MAE144 HW 1

Brian Kong A16996735

Question 1

https://github.com/TheBrianKong/ucsdmae144

I don't think I'll need to make a `README.md` because the file names should be distinguishable on their own or in well-named folders.

(also because no one reads the `README.md` anyway, unless they have to).

Question 2

Question 2a

$$G(s) = \frac{(s+2)(s-2)(s+5)(s-5)}{(s+1)(s-1)(s+3)(s-3)(s+6)(s-6)} = \frac{b(s)}{a(s)}$$

Make $D(s) = \frac{Y(s)}{X(s)}$ S.T. $T(s) = \frac{G(s)D(s)}{1+G(s)D(s)} = \frac{g(s)}{f(s)}$ with poles at $s = \{-1, 1, -3, 3, -6, 6\}$

Hint: form the denominator polynomial f(s), then call **RR_Diophantine**. The code that you need to write for this problem is literally a 2-liner: one to call **RR_Diophantine** properly, then one to double check that it worked. Note that, amongst all solutions of a(s)*x(s)+b(s)*y(s)=f(s), **RR_Diophantine** returns the answer with the smallest order for y(s).

```
f1 = RR_poly([-1 1 -3 3 -6 6],1); a = RR_poly([-5 5 -4 4],1); b = RR_poly([-2
2],1);
[x,y]=RR_diophantine(a,b,f1)
```

```
x =
  RR_poly with properties:
poly:     1.0000         0   -2.0952
roots:   -1.4475    1.4475
    n: 2
y =
  RR_poly with properties:
poly:    -2.9048         0 -128.5238
roots:   0.0000 - 6.6518i   0.0000 + 6.6518i
    n: 2
```

```
test=trim(a*x+b*y), residual1=norm(f1-test)
```

```
test =
  RR_poly with properties:
poly:     1.0000         0  -46.0000         0  369.0000         0 -324.0000
roots:   -6.0000   -3.0000   -1.0000    1.0000    3.0000    6.0000
    n: 6
residual1 = 1.2711e-13
```

```
G = RR_tf(RR_poly([-2 2 5 -5],1), RR_poly([-1 1 -3 3 -6 6],1))
```

```
G =
```

```
  RR_tf with properties:
num:     1     0   -29     0   100
den:     1     0   -46     0   369     0  -324
Continuous-time transfer function
  m=4, n=6, n_r=n-m=2, strictly proper, K=     1
  z:   -5.0000   -2.0000    2.0000    5.0000
  p:   -6.0000   -3.0000   -1.0000    1.0000    3.0000    6.0000
```

```
% We can conclude that it does gives what we want with a minimal residual
D = RR_tf(y,x)
```

```
D =
  RR_tf with properties:
num:   -2.9048         0 -128.5238
den:    1.0000         0   -2.0952
Continuous-time transfer function
  m=2, n=2, n_r=n-m=0, semiproper, K=   -2.9048
  z:   0.0000 - 6.6518i   0.0000 + 6.6518i
  p:   -1.4475    1.4475
```

```
T = G*D/(1+G*D) % check the properties of our T(s)
```

```
T =
  RR_tf with properties:
num:   1.0e+04 *
  -0.0003 + 0.0000i   0.0000 + 0.0000i  -0.0044 - 0.0000i   0.0000 + 0.0000i   0.3437 + 0.0000i  -0.0000 + 0.0000i
den:   1.0e+04 *
   0.0001 + 0.0000i   0.0000 + 0.0000i  -0.0051 - 0.0000i  -0.0000 - 0.0000i   0.0421 + 0.0000i  -0.0000 + 0.0000i
Continuous-time transfer function
  m=6, n=8, n_r=n-m=2, strictly proper, K=   -2.9048
  z:  -2.0000 + 0.0000i   2.0000 + 0.0000i  -5.0000 + 0.0000i   5.0000 + 0.0000i  -0.0000 - 6.6518i  -0.0000 + 6.651
  p:  -1.9343 + 0.0000i   1.9343 + 0.0000i   0.0000 - 2.4203i   0.0000 + 2.4203i   3.7834 + 0.0000i  -3.7834 + 0.000
```

Question 2b

```
fprintf("m = %d, n = %d",length(D.z),length(D.p))
```

```
m = 2, n = 2
```

From this we can determine that it is indeed proper since we have an equal power in the numerator and denominator of our controller.

If the controller is NOT proper ($m > n$), then we need to add an additional $m - n$ poles, such that the new tf has equal number of zeros and poles.

Question 3

input: RR_tf([num coeff], [den coeff]), h, omega_bar

```
% Test: Ds = RR_tf([1 z1],[1 p1 0])
syms z1 p1;
Ds = RR_tf([1 z1],[1 p1 0])
```

```
Ds =
```

```
  RR_tf with properties:
num:[1, z1]
den:[1, p1, 0]
Continuous-time transfer function
  m=1, n=2, n_r=n-m=1, strictly proper, K=1
  z:-z1
  p:[0, -p1]
```

```
Gz = EPK_C2D_matched(Ds,0.01)
```

```
om_bar = 0
Gz =
  RR_tf with properties:
num:-(z1/200 - 1)/(z1/200 + 1)
den:[1, -(p1/200 - 1)/(p1/200 + 1)]
Discrete-time transfer function with h=    0.0100
  m=0, n=1, n_r=n-m=1, strictly proper, K=-(z1/200 - 1)/(z1/200 + 1)
  z:  p:(p1/200 - 1)/(p1/200 + 1)
```

```
c2d(tf([1 1],[1 10 0]),0.01,'matched')
```

```
ans =

  0.009564 z - 0.009469
  ----------------------
  z^2 - 1.905 z + 0.9048

Sample time: 0.01 seconds
Discrete-time transfer function.
Model Properties
```

Well, my code is very incorrect as of now; I have run into so many different incompatibilities in the RR codebase and syms:

I have been alternating between two different approaches, and I was able to take one further than the other, but the algebra seems very incorrect for trying to implement the 9.31 version of Tustin's approx in the textbook . Also, the way that my method is set up, I get the following error when attempting to substitute syms vars:

```
subs(Gz,[z1 p1],[1 10])
```

```
Error using sym>tomupad
Unable to convert 'RR_tf' to 'sym'.
Error in sym (line 249)
                S.s = tomupad(x);
Error in subs (line 70)
    r_unique_name = subs(sym(f_unique_name),varargin{:});
Related documentation
```

This `tomupad` function has been the bane of my existence for the past several hours.

If we get our custom function to work as intended, it would have benefits over the standard MATLAB `c2d` function, providing methods to use symbolic variables for substitution later, and possibly handle the distortion/ warping with our f factor.

I'm dead tired, exhausted, and lacking sleep, and hope to see what is the right solution tomorrow!

```
function [Gz] = EPK_C2D_matched(Gs,h,om_bar) % epk= eun pyo kong
```

```
    if nargin == 2, om_bar = 0, f = 1; % basically skip computation for f in this
default case of omega = 0
    else f=2*(1-cos(om_bar*h))/(omg_bar*h*sin(omg_bar*h)); end
    m=Gs.num.n; n=Gs.den.n; % get our m and n values for Gs
    c = 2/(f*h); % a portion of our approximation with tustin and prewarping from z
to s
    %b=RR_poly(0); a=b;% initialize tf for a Gz = tf(b,a)
    b = sym(zeros(1,m)); a= sym(zeros(1,n));
    % we iterate for every pole and zero and map them to DT, appending to our poly
poles or zeros in a or b
    fac1=RR_poly([1 1]); fac2=RR_poly([1 -1]); % this allows us to bypass an issue
from further down the line:
    for j=1:m % zeros: b
        if isa(Gs.z(j),'sym') || Gs.z(j+1)<1e8
            % essentially doing z = e^{sh} and updating our b or a
            b(j) = (1+f*Gs.z(j)*h/2)/(1-f*Gs.z(j)*h/2); % for j = 1:m
            %b=b+Gs.num.poly(m+1-j)*c^j*fac1^(n-j)*fac2^j;
        else
            % case where infinite zero -> z= -1
            %b=b-1*c^j*fac1^(n-j)*fac2^j;
            b(j) = (-1);
        end
    end
    for j=1:n % poles: a
        a(j)=(1+f*Gs.p(j)*h/2)/(1-f*Gs.p(j)*h/2); % for j = 1:n, same reason as
above
        % a=a+Ds.den.poly(n+1-j)*c^j*fac1^(n-j)*fac2^j;
    end
    Gz=RR_tf(RR_poly(b),RR_poly(a));
    Gz.h = h;
end
```