# LINEAR DYNAMICAL SYSTEMS REPORT

**School or Department:** Wu Han University

**Grade and Specialty:** Excellent engineer 2016

**Name:** Guo Yang, Kong Chuishun, Tang Rui

**Advisor:** Fangling Pu

May 7 2018

**【Abstract】**In this lab, we consider vector-valued linear and affine functions, and systems of linear equations. And we use a python program to familiar it. In the application, we introduce a method to train Binarized Neural Networks (BNNs) - neural networks with binary weights and activations.

**【Key words】** linear equation; BNN, Multidimensional linear function model

# Table of Contents

# Chapter 1 Introduction

## 1.1 Linear dynamical systems

Suppose $x_1,x_2,...$ is a sequence of n-vectors. The index (subscript) denotes time or period, and is written as t; $x_t$, the value of the sequence at time (or period) t, is called the state at time t. We can think of $x_t$ as a vector that changes over time, i.e., one that changes dynamically. In this context, the sequence $x_1,x_2,...$ is sometimes called a trajectory or state trajectory. We sometimes refer to $x_t$ as the current state of the system (implicitly assuming the current time is t), and $x_{t+1}$ as the next state, $x_{t-1}$ as the previous state, and so on.

The state $x_t$ can represent a portfolio that changes daily, or the positions and velocities of the parts of a mechanical system, or the quarterly activity of an economy. If $x_t$ represents a portfolio that changes daily, (x5)3 is the amount of asset 3 held in the portfolio on (trading) day 5.

A linear dynamical system is a simple model for the sequence, in which each xt+1 is a linear function of $x_t$:

$$x_{t+1} = A_t x_t, \quad t = 1, 2, .... \qquad (1.1)$$

Here the n×n matrices $A_t$ are called the dynamics matrices. The equation above is called the dynamics or update equation, since it gives us the next value of x, i.e., $x_{t+1}$, as a function of the current value $x_t$. Often the dynamics matrix does not depend on t, in which case the linear dynamical system is called time-invariant.

If we know $x_t$ (and $A_t,A_{t+1},...$) we can determine $x_{t+1},x_{t+2},...$ simply by iterating the dynamics equation (9.1). In other words: If we know the current value of x, we can find all future values. In particular, we do not need to know the past states. This is why xt is called the state of the system. It contains all the information needed at time t to determine the future evolution of the system.

## 1.2 Linear dynamical system with input.

There are many variations on and extensions of the basic linear dynamical system model (1.1), some of which we will encounter later. As an example, we can add additional terms to the update equation:

$$x_{t+1} = A_t x_t + B_t u_{t + ct} \quad t = 1, 2, .... \qquad (1.2)$$

Here ut is an m-vector called the input, Bt is the n × m input matrix, and the n-vector ct is called the offset, all at time t. The input and offset are used to model other factors that affect the time evolution of the state. Another name for the input ut is exogenous variable, since, roughly speaking, it comes from outside the system.

## 1.3 Markov model.

The linear dynamical system (1.1) is sometimes called a Markov model (after the mathematician Andrey Markov). Markov studied systems in which the next state value depends on the current one, and not on the previous state values $x_{t-1},x_{t-2},....$ The linear dynamical system (1.1) is the special case of a Markov system where the next state is a linear function of the current state. In a variation on the Markov model, called a (linear) K-Markov model, the next state xt+1 depends on the current state and K −1 previous states. Such a system has the form

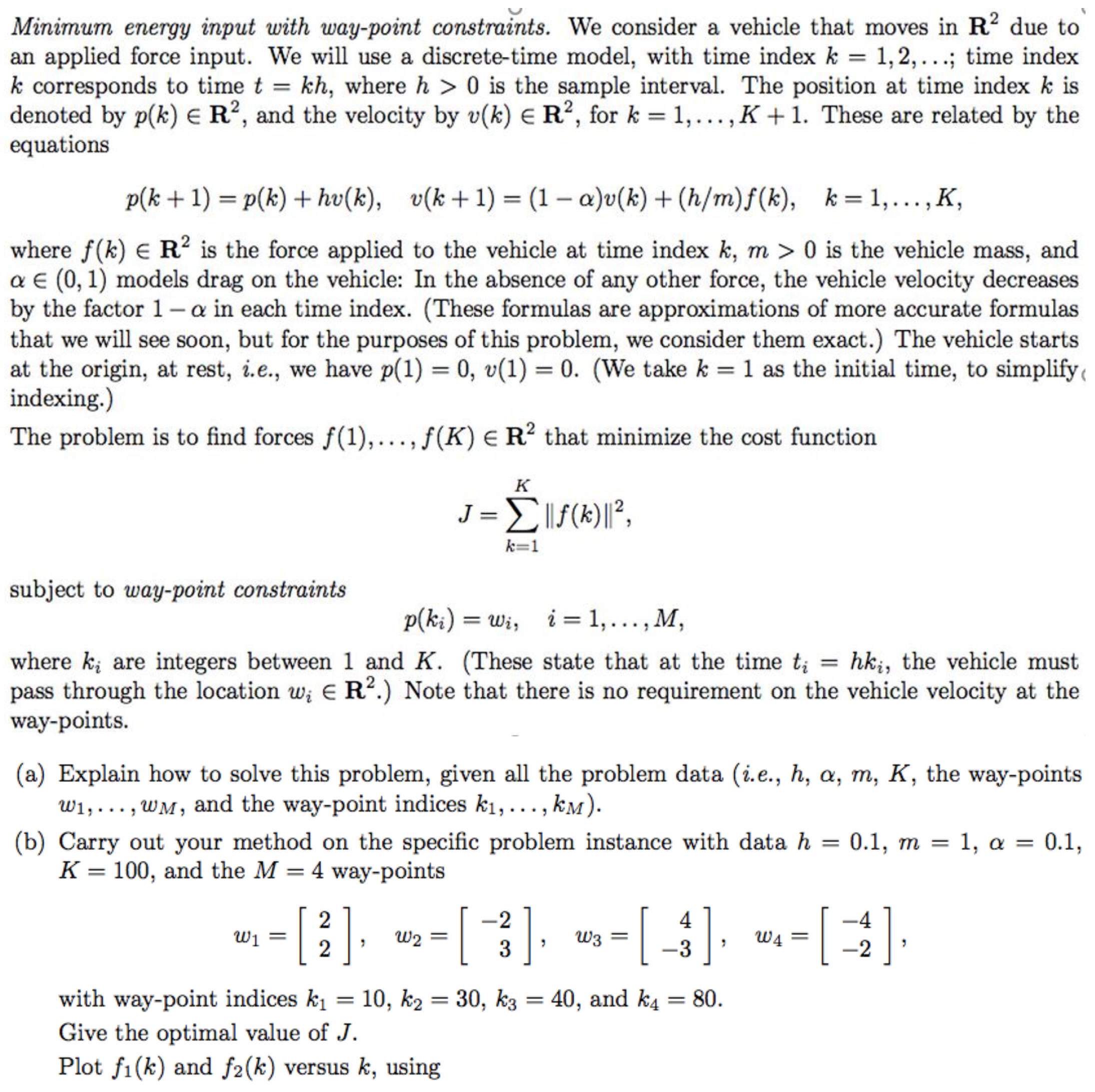
$$x_{t+1} = A_1 x_t + \cdots + A_K x_{t-K+1} \quad t = K, K+1, \ldots. \tag{1.3}$$

Models of this form are used in time series analysis and econometrics, where they are called (vector) auto-regressive models. When K = 1, the Markov model is the same as a linear dynamical system (1.1). When K > 1, the Markov model can be reduced to a standard linear dynamical system with an appropriately chosen state.

**1.4 Simulation.**

If we know the dynamics (and input) matrices, and the state at time t, we can find the future state trajectory $x_{t+1}$, $x_{t+2}$,... by iterating the equation (1.1) or (1.2), provided we also know the input sequence $u_t$, $u_{t+1}$,...). This is called simulating the linear dynamical system. Simulation makes predictions about the future state of a system. (To the extent that (1.1) is only an approximation or model of some real system, we must be careful when interpreting the results.) We can carry out what-if simulations, to see what would happen if the system changes in some way, or if a particular set of inputs occurs.

# Chapter 2 Question



*Minimum energy input with way-point constraints.* We consider a vehicle that moves in $\mathbf{R}^2$ due to an applied force input. We will use a discrete-time model, with time index $k = 1, 2, \ldots$; time index $k$ corresponds to time $t = kh$, where $h > 0$ is the sample interval. The position at time index $k$ is denoted by $p(k) \in \mathbf{R}^2$, and the velocity by $v(k) \in \mathbf{R}^2$, for $k = 1, \ldots, K+1$. These are related by the equations

$$p(k+1) = p(k) + hv(k), \quad v(k+1) = (1-\alpha)v(k) + (h/m)f(k), \quad k = 1, \ldots, K,$$

where $f(k) \in \mathbf{R}^2$ is the force applied to the vehicle at time index $k$, $m > 0$ is the vehicle mass, and $\alpha \in (0,1)$ models drag on the vehicle: In the absence of any other force, the vehicle velocity decreases by the factor $1-\alpha$ in each time index. (These formulas are approximations of more accurate formulas that we will see soon, but for the purposes of this problem, we consider them exact.) The vehicle starts at the origin, at rest, *i.e.*, we have $p(1) = 0$, $v(1) = 0$. (We take $k = 1$ as the initial time, to simplify indexing.)

The problem is to find forces $f(1), \ldots, f(K) \in \mathbf{R}^2$ that minimize the cost function

$$J = \sum_{k=1}^{K} \|f(k)\|^2,$$

subject to *way-point constraints*

$$p(k_i) = w_i, \quad i = 1, \ldots, M,$$

where $k_i$ are integers between 1 and $K$. (These state that at the time $t_i = hk_i$, the vehicle must pass through the location $w_i \in \mathbf{R}^2$.) Note that there is no requirement on the vehicle velocity at the way-points.

(a) Explain how to solve this problem, given all the problem data (*i.e.*, $h$, $\alpha$, $m$, $K$, the way-points $w_1, \ldots, w_M$, and the way-point indices $k_1, \ldots, k_M$).

(b) Carry out your method on the specific problem instance with data $h = 0.1$, $m = 1$, $\alpha = 0.1$, $K = 100$, and the $M = 4$ way-points

$$w_1 = \begin{bmatrix} 2 \\ 2 \end{bmatrix}, \quad w_2 = \begin{bmatrix} -2 \\ 3 \end{bmatrix}, \quad w_3 = \begin{bmatrix} 4 \\ -3 \end{bmatrix}, \quad w_4 = \begin{bmatrix} -4 \\ -2 \end{bmatrix},$$

with way-point indices $k_1 = 10$, $k_2 = 30$, $k_3 = 40$, and $k_4 = 80$.

Give the optimal value of $J$.

Plot $f_1(k)$ and $f_2(k)$ versus $k$, using

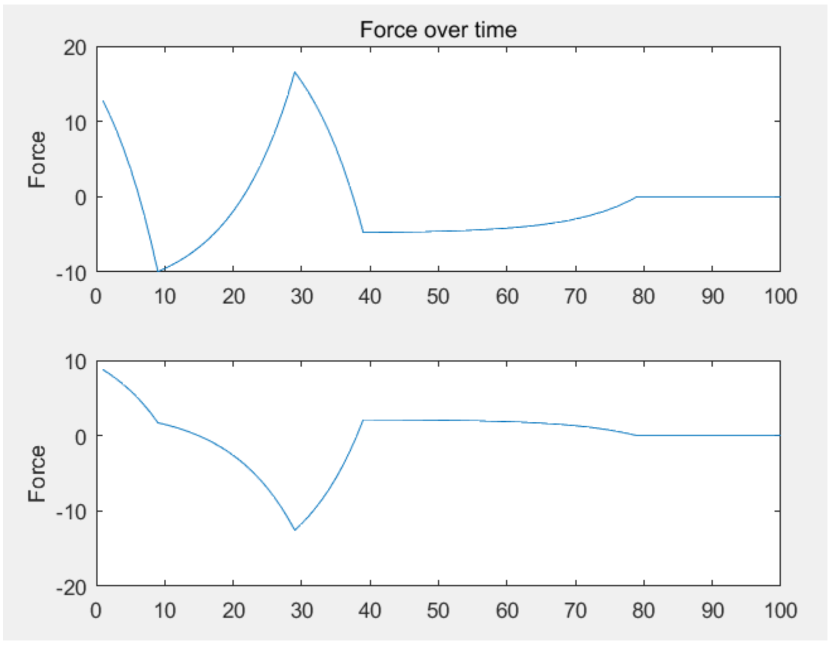
```
subplot(211); plot(f(1,:));
subplot(212); plot(f(2,:));
```

We assume here that f is a $2 \times K$ matrix, with columns $f(1), \ldots, f(K)$.
Plot the vehicle trajectory, using `plot(p(1,:),p(2,:))`. Here p is a $2 \times (K+1)$ matrix with columns $p(1), \ldots, p(K+1)$.

# Chapter 3 Solution

## 3.1 Specific operation

### 2.3 Analysis about Results



# Bibliography

[1]Bahdanau,Dzmitry,Cho,Kyunghyun,andBengio,Yoshua. Neural machine translation by jointly learning to align and translate. In ICLR'2015, arXiv:1409.0473, 2015.

# Appendix

*code:*

```
close all;
clear all;

h = .1;
```

```
a = .1;
K = 100;

w1 = [2; 2];
w2 = [-2; 3];
w3 = [4; -3];
w4 = [-4; -2];

k1 = 10;
k2 = 30;
k3 = 40;
k4 = 80;

A = [1 0 h 0; 0 1 0 h; 0 0 1-a 0; 0 0 0 1-a];
B = [0 0; 0 0; h 0; 0 h];

% A1
A1 = [];
for ki = 1:k1-1,
    A1 = [A1 A^(k1-1-ki)*B];
end
z1 = [zeros(1,200-2*(k1-1))];
z1 = [z1;z1;z1;z1];
A1 = [eye(2);0 0; 0 0]'*[A1 z1];

%A2
A2 = [];
for ki = 1:k2-1,
    A2 = [A2 A^(k2-1-ki)*B];
end
z2 = [zeros(1,200-2*(k2-1))];
z2 = [z2;z2;z2;z2];
A2 = [eye(2);0 0; 0 0]'*[A2 z2];


%A3
A3 = [];
for ki = 1:k3-1,
    A3 = [A3 A^(k3-1-ki)*B];
end
z3 = [zeros(1,200-2*(k3-1))];
z3 = [z3;z3;z3;z3];
A3 = [eye(2);0 0; 0 0]'*[A3 z3];
```

```
%A4
A4 = [];
for ki = 1:k4-1,
   A4 = [A4 A^(k4-1-ki)*B];
end
z4 = [zeros(1,200-2*(k4-1))];
z4 = [z4;z4;z4;z4];
A4 = [eye(2);0 0; 0 0]'*[A4 z4];


Aall = [A1;A2;A3;A4];
W = [w1;w2;w3;w4];

f = Aall'*inv(Aall*Aall')*W;
J = norm(f)^2

f1 = [];
f2 = [];
for i = 1:length(f),
   if (mod(i,2)) == 0,
      f2 = [f2; f(i)];
   else
      f1 = [f1; f(i)];
   end
end
f = [f1 f2];

subplot(211); plot(f(:,1)); title('Force over time');
ylabel('Force');
subplot(212); plot(f(:,2)); ylabel('Force');

pause;


% Plotting p
x = [0;0;0;0];
for i = 1:K,
   xnext = A*x(:,i) + B*[f(i,1); f(i,2)];
   x = [x xnext];
end

close all;
p = [eye(2);0 0; 0 0]'*x;
plot(p(1,:),p(2,:));
```

```
title('trajectory'); xlabel('p1'); ylabel('p2');
```