Morteza Kazemi | 97243054
Amir Masoud Shaker | 97243081
11/21/2021

# Homework #4

## Q1

| Syntax | Meaning | Description |
|---|---|---|
| 1) polyval(p,x)<br><br>2) [y,delta] = polyval(p,x,S)<br><br>3) y = polyval(p,x,[],mu) or [y,delta] = polyval(p,x,S,mu) | Polynomial evaluation | 1) This syntax evaluates the polynomial p at each point in x.<br><br>The argument p is a vector of length n+1 and its elements are coefficients of an nth-degree polynomial.<br><br>2) This syntax uses the optional output structure S to generate error estimates. delta is an estimate of the standard error in predicting a future observation at x by p(x).<br><br>3) This syntax uses the optional output mu to center and scale the data.<br><br>mu(1) is mean(x), and mu(2) is std(x). |

Input Arguments:

p — Polynomial coefficients, specified as a vector. For example, the vector [3 0 5] represents the polynomial $3 * x^2 + 0 * x^1 + 5 * x^0$

Data Types: single | double + Complex Number Support x — Query points, specified as a vector. polyval evaluates the polynomial p at the points in x and returns the corresponding function values in y.

Data Types: single | double + Complex Number Support

S — Error estimation structure. This structure is an optional output from [p,S] = polyfit(x,y,n) that can be used to obtain error estimates. S contains the following fields:

Fields:

| | |
|---|---|
| R: | Triangular factor from a QR decomposition of the Vandermonde matrix of x |
| Df: | Degrees of freedom |
| Normr: | Norm of the residuals |

mu — Centering and scaling values, specified as a two-element vector. This vector is an optional output from [p,S,mu] = polyfit(x,y,n) that is used to improve the numerical properties of fitting and evaluating the polynomial p. The value mu(1) is mean(x), and mu(2) is std(x). These values are used to center the query points in x at zero with unit standard deviation.

Output Arguments:

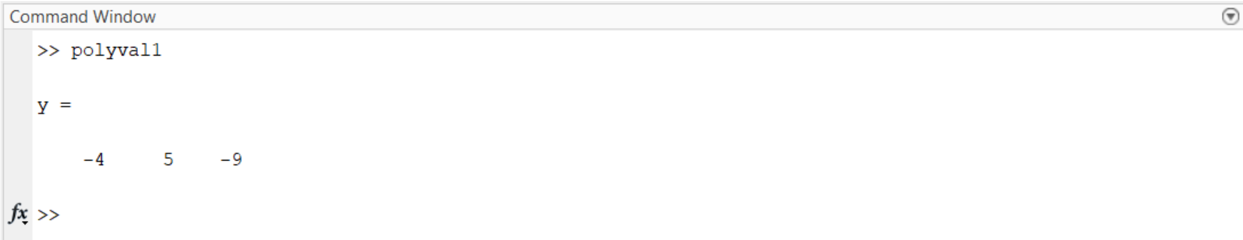
y — Function values, returned as a vector of the same size as the query points x. The vector contains the result of evaluating the polynomial p at each point in x.

delta — Standard error for prediction, returned as a vector of the same size as the query points x. Generally, an interval of y ± Δ corresponds to a roughly 68% prediction interval for future observations of large samples, and y ± 2Δ a roughly 95% prediction interval.

Examples:

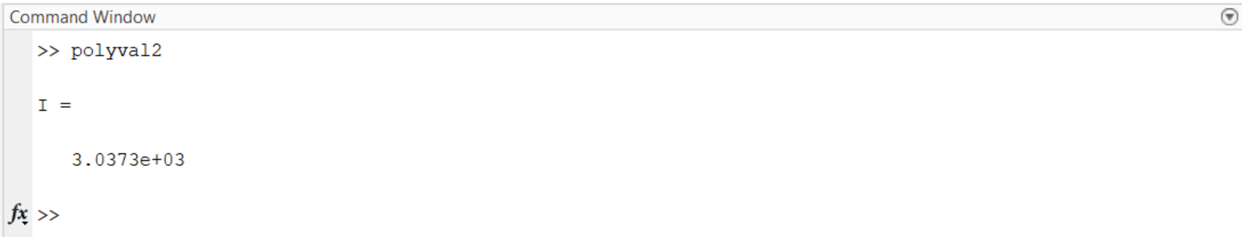
Polyval1:

```
p = [2 3 -9];
% coefficients of the function
x = [1 2 0];
% some points
y = polyval(p,x)
% evaluate the polynomial function at the points x=1,2,0
```



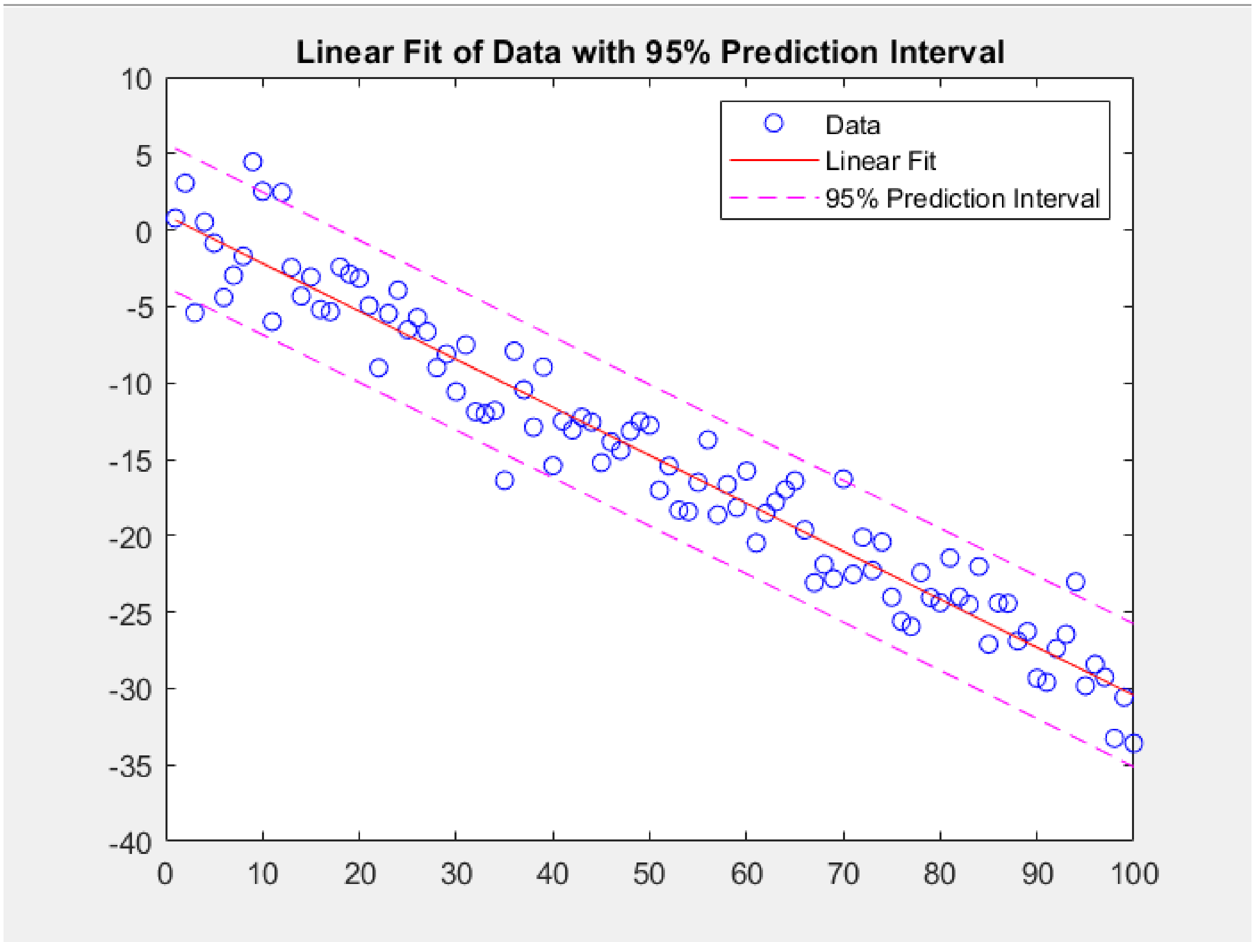

Polyval2:

```
p = [1 2 -5 15];
% coefficients of the polynomial function
q = polyint(p);
% integrate the polynomial function
lower_limit = 2;
% lower limit of the integral
upper_limit = 10;
% upper limit of the integral
I = diff(polyval(q,[lower_limit upper_limit]))
% find the value of the integral
```



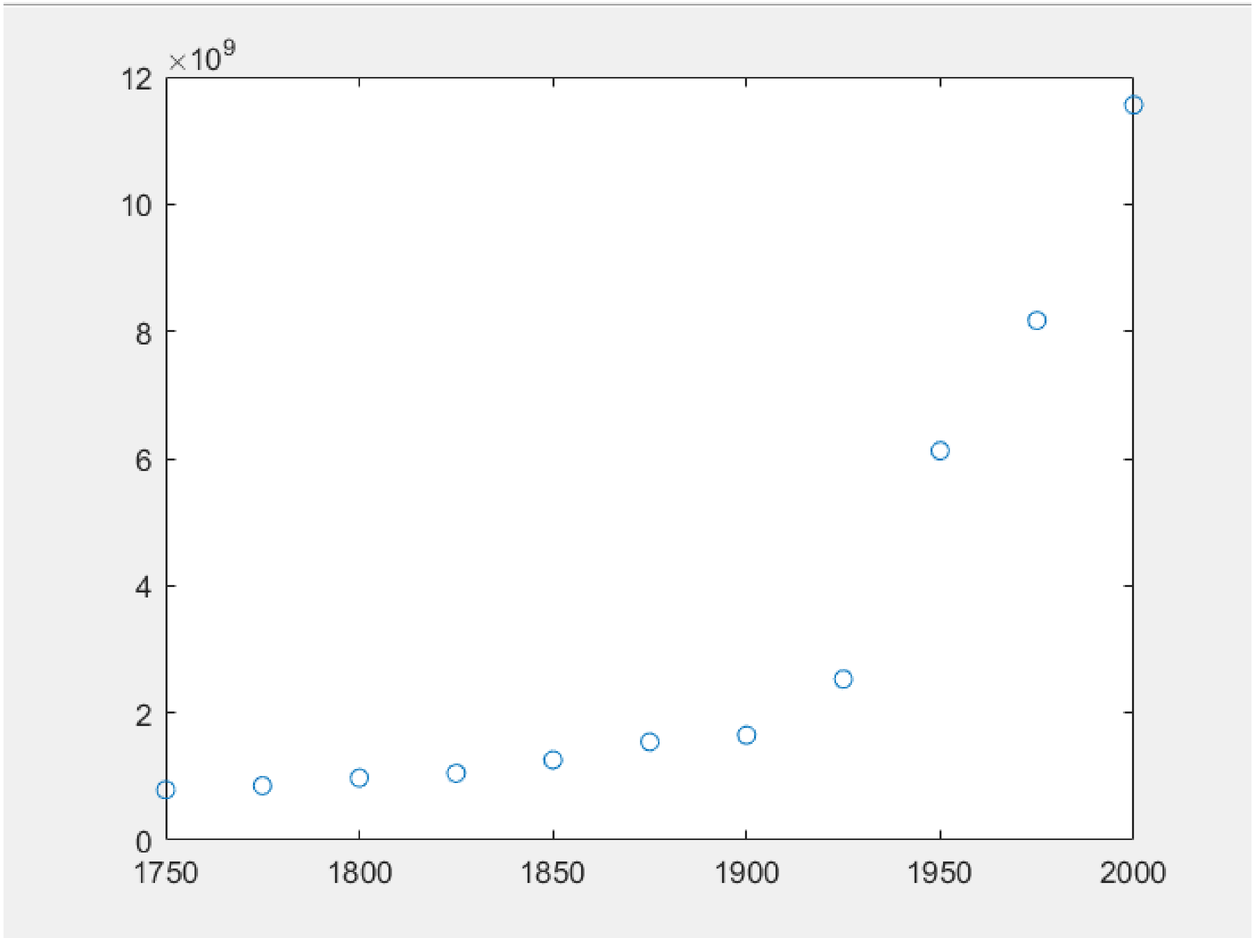
Polyval3:

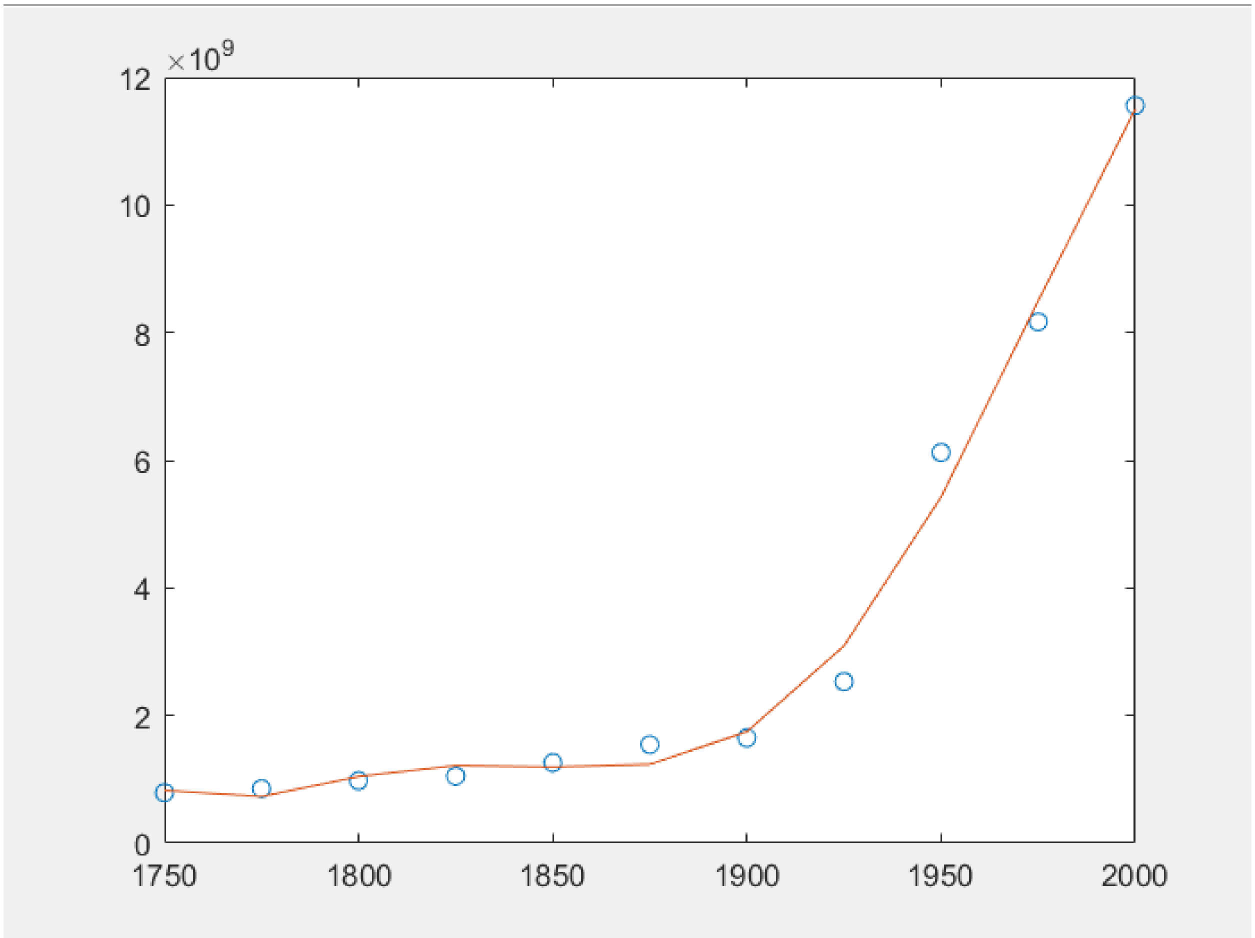
```
x = 1:100;
% create datapoints
y = -0.3*x + 2*randn(1,100);
% create datapoints
[p,S] = polyfit(x,y,1);
% fit a first degree polynomial to the data
% return the coefficients for the linear fit and the error estimation structure
[y_fit,delta] = polyval(p,x,S);
plot(x,y,'bo')
hold on
plot(x,y_fit,'r-')
plot(x,y_fit+2*delta,'m--',x,y_fit-2*delta,'m--')
title('Linear Fit of Data with 95% Prediction Interval')
legend('Data','Linear Fit','95% Prediction Interval')
% Plot the original data, linear fit, and 95% prediction interval y±2?
```



Polyval4:

```
x = 1:100;
% create datapoints
y = -0.3*x + 2*randn(1,100);
% create datapoints
[p,S] = polyfit(x,y,1);
% fit a first degree polynomial to the data
% return the coefficients for the linear fit and the error estimation structure
[y_fit,delta] = polyval(p,x,S);
plot(x,y,'bo')
hold on
plot(x,y_fit,'r-')
plot(x,y_fit+2*delta,'m--',x,y_fit-2*delta,'m--')
title('Linear Fit of Data with 95% Prediction Interval')
legend('Data','Linear Fit','95% Prediction Interval')
% Plot the original data, linear fit, and 95% prediction interval y±2?
```





$\times 10^9$
12
10
8
6
4
2
0
1750
1800
1850
1900
1950
2000

| Syntax | Meaning | Description |
| --- | --- | --- |
| r = roots(p) | Polynomial roots | This syntax returns the roots of the polynomial represented by p as a column vector. |

Input Arguments:

p — Polynomial coefficients, specified as a vector.

Data Types: single | double + Complex Number Support

Examples:

Roots1:

```
p = [1 -1 -2];
% coefficients of the polynomial function
r = roots(p)
% find the roots of the polynomial function
```

Command Window

```
>> roots1

r =

     2
    -1

fx >>
```

Roots2:

```
p = [1 0 0 -9];
% coefficients of the polynomial function
r = roots(p)
% find the roots of the polynomial function
```

```
Command Window
>> roots2

r =

  -1.0400 + 1.8014i
  -1.0400 - 1.8014i
   2.0801 + 0.0000i

fx >>
```

| Syntax | Meaning | Description |
|---|---|---|
| 1) p = poly(r)<br>2) p = poly(A) | Polynomial with specified roots or characteristic polynomial | 1) This syntax returns the coefficients of the polynomial whose roots are the elements of r (r is a vector).<br>2) This syntax returns the n+1 coefficients of the characteristic polynomial of the matrix, det(λI – A) (A is a matrix). |

Input Arguments:

r — Polynomial roots, specified as a vector.

Data Types: single | double + Complex Number Support

A — Input matrix

Data Types: single | double + Complex Number Support

Output Arguments

p — Polynomial coefficients, returned as a row vector.

If the input is a square n-by-n matrix, A, then p contains the coefficients for the characteristic polynomial of A. If the input is a vector of roots, r, then p contains the coefficients for the polynomial whose roots are in r.

Examples:

Poly1:

```
M = [2 4 6; 8 10 12; 14 16 18]
% creat matrix M
e = eig(M)
% compute the eigenvalues of matrix M
p = poly(e)
% determine the characteristic polynomial from the values in e
```

Command Window

```
>> poly1

M =

     2     4     6
     8    10    12
    14    16    18


e =

   32.2337
   -2.2337
   -0.0000


p =

    1.0000  -30.0000  -72.0000   -0.0000

>>
```

Poly2:

```
M = [1 3 5; 7 9 11; 13 15 17]
% creat matrix M
p = poly(M)
% calculate the characteristic polynomial of matrix M
r = roots(p)
% find the roots of p which are the eigenvalues of matrix M
```

Command Window

```
>> poly2

M =

     1     3     5
     7     9    11
    13    15    17


p =

    1.0000  -27.0000  -72.0000   -0.0000


r =

   29.4452
   -2.4452
   -0.0000

fx >>
```

# Q2

```
clc;
clear;

coeff = input('please enter the coefficients: ');
% gets the coefficients of the polynomial function
r = roots(coeff);
% computes real and imaginary roots of the polynomial function
is_real = imag(r) == 0;
% creates a boolean vector which containes 1 for real roots and 0 for imaginary
% roots
is_imaginary = imag(r) ~= 0;
% creates a boolean vector which containes 1 for imaginary roots and 0 for real
% roots

num_of_real = sum(is_real);
% gets the number of real roots of the polynomial function
num_of_imaginary = sum(is_imaginary);
% gets the number of imaginary roots of the polynomial function

fprintf('\nnumber of real roots : %d \n', num_of_real);
fprintf('\nnumber of imaginary roots : %d \n', num_of_imaginary);
```

Command Window

```
please enter the coefficients: [1 0 0 0 -1]

number of real roots : 2

number of imaginary roots : 2
fx >>
```

## Q3

| Syntax | Meaning | Description |
|---|---|---|
| 1) w = conv(u,v)<br>2) w = conv(u,v,shape) | Convolution and polynomial multiplication | 1) This syntax returns the convolution of vectors u and v.<br>If u and v are vectors of polynomial coefficients, this function will actually multiply them together.<br>2) This syntax returns a subsection of the convolution, as specified by shape.<br>conv(u,v,'same') returns only the central part of the convolution, the same size as u.<br>conv(u,v,'valid') returns only the part of the convolution computed without the zero-padded edges. |

Input Arguments:
u,v — Input vectors, specified as either row or column vectors. The vectors u and v can be different lengths or data types.
When u or v are of type single, then the output is of type single. Otherwise, conv converts inputs to type double and returns type double.

Data Types: double | single | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64 | logical
+ Complex Number Support

shape — Subsection of the convolution, specified as 'full', 'same', or 'valid'.

- 'full' : Full convolution (default).

- 'same': Central part of the convolution of the same size as u.

- 'valid': Only those parts of the convolution that are computed without the zero-padded edges. Using this option, length(w) is max(length(u)-length(v)+1,0), except when length(v) is zero. If length(v) = 0, then length(w) = length(u).

Examples:

Conv1:

```
u = [2 1 -1];
% coefficients of polynomial function u
v = [5 6];
% coefficients of polynomial function v
w = conv(u,v)
% multiply the polynomials
```

```
Command Window
>> conv1

w =

    10    17     1    -6

fx >>
```

## Conv2:

```
u = [1 1 1];
% create vector u
v = [1 1 0 0 0 1 1];
% create vector v
w = conv(u,v)
% convolve u and v
```

```
Command Window
>> conv_2

w =

     1     2     2     1     0     1     2     2     1

fx >>
```

## Conv3:

```
u = [-1 2 3 -2 0 1 2];
% create vector u
v = [2 4 -1 1];
% create vector v
w = conv(u,v,'same')
% find the central part of the convolution of u and v
```

```
Command Window
>> conv3

w =

    15     5    -9     7     6     7    -1

fx >>
```

| Syntax | Meaning | Description |
| --- | --- | --- |
| [q,r] = deconv(u,v) | Deconvolution and polynomial division | This syntax deconvolves a vector v out of a vector u using long division, and returns the quotient q and remainder r.<br><br>If u and v are vectors of polynomial coefficients, this function will actually divide u by v. |

Input Arguments:
u,v — Input vectors, specified as either row or column vectors. u and v can be different lengths or data types.
If one or both of u and v are of type single, then the output is also of type single. Otherwise, deconv returns type double.
The lengths of the inputs should generally satisfy length(v) <= length(u). However, if length(v) > length(u), then deconv returns the outputs as q = 0 and r = u.

Data Types: double | single + Complex Number Support

Output Arguments:
q — Quotient, returned as a row or column vector such that u = conv(v,q)+r.
r — Remainder, returned as a row or column vector such that u = conv(v,q)+r.

Data Types: double | single

Examples:

Deconv1:

```
u = [2 7 4 9];
% coefficients of polynomial function u
v = [1 0 1];
% coefficients of polynomial function v
[q,r] = deconv(u,v)
% divide the u by the v which results in quotient coefficients and remainder
coefficients
```

```
Command Window
>> deconv1

q =

     2     7


r =

     0     0     2     2

fx >>
```

# Q4

```
clc;
clear;

N = [1, 5, 11, 13];
% creates the polynomial function N
D = [1, 2, 4];
% creates the polynomial function D
[Q,R] = deconv(N,D)
% computes the quotient and remainder
fprintf('\nH(s) = N(s)/D(s)\n');
fprintf('\nN(s)/D(s) = Q(s) + R(s)/D(S)\n');
fprintf('\nN(s) = Q(s).D(S) + R(s)\n');
fprintf('\ns^3 + 5s^2 + 11s + 13 = (s + 3)(s^2 + 2s + 4) + (s + 1)\n');
fprintf('\nWe are done!\n');
```

Command Window

```
Q =

     1     3


R =

     0     0     1     1


H(s) = N(s)/D(s)

N(s)/D(s) = Q(s) + R(s)/D(S)

N(s) = Q(s).D(S) + R(s)

s^3 + 5s^2 + 11s + 13 = (s + 3)(s^2 + 2s + 4) + (s + 1)

We are done!
>>
```

## Q5

| Syntax | Meaning | Description |
|---|---|---|
| 1) k = polyder(p)<br>2) k = polyder(a,b)<br>3) [q,d] = polyder(a,b) | Polynomial differentiation | 1) returns the derivative of the polynomial represented by the coefficients in p<br>2) returns the derivative of the product of the polynomials a and b<br>3) returns the derivative of the quotient of the polynomials a and b<br>$\frac{q(x)}{d(x)} = \frac{d}{dx}\left[\frac{a(x)}{b(x)}\right]$ |

Input Arguments:

p — Polynomial coefficients (vector)
For example, the vector [3 0 5] represents the polynomial $3 * x^2 + 0 * x^1 + 5 * x^0$
a,b — Polynomial coefficients (specified as two separate arguments of row vectors.)

Data Types: single | double + Complex Number Support

Output Arguments:

k — Differentiated polynomial coefficients (row vector)
q — Numerator polynomial (row vector)
d — Denominator polynomial (row vector)

Examples:

```
clc;
clear;

p1 = [-9 0 5 0 5]; % -9x^4 + 5x^2 + 5
p2 = [1 0]; % x
q1 = polyder(p1) % q1(x) = dp1(x)/dx
q2 = polyder(p1, p2) % q2(x) = dp1(x)p2(x)/dx
[q3_q, q3_d] = polyder(p1, p2) % q3_q(x)/q3_d(x) = d(p1(x)/p2(x))/dx
```

```
Command Window

q1 =

   -36     0    10     0


q2 =

   -45     0    15     0     5


q3_q =

   -27     0     5     0    -5


q3_d =

     1     0     0
```

| Syntax | Meaning | Description |
|---|---|---|
| 1) q = polyint(p,k)<br>2) q = polyint(p) | Polynomial integration | 1) returns the integral of the polynomial represented by the coefficients in p using a constant of integration k.<br>2) assumes a constant of integration k = 0. |

Input Arguments:
p — Polynomial coefficients, specified as a vector.

k — Constant of integration, specified as a numeric scalar.

Data Types: single | double + Complex Number Support

Output Arguments:

q — Integrated polynomial coefficients, returned as a row vector.

Examples:

```
clc;
clear;

p1 = [-9 0 5 0 5]; % -9x^4 + 5x^2 + 5
p2 = [1 0]; % x
k = 3; % integration constant

q1 = polyint(p1) % integration of p1
q1_with_k = polyint(p1, k) % integration with a constant k

q2 = polyint(conv(p1, p2)) % integration of p1*p2
q2_with_k = polyint(conv(p1, p2), k) % integration of p1*p2 with a constant k
```

Command Window

```
q1 =

   -1.8000         0    1.6667         0    5.0000         0


q1_with_k =

   -1.8000         0    1.6667         0    5.0000    3.0000


q2 =

   -1.5000         0    1.2500         0    2.5000         0         0


q2_with_k =

   -1.5000         0    1.2500         0    2.5000         0    3.0000
```

Calculations for N(s) and H(s):

```
clc;
clear;

N = [1 5 11 13]; % s^3 + 5s^2 + 11s + 13
D = [1 2 4]; % s^2 + 2s + 4
%[H_q, H_r] = deconv(N, D); % N/D

disp('Function: N');
first_derivative = polyder(N) % first order derivative of N
second_derivative = polyder(first_derivative) % second order dervative of N
integral = polyint(N) %integration without constant

disp('Function: H');
[first_derivative_q, first_derivative_d] = polyder(N,D) % first order derivative of H
[second_derivative_q, second_derivative_d] =
polyder(first_derivative_q,first_derivative_d) % second order dervative of H
integral = polyint(deconv(N, D)) %integration without constant
```

Command Window

```
Function: N

first_derivative =

     3    10    11


second_derivative =

     6    10


integral =

    0.2500    1.6667    5.5000   13.0000         0

Function: H

first_derivative_q =

     1     4    11    14    18


first_derivative_d =

     1     4    12    16    16
```

```
second_derivative_q =

     2    10     8   -16   -80   -64


second_derivative_d =

     1     8    40   128   304   512   640   512   256


integral =

    0.5000    3.0000         0
```

## Q6

```
clc;
clear;

m1 = [-1 0 1]; % m1 = -x^2 + 1
m2 = [1 -2 3]; % m2 = x^2 -2x + 3
m1_to3 = conv(conv(m1, m1), m1); %m1^3
m3 = conv(m1_to3, m2); % m1^3*m2
final = polyint(m3);

% integration boundaries
a = -3;
b = 4;

value_for_a_b = polyval(final,[a b]); % polyval(p,x) evaluates the polynomial p at
each point in x.

% diff(X) calculates differences between adjacent elements of X along the
% first array dimension. here we only have two elements and it gives (value for b -
value for a)
I = diff(value_for_a_b) %integration from a to b
```

```
Command Window

I =

  -1.8383e+04
```

We checked the answer here: https://www.integral-calculator.com/

YOUR INPUT:
$f(x) =$

$$\left(1-x^2\right)^3\left(x^2-2x+3\right)$$

**Note:** Your input has been rewritten/simplified.

No further simplification found!

DEFINITE INTEGRAL:
$\int_{-3}^{4} f(x)\,\mathrm{d}x =$

$$-\frac{3308963}{180}$$

No further simplification found!

**Approximation:**

$-18383.12777777778$

## Q7

| Syntax | Meaning | Description |
|---|---|---|
| 3) [r,p,k] = residue(b,a)<br><br>4) [b,a] = residue(r,p,k) | Partial fraction expansion (decomposition) | 3) finds the residues, poles, and direct term of a Partial Fraction Expansion of the ratio of two polynomials, where the expansion is:<br><br>$\frac{b(s)}{a(s)}=\frac{b_m s^m+b_{m-1}s^{m-1}+\ldots+b_1 s+b_0}{a_n s^n+a_{n-1}s^{n-1}+\ldots+a_1 s+a_0}=\frac{r_n}{s-p_n}+\ldots+\frac{r_2}{s-p_2}+\frac{r_1}{s-p_1}+k(s).$<br><br>The inputs to residue are vectors of coefficients of the polynomials b = [bm ... b1 b0] and a = [an ... a1 a0]. The outputs are the residues r = [rn ... r2 r1], the poles p = [pn ... p2 p1], and the polynomial k. For most textbook problems, k is 0 or a constant.<br><br>4) converts the partial fraction expansion back to the ratio of two polynomials and returns the coefficients in b and a. |

Notes:

If the degree of the numerator is equal to the degree of the denominator, the output k can be nonzero and if the degree of the numerator is greater than the degree of the denominator, the output k is a vector that represents the coefficients of a polynomial in s.

Input Arguments:

b/a — Coefficients of numerator/denominator polynomial (vector of numbers)

Specified as a vector of numbers representing the coefficients of the polynomial in descending powers of s.

Data Types: single | double + Complex Number Support

Output Arguments:

r/p — Residues/Poles/ Direct term of partial fraction expansion (r and p: column vector of numbers that are specified in the table above, k: row vector of numbers that specify the coefficients of the polynomial in descending powers of s.)

Examples:

```
clc;
clear;
%F = b/a

print_residue([5 3], [3 4 5]); % complex roots (Degree of Numerator < Degree of
Denominator)
print_residue([5 3 1], [1 2 1]); % real roots (Degree of Numerator = Degree of
Denominator)
print_residue([5 3 1 0], [1 2 1]); % real roots (Degree of Numerator > Degree of
Denominator)


function print_residue(b, a)
    disp('converted to partial expansion:');
    [r, p, k] = residue(b, a)
    disp('converted back to polynomial coefficients:');
    [b_conv_back, a_conv_back] = residue(r, p, k)
end
```

```
Command Window
converted to partial expansion:

r =

   0.8333 + 0.0503i
   0.8333 - 0.0503i


p =

  -0.6667 + 1.1055i
  -0.6667 - 1.1055i


k =

     []

converted back to polynomial coefficients:

b_conv_back =

    1.6667    1.0000


a_conv_back =

    1.0000    1.3333    1.6667
```

```
Command Window
converted to partial expansion:

r =

    -7
     3


p =

    -1
    -1


k =

     5

converted back to polynomial coefficients:

b_conv_back =

     5     3     1


a_conv_back =

     1     2     1
```

```
Command Window
converted to partial expansion:

r =

    10
    -3


p =

    -1
    -1


k =

     5    -7

converted back to polynomial coefficients:

b_conv_back =

     5     3     1     0


a_conv_back =

     1     2     1
```

## Expanding Y(s):

```
clc;
clear;

%Y(s) = b(s)/a(s)

b = [1 3];
a = [1 6 8 0];

[r, p, k] = residue(b, a)
```

```
Command Window
r =

   -0.1250
   -0.2500
    0.3750


p =

    -4
    -2
     0


k =

     []
```