In [45]:

```
from pylab import *
%run pam11.py
```

In [46]:

```
%matplotlib inline
```

Lab05, Experiment No 1

showpsd Python Function

Part (a)

To add the showpsd function to the showfun module. It will be used to plot (an approximation to) the PSD of a CT signal x(t) from its samples x(nTs), taken with sampling rate Fs = 1/Ts. Also, to add the enhancement to showpsd (as described in the function header), in a similar fashion as we did for the showft function.

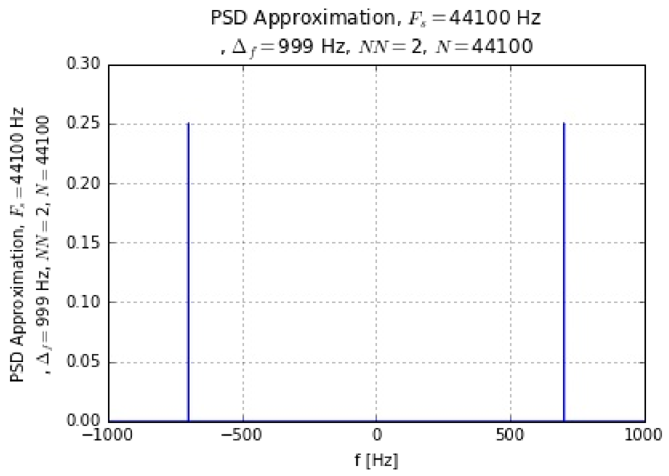
Below is the "showpsd0" function appended to showfun.py file to show the power spectral density of a DT signal. This is the default code to show psd (already given in the lab manual).

In [47]:

```
def showpsd0(xt, Fs, ff_lim, N):
    """
    Plot (DFT/FFT approximation to) power spectral density (PSD) of x(t).
    Displays S_x(f) either linear and absolute or normalized in dB.
    >>>>> showpsd(xt, Fs, ff_lim, N) <<<<<
    where xt:sampled CT signal x(t)
        Fs:sampling rate of x(t)
        ff_lim = [f1,f2,llim]
        f1: lower frequency limit for display
        f2:upper frequency limit for display
        llim = 0: display S_x(f) linear and absolute
        llim < 0: display 10*log_{10}(S_x(f))/max(S_x(f))
        in dB with lower display limit llim dB
        N:blocklength
    """
    # ***** Determine number of blocks, prepare x(t) *****
    N = int(min(N, len(xt)))                         # N <= length(xt) needed
    NN = int(floor(len(xt)/float(N)))                # Number of blocks of length N
    xt = xt[0:N*NN]                                  # Truncate x(t) to NN blocks
    xNN = reshape(xt,(NN,N))                         # NN row vectors of length N

    # ***** Compute DFTs/FFTs, average over NN blocks *****
    Sxf = np.power(abs(fft(xNN)),2.0)                # NN FFTs, mag squared
    if NN > 1:
        Sxf = sum(Sxf, axis=0)/float(NN)

    Sxf = Sxf/float(N*Fs)                             # Correction factor DFT -> PSD
    Sxf = reshape(Sxf,size(Sxf))
    ff = Fs*array(arange(N),int64)/float(N)           # Frequency axis

    if ff_lim[0] < 0:                                 # Negative f1 case
        ixp = where(ff<0.5*Fs)[0]                     # Indexes of pos frequencies
        ixn = where(ff>=0.5*Fs)[0]                    # Indexes of neg frequencies
        ff = hstack((ff[ixn]-Fs,ff[ixp]))             # New freq axis
        Sxf = hstack((Sxf[ixn],Sxf[ixp]))             # Corresponding S_x(f)

    # ***** Determine maximum, trim to ff_lim *****
    maxSxf = max(Sxf)                                 # Maximum of S_x(f)
    ixf = where(logical_and(ff>=ff_lim[0], ff<ff_lim[1]))[0]
    ff = ff[ixf]                                      # Trim to ff_lim specs
    Sxf = Sxf[ixf]

    # ***** Plot PSD *****
    strgt = 'PSD Approximation, $F_s=${:d} Hz\n'.format(Fs)
    strgt = strgt + ', $\\Delta_f=${:.3g} Hz'.format(ff[-1])
    strgt = strgt + ', $NN=${:d}, $N=${:d}'.format(NN, N)
    f1 = figure()
    af1 = f1.add_subplot(111)
    af1.plot(ff, Sxf, '-b')
    af1.grid()
    af1.set_xlabel('f [Hz]')
    af1.set_ylabel(strgt)
    af1.set_title(strgt)
    show()
```

In [48]:

```
Fs = 44100                              # Sampling rate
f1 = 700                                # Test frequency 1
f2 = 720                                # Test frequency 1
tlen = 2                                # Duration in seconds
tt = arange(round(tlen*Fs))/float(Fs)   # Time axis
x1t = sin(2*pi*f1*tt)                   # Sine with freq f1
x2t = 0.01*cos(2*pi*f2*tt)              # Attenuated cosine with freq f2
xt = x1t+x2t                            # Combined sinusoidal signal
showpsd0(xt,Fs,[-1000,1000,0],Fs)       # Plot S_x(f)
```



In the above plot, we can't observe the x2t signal as it's strength is very low. So, we need to modify the above code so that it can show the weak strength signal too.

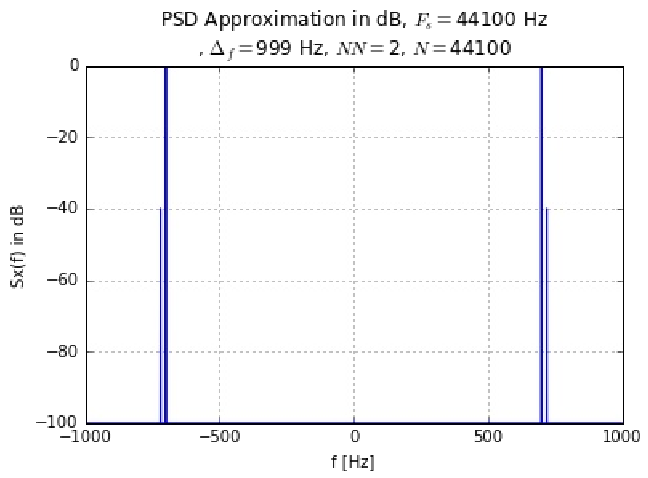
Modifying the "showpsd0" function to "showpsd" function (and appending it to showfun.py file) so that it can show the PSD in dB too.

In [49]:

```
def showpsd(xt, Fs, ff_lim, N):
    """
    Plot (DFT/FFT approximation to) power spectral density (PSD) of x(t).
    Displays S_x(f) either linear and absolute or normalized in dB.
    >>>>> showpsd(xt, Fs, ff_lim, N) <<<<<
    where xt:sampled CT signal x(t)
        Fs:sampling rate of x(t)
        ff_lim = [f1,f2,llim]
        f1: lower frequency limit for display
        f2:upper frequency limit for display
        llim = 0: display S_x(f) linear and absolute
        llim < 0: display 10*log_{10}(S_x(f))/max(S_x(f))
        in dB with lower display limit llim dB
        N:blocklength
    """
    # ***** Determine number of blocks, prepare x(t) *****
    N = int(min(N, len(xt)))                            # N <= length(xt) needed
    NN = int(floor(len(xt)/float(N)))                   # Number of blocks of length N
    xt = xt[0:N*NN]                                     # Truncate x(t) to NN blocks
    xNN = reshape(xt,(NN,N))                            # NN row vectors of length N
    llim = ff_lim[2]                                    # to display Sxf in db or linear

    # ***** Compute DFTs/FFTs, average over NN blocks *****
    Sxf = np.power(abs(fft(xNN)),2.0)                   # NN FFTs, mag squared
    if NN > 1:
        Sxf = sum(Sxf, axis=0)/float(NN)

    Sxf = Sxf/float(N*Fs)                                # Correction factor DFT -> PSD
    Sxf = reshape(Sxf,size(Sxf))
    ff = Fs*array(arange(N),int64)/float(N)              # Frequency axis

    if ff_lim[0] < 0:                                    # Negative f1 case
        ixp = where(ff<0.5*Fs)[0]                        # Indexes of pos frequencies
        ixn = where(ff>=0.5*Fs)[0]                       # Indexes of neg frequencies
        ff = hstack((ff[ixn]-Fs,ff[ixp]))                # New freq axis
        Sxf = hstack((Sxf[ixn],Sxf[ixp]))                # Corresponding S_x(f)

    # ***** Determine maximum, trim to ff_lim *****
    maxSxf = max(Sxf)                                    # Maximum of S_x(f)

    if llim < 0:
        Sxf  = 10*log10((Sxf/maxSxf))
        ix = where(Sxf <= llim)
        Sxf[ix] = llim*ones(len(ix))


    ixf = where(logical_and(ff>=ff_lim[0], ff<ff_lim[1]))[0]
    ff = ff[ixf]                                         # Trim to ff_lim specs
    Sxf = Sxf[ixf]

    # ***** Plot PSD *****
    if llim < 0:
        strgt = 'PSD Approximation in dB, $F_s=${:d} Hz\n'.format(Fs)
        strgt1 = 'Sx(f) in dB'
    else:
        strgt = 'PSD Approximation, $F_s=${:d} Hz\n'.format(Fs)
        strgt1 = 'Sx(f)'
    strgt = strgt + ', $\\Delta_f=${:.3g} Hz'.format(ff[-1])
    strgt = strgt + ', $NN=${:d}, $N=${:d}'.format(NN, N)
    f1 = figure()
    af1 = f1.add_subplot(111)
    af1.plot(ff, Sxf, '-b')
    af1.grid()
    af1.set_xlabel('f [Hz]')
    af1.set_ylabel(strgt1)
    af1.set_title(strgt)
    show()
```

In [50]:

```
Fs = 44100                              # Sampling rate
f1 = 700                                # Test frequency 1
f2 = 720                                # Test frequency 1
tlen = 2                                # Duration in seconds
tt = arange(round(tlen*Fs))/float(Fs)   # Time axis
x1t = sin(2*pi*f1*tt)                   # Sine with freq f1
x2t = 0.01*cos(2*pi*f2*tt)              # Attenuated cosine with freq f2
xt = x1t+x2t                            # Combined sinusoidal signal
showpsd(xt,Fs,[-1000,1000,-100],Fs)     # Plot S_x(f)
```



In [ ]:

Now in the above plot, we can see both the sine component i.e x1t and x2t signals both.

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

Part (b)

To add a feature to showpsd so that it displays the total power Px , as well as Px(f1, f2) for f1 = ff_lim[0] and f2 = ff_lim[1], in the title bar.

Modifying the "showpsd" function so that it displays both the PSD of the signal and also the power content of the signal for a given frequency range.

In [51]:

```
def showpsd(xt, Fs, ff_lim, N):
    """
    Plot (DFT/FFT approximation to) power spectral density (PSD) of x(t).
    Displays S_x(f) either linear and absolute or normalized in dB.
    >>>>> showpsd(xt, Fs, ff_lim, N) <<<<<
    where xt:sampled CT signal x(t)
        Fs:sampling rate of x(t)
        ff_lim = [f1,f2,llim]
        f1: lower frequency limit for display
        f2:upper frequency limit for display
        llim = 0: display S_x(f) linear and absolute
        llim < 0: display 10*log_{10}(S_x(f))/max(S_x(f))
        in dB with lower display limit llim dB
        N:blocklength
    """
    # ***** Determine number of blocks, prepare x(t) *****
    N = int(min(N, len(xt)))                          # N <= length(xt) needed
    NN = int(floor(len(xt)/float(N)))                 # Number of blocks of length N
    xt = xt[0:N*NN]                                   # Truncate x(t) to NN blocks
    xNN = reshape(xt,(NN,N))                          # NN row vectors of length N
    llim = ff_lim[2]                                  # to display Sxf in db or linear

    # ***** Compute DFTs/FFTs, average over NN blocks *****
    Sxf = np.power(abs(fft(xNN)),2.0)                 # NN FFTs, mag squared
    if NN > 1:
        Sxf = sum(Sxf, axis=0)/float(NN)

    Sxf = Sxf/float(N*Fs)                              # Correction factor DFT -> PSD
    Sxf = reshape(Sxf,size(Sxf))
    ff = Fs*array(arange(N),int64)/float(N)            # Frequency axis

    if ff_lim[0] < 0:                                  # Negative f1 case
        ixp = where(ff<0.5*Fs)[0]                      # Indexes of pos frequencies
        ixn = where(ff>=0.5*Fs)[0]                     # Indexes of neg frequencies
        ff = hstack((ff[ixn]-Fs,ff[ixp]))              # New freq axis
        Sxf = hstack((Sxf[ixn],Sxf[ixp]))              # Corresponding S_x(f)

    Px = (cumsum(Sxf)*Fs/N)[-1]                        # Calculating total power of the signal

    # ***** Determine maximum, trim to ff_lim *****
    maxSxf = max(Sxf)                                  # Maximum of S_x(f)

    ixf = where(logical_and(ff>=ff_lim[0], ff<ff_lim[1]))[0]
    ff = ff[ixf]                                       # Trim to ff_lim specs
    Sxf = Sxf[ixf]

    Px_prime = (cumsum(Sxf)*Fs/N)[-1]                  # Calculating the power of the signal in
                                                       # frequency range of [ff_lim[0], ff_lim[1]]
    Px_prime_perc = (Px_prime/Px) * 100.0              # Changing the power content in percentage

    # ***** Changing Sxf into dB when llim is less than zero *****
    if llim < 0:
        Sxf  = 10*log10((Sxf/maxSxf))
        ix = where(Sxf <= llim)
        Sxf[ix] = llim*ones(len(ix))


    # ***** Plot PSD *****
    if llim < 0:
        strgt = '$P_x=${:0.2f}, $P_x(f1,f2)=${:0.2f}%, $F_s=${:d} Hz\n'.format(Px,Px_prime_perc,Fs)
        strgt1 = 'Sx(f) in dB'
    else:
        strgt = '$P_x=${:0.2f}, $P_x(f1,f2)=${:0.5f}, $F_s=${:d} Hz\n'.format(Px,Px_prime,Fs)
        strgt1 = 'Sx(f)'
    strgt = strgt + ', $\\Delta_f=${:.3g} Hz'.format(ff[-1])
    strgt = strgt + ', $NN=${:d}, $N=${:d}'.format(NN, N)
    f1 = figure()
    af1 = f1.add_subplot(111)
    af1.plot(ff, Sxf, '-b')
    af1.grid()
    af1.set_xlabel('f [Hz]')
    af1.set_ylabel(strgt1)
    af1.set_title(strgt)
    show()
```

In [52]:

```
Fs = 44100                               # Sampling rate
tlen =1                                # Duration in seconds
tp = 0.1                                 # Rectangular Pulse Width
nones = int(tp*Fs)                       # Number of ones in the sequence
nzeros = int(tlen*Fs - nones)            # Number of zeros in the sequence
xt = array(hstack((zeros(int(nzeros/2)),ones(nones),zeros(int(tlen*Fs)-nones-int(nzeros/2)))),int8)
```

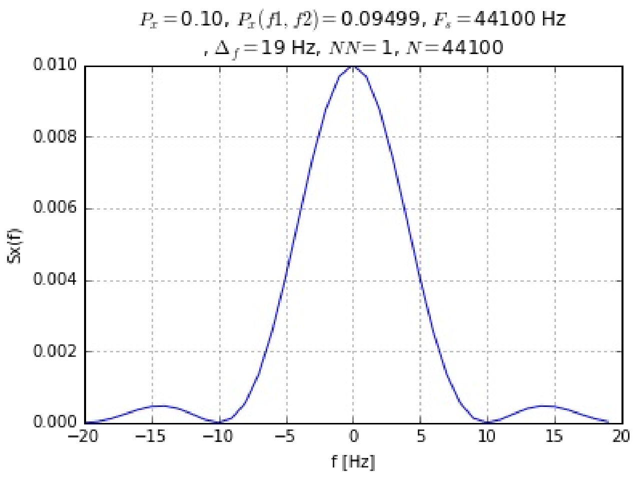
Generating the PSD of x(t) as required in Lab Manual (not in dB)

In [ ]:

```
```

In [53]:

```
showpsd(xt,Fs,[-20,20,0],Fs)        # Plot S_x(f)
```



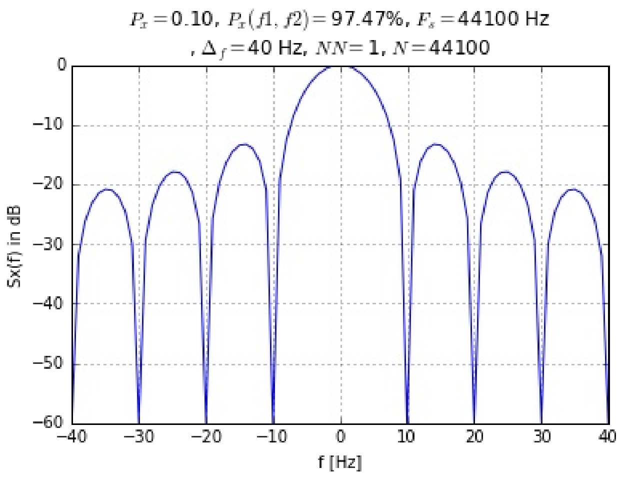
In [ ]:

```
```

Generating the PSD of x(t) as required in Lab Manual (in dB)

In [54]:

```
showpsd(xt,Fs,[-40,41,-60],Fs)        # Plot S_x(f)
```



In [ ]:

```
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

Part (c)

To generate about 5 sec of random binary polar PAM signals with rectangular, triangular, and sinc (tail length ≈ 20, no windowing) pulses by using FB = 100 and Fs = 44100 Hz and displaying the PSDs of the three signals in dB, with lower limit -60 dB, over the frequency range [−500-500 Hz].

Generating random binary polar PAM signal using 'rect' pulse.

In [55]:

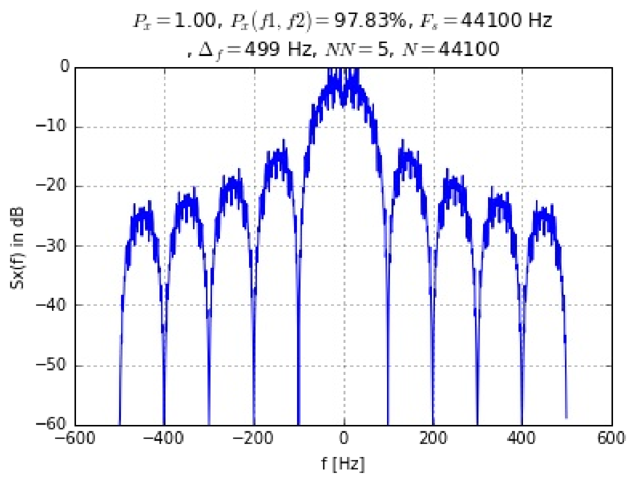
```
tlen = 5                              # Required time (as in Lab Manual) of the signal
FB = 100                              # Bit Rate
Fs = 44100                            # Sampling Frequency
nbits = int(tlen*FB)                  # Number of bits in the "tlen" time
an = 2*(around(2*rand(nbits))%2)-1    # Generating a sequence of zeros and ones
```

In [56]:

```
tt,st=pam11(an, FB,Fs,'rect')         # Generating the signal samples using pam11 function
```

In [57]:

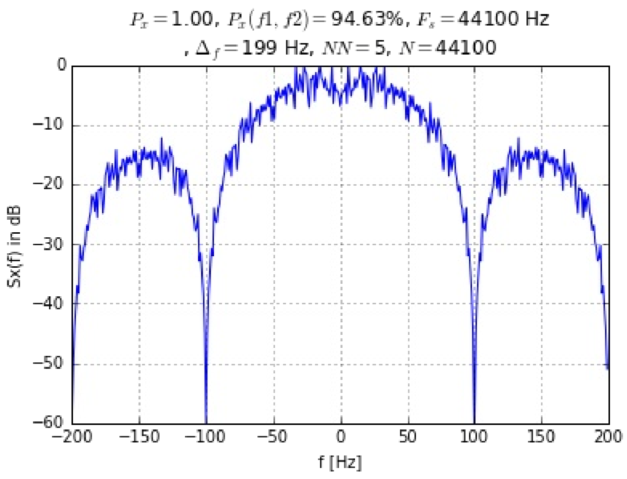
```
showpsd(st,Fs,[-500,500,-60],Fs)      # Plotting PSD in frequency range of -500 to 500
```



Now we will be plotting the PSD in frequency range of -200 to 200 (-2*FB, 2*FB).

In [58]:

```
showpsd(st,Fs,[-200,200,-60],Fs)        # Plotting PSD in frequency range of -200 to 200 (-2*FB, 2*FB)
```



Observation: The PSD of rectangular pulse is sinc square and and we are getting the same in the above plot. Also, in the -2 *FB to 2* FB range, a reasonable amount of energy is present in this bandwidth.

Generating random binary polar PAM signal using 'tri' pulse.

In [59]:

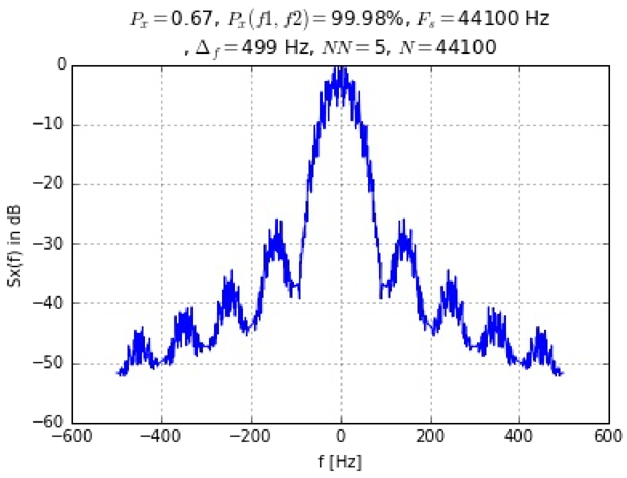
```
tlen = 5                                # Required time (as in Lab Manual) of the signal
FB = 100                                # Bit Rate
Fs = 44100                              # Sampling Frequency
nbits = int(tlen*FB)                    # Number of bits in the "tlen" time
an = 2*(around(2*rand(nbits))%2)-1      # Generating a sequence of zeros and ones
```

In [60]:

```
tt,st=pam11(an, FB,Fs,'tri')            # Generating the signal samples using pam11 function
```

In [61]:

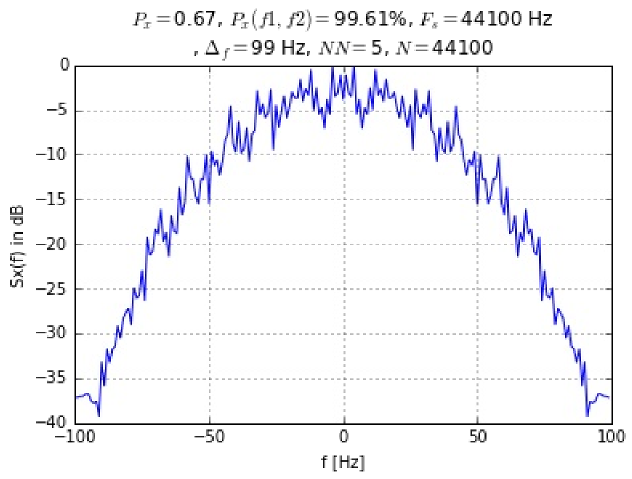
```
showpsd(st,Fs,[-500,500,-60],Fs)        # Plotting PSD in frequency range of -500 to 500
```



Now we will be plotting the PSD in frequency range of -100 to 100 (-FB, FB).

In [62]:

```
showpsd(st,Fs,[-100,100,-60],Fs)       # Plotting PSD in frequency range of -100 to 100 (-FB, FB)
```



Observation: The PSD of rectangular pulse is sinc square and and we are getting the same in the above plot. Also, in the -FB to FB range, a reasonable amount of energy is present in this bandwidth.

Generating random binary polar PAM signal using 'sinc' pulse

In [63]:

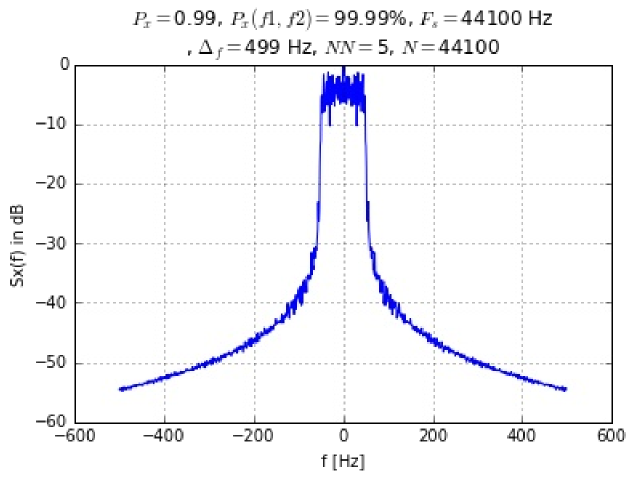
```
tlen = 5                               # Required time (as in Lab Manual) of the signal
FB = 100                               # Bit Rate
Fs = 44100                             # Sampling Frequency
nbits = int(tlen*FB)                   # Number of bits in the "tlen" time
an = 2*(around(2*rand(nbits))%2)-1     # Generating a sequence of zeros and ones
```

In [64]:

```
tt,st=pam11(an, FB,Fs,'sinc', [10,0])           # Generating the signal samples using pam11 function
```

In [65]:

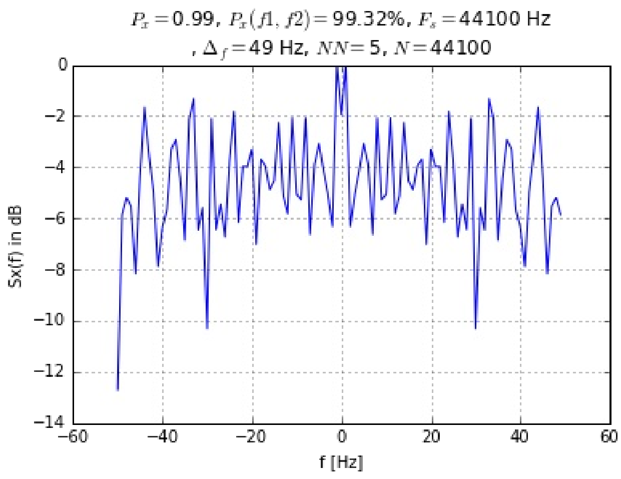
```
showpsd(st,Fs,[-500,500,-60],Fs)
```



Now we will be plotting the PSD in frequency range of -50 to 50 (-FB/2, FB/2).

In [66]:

```
showpsd(st,Fs,[-50,50,-60],Fs)       # Plotting PSD in frequency range of -50 to 50 (-FB/2, FB/2)
```



Observation: The PSD of rectangular pulse is sinc square and and we are getting the same in the above plot. Also, in the -FB/2 to FB/2 range, a reasonable amount of energy is present in this bandwidth.

In [ ]:

```
```

In [ ]:

```
```

Part 3 (d) is done in GNU Radio and a PDF report and the GRC file are attached in the folder. File name: Lab05_Experiment_1d.pdf, Lab05_Experiment_1d.grc.

In [ ]:

```
```