Appendix F

MATLAB's Symbolic Math Toolbox Tutorial

F.1 Introduction

Readers who are studying MATLAB may want to explore the additional functionality of MATLAB's Symbolic Math Toolbox. Before proceeding, the reader should have studied Appendix B, the MATLAB tutorial, including Section B.1, which is applicable to this appendix.

MATLAB's Symbolic Math Toolbox Version 8.0 in addition to MATLAB Version 9.3 (R2017b) and the Control System Toolbox Version 10.3 is required in order to add symbolic mathematics capability to your M-files.

The M-files in this appendix are available elsewhere on this Web site.

Symbolic math commands are used in your MATLAB M-files right along with your standard MATLAB statements. The only additional requirement is to declare symbolic variables before they are used with the statement `syms x1 x2...`, where `xi` are symbolic variables.

Some of the added capabilities that the Symbolic Math Toolbox yields for control systems analysis and design include the following:

1. Functions and equations can be entered symbolically. That is, alpha characters as well as numerical characters can be used in your M-files. For example, you can enter `B=x^2+3*x+7`, instead of `B=[1 3 7]`. You could even enter `B=a*x^2+b*x+c` and obtain its factors as

```
[              2            1/2]
[         -b + (b - 4 a c)      ]
[1/2 --------------------------]
[               a              ]
[                              ]
[              2            1/2]
[         -b - (b - 4 a c)      ]
[1/2 --------------------------]
[               a              ]
```

2. Symbolic expressions can be manipulated algebraically and simplified.
3. Transfer functions can be typed almost as written, making your M-files more readable. For example, the statement, `G=(s+1)*(s+2)/[(s^2+3*s+10)*(s+4)]` would replace the three statements, `numg=poly([-1 -2])`, `deng=conv([1 3 10], [1 4])`, and `G=tf(numg,deng)`.

4. Laplace and z-transforms as well as their inverses can be entered and found in symbolic form.
5. Functions can be "pretty printed" for clarity in the **MATLAB Command Window** and printed output.

These are only a few advantages of using the Symbolic Math Toolbox. This appendix will explore more. The reader is encouraged not to stop exploration at the end of Appendix F, since there is so much more than can be covered here. The Bibliography at the end of this appendix gives references for further pursuit.

The M-files for Appendix F can be found in the Control Systems Engineering Toolbox. Symbolic Math Toolbox examples are included for Chapters 2, 3, 4, 6, and 13. The reader is encouraged, however, to apply what is learned to other chapters.

F.2 Symbolic Math Toolbox Examples

Chapter 2: Modeling in the Frequency Domain

ch2apF1 MATLAB's calculating power is greatly enhanced using the Symbolic Math Toolbox. In this example, we demonstrate its power by calculating inverse Laplace transforms of $F(s)$. The beginning of any symbolic calculation requires defining the symbolic objects. For example, the Laplace transform variable, s, or the time variable, t, must be defined as a symbolic object. This definition is performed using the `syms` command. Thus, `syms s` defines `s` as a symbolic object; `syms t` defines `t` as a symbolic object; and `syms s t` defines both `s` and `t` as symbolic objects. We need only define objects that we input to the program. Variables produced by the program need not be defined. Thus, if we are finding inverse Laplace transforms, we need only define s as a symbolic object, since t results from the calculation. Once the object is defined, we can then type F as a function of s as we normally would write it. We do not have to use vectors to represent the numerator and denominator. The Laplace transforms or time functions can also be printed in the **MATLAB Command Window** as we normally would write it. This form is called *pretty printing*. The command is `pretty(F)`, where `F` is the function we want to pretty print. In the code below, you can see the difference between normal printing and pretty printing if you run the code without the semicolons at the steps where the functions, `F` or `f`, are defined. Once `F` is defined as $F(s)$, we can find the inverse Laplace transform using the command `ilaplace(F)`. In the following example, we find the inverse Laplace transforms of the frequency functions in the examples used for Cases 2 and 3 in Section 2.2 in the text.

```
'(ch2apF1)'                          % Display label.
syms s                               % Construct symbolic object for
                                     % Laplace variable 's'.
'Inverse Laplace transform'          % Display label.
F=2/[(s+1)*(s+2)^2];                 % Define F(s) form case 2 example.
'F(s) from case 2'                   % Display label.
pretty (F)                           % Pretty print F(s)
f=ilaplace(F);                       % Find inverse Laplace transform.
'f(t) for case 2'                    % Display label.
pretty(f)                            % Pretty print f(t) for Case 2.
F=3/[s*(s^2+2*s+5)];                 % Define F(s) from Case 3 example.
'F(s) for Case 3'                    % Display label.
pretty(F)                            % Pretty print F(s) for Case 3.
```

```
f=ilaplace(F);                    % Find inverse Laplace transform.
'f(t) for Case 3'                 % Display label.
pretty(f)                         % Pretty print f(t) for Case 3.
Pause
```

ch2apF2 In this example, we find Laplace transforms of time functions using the command, `laplace(f)`, where `f` is a time function, $f(t)$. As an example, we use the time functions that resulted from the calculations in Cases 2 and 3 in Section 2.2 in the text and work in reverse to obtain their Laplace transforms. We will see that the command, `laplace(f)`, yields $F(s)$ in partial fractions. In addition to pretty printing discussed in the previous example, the Symbolic Math Toolbox contains other commands that can change the look of the displayed result for readability and form. Some of these commands are: `collect(F)`—collect common coefficient terms of `F`; `expand(F)`—expands product of factors of `F`; `factor(F)`—factors `F`; `simple(F)`—finds simplest form of `F` with the least number of terms; `simplify(F)`—simplifies `F`; `vpa(expression, places)`—standing for *variable precision arithmetic*, this command converts fractional symbolic terms into decimal terms with a specified number of decimal places. For example, the symbolic fraction, 3/16, would be converted to 0.1875 if the argument, `places`, were 4. In the example below, we find the Laplace transform of a time function. The result is displayed as partial fractions. To combine the partial fractions, we use the command, `simplify(F)`, where `F` is the Laplace transform of $f(t)$ found using `laplace(f)`. Finally, we use `F=vpa(F,3)` to convert the symbolic fractions to decimals in the displayed result.

```
'(ch2apF2)'                       % Display label.
syms t                            % Construct symbolic object for
                                  % time variable 't'.
'Laplace transform'               % Display label.
'f(t) from Case 2'                % Display label.
f=2*exp(-t)-2*t*exp(-2*t)-2*exp(-2*t);
                                  % Define f(t) from Case 2 example.
pretty(f)                         % Pretty print f(t) from Case 2
                                  % example.
'F(s) for Case 2'                 % Display label.
F=laplace(f);                     % Find Laplace transform.
pretty(F)                         % Pretty print partial fractions of
                                  % F(s) for Case 2.
F=simplify(F);                    % Combine partial fractions.
pretty(F)                         % Pretty print combined partial
                                  % fractions.
'f(t) for Case 3'                 % Display label.
f=3/5-3/5*exp(-t)* [cos(2 * t)+(1/2)*sin(2 * t)];
                                  % Define f(t) from Case 3 example.
pretty (f)                        % Pretty print f(t) for Case 3.
'F(s) for Case 3 - Symbolic fractions'
                                  % Display label.
F=laplace(f);                     % Find Laplace transform.
pretty(F)                         % Pretty print partial fraction of
                                  % F(s) for Case 3.
'F(s) for Case 3 - Decimal representation'
                                  % Display label.
F=vpa(F,3);                       % Convert symbolic numerical
                                  % fractions to 3-place decimal
                                  % representation for F(s).
pretty(F)                         % Pretty print decimal
                                  % representation.
```

```
'F(s) for Case 3 - Simplified        % Display label.
F=simplify(F);                       % Combine partial fractions.
pretty(F)                            % Pretty print combined partial
                                     % fractions.
pause
```

ch2apF3 MATLAB's Symbolic Math Toolbox may be used to simplify the input of complicated transfer functions as follows: Initially, input the transfer function $G(s) =$ `numg/deng` via symbolic math statements. Then convert $G(s)$ to an LTI transfer function object. This conversion is done in two steps. The first step uses the command `[numg, deng]=numden(G)` to extract the symbolic numerator and denominator of `G`. The second step converts, separately, the numerator and denominator to vectors using the command `sym2poly(S)`, where `S` is a symbolic polynomial. The last step consists of forming the LTI transfer function object by using the vector representation of the transfer function's numerator and denominator. As an example, we form the LTI object $G(s) = [54(s + 27)(s^3+52s^2+ 37s +73)]/[s(s^4+872s^3+437s^2+ 89s+ 65)\ (s^2 + 79s +36)]$, making use of MATLAB's Symbolic Math Toolbox for simplicity and readability.

```
'(ch2apF3)'                          % Display label.
syms s                               % Construct symbolic object for
                                     % frequency variable 's'.
G=54*(s+27)*(s^3+52*s^2+37*s+73)...
  /(s*(s^4+872*s^3+437*s^2+89*s+65)*(s^2+79*s+36));
                                     % Form symbolic G(s).
'Symbolic G(s)'                      % Display label.
pretty(G)                            % Pretty print symbolic G(s).
[numg,deng]=numden(G);               % Extract symbolic numerator and
                                     % denominator.
numg=sym2poly(numg);                 % Form vector for numerator of
                                     % G(s).
deng=sym2poly(deng);                 % Form vector for denominator of
                                     % G(s).
'LTI G(s) in Polynomial Form'        % Display label.
Gtf=tf(numg,deng)                    % Form and display LTI object for
                                     % G(s) in polynomial form.
'LTI G(s) in Factored Form'          % Display label.
Gzpk=zpk(Gtf)                        % Convert G(s) to factored form.
pause
```

ch2apF4 (Example 2.10) MATLAB's Symbolic Math Toolbox may be used to simplify the solution of simultaneous equations by using Cramer's rule. A system of simultaneous equations can be represented in matrix form by $\mathbf{Ax} = \mathbf{B}$, where $\mathbf{A}$ is the matrix formed from the coefficients of the unknowns in the simultaneous equations, $\mathbf{x}$ is a vector containing the unknowns, and $\mathbf{B}$ is a vector containing the inputs. Cramer's rule states that x_k the kth element of the solution vector, $\mathbf{x}$, is found using $x_k = \det(\mathbf{A_k})/\det(\mathbf{A})$, where $\mathbf{A}_k$ is the matrix formed by replacing the kth column of matrix $\mathbf{A}$ with the input vector, $\mathbf{B}$. In the text, we refer to det($\mathbf{A}$) as "delta." In MATLAB, matrices are written with a space or comma separating the elements of each row. The next row is indicated with a semicolon or carriage return. The entire matrix is then enclosed in a pair of square brackets. Applying the above to the solution of Example 2.10: `A=[(R1+L*s)-L*s;-L*s (L*s+R2+(1/(c*s)))]` and `Ak=[(R1+L*s) V;-L*s 0]`. The function `det(matrix)` evaluates the determinant of the square matrix argument. Let us now find the transfer function $G(s) = I_2(s)/V(s)$, asked for in Example 2.10. The command `simplify(S)`, where `S` is a symbolic function, is introduced in the solution. `Simplify(S)` simplifies the solution by shortening the

length of S. The use of `simplify(I2)` shortens the solution by combining like powers of the Laplace variable, *s*.

```
'(ch2apF4) Example 2.10'            % Display label.
syms s R1 R2 L c V                  % Construct symbolic objects for
                                    % frequency variable 's', and
                                    % 'R1','R2', 'L','c', and 'V'.
                                    % Note: Use lower-case 'c'
                                    % in declaration for
                                    % capacitor.
A2=[(R1+L*s)V;-L*s 0]               % Form Ak = A2.
A=[(R1+L*s)-L*s;-L*s (L*s+R2+(1/(c*s)))]
                                    % Form A.
I2=det(A2)/det(A);                  % Use Cramer's rule to solve for
                                    % I2(s).
I2=simplify(I2);                    % Reduce complexity of I2(s)
G=I2/V;                             % Form transfer function,
                                    % G(s) = I2(s)/V(s).
'G(s)'                              % Display label.
pretty (G)                          % Pretty print G(s).
pause
```

Chapter 3: Modeling in the Time Domain

ch3apF1 (Example 3.6) MATLAB's Symbolic Math Toolbox may be used to perform matrix operations. The code for these operations is intuitive and readable. The operations are addition `(+)`, subraction `(-)`, inverse `(^-1)`, and matrix raised to a power `n (^n)`. We demonstrate by solving Example 3.6 in the text using Eq. 3.73 directly.

```
'(ch3apF1) Example 3.6'             % Display label.
syms s                              % Construct symbolic object for
                                    % frequency variable 's'.
A=[0 1 0;0 0 1;-1 -2 -3];           % Create matrix A.
B=[10;0;0];                         % Create vector B.
C=[1 0 0];                          % Create vector C.
D=0;                                % Create D.
I=[1 0 0;0 1 0;0 0 1];              % Create identity matrix.
'T(s)'                              % Display label.
T=C*((s*I-A)^-1)* B+D;              % Find transfer function.
pretty(T)                           % Pretty print transfer function.
pause
```

Chapter 4: Time Response

Ch4apF1 (Example 4.11) MATLAB's Symbolic Math Toolbox, with its ability to perform matrix operations, lends itself to the Laplace transform solution of state equations. Also, the command `[V,D]=eig(A)` allows us to find the eigenvalues of a square matrix, A, which are the diagonal elements of diagonal matrix D. We demonstrate by solving Example 4.11.

```
'(ch4apF1) Example 4.11'            % Display label.
syms s                              % Construct symbolic object for
                                    % frequency variable 's'.
'a'                                 % Display label.
A=[0 1 0;0 0 1;-24 -26 -9];         % Create matrix A.
B=[0;0;1];                          % Create vector B.
X0=[1;0;2];                         % Create initial condition vector,
                                    % X(0).
```

```
U=1/(s+1);                          % Create U(s).
I=[1 0 0;0 1 0;0 0 1];              % Create identity matrix.
X=((s*I-A)^-1)*(X0+B*U);            % Find Laplace transform of state
                                    % vector.
x1=ilaplace(X(1));                  % Solve for X1(t).
x2=ilaplace(X(2));                  % Solve for X2(t).
x3=ilaplace(X(3));                  % Solve for X3(t).
y=x1+x2;                            % Solve for output, y(t).
y=vpa(y,3);                         % Convert fractions to decimals.
'y(t)'                              % Display label.
pretty(y)                           % Pretty print y(t).
'b'                                 % Display label.
[V,D]=eig(A);                       % Find eigenvalues, which are the
                                    % diagonal elements of D.
'Eigenvalues on diagonal'           % Display label.
D                                   % Display D.
pause
```

ch4apF2 (Example 4.12/4.13) In this example, we use MATLAB's Symbolic Math Toolbox to solve state equations in the time domain. We make use of the Symbolic Math Toolbox's ability to perform integration. We first solve for the state-transition matrix by taking the inverse Laplace transform of $(s\mathbf{I} - \mathbf{A})^{-1}$. We then use the convolution integral to obtain the solution. Integration is performed using the command `int(S,v,a,b)`, where `S` is the function to be integrated, `v` is the variable of integration, `a` is the lower limit of integration, and `b` is the upper limit of integration. As an example we solve Example 4.12 in the text. The state-transition matrix is obtained by the method demonstrated in Example 4.13 in the text.

```
'(ch4apF2) Example 4.12/4.13'       % Display label.
syms s t tau                        % Construct symbolic object for
                                    % frequency variable 's','t',
                                    % and 'tau'.
'a'                                 % Display label.
A=[0 1;-8 -6]                       % Create matrix A.
B=[0;1]                             % Create vector B.
X0=[1;0]                            % Create initial condition vector,
                                    % X(0).
U=1                                 % Create u(t).
I=[1 0;0 1];                        % Create identity matrix.
'E=(s*I-A)^-1'                      % Display label.
E=((s*I-A)^-1)                      % Find Laplace transform of state-
                                    % transition matrix,(sI-A)^-1.
Fi11=ilaplace(E(1,1));              % Take inverse Laplace transform
Fi12=ilaplace(E(1,2));              % of each element
Fi21=ilaplace(E(2,1));              % of (sI-A)^-1
Fi22=ilaplace(E(2,2));              % to find state-transition matrix.
'Fi(t)'                             % Display label.
Fi=[Fi11 Fi12;Fi21 Fi22];           % Form state-transition matrix,
                                    % Fi(t).
pretty(Fi)                          % Pretty print state-transition
                                    % matrix, Fi(t).
Fitmtau=subs(Fi,t,t-tau);           % Form Fi(t-tau).
'Fi(t-tau)'                         % Display label.
pretty(Fitmtau)                     % Pretty print Fi(t-tau).
X=Fi*X0+int(Fitmtau*B*1,tau,0,t);   % Solve for X(t).
X=expand(X);                        % Expand X for clearer display.
```

```
'X(t)'                                  % Display label.
pretty(X)                               % Pretty print X (t).
pause
```

Chapter 6: Stability

ch6apF1 (Example 6.2) MATLAB's Symbolic Math Toolbox may be used conveniently to calculate the values in a Routh table. The toolbox is particularly useful for more complicated tables, where symbolic objects, such as epsilon, are used. In this example, we represent each row of the Routh table by a vector. Expressions are written for subsequent row elements by using the equations given in Table 6.2 of the text. The MATLAB command `det(M)` is used to find the determinant of the square matrix, `M`, as shown for each row element in Table 6.2. Further, we test the previous row's first element to see if it is zero. If it is zero, it is replaced by epsilon, `e`, in the next row's calculation. The preceding logic is performed using MATLAB's IF/ELSE/END as shown in the code below.

We now demonstrate the making of a Routh table using the Symbolic Math Toolbox for a problem that requires the epsilon method to complete the table. The following program produces the Routh table for Example 6.2 in the text. Also, for clarity, we convert all rows to symbolic objects, simplify, and pretty print after forming the table. CAUTION: In general, the results of this program are not valid if an entire row is zero as `e` approaches zero, such as `[e 0 0 0]`. This case must be handled differently, as discussed in text Section 6.3 in the subsection, "Entire Row Is Zero."

```
'(ch6apF1) Example 6.2'                 % Display label.
% -det([si() si(); sj() sj()])/sj()
                                        % Template for use in each cell.
syms e                                  % Construct a symbolic object for
                                        % epsilon.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
s5=[1 3 5 0 0];                         % Create s^5 row of Routh table.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
s4=[2 6 3 0 0];                         % Create s^4 row of Routh table.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if -det([s5(1) s5(2);s4(1) s4(2)])/s4(1)==0
  s3=[e...
    -det([s5(1) s5(3);s4(1) s4(3)])/s4(1) 0 0];
                                        % Create s^3 row of Routh table
                                        % if 1st element is 0.
else
  s3=[-det([s5(1) s5(2);s4(1) s4(2)])/s4(1)...
    -det([s5(1) s5(3);s4(1) s4(3)])/s4(1) 0 0];
                                        % Create s^3 row of routh table
                                        % if 1st element is not zero.
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if -det([s4(1) s4(2);s3(1) s3(2)])/s3(1)==0
  s2=[e...
    -det([s4(1) s4(3);s3(1) s3(3)])/s3(1) 0 0];
                                        % Create s^2 row of Routh table
                                        % If 1st element is 0.
else
  s2=[-det([s4(1) s4(2);s3(1) s3(2)])/s3(1)...
    -det([s4(1) s4(3);s3(1) s3(3)])/s3(1) 0 0];
                                        % Create s^2 row of Routh table
                                        % if 1st element is not zero.
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if -det([s3(1) s3(2);s2(1) s2(2)])/s2(1)==0
  s1=[e...
    -det([s3(1) s3(3);s2(1) s2(3)])/s2(1) 0 0];
                                    % Create s^1 row of Routh table
                                    % if 1st element is 0.
else
s1=[-det([s3(1) s3(2);s2(1) s2(2)])/s2(1)...
  -det([s3(1) s3(3);s2(1) s2(3)])/s2(1) 0 0];
                                    % Create s^1 row of Routh table
                                    % if 1st element is not zero.
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
s0=[-det([s2(1) s2(2);s1(1) s1(2)])/s1(1)...
  -det([s2(1) s2(3);s1(1) s1(3)])/s1(1) 0 0];
                                    % Create s^0 row of Routh table.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
's5'                                % Display label.
s5=sym(s5);                         % Convert s5 to a symbolic object.
s5=simplify(s5);                    % Simplify terms in s^5 row.
pretty(s5)                          % Pretty print s^5 row.
's4'                                % Display label.
s4=sym(s4);                         % Convert s4 to a symbolic object.
s4=simplify(s4);                    % Simplify terms in s^4 row.
pretty(s4)                          % Pretty print s^4 row.
's3'                                % Display label.
s3=sym(s3);                         % Convert s3 to a symbolic object.
s3=simplify(s3);                    % Simplify terms in s^3 row.
pretty(s3)                          % Pretty print s^3 row.
's2'                                % Display label.
s2=sym(s2);                         % Convert s2 to a symbolic object.
s2=simplify(s2);                    % Simplify terms in s^2 row.
pretty(s2)                          % Pretty print s^2 row.
's1'                                % Display label.
s1=sym(s1);                         % Convert s1 to a symbolic object.
s1=simplify(s1);                    % Simplify terms in s^1 row.
pretty(s1)                          % Pretty print s^1 row.
's0'                                % Display label.
s0=sym(s0);                         % Convert s0 to a symbolic object.
s0=simplify(s0);                    % Simplify terms in s^0 row.
pretty(s0)                          % Pretty print s^0 row.
pause
```

ch6apF2 (Example 6.9)

MATLAB's Symbolic Math Toolbox also may be used conveniently to calculate the values in a Routh table that contains a variable gain, *K*. The technique is similar to the previous example, ch6sp1, except that `K`, rather than `e`, is used as the symbolic object. We now demonstrate the solution of Example 6.9 in the text using MATLAB and MATLAB's Symbolic Math Toolbox.

```
'(ch6apF2) Example 6.9'             % Display label.
% -det([si() si();sj() sj()])/sj()
                                    % Template for use in each cell.
syms K                              % Construct a symbolic object for
                                    % gain, K.
s3=[1 77 0 0];                      % Create s^3 row of Routh table.
```

```
s2=[18 K 0 0];                          % Create s^2 row of Routh table.
s1=[-det([s3(1) s3(2);s2(1) s2(2)])/s2(1)...
  -det([s3(1) s3(3);s2(1) s2(3)])/s2(1) 0 0];
                                        % Create s^1 row of Routh table.
s0=[-det([s2(1) s2(2);s1(1) s1(2)])/s1(1)...
  -det([s2(1) s2(3);s1(1) s1(3)])/s1(1) 0 0];
                                        % Create s^0 row of Routh table.
's3'                                    % Display label.
s3=sym(s3);                             % Convert s3 to a symbolic object.
s3=simplify(s3);                        % Simplify terms in s^3 row.
pretty(s3)                              % Pretty print s^3 row.
's2'                                    % Display label.
s2=sym(s2);                             % Convert s2 to a symbolic object.
s2=simplify(s2);                        % Simplify terms in s^2 row.
pretty(s2)                              % Pretty print s^2 row.
's1'                                    % Display label.
s1=sym(s1);                             % Convert s1 to a symbolic object.
s1=simplify(s1);                        % Simplify terms in s^1 row.
pretty(s1)                              % Pretty print s^1 row.
's0'                                    % Display label.
s0=sym(s0);                             % Convert s0 to a symbolic object.
s0=simplify(s0);                        % Simplify terms in s^0 row.
pretty(s0)                              % Pretty print s^0 row.
pause
```

Chapter 13: Digital Control Systems

ch13apF1 (Example 13.1) MATLAB's Symbolic Math Toolbox and the command `ztrans(f)` can be used to find the z-transform of a time function, `f`, represented as $f(nT)$. MATLAB assumes that the default sampled-time independent variable is n and the default transform independent variable is z. If you want to use k instead of n, that is, $f(kT)$, use `ztrans(f,k,z)`. This command overrides MATLAB's defaults and assumes the sampled-time independent variable to be k. Let us solve Example 13.1 using MATLAB's Symbolic Math Toolbox.

```
'(ch13apF1) Example 13.1'               % Display label.
syms n T                                % Construct symbolic objects for
                                        % 'n' and 'T'.
'f(nT)'                                 % Display label.
f=n*T;                                  % Define f(nT).
pretty(f)                               % Pretty print f(nT).
'F(z)'                                  % Display label.
F=ztrans(f);                            % Find z-transform, F(z).
pretty(F)                               % Pretty print F(z).
pause
```

ch13apF2 (Example 13.2) MATLAB's Symbolic Math Toolbox and the command `iztrans(F)` can be used to find the time-sampled function represented as $f(nT)$, given its z-transform, $F(z)$. If you want the sampled time function returned as $f(kT)$, then change MATLAB's default independent sampled-time variable by using the command `iztrans(F,k)`. Let us solve Example 13.2 using MATLAB's Symbolic Math Toolbox.

```
'(ch13apF2) Example 13.2'               % Display label.
syms z k                                % Construct symbolic objects for
                                        % 'z' and 'k'.
'F(z)'                                  % Display label.
```

```
F=0.5*z/((z-0.5)*(z-0.7));          % Define F(z).
pretty (F)                          % Pretty print F(z).
'f(kT)'                             % Display label.
f=iztrans(F,k);                     % Find inverse z-transform, f(kT).
pretty(f)                           % Pretty print f(kT).
'f(nT)'                             % Display label.
f=iztrans(F);                       % Find inverse z-transform, f(nT).
pretty(f)                           % Pretty print f(nT).
pause
```

ch13apF3 (Example 13.4) MATLAB's Symbolic Math Toolbox can be used to find the z-tansform of a transfer function, $G(s)$, in cascade with a z.o.h. Two new commands are introduced. The first, `compose(f,g)`, allows a variable g to replace the variable t in $f(t)$. We use this command to replace t in $g_2(t)$ with nT before taking the z-transform. The other new command is `subs(S,old,new)`. `Subs` stands for symbolic substitution. `Old` is a variable contained in `S`. `New` is a numerical or symbolic quantity to replace `old`. We use `subs` to replace T in $G(z)$ with a numerical value. To find the z-transform of a transfer function, $G(s)$, in cascade with a z.o.h. by using MATLAB's Symbolic Math Toolbox, we perform the following steps: (1) Construct $G_2(s) = G(s)/s$; (2) find the inverse Laplace transform of $G_2(s)$; (3) replace t with nT in $g_2(t)$; (4) find $G(z) = (1 - z^{-1})G_2(z)$; (5) substitute a numerical value for T. Let us solve Example 13.4 using MATLAB's Symbolic Math Toolbox.

```
'(ch13apF3) Example 13.4'           % Display label.
syms s z n T                        % Construct symbolic objects for
                                    % 's', 'z', 'n', and 'T'.
G2s=(s+2)/(s*(s+1));                % Form G2(s)=G(s)/s.
'G2(s)=G(s)/s'                      % Display label.
pretty(G2s)                         % Pretty print G2(s).
'g2(t)'                             % Display label.
g2t=ilaplace(G2s);                  % Find g2(t).
pretty(g2t)                         % Pretty print g2(t).
g2nT=compose(g2t,n*T);              % Find g2 (nT).
'g2(nT)'                            % Display label.
pretty(g2nT)                        % Pretty print g2(nT).
Gz=(1-z^-1)* ztrans(g2nT);          % Find G(z)=(1-z^-1)G2(z).
Gz=simplify(Gz);                    % simplify G(z).
'G(z)=(1-z^-1)G2(z)'                % Display label.
pretty(Gz)                          % Pretty print G(z).
Gz=subs(Gz,T,0.5);                  % Let T=0.5 in G(z).
Gz=vpa(simplify(Gz),4);             % Simplify G(z) and evaluate
                                    % numerical values to 4 places.
'G(z) evaluated for T=0.5'          % Display label.
pretty(Gz)                          % Pretty print G(z) with numerical
                                    % values.
pause
```

F.3 Command Summary

`diff (S,'x')`	Differentiate the symbolic function, S, with respect to variable, x.
`compose(f,g)`	Substitute $g(y)$ for x in $f(x)$.
`expand(x)`	Expand a symbolic function.
`ilaplace(X)`	Find inverse Laplace transform of X(s).
`int(S,v,a,b)`	Integrate S with respect to v from lower limit a to upper limit b.

`iztrans(F,k)`	Find inverse *z*-transform. Finds *f(kT)* given *F(z)*. Without optional field, *k*, finds *f(nT)*.
`laplace(x)`	Find Laplace transform of *x(t)*.
`numden(G)`	Extract symbolic numerator and denominator from *G(s)*.
`pretty(x)`	Pretty print *x*.
`simple(x)`	Find simplest from of symbolic object *x*.
`simplify(x)`	Simplify *x*.
`subs(S,old,new)`	Substitute new for old in symbolic *S*.
`sym(v)`	Convert *v* to a symbolic object.
`syms x y z`	Declare *x, y*, and *z* to be symbolic objects.
`sym2poly(P)`	Convert symbolic polynomial, *P*, to a vector.
`vpa(x,D)`	Use variable precision arithmetic. Convert fractional symbolic values to decimal with *D* places.
`ztrans(f)`	Find *z*-transform of *f(nT)*.

Bibliography

MathWorks. *Control System Toolbox™ Getting Started Guide R2018b.* MathWorks, Natick, MA, 2000–2018.

MathWorks. *MATLAB® Primer R2018.* MathWorks, Natick, MA, 1984–2018.

MathWorks. *Symbolic Math Toolbox™ User's Guide R2018.* MathWorks, Natick, MA, 1993–2018.