# Functional Mechanical Design

Exercise 1

## Motion Law

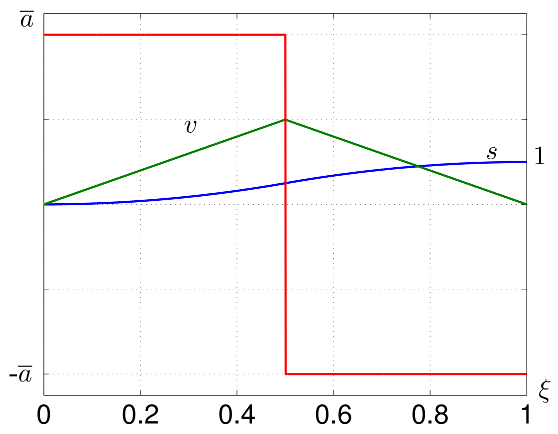

Figure 1: motion law with constant acceleration.

In the field of mechanical automation the motion law of the follower can be generally break up in a sequence of rising and falling intervals (with possible dwelling period), each of them characterized by a lift $h$ and its corresponding time of advancement $t_a$ (rising time or falling time).

A particular case that worth to be mentioned is that one which entails a single change of the acceleration sign of the motion law $y = y(t)$ within the domain of definition, satisfying the following boundary conditions:

$$y(0) = 0, \qquad \dot{y}(0) = 0;$$

$$y(t_a) = h, \qquad \dot{y}(t_a) = 0.$$

Concerning the study of the motion law that satisfy the aforementioned conditions, it is a good choice to highlight the influence of the shape of the law referring to adimensional (normalized) diagrams. In order to make the motion law adimensional with respect to time (or rotation) and lift it is convenient to divide the abscissa and the ordinate by the time $t_a$ and lift $h$, respectively. This way we can refer to the adimensional diagram $s(\xi)$ constrained to the motion law $y = y(t)$ by the following relationship:

$$y = hs\,, \qquad t = t_a\xi\,. \tag{1}$$

Where $s = s(\xi)$ is a function defined between $0 < \xi < 1$ with extreme values equal to: $s(0) = 0$ e $s(1) = 1$.

In the following practice you will be asked to write some Matlab functions receiving as input a value of $\xi$ bounded between 0 and 1 and output, in adimensional form, the corresponding value of position, velocity and acceleration of the considered motion law. In particular, 5 functions for the following motion laws need to be developed:

- motion law with constant acceleration and symmetric velocity profile (diagram shown in figure 1);
- trapezoidal velocity motion law;
- cycloidal motion law;
- cubic motion law;
- modified trapezoidal motion law[1]. Such motion law features an acceleration diagram made up of seven stages characterized by a constant acceleration in the second, fourth and sixth stage. On the other hand, the first, third, fifth and seventh stages are characterized by a sinusoidal acceleration (see figure 2).

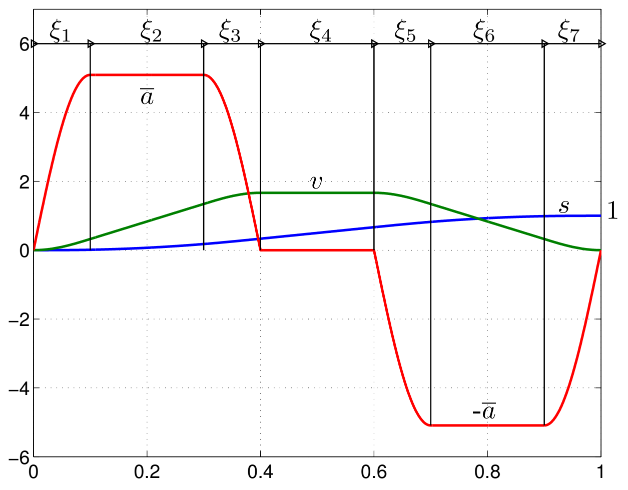

Figure 2: modified trapezoidal motion law.

In addition to the parameter $\xi$ the function should accept another input (preferably an array structure) with the parameters needed to describe the motion law in each stage. For example, in the modified trapezoidal motion law seven parameters are required to define the adimensional duration of each stage.

Eventually, the functions will be merged together to describe a dimensional motion law characterized by the following phases, considering the angle $\theta$ along the abscissa and the angle $\alpha$ on the ordinate:

- $0° < \theta < 60°$: dwell ($\alpha = 0°$)
- $60° < \theta < 120°$: rising ($\alpha$ starts from $0°$ and gets to $\alpha = 20°$ with cycloidal motion law)
- $120° < \theta < 180°$: dwell (constant angle of $\alpha = 20°$)
- $180° < \theta < 240°$: return ($\alpha$ return at the value 0 with a modified trapezoidal motion law)
- $240° < \theta < 360°$: dwell (constant angle of $\alpha = 0°$)

[1]All the needed formulas can be found starting from page 108 in the text book *Meccanismi per macchine automatiche*.

## Remarks

An example of a possible implementation of the Matlab script is shown below.
Main file:

```
close all
clc
clear all

%% choose between true and false:
%TRUE=COMPLEX MOTION LAW MADE UP OF ELEMENTARY PROFILES
%FALSE=SIMPLE MOTION LAW

if false

        MotionLaw = {'Dwell' 'CIC' 'Dwell' 'CIC' 'Dwell'};
    LiftH_deg = [0 20 0 -20 0];
    LiftH = deg2rad( LiftH_deg );
    Abscissa = [60 45 75 90 90]; % duration of each phase
else % Simple motion law
    MotionLaw = {'TVP'}; % Select one among TVP , ACS , CUB , CIC
    LiftH = pi;
    Time = 1;
    Abscissa=360;
end

N = 1000; % total number of discretization intervals
abscissa = linspace (0 ,sum( Abscissa ) ,N); % vector with N elements

% query the function
data = motionlaw(MotionLaw , LiftH , Abscissa , N);
figure
subplot(3,1,1) ;plot(abscissa,data.pos);
ylabel('Disp.[ rad ]');grid ;
title ('MotionLaw');
subplot(3,1,2); plot(abscissa,data.vel);
ylabel('Vel.[ rad/ab ]'); grid ;
subplot(3,1,3); plot(abscissa,data.acc);
ylabel('Acc.[ rad/ab\^2 ]');  grid ;
xlabel('Abscissa');
```

Function to be developed: (an example of trapezoidal velocity profile is shown here)

```
function [out] = motionlaw (MotionLaw , h , Ab, N)

% Input parameters
%
% MotionLaw = Identifier :
% 'ACS' : constant acceleration symmetric law
% 'TVP' : trapezoidal velocity profile
% 'CUB' : cubic law
% 'CIC' : cycloidal law
% 'Dwell ' : dwell
%
% 'h' = Lift
% 'Ta' = Time of advancement
% 'N' = Total number of discretization point s
%
% REMARK: The first three input s must be of the same size
%

if length(MotionLaw) ~=length(h) || length(h) ~= length(Ab)
error ('MotionLaw , h and Ta arrays must have the same size' );
end


% Discretization
t = linspace (0,sum(Ab),N);


% Memory allocation
out . pos = zeros (1 ,N);
out . vel = zeros (1 ,N);
out . acc = zeros (1  ,N);


% Set of instructions at each discretization point
for k = 1:N
        % we need to know on which stage we are by returning the index of the
        % abscissa      vector
        CurInt = find(cumsum(Ab)>=t(k),1,'first');

    % we need to calculate epsilon
    epsilon = (t(k) - sum(Ab(1:(CurInt-1)))) / Ab(CurInt);
    % switch case statement to select the needed law
    switch char (MotionLaw(CurInt))
    case {'ACS'}
        AdimLaw = ACS(epsilon);
    case {'TVP'}
        AdimLaw = TVP(epsilon);
    case {'CUB'}
        AdimLaw = Cubic(epsilon);
    case {'CIC'}
        AdimLaw = Cycloidal(epsilon);
    case {'Dwell'}
        AdimLaw = Dwell(epsilon);
    case {'ModTVP'}
        AdimLaw = ModTVP( epsilon,delta );
    otherwise
            error('s__motion_law_not_recognized',char(MotionLaw(CurInt)));
    end

% Output in terms of position , velocity and acceleration
 out.pos(k) = AdimLaw.s*h(CurInt);
 out.vel (k) =AdimLaw.v*h(CurInt)/Ab(CurInt);
 out.acc(k) =AdimLaw.a*h(CurInt)/Ab(CurInt)^2;


end
end










function [out] = TVP(epsilon)

 ev = 1/3;
 cv = 1/(1-ev);
 ca = 1/ev/(1-ev);


if 0 <= epsilon && epsilon <=ev
    out.a =  ca;
    out.v =  ca*epsilon;
    out.s =  1/2*ca*epsilon^2;
elseif ev < epsilon && epsilon <=(1-ev)
     out.a = 0;
     out.v = ca*ev;
     out.s = ca*ev*epsilon-1/2*ca*ev^2;
else
     out.a = -ca;
     out. v = ca*(1-epsilon);
     out. s = ca*(epsilon-epsilon^2/2+ev-ev^2-1/2);
end
end
```