# The Adaptive Dynamic Programming Toolbox

User's Manual

Xiaowei Xing and Dong Eui Chang
{xwxing, dechang}@kaist.ac.kr

Dec. 29, 2020

Control Lab, School of Electrical Engineering
Korea Advanced Institute of Science and Technology


# Contents

# 1 Introduction

## 1.1 Overview

The Adaptive Dynamic Programming Toolbox (ADPT) is a software package written in MATLAB for optimal control problems. Based on adaptive dynamic programming, the ADPT generates approximate optimal feedback controls for linear and nonlinear systems in the continuous-time domain. The ADPT meets following properties:

- Flexibility: Two workings modes are provided: model-based mode and model-free mode. The model-based working mode deals with situations where the system model is known. In the model-free working mode, the ADPT takes measurements of system trajectories as input to produce the approximate optimal feedback control, removing the requirement of knowledge of system equations. Moreover, multiple options are provided for the both working modes such that the user can customize problems conveniently.

- Self-containedness: The ADPT is completely self-contained. That is, no external packages are required to work with the ADPT. Besides, two MATLAB files that contains information of the approximate optimal control and the approximate optimal cost function respectively would be generated after computations inside the ADPT. These two file can be applied in any MATLAB environment.

- User-friendliness: System equations can be easily declared using symbolic expressions. Moreover, the syntax of ADPT is quite intuitive and straight forward.

## 1.2 Installation

The ADPT is able to work on Windows, Linux and Mac. To use the ADPT, please make sure that the version of MATLAB is not earlier than R2008b.

**Remark.** Compatible testing has been done with MATLAB R2014b, MATLAB R2017a and MATLAB R2019a. However, there may be a problem using ADPT with MATLAB earlier than R2014b due to modifications of some functions. If an error occurs, please refer to Section 1.3.

**Remark.** Scripts may be different when using the ADPT with different MATLAB versions. For example, starting in R2016b, MATLAB scripts, including live scripts,

can contain code to define functions which are called local functions. Some comments about this change have been added in example codes. For more questions, please refer to Section 1.3.

Installation steps are as follows.

1. Download source files of ADPT at

   `https://github.com/Everglow0214/The_Adaptive_Dynamic_Programming_Toolbox`.

2. Suppose that the source files are saved:

   (a) in the folder `D:\username\adpt_src` on Windows, please type

   ```
   addpath D:\username\adpt_src
   savepath
   ```

   in the MATLAB command window;

   (b) or in the folder `/home/username/adpt_src` on Linux, please type

   ```
   addpath /home/username/adpt_src
   savepath
   ```

   in the MATLAB command window;

   (c) or in the folder `/Users/username/adpt_src` on Mac, please type

   ```
   addpath /Users/username/adpt_src
   savepath
   ```

   in the MATLAB command window.

## 1.3 Feedback

If you have problems using ADPT, please send an email to

`xwxing@kaist.ac.kr`

or

`dechang@kaist.ac.kr`

with the following:

1. The version of MATLAB, and the operating system you are using.
2. A minimal code example such that the problem can be reproduced.

# 2 Software Usages

## 2.1 Problem Setup

The Adaptive Dynamic Programming Toolbox is developed to solve optimal control problems where the system model is in the form of

$$\dot{x} = f(x) + g(x)u \tag{1}$$

with $x \in \mathbb{R}^n$ the state, $u \in \mathbb{R}^m$ the control, $f : \mathbb{R}^n \to \mathbb{R}^n$ and $g : \mathbb{R}^n \to \mathbb{R}^{n\times m}$ locally Lipschitz continuous mappings, and $f(0) = 0$, and the cost to be minimized is in the form of

$$J(x_0, u) = \int_0^\infty (q(x(t)) + u(t)^T R u(t))\ \mathrm{d}t \tag{2}$$

with $x_0 = x(0)$ the initial state, $q : \mathbb{R}^n \to \mathbb{R}_{\geq 0}$ a positive definite function, and $R = R^T \in \mathbb{R}^{m\times m}$ a positive definite matrix.

The objective is to find a control policy $u$ that minimizes $J(x_0, u)$ given $x_0$ in the following two cases. One is the model-based case, i.e., the system model (1) is known, and the other is the model-free case, i.e., the system model (1) is not known but data of system trajectories are available.

## 2.2 Working Modes

Two working modes are provided by ADPT: the model-based mode and the model-free mode. The knowledge of system equations is required in the model-based working mode. In the model-free working mode, the ADPT produces the approximate optimal feedback control from measurements of system trajectories, removing the requirement of knowledge of system dynamics. Both the working modes share the following principle.

For $x \in \mathbb{R}^n$, define the optimal control by

$$u^*(x) = \arg\min_u J(x, u)$$

and the optimal cost function by

$$V^*(x) = \min_u J(x, u).$$

Define the monomials of the state $x = (x_1, x_2, \cdots, x_n) \in \mathbb{R}^n$ by

$$
\begin{aligned}
&x_1, x_2, x_3, \cdots, x_n,\\
&x_1^2, x_1x_2, x_1x_3, \cdots, x_2^2, x_2x_3, x_2x_4, \cdots, x_n^2,\\
&x_1^3, x_1^2x_2, x_1^2x_3, \cdots, x_1^2x_n, x_1x_2^2, x_1x_2x_3, x_1x_2x_4, \cdots, x_1x_n^2, \cdots, x_n^3,\\
&\cdots.
\end{aligned}
$$

The approximate optimal control $\hat{u}^*$ and the optimal cost function $\hat{V}^*$ generated by ADPT are linear combinations of monomials of the state. The highest degree of monomials to represent $\hat{u}^*$ is referred to as the *approximation degree*. For $\hat{V}^*$, the highest degree of monomials is 1 larger than the approximation degree. For a pre-fixed approximation degree $d$, define

$$
\begin{aligned}
\Phi_d(x) &= (x_1, x_2, \cdots, x_n, x_1^2, x_1x_2, \cdots, x_n^d) \in \mathbb{R}^{N_1},\\
\Phi_{d+1}(x) &= (x_1, x_2, \cdots, x_n, x_1^2, x_1x_2, \cdots, x_n^{d+1}) \in \mathbb{R}^{N_2},
\end{aligned}
$$

where

$$
N_1 = \sum_{i=1}^{d} \binom{i+n-1}{n-1}, \quad N_2 = \sum_{i=1}^{d+1} \binom{i+n-1}{n-1}.
$$

Then in the ADPT $\hat{u}^*$ and $\hat{V}^*$ are represented by

$$
\hat{u}^*(x) = W\Phi_d(x), \tag{3}
$$

$$
\hat{V}^*(x) = \Phi_{d+1}(x)^T c, \tag{4}
$$

where $W \in \mathbb{R}^{m \times N_2}$ and $c \in \mathbb{R}^{N_1 \times 1}$ are composed of coefficients corresponding to the monomials in $\Phi_d(x)$ and $\Phi_{d+1}(x)$.

To obtain $W$ and $c$, the ADPT first generates some trajectories with some initial states under an initial control and some exploration signals by default in the model-based working mode, or is fed with trajectory measurements recorded by the user in the model-free working mode. Coefficient matrices $W$ and $c$ are then calculated in an iterative way by ADPT, after which two MATLAB files, `uAdp.p` and `VAdp.p`, that save information about $\hat{u}^*(x)$ and $\hat{V}^*(x)$ would be generated. Both the files take $x$, system state, as input argument, and return the feedback control $\hat{u}^*(x)$ and the cost $\hat{V}^*(x)$, respectively. These two files are independent of ADPT, and can be executed in any MATLAB environment.

## 2.3 Functions

### 2.3.1 Model-Based Working Mode

For this mode, generally the user only needs to define the system model, the cost function and the approximation degree before calling the function `adpModelBased`. Some options may be specified for `adpModelBased` by passing an extra argument which is returned by the function `adpSetModelBased`, as shown below.

```
[w,c,opt] = adpModelBased(f,g,x,n,u,m,q,R,t,d)
[w,c,opt] = adpModelBased(f,g,x,n,u,m,q,R,t,d,opt)
```

- Inputs
  - `f`: $f(x)$ in (1) (symbolic representation)
  - `g`: $g(x)$ in (1) (symbolic representation or constant matrix)
  - `x`: system state (symbolic representation with size $n \times 1$)
  - `n`: state dimension
  - `u`: control input (symbolic representation with size $m \times 1$)
  - `m`: control dimension
  - `q`: $q(x)$ in (2) (symbolic representation)
  - `R`: matrix $R$ in (2)
  - `t`: time (symbolic representation)
  - `d`: approximation degree
  - `opt`: (temporary) settings of the function `adpModelBased` (struct) `opt` is the output of the function `adpSetModelBased`.
- Outputs
  - `w`: matrix $W$ matrix in (3)
  - `c`: vector $c$ in (4)
  - `opt`: settings of the function `adpModelBased`
- Remark
  - Default settings would be applied if `opt` is not passed to `adpModelBased`.

```
opt = adpSetModelBased(n1,v1,n2,v2,...,n15,v15)
```

- Inputs
  - `n1,n2,...,n15`: names of options
  - `v1,v2,...,v15`: values to be assigned to `n1,n2,...,n15`
- Outputs
  - `opt`: (temporary) settings of the function `adpModelBased`
- Remark
  - The number of input arguments can be $0, 2, 4, \cdots, 30$.
- Supported options: name (default)
  - For initial states: `'xInit'` ([ ]), `'xInitNum'` (2), `'xInitMin'` (0.3), `'xInitMax'` (0.9), `'tSpan'` ($[0, 8]$), `'odeOpt'` (`odeset('RelTol',1e-6,'AbsTol',1e-6)`)
    - Two default initial states are generated randomly, with the value of each initial state in an open interval (`'xInitMin'`, `'xInitMax'`) or (−`'xInitMax'`, −`'xInitMin'`).
    - The user may choose the number (`'xInitNum'`) and the range (`'xInitMin'` and `'xInitMax'`) of initial states.
    - The user may also directly specify initial states (`'xInit'`) by a row vector or a matrix with each row indicating one initial condition.
    - Trajectories will be calculated starting from these initial states for a time interval $[0,$ `'tSpan'`$]$.
    - Trajectories are calculated using the Runge-Kutta method. Settings of the Runge-Kutta method can be modified by calling the function `odeset`.
  - For the initial control: `'u0Symb'` ([ ])
    - The default initial control is calculated from the linear quadratic method after linearization around the origin, with matrix $Q$ obtained from the Hessian matrix of $q(x)$ in (2).
    - The user may also directly specify the initial control (`'u0Symb'`) in a symbolic form.

- For exploration signals: `'explSymb'` ([ ]), `'explAmpl'` (0.8), `'numFreq'` (4)
  - Default exploration signals are sums of four sinusoidal signals with different frequencies.
  - The frequencies in the default exploration signals consist of rational numbers and irrational numbers and are chosen randomly from a predefined set.
  - The user may specify the number (`'numFreq'`) and the amplitude (`'explAmpl'`) of sinusoidal signals in one exploration signal.
  - The number of exploration signals automatically matches with the control dimension and the number of initial states.
  - All of these default exploration signals are different.
  - The user may also directly specify exploration signals (`'explSymb'`).
- For basis functions: `'basisOpt'` (`'mono'`)
  - Supported basis functions:
    monomials (`'mono'`)
- Others: `'stride'` (1), `'crit'` (1), `'epsilon'` (0.001), `'maxIter'` (100)
  - The user may specify data to be used in the iteration process inside ADPT by choosing the stride (`'stride'`) in each trajectory. For example, suppose two trajectories are generated as

    $$[x_{11}; x_{12}; \cdots; x_{1a}; x_{21}; x_{22}; \cdots; x_{2b}],$$

    where $x_{1\sim}$ indicates a value in the first trajectory and $x_{2\sim}$ indicates a value in the second trajectory. The user may only use $x_{11}, x_{13}, x_{15}, \cdots$ and $x_{21}, x_{24}, x_{27}, \cdots$ by setting `'stride'` to $[2; 3]$.
  - The stop criterion (`'crit'`) can be:
    0: $\|c_i - c_{i-1}\| \leq$ `'epsilon'` (the criterion used in [3]);
    1: $\|c_i - c_{i-1}\|^2 + \|W_i - W_{i-1}\|^2 \leq$ `'epsilon'`$^2$;
    0: $\|c_i - c_{i-1}\| \leq$ `'epsilon'` $\cdot \|c_{i-1}\|$;
    1: $\|c_i - c_{i-1}\|^2 + \|W_i - W_{i-1}\|^2 \leq$ `'epsilon'`$^2 \cdot (\|c_{i-1}\|^2 + \|W_{i-1}\|^2)$.
  - The maximum number of iterations (`'maxIter'`) can also be specified.

- Example

```
xInit = [ -3, 2;
           2.2, 3];
```

```
tSpan = [0, 10;
         0,  8];
opt = adpSetModelBased('xInit',xInit,'tSpan',tSpan);
```

### 2.3.2 Model-Free Working Mode

This mode deals with situations where the system model is unknown. After recording values of states, initial controls, exploration signals and timestamps, the user can call the function `adpModelFree` to calculate coefficients. Some options may also be specified for `adpModelFree` by passing an extra argument which is the output of the function `adpSetModelFree`, as shown below.

```
[w,c,opt] = adpModelFree(t,x,n,u0,m,e,d,q,R)
[w,c,opt] = adpModelFree(t,x,n,u0,m,e,d,q,R,opt)
```

- Inputs
  - `t`: time recorded (column vector)
    For every consecutive time interval, the starting time should be 0 or should be shifted to 0. For example, $t$ may be like
    $$[t_{11}; t_{12}; t_{13}; \cdots; t_{1a}; t_{21}; t_{22}; t_{23}; \cdots; t_{2b}],$$
    where $t_{1\sim}$and $t_{2\sim}$ are non-consecutive time intervals and $t_{11} = t_{21} = 0$.
  - `x`: state values recorded
    `x` is a column vector or a matrix, with each row indicating a state value at the corresponding time that is recorded in `t`.
  - `n`: state dimension
    `n` is same as the number of columns of `x`.
  - `u0`: control inputs recorded
    `u0` is a column vector or a matrix, with each row indicating a control value at the corresponding time that is recorded in `x`.
  - `m`: control dimension
    `m` is same as the number of columns of `u0`.
  - `e`: exploration signals recorded
    `e` is a column vector or a matrix, with each row indicating an exploration signal value at the corresponding time that is recorded in `t`. The number of columns of `e` is `m`.

  - `d`: approximation degree
  - `q`: $q(x)$ in (2) (function handle)
    For example, $q(x)$ can be defined as

```
q = @(x) 2*x(1)^2 + 3*x(2)^2;
```

  - `R`: matrix $R$ in (2)
  - `opt`: (temporary) settings of the function `adpModelFree` (struct)
    `opt` is the output of the function `adpSetModelFree`.

- Outputs
  - `w`: matrix $W$ matrix in (3)
  - `c`: vector $c$ in (4)
  - `opt`: settings of the function `adpModelFree`

- Remark
  - Default settings would be applied if `opt` is not passed to `adpModelFree`.

```
opt = adpSetModelFree(n1,v1,n2,v2,...,n5,v5)
```

- Inputs
  - `n1,n2,...,n5`: names of options
  - `v1,v2,...,v5`: values to be assigned to `n1,n2,...,n5`

- Outputs
  - `opt`: (temporary) settings of the function `adpModelFree`

- Remark
  - The number of input arguments can be $0, 2, 4, \cdots, 10$.

- Supported options: name (default)
  - For basis functions: `'basisOpt'` (`'mono'`)
    * Supported basis functions:
      monomials (`'mono'`)

- Others: `'stride'` (1), `'crit'` (1), `'epsilon'` (0.001), `'maxIter'` (100)
  - The user may specify data to be used in the iteration process inside ADPT by choosing the stride (`'stride'`) in each trajectory. For example, suppose two trajectories are generated as

    $$[x_{11}; x_{12}; \cdots; x_{1a}; x_{21}; x_{22}; \cdots; x_{2b}],$$

    where $x_{1\sim}$ indicates a value in the first trajectory and $x_{2\sim}$ indicates a value in the second trajectory. The user may only use $x_{11}, x_{13}, x_{15}, \cdots$ and $x_{21}, x_{24}, x_{27}, \cdots$ by setting `'stride'` to $[2; 3]$.
  - The stop criterion (`'crit'`) can be:
    0: $\|c_i - c_{i-1}\| \leq$ `'epsilon'` (the criterion used in [3]);
    1: $\|c_i - c_{i-1}\|^2 + \|W_i - W_{i-1}\|^2 \leq$ `'epsilon'`$^2$;
    0: $\|c_i - c_{i-1}\| \leq$ `'epsilon'` $\cdot \|c_{i-1}\|$;
    1: $\|c_i - c_{i-1}\|^2 + \|W_i - W_{i-1}\|^2 \leq$ `'epsilon'`$^2 \cdot (\|c_{i-1}\|^2 + \|W_{i-1}\|^2)$.
  - The maximum number of iterations (`'maxIter'`) can also be specified.

- Example

```
stride = [2;3];
opt = adpSetModelBased('stride',stride);
```

# 3 Examples

In this section we introduce some examples about how to use the Adaptive Dynamic Programming Toolbox. An application of the ADPT to a linear system in studied in Section 3.1. Applications to nonlinear systems are introduced in Section 3.2 and 3.3. Source codes for these examples are available in the folder `exmaples` at

`https://github.com/Everglow0214/The_Adaptive_Dynamic_Programming_Toolbox.git`.

The approximate optimal control is computed with the file `main.m` and is tested with the file `test.m`.

## 3.1 Turbocharged Diesel Engine System

Consider a turbocharged diesel engine system that can be described as [4]

$$\dot{x} = Ax + Bu, \tag{5}$$

where $x \in \mathbb{R}^6$ is the state, $u \in \mathbb{R}^2$ is the control, and

$$A = \begin{bmatrix} -0.4125 & -0.0248 & 0.0741 & 0.0089 & 0 & 0 \\ 101.5873 & -7.2651 & 2.7608 & 2.8068 & 0 & 0 \\ 0.0704 & 0.0085 & -0.0741 & -0.0089 & 0 & 0.0020 \\ 0.0878 & 0.2672 & 0 & -0.3674 & 0.0044 & 0.3962 \\ -1.8414 & 0.0990 & 0 & 0 & -0.0343 & -0.0330 \\ 0 & 0 & 0 & -359.0000 & 187.5364 & -87.0316 \end{bmatrix},$$

$$B = \begin{bmatrix} -0.0042 & 0.0064 \\ -1.0360 & 1.5849 \\ 0.0042 & 0 \\ 0.1261 & 0 \\ 0 & -0.0168 \\ 0 & 0 \end{bmatrix}.$$

The integral cost (2) is select with

$$q(x) = x^T Q x, \quad Q = \texttt{diag(1,1,0.1,0.1,0.1,0.1)}, \quad R = I_2.$$

Source codes for this example can be found in the folder

`examples/turbocharged_diesel_engine_system`.

The codes of using the ADPT to solve this optimal control problem in the model-based mode are shown in Listing 1. Since the solution to the linear quadratic problem is a linear state feedback controller, the approximation degree is set to 1 in line 1. The system model (5) is defined in lines 9 – 22 and the cost function is defined in lines 24 – 26. After specifying the options in lines 28 – 30, the function `adpModelBased` is called to return the coefficients $W$ and $c$ in line 32.

```
1  d = 1; % approximation degree
2  n = 6; % state dimension
3  m = 2; % control dimension;
4  % Symbolic variables.
5  x = sym('x',[n,1]);
6  u = sym('u',[m,1]);
7  t = sym('t');
8  % Define the system.
9  A = [-0.4125, -0.0248, 0.0741, 0.0089, 0, 0;
10      101.5873, -7.2651, 2.7608, 2.8068, 0, 0;
11      0.0704, 0.0085, -0.0741, -0.0089, 0, 0.0200;
12      0.0878, 0.2672, 0, -0.3674, 0.0044, 0.3962;
13      -1.8414, 0.0990, 0, 0, -0.0343, -0.0330;
14      0, 0, 0, -359.0000, 187.5364, -87.0316];
15 B = [-0.0042,  0.0064;
16      -1.0360,  1.5849;
17       0.0042,       0;
18       0.1261,       0;
19            0, -0.0168;
20            0,       0];
21 f = A * x;
22 g = B;
23 % Define the integral cost.
24 Q = diag([1,1,0.1,0.1,0.1,0.1]);
25 q = x'*Q*x;
26 R = eye(m);
27 % Specify options.
28 xInit = -3+6*rand(4,6);
29 tSpan = ones(4,1)*[0,10];
30 adpOpt = adpSetModelBased('xInit',xInit,'tSpan',tSpan);
31 % Execute ADP iterations.
```

```
32 [w,c] = adpModelBased(f,g,x,n,u,m,q,R,t,d,adpOpt);
```

Listing 1: Example codes for the model-based working mode.

For the model-free working mode, the codes are shown in Listing 2. The initial control $u_0$ is in the form of $u_0(x) = -Kx$ with the feedback gain $K$ defined in line 20. Exploration signals are composed of sinusoidal signals as shown in 21 – 26. Four random initial states are given in line 30, with the corresponding time span for simulation given in line 31, where the time interval $[0, 8]$ is divided into sub-intervals of size 0.002 so that trajectory data are recorded every 0.002 sec in lines 36 – 41. The timestamps are save in the column vector `t_save` in line 39, and the values of states are saved in the matrix `x_save` in line 40, with each row in `x_save` corresponding to the same row in `t_save`. Similarly, the values of the initial control and exploration signals are saved in matrices `u0_save` and `expl_save` in lines 42 – 43. These measurements are passed to the function `adpModelFree` to compute the approximate optimal control and the approximate optimal cost function in lines 57 – 58.

```
1  % Generate data.
2  x = sym('x',[6,1]);
3  t = sym('t');
4  % Define the system.
5  A = [-0.4125, -0.0248, 0.0741, 0.0089, 0, 0;
6       101.5873, -7.2651, 2.7608, 2.8068, 0, 0;
7       0.0704, 0.0085, -0.0741, -0.0089, 0, 0.0200;
8       0.0878, 0.2672, 0, -0.3674, 0.0044, 0.3962;
9       -1.8414, 0.0990, 0, 0, -0.0343, -0.0330;
10      0, 0, 0, -359.0000, 187.5364, -87.0316];
11 B = [-0.0042,  0.0064;
12      -1.0360,  1.5849;
13       0.0042,       0;
14       0.1261,       0;
15            0, -0.0168;
16            0,       0];
17 f = A * x;
18 g = B;
19 % Feedback gain of the initial control.
20 K = ones(2,6);
21 % Exploration signals.
22 e1 = 0.8*(sin(7*t)+sin(1.1*t)+sin(sqrt(3)*t)+...
```

```
  sin(sqrt(6)*t));
e2 = 0.8*(sin(2*t)+sin(2*pi*t)+sin(sqrt(10)*t)+...
  sin(11*t));
explSymb = [e1,e2];
expl = matlabFunction(explSymb,'Vars',t);
% To be used in the function ode45.
dx = matlabFunction(f+g*(-K*x+explSymb'),'Vars',{t,x});
xInit = -3+6*rand(4,6);
tSpan = ones(4,1)*[0:0.002:8];
odeOpt = odeset('RelTol',1e-6,'AbsTol',1e-6);
% Record data.
t_save = []; % timestamps
x_save = []; % states
for i = 1:size(xInit,1)
  [time,states] = ode45(@(t,x)dx(t,x),tSpan(i,:),...
    xInit(i,:),odeOpt);
    t_save = [t_save; time];
    x_save = [x_save; states];
end
u0_save = -x_save*K; % controls
expl_save = expl(t_save); % exploration signals
% Main part.
d = 1; approximation degree
n = 6; state dimension
m = 2; control dimension
Q = diag([1,1,0.1,0.1,0.1,0.1]);
q = @(x) x'*Q*x;
R = eye(m);
% Specify options.
stride = 3;
epsilon = 0.0001;
adpOpt = adpSetModelFree('stride',stride,'epsilon',...
  epsilon);
% Execute ADP iterations.
[w,c] = adpModelFree(t_save,x_save,n,u0_save,m,...
  expl_save,d,q,R,adpOpt);
```

Listing 2: Example codes for the model-free working mode.

## 3.2 Mass-Spring-Damper System

Consider a mass-spring-damper system given by [2]

$$\begin{aligned}\dot{x}_1 &= x_2,\\ \dot{x}_2 &= -\frac{k_1}{M}x_1 - \frac{k_2}{M}x_1^3 - \frac{b}{M}x_2 + \frac{1}{M}u,\end{aligned}$$

where $x_1$, $x_2$ and $M$ denote the position, velocity and mass of the object; $k_1$ and $k_2$ are linear and nonlinear stiffness of the spring; $b$ is the linear damping ratio of the damper; and $u$ is the control. The system is shown in Figure 1. Let $x = (x_1, x_2) \in \mathbb{R}^2$. Then the system is in the form of (1) with

$$f(x) = \begin{bmatrix} x_2 \\ \frac{1}{M}(-k_1x_1 - k_2x_1^3 - bx_2)\end{bmatrix}, \quad g(x) = \begin{bmatrix} 0 \\ \frac{1}{M}\end{bmatrix}.$$

Choose the cost in the form of (2) with

$$q(x) = 5x_1^2 + 3x_2^2, \quad R = 2,$$

and system parameters

$$k_1 = 3, \quad k_2 = 2, \quad b = 2, \quad M = 5.$$

This optimal control problem is solved by ADPT in both the model-based mode and model-free mode, with the codes available in the folder

`examples/mass_spring_damper_system`.

Structures of these codes are quite similar with those in Section 3.1 and are omitted here.

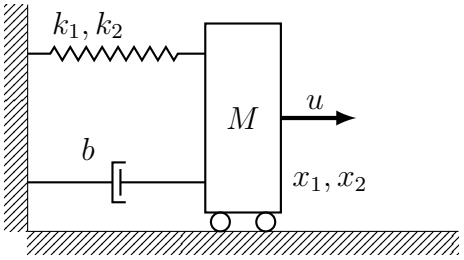


Fig. 1. The mass-spring-damper system.

## 3.3 Satellite Attitude System

The fully-actuated satellite system extended by stable embedding technique [1] are represented by quaternions as [5, 6]

$$\dot{q} = \frac{1}{2} q\Omega - \alpha(\|q\|^2 - 1)q, \tag{6}$$

$$\dot{\Omega} = \mathbb{I}^{-1}((\mathbb{I}\Omega) \times \Omega) + \mathbb{I}^{-1}u, \tag{7}$$

where $q$ is a quaternion which is extended to the the whole quaternion space from the unit quaternion space, representing the attitude of the satellite, $\Omega \in \mathbb{R}^3$ is the body angular velocity vector, $\mathbb{I} \in \mathbb{R}^{3\times 3}$ is the moment of inertial matrix, $u \in \mathbb{R}^3$ is the control input, and $\alpha > 0$ is a constant. The quaternion multiplication is carried out for $q\Omega$ on the right-hand side of (6) where $\Omega$ is treated as a pure quaternion.

Consider the problem of stabilizing the system (6) and (7) at the equilibrium point $(q_e, \Omega_e) = ((1, 0, 0, 0), (0, 0, 0))$. The error dynamics is given by

$$\dot{e}_q = \frac{1}{2}(e_q + q_e)e_\Omega - \alpha(\|e_q + q_e\|^2 - 1)(e_q + q_e),$$
$$\dot{e}_\Omega = \mathbb{I}^{-1}((\mathbb{I}e_\Omega) \times e_\Omega) + \mathbb{I}^{-1}u,$$

where $e_q = q - q_e$ and $e_\Omega = \Omega - \Omega_e$ are state errors. An optimal control problem is posed to design a stabilizing controller with the integral cost (2) with

$$q(x) = x^T Q x, \quad x = (e_q, e_\Omega) \in \mathbb{R}^7, \quad Q = 2I_7, \quad R = I_3.$$

The inertial matrix $\mathbb{I}$ is set to $\mathbb{I} =$ `diag(0.1029,0.1263,0.0292)`. The parameter $\alpha$ is set to $\alpha = 1$. This optimal control problem is solved by ADPT in the both working modes, with the codes available in the folder

`examples/satellite_attitude_system`.

Structures of these codes are quite similar with those in Section 3.1 and are omitted here.

# References

[1] D. E. Chang. On controller design for systems on manifolds in euclidean space. *Int. J. Robust Nonlinear Contr.*, 28:4981–4998, 2018.

[2] S. T. Glad. Control of nonlinear systems. URL: `https://www.control.isy.liu.se/student/graduate/nonlin/book.pdf`, 2009.

[3] Y. Jiang and Z.-P. Jiang. *Robust Adaptive Dynamic Programming*. John Wiley & Sons, Inc., Hoboken, NJ, USA, 2017.

[4] M. Jung, K. Glover, and U. Christen. Comparison of uncertainty parameterisations for $H_\infty$ robust control of turbocharged diesel engines. *Control Eng. Pract.*, 13:15–25, 2005.

[5] W. Ko. A stable embedding technique for control of satellite attitude represented in unit quaternions. Master's thesis, Korea Advanced Institute of Science and Technology, South Korea, 2020.

[6] W. Ko, K. S. Phogat, N. Petit, and D. E. Chang. Tracking controller design for satellite attitude under unknown constant disturbance using stable embedding. arXiv preprint, URL: `https://arxiv.org/abs/2010.01290`.