Symbolic Expression in MATLAB using Symbolic Math ToolBox

- Symbolic Math Toolbox™ provides functions for solving, plotting, and manipulating symbolic math equations.
- The toolbox provides libraries of functions in common mathematical areas such as calculus, linear algebra, algebraic and ordinary differential equations, equation simplification, and equation manipulation.
- Symbolic Math Toolbox lets you analytically perform differentiation, integration, simplification, transforms, and equation solving.
- Your computations can be performed either analytically or using variable precision arithmetic, with the results displayed in mathematical typeset. Y

Create Symbolic Numbers:

- You can create symbolic numbers by using sym. Symbolic numbers are exact representations, unlike floating-point numbers.
- To Create a symbolic number by using sym and compare it to the same floating-point number.

```
>> sym(1/3)  % ans = 1/3
>> 1/3   % ans =     0.3333
```

- The symbolic number is represented in exact rational form, while the floating-point number is a decimal approximation.
- The symbolic result is not indented, while the standard MATLAB® result is indented.
- Calculations on symbolic numbers are exact.
- In following example, the symbolic result is exact, while the numeric result is an approximation.

```
>> sin(sym(pi))  % ans = 0
>> sin(pi)   %ans =  1.2246e-16
```

Create Symbolic Variables:

- You can use two ways to create symbolic variables, syms and sym. The syms syntax is a shorthand for sym.
- Create symbolic variables x and y using syms and sym respectively.

```
>> syms x
>> y = sym('y')
```

- The first command creates a symbolic variable x in the MATLAB workspace with the value x assigned to the variable x.
- The second command creates a symbolic variable y with value y. Therefore, the commands are equivalent.
- With syms, you can create multiple variables in one command. Create the variables a, b, and c.

```
>> syms a b c
```

Create Symbolic Expressions:

- Suppose you want to use a symbolic variable to represent the golden ratio

$$\varphi = \frac{1+\sqrt{5}}{2}$$

- The Command is

```
>> phi = (1 + sqrt(sym(5)))/2;
```

- Now you can perform various mathematical operations on phi. For example,
  >> f = phi^2 - phi - 1
- Now suppose you want to study the quadratic function $f = ax^2 + bx + c$. First, create the symbolic variables a, b, c, and x:
  >> syms a b c x, f = a*x^2 + b*x + c;

Create Symbolic Functions:

- You also can use sym and syms to create symbolic functions.
- For example, you can create an arbitrary function f(x, y) where x and yare function variables.
- The simplest way to create an arbitrary symbolic function is to use syms:
  >>syms f(x, y)
- This syntax creates the symbolic function f and symbolic variables x and y.
- If instead of an arbitrary symbolic function you want to create a function defined by a particular mathematical expression, use this two-step approach.
- First, create symbolic variables representing the arguments of the function:
  >> syms x y
- Then assign a mathematical expression to the function.
- In this case, the assignment operation also creates the new symbolic function:
  >> f(x, y) = x^3*y^3
- After creating a symbolic function, you can differentiate, integrate, or simplify it, substitute its arguments with values, and perform other mathematical operations.

Differentiation:

- **diff() :** This function is used to Differentiate symbolic expression or function
- **Syntax:**
  - diff(F) differentiates F with respect to the variable determined by symvar(F,1).
  - diff(F,var) differentiates F with respect to the variable var.
  - diff(F,n) computes the nth derivative of F with respect to the variable determined by symvar.
  - diff(F,var,n) computes the nth derivative of F with respect to the variable var.

  where, F — Expression or function to differentiate,
  var — Differentiation variable
  n — Differentiation order

Examples:

1. **Differentiation of Univariate Function**
   >> syms x
   >> f(x) = sin(x^2);
   >> df = diff(f,x) % df(x) = 2*x*cos(x^2)

2. **Differentiation with Respect to Particular Variable**
   >> syms x t
   >> diff(sin(x*t^2)) % ans = t^2*cos(t^2*x)

   Note: Because you did not specify the differentiation variable, diff uses the default variable defined by symvar. For this expression, the default variable is x:
   >> symvar(sin(x*t^2),1) %ans = x

   Now, find the derivative of this expression with respect to the variable t:
   >> diff(sin(x*t^2),t) %ans = 2*t*x*cos(t^2*x)

3. **Higher-Order Derivatives of Univariate Expression**

- To find the 4th, 5th, and 6th derivatives of this expression:

```
>> syms t
>> d4 = diff(t^6,4),  d5 = diff(t^6,5), d6 = diff(t^6,6)
% d4 = 360*t^2, d5 = 720*t,  d6 = 720
```

4. **Higher-Order Derivatives of Multivariate Expression with Respect to Particular Variable**

- To Find the second derivative of this expression with respect to the variable y:

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

5. **Mixed Derivatives**

- To Differentiate this expression with respect to the variables x and y:

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

- You also can compute mixed higher-order derivatives by providing all differentiation variables:

```
>> syms x y
>> diff(x*sin(x*y), x, x, x, y)
% ans = x^2*y^3*sin(x*y) - 6*x*y^2*cos(x*y) - 6*y*sin(x*y)
```

Points to remember:

To improve performance, diff assumes that all mixed derivatives commute. For example,

$$\frac{\partial}{\partial x}\frac{\partial}{\partial y}f(x,y) = \frac{\partial}{\partial y}\frac{\partial}{\partial x}f(x,y)$$

int() for calculating integrals:

- **int():** This function is used for Definite and indefinite integrals.
- **Syntax and description**
  - int(expr,var): computes the indefinite integral of expr with respect to the symbolic scalar variable var. Specifying the variable var is optional. If you do not specify it, int uses the default variable determined by symvar. If expr is a constant, then the default variable is x.
  - int(expr,var,a,b): computes the definite integral of expr with respect to var from a to b. If you do not specify it, int uses the default variable determined by symvar. If expr is a constant, then the default variable is x.
  - int(expr,var,[a,b]), int(expr,var,[a b]), and int(expr,var,[a;b]) are equivalent to int(expr,var,a,b).

where,

  - expr — Integrand, specified as a symbolic expression or function, a constant, or a vector or matrix of symbolic expressions, functions, or constants.
  - var — Integration variable, specified as a symbolic variable. If you do not specify this variable, int uses the default variable determined bysymvar(expr,1). If expr is a constant, then the default variable is x.

- a – lower bound
- b - upper bound

Examples:

1. **Indefinite Integral of Univariate Expression**

   To Find an indefinite integral of this univariate expression:

   ```
   >> syms x
   >> int(-2*x/(1 + x^2)^2) % ans = 1/(x^2 + 1)
   ```

2. **Indefinite Integrals of Multivariate Expression**

   To find indefinite integrals of this multivariate expression with respect to the variables x and z:

   ```
   >> syms x z
   >> int(x/(1 + z^2), x), int(x/(1 + z^2), z)
   % ans = x^2/(2*(z^2 + 1)),  ans = x*atan(z)
   ```

   If you do not specify the integration variable, int uses the variable returned by symvar. For this expression, symvar returns x:

   ```
   >> symvar(x/(1 + z^2), 1)  % ans = x
   ```

3. **Definite Integrals of Univariate Expressions**

   To Integrate this expression from 0 to 1:

   ```
   >> syms x
   >> int(x*log(1 + x), 0, 1)
   % ans = 1/4
   ```

   To Integrate this expression from sin(t) to 1 specifying the integration range as a vector:

   ```
   >> syms x t
   >> int(2*x, [sin(t), 1]) % ans = cos(t)^2
   ```

4. **Integrals of Matrix Elements**

   To Find indefinite integrals for the expressions listed as the elements of a matrix:

   ```
   >>  syms a x t z
   >>  int([exp(t), exp(a*t); sin(t), cos(t)])
   % ans = [  exp(t), exp(a*t)/a] [ -cos(t),     sin(t)]
   ```

5. **Apply IgnoreAnalyticConstraints**

   To Compute this indefinite integral. By default, int uses strict mathematical rules. These rules do not let int rewrite asin(sin(x)) andacos(cos(x)) as x.

   ```
   >> syms x
   >> int(acos(sin(x)), x)
   % ans = x*acos(sin(x)) + (x^2*sign(cos(x)))/2
   ```

   If you want a simple practical solution, try IgnoreAnalyticConstraints:

   ```
   >> int(acos(sin(x)), x, 'IgnoreAnalyticConstraints', true)
   % ans = -(x*(x - pi))/2
   ```

6. **Ignore Special Cases**

   To Compute this integral with respect to the variable x:

   ```
   >> syms x t
   >>  int(x^t, x)
   ```

   - By default, int returns the integral as a piecewise object where every branch corresponds to a particular value (or a range of values) of the symbolic parameter t:

     ```
     %ans = piecewise(t == -1, log(x), t ~= -1, x^(t + 1)/(t + 1))
     ```

   To ignore special cases of parameter values, use IgnoreSpecialCases:

   ```
   >> int(x^t, x, 'IgnoreSpecialCases', true)
   ```

   With this option, int ignores the special case t=-1 and returns only the branch where t<>–1:

```
% ans = x^(t + 1)/(t + 1)
```

7. **Find Cauchy Principal Value**

To compute this definite integral, where the integrand has a pole in the interior of the interval of integration. Mathematically, this integral is not defined.

```
>>syms x
>> int(1/(x - 1), x, 0, 2)
% ans = NaN
```

However, the Cauchy principal value of the integral exists. Use PrincipalValue to compute the Cauchy principal value of the integral:

```
>>  int(1/(x - 1), x, 0, 2, 'PrincipalValue', true)
% ans = 0
```

8. **Approximate Indefinite Integrals**

If int cannot compute a closed form of an integral, it returns an unresolved integral:

```
>>  syms x
>>  F = sin(sinh(x));
>>  int(F, x) % ans = int(sin(sinh(x)), x)
```

If int cannot compute a closed form of an indefinite integral, try to approximate the expression around some point using taylor, and then compute the integral. For example, approximate the expression around $x = 0$:

```
>> int(taylor(F, x, 'ExpansionPoint', 0, 'Order', 10), x)
% ans = x^10/56700 - x^8/720 - x^6/90 + x^2/2
```

9. **Approximate Definite Integrals**

To compute this definite integral:

```
>>  syms x
>>  F = int(cos(x)/sqrt(1 + x^2), x, 0, 10)
% F = int(cos(x)/(x^2 + 1)^(1/2), x, 0, 10)
```

If int cannot compute a closed form of a definite integral, try approximating that integral numerically using vpa. For example, approximateF with five significant digits:

```
>> vpa(F, 5)
% ans = 0.37571
```

Integral() for calculating numerical integration

- **integral() :** function used to calculate Numerical integration
- **Syntax:**

```
q = integral(fun,xmin,xmax):
```

- numerically integrates function fun from xmin to xmax using global adaptive quadrature and default error tolerances.

Here,

- fun — Integrand specified as a function handle, which defines the function to be integrated from xmin to xmax.
- xmin — Lower limit of x
- xmax — Upper limit of x

- **Examples**

1. **Improper Integral** : Create the function $f(x) = e^{-x^2}(\ln x)^2$. Evaluate the integral from x=0 to x=Inf.

```
>> fun = @(x) exp(-x.^2).*log(x).^2;
>> q = integral(fun,0,Inf)
% q = 1.9475
```

2. **Parameterized Function:** Create the function $f(x) = 1/(x^3 - 2x - c)$ with one parameter, c. Evaluate the integral from x=0 to x=2 at c=5.

```
>> fun = @(x,c) 1./(x.^3-2*x-c);
>> q = integral(@(x)fun(x,5),0,2)
% q = -0.4605
```

3. **Singularity at Lower Limit:** Create the function $f(x) = \ln(x)$. Evaluate the integral from x=0 to x=1 with the default error tolerances.

```
>> fun = @(x)log(x);
>> format long
>> q1 = integral(fun,0,1)
% q1 =  -1.000000010959678
```

4. **Vector-Valued Function:** Create the vector-valued function $f(x) = [\sin x,\ \sin 2x,\ \sin 3x,\ \sin 4x,\ \sin 5x]$ and integrate from x=0 to x=1. Specify 'ArrayValued',true to evaluate the integral of an array-valued or vector-valued function.

```
>> fun = @(x)sin((1:5)*x);
>> q = integral(fun,0,1,'ArrayValued',true)
% q =    0.4597    0.7081    0.6633    0.4134    0.1433
```

5. **Improper Integral of Oscillatory Function:** Create the function $f(x) = x^5 e^{-x} \sin x$. Evaluate the integral from x=0 to x=Inf

```
>> fun = @(x)x.^5.*exp(-x).*sin(x);
>> format long
>> q = integral(fun,0,Inf,)
% q =  -14.999999999998364
```

gamma(): Gamma function

- **Syntax:**

```
Y = gamma(X)
```

- function returns the gamma function evaluated at the elements of X.

Examples:

- **Evaluate Gamma Function:** Evaluate the gamma function with a scalar and a vector.

1. Evaluate $\Gamma(0.5)$, which is equal to $\sqrt{\pi}$.

```
>> y = gamma(0.5) % y = 1.7725
```

2. Evaluate several values of the gamma function between [-3.5 3.5].

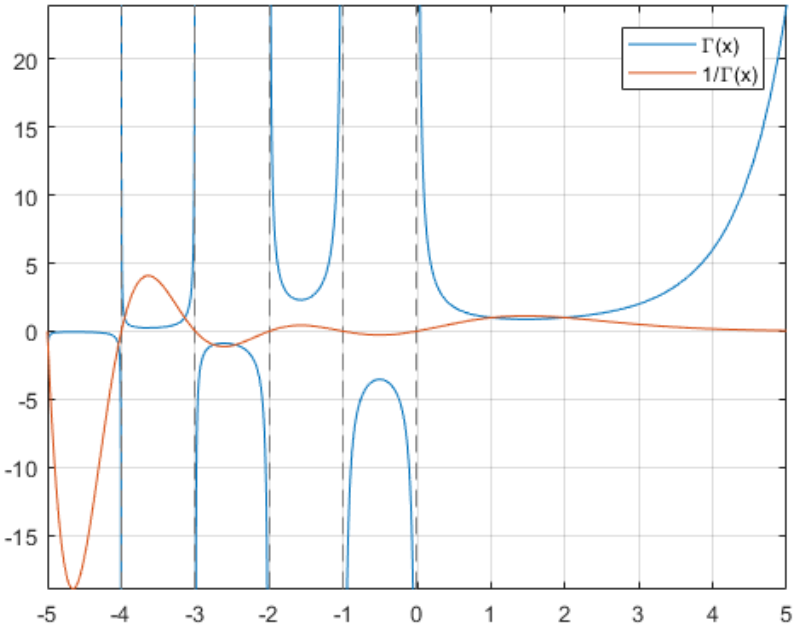
```
>> x = -3.5:3.5, y = gamma(x)

% y =    0.2701   -0.9453    2.3633   -3.5449    1.7725    0.8862    1.3293

% 3.3234
```

- **Plot Gamma Function:** Plot the gamma function and its inverse.

1. Use fplot to plot the gamma function and its inverse.

2. The gamma function increases quickly for positive arguments and has simple poles at all negative integer arguments (as well as 0).
3. The function does not have any zeros. Conversely, the inverse gamma function has zeros at all negative integer arguments (as well as 0).

```
>> fplot(@gamma), hold on, fplot(@(x) 1./gamma(x))

>> legend('\Gamma(x)','1/\Gamma(x)'), hold off, grid on
```



beta(): Beta function

- **Syntax:** beta(x,y)
- Function returns the beta function of x and y.
- Here,
  - x : Symbolic number, variable, expression, function, or a vector or matrix of symbolic numbers, variables, expressions, or functions. If x is a vector or matrix, beta returns the beta function for each element of x
  - y: Symbolic number, variable, expression, function, or a vector or matrix of symbolic numbers, variables, expressions, or functions. If y is a vector or matrix, beta returns the beta function for each element of y

Examples:

1. **Compute the beta function for these numbers. Because these numbers are not symbolic objects, you get floating-point results:**

```
>> [beta(1, 5), beta(3, sqrt(2)), beta(pi, exp(1)), beta(0, 1)]

% ans =    0.2000    0.1716    0.0379       Inf
```

2. **Compute the beta function for the numbers converted to symbolic objects:**

```
>> [beta(sym(1), 5), beta(3, sym(2)), beta(sym(4), sym(4))]
```

```
% ans = [ 1/5, 1/12, 1/140]
```

3. **If one or both parameters are complex numbers, convert these numbers to symbolic objects:**

```
>> [beta(sym(i), 3/2), beta(sym(i), i), beta(sym(i + 2), 1 - i)]

% ans = [ (pi^(1/2)*gamma(1i))/(2*gamma(3/2 + 1i)),
gamma(1i)^2/gamma(2i),...

%  (pi*(1/2 + 1i/2))/sinh(pi)]
```

4. **Compute the beta function for negative parameters. If one or both arguments are negative numbers, convert these numbers to symbolic objects:**

```
>> [beta(sym(-3), 2), beta(sym(-1/3), 2), beta(sym(-3), 4),  beta(sym(-3), -2)]

% ans = [ 1/6, -9/2, Inf, Inf]
```

5. **Call beta for the matrix A and the value 1. The result is a matrix of the beta functions beta(A(i,j),1):**

```
>> A = sym([1 2; 3 4]), beta(A,1)

% ans = [   1, 1/2] [ 1/3, 1/4]
```

6. **Differentiate the beta function, then substitute the variable t with the value 2/3 and approximate the result using vpa:**

```
>> syms t

>> u = diff(beta(t^2 + 1, t)), vpa(subs(u, t, 2/3), 10)

% u =  beta(t, t^2 + 1)*(psi(t) + 2*t*psi(t^2 + 1) -... psi(t^2 + t + 1)*(2*t + 1))

% ans = -2.836889094
```

7. **Expand these beta functions:**

```
>> syms x y

>> expand(beta(x, y)),  expand(beta(x + 1, y - 1))

% ans = (gamma(x)*gamma(y))/gamma(x + y)

% ans = -(x*gamma(x)*gamma(y))/(gamma(x + y) - y*gamma(x + y))
```

Points to Remember:

1. The beta function is uniquely defined for positive numbers and complex numbers with positive real parts. It is approximated for other numbers.
2. Calling beta for numbers that are not symbolic objects invokes the MATLAB® beta function. This function accepts real arguments only. If you want to compute the beta function for complex numbers, use sym to convert the numbers to symbolic objects, and then call beta for those symbolic objects.
3. If one or both parameters are negative numbers, convert these numbers to symbolic objects using sym, and then call beta for those symbolic objects.

4. If the beta function has a singularity, beta returns the positive infinity Inf.
5. beta(sym(0),0), beta(0,sym(0)), and beta(sym(0),sym(0)) return NaN.
6. beta(x,y) = beta(y,x) and beta(x,A) = beta(A,x).
7. At least one input argument must be a scalar or both arguments must be vectors or matrices of the same size. If one input argument is a scalar and the other one is a vector or a matrix, beta(x,y) expands the scalar into a vector or matrix of the same size as the other argument with all elements equal to that scalar.

erf(): Error function

Syntax:

erf(X): erf(X) represents the error function of X. If X is a vector or a matrix, erf(X) computes the error function of each element of X.

Examples:

1. **Error Function for Floating-Point and Symbolic Numbers:** Depending on its arguments, erf can return floating-point or exact symbolic results. Compute the error function for these numbers. Because these numbers are not symbolic objects, you get the floating-point results:

   >> A = [erf(1/2), erf(1.41), erf(sqrt(2))]

   % A = 0.5205 0.9539 0.9545

2. **Compute the error function for the same numbers converted to symbolic objects. For most symbolic (exact) numbers, erf returns unresolved symbolic calls:**

   >> symA = [erf(sym(1/2)), erf(sym(1.41)), erf(sqrt(sym(2)))]

   % symA = [erf(1/2), erf(141/100), erf(2^(1/2))]

3. **Use vpa to approximate symbolic results with the required number of digits:**

   >> vpa (symA,10)

   % ans = [0.5204998778, 0.9538524394, 0.9544997361]

4. **Error Function for Variables and Expressions:** For most symbolic variables and expressions, erf returns unresolved symbolic calls. Compute the error function for x and sin(x) + x*exp(x):

   >> syms x

   >> f = sin(x) + x*exp(x);

   >> erf(x) % ans = erf(x)

   >> erf(f) % ans = erf(sin(x) + x*exp(x))

5. **Error Function for Vectors and Matrices:** If the input argument is a vector or a matrix, erf returns the error function for each element of that vector or matrix. Compute the error function for elements of matrix M and vector V:

   >> M = sym([0 inf; 1/3 -inf]);

```
>> V = sym([1; -i*inf]);

>> erf(M)

% ans = [        0,  1] [ erf(1/3), -1]

>> erf(V)   % ans =  erf(1)  -Inf*1i
```

6. **Special Values of Error Function:** erf returns special values for particular parameters. Compute the error function for x = 0, x = ∞, and x = –∞. Use sym to convert 0 and infinities to symbolic objects. The error function has special values for these parameters:

```
>> [erf(sym(0)), erf(sym(Inf)), erf(sym(-Inf))]

% ans = [ 0, 1, -1]
```

7. **Compute the error function for complex infinities. Use sym to convert complex infinities to symbolic objects:**

```
>> [erf(sym(i*Inf)), erf(sym(-i*Inf))]

% ans = [ Inf*1i, -Inf*1i]
```

8. **Handling Expressions That Contain Error Function:** Many functions, such as diff and int, can handle expressions containing erf.
   a. **Compute the first and second derivatives of the error function:**

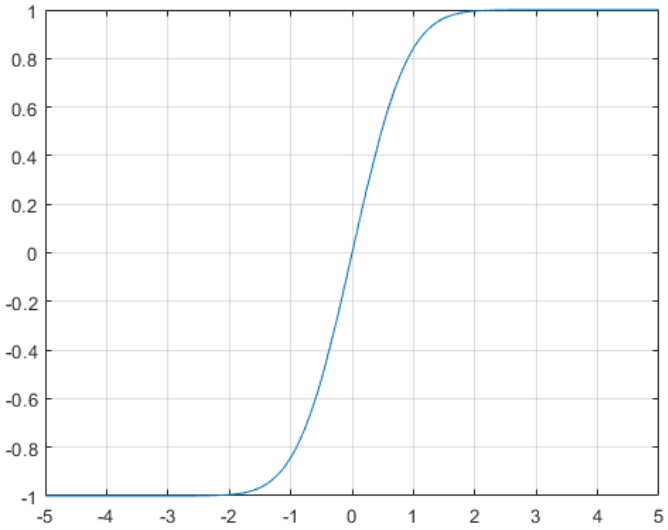
```
>> syms x

>> diff(erf(x), x)

>> diff(erf(x), x, 2)

%ans = (2*exp(-x^2))/pi^(1/2)

% ans = -(4*x*exp(-x^2))/pi^(1/2)
```

   b. **Compute the integrals of these expressions:**

```
>> int(erf(x), x)

>> int(erf(log(x)), x)

% ans = exp(-x^2)/pi^(1/2) + x*erf(x)

%  ans =  x*erf(log(x)) - int((2*exp(-log(x)^2))/pi^(1/2), x)
```

Plot Error Function: Plot the error function on the interval from -5 to 5.

```
>> syms x, fplot(erf(x),[-5,5]),  grid on
```



Points to remember:

1. Calling erf for a number that is not a symbolic object invokes the MATLAB® erf function. This function accepts real arguments only. If you want to compute the error function for a complex number, use symto convert that number to a symbolic object, and then call erf for that symbolic object.
2. For most symbolic (exact) numbers, erf returns unresolved symbolic calls. You can approximate such results with floating-point numbers using vpa.

References:

1) Gautschi, W. "Error Function and Fresnel Integrals." Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables. (M. Abramowitz and I. A. Stegun, eds.). New York: Dover, 1972.

2) Zelen, M. and N. C. Severo. "Probability Functions." Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables. (M. Abramowitz and I. A. Stegun, eds.). New York: Dover, 1972,

3) **MATLAB Documentation**