مدل سازی سیستم مکانیکی فنر و دمپر

برای انجام این مدل سازی، ابتدا معادلات دینامیکی سیستم را مینویسیم. برای اینکار از قانون دوم نیوتن استفاده می کنیم؛ طبق این قانون برآیند نیروهای وارد بر جسم باید با حاصل ضرب جرم در شتاب آن مساوی باشد. پس اعمال این قانون بر هر جسم، برای بدست آوردن تابع تبدیل و فضای حالت دو راهکار پیش رو داریم:

۱. بدست آوردن معادلات حالت و سپس محاسبه تابع تبدیل از روی آن
۲. بدست آوردن تابع تبدیل و محاسبه ماتریس ها و معادلات حالت از روی آن.

ما از روش اول مدل سازی را انجام داده ایم. حال معادلات دینامیکی این دو جسم را می‌نویسیم:

$$\begin{cases} M_2\ddot{x}_1 = -D_2\dot{x}_1 - k_2x_1 - k_1(x_1 - x_2) - D_1(\dot{x}_1 - \dot{x}_2) \\ M_1\ddot{x}_2 = -k_1(x_2 - x_1) - D_1(\dot{x}_2 - \dot{x}_1) - f(t) \end{cases}$$

بعد از ساده‌سازی خواهیم داشت:

$$\begin{cases} \ddot{x}_1 = \dfrac{-(D_1 + D_2)\dot{x}_1 - (k_1 + k_2)x_1 + D_1\dot{x}_2 + k_1x_2}{M_2} \\[2ex] \ddot{x}_2 = \dfrac{D_1\dot{x}_1 + k_1x_1 - D_1\dot{x}_2 - k_1x_2 - f(t)}{M_1} \end{cases}$$

حال متغیرهای حالت را بصورت زیر تعریف می‌کنیم:

$$y = \begin{bmatrix} x_1 \\ \dot{x}_1 \\ x_2 \\ \dot{x}_2 \end{bmatrix},\ u(t) = f(t) \quad \rightarrow \quad \dot{y}(t) = Ay(t) + Bu(t)$$

و اما در نهایت معدلات حالت بصورت زیر خواهد بود:

$$\begin{cases} \dot{y} = \begin{bmatrix} \dot{x}_1 \\ \ddot{x}_1 \\ \dot{x}_2 \\ \ddot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{-k_1-k_2}{M_2} & \frac{-D_1-D_2}{M_2} & \frac{k_1}{M_2} & \frac{D_1}{M_2} \\ 0 & 0 & 0 & 1 \\ \frac{k_1}{M_1} & \frac{D_1}{M_1} & \frac{-k_1}{M_1} & \frac{-D_1}{M_1} \end{bmatrix} \times \begin{bmatrix} x_1 \\ \dot{x}_1 \\ x_2 \\ \dot{x}_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ -1 \end{bmatrix} f(t) \\ \\ z(t) = Cy(t) + Df(t) \end{cases}$$

و سپس با به دست آمدن ماتریس‌های حالت، می‌توان تابع تبدیل را بدست آورد:

$$G(s) = C \times (sI - A)^{-1} \times B + D$$

در ادامه راجع به مقدار بردار C و D توضیح داده شده است.

پیاده سازی مدل در متلب

برای انجام مدل‌سازی کامپیوتری، قاعدتا باید برای پارامترهای سیستم مانند جرم، ضریب سختی فنرها و ضریب دمپرها نیاز به مقداردهی داریم:

```
1  clc;clear;close all;
2  % Define system parameters
3  M1 = 1; M2 = 1; k1 = 1; k2 = 6; D1 = 0.5; D2 = 0.1;
4
```

در خط اول workspace متلب کاملا پاک کرده‌ایم. انجام این‌کار برای پیشگیری از تداخل متغیرهای قبلی موجود در متلب [که حاصل اجراهای قبلی اسکریپت‌ها می‌باشد] در محاسبات کنونی ما حیاتی است.

سپس در خط سوم جرم‌ها، ضرایب سختی فنرها و ضرایب دمپرها را مقداردهی کرده‌ایم. این قسمت بخش ورودی اسکریپت ما تلقی می‌شود که هرکس با تغییر آن‌ها می‌تواند خروجی‌ها را برای سیستم دلخواه خود بدست آورد.

```
inM1 = input('[default = 1] M1 = ', 's');
if ~isempty(inM1)
   M1 = str2double(inM1);
end
inM2 = input('[default = 1] M2 = ', 's');
if ~isempty(inM2)
   M2 = str2double(inM2);
end
in_k1 = input('[default = 1] k1 = ', 's');
if ~isempty(in_k1)
   k1 = str2double(in_k1);
end
in_k2 = input('[default = 6] k2 = ', 's');
if ~isempty(in_k2)
   k2 = str2double(in_k2);
end
inD1 = input('[default = 0.5] D1 = ', 's');
if ~isempty(inD1)
   D1 = str2double(inD1);
end
inD2 = input('[default = 0.1] D2 = ', 's');
if ~isempty(inD2)
   D2 = str2double(inD2);
end
```

و اما در صورتی که کاربر حین اجرا بخواهد، مقدار متفاوتی نسبت به مقادیر پیشفرضی که در کد در نظر گرفته شده وارد کند ما خطوط ۵ الی ۲۸ را نوشته ایم. تمام کاری که این بخش می‌کند این است که به ازای پارامتر های مختلف سیستم، از کاربر ورودی دریافت می‌کند، در صورتی که کاربر ورودی وارد نکند(صرفا Enter را فشار دهد) مقدار پیش فرض پارامتر مربوطه تغییری نمی‌کند؛ در غیر اینصورت اگر کاربری عدد جدیدی وارد کرد، مقدار پارامتر با مقدار ورودی کاربر جایگزین می‌شود.

```
30 -    fprintf('Solve for this inputs: M1=%.1f, M2=%.1f, ', M1, M2, k1, k2, D1, D2);
31 -    fprintf('k1=%.1f, k2=%.1f, D1=%.1f, D2=%.1f :\n\n', k1, k2, D1, D2);
```

و نهایتا در این دو خط، مقدار نهایی تمام پارامترهای مسئله ثبت می‌شود. صرفا مقادیر این پارامترها را چاپ کرده ایم؛ اینکار صرفا برای این است که پس از اجرای برنامه، Command Window متلب تمام اطلاعات مربوطه به سوال را نمایش دهد و تاثیری در محاسبات ما ندارد.

```
33 -    f = @(t) abs(sin(t));
34 -    inF = input('{ default: f(t) = |sin(t)| }  f(t) = ', 's');
35 -    if ~isempty(inF)
36 -       f = str2func(['@(t)' inF]);
37 -    end
38 -    display(f);
```

با توجه به اینکه سیستم یک ورودی f(t) هم دارد، در خطوط ۳۳ الی ۳۸ ما تابع f را هم تغیعریف کرده ایم. همانند ورودی های قبلی کاربر میتواند از تابع پیشفرض استفاده کند (از طریق نادیده گرفتن ورودی) و یا تابع موردنظر خود را بر حسب t وارد کند. در انتها در خط ۳۸ تابع نهایی چاپ می‌شود.

```
40 -    A = [0, 1, 0, 0;
41          -(k1+k2)/M2, -(D1+D2)/M2, - k1/M2, -D1/M2;
42          0, 0, 0, 1;
43          k1/M1, D1/M1, -k1/M1, -D1/M1
44      ];
45 -    B = [0; 0; 0; -1/M1];
46 -    C = [1, 0, 0, 0];
47 -    D = 0;
48      % Display state space matrices
49 -    disp('State Space Matrices:');
50 -    display(A);
51 -    display(B);
52 -    display(C);
53 -    disp(D);
```

حال در خطوط ۴۰ الی ۴۵، ماتریس های A و B حالت را طبق محاسبات ابتدایی و معادلات حالت بدست آمده مینویسیم. در خط ۴۶ و ۴۷ ماتریس های حالت C و D را بدست می اوریم؛ مقدار این دو ماتریس وابسته به این است که ما کدام متغیر را بعنوان خروجی می‌خواهیم. برای مثال ما در اینجا خروجی را برابر با جابه‌جایی جسم ۱ (x_1) در نظره گرفته ایم و با توجه به این‌که ورودی بصورت مستقیم در خروجی دلخواه ما (x_1) تاثیر ندارد، پس داریم:

$$y = (x_1, \dot{x}_1, x_2, \dot{x}_2)$$

$$\begin{cases} \dot{y} = Ay + Bu \\ z = Cy + Du \end{cases}, \qquad z = y_1 \rightarrow \begin{cases} C = [1\ 0\ 0\ 0] \\ D = 0 \end{cases}$$

که این همان خطوط ۴۶ و ۴۷ کد ماست؛ در خطوط ۴۹ الی ۵۳ هم ما این ماتریس ها را نمایش می-دهیم.

```
55 -     disp('Approach 1. Use matlab ss2tf function: ');
56 -     disp('Approach 2. Transfer Matrix is ontained via: G(s) = C * (sI - A) \ B + D');
57 -     choice = input('Which approach do you want to use? (Enter 1 or 2) ');
58 -     while choice ~= 1 || choice ~= 2
59 -         choice = input('Wrong choice; Please just Enter 1 or 2: ');
60 -     end
```

حالا نوبت به محاسبه تابع تبدیل از روی ماتریس‌های حالت می‌رسد. برای این کار هم از دو روش می-تونیم استفاده کنیم:

۱. استفاده از تابع آماده متلب به نام: ss۲tf
۲. استفاده از فرمول ریاضی مربوطه که به صورت زیر می‌باشد:

$$G(s) = C \times (sI - A)^{-1} \times B + D$$

دوباره در این باره ما به کاربر حق انتخاب می‌دهیم؛ در خطوط ۵۵ الی ۶۰ از کاربر می‌خواهیم یکی از این دو روش را انتخاب کند. خطوط ۵۸ الی ۶۰ برای این است که اطمینان حاصل کنیم کاربر ورودی اشتباهی وارد نکند. یعنی تا زمانی که کاربر یکی از گزینه های '۱' یا '۲' را وارد نکند، دریافت ورودی تکرار می‌شود.

```
    G_s = [];
    if choice == 1
        [denominator, numerator] = ss2tf(A, B, C, D);
        display(numerator);
        display(denominator);
        G_s = tf(numerator, denominator);
        display(G_s);
    elseif choice == 2
        n, m = size(A);  % for constructing I we need dimension of A
        s = tf('s');
        sI_A = (s*eye(n) - A);
        disp('sI - A = ');
        disp(sI_A);
        disp('(sI - A) ^ -1 = ');
        disp(inv(sI_A));
        % inv_sI_A_cross_B = inv(sI_A) * B;  % this line could be used, but
        % MatLab says too: This method is slower and less accurate than the
        % next line approach:
        inv_sI_A_cross_B = sI_A \ B;
        disp('C * [(sI - A)^-1] * B \ ');
        G_s0 = C * inv_sI_A_cross_B;
        disp(G_s0);
        G_s = G_s0 + D;
        display(G_s);
    end
```

حال با توجه به انتخاب کاربر، برنامه طبق یکی از روش های مذکور، تابع تبدیل را محاسبه می‌کند. محاسبات این بخش بصورت گام به گام انجام شده است. مثلا در روش دوم، تمام مراحل ضرب و جمع ماتریسی، مرحله به مرحله چاپ می‌گردد.

```
    % Define model functions (state-space equations)
    ode = @(t, y) [
        y(2);
        (-(k1+k2)*y(1) - (D1+D2)*y(2) - k1*y(3) - D1*y(4))/M2;
        y(4);
        (k1*y(1) + D1*y(2) - k1*y(3) - D1*y(4) - f(t))/M1
    ];

    % Initial conditions
    initial_conditions = [0; 0; 0; 0];

    % Simulation time span
    t_i = input('Specify the start time of the simulation [usually its Zero]: ');
    t_f = input('Specify the final time of the simulation: ');
    while t_i < 0 || t_f < 0 || t_f <= t_i
        disp('Please provide valid time data; conditions: t_i >= 0, t_f >= 0, t_f > t_i');
        t_i = input('t_i = ');
        t_f = input('t_f = ');
    end
    t_span = [t_i t_f];

    % Solve state-space equations
    [t, Y] = ode45(ode, t_span, initial_conditions);
```

در خطوط ۸۹ الی ۱۰۷ ما مشخصات لازم را برای حل معادلات حالت سیستم مربوطه فراهم می آوریم. ماتریس ode مشخص کننده معادلات حالت است که ابتدای کار از طریق معادلات دینامیکی بدست آوردیم. بردار initial_conditions همانطور که از نامش بر می‌آید شرایط اولیه معادلات را مشخص می‌کند. این شرایط اولیه درواقع همان مکان و سرعت اولیه در لحظه $t=t_i$ برای هر دو جسم می‌باشند.

سپس بردار زمانی را تعریف کردیم به نام t_span؛ عنصر اول آن زمان شروع شبیه‌سازی و درایه دوم آن هم زمان پایانی است.

در نهایت در خط ۱۱۰ از طریق تابع ode۴۵ معادله را به ازای ورودی‌های دریافت شده برای سیستم حل می‌کنیم؛ این تابع در خروجی به ما مقادیر مکان و سرعت هر جسم را بصورت جداگانه می‌دهد که می‌توانیم از این مقادیر برای رسم استفاده کنیم:

```
112      % Display results
113 -    figure;
114 -    x1 = Y(:, 1);
115 -    dx1 = Y(:, 2);
116 -    x2 = Y(:, 3);
117 -    dx2 = Y(:, 4);
118
119 -    plot(t, x2, 'r', t, dx1, 'b', t, x2, 'g', t, dx2, 'm');
120 -    legend('x1', 'dx1', 'x2', 'dx2');
121 -    xlabel('Time');
122 -    ylabel('State');
123 -    title('System Response to Step Input');
```

گفتیم که خروجی تابع ode۴۵ مقادیر سرعت و مکان هر جسم در زمان تعیین شده t_span می‌باشد. در خطوط ۱۱۳ الی ۱۱۷ این مقادیر را در متغیرهای مربوطه ذخیره میکنیم و سپس در خطوط ۱۱۹ الی ۱۲۳ تمام این مولفه‌ها در در یک figure نمودار رسم می‌کنیم.

```
% Plot step, ramp and pulse responses:
%input for ramp plot

figure,
step(G_s),
hold on,
impulse(G_s),
legend('Step Response', 'Impulse Response'),
title('Step and Impulse responses');
```

همانند قبل اینجا هم رسم انجام می‌دهیم؛ اما این‌بار از تابع تبدیل استفاده میکنیم. خط ۱۲۹ پاسخ پله را با استفاده از step و خط ۱۳۱ پاسخ ضربه را با استفاده از impulse رسم میکند.

```
135 -    t = t_i:.1:t_f;
136 -    u = sin(t);
137 -    figure,
138 -    lsim(G_s, u, t, initial_conditions), legend('Ramp'), title('Ramp Response');
```

حال پاسخ ورودی شیب این سیستم را رسم می‌کنیم؛ با این تفاوت که اینجا نیاز به تعریف یک ورودی هم داریم که در خط ۱۳۶ تعریف شده است.

```
140      % Compute the eigenvalues (roots) of the system
141 -    eigenvalues = eig(A);
142 -    disp('Here are the eigen values of A:');
143 -    display(eigenvalues);
144      % Plot the poles (roots) on the complex plane
145 -    figure;
146 -    rlocus(G_s);
147 -    title('Pole-Zero Map');
```

و در نهایت در خط ۱۴۱ الی ۱۴۳ مقادیر ویژه ماتریس حالت A را بدست می‌آوریم و در ادامه خطوط هم توسط تابع rlocus مکان هندسی تابع تبدیل G_s را رسم میکنیم.