Analytical Mechanics: MATLAB

Shinichi Hirai

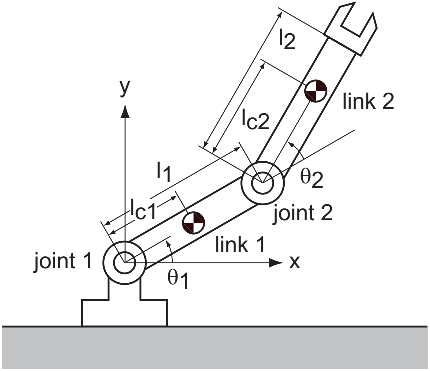
Dept. Robotics, Ritsumeikan Univ.

Agenda

1. Vector and Matrix
2. Graph
3. Ordinary Differential Equations
4. Optimization
5. Parameter Passing
6. Random Numbers
7. Summary

Problem

We drive a 2-DOF open loop menipulator based on joint PID control. Let us simulate the motion of the manipulator.



Problem

Step 1. Derive equations of motion (kinematics / dynamics)

Step 2. Numerically solve the derived equations of motion

Step 3. Describe the derived numerical solution by graphs or movies (visualization)

Step 4. Analyze the simulated motion

Problem

Soloving a set of simultaneous linear equations

$$\begin{bmatrix} H_{11} & H_{12} \\ H_{12} & H_{22} \end{bmatrix} \begin{bmatrix} \dot{\omega}_1 \\ \dot{\omega}_2 \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix}$$

$$\Downarrow$$

Solving a set of ordinary differential equations

$$\dot{\theta}_1 = \omega_1$$

$$\dot{\theta}_2 = \omega_2$$

$$\begin{bmatrix} \dot{\omega}_1 \\ \dot{\omega}_2 \end{bmatrix} = \begin{bmatrix} \cdots(\theta_1, \theta_2, \omega_1, \omega_2) \\ \cdots(\theta_1, \theta_2, \omega_1, \omega_2) \end{bmatrix}$$

What is MATLAB?

1. Software for numerical calculation
2. can handle vectors or matrices directly
3. Functions such as ODE solvers and optimization
4. Toolboxes for various applications
5. both programming and interactive calculation

What is MATLAB?

MATLAB environment

MATLAB Total Academic Headcount (TAH)
MATLAB with all toolboxes is available

Information

`https://it.support.ritsumei.ac.jp/hc/ja`

What is MATLAB?

- Install MATLAB into your own PC or mobile
- Sample programs are on the web of the class

Vector and Matrix

Column vector

```
x = [ 2; 3; -1 ];
```

Row vector

```
y = [ 2, 3, -1 ];
```

Matrix

```
A = [  4, -2,  1; ...
      -2,  5,  2; ...
      -2,  3,  2 ];
```

Vector and Matrix

Symbol ... implies that the sentense continues.

Column vector

```
x = [ 2; ...
      3; ...
     -1 ];
```

Column vector

```
x = [ 2; 3; -1 ];
```

Vector and Matrix

Multiplication

```
p = A*x;
q = y*A;
```

```
>> p

p =

     1
     9
     3

>>
```

Vector and Matrix

Multiplication

```
p = A*x;
q = y*A;
```

```
>> q

q =

     4     8     6

>>
```

Matrix operations

```
>> A

A =

     4    -2     1
    -2     5     2
    -2     3     2

>> A(3,2)

ans =

     3
```

Matrix operations

```
>> A
A =
     4    -2     1
    -2     5     2
    -2     3     2

>> A(3,2) = 6;

>> A
A =
     4    -2     1
    -2     5     2
    -2     6     2
```

Matrix operations

```
>> A(3,:)

ans =

    -2     3     2

>> A(:,2)

ans =

    -2
     5
     3
```

Matrix operations

```
>> A
A =
     4    -2     1
    -2     5     2
    -2     3     2

>> A(:,2) = [ 0; 2; 1 ];

>> A
A =
     4     0     1
    -2     2     2
    -2     1     2
```

Matrix operations

```
>> A
A =
     4    -2     1
    -2     5     2
    -2     3     2

>> A(3,:) = [ 3, -5, -1 ];

>> A
A =
     4    -2     1
    -2     5     2
     3    -5    -1
```

Matrix operations

```
>> A
A =
     4    -2     1
    -2     5     2
    -2     3     2

>> B = A([1,3],:);

>> B
B =
     4    -2     1
    -2     3     2
```

Matrix operations

```
>> A
A =
     4    -2     1
    -2     5     2
    -2     3     2

>> C = A(:,[2,1]);

>> C
C =
    -2     4
     5    -2
     3    -2
```

Basic row operations

`A(3,:) = 5*A(3,:);`	multiply the 3rd row by 5
`A(1,:) = A(1,:) + 4*A(2,:);`	add 4-times of the 2nd row to the 1st row
`A([3,1],:) = A([1,3],:);`	exchange the 1st and the 3rd rows

Solving simultaneous linear equation

```
A = [  4, -2,  1; ...
      -2,  5,  2; ...
      -2,  3,  2 ];
p = [ 1; 9; 3 ];
```

Solve a simultaneous linear equation $A\boldsymbol{x} = \boldsymbol{p}$

```
>> x = A\p;
>> x
x =
     2
     3
    -1

>> A*x

ans =
```

Solving simultaneous linear equation

- operator \ is general but less effective
- when coefficient matrix is positive-definite and symmetric, apply Cholesky decomposition
- inertia matrices are positive-definite and symmetric

Cholesky decomposition

positive-definite and symmetric matrix M can be decomposed as

$$M = U^{\mathrm{T}} U$$

where U is an upper trianglular matrix.

$$M\boldsymbol{x} = \boldsymbol{b} \quad \Longrightarrow \quad U^{\mathrm{T}} U \boldsymbol{x} = \boldsymbol{b} \quad \Longrightarrow \quad \left\{ \begin{array}{c} U^{\mathrm{T}} \boldsymbol{y} = \boldsymbol{b} \\ U\boldsymbol{x} = \boldsymbol{y} \end{array} \right.$$

Cholesky decomposition

program Cholesky.m

```
fprintf('Cholesky decomposition\n');

M = [ 4, -2, -2; ...
     -2,  2,  0; ...
     -2,  0,  3 ];

U = chol(M);
U

U'*U
```

Cholesky decomposition

```
>> Cholesky
Cholesky decomposition

U =
     2    -1    -1
     0     1    -1
     0     0     1

ans =
     4    -2    -2
    -2     2     0
    -2     0     3
```

Cholesky decomposition

program

```
b = [ 4; 2; -7 ];
y = U'\b;
x = U\y;
x
```

result

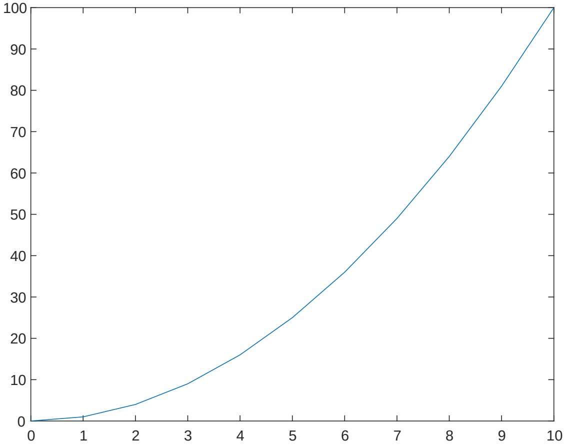
```
x =
     2
     3
    -1
```

Graph

```
>> x = [0:10]'
x =
     0
     1
     2
     3
    ...
>> f = x.*x
f =
     0
     1
     4
     9
    ...
```

Graph

```
>> plot(x,f)
```



Element-wise operations

Operators such as $.*$ and $./$ perform element-wise operation.

$$\begin{bmatrix} 2 \\ 5 \\ -3 \end{bmatrix} .* \begin{bmatrix} 3 \\ -1 \\ -3 \end{bmatrix} = \begin{bmatrix} 6 \\ -5 \\ 9 \end{bmatrix}$$

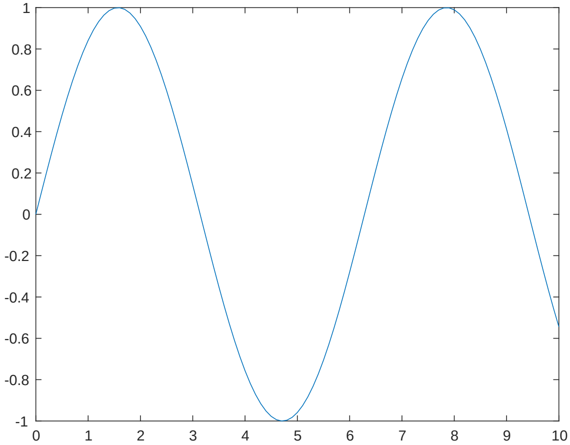
$$\begin{bmatrix} 6 \\ -5 \\ 1 \end{bmatrix} ./ \begin{bmatrix} 3 \\ -1 \\ 2 \end{bmatrix} = \begin{bmatrix} 2 \\ 5 \\ 1/2 \end{bmatrix}$$

Graph

```
>> t = [0:0.1:10]'
t =
         0
    0.1000
    0.2000
    0.3000
    ...
>> x = sin(t)
x =
         0
    0.0998
    0.1987
    0.2955
    ...
```

Graph

```
>> plot(t,x)
```



Vectorized functions

Functions such as cos, sin, exp, and log accept vectors as their arguments.

$$\sin \begin{bmatrix} 0 \\ \pi/6 \\ \pi/3 \end{bmatrix} = \begin{bmatrix} \sin(0) \\ \sin(\pi/6) \\ \sin(\pi/3) \end{bmatrix} = \begin{bmatrix} 0 \\ 1/2 \\ \sqrt{3}/2 \end{bmatrix}$$

$$\exp \begin{bmatrix} 0 \\ \log 2 \\ \log 3 \end{bmatrix} = \begin{bmatrix} \exp(0) \\ \exp(\log 2) \\ \exp(\log 3) \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

Graph

file draw_graph.m

```
t = [0:0.1:10]';
x = sin(t);
plot(t,x);
title(';time and position');    % title of the graph
xlabel('time');      % label of horizontal axis
ylabel('position'); % label of vertical axis
ylim([-1.5,1.5]);   % range of vertical axis
saveas(gcf,'draw_sine_graph.png');
        % save the graph to the specified file
```

runnig file draw_graph.m draws a graph and save the graph to an image file.

Solving Ordinary Differential Equations

van der Pol equation

$$\ddot{x} - 2(1 - x^2)\dot{x} + x = 0$$

$$\Downarrow$$

$$\begin{cases} \dot{x} = v \\ \dot{v} = 2(1 - x^2)v - x \end{cases}$$

$$\Downarrow$$

$$\boldsymbol{q} = \begin{bmatrix} x \\ v \end{bmatrix}, \qquad \dot{\boldsymbol{q}} = \boldsymbol{f}(t, \boldsymbol{q}) = \begin{bmatrix} v \\ 2(1 - x^2)v - x \end{bmatrix}$$

Solving Ordinary Differential Equations

File van_der_Pol.m describes function $\boldsymbol{f}(t, \boldsymbol{q})$

```
function dotq = van_der_Pol (t, q)
    x = q(1);
    v = q(2);
    dotx = v;
    dotv = 2*(1-x^2)*v - x;
    dotq = [dotx; dotv];
end
```

File name "van_der_Pol" should be consistent to function name "van_der_Pol".

Solving Ordinary Differential Equations

Program van_der_Pol_solve.m

```
interval = 0.00:0.10:10.00;
qinit = [ 2.00; 0.00 ];
[time, q] = ode45(@van_der_Pol, interval, qinit);
```

Solving Ordinary Differential Equations

Draw a graph of time t and variable x

```
plot(time, q(:,1), '-');
```

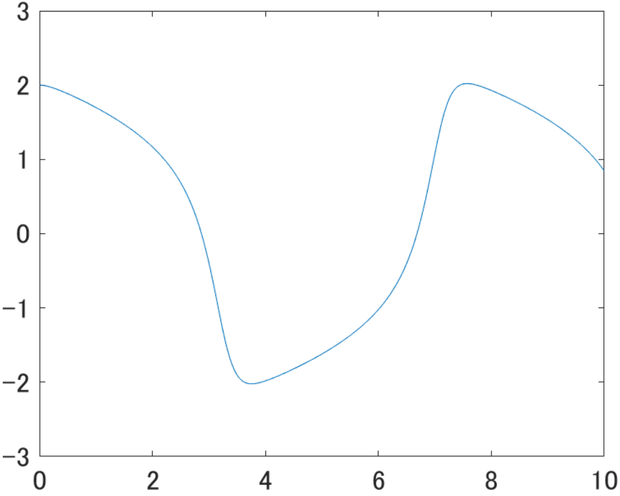
Draw a graph of time t and variable v

```
plot(time, q(:,2), '-');
```

'-'	solid line
'--'	broken line
'-.'	dot-dash line
':'	dotted line

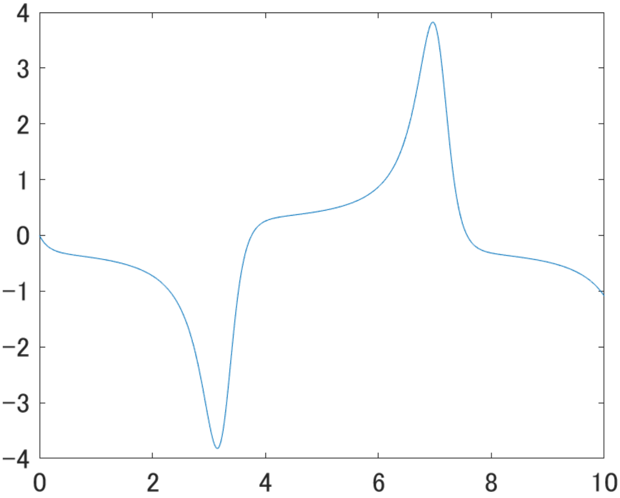
Solving Ordinary Differential Equations

graph of time t and variable x



Solving Ordinary Differential Equations

graph of time t and variable v



Optimization

Minimizing Rosenbrock function

$$\text{minimize} \quad f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

File Rosenbrock.m

```
function f = Rosenbrock( x )
    x1 = x(1); x2 = x(2);
    f = 100*(x2 - x1^2)^2 + (1 - x1)^2;
end
```

Optimization

File Rosenbrock_minimize.m

```
xinit = [ -1.2; 1.0 ];
[xmin, fmin] = fminsearch(@Rosenbrock, xinit);
xmin
fmin
```

Result

```
>> Rosenbrock_minimize
xmin =
    1.0000
    1.0000
fmin =
   8.1777e-10
```

ODE with Parameter

ordinary differential equation

$$\ddot{x} + b\dot{x} + 9x = 0$$

where b is a parameter

$$\Downarrow$$

$$\begin{aligned} \dot{x} &= v \\ \dot{v} &= -bv - 9x \end{aligned}$$

Global Variable

Function

```
    function dotq = damped_vibration (t, q)
        global b;
        x = q(1); v = q(2);
        dotx = v; dotv = -b*v - 9*x;
        dotq = [dotx; dotv];
    end
```

Program

```
global b;
interval = [0,10];
qinit = [2.00;0.00];
b = 1.00;
[time,q] = ode45(@damped_vibration,interval,qinit);
```

Nested Function

Function with arguments of time, state variable vector, and parameter

```
function dotq = damped_vibration_param (t, q, b)
    x = q(1); v = q(2);
    dotx = v; dotv = -b*v - 9*x;
    dotq = [dotx; dotv];
end
```

Program

```
interval = [0,10];
qinit = [2.00;0.00];
b = 1.00;
damped_vibration = @(t,q) damped_vibration_param (t,q,b);
[time,q] = ode45(damped_vibration,interval,qinit);
```

Global Variable vs Nested Function

Global Variable

Simple program
Global variables may conflict against local variables

Nested Function

Somewhat complicated
Must perform function defininion whenever paramerer values change
Never conflict with other variables

Uniform Random Numbers

Uniform Random Numbers in interval $(0, 1)$

```
rng('shuffle', 'twister');
for k=1:10
    x = rand;
    s = num2str(x);
    disp(s);
end
```

Symbol `'shuffle'` generates different random numbers whenever the program runs.

Uniform Random Numbers

Uniform Random Numbers in interval $(0, 1)$

```
rng(0, 'twister');
for k=1:10
    x = rand;
    s = num2str(x);
    disp(s);
end
```

specifying seed 0 generates unique random numbers whenever the program runs.

dice.m

```
function k = dice()
%   simulating a dice
    x = rand;
    if x < 1/6.00           k = 1;
    elseif x < 2/6.00       k = 2;
    elseif x < 3/6.00       k = 3;
    elseif x < 4/6.00       k = 4;
    elseif x < 5/6.00       k = 5;
    else                    k = 6;
    end
end
```

dice_run.m

```
for i=1:10
    s = [];
    for j=1:10
        k = dice();
        s = [s, ' ', num2str(k)];
    end
    disp(s);
end
```

dice_run.m

```
>> dice_run
 2 4 6 5 6 3 3 4 2 5
 4 4 2 1 3 3 5 5 5 1
 1 4 2 6 6 2 5 6 4 6
 6 4 5 4 1 3 4 4 3 6
 5 6 3 4 6 6 5 2 4 1
 3 5 6 5 3 5 3 6 6 6
 3 3 2 5 6 6 4 4 1 6
 3 2 6 5 6 2 5 4 1 3
 2 5 2 6 5 3 3 5 6 4
 4 2 3 5 6 5 1 5 3 3
>> dice_run
 1 5 2 2 3 3 4 4 3 3
 4 4 6 5 3 5 1 1 1 1
 2 2 1 4 1 1 4 6 6 4
 6 4 4 2 3 3 1 6 1 3
```

Summary

Numerical calculation using MATLAB

- linear calculation (vectors and matrices)
- solving simultaneous linear equations
- solving ordinary differential equations numerically
- optimization
- parameter passing
- random numbers