tf.tfmat Exploring transfer functions in Matlab

Matlab includes several nice functions for working with transfer functions. We explore some here.

The `tf` command and its friends

The `tf` command allows us to create LTI transfer function objects (which we'll abbreviate as "`tf` objects") that are recognized by `lsim`, `step`, and `initial`.
Consider the transfer function

$$H(s) = \frac{s+1}{s^3+3s^2+7s+1}. \tag{1}$$

We can make a Matlab model as follows.

```
sys = tf([1,1],[1,3,7,1])
```

```
sys =

          s + 1
  ---------------------
  s^3 + 3 s^2 + 7 s + 1

Continuous-time transfer function.
```

Alternatively, we could define `s` as a transfer function model itself.

```
s = tf([1,0],[1]); % tf is 1*s+0/1 = s
(s+1)/(s^3+3*s^2+7*s+1)
```

```
ans =

          s + 1
  ---------------------
  s^3 + 3 s^2 + 7 s + 1

Continuous-time transfer function.
```

Algebraic operations with `tf`s

Say we have two transfer functions $G(s)$ and $H(s)$ (already defined as `sys`). We might want to concatenate them. The idea is that we might

take the output of $G(s)$ and use that as the input to $H(s)$. In this case, the transfer function from the input of $G(s)$ to the output of $H(s)$ is just the multiplication

$$G(s)H(s). \tag{2}$$

```
G = 1/(s+2); % or tf([1],[1,2])
G*sys
```

```
ans =

              s + 1
  -------------------------------
  s^4 + 5 s^3 + 13 s^2 + 15 s + 2

Continuous-time transfer function.
```

Note that we have seen that Matlab handles addition and multiplication of scalars and `tf`s as well as the products of `tf`s. (It will also handle division.)

State-space models to `tf` models.

Consider the state-space model with standard matrices as shown below.

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

We can create a `ss` model as usual.

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

First, let's form a transfer function symbolically

We know the transfer function matrix is given by

$$C(sI - A)^{-1}B + D. \tag{3}$$

```
syms S
sys_tf_s = C*inv(S*eye(size(A)) - A)*B + D
```

```
sys_tf_s =

             1/(S + 2)
 1/(S + 2) + 1/(S + 3)
         1/(S + 3) + 1
```

This gave us three symbolic transfer functions in a 3×1 matrix, the first being that for the input to the first output, the second for the input to the second output, etc.

Or we can convert the *ss* model to a *tf* model

We can actually simply pass the `ss` model to the `tf` function.

```
sys_tf = tf(sys_ss)
```

```
sys_tf =

  From input to output...
         1
   1:  -----
       s + 2

          2 s + 5
   2:  -------------
       s^2 + 5 s + 6

       s + 4
   3:  -----
       s + 3

Continuous-time transfer function.
```

Note that the function `ss2tf` has a serious bug and should not be trusted.

Poles, zeros, and stability

Let's take a look at the poles and zeros of `sys`.

```
p_sys = pole(sys)
z_sys = zero(sys)
```

```
p_sys =

  -1.4239 + 2.1305i
  -1.4239 - 2.1305i
```

```
  -0.1523 + 0.0000i


z_sys =

    -1
```

Stability can be evaluated from `p_sys`. The system is stable because the real parts of all poles are negative.
Let's take a look at the pole-zero map.

```
figure;
pzmap(sys)
```

The resulting figure is shown in Fig. tfmat.1.

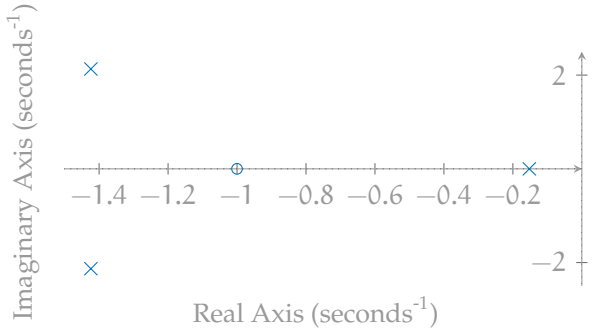

Figure tfmat.1: the pole-zero map.

Simulating with `tf`s

All the simulation functions we've used for `ss` models (`lsim`,`step`,`impulse`,`initial`) will also work for `tf` models. Let's try a `impulse` response on our original `sys` transfer function model.

```
t = linspace(0,15,200);
y = impulse(sys,t);
```

Plot.

```
figure
plot(t,y);
xlabel('time (s)')
ylabel('impulse response')
```

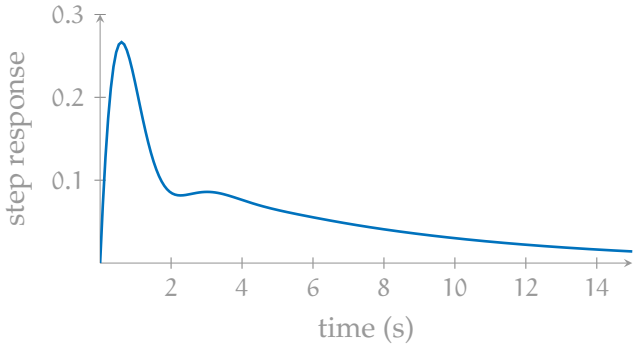

Figure tfmat.2: the impulse response.

The resulting figure is shown in Fig. tfmat.2.