Random Processes


```
%pylab inline
#%matplotlib qt
import sk_dsp_comm.sigsys as ss
import sk_dsp_comm.digitalcom as dc
import scipy.signal as signal
from IPython.display import Audio, display
from IPython.display import Image, SVG
```

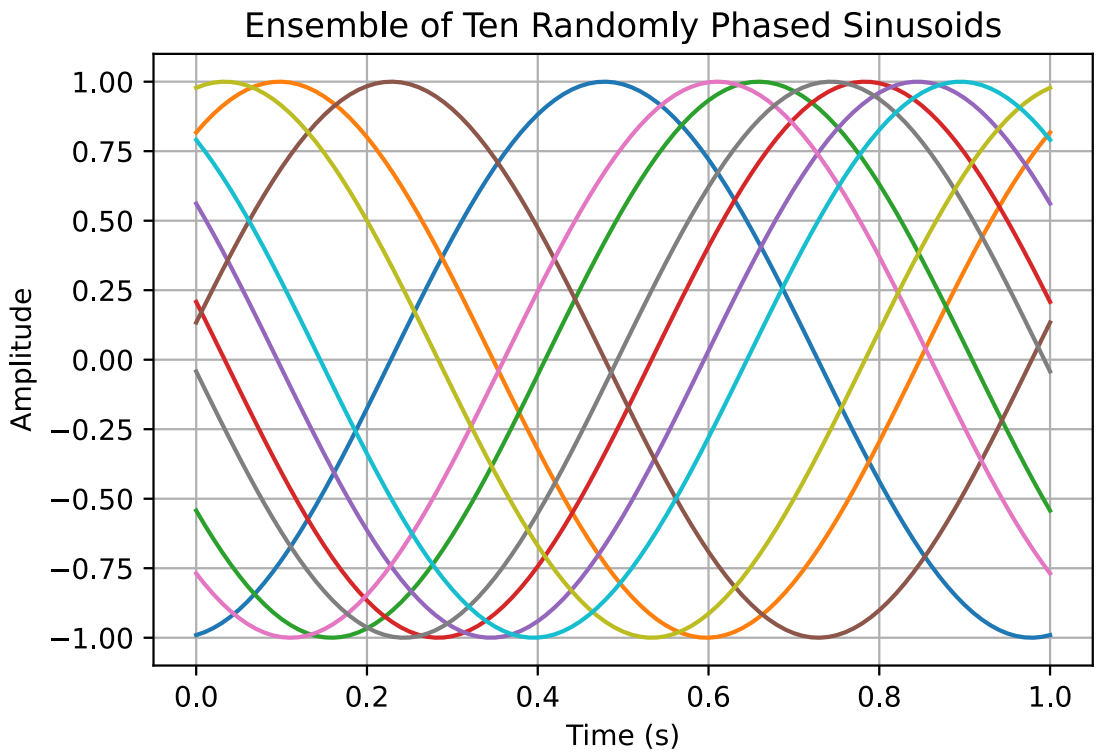
```
Populating the interactive namespace from numpy and matplotlib
```

```
pylab.rcParams['savefig.dpi'] = 100 # default 72
#pylab.rcParams['figure.figsize'] = (6.0, 4.0) # default (6,4)
#%config InlineBackend.figure_formats=['png'] # default for inline viewing
%config InlineBackend.figure_formats=['svg'] # SVG inline viewing
#%config InlineBackend.figure_formats=['pdf'] # render pdf figs for LaTeX
```

A Randomly Phased Sinusoid

Write some code to create a randomaly phased sinusoid, similar to notes Example 4.3.

```
A = 1
fo = 1
for k in range(10):
    phi = 2*pi*rand(1)
    t = arange(0,1+.01,0.01)
    x = A*cos(2*pi*fo*t + phi)
    #plot(t,x,'b')
    plot(t,x)
grid()
ylim([-1.1,1.1])
xlabel('Time (s)')
ylabel('Amplitude')
title('Ensemble of Ten Randomly Phased Sinusoids');
```



Getting Started with Computer Exercise 7.1

In this problem you generate an ensemble of randomly phased sinusoids. It is convenient to place each waveform in a row so the ensemble is a matrix of n rows by m columns. Sampling across a row allows a time average to be performed while sampling down a column forms a ensemble of statistical average (of course using sample statistics).

You can form the data set using `for` loops but you can also Numpy 2D `ndarrays` and the matrix multiply operator `@`, which comes with Python 3.5+. How to do it without any loops? Use the matrix *outer product*, which forms a $n \times m$ matrix when multiplying vectors, i.e., $(n \times 1) \cdot (1 \times m) = n \times m$.

Consider the argument of the cosine function: $2\pi \cdot f_0 \cdot t + \theta$ and form the $n \times m$ matrix $\mathbf{X}$. Both t and θ need to be matrices, but in the case of t all of the columns entries are identical and in the case of θ all of the rows entries are identical.

```
# A starting point for theta
theta = rand(3,1) @ ones((1,5))
theta
```

```
array([[ 0.17746726,  0.17746726,  0.17746726,  0.17746726,  0.17746726],
       [ 0.63376412,  0.63376412,  0.63376412,  0.63376412,  0.63376412],
       [ 0.72742752,  0.72742752,  0.72742752,  0.72742752,  0.72742752]])
```

```
# A starting point for t
t = ones((3,1)) @ array([arange(0,5)])
t
```

```
array([[ 0.,  1.,  2.,  3.,  4.],
       [ 0.,  1.,  2.,  3.,  4.],
       [ 0.,  1.,  2.,  3.,  4.]])
```

```
X = cos(t + theta)
X.shape
```

```
(3, 5)
```

```
X
```

```
array([[ 0.98429397,  0.38326539, -0.57013563, -0.99935658, -0.5097737 ],
       [ 0.80580418, -0.0629262 , -0.87380251, -0.88130883, -0.07854387],
       [ 0.74688745, -0.15599153, -0.91545261, -0.83325078,  0.01503797]])
```

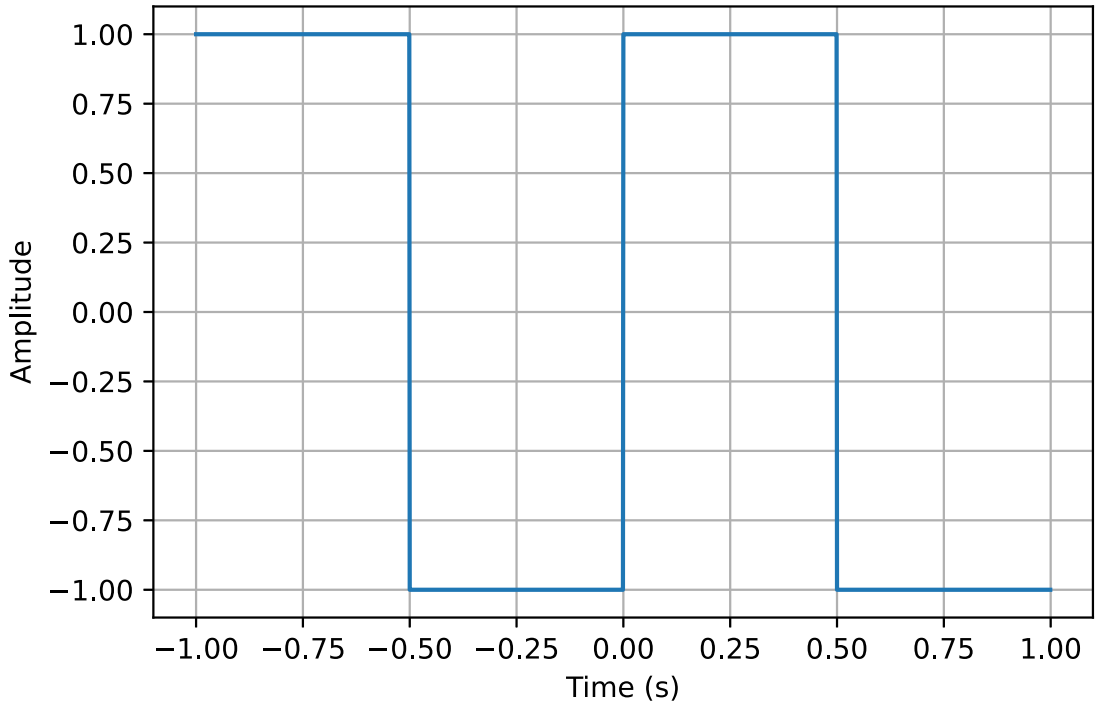
More to say later.

Problem 7.5

Waveform Creation

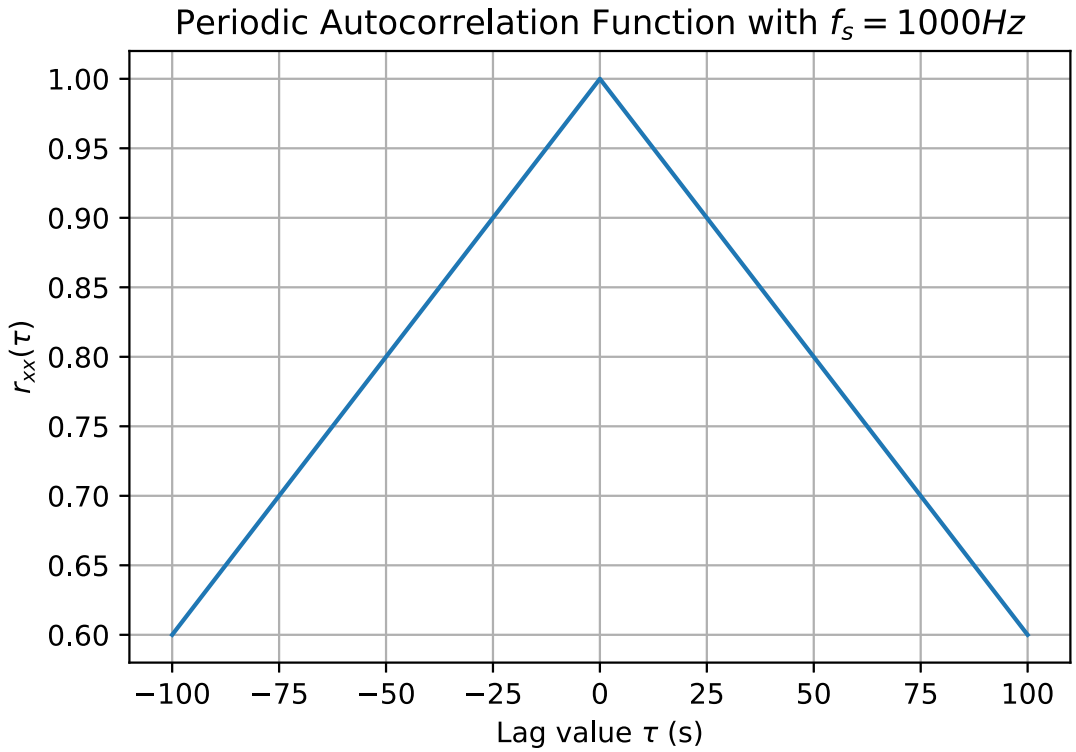
Generating a sampled version of the squarewave. Here I set $f_s = 1000$ Hz.

```
t = arange(-1,1,0.001)
s = sign(sin(2*pi*t))
plot(t,s)
ylim([-1.1,1.1])
grid()
xlabel('Time (s)')
ylabel('Amplitude');
```




```
rs, lags = dc.xcorr(s,s,100)
plot(lags,rs.real)
grid()
xlabel(r'Lag value $\tau$ (s)')
ylabel(r'$r_{xx}(\tau)$')
title(r'Periodic Autocorrelation Function with $f_s =1000 Hz$')
```

```
Text(0.5, 1.0, 'Periodic Autocorrelation Function with $f_s =1000 Hz$')
```

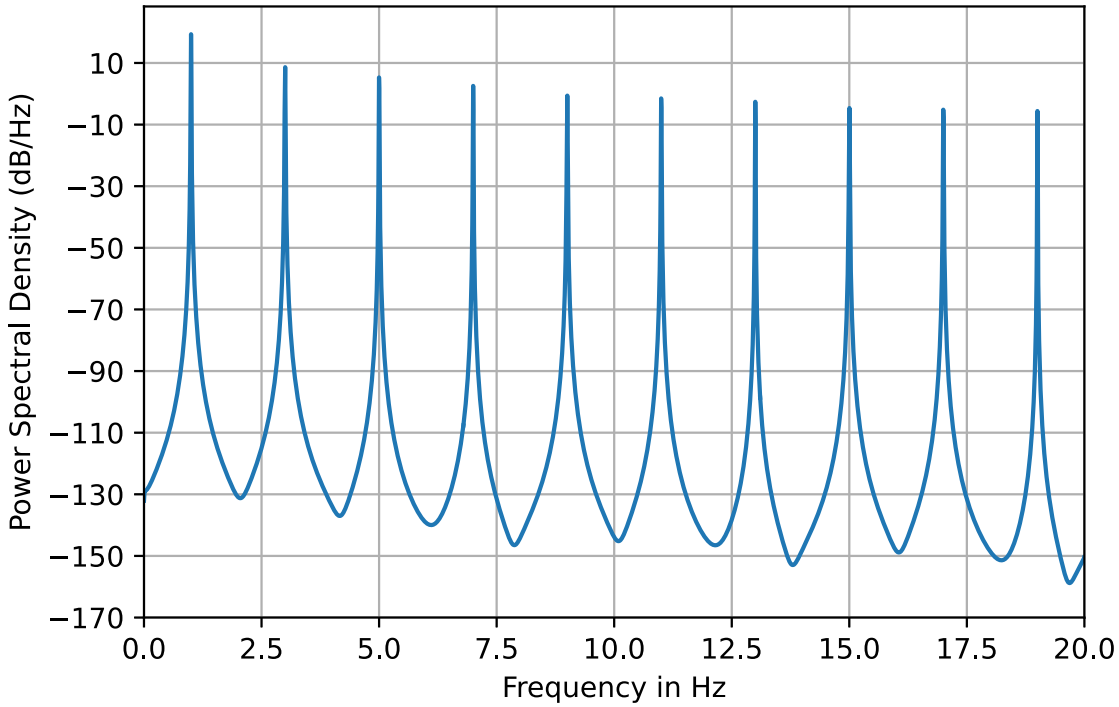


When using `dc.xcorr` the quantity $R_{xy}[k]$ is normalized by $\sqrt{E_1 E_2}$ where $E_1 = \sum |x_1[n]|^2$ and $E_2 = \sum |x_2[n]|^2$. To emphasize that fact I have labeld the axis as $r_{xx}(\tau) = R_{xx}(\tau)/\sqrt{E_1 E_2}$.

Squarewave Power Spectral Density

```
t = arange(-100,100,0.01)
s = sign(sin(2*pi*t))
psd(s,2**14,100);
xlim([0,20]);
xlabel('Frequency in Hz')
```

```
Text(0.5, 0, 'Frequency in Hz')
```



No surprise that the power sepctrum of the randomly phased squarewave contains odd harmonics. He you see them starting at the fundamental of 1 Hz and the 3rd, 5th, 7th, etc. harmonics at 3, 5, 7,etc. Hz.

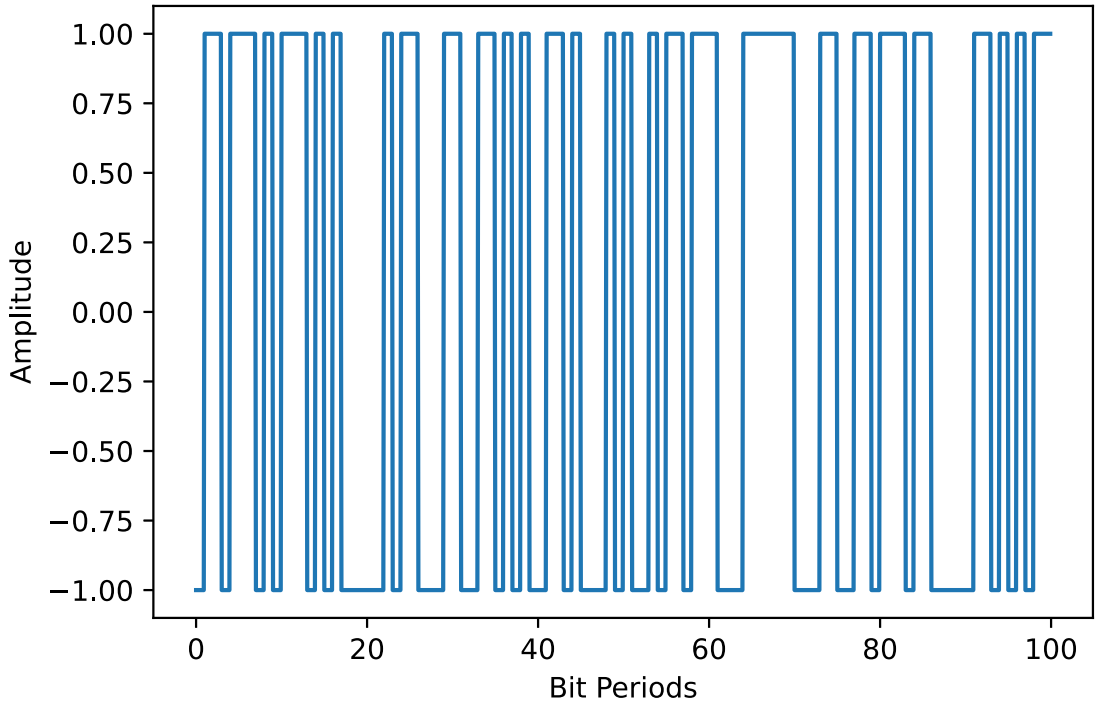
Random Pulse Train

In the digital comm code module (`digitalcom.py`), I provide a time averaged auto/cross correlation function: `rx, lags = dc.xcorr(x1,x2,Nlags)` . A random pulse train, similar to that described in the notes, can be found in the function `x,b,data = dc.NRZ_bits(N_bits, Ns, pulse_shape)` .

The random pulse train below is generated using 10 samples per bit.

```
x,b,data = dc.nrz_bits(100, 10, 'rect')
t = arange(len(x))*0.1
plot(t,x)
ylim([-1.1,1.1])
xlabel(r'Bit Periods')
ylabel(r'Amplitude')
```

```
Text(0, 0.5, 'Amplitude')
```



• Estimating the Power Spectrum

The PSD function is actually part of the graphics package `matplotlib`.

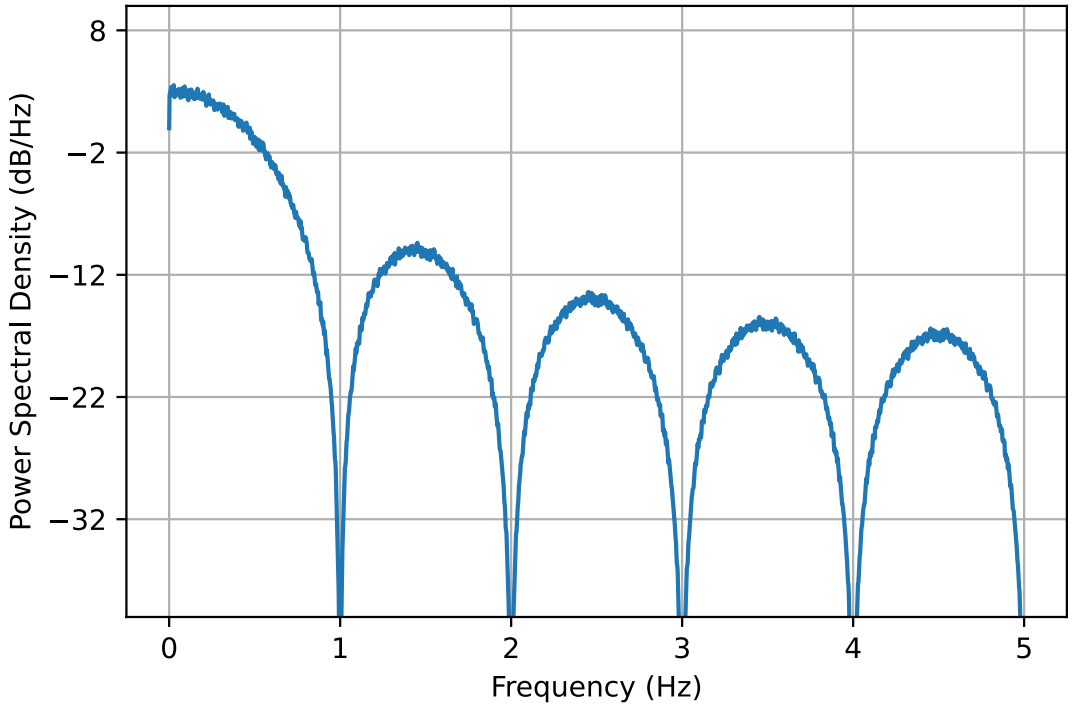
Call signature::

```
psd(x, NFFT=256, Fs=2, Fc=0, detrend=mlab.detrend_none,
    window=mlab.window_hanning, noverlap=0, pad_to=None,
    sides='default', scale_by_freq=None, **kwargs)
```

The power spectral density by Welch's average periodogram method. The vector *x* is divided into *NFFT* length segments. Each segment is detrended by function *detrend* and windowed by function *window*. *noverlap* gives the length of the overlap between segments. The $|\mathrm{fft}(i)|^2$ of each segment i are averaged to compute *Pxx*, with a scaling to correct for power loss due to windowing. *Fs* is the sampling frequency.

To get a good quality PSD estimate requires a large number of symbols. Here I have increased the symbol count to `N_bits = 1000000`.

```
x,b,data = dc.nrz_bits(100000, 10, 'rect')
psd(x,2**12,10)
xlabel('Frequency (Hz)')
ylim([-40,10]);
```

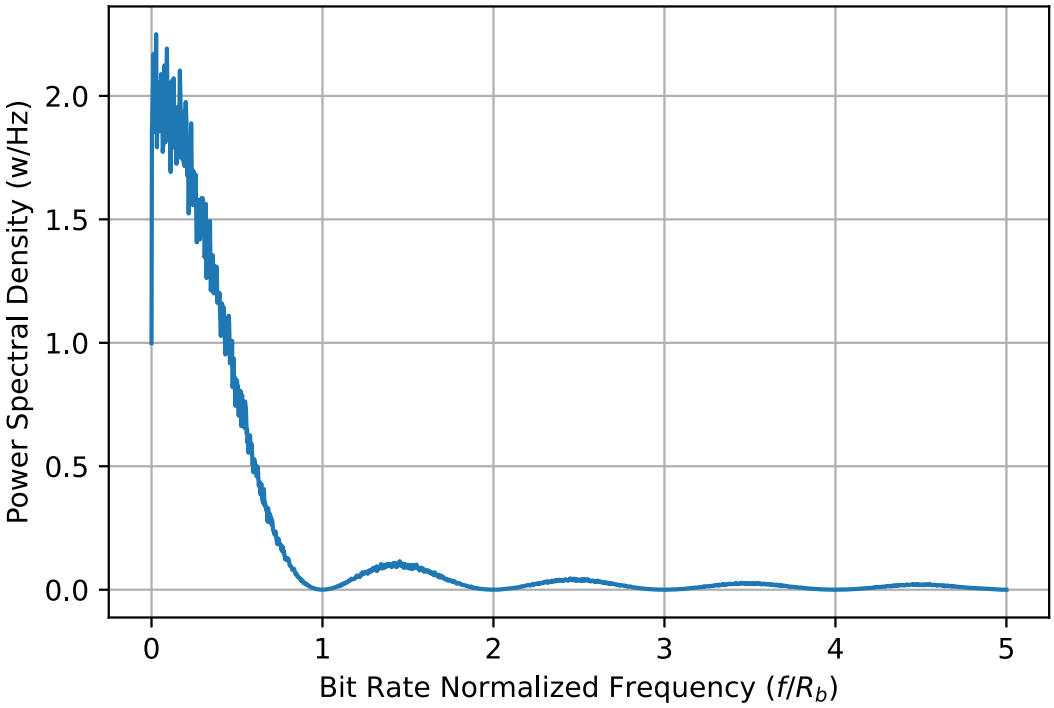


Consider a linear scale plotting of the PSD using `Pxx,ss.my_psd()`:

```
len(x)/2**12
```

```
244.140625
```

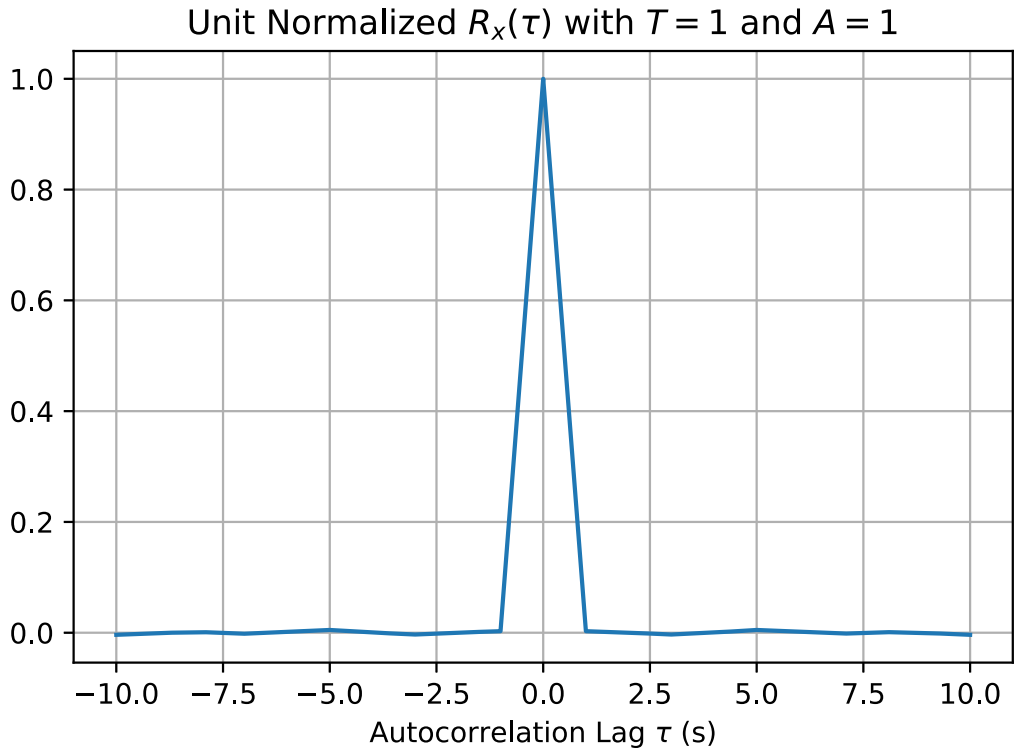
```
Pxx,freq = ss.my_psd(x,2**12,10)
plot(freq,Pxx)
grid()
xlabel('Bit Rate Normalized Frequency ($f/R_b$)')
ylabel('Power Spectral Density (w/Hz)');
```



- In the above figure notice that the $\mathrm{sinc}^2(x)$ spectral shaping is evident.

```
# Correlate x with x
# Output 100 lags either side of zero
# Since lags are in samples, the lag in seconds is +/-(100/100) = +/-1 s
Rx, tau_k = dc.xcorr(x,x,100)
```

```
plot(tau_k/10,real(Rx))
title(r'Unit Normalized $R_x(\tau)$ with $T =1$ and $A = 1$')
xlabel(r'Autocorrelation Lag $\tau$ (s)')
grid()
```



• Randomly Phased Square–Wave as a Special Case

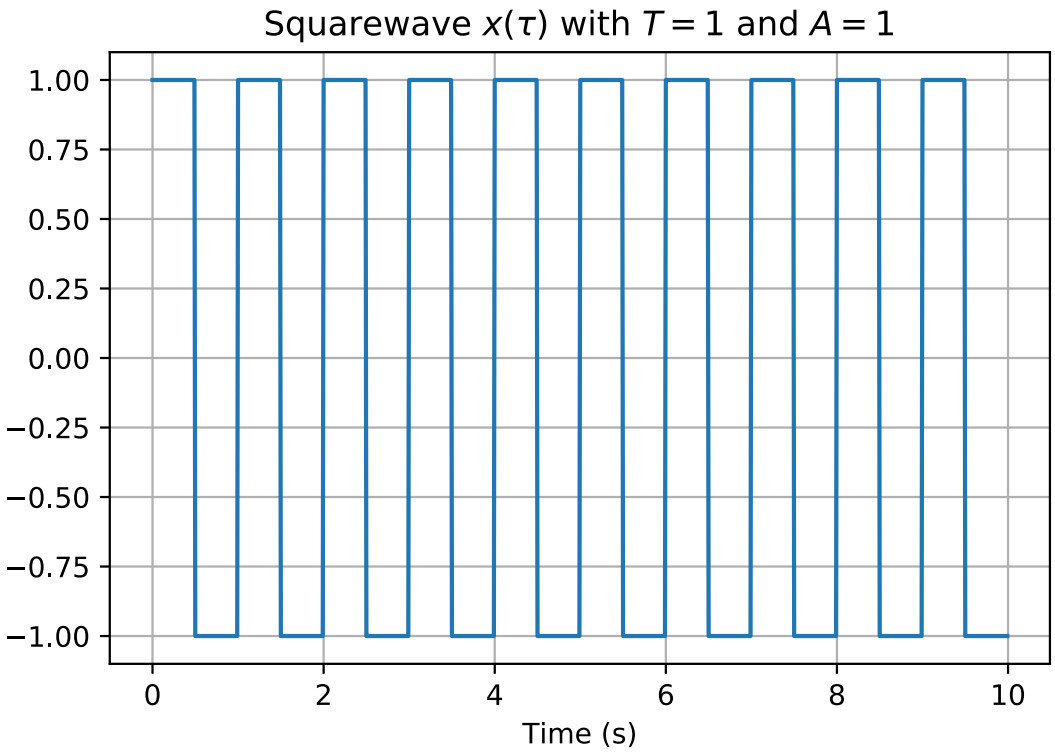
We can view a square–wave as a special case of a random bit stream. The pulse shape of the square–wave is $p(t) = A[u(t) - 2u(t - T/2) + u(t - T)]$ and the random bits are now all 1's correlated, that is $R_m = 1$.

To build up a waveform model we can use `sign()` of a `sin()` function. Assume $T = 1$ and $A = 1$

```
fs = 100
t = arange(0,10,1/100) # fs = 100 Hz
A = 1
T = 1
#x = A*sign(sin(2*pi*t/T))
# Create a modulo T time axis
t_mod = mod(t,T)
x = A*(ss.step(t_mod) - 2*ss.step(t_mod-T/2) + ss.step(t_mod-T))
```

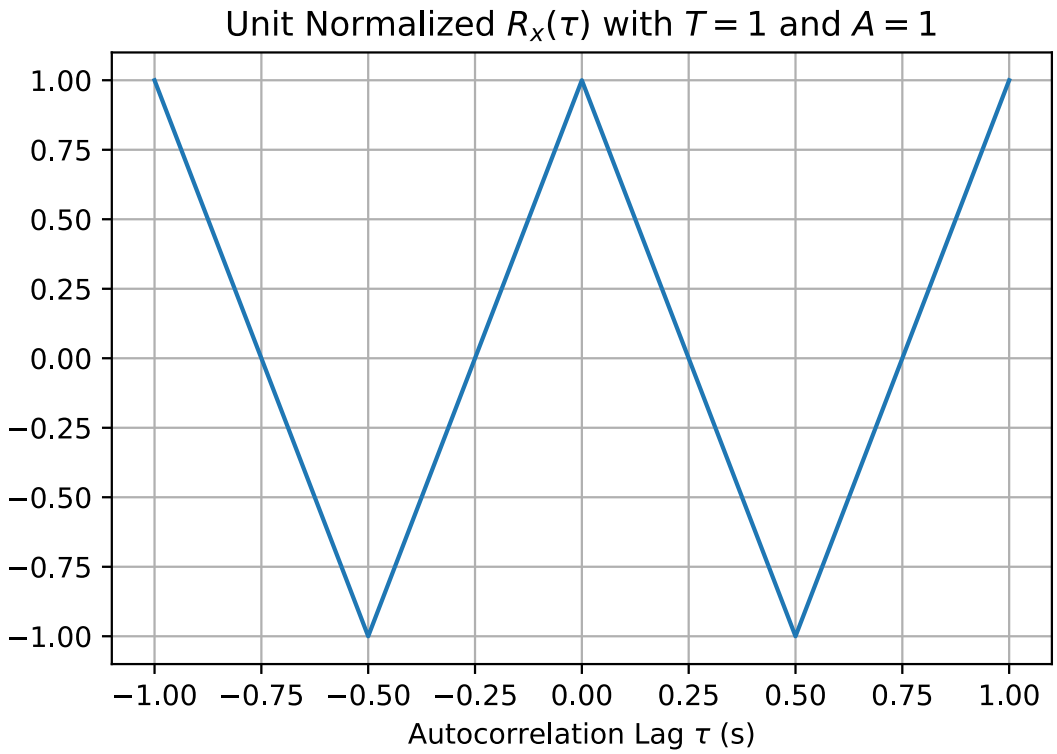
```
plot(t,x)
title(r'Squarewave $x(\tau)$ with $T =1$ and $A = 1$')
xlabel(r'Time (s)')
grid()
ylim([-1.1,1.1])
```

```
(-1.1, 1.1)
```




```
# Correlate x with x
# Output 100 lags either side of zero
# Since lags are in samples, the lag in seconds is +/-(100/100) = +/-1 s
Rx, tau_k = dc.xcorr(x,x,100)
```

```
plot(tau_k/fs,real(Rx))
title(r'Unit Normalized $R_x(\tau)$ with $T =1$ and $A = 1$')
xlabel(r'Autocorrelation Lag $\tau$ (s)')
grid()
```



- Note since $x(t)$ is periodic we know that $R_x(\tau)$ is also periodic
- Note also that as expected $R_x(0) \geq R_x(\tau)$ for all $\tau \neq 0$