



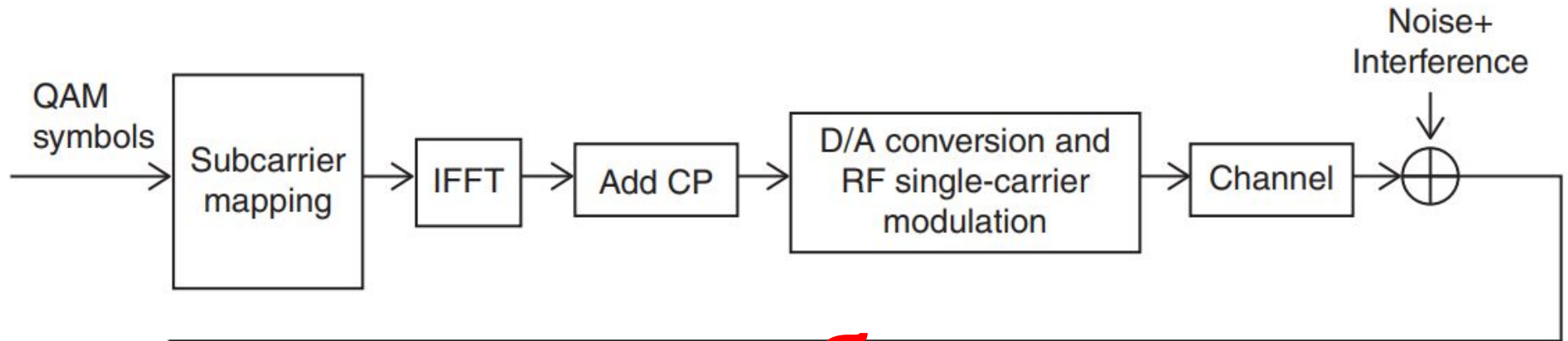
# Методы обработки сигналов в телекоммуникационных системах

Тимошенко Александр Геннадиевич

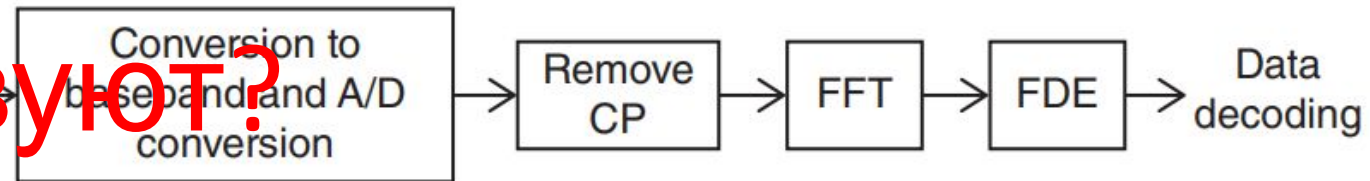
4226а

Некоторые слайды на английском  
языке

**Какие методы необходимо применять в предлагаемой схеме?**



**А какие методы вообще существуют?**





## **А какие методы вообще существуют?**

- А какие сигналы мы обрабатываем?

## Analog signals

Infinite precision in amplitude

Continuous signal

Continuous mathematics



## Digital signals

Finite precision in amplitude

Only present at certain points

Discrete mathematics





## Analog signals

восстановление

разделение информационных  
потоков

подавление шумов

сжатие данных

фильтрация

усиление сигналов

## Digital signals

спектральный анализ

линейная фильтрация

свертка традиционных типов

частотно-временной анализ

нелинейная обработка

адаптивная фильтрация

многоскоростная обработка

секционная свертка



## Базовые операции

задержка сигналов

сложение сигналов

вычитание сигналов

умножение сигналов

деление сигналов

интегрирование

дифференцирование

фильтрация

спектральный анализ

линейная фильтрация

свертка традиционных типов

частотно-временной анализ

нелинейная обработка

адаптивная фильтрация

многоскоростная обработка

секционная свертка



## Какие ещё виды обработки сигналов возможны?

- Статистическая обработка сигналов
- Обработка звука
  - Распознавание речи
- Обработка изображений
  - Обработка видео
  
- Обработка сигналов нужна для обработки информации
  - Включая экспериментальные данные



## Signal processing methods

- Today:
- Part 1
  - Digital signals
- Part 2
  - Basic programming in Python
  - Data types (in Python)
  - Correct notations in coding
  - Visualizing your signals



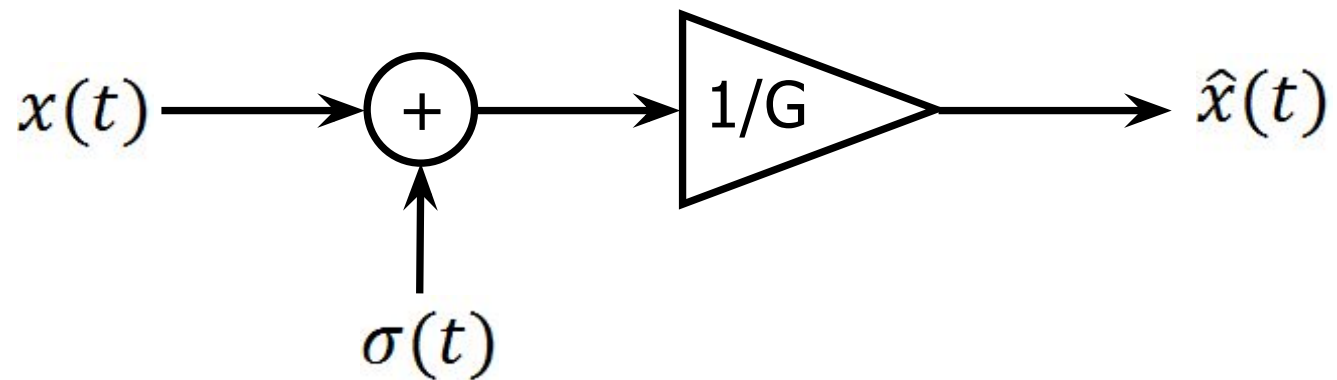


## Why digital signals are useful?

- Storage (generalized computer memory, compatibility)
- Processing (coding, no mechanical or physical interactions)
- Transmission(signal regeneration)

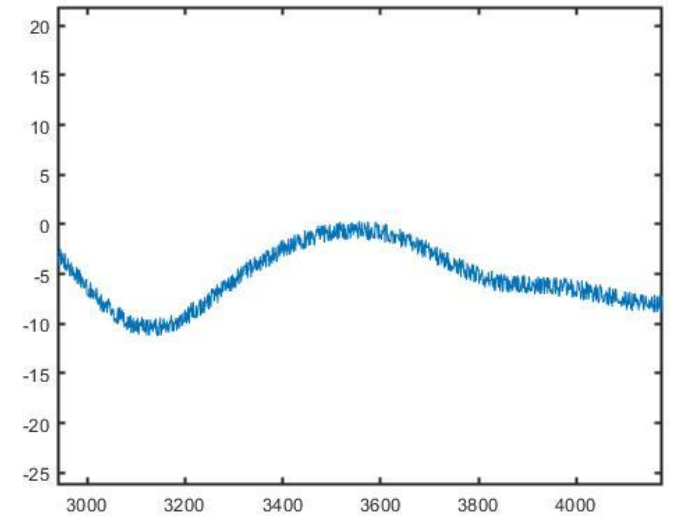
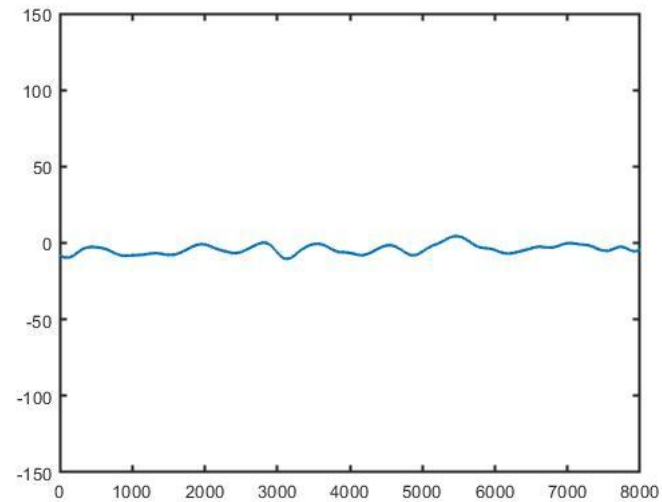
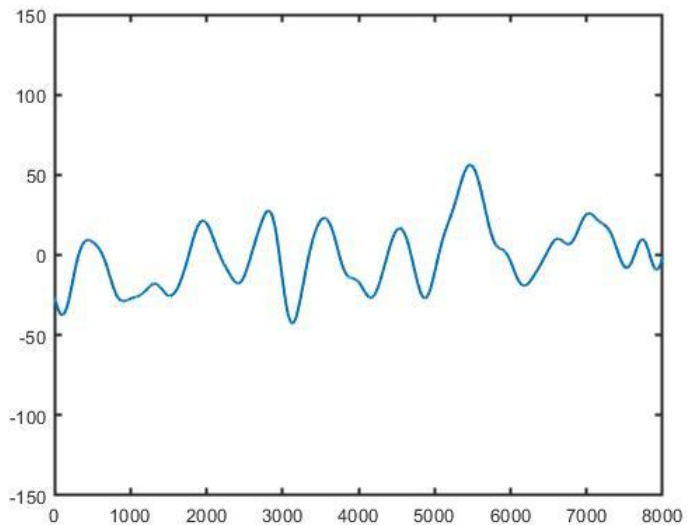
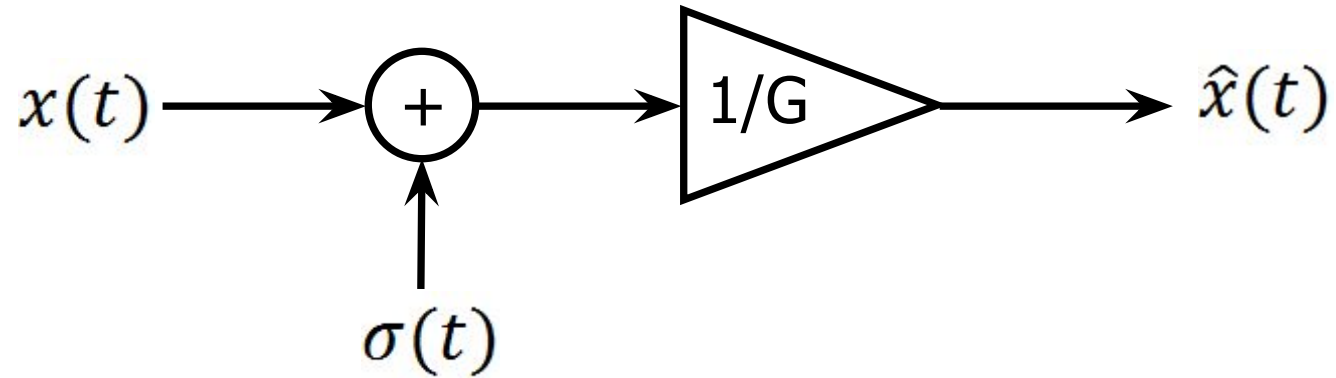


## Transmission - signal regeneration

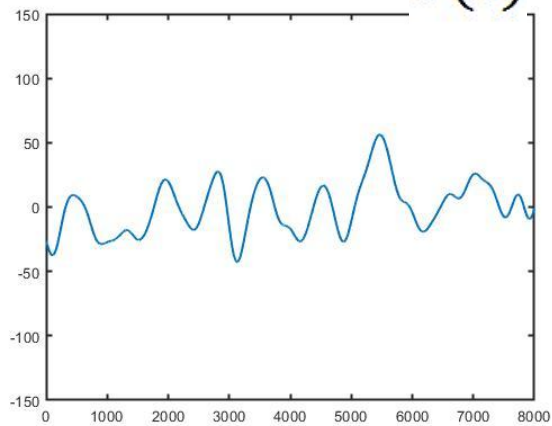
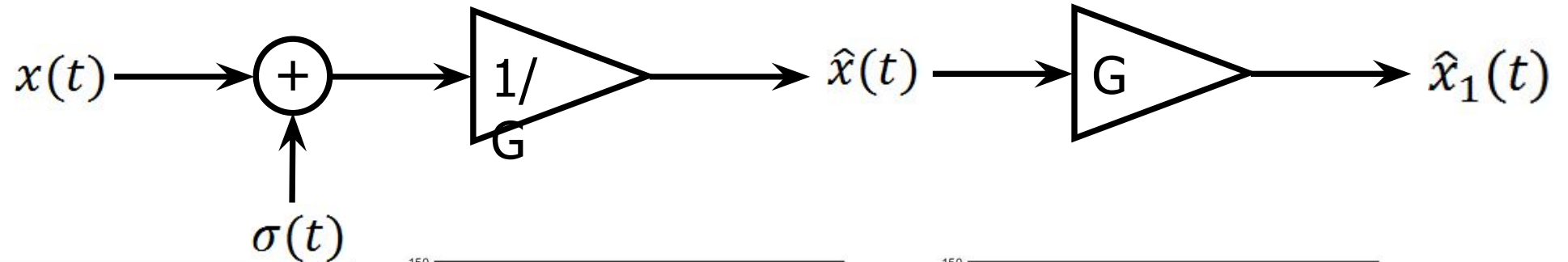




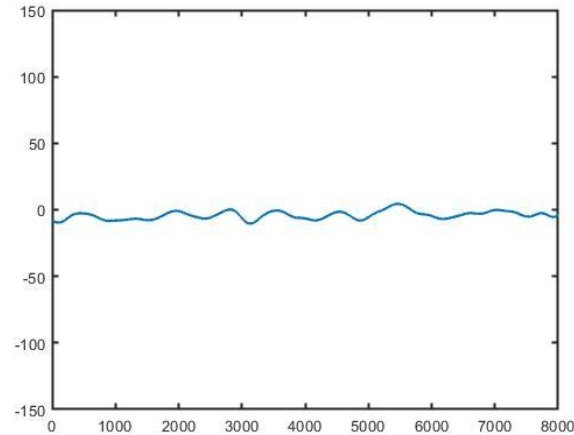
## Transmission - signal regeneration



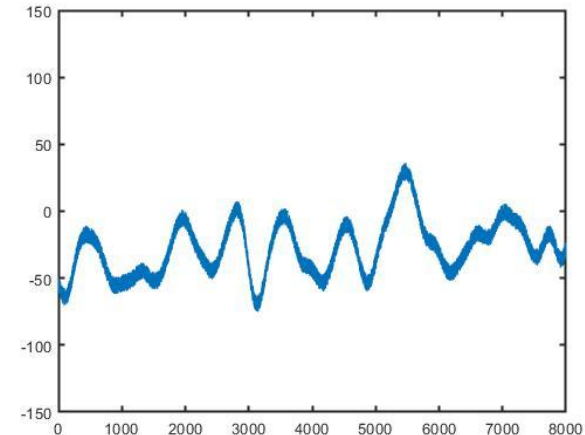
## Transmission - signal regeneration



$x(t)$



$\hat{x}(t) = x(t)/G + \sigma(t)$

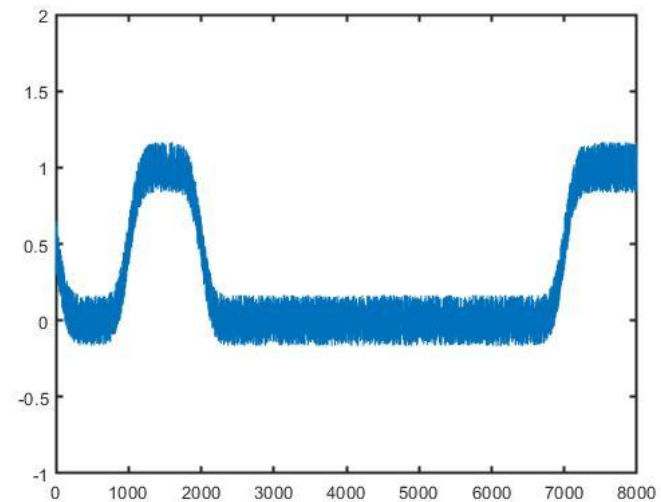
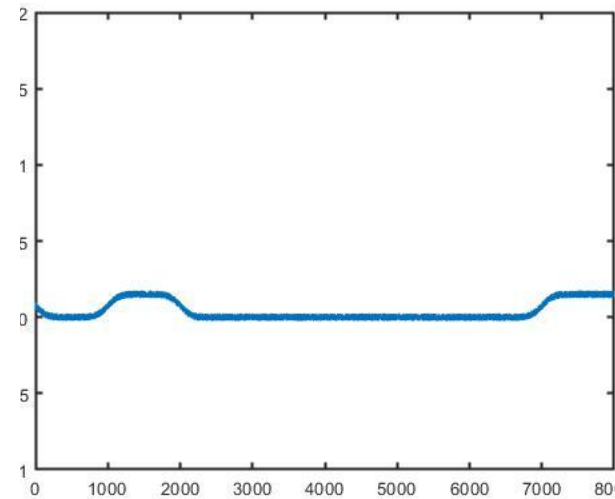
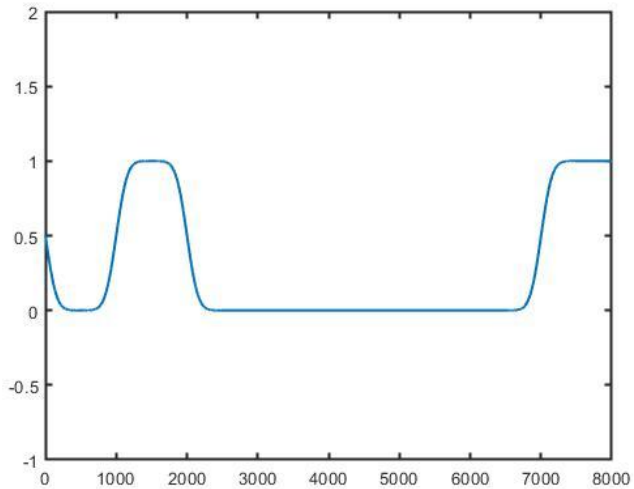
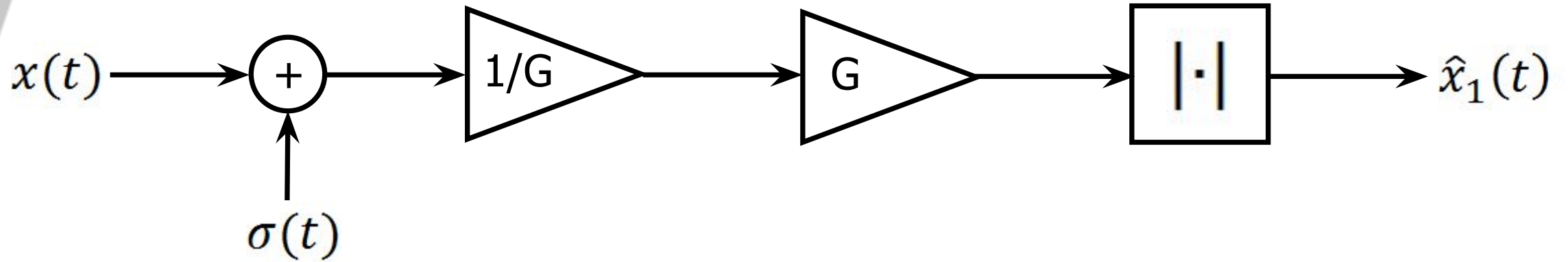


$$\hat{x}_1(t) = [x(t)/G + \sigma(t)]G$$

$$\hat{x}_1(t) = x(t) + G\sigma(t)$$

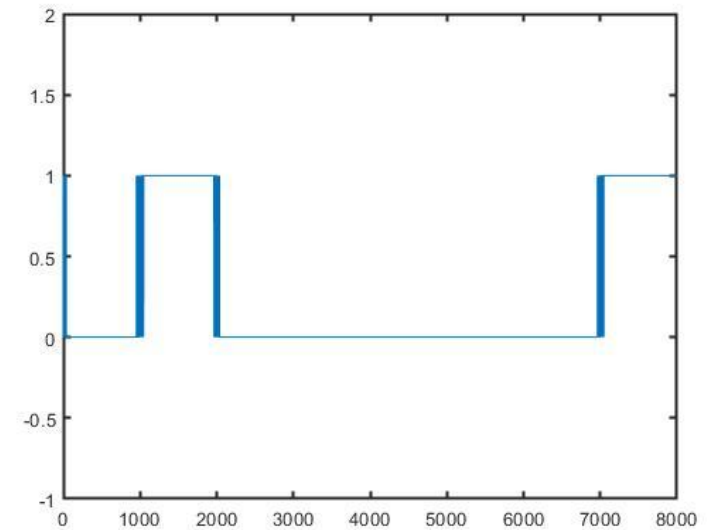
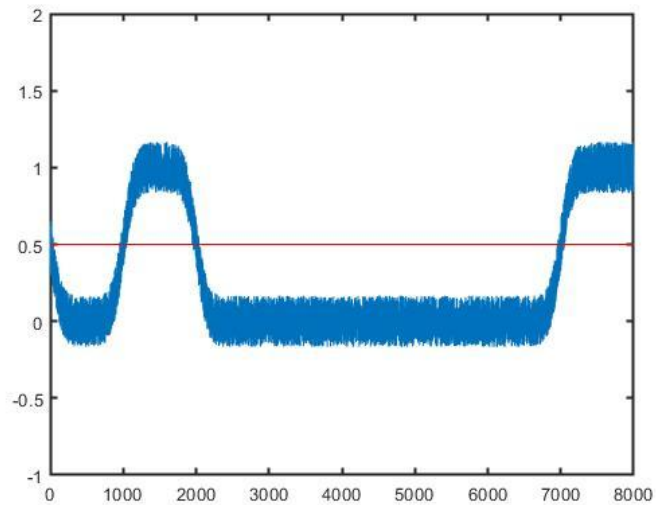
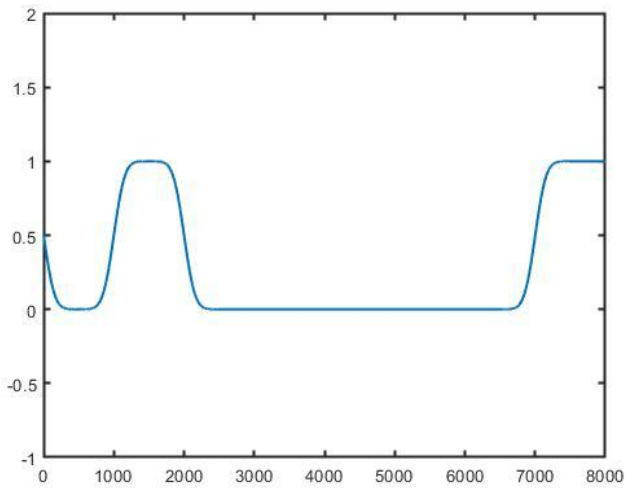
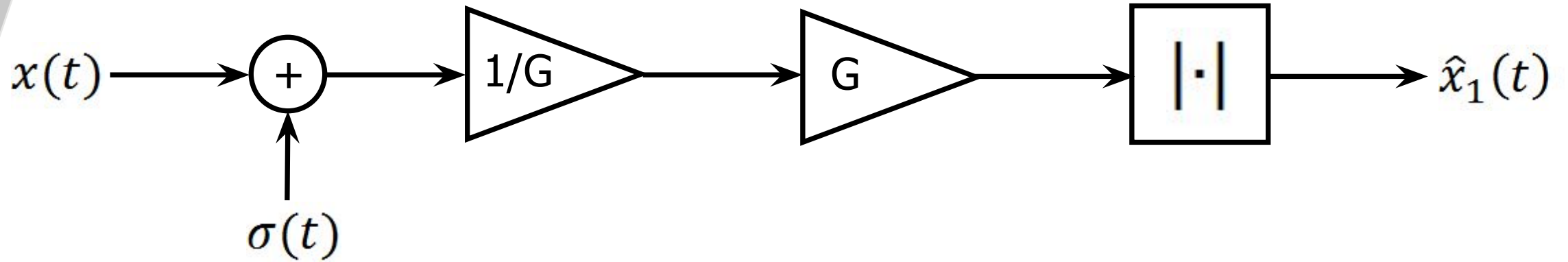


## Transmission - signal regeneration





## Transmission - signal regeneration





## Why digital signals are useful?

- Storage (generalized computer memory, compatibility)
- Processing (coding, no mechanical or physical interactions)
- Transmission(signal regeneration)



## Digital signals

- One dimensional (for now)
- Notation  $x[n]$  where «n» is an integer
- Two sides sequences  $x : \mathbb{Z} \rightarrow \mathbb{C}$
- n is a-dimensional
  
- Analysis (by taking periodic measurements)
- Synthesis (program that generates a stream of samples)

$\mathbb{Z}$  (*set of integers*)

$\mathbb{C}$  (*set of complex numbers*)



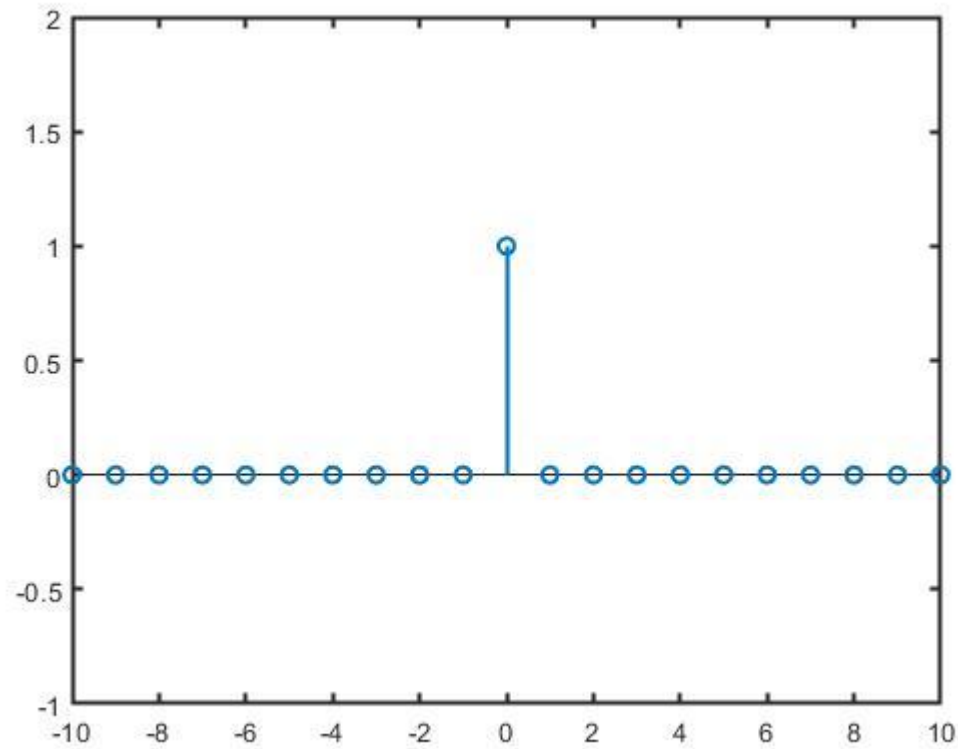


## **Digital Signals – common signals**

- Delta Dirac function
- Unit step function
- Exponential decay
- Sinusoidal signals



## Delta signal

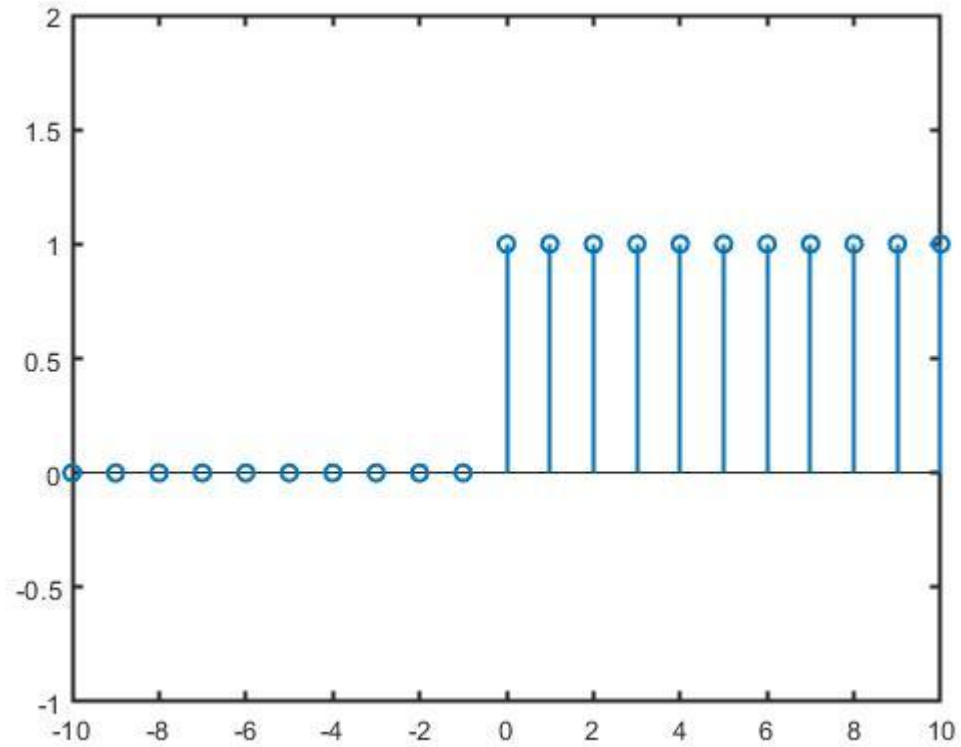


$$x[n] = \delta[n]$$

$$x[n] = \begin{cases} 0, & \text{when } n \neq 0 \\ 1, & \text{when } n = 0 \end{cases}$$



## Unit Step function



$$\begin{matrix} \square & \square & \square & \square & \square \\ \hline \square & \square & \square & \square & \square \end{matrix} = \begin{matrix} \square & \square \\ \hline \square & \square \end{matrix} \left[ \begin{matrix} \square & \square \\ \hline \square & \square \end{matrix} \right]$$

$$\begin{matrix} \square & \square & \square & \square & \square \\ \hline \square & \square & \square & \square & \square \end{matrix} = \begin{matrix} \square & \square \\ \hline \square & \square \end{matrix} \left[ \begin{matrix} \square & \square \\ \hline \square & \square \end{matrix} \right]$$

$$\begin{matrix} \square & \square & \square & \square & \square \\ \hline \square & \square & \square & \square & \square \end{matrix} = \begin{matrix} \square & \square \\ \hline \square & \square \end{matrix} \left[ \begin{matrix} \square & \square \\ \hline \square & \square \end{matrix} \right]$$

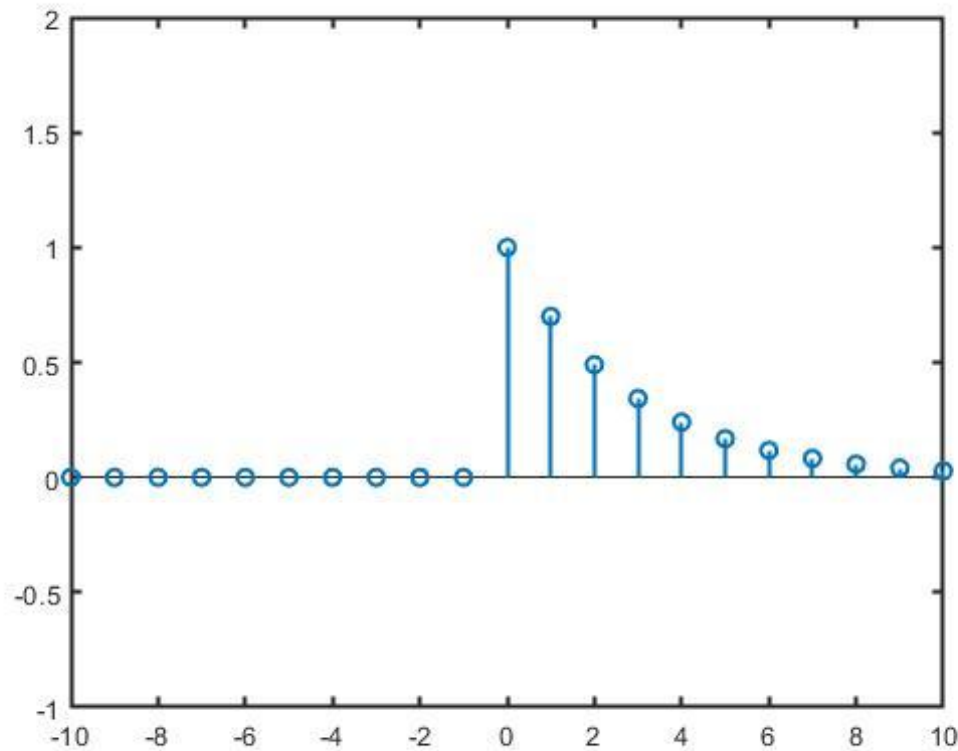
$$\begin{matrix} \square & \square & \square & \square & \square \\ \hline \square & \square & \square & \square & \square \end{matrix} = 1 \left[ \begin{matrix} \square & \square \\ \hline \square & \square \end{matrix} \right]$$

$$\begin{matrix} \square & \square & \square & \square & \square \\ \hline \square & \square & \square & \square & \square \end{matrix} = \begin{matrix} \square & \square \\ \hline \square & \square \end{matrix} \left[ \begin{matrix} \square & \square \\ \hline \square & \square \end{matrix} \right]$$

$$x[n] = \begin{cases} 0, & \text{when } n < 0 \\ 1, & \text{when } n \geq 0 \end{cases}$$



## Exponential functions

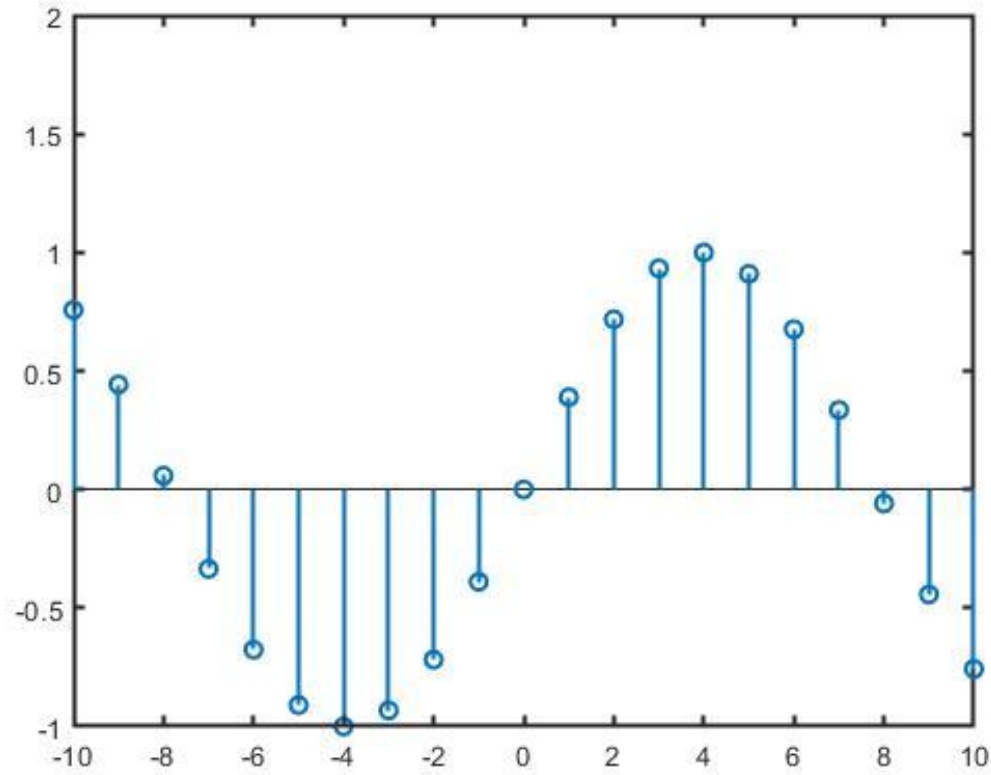


$$x[n] = \begin{cases} 0, & \text{when } n < 0 \\ a^n, & \text{when } n \geq 0 \end{cases}$$

*when  $a < 1$  converges to 0*



## Sinusoidal functions



$$x[n] = \sin(w_0 n + \theta)$$



## Signal classes

- Finite length  $x[n]$  for  $n = 0, 1, \dots, N-1$
- Infinite length
- Periodic
- Finite-support



## Finite length

□ Sequence notation

$$x[n] = 0, 1, \dots, N - 1$$

□ Vector notation

$$x = [x_0 \ x_1 \ \dots \ x_{N-1}]^T$$

□ Easy to process in numerical packages (numPy)



## Infinite length

- Sequence notation
- Good for theorems

$$x[n], n \in \mathbb{Z}$$





## Periodic

- N-periodic Sequence  $\tilde{x}[n] = \tilde{x}[n + kN], \quad n, k, N \in \mathbb{Z}$
- The same information as finite signals with length N
- Useful as a bridge between finite and infinite signals



## Finite support signals

$$x[n] = \begin{cases} x[n], & \text{if } 0 \leq n < N \\ 0, & \text{otherwise} \end{cases} \quad n \in \mathbb{Z}$$

- The same information as finite signals of length  $N$
- Useful as a bridge between finite and infinite signals



## Energy and Power

$$E_x = \sum_{n=-\infty}^{\infty} |x[n]|^2$$

$$P_x = \lim_{N \rightarrow \infty} \frac{1}{2N + 1} \sum_{n=-N}^N |x[n]|^2$$



## Elementary operators

□ Delay  $y[n] = x[n - k] \quad k \in \mathbb{Z}$

□ Scaling  $y[n] = ax[n]$

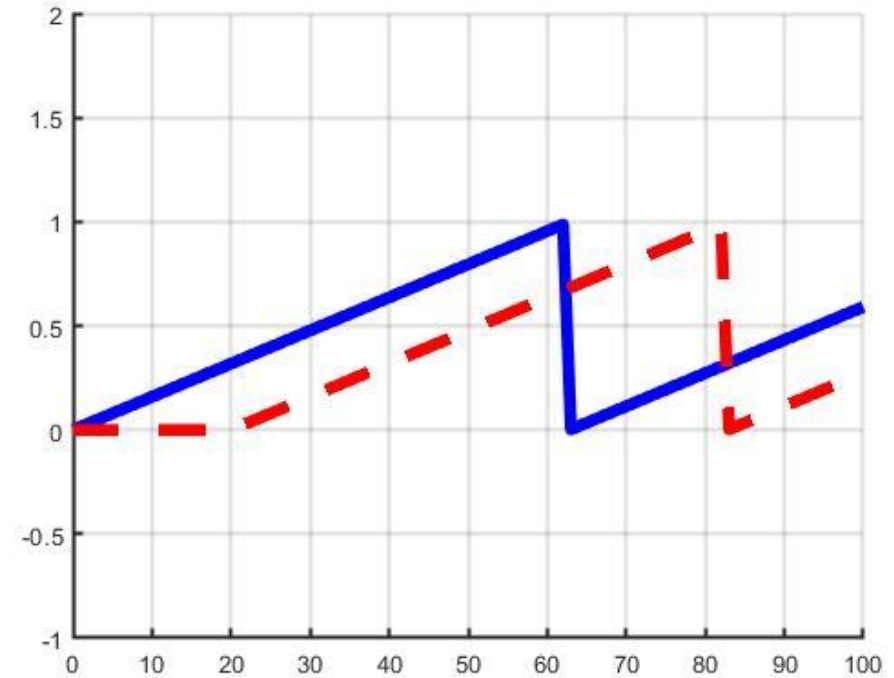
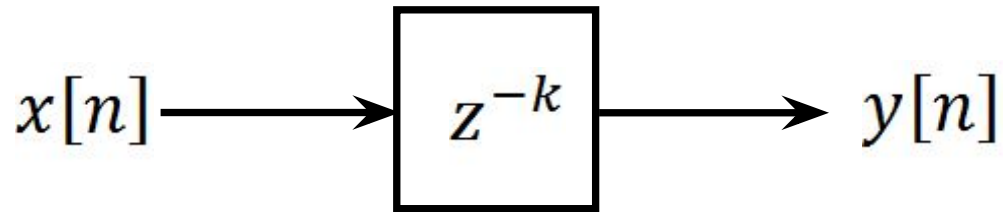
□ Summation  $y[n] = x[n] + f[n]$



## Elementary operators

□ Delay

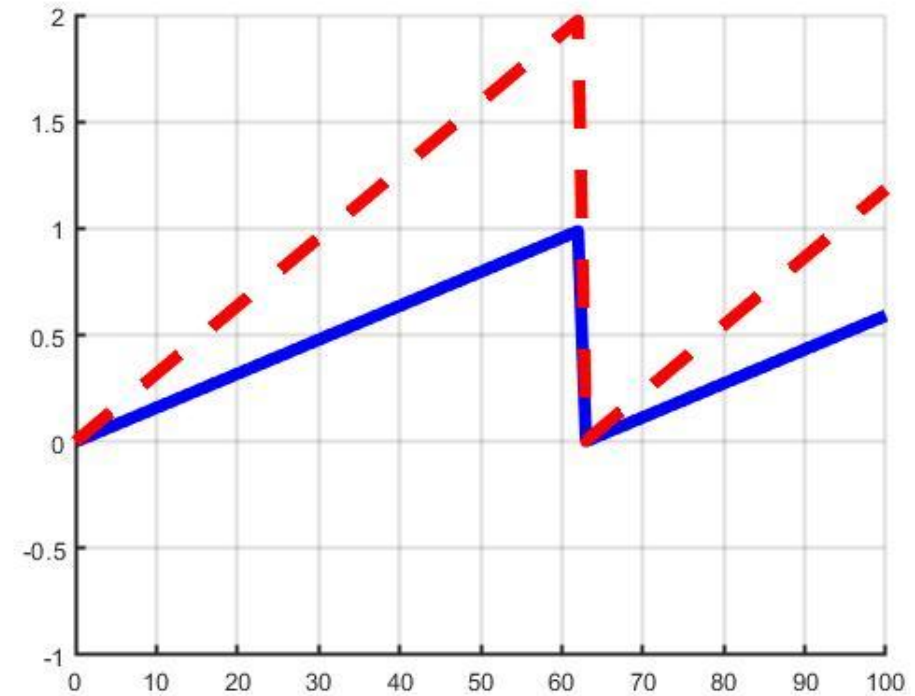
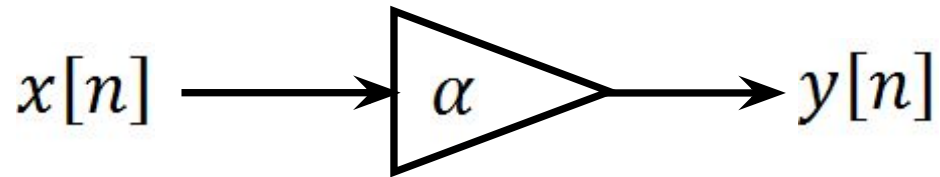
$$y[n] = x[n - k] \quad k \in \mathbb{Z}$$





## Elementary operators

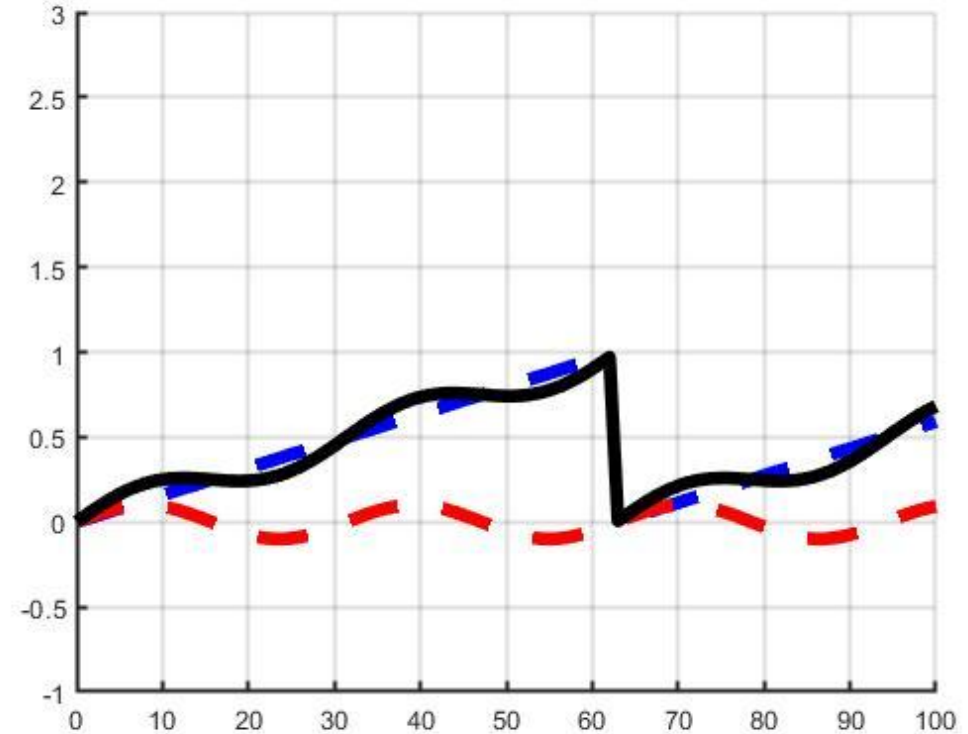
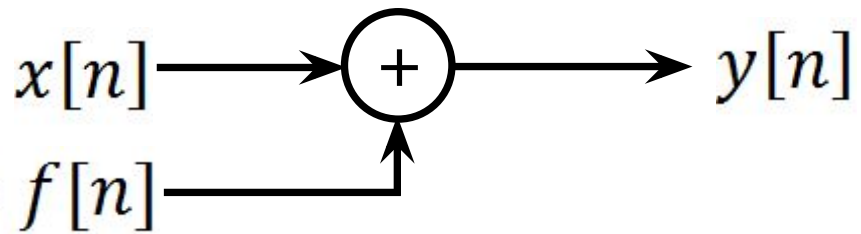
□ Scaling  $y[n] = ax[n]$

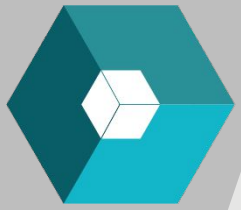




## Elementary operators

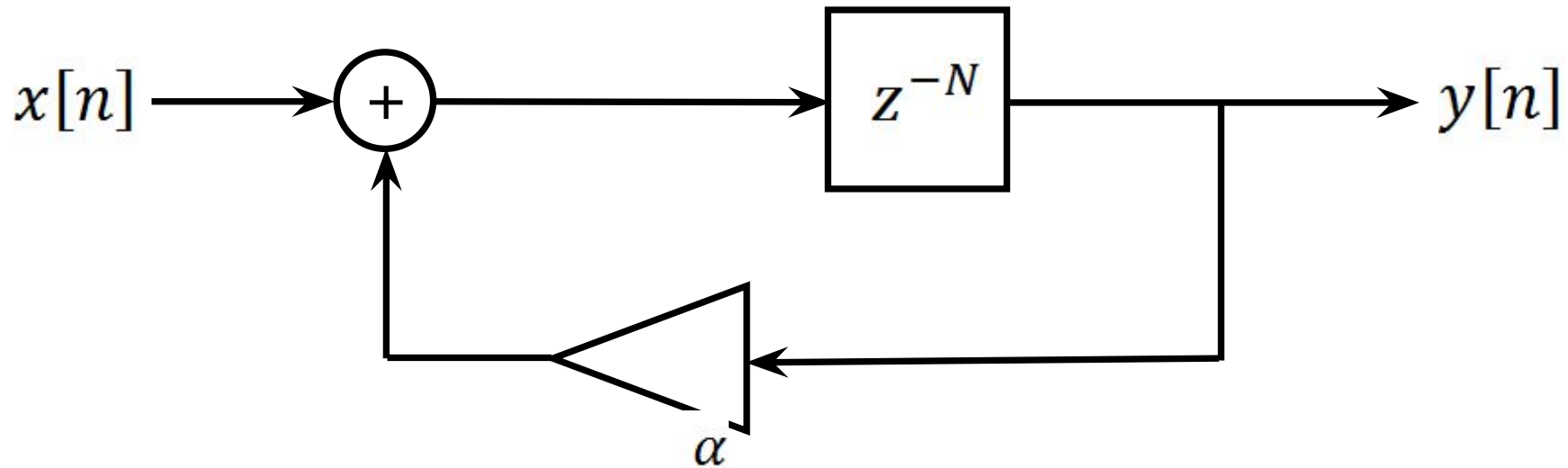
□ Summation  $y[n] = x[n] + f[n]$





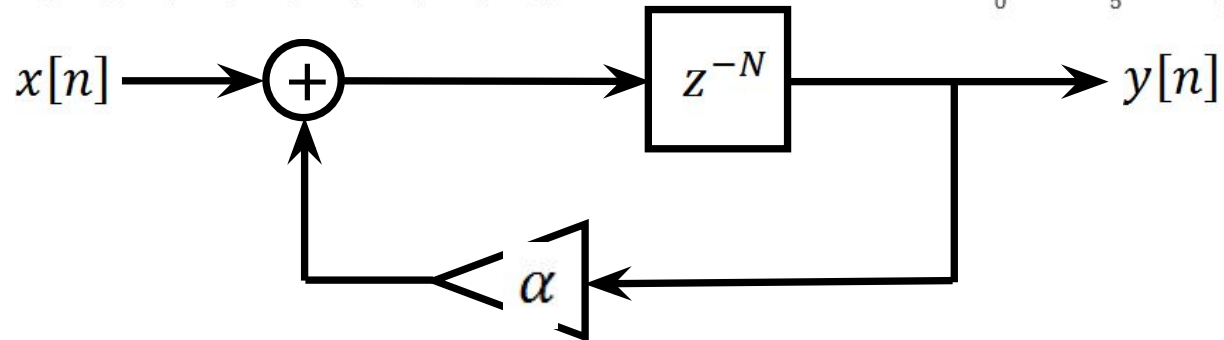
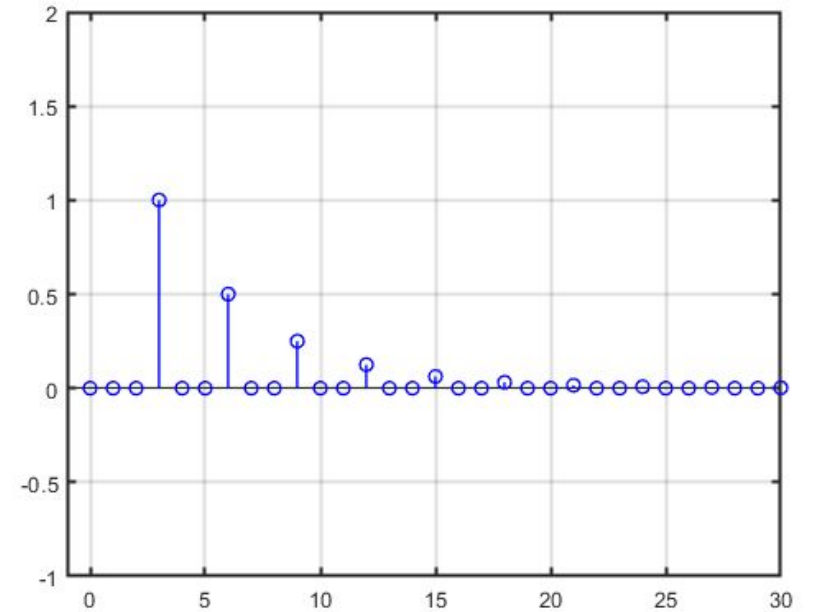
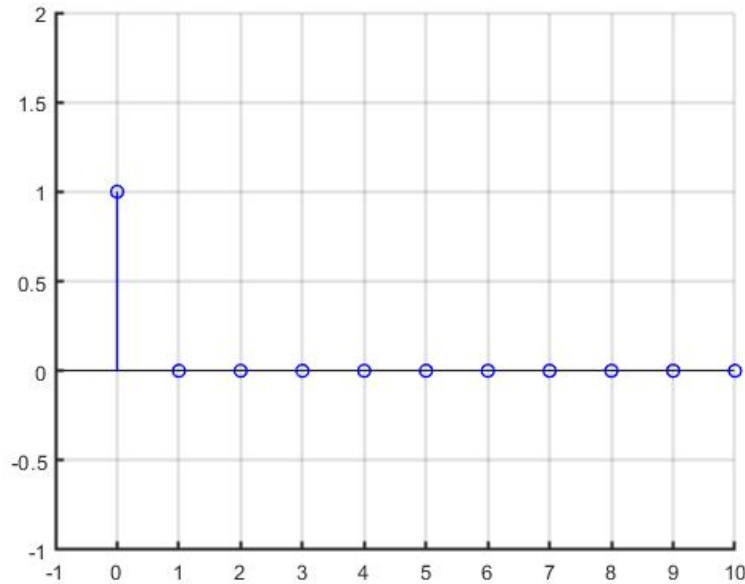
## Karplus-Strong algorithm - simplified

$$y[n] = x[n - N] + \alpha \cdot y[n - N]$$





## Karplus-Strong algorithm - simplified

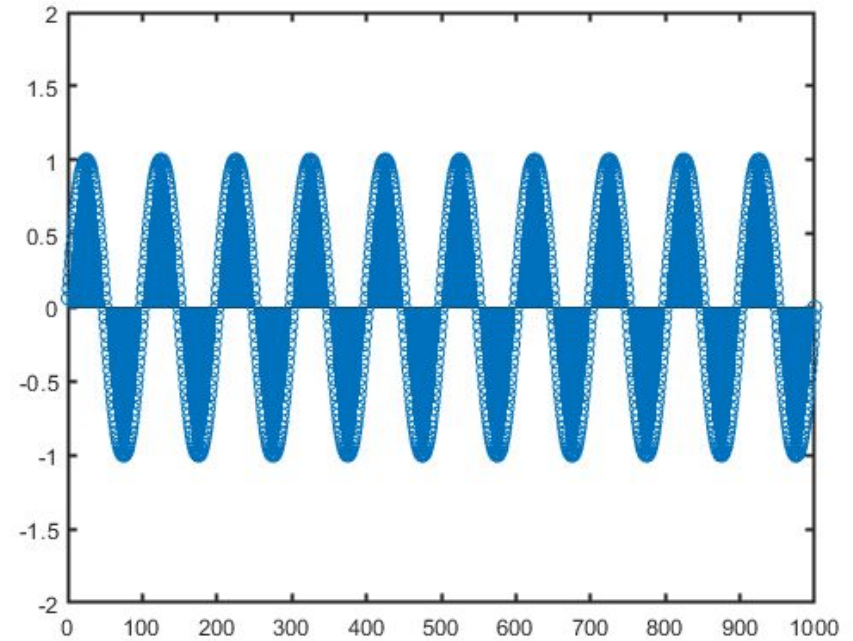
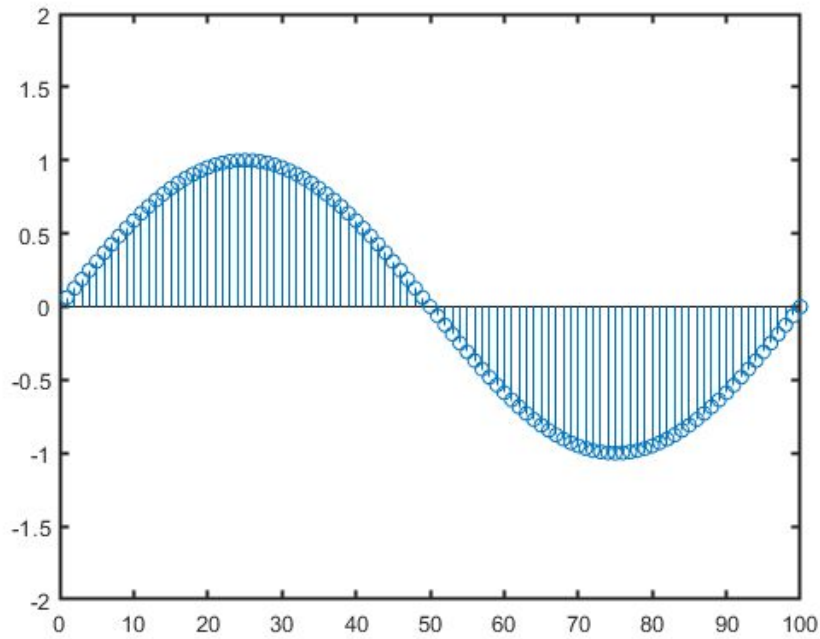


## Karplus-Strong algorithm - simplified

$$y[n] = x[n - N] + \alpha \cdot y[n - N]$$

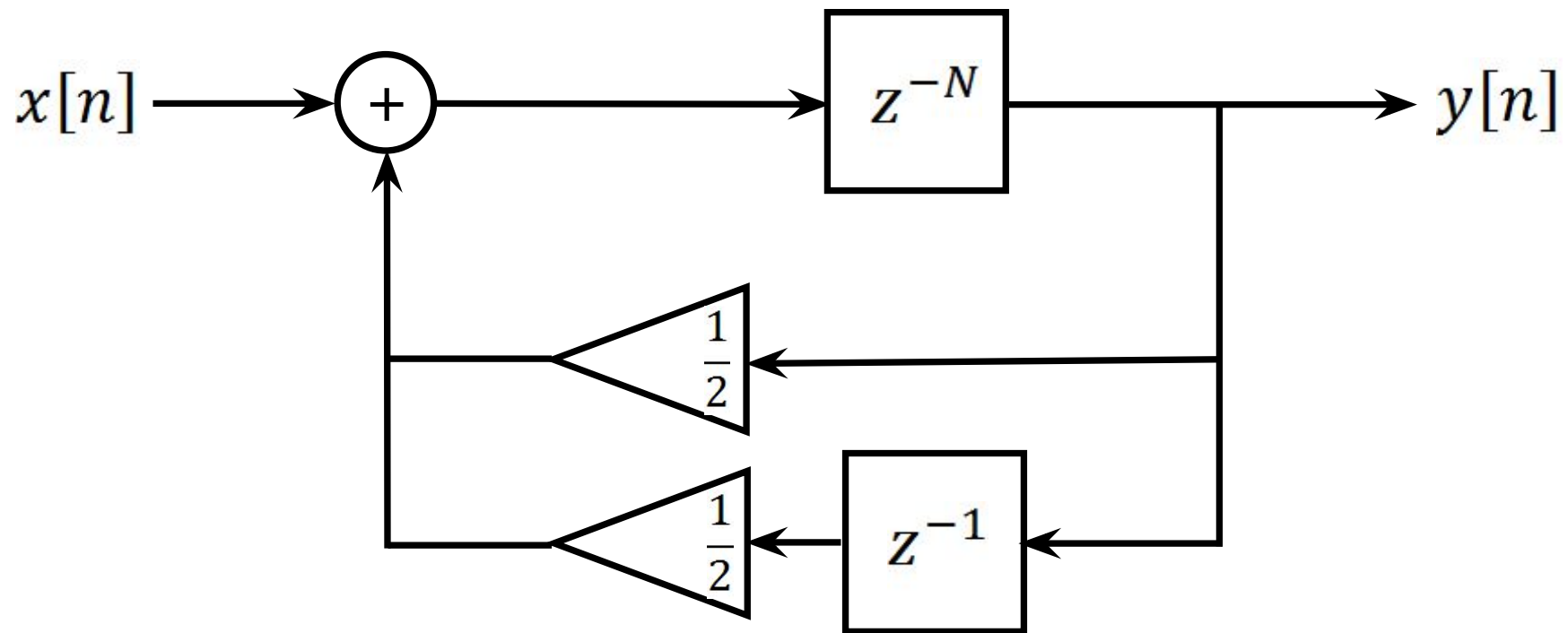
$$\bar{x}(n) = \sin\left(2\pi \frac{n}{100}\right)$$

$$N = 100 \quad \alpha = 1$$

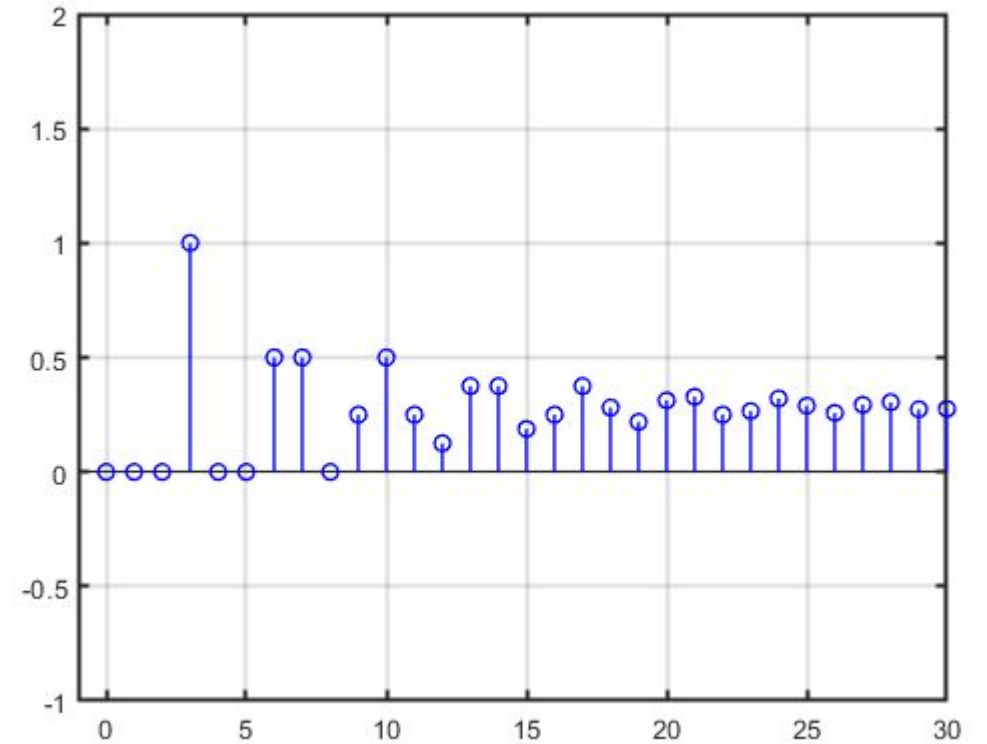
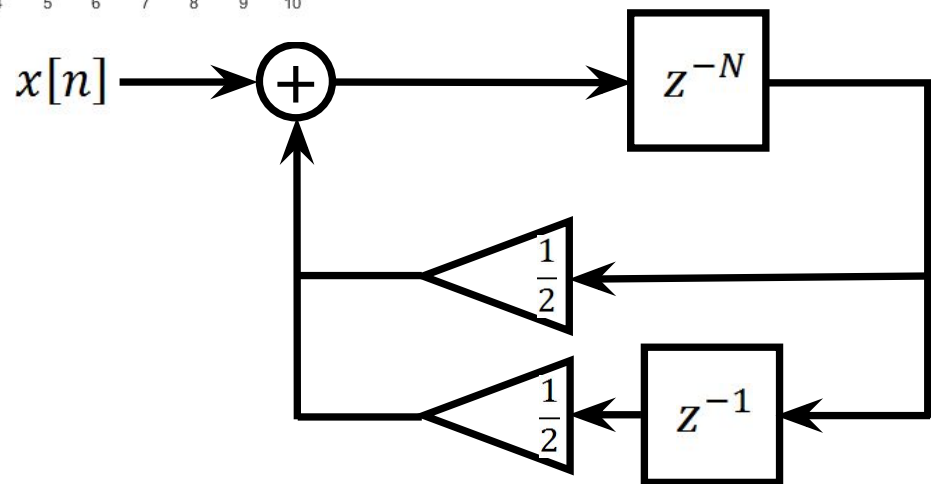
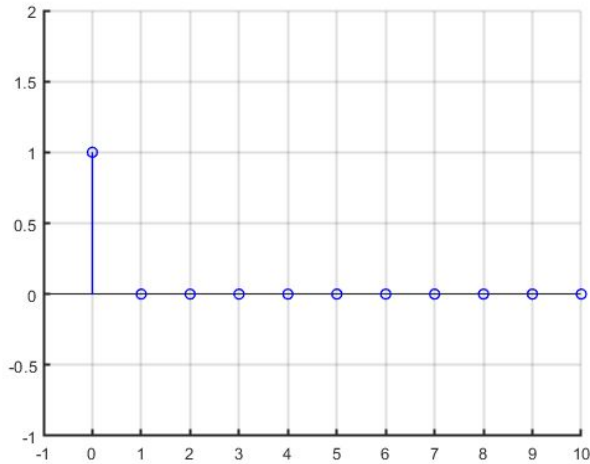


## Karplus-Strong algorithm

$$y[n] = x[n - N] + \frac{y[n - N] + y[n - (N + 1)]}{2}$$

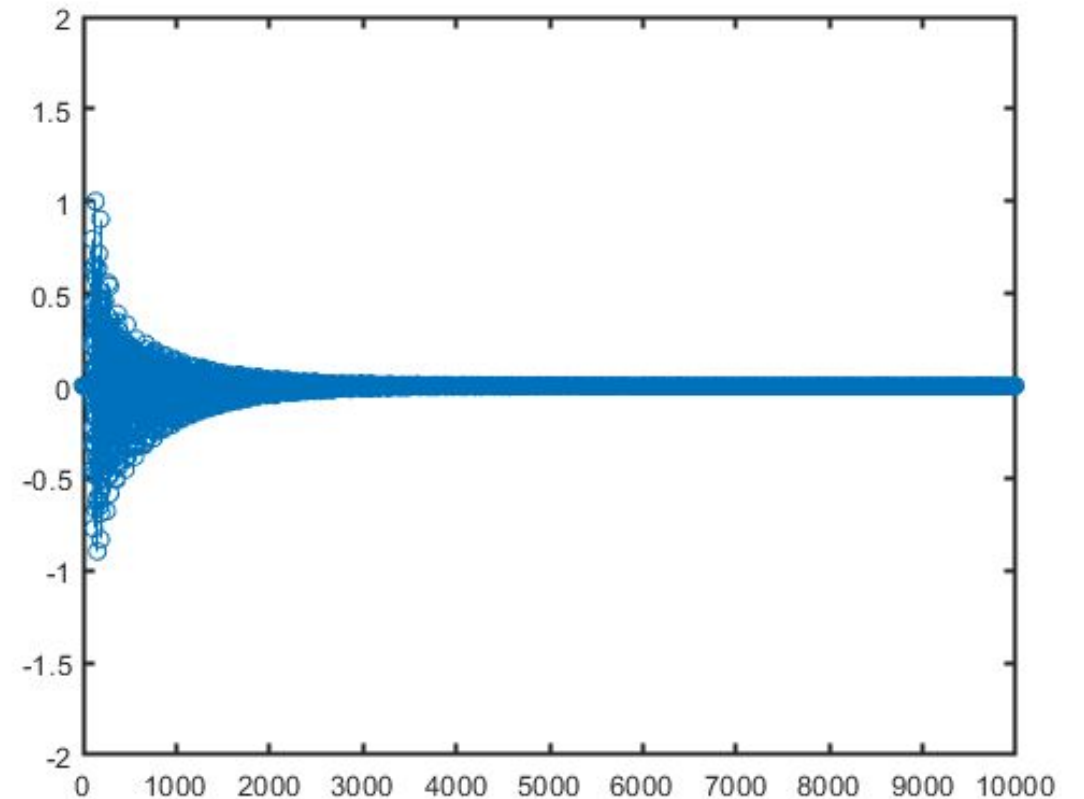
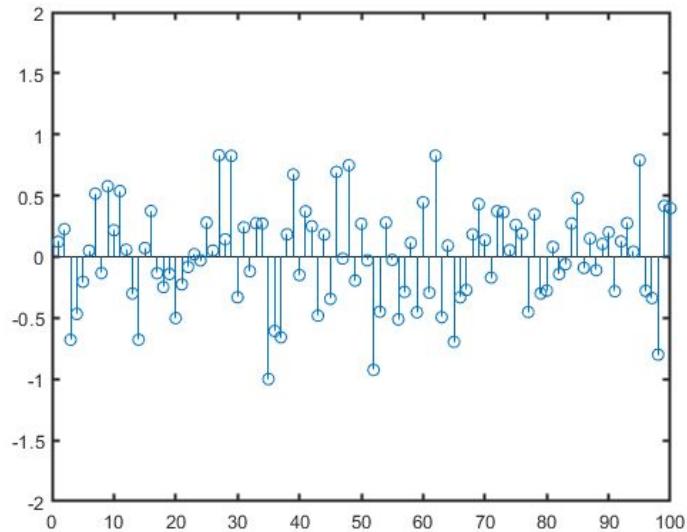


## Karplus-Strong algorithm

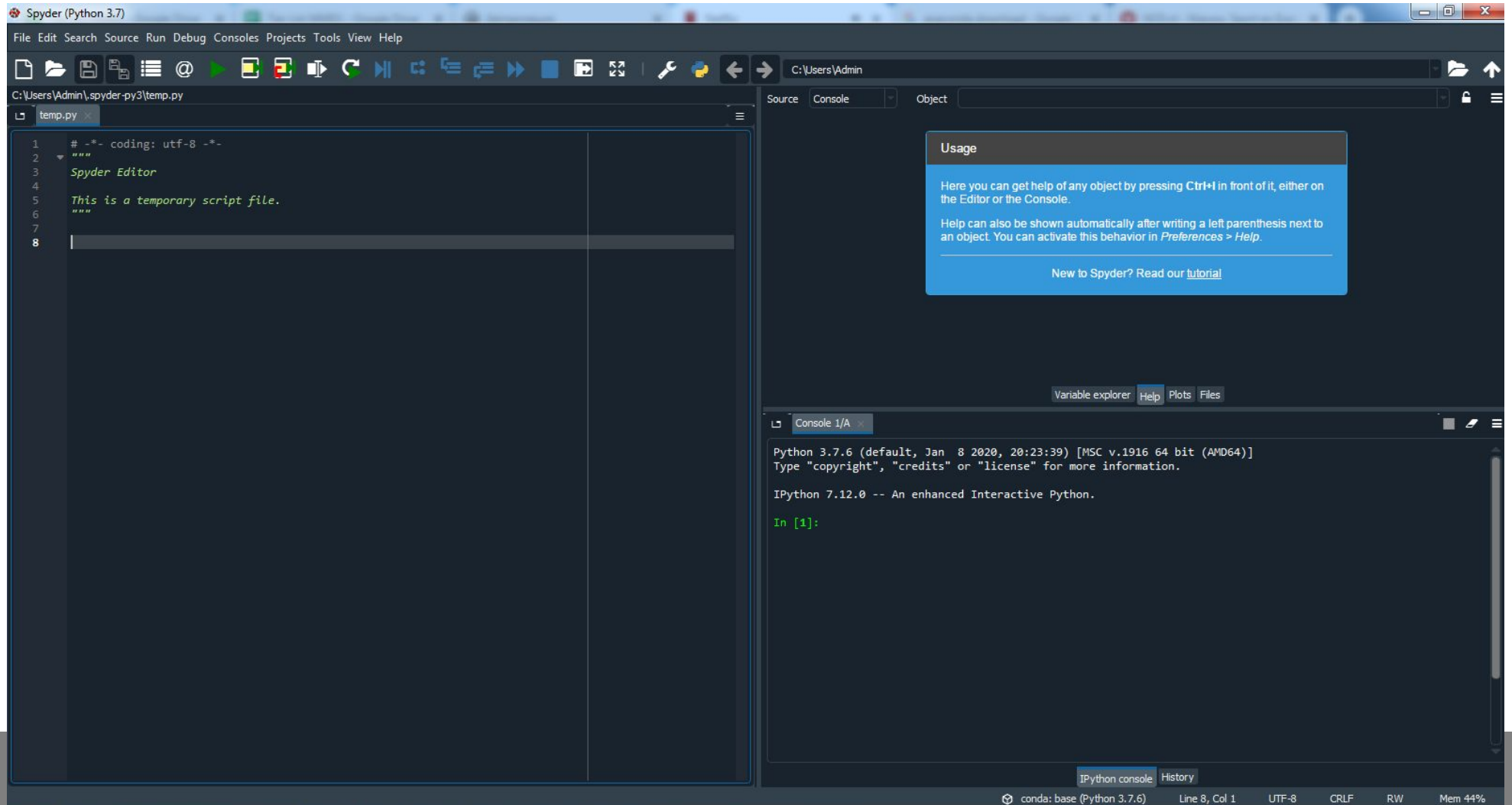


## Karplus-Strong algorithm

*$N$  controls the frequency (pitch)  
 $a$  controls the envelope (decay)  
 $\bar{x}(n)$  controls the color (Timbre)*



# Python 3 Anaconda - Spyder



The screenshot displays the Spyder Python IDE interface. The main editor window shows a file named `temp.py` with the following content:

```
1 # -*- coding: utf-8 -*-  
2 """  
3 Spyder Editor  
4  
5 This is a temporary script file.  
6 """  
7  
8
```

The interface includes a menu bar (File, Edit, Search, Source, Run, Debug, Consoles, Projects, Tools, View, Help), a toolbar with various icons, and a file explorer showing the current directory `C:\Users\Admin`. A "Usage" dialog box is open, providing information on how to access help for objects. The console window at the bottom shows the Python 3.7.6 environment and the IPython 7.12.0 prompt.

Usage

Here you can get help of any object by pressing **Ctrl+H** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

[New to Spyder? Read our tutorial](#)

Variable explorer Help Plots Files

Console 1/A

Python 3.7.6 (default, Jan 8 2020, 20:23:39) [MSC v.1916 64 bit (AMD64)]  
Type "copyright", "credits" or "license" for more information.

IPython 7.12.0 -- An enhanced Interactive Python.

In [1]:

IPython console History

conda: base (Python 3.7.6) Line 8, Col 1 UTF-8 CRLF RW Mem 44%



## Data types

Text Type: str

Numeric Types: int, float, complex

Sequence Types: list, tuple, range

Mapping Type: dict

Set Types: set, frozenset

Boolean Type: bool

Binary Types: bytes, bytearray, memoryview

## Data types

You can assign the datatype manually (like programming in C)

OR

You can assign the datatype automatically

```
Console 1/A  
Python 3.7.6 (default, Jan 8 2020, 20:23:39) [MSC v.1916 64 bit (AMD64)]  
Type "copyright", "credits" or "license" for more information.  
  
IPython 7.12.0 -- An enhanced Interactive Python.  
  
In [1]: x = 5  
...: print(type(x))  
<class 'int'>  
  
In [2]: x = float(5)  
...: print(type(x))  
<class 'float'>
```



## Data types

Some data types can interact

Try assigning the INTEGER 5 to variable 'x'

Try assigning the float 7 to variable 'y'

Multiply 'x' and 'y' and assign the result to 'z' (you type 'z=x\*y')

Determine the data type of 'z'

```
In [8]: x=5
In [9]: y=float(7)
In [10]: z=x*y
In [11]: print(type(z))
```



## Importing libraries

- For example try to print the number  $\pi$

```
In [1]: pi
Traceback (most recent call last):

  File "<ipython-input-1-f84ab820532c>", line 1, in <module>
    pi
NameError: name 'pi' is not defined

In [2]: math.pi
Traceback (most recent call last):

  File "<ipython-input-2-c49acc181da4>", line 1, in <module>
    math.pi
NameError: name 'math' is not defined
```



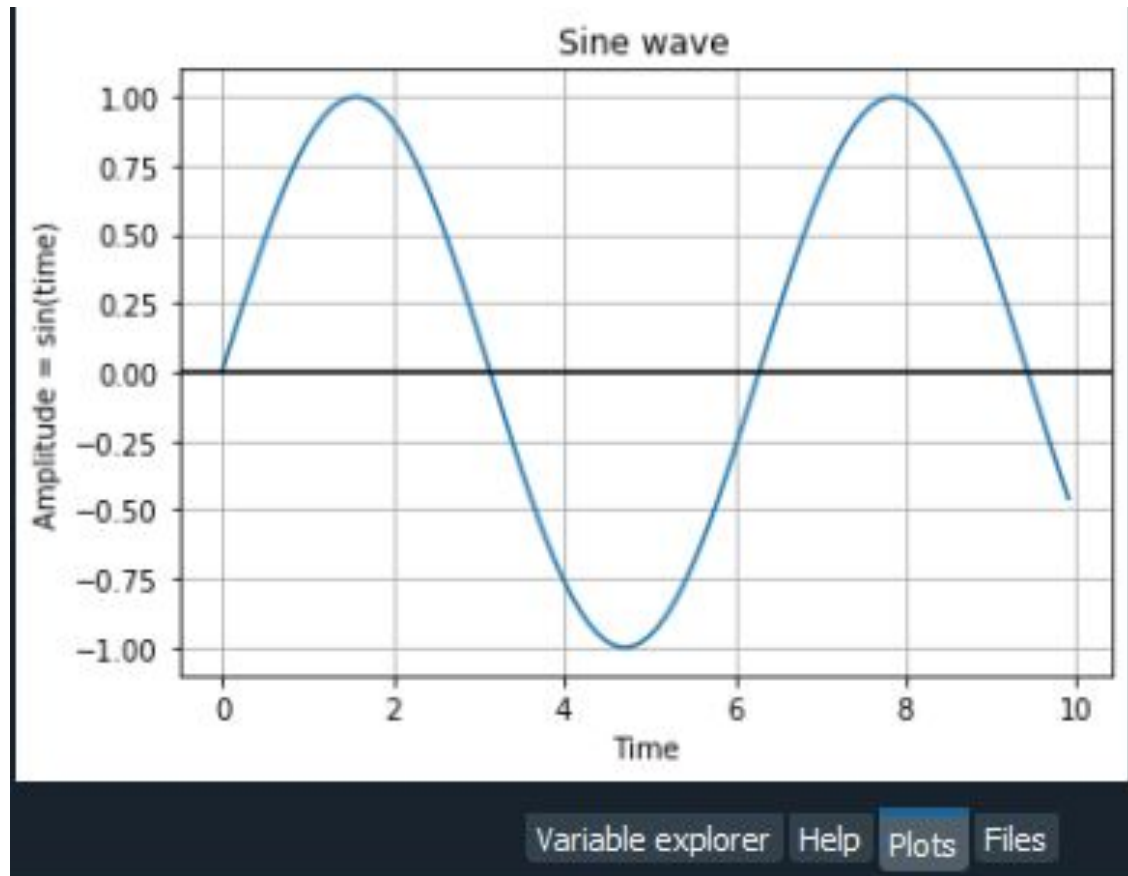
## Importing functions

- For example try to print the number  $\pi$

```
In [3]: import math
...: print(math.pi)
3.141592653589793
```



## Visualizing numbers – plotting a sine wave





## Visualizing numbers – plotting a sine wave

Import module numpy

Import the function pyplot from the module matplotlib

```
temp.py x
1 # -*- coding: utf-8 -*-
2 import numpy as np
3 import matplotlib.pyplot as plot
4
```



## Visualizing numbers – plotting a sine wave

Create time points for the x-axis

Numbers 0 until 10 with step size 0.01

(so you will get a range from [0 0.01 0.02 0.03 ... 1] )

```
# Get x values of the sine wave  
time = np.arange(0, 10, 0.1);
```



## Visualizing numbers – plotting a sine wave

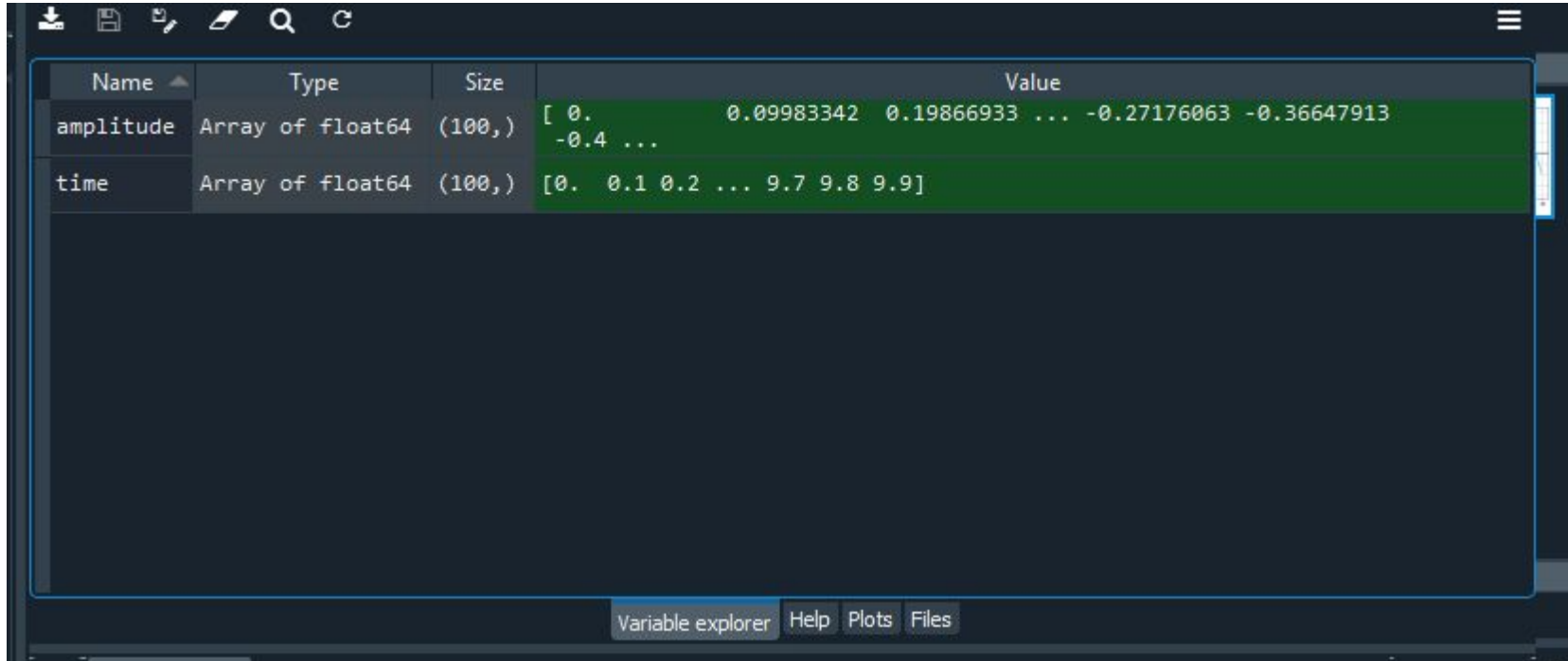
Create the amplitude points for each point in time

Use the sin function of the numpy module

(which you loaded and renamed as 'np')

```
# Amplitude of the sine wave is sine of a variable like time  
amplitude = np.sin(time)
```

## Visualizing numbers – plotting a sine wave



The screenshot shows a software interface with a variable explorer window. The window displays two variables: 'amplitude' and 'time'. Both are arrays of float64 type, each with a size of (100,). The 'amplitude' array contains values ranging from 0.0 to -0.36647913, and the 'time' array contains values ranging from 0.0 to 9.9.

Name	Type	Size	Value
amplitude	Array of float64	(100,)	[ 0. 0.09983342 0.19866933 ... -0.27176063 -0.36647913 -0.4 ...
time	Array of float64	(100,)	[0. 0.1 0.2 ... 9.7 9.8 9.9]

Variable explorer Help Plots Files



## Visualizing numbers – plotting a sine wave

Create the plot.

```
# Plot a sine wave using time and amplitude obtained for the sine wave  
plot.plot(time, amplitude)
```

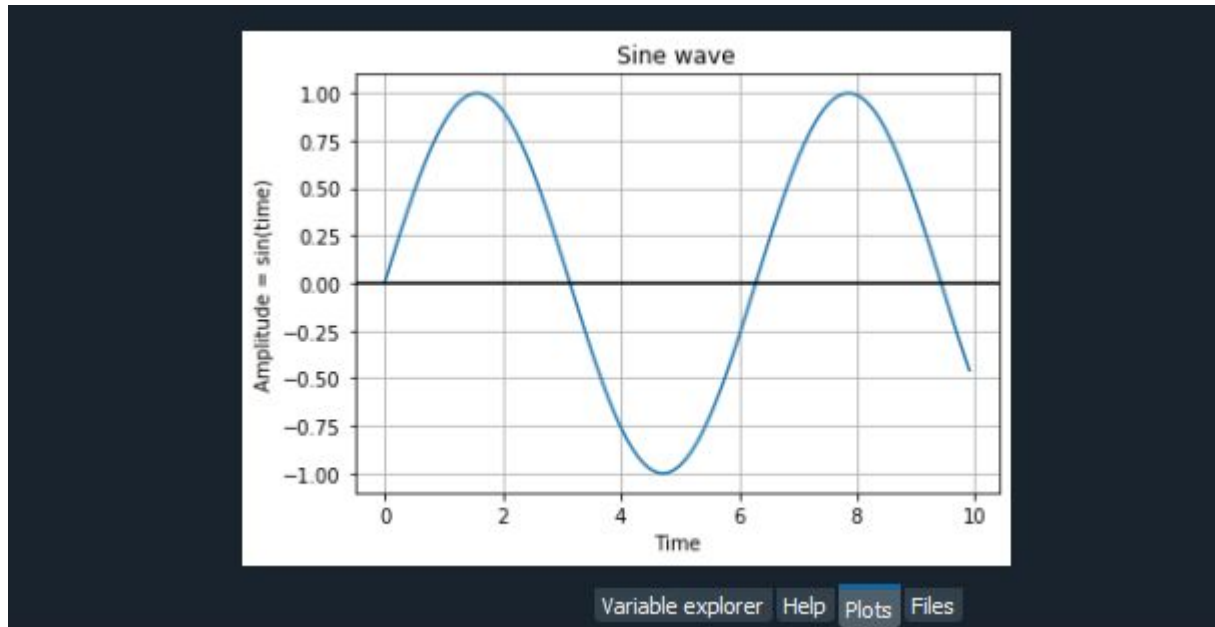
Create the mark up for the plot

```
# Give a title for the sine wave plot  
plot.title('Sine wave')  
# Give x axis label for the sine wave plot  
plot.xlabel('Time')  
# Give y axis label for the sine wave plot  
plot.ylabel('Amplitude = sin(time)')  
plot.grid(True, which='both')  
plot.axhline(y=0, color='k')
```

## Visualizing numbers – plotting a sine wave

Display the plot.

```
22  
23 plot.show()
```





## Python

Try to adhere to good practice!!

Code is written once, but read multiple times.

Find ground rules here:

<https://www.python.org/dev/peps/pep-0008/>



## Digital signals - limitations

$$1 + 1e20 - 1e20 = ?$$

$$1 + (1e20 - 1e20) = ?$$

Why?



- $(1e20 \cdot 1e20) \cdot 1e-20 = ?$
- $1e20 \cdot (1e20 \cdot 1e-20) = ?$
- $1e20 \cdot (1e20 - 1e20) = ?$
- $1e20 \cdot 1e20 - 1e20 \cdot 1e20 = ?$



## Data types - Integer (int)

```
In [16]: a = 100
```

```
In [17]: a.bit_length()  
Out[17]: 7
```

```
In [25]: bin(100)  
Out[25]: '0b1100100'
```

```
In [26]: bin(-4)  
Out[26]: '-0b100'
```

```
In [38]: a=1
```

```
In [39]: b=1
```

```
In [40]: a is b  
Out[40]: True
```

```
In [46]: a==b  
Out[46]: True
```

```
In [41]: a=257
```

```
In [42]: b=257
```

```
In [43]: a is b  
Out[43]: False
```

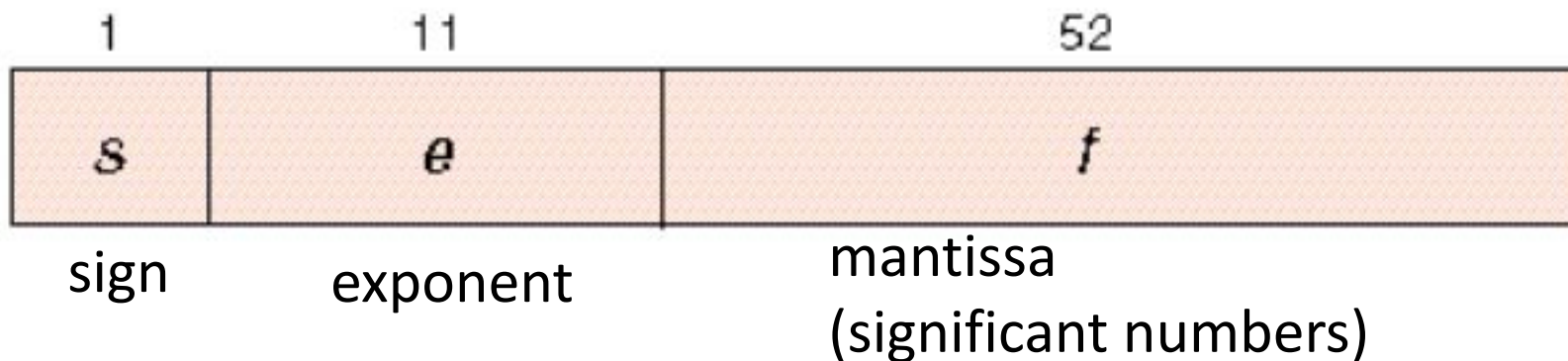
## Floating point - Single precision (single) / double precision (double)

double precision (64 bit) floating numbers

$$-123.4567890123 = -1.234567890123 e2$$

sign
mantissa
exponent

(significant numbers)





## Floating point - Single precision (single) / double precision (double)

double precision (64 bit) floating numbers



$$x = \pm (1 + f) \cdot 2^e$$

Giving the decimal interval for  $e$  as  
 **$-1022 \leq e \leq 1023$**

**Using offset binary representation**

With fraction  $f$

$$0 \leq f < 1$$

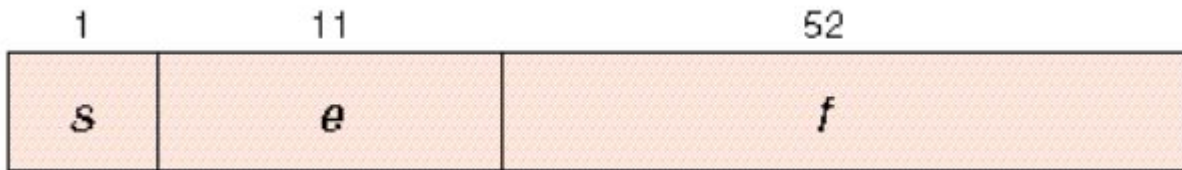
In the interval where  
 $2^{52} \cdot f$  is an integer

$$0 \leq 2^{52} \cdot f < 2^{52}$$



## Floating point - Single precision (single) / double precision (double)

double precision (64 bit) floating numbers



With exponent  $e$  as integer in the interval

$$-1022 \leq e \leq 1023$$

$$x = \pm (1 + f) \cdot 2^e$$

```
In [11]: a = float(2) ** 1023
```

```
In [12]: a = float(2) ** 1024
Traceback (most recent call last):
```

```
File "<ipython-input-12-a340186a386a>", line 1, in <module>
    a = float(2) ** 1024
```

```
OverflowError: (34, 'Result too large')
```

With fraction  $f$

$$0 \leq f < 1$$

In the interval where  $2^{52} \cdot f$  is an integer

$$0 \leq 2^{52} \cdot f < 2^{52}$$

```
In [9]: a = float(10) ** 308
```

```
In [10]: a = float(10) ** 309
Traceback (most recent call last):
```

```
File "<ipython-input-10-92873524ff56>", line 1, in <module>
    a = float(10) ** 309
```

```
OverflowError: (34, 'Result too large')
```



## Не сравнивайте числа с плавающей запятой

```
float f = 0.1;
while (f != 1.0) {
    f += 0.1;
}
```

```
f=0.2000000030
f=0.3000000119
f=0.4000000060
f=0.5000000000
f=0.6000000238
f=0.7000000477
f=0.8000000715
f=0.9000000954
f=1.0000001192
f=1.1000001431
f=1.2000001669
f=1.3000001907
f=1.4000002146
f=1.5000002384
f=1.6000002623
```

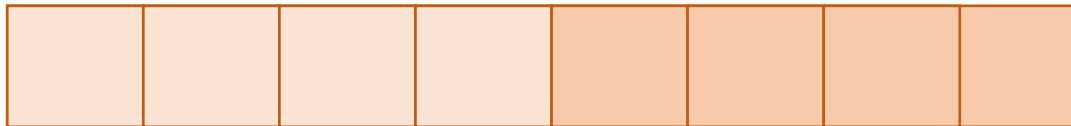
## **Будьте аккуратны с числами с плавающей запятой**

- Точность уменьшается по мере роста величины
- Ошибки переполнения и округления
  - Ракета-носитель «Ариан-5» 4 июня 1996 была подорвана на 34-й секунде полёта
    - Конвертация данных из 64-разрядного числа с плавающей запятой в 16-разрядное привела к зависанию компьютера
  - Точность ракет Patriot в 1991 (Война в Персидском заливе) была ограничена ошибкой округления



## Data types - Fixed point

A fixed decimal point. Often used in low level devices.



example

10001011

$$2^3 + 2^{-1} + 2^{-3} + 2^{-4} = 8.675$$

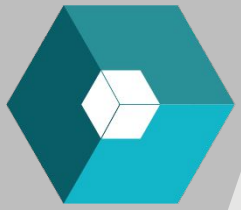
```
>>> from decimal import *
>>> getcontext()
Context(prec=28, rounding=ROUND_HALF_EVEN, Emin=-999999999, Emax=999999999,
        capitals=1, flags=[], traps=[Overflow, DivisionByZero,
        InvalidOperation])

>>> getcontext().prec = 7           # Set a new precision
```



## **5 steps of a method/function in any language**

- 1 Initialization
- 2 input
- 3 process
- 4 output
- 5 Termination



## 1) Define a function

```
def myFirstFunction(input):
```

```
    print("This will be the output, %s" %(input))
```

A screenshot of a code editor window titled 'myFirstScript.py'. The editor shows a Python script with a docstring and a function definition. The docstring includes creation and modification dates and the author's name. The function 'myFirstFunction' takes an 'input' parameter and prints a message using string formatting. The file explorer on the right shows the file structure, including a '\_\_pycache\_\_' folder and two Python files: 'KarplusStrong.py' and 'myFirstScript.py'.

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Sat Aug 29 18:53:26 2020
4  @author: Admin
5
6  Last Modified on Sat Aug 29 18:53:26 2020
7  @author: Admin
8
9
10 """
11
12
13 def myFirstFunction(input):
14     print("This will be the output,%s" %(input))
15
```



## Importing the function

```
In [53]: import myFirstScript
```

## Calling the function

```
In [55]: myFirstScript.myFirstFunction( 'aaaa' )  
This will be the output,aaaa
```



## Data types

Text Type: str

Numeric Types: int, float, complex

Sequence Types: list, tuple, range

Mapping Type: dict

Set Types: set, frozenset

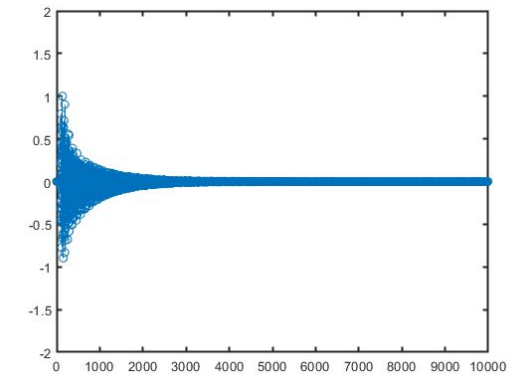
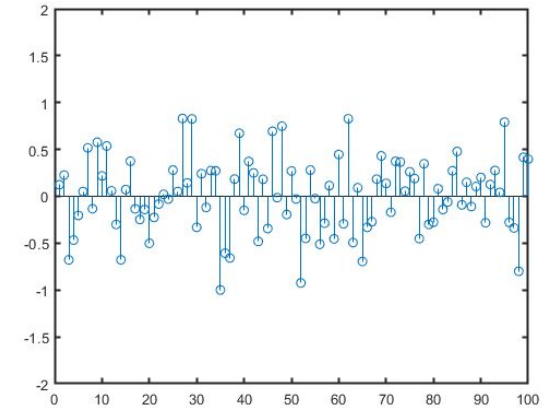
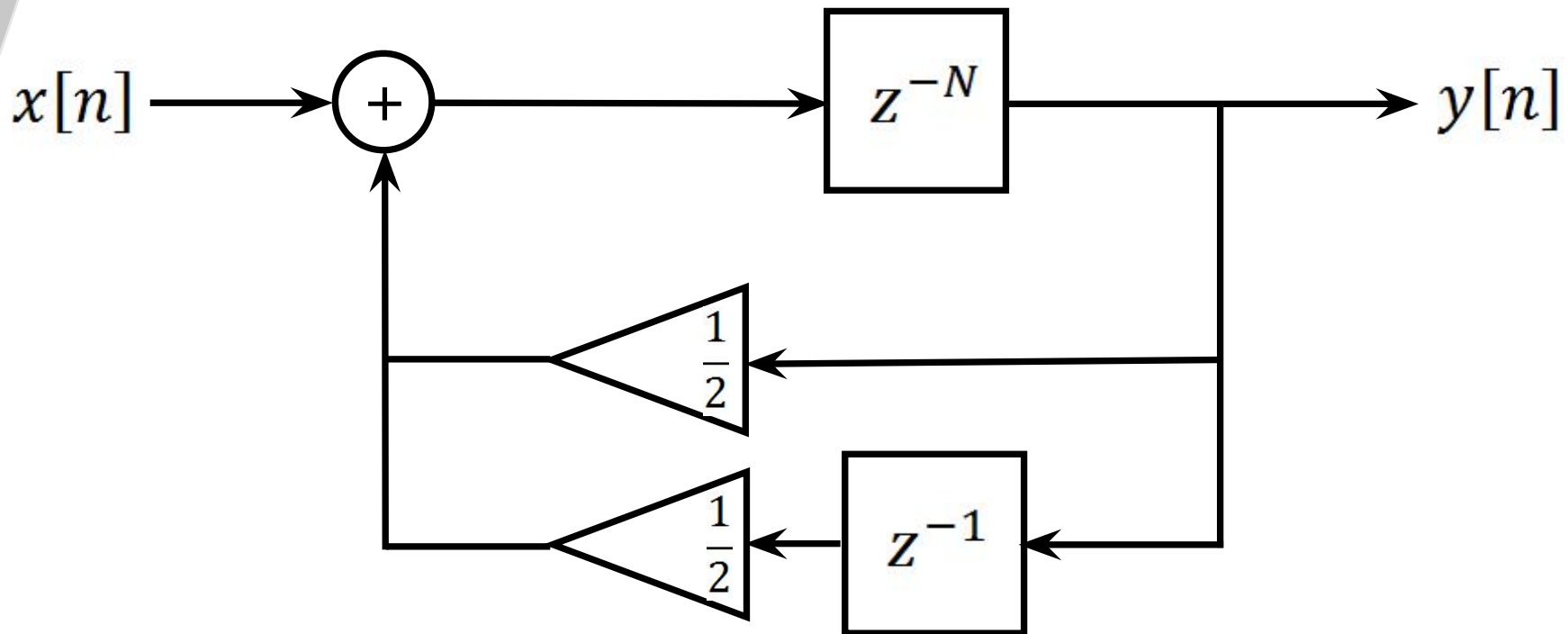
Boolean Type: bool

Binary Types: bytes, bytearray, memoryview



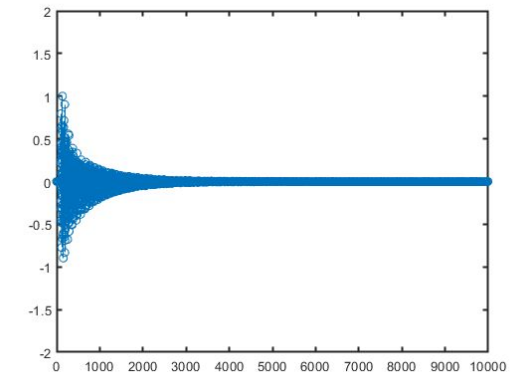
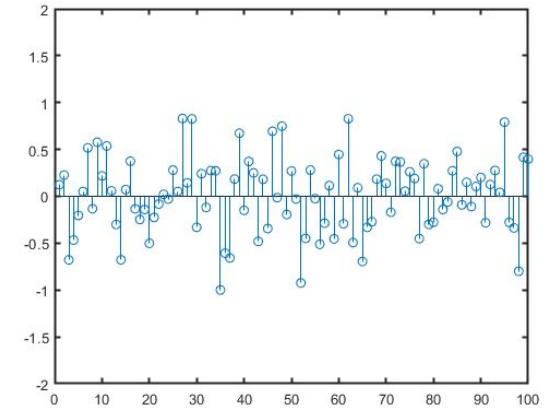
## Writing a Karplus-Strong algorithm in Python

$$y[n] = x[n - N] + \frac{y[n - N] + y[n - (N + 1)]}{2}$$



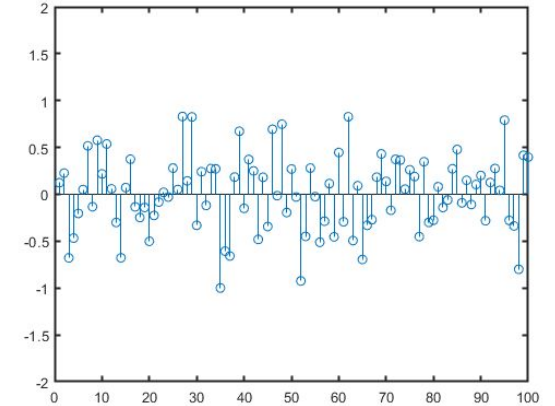
## Writing a Karplus-Strong algorithm in Python

- Generate a random input
- Process the input using the algorithm
- Show the Result
- Save the result



## Writing a Karplus-Strong algorithm in Python

- Generate a random input



[Understanding the Karplus-Strong with Python \(Synthetic Guitar Sounds Included\) | Frolian's blog \(flothsof.github.io\)](#)



## Data types

Text Type: str

Numeric Types: int, float, complex

Sequence Types: list, tuple, range

Mapping Type: dict

Set Types: set, frozenset

Boolean Type: bool

Binary Types: bytes, bytearray, memoryview

## Writing a Karplus-Strong algorithm in Python

- Generate a random input

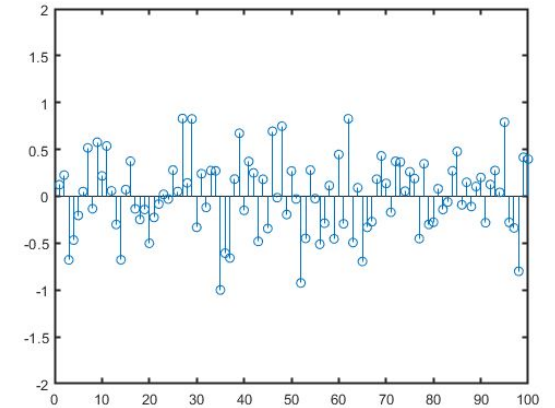
`numpy.random.uniform`

- Syntax

*`import numpy as np`*

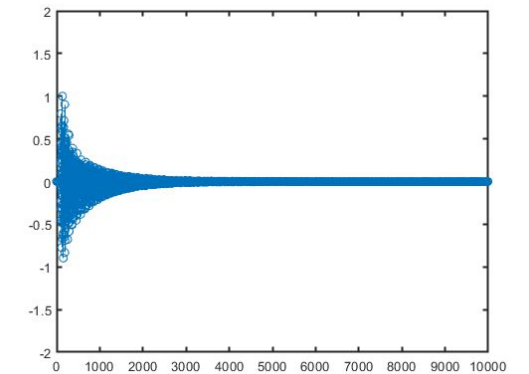
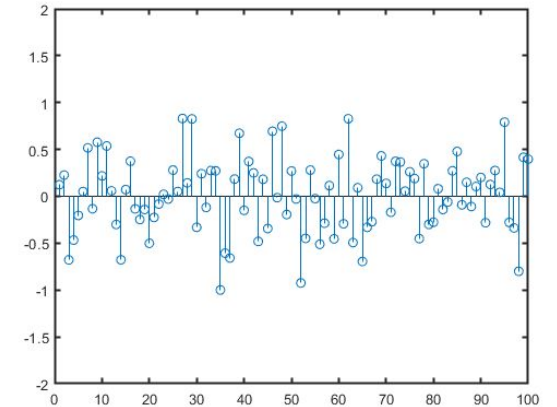
`np.random.uniform(low=0.0, high=1.0, size=None)`

`np.random.uniform(-1,1,1000)`



## Writing a Karplus-Strong algorithm in Python

- ~~Generate a random input~~
- Process the input using the algorithm
- Show the Result
- Save the result



## Writing a Karplus-Strong algorithm in Python

### First solution

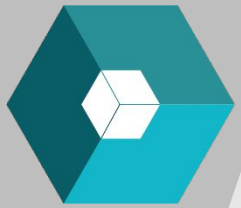
$$y[n] = x[n - N] + \frac{y[n - N] + y[n - (N + 1)]}{2}$$

- 1) Create variable **y** as a copy of input **x**;  
Append **N** zeros in front of **y**

**OR**

Create **y** as zeros for a length of **x** plus **N**  
Replace the indexed values in **y** with values of **x**

- 2) Write a for-loop, which goes through all indexes of **y**  
Replace indexed values of **y**, with a sum of the current value plus scaled previous values



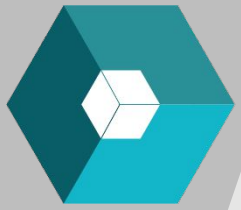
## Writing a Karplus-Strong algorithm in Python

### First solution

$$y[n] = x[n - N] + \frac{y[n - N] + y[n - (N + 1)]}{2}$$

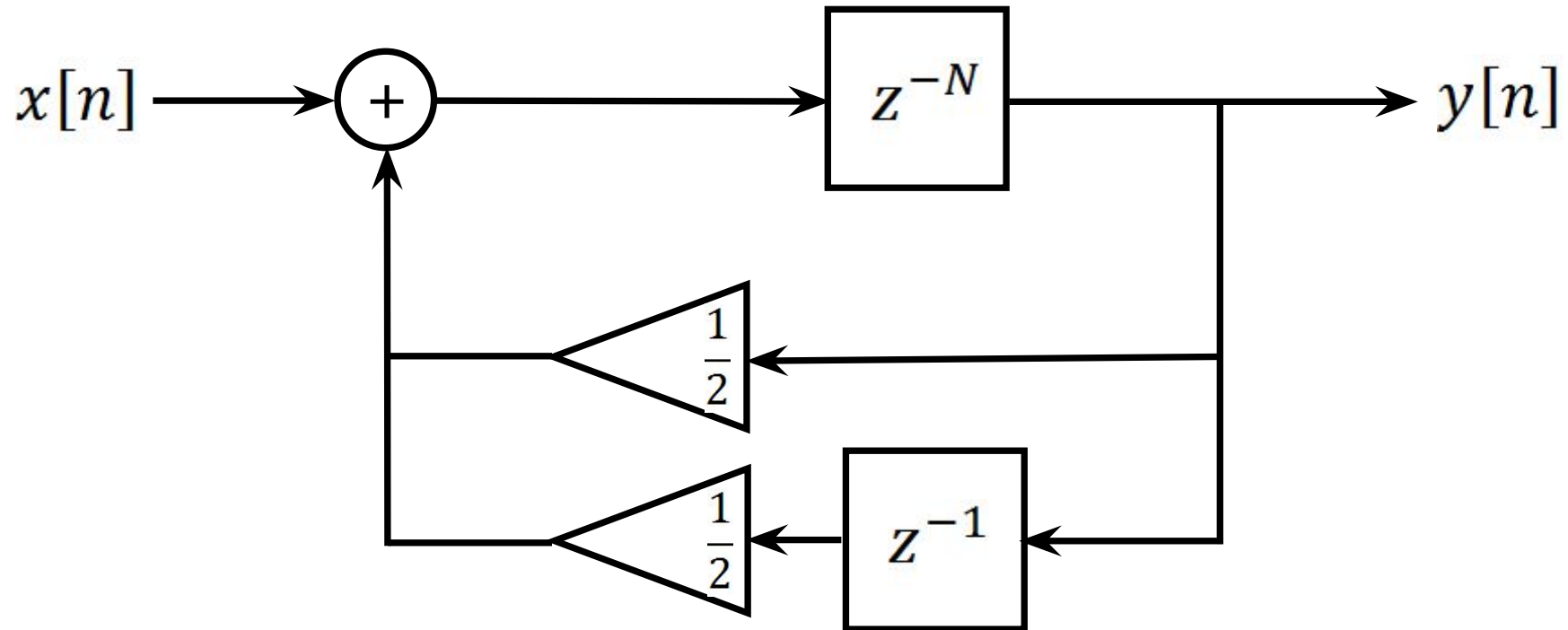
```
def KS1(signal, delayN, envelopeA, totalLength):  
    import numpy as np  
    # M = len(signal)  
    y = np.zeros((delayN))  
    y = np.append(y, signal)  
    y = np.append(y, np.zeros(totalLength - len(y)))  
  
    for index in range(delayN, totalLength):  
        y[index] = y[index] + envelopeA/2 * ( y[index - delayN] + y[index - (delayN+1)] )  
    return y
```





## Writing a Karplus-Strong algorithm in Python

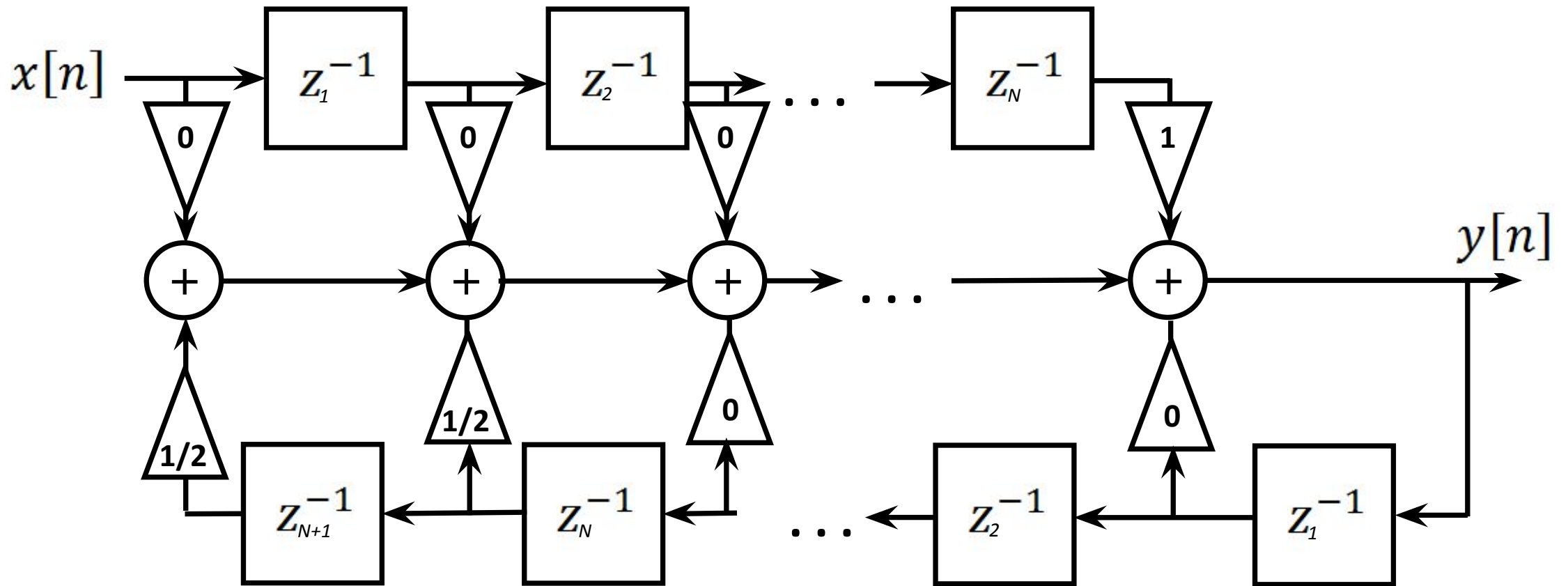
Second solution





## Writing a Karplus-Strong algorithm in Python

Second solution





## Writing a Karplus-Strong algorithm in Python

Second solution:

IIR Filter

**b-coefficients**

$b_0, b_1, \dots, b_{N-1}$  are 0

$b_N$  is 1

**a-coefficients**

$a_0$  is 1

$a_1 \dots a_{N-1}$  are 0

$a_N$  and  $a_{N+1}$  are 0.5



## Writing a Karplus-Strong algorithm in Python

```
#using a filter
randomInput=np.append(randomInput, np.zeros(totalLength-len(randomInput)))
b = np.append(np.zeros(delay), [1])
a = np.append([1], np.zeros(delay-1))
a = np.append(a, [-alpha/2, -alpha/2])
outputSignal = signal.lfilter(b, a, randomInput)
plot.plot(outputSignal)
plot.show()
```



## Save your output

```
fs = 44100 # 44100 samples per second
audio = outputSignal * (2**15 - 1) / np.max(np.abs(outputSignal))
audio = audio.astype(np.int16)
write('test.wav', 44100, audio)
```

## Homework Task 1

- 1) Write your own function and generate outputs (for example your own random number generator, a lookup table etc.)
- 2) Change the code to match «good practice»

<https://www.python.org/dev/peps/pep-0008/>

- 3) Create a function that has multiple outputs, and optimize the memory usage and processing speed.
- 4) Retrieve the outputs from the function and display them (or plot them)

## Полезная литература

- Методы и техника обработки сигналов при физических измерениях (2 тома)
- Signal processing for 5G
- Multi-Carrier Communication Systems with Examples in MATLAB: A New Perspective
- An Image Processing Tour of College Mathematics Yevgeniy V. Galperin
- Fundamentals of IMAGE, AUDIO, and VIDEO PROCESSING Using MATLAB: With Applications To PATTERN RECOGNITION
- Introduction to Modeling and Simulation with MATLAB and Python
- PySDR: A Guide to SDR and DSP using Python
- Python documentation