07.7 ssresp.sim Simulating state-space response

1 Ahem.[7]

For many nonlinear models, numerical solution of the state equation is required. For linear models, we can always solve them analytically using the methods of this chapter. However, due to its convenience, we will often want to use numerical techniques even when analytic ones are available. Matlab has several built-in and *Control Systems Toolbox* functions for analyzing state-space system models, especially *linear* models. We'll explore a few, here.

Consider, for instance, a linear state model with the following A, B, C, and D matrices:

$$A = \begin{bmatrix} -3 & 4 & 5 \\ 0 & -2 & 3 \\ 0 & -6 & 1 \end{bmatrix} \quad B = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \quad C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix} \quad D = \begin{bmatrix} 0 \\ 0 \end{bmatrix}. \tag{1a}$$

```
A = [-3,4,5;0,-2,3;0,-6,1];
B = [1;0;1];
C = [1,0,0;0,-1,0];
D = [0;0];
```

For a step input $u(t) = 3u_s(t)$ and initial state $x(0) = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}^\top$, let's compare analytic and numerical solutions for the output response $y(t)$.

```
u = @(t) 3*ones(size(t)); % for t>=0
x_0 = [1; 2; 3];
```

[7]The source of this lecture can be downloaded as a Matlab m-file at http://ricopic.one/dynamic_systems/source/simulating_state_space_response.m.

Analytic solution

For an analytic solution, we'll use a rearranged version of **??**.[8]

$$\mathbf{y}(t) = C\Phi(t)\mathbf{x}(0) + C\Phi(t)\int_0^t \Phi(-\tau)B\mathbf{u}(\tau)d\tau + D\mathbf{u}(t). \tag{2a}$$

First, we need the state transition matrix $\Phi(t)$, so we consider the eigenproblem.

```
[M,L] = eig(A)
```

```
M =

   1.0000 + 0.0000i   0.7522 + 0.0000i   0.7522 + 0.0000i
   0.0000 + 0.0000i   0.3717 + 0.0810i   0.3717 - 0.0810i
   0.0000 + 0.0000i   0.0787 + 0.5322i   0.0787 - 0.5322i


L =

  -3.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i  -0.5000 + 3.9686i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i  -0.5000 - 3.9686i
```

Note that, when assigning its output to two variables `M` and `L`, the `eig` function returns the modal matrix to `M` and the eigenvalue matrix to `L`. The modal matrix of eigenvectors `M` has each column (eigenvector) normalized to unity. Also notice that `M` and `L` are *complex*. The imaginary parts of two eigenvalues and their corresponding eigenvectors are significant. Finally, since the *real* parts of the all eigenvalues are *negative*, the system is stable. The "diagonal"-basis state transition matrix $\Phi'(t)$ is simply

$$\Phi'(t) = e^{\Lambda t}. \tag{3}$$

Let's define this as an "anonymous" function.

[8]Although we call this the "analytic" solution, we are not solving for a detailed symbolic expression, although we *could*. In fact, Eq. 2 *is* the analytic solution and what follows is an attempt to represent it graphically.

```
Phi_p = @(t) diag(diag(exp(L*t))); % diags to get diagonal mat
```

The original-basis state transition matrix $\Phi(t)$ is, from **??**,

$$\Phi(t) = M\Phi'(t)M^{-1}. \tag{4}$$

```
M_inv = M^-1; % compute just once, not on every call
Phi = @(t) M*Phi_p(t)*M_inv;
```

Free response

The free response is relatively straightforward to compute.

```
t_a = 0:.05:5; % simulation time
y_fr = NaN*ones(size(C,1),length(t_a)); % initialize
for i = 1:length(t_a)
    y_fr(:,i) = C*Phi(t_a(i))*x_0;
end
y_fr(:,1:3) % first three columns
```

```
ans =

   1.0000 - 0.0000i   1.8922 - 0.0000i   2.5646 - 0.0000i
  -2.0000 + 0.0000i  -2.2030 + 0.0000i  -2.3105 + 0.0000i
```

A time array `t_a` was defined such that `Phi` could be evaluated. The first three columns of y_{fr} are printed for the first three moments in time. Note how there's a "hanging chad" of imaginary components. Before we `realize` them, let's make sure they're negligibly tiny.

```
max(max(abs(imag(y_fr))))
y_fr = real(y_fr);
```

```
ans =

   5.2907e-16
```

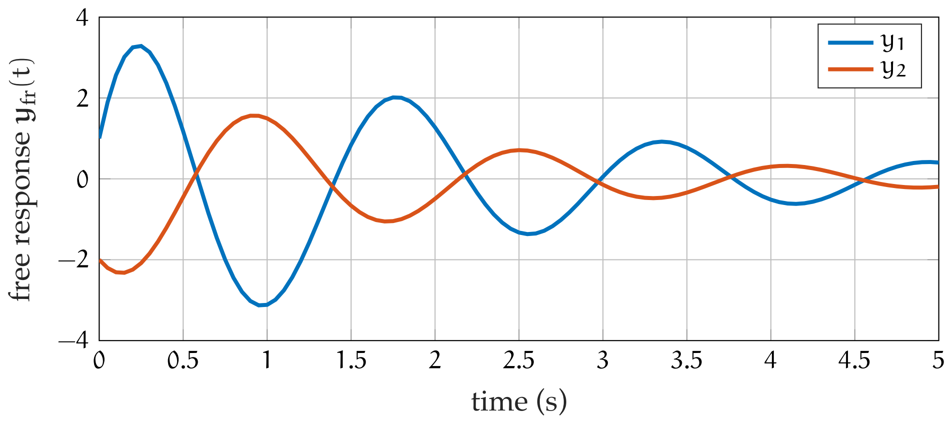

Figure sim.1: free response y_{fr}.

The results are plotted in Fig. sim.1. As we might expect from the eigenvalues, the free responses of both outputs oscillate and decay.

Forced response

Now, there is the matter of integration in Eq. 2. Since Matlab does not excel in symbolic manipulation, we have chosen to avoid attempting to write the solution, symbolically.[9] For this reason, we choose a simple numerical (trapezoidal) approximation of the integral using the `trapz` function. First, the integrand can be evaluated over the simulation interval.

```
integrand_a = NaN*ones(size(C,2),length(t_a)); % initialize
for i = 1:length(t_a)
    tau = t_a(i);
    integrand_a(:,i) = Phi(-tau)*B*u(tau);
end
```

Now, numerically integrate.

```
integral_a = zeros(size(integrand_a));
for i = 2:length(t_a)
    i_up = i; % upper limit of integration
```

[9]Mathematica or SageMath would be preferrable for this.

```
    integral_a(:,i) = ... % transposes for trapz
        trapz(t_a(1:i_up)',integrand_a(:,1:i_up)')';
end
```

Now, evaluate the forced response at each time.

```
y_fo = NaN*ones(size(C,1),length(t_a)); % initialize
for i = 1:length(t_a)
    y_fo(:,i) = C*Phi(t_a(i))*integral_a(:,i);
end
y_fo(:,1:3) % first three columns
```

```
ans =

   0.0000 + 0.0000i   0.1583 - 0.0000i   0.3342 - 0.0000i
   0.0000 + 0.0000i  -0.0109 + 0.0000i  -0.0426 + 0.0000i
```

```
max(max(abs(imag(y_fo))))
y_fo = real(y_fo);
```

```
ans =

   2.1409e-16
```

The forced response is shown in Fig. sim.2, which shows damped oscillations.

Total response

The total response is found from the sum of the free and forced responses: $y(t) = y_{\text{fr}} + y_{\text{fo}}$. We can simply sum the arrays.

```
y_t = y_fr + y_fo;
```

The result is plotted in Fig. sim.3.

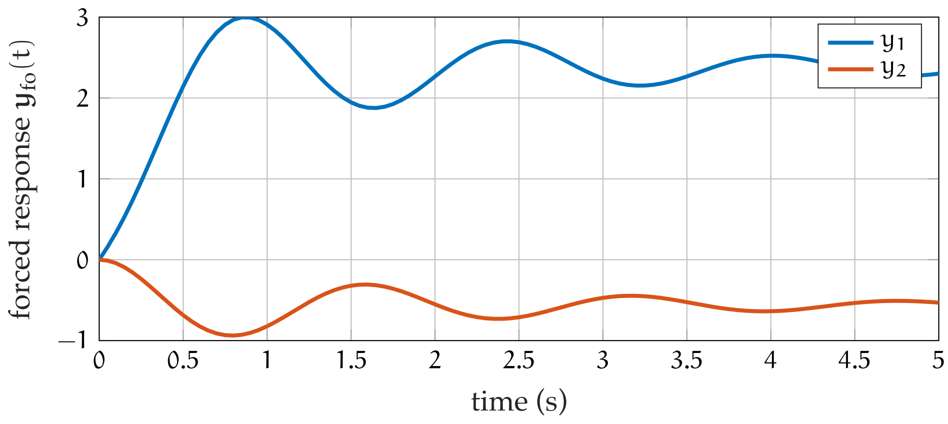

Figure sim.2: forced response y_{fo}.

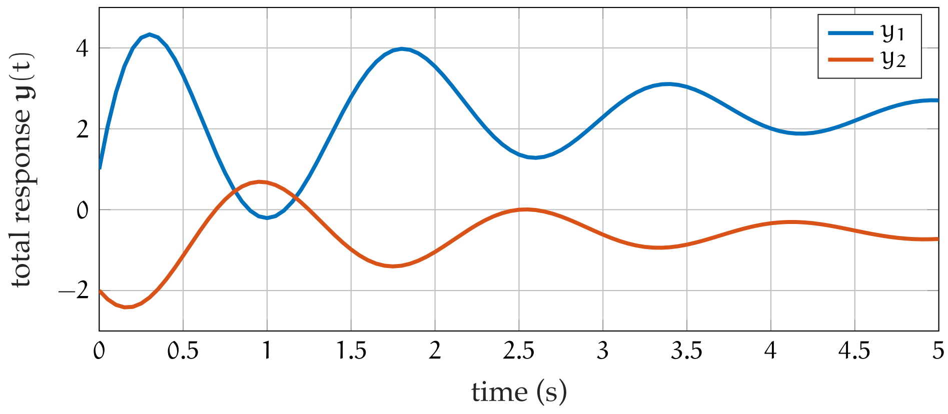

Figure sim.3: total response y.

Numerical solution

The numerical solution of the state equations is rather simple using Matlab's `ss` and `step` or `lsim` commands, as we show, here. First, we define an `ss` model object—a special kind of object that encodes a state-space model.

```
sys = ss(A,B,C,D);
```

At this point, using the `step` function would be the easiest way to solve for the step response. However, we choose the more-general `lsim` for

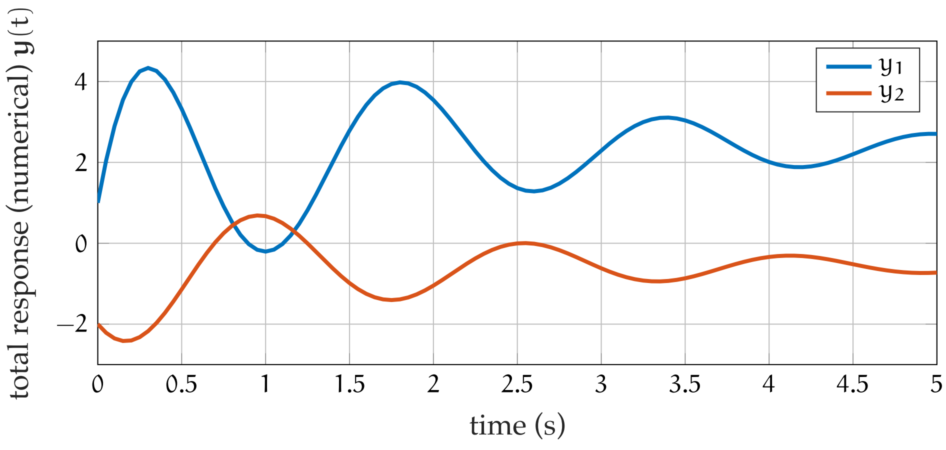

Figure sim.4: total response y from `lsim`.

demonstration purposes.

```
y_t_num = lsim(sys,u(t_a),t_a,x_0);
```

This total solution is shown in Fig. sim.4.

```
d_y = y_t-y_t_num';
```

Fig. sim.5 shows a plot of the differences between the analytic total solution `y_t` and the numerical `y_t_num` for each output. Note that calling this "error" is a bit presumptuous, given that we used numerical integration in the analytic solution. If a more accurate method is desired, working out the solution, symbolically, is the best.

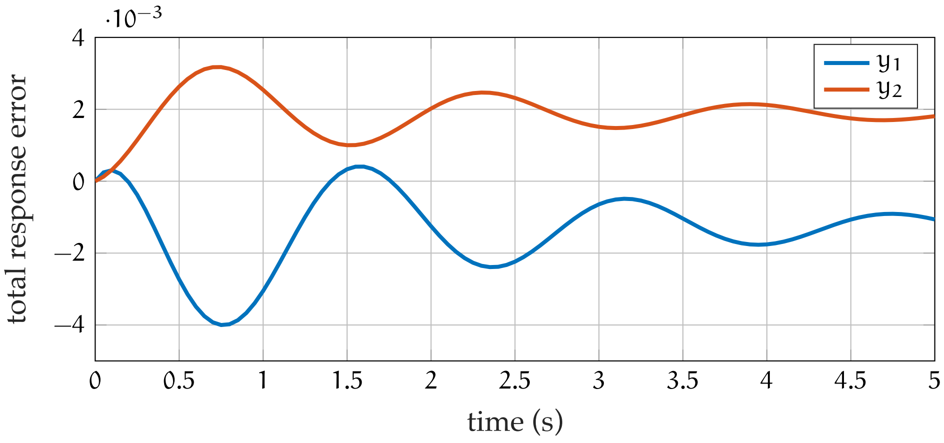

Figure sim.5: total response error `y_t-y_t_num`.