

MATLAB Course

For Engineers

lecture 7



Symbolic Math

The symbolic **Math toolbox** is a collection of Matlab **functions** that are used for execution of **symbolic** operators. Specifically the followings:

- Symbolic **algebra**.
- Symbolic methods for **solving algebraic and transcendental equations**.
- Symbolic methods for **solving ordinary differential equations**.
- Symbolic **calculus**, including **integration, differentiation, limits, and series**.
- Laplace** transforms.
- Selected topics in **linear algebra**, including symbolic methods for obtaining **determinants, matrix inverses, and eigenvalues**.

The **sym** function can be used to **create** “**symbolic objects**” in MATLAB.

If the **input** argument to **sym** is a **string**, the **result** is a **symbolic number** or **variable**. If the input argument is a **numeric** scalar or matrix, the result is a symbolic representation of the given **numeric** values.

For example, typing **x = sym('x')** creates the **symbolic** variable with name **x**, and typing **y = sym('y')** creates a **symbolic** variable named **y**.

Typing **x = sym('x', 'real')** tells MATLAB to assume that **x** is **real**. Typing **x = sym('x', 'unreal')** tells MATLAB to assume that **x** is **not real**.

The **syms** function enables you to combine **more** than **one** such statement into a **single statement**.

For example, typing **syms x** is equivalent to typing **x = sym('x')**, and typing **syms x y u v** creates the **four** symbolic variables **x**, **y**, **u**, and **v**.

You can use the **sym** function to create **symbolic constants** by using a **numerical** value for the argument. **For example**, typing

```
pi = sym('pi')  
fraction = sym('1/3')  
and  
sqroot2 = sym('sqrt(2)')
```

create symbolic **constants** that **avoid** the **floating-point** approximations inherent in the values of π , $1/3$, and $\sqrt{2}$.

You can use **symbolic** variables in expressions and as arguments of functions. You use the operators $+$ $-$ $*$ $/$ $^$ and the **built-in functions** just as you use them with **numerical calculations**. **For example**, typing

```
>>syms x y
>>s = x + y;
>>r = sqrt(x^2 + y^2);
```

creates the **symbolic** variables **s** and **r**. The terms **s = x + y** and **r = sqrt(x² + y²)** are **examples** of symbolic *expressions*.

The **vector** and **matrix** notation used in MATLAB also applies to symbolic variables. **For example**, you can **create** a **symbolic** matrix **A** as follows:

```
>>n = 3;  
>>syms x;  
>>A = x.^([0:n]' * [0:n])  
A =  
[ 1, 1, 1, 1]  
[ 1, x, x^2, x^3]  
[ 1, x^2, x^4, x^6]  
[ 1, x^3, x^6, x^9]
```

By **contrast**, the function `findsym(E,n)` returns the **n** symbolic variables in **E** **closest** to **x**, with the tie breaker going to the variable **closer** to **z**.

```
>>syms b x1 y
```

```
>>findsym(6*b+y)
```

```
ans =
```

```
    b, y
```

```
>>findsym(6*b+y+x) %Note:x has not been declared symbolic.
```

```
??? Undefined function or variable 'x'.
```

```
>>findsym(6*b+y,1) %Find the one variable closest to x
```

```
ans =
```

```
    y
```

```
>>findsym(6*b+y+x1,1) %Find the one variable closest to x
```

```
ans =
```

```
    x1
```

```
>>findsym(6*b+y*i) %i is not symbolic
```

```
ans =
```

```
    b, y
```

The function `collect(E)` collects coefficients of like powers in the expression `E`. If there is more than one variable, you can use the optional form `collect(E, v)`, which collects all the coefficients with the same power of `v`.

```
>>syms x y
>>E = (x-5)^2+(y-3)^2;
>>collect(E)
ans =
      x^2-10*x+25+(y-3)^2
>>collect(E, y)
ans =
      y^2-6*y+(x-5)^2+9
```

The `expand` and `simplify` functions.

```
>>syms x y
>>expand((x+y)^2) % applies algebra rules
ans =
    x^2+2*x*y+y^2
>>expand(sin(x+y)) % applies trig identities
ans =
    sin(x)*cos(y)+cos(x)*sin(y)
>>simplify(6*((sin(x))^2+(cos(x))^2))...
% applies another trig identity
ans =
    6
```

The `factor` function.

```
>>syms x y
>>factor(x^2-1)
ans =
      (x-1) * (x+1)
```

The function `subs(E, old, new)` substitutes **new** for **old** in the expression **E**, where **old** can be a **symbolic variable** or **expression** and **new** can be a **symbolic variable**, **expression**, or **matrix**, or a **numeric** value or **matrix**. **For example,**

```
>>syms x y
>>E = x^2+6*x+7;
>>F = subs(E, x, y)
F =
      y^2+6*y+7
```

If you want to tell MATLAB that f is a **function** of the variable t , type `f = sym('f(t)')`. There after, `f` behaves like a function of t , and you can **manipulate** it with the toolbox commands.

For example, to create a new function $g(t) = f(t + 2) - f(t)$,

```
>>syms t
>>f = sym('f(t)');
>>g = subs(f, t, t+2) - f
g =
    f(t+2) - f(t)
```

Once a specific function is defined for $f(t)$, the function $g(t)$ will be available.

Use the **subs** and **double** functions to **evaluate** an expression numerically. Use **subs (E, old, new)** to replace old with a numeric value **new** in the expression **E**. The result is of class **double**. **For example,**

```
>>syms x
>>E = x^2+6*x+7;
>>G = subs (E, x, 2)
G =
    23
>>class (G)
ans =
    double
```

The MATLAB function `ezplot(E)` generates a **plot** of a symbolic expression **E**, which is a function of **one** variable. The default range of the **independent** variable is the interval `[-2pi, 2pi]` unless this interval contains a **singularity**.

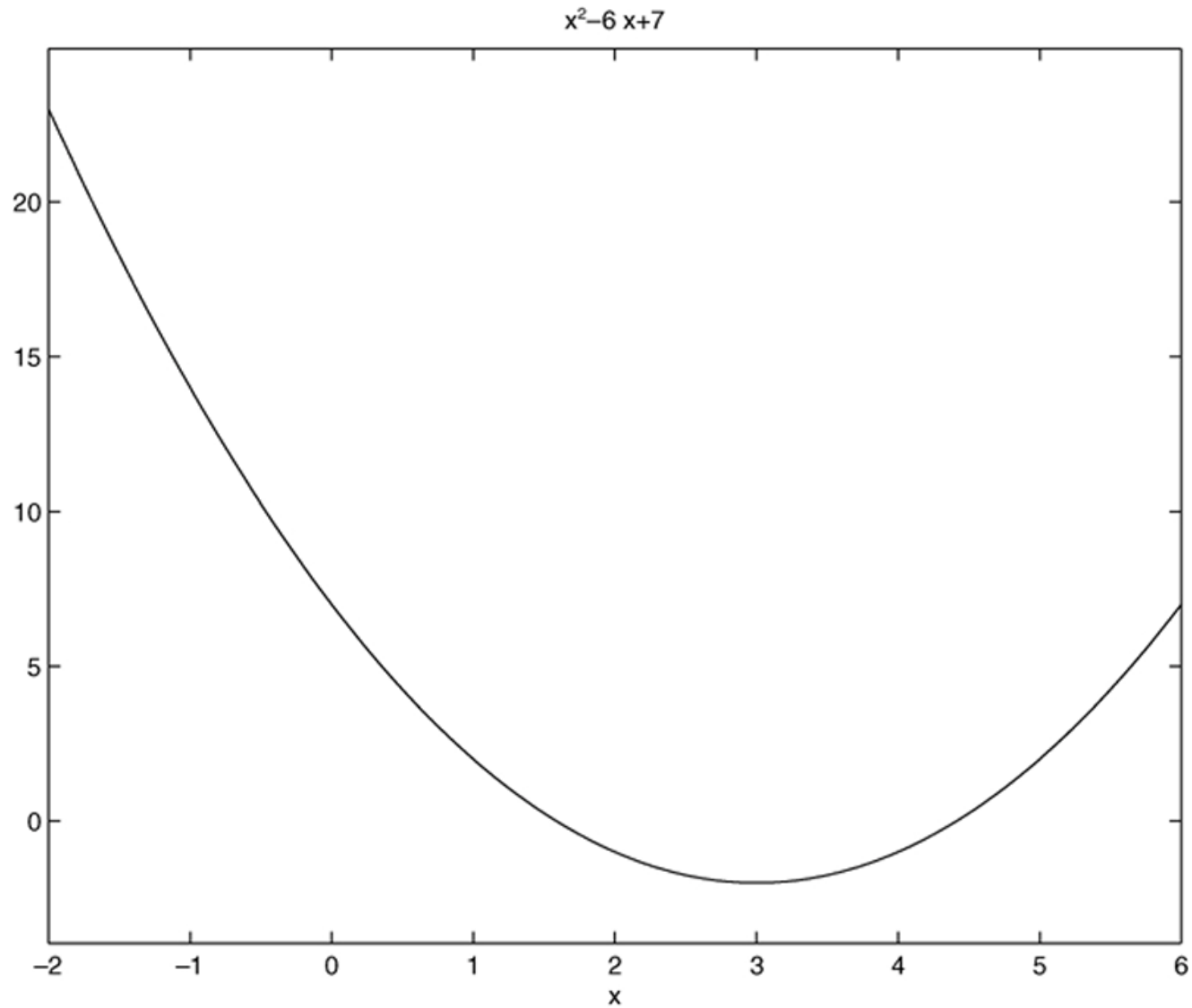
The **optional** form `ezplot(E, [xmin xmax])` generates a **plot** over the range from **xmin** to **xmax**.

For example, plot the $E = x^2 - 6x + 7$.

```
>> syms x
>> E = x^2 - 6*x + 7;
>> ezplot(E, [-2, 6])
```

See the graph in the **next** slide.

Plot of the function $E = x^2 - 6x + 7$ generated by the `ezplot`.



The `solve` function.

There are **three** ways to use the **solve** function.
For example, to **solve** the equation $x + 5 = 0$,
The **one** way is

```
>>eq1 = 'x+5=0' ;  
>>solve (eq1)  
ans =  
    -5
```

The **second** way is

```
>>solve ('x+5=0' )  
ans =  
    -5
```

The **third** way is

```
>>syms x
>>solve(x+5)
ans =
    -5
```

You can **store** the **result** in a **named** variable as follows:

```
>>syms x
>>x = solve(x+5)
x =
    -5
```

To solve the equation $e^{2x} + 3e^x = 54$, the session is

```
>>solve('exp(2*x)+3*exp(x)=54')
```

```
ans = [ log(-9)+i*pi]
```

```
      [ log(6) ]
```

```
>>eval(ans)      ans=  1.7918
```

Other examples: $2.1972 + 3.1416i$

```
>>eq2 = 'y^2+3*y+2=0' ;
```

```
>>solve(eq2)
```

```
ans =
```

```
      [-2]
```

```
      [-1]
```

```
>>eq3 = 'x^2+9*y^4=0' ;
```

```
>>solve(eq3) %Note that x is presumed to be  
the unknown variable
```

```
ans =
```

```
      [ 3*i*y^2]
```

```
      [-3*i*y^2]
```

When **more** than **one** variable occurs in the expression, MATLAB assumes that the variable closest to **x** in the **alphabet** is the variable to be found. You can specify the **solution** variable using the **syntax** `solve(E, 'v')`, where **v** is the solution variable.

For example: $x+5y^2-z+1=0$, $x+z+y=0$, $2z\sqrt{y}=0$

```
>>syms y z x
```

```
>>[x,y,z]=solve(x+5*y^2-z+1,x+y+z,2*z*sqrt(y))
```

Differentiation with the `diff` function.

```
>>syms n x y
```

```
>>diff(x^n)
```

```
ans =
```

```
    x^n*n/x
```

```
>>simplify(ans)
```

```
ans =
```

```
    x^(n-1)*n
```

```
>>diff(log(x))
```

```
ans =
```

```
    1/x
```

```
>>diff((sin(x))^2)
```

```
ans =
```

```
    2*sin(x)*cos(x)
```

If the expression contains **more** than **one** variable, the **diff** function **operates** on the variable **x**, or the variable **closest** to **x**, **unless** told to do otherwise. When there is **more** than **one** variable, the diff function computes the **partial derivative**.

```
>>diff(sin(x*y))  
ans =  
      cos(x*y)*y
```

The function **diff(E,v)** returns the **derivative** of the expression **E** with respect to the variable **v**.

```
>>syms x y  
>>diff(x*sin(x*y),y)  
ans =  
      x^2*cos(x*y)
```

The function `diff(E, n)` returns the ***n*th derivative** of the expression **E** with respect to the **default** independent variable.

```
>>syms x
>>diff(x^3,2)
ans =
    6*x
```

The function `diff(E, v, n)` returns the ***n*th derivative** of the expression **E** with respect to the variable **v**.

```
>>syms x y
>>diff(x*sin(x*y),y,2)
ans =
    -x^3*sin(x*y)
```

Integration with the `int` function.

```
>>syms x
```

```
>>int(2*x)
```

```
ans =
```

```
    x^2
```

The function `int(E)` returns the **integral** of the expression **E** with respect to the default **independent** variable.

```
>>syms n x y
```

```
>>int(x^n)
```

```
ans =
```

```
    x^(n+1) / (n+1)
```

```
>>int(1/x)
```

```
ans =
```

```
    log(x)
```

```
>>int(cos(x))
```

```
ans =
```

```
    sin(x)
```

The form `int(E, v)` returns the **integral** of the expression **E** with respect to the variable **v**.

```
>>syms n x
>>int(x^n,n)
ans =
      1/log(x) *x^n
```

The form `int(E, a, b)` returns the **integral** of the expression **E** with respect to the default **independent** variable evaluated over the interval **[a, b]**, where **a** and **b** are **numeric** expressions.

```
>>syms x
>>int(x^2,2,5)
ans =
      39
```

The form `int(E, v, a, b)` returns the **integral** of the expression **E** with respect to the variable **v** **evaluated** over the interval **[a, b]**, where **a** and **b** are **numeric** quantities.

```
>>syms x y
>>int(xy^2,y,0,5)
ans =
      125/3*x
```

The form `int(E, m, n)` returns the **integral** of the expression **E** with respect to the **default** independent variable **evaluated** over the interval **[m, n]**, where **m** and **n** are symbolic expressions.

```
>>syms t x
>>int(x,1,t)
ans =
      1/2*t^2-1/2
int(sin(x),t,exp(t))
ans =
      -cos(exp(t)) + cos(t)
```

Taylor Series.

The `taylor(f, n, a)` function gives the first `n-1` terms in the **Taylor series** for the function defined in the expression `f`, evaluated at the point `x = a`. If the **parameter a** is **omitted** the function returns the series evaluated at `x = 0`.

```
>>syms x
```

```
>>f = exp(x) ;
```

```
>>taylor(f, 4)
```

```
ans =
```

$$1+x+\frac{1}{2}x^2+\frac{1}{6}x^3$$

```
>>taylor(f, 3, 2)
```

```
ans =
```

$$\exp(2) + \exp(2) * (x-2) + \frac{1}{2} * \exp(2) * (x-2)^2$$

Series summation.

The `symsum(E, a, b)` function returns the **sum** of the expression **E** as the default symbolic variable varies from **a** to **b**.

```
>>syms k n
>>symsum(k, 0, 10)
ans =
    55
>>symsum(k^2, 1, 4)
ans =
    30
>>symsum(k, 0, n-1)
ans =
    1/2*n^2-1/2*n
```

Finding limits.

The basic form `limit(E)` finds the limit as $x \rightarrow 0$.

```
>>syms a x
>>limit(sin(a*x)/x)
ans =
     a
```

The form `limit(E,v,a)` finds the limit as $u \rightarrow a$.

```
>>syms h x
>>limit((x-3)/(x^2-9),3)
ans =
     1/6
>>limit((sin(x+h)-sin(x))/h,h,0)
ans =
    cos(x)
```

The forms `limit(E,v,a,'right')` and `limit(E,v,a,'left')` specify the direction of the limit.

```
>>syms x
```

```
>>limit(1/x,x,0,'left')
```

```
ans =
```

```
    -inf
```

```
>>limit(1/x,x,0,'right')
```

```
ans =
```

```
    inf
```

Solving differential equations with the `dsolve` function.

The `dsolve` function's syntax for solving a single equation is `dsolve ('eqn')`. The function returns a symbolic solution of the ODE specified by the symbolic expression `eqn`.

```
>>dsolve ('Dy+2*y=12')  
ans =  
6+C1*exp(-2*t)
```

There can be **symbolic constants** in the equation.

```
>>dsolve ('Dy=sin(a*t)')  
ans =  
(-cos(a*t)+C1*a)/a
```

Here is a **second-order** example:

```
dsolve('D2y=c^2*y')
```

```
ans =
```

```
C1*exp(-c*t) + C2*exp(c*t)
```

Sets of equations can be solved with **dsolve**. The appropriate syntax is **dsolve('eqn1', 'eqn2', ...)**.

```
>>[x, y] = dsolve('Dx=3*x+4*y', 'Dy=-4*x+3*y')  
x = C1*exp(3*t)*cos(4*t)+C2*exp(3*t)*sin(4*t)  
y = -C1*exp(3*t)*sin(4*t)+C2*exp(3*t)*cos(4*t)
```

Conditions on the solutions at specified values of the **independent** variable can be **handled** as follows.

The form

```
dsolve('eqn', 'cond1', 'cond2', ...)
```

returns a symbolic solution of the **ODE** specified by the symbolic expression **eqn**, subject to the conditions specified in the expressions **cond1**, **cond2**, and so on.

If **y** is the **dependent** variable, these **conditions** are specified as follows: $y(a) = b$, $Dy(a) = c$, $D^2y(a) = d$, and so on.

For example,

```
>>dsolve('D2y=c^2*y', 'y(0)=1', 'Dy(0)=0')  
ans =  
1/2*exp(c*t)+1/2*exp(-c*t)
```

Arbitrary boundary conditions, such as $y(0) = c$, can be used.

```
>>dsolve('Dy+a*y=b', 'y(0)=c')  
ans =  
1/a*b+exp(-a*t)*(-1/a*b+c)
```

Sets of equations with specified **boundary conditions** can be solved as follows.

The function

```
dsolve('eqn1', 'eqn2', ..., 'cond1', 'cond2', ...)
```

returns a symbolic solution of a set of equations specified by the symbolic expressions **eqn1**, **eqn2**, and so on, subject to the **initial conditions** specified in the expressions **cond1**, **cond2**, and so on.

For example,

```
>>dsolve('Dx=3*x+4*y', 'Dy=-4*x+3*y',  
'x(0)=0', 'y(0)=1')
```

```
[x,y] =  
      x = exp(3*t)*sin(4*t)  
      y = exp(3*t)*cos(4*t)
```

It is **not** necessary to specify only **initial conditions**. The conditions can be specified at different values of t .

```
>>dsolve('D2y+9*y=0', 'y(0)=1', 'Dy(pi)=2')  
ans =  
      -2/3*sin(3*t)+cos(3*t)
```

LAPLACE TRANSFORM

```
>>syms b t
>>laplace(t^3)
ans =
      6/s^4
>>laplace(exp(-b*t))
ans =
      1/(s+b)
>>laplace(sin(b*t))
ans =
      b/(s^2+b^2)
```

INVERSE TRANSFORM

```
>>syms b s
>>ilaplace(1/s^4)
ans =
      1/6*t^3
>>ilaplace(1/(s+b))
ans =
      exp(-b*t)
>>ilaplace(b/(s^2+b^2))
ans =
      sin(b*t)
```

Problem :

Find the intersection points between circle and line in xy plane

$$(x-2)^2+(y-4)^2=16, \quad y=x/2+1$$