MATLAB Session 2 Web Support Supplement

Partial Fraction and Transfer Functions	
Symbolic algebra functions:	
`poly2sym`	Convert a polynomial to its algebraic representation
`sym`	Create a symbolic object
`sym2poly`	Convert an algebraic expression to a polynomial
`laplace`	Symbolic Laplace transform
`ilaplace`	Symbolic inverse Laplace transform
`pretty`	Windowdress a symbolic expression

M2.3 Symbolic algebra tools

Features covered in this session:

- Use of `sym` to create symbolic variables
- Perform partial fractions, Laplace transform and its inverse with symbolic algebra

Since version 4, MATLAB has incorporated the symbolic algebra engine from Maple. We do not need this capability to understand or learn control. Thus, we will not present a formal coverage of the symbolic algebra toolbox. However, it is important to be informed of this technique and it is sort of fun to carry out at least some simple tasks with symbolic algebra. The following commands are meant to serve as an illustration of this technique. For the list of functions and MATLAB demonstrations, enter:

```
help symbolic
```

MATLAB understands, in most cases, our desire to define an algebraic expression when we enclose it in quotes:

```
p = 'a*x^2+b*x+c'
```

This is especially true when we use the expression as a function argument. For example, we can differentiate the polynomial with

```
diff(p)
```

and MATLAB should return `2*a*x+b` as the answer. To ensure that there is no misunderstanding, we can always define a symbolic expression explicitly with the `sym` function:

```
p = sym('a*x^2+b*x+c')
```

We can easily convert a polynomial to its algebraic equivalent with `poly2sym()`. This is nice since we work mostly with polynomials and they are a lot easier to type. This is a simple example:

```
p=[1 2 0 3];
sp=poly2sym(p)      % sp is the symbolic representation of p
pretty(sp)          % pretty() is just what it says

sym2poly(sp)        % converts from symbolic back to polynomial
```

There are, of course, many other functions available to manipulate algebraic expressions, but we will skip them and jump right to doing Laplace transform. In the following three examples, we first do the Laplace transform of f(t) with `laplace()`, and afterward, use the inverse to recover f(t) with `ilaplace()`. The first example uses a formal definition of an expression, which helps with clarity as shown in the third example. The second example shows how we can use an expression as an argument.

```
f=sym('exp(-a*t)')                    % f(t) = e(-at)
F=laplace(f)
fcheck=ilaplace(F)

laplace(sym('sin(freq*t)'))           % f(t) = sin(ωt)
ilaplace(ans)

f=sym('exp(-a*t)*sin(freq*t)')        % f(t) = e(-at)sin(ωt)
laplace(f)
ilaplace(ans)
```

Take note how we have used `freq`, but not `w`, to represent frequency. MATLAB uses a quirky rule (the highest alphabet, among others) to determine which notation is the independent variable. We can choose our own with functions such as differentiation, but not with Laplace transform. Thus using `w` as the frequency only works with a simple sine function, as in

```
syms w x s;
laplace(sin(w*x),s)
```

but not when we have a product of exponential and sinusoidal functions. It is safer to use `freq` to represent frequency.

We will now switch topics and use the symbolic algebra toolbox for doing partial fractions, which of course, is redundant since we can do the inverse transform with `ilaplace()`. As an example, we repeat Example 2.1 in text. First, we generate the algebraic representation of the two polynomials and use them to make the symbolic transfer function:

```
q=[6 0 -12];
sq=poly2sym(q);
sp=poly2sym([1 1 -4 -4]);     % skipping one step for p(s)

g=sq/sp                % makes the transfer function q/p
pretty(g)
```

With the transfer function in place, we find the partial fraction expansion and the time-domain function with:

```
diff(int(g))
subs(ans,'s','x')      % substitutes "s" for "x"
pretty(ans)
ilaplace(ans)          % gets the time-domain function
```

The idea behind the partial fraction expansion is to use the fact that when we perform a symbolic integration, the function `int()` will usually separate terms and differentiating the answer will re-produce the original expression in separate terms. The second and third commands are for window-dressing and are optional.

To repeat our comment on redundancy, we could simply have executed:

```
ilaplace(g)
```

To double check, we take the Laplace transform of the result to recover the transfer function:

```
laplace(ans)            % confirms the same Laplace transform
pretty(ans)
```

We finish with an example that we worked on earlier.

```
poles=[0 -1 -2 -3];
p=poly(poles);
sp=poly2sym(p)
factor(sp)              % same as s(s+1)(s+2)(s+3)
tf=symdiv('1',sp)       % makes the transfer function
ilaplace(tf)            % finds the time domain function
```

As always, our "hand calculation" adds one extra step. We first need to do the partial fraction expansion before we can easily find the inverse transform:

```
diff(int(tf))
ilaplace(ans)
```

MATLAB can do much more. We can use symbolic algebra, for example, to find the analytical solution of an ordinary differential equation. However, that would divert our attention from understanding how we can infer time domain responses from only pole positions without doing much work. We will stop our illustration of symbolic algebra here.