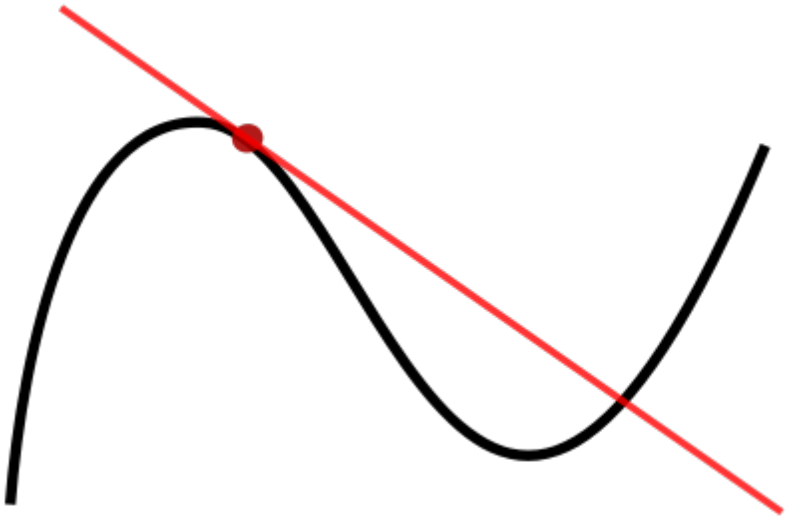
Symbolic Arithmetic

- Symbolic Arithmetic in MatLab is based on the Maple kernel integrated in the Symbolic Math Toolboxes and allow the user to perform calculations symbolically in the MatLab environment.

- The computational engine underlying the toolboxes is the kernel of Maple, a symbolic mathematical system developed primarily at the University of Waterloo, Canada and, more recently, at the Eidgenössiche Technische Hochschule, Zürich, Switzerland. Maple is marketed and supported by Waterloo Maple, Inc.

Symbolic Differentiation

- Differentiation: find the slope of the tangent line of a function. Consider this example from wikipedia: The black curve shows the function plot.



The red line shows the tangent line at a curve point, the slope of this

tangent line is the derivative value.

- Consider an example of symbolic differentiation in

  L15differentiation_example.m:

```
% x and y are symbolic variables
syms x y
f=x^3*cos(x)-y^2*sin(log(x))
% differentiate f with respect to x
fx=diff(f,x)
% differentiate f with respect to y
fy=diff(f,y)
gradient_magnitude=sqrt(fx^2+fy^2)
fprintf('Simplification of gradient magnitude expression');
simplify(gradient_magnitude)
```

The output of this code is:

```
f =
x^3*cos(x) - y^2*sin(log(x))

fx =
3*x^2*cos(x) - x^3*sin(x) - (y^2*cos(log(x)))/x

fy =
-2*y*sin(log(x))

gradient_magnitude =
((x^3*sin(x) - 3*x^2*cos(x) +
(y^2*cos(log(x)))/x)^2 +
4*y^2*sin(log(x))^2)^(1/2)

Simplification of gradient magnitude expression
ans =
((x^3*sin(x) - 3*x^2*cos(x) +
(y^2*cos(log(x)))/x)^2 +
4*y^2*sin(log(x))^2)^(1/2)
```

This code sets $f(x,y) = x^3\cos(x) - y^2\sin(\log(x))$ and computes:

$$\begin{aligned} f_x &= 3x^2\cos(x) - x^3\sin(x) - \frac{(y^2\cos(\log(x)))}{x} \quad \text{and} \\ f_y &= \frac{\partial f}{\partial y} = -2y\sin(\log(x)). \end{aligned}$$

The magnitude of the gradient (f_x, f_y) is found as:

$$||(f_x,f_y)||_2 = \sqrt{f_x^2 + f_y^2} = \left[\left(x^3\sin(x) - 3x^2\cos(x) + \frac{(y^2\cos(\log(x)))}{x}\right)^2 + 4y^2\sin(\log(x))^2\right]^{(1/2)}.$$

Simplification of gradient magnitude expression yields the same expression: it is already simplified.

- Continuing, the MatLab code:

```
x=2; y=3;
fprintf('fx=');
eval(fx)
fprintf('fy=');
eval(fy)
fprintf('Gradient magnitude=');
eval(gradient_magnitude)
```

produces the output:

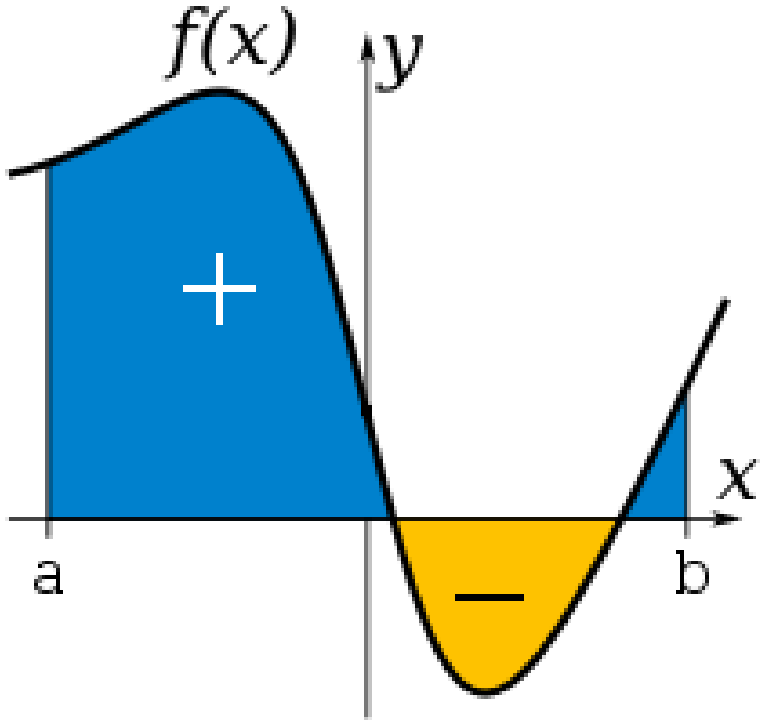
```
fx=
ans =
  -15.7297
fy=
ans =
   -3.8338
Gradient magnitude=
ans =
   16.1902
```

- That is, when we set $x = 2$ and $y = 3$ we can use the function `eval` to evaluate f_x, f_y and $||(f_x, f_y)||_2$ numerically as -15.7297, -3.8338 and 16.1902, respectively.

- MatLab also supports numerical differentiation. For example, **del2** compute the discrete Laplacian, **diff** computes differences as approximations to derivatives, **gradient** computes the numerical gradient and **polyder** computes polynomial differentiation.

Symbolic Integration

- Integration is finding the area under a curve. More precisely, $\int_a^b f(x)$ is defined as the signed area of the region in the xy-plane bounded by the graph of f, the x-axis, and the vertical lines $x = a$ and $x = b$, such that area above the x-axis adds to the total while the area below the x-axis subtracts from the total The figure below shows the situation:



The "blue" areas between a and b are added to the integral's value while the yellow area is subtracted from the integral's value.

- A trivial example (the >> shows the answer returned by MatLab):

```
syms f
f=x^2
>> f = x^2
int(f,x)
>> ans = x^3/3
```

We see that $\int x^2 = \frac{x^3}{3}$.

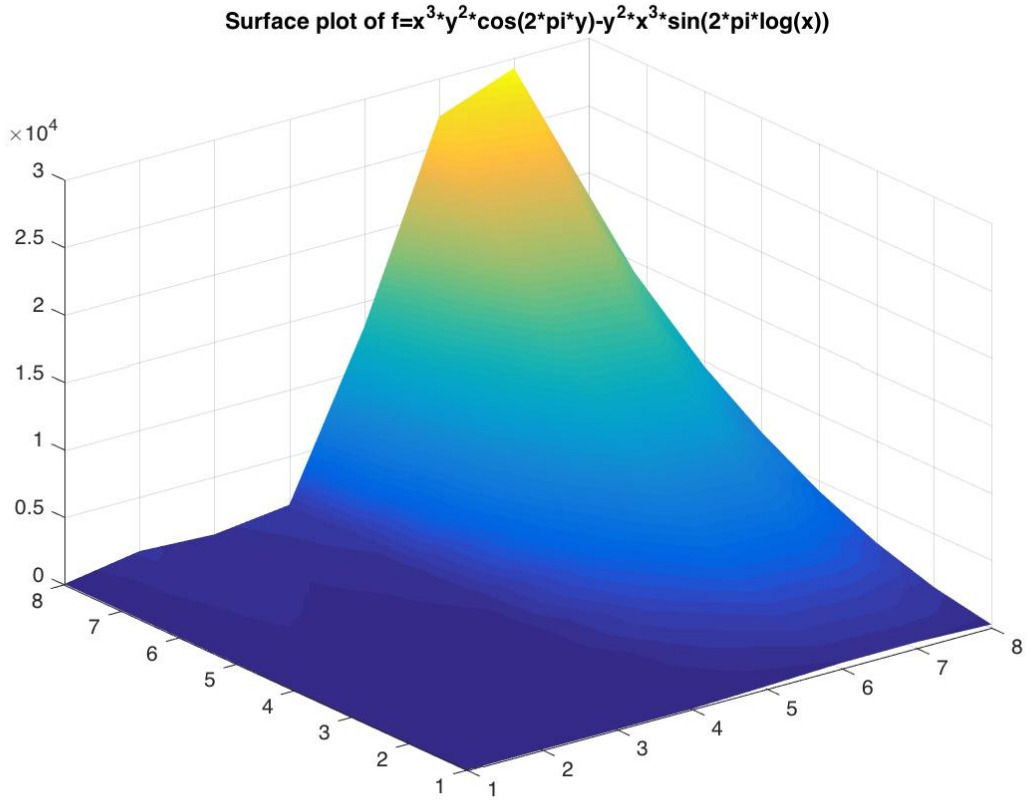
- For a more complex example, consider the following MatLab code in **L15integration_example.m**:

```
syms f x y
fprintf('%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%\n');
fprintf('The function:\n');
f=x^3*y^2*cos(2*pi*y)-y^2*x^3*sin(2*pi*log(x))

% set limits for integration
xmin=1;
xmax=8;
ymin=1;
ymax=8;

% plot the surface - the areas under the surface
% is the integral's value
[X,Y]=meshgrid(xmin:xmax,ymin:ymax);
Z=X.^3*Y.^2.*cos(2*pi.*Y)-Y.^2*X.^3.*sin(2*pi.*log(X));
surf(X,Y,Z)
% use smooth shading
shading interp
title('Surface plot of f=x^3*y^2*cos(2*pi*y)-y^2*x^3*sin(2*pi*log(x))');
print integral_example.jpg -djpeg
```

- This code integrates the function:

$$f(x, y) = x^3 y^2 \cos(2\pi y) - y^2 x^3 \sin(2\pi \log(x))$$

The figure below shows a plot of this surface with x and y ranging from 1 to 8.



Surface plot of $f(x, y) = x^3 y^2 \cos(2\pi y) - y^2 x^3 \sin(2\pi \log(x))$ for both x and y ranging from 1 to 8.

- The commands `int(f,x)`, `int(f,y)` and `int(f,x,y)` integrate this function with respect to x and y (we get 2 symbolic functions in terms of y and x) and a symbolic expression of the double integral with respect to x and then y. These are called **indefinite** integrals as the result can only be evaluated symbolically. Continuing with the code:

```
fprintf('%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%\n');
% integrate indefinite integral of f with respect to x
fprintf('Integration of f wrt to x');
int(f,x)

% integrate indefinite integral of f with respect to y
fprintf('Integration of f wrt to y');
int(f,y)

% integrate indefinite integral of f with respect to x and y
fprintf('Integration of f wrt to x and then y');
int(f,x,y)
```

We get the output:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Integration of f wrt to x
ans = (x^4*y^2*(cos(2*pi*y) -
      sin(2*pi*log(x)) +
      (pi^2*cos(2*pi*y))/4 +
      (pi*cos(2*pi*log(x)))/2))/(pi^2 + 4)
Integration of f wrt to y
ans = (x^3*y^2*sin(2*pi*y))/(2*pi) -
      (x^3*sin(2*pi*y))/(4*pi^3) -
      (x^3*y^3*sin(2*pi*log(x)))/3 +
      (x^3*y*cos(2*pi*y))/(2*pi^2)
Integration of f wrt to x and then y
ans = y^2*((x^4*(sin(2*pi*log(x)) -
      (pi*cos(2*pi*log(x)))/2))/(pi^2 + 4) -
      (y^4*(sin(2*pi*log(y)) -
      (pi*cos(2*pi*log(y)))/2))/(pi^2 + 4)) +
      (y^6*cos(2*pi*y))/4 - (x^4*y^2*cos(2*pi*y))/4
```

- We can evaluate the integral of f between limits 1 and 8 in x and y:

```
fprintf('%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%\n');
% evaluate definite integral of f from xmin to xmax in x
fprintf('Evaluate f wrt x from %d to %d\n',xmin,xmax);
fprintf('Answer is a symbolic expression in terms of y\n');
int(f,x,xmin,xmax)

% evaluate definite integral of f from ymin to ymax in y
fprintf('Evaluate f wrt y from %d to %d\n',ymin,ymax);
fprintf('Answer is a symbolic expression in terms of x\n');
int(f,y,ymin,ymax)
```

to get:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Evaluate f wrt x from 1 to 8
Answer is a symbolic expression in terms of y
ans = (4095*y^2*cos(2*pi*y))/4 -
      (y^2*(4096*sin(2*pi*log(8)) -
       pi*(2048*cos(2*pi*log(8))-1/2)))/(pi^2 + 4)

Evaluate f wrt y from 1 to 8
Answer is a symbolic expression in terms of x
ans = (7*x^3)/(2*pi^2) - (511*x^3*sin(2*pi*log(x)))/3
```

- We can numerically evaluate these integrals symbolically and numerically using:

```
fprintf('%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%\n');
% evaluate definite integral of f from ymin to ymax in y

fprintf('Evaluate f wrt to y between %d and %d as fy\n',ymin,ymax);
fy=int(f,y,ymin,ymax)
fprintf('Evaluate fy wrt to x between %d and %d\n',xmin,xmax);
% correct answer given symbolically
int(fy,x,xmin,xmax)
% need to evaluate the expression
% to get numerical answer
fprintf('Numerical evaluation:');
eval(int(fy,x,xmin,xmax))
```

to get:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Evaluate f wrt to y between 1 and 8 as fy
fy = (7*x^3)/(2*pi^2) - (511*x^3*sin(2*pi*log(x)))/3

Evaluate fy wrt to x between 1 and 8
ans = 28665/(8*pi^2) - (511*(4096*sin(2*pi*log(8)) -
```

```
      pi*(2048*cos(2*pi*log(8)) - 1/2)))/(3*(pi^2 + 4))

Numerical evaluation:
ans = 4.5640e+04
```

while the MatLab code:

```
fprintf('%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%\n');
fprintf('Evaluate f wrt to x between %d and %d as fx\n',xmin,xmax);
fx=int(f,x,xmin,xmax)
fprintf('Evaluate fx wrt to y between %d and %d\n',ymin,ymax);
% correct answer given symbolically
int(fx,y,ymin,ymax)
% need to evaluate the expression
% to get numerical answer
fprintf('Numerical evaluation:');
eval(int(fx,y,ymin,ymax))
```

produces:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Evaluate f wrt to x between 1 and 8 as fx
fx = (4095*y^2*cos(2*pi*y))/4 -
```

```
      (y^2*(4096*sin(2*pi*log(8)) -
       pi*(2048*cos(2*pi*log(8)) - 1/2)))/(pi^2 + 4)

Evaluate fx wrt to y between 1 and 8
ans = 28665/(8*pi^2) - (511*(4096*sin(2*pi*log(8)) -
      pi*(2048*cos(2*pi*log(8)) - 1/2)))/(3*(pi^2 + 4))

Numerical evaluation:
ans = 4.5640e+04
```

- To fully evaluate a double integral we first evaluate in either x or y as above but using the limits 1 to 8 for both x and y and then evaluate the resulting expression in y or x, again with limits 1 to 8, to get the final result. The results below are for integration both ways (we get the same result). Note that the result is first expressed symbolically but when we `eval` it we get a number (the symbolic expression and the number are the same). This is called **definite** integration as the limits in both x and

y are used. The result is always a number. The MatLab code shows this relationship is always true:

```
fprintf('%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%\n');
fprintf('Evaluate double integral of f directly\n');
fprintf('wrt x from %d to %d and then',xmin,xmax);
fprintf('wrt y from %d to %d\n',ymin,ymax);
int(int(f,y,ymin,ymax),x,xmin,xmax)
% need to evaluate the expression
% to get numerical answer
fprintf('Numerical evaluation:');
eval(int(int(f,y,ymin,ymax),x,xmin,xmax))

fprintf('\nEvaluate double integral of f directly\n');
fprintf('wrt y from %d to %d and then',ymin,ymax);
fprintf('wrt x from %d to %d\n',xmin,xmax);
int(int(f,x,xmin,xmax),y,ymin,ymax)
% need to evaluate the expression
% to get numerical answer
fprintf('Numerical evaluation:');
eval(int(int(f,x,xmin,xmax),y,ymin,ymax))
```

which produces:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Evaluate double integral of f directly
wrt x from 1 to 8 and then wrt y from 1 to 8
ans = 28665/(8*pi^2) - (511*(4096*sin(2*pi*log(8)) -
      pi*(2048*cos(2*pi*log(8)) - 1/2)))/(3*(pi^2 + 4))

Numerical evaluation:
ans = 4.5640e+04

Evaluate double integral of f directly
wrt y from 1 to 8 and then wrt x from 1 to 8
ans = 28665/(8*pi^2) - (511*(4096*sin(2*pi*log(8)) -
      pi*(2048*cos(2*pi*log(8)) - 1/2)))/(3*(pi^2 + 4))

Numerical evaluation:
ans = 4.5640e+04
```

- Try this double integration at home with paper and pen if you don't believe the results!!!

Numerical Integration

- We can always verify the numerical evaluation of symbolic integration by using the **integral2** function, which computes the integral using numerical methods.

- Consider the following double integration done in the previous section. The following MatLab code (in **L15numerical_integration1.m**):

```
syms f x y

% set limits for integration
xmin=1;
xmax=8;
ymin=1;
ymax=8;

%% Symbolic Integration
f=x^3*y^2*cos(2*pi*y)-y^2*x^3*sin(2*pi*log(x));
fprintf('The value of  symbolic integration: %20.12f\n',...
```

```
        eval(int(int(f,y,ymin,ymax),x,xmin,xmax)));

% Numerical Integration
[X,Y]=meshgrid(xmin:xmax,ymin:ymax);
fun=@(X,Y) (X.^3.*Y.^2.*cos(2*pi*Y)-Y.^2.*X.^3.*sin(2*pi*log(X));
fprintf('The value of numerical integration: %20.12f\n',...
        integral2(fun,xmin,xmax,ymin,ymax));
```

produces the output:

```
The value of  symbolic integration:   45640.283341172450
The value of numerical integration:   45640.283383236834
```

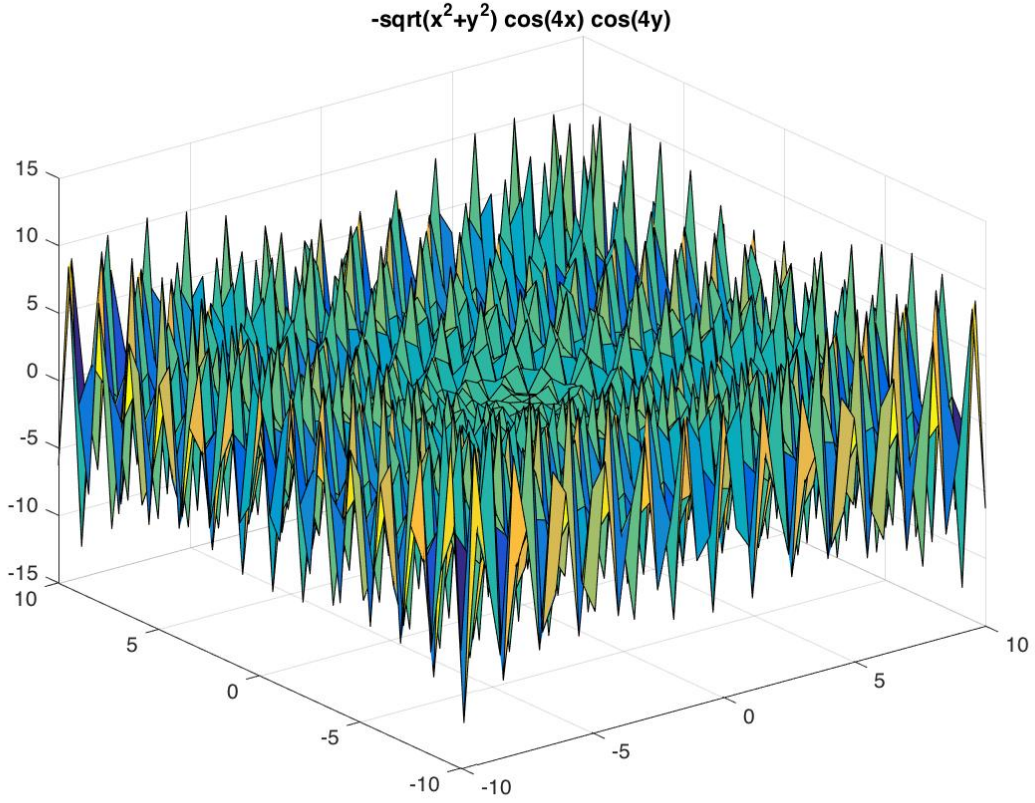
We can see we obtain almost identical results (within roundoff error).

- Sometimes symbolic integration does not give a closed form solution.

  Consider the following MatLab code (in **L15numerical_integration2.m**):

```
syms x y f

xmin=-10.0;
xmax=10.0;
ymin=-10.0;
ymax=10.0;
[X,Y]=meshgrid(xmin:0.5:xmax,ymin:0.5:ymax);

Z=exp(-sqrt(X.^2+Y.^2).*cos(4*X).*cos(4*Y));
surf(X,Y,Z)
title('sqrt(x^2+y^2) cos(4x) cos(4y)');
print non_integratable_fct.jpg -djpeg

%% Symbolic Integration
f=exp(-sqrt(x^2+y^2))*cos(4*x)*cos(4*y)
fprintf('The symbolic expression of the integral is:\n'); % line 16
int(int(f,y,ymin,ymax),x,xmin,xmax)                       % line 17
fprintf('The value of  symbolic integration: \n');        % line 18
eval(int(int(f,y,ymin,ymax),x,xmin,xmax))                 % line 19

% Numerical Integration
fun = @(X,Y)(exp(-sqrt(X.^2+Y.^2)).*cos(4*X).*cos(4*Y));
fprintf('The value of numerical integration: %20.12f\n',...
         integral2(fun,xmin,xmax,ymin,ymax));
```

- The following graph is produced:



A function that is non-integratable by MatLab.

The result of the code produces:

```
f = cos(4*x)*cos(4*y)*exp(-(x^2 + y^2)^(1/2))

The symbolic expression of the integral is:
int(int(cos(4*x)*cos(4*y)*exp(-(x^2+y^2)^(1/2)),y,-10,10),x,-10,10)

The value of  symbolic integration:
int(int(cos(4*x)*cos(4*y)*exp(-(x^2+y^2)^(1/2)),y,-10,10),x,-10,10)

The value of numerical integration:       -1.883399238678
```

- We can see that the symbolic integration does not work. It just returns the MatLab formula for integration as the symbolic integration result. Its numerical evaluation works fine.

- There is no reason to believe the numerical integration result is not correct!!!

- There are functions **integral** and **integral2** and **integral3** to perform 1D, 2D and 3D integration (higher order integration appears not to be directly supported, but you can do these by building up individual calls to verb!int!).

- There are other numerical integration functions as well. **quadgk** numerically evaluates integrals using adaptive Gauss-Kronrod quadrature, **quad2d** numerically evaluates double integrals using the tiled method, **cumtrapz** performs cumulative trapezoidal numerical integration, **trapz** performs trapezoidal numerical integration and **polyint** does polynomial integration.

- We can also specify via name/property pairs the **AbsTol** and **RelTol** values for the integration to functions **integral2**.

- From MathWorks documentation: **integral2** uses the absolute error tolerance to limit an estimate of the **absolute error**, $|qQ|$, where q is the computed value of the integral and Q is the (unknown) exact value. **integral2** might provide more decimal places of precision if you decrease the absolute error tolerance. The default value is 1e-10.

- From MathWorks documentation: **integral2** uses the relative error tolerance to limit an estimate of the **relative error**, $|qQ|/|Q|$, where q is the computed value of the integral and Q is the (unknown) exact value. **integral2** might provide more significant digits of precision if you decrease the relative error tolerance. The default value is 1e-6.

- Details on how to estimate Q are not given.

Symbolic Matrix Calculations

- We can also perform symbolic matrix calculations in MatLab. Consider the following MatLab segments in **L15matrix_example.m**:

```
% declare 8 symbolic variables
syms a1 b1 c1 d1 a2 b2 c2 d2
% set matrices A and B
fprintf('Symbolic matrix A:\n');
A=[a1 b1;
   c1 d1]
fprintf('Symbolic matrix B:\n');
B=[a2 b2;
   c2 d2]
% add A and B symbolically
fprintf('C=A+B\n');
C=A+B
% multiply A and B symbolically
fprintf('D=A*B\n');
D=A*B
```

produces the following output:

```
Symbolic matrix A:
A =
[ a1, b1]
[ c1, d1]
Symbolic matrix B:
B =
[ a2, b2]
[ c2, d2]
C=A+B
C =
[ a1 + a2, b1 + b2]
[ c1 + c2, d1 + d2]
D=A*B
D =
[ a1*a2 + b1*c2, a1*b2 + b1*d2]
[ a2*c1 + c2*d1, b2*c1 + d1*d2]
```

- The MatLab segment shows the numerical evaluation:

```
% set symbolic variables to values
a1=1; b1=2; c1=3; d1=4;
a2=9; b2=8; c2=7; d2=6;
% evaluate matrices A and B
fprintf('Numeric matrix A:\n');
eval(A)
fprintf('Numeric matrix B:\n');
eval(B)
% evaluate the addition of A and B
fprintf('Numeric Addition C=A+B:\n');
eval(C)
% evaluate the multiplication of A and B
fprintf('Numeric Multiplication D=A*B:\n');
eval(D)
```

has the following output:

```
Numeric matrix A:
ans =
     1     2
     3     4
Numeric matrix B:
ans =
     9     8
     7     6
Numeric Addition C=A+B:
ans =
    10    10
    10    10
Numeric Multiplication D=A*B:
ans =
    23    20
    55    48
```

- The following MatLab code computes the symbolic inverse of a matrix and verifies it correctness:

```
% compute the symbolic inverse of A as E
fprintf('Symbolic inverse of A saved as E:\n');
E=inv(A)
% evaluate E using a1 b1 c1 d1
fprintf('Numeric inverse of A saved as E:\n');
eval(E)
% multiply A and E - should get
% the identity matrix
fprintf('Symbolic identity matrix E*A:\n');
I=E*A
% evaluate I
fprintf('Numeric identity matrix E*A:\n');
eval(I)
```

with the following output:

```
Symbolic inverse of A saved as E:
E =
[ d1/(a1*d1-b1*c1),-b1/(a1*d1-b1*c1)]
[-c1/(a1*d1-b1*c1), a1/(a1*d1-b1*c1)]
Numeric inverse of A saved as E:
ans =
   -2.0000    1.0000
    1.5000   -0.5000
Symbolic identity matrix E*A:
I =
[(a1*d1)/(a1*d1-b1*c1)-(b1*c1)/(a1*d1-b1*c1),0]
[0, (a1*d1)/(a1*d1-b1*c1)-(b1*c1)/(a1*d1-b1*c1)]
Numeric identity matrix E*A:
ans =
     1     0
     0     1
```

- Lastly we can symbolically solve a system of equations:

```
% Solve system of equations,  P x = q
syms x1 x2 q1 q2
fprintf('Solve P x=q\n');
P=A
q=[q1; q2]
x=[x1; x2]
% solve for x
fprintf('Symbolic x:\n');
x=P\q
% set q
q1=1; q2=2;
% evaluate x
fprintf('Numeric x:\n');
eval(x)
% residual vector
fprintf('Symbolic residual vector r:\n');
r=P*x-q
fprintf('Numeric residual vector r:\n');
eval(r)
```

with the following output:

```
Solve P x=q
P =
[ a1, b1]
[ c1, d1]
q =
 q1
 q2
x =
 x1
 x2
Symbolic x:
x =
 -(b1*q2-d1*q1)/(a1*d1-b1*c1)
   (a1*q2-c1*q1)/(a1*d1-b1*c1)
Numeric x:
ans =
         0
    0.5000
Symbolic residual vector r:
```

```
r =
(b1*(a1*q2-c1*q1))/(a1*d1-b1*c1)-q1-
  (a1*(b1*q2-d1*q1))/(a1*d1-b1*c1)
(d1*(a1*q2-c1*q1))/(a1*d1-b1*c1)-q2-
  (c1*(b1*q2-d1*q1))/(a1*d1-b1*c1)
Numeric residual vector r:
ans =
     0
     0
```

Symbolic Roots of Polynomials

- We can solve for the roots of a polynomial. We generate a polynomial with known roots and then solve for these roots. Consider the following MatLab code in **L15polynomial_roots_example.m**:

```
% find the roots of a polynomial
syms x
% set up polynomial with roots
% 7, 3, 5.3 and -5.3
fprintf('The polynomial with roots ');
fprintf('7, 3, 5.3 and -5.3\n');
f=(x-7)*(x-3)*(x-5.3)*(x+5.3)
% solve for roots of f
fprintf('Computed roots:\n');
solve(f,x)
% evaluate the roots - convert the
% symbolic expressions into numbers
```

```
fprintf('Evaluated roots:\n');
eval(solve(f,x))
```

- This code produces the result:

```
The polynomial with roots 7, 3, 5.3 and -5.3
f = (x - 3)*(x - 7)*(x - 53/10)*(x + 53/10)
Computed roots:
ans =
      3
      7
  53/10
 -53/10
Evaluated roots:
ans =
    3.0000
    7.0000
    5.3000
   -5.3000
```