7 SYMBOLIC COMPUTATIONS

Table of Contents

Initialization

```
% reduce needless whitespace
format compact
% reduce irritations
more off
% start a diary
%diary lectureN.txt
```

INTRODUCTION

Normally Matlab generates *numbers*, not *formulae*. If you ask Matlab to find the roots where a function is zero, maybe using `fzero`, it gives you numbers. If you ask Matlab to integrate a function from a lower limit to an upper limit using `integral` or `quad`, it gives you a number. Etcetera,

But sometimes you want a formula instead of a number. For example, you might want the derivative or antiderivative of a function. Either one is a formula, not a number. Also, sometimes you want to see an *exact* number, and non-integer numbers in Matlab have round-off errors.

What you need for such purposes is a *symbolic computations* program. Such a program is available inside Matlab as the "Symbolic Math Toolbox". Normally MathWorks charges separately for this package. However, it is included in the "Student Edition", and within Matlab on the COE computers.

In this section we will illustrate how normal Matlab computations and symbolic ones differ. We will look at a simple function, a quadratic in fact.

One thing to remember: Be sure to inform Matlab with the `syms` or `sym` command when variables and/or numbers are intended to be symbolic. Normal variables are names of storage locations with a number in it. But a symbolic variable does not store a number; at all times it can stand for *any* number. So a normal variable `x` is very different from a symbolic variable `x`, and Matlab must know which of the two `x` is.

```
disp(' ')
```

```
% the example quadratic as a normal Matlab function
```

```
qNum = @(x) -4*x.^2+3*x+12

% we can integrate it between, say, 0 and 2
disp('Integrate from 0 to 2 using "integral":')
qNumInt02=integral(qNum,0,2)
disp('We got a floating point number.')

% find a root
disp('Find one of the two roots using "fzero":')
qNumRoot=fzero(qNum,1)
disp('We got a floating point number.')

% tell Matlab that, from now on, x is a symbolic variable
disp(' ')
disp('Let''s do some *symbolic math* now:')
syms x

% the example quadratic as a symbolic function
disp('Create a symbolic quadratic:')
qSym=-4*x^2+3*x+12

% we can integrate it between, say, 0 and 2
disp('Integrate symbolically from 0 to 2 using "int":')
qSymInt02=int(qSym,x,[0 2])
disp('We got an *exact* number.')

% but we can also find the anti-derivative
disp('Find the symbolical anti-derivative using "int":')
qSymInt=int(qSym,x)
disp('We got a symbolic function, the anti-derivative.')
disp('Reformat it with "expand" to check:')
expand(qSymInt)

% and we can find the derivative
disp('Differentiate symbolically using "diff":')
qSymDiff=diff(qSym,x)
disp('We got a symbolic function, the derivative.')

% we can find both roots exactly (note ==, not =)
disp('Find both roots exactly using "solve" (note ==):')
qSymRoots=solve(qSym == 0,x)
disp('We got *both* roots as *symbolic expressions*.')
disp('To see the numbers, use "double":')
double(qSymRoots)

% we can factor the quadratic, but with some trouble
disp('Try to factor the quadratic:')
qSymFactors=factor(qSym)
disp('Oops!  Try "help sym/factor", not "help factor".')
%help sym/factor
disp('Try again to factor the quadratic:')
qSymFactors=factor(qSym,'FactorMode','full')
disp('Use "prod" to combine the factors:')
prod(qSymFactors)
```

```
% we can easily factor a quadratic with rational roots
disp('Try to factor a quadratic with rational roots:')
qSymRat=-4*x^2+3*x+27
qSymRatFactors=factor(qSymRat)
```

```
qNum =
    @(x)-4*x.^2+3*x+12
Integrate from 0 to 2 using "integral":
qNumInt02 =
   19.3333
We got a floating point number.
Find one of the two roots using "fzero":
qNumRoot =
    2.1472
We got a floating point number.

Let's do some *symbolic math* now:
Create a symbolic quadratic:
qSym =
- 4*x^2 + 3*x + 12
Integrate symbolically from 0 to 2 using "int":
qSymInt02 =
58/3
We got an *exact* number.
Find the symbolical anti-derivative using "int":
qSymInt =
(x*(- 8*x^2 + 9*x + 72))/6
We got a symbolic function, the anti-derivative.
Reformat it with "expand" to check:
ans =
- (4*x^3)/3 + (3*x^2)/2 + 12*x
Differentiate symbolically using "diff":
qSymDiff =
3 - 8*x
We got a symbolic function, the derivative.
Find both roots exactly using "solve" (note ==):
qSymRoots =
 3/8 - 201^(1/2)/8
 201^(1/2)/8 + 3/8
We got *both* roots as *symbolic expressions*.
To see the numbers, use "double":
ans =
   -1.3972
    2.1472
Try to factor the quadratic:
qSymFactors =
[ -1, 4*x^2 - 3*x - 12]
Oops!  Try "help sym/factor", not "help factor".
Try again to factor the quadratic:
qSymFactors =
[ -1, x + 201^(1/2)/8 - 3/8, x - 201^(1/2)/8 - 3/8]
Use "prod" to combine the factors:
```

```
ans =
(x + 201^(1/2)/8 - 3/8)*(201^(1/2)/8 - x + 3/8)
Try to factor a quadratic with rational roots:
qSymRat =
- 4*x^2 + 3*x + 27
qSymRatFactors =
[ -1, 4*x + 9, x - 3]
```

SIMPLIFYING ANSWERS

In classes like Analysis in Mechanical Engineering, you are required to simplify your answers. Symbolic math to the rescue!

Watch it, however. If you try to simplify, say, a numeric ratio like 629/969 as

```
          simplify(629/969)
```

then Matlab sees "629/969", evaluates that as 0.6491... and gives that to the symbolic `simplify` function. Of course `simplify` cannot make any sense out of 0.6491.... In fact, it will refuse to cooperate. What you need to do is tell Matlab that the entire "629/969" is to be treated as a symbolic expression, to be given to `simplify` "as is". You can do that with the `sym` function.

```
disp(' ')

% not so easy to see that 17 is a common factor
disp('simplify(629/969) is *wrong*!')
%ratioSimplified=simplify(629/969)
disp('simplify(sym(''629/969'') is right:')
ratioSimplified=simplify(sym('629/969'))
```

```
simplify(629/969) is *wrong*!
simplify(sym('629/969') is right:
ratioSimplified =
37/57
```

VERY HIGH ACCURACY

Function `vpa`, ("variable precision arithmetic"), will give you numbers to arbitrarily high accuracy.

But watch it again. Let's try the result pi^2/6 of the infinite sum we talked about in the previous lesson.

```
disp(' ')

% Matlab gives vpa a number equal to pi^2/6 to 16 digits:
disp('vpa(pi^2/6) is *wrong*:')
piSqOver6=vpa(pi^2/6,50)

% Matlab gives vpa the symbolic string pi^2/6:
disp('vpa(sym(''pi^2/6'')) is correct:')
piSqOver6=vpa(sym('pi^2/6'),50)
```

```
vpa(pi^2/6) is *wrong*:
```

```
piSqOver6 =
1.6449340668482264060656916626612655818462371826172
vpa(sym('pi^2/6')) is correct:
piSqOver6 =
1.6449340668482264364724151666460251892189499012068
```

SOLVING EQUATIONS

The Symbolic Toolbox can solve quite a lot of equations exactly.

If it cannot, it will drop back to a numerical solution.

Solving quadratic equations

Suppose you no longer remembered the solution to the quadratic equation

$$ax^2 + bx + c = 0$$

The symbolic toolbox can give it to you

```
disp(' ')

% tell Matlab that the variables are symbolic
syms a b c x qGen

% define the generic quadratic polynomial
qGen=a*x^2+b*x+c

% find the symbolic solution
disp('Find the roots of this quadratic using "solve"')
qGenRoots=solve(qGen == 0,x)
disp('Reformat using "pretty":')
pretty(qGenRoots)

% see whether we get back our previous solution
disp('Test it for -4x^2 + 3x +12 = 0:')
qGenRootsTest=subs(qGenRoots,{a b c},{-4 3 12})

% we can create a normal Matlab function for the roots
disp('Create a *numerical* function for the roots:')
qGenRootsFun=matlabFunction(qGenRoots)

% test it
disp('Test that for -4x^2 + 3x +12 = 0:')
qGenRootsFunTest=qGenRootsFun(-4,3,12)
```

```
qGen =
a*x^2 + b*x + c
Find the roots of this quadratic using "solve"
qGenRoots =
 -(b + (b^2 - 4*a*c)^(1/2))/(2*a)
 -(b - (b^2 - 4*a*c)^(1/2))/(2*a)
```

```
Reformat using "pretty":
/             2          \
|   b + sqrt(b  - 4 a c) |
| - -------------------- |
|            2 a         |
|                        |
|             2          |
|   b - sqrt(b  - 4 a c) |
| - -------------------- |
\            2 a         /

Test it for -4x^2 + 3x +12 = 0:
qGenRootsTest =
 201^(1/2)/8 + 3/8
 3/8 - 201^(1/2)/8
Create a *numerical* function for the roots:
qGenRootsFun =
    @(a,b,c)[(b.*(-1.0./2.0)-sqrt(a.*c.*-4.0+b.^2).*(1.0./2.0))./a;
(b.*(-1.0./2.0)+sqrt(a.*c.*-4.0+b.^2).*(1.0./2.0))./a]
Test that for -4x^2 + 3x +12 = 0:
qGenRootsFunTest =
    2.1472
   -1.3972
```

SKIP: Cubic and quartic equations

Just like the formulae above for the roots of the quadratic equation, there are formulae for the roots of the cubic,

$$ax^3 + bx^2 + cx + d = 0$$

and quartic,

$$ax^4 + bx^3 + cx^2 + dx + e = 0$$

equations.

But there are *no* formulae for quintic and higher equations. In fact, it has been shown that such formulae *cannot* exist. And even for cubic equations, the formulae are quite messy. We can clean up a bit if we divide the entire equation by *a* and then replace *x* by *X*-_b_/3_a_. That brings the cubic into the simpler form

$$X^3 + CX + D = 0$$

where *C* and *D* can easily be found in terms of *a*, *b*, *c*, and *d*. We will find the general roots of this equation.

```
disp(' ')

% tell Matlab that the variables are symbolic
syms X C D

% define the generic simplified cubic polynomial
disp('Create a symbolic simplified cubic:')
cGen=X^3+C*X+D
```

```
% find the symbolic solution
disp('Try finding the roots using "solve":')
cGenRoots=solve(cGen == 0,X)
disp('Oops.  Try "help solve"')
%help solve
cGenRoots=solve(cGen == 0,X,'MaxDegree',3)
disp('Reformat using "pretty":')
pretty(cGenRoots)

% the roots of x^3 - 3 x - 2 = 0 should be -1, -1, 2
disp('Try it for x^3 - 3x - 2 = 0 (roots -1, -1, 2):')
cGenRootsTest=subs(cGenRoots,{C D},{-3 -2})

% we can create a normal Matlab function for the roots
disp('Create a *numerical* function for the roots:')
cGenRootsFun=matlabFunction(cGenRoots)
% test it
disp('Test that for x^3 - 3x -2 = 0:')
cGenRootsFunTest=cGenRootsFun(-3,-2)

% let's play with a quartic whose roots we know
disp('A symbolic quartic with roots we know:')
quartic=((x+1)^2-2)*((x-2)^2+5)
% expand it to make it harder for Matlab
disp('Hide the roots from the symbolic toolbox:')
quarticExpanded=expand(quartic)
% find the roots
disp('let "solve" find the roots:')
quarticSolved=solve(quarticExpanded == 0,x)
% refactor it to check
disp('Check that "factor" agrees too:')
quarticFactored=factor(quarticExpanded)
```

```
Create a symbolic simplified cubic:
cGen =
X^3 + C*X + D
Try finding the roots using "solve":
cGenRoots =
 root(D + C*z + z^3, z, 1)
 root(D + C*z + z^3, z, 2)
 root(D + C*z + z^3, z, 3)
Oops.  Try "help solve"
cGenRoots =

                                        ((C^3/27 + D^2/4)^(1/2) -
 D/2)^(1/3) - C/(3*(- D/2 + (C^3/27 + D^2/4)^(1/2))^(1/3))
 C/(6*(- D/2 + (C^3/27 + D^2/4)^(1/2))^(1/3)) - ((C^3/27 +
 D^2/4)^(1/2) - D/2)^(1/3)/2 - (3^(1/2)*(((C^3/27 + D^2/4)^(1/2) -
 D/2)^(1/3) + C/(3*(- D/2 + (C^3/27 + D^2/4)^(1/2))^(1/3)))*1i)/2
 (3^(1/2)*(((C^3/27 + D^2/4)^(1/2) - D/2)^(1/3) + C/(3*(- D/2 +
 (C^3/27 + D^2/4)^(1/2))^(1/3)))*1i)/2 - ((C^3/27 + D^2/4)^(1/2) -
 D/2)^(1/3)/2 + C/(6*(- D/2 + (C^3/27 + D^2/4)^(1/2))^(1/3))
```

```
Reformat using "pretty":
/           C    \
|    #2 - ----   |
|         3 #3   |
|                |
|   C    #2      |
| ---- - -- - #1 |
| 6 #3    2      |
|                |
|      #2     C  |
| #1 - -- + ---- |
\       2   6 #3 /

where

                 /        C  \
         sqrt(3) | #2 + ---- | 1i
                 \      3 #3 /
   #1 == ------------------------
                     2

         /     /  3    2 \     \1/3
         |     | C    D  |    D |
   #2 == | sqrt| -- + -- | - - |
         \     \ 27    4 /   2 /

         /           /  3    2 \ \1/3
         |   D       | C    D  | |
   #3 == | - - + sqrt| -- + -- | |
         \   2       \ 27    4 / /


Try it for x^3 - 3x - 2 = 0 (roots -1, -1, 2):
cGenRootsTest =
  2
 -1
 -1
Create a *numerical* function for the roots:
cGenRootsFun =
    @(C,D)
[(D.*(-1.0./2.0)+sqrt(C.^3.*(1.0./2.7e1)+D.^2.*(1.0./4.0))).^(1.0./3.0)-
C.*1.0./
(D.*(-1.0./2.0)+sqrt(C.^3.*(1.0./2.7e1)+D.^2.*(1.0./4.0))).^(1.0./3.0).*(1.0./3.0)
(D.*(-1.0./2.0)+sqrt(C.^3.*(1.0./2.7e1)+D.^2.*(1.0./4.0))).^(1.0./3.0).*(1.0./3.0)
(D.*(-1.0./2.0)+sqrt(C.^3.*(1.0./2.7e1)+D.^2.*(1.0./4.0))).^(1.0./3.0).*(1.0./2.0)
(D.*(-1.0./2.0)+sqrt(C.^3.*(1.0./2.7e1)+D.^2.*(1.0./4.0))).^(1.0./3.0).*(1.0./6.0)
(D.*(-1.0./2.0)+sqrt(C.^3.*(1.0./2.7e1)+D.^2.*(1.0./4.0))).^(1.0./3.0).*(1.0./3.0)
(D.*(-1.0./2.0)+sqrt(C.^3.*(1.0./2.7e1)+D.^2.*(1.0./4.0))).^(1.0./3.0).*(1.0./2.0)
(D.*(-1.0./2.0)+sqrt(C.^3.*(1.0./2.7e1)+D.^2.*(1.0./4.0))).^(1.0./3.0).*(1.0./6.0)
Test that for x^3 - 3x -2 = 0:
cGenRootsFunTest =
     2
    -1
    -1
```

```
A symbolic quartic with roots we know:
quartic =
((x + 1)^2 - 2)*((x - 2)^2 + 5)
Hide the roots from the symbolic toolbox:
quarticExpanded =
x^4 - 2*x^3 + 22*x - 9
let "solve" find the roots:
quarticSolved =
  - 2^(1/2) - 1
 2 - 5^(1/2)*1i
 2 + 5^(1/2)*1i
    2^(1/2) - 1
Check that "factor" agrees too:
quarticFactored =
[ x^2 - 4*x + 9, x^2 + 2*x - 1]
```

Solving other equations

Some other equations the Toolbox manages to solve.

```
disp(' ')

% make sure x and y are symbols
syms x y

% exponentials can be negative for complex arguments
disp('Solve e^x = -1:')
expSol=solve(exp(x) == -1,x)

% the next expression to solve, but we will write it out
disp(' ')
disp('Let''s try to solve a x y - b x - 1 = 0 now:')
qxySym=a*x*y - b*x - 1
% a x y - b x - 1 = 0 can be solved for either x or y
disp('Solve for x:')
xSol=solve(a*x*y - b*x - 1 == 0, x)
disp('Solve for y instead:')
ySol=solve(a*x*y - b*x - 1 == 0, y)
% sometimes rewriting the equation can help
disp('Use "collect" to identify the x-coefficients::')
xCoefs=collect(a*x*y-b*x-1,x)

% solve can solve systems of equations
disp(' ')
disp('Solve two linear equations in two unknowns:')
[xSol,ySol] = solve(x+y == 7, x-y == 1,x,y)

% if solve fails to do so, Matlab switches to numerical
disp(' ')
disp('Try to solve two non-linear linear equations:')
[xSol,ySol] = solve(x^2+cos(y) == 7, cosh(x)-y == 1,x,y)
```

```
Solve e^x = -1:
```

```
expSol =
pi*1i

Let's try to solve a x y - b x - 1 = 0 now:
qxySym =
a*x*y - b*x - 1
Solve for x:
xSol =
-1/(b - a*y)
Solve for y instead:
ySol =
(b*x + 1)/(a*x)
Use "collect" to identify the x-coefficients::
xCoefs =
(a*y - b)*x - 1

Solve two linear equations in two unknowns:
xSol =
4
ySol =
3

Try to solve two non-linear linear equations:
Warning: Cannot solve symbolically. Returning a numeric approximation
 instead.
xSol =
-2.5330484287641517693678062683806
ySol =
5.3356247748635880949060107078797
```

PARTIAL FRACTIONS

In analyzing the dynamics of controlled systems, you often encounter ratios of big polynomials. Then you want to take these ratios apart in simpler fractions. The reason is that these simpler fractions tell you many important things. For example, they tell you whether, if the system is disturbed, it will return to its normal position, versus, say, crash. And, if so, they will also tell you how fast the system will return to normal.

Symbolic function `partfrac` can do it.

```
disp(' ')

% make sure x is still symbolic
syms x

% example symbolic ratio
disp('Create an example symbolic ratio:')
ratSym=(2*x^2-3*x+1)/(x^3+2*x^2-9*x-18)

% you could first look at the factors
disp('Let''s look at the factors first:')
ratSymFactors=factor(ratSym)
disp('Multiply them out:')
prod(ratSymFactors)
```

```
% get the partial fractions
disp('Now get the partial fractions')
ratPartFrac=partfrac(ratSym)
```

```
Create an example symbolic ratio:
ratSym =
-(2*x^2 - 3*x + 1)/(- x^3 - 2*x^2 + 9*x + 18)
Let's look at the factors first:
ratSymFactors =
[ 2*x - 1, x - 1, 1/(x - 3), 1/(x + 3), 1/(x + 2)]
Multiply them out:
ans =
((2*x - 1)*(x - 1))/((x + 2)*(x - 3)*(x + 3))
Now get the partial fractions
ratPartFrac =
1/(3*(x - 3)) - 3/(x + 2) + 14/(3*(x + 3))
```

FUNCTION MANIPULATIONS

Below are some more example manipulations

```
disp(' ')

% make sure x, etc, are still symbolic
syms x a b c

% dealing with integrals you cannot do
disp(' ')
disp('Try two hard integrals from a basic table book:')
disp('Try 1/(x*(a*x^2+b*x+c)^(1/2)):')
intHard1=int(1/(x*(a*x^2+b*x+c)^(1/2)))
disp('Try 1/(x*(a*x^2+b*x+c)^(3/2)):')
intHard2=int(1/(x*(a*x^2+b*x+c)^(3/2)))
disp('But the solution is in a basic table book!')
intHard2=int(1/(x*(a*x^2+b*x+c)^(3/2)), ...
    'IgnoreSpecialCases',1,'IgnoreAnalyticConstraints',1)
disp('Do not pay too much for the Toolbox!')

% sometimes int is useful to identify a function
disp(' ')
disp('The integral of exp(-x^2):')
intExpMinusxSqr=int(exp(-x^2),x)
disp('The integral of sin(x)/x:')
intSinxOverx=int(sin(x)/x,x)

% writing the Taylor series of, say, sinint
disp(' ')
disp('Try to learn about the Taylor series of Si:')
sinxOverxTaylor=taylor(sin(x)/x)
sinxOverxTaylor=taylor(sin(x)/x,'Order',10)
sinintTaylor=int(sinxOverxTaylor)

% examine the logic of the terms
```

```
disp('Examine t(3)/t(1):')
t3ot1=simplify(-(x^3/18)/x)
factors18=factor(18)
disp('Examine t(5)/t(3):')
t5ot3=simplify(-(x^5/600)/(x^3/18))
factors100=factor(100)
disp('Examine t(7)/t(5):')
t7ot5=simplify(-(x^7/35280)/(x^5/600))
factors294=factor(294)
disp('Examine t(9)/t(7):')
t9ot7=simplify(-(x^9/3265920)/(x^7/35280))
factors648=factor(648)

% Matlab has "funtool" to play with functions:
%funtool
```

```
Try two hard integrals from a basic table book:
Try 1/(x*(a*x^2+b*x+c)^(1/2)):
intHard1 =
-log(b/2 + c/x + (c^(1/2)*(a*x^2 + b*x + c)^(1/2))/x)/c^(1/2)
Try 1/(x*(a*x^2+b*x+c)^(3/2)):
intHard2 =
int(1/(x*(a*x^2 + b*x + c)^(3/2)), x)
But the solution is in a basic table book!
intHard2 =
int(1/(x*(a*x^2 + b*x + c)^(3/2)), x, 'IgnoreSpecialCases', true,
 'IgnoreAnalyticConstraints', true)
Do not pay too much for the Toolbox!

The integral of exp(-x^2):
intExpMinusxSqr =
(pi^(1/2)*erf(x))/2
The integral of sin(x)/x:
intSinxOverx =
sinint(x)

Try to learn about the Taylor series of Si:
sinxOverxTaylor =
x^4/120 - x^2/6 + 1
sinxOverxTaylor =
x^8/362880 - x^6/5040 + x^4/120 - x^2/6 + 1
sinintTaylor =
x^9/3265920 - x^7/35280 + x^5/600 - x^3/18 + x
Examine t(3)/t(1):
t3ot1 =
-x^2/18
factors18 =
     2     3     3
Examine t(5)/t(3):
t5ot3 =
-(3*x^2)/100
factors100 =
     2     2     5     5
Examine t(7)/t(5):
t7ot5 =
-(5*x^2)/294
factors294 =
     2     3     7     7
Examine t(9)/t(7):
t9ot7 =
-(7*x^2)/648
factors648 =
     2     2     2     3     3     3     3
```

End lesson 7

Published with MATLAB® R2015b