Using SymPy for Root Finding

```
%pylab inline
#%matplotlib qt
import sk_dsp_comm.sigsys as ss
import scipy.signal as signal
import sk_dsp_comm.digitalcom as dc
import scipy.special as special
import scipy.stats as stats
```

```
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
```

Set up `SymPy`

```
from IPython.display import display
from sympy.interactive import printing
printing.init_printing(use_latex='mathjax')
import sympy as sym
```

Define some symbolic variables for use in the remainder of this example.

```
Pe, SNR, SNRdB, SJR, SJRdB = sym.symbols("P_E, SNR, SNR_dB, SJR, SJR_dB",positive=True)
x = sym.symbols("x", real=True)
```

Define the Q(x) Function

```
Qsymp = 1/2*sym.erfc(x/sym.sqrt(2))
Qsymp
```

$0.5 \operatorname{erfc}\left(\frac{\sqrt{2} x}{2}\right)$

Problem Definition

Here we conider a simple model for the BEP of BPSK receiver impaired by a bradband noise jammer. The BEP is modeled with the addition of term N_j which represents a jamming noise spectral density. The BEP expression is

$$P_E = Q\left(\sqrt{\frac{2E_b}{N_0 + N_j}}\right) = Q\left(\sqrt{\frac{2}{N_0/E_b + N_j/E_b}}\right) \qquad (1)$$

Next we write the P_E expression using the Q-function defined using `sympy` functions, where we define $\mathrm{SNR} = E_b/N_0$ and $\mathrm{SJR} = E_b/N_j$. In working with `SymPy` it is very common to make variable substitutions, such as in the creation of the `Pe` symbolic function below:

```
Pe = Qsymp.subs({x:sym.sqrt(2/(1/SNR+1/SJR))})
Pe
```

$$0.5\,\mathrm{erfc}\left(\frac{1}{\sqrt{\frac{1}{SNR}+\frac{1}{SJR}}}\right)$$

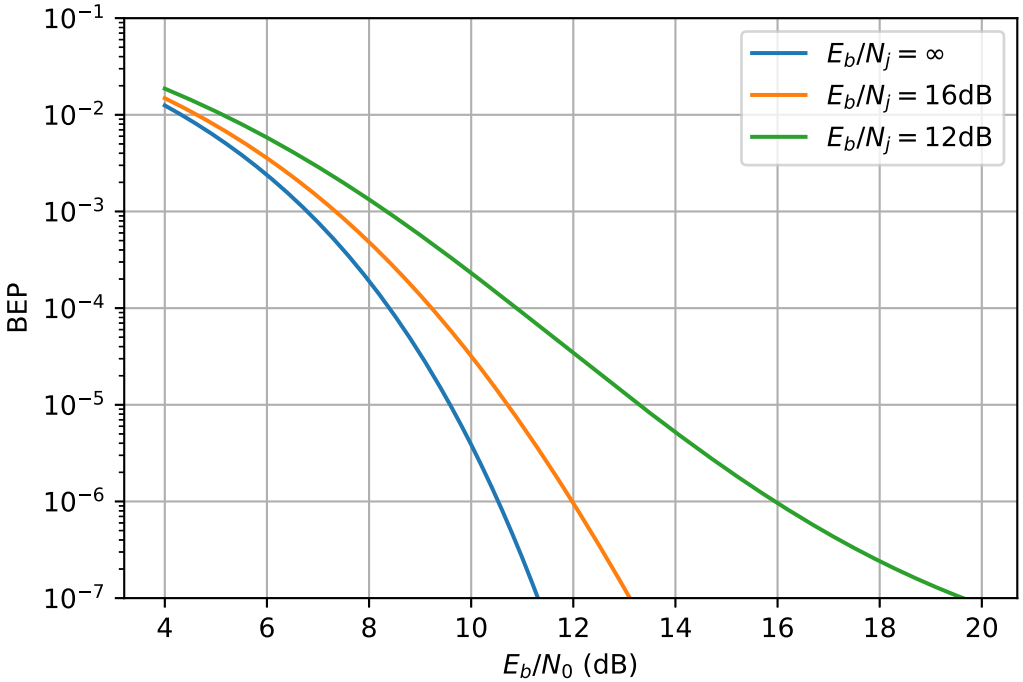
Next we plot some BEP curves by filling arrays using a `for` loop, since the `sympy` function is not immediately vectoriazable as it would be in pure `numpy`. NOte another option is use `sym.lambdify()` on the symbolic expression, but I an running into errors with `erfc` not found. Keep it simple for now, just not as fast using a `for` loop.

We choose a range of 4 to 20 dB in 0.1 dB steps and use SJR values in dB of ∞, 16, and 12 dB:

```
SNR_dB_num = arange(4,20,.1)
Pe_val1 = zeros_like(SNR_dB_num)
Pe_val2 = zeros_like(SNR_dB_num)
Pe_val3 = zeros_like(SNR_dB_num)
for k,SNR_dB_k in enumerate(SNR_dB_num):
    Pe_val1[k] = Pe.subs({SNR:10**(SNR_dB_k/10),SJR:sym.oo})
    Pe_val2[k] = Pe.subs({SNR:10**(SNR_dB_k/10),SJR:10**(16/10)})
    Pe_val3[k] = Pe.subs({SNR:10**(SNR_dB_k/10),SJR:10**(12/10)})
```

Plot the results:

```
semilogy(SNR_dB_num,Pe_val1)
semilogy(SNR_dB_num,Pe_val2)
semilogy(SNR_dB_num,Pe_val3)
ylabel(r'BEP')
xlabel(r'$E_b/N_0$ (dB)')
ylim([1e-7,1e-1])
legend((r'$E_b/N_j = \infty$',r'$E_b/N_j = 16$dB',r'$E_b/N_j = 12$dB'),loc='best')
grid();
```



The above lets use see how SJR pushed the BEP curves to the right and causes flaring. Suppose now we want to use `sym.nsolve` to find the BEP degradation in dB at $P_E = 10^{-6}$. As a quick check we can use `nsolve` to find the E_b/N_0 value in dB when $\text{SJR} \to \infty$. This will give us the $E_b?N_0$ crossing point for $P_E = 10^{-6}$, which is the refernce point for degradation calculations that follow.

```
sym.nsolve(1e-6 - Pe.subs({SNR:10**(SNRdB/10),SJR:sym.oo}),10)
```

10.5298316995714

No surprises here, we have seen the 10.529 dB number when using the inverse Q-function via `scipystats.norm.isf()`, i.e.,

```
Pe = [1e-4,1e-5,1e-6,1e-7]
for n, Pe_n in enumerate(Pe):
    EbN0_dB = 10*log10(stats.norm.isf(Pe[n])**2/2)
    print('Pe = %2.1e <==> (Eb/No) = %6.3f dB' % (Pe_n,EbN0_dB))

Pe = 1.0e-04 <==> (Eb/No) =  8.398 dB
Pe = 1.0e-05 <==> (Eb/No) =  9.588 dB
Pe = 1.0e-06 <==> (Eb/No) = 10.530 dB
Pe = 1.0e-07 <==> (Eb/No) = 11.309 dB
```

Note the second argument is the initial guess on the solution. Here I have chosen 10dB. If the guess is too far away from the solution/root, an error occurs. Also in `Sympy` infinity is denoted using `sym.oo`.

Moving on, build up a solution for the degradation when $\text{SJR}_{dB} = 12$ dB. Here we use `sym.N()` to convert to a numerical answer with 6 digit precision.

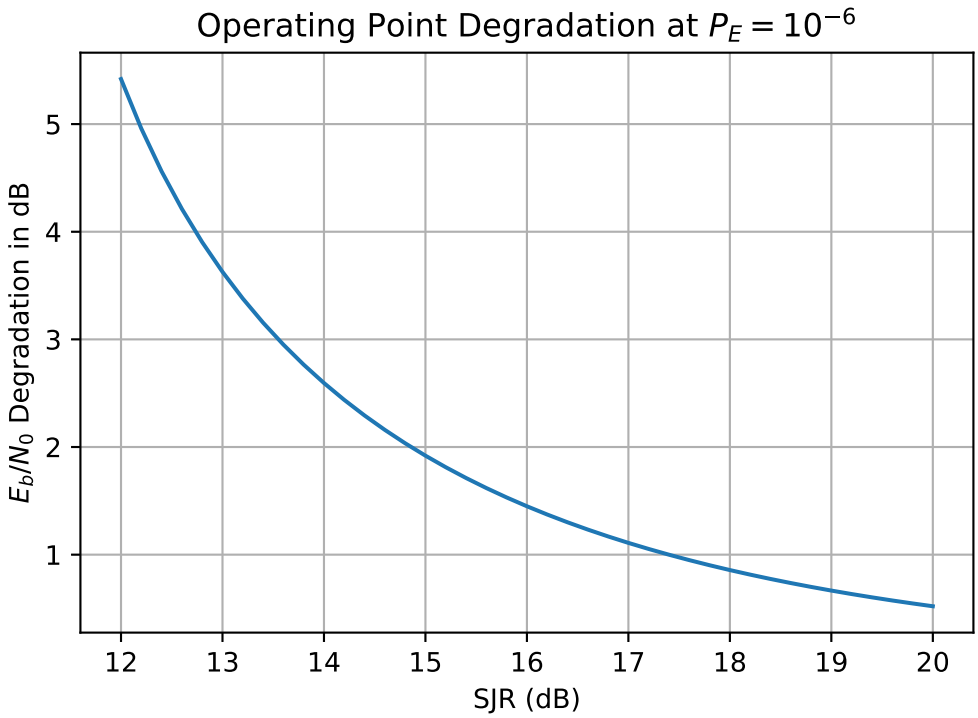
```
SJRdB = 12
Pe_thresh = 1e-6
sym.N(sym.nsolve(Pe_thresh - Pe.subs({SNR:10**(SNRdB/10),SJR:10**(SJRdB/10)}),18) \
-sym.nsolve(Pe_thresh - Pe.subs({SNR:10**(SNRdB/10),SJR:sym.oo}),10),6)
```

5.41854

Create a degradation plot in terms of SJR in dB. Note we must have $\mathrm{SJR}_{dB} > \mathrm{SNR}_{dB}$ otherwise there is no value of E_b/N_0 that can overcome the jammer power. Some added parameters are included on the first use of `nsolve` to help with convergence over a range of operating points, e.g.,

```
# The (10,20) sets a range of values expected for the solution
# The solver type is set to bisection
# Error reporting is turned off with verify=False,
# which means you are on your own to verify the correct solution
sym.N(sym.nsolve(Pe_thresh - Pe.subs({SNR:10**(SNRdB/10),SJR:10**(SJRdB_k/10)}),
                                    (10,20),solver='bisect',verify=False)
```

```
SJRdB = arange(12,20 +.2,0.2)
Pe_thresh = 1e-6
Pe_deg_dB = zeros_like(SJRdB)
for k, SJRdB_k in enumerate(SJRdB):
    Pe_deg_dB[k] = sym.N(sym.nsolve(Pe_thresh - Pe.subs({SNR:10**(SNRdB/10),SJR:10**
(SJRdB_k/10)}),
                                    (10,20),solver='bisect',verify=False) \
    -sym.nsolve(Pe_thresh - Pe.subs({SNR:10**(SNRdB/10),SJR:sym.oo}),10),6)
plot(SJRdB,Pe_deg_dB)
ylabel(r'$E_b/N_0$ Degradation in dB')
xlabel(r'SJR (dB)')
title(r'Operating Point Degradation at $P_E = 10^{-6}$')
grid();
```



Hopefully the above will help you get started with using `sym.nsolve` for other applications.