MAE 144 HW 1
Dehao Lin

1. Provide a link to your GitHub page: https://github.com/deh98/MAE-144

2a. Consider the (pathological...) plant $G(S) = \frac{(S+2)(s-2)(s+5)(s-5)}{(s+1)(s-1)(s+3)(s-3)(s+6)(s-6)} = \frac{b(s)}{a(s)}$. Compute a controller $D(s) = \frac{y(s)}{x(s)}$ that puts the 6 poles of $T(s) = \frac{G(s)D(s)}{1+G(s)D(s)} = \frac{g(s)}{f(s)}$ at $s = \{-1,-1,-3,-3,-6,-6,\}$.

$Residual = 1.1369 \times 10^{-13}$

2b. Is the controller determined in 1a proper? If not, change your target f(s) by adding k additional poles to T(s) at, say, s=-20, for a sufficiently large k such that the answer returned by RR_Diophantine is proper, compute the modified D(s), and check that it worked. (How big does k need to be?). Discuss.

Output of my code:

```
>> y
        y =
          RR_poly with properties:
        poly:     0.6710    2.2869  -21.5818  -62.0409   42.6886   81.5318
        roots:   -6.0000   -3.0000   -1.0000    1.2681    5.3235
           n: 5
>> x
        x =
          RR_poly with properties:
        poly:    -0.6710   -2.2869   10.1755   24.1641
        roots:   -5.0009   -2.0030    3.5955
           n: 3
```

Because the number of zeros is greater than the numbers of poles, the controller in part 2a is not proper.

If the s=-20, f(s) becomes borderline proper at k=5, and strictly proper when k=6.

3. Study the RR_C2D_zoh and RR_C2D_tustin codes linked in section 9.3 of RR. Then, write a succinct, appropriately-commented code in a similar style, called XYZ_C2D_matched (replace XYZ with your initials), that performs a matched z-transform, as described in footnote 14 of section 9.3 of RR, to convert a D(s) to a corresponding D(z). Your code should have an option for inputting the frequency of interest omega_bar as discussed in point (iii) of footnote 14, and it should have an option to return either a semi-causal or a strictly-causal D(z) as discussed in point (ii) of footnote 14 (default values for both of these options should be used if they are not specified in the call to the function). Write your code in such a way that it can handle D(s) incorporating symbolic variables. For D(s)=(s+z1)/[s*(s+p1)], where z1 and p1 are symbolic variables, compute the corresponding D(z) using your code (and, for comparison, by hand). For z1=1 and p1=10, compare the result D(z) generated by your code to that returned by using the

`matched' option of MATLAB's built-in c2d function. Discuss. Is your code "better" than that provided by MATLAB? If so, discuss how. :)

My code is not better than the code provided by MATLAB because this matching method is ad hog, and might not return the most accurate Dz. (I saw some difference as I use the same transfer function on the Tustin methods and mine)

```
function [dz] = DL_C2D_matched(ds,h, omgBar)
% Matched pole-zero method that transform continuous time transfer
% function to descrete time transfer function.
% Ds - Continuouse time transfer function
% omgBar - frequency of interest


    % if ~isa(h,'RR_poly'), h = 0.001; end % estimate a small value time step
    % if ~isa(omgBar,'RR_poly'), omgBar = 0; end % check if omega bar input exists
    h = 0.001;
    omgBar = 0;

    zeros = RR_roots(ds.num); % finding the zeros and poles of continuous tf
    poles = RR_roots(ds.den);
    z_zeros = exp(zeros .* h); % initial s-to-z conversion
    z_poles = exp(poles .* h);

    m = ds.num.n; % check infinite zeros' existance
    n = ds.den.n;
    infz = m-n;
    if infz > 0 % including infinite zeros into the numerator, but include one less to make tf
proper
        for j = 1:(infz-1), z_zeros = [z_zeros -1]; end
    end

    prodnum = 1; % product of numerator
    for j = 1:numel(z_zeros) prodnum = prodnum * (h * omgBar + z_zeros(j)); end
    prodden = 1; % product of denominator
    for j = 1:numel(z_poles) prodden = prodden * (h * omgBar + z_poles(j)); end
    kfac = prodnum/prodden; % finding the correct gain

    dz = kfac * RR_tf(z_zeros,z_poles); % return the fully transformed descrete transfer function
    dz.h = h;

end
```