), \]
\[ v'(t) = 10(1 - u^2(t))v(t) - u(t), \]  
(2.1) \hspace{2cm} (2.2)

with initial conditions

\[ u(0) = 1, \quad v(0) = 5. \]  
(2.3)

We can rewrite this problem in vector form

\[ \mathbf{x}'(t) = \mathbf{f}(t, \mathbf{x}(t)), \]  
(2.4)

where

\[ \mathbf{x}(t) = \begin{pmatrix} u(t) \\ v(t) \end{pmatrix}, \quad \mathbf{f}(t, \mathbf{x}(t)) = \begin{pmatrix} v(t) \\ 10(1 - u^2(t))v(t) - u(t) \end{pmatrix}. \]

2.1 BDF3 with predictor

Implement the (fixed-stepsize) backward differentiation method of stepnumber 3. Take the 1st step with the 1-step method and the 2nd with the 2-step method. For a predictor for the \( k \)-step method, use the \( k \)-step extrapolation formula from Problem 1. To solve the implicit equations, provide the option of using a Newton-Raphson or a chord iteration. For a convergence test, require that the max norm of the difference between two successive iterates be no greater than \( 10^{-7} \).

The backward differentiation methods are:

1. BDF(1)

\[ y_{n+1} = y_n + hf_{n+1}. \]  
(2.5)

2. BDF(2)

\[ y_{n+2} = \frac{2}{3} \left( 2y_{n+1} - \frac{1}{2} y_n + hf_{n+2} \right). \]  
(2.6)

3. BDF(3)

\[ y_{n+3} = \frac{6}{11} \left( 3y_{n+2} - \frac{3}{2} y_{n+1} + \frac{1}{3} y_n + hf_{n+3} \right). \]  
(2.7)
For the problem Eq.(2.4), the Jacobian matrix is

\[
\frac{\partial \vec{f}}{\partial \vec{x}} = \begin{pmatrix} 0 & 1 \\ -20x_1x_2 - 1 & 10(1 - x_1^2) \end{pmatrix},
\] 

(2.8)

where \(x_1 = u\), \(x_2 = v\).

The Newton-Raphson iterations for the BDFs are

1. BDF(1)

\[
\left( I - h \frac{\partial \vec{f}}{\partial \vec{x}}(x_{n+1}^{[k+1]}) \right) E_{n+1}^{[k]} = x_{n+1}^{[k]} - h \vec{f}(x_{n+1}^{[k]}) - x_n,
\]

\(x_{n+1}^{[k+1]} = x_{n+1}^{[k]} - E_{n+1}^{[k]}\).

2. BDF(2)

\[
\left( I - \frac{2h}{3} \frac{\partial \vec{f}}{\partial \vec{x}}(x_{n+2}^{[k+1]}) \right) E_{n+2}^{[k]} = x_{n+2}^{[k]} - \frac{2h}{3} \vec{f}(x_{n+2}^{[k]}) - \frac{4}{3} x_{n+1} + \frac{1}{3} x_n,
\]

\(x_{n+2}^{[k+1]} = x_{n+2}^{[k]} - E_{n+2}^{[k]}\).

3. BDF(3)

\[
\left( I - \frac{6h}{11} \frac{\partial \vec{f}}{\partial \vec{x}}(x_{n+3}^{[k+1]}) \right) E_{n+3}^{[k]} = x_{n+3}^{[k]} - \frac{6h}{11} \vec{f}(x_{n+3}^{[k]}) - \frac{18}{11} x_{n+2} + \frac{9}{11} x_{n+1} - \frac{2}{11} x_n,
\]

\(x_{n+3}^{[k+1]} = x_{n+3}^{[k]} - E_{n+3}^{[k]}\).

The Matlab code of BDF3 with Newton-Raphson iteration is:

```
function [x,t] = bdf3(t0,h,N)
%%% solve the problem x'(t) = f(t,x) using 3-step BDF and Newton iteration
%%% t0: initial time
%%% h: step size
%%% N: the number of time steps
%%% [x,t]: numerical solution (vector)

%%% set up
x0 = [1;5];
```
t = [t0:h:(t0+N*h)];
x = zeros(2,N+1);
x(:,1) = x0;
ep = 1e-7;

%% BDF1 for the first step
x1 = x0;
e1 = 1;
n=0;
while e1>ep
    A = eye(2)-h*df(t0+h,x1);  
b = x1-h*f(t0+h,x1)-x0;
e11 = A\b;
x1 = x1 -e11;
e1 = norm(e11);
n=n+1;
    if n>100
        break
    end
end
x(:,2)=x1;

%% BDF2 for the second step
x2 = 2*x1-x0;
e2 = 1;
n=0;
while e2>ep
    A = eye(2)-2*h/3*df(t0+2*h,x2);  
b = x2-2*h/3*f(t0+2*h,x2)-4*x1/3 + x0/3;
e22 = A\b;
x2 = x2 -e22;
e2 = norm(e22);
n=n+1;
    if n>100
        break
    end
end
x(:,3)=x2;
%% BDF3
for k=3:N
    \% initial guess
    x3 = 3*x(:,k) - 3*x(:,k-1) + x(:,k-2);
    \% Newton iteration
    e3 = 1;
    n=0;
    while e3>ep
        A = eye(2) - 6/11*h*df(t0+k*h,x3);
        b = x3 - 6/11*h*f(t0+k*h,x3) - 18/11*x(:,k) + 9/11*x(:,k-1) - 2/11*x(:,k-2);
        e33 = A\b;
        e3 = norm(e33);
        x3 = x3 - e33;
        n=n+1;
        if n>100
            fprintf('not converging')
            break
        end
    end
    x(:,k+1) = x3;
end

And the chord iterations are

1. BDF(1)
\[
\begin{pmatrix}
I - h \frac{\partial \tilde{f}}{\partial \tilde{x}} (x^{[0]}_{n+1})
\end{pmatrix} E^{[k]}_{n+1} = x^{[k]}_{n+1} - h \tilde{f}(x^{[k]}_{n+1}) - x_n,
\]
\[
x^{[k+1]}_{n+1} = x^{[k]}_{n+1} - E^{[k]}_{n+1}.
\]

2. BDF(2)
\[
\begin{pmatrix}
I - \frac{2h}{3} \frac{\partial \tilde{f}^3}{\partial \tilde{x}} (x^{[0]}_{n+2})
\end{pmatrix} E^{[k]}_{n+2} = x^{[k]}_{n+2} - \frac{2h}{3} \tilde{f}(x^{[k]}_{n+2}) - \frac{4}{3} x_{n+1} + \frac{1}{3} x_n,
\]
\[
x^{[k+1]}_{n+2} = x^{[k]}_{n+2} - E^{[k]}_{n+2}.
\]
3. BDF(3)

\[
\begin{aligned}
  \left( I - \frac{6h}{11} \frac{\partial}{\partial x} (x_{n+3}^{[0]}) \right) E_{n+3}^{[k]} &= x_{n+3}^{[k]} - \frac{6h}{11} f(x_{n+3}^{[k]}) - \frac{18}{11} x_{n+2} + \frac{9}{11} x_{n+1} - \frac{2}{11} x_n, \\
  x_{n+3}^{[k+1]} &= x_{n+3}^{[k]} - E_{n+3}^{[k]}.
\end{aligned}
\]

The Matlab code for the BDF3 with chord iteration is

```matlab
function [x,t] = bdf3_chord(t0,h,N)

%%% solve the problem x'(t) = f(t,x) using 3-step BDF and chord iteration
%%% t0: initial time
%%% h: step size
%%% N: the number of time steps
%%% [x,t]: numerical solution (vector)

%%% set up
x0 = [1;5];
t = [t0:h:(t0+N*h)];
x = zeros(2,N+1);
x(:,1) = x0;
ep = 1e-7;

%%% BDF1 for the first step
x1 = x0;
e1 = 1;
n=0;
A1 = eye(2) - h*df(t0+h,x1);
while e1>ep
    b = x1-h*f(t0+h,x1)-x0;
e11 = A1\b;
x1 = x1 -e11;
e1 = norm(e11);
n=n+1;
if n>100
    break
end
end
x(:,2)=x1;
```
%% BDF2 for the second step
x2 = 2*x1 - x0;
e2 = 1;
n=0;
A2 = eye(2) - 2*h/3*df(t0+2*h,x2);
while e2>ep
    b = x2 - 2*h/3*f(t0+2*h,x2) - 4*x1/3 + x0/3;
e22 = A2\b;
x2 = x2 - e22;
e2 = norm(e22);
n=n+1;
if n>100
    break
end
x(:,3)=x2;

%% BDF3
for k=3:N
    % initial guess
    x3 = 3*x(:,k) - 3*x(:,k-1) + x(:,k-2);
    % chord iteration
    e3 = 1;
n=0;
A3 = eye(2) - 6/11*h*df(t0+k*h,x3);
while e3>ep
    b = x3 - 6/11*h*f(t0+k*h,x3) - 18/11*x(:,k) + 9/11*x(:,k-1) - 2/11*x(:,k-2);
e33 = A3\b;
e3 = norm(e33);
x3 = x3 - e33;
n=n+1;
if n>100
    sprintf('not converging')
    break
end
end
x(:,k+1) = x3;
end

Besides, the Matlab codes for the function \textit{f.m} and its Jacobian \textit{df.m} are

\begin{verbatim}
function y = f(t,x)
    \texttt{\%\% Van del Pol Oscillator}
    y = zeros(2,1);
    y(1) = x(2);
    y(2) = 10*(1-(x(1))^2)*x(2) - x(1);

function y = df(t,x)
    \texttt{\%\% Van del Pol Oscillator: Jacobian matrix}
    y = zeros(2,2);
    y(1,2) = 1;
    y(2,1) = -20*x(1)*x(2) - 1;
    y(2,2) = 10*(1-(x(1))^2);
\end{verbatim}

2.2 Largest step size
Let $\mathcal{H} = \{0.010, 0.011, \ldots, 0.020\}$. Determine the largest step size $h \in \mathcal{H}$ for which the integrator succeeds when using the Newton-Raphson iteration.

The numerical results for Newton-Raphson iteration with $h = 0.01$ and $h = 0.02$ are shown in Figs.1-2, which shows that $\forall h \in \mathcal{H}$ is OK.

However, for the explicit 3-step Adams formula, if $h = 0.02$, then there is little bit oscillation in the numerical solution. It verifies that BDF has larger stable region than explicit Adams.

2.3 Numerical solution of Newton-Raphson iteration
The numerical results for $h = 0.02$ are shown in Figs.2, in which the exact solution is obtained by setting $h = 0.002$.

2.4 Numerical solution of chord iteration
The numerical results for chord iteration with $h = 0.01$ and $h = 0.02$ are shown in Figs.3-4, which shows that $\forall h \in \mathcal{H}$ is OK.
Besides, we know that for this problem, the largest step sizes of the Newton-Raphson iteration and chord iteration are the same, while the chord iteration enjoys less computational costs.

Figure 1: BDF3 with Newton-Raphson iteration $h = 0.01$

Figure 2: BDF3 with Newton-Raphson iteration $h = 0.02$
Figure 3: BDF3 with chord iteration $h = 0.01$

Figure 4: BDF3 with chord iteration $h = 0.02$
3 BDF2

**Definition 3.1.** A numerical method is A-stable if its region of absolute stability contains the left half-plane $h \Re(\lambda) < 0$.

**Lemma 3.1.** Prove the 2-step backward differentiation method is A-stable.

**Proof.** Consider the model problem

$$y' = \lambda y.$$  \hspace{1cm} (3.1)

The 2-step backward differentiation for this problem is

$$\frac{3}{2}y_n - 2y_{n-1} + \frac{1}{2}y_{n-2} = \lambda h y_n.$$  \hspace{1cm} (3.2)

Let $z = h\lambda$, then the stability polynomial is

$$\pi(\xi; z) = \rho(\xi) - z\sigma(\xi),$$

$$= \left(\frac{3}{2} - z\right) \xi^2 + 2\xi + \frac{1}{2}.$$  

The roots of the stability polynomial are

$$\xi_1 = \frac{2 + \sqrt{2z + 1}}{3 - 2z},$$

$$\xi_2 = \frac{2 - \sqrt{2z + 1}}{3 - 2z}.$$  

Claim that $\forall z = -a + bi$, $a > 0, a, b \in \mathbb{R}$, we have $|\xi_1| < 1$ and $|\xi_2| < 1$.

$$|\xi| < 1,$$

$$\Rightarrow |2 \pm \sqrt{2z + 1}| < |3 - 2z|,$$

$$\Rightarrow |2 \pm \sqrt{2z + 1}|^2 < |3 - 2z|^2,$$

$$\Rightarrow 4 + |\sqrt{2z + 1}|^2 \pm 4\Re(\sqrt{2z + 1}) < 9 + 4|z|^2 - 12\Re(z),$$

$$\Rightarrow |\sqrt{2z + 1}|^2 - 4|z|^2 + 12\Re(z) \pm 4\Re(\sqrt{2z + 1}) < 5.$$  

The stable region is given in Fig.5, which contains the left-half plane. Hence, BDF2 is A-stable.
4 Index of DAE

Consider the equations

\[ \begin{align*}
C' & = -R, \\
T' & = -(T - T_C), \\
0 & = R - e^{-1/T}C.
\end{align*} \]  

(4.1) (4.2) (4.3)

4.1 Hessenberg index-1

Assuming \( T_C(t) \) is given, what is the index of this DAE? Is it in Hessenberg form?

Solution: Recall that the general DAE of Hessenberg index-1 is in this form

\[ \begin{align*}
x' & = f(t, x, z), \\
0 & = g(t, x, z),
\end{align*} \]  

(4.4) (4.5)

where \( x, z, f, g \) may be vector.

In this case, just choose \( x = (C, T)^T \) and \( z = R \), then Eqs.(4.1)-(4.3) are the in Hessenberg form with index 1.
Besides, if we differentiate Eq.(4.3) once, then we can get
\[ R' = e^{-1/T}C' + e^{-1/T}T^{-2}T'C, \]
\[ = -Re^{-1/T} - e^{-1/T}T^{-2}C(T - T_C). \]

It follows that the index of this DAE is 1.

**4.2 Hessenberg index-2**

Assuming \( T_C \) is an additional unknown, to be determined so that
\[ C = u(t) \] (4.6)

is satisfied, where \( u(t) \) is given, what is the index of this DAE? Is it in Hessenberg form? Justify your answers.

**Solution:** Let us combine Eqs.(4.1)-(4.3) and Eq.(4.6).

\[
\begin{cases}
C' = -R, \\
T' = -(T - T_C), \\
0 = R - e^{-1/T}C, \\
0 = C - u, \\
0 = u' + e^{-1/T}u,
\end{cases}
\] (4.7)

\[
\Rightarrow \begin{cases}
T' = -(T - T_C), \\
0 = u' + e^{-1/T}u,
\end{cases}
\] (4.8)

where \( u \) is given.

Recall that the general DAE of Hessenberg index-2 is in this form
\[ x' = f(t, x, z), \] (4.9)
\[ 0 = g(t, x), \] (4.10)

where \( x, z, f, g \) may be vector.

Now it is clear that Eq.(4.8) is in Hessenberg form but Eq.(4.7) is not.

Finally, the index of this problem is 2.
5 Numerical algorithm of DAE

Consider a semi-explicit index-1 DAE of the form

\[
\begin{align*}
  f(t, x, z, x') &= 0, \\
  g(t, x, z) &= 0,
\end{align*}
\]

with initial condition \(x(0) = x_0\). Assume \(\partial f/\partial x'\) and \(\partial g/\partial z\) are square and nonsingular. The initialization problem is to find \(x'_0 = x'(0)\) and \(z_0 = z(0)\). Describe a good solution algorithm and be precise.

Solution:

1. Since \(x(0) = x_0\) is known and \(\partial g/\partial z\) is nonsingular, we can solve for \(z_0 = z(0)\) from Eq.(5.2) by Newton-Raphson iteration:

\[
  z_0^{[k+1]} = z_0^{[k]} - \left( \frac{\partial g}{\partial z}(0, x_0, z_0^{[k]}) \right)^{-1} g(0, x_0, z_0^{[k]}).
\]

2. Since both \(x(0) = x_0\) and \(z_0 = z(0)\) are known and \(\partial f/\partial x'\) are nonsingular, we can solve for \(x'_0 = x'(0)\) from from Eq.(5.1) by Newton-Raphson iteration:

\[
  (x'_0)^{[k+1]} = (x'_0)^{[k]} - \left( \frac{\partial f}{\partial z}(0, x_0, z_0, (x'_0)^{[k]}) \right)^{-1} f(0, x_0, z_0, (x'_0)^{[k]}).
\]

Now, the initialization problem is done.
1 Pendulum problem: reformulating

Consider the pendulum example:

\[ x_1'' = -\lambda x_1, \quad (1.1) \]
\[ x_2'' = -\lambda x_2 - g, \quad (1.2) \]
\[ 0 = x_1^2 + x_2^2 - 1, \quad (1.3) \]

where \( \lambda(t) \) is an unknown function and \( g \) is the known scaled constant of gravity.

Let us use the notation \( q \) for the position coordinates and \( v = q' \) for the velocities. Then the DAE is written as a first-order system

\[ q_1' = v_1, \quad (1.4) \]
\[ q_2' = v_2, \quad (1.5) \]
\[ v_1' = -\lambda q_1, \quad (1.6) \]
\[ v_2' = -\lambda q_2 - g, \quad (1.7) \]
\[ 0 = q_1^2 + q_2^2 - 1. \quad (1.8) \]

Taking differentiation of Eq.(1.8), we can get

\[ q_1 q_1' + q_2 q_2' = 0, \]
\[ \Rightarrow \quad q_1 v_1 + q_2 v_2 = 0. \quad (1.9) \]
Differentiating the velocity constraint Eq.(1.9) and substituting for $q'$ and $v'$ yields

\[ q_1 v_1' + q_2 v_2' + v_1^2 + v_2^2 = 0, \]
\[ \Rightarrow q_1(-\lambda q_1) + q_2(-\lambda q_2 - g) + v_1^2 + v_2^2 = 0, \]
\[ \Rightarrow -\lambda(q_1^2 + q_2^2) - gq_2 + v_1^2 + v_2^2 = 0, \]
\[ \Rightarrow -\lambda - gq_2 + v_1^2 + v_2^2 = 0, \]
\[ \Rightarrow \lambda = -gq_2 + v_1^2 + v_2^2. \] (1.10)

Given the initial values $q_1(0) = 0.6, v_1(0) = 0, q_2(0) < 0$, thanks to Eqs.(1.8)-(1.10), we can find the consistent initial conditions

\[ q_1(0) = 0.6, \quad q_2(0) = -0.8, \quad v_1(0) = 0, \quad v_2(0) = 0, \quad \lambda(0) = 25.6. \] (1.11)

**Remark 1.1.** Now there are two ways to solve this pendulum problem:

1. **Method I:** Differentiating (1.10) to get $\lambda'$ and combining Eqs.(1.4)-(1.7) yields the system of ODE,

\[ q_1' = v_1, \] (1.12)
\[ q_2' = v_2, \] (1.13)
\[ v_1' = -\lambda q_1, \] (1.14)
\[ v_2' = -\lambda q_2 - g, \] (1.15)
\[ \lambda' = -3gv_2 - 2\lambda(v_1q_1 + v_2q_2). \] (1.16)

We can solve this ODE with initial conditions (1.11).

2. **Method II:** Substituting Eq.(1.10) into Eqs.(1.4)-(1.7) yields the system of ODE

\[ q_1' = v_1, \] (1.17)
\[ q_2' = v_2, \] (1.18)
\[ v_1' = -(v_1^2 + v_2^2 - gq_2)q_1, \] (1.19)
\[ v_2' = -(v_1^2 + v_2^2 - gq_2)q_2 - g, \] (1.20)

with invariant equation corresponding to Eqs.(1.8)-(1.9),

\[ 0 = q_1^2 + q_2^2 - 1, \] (1.21)
\[ 0 = q_1v_1 + q_2v_2. \] (1.22)

We can just solve ODE (1.17)-(1.20) with initial conditions (1.11). Finally, we use Eq.(1.8) to compute $\lambda$. 

---

2
Numerical tests show that both of Method I and Method II are good if the initial conditions are consistent.

2 Pendulum problem: numerical solution

2.1 BDF(2) and Backward Euler

Let us consider the DAE or (ODE)

\[ F(y, y') = 0. \] (2.1)

1. Backward Euler

\[ F \left( y_n, \frac{y_n - y_{n-1}}{h} \right) = 0. \] (2.2)

Solving Eq.(2.2) by Newton-Raphson iteration yields

\[ E_n^{[k]} = \left[ \frac{\partial F^{[k]}}{\partial y} + \frac{1}{h} \frac{\partial F^{[k]}}{\partial y'} \right]^{-1} F^{[k]}, \] (2.3)

\[ y_n^{[k+1]} = y_n^{[k]} - E_n^{[k]}, \] (2.4)

where

\[ F^{[k]} = F \left( y_n^{[k]}, \frac{y_n^{[k]} - y_{n-1}}{h} \right). \]

2. BDF(2)

\[ F \left( y_n, \frac{3}{2} y_n - 2 y_{n-1} + \frac{1}{2} y_{n-2} \right) = 0. \] (2.5)

Solving Eq.(2.5) by Newton-Raphson iteration yields

\[ E_n^{[k]} = \left[ \frac{\partial F^{[k]}}{\partial y} + \frac{3}{2h} \frac{\partial F^{[k]}}{\partial y'} \right]^{-1} F^{[k]}, \] (2.6)

\[ y_n^{[k+1]} = y_n^{[k]} - E_n^{[k]}, \] (2.7)

where

\[ F^{[k]} = F \left( y_n^{[k]}, \frac{3}{2} y_n^{[k]} - 2 y_{n-1} + \frac{1}{2} y_{n-2} \right). \]
The Matlab code for the BDF2 is

```matlab
function yn = bdf(h,y,F_handle)
%%% use the Backward Euler and BDF(2) to solve F(y,y')=0.
%%% y = [ ..., y_{n-3},y_{n-2},y_{n-1}] 

%%% prepare
ep = 1e-7;

%%% solve
s = size(y);
s1 = s(1);
s2 = s(2);
e = [];
if s2<1.5 
    % Backward Euler
    yn = y;
e1 = 100;
    yp = zeros(s1,1);
    while e1>ep 
        [F, dfdy, dfdyp] = F_handle(yn,yp);
        A = dfdy + dfdyp/h;
        e11 = A\F;
        yn = yn-e11;
        e2 = norm(e11,inf);
        ee = [ee,e2];
        if e2>e1
            fprintf('not convergent')
            break
        end
        e1 = e2;
        yp = (yn - y)/h;
    end
else 
    % BDF(2)
    y1 = y(:,end);
y2 = y(:,end-1);
end
```


e1 = 100;
yn = y1;
yp = zeros(s1,1);
while e1>ep
    [F, dfdy, dfdyp] = F_handle(yn,yp);
    A = dfdy + 1.5*dfdyp/h;
    e11 = A/F;
    yn = yn - e11;
    e2 = norm(e11,inf);
    ee = [ee,e2];
    if e1>e1
        fprintf('not convergent')
        break
    end
    e1 = e2;
    yp = (1.5*yn - 2*y1 + 0.5*y2)/h;
end

%% display error
ne = length(ee);
for i=1:ne
    fprintf('
');
end

2.2 Jacobian matrices for pendulum problem

To compute $\frac{\partial F}{\partial y}$ and $\frac{\partial F}{\partial y'}$, we use this matlab code.

1. Method I:

    function [F,dfdy,dfdyp]=F_etc1(y,yp)
    %%% Pendulum problem
    %%% y and yp should be columns with dim = 5;
yy = zeros(5,1);
yy(1) = y(3);
yy(2) = y(4);
yy(3) = -y(5)*y(1);
yy(4) = -y(5)*y(2) - 32;
yy(5) = -2*y(1)*y(3)*y(5) - 2*y(2)*y(4)*y(5) - 96*y(4);

F = yp-yy;
dfdy = [0 0 1 0 0;0 0 0 1 0; -y(5) 0 0 0 -y(1); 0 -y(5) 0 0 -y(2);-2*y(3)*y(5)
      ... -2*y(4)*y(5) -2*y(5)*y(1) -96-2*y(2)*y(5) -2*(y(1)*y(3)+y(2)*y(4))];
dfdyp = eye(5);

2. Method II:

function [F,dfd,y,dfdyp]=F_etc(y,yp)
    q = y(3)^2 + y(4)^2 - 32*y(2);
    yy = zeros(4,1);
    yy(1) = y(3);
    yy(2) = y(4);
    yy(3) = -q*y(1);
    yy(4) = -q*y(2) - 32;

    F = yp-yy;
    dfdy = [-0 0 1 0 0;0 0 0 1 0; -q 32*y(1) -2*y(3) -2*y(4); 0 64*y(2) -2*y(3) -2*y(4)];
    dfdyp = eye(4);

2.3 Numerical results

Here, we choose $g = 32$ and step size $h = 0.002$, iteration tolerance $1e-7$.

The numerical results obtained by Method I with 5 steps are shown in Table 1. The max norm of the difference between two successive iterates in Newton iteration are shown in Table 2.

The numerical results obtained by Method II with 5 steps are shown in Table 3. The max norm of the difference between two successive iterates in Newton iteration are shown
in Table 4. Furthermore, in Fig. 1, we also compute the two constraints in Eqs.(1.21)-(1.22). It follows that the numerical method works very well with time step size $h = 0.002$.

Table 1: Numerical solution obtained by Method I ($h = 0.002$, first 5 step)

<table>
<thead>
<tr>
<th></th>
<th>$t = 0$</th>
<th>$t = h$</th>
<th>$t = 2h$</th>
<th>$t = 3h$</th>
<th>$t = 4h$</th>
<th>$t = 5h$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_1$</td>
<td>0.6000</td>
<td>0.5999</td>
<td>0.5998</td>
<td>0.5997</td>
<td>0.5995</td>
<td>0.5992</td>
</tr>
<tr>
<td>$q_2$</td>
<td>-0.8000</td>
<td>-0.8000</td>
<td>-0.8001</td>
<td>-0.8002</td>
<td>-0.8004</td>
<td>-0.8006</td>
</tr>
<tr>
<td>$v_1$</td>
<td>0</td>
<td>-0.0307</td>
<td>-0.0614</td>
<td>-0.0922</td>
<td>-0.1229</td>
<td>-0.1537</td>
</tr>
<tr>
<td>$v_2$</td>
<td>0</td>
<td>-0.0230</td>
<td>-0.0461</td>
<td>-0.0691</td>
<td>-0.0920</td>
<td>-0.1150</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>25.6000</td>
<td>25.6044</td>
<td>25.6118</td>
<td>25.6231</td>
<td>25.6386</td>
<td>25.6585</td>
</tr>
</tbody>
</table>

Table 2: Max norm of the difference between two successive iterates (Method I)

<table>
<thead>
<tr>
<th></th>
<th>$N = 1$</th>
<th>$N = 2$</th>
<th>$N = 3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t = h$</td>
<td>3.0722e-02</td>
<td>3.0187e-07</td>
<td>1.0345e-15</td>
</tr>
<tr>
<td>$t = 2h$</td>
<td>2.0483e-02</td>
<td>1.0242e-02</td>
<td>3.7286e-08</td>
</tr>
<tr>
<td>$t = 3h$</td>
<td>2.0486e-02</td>
<td>1.0243e-02</td>
<td>5.2228e-08</td>
</tr>
<tr>
<td>$t = 4h$</td>
<td>2.0491e-02</td>
<td>1.0245e-02</td>
<td>7.2162e-08</td>
</tr>
<tr>
<td>$t = 5h$</td>
<td>2.0497e-02</td>
<td>1.0247e-02</td>
<td>9.3782e-08</td>
</tr>
</tbody>
</table>

Table 3: Numerical solution obtained by Method II ($h = 0.002$, first 5 step)

<table>
<thead>
<tr>
<th></th>
<th>$t = 0$</th>
<th>$t = h$</th>
<th>$t = 2h$</th>
<th>$t = 3h$</th>
<th>$t = 4h$</th>
<th>$t = 5h$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_1$</td>
<td>0.6000</td>
<td>0.5999</td>
<td>0.5998</td>
<td>0.5997</td>
<td>0.5995</td>
<td>0.5992</td>
</tr>
<tr>
<td>$q_2$</td>
<td>-0.8000</td>
<td>-0.8000</td>
<td>-0.8001</td>
<td>-0.8002</td>
<td>-0.8004</td>
<td>-0.8006</td>
</tr>
<tr>
<td>$v_1$</td>
<td>0</td>
<td>-0.0307</td>
<td>-0.0614</td>
<td>-0.0922</td>
<td>-0.1229</td>
<td>-0.1536</td>
</tr>
<tr>
<td>$v_2$</td>
<td>0</td>
<td>-0.0230</td>
<td>-0.0461</td>
<td>-0.0691</td>
<td>-0.0920</td>
<td>-0.1150</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>25.6000</td>
<td>25.6029</td>
<td>25.6098</td>
<td>25.6210</td>
<td>25.6365</td>
<td>25.6563</td>
</tr>
</tbody>
</table>
Table 4: Max norm of the difference between two successive iterates (Method II)

<table>
<thead>
<tr>
<th></th>
<th>N = 1</th>
<th>N = 2</th>
<th>N = 3</th>
<th>N = 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>t = h</td>
<td>3.0719e-02</td>
<td>2.3586e-06</td>
<td>6.2611e-14</td>
<td></td>
</tr>
<tr>
<td>t = 2h</td>
<td>2.0482e-02</td>
<td>1.0241e-02</td>
<td>4.1062e-06</td>
<td>1.1273e-09</td>
</tr>
<tr>
<td>t = 3h</td>
<td>2.0486e-02</td>
<td>1.0242e-02</td>
<td>6.4638e-06</td>
<td>2.7052e-09</td>
</tr>
<tr>
<td>t = 4h</td>
<td>2.0491e-02</td>
<td>1.0243e-02</td>
<td>8.8197e-06</td>
<td>4.9581e-09</td>
</tr>
<tr>
<td>t = 5h</td>
<td>2.0498e-02</td>
<td>1.0245e-02</td>
<td>1.1173e-05</td>
<td>7.8845e-09</td>
</tr>
</tbody>
</table>

Figure 1: Numerical solution obtained by Method II for h = 0.002 with 5000 steps
3 Pendulum problem: nonconsistent initial conditions

Consider the Method I and choose the initial conditions as

\[ q_1(0) = 0.6, \quad q_2(0) = -0.8, \quad v_1(0) = 0, \quad v_2(0) = 0, \quad \lambda(0) = -12.8, \]

which is not consistent.

Then the Method I fails. It implies that if the initial error is very large, then it does not converge. The reason is the Newton iteration relies heavily on the initial guess.

Following is the output.

\[
\begin{align*}
\text{h} & = 0.010000 \\
\text{err} & = 22.400 \\
\text{err} & = 0.016415 \\
\text{err} & = 4.0782e-07 \\
\text{err} & = 2.8023e-13 \\
\text{solution} & = \\
& = 0.59795 -0.80154 -0.20539 -0.15363 9.58358 \\
\text{err} & = 22.531 \\
\text{err} & = 0.016651 \\
\text{err} & = 1.0333e-07 \\
\text{err} & = 7.2742e-13 \\
\text{solution} & = \\
& = 0.59451 -0.80409 -0.41252 -0.30583 9.41903 \\
\text{err} & = 0.020004 \\
\text{err} & = 0.029514 \\
\text{not converging}
\end{align*}
\]

\[
\begin{align*}
\text{h} & = 0.0050000 \\
\text{err} & = 22.400 \\
\text{err} & = 0.0040980 \\
\text{err} & = 6.3890e-09 \\
\text{solution} & = \\
\end{align*}
\]
0.599488  -0.800384  -0.102474  -0.076804  9.595902
 err = 22.433
 err = 0.0041126
 err = 1.5898e-09

solution =

0.59863  -0.80102  -0.20517  -0.15343  9.55490
 err = 0.0048348
 err = 0.0073059
 not converging
4 Wiener process

Let $0 = t_0 < t_1 < \cdots < t_{k-1} < t_{k+1} < \cdots < t_N$. Let $W_i$, $i \neq k$ and $0 \leq i \leq N$, be Gaussian random variables satisfying $E[W_i] = 0$ and $E[W_i W_j] = \min\{t_i, t_j\}$. Let $Z$ be a standard Gaussian random variable independent of $W_i$, $i \neq k$, $0 \leq i \leq N$. Define

$$W_k = aW_{k-1} + bW_{k+1} + \sqrt{ab\sqrt{t_{k+1} - t_{k-1}}}Z$$

(4.1)

where $a = (t_{k+1} - t_k)/(t_{k+1} - t_{k-1})$ and $b = (t_k - t_{k-1})/(t_{k+1} - t_{k-1})$. Determine $E[W_k]$ and $E[W_k W_j]$, $j = 0, 1, \ldots, N$. And of course simplify! What does formula (4.1) do for us?

**Solution:** It is easy to know that

$$E[W_k] = 0.$$  \hspace{1cm} (4.2)

For $E[W_k W_j]$, let us consider the $j = 0, 1, \ldots, N$ in different ranges:

1. if $j = 0, 1, \ldots, k - 1$, then

$$E[W_k W_j] = a\min\{t_{k-1}, t_j\} + b\min\{t_{k+1}, t_j\},$$

$$= (a + b)t_j,$$

$$= t_j.$$  \hspace{1cm} (4.3)

2. if $j = k$, then

$$E[W_k W_j] = a^2E[W_{k-1}^2] + b^2E[W_{k+1}^2] + ab(t_{k+1} - t_{k-1})E[Z^2] + 2abE[W_{k-1} W_{k+1}],$$

$$= a^2t_{k-1} + b^2t_{k+1} + ab(t_{k+1} - t_{k-1}) + 2abt_{k-1}$$

$$= t_k.$$  \hspace{1cm} (4.4)

3. if $j = k + 1, \ldots, N$, then

$$E[W_k W_j] = a\min\{t_{k-1}, t_j\} + b\min\{t_{k+1}, t_j\},$$

$$= at_{k-1} + bt_{k+1},$$

$$= t_k.$$  \hspace{1cm} (4.5)

It follows that Eq.(4.1) is some kind of interpolation at $t = t_k$ if we know $0 = t_0 < t_1 < \cdots < t_{k-1} < t_{k+1} < \cdots < t_N$. In other words, if we add one more point as (4.1) into a sequence of Wiener process, then the new sequence is also a Wiener process.
5 Riemann-Stieltjes integral

Express the Riemann-Stieltjes integral $\int_0^T f(t) dW(t)$ as a Riemann integral by integrating by parts under the assumption that $f \in C^1[0, T]$.

Solution:

$$\int_0^T f(t) dW(t) = W(t) f(t) \bigg|_0^T - \int_0^T W(t) df(t),$$

$$= W(T) f(T) - \int_0^T W(t) f'(t) dt.$$ 

6 Standard Gaussian

How would one generate a random number having the same distribution as $\int_0^T W(t) dt$ given the ability to generate a standard Gaussian $Z$?

Solution:

$$\int_0^T W(t) dt = W(t) t \bigg|_0^T - \int_0^T t dW(t),$$

$$= W(T) T - \left( \int_0^T t^2 dt \right)^{1/2} Z,$$

$$= \sqrt{T^3} Z - \left( \frac{T^3}{3} \right)^{1/2} Z,$$

$$= \sqrt{\frac{4}{3}} T^3 Z.$$

It follows that if we have the standard Gaussian $Z(0, 1)$, then we just need to multiply this by $\sqrt{\frac{4}{3}} T^3$ to generate $\int_0^T W(t) dt$. 
7 Euler-Maruyama method

Theorem 7.1. For the case \( f(x) = ax \) and \( g(x) = bx \), the Euler-Maruyama method takes the form

\[
x_{k+1} = (1 + ha)x_k + \sqrt{hb}Z_kx_k,
\]

where each \( Z_k \sim N(0, 1) \) which is independent with \( x_k \). Suppose the initial condition \( x(0) = x_0 \) is deterministic.

Consider the limit where \( h \to 0 \) and \( n \to \infty \) with \( nh = t_f \) fixed. Show that

\[
E[x_n] \to e^{at_f}x_0,
\]

\[
\text{var}[x_n] \to e^{2at_f}(e^{b^2t_f} - 1)x_0^2.
\]

Proof. Since \( Z_k \) and \( x_k \) are independent with each other, we know that

\[
E[x_kZ_k] = E[x_k]E[Z_k] = 0,
\]

\[
E[x_k^2Z_k] = E[x_k^2]E[Z_k] = 0,
\]

\[
E[x_kZ_k^2] = E[x_k^2]E[Z_k^2] = E[x_k^2].
\]

Taking expectation in Eq. (7.1), we find that

\[
E[x_{k+1}] = E \left[ x_k(1 + ha) + \sqrt{hb}Z_k \right],
\]

\[
= (1 + ha)E[x_k] + \sqrt{hb}E[x_kZ_k],
\]

\[
= (1 + ha)E[x_k].
\]

It follows that

\[
E[x_n] = (1 + ha)^nx_0 = (1 + ha)^{tf}x_0,
\]

\[
\Rightarrow \lim_{n \to \infty} E[x_n] = \lim_{h \to 0}(1 + ha)^{tf}x_0 = e^{at_f}x_0.
\]

Now we have proved (7.2).
Similarly, squaring both sides of Eq.(7.1) and then taking the expectation, we find
\[
E[x_{k+1}^2] = E \left[ x_k^2 (1 + ha)^2 + 2(1 + ha) \sqrt{h} b x_k Z_k + h b^2 x_k^2 Z_k^2 \right],
\]
\[
= (1 + ha)^2 E[x_k^2] + 2(1 + ha) \sqrt{h} b E[x_k^2] Z_k + h b^2 E[x_k^2] Z_k^2,
\]
\[
= ((1 + ha)^2 + h b^2) E[x_k^2],
\]
(7.8)

Divided by \(E[x_k^2]\) and taking natural logarithms on both sides of Eq.(7.8) yields
\[
\log \left( \frac{E[x_{k+1}^2]}{E[x_k^2]} \right) = \log \left( (1 + ha)^2 + h b^2 \right),
\]
\[
= \log \left( 1 + (2a + b^2) h + O(h^2) \right),
\]
\[
= (2a + b^2) h + O(h^2).
\]
(7.9)

Taking summation on both sides of (7.9), we can get
\[
\sum_{k=0}^{n-1} \log \left( \frac{E[x_{k+1}^2]}{E[x_k^2]} \right) = n(2a + b^2) h + nO(h^2),
\]
\[
\Rightarrow \log \left( \frac{E[x_n^2]}{E[x_0^2]} \right) = t_f (2a + b^2) + O(h),
\]
\[
\Rightarrow E[x_n^2] = x_0^2 e^{t_f (2a + b^2)} + O(h),
\]
\[
\Rightarrow \lim_{n \to \infty} E[x_n^2] = \lim_{h \to 0} x_0^2 e^{t_f (2a + b^2)} + O(h) = x_0^2 e^{t_f (2a + b^2)}. \tag{7.10}
\]

It follows that
\[
\lim_{n \to \infty} \text{var}[x_n^2] = \lim_{n \to \infty} \left( E[x_n^2] - E[x_n]^2 \right),
\]
\[
= x_0^2 e^{t_f (2a + b^2)} - \left( e^{a t_f} x_0 \right)^2,
\]
\[
= e^{2 a t_f} \left( e^{b^2 t_f} - 1 \right) x_0^2. \tag{7.11}
\]

Now we are done.
CS 614: Homework 5

Yingwei Wang

Department of Mathematics, Purdue University

1 Stratonowich integral

From first principles, determine the Stratonowich integral

\[ \int_0^T W(t) \circ dW(t). \]

The Stratonowich integral uses the midpoints rather than the left endpoints of the subintervals of the partitions whose limit defines the integral.

Solution:

\[
\int_0^T W(t)dW(t) \\
= \lim_{h \to 0} \sum_{n=1}^{N} W(t_{n-1/2})(W(t_n) - W(t_{n-1})) \\
= \lim_{h \to 0} \sum_{n=1}^{N} \left( \frac{1}{2} W(t_n)^2 - \frac{1}{2} W(t_{n-1})^2 + \frac{1}{2} \left( (W(t_{n-1/2}) - W(t_{n-1}))^2 - (W(t_n) - W(t_{n-1/2}))^2 \right) \right) \\
= \frac{1}{2} W(T)^2 + \lim_{h \to 0} \frac{1}{2} \sum_{n=1}^{N} \left( (t_{n-1/2} - t_{n-1})(Z_n)^2 - (t_n - t_{n-1/2})(\tilde{Z}_n)^2 \right)
\]
where the $Z_n$ and $\tilde{Z}_n$ are independent Gaussian random variables, for which $\mathbb{E}((Z_n)^2) = \mathbb{E}((\tilde{Z}_n)^2) = 1$ and $\text{Var}((Z_n)^2) = \text{Var}((\tilde{Z}_n)^2) = 2$.

Hence, we have

$$
\mathbb{E} \left( \sum_{n=1}^{N} \left( (t_{n-1/2} - t_{n-1}) (Z_n)^2 - (t_n - t_{n-1/2}) (\tilde{Z}_n)^2 \right) \right) = 0.
$$

and

$$
\text{Var} \left( \sum_{n=1}^{N} \left( (t_{n-1/2} - t_{n-1}) (Z_n)^2 - (t_n - t_{n-1/2}) (\tilde{Z}_n)^2 \right) \right) \leq 2hT.
$$

Therefore, in the sense of Stratonovich integral,

$$
\int_{0}^{T} W(t) dW(t) = \frac{1}{2} W(T)^2 \quad \text{w.p.1.} \quad (1.1)
$$

2 Ito formula

2.1 About the $\int_{0}^{T} W(t) dW(t)$

Let $y = W_t$, then $dy = dW_t$. Let $g(y) = \frac{1}{2} y^2$, then by Ito formula

$$
dg(y) = g_t dt + g_y dy + \frac{1}{2} g_{yy} dy dy,
$$

$$
= 0 + y dW_t + \frac{1}{2} dt.
$$

Taking integration on both sides leads to

$$
\frac{1}{2} W(T)^2 = \int_{0}^{T} W_t dW_t + \frac{1}{2} T.
$$
It follows that

\[ \int_0^T W_t dW_t = \frac{1}{2} W(T)^2 - \frac{1}{2} T, \]
\[ = (N(0, T/2))^2 - \frac{1}{2} T, \]
\[ = \left( \int_0^T \frac{1}{\sqrt{2}} dW_t \right)^2 - \frac{1}{2} T. \]

Here, \( f(t) = \frac{1}{\sqrt{2}} \) is a function independent of \( W(t) \) for which the integrals \( \int_0^T f(t)^2 dt = \frac{1}{2} T \) is known.

### 2.2 About the \( \int_0^T W(t)^2 dW(t) \)

Let \( y = W_t \), then \( dy = dW_t \). Let \( g(y) = \frac{1}{4} y^3 \), then by Ito formula

\[ dg(y) = g_t dt + y_y dy + \frac{1}{2} y_y y_{yy} dy, \]
\[ = 0 + y^2 dW_t + y dt. \]

Taking integration on both sides leads to

\[ \frac{1}{3} W(T)^3 = \int_0^T W(t)^2 dW_t + \int_0^T W(t) dt, \]
\[ = \int_0^T W(t)^2 dW_t + TW(T) - \int_0^T tdW_t. \]

It follows that

\[ \int_0^T W_t^2 dW_t = \frac{1}{3} W(T)^3 - TW(T) + \int_0^T tdW_t, \]
\[ = \frac{1}{3} W(T)^3 - N(0, T^3) + \int_0^T tdW_t, \]
\[ = \frac{1}{3} W(T)^3 - \int_0^T TdW_t + \int_0^T tdW_t. \]

Here, \( f_1(t) = T \) and \( f_2(t) = t \).
3 Weak Ito-Taylor expansions

For the algorithm for one step of the second order Taylor method given in Section 7.2.6 of the class notes,

\[ R_1 = \int_0^h dW_t = N(0, h), \]  
\[ R_2 = \int_0^h t dW_t = N \left( 0, \frac{1}{3} h^3 \right). \]

show how to generate \( R_1 \) and \( R_2 \) from two independent standard Gaussian random variables \( Z_1 \) and \( Z_2 \).

By Eqs.(3.1)-(3.2), we can get

\[ R_1 + R_2 = \int_0^h (1 + t) dW_t = N \left( 0, \frac{1}{3} h^3 + h^2 + h \right). \]  

Let

\[ R_1 = c_1 Z_1, \]  
\[ R_2 = c_2 Z_1 + c_3 Z_2. \]

By Eqs.(3.1)-(3.3), we can get

\[ c_1^2 = h, \]  
\[ c_2^2 + c_3^2 = \frac{1}{3} h^3, \]  
\[ (c_1 + c_2)^2 + c_3^2 = \frac{1}{3} h^3 + h^2 + h. \]

It follows that

\[ c_1 = h^{1/2}, \]  
\[ c_2 = \frac{1}{2} h^{3/2}, \]  
\[ c_3 = \frac{1}{2\sqrt{3}} h^{3/2}. \]
Finally, $R_1$ and $R_2$ can be generated in this way

\begin{align}
R_1 &= h^{1/2}Z_1, \\
R_2 &= \frac{1}{2}h^{3/2}Z_1 + \frac{1}{2\sqrt{3}}h^{3/2}Z_2,
\end{align}

(3.12) (3.13)

where $Z_1$ and $Z_2$ are two independent standard Gaussian random variables.

4 Symplectic Euler method

The nonlinear oscillator

\[ u''(t) + f(u) = 0 \]  

(4.1)

is equivalent to the first-order system

\[ \begin{align*}
u'(t) &= -v(t), \\
v'(t) &= f(u(t)),
\end{align*} \]

(4.2)

with initial conditions $u(0) = 1$, $v(0) = 0$.

Symplectic Euler method applied to problem (4.2) leads to

\[ \begin{align*}
u_{n+1} &= u_n - hv_n, \\
v_{n+1} &= v_n + hf(u_{n+1}),
\end{align*} \]

(4.3)

with $u_0 = 1$, $v_0 = 0$.

We suppose that the modified equation is a system of two ODEs with dependent variables $x(t)$ and $y(t)$. The LTE of the given method is, therefore,

\[ \hat{T}_{n+1} = \begin{bmatrix} x(t + h) - x(t) + hy(t) \\
y(t + h) - y(t) - hf(x(t + h)) \end{bmatrix}, \quad t = nh. \]

(4.4)

which, by Taylor expansion, becomes

\[ \hat{T}_{n+1} = h \begin{bmatrix} x'(t) + \frac{1}{2}hx''(t) + y(t) \\
y'(t) + \frac{1}{2}hy''(t) - hf(x(t)) - h\frac{df}{dx}x'(t) \end{bmatrix} + \mathcal{O}(h^3). \]

(4.5)
We now suppose that the modified equations take the form

\[
\begin{align*}
    x'(t) &= -y(t) + ha(x, y), \\
y'(t) &= f(x(t)) + hb(x, y),
\end{align*}
\]  

(4.6)

where the functions \(a(x, y)\) and \(b(x, y)\) are to be determined. Differentiating these with respect to \(t\) gives

\[
\begin{align*}
    x''(t) &= -y'(t) + O(h) = -f(x(t)) + O(h), \\
y''(t) &= \frac{df}{dx}x'(t) + O(h) = -\frac{df}{dx}y(t) + O(h),
\end{align*}
\]  

(4.7)

Substitution into Eq.(4.5) then leads to

\[
\hat{T}_{n+1} = h^2 \begin{bmatrix} a(x, y) - \frac{1}{2}f(x) \\ b(x, y) + \frac{1}{2}\frac{df}{dx}y \end{bmatrix} + O(h^3). 
\]

(4.8)

Therefore, \(\hat{T}_{n+1} = O(h^3)\) on choosing \(a(x, y) = \frac{1}{2}f(x)\) and \(b(x, y) = -\frac{1}{2}\frac{df}{dx}y\). Our modified system of equations is, therefore,

\[
\begin{align*}
    x'(t) &= -y(t) + \frac{1}{2}hf(x(t)), \\
y'(t) &= f(x(t)) - \frac{1}{2}h\frac{df(x(t))}{dx}y(t).
\end{align*}
\]  

(4.9)

Suppose that \(F(u)\) is such that \(f(u) = \frac{dF(u)}{du}\). Then

\[
\begin{align*}
    \frac{d}{dt}[2F(x(t)) - hf(x(t))y(t) + y^2(t)], \\
    &= 2f(x)x'(t) - h(\frac{df(x)}{dx}x'(t)y(t) + f(x)y'(t)) + 2y(t)y'(t), \\
    &= (2f - h\frac{df(x(t))}{dx}y)x'(t) + (2y - hf)y'(t), \\
    &= 0.
\end{align*}
\]

Hence, the solutions lie on the family of curves

\[
2F(x(t)) - hf(x(t))y(t) + y^2(t) = \text{constant}. 
\]  

(4.10)
5 Angular momentum conservation

5.1 Conservation of the exact solution

Two particles in motion interacting through a central force satisfy

\[
\vec{r}_i' = \frac{1}{m_i} \vec{p}_i, \quad \vec{p}_i' = -\frac{\partial}{\partial \vec{r}_i} U(\|\vec{r}_2 - \vec{r}_1\|), \quad i = 1, 2. \tag{5.1}
\]

Lemma 5.1. The total angular momentum \( \sum_i \vec{r}_i \times \vec{p}_i \) is conserved.

Proof. Let us compute the changing rate of the total angular momentum.

\[
\frac{d}{dt} \left( \sum_i \vec{r}_i \times \vec{p}_i \right) = \sum_i \left( \vec{r}_i' \times \vec{p}_i + \vec{r}_i \times \vec{p}_i' \right),
\]

\[
= \sum_i \left( \frac{1}{m_i} \vec{p}_i \times \vec{p}_i - \vec{r}_i \times \frac{\partial}{\partial \vec{r}_i} U(\|\vec{r}_2 - \vec{r}_1\|) \right),
\]

\[
= \sum_i \left( -\vec{r}_i \times \frac{\partial}{\partial \vec{r}_i} U(\|\vec{r}_2 - \vec{r}_1\|) \right),
\]

\[
= -\vec{r}_1 \times \frac{\partial}{\partial \vec{r}_1} U(\|\vec{r}_2 - \vec{r}_1\|) - \vec{r}_2 \times \frac{\partial}{\partial \vec{r}_2} U(\|\vec{r}_2 - \vec{r}_1\|),
\]

\[
= -\vec{r}_1 \times U' \frac{\partial \|\vec{r}_2 - \vec{r}_1\|}{\partial \vec{r}_1} - \vec{r}_2 \times U' \frac{\partial \|\vec{r}_2 - \vec{r}_1\|}{\partial \vec{r}_2}. \tag{5.2}
\]

Let us consider the \( \|\vec{r}_2 - \vec{r}_1\| \).

\[
\|\vec{r}_2 - \vec{r}_1\| = \left( \|\vec{r}_1\|^2 + \|\vec{r}_2\|^2 - 2(\vec{r}_1, \vec{r}_2) \right)^{1/2},
\]

\[
\Rightarrow \begin{cases} 
\frac{\partial \|\vec{r}_2 - \vec{r}_1\|}{\partial \vec{r}_1} = \frac{\vec{r}_1 - 2\vec{r}_2}{2\|\vec{r}_2 - \vec{r}_1\|}, \\
\frac{\partial \|\vec{r}_2 - \vec{r}_1\|}{\partial \vec{r}_2} = \frac{\vec{r}_2 - 2\vec{r}_1}{2\|\vec{r}_2 - \vec{r}_1\|}. \tag{5.3}
\end{cases}
\]

Plugging Eq.5.3 into Eq.(5.2) yields

\[
\frac{d}{dt} \left( \sum_i \vec{r}_i \times \vec{p}_i \right) = 0. \tag{5.4}
\]
It follows that
\[ \sum \vec{r}_i \times \vec{p}_i = \text{constant.} \] (5.5)

### 5.2 Conservation of the numerical solution

One step of the symplectic Euler method is given by
\[ \vec{p}^1_i = \vec{p}_i + h \vec{F}_i, \quad \vec{r}^1_i = \vec{r}_i + \frac{h}{m_i} \vec{p}^1_i, \] (5.6)

where \( \vec{F}_i = -(\partial / \partial \vec{r}_i)U(\|\vec{r}_2 - \vec{r}_1\|) \) and the superscript 1 denotes the value at the new time point.

**Lemma 5.2.** The numerical total angular momentum \( \sum \vec{r}^1_i \times \vec{p}^1_i \) obtained by symplectic Euler method is conserved.

**Proof.** Similarly as the problem 4, we suppose that the modified equation is a system of ODEs with dependent variables \( x_i(t), y_i(t), \quad i = 1, 2 \). The LTE of the given method is, therefore,
\[ \hat{T}_{n+1} = \begin{bmatrix} x_i(t + h) - x_i(t) + \frac{h}{m} y_i(t + h) \\ y_i(t + h) - y_i(t) - hF(x_1, x_2) \end{bmatrix}, \quad t = nh. \] (5.7)

which, by Taylor expansion, becomes
\[ \hat{T}_{n+1} = h \begin{bmatrix} x'_i(t) + \frac{1}{2} hx''_i(t) - \frac{1}{m} (y_i(t) + hy'_i(t)) \\ y'_i(t) + \frac{1}{2} hy''_i(t) - hF(x_1, x_2) \end{bmatrix} + O(h^3). \] (5.8)

We now suppose that the modified equations take the form
\[
\begin{align*}
x'_i(t) &= \frac{1}{m} y(t) + ha_i(x, y), \\
y'_i(t) &= F_i(x_1, x_2) + hb_i(x, y),
\end{align*}
\] (5.9)
where the functions \( a_i(x, y) \) and \( b_i(x, y) \) are to be determined. Differentiating these with respect to \( t \) gives

\[
\begin{align*}
  x_i''(t) &= \frac{1}{m} y'_i(t) + \mathcal{O}(h) = \frac{1}{m} F_i(x_1, x_2) + \mathcal{O}(h), \\
  y_i''(t) &= \frac{\partial F_i}{\partial x_1} x'_1 + \frac{\partial F_i}{\partial x_2} x'_2 + \mathcal{O}(h) = \frac{1}{m} \left( \frac{\partial F_i}{\partial x_1} y_1 + \frac{\partial F_i}{\partial x_2} y_2 \right) + \mathcal{O}(h),
\end{align*}
\]

It follows that

\[
\hat{T}_{n+1} = h^2 \begin{bmatrix} a_i(x, y) - \frac{1}{2m} F_i(x_1, x_2) \\ b_i(x, y) + \frac{1}{2m} \left( \frac{\partial F_i}{\partial x_1} y_1 + \frac{\partial F_i}{\partial x_2} y_2 \right) \end{bmatrix} + \mathcal{O}(h^3). \tag{5.11}
\]

Therefore, \( \hat{T}_{n+1} = \mathcal{O}(h^3) \) on choosing \( a_i(x, y) = \frac{1}{2m} F_i(x_1, x_2) \) and \( b_i(x, y) = -\frac{1}{2m} \left( \frac{\partial F_i}{\partial x_1} y_1 + \frac{\partial F_i}{\partial x_2} y_2 \right) \).

Our modified system of equations is, therefore,

\[
\begin{align*}
  x'_i(t) &= \frac{1}{m} y(t) + \frac{h}{2m} F_i(x_1, x_2), \\
  y'_i(t) &= F_i(x_1, x_2) - \frac{h}{2m} \left( \frac{\partial F_i}{\partial x_1} y_1 + \frac{\partial F_i}{\partial x_2} y_2 \right),
\end{align*}
\]

with \( i = 1, 2 \).

Now it is easy to know that

\[
\frac{d}{dt} (x_1 \times y_1 + x_2 \times y_2) = 0. \tag{5.13}
\]

It follows that The numerical total angular momentum \( \sum_i \vec{r}_i \times \vec{p}_i \) obtained by symplectic Euler method is also conserved.
1 Rugnge-Kutta method and linear invariants

Definition 1.1. Given a general ODE system

\[ x'(t) = f(x), \] (1.1)

with \( x(t) = (x_1(t), \cdots, x_2(t))^T \in \mathbb{R}^m \) and \( f : \mathbb{R}^m \to \mathbb{R}^m \), we say that there is a linear variant if some linear combination of the solution components is always preserved

\[ \frac{d}{dt}(c_1 x_1(t) + \cdots + c_m x_m(t)) = 0, \] (1.2)

where \( c_1, \cdots, c_m \) are constants.

We may write this more compactly as

\[ \frac{d}{dt} c^T x(t) = 0, \] (1.3)

where \( c = (c_1, \cdots, c_m)^T \in \mathbb{R}^m \).

Because each \( x'_i \) is given by \( f_i(x) \), this equivalent to

\[ c_1 f_1(x) + \cdots + c_m f_m(x) = 0, \] (1.4)

which we may write as

\[ c^T f(x(t)) = 0, \quad \forall x \in \mathbb{R}^m. \] (1.5)
Theorem 1.1. Any Runge-Kutta method will preserve linear invariants of an ODE.

Proof. Suppose we have an $s$-stage Runge-Kutta scheme

\[ k_i = f \left( t_n + c_i h, x_n + h \sum_{j=1}^{s} a_{ij} k_j \right), \quad i = 1, \cdots, s, \]  

(1.6)

\[ x_{n+1} = x_n + h \sum_{i=1}^{s} b_i k_i, \quad c_i = \sum_{j=1}^{s} a_{ij}, \quad i = 1, \cdots, s. \]  

(1.7)

Suppose $c = (c_1, \cdots, c_m)^T \in \mathbb{R}^m$ is a constant vector such that Eq.(1.5) holds. Then

\[ c^T x_{n+1} = c^T x_n + h \sum_{i=1}^{s} b_i c^T f \left( x_n + h \sum_{j=1}^{s} a_{ij} k_j \right), \]

\[ = c^T x_n, \]  

(1.8)

where the last equality holds thanks to Eq.(1.5).

It follows that as long as the initial values satisfy $c^T x(0) = \text{constant}$, then all of the $x_n$ will keep this property. \qed

2 Theta method

For a system of ODEs $y' = f(y)$ in autonomous form, the theta method is given by

\[ y_n = y_{n-1} + h(\theta f(y_n) + (1 - \theta) f(y_{n-1})) \]  

(2.1)

where $\theta$ is a method parameter in the range $0 \leq \theta \leq 1$.

2.1 A-stable

Consider the model problem

\[ x'(t) = \lambda x(t). \]  

(2.2)

Applying the theta method to this problem, we can get

\[ x_{n+1} = x_n + h\lambda(\theta x_{n+1} + (1 - \theta)x_n), \]

\[ (1 - \hat{h}\theta)x_{n+1} = (1 + \hat{h}(1 - \theta))x_n, \quad \hat{h} = h\lambda, \]

\[ \Rightarrow \quad x_{n+1} = \frac{1 + \hat{h}(1 - \theta)}{1 - \hat{h}\theta} x_n. \]
It follows that the region of absolute stability is

\[
\left\{ \hat{h} \in \mathbb{C} : \left| \frac{1 + \hat{h}(1 - \theta)}{1 - h\theta} \right| < 1 \right\}.
\] (2.3)

Let \( \hat{h} = x + iy \), then

\[
\left| \frac{1 + \hat{h}(1 - \theta)}{1 - h\theta} \right| < 1,
\]

\[
\Rightarrow |1 + (1 - \theta)x + i(1 - \theta)y|^2 < |1 - \theta x - i\theta y|^2,
\]

\[
\Rightarrow (1 + (1 - \theta)x)^2 + (1 - \theta)^2y^2 < (1 - \theta x)^2 + \theta^2y^2,
\]

\[
\Rightarrow (1 - 2\theta)x^2 + 2x + (1 - 2\theta)y^2 < 0.
\] (2.4)

Let us consider inequality (2.4).

1. If \( \theta = 1/2 \), then (2.4) indicates that the region of the absolute stability is \( \{ \hat{h} = x + iy : x < 0 \} \), which implies A-stable.

2. If \( 0 \leq \theta < 1/2 \), then \( 1 - 2\theta > 0 \). Divided by \( (1 - 2\theta) \) on both sides of inequality (2.4) leads to

\[
\left( x + \frac{1}{1 - 2\theta} \right)^2 + y^2 < \left( \frac{1}{1 - 2\theta} \right)^2.
\] (2.5)

It follows that the region of the absolute stability is a disc centered at \( x_0 = -1/(1 - 2\theta), y_0 = 0 \) with radius \( r = 1/(1 - 2\theta) \). Hence, in this case, the method is not A-stable.

3. If \( 1/2 < \theta \leq 1 \), then \( 1 - 2\theta < 0 \). Divided by \( (1 - 2\theta) \) on both sides of inequality (2.4) leads to

\[
\left( x + \frac{1}{1 - 2\theta} \right)^2 + y^2 > \left( \frac{1}{1 - 2\theta} \right)^2.
\] (2.6)

It follows that the region of the absolute stability is the region out of the disc centered at \( x_0 = -1/(1 - 2\theta) > 0, y_0 = 0 \) with radius \( r = 1/(1 - 2\theta) \). Hence, in this case, the method is A-stable.

In conclusion, for \( 1/2 \leq \theta \leq 1 \), the method is A-stable.
2.2 Butcher array

Determine the Butcher tableau (Butcher array) for the theta method (as an implicit Runge-Kutta method).

\[
\begin{array}{c|cc}
0 & \\
1 & 1-\theta & \theta \\
1-\theta & \theta \\
\end{array}
\]  

(2.7)

3 Hamiltonian system

3.1 Introduction

Consider the Hamiltonian

\[ H = \frac{1}{2}M^{-1}p^2 + U(q) \]  

(3.1)

where \( U(q) \) is the Morse potential given by

\[ U(q) = D(1 - e^{-S(q-q_0)})^2 \]

with \( D = 90.5 \times 0.0004814, S = 1.814, q_0 = 1.41, \) and \( M = 0.9953. \) Here \( q \) represents the distance between a pair of bonded atoms.

Write a MATLAB function \( \text{Ham} \), which given a 2-dimensional array \([q, p]^T\) returns the value of the Hamiltonian \( H(q, p) \) (for the Morse potential).

\begin{verbatim}
function h = ham(x)
    D = 90.5*0.0004814;
    S = 1.814;
    q0 = 1.41;
    M = 0.9953;
    h = (x(2))^2/(2*M) + D*(1-exp(-S*(x(1)-q0)))^2;
end
\end{verbatim}

Then the system of ODEs generated by such a separable Hamiltonian has the form

\[ q'(t) = M^{-1}p, \]  

(3.2)

\[ p'(t) = -2DSe^{-S(q-q_0)}(1 - e^{-S(q-q_0)}). \]  

(3.3)
Write a `rhs` function, which given a scalar position `q` and momentum `p`, returns the velocity and the force.

```matlab
function y = rhs(x)
    g = zeros(2,1);
    g(1) = x(2)/M;
    g(2) = -2*D*S*exp(-S*(x(1)-q0))*(1-exp(-S*(x(1)-q0)));
end
```

3.2 Kutta’s Simpson method

This is the “(classical) Runge–Kutta” method:

\[
\begin{align*}
    k_1 &= f(t_n, y_n), \\
    k_2 &= f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_1), \\
    k_3 &= f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_2), \\
    k_4 &= f(t_n + h, y_n + h k_3), \\
    y_{n+1} &= y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4).
\end{align*}
\]

For \( y' = f(t) \) this is Simpson’s rule.

Write a `RKstep` function which takes one step of Kutta’s Simpson method. It will take as arguments a 2-dimensional array \([q_{n-1}, p_{n-1}]^T\) and a stepsize \( h \) and return a 2-dimensional array \([q_n, p_n]^T\).

```matlab
function x1 = rkstep(x0,h)
    %%% One step of the 4th order RK for Hamilton system
end
```
both x0 and x1 should be column vectors.
h is step size.
hk1 = h*rhs(x0);
hk2 = h*rhs(x0+hk1/2);
hk3 = h*rhs(x0+hk2/2);
hk4 = h*rhs(x0+hk3);
x1 = x0 + (hk1+2*hk2+2*hk3+hk4)/6;

3.3 Leapfrog method

The leapfrog method for this problem is

\[ M \frac{q_{n+1} - 2q_n + q_{n-1}}{h^2} = F(q_n), \]  
\[ M \frac{q_{n+1} - q_{n-1}}{2h} = p_n, \]

where \( F(q) = -2DSe^{-S(q-q_0)}(1 - e^{-S(q-q_0)}) \).

The leapfrog method can be expressed as follows: We begin the step with values \( q_{n-1}, p_{n-1}, \) and \( F_{n-1} = F(q_{n-1}) \) obtained from the previous step. Then we compute

- \( \phi_{(h/2)K} \) (half-kick):
  \[ p_{n-1/2} = p_{n-1} + \frac{h}{2}F_{n-1}, \]

- \( \phi_{hK} \) (drift):
  \[ q_n = q_{n-1} + hM^{-1}p_{n-1/2}, \]

- \( \phi_{(h/2)U} \) (half-kick):
  \[ F_n = F(q_n), \]
  \[ p_n = p_{n-1/2} + \frac{h}{2}F_n. \]

Implement it by writing an \texttt{LFstep} function which takes one step of the leapfrog method. It will take as arguments a 2-dimensional array \([q_{n-1}, p_{n-1}]^\top\) and a stepsize \( h \) and return a 2-dimensional array \([q_n, p_n]^\top\).

function x2 = lpstep(x0,h)
% One step of the 4th order RK for Hamilton system
both $x_0$ and $x_1$ should be column vectors. 
$h$ is step size.

```matlab
M = 0.9953;
%% half kick
y0 = rhs(x0);
p1 = x0(2) + y0(2)*h/2;

%% drift
q2 = x0(1) + h*p1/M;

%% half kick
x1 = [q2;p1];
y1 = rhs(x1);
p2 = p1 + y1(2)*h/2;

%% output
x2 = [q2;p2];
```

### 3.4 Plot

Suppose the initial values are

\[
q(0) = 1.4155, \\
p(0) = 1.545 M/48.888.
\]

Run `RKstep` for 250 steps of size $h = 8$, and plot the deviation in the Hamiltonian from its initial value. Run `LFstep` for 1000 steps of size $h = 2, h = 2.3684, h = 2.3685$, and, for each of these, plot the deviation in the Hamiltonian.
Figure 1: Deviation in the Hamiltonian (RKstep, $h = 8, N = 250$)

Figure 2: Deviation in the Hamiltonian (RKstep, $h = 2, N = 100$)
Figure 3: Phase (RKstep, $h = 8, N = 250$)

Figure 4: Phase (RKstep, $h = 2, N = 100$)
Figure 5: Deviation in the Hamiltonian (LFstep, $h = 2, N = 1000$)

Figure 6: Deviation in the Hamiltonian (LFstep, $h = 2.3684, N = 1000$)
Figure 7: Deviation in the Hamiltonian (LFstep, $h = 2.3685$, $N = 1000$)

Figure 8: Phase (LFstep, $h = 2$, $N = 1000$) left: first 100 steps; right: overall
Figure 9: Phase (LFstep, $h = 2.3684, N = 1000$) left: first 100 steps; right: overall

Figure 10: Phase (LFstep, $h = 2.3685, N = 1000$) left: first 100 steps; right: overall
3.5 Comparison

How does the leapfrog method compare to Kutta’s Simpson method for equal work?

In each step of the Kutta’s Simpson method, the main computational cost is 4 times of $rhs$, while in each step of leapfrog method, the $rhs$ just need to compute once. In order to compare the efficiency, we should choose $h = 8$ in Kutta’s Simpson method while $h = 2$ in leapfrog method.

Please see the above figures. If $h = 8$ in Kutta’s Simpson method, then the numerical results are terrible; while leapfrog method with $h = 2$ is fine. Besides, I also plot the results obtained by Kutta’s Simpson method with $h = 2$, which are much better.

The numerical solution obtained by leapfrog method keep the Hamiltonian energy very well while the Kutta’s Simpson method does not.

3.6 Stepsize

What do you observe when the step size changes for the leapfrog method from $h = 2.3684$ to $h = 2.3685$?

Just a slight difference. See Figs.6-7 and Figs.9-10.