Chapter 2

- **Simulation Techniques**

References:

- S.M.Kay, *Fundamentals of Statistical Signal Processing: Estimation Theory,* Prentice Hall, 1993
- C.L.Nikias and M.Shao, *Signal Processing with Alpha-Stable Distribution and Applications*, John Wiley & Sons, 1995
- V.K.Ingle and J.G.Proakis, *Digital Signal Processing Using MATLAB V.4*, PWS Publishing Company, 1997
- E.Part-Enander, A.Sjoberg, B.Melin and P.Isaksson, *The MATLAB Handbook*, Addison-Wesley, 1996
- S.K.Park and K.W.Miller, "Random number generators: good ones are hard to find," *Communications of the ACM*, vol.31, no.10, Oct. 1988

Simulation Techniques

Signal Generation

1. Deterministic Signals

It is trivial to generate deterministic signals given the synthesis formula, e.g., for a single real tone, it is generated by

$$x(n) = A\cos(\omega n + \theta), \qquad n = 0,1,\cdots,N-1$$

MATLAB code:

```
N=10;                          % number of samples is 10
A=1;                           % tone amplitude is 1
w=0.2;                         % frequency is 0.2
p=1;                           % phase is 1

for n=1:N
      x(n)=A*cos(w*(n-1)+p);   % note that index should be > 0
end
```

An alternative approach is

```
n=0:N-1;                        % define a vector of size N
x = A.*cos(w.*n+p);             % the first time index is also 1
                                % “.*” is used in vector multiplication
```

Both give

```
x =

  Columns 1 through 7

    0.5403    0.3624    0.1700   -0.0292   -0.2272   -0.4161   -0.5885

  Columns 8 through 10

   -0.7374   -0.8569   -0.9422
```

Q.: Which approach is better? Why?

Example 2.1

Recall the simple mathematical model of a musical signal:

$$x(t) = a(t) \sum_{m=1}^{\infty} c_m \cos(2\pi m f_0 t + \phi_m)$$

A further simplified form is

$$x(t) = \cos(2\pi f_0 t)$$

where each music note has a distinct f_0.

Let's the following piece of music:

A A	E E	F# F#	E E	
D D	C#C#	B B	A A	
E E	D D	C# C#	B B	(repeat once)

(repeat first two lines once)

The American Standard pitch for each of these notes is:

A:	440.00 Hz
B:	493.88 Hz
C#:	554.37 Hz
D:	587.33 Hz
E:	659.26 Hz
F#:	739.99 Hz

Assuming that each note lasts for 0.5 second and a sampling frequency of 8000 Hz, the MATLAB code for producing this piece of music is:

```
a=sin(2*pi*440*(0:0.000125:0.5));        % frequency for A
b=sin(2*pi*493.88*(0:0.000125:0.5));     % frequency for B
cs=sin(2*pi*554.37*(0:0.000125:0.5));    % frequency for C#
d=sin(2*pi*587.33*(0:0.000125:0.5));     % frequency for D
e=sin(2*pi*659.26*(0:0.000125:0.5));     % frequency for E
fs=sin(2*pi*739.99*(0:0.000125:0.5));    % frequency for F#
```

```
line1=[a,a,e,e,fs,fs,e,e];                    % first line of song
line2=[d,d,cs,cs,b,b,a,a];                    % second line of song
line3=[e,e,d,d,cs,cs,b,b];                    % third line of song

song=[line1,line2,line3,line3,line1,line2];   % composite song

sound(song,8000);            % play sound with 8kHz sampling frequency

wavwrite(song,'song.wav');   % save song as a “wav” file
```

Note that in order to attain better music quality (e.g., flute, violin), we should use the more general model:

$$x(t) = a(t) \sum_{m=1}^{\infty} c_m \cos(2\pi m f_0 t + \phi_m)$$

Q.: How many discrete-time samples in the 0.5 second note with 8000 Hz sampling frequency?

Q.: How to change the sampling frequency to 16000 Hz?

2. Random Signals

- Uniform Variable

A uniform random sequence can be generated by

$$x(n) = seed_n = (a \cdot seed_{n-1}) \bmod(m), \qquad n = 1,2,\cdots$$

where $seed_0$, a and m are positive integers. The numbers generated should be (approximately) uniformly distributed between 0 and $(m-1)$.
A set of choice for a and m which generates good uniform variables is

$$a = 16807 \qquad \text{and} \qquad m = 2147483647$$

This uniform PDF can be changed easily by scaling and shifting the generation formula. For example, a random number which is uniformly between –0.5 and 0.5 is given by

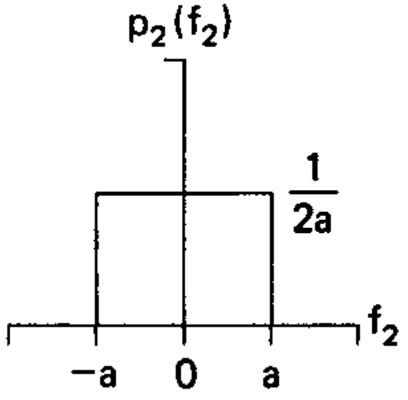
$$seed_n = (a \cdot seed_{n-1}) \bmod(m)$$

$$x(n) = \frac{seed_n}{m} - 0.5$$

The power of $x(n)$ is

$$\mathrm{var}(x) = \int_{-0.5}^{0.5} x^2 \cdot p(x) \cdot dx = \int_{-0.5}^{0.5} x^2 \cdot dx$$

$$= \frac{1}{12}$$



Note that $x(n)$ is independent (white).

To generate a white uniform number with variance σ_x^2:

$$seed_n = (a \cdot seed_{n-1}) \bmod(m)$$

$$x(n) = \left(\frac{seed_n}{m} - 0.5\right) \cdot \sqrt{12} \cdot \sigma_x$$

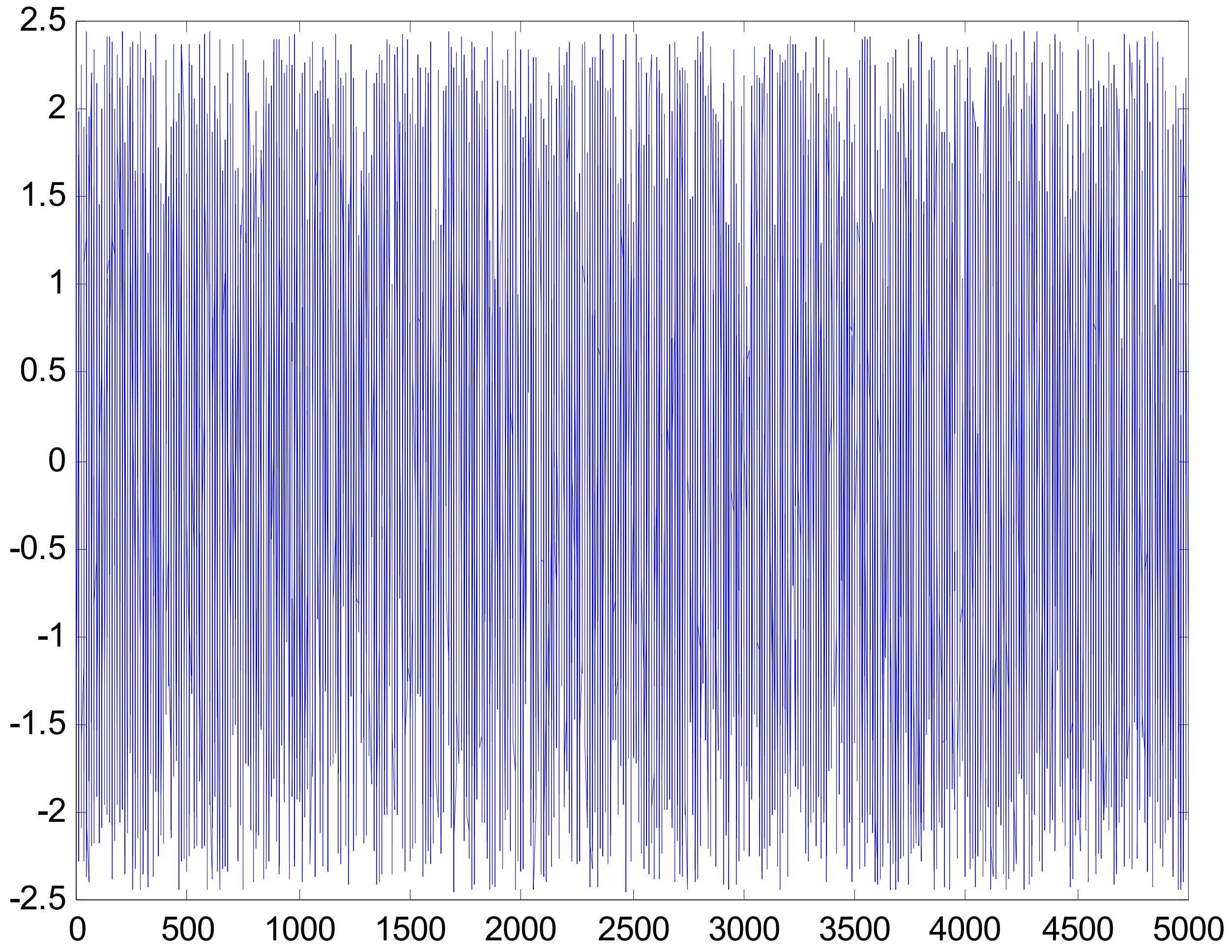
MATLAB code for generating zero-mean uniform numbers with power 2:

```
N=5000;                                  % number of samples is 5000
power = 2;                               % signal power is 2
u = (rand([1,N])-0.5).*sqrt(12*power);   % “rand” give a uniform number
                                         %  in [0,1]
```

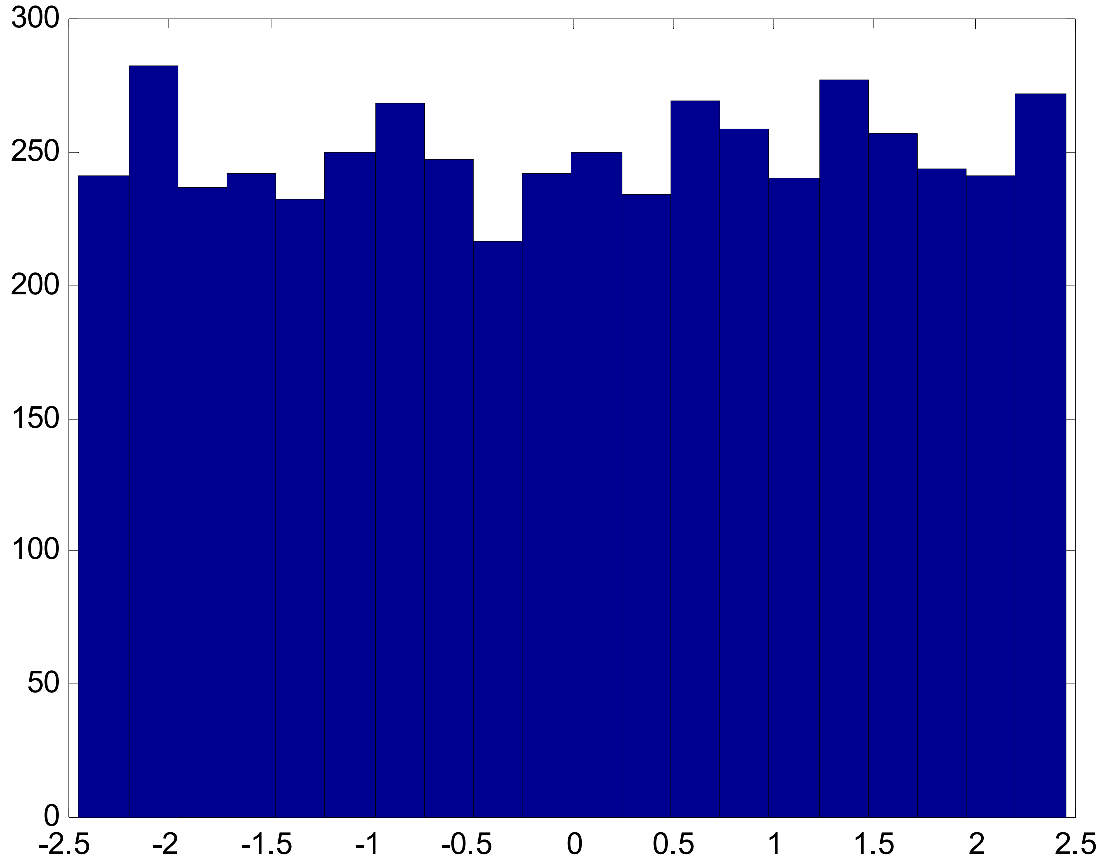
Evaluation of MATLAB uniform random numbers:

```
m = mean(u)                    % * “mean” computes the time average

⇒                m =  0.0172


p = mean(u.*u)                 %  compute power

⇒                p = 2.0225


y = mean((u-m).*(u-m))         %  compute variance

⇒                v = 2.0222
```

```
plot(u);                        % plot the signal
```

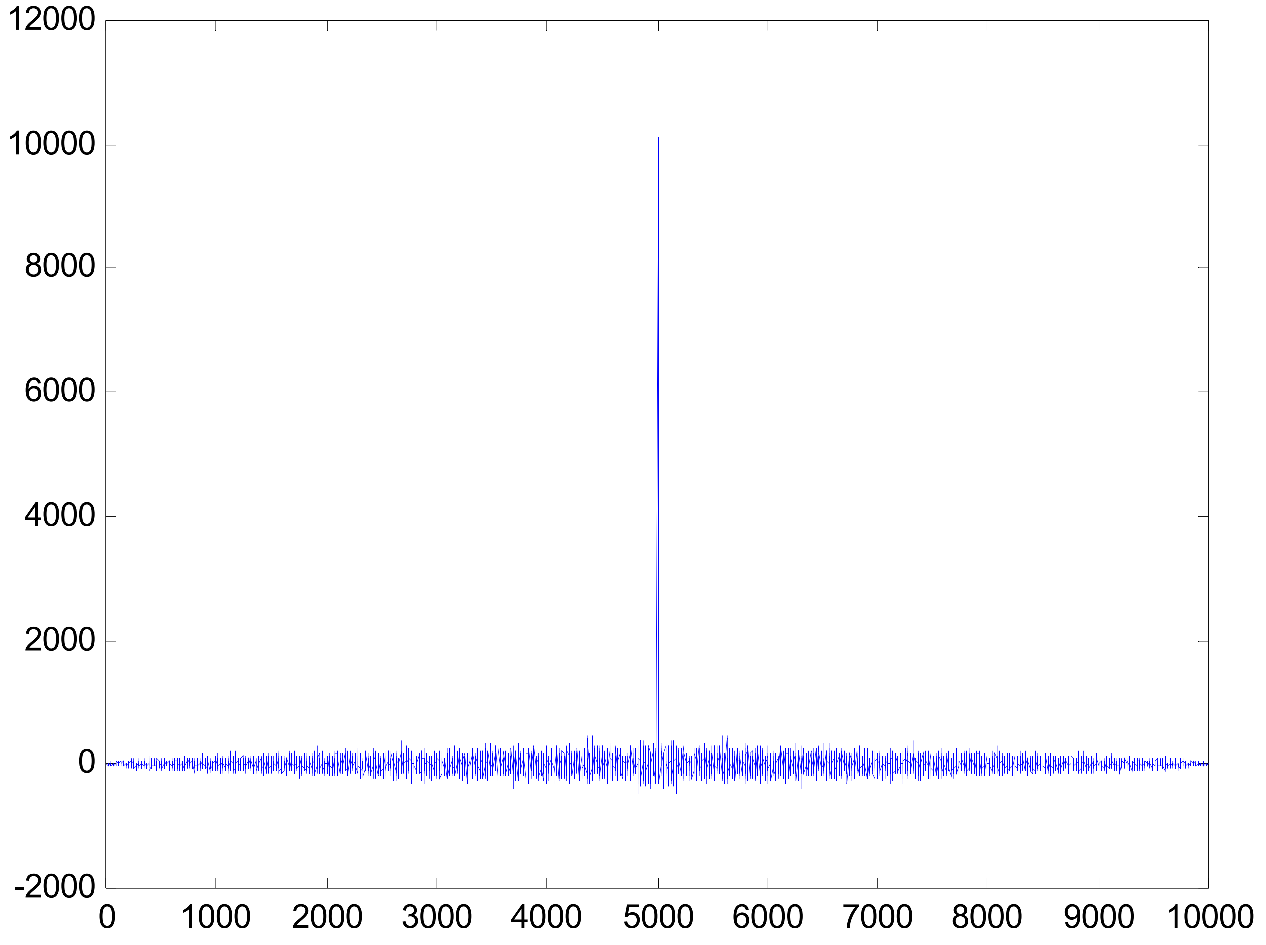


```
hist(u,20)                    % plot the histogram for u
                              % with 20 bars
```

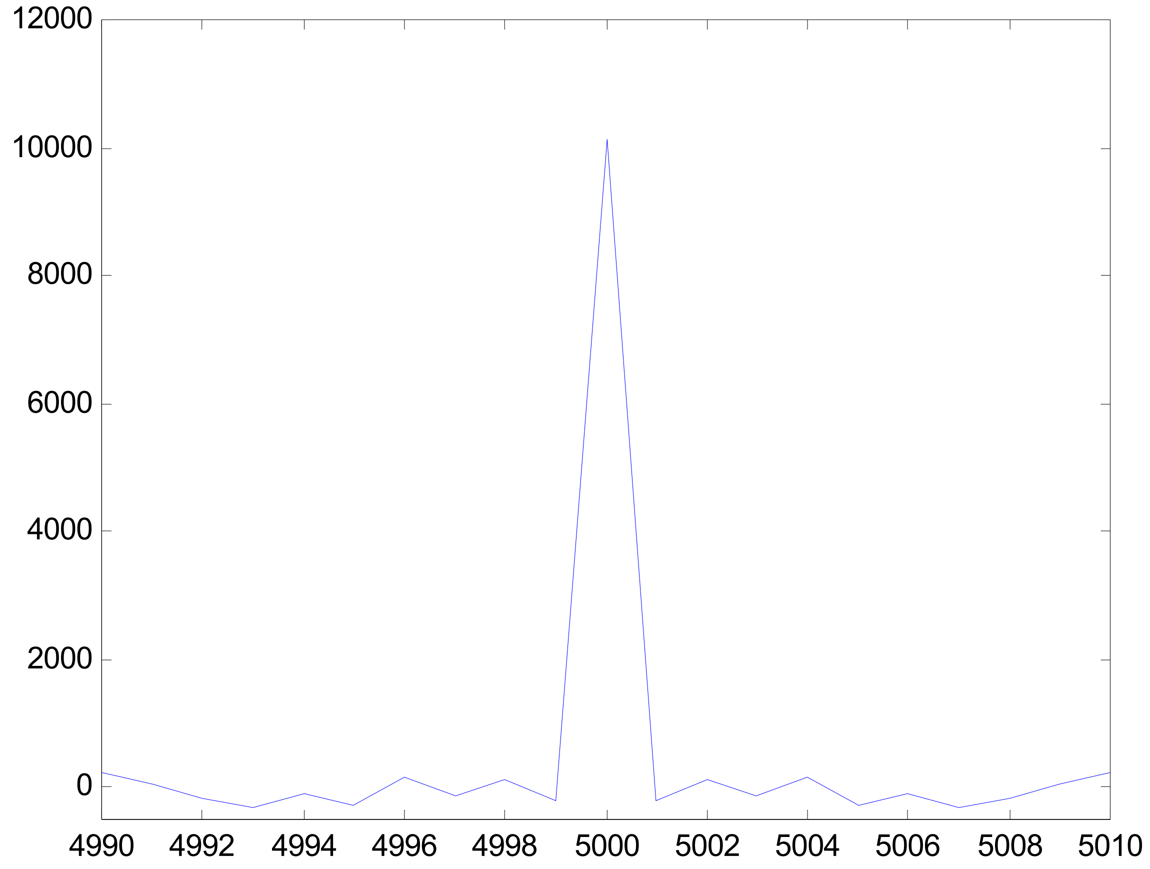


Q.: Is the random generator acceptable? Does ergodicity hold?

```
a = xcorr(u);                    %  compute the autocorrelation
plot(a)                          %  plot the autocorrelation
```



```
axis([4990, 5010, -500, 12000])   % change the axis
```



The time index at 5000 corresponds to $R_{uu}(0)$

$\Rightarrow$ white

- Gaussian Variable

Given a pair of independent uniform numbers which are uniformly distributed between [0,1], say, (u_1, u_2), a pair of independent Gaussian numbers, which have zero-mean and unity variance, can be generated from:

$$w_1 = \sqrt{-2\ln(u_1)} \cdot \cos(2\pi u_2)$$

$$w_2 = \sqrt{-2\ln(u_1)} \cdot \sin(2\pi u_2)$$

This is known as the Box-Mueller transformation. Note that the Gaussian numbers are white.

MATLAB code for generating zero-mean Gaussian numbers with power 2:

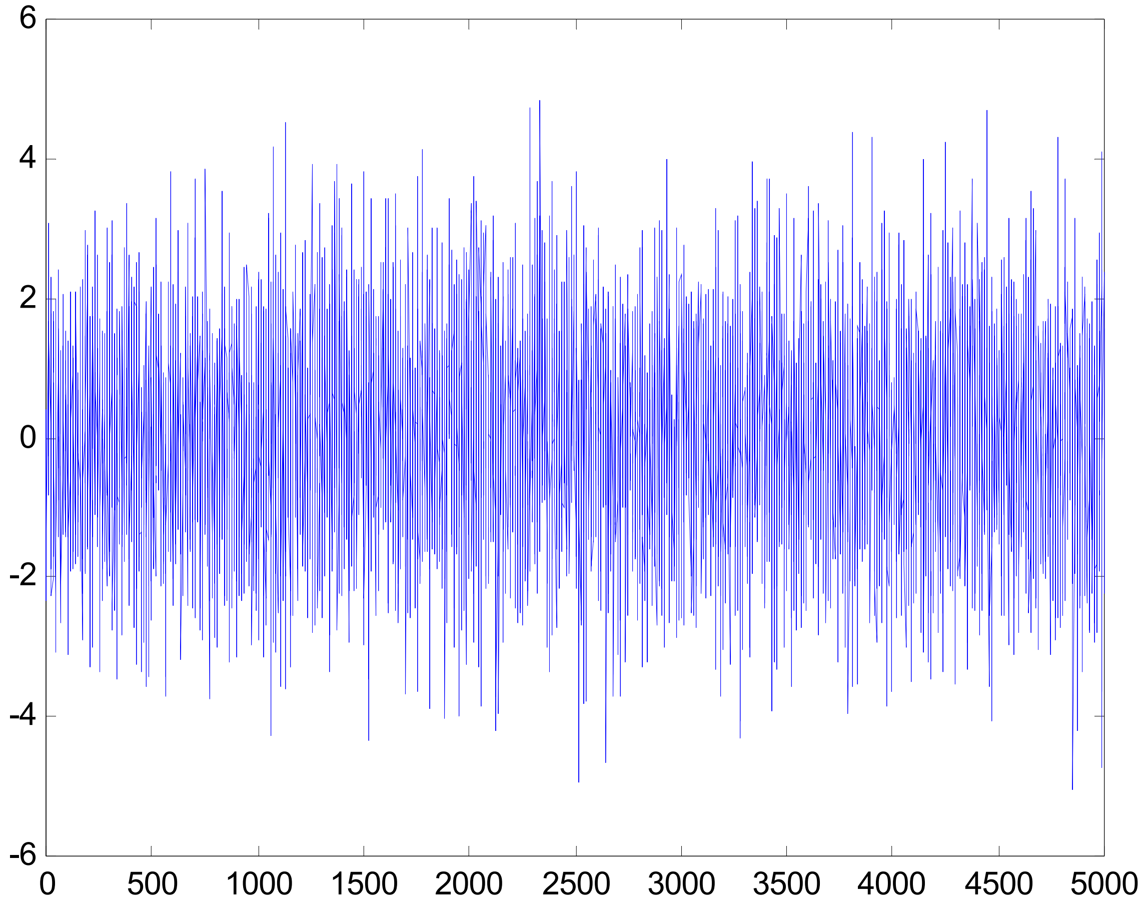
```
N=5000;                              % number of samples is 5000
power = 2;                           % signal power is 2
w = randn([1,N]).*sqrt(power);       % “randn” give Gaussian number
```

% with mean 0 and variance 1

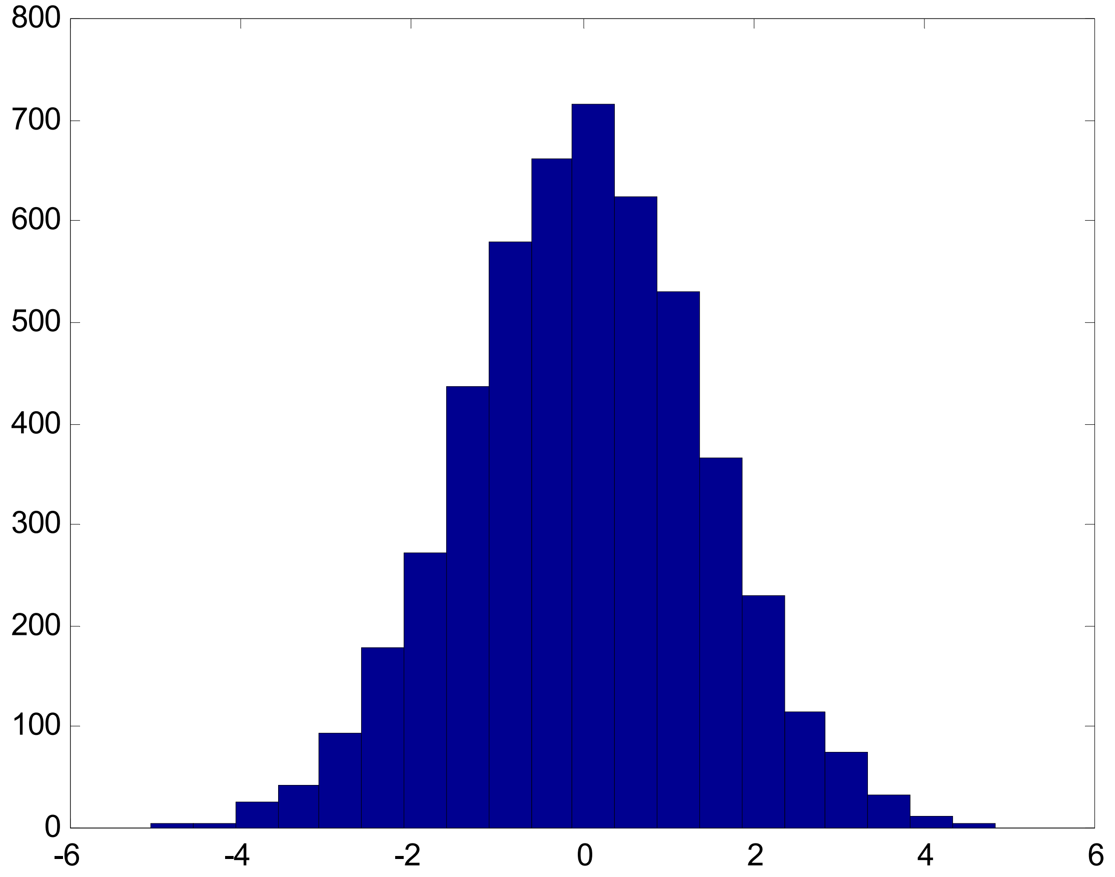
Evaluation of MATLAB Gaussian random numbers:

```
m = mean(w)                        % * “mean” computes the time average

⇒                m =  0.0123


p = mean(w.*w)                     %  compute power

⇒                p = 2.0158


y = mean((w-m).*(w-m))             %  compute variance

⇒                v = 2.0157
```

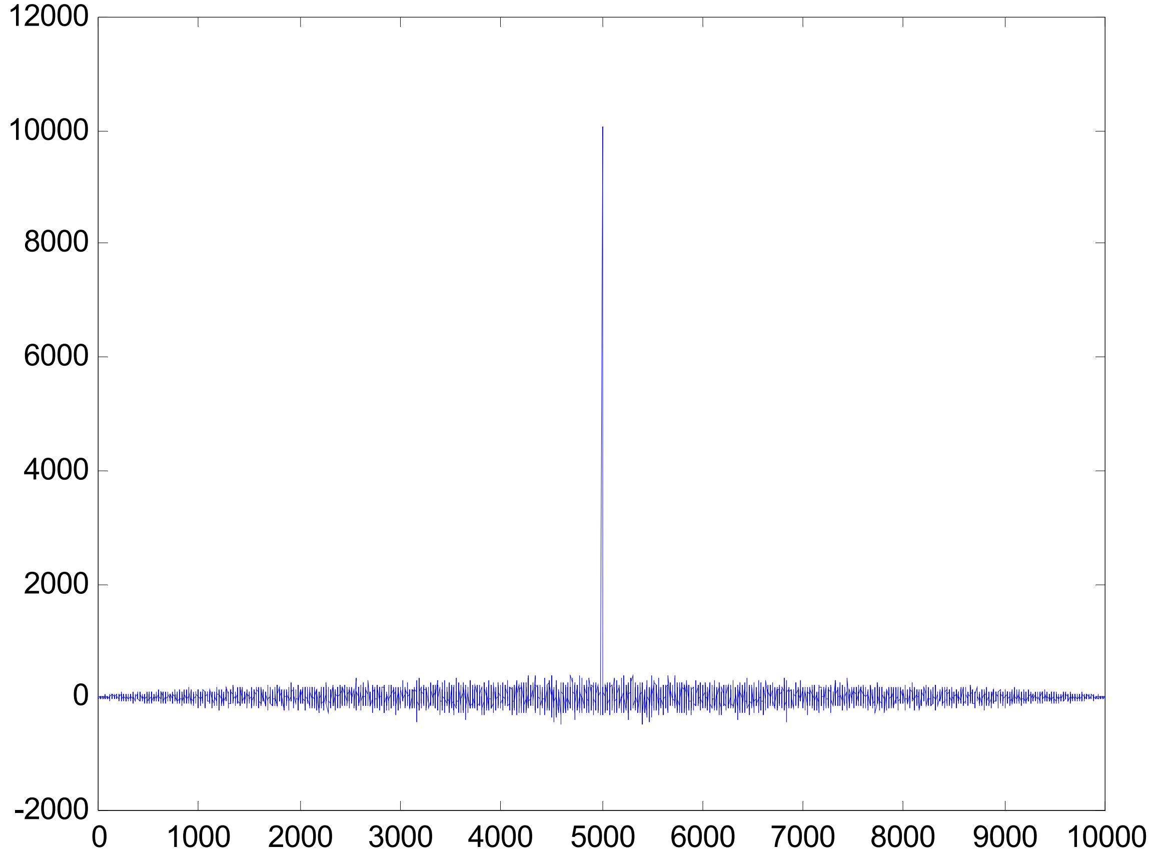
```
plot(w);                                   % plot the signal
```



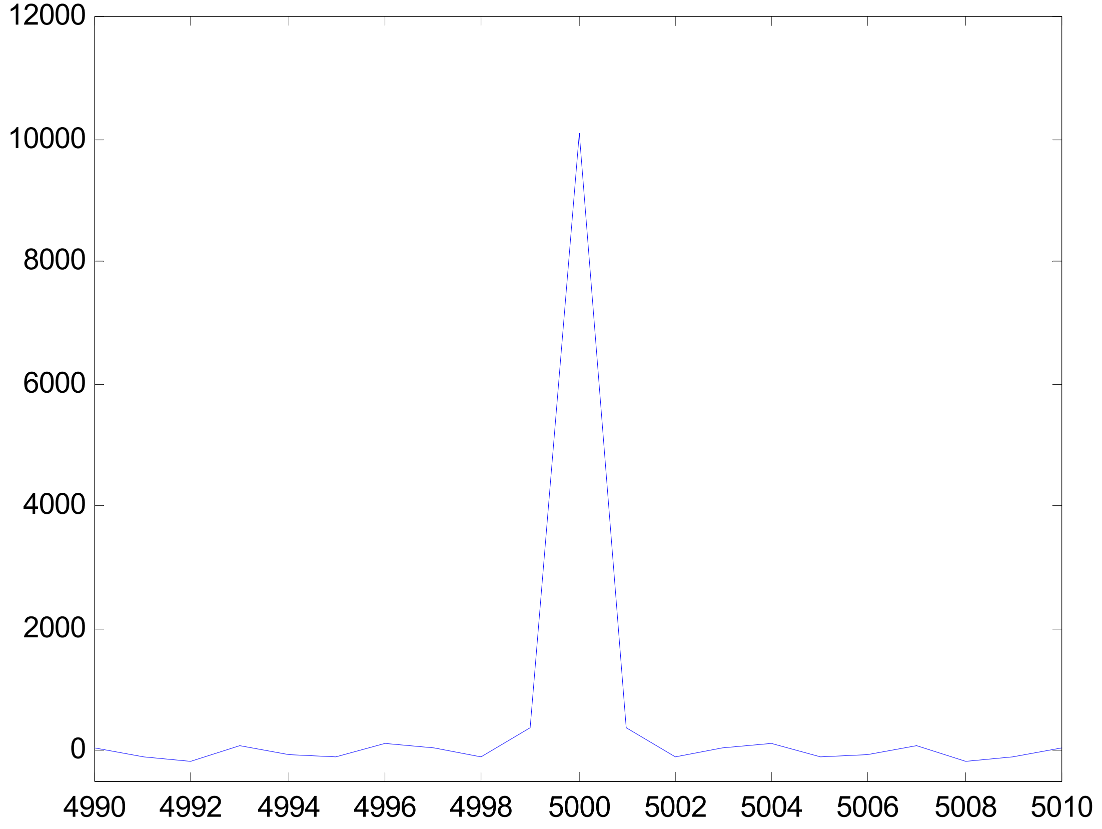
```
hist(w,20)                    % plot the histogram for w
                              % with 20 bars
```



```
a = xcorr(w);          % compute the autocorrelation
plot(a)                % plot the autocorrelation
```



axis([4990, 5010, -500, 12000])   % change the axis



The time index at 5000 corresponds to $R_{ww}(0)$

▪ Impulsive Variable

The main feature of impulsive or impulse process is that its value can be very large. A mathematical model for impulsive noise is called α-stable process, where $0 < \alpha \le 2$.

α-stable process is a generalization of Gaussian process ($\alpha = 2$) and Cauchy process ($\alpha = 1$)

The variable is more impulsive for a smaller α

A α-stable variable is generated using two independent variables: Φ which is uniform on $(-0.5\pi, 0.5\pi)$, and W which is exponentially distributed with unity mean, where W is produced from

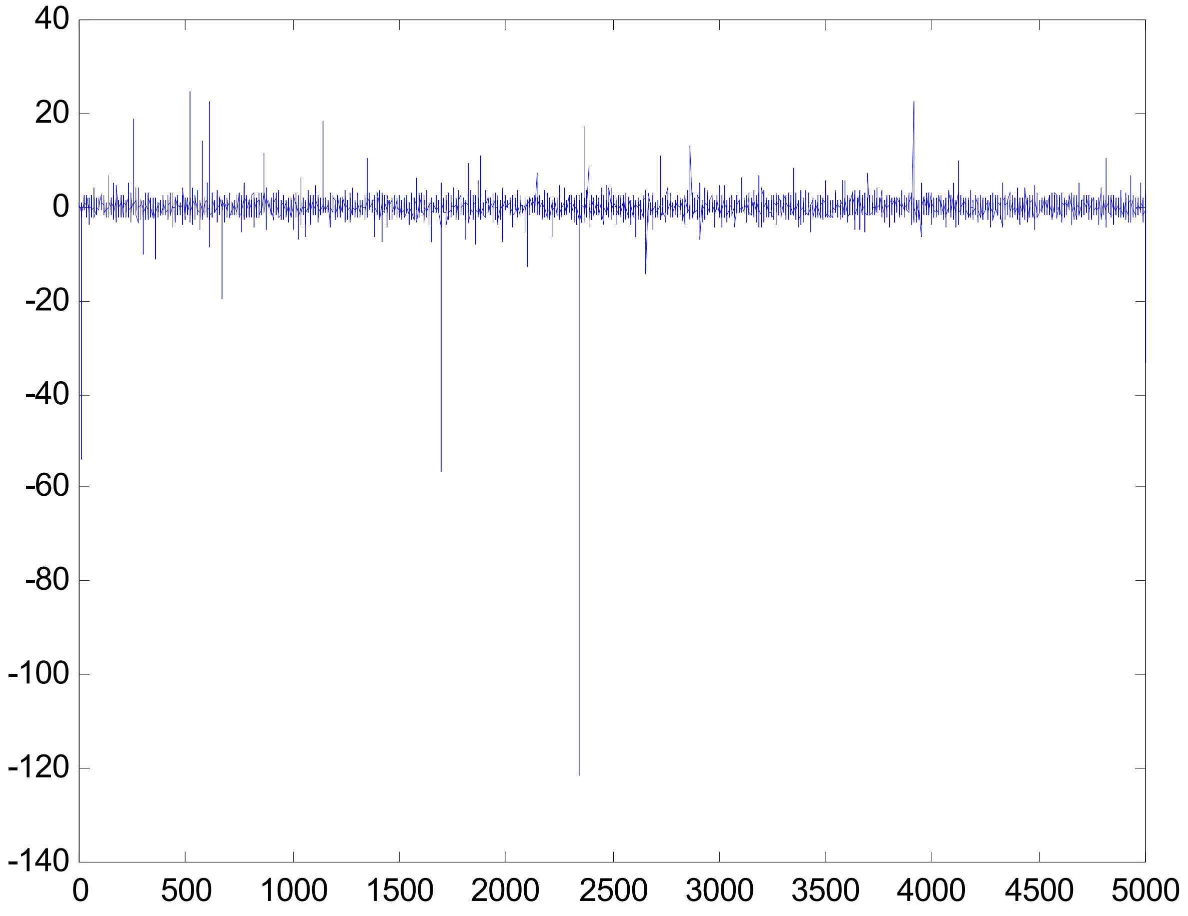
$$W = -\ln(u)$$

where u is a uniform variable distributed on [0,1]

MATLAB code for $0 < \alpha < 2$ and $\alpha \neq 1$

```
alpha = 1.8;                          % alpha is set to 1.8
beta = 0;                             % beta is a symmetric parameter
N=5000;
phi = (rand(1,N)-0.5)*pi;
w = -log(rand(1,N));
k_alpha = 1 - abs(1-alpha);
beta_a = 2*atan(beta*tan(pi*alpha/2.0))/(pi*k_alpha);
phi_0 = -0.5*pi*beta_a*k_alpha/alpha;
epsilon = 1 - alpha;
tau = -epsilon*tan(alpha*phi_0);
a = tan(0.5.*phi);
B = tan(0.5.*epsilon.*phi)./(0.5.*epsilon.*phi);
b = tan(0.5.*epsilon.*phi);
z = (cos(epsilon.*phi)-tan(alpha.*phi_0).*sin(epsilon.*phi))./(w.*cos(phi));
d = (z.^(epsilon./alpha) - 1)./epsilon;
i = (2.*(a-b).*(1+a.*b) - phi.*tau.*B.*(b.*(1-a.^2)-2.*a)).*(1+epsilon.*d)./((1-a.^2).*(1+b.^2))+tau.*d;
```

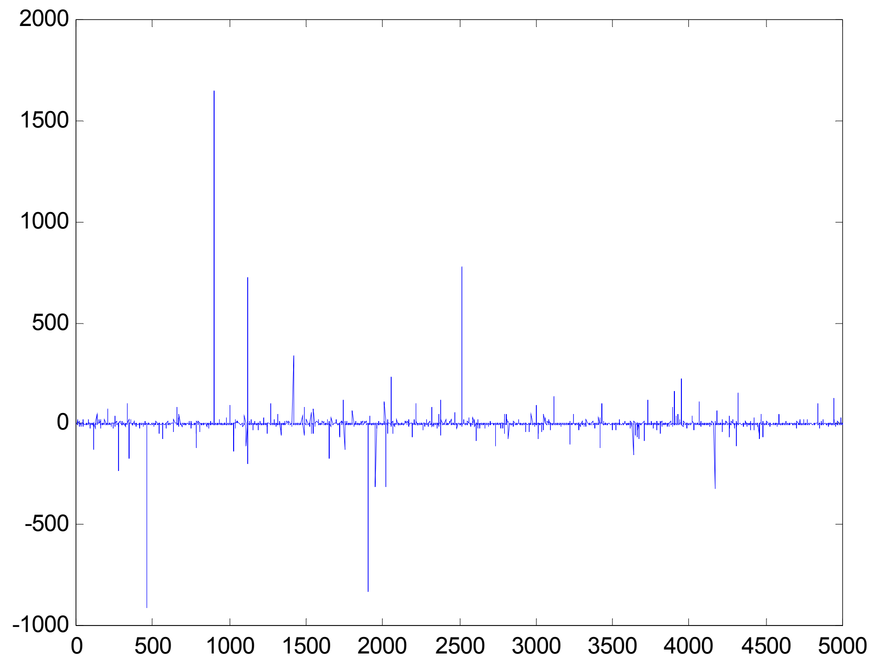
plot(i);

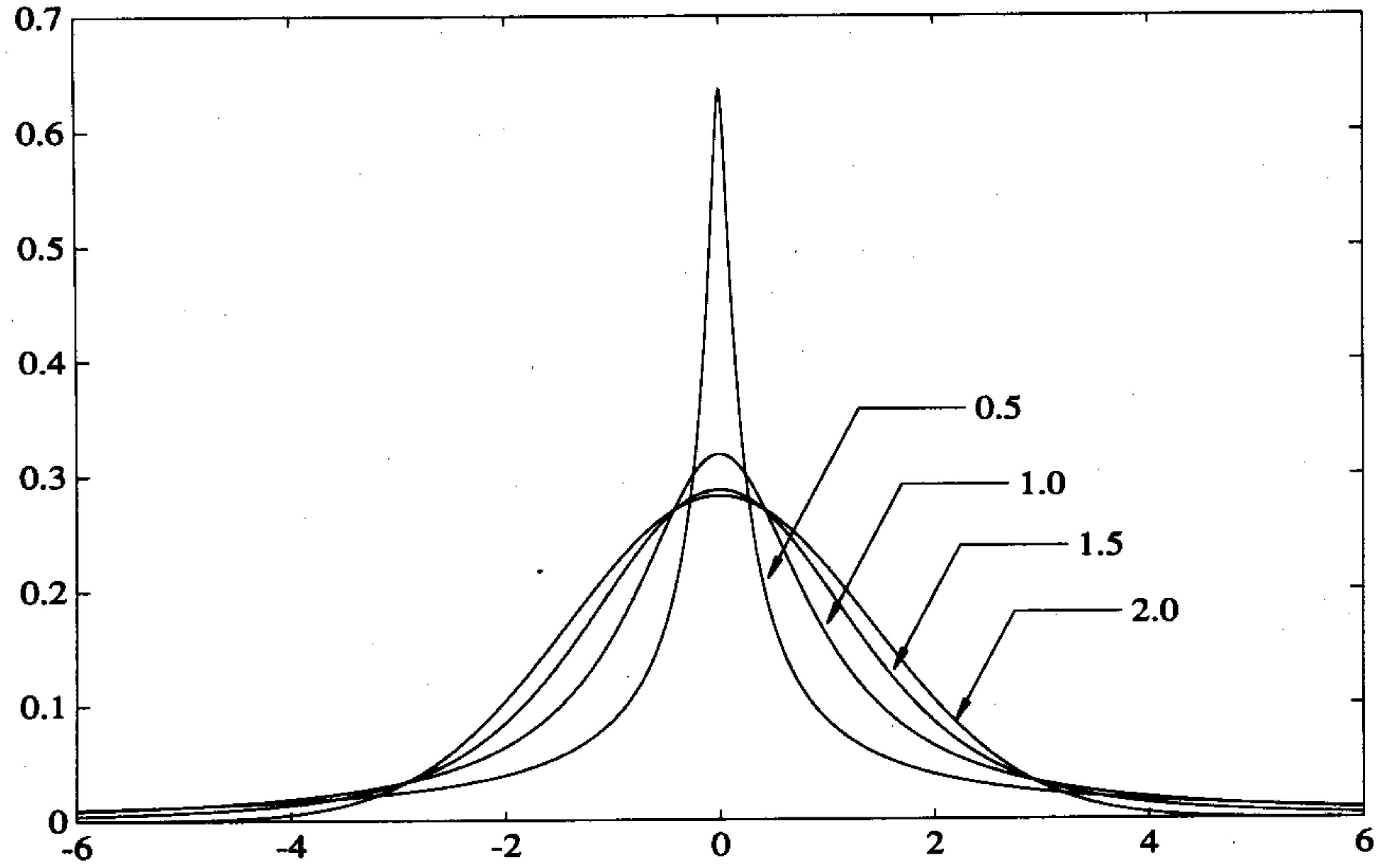


40
20
0
-20
-40
-60
-80
-100
-120
-140
0
500
1000
1500
2000
2500
3000
3500
4000
4500
5000

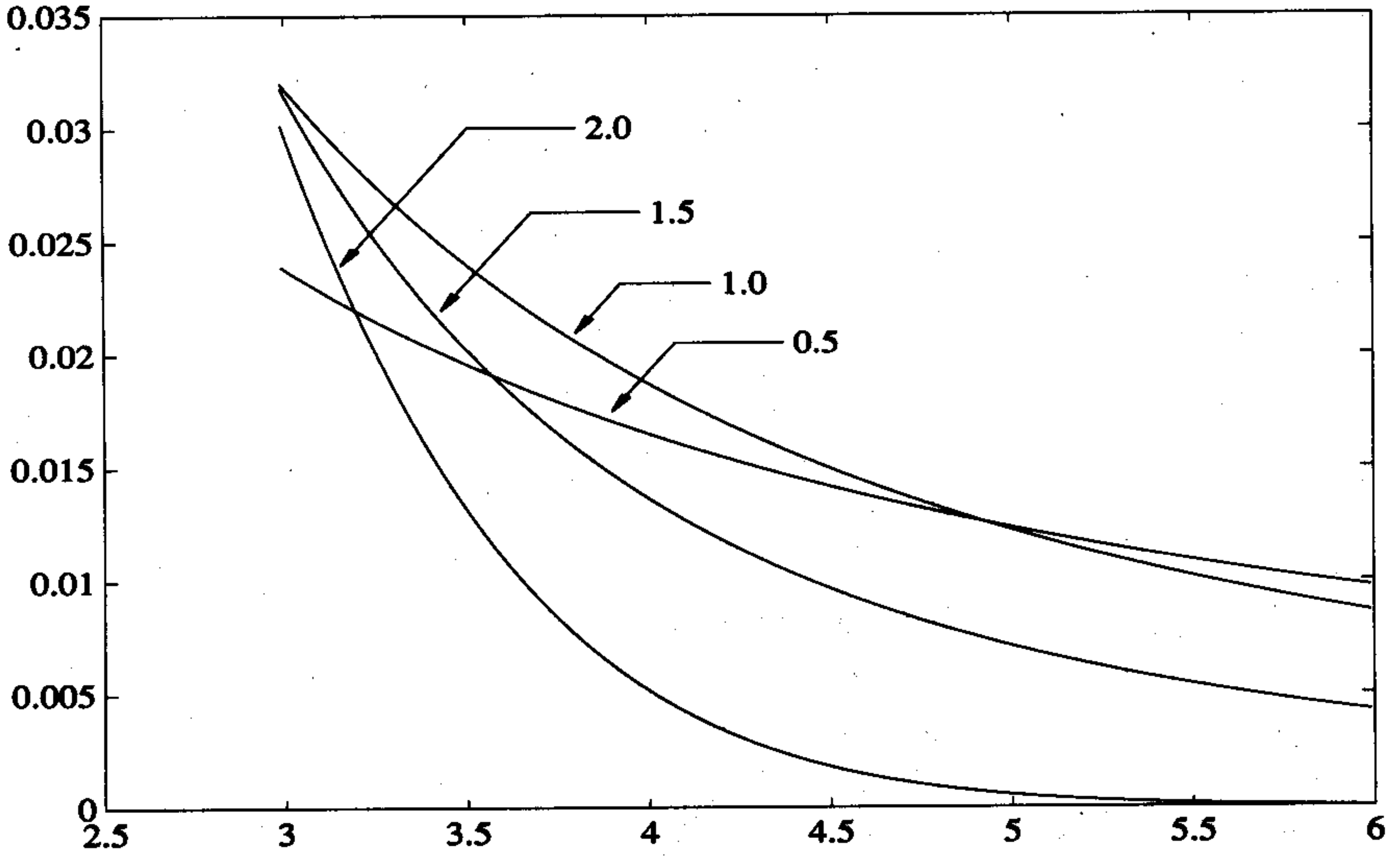
MATLAB code for $\alpha = 1$

```
N=5000;
phi = (rand(1,N)-0.5)*pi;
a = tan((0.5.*phi));
i = 2.*a./(1-a.^2);
plot(i)
```





PDF for different α



The “impulsiveness” is due to the heavier tails, i.e., PDF go to zero slowly

▪ AR, MA and ARMA Processes

MA process is generated from

$$x(n) = b_0 w(n) + b_1 w(n-1) + \cdots + b_N w(n-N)$$

where $\{ w(n) \}$ is a white noise sequence. Only the transient signal is needed to remove.

e.g., for a second-order MA process

$$x(n) = b_0 w(n) + b_1 w(n-1)$$

$\because \; w(n) = 0, \; n < 0$

$\Rightarrow$
$$\begin{aligned} x(0) &= b_0 w(0) + b_1 w(-1) = b_0 w(0) \\ x(1) &= b_0 w(1) + b_1 w(0) \\ x(2) &= b_0 w(2) + b_1 w(1) \\ &\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \end{aligned}$$

The transient signal is $x(0)$. We should choose $\{x(1), x(2), \cdots\}$

MATLAB code for generating 50 samples of MA process with $b_0 = 1, b_1 = 2$:

```
b0=1;
b1=2;
N=50;
w=randn(1,N+1);                      % generate N+1 white noise samples
for n=1:N
     x(n) = b0*w(n+1)+b1*w(n);       % shift “w” by one sample
end
```

Alternatively, we can use the convolution function in MATLAB:

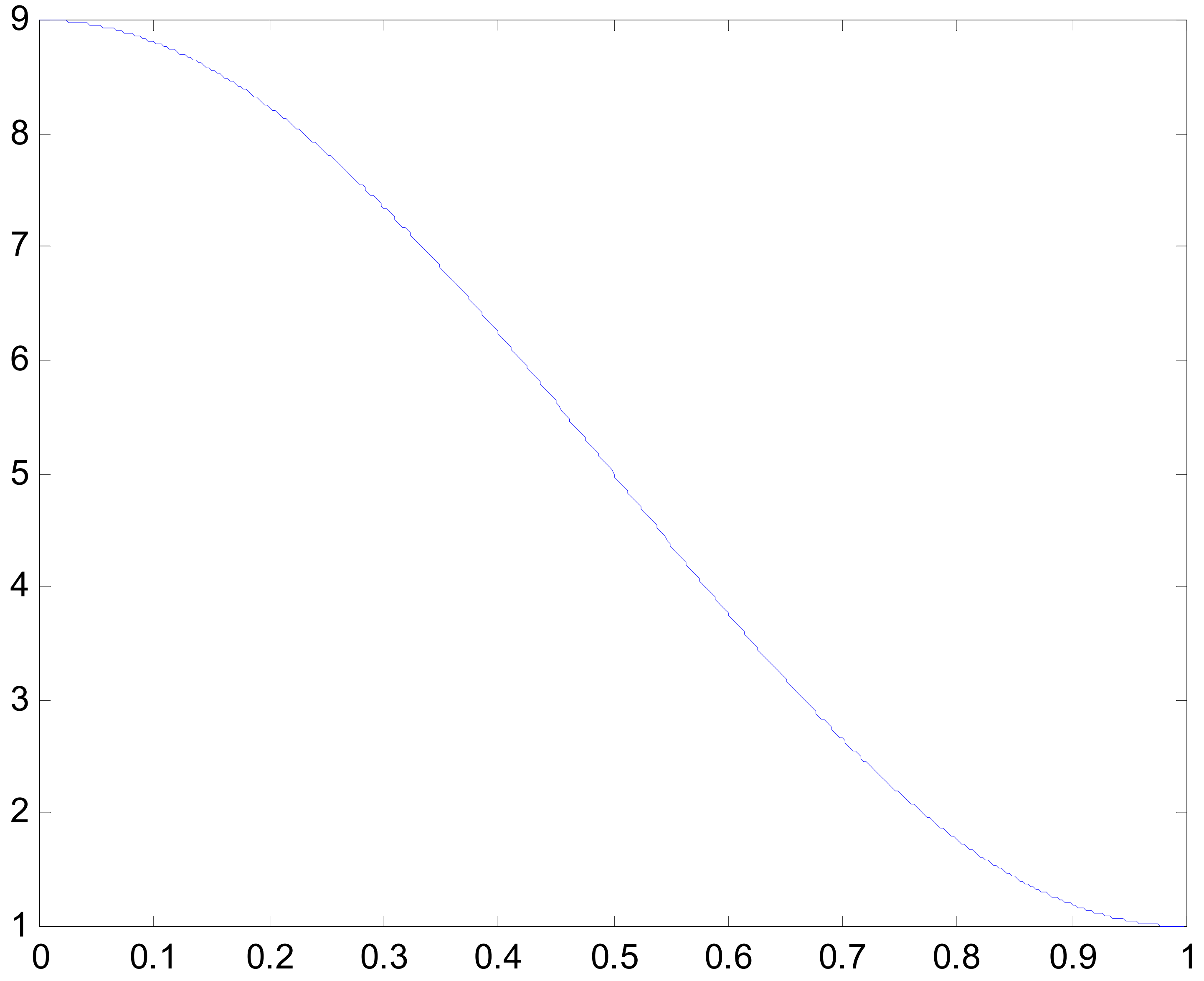
```
b0=1;
b1=2;
N=50;
w=randn(1,N+1);                      % generate N+1 white noise samples
b= [b0 b1];                          % “b” is an vector
y=conv(b,w);                         % signal length is “N+1”+”2”-1
x=y(2:N+1);                          % remove the transient signals
```

From (1.44), the PSD for MA process is

$$\Phi_{xx}(\omega) = \left|1 + 2e^{-j\omega}\right|^2 \cdot \sigma_w^2 = \left|1 + 2e^{-j\omega}\right|^2$$

It can be plotted using the freqz command in MATLAB:

```
b0=1;
b1=2;
b= [b0 b1];
a=1;
[H,W] = freqz(b,a);                    % “H” is complex frequency response
PSD = abs(H.*H);
plot(W/pi,PSD);
```


9
8
7
6
5
4
3
2
1
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

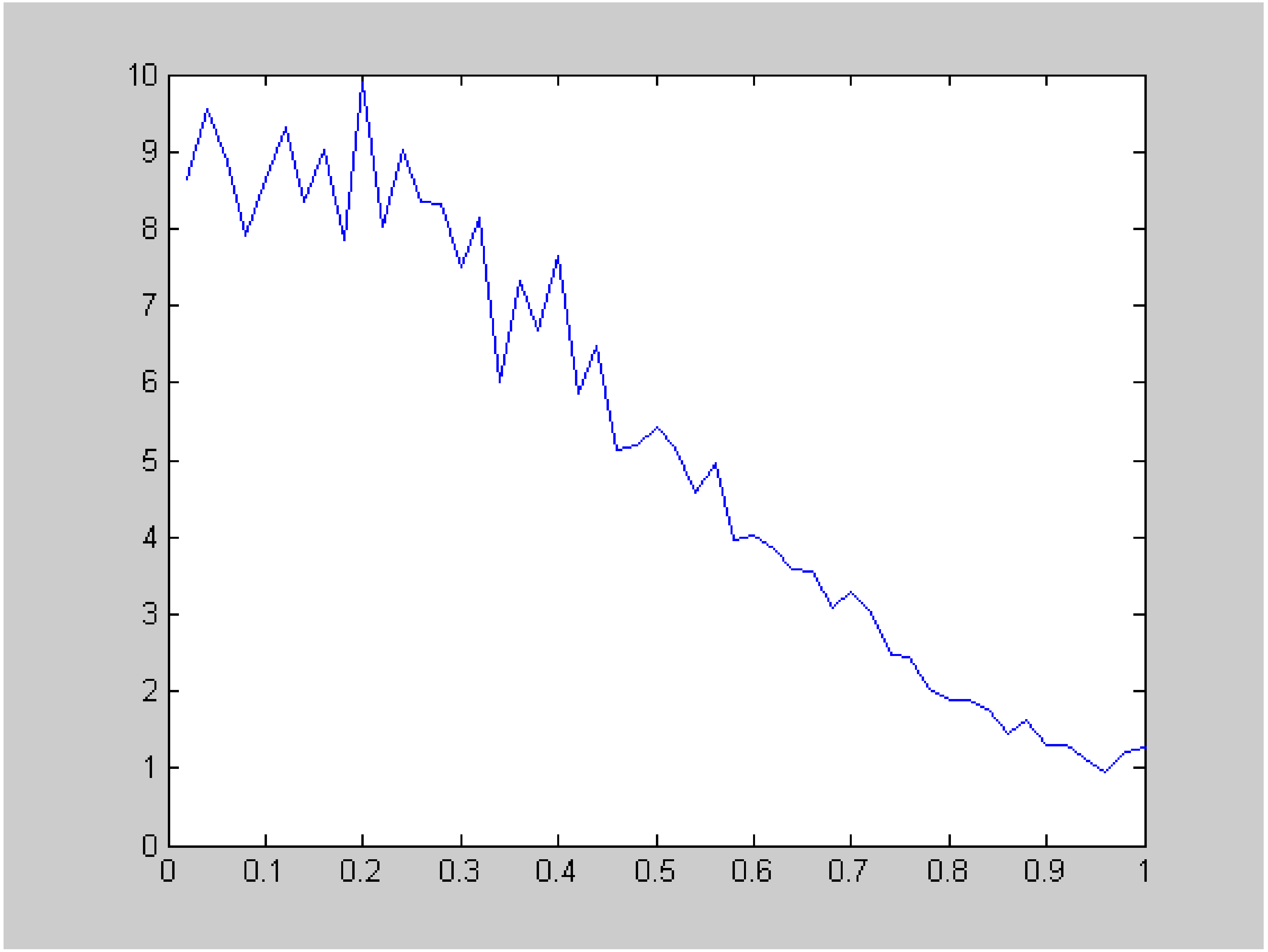
To evaluate the MA process generated by MATLAB, we use (1.38):

$$\Phi_{xx}(\omega) = \lim_{N\to\infty} E\left\{ \frac{1}{N} \left| \sum_{n=0}^{N-1} x(n) e^{-j\omega n} \right|^2 \right\}$$

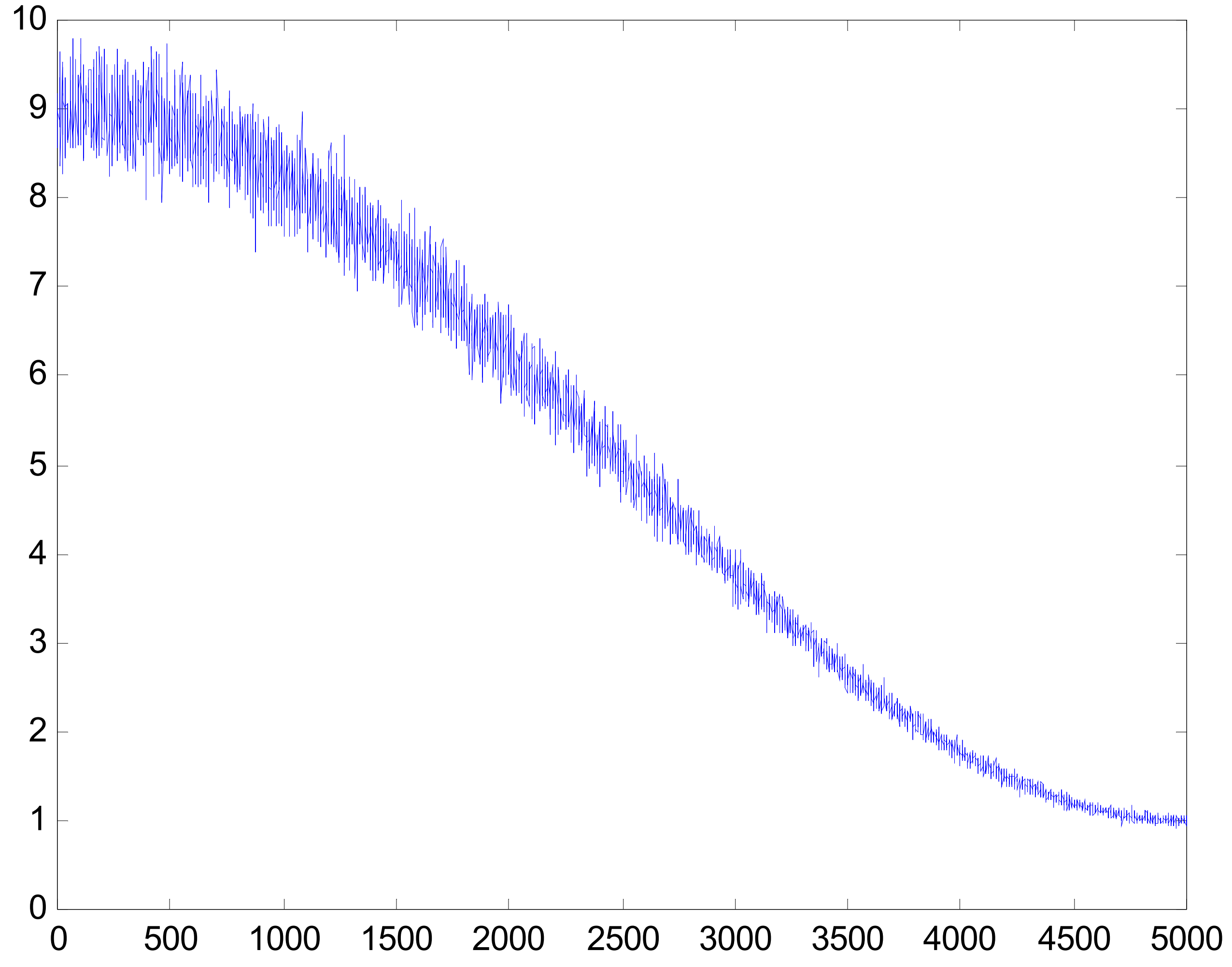
- $N \to \infty \quad \Rightarrow \quad N \to 100$
- $E\{\} \quad \Rightarrow$ average of 100 independent simulations

MATLAB code:

```
N=100;
b= [1 2];                          % “b” is a vector
for m=1:100                        % perform 100 independent runs
     w=randn(1,N+1);               % generate N+1 white noise samples
     y=conv(b,w);                  % signal length is “N+1”+”2”-1
     x=y(2:N+1);                   % remove the transient signals
     p(m,:) = abs(fft(x).*fft(x));
end
psd = mean(p)./100;
index = 1/50:1/50:2;
plot(index,psd);
axis([0, 1, 0 10]);
```


10
9
8
7
6
5
4
3
2
1
0
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

- $N \to \infty \quad \Rightarrow \quad N \to 10000$
- $E\{\}$ $\Rightarrow$ average of 10000 independent simulations



Transient signals are also needed to remove in AR & ARMA processes because of non-stationarity due to the poles:

$$x(n) = a_1 x(n-1) + a_2 x(n-2) + \cdots + a_M x(n-M) + w(n)$$

$$x(n) = a_1 x(n-1) + a_2 x(n-2) + \cdots + a_M x(n-M)$$
$$+ b_0 w(n) + b_1 w(n-1) + \cdots + b_N w(n-N)$$

e.g., for a first–order AR process: $x(n) = ax(n-1) + w(n)$

$$R_{xx}(n, n+m) = a^m \left(\frac{1 - a^{2(n+1)}}{1 - a^2} \right) \sigma_w^2$$

$\Rightarrow$ nonstationary because $R_{xx}(n, n+m)$ depends on n

$\Rightarrow$ for sufficiently large n, say, $\left| a^{2(n+1)} \right| << 1$, we can consider it stationary.

$\Rightarrow$ since a is the pole, extension to general AR and ARMA processes:

$$\left| p_i^{2(n+1)} \right| << 1, \qquad \text{for all poles } \{p_i\}$$

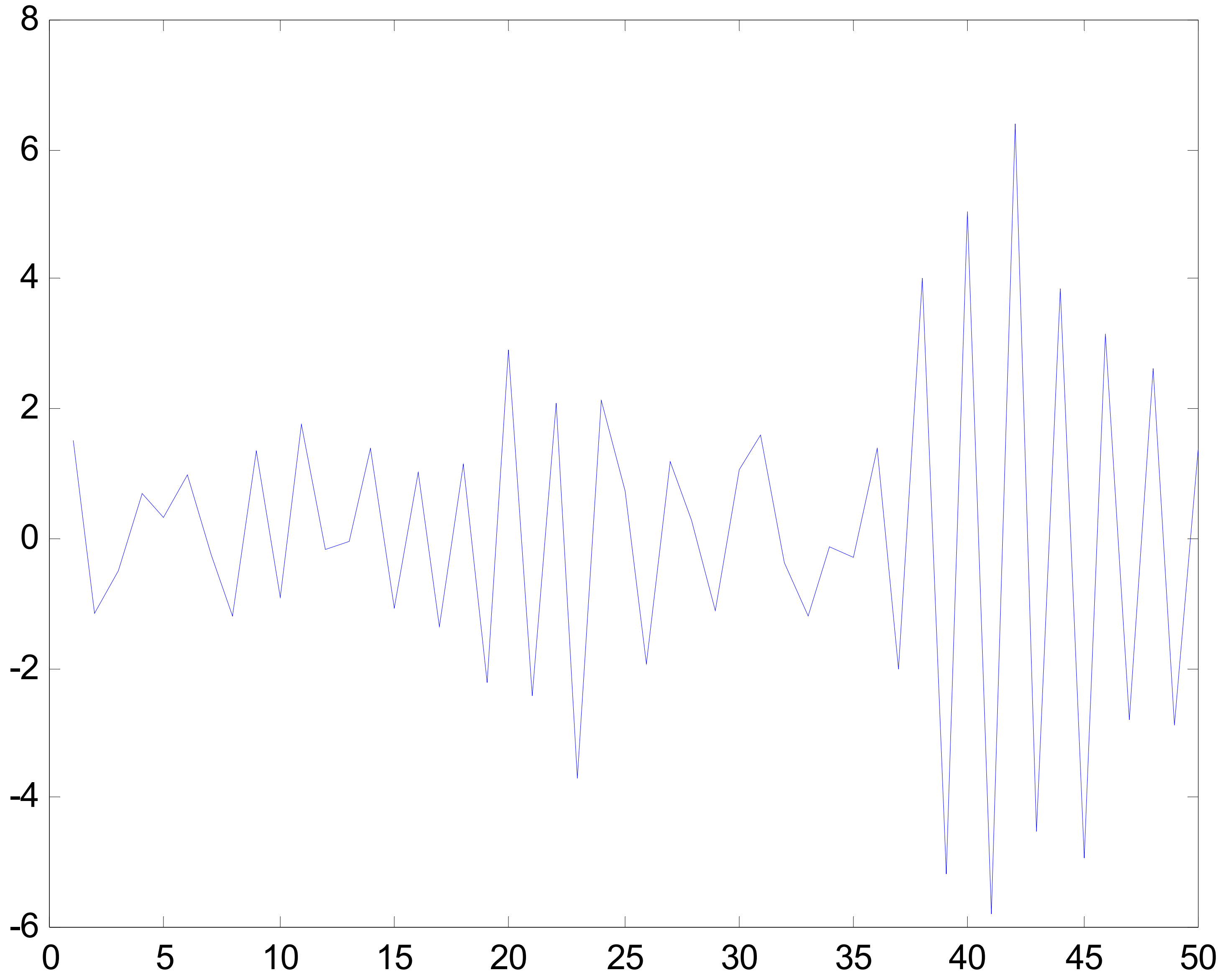
Suppose $\left| a^{2(n+1)} \right| \le 0.0001$ is required and the AR parameter is $a = -0.9$. The required n is calculated as

$$\left| (-0.9)^{2(n+1)} \right| = 0.9^{2(n+1)} \le 0.0001$$

$$\Rightarrow n \ge 43$$

MATLAB code for generating 50 samples of the AR process:

```
M = 43;
N = 50;
a = -0.9;
y(1) = 0;
for n=2:M+N
     y(n) = a*y(n-1)+randn;
end
x=y(M+1:M+N);
plot(x);
```


8
6
4
2
0
-2
-4
-6
0
5
10
15
20
25
30
35
40
45
50

Digital Filtering

Given an input signal $x(n)$ and the transfer function $H(z)$, it is easy to generate the corresponding output signal, say, $y(n)$

For FIR system, we can follow the MA process, while for IIR system, we can follow the ARMA process. The transient signals can be removed if necessary as in the MA, AR and/or ARMA processes.

Given $H(z)$, the impulse response can be computed via inverse DTFT:

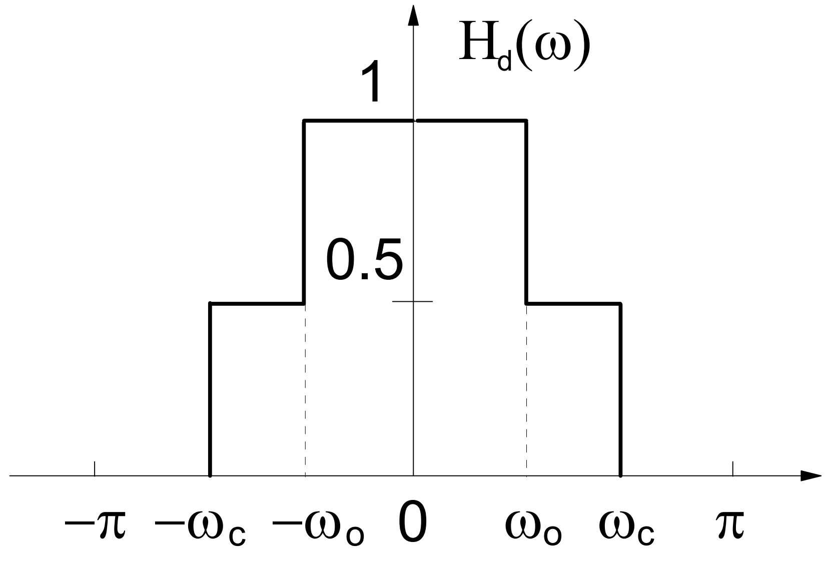
$$h(n) = \frac{1}{2\pi}\int_{-\pi}^{\pi} H(\omega)e^{j\omega n}d\omega$$

Frequency spectrum for $H(z) \Rightarrow$ impulse response $\{h(n)\}$

$$\Rightarrow \qquad y(n) = h(n) \otimes x(n) = \sum_{k=-\infty}^{\infty} h(n-k)x(k) = \sum_{k=-\infty}^{\infty} x(n-k)h(k)$$

<u>Example 2.2</u>

Compute the impulse response for $H_d(z)$ with the following DTFT spectrum, and $\omega_o = 0.2\pi$ and $\omega_c = 0.4\pi$.



$$h_d(n) = \frac{1}{2\pi}\int_{-\pi}^{\pi} H_d(\omega) e^{j\omega n} d\omega$$

$$= \frac{1}{2\pi}\int_{-\omega_c}^{\omega_c} 0.5 \cdot e^{j\omega n} d\omega + \frac{1}{2\pi}\int_{-\omega_o}^{\omega_o} 0.5 \cdot e^{j\omega n} d\omega = \frac{\sin(\omega_c n)}{2\pi n} + \frac{\sin(\omega_o n)}{2\pi n}$$

$$\Rightarrow \qquad h_d(n) = \frac{\sin(0.2\pi n) + \sin(0.4\pi n)}{2\pi n}, \qquad n = \cdots, -1, 0, 1, \cdots$$

Note that $h_d(0)$ can be obtained by using L'Hospital's rule or:

$$h_d(0) = \frac{1}{2\pi}\int_{-\pi}^{\pi} H_d(\omega)e^{j\omega\cdot 0}d\omega = \frac{1}{2\pi}\int_{-\pi}^{\pi} H_d(\omega)d\omega$$

$$= \frac{1}{2\pi}\int_{-\omega_c}^{\omega_c} 0.5d\omega + \frac{1}{2\pi}\int_{-\omega_o}^{\omega_o} 0.5d\omega = \frac{\omega_c + \omega_o}{2\pi} = 0.3$$

Combining the results:

$$h_d(n) = \begin{cases} 0.3, & n = 0 \\ \dfrac{\sin(0.2\pi n) + \sin(0.4\pi n)}{2\pi n}, & \text{otherwise} \end{cases}$$

$$\Rightarrow \qquad y(n) = h_d(n) \otimes x(n) = \sum_{k=-\infty}^{\infty} x(n-k)h_d(k) \approx \sum_{k=-M}^{M} x(n-k)h_d(k)$$

Example 2.3

Compute the impulse response of a time-shift function which time-shifts a signal by a non-integer delay D.

$$y(n) = x(n-D) \Rightarrow Y(\omega) = e^{-j\omega D} \cdot X(\omega), \qquad H(\omega) = \exp(-j\omega D)$$

$$h(n) = \frac{1}{2\pi}\int_{-\pi}^{\pi} H(\omega) e^{j\omega n} d\omega = \frac{1}{2\pi}\int_{-\pi}^{\pi} e^{-j\omega D} \cdot e^{j\omega n} d\omega = \frac{1}{2\pi}\int_{-\pi}^{\pi} e^{j\omega(n-D)} d\omega$$

$$= \operatorname{sinc}(n-D)$$

where

$$\operatorname{sinc}(x) = \frac{\sin(\pi x)}{\pi x}$$

$$y(n) = x(n) \otimes \operatorname{sinc}(n-D) = \sum_{k=-\infty}^{\infty} x(n-k)\operatorname{sinc}(k-D)$$

$\Rightarrow$

$$\approx \sum_{k=-M}^{M} x(n-k)\operatorname{sinc}(k-D)$$

Questions for Discussion

1. Observe that the following signal:

$$y(n) = \sum_{k=-10}^{10} x(n-k)\,\mathrm{sinc}(k-D) \approx x(n-D)$$

which depends on future data $\{x(n+1), x(n+2), \cdots, x(n+10)\}$.

This is referred to as a non-causal system. How to generate the output of the non-causal system in practice?

2. The spectrum for the Hilbert transform is

$$H(\omega) = \begin{cases} -j, & 0 < \omega \le \pi \\ j, & -\pi \le \omega < 0 \end{cases}$$

Use a FIR filter with 15 coefficients to perform the Hilbert transform of a discrete-time signal $x[n]$. Let the resultant signal be $y[n]$.