Part 1 — Poles and zeros

```
clear
```

1ab. Roots

Calculate the roots of each of the following polynomials

```
% Each polynomial is represented by each row of the matrix
P12 = [ 1 1 2 8 7 15 12 ; ...
        1 1 4 3 7 15 18 ];

% the number of polynomials
nRows = size(P12, 1);

% print the polynominal forms of each in turns of s
syms P [1 2], syms s
for k=1:nRows
    disp(P(k) == poly2sym(P12(k,:), s))
end % next k
```

$$P_1 = s^6 + s^5 + 2\,s^4 + 8\,s^3 + 7\,s^2 + 15\,s + 12$$
$$P_2 = s^6 + s^5 + 4\,s^4 + 3\,s^3 + 7\,s^2 + 15\,s + 18$$

```
% allocate cell array of roots
P_roots = cell(1, nRows);
% loop through the rows of P12
for k=1:nRows
    % calculate the roots for each polynomial
    P_roots{k} = roots(P12(k,:));
end % next k
```

The roots for each polynomial are

```
% display the roots in two tables showing both forms
P1_roots_Cartesian = complexTable(P_roots{1})
```

P1_roots_Cartesian = 6×3 table

	CartesianForm	r	thetaDeg
1	0.9979 + 1.6070i	1.8916	58.1624
2	0.9979 - 1.6070i	1.8916	301.8376
3	-1.8615 + 0.0000i	1.8615	180
4	-0.1302 + 1.4299i	1.4358	95.2009
5	-0.1302 - 1.4299i	1.4358	264.7991
6	-0.8739 + 0.0000i	0.8739	180

```
P2_roots_Cartesian = complexTable(P_roots{2})
```

P2_roots_Cartesian = 6×3 table

	CartesianForm	r	thetaDeg
1	1.0375 + 1.3227i	1.6811	51.8922
2	1.0375 - 1.3227i	1.6811	308.1078
3	-0.4632 + 1.9333i	1.9880	103.4723
4	-0.4632 - 1.9333i	1.9880	256.5277
5	-1.0743 + 0.6764i	1.2695	147.8047
6	-1.0743 - 0.6764i	1.2695	212.1953

2. Polynomial form

Calculate the polynomial form and roots of

```
% the polynomial
P3_poly = poly(-[5 2 3 -1 -2 4]);

% display factored out in terms of s
syms P3 s
disp(P3 == prod(factor(poly2sym(P3_poly, s))))
```

$$P_3 = (s-1)\,(s-2)\,(s+2)\,(s+3)\,(s+4)\,(s+5)$$

The polynomial form is

```
syms s
P3_s = poly2sym(P3_poly, s)
```

P3_s = $s^6 + 11\,s^5 + 31\,s^4 - 31\,s^3 - 200\,s^2 - 52\,s + 240$

The roots of the polynomial are

```
P3_roots = roots(P3_poly)
```

```
P3_roots = 6×1
   -5.0000
   -4.0000
   -3.0000
   -2.0000
    2.0000
    1.0000
```

3a. Converting to polynomial numerator and denominator.

Represent

```
% the transfer function in zero-pole-gain form
```

```
G1 = zpk(-[2 3 -6 8], -[0 7 -2 10 -3], 9)
```

```
G1 =
 
  9 (s+2) (s+3) (s+8) (s-6)
  --------------------------
  s (s+7) (s+10) (s-3) (s-2)
 
Continuous-time zero/pole/gain model.
```

using polynomials in the numerator and denominator.

In polynomial numerator and denominator, the transfer function

```
G1_tf = tf(G1)
```

```
G1_tf =
 
  9 s^4 + 63 s^3 - 288 s^2 - 2052 s - 2592
  ----------------------------------------
   s^5 + 12 s^4 - 9 s^3 - 248 s^2 + 420 s
 
Continuous-time transfer function.
```

3b. Converting to zero-pole-gain form.

Represent

```
% the transfer function in polynomial numerator and denominator form
G2 = tf([1 17 99 223 140], [1 32 363 2092 5052 4320])
```

```
G2 =
 
         s^4 + 17 s^3 + 99 s^2 + 223 s + 140
  ------------------------------------------------
  s^5 + 32 s^4 + 363 s^3 + 2092 s^2 + 5052 s + 4320
 
Continuous-time transfer function.
```

```

% $G_2(s) = \frac{s^4 + 17s^3+ 99s^2 + 223s + 140}
%                {s^5 + 32s^4 + 363s^3 + 2092s^2 + 5052s + 4320}}$
```

using factored forms of the polynomials in the numerator and denominator.

In zero-pole-gain form, the transfer function

```
G2_zpk = zpk(G2)
```

```
G2_zpk =
 
               (s+7) (s+5) (s+4) (s+1)
  ------------------------------------------------------
  (s+16.79) (s^2 + 4.097s + 4.468) (s^2 + 11.12s + 57.6)
 
Continuous-time zero/pole/gain model.
```

4abc. Partial fraction expansion

Calculate the partial fraction expansion of each of the following transfer functions.

$$G_3 = \frac{5(s+2)}{s(s^2+8s+15)}, \quad G_4 = \frac{5(s+2)}{s(s^2+6s+9)}, \quad G_5 = \frac{5(s+2)}{s(s^2+6s+34)},$$

which have the zero-pole-gain forms

```
% allocate G polynomials
%   2 factors per line
%   3-nominal max for each factor
%   2 lines for numerator and denominator
%   5 functions (first 2 unused)
G_roots = zeros(2, 3, 2, 5);

% get the size of G
[nFactors, nRootSummands, ~, nTfs] = size(G_roots);

% G3
G_roots(:, :, :, 3) = cat(3, [0 0 5 ; 0 1 2], [0 1 0; 1 8 15]);
% G4
G_roots(:, :, :, 4) = cat(3, [0 0 5 ; 0 1 2], [0 1 0; 1 6 9]);
% G5
G_roots(:, :, :, 5) = cat(3, [0 0 5 ; 0 1 2], [0 1 0; 1 6 34]);

% convolve the factors in each line giving G_poly:
% the resulting length of convolution for operands of length (M + 1),
% (N + 1) is (M + N + 1), so C convolutions of N-vectors each will give
% a length of CN - (C - 1) = (C - 1)N + 1
nPolySummands = (((nRootSummands - 1)*nFactors) + 1);
% allocate
G_poly = zeros(nPolySummands, 2, nTfs);
% loop through the transfer functions (skip first 2)
for (iTf=3:nTfs)
    for (iLine=1:2)
        G_poly(:, iLine, iTf) = convRows(G_roots(:, :, iLine, iTf));
    end % next iLine
end % next iTf

% print the polynominal forms of each transfer function:
% allocate room for the transfer functions
G = cell(1,nTfs);
% loop through the transfer functions
for iTf=3:nTfs
    G{iTf} = zpk(tf(G_poly(:, 1, iTf)', G_poly(:, 2, iTf)'));
end % next iTf
% print each one
G3 = G{3}
```

```
G3 =

     5 (s+2)
  -------------
  s (s+5) (s+3)

Continuous-time zero/pole/gain model.
```

```
G4 = G{4}
```

```
G4 =

   5 (s+2)
  ---------
  s (s+3)^2

Continuous-time zero/pole/gain model.
```

```
G5 = G{5}
```

```
G5 =

       5 (s+2)
  -----------------
  s (s^2 + 6s + 34)

Continuous-time zero/pole/gain model.
```

The partial fraction expansions are

```
% an (n + 1)-nomial will have n roots.
nPolesPred = (nPolySummands - 1);
% allocate R (coefficients), P (poles), K (direct term)
RP = zeros(nPolesPred, 2, nTfs);
% all the transfer functions have degree 1-(1+2) = -2,
% so expect no direct function
K = zeros(0, nTfs);
% loop through the transfer functions
for iTf=3:nTfs
    % the #roots predicted may be more than necessary
    [G345_R, G345_P, G345_K] = ...
        residue(G_poly(:, 1, iTf), G_poly(:, 2, iTf));
    % so 0-pad R, P, K
    nPolesActual = numel(G345_P);
    G345_R = [ zeros((nPolesPred - nPolesActual), 1) ; G345_R ];
    G345_P = [ zeros((nPolesPred - nPolesActual), 1) ; G345_P ];
    % collect the residue
    RP(:, 1, iTf) = G345_R;
    RP(:, 2, iTf) = G345_P;
    K(:, iTf) = G345_K;
end % next iTf
```

Well, we see that each of their residues (column #1), poles (column #2 in RP matrix), and their direct functions (K)

```
% copy R, P, K for each function, filtering out zero rows
% (for future use)
G3_RP = RP((RP(:,1,3) ~= 0), :, 3)
```

```
G3_RP = 3×2
   -1.5000   -5.0000
    0.8333   -3.0000
    0.6667         0
```

```
G3_K = K(:, 3)
```

```
G3_K =

  0×1 empty double column vector
```

```
G4_RP = RP((RP(:,1,4) ~= 0), :, 4)
```

```
G4_RP = 3×2
   -1.1111   -3.0000
    1.6667   -3.0000
    1.1111         0
```

```
G4_K = K(:, 4)
```

```
G4_K =

  0×1 empty double column vector
```

```
G5_RP = RP((RP(:,1,5) ~= 0), :, 5)
```

```
G5_RP = 3×2 complex
  -0.1471 - 0.4118i  -3.0000 + 5.0000i
  -0.1471 + 0.4118i  -3.0000 - 5.0000i
   0.2941 + 0.0000i   0.0000 + 0.0000i
```

```
G5_K = K(:, 5)
```

```
G5_K =

  0×1 empty double column vector
```

Thus the partial fraction expansions

```
% cause syntax error if misspelled
OMITNAN = 'omitnan';

% allocate room for the transfer functions
syms G_partial [1 nTfs]
% in terms of s
syms s
% loop through the transfer functions
for iTf=3:nTfs
    % filter out zero rows
```

```
    Tf_RP = RP((RP(:,1,iTf) ~= 0), :, iTf);
    % get the poles
    Tf_poles = Tf_RP(:, 2);
    % loop through remaining rows
    nTf_RP = numel(Tf_poles);
    % initialize to 0
    G_partial(iTf) = 0;
    % loop through the residue-pole rows
    for iRP = 1:nTf_RP
        % count the number of repeats so far for this pole (including
        % this one):
        % * poles are ordered so that an increase in previous instances
        % means an increase in order of the current instance
        nInstances = (sum(Tf_poles(1:iRP) == Tf_poles(iRP)));
        % add the fraction to G_partial: R/(s - P)^n
        G_partial(iTf) = (G_partial(iTf) + (Tf_RP(iRP,1)./(s - Tf_poles(iRP))^nInstances));
    end % next iRP
    % add all of the direct terms
    G_partial(iTf) = G_partial(iTf) + sum(K(:, iTf));
end % next iTf
% print each one
G3_partial = G_partial(3)
```

G3_partial =

$$\frac{5}{6\,(s+3)}-\frac{3}{2\,(s+5)}+\frac{2}{3\,s}$$

```
G4_partial = G_partial(4)
```

G4_partial =

$$\frac{5}{3\,(s+3)^2}-\frac{10}{9\,(s+3)}+\frac{10}{9\,s}$$

```
G5_partial = G_partial(5)
```

G5_partial =

$$\frac{5}{17\,s}+\frac{-\frac{5}{34}-\frac{7}{17}\,\mathrm{i}}{s+3-5\,\mathrm{i}}+\frac{-\frac{5}{34}+\frac{7}{17}\,\mathrm{i}}{s+3+5\,\mathrm{i}}$$

```
function complexTable = complexTable(complex)
```

complexTable(complex)

Creates a table showing the Cartesian forms, magnitudes and angles (in [0, 360) [deg]) of the given complex numbers.

Input Arguments

complex : double = vector of complex numbers

Output Arguments

complexTable : table (3-columns) = the table of the Cartesian forms, magnitudes and angles (in [0, 360) [deg]) of each complex numbers

```
    CartesianForm = complex;
    r = abs(complex);
    thetaDeg = angle360(complex);
    complexTable = table(CartesianForm, r, thetaDeg);
end % function complexTable(complex)


function angle360 = angle360(vector)
```

angle360(vector)

Gets an angle from a vector of complex numbers in the domain of [0, 360) [deg].

Input Arguments

vector : double = representing the vector of complex numbers

Output Arguments

acc : the vector of arrays in the domain of [0, 360) [deg

```
    angle360 = mod(rad2deg(angle(vector)) + 360, 360);
end % function angle360(vector)


function acc = convRows(matrix)
```

conv_rows(matrix)

Convolves the rows of a matrix into a row vector.

Input Arguments

T : double = 2D array whose rose to convolve

Output Arguments

acc : the resulting convolved row vector

```
    % initialize
    acc = 1;
    % transpose to loop the matrix by row
    % because Matlab defaults to by column
```

```
    for row = matrix'
        % convolve the accumulator with the next row.
        % note that row will be vertical as a column requiring another
        % transpose
        acc = conv(acc, row');
    end % next row
end % function conv_rows(matrix)
```