MATLAB AND MATHSCRIPT: A SHORT INTRODUCTION FOR USE IN SIGNALS AND SYSTEMS

BACKGROUND:

"A computer will always do exactly what you tell it to do. But that may not be what you had in mind"–a quote from the 1950's.

This Appendix is a short introduction to MATLAB and Mathscript *for this book*. It is not comprehensive; only commands directly applicable to signals and systems are covered. No commands in any of MATLAB's Toolboxes are included, since these commands are not included in basic MATLAB or Mathscript. Programming concepts and techniques are not included, since they are not used anywhere in this book.

MATLAB:

MATLAB is a computer program developed and sold by the Mathworks, Inc. It is the most commonly used program in signal processing, but it is used in all fields of engineering.

"MATLAB" is an abbreviation of MATrix LABoratory. It was originally based on a set of numerical linear algebra programs, written in FORTRAN, called LINPACK. So MATLAB tends to formulate problems in terms of vectors and arrays of numbers, and often solves problems by formulating them as linear algebra problems.

For example, MATLAB computes the roots of a polynomial by computing the eigenvalues of the companion matrix formed from the polynomial coefficients. When using MATLAB, one usually formulates a problem in terms of arrays or vectors of numbers, which are defined below.

Mathscript:

Mathscript is a computer program developed and sold by National Instruments, as a module in LabView. The basic commands used by MATLAB also work in Mathscript, but higher-level MATLAB commands, and those in Toolboxes, usually do not work in Mathscript. Unless otherwise noted, all MATLAB commands used in this book and CD also work in Mathscript.

A student version of Mathscript is included on this CD. *Access to MATLAB is not required to use this book.* In the sequel, we use "M/M" to designate "MATLAB or Mathscript."

Getting Started:

To install the student version of Mathscript included on this CD, follow the instructions.

When you run M/M, a **prompt** >> will appear when it is ready. Then you can type commands. Your first command should be >>`cd mydirectory`, to change directory to your working directory, which we call "mydirectory" here.

We will use `this font` to represent typed commands and generated output. You can get help for any command, such as `plot`, by typing at the prompt `help plot`.

Some basic things to know about M/M:

- Putting a **semicolon** ";" at the end of a command suppresses output; without it M/M will type the results of the computation. This is harmless, but it is irritating to have numbers flying by on your screen.
- Putting **ellipses** "..." at the end of a command means it is continued on the next line. This is useful for long commands.
- Putting "%" at the beginning of a line makes the line a **comment**; it will not be executed. Comments are used to explain what the program is doing at that point.
- `clear` eliminates all present variables. Programs should start with a `clear`.
- `whos` shows all variables and their sizes.
- M/M variables are case-sensitive: `t` and `T` are different variables.
- `save myfile X,Y` saves the variables `X` and `Y` in the file myfile.mat for use in another session of M/M at another time.
- `load myfile` loads all variables saved in myfile.mat, so they can now be used in the present session of M/M.
- `quit` ends the present session of M/M.

.m Files:

An M/M program is a list of commands executed in succession. Programs are called "m-files" since their extension is ".m," or "scripts."

To write an .m file, at the upper left, click:
File→New→m-file
This opens a window with a text editor.
Type in your commands and then do this:
File→Save as→myname.m
Make sure you save it with an .m extension. Then you can run the file by typing its name at the prompt: >> `myname`. Make sure the file name is not the same as a Matlab command! Using your own name is a good idea.

You can access previously-typed commands using uparrow and downarrow on your keyboard.

To *download a file* from a web site or the CD, right-click on it, select **save target as**, and use the menu to select the proper file type (specified by its file extension).

BASIC COMPUTATION:

Basic Arithmetic:

- Addition: `3+2` gives `ans=5`
- Subtraction: `3-2` gives `ans=1`
- Multiplication: `2*3` gives `ans=6`
- Division: `6/2` gives `ans=3`
- Powers: `2^3` gives `ans=8`
- Others: `sin,cos,tan,exp,log,log10`
- Square root: `sqrt(49)` gives `ans=7`
- Conjugate: `conj(3+2j)` gives `ans=3-2i`

Both `i` or `j` represent $\sqrt{-1}$; answers use `i`. `pi` represents π. `e` does not represent 2.71828.

Entering Vectors and Arrays:

To enter *row vector* [1 2 3] and store it in `A` type at the prompt `A=[1 2 3];` or `A=[1,2,3];`

To enter the same numbers as a *column vector* and store it in `A`, type at the prompt *either* `A=[1;2;3];` *or* `A=[1 2 3];A=A';` Note `A=A'` replaces `A` with its transpose. "Transpose" means "convert rows to columns, and vice-versa."

To enter a vector of consecutive or equally-spaced numbers, follow these examples:

- `[2:6]` gives `ans=2 3 4 5 6`
- `[3:2:9]` gives `ans=3 5 7 9`
- `[4:-1:1]` gives `ans=4 3 2 1`

To enter an *array* or matrix of numbers, type, for example, `B=[3 1 4;1 5 9;2 6 5];` This gives the array `B` and its transpose `B'`

$$B = \begin{bmatrix} 3 & 1 & 4 \\ 1 & 5 & 9 \\ 2 & 6 & 5 \end{bmatrix} \qquad B' = \begin{bmatrix} 3 & 1 & 2 \\ 1 & 5 & 6 \\ 4 & 9 & 5 \end{bmatrix}$$

Other basics of arrays:

- `ones(M,N)` is an $M \times N$ array of "1"
- `zeros(M,N)` is an $M \times N$ array of "0"
- `length(X)` gives the length of vector `X`
- `size(X)` gives the size of array `X`
  For `B` above, `size(B)` gives `ans=3 3`
- `A(I,J)` gives the (I,J)th element of `A`.
  For `B` above, `B(2,3)` gives `ans=9`

Array Operations:

Arrays add and subtract point-by-point:
`X=[3 1 4];Y=[2 7 3];X+Y` gives `ans=5 8 7`
But `X*Y` generates an error message.
To compute various types of vector products:

- To multiply element-by-element, use `X.*Y`
  This gives `ans=6 7 12`.
  To divide element-by-element, `X./Y`
- To find the inner product of `X` and `Y`
  (3)(2)+(1)(7)+(4)(3)=25, use `X*Y'`
  This gives `ans=25`
- To find the outer product of `X` and `Y`
  $\begin{bmatrix} (3)(2) & (3)(7) & (3)(3) \\ (1)(2) & (1)(7) & (1)(3) \\ (4)(2) & (4)(7) & (4)(3) \end{bmatrix}$ use `X'*Y`
  This gives the above matrix.

A common problem is thinking you have a row vector when in fact you have a column vector. Check by using `size(X)`; here that gives `ans=1,3` which tells you that `X` is a 1×3 (row) vector.

- These functions operate on each element of an array separately, giving another array: `sin,cos,tan,exp,log,log10,sqrt` `cos([0:3]*pi)` gives `ans=1 -1 1 -1`
- To compute n^2 for $n = 0, 1 \ldots 5$, use `[0:5].^2` gives `ans=0 1 4 9 16 25`
- To compute 2^n for $n = 0, 1 \ldots 5$, use `2.^[0:5]` gives `ans=1 2 4 8 16 32`

Some other array operations:

- `A=[1 2 3;4 5 6];(A(:))'` Stacks A by columns into a column vector and transposes the result to a row vector. Here, this gives `ans=1 4 2 5 3 6`
- `reshape(A(:),2,3)` Unstacks the column vector to a 2×3 array which, in this case, is the original array `A`.
- `X=[1 4 1 5 9 2 6 5];C=X(2:8)-X(1:7)` Takes differences of successive values of `X`. Here, this gives `C=3 -3 4 4 -7 4 -1`
- `D=[1 2 3]; E=[4 5 6]; F=[D E]` This *concatenates* the vectors `D` and `E` (i.e., it appends `E` after `D` to get vector `F`) Here, this gives `F=1 2 3 4 5 6`
- `I=find(A>2)` stores in `I` locations (indices) of elements of vector `A` that exceed 2. `find([3 1 4 1 5]<2)` gives `ans=2 4`
- `A(A>2)=0` sets to 0 all values of elements of vector `A` exceeding 2. `A=[3 1 4 1 5]; A(A<2)=0` gives `A=3 0 4 0 5`

M/M indexing of arrays starts with 1, while signals and systems indexing starts with 0. For example, the DFT is defined using index $n = 0, 1 \ldots N-1$, for $k = 0, 1 \ldots N-1$. `fft(X)`, which computes the DFT of `X`, performs

```
fft(X)=X*exp(-j*2*pi*[0:N-1]'*[0:N-1]/N);
```

Solving Systems of Equations:

To solve the linear system of equations

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 17 \\ 39 \end{bmatrix}$$

using `A=[1 2;3 4];Y=[17;39];X=A\Y;X'` gives `ans=5.000 6.000`, which is the solution $[x\ y]'$.

To solve the complex system of equations

$$\begin{bmatrix} 1+2j & 3+4j \\ 5+6j & 7+8j \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 16+32j \\ 48+64j \end{bmatrix}$$

`[1+2j 3+4j;5+6j 7+8j]\[16+32j;48+64j]` gives `ans=`$\frac{2-2i}{6+2i}$, which is the solution.

These systems can also be solved using `inv(A)*Y`, but this is a bad idea, since computing the matrix inverse of `A` takes much more computation than just solving the system of equations. Computing a matrix inverse can lead to numerical difficulties for large matrices.

PLOTTING:

Plotting Basics:

To plot a function $x(t)$ for $a \le t \le b$:

- Generate, say, 100 values of t in $a \le t \le b$ using `T=linspace(a,b,100);`
- Generate and store 100 values of $x(t)$ in `X`
- Plot each computed value of `X` against its corresponding value of `T` using `plot(T,X)`
- If you are making several different plots, put them all on one page using `subplot`. `subplot(324),plot(T,X)` divides a figure into a 3-by-2 array of plots, and puts the `X` vs. `T` plot into the 4th place in the array (the middle of the right-most column).

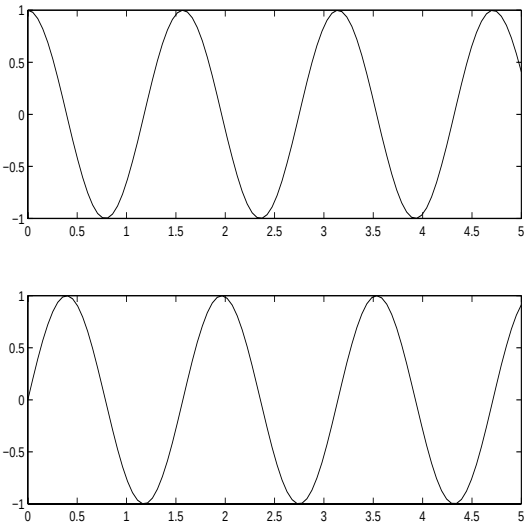
Print out the current figure (the one in the foreground; click on a figure to bring it to the foreground) by typing `print`

Print the current figure to a encapsulated postscript file myname.eps by typing `print -deps2 myname.eps`. Type `help print` for a list of printing options for your computer. For example, use `-depsc2` to save a figure in color.

To make separate plots of $\cos(4t)$ and $\sin(4t)$ for $0 \le t \le 5$ in a single figure, use the following:

```
T=linspace(0,5,100);X=cos(4*T);Y=sin(4*T);
subplot(211),plot(T,X)
subplot(212),plot(T,Y)
```

These commands produce the following figure:



The default is that `plot(X,Y)` plots each of the 100 ordered pairs `(X(I),Y(I))` for `I`=1...100, and connects the points with straight lines. If there are only a few data points to be plotted, they should be plotted as individual ordered pairs, not connected by lines. This can be done using `plot(X,Y,'o')` (see Fig. 6-76 below).

Plotting Problems:

Common problems encountered using `plot`:
`T` and `X` must have the same lengths; and
Neither `T` nor `X` should be complex; use `plot(T,real(X))` if necessary.

The above `linspace` command generates 100 equally-spaced numbers between a and b, *including a and b*. This is *not* the same as sampling $x(t)$ with a sampling interval of $\frac{b-a}{100}$. To see why:

- `linspace(0,1,10)` gives 10 numbers between 0 and 1 inclusive, spaced by 0.111
- `[0:.1:1]` gives 11 numbers spaced by 0.1

Try the following yourself on M/M:

- `T=[0:10];X=3*cos(T);plot(T,X)` This should be a very jagged-looking plot, since it is only sampled at 11 integers, and the samples are connected by lines.
- `T=[0:0.1:10];X=3*cos(T);plot(T,X)` This should be a much smoother plot, since there are now 101 (not 100) samples.
- `T=[1:4000];X=cos(2*pi*440*T/8192);` `sound(X,8192)` This is musical note "A." `sound(X,Fs)` plays `X` as sound, at a sampling rate of `Fs` samples/second.
- `plot(X)`. This should be a blue smear! It's about 200 cycles squished together.
- `plot(X(1:100))` This "zooms in" on the first 100 samples of X to see the sinusoid.

More Advanced Plotting:

Plots should be labelled and annotated:

- `title('Myplot')` adds the title "Myplot"
- `xlabel('t')` labels the x-axis with "t"
- `ylabel('x')` labels the y-axis with "x"
- `$\omega$` produces ω in `title,xlabel` and `ylabel` Similarly for other Greek letters. Note ', not ', should be used everywhere.
- `axis tight` contracts the plot borders to the limits of the plot itself
- `axis([a b c d])` changes the horizontal axis limits to $a \le x \le b$ and the vertical axis limits to $c \le y \le d$.
- `grid on` adds grid lines to the plot
- `plot(T,X,'g',T,Y,'r')` plots on the same plot (`T,X,Y` must all have the same lengths) `X` vs. `T` in green and `Y` vs. `T` in red.

Plotting Examples:

A good way to learn how to plot is to study specific examples. The figures in this book were redrawn from figures generated using MATLAB. Four specific illustrative examples follow.

Figure 4-32(b):

This example shows how to plot two different functions in a single plot, using different colors, and insert a title. The following .m file

```
clear;T=linspace(0,600,1000);
A=0.01;K=0.04;B=A+K;
SA=3*(1-exp(-A*T));
SB=3*(1-exp(-B*T));
plot(T,SA,'b',T,SB,'r')
title('STEP RESPONSE WITH
AND WITHOUT FEEDBACK')
grid on,print -depsc2 m1.eps
```

generates the following figure:

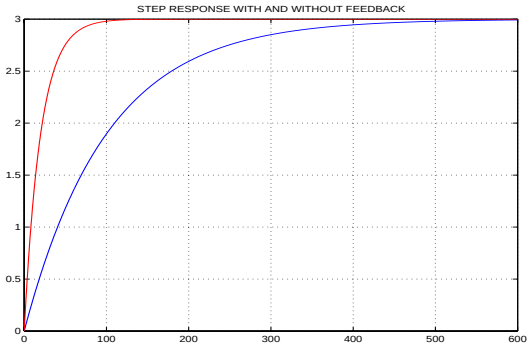


Figure 6-24(b):

This example demonstrates `./` and `.*` The following .m file

```
clear;W=linspace(-8,8,1000);V=j*W;
N=(V+2j).*(V-2j).*(V-.1+4j).*(V-.1-4j);
D1=(V+.5+1j).*(V+.5-1j).*(V+.5+3j);
D2=(V+.5-3j).*(V+.5+5j).*(V+.5-5j);
plot(W,abs(N./D1./D2),'r'),grid on
```

generates the following figure:

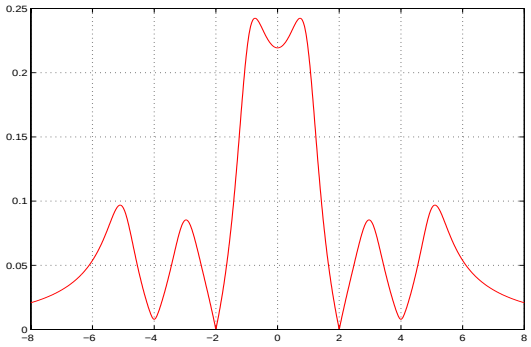


Figure 6-38(b):

This example shows how to use a loop: `H=H./(V-P(I));` is executed for `I=1,2,3,4,5` in succession. The following .m file

```
clear;W=linspace(-2,2,1000);V=j*W;
P=exp(j*2*pi*[3:7]/10);
H=ones(1,1000);
for I=1:5;H=H./(V-P(I));end
subplot(211),plot(W,abs(H)),grid on
```

generates the following figure:

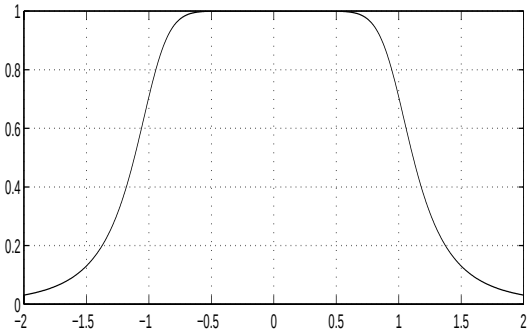
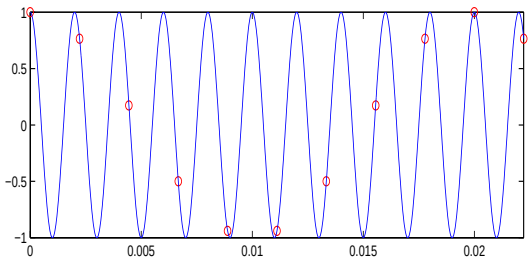


Figure 6-76:

This example shows how to use `hold` to superpose two plots, how to plot individual points, use `subplot` to change the aspect ratio of a figure, and `axis tight` to tighten it. It also uses `[a:0.001:b]`, not `linspace(a,b,1000)`, to sample every 0.001. The following .m file

```
T1=[0:1/45000:1/45];
X1=cos(2*pi*500*T1);
T2=[0:1/450:1/45];
X2=cos(2*pi*500*T2);
subplot(211),plot(T2,X2,'or'),hold,
subplot(211),plot(T1,X1),axis tight
```

generates the following figure:



These do not include the computer examples, whose programs are listed elsewhere on this CD.

PARTIAL FRACTIONS:

Rectangular-to-Polar Complex Conversion:

If an M/M result is a complex number, then it is presented in its rectangular form `a+bj`. M/M recognizes both `i` and `j` as $\sqrt{-1}$, so that complex numbers can be entered as `3+2j`.

To convert a complex number `X` to polar form, use `abs(X),angle(X)` to get its magnitude and phase (in radians), respectively. To get its phase in degrees, use `angle(X)*180/pi`

Note `atan(imag(X)/real(X))` will **not** give the correct phase, since this formula is only valid if the real part is positive. `angle` corrects this.

The real and imaginary parts of `X` are found using `real(X)` and `imag(X)`, respectively.

Polynomial Zeros:

To compute the zeros of a polynomial, enter its coefficients as a *row* vector `P` and use `R=roots(P)`. For example, to find the zeros of $3x^3$–$21x$+18 (the roots of $3x^3$–$21x$+18=0) use `P=[3 0 -21 18];R=roots(P);R'`, giving `ans= -3.0000 2.0000 1.0000`, which are the roots.

To find the monic (leading coefficient one) polynomial having a given set of numbers for zeros, enter the numbers as a *column* vector `R` and use `P=poly(R)`. For example, to find the polynomial having $\{1, 3, 5\}$ as its zeros, use `R=[1;3;5];P=poly(R)`, giving `P=1 -9 23 -15`. The polynomial is therefore x^3–$9x^2$+$23x$–15.

Note that polynomial are stored as row vectors, and roots are stored as column vectors.

Pole-zero diagrams are made using `zplane`, in the MATLAB Signal Processing Toolbox. To produce the pole-zero diagram of $\mathbf{H(z)}=\frac{\mathbf{z^2+3z+2}}{\mathbf{z^2+5z+6}}$, type `zplane([1 3 2],[1 5 6])`. The unit circle $|\mathbf{z}|$=1 is also plotted, as a dotted line.

Partial Fraction Expansions:

Partial fraction expansions are a vital part of signals and systems, and their computation is onerous (see Chapter 3). M/M computes partial fraction expansions using `residue`. Specifically,

$$\mathbf{H(s)} = \frac{b_0\mathbf{s}^M + b_1\mathbf{s}^{M-1} + \ldots + b_M}{a_0\mathbf{s}^N + a_1\mathbf{s}^{N-1} + \ldots + a_N}$$

has the partial fraction expansion (if $M \leq N$)

$$\mathbf{H(s)} = K + \frac{\mathbf{R_1}}{\mathbf{s-p_1}} + \ldots + \frac{\mathbf{R_N}}{\mathbf{s-p_N}}$$

The poles $\{\mathbf{p_i}\}$ and residues $\{\mathbf{R_i}\}$ can be computed from coefficients $\{a_i\}$ and $\{b_i\}$ using

`B=`$[b_0\ b_1 \ldots b_M]$`;A=`$[a_0\ a_1 \ldots a_N]$
`[R P]=residue(B,A);[R P]`

The residues $\{R_i\}$ are given in column vector `R`, and poles $\{p_i\}$ are given in column vector `P`.

To compute the partial fraction expansion of $\mathbf{H(s)} = \frac{3\mathbf{s}+6}{\mathbf{s}^2+5\mathbf{s}+4}$, use the command

`[R P]=residue([3 6],[1 5 4]);[R P]`

This gives $\begin{bmatrix} 2 & -4 \\ 1 & -1 \end{bmatrix}$, so `R=`$\begin{bmatrix} 2 \\ 1 \end{bmatrix}$ and `P=`$\begin{bmatrix} -4 \\ -1 \end{bmatrix}$ from which we read off $\mathbf{H(s)}=\frac{2}{\mathbf{s}+4}+\frac{1}{\mathbf{s}+1}$.

In practice, the poles and residues both often occur in complex conjugate pairs. Then use

$$Re^{pt} + R^*e^{p*t} = 2|R|e^{at}\cos(\omega t + \theta),$$

$R=|R|e^{j\theta}$ and $p=a+j\omega$, to simplify the result.

To compute the partial fraction expansion of $\mathbf{H(s)} = \frac{\mathbf{s}+7}{\mathbf{s}^2+8\mathbf{s}+25}$, use the command

`[R P]=residue([1 7],[1 8 25]);[R P]`

This gives $\begin{bmatrix} 0.5000 - 0.5000i & -4.000 + 3.000i \\ 0.5000 + 0.5000i & -4.000 - 3.000i \end{bmatrix}$ from which we have $\mathbf{H(s)}=\frac{0.5-j0.5}{\mathbf{s}+4-j3}+\frac{0.5+j0.5}{\mathbf{s}+4+j3}$

which has the inverse Laplace transform

$h(t)$=(0.5–j0.5)$e^{(-4+j3)t}$+(0.5+j0.5)$e^{(-4-j3)t}$

From `abs(0.5-0.5j),angle(0.5-0.5j)`,

$h(t)=2\frac{\sqrt{2}}{2}e^{-4t}\cos(3t-\frac{\pi}{4})=\sqrt{2}e^{-4t}\cos(3t-\frac{\pi}{4})$.

Both $h(t)$ expressions are valid for $t > 0$.

If $\mathbf{H(s)}$ is proper but not strictly proper, the constant K is nonzero. It is computed using

`[R P K]=residue(B,A);[R P],K`

since `K` has size different from `R` and `P`.

To find the partial fraction expansion of $\mathbf{H(s)}=\frac{\mathbf{s}^2+8\mathbf{s}+9}{\mathbf{s}^2+3\mathbf{s}+2}$, use the command

`[R P K]=residue([1 8 9],[1 3 2]);[R P]`

K gives $\begin{bmatrix} 3 & -2 \\ 2 & -1 \end{bmatrix}$, K=1 so R=$\begin{bmatrix} 3 \\ 2 \end{bmatrix}$, P=$\begin{bmatrix} -2 \\ -1 \end{bmatrix}$ from which we read off $\mathbf{H}(\mathbf{s})=1+\frac{3}{\mathbf{s}+2}+\frac{2}{\mathbf{s}+1}$.

Double poles are handled as follows:

To find the partial fraction expansion of $\mathbf{H}(\mathbf{s})=\frac{8\mathbf{s}^2+33\mathbf{s}+30}{\mathbf{s}^3+5\mathbf{s}^2+8\mathbf{s}+4}$, use the command

```
[R P]=residue([8 33 30],[1 5 8 4]);
```

[R P] gives $\begin{bmatrix} 3 & -2 \\ 4 & -2 \\ 5 & -1 \end{bmatrix}$, so R=$\begin{bmatrix} 3 \\ 4 \\ 5 \end{bmatrix}$, P=$\begin{bmatrix} -2 \\ -2 \\ -1 \end{bmatrix}$

We then read off $\mathbf{H}(\mathbf{s})=\frac{3}{\mathbf{s}+2}+\frac{4}{(\mathbf{s}+2)^2}+\frac{5}{\mathbf{s}+1}$.

In practice, we are interested not in an analytic expression for $h(t)$, but in computing $h(t)$ sampled every Ts seconds. These samples can be computed directly from R and P, for $0 \le t \le T$:

```
t=[0:Ts:T];H=real(R.'*exp(P*t));
```

Since R and P are column vectors, and t is a row vector, H is the inner products of R with each column of the array exp(P*t). R.' transposes R without also taking complex conjugates of its elements. real is necessary since roundoff error creates a tiny (incorrect) imaginary part in H.

Frequency Response:

polyval(P,W) evaluates the polynomial whose coefficents are stored in row vector P at the elements of vector W. For example, to evaluate the polynomial x^2–$3x$+2 at x=4, polyval([1 -3 2],4) gives ans=6

The continuous-time **frequency response** of

$$\mathbf{H}(\mathbf{s}) = \frac{b_0\mathbf{s}^M + b_1\mathbf{s}^{M-1} + \ldots + b_M}{a_0\mathbf{s}^N + a_1\mathbf{s}^{N-1} + \ldots + a_N}$$

can be plotted for $0 \le \omega \le W$ using

B=[b_0 $b_1 \ldots b_M$];A=[a_0 $a_1 \ldots a_N$]

```
w=linspace(0,W,1000);
H=polyval(B,j*w)./polyval(A,j*w);
subplot(211),plot(w,abs(H))
```

Discrete-Time Commands

- stem(X) produces a stem plot of X
- conv(X,Y) convolves X and Y
- fft(X,N) computes the N-point DFT of X
- ifft(F) computes the inverse DFT of F. Due to roundoff error, use real(ifft(F)).
- sinc(X) compute $\frac{\sin(\pi x)}{\pi x}$ for each element.

Figure 6-30

We combine many of the above commands to show how Fig. 6-30 in the book was produced.

Example 6-8 plots the impulse and magnitude frequency responses of a comb filter that eliminates 1-kHz and 2-kHz sinusoids, using poles with a real part of –100. Both plots are given in Fig. 6-30, which was redrafted from plots from: (% denotes a comment statement)

```
F=linspace(0,3000,100000);
W=j*2*pi*F;A=100;
%Compute frequency response:
Z=j*2*pi*1000*[-2 -1 1 2];
N=poly(Z);D=poly(Z-A);
FH=polyval(N,W)./polyval(D,W);
subplot(211),plot(F,abs(FH))
%Compute impulse response:
T=linspace(0,0.05,10000);
[R P K]=residue(N,D);
H=real(R.'*exp(P*T));
subplot(212),plot(T,H)
```