Μάθημα 9: Η βιβλιοθήκη commlib

Θωμάς Καμαλάκης

16 Σεπτεμβρίου 2023

1 Εισαγωγή

Στα προηγούμενα μαθήματα παρουσιάσαμε ορισμένα μαθηματικά εργαλεία που θα χρησιμοποιήσουμε στη συνέχεια στην ανάλυση των συστημάτων επικοινωνιών. Επίσης είδαμε και μερικά παραδείγματα όπου χρησιμοποιήσαμε την `Python` για να ερμηνεύσουμε ή επιβεβαιώσουμε τη θεωρητική μας προσέγγιση. Αν δείτε τα listings που παρουσιάσαμε μέχρι τώρα θα καταλάβετε ότι τμήματα του κώδικα επαναλαμβανόντουσαν ξανά και ξανά κάτι που αντιβαίνει σε μία αρχή του καλού προγραμματισμού: μην επαναλαμβάνεσαι (do not repeat yourself - DRY).

Θα πρέπει με κάποιο τρόπο να αποφύγουμε την άσκοπη επανάληψη η οποία πολύ συχνά οδηγεί σε σφάλματα και αρκετές ώρες όπου κοιτάμε την οθόνη και δεν καταλαβαίνουμε τι έχει πάει στραβά. Ιδανικά θα θέλαμε να σπάσουμε μία φορά το κεφάλι μας να φτιάξουμε κάτι και από εκεί και πέρα να το χρησιμοποιούμε ως "μαύρο κουτί". Και μία που μιλάμε για μαύρα κουτιά, θα ήταν καλύτερο να φτιάξουμε δικούς μας τύπους δεδομένων που να αντιπροσωπεύουν με βέλτιστο τρόπο το τι μελετάμε. Πάρτε για παράδειγμα ένα σήμα. Μέχρι τώρα χρησιμοποιούμε ένα `numpy array` για το χρόνο `t` και ένα `array` για τα δείγματα του σήματος τις χρονικές στιγμές που περιέχει το `t`. Επίσης έχουμε μία σειρά από συναρτήσεις που κάνουν κάτι χρήσιμο βασισμένα στα `t` και `x`, π.χ. υπολογίζουν το φάσμα κτλ. Δεν θα ήταν καλύτερο να είχαμε έναν τύπο δεδομένων που να τον λέμε `signal` που μεταξύ άλλων να περιέχει το `t` και `x` και όλες οι απαραίτητες επεξεργασίες να γίνονται απευθείας σε αυτόν. Για να το πούμε με μία ορολογία που είναι εξοικειωμένοι όσοι σπουδάζουν πληροφορική θα πρέπει να υιοθετήσουμε μία *αντικειμενοστραφή* προσέγγιση και να υλοποιήσουμε τις κατάλληλες *κλάσεις* όπως η `signal`.

Δεν είναι ανάγκη να πετάξουμε ότι έχουμε κάνει μέχρι τώρα καθώς ορισμένα μέρη είναι πολύ χρήσιμα. Ας μαζέψουμε ότι χρήσιμο υπάρχει ένα αρχείο `utils.py` το οποίο θα συμπεριλαμβάνουμε στο αρχείο που ορίζουμε τις διάφορες κλάσεις `commlib.py`.

2 Η βιβλιοθήκη `utils.py`

Η βιβλιοθήκη `utils.py` φαίνεται στο listing 1. Περιλαμβάνει βασικές συναρτήσεις που έχουν να κάνουν με υπολογισμούς σημάτων και στατιστικών κατανομών. Όπως είπαμε και παραπάνω στην ουσία το `utils.py` περιλαμβάνει συναρτήσεις που είχαμε δει στα προηγούμενα μαθήματα και έχει διαδικαστική (procedural) λογική: δεν χρησιμοποιούνται κλάσεις και επομένως δεν ορίζονται νέοι τύποι δεδομένων. Αποτελεί μία συλλογή από χρήσιμα εργαλεία που θα μας κάνουν λίγο πιο εύκολη τη ζωή στην `commlib.py`

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  from scipy.special import comb, erfc
4
5  plt.rcParams.update({
6      "text.usetex": True,
7      "font.family": "serif",
8      "font.serif": ["Palatino"],
9      "font.size" : 14,
10     "lines.linewidth" : 2,
11 })
12
13
14 # Basic signals
15 def carrier(t, f0, A, phi = 0):
16     return A * np.cos( 2 * np.pi * f0 * t + phi )
17
18 def square_pulse(t, T1, A = 1):
19     samples = np.zeros(t.size)
20     i = np.where( np.abs(t) <= 0.5 * T1)
```

```
    samples[ i ] = A
    return samples

def step_signal(t):
    samples = np.zeros(t.size)
    i = np.where( t>=0 )
    samples[ i ] = 1
    return samples

# Digital signal representation
def to_amps(m, b):
    q = len( list(m.keys()) [0] )
    bs = [ b[i:i+q] for i in range(0, len(b), q) ]
    vals = [ m[g] for g in bs ]
    return np.array(vals)

def dig_waveform(amps, t, TS):
    Namps = amps.size
    x = np.zeros(t.size)
    k = np.floor( t/TS + 0.5 ).astype(int)
    u = np.where( (k>=0) & (k<Namps) )
    x[ u ] = amps[ k[u] ]
    return x

# Signal utilities
def inst_power(x):
    return np.abs(x) ** 2

def energy(t, x):
    return np.trapz(inst_power(x), t)

def avg_power(t, x):
    ta = min(t)
    tb = max(t)
    return energy(t, x) / (tb - ta)

def calc_spectrum(t, x):
    Dt = t[1] - t[0]
    N = t.size
    Df = 1 / Dt / N
    f = np.arange( -N / 2, N / 2 ) * Df
    X = Dt * np.fft.fftshift(
              np.fft.fft(
              np.fft.fftshift(x) ) )
    return f, X

def calc_inv_spectrum(f, X):
    Df = f[1] - f[0]
    N = f.size
    Dt = 1 / Df / N
    t = np.arange( -N / 2, N/2 ) * Dt
    x = 1 / Dt * np.fft.fftshift(
              np.fft.ifft(
              np.fft.fftshift(X) ) )
    return t, x

# System output
def system_output(t, x, H):
    X, f = calc_spectrum(x, t)
    Y = X * H(f)
    return calc_inv_spectrum(Y, f)


# distibutions
def binomial(q, Q, p):
    return comb(Q, q) * p ** q * (1-p) ** (Q-q)

# PDF estimation
def calc_pdfcdf(x, y):
    N = x.size
    Dy = y[1] - y[0]
    pdf = np.zeros( y.size )
    cdf = np.zeros( y.size )

    for i, yy in enumerate(y):
        Ny = np.sum( (x >= yy-Dy/2) & ( x<yy+Dy/2 ) )
```

```
        pdf[i] =  Ny / N / Dy
        cdf[i] = np.sum( x <= yy ) / N

    return pdf, cdf

def Q(x):
    return 0.5 * erfc( x / np.sqrt(2) )

def normpdf(x, mu, sigma):
    return  1 / np.sqrt(2 * np.pi) * np.exp( - (x-mu) ** 2.0 / 2 / sigma ** 2.0)

def normcdf(x, mu, sigma):
    z = (x - mu) / sigma
    return 1 - Q(z)
```

Listing 1: utils.py

3 Η κλάση signal

Το listing 2 δείχνει το αρχείο commlib.py. Η πρώτη κλάση που έχουμε υλοποιήσει είναι η signal. Όπως συζητήσαμε και παραπάνω, η κλάση έχει δύο σημαντικά γνωρίσματα: το t που είναι ο άξονας του χρόνου και το samples που περιέχει τα δείγματα του σήματος που αντιστοιχούν στις χρονικές στιγμές του t. Στην __init__ αρχικοποιούνται αυτά τα γνωρίσματα μαζί με ορισμένα βοηθητικά. Έτσι μπορεί κανείς να αρχικοποιήσει (αν περάσει τις αντίστοιχες παραμέτρους κατά την αρχικοποίηση του αντικειμένου) το φάσμα spec και τον άξονα των συχνοτήτων f. Στο Dt και στο Df αποθηκεύεται η διαδοχική χρονική και συχνοτική διαφορά που αντιστοιχεί σε διαδοχικές τιμές στον άξονα του χρόνου και των συχνοτήτων. Επίσης το γνώρισμα N περιέχει το πλήθος των δειγμάτων του σήματος.

Υπάρχουν και ορισμένες χρήσιμες μέθοδοι που ορίζονται για την signal:

- plot: κάνει μία γραφική παράσταση του σήματος στο πεδίο του χρόνου.
- energy: υπολογίζει την ενέργεια του σήματος στο διάστημα που αντιστοιχεί στον άξονα του χρόνου.
- avg_power: υπολογίζει την μέση ενέργεια του σήματος.
- calc_spec: υπολογισμός του φάσματος μέσω του FFT
- calc_inv_spec: αντιστροφή του φάσματος μέσω του IFFT

```
import numpy as np
import matplotlib.pyplot as plt
from utils import energy, avg_power, calc_spectrum, calc_inv_spectrum

class signal:
    """
    The basic signal class
    """
    def __init__(self, t = None, samples = None, f = None, spec = None):
        """
        Initialize the signal class
        """
        self.t = t
        if self.t is not None:
            self.Dt = t[1] - t[0]
            self.N = t.size
            self.T = np.max(t) - np.min(t)
        else:
            self.N = 0
            self.Dt = None


        self.samples = samples
        self.f = f
        self.spec = spec

        if self.f is not None:
            self.Df = f[1] - f[0]
            self.N = f.size
        else:
```

```
            self.Df = None

    def plot(self, line_type = '-', what = None, label = None, xlabel = None, ylabel = None):
        """
        Plot the signal
        """

        y = self.samples
        x = self.t
        if what == 'real':
            y = np.real(x)
        elif what == 'imag':
            y = np.imag(x)
        elif what == 'abs':
            y = np.abs(x)
        elif what == 'spec':
            y = np.abs( self.spec )
            x = self.f


        plt.plot(x, y, line_type, label = label)

        if xlabel is not None:
            plt.xlabel(xlabel)
        if ylabel is not None:
            plt.ylabel(ylabel)

    def energy(self):
        """
        Calculate signal energy
        """
        return energy(self.t, self.samples)

    def avg_power(self):
        """
        Calcualate average power
        """
        return avg_power(self.t, self.samples)

    def calc_spec(self):
        """
        Calculate spectrum using FFT
        """
        self.f, self.spec = calc_spectrum(self.t, self.samples)
        self.Df = self.f[1]-self.f[0]
        return self.spec, self.f

    def calc_inv_spec(self):
        """
        Calculate inverse spectrum using FFT
        """

        t, samples = calc_inv_spectrum(self.f, self.spec)

        if self.t is None:
            self.t = t
            self.Dt = t[2]-t[1]

        self.samples = samples
        return self.samples, self.t

    def __add__(self, b):
        """
        Sum two signals
        """
        return signal(t = self.t, samples = self.samples + b.samples)

    def __mul__(self, b):
        """
        Multiply signal with another signal or a constant
        """
        if np.isscalar(b):
            return signal(t = self.t, samples = self.samples * b)
        else:
            return signal(t = self.t, samples = self.samples * b.samples)

```

```
    def __rmul__(self, b):
        """
        Multiply signal with another signal or a constant
        """
        if np.isscalar(b):
            return signal(t = self.t, samples = self.samples * b)
        else:
            return signal(t = self.t, samples = self.samples * b.samples)

    def __pow__(self, c):
        """
        Raise signal samples to a power
        """
        return signal(t = self.t, samples = self.samples ** c)


class carrier(signal):
    """
    Carrier signal
    """
    def __init__(self, t, f0, A = 1, phi = 0.0):
        samples = A * np.cos( 2 * np.pi * f0 * t + phi)
        super().__init__( t = t, samples = samples )

class square_pulse(signal):
    """
    Square pulse
    """
    def __init__(self, t, T1, tcenter = 0.0, amp = 1.0):
        self.tcenter = tcenter
        samples = np.zeros(t.size)
        i = np.where( np.abs (t - tcenter) <= 0.5 * T1)
        samples[ i ] = amp
        self.amp = amp
        self.T1 = T1
        super().__init__( t = t, samples = samples )

class awgn(signal):
      """
      AWG noise
      """
      def __init__(self, t, N0):
          self.N0 = N0
          Dt = t[1] - t[0]
          self.sigma = np.sqrt( N0 / 2 / Dt )
          samples = self.sigma * np.random.randn(t.size)
          super().__init__( t = t, samples = samples )

class digital_signal(signal):
    """
    Digital signal
    """
    def __init__(self, TS = 1, samples_per_symbol = 10,
                 tinitial = 0, tguard = 0, symbols = None):

        super().__init__()
        self.TS = TS
        self.samples_per_symbol = samples_per_symbol
        self.tinitial = tinitial
        self.tguard = tguard

        if symbols is not None:
            self.modulate_from_symbols(symbols)

    def modulate_from_symbols( self, symbols ):
        """
        shape signal according to symbol sequence
        """
        self.Tmin = self.tinitial - self.tguard
        self.Tmax = self.tinitial + symbols.size * self.TS +  self.tguard
        self.Dt = self.TS / self.samples_per_symbol
        self.t = np.arange(self.Tmin, self.Tmax, self.Dt)
        self.samples = np.zeros( self.t.size )
        self.symbols = symbols
        self.N = self.t.size

```

```
        i = np.floor( (self.t - self.tinitial) / self.TS).astype(int)
        j = np.where( np.logical_and(i >= 0, i < symbols.size ) )

        self.samples[j] = symbols[ i[j] ]

class system:
    """
    Abstract system class
    """
    def __init__(self, input_s = None,
                       output_s = None,
                       action = None,
                       params = None):

        self.input_s = input_s
        self.output_s = output_s
        self.action = action
        self.params = params

    def set_input(self, input_s):
        """
        Set system input
        """
        self.input_s = input_s

    def set_output(self, output_s):
        """
        Set system output
        """
        self.output_s = output_s

    def set_action(self, action):
        """
        Set system action callable
        """
        self.action = action

    def set_params(self, params):
        """
        Set system params
        """
        self.params = params

    def calc_output(self):
        """
        Calculate system output applying the action callable
        """
        if self.params is None:
            self.output_s = self.action( self.input_s )
        else:
            self.output_s = self.action( self.input_s, **self.params)

        return self.output_s

    def calc_spec_input(self):
        """
        Calculate spectrum input using FFT
        """
        self.input_s.calc_spectrum()

    def calc_spec_output(self):
        """
        Calculate spectrum output using FFT
        """
        self.output_s.calc_spectrum()

    def calc_inv_input(self):
        """
        Calculate spectrum time domain input samples using IFFT
        """
        self.input_s.calc_inv_spectrum()

    def calc_inv_output(self):
        """
        Calculate spectrum time domain output samples using IFFT
        """
```

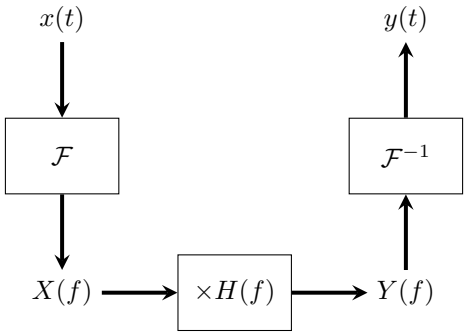
```
259         self.output_s.calc_inv_spectrum()
260
261 class lti_system(system):
262     """
263     Linear time invariant (LTI) system
264     """
265     def action(self, x):
266         """
267         Apply the action of the LTI system
268         """
269         X, f = x.calc_spec()
270         if callable(self.H):
271             Y = self.H(f) * X
272         else:
273             Y = self.H * X
274
275         y = signal(t = x.t, f = f, spec = Y)
276         y.calc_inv_spec()
277         return y
278
279     def __init__(self, input_s = None,
280                        output_s = None,
281                        H = None):
282
283         self.input_s = input_s
284         self.output_s = output_s
285         self.H = H
286         self.params = None
```

Listing 2: commlib.py

4 Άλλες κλάσεις σημάτων

Χρησιμοποιώντας την `signal` ορίζουμε παράγωγες κλάσεις (παιδιά της `signal`) που κληρονομούν τα γενικά γνωρίσματα και τις μεθόδους της και ταυτόχρονα εξειδικεύουν κάποια στοιχεία. Έχουμε τις εξής κλάσεις

- `carrier`: υλοποιεί ένα φέρον σήμα της μορφής
$$x(t) = A\cos\left(2\pi f_0 t + \phi\right) \tag{1}$$
- `square_pulse`: υλοποιεί ένα τετραγωνικό σήμα με πλάτος A του οποίου το κέντρο t_c καθορίζεται από το γνώρισμα `tcenter`:
$$x(t) = \begin{cases} A & , -\frac{T_1}{2} \leq t - t_c \leq \frac{T_1}{2} \\ 0 & , \text{διαφορετικά} \end{cases} \tag{2}$$
- `digital_signal`: Πρόκειται για την αναπαράσταση ενός ψηφιακού σήματος της μορφής:
$$x(t) = \sum_{k=0}^{K-1} A_k p(t - kT_S - t_i) \tag{3}$$
όπου τα $p(t)$ είναι τετραγωνικοί παλμοί με διάρκεια T_S που ξεκινούν από το $t = 0$ (ενώ μέχρι τώρα χρησιμοποιούσαμε παλμούς)
$$p(t) = \begin{cases} 1 & , t_{\leq} t \leq T_1 \\ 0 & , \text{διαφορετικά} \end{cases} \tag{4}$$
Η λογική της κλάσης ακολουθεί τον τρόπο που χρησιμοποιήσαμε για να συνθέσουμε το ψηφιακό σήμα στο 1ο μάθημα. Ωστόσο προσέξτε μερικές προσθήκες:
  - ορίζουμε το πλήθος των δειγμάτων ανά διάρκεια συμβόλου `samples_per_symbol`. Αυτό σημαίνει ότι για κάθε ένα σύμβολο που θεωρούμε θα υπάρχουν `samples_per_symbol` δείγματα του σήματος.
  - το σήμα έχει μία διάρκεια που ισούται με `tguard` στην οποία είναι μηδέν και το πρώτο σύμβολο ξεκινάει από το `tinitial`. Αυτό σημαίνει ότι το t_i ισούται με `tinitial`. Επίσης μετά τον παλμό που αντιστοιχεί στο τελευταίο σύμβολο υπάρχει πάλι μία διάρκεια `tguard` στην οποία το σήμα είναι μηδέν. Τα διαστήματα αυτά χρησιμοποιούνται συχνά για να αυξήσουν την ακρίβεια στους υπολογισμούς στο πεδίο των συχνοτήτων.
  - η μέθοδος `modulate_from_symbols` χρησιμοποιείται για να αποτυπώσει τα σύμβολα `symbols` στο σήμα.
- `awgn`: υλοποιεί ένα θόρυβο AWG. Θα δώσουμε μερικές λεπτομέρειες υλοποίησης παρακάτω.



Εικόνα 1: Περιγραφή ενός LTI συστήματος με χρήση του πεδίου των συχνοτήτων.

5 Η κλάση `lti_system`

Υλοποιούμε δύο κλάσεις που αφορούν τα συστήματα την `system` που περιγράφει ένα γενικό σύστημα και την παράγωγη κλάση `lti_system` η οποία περιγράφει ένα σύστημα LTI. Η κλάση `system` έχει τα εξής γνωρίσματα:

- `input_s`: σήμα εισόδου
- `output_s`: σήμα εξόδου
- `action`: ένα callable που περιγράφει την δράση του συστήματος.
- `params`: παράμετροι που χρησιμοποιούνται κατά την κλήση του `action`. Γενικά θεωρούμε ότι το πρώτο όρισμα του `action` είναι ένα σήμα ενώ αν υπάρχουν επιπλέον παράμετροι που καθορίζουν την δράση του συστήματος αυτές περνάνε ως keyword ορίσματα στην action. Θα δούμε ένα παράδειγμα και πιο μετά.

Υπάρχουν και διάφορες μέθοδοι που ορίζονται στην `system`:

- `set_input`: Θέτουμε την είσοδο του συστήματος.
- `set_output`: Θέτουμε την έξοδο του συστήματος.
- `set_action`: Ορίζουμε την δράση του συστήματος
- `set_params`: Ορίζουμε τις παραμέτρους της δράσης του συστήματος.
- `calc_output`: Υπολογίζουμε την έξοδο βάσει της οριζόμενης δράσης.
- `calc_spec_input`: Υπολογισμός του φάσματος εισόδου του συστήματος.
- `calc_spec_output`: Υπολογισμός του φάσματος εξόδου του συστήματος.
- `calc_inv_input`: Υπολογισμός του αντίστροφου μετασχηματισμού Fourier του σήματος εισόδου του συστήματος.
- `calc_inv_output`: Υπολογισμός του αντίστροφου μετασχηματισμού Fourier του σήματος εξόδου του συστήματος.

Η κλάση `lti_system` κληρονομεί τα χαρακτηριστικά και τις μεθόδους της `system`. Ωστόσο σε ότι αφορά την δράση του συστήματος αυτή ακολουθεί την λογική που είδαμε στο μάθημα 5 και επαναλαμβάνουμε εδώ για ευκολία στην εικόνα 1. Επομένως ορίζουμε το όρισμα `action` που περιγράφει τη δράση του συστήματος βάσει της συνάρτησης μεταφοράς $H(f)$ που αντιστοιχεί στο γνώρισμα `H` του αντικειμένου.

6 Παραδείγματα

6.1 Παράδειγμα μη γραμμικού συστήματος

```python
from commlib import carrier, system
import numpy as np
import matplotlib.pyplot as plt

f0 = 1
Tmax = 5
Tmin = 0
Nt = 1024

def nl_action(s, b = 0.1):
    return s + b * s ** 2

t = np.linspace(Tmin, Tmax, Nt)
x = carrier(t, f0, phi = -np.pi / 2)

S1 = system(input_s = x,
            action = nl_action,
            params = {'b' : 0.01})
S2 = system(input_s = x,
            action = nl_action,
            params = {'b' : 0.1})

plt.close('all')
plt.figure()
x.plot(xlabel = '$t$', label = '$x(t)$')
y1 = S1.calc_output()
y1.plot(label = '$y(t)$')
plt.legend()
y1.calc_spec()
plt.figure()
y1.plot(what = 'spec', ylabel = '$|Y(f)|$', xlabel = '$f$')
plt.xlim([-4, 4])

plt.figure()
y2 = S2.calc_output()
x.plot(xlabel = '$t$', label = '$x(t)$')
y2.plot(label = '$y(t)$')
plt.legend()
y2.calc_spec()
plt.figure()
y2.plot(what = 'spec', ylabel = '$|Y(f)|$', xlabel = '$f$')
plt.xlim([-4, 4])
```

Listing 3: example.py

Στο listing 3 δείχνουμε ένα πρώτο παράδειγμα χρήσης της `commlib` στην περίπτωση ενός συστήματος $\mathcal{S}$ που επιδρά σε ένα σήμα εισόδου $x(t)$ ως εξής:

$$y(t) = \mathcal{S}\{x(t)\} = x(t) + bx^2(t) \tag{5}$$

Μπορούμε να δείξουμε ότι το σύστημα δεν είναι γραμμικό επομένως δεν είναι και LTI. Ως είσοδο στο σήμα θεωρούμε ένα φέρον της μορφής (1). Στην εικόνα 2 δείχνουμε την είσοδο και την έξοδο στο πεδίο του χρόνου καθώς και το φάσμα της εξόδου στην περίπτωση όπου $b = 0.01$ ενώ το ίδιο κάνουμε και στην εικόνα 3 για $b = 0.1$. Έχει ένα ενδιαφέρον να παρατηρήσουμε ότι υπάρχει μία παραμόρφωση η οποία αυξάνει εξαιτίας του παράγοντα b. Αυτό είναι φανερό και στο φάσμα της εξόδου όπου πέρα από τις φασματικές συνιστώσες στο $f = \pm f_0$ εμφανίζονται και συνιστώσες στο $f = 0$ και στο $f = \pm 2f_0$

Μπορούμε να δούμε και θεωρητικά γιατί έχουμε αυτή την συμπεριφορά. Αν αντικαταστήσουμε την (1) στην (5) τότε θα βρούμε:

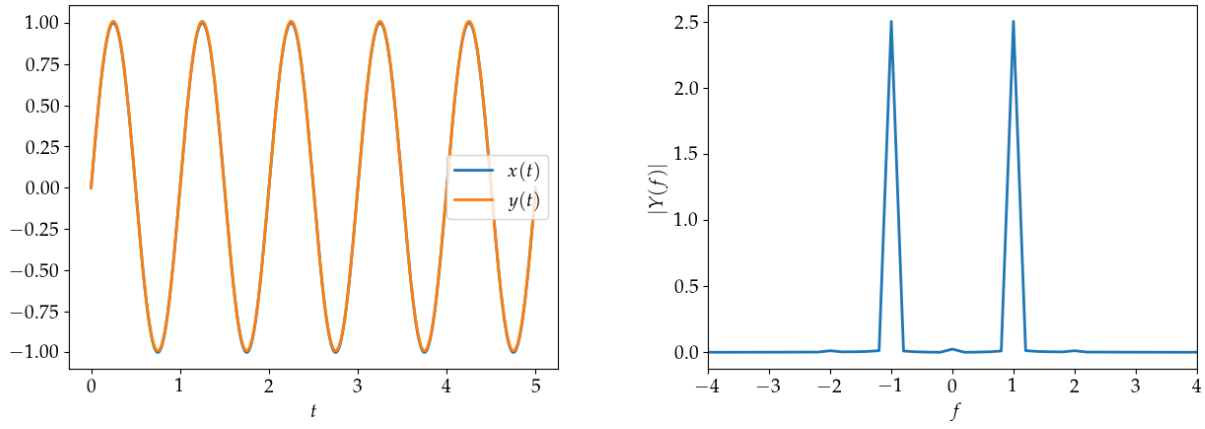
$$y(t) = A\cos(2\pi f_0 t) + bA^2\cos^2(2\pi f_0 t) = A\cos(2\pi f_0 t) + bA^2\frac{\cos(4\pi f_0 t)}{2} + \frac{bA^2}{2} = \\ \frac{bA^2}{2} + Ae^{j2\pi f_0 t} + Ae^{-j2\pi f_0 t} + \frac{bA^2}{4}e^{j4\pi f_0 t} + \frac{bA^2}{4}e^{-j4\pi f_0 t} \tag{6}$$

Το φάσμα εξόδου θα είναι:

$$Y(f) = \frac{bA^2}{2}\delta(f) + A\delta(f - f_0) + A\delta(f + f_0) + \frac{bA^2}{4}\delta(f - 2f_0)\frac{bA^2}{4}\delta(f + 2f_0) \tag{7}$$

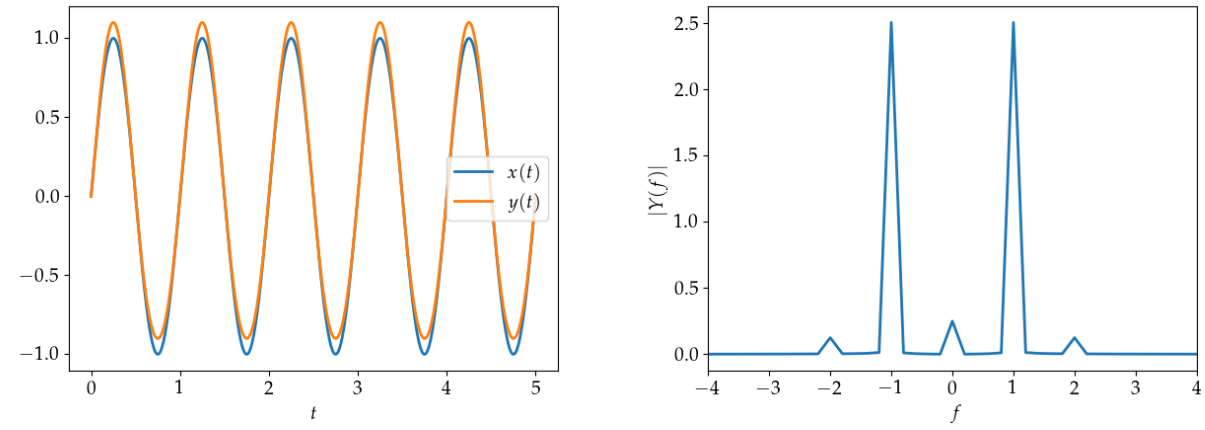
Επομένως το φάσμα εξόδου περιέχει δύο συναρτήσεις δ με πλάτος $A_1 = A$ στις συχνότητες $f = \pm f_0$, άλλες δύο με πλάτος $A_2 = \frac{bA^2}{4}$ στο $f = \pm 2f_0$ και μία στο $f = 0$ με πλάτος $A_0 = \frac{bA^2}{2}$. Επομένως

$$\frac{A_2}{A_1} = \frac{bA}{4} \tag{8}$$



(a) (b)

Εικόνα 2: (a) Το σήμα εισόδου και εξόδου στο πεδίο του χρόνου και (b) το σήμα εξόδου στο πεδίο των συχνοτήτων όταν $b = 0.01$.



(a) (b)

Εικόνα 3: (a) Το σήμα εισόδου και εξόδου στο πεδίο του χρόνου και (b) το σήμα εξόδου στο πεδίο των συχνοτήτων όταν $b = 0.1$.

δηλαδή ο λόγος του πλάτος A_2/A_1 αυξάνει με το β όπως φαίνεται στις εικόνες 2 και 3. Μία παρόμοια σχέση ισχύει για το A_0/A_1. Επίσης έχει ένα ενδιαφέρον να παρατηρήσουμε ότι $A_0 = 2A_2$ και επομένως στο $f = 0$ θα πρέπει το πλάτος του φάσματος να είναι διπλάσιο από ότι στο $f = \pm 2f_0$.

6.2 Παράδειγμα LTI συστήματος

Θα δώσουμε και ένα παράδειγμα για την περίπτωση ενός LTI συστήματος. Θεωρούμε ότι το σύστημα μας έχει μία συνάρτηση μεταφοράς:

$$H(f) = \exp\left(-\frac{f^2}{2B^2}\right) \tag{9}$$

και ως είσοδο έχουμε ένα σήμα της μορφής (5) που περιγράφεται από την κλάση `digital_signal`. Στην εικόνα 4 έχουμε παραστήσει γραφικά την είσοδο και την έξοδο για διάφορετικές τιμές του B. Παρατηρούμε ότι όσο μεγαλύτερο είναι το B οπότε το $H(f)$ εκτείνεται και σε υψηλές συχνότητες τόσο πιο πολύ μοιάζει η είσοδος στην έξοδο. Αντίθετα όσο μικραίνει το B "κόβονται" όλο και περισσότερο οι υψηλές συχνότητες και το σήμα εξόδου δεν μπορεί να παρακολουθήσει τις απότομες μεταβολές του σήματος εισόδου. Αν παρατηρήσουμε ιδιαίτερα το σήμα εξόδου στην εικόνα 4a θα δούμε ότι οι παλμοί $p(t)$ που αποτελούν το σήμα έχουν διευρυνθεί και επομένως το ένα σύμβολο πέφτει (αλληλεπιδρά δηλαδή με το γειτονικό του) οπότε μιλάμε για *αλληλεπίδραση συμβόλων* (intersymbol interference - ISI). Είναι ένα φαινόμενο το οποίο παίζει πολύ σημαντικό ρόλο σε κανάλια που έχουν την τάση να παρουσιάζουν σημαντική εξασθένιση στις υψηλές συχνότητες και πρέπει να βρεθεί ένας τρόπος αντιστάθμισης του.

```
1  from commlib import digital_signal, lti_system
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5  symbols = np.array([-1, 1, 3, 7, 5, 1, 3])
6  Ns = symbols.size
7  TS = 1
8  BB = [1, 2, 10]
9  tguard = 2 * TS
10 samples_per_symbol = 20
11
12 x = digital_signal(TS = TS,
13                     samples_per_symbol = samples_per_symbol,
14                     tguard = tguard, symbols = symbols)
15
16 plt.close('all')
17 for B in BB:
18     H = lambda f: np.exp(-f ** 2 / 2 / B ** 2)
19
20     S = lti_system(input_s = x, H = H)
21     y = S.calc_output()
22     plt.figure()
23     x.plot(xlabel = '$t$', label = '$x(t)$')
24     y.plot(xlabel = '$t$', label = '$y(t)$')
25     plt.legend()
26     plt.title('$B=%6.1f$' % B)
```

Listing 4: example_lti.py

7 Η κλάση awgn

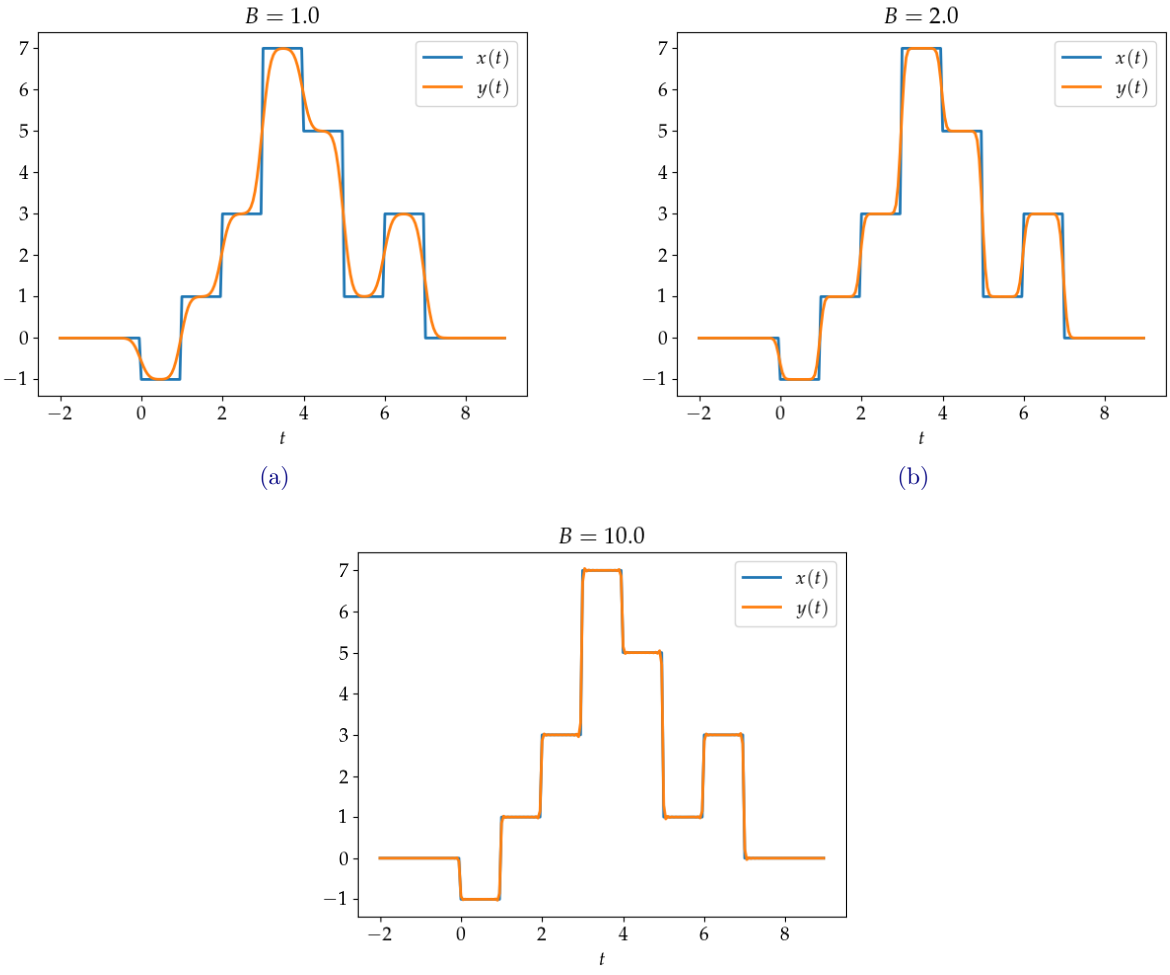
Έχει ενδιαφέρον να προσπαθήσουμε να μοντελοποιήσουμε τον θόρυβο AWG. Ας θεωρήσουμε αρχικά ότι έχουμε ένα `numpy array n` του οποίου τα στοιχεία n_k παράγονται από μία τυχαία γεννήτρια δειγμάτων με μηδενική μέση τιμή, διακύμανση $\sigma^2 = s^2/\Delta$ ενώ είναι ανεξάρτητα μεταξύ τους. Αν τα n_k αντιστοιχούν σε ένα αναλογικό σήμα $n(t)$ στις χρονικές στιγμές $t = t_k$ που διαφέρουν μεταξύ του κατά Δ, τότε η συνάρτηση αυτοσυσχέτισης του $n(t)$ θα είναι

$$R_{nn}(t_p, t_q) = \mathbb{E}\{n(t_p)n(t_q)\} = \mathbb{E}\{n_p n_q\} = \begin{cases} \frac{s^2}{\Delta} & , t_p = t_q \\ 0 & , \text{διαφορετικά} \end{cases} \tag{10}$$

Αν προσέξουμε αυτή την σχέση παρατηρούμε ότι όταν $\Delta \to 0$ το όριο τείνει στο άπειρο ενώ όταν $t_p \neq t_q$ η συνάρτηση είναι μηδέν. Επομένως μπορούμε να θεωρήσουμε ότι η συνάρτηση αυτοσυσχέτισης R_{nn} προσεγγίζει μία συνάρτηση δ, δηλαδή:

$$R_{nn}(t_p, t_q) \cong s^2\delta(t_p - t_q) \tag{11}$$

Δεδομένου ότι τα δείγματα n_k έχουν κανονική κατανομή με μέση τιμή μηδέν και συνάρτηση αυτοσυσχέτισης που σύμφωνα με την (11) προσεγγίζεται από μία συνάρτηση $\delta(t)$. Προκύπτει επομένως ότι έχει τις ιδιότητες του



(a) (b) (c)

Εικόνα 4: Το σήμα εισόδου $x(t)$ και το σήμα εξόδου $y(t)$ για ένα LTI σύστημα.

θορύβου AWG αρκεί να αντικαταστήσουμε το s^2 με $N_0/2$,

$$s^2 \cong \frac{N_0}{2} \tag{12}$$

οπότε η διακύμανση των n_k θα πρέπει να επιλεγεί ως εξής:

$$\sigma^2 = \mathbb{E}\left\{n_k^2\right\} = \frac{N_0}{2\Delta} \tag{13}$$

```
from commlib import awgn, lti_system
import numpy as np
import matplotlib.pyplot as plt

N0 = 1
Nt = 65536
iv = int(Nt / 2)
Niters = 1000
Tmax = 10
t = np.linspace(0, Tmax, Nt)
n = awgn( t, N0 )

plt.close('all')
n.plot(xlabel = '$t$', label = '$n(t)$')
BB = np.array([1, 2, 10])
sigmas2 = np.zeros(BB.size)

for i, B in enumerate(BB):
    H = lambda f: (np.abs(f) <= B).astype(float)
    for k in range(Niters):
        n = awgn(t, N0)
        S = lti_system(input_s = n, H = H)
        y = S.calc_output()
        if k==0:
            plt.figure()
            y.plot(xlabel = '$t$', label = '$y(t)$')
            plt.legend()
            plt.title('$B=%6.1f$' % B)

        sigmas2[i] += np.real(y.samples[iv]) ** 2 / Niters

print(sigmas2)
```

Listing 5: awgn.py

Στο listing 5 δείχνουμε ένα παράδειγμα χρήσης της κλάσης **awgn** όπου δημιουργούμε έναν θόρυβο AWG $n(t)$ και τον περνάμε από ένα σύστημα με συνάρτηση μεταφοράς με τετραγωνική απόκριση:

$$H(f) = \begin{cases} 1 & , -B \leq f \leq B \\ 0 & , \text{διαφορετικά} \end{cases} \tag{14}$$

Στην εικόνα 5 δείχνουμε την είσοδο $n(t)$ και την έξοδο $y(t)$ που παράγονται για διάφορες τιμές του B. Σύμφωνα με αυτά που ξέρουμε για τα τυχαία σήματα, η φασματική πυκνότητα ισχύος του $n(t)$ είναι

$$S_n(f) = \frac{N_0}{2} \tag{15}$$

ενώ η φασματική πυκνότητα της εξόδου $y(t)$ θα είναι:

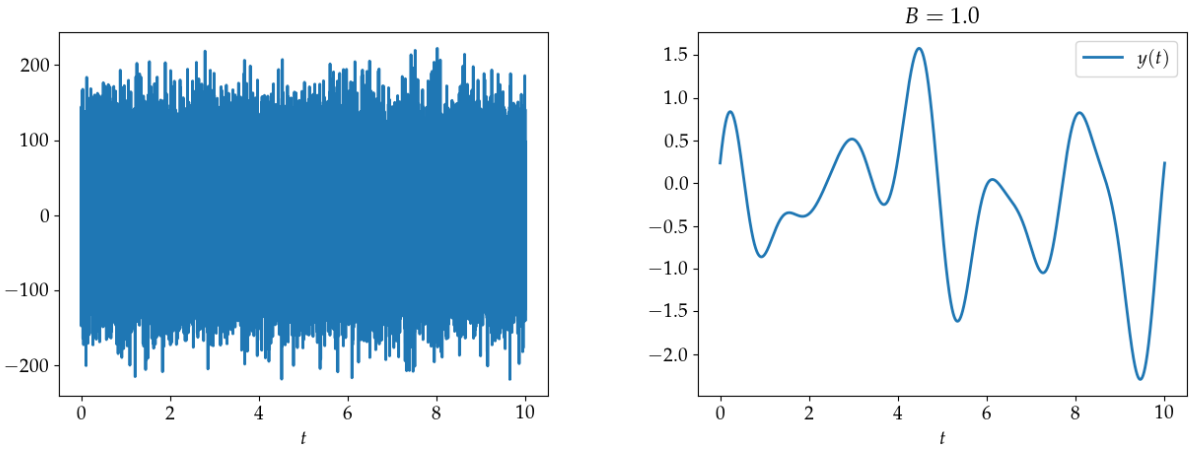
$$S_y(f) = S_n(f)|H(f)|^2 = \begin{cases} \frac{N_0}{2} & , -B \leq f \leq B \\ 0 & , \text{διαφορετικά} \end{cases} \tag{16}$$

Σύμφωνα με την παραπάνω η $S_y(f)$ περιέχει όλες τις φασματικές συνιστώσες του $S_n(f)$ στις χαμηλές συνχότητες, $|f| \leq B$ και κόβει όσες είναι σε υψηλές συχνότητες $|f| > B$. Παρατηρείστε ότι αυτό επηρεάζει την συμπεριφορά των δειγμάτων του $y(t)$ όσο αλλάζει το B. Για μεγάλο B (εικόνα 4c), οι μεταβολές της εξόδου είναι αρκετά γρήγορες καθώς το $H(f)$ είναι ευρύ και η φασματική πυκνότητα εξόδου $S_y(f)$ είναι μη μεδενική σε πιο υψηλές συχνότητες. Όσο όμως το B μικραίνει, οι μεταβάσεις του $y(t)$ γίνονται πιο αργές καθώς τώρα η $H(f)$ "κόβει" και μικρότερες συχνότητες.

Η ισχύς του σήματος είναι:

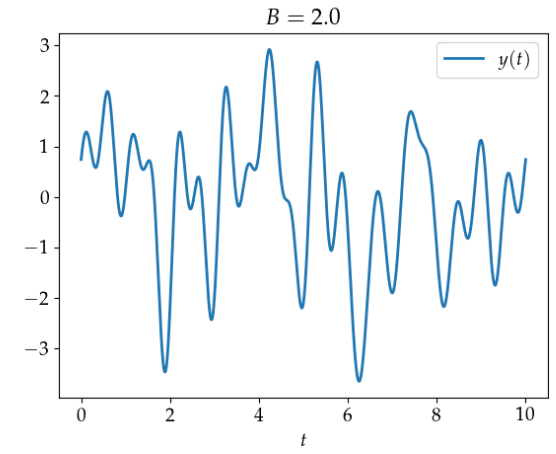
$$\mathcal{P}_y = \mathbb{E}\left\{y^2(t)\right\} = \int_{-\infty}^{+\infty} S_y(f)\mathrm{d}f = N_0 B \tag{17}$$

Το listing 5 υπολογίζει αριθμητικά την ισχύ του σήματος και όπως φαίνεται και από τα αποτελέσματα που τυπώνει στην οθόνη υπάρχει καλή συμφωνία με την (17).

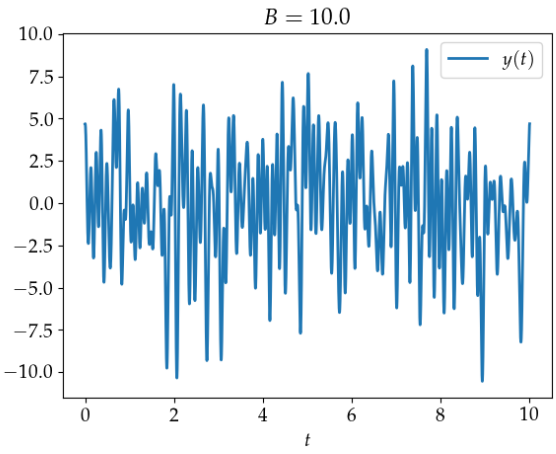


(a)

(b)



(c)



(d)

Εικόνα 5: Το σήμα εισόδου $n(t)$ και το σήμα εξόδου $y(t)$ για ένα LTI σύστημα με συνάρτηση μεταφοράς (14).