Τηλεπικοινωνιακά Συστήματα

Διάλεξη 8η

Θωμάς Καμαλάκης

Χαροκόπειο Πανεπιστήμιο Αθηνών

Οκτώβριος 2020

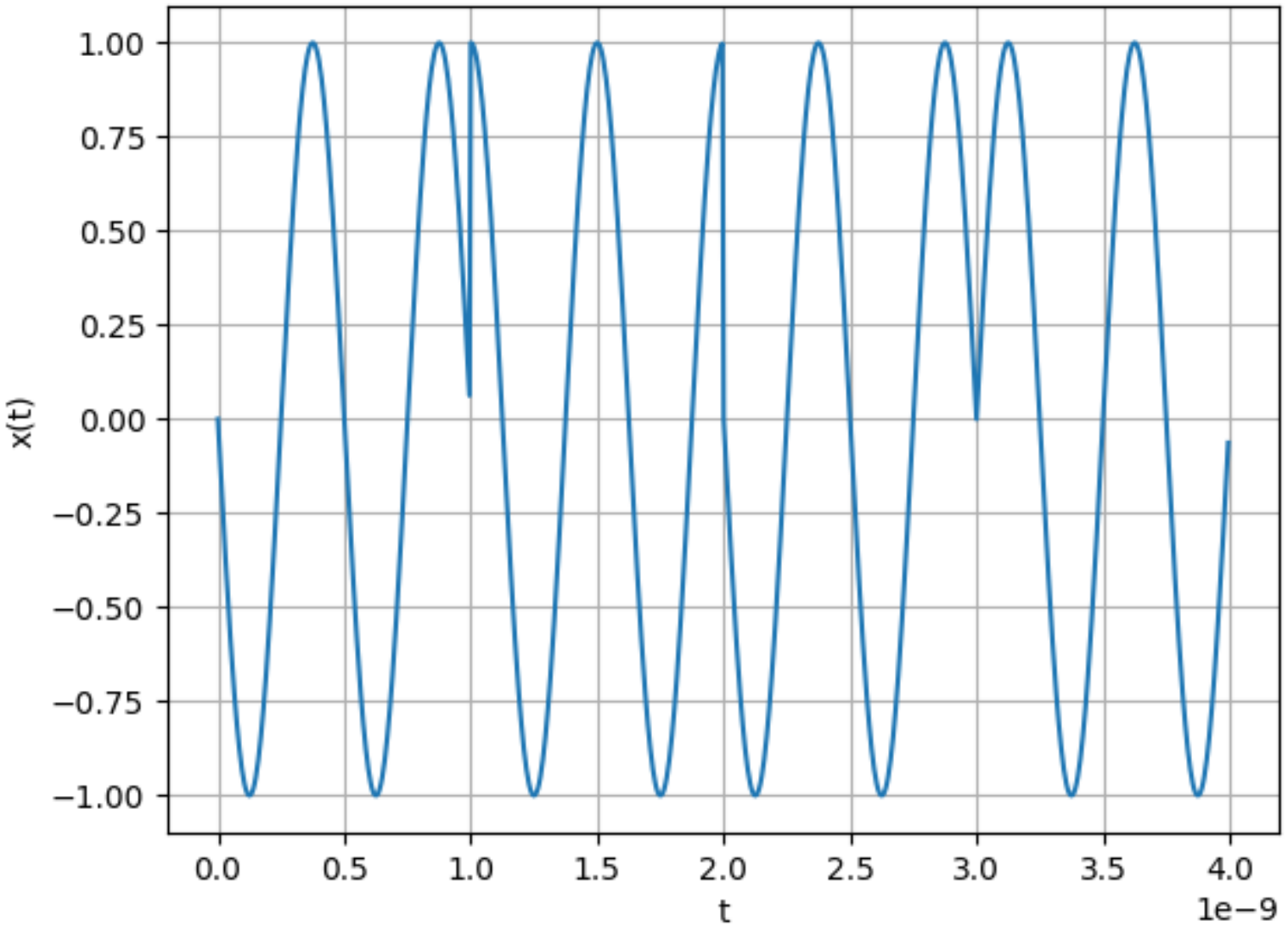
Περιεχόμενα

psk.py

```
1  import commlib as cl
2  import matplotlib.pyplot as plt
3
4  M = 4
5  TS = 1e—9
6  samples_per_symbol = 200
7  f0 = 2e9
8  Nbits = 8
9
10 c = cl.psk_constellation( M = M )
11 x = cl.digital_signal( TS = TS, samples_per_symbol = samples_per_symbol,
12                       constellation = c, fcarrier = f0 )
13 bits = cl.random_bits( Nbits )
14 x.modulate_from_bits( bits )
15 x.plot( close_all = True )
16 plt.grid()
```

commlib.py

```
386     def modulate_from_symbols( self, symbols ):
387
388         self.Tmin = self.tinitial — self.tguard
389         self.Tmax = self.Tmin + symbols.size * self.TS + 2 * self.tguard
390         self.Dt = self.TS / self.samples_per_symbol
391         self.t = np.arange(self.Tmin, self.Tmax, self.Dt)
392         self.samples = np.zeros( self.t.size )
393         self.symbols = symbols
394         self.N = self.t.size
395
396         i = np.floor( (self.t — self.tinitial) / self.TS).astype(int)
397         j = np.where( np.logical_and(i >= 0, i < symbols.size ) )
398
399         phase = 2 * np.pi * self.fcarrier * self.t[j] + self.phi0
400
401         self.samples[j] = np.real(symbols[ i[j] ]) * np.cos( phase ) \
402                         — np.imag(symbols[ i[j] ]) * np.sin( phase )
```


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
x(t)
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
t
1e−9

Ενέργεια σήματος ($L = 1$)

- Εφόσον:

$$\int_{(k-1)T_S}^{kT_S} |x(t)|^2 \mathrm{d}t = \int_{(k-1)T_S}^{kT_S} |x_1 p_1(t) + x_2 p_2(t)|^2 \mathrm{d}t =$$

$$\int_{(k-1)T_S}^{kT_S} (x_1^2 p_1^2(t) + x_2^2 p_2^2(t) + 2x_1 x_2 p_1(t) p_2(t)) \mathrm{d}t =$$

$$x_1^2 \int_{(k-1)T_S}^{kT_S} p_1^2(t) \mathrm{d}t + x_2^2 \int_{(k-1)T_S}^{kT_S} p_2^2(t) \mathrm{d}t + 2x_1 x_2 \int_{(k-1)T_S}^{kT_S} p_1 p_2(t) \mathrm{d}t$$

$$= x_1^2 + x_2^2 = 1$$

- έχουμε για την αναμενόμενη ενέργεια του σήματος στην διάρκεια του $k^{\text{οστού}}$ συμβόλου,

$$\mathcal{E}_k = \mathbb{E}\left\{ \int_{(k-1)T_S}^{kT_S} |x(t)|^2 \mathrm{d}t \right\} = 1$$

Ισχύς του σήματος

- Η μέση ισχύς του σήματος είναι:

$$\mathcal{P} = \mathbb{E}\left\{\lim_{\tau\to\infty}\frac{1}{\tau}\int_{-\frac{\tau}{2}}^{+\frac{\tau}{2}}|x(t)|^2\mathrm{d}t\right\} = \lim_{n\to\infty}\mathbb{E}\left\{\frac{1}{2nT_\mathrm{S}}\int_{-nT_\mathrm{S}}^{+nT_\mathrm{S}}|x(t)|^2\mathrm{d}t\right\}$$

- Αλλά δεδομένου ότι:

$$\int_{-nT_\mathrm{S}}^{+nT_\mathrm{S}}|x(t)|^2\mathrm{d}t = 2nT_\mathrm{S}$$

- Θα έχουμε:

$$\mathcal{P} = 1\frac{1}{T_\mathrm{S}}$$

Πηλίκο σήμα-προς-θόρυβο

- Στην περίπτωση όπου έχουμε απώλειες $L \neq 1$ τότε απλά έχουμε:

$$\mathcal{E}_k = L$$

- Το πηλίκο σήμα-προ-θόρυβο *συμβόλου*

$$\text{SNR}_\text{S} = \frac{\mathcal{E}_k}{N_0} = \frac{L}{N_0}$$

- Το πηλίκο σήμα-προ-θόρυβο *bit*

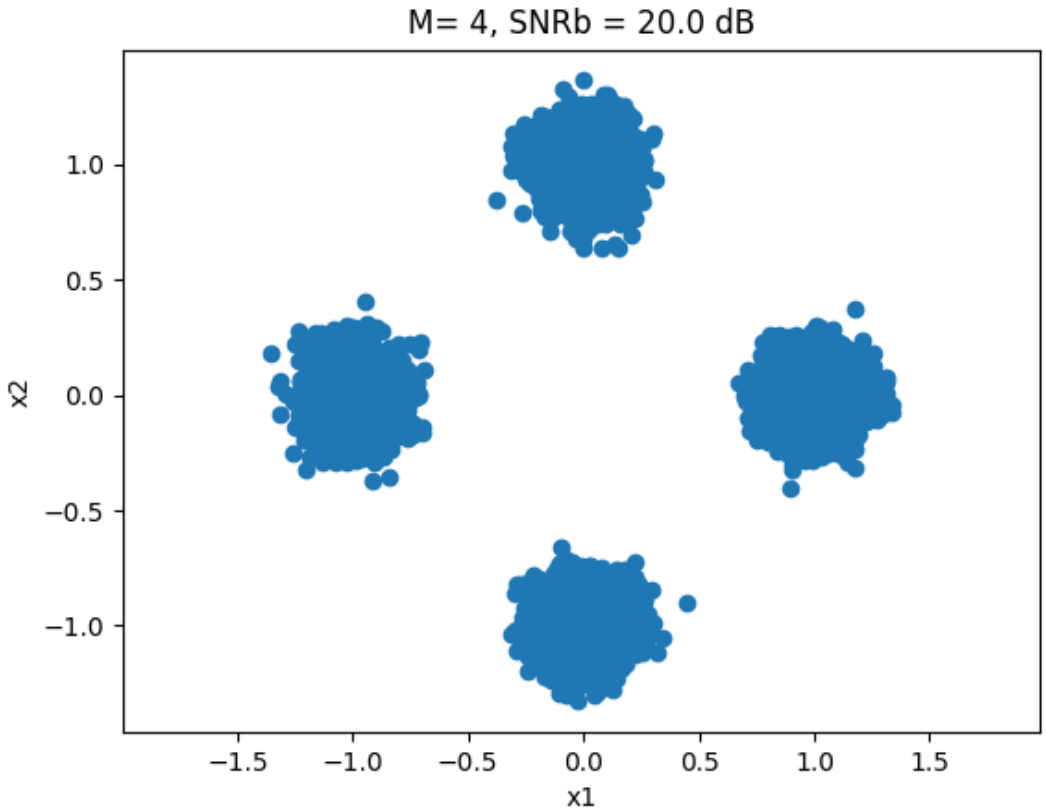
$$\text{SNR}_\text{b} = \frac{\mathcal{E}_k}{N_0 \log_2 M} = \frac{L}{N_0 \log_2 M}$$

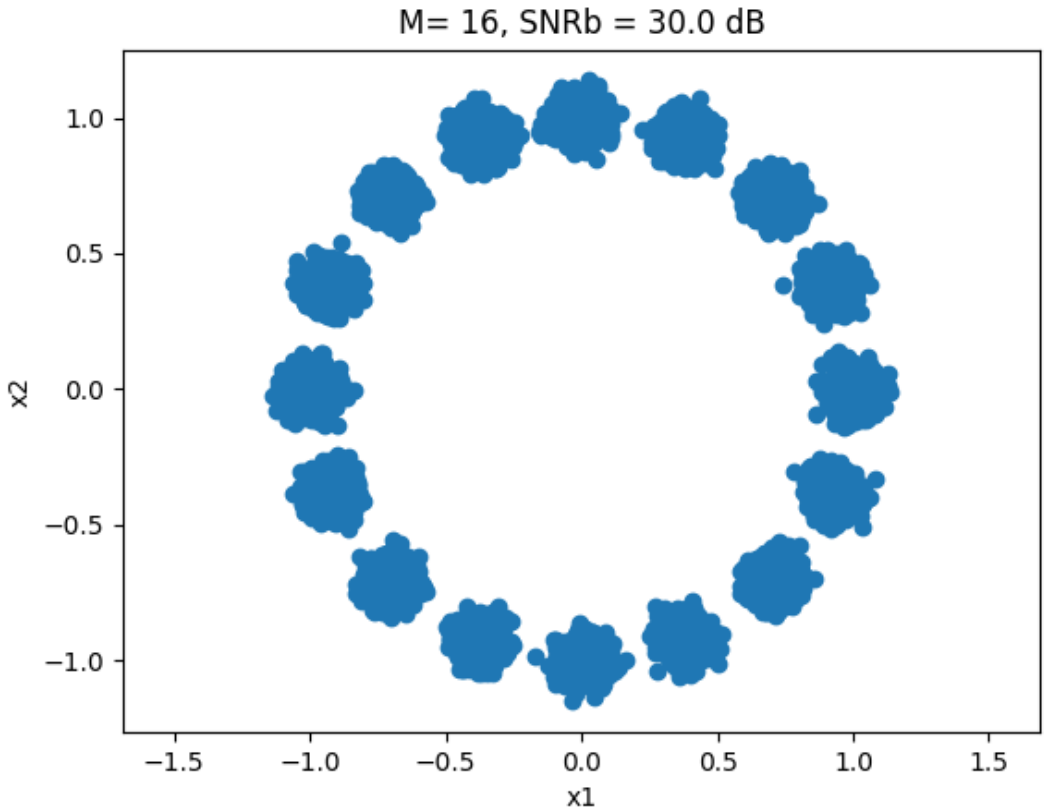
Προσομοίωση του PSK - commlib.py

```
class psk_simulation(monte_carlo):

    def __init__(self, M = 16, SNRbdB = 10,
       max_symbol_errors = 100, **kwargs):

        super().__init__( **kwargs )
        self.M = M
        self.m = np.log2(M).astype(int)
        self.SNRbdB = SNRbdB
        self.SNRb = 10 ** (SNRbdB / 10)
        self.constellation = psk_constellation(M,
       SNRbdB = SNRbdB)
        self.N0 = 1 / self.SNRb / np.log2(M)
        self.sigma = np.sqrt(self.N0 / 2)
        self.symbol_errors = 0
        self.bit_errors = 0
        self.max_symbol_errors =
       max_symbol_errors
        self.realizations = np.zeros( self.
       max_iterations, dtype = complex )

    def generate(self):
        bits = random_bits( self.m )
        self.symbol = self.constellation.
       bits_to_symbols( bits )
        self.noise = self.sigma * np.random.randn
       ( 1 ) + 1j * self.sigma * np.random.randn
       (1)
        self.input_bits = bits

    def apply(self):
        self.output = self.symbol + self.noise
       output )

    def measure(self):
        self.bit_errors += np.sum( np.abs(self.
       decoded_bits - self.input_bits) ).astype(
       int)
        self.symbol_errors += int(self.symbol !=
       self.decoded_symbol)
#---pskmeasure
    def terminate(self):
        return self.symbol_errors >= self.
       max_symbol_errors

    def append_to_realizations(self):
        self.realizations[self.ci] = self.output
       [0]

    def report_iteration(self,i):
        if i != 0:
            ser = self.symbol_errors / i
            print('iteration %d / %d, symbol
       errors = %d, SER = %e' %(i, self.
       max_iterations, self.symbol_errors, ser) )

    def plot_constellation(self):
        plt.figure()
        self.plot_realizations()
        plt.xlabel('x1')
        plt.ylabel('x2')
        plt.title('M= %d, SNRb = %2.1f dB' %(self
       .M, self.SNRbdB))
        plt.axis('equal')
```

Προσομοίωση του PSK - pskconst.py

```
1  import commlib as cl
2  import matplotlib.pyplot as plt
3
4  M = 4
5  s = cl.psk_simulation(M = M, SNRbdB = 20, keep_realizations = True, max_iterations = 10000,
6                        max_symbol_errors = 10000)
7  s.execute()
8  plt.close('all')
9  s.plot_constellation()
10
11 s = cl.psk_simulation(M = 16, SNRbdB = 30, keep_realizations = True, max_iterations = 10000,
12                       max_symbol_errors = 10000)
13 s.execute()
14 s.plot_constellation()
```


M= 4, SNRb = 20.0 dB
x2
x1
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5


M= 16, SNRb = 30.0 dB
x2
x1
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

Ο δέκτης PSK

- Όπως και στο PAM, ο δέκτης ψάχνει να βρει το πλησιέστερο σύμβολο

$$Y_k = \exp(j\Phi_k) = Y_{k1} + jY_{k2}$$

στο $y = y_1 + jy_2$ όπου:

$$y_1 =< y(t), p_1(t) >= x_1 + n_1 = \cos(\Phi_m) + n_1$$

$$y_2 =< y(t), p_2(t) >= x_2 + n_2 = \sin(\Phi_m) + n_2$$

- με $\Phi_k = 2\pi(k-1)/M$ και όπως είδαμε και στην προηγούμενη διάλεξη τα n_1 και n_2 είναι ανεξάρτητες Gaussian μεταβλητές με μηδενική μέση τιμή και διακύμανση:

$$\mathbb{E}\left\{n_1^2\right\} = \mathbb{E}\left\{n_2^2\right\} = \frac{N_0}{2}$$

Ο δέκτης PSK

- Η απόσταση του y από ένα σύμβολο Y_k είναι:

$$|y-Y_k|^2 = (y_1-Y_{k1})^2+(y_2-Y_{k2})^2 = y_1^2+y_2^2+Y_{k1}^2+Y_{k2}^2-2(y_1Y_{k1}+y_2Y_{k2}) = y_1^2+y_2^2+|Y_k|^2-2|y||Y_k|\cos(\phi-\Phi_k) = y_1^2+y_2^2+1-2|y|\cos(\phi-\Phi_k)$$

- όπου ϕ είναι το όρισμα του μιγαδικού αριθμού y:

$$y = |y|\mathrm{e}^{j\phi}$$

- Επομένως για δεδομένο $y = y_1 + jy_2$ το σύμβολο Y_k το οποίο είναι πλησιέστερα στο y είναι αυτό που έχει το *μεγαλύτερο*

$$\cos(\phi-\Phi_k) = \cos\left(\phi-\frac{2\pi(k-1)}{M}\right)$$

Ο δέκτης PSK

- Αν υποθέσουμε ότι ξεκινούμε από το σύμβολο

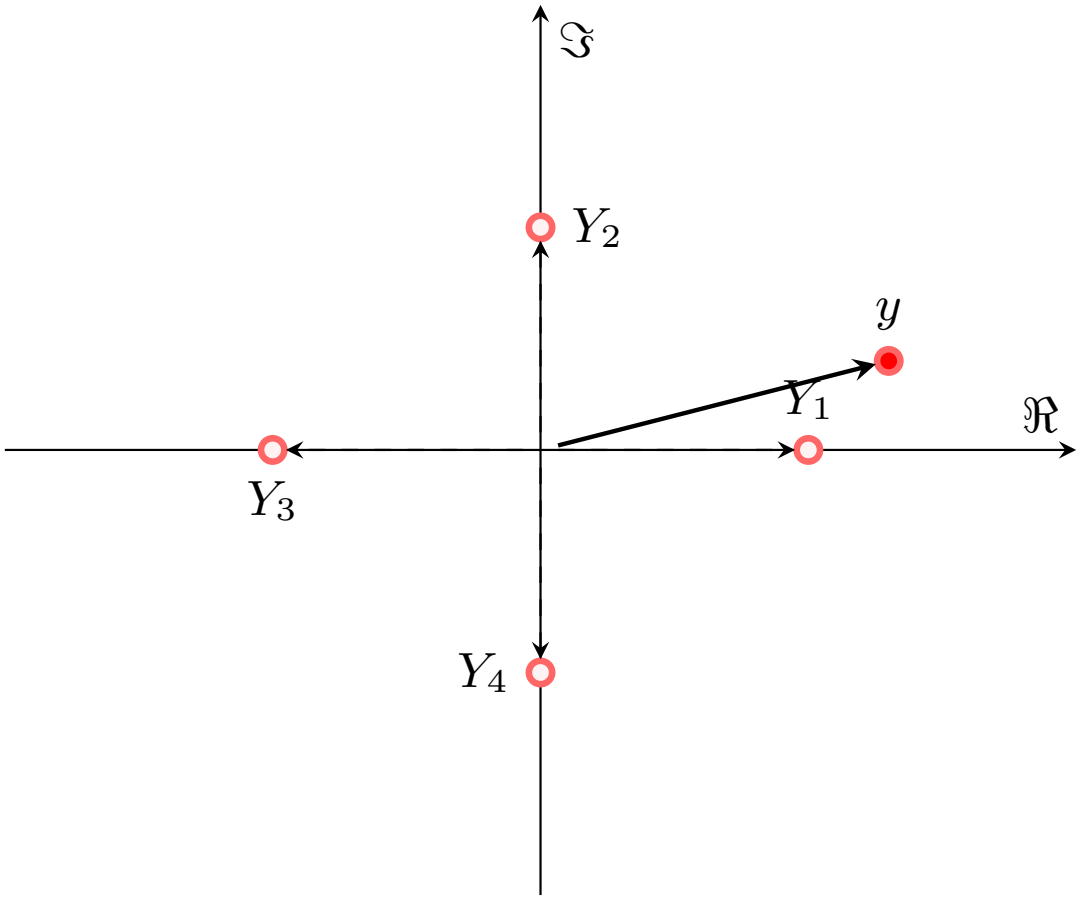
$$x = \exp(j\Phi_0) = 1$$

- Τότε θα έχουμε:

$$y = 1 + n \tag{1}$$

- όπου $n = n_1 + jn_2$.
- το ϕ είναι το όρισμα του y. Για να είναι σωστή η αποκωδικοποίηση θα πρέπει το $\cos(\phi)$ να είναι μεγαλύτερο από όλα τα άλλα $\cos\left(\phi - \frac{2\pi(k-1)}{M}\right)$

ο δέκτης PSK



Σχήμα: Ο αστερισμός του 4-PSK

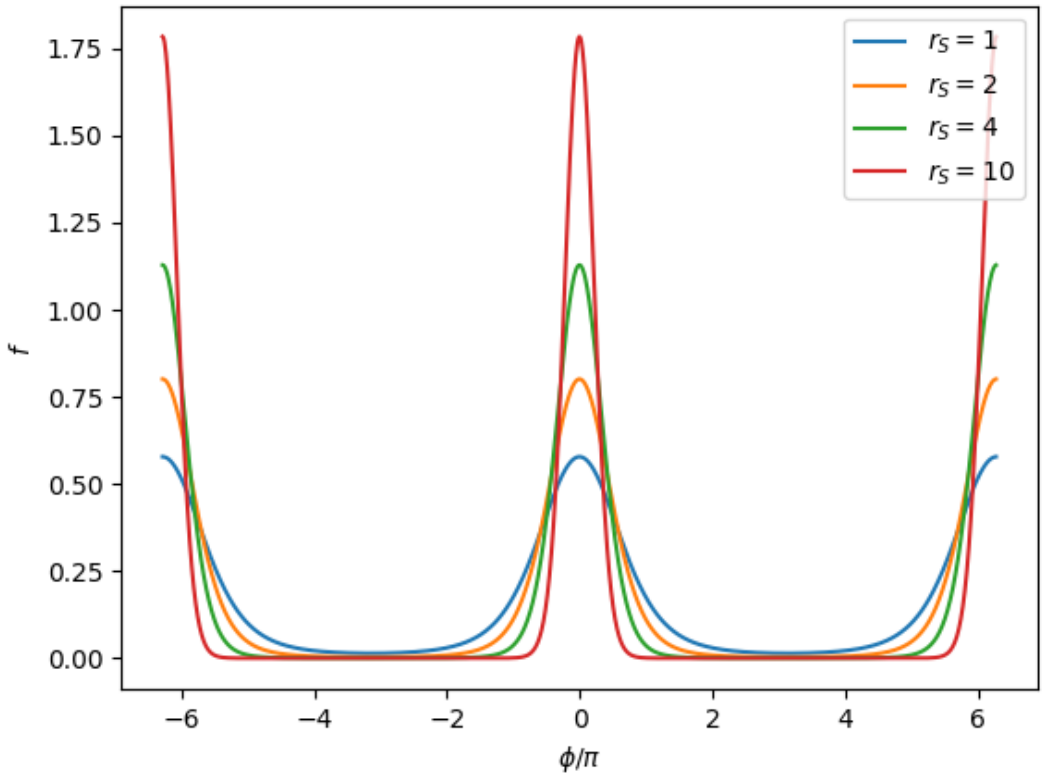
Στατιστική των σφαλμάτων

- Οι φάσεις $2\pi(k-1)/M$ χωρίζουν το διάστημα $[0, 2\pi]$ σε M τμήματα. Είναι φανερό ότι για να είναι μεγαλύτερο το $\cos(\phi)$ σε σχέση με τα άλλα συνημίτονα θα πρέπει:
- $0 \leq \phi \leq \frac{\pi}{M}$ ή
- $2\pi - \frac{\pi}{M} \leq \phi \leq 2\pi$
- Η πυκνότητα πιθανότητας του ϕ είναι:

$$f(\phi) = \frac{1}{2\pi} \mathrm{e}^{-\rho_S \sin^2(\phi)} \int_0^{+\infty} v \mathrm{e}^{-(v-\sqrt{2\rho_S}\cos(\phi))^2/2} \mathrm{d}v$$

- οπου το r_S είναι το πηλίκο σήμα-προς-θόρυβο.

Στατιστική των σφαλμάτων



Στατιστική των σφαλμάτων

- Παρατηρούμε ότι η συνάρτηση $f(\phi)$ είναι περιοδική, δηλαδή $f(\phi + 2\pi) = f(\phi)$.
- Επομένως:

$$P_{\mathrm{e}} = 1 - \int_0^{\frac{\pi}{M}} f(\phi)\mathrm{d}\phi - \int_{2\pi-\frac{\pi}{M}}^{2\pi} f(\phi)\mathrm{d}\phi = 1 - \int_0^{\frac{\pi}{M}} f(\phi)\mathrm{d}\phi - \int_{-\frac{\pi}{M}}^{0} f(\phi)\mathrm{d}\phi =$$
$$\int_{-\pi}^{-\frac{\pi}{M}} f(\phi)\mathrm{d}\phi + \int_{\frac{\pi}{M}}^{\pi} f(\phi)\mathrm{d}\phi = 2\int_{\frac{\pi}{M}}^{\pi} f(\phi)\mathrm{d}\phi$$

- Μπορούμε να υπολογίσουμε την πιθανότητα σφάλματος συμβόλου με αριθμητική ολοκλήρωση.
- Η πιθανότητα σφάλματος συμβόλου θα είναι $P_{\mathrm{b}} \cong P_{\mathrm{e}} / \log_2 M$.

Στατιστική των σφαλμάτων - commlib.py

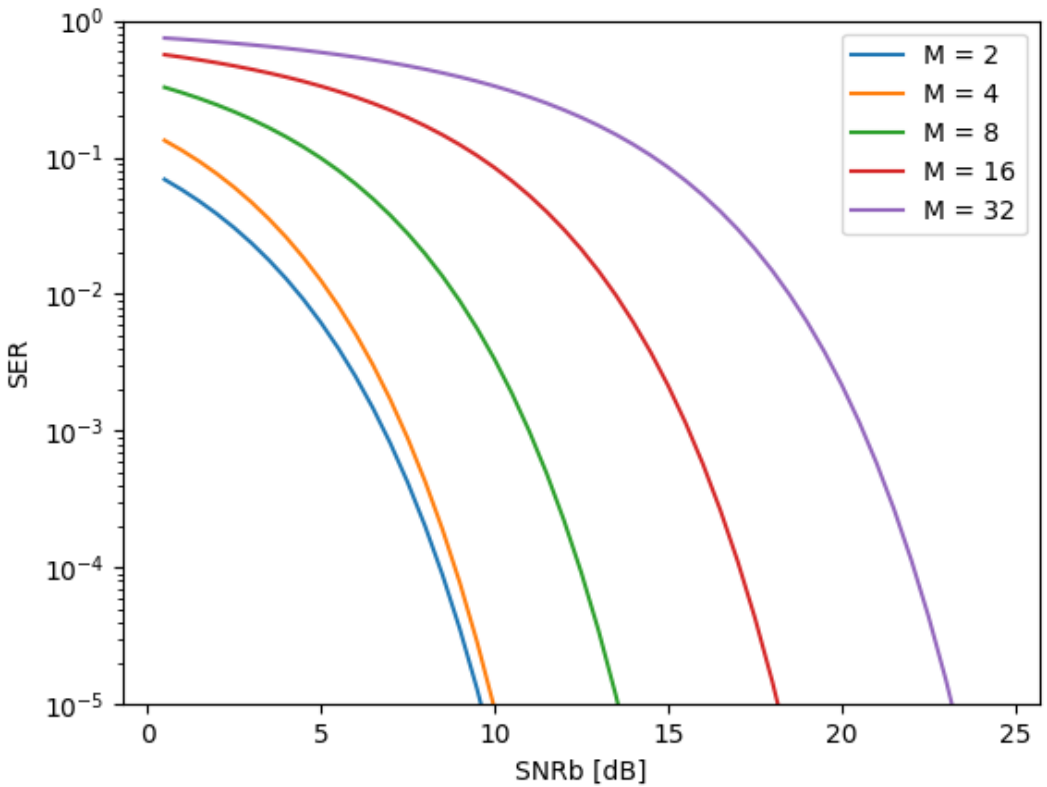
```
class psk_constellation(constellation):

    def __init__(self, M, R = 1,title = None,
       SNRbdB = 10):
        super().__init__(title = title)

        self.M = M
        self.m = np.log2(M).astype(int)
        self.SNRbdB = SNRbdB
        self.R = R
        self.SNRb = 10 ** (SNRbdB/10)
        self.SNRS = self.SNRb * self.m
        self.vmax = 100
        self.Nphi = 1000

        symbols = np.zeros( M, dtype = complex )
        for i in range( M ):
            symbols [ i ] = R * np.exp( 1j * 2 *
      np.pi / M * i )

        self.set_symbols( symbols )
        self.set_gray_bits( self.m )

    def fphi(self, phis):
        rS = self.SNRS
        v = np.arange(0, self.vmax, self.vmax /
      self.Nphi)
        f = np.zeros(phis.size)

        for i, phi in enumerate(phis):
            g = v * np.exp( -0.5 * ( v - np.sqrt
      (2*rS) * np.cos(phi) ) ** 2.0 )
            self.v = v
            self.g = g
            f[i] = 1/(2*np.pi) * np.exp( -rS*np.
      sin(phi) ** 2.0) * np.trapz(g,v)

        return f

    def ser(self, Npoints = 100):

        phi = np.arange(-np.pi / self.M, np.pi /
      self.M, 2 * np.pi / self.M / Npoints)
        fphi = self.fphi(phi)
        return 1 - np.trapz(fphi, phi)

    def ber(self, Npoints = 1000):
        return self.ser(Npoints = Npoints) / self
      .m
```

Στατιστική των σφαλμάτων - pskservsM.py

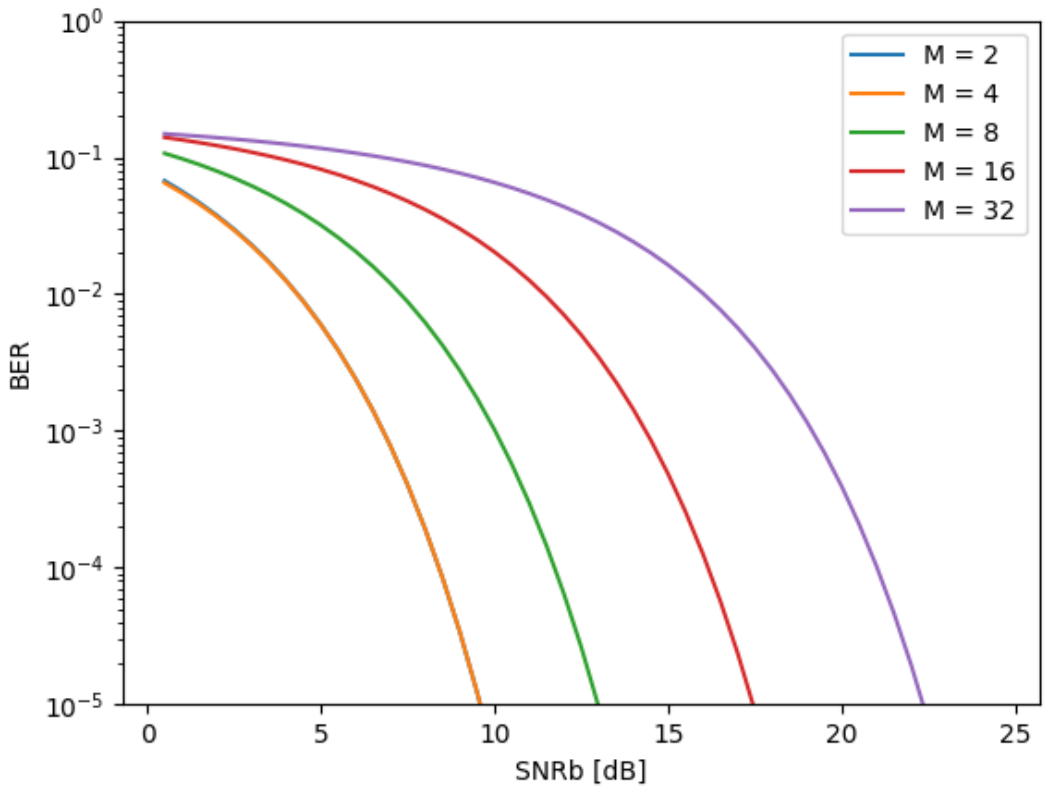
```
from commlib import psk_constellation
import matplotlib.pyplot as plt
import numpy as np

SNRbdBs = np.arange(0.5, 25, 0.5)
n = np.arange(1,7,1)
Ms = np.array([2, 4, 8, 16, 32])
Ms = Ms.astype(int)

Pest = np.zeros( [SNRbdBs.size, Ms.size] )
Pebt = np.zeros( [SNRbdBs.size, Ms.size] )

threshold = 1e-4

for i, SNRbdB in enumerate(SNRbdBs):
    for j, M in enumerate(Ms):
        c = psk_constellation(M = M, SNRbdB =
       SNRbdB)
        Pest[i,j] = c.ser()
        Pebt[i,j] = c.ber()
        print('M = %d, SNRbdB = %6.2f Pe = %e' %
       (M,SNRbdB, Pest[i,j]))
```

```
plt.close('all')

for j, M in enumerate(Ms):
    plt.figure(1)
    plt.semilogy( SNRbdBs, Pest[:,j], label = 'M
       = %d' % M)
    plt.figure(2)
    plt.semilogy( SNRbdBs, Pebt[:,j], label = 'M
       = %d' % M)

plt.figure(1)
plt.xlabel('SNRb [dB]')
plt.ylabel('SER')
plt.legend()
plt.ylim([1e-5, 1])

plt.figure(2)
plt.xlabel('SNRb [dB]')
plt.ylabel('BER')
plt.legend()
plt.ylim([1e-5, 1])
```

Επιδόσεις του PSK



Επιδόσεις του PSK



Επιδόσεις του PSK

- Το P_b του 4-PSK είναι το ίδιο με το 2-PSK.
- Αυτό συμβαίνει επειδή το 2-PSK είναι στην ουσία το ίδιο με το 2-PAM. Τα σύμβολα είναι πάνω στον πραγματικό άξονα (± 1).
- Το 4-PSK είναι σαν να έχουμε δύο συστήματα 2-PAM: Ένα στον πραγματικό άξονα (± 1) και ένα στον φανταστικό άξονα ($\pm j$).
- Στην περίπτωση του 2-PSK, η πιθανότητα σφάλματος μπορεί να υπολογιστεί από τον τύπο του 2-PAM με $\beta = 1$:

$$P_{\text{e}2} = Q\left(\frac{\beta}{\sigma}\right) = Q\left(\frac{2}{N_0}\right)$$

- Στην περίπτωση του 4-PSK, για να συμβεί σφάλμα πρέπει να κάνουμε λάθος στο PAM του πραγματικού άξονα, στο PAM του φανταστικού άξονα ή και στα δύο PAM:

$$P_{\text{e}4} = P_{\text{e}2}(1 - P_{\text{e}2}) + (1 - P_{\text{e}2})P_{\text{e}2} + P_{\text{e}2}^2 = 2P_{\text{e}2} - P_{\text{e}2}^2$$

Επιδόσεις του PSK

- Αν το P_{e2} είναι μικρό, τότε

$$P_{e4} = 2P_{e2} - P_{e2}^2 \cong 2P_{e2}$$

- ενώ η πιθανότητα σφάλματος *bit* θα είναι:

$$P_{b4} \cong \frac{P_{e4}}{2} \cong \frac{2P_{e2}}{2} = P_{b2}$$

- Επομένως οι πιθανότητες σφάλματος bit είναι οι ίδιες για τα δύο συστήματα!
- Ωστόσο το 4-PSK υπερέχει ξεκάθαρα καθώς μεταδίδει 2 φορές περισσότερα bit από ότι το 2-PSK.

Θεωρία vs simuation - commlib.py

```
class psk_simulation(monte_carlo):

    def __init__(self, M = 16, SNRbdB = 10,
       max_symbol_errors = 100, **kwargs):

        super().__init__( **kwargs )
        self.M = M
        self.m = np.log2(M).astype(int)
        self.SNRbdB = SNRbdB
        self.SNRb = 10 ** (SNRbdB / 10)
        self.constellation = psk_constellation(M,
       SNRbdB = SNRbdB)
        self.N0 = 1 / self.SNRb / np.log2(M)
        self.sigma = np.sqrt(self.N0 / 2)
        self.symbol_errors = 0
        self.bit_errors = 0
        self.max_symbol_errors =
       max_symbol_errors
        self.realizations = np.zeros( self.
       max_iterations, dtype = complex )

    def generate(self):
        bits = random_bits( self.m )
        self.symbol = self.constellation.
       bits_to_symbols( bits )
        self.noise = self.sigma * np.random.randn
       ( 1 ) + 1j * self.sigma * np.random.randn
       (1)
        self.input_bits = bits

    def apply(self):
        self.output = self.symbol + self.noise
        [self.decoded_symbol, self.decoded_bits,
       _ ] = self.constellation.decode( self.
       output )

    def measure(self):
        self.bit_errors += np.sum( np.abs(self.
       decoded_bits - self.input_bits) ).astype(
       int)
        self.symbol_errors += int(self.symbol !=
       self.decoded_symbol)
```

Θεωρία vs simuation - commlib.py

```
    def terminate(self):
        return self.symbol_errors >= self.
       max_symbol_errors

    def append_to_realizations(self):
        self.realizations[self.ci] = self.output
       [0]

    def report_iteration(self,i):
        if i != 0:
            ser = self.symbol_errors / i
            print('iteration %d / %d, symbol
```

```
          errors = %d, SER = %e' %(i, self.
          max_iterations, self.symbol_errors, ser) )

    def plot_constellation(self):
        plt.figure()
        self.plot_realizations()
        plt.xlabel('x1')
        plt.ylabel('x2')
        plt.title('M= %d, SNRb = %2.1f dB' %(self
       .M, self.SNRbdB))
        plt.axis('equal')
```

Θεωρία vs simuation - pskber.py

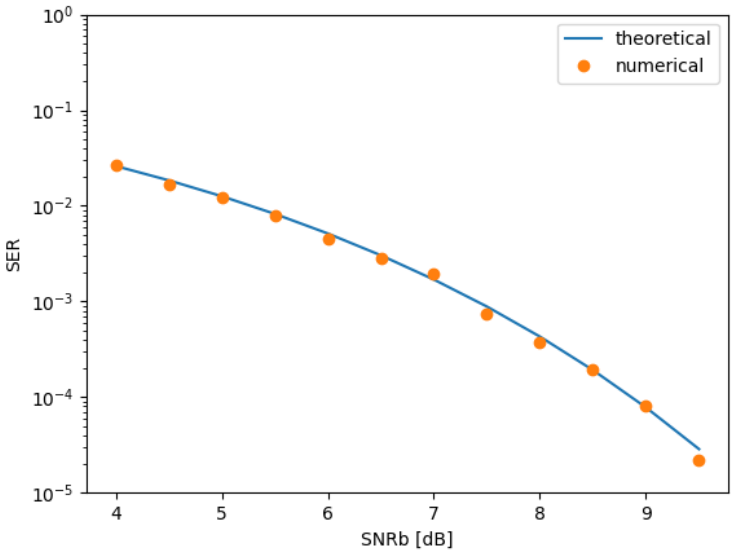
```
from commlib import psk_simulation,
      psk_constellation
import matplotlib.pyplot as plt
import numpy as np

SNRbdBs = np.arange(4, 18, 0.5)
M = 16

Pes = np.zeros( SNRbdBs.size )
Peb = np.zeros( SNRbdBs.size )
Pest = np.zeros( SNRbdBs.size )
Pebt = np.zeros( SNRbdBs.size )

threshold = 1e-5

for i, SNRbdB in enumerate(SNRbdBs):

    s = psk_simulation(max_iterations = int(1e6),
                       M = M, SNRbdB = SNRbdB,
      report_step = 100000,
                       keep_realizations = False,
                       max_symbol_errors = 100,
                       report = True)
    Pest[i] = s.constellation.ser()
    c = psk_constellation(M = M, SNRbdB = SNRbdB)
    Pest[i] = c.ser()
    Pebt[i] = c.ber()
    s.execute()
    Pes[i] = s.symbol_errors / s.
       iterations_performed
    Peb[i] = s.bit_errors / s.
       iterations_performed / np.log2(M)
    if Pes[i] < threshold:
        break
    print('M = %d, SNRbdB = %6.2f Pe = %e / %e' %
        (M,SNRbdB, Pes[i], Pest[i]))
    print('M = %d, SNRbdB = %6.2f Pe = %e' % (M,
       SNRbdB, Pest[i]))
```

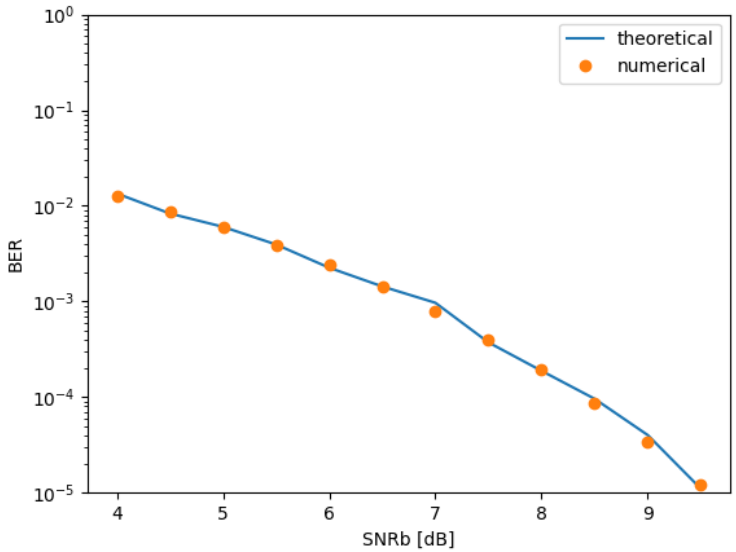
Θεωρία vs simuation - pskber.py

```
plt.close('all')

plt.figure(1)
plt.semilogy( SNRbdBs, Pest, label = 'theoretical
       ')
plt.semilogy( SNRbdBs, Pes, 'o', label = '
       numerical')

plt.figure(2)
plt.semilogy( SNRbdBs, Peb, label = 'theoretical'
       )
plt.semilogy( SNRbdBs, Pebt, 'o', label = '
       numerical')
```

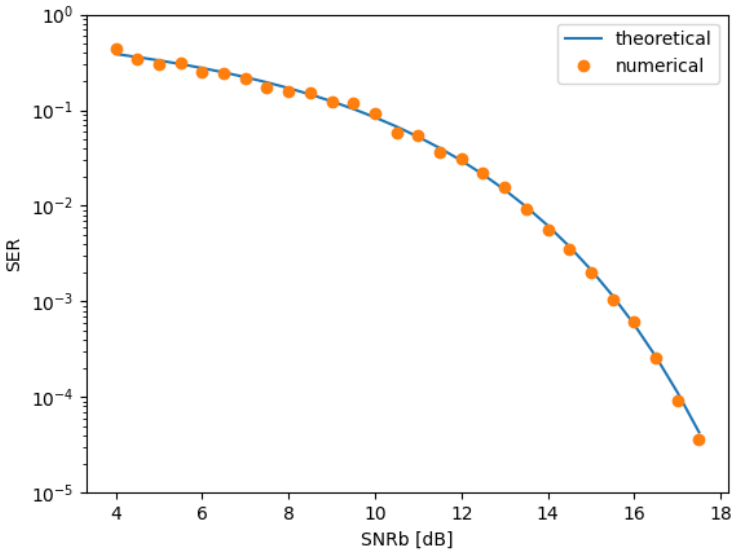
```
plt.figure(1)
plt.xlabel('SNRb [dB]')
plt.ylabel('SER')
plt.legend()
plt.ylim([1e-5, 1])

plt.figure(2)
plt.xlabel('SNRb [dB]')
plt.ylabel('BER')
plt.legend()
plt.ylim([1e-5, 1])
```

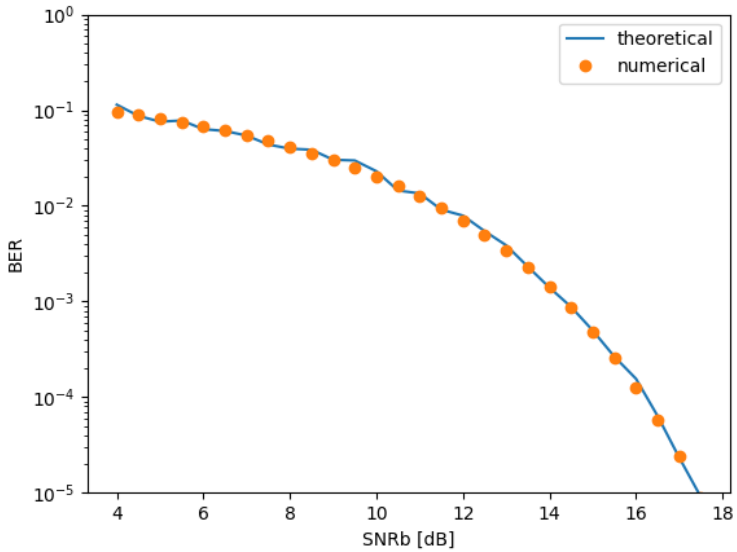
Θεωρία vs simulation - $M = 4$





Θεωρία vs simulation - $M = 16$



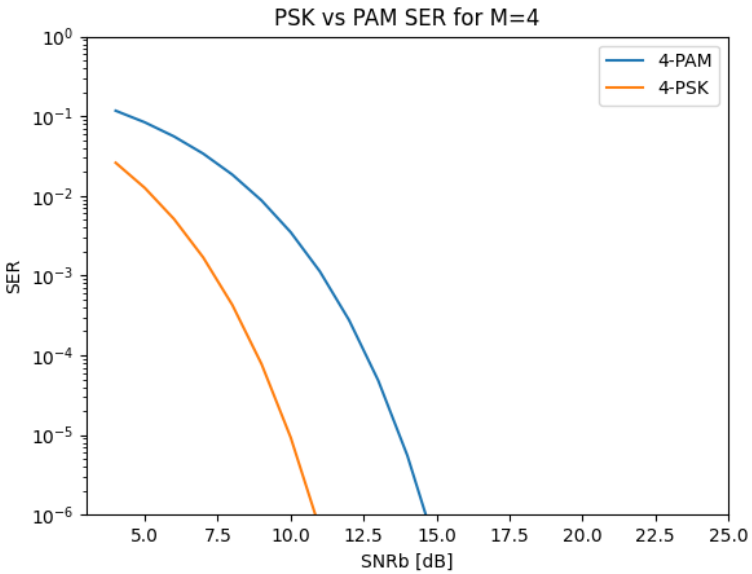


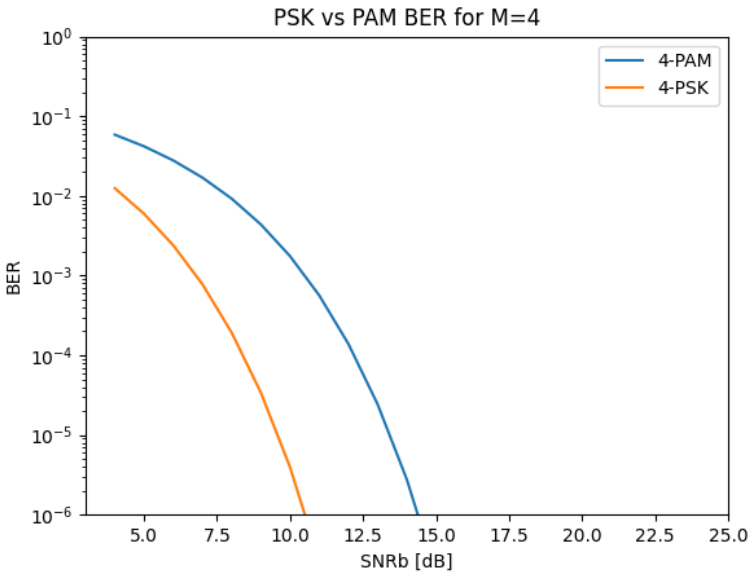
PAM vs PSK - pamvspsk.py

```
from commlib import pam_constellation,
       psk_constellation
import matplotlib.pyplot as plt
import numpy as np

SNRbdBs = np.arange(4, 25, 1)
M = 4

PesPAM = np.zeros( SNRbdBs.size )
PebPAM = np.zeros( SNRbdBs.size )
PesPSK = np.zeros( SNRbdBs.size )
PebPSK = np.zeros( SNRbdBs.size )

for i, SNRbdB in enumerate(SNRbdBs):

    cpsk = psk_constellation(M, SNRbdB = SNRbdB)
    cpam = pam_constellation(M, SNRbdB = SNRbdB)

    PesPAM[i] = cpam.ser()
    PesPSK[i] = cpsk.ser()
    PebPAM[i] = cpam.ber()
    PebPSK[i] = cpsk.ber()

plt.close('all')
```

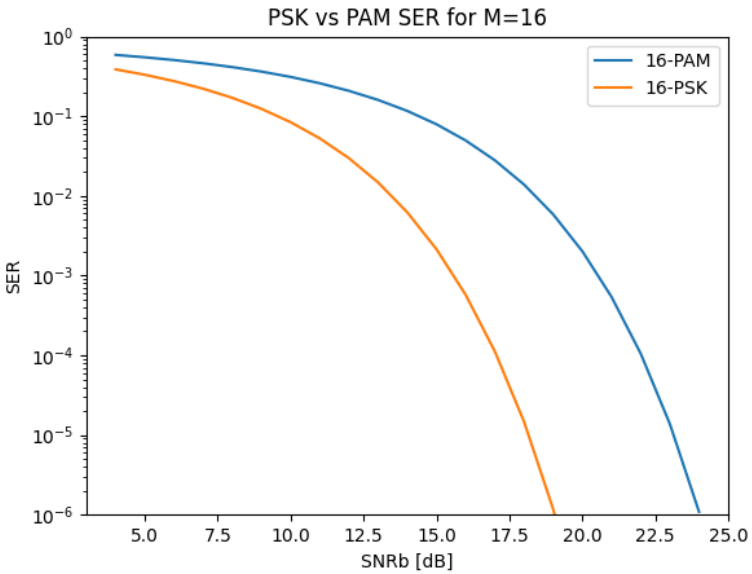
```
plt.figure(1)
plt.semilogy( SNRbdBs, PesPAM, label = '%d-PAM' %
       M)
plt.semilogy( SNRbdBs, PesPSK, label = '%d-PSK' %
       M)
plt.xlabel('SNRb [dB]')
plt.ylabel('SER')
plt.legend()
plt.title('PSK vs PAM SER for M=%d' %M)
plt.ylim([1e-6, 1])

plt.figure(2)
plt.semilogy( SNRbdBs, PebPAM, label = '%d-PAM' %
       M)
plt.semilogy( SNRbdBs, PebPSK, label = '%d-PSK' %
       M)
plt.xlabel('SNRb [dB]')
plt.ylabel('BER')
plt.legend()
plt.title('PSK vs PAM BER for M=%d' %M)
plt.ylim([1e-6, 1])
```

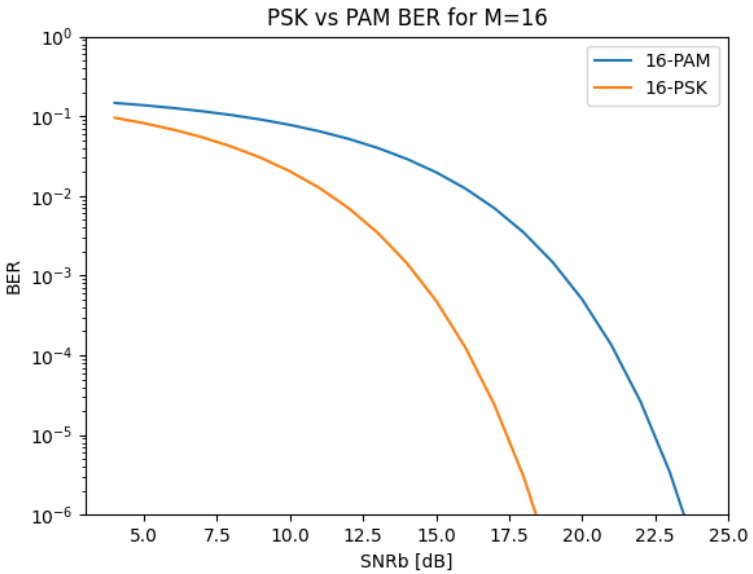
PAM vs PSK, $M = 4$


PSK vs PAM SER for M=4
4-PAM
4-PSK
SER
SNRb [dB]
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
5.0
7.5
10.0
12.5
15.0
17.5
20.0
22.5
25.0


PSK vs PAM BER for M=4
4-PAM
4-PSK
BER
SNRb [dB]
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
5.0
7.5
10.0
12.5
15.0
17.5
20.0
22.5
25.0

PAM vs PSK, $M = 16$





Το φάσμα του PSK

- Η κυματομορφή PSK γράφεται ως εξής:

$$Y(t) = Q(t) - I(t)$$

$$Q(t) = \sqrt{\frac{2}{\mathcal{E}_p}} \cos(2\pi f_0 t) \sum_k \cos(\phi_k) p(t - kT_S) = \sum_k c_k p_1(t - kT_S)$$

$$I(t) = \sqrt{\frac{2}{\mathcal{E}_p}} \sin(2\pi f_0 t) \sum_k \sin(\phi_k) p(t - kT_S) = \sum_k s_k p_2(t - kT_S)$$

- Η συνάρτηση συσχέτισης γράφεται:

$$R_{YY}(t, t+\tau) = \mathbb{E}\{Y(t)Y(t+\tau)\} =$$
$$\mathbb{E}\{Q(t)Q(t+\tau)\} - \mathbb{E}\{I(t)I(t+\tau)\} - \mathbb{E}\{Q(t)I(t+\tau)\} + \mathbb{E}\{I(t)Q(t+\tau)\}$$

Το φάσμα του PSK

- Οι αναμενόμενες τιμές που έχουν τα γινόμενα του I με το Q υπολογίζονται ως εξής:

$$\mathbb{E}\{Q(t)I(t+\tau)\} = \sum_{kl} \mathbb{E}\{c_k s_l\}\, p_1(t-kT_S)p_2(t+\tau-lT_S)$$

- Δεδομένου ότι $c_k = \cos\phi_k$ και $s_l = \sin\phi_l$ θα έχουμε εάν $k \neq l$:

$$\mathbb{E}\{c_k s_l\} = \mathbb{E}\{c_k\}\,\mathbb{E}\{s_l\} = 0$$

- Ενώ:

$$\mathbb{E}\{c_k s_k\} = \frac{1}{M}\sum_k \cos(\phi_k)\sin(\phi_k) = \frac{1}{2M}\sum_k \sin(2\phi_k) = 0$$

- Άρα τελικά:

$$\mathbb{E}\{Q(t)I(t+\tau)\} = \mathbb{E}\{Q(t+\tau)I(t)\} = 0$$

Το φάσμα του PSK

- Επομένως η συνάρτηση αυτοσυσχέτισης του PSK γράφεται:

$$R_{YY}(t, t+\tau) = R_{QQ}(t, t+\tau) + R_{II}(t, t+\tau)$$

- και η φασματική πυκνότητα ισχύος του PSK είναι:

$$S_Y(f) = S_Q(f) + S_I(f)$$

- όπου $S_Q(f)$ και $S_I(f)$ οι φασματικές πυκνότητες ισχύος των $Q(t)$ και $I(t)$.
- τα $Q(t)$ και $I(t)$ είναι κυματομορφές ζωνοπερατών PAM με σύμβολα c_k και s_k που έχουν μηδενική μέση τιμή, $\mathbb{E}\{c_k\} = \mathbb{E}\{s_k\} = 0$.
- στην περίπτωση αυτή θα εχουμε:

$$S_Q(f) = \frac{\mathbb{E}\{c_k^2\}}{4T_S}\left(|P(f-f_0)|^2 + |P(f+f_0)|^2\right)$$

Το φάσμα του PSK

- και μία παρόμοια σχέση ισχύει για το $S_I(f)$:

$$S_I(f) = \frac{\mathbb{E}\left\{s_k^2\right\}}{4T_\mathrm{S}} \left(|P(f-f_0)|^2 + |P(f+f_0)|^2\right)$$

- και τελικά:

$$S_Y(f) = \frac{\mathbb{E}\left\{c_k^2 + s_k^2\right\}}{4T_\mathrm{S}} \left(|P(f-f_0)|^2 + |P(f+f_0)|^2\right) \qquad (2)$$

- οπότε το φάσμα του PSK είναι το ίδιο με αυτό του PAM!
- Για τετραγωνικούς παλμούς το εύρος ζώνης θα είναι $B = 2/T_\mathrm{S}$
- Η φασματική απόδοση θα είναι $\frac{1}{2}\log_2 M$ bit/s/Hz.