Hands on OFDM crash course

(**O**rthogonal **F**requency-**d**ivision **M**ultiplexing: standard modulation format for 5G New Radio)

"Preach the Gospel at all times. Use words if necessary."
St. Francis of Assisi

"Talk is cheap. Show me the code "
Linus Torvalds

“Simplicity is the ultimate sophistication.”
Leonardo da Vinci

NOKIA

Organization, Way of Work

You should always **submit a full set of slides**, regardless of the number of solved ones.

The **name of** the presentation **file** should look like this: Surname_First_NumerLastSolvedSlide, (do not use Polish letters), example "Zoladz_Miroslaw_50.pptx"

Organization, Materials content and modification

Most of the slides contain screenshots of the **scripts to be completed**.

Their **text versions** can be found in the appendix attached to the presentation.

The paths to the script files are given either directly **on the slide or in the slide notes**

Problem solutions should be placed **directly in the presentation**.

In the case of scripts, **screenshots of their solutions should be placed ON the task scripts**.

Screenshots should contain **ONLY code and line numbers**.

Please **don't** give **console screenshots** unless it is explicitly stated in the command.

Organization, Materials content and modification, cont.

In addition, please:

- do **not remove/move** any elements of the presentation, in particular slides,
- do **not remove my comments** from the code,
- **don't comment out lines** of code (either something is or isn't),
- **don't use** comprehension lists or maps, just plain for.

Please **ask questions via chat** or in a meeting, **not on slides**

Python tool

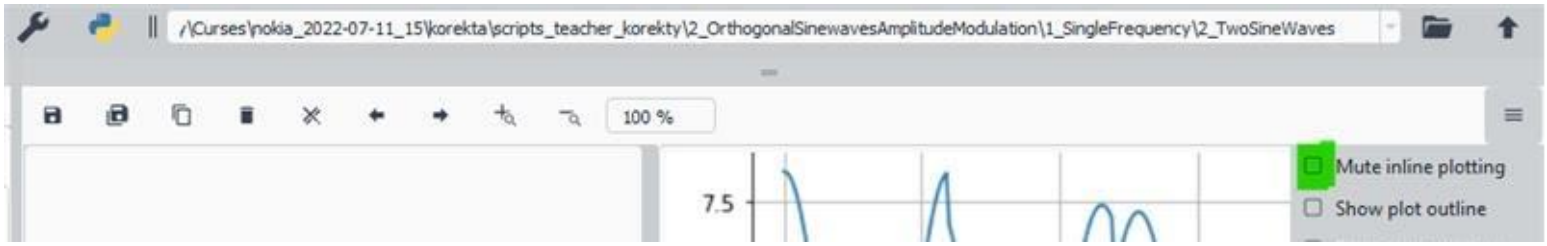
Preferred environment : https://www.anaconda.com/products/distribution

(Other tools allowed but we do not support you in case of problems)

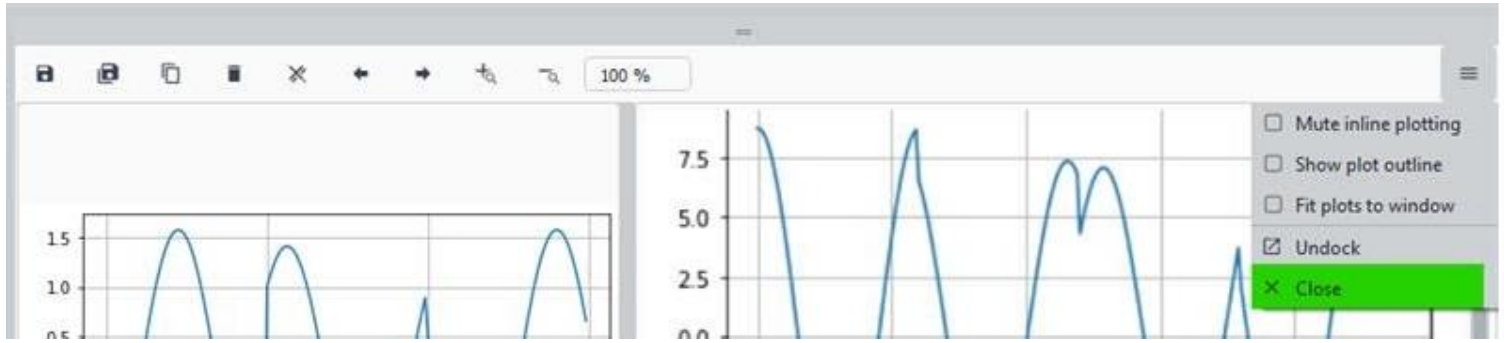
Preferred application: "Spider“ (from Anaconda)

Preferred settings: Prints and pot in the same window**:**

1) Uncheck "Mute..“



2) Close all panels in Console



Useful shortcuts

Console clear: Ctrl-l

Code block commenting\uncommenting : select blok, press “ctrl-1”

Course plan

1. Sinewaves orthogonality
2. Orthogonal sinewaves amplitude modulation
3. Discrete Fourier Transformation

Dot product of vectors

Dot product of signals

Sum of periodical signals

Dot product of sinusoids

2D vectors

- Dot product formula
- Orthogonal examples
- Dot product vs. angle

Vector as a list of numbers

- 2D example
- 3D example

Vector as a signal - signal as a vector

Dot product of signals

Signal's similarity

Dot versus phase shift

Dot versus amplitudes

Interferences

NOKIA

Dot product **formula**

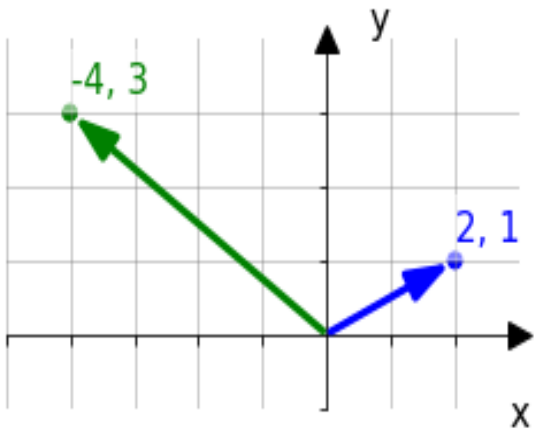
Vector coordinates: b(blue) = (x, y); g(green) = (x, y)

Product formula: b*g = (x*x) + (y*y);

input: vectors,

output: scalar <- sum of multiplication of corresponding coordinates

g = (-4,3), b = (2, 1)

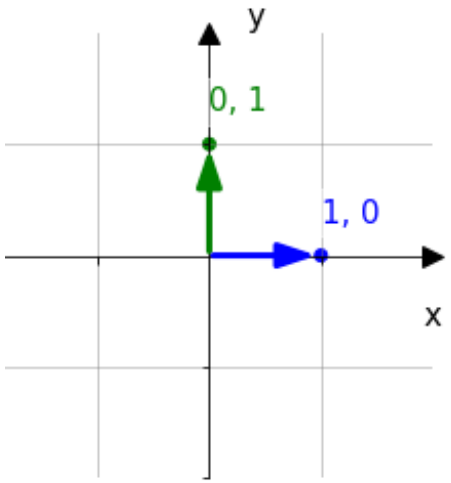


(-4, 3) * (2, 1) = (-4*2)+(3*1) = -5

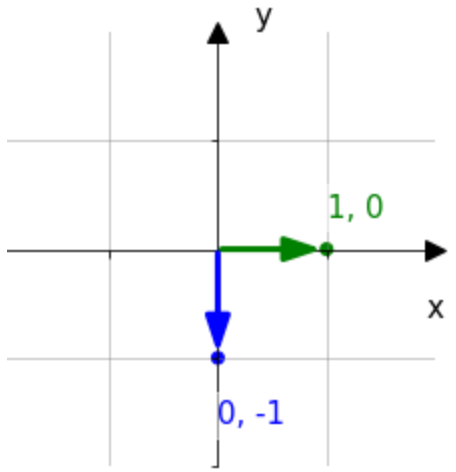
NOKIA

teacher\app_dot_2d_draw_and_calc.py

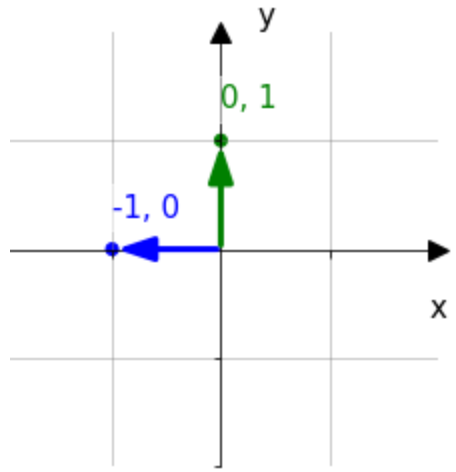
Dot Product: **Orthogonal vectors examples**



(x, y) * (x, y) = (0, 1) * (1, 0) = (0*1)+(1*0) = 0



(1*0)+(0*-1) = 0



(0*-1)+(1*0) = 0

In all cases dot product equals to zero.

Maybe because one of the corresponding coordinates is always zero ?

0, 1	1, 0	0, 1
1, 0	0,-1	-1, 0
0, 0	0, 0	0, 0

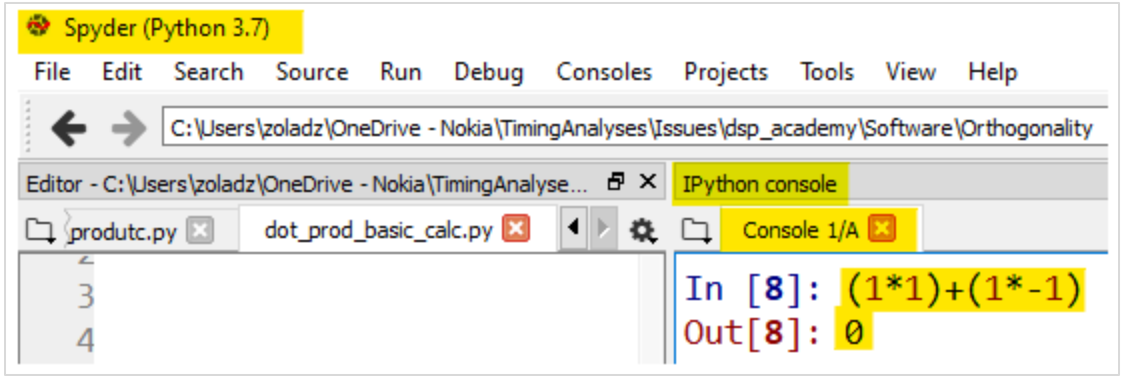
NOKIA

teacher\app_dot_2d_draw_and_calc.py

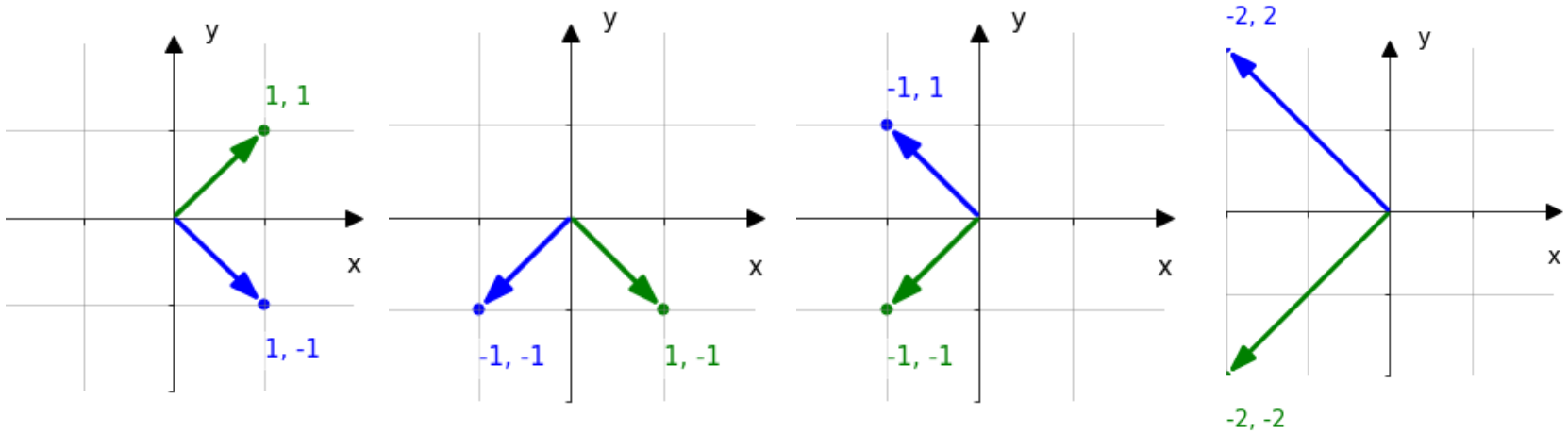
Dot Product: Orthogonal vectors examples: **Nonzero coordinates**

Calculate dot product of following vectors.

Use Spider REPL



REPL (stands from **Read–Eval–Print Loop**),
A simple interactive computer programming environment that takes single user inputs, executes them, and returns the result to the user. [Wikipedia]



Calculations	(1*1)+(1*-1)	(-1*1)+(-1*-1)	(-1*-1)+(1*-1)	(-2*-2)+(2*-2)
result	0	0	0	0

Still zero ? Maybe because we use the coordinates of the same absolute value (1 or 2)?

NOKIA

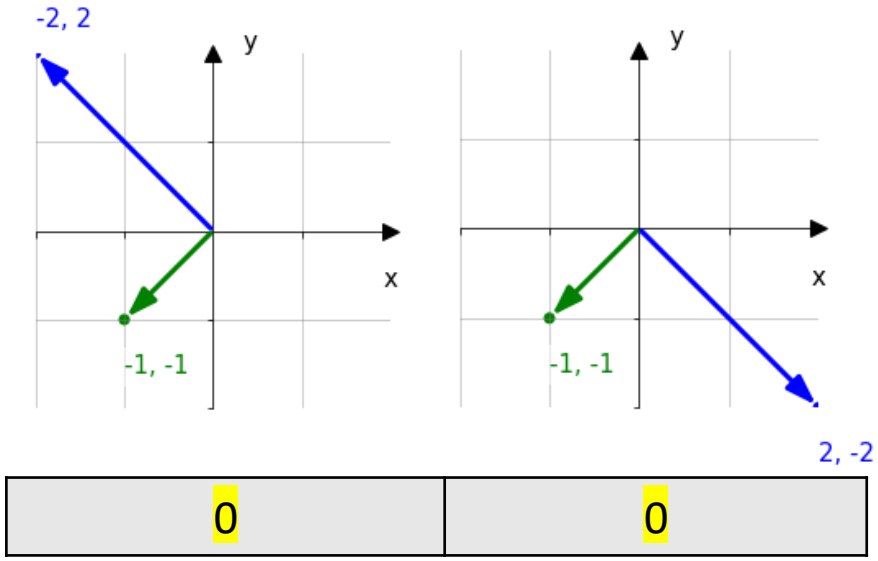
teacher\app_dot_2d_draw_and_calc.py

Dot Product: Orthogonal vectors examples: Nonzero coordinates, **cont.**

Calculate dot product of following vectors.

To work more efficiently instead of REPL use script on the right

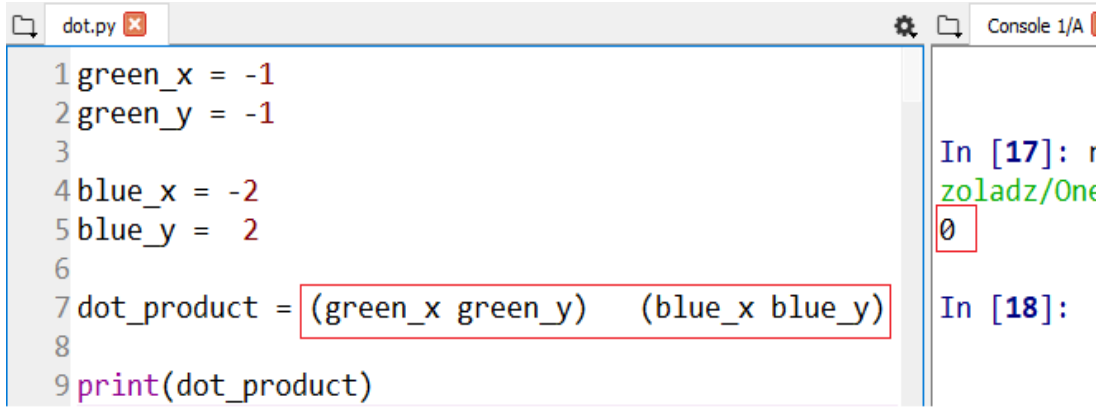
Note: correct mistakes in dot product formula



Still zero ? Maybe because we use the same absolute value of a **single vector** coordinates (1 or 2)?

```
green_x = -1
green_y = -1

blue_x = -2
blue_y =  2

dot_product = (green_x+blue_y) * (green_y+blue_y)

print(dot_product)
```

Fix dot product formula
Run the script (F5)
Find result in console



NOKIA

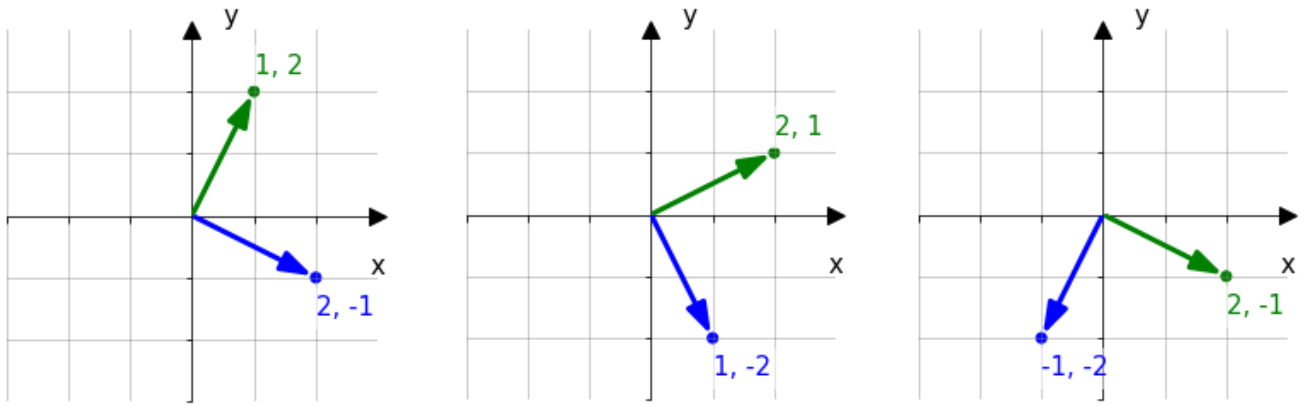
Dot Product: Orthogonal vectors examples: Nonzero coordinates, **cont.**

Calculate dot product of following vectors.

Use script with, more concise, vector coordinates notation (->).

(Use the same file)

```
1  green = [1,2]
2  blue = [2,-1]
3
4  dot_product = (green[0]*green[1]) + (blue[0]*blue[1])
5  print(dot_product)
6
7
```



0	0	0

Still zero ? Maybe because we use integer coordinates?

NOKIA

Dot Product: Orthogonal vectors examples: Nonzero coordinates, **cont.**

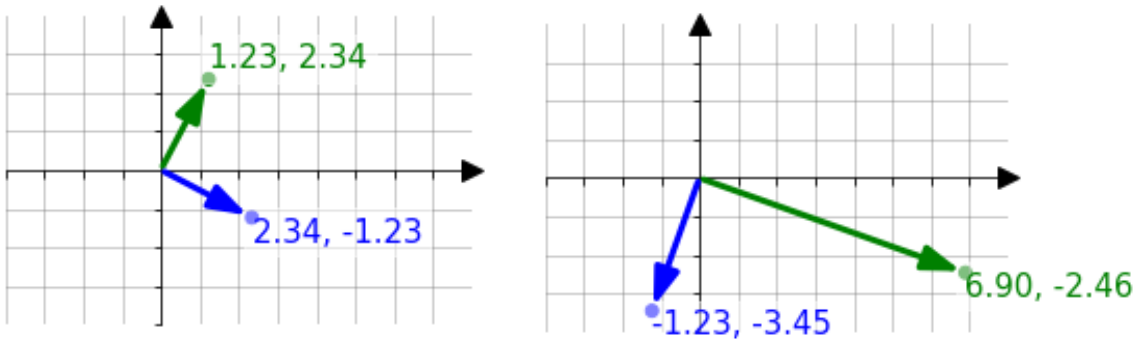
1. Calculate dot product of following vector configurations.

Use script version with “for” loop for iteration through vector indices. In this case these are 0 and 1.

This code works for vectors of any length.

```
green = [ 2,-1]
blue  = [-1,-2]

dot_product = 0
for i in range(len(green)): # i = 0,..len(green)-1
    dot_product += green[i] * blue[i]

print(dot_product)
```

1_SinewavesOrthogonality\1_DotProductOfVectors\3_dot_calc_**loop**_usage.py



0.0	0.0

2. Avoid loop-ing by using “**sum**” function from

```
import numpy as np

green = np.array([ 2,-1])
blue  = np.array([-1,-2])

dot_product = np.sum(green*blue)

print(dot_product)
```

1_SinewavesOrthogonality\1_DotProductOfVectors\3_dot_calc_**sum**_usage.py

Evidently dot product of any perpendicular vectors equals to zero. How about other angles ?

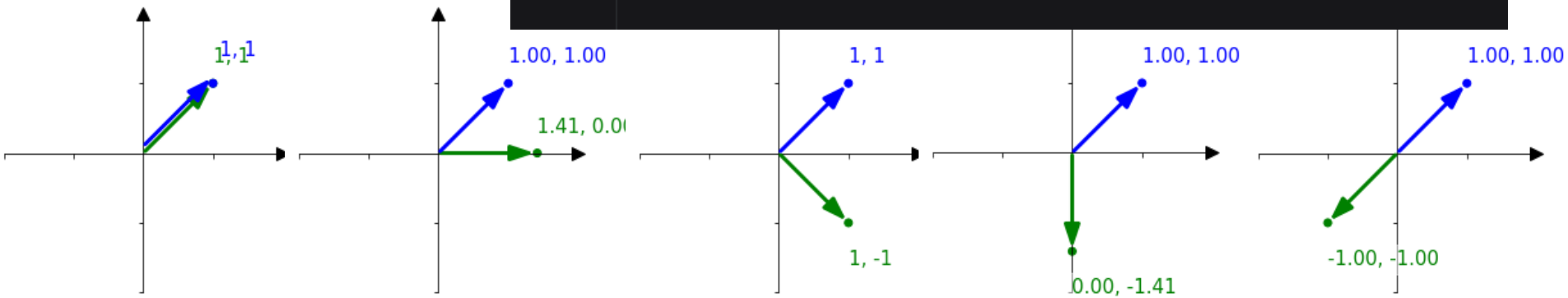
NOKIA

Dot Product: **Non orthogonal** vectors examples

Calculate dot product for the selected angles between vectors.

Take advantage of “dot” function from “numpy” library

```python
import numpy as np

green = [1,1]
blue  = [1,1]

dot = np.dot(green, blue)
print(dot)
```



dot product	2	1.41	0	-1.41	-2.0

For which angle dot product has maximum value and for which a minimum value?

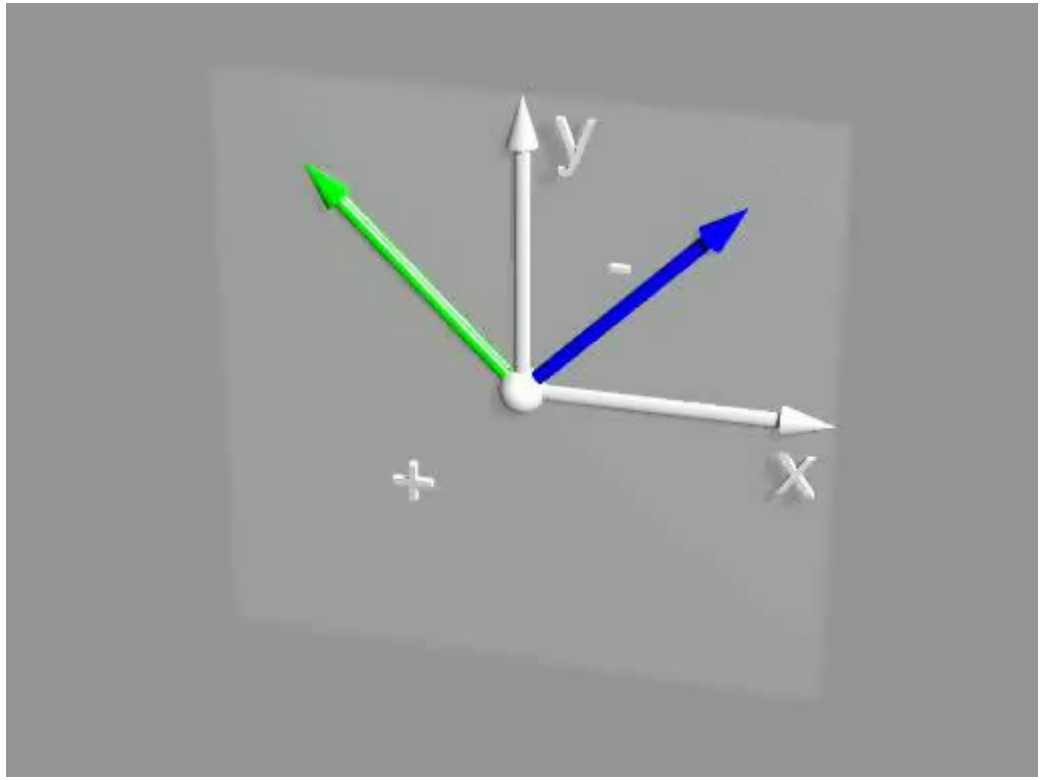
Maximum for 0 and minimum for 180 degrees between vectors

NOKIA

Dot Product: Non orthogonal vectors examples, **cont.**

Results from previous slide let us suppose that the **dot product** :

- takes its **maximum** when the vectors points at **same direction**,
- equals to **zero** when the vectors are **orthogonal**
- takes its **minimum** when the vectors points at the **opposite direction**.



Click to play

2d_vect_dot.pov

NOKIA

Dot Product v.s. vectors angle

Print dot product between green and blue vector for 360° angle range with 10° steps.

Assume that green vector is fixed at [0,1] and blue vector rotates.

Expected result

```
000: +1.000
010: +0.985
020: +0.940
...
360: +1 .000
```

```python
import numpy as np
import math

green = [0,1]
1 usage
def rotate_vector(vector, angle):
    rad_angle = np.radians(angle)
    x = vector[0] * math.cos(rad_angle) - vector[1] * math.sin(rad_angle)
    y = vector[0] * math.sin(rad_angle) + vector[1] * math.cos(rad_angle)
    vector_rotated = np.array([x, y])
    return vector_rotated

for i in range(0,370,10):
    blue = rotate_vector(green, i)
    dot_product = np.dot(green, blue)
    dot_rounded = np.round(dot_product, decimals: 10)
    print(f'{i:03d}:{dot_rounded  :+0.3f}')
```

```
000:+1.000
010:+0.985
020:+0.940
030:+0.866
040:+0.766
050:+0.643
060:+0.500
070:+0.342
080:+0.174
090:+0.000
100:-0.174
110:-0.342
120:-0.500
130:-0.643
140:-0.766
150:-0.866
160:-0.940
170:-0.985
180:-1.000
190:-0.985
200:-0.940
210:-0.866
220:-0.766
230:-0.643
240:-0.500
250:-0.342
260:-0.174
270:-0.000
280:+0.174
290:+0.342
300:+0.500
310:+0.643
320:+0.766
330:+0.866
340:+0.940
350:+0.985
360:+1.000
```

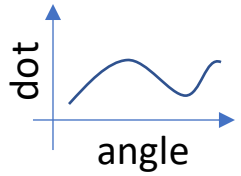
Auxiliary examples (not solution for task)

```python
# Using "range" function and "for" loop
# for iterating from 5 to 110 with step 7.
for i in range(5,110,7): # yes, 110
    print(i)

# Scaling above range to range (0,1)
# just to have number with decimal point.
    print(i, ':', (i-5)/(110-5))

# Using f-strings to control "numbers format".
# Replaces 1b
    print(f'{i:03d}: {(i-5)/(110-5) :+0.3f}')
```

```python
# Rotating vector by 45 deg
# by using function "rotate_vector"

from mylib import rotate_vector

vector = [0,1]
rot_vector= rotate_vector(vector,45)
print(rot_vector)
```

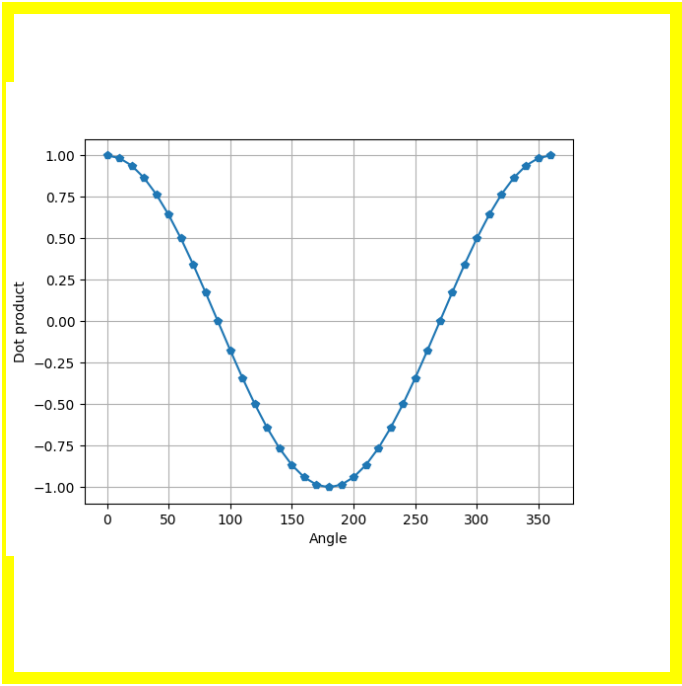
Dot Product v.s. vectors angle, **cont.**

"Replace" printing from previous slide with plotting.



8_dot_calc_angle_range_plot.py

```python
import matplotlib.pyplot as plt
import numpy as np
from mylib import rotate_vector
v = np.array([0, 1])

angle_list = []
dot_list = []

for angle in range(0,370,10):
    angle_list.append(angle)
    v_rot = rotate_vector(v, angle)
    dot = np.sum(v_rot*v)
    dot_list.append(dot)

plt.plot( *args: angle_list,dot_list, '-p')

plt.xlabel("Angle")
plt.ylabel("Dot product")

plt.grid()
plt.show()
```



Auxiliary examples (not solution for task)

7_list_example.py

```python
# Generating copy of list "w"
v = [1,3,6,7]

w = list()
for i in v:
    w.append(i)
print(w)
```

6 plot example.py

```python
# Drawing plot

import matplotlib.pyplot as plt

v = [1,3, 6, 7]
w = [1,9,36,49]

plt.plot(v, w, '-p')
plt.xlabel('V')
plt.ylabel('W')
plt.grid()
```

NOKIA

You are here

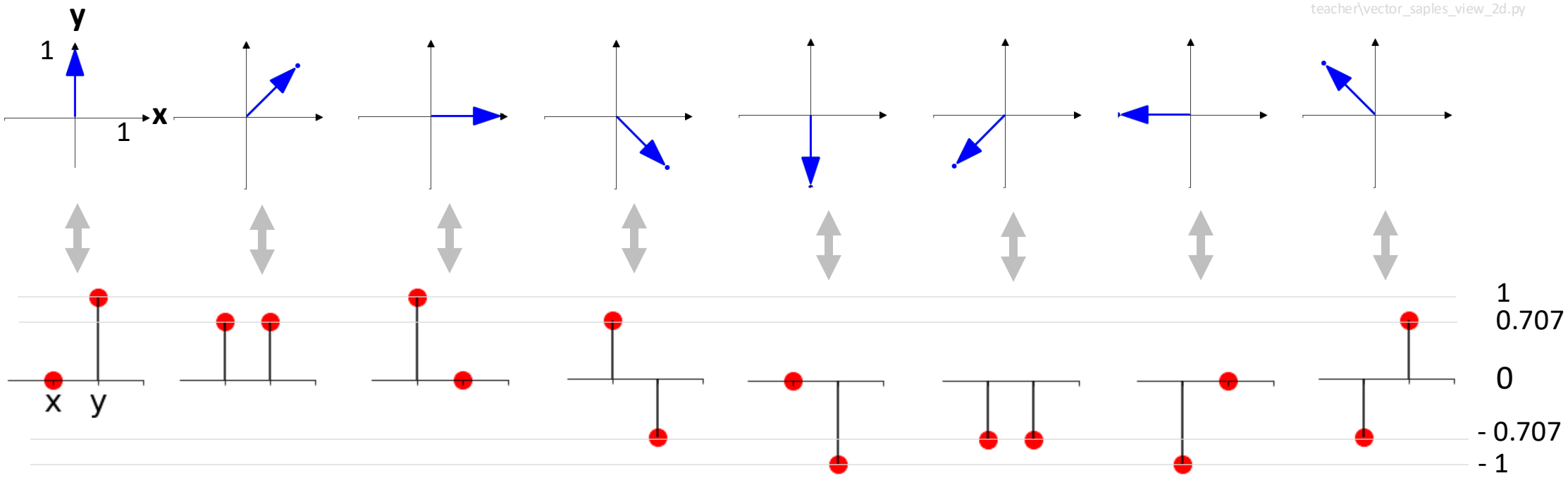
1. Sinewaves orthogonality
2. Orthogonal sinewaves amplitude modulation
3. Discrete Fourier Transformation

- Dot product of vectors
  - 2D vectors
    - Dot product formula
    - Orthogonal examples
    - Dot product vs. angle
  - Vector as a list of numbers
    - 2D example
    - 3D example
- Dot product of signals
  - Vector as a signal - signal as a vector
  - Dot product of signals
  - Signal's similarity
- Sum of periodical signals
- Dot product of sinusoids
  - Dot versus phase shift
  - Dot versus amplitudes
  - Interferences

NOKIA

Vector as a list of numbers: 2D example

2D vector can be presented on a single axis as a **list of values**



NOKIA

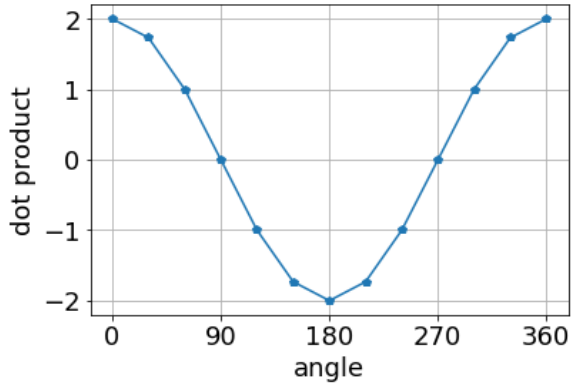
Vector as a list of numbers: 2D example, **cont.**

Find vectors orthogonal to vector A

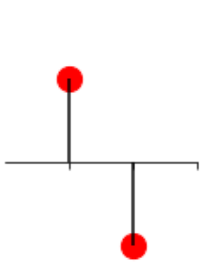
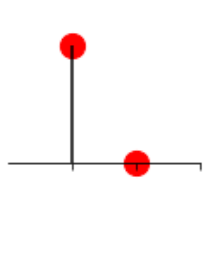
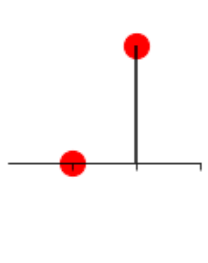
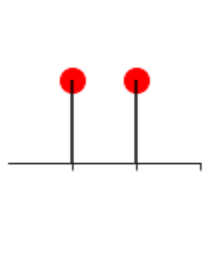
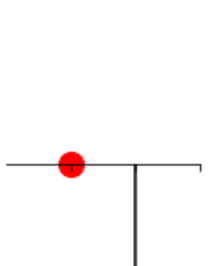
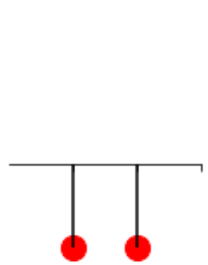
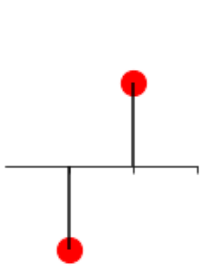
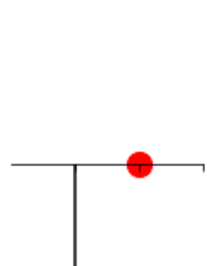
vector A	Orthogonal	Orthogonal	**Anti**parallel
4	7	1	6
8	5	3	2

Replace question marks with vector **index**.
Use REPL or script

cheat sheet



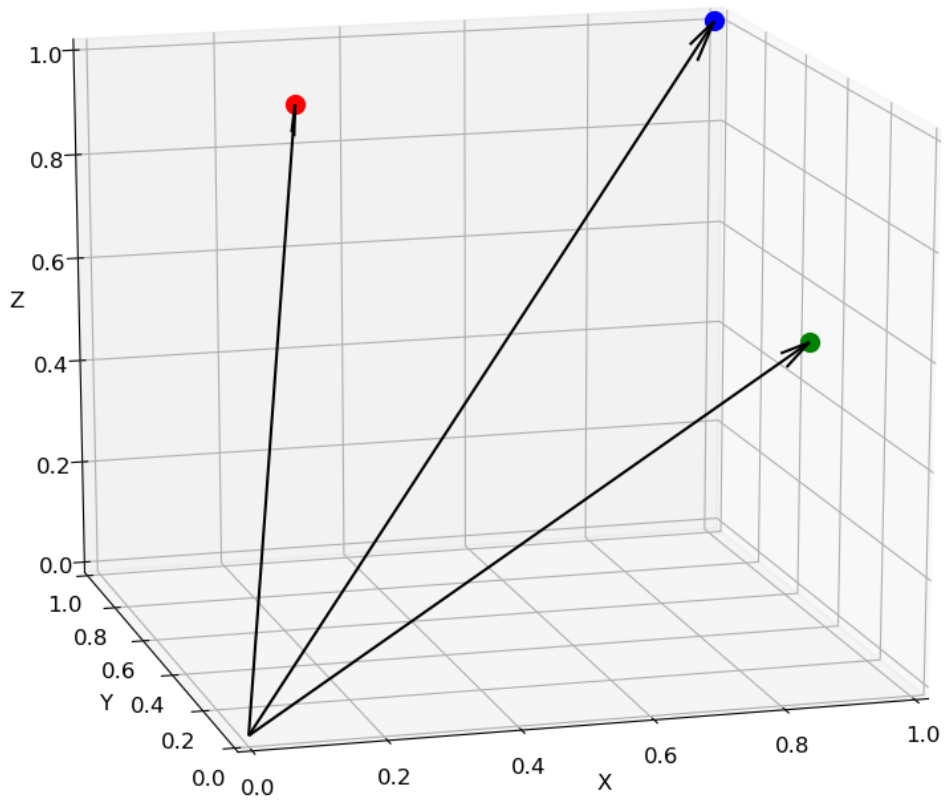
teacher\aux_dot_calc_angle_range_plot.py

1	2	3	4	5	6	7	8
							
0.707, -0.707	1, 0	0, 1	0.707, 0.707	0, -1	-0.707, -0.707	-0.707, 0.707	-1, 0

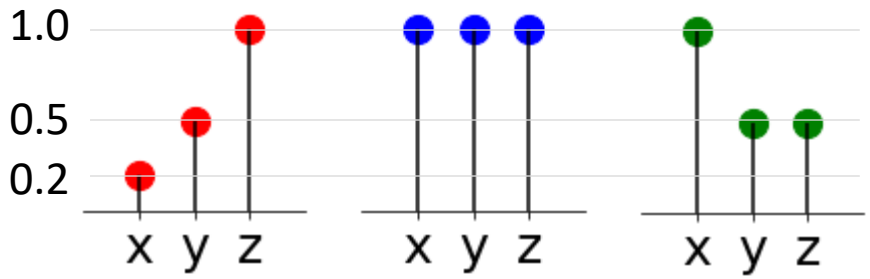
NOKIA

Vector as a list of numbers: **3D example**

Also, **3D vector** can be presented on a **single axis**.



Teacher\vector_3d_cartesian_viev.py

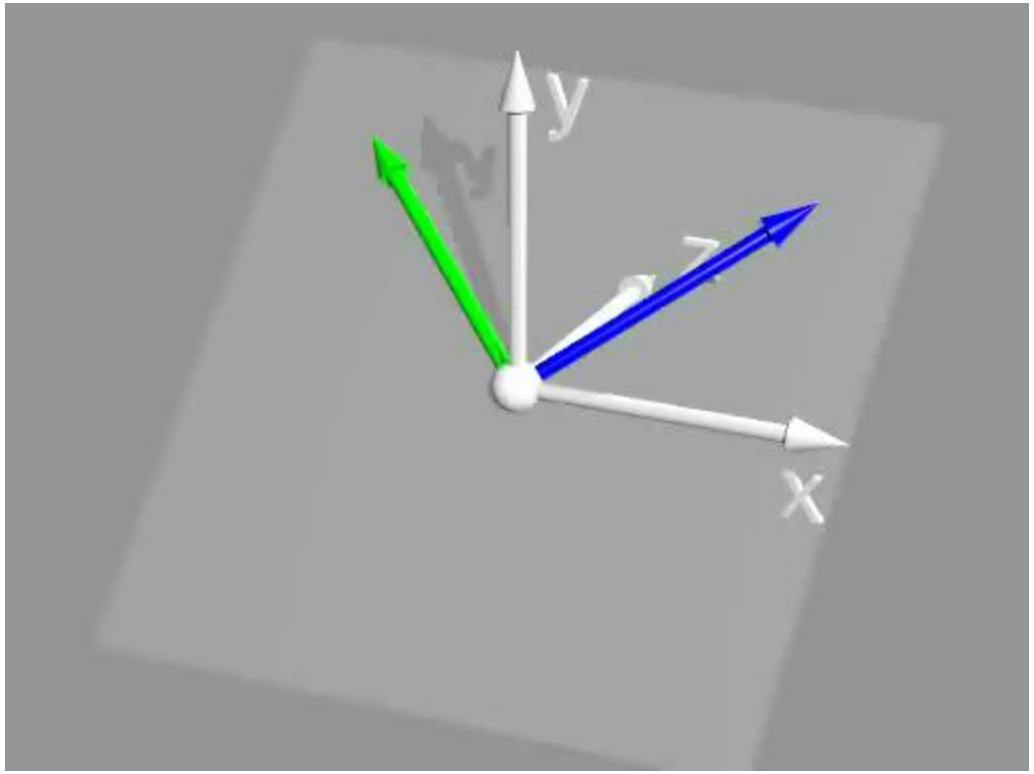


NOKIA

Vector as a list of numbers: 3D example, **cont.**

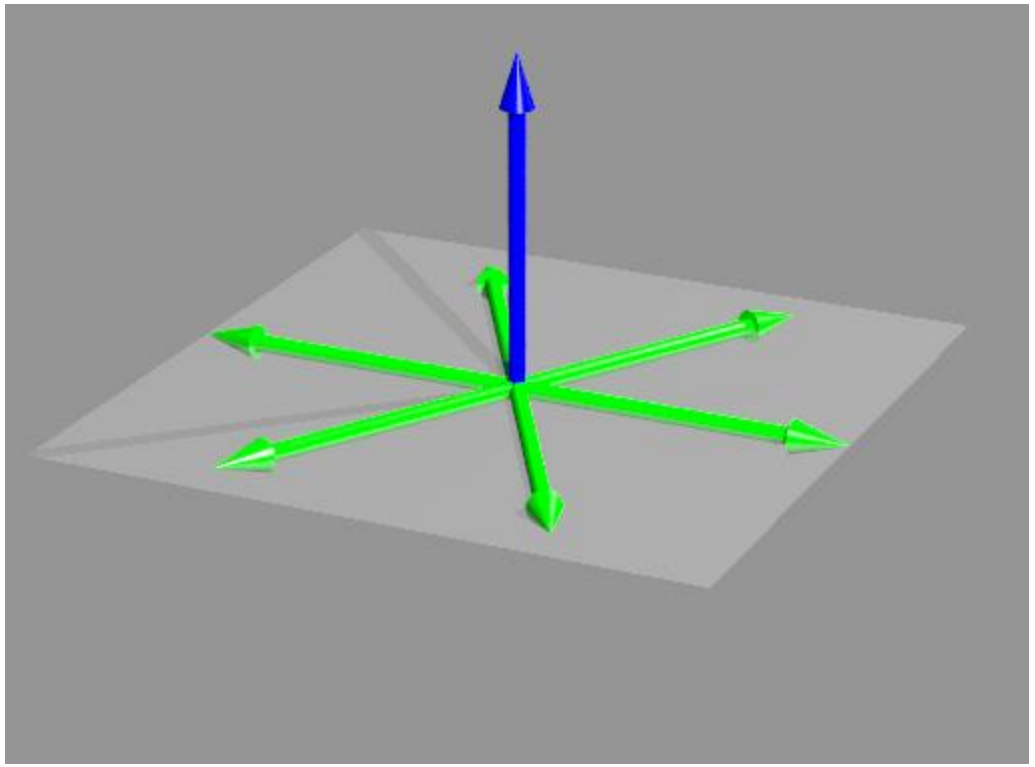
As in case of 2D vectors a **dot product** allows to check how two 3D vectors are oriented towards each other (parallel, antiparallel, orthogonal and so on)

Click to see how does dot product “work” for 3D vectors



3d_vect_dot.pov

Unlike the 2D vector, the specified **3D vector** can have an **infinite number of orthogonal vectors**



3d_vect_dot_infit othog.pov

NOKIA

You are here

1. Sinewaves orthogonality
2. Orthogonal sinewaves amplitude modulation
3. Discrete Fourier Transformation

Dot product of vectors

Dot product of signals

Sum of periodical signals

Dot product of sinusoids

2D vectors

- Dot product formula
- Orthogonal examples
- Dot product vs. angle

Vector as a list of numbers

- 2D example
- 3D example

Vector as a signal - signal as a vector

Dot product of signals

Signal's similarity

Dot versus phase shift

Dot versus amplitudes

Interferences

NOKIA

Vector as a signal - signal as a vector

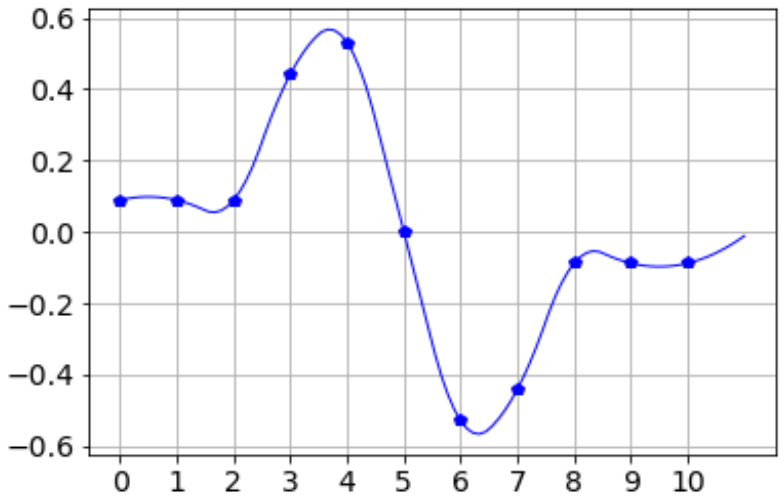
In fact, any **list of values** (scalars) can be considered **a vector in a space**.
The **dimension** of space equals to **values number**.

Examples:

2D: (2.3, 0.4):	x=2.3, y=0.4
3D: (2.3, 0.4, 12.7):	x=2.3, y=0.4, z= 12.7
4D: (2.3, 0.4, 12.7, 9.2):	x=2.3, y=0.4, z= 12.7, ź=9.2
ND: (2.3, 0.4, 12.7, 9.2, … , 17.3):	x=2.3, y=0.4, z= 12.7, ź=9.2, … Ώ=17.3

0 , 1 , 2 , 3 , … , **N** <- coordinate indexes

On the other hand, any signal is a list of values, so **any signal** can be considered a **vector in N dimensional space**.

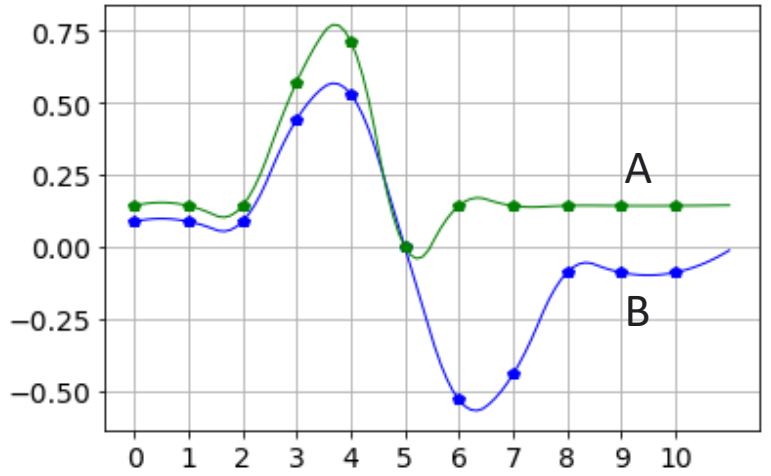


NOKIA

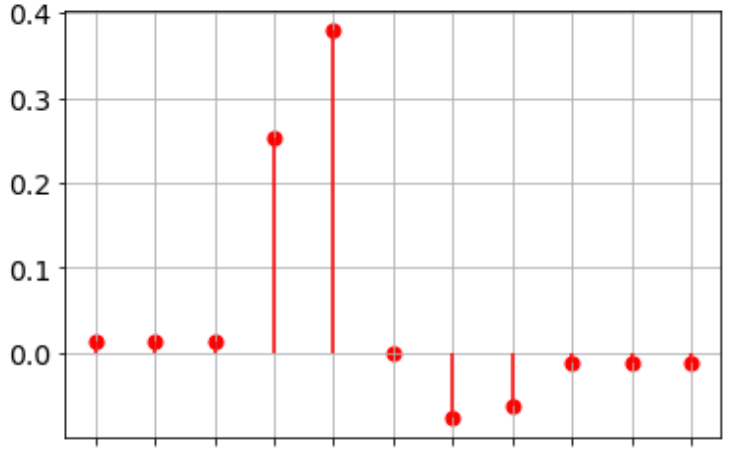
Dot Product of signals

As it was said dot product let us measure how two vectors are oriented towards each other, so, **dot product** let us also measure **relation between two signals** (parallel, antiparallel, orthogonal, and so one).

One can say that **dot product** let us evaluate a **degree of similarity** between **two signals**.



A * B



Teacher\vector-signal_dot_plot.py

dot product -> $\sum_{i=0}^{10} A[i] * B[i] = 0.4924$

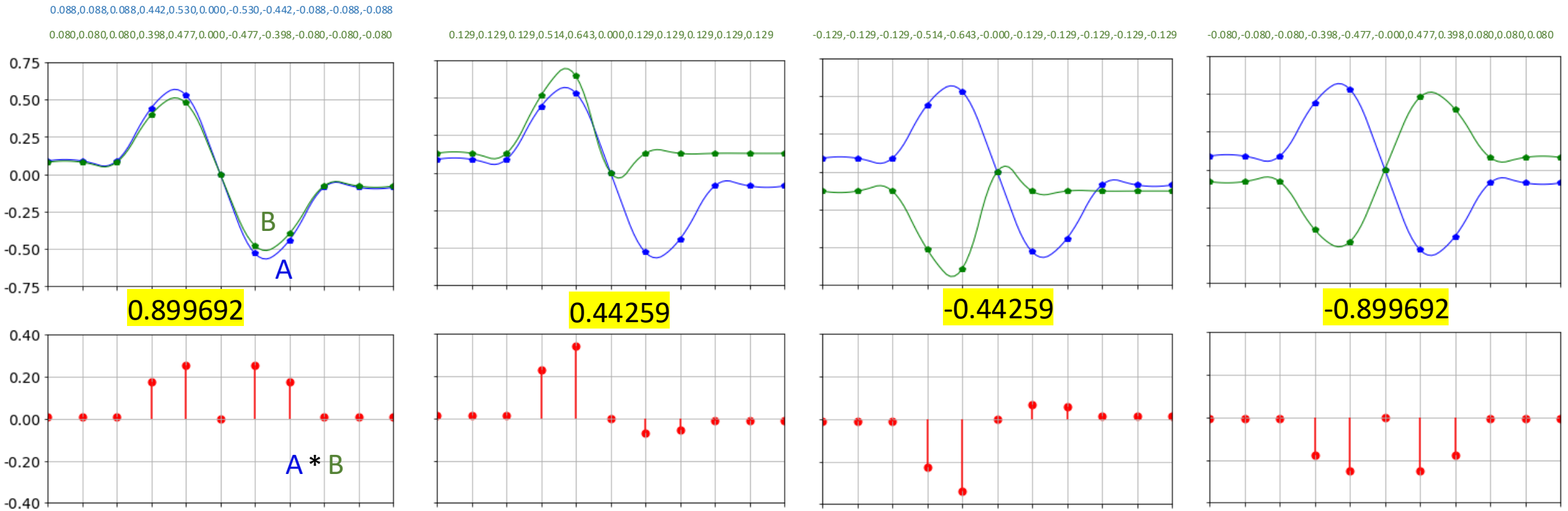
NOKIA

Signals similarity evaluation by using do product

Use Python to replace question marks with dot product.

Use signal samples value given above the plots

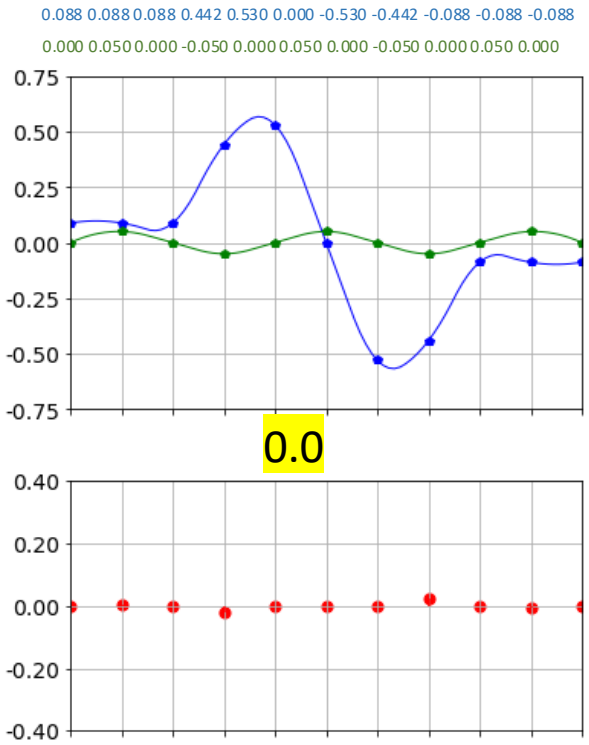
Use round(x,6) when printing dot value

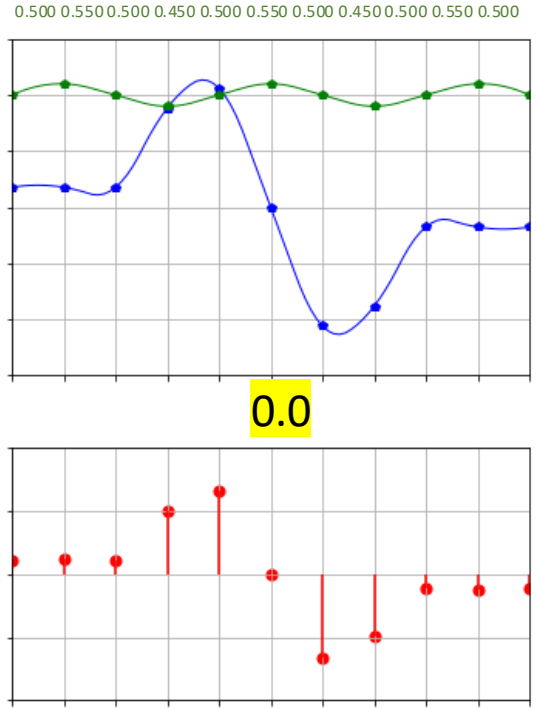


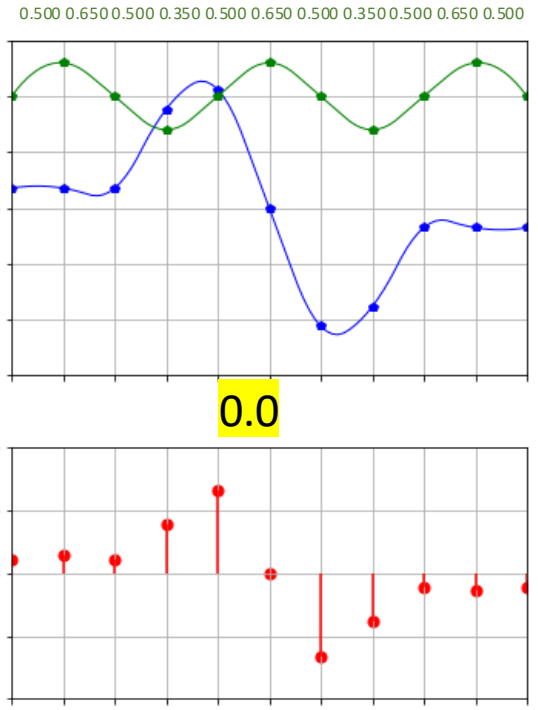
NOKIA

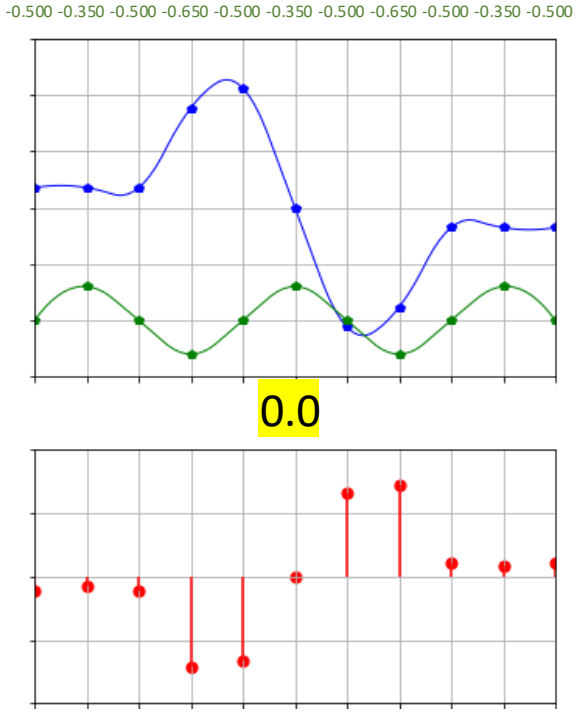
Signals similarity evaluation by using do product, **cont.**

Please continue







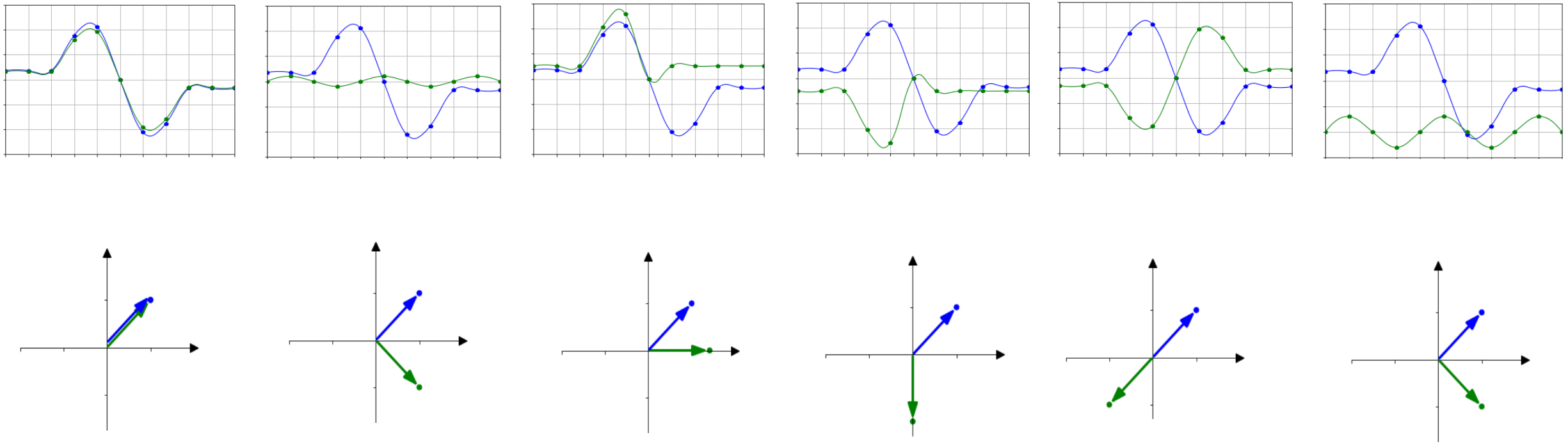


NOKIA

Signals similarity evaluation by using dot product, **cont.**

Find analogy between signal pairs and 2D vectors pairs

Put vectors figures below proper signal figures



NOKIA

Plots customization: **Example**

Example of generation of vectors (1D array) with sinusoid and triangle waves

data preparation

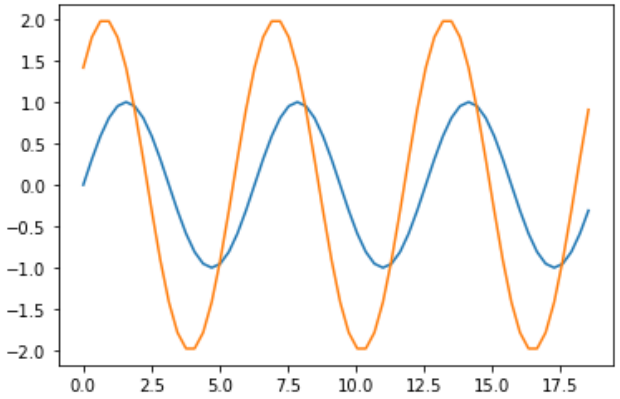
```
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal #required for triangle signal generation

# DATA CREATION

# definition of pi constant by using numpy library
pi = np.pi

# time vector.
# from zero to 2pi in 30 steps
t = np.linspace(0, 3*2*pi,60, endpoint=False) #

# "simple" sinusoid
sin_a = np.sin(t)

# sinusoid with amplitude equal to 2 and time shift qual to pi/4
sin_b = 2*np.sin(t+pi/4)

# triangular waveforms
#"0.5" denotes proportion between positive and negative slope
trian_a = signal.sawtooth(t,0.5)
trian_b = 2*signal.sawtooth(t+pi/2, 0.5)
trian_c = trian_a+trian_b
```


NOKIA

Plots customization: **Example cont.**

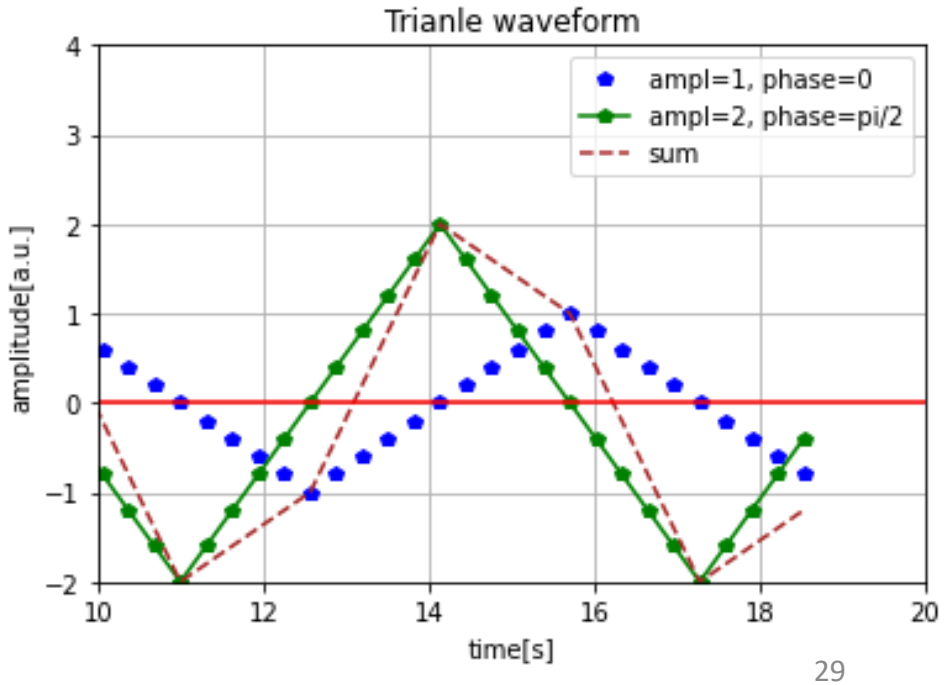
Simple plotting

```
# adding waveforms to figure
plt.plot(t,sin_a)
plt.plot(t,sin_b)

# drawing figure
plt.show()
```



Plots customization

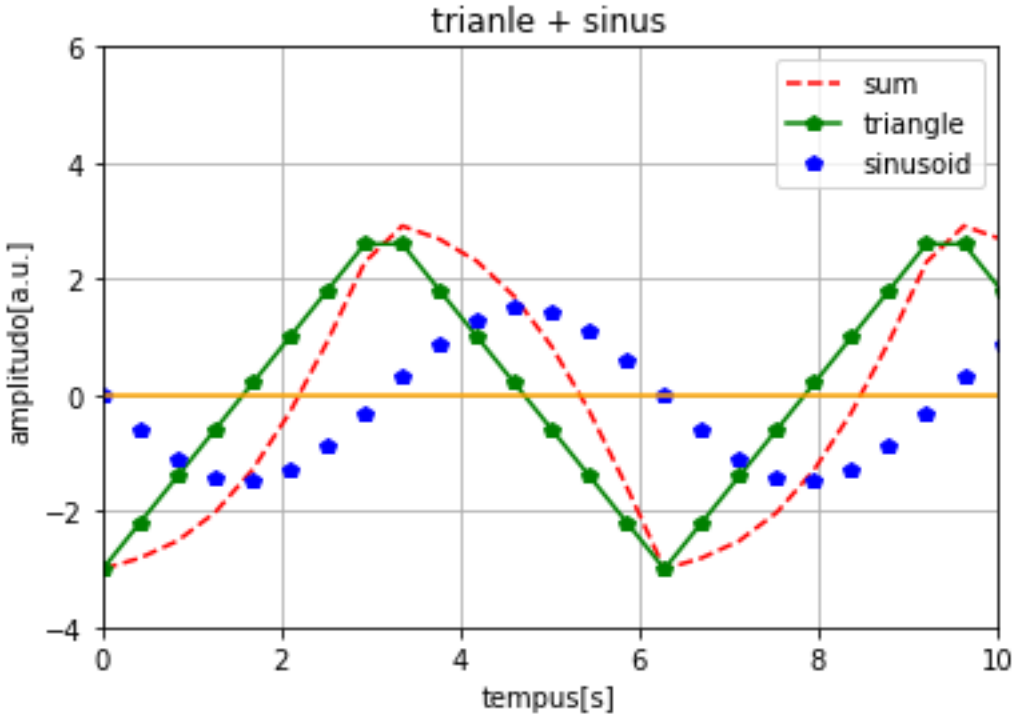
```
# adding waveforms to figure
plt.plot(t,trian_a, 'p', label='ampl=1, phase=0', color='blue')
plt.plot(t,trian_b,'-p', label='ampl=2, phase=pi/2', color='green')
plt.plot(t,trian_c,'--', label='sum', color='brown')

# customizing figure
plt.title('Trianle waveform')
plt.xlabel('time[s]')
plt.ylabel('amplitude[a.u.]')
plt.xlim(10,20)
plt.ylim(-2,4)
plt.axhline(y=0,color='red')
plt.grid()
plt.legend()

# drawing figure
plt.show()
```

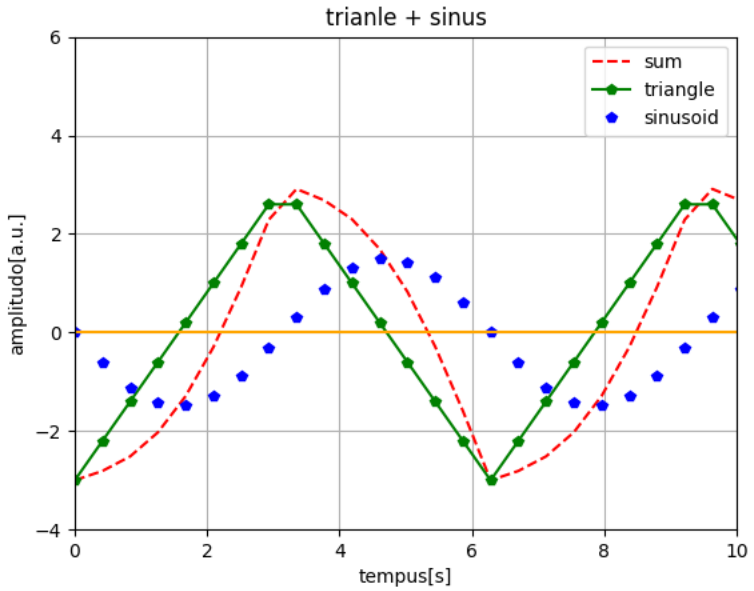


Plots customization: **Exercise**

Generate figure as below.

Don’t forget axes titles, axes ranges and legend






```
import numpy as np
import matplotlib.pyplot
from scipy import signal

pi = np.pi
t = np.linspace( start: 0,

trian_a = 1.5*np.sin(-t)
trian_b = 3*signal.sawto
trian_c = trian_a+trian_

plt.plot( *args: t,trian_c
plt.plot( *args: t,trian_b
plt.plot( *args: t,trian_a


plt.title('trianle + sin
plt.xlabel('tempus[s]')
plt.ylabel('amplitudo[a.
plt.xlim( *args: 0,10)
plt.ylim( *args: -4,6)
plt.axhline(y=0,color='o
plt.grid()
plt.legend()

plt.show()
```

You are here

1. Sinewaves orthogonality
2. Orthogonal sinewaves amplitude modulation
3. Discrete Fourier Transformation

Dot product of vectors

- 2D vectors
  - Dot product formula
  - Orthogonal examples
  - Dot product vs. angle
- Vector as a list of numbers
  - 2D example
  - 3D example

Dot product of signals

- Vector as a signal - signal as a vector
- Dot product of signals
- Signal's similarity

Sum of periodical signals

Dot product of sinusoids

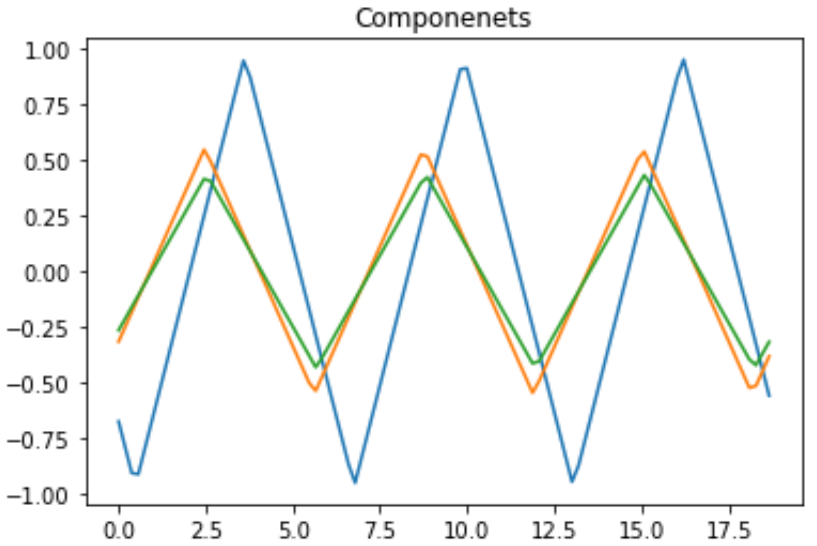
- Dot versus phase shift
- Dot versus amplitudes
- Interferences

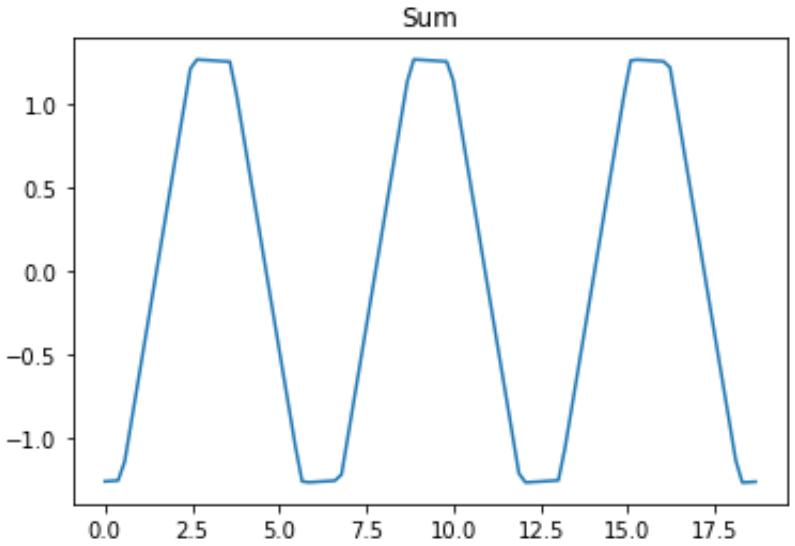
NOKIA

Sum of periodic signals (same frequency)

Run script for 2 rectangular waveform and sawtooth waveform (see details in code comments)

```
…
from scipy import signal
…

t = np.linspace(0, 3*(2*pi),100, endpoint=False)

y_sum = np.zeros(len(t)) # sumaric vector filled with zeros

for i in range(3): # nr of components

    # generating random values of phase shift and amplitude
    # "rand" function generates single random nr in range from 0 to 1
    phase = np.random.rand()*(2*pi)
    ampl  = np.random.rand()

    # generating different waveforms
    # select single waveform by commenting the remaining ones

    y = ampl * np.sign(np.sin(t+phase))      # rectangle
    #y = ampl * signal.sawtooth(t+phase,0.5) # triangle

    plt.plot(t,y,'-')
    y_sum = y_sum + y

plt.title('Componenets')
plt.show()

plt.title('Sum')
plt.plot(t,y_sum,'-')
plt.show()
```



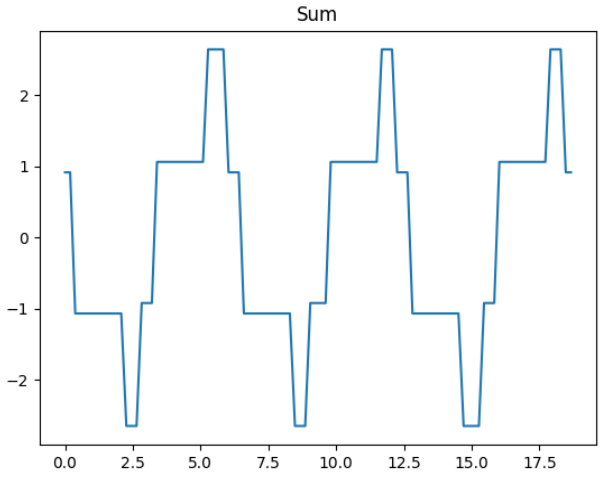


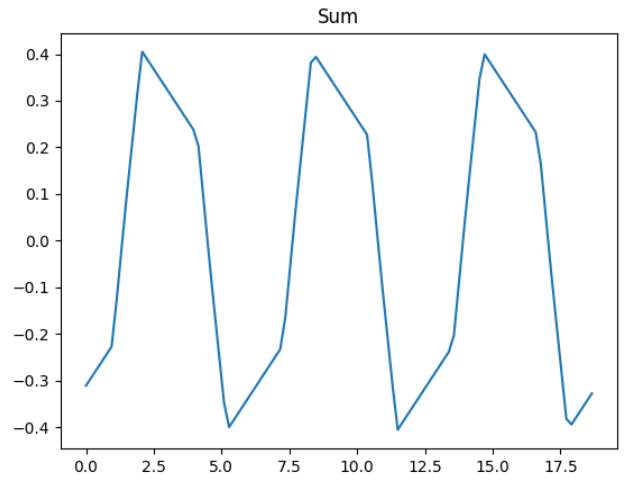
Sum of periodic signals (same frequency), **cont.**

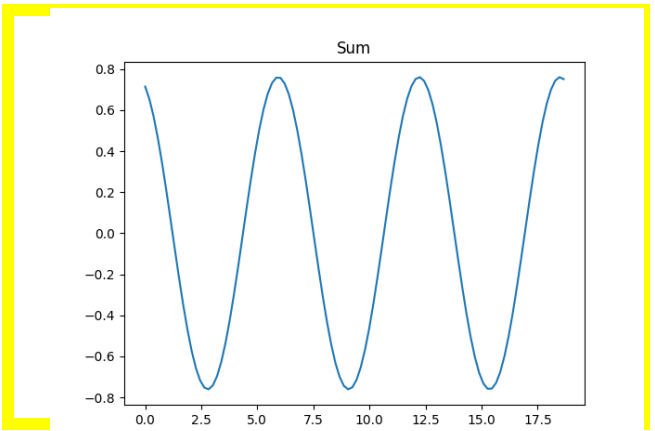
Extend the script with sinus wave generation

Put sum plots (only)

```python
import numpy as np
from matplotlib import pyplot as plt
from scipy import signal

pi = np.pi

t = np.linspace( start: 0, 3 * (2 * pi),  num: 100, endpoint=False)

y_sum = np.zeros(len(t))  # sumaric vector filled with zeros

for i in range(3):  # nr of components

    # generating random values of phase shift and amplitude
    # "rand" function generates single random nr in range from 0 to 1
    phase = np.random.rand() * (2 * pi)
    ampl = np.random.rand()

    # generating different waveforms
    # select single waveform by commenting the remaining ones

    #y = ampl * np.sign(np.sin(t + phase))  # rectangle
    y = ampl * signal.sawtooth(t+phase, width: 0.5) # triangle
    #y = ampl * np.sin(t+phase) #sinus

    plt.plot( *args: t, y, '-')
    y_sum = y_sum + y

plt.title('Componenets')
plt.show()

plt.title('Sum')
plt.plot( *args: t, y_sum, '-')
plt.show()
```







NOKIA

Sum of periodic signals (same frequency): **conclusion**

Which waveform type is exceptional and why ?

Exceptional waveform is sinusoid because negative and positive slopes have the same propotion

You are here

1. Sinewaves orthogonality
2. Orthogonal sinewaves amplitude modulation
3. Discrete Fourier Transformation

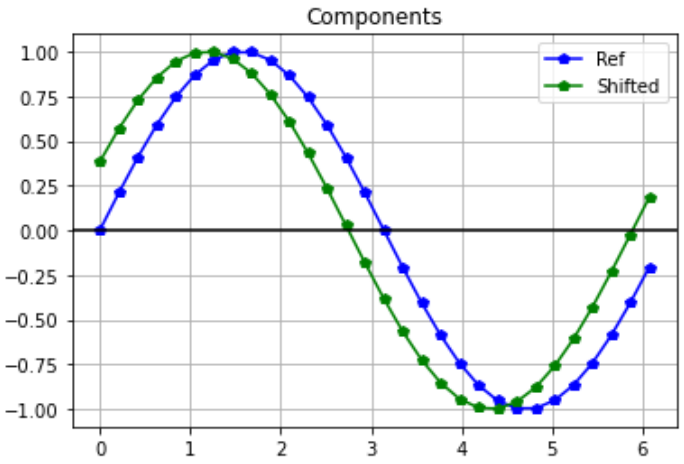
- Dot product of vectors
  - 2D vectors
    - Dot product formula
    - Orthogonal examples
    - Dot product vs. angle
  - Vector as a list of numbers
    - 2D example
    - 3D example
- Dot product of signals
  - Vector as a signal - signal as a vector
  - Dot product of signals
  - Signal's similarity
- Sum of periodical signals
- Dot product of sinusoids
  - Dot versus phase shift
  - Dot versus amplitudes
  - Interferences

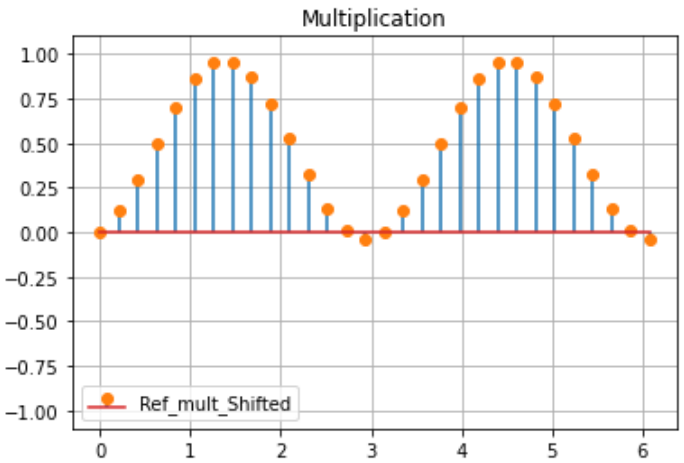
NOKIA

dot vs. phase shift

Fix and complete following script to generate plots and prints as below

```python
import numpy as np
import matplotlib.pyplot as plt
pi = np.pi

# PARAMETER
PHASE_SHIFT = pi/2+pi/2

# VECTORS
t = np.linspace( start: 0, 2*pi, num: 30, endpoint=False)

Ref = np.sin(t)
Shifted = np.sin(t+PHASE_SHIFT)
Ref_mult_Shifted = Ref*Shifted
dot_product = np.sum(Ref_mult_Shifted) #use Ref_mult_Shifted

# PLOTS (HINT: use separate plots, not one with grid)

# components
plt.plot( *args: t, Ref, '-p', color='blue')
plt.plot( *args: t, Shifted, '-p', color='green')
plt.legend(["Ref", "Shifted"])
plt.title("Components")
plt.grid()
plt.axhline(y=0, color='black')
plt.show()
# multiplication, HINT: use "stem" function for ploting
plt.stem( *args: t, Ref_mult_Shifted, markerfmt='darkorange')
plt.legend(["Ref_mult_Shifted"])
plt.grid()
plt.axhline(y=0, color='red')
plt.ylim([-1,1])
plt.show()
# print phase shift and dot product value
print(PHASE_SHIFT,f' dot = {dot_product:0.2f}',)
```

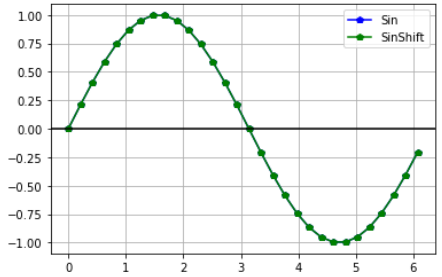


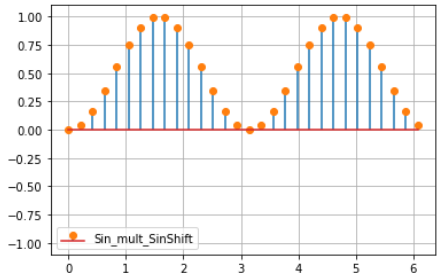



```
phase_shift = 0.39
dot_product = 13.86
```


NOKIA

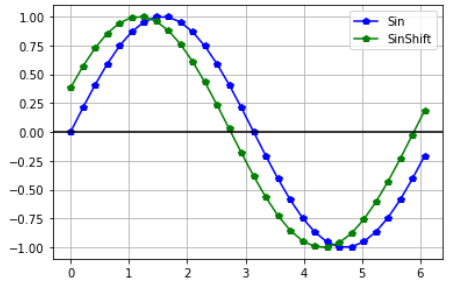
0

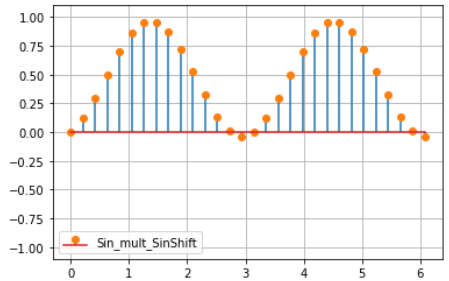




dot product = 15.00

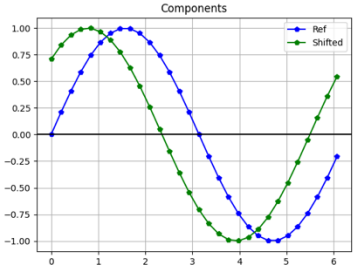
pi/8

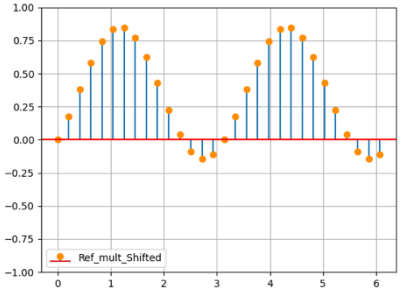




dot product = 13.86

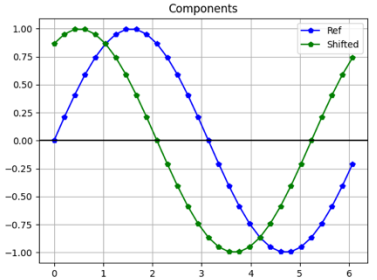
pi/4

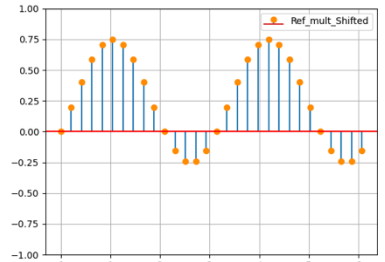




dot product = 10.61

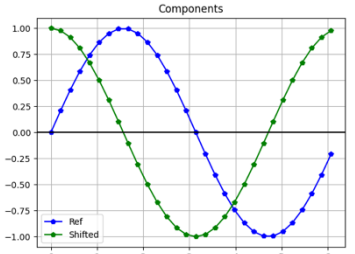
pi/3

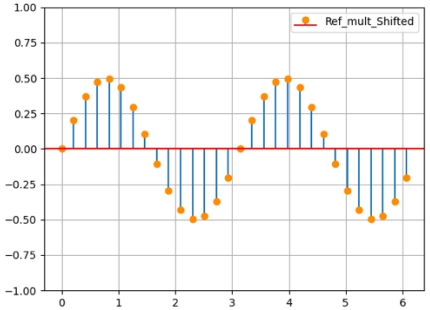




dot product = 7.50

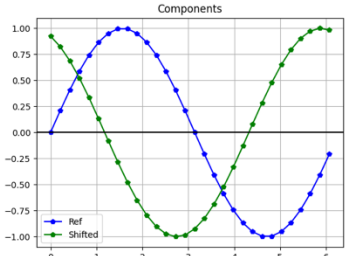
pi/2

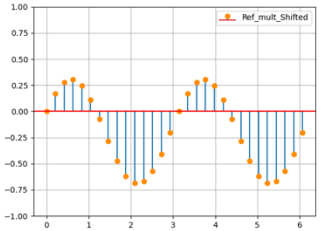




dot product = 0.00

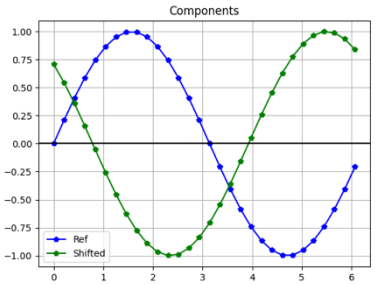
np.pi/2 + np.pi/8

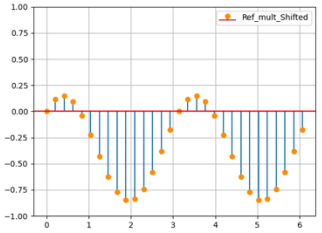




dot product = -5.74

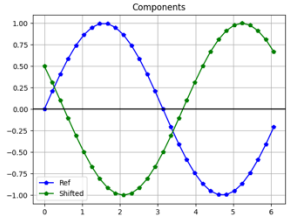
np.pi/2 + np.pi/4

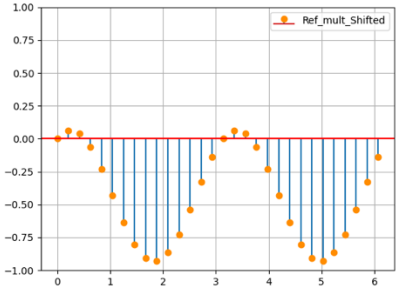




dot product = -10.61

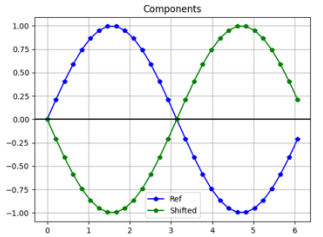
np.pi/2 + np.pi/3

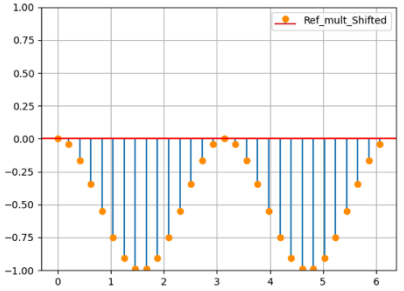




dot product = -12.99

np.pi/2 + np.pi/2





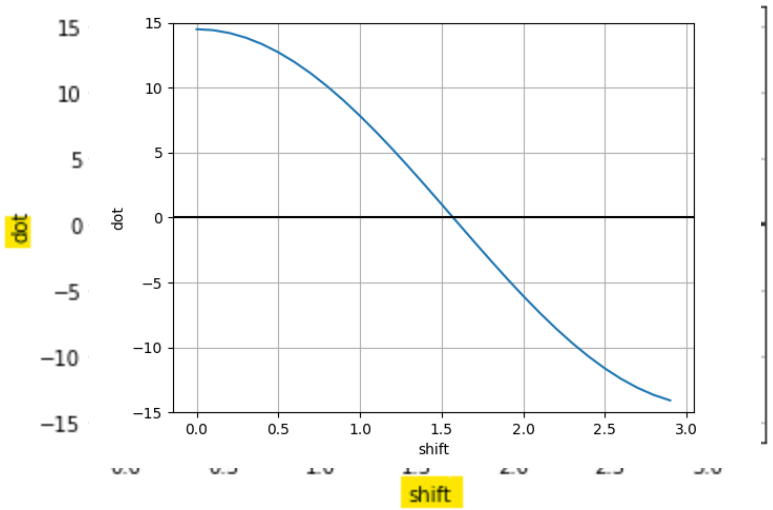
dot product = -15.00

dot vs. phase shift, cont.

Plot relationship between **dot product** and **phase shift**

Shift values should be in the range as on the plot below however,

its values do not have to be limited to values from previous slide (linspace)

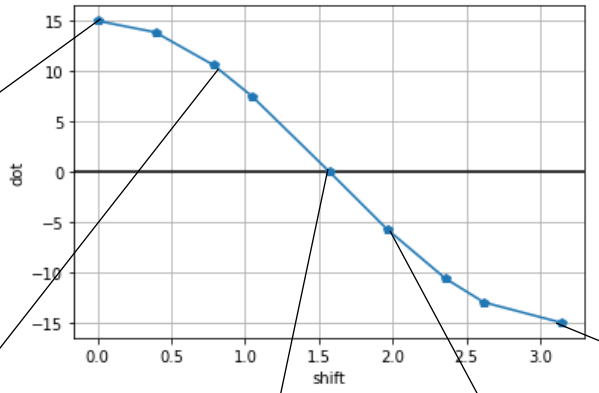


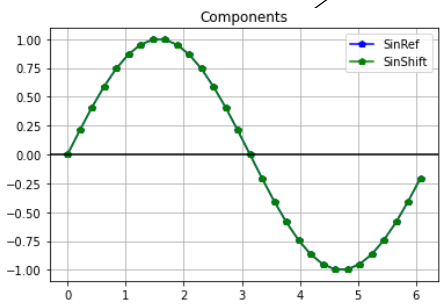

```python
import numpy as np
import matplotlib.pyplot as plt

pi = np.pi

t = np.linspace( start: 0, 2 * pi,  num: 30)
phaseShift = np.linspace( start: 0,  stop: 3,  num: 30, endpoint=False)
Ref = np.sin(t)

dot_products = []

for phase in phaseShift:
    shifted = np.sin(t + phase)
    dot_product = np.dot(shifted, Ref)
    dot_products.append(dot_product)

plt.plot( *args: phaseShift, dot_products)
plt.xlabel('shift')
plt.ylabel('dot')
plt.ylim( *args: -15,15)
plt.axhline( y: 0, color='black')
plt.grid()
plt.show()
```

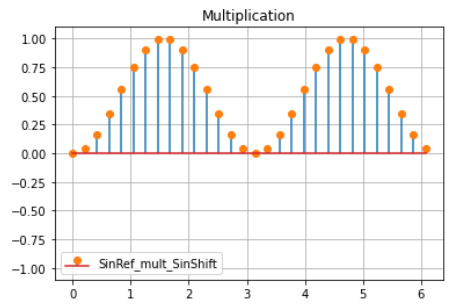

NOKIA

dot vs. phase shift, cont.

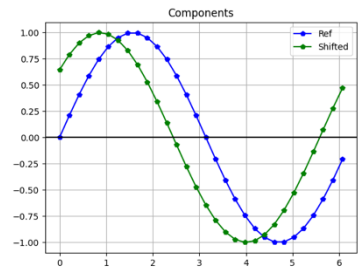
Base on results from previous page
assign plots to points on “phase shift – dot value” plot

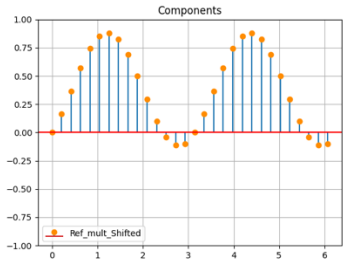




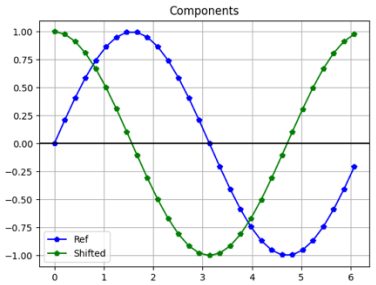


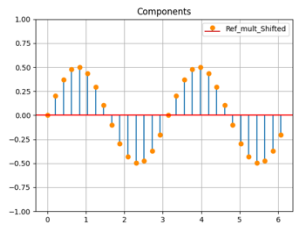
phase_shift = 0.00
dot_product = 15.00



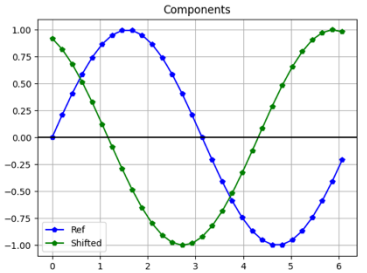


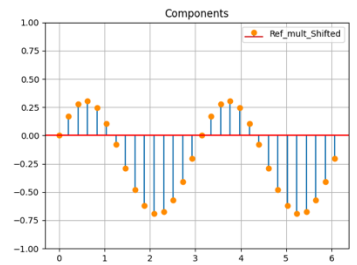
phase shift = 0.70
dot product = 11.47



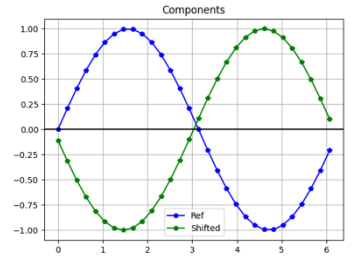


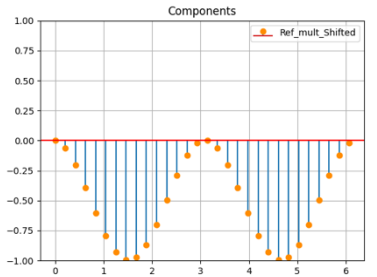
phase shift = 1.57
dot product = 0.01





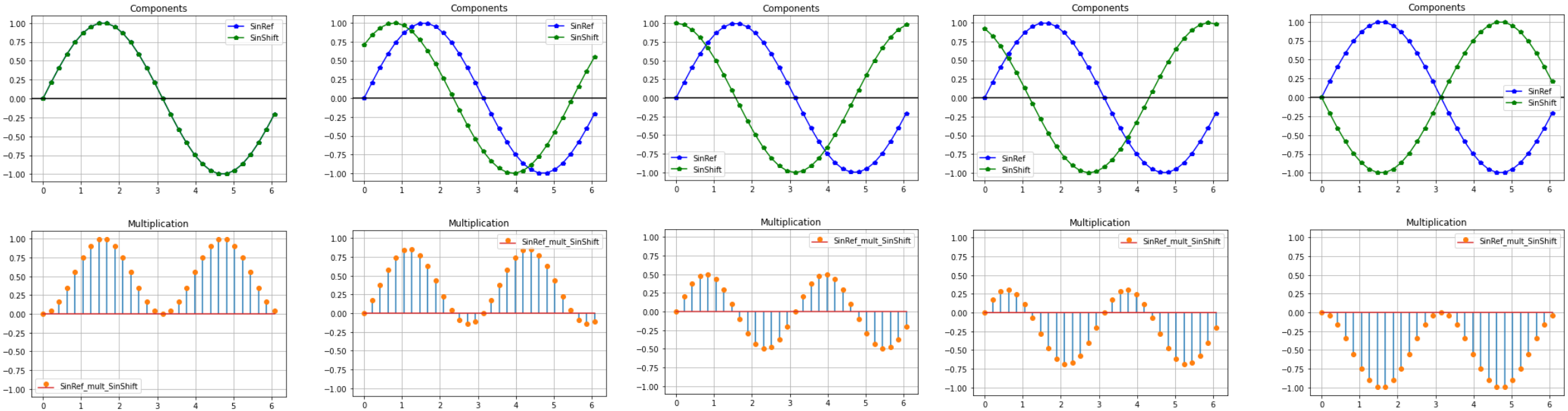
phase shift = 1.97
dot product = -5.83

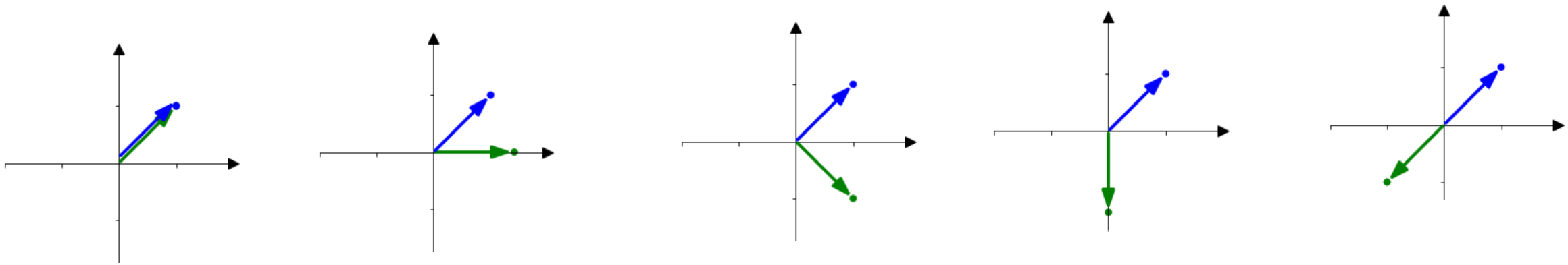




phase shift = 3.25
dot product = -14.9

dot vs. phase shift, cont





Put vector figures below proper sinusoids plots

You are here

1. Sinewaves orthogonality
2. Orthogonal sinewaves amplitude modulation
3. Discrete Fourier Transformation

- Dot product of vectors
  - 2D vectors
    - Dot product formula
    - Orthogonal examples
    - Dot product vs. angle
  - Vector as a list of numbers
    - 2D example
    - 3D example
- Dot product of signals
  - Vector as a signal - signal as a vector
  - Dot product of signals
  - Signal's similarity
- Sum of periodical signals
- Dot product of sinusoid
  - Dot versus phase shift
  - Dot versus amplitudes
  - Interferences

NOKIA

dot versus amplitudes: **sinus amplitude measurement**

Examine **dot product** of two sinewaves of the **same frequency and phase** for time vectors of different size.

Use following script to fill tables with dot product of sinusoids of selected amplitudes for two different TIME_VECTOR_SIZEs

```
# PARAMS
TIME_VECTOR_SIZE = 30

A = 1
B = 2

# CALCULATIONS
t = np.linspace(0, 2*pi,TIME_VECTOR_SIZE, endpoint=False)

Sin_A = A * np.sin(t)
Sin_B = B * np.sin(t)
dot = np.dot(Sin_A,Sin_B)

# PRESENTATION
print(f'A = {A}')
print(f'B = {B}')
print(f'dot = {dot:0.2f}')
```

TIME_VECTOR_SIZE=**15**

dot	A		
B	0	1	2
0	0.00	0.00	0.00
1	0.00	7.50	15.00
2	0.00	15.00	30.00

TIME_VECTOR_SIZE=**30**

dot	AMP_A		
AMP_B	0	1	2
0	0.00	0.00	0.00
1	0.00	15.00	30.00
2	0.00	30.00	60.00

Base on table content write formula illustrating relationship between **dot product** value and values of **A, B** and **TIME_VECTOR_SIZE**,

dot_product(AMP_A, AMP_B, TIME_**VECTOR_SIZE**,) =

A*B*TIME_VECTOR_SIZE/2

NOKIA

dot versus amplitudes : **normalization**

Value of dot product of two “in-phase” sinuses depends on their: amplitudes and resolution (TIME_VECTOR_SIZE).
In order to use dot product to measure amplitude of sinus one must normalize dot product, i.e., make it independent from time vector resolution

Extend previous script with “A_rx“ evaluation.
The value (not formula) of “A_rx” should:

- Be equal to value of A_tx
- be calculated base on dot product
- should not depend on TIME_VECTOR_SIZE changes (see tables)

```python
import numpy as np
import matplotlib.pyplot as plt
pi = np.pi

# PARAMS
TIME_VECTOR_SIZE = 100

A_tx = 1
B = 2

# CALCULATIONS
t = np.linspace( start: 0, 2*pi,TIME_VECTOR_SIZE, endpoint=False)

Sin_A = A_tx * np.sin(t)
Sin_B = B * np.sin(t)
dot = np.dot(Sin_A,Sin_B)
A_rx = dot/B/TIME_VECTOR_SIZE*2

# PRESENTATION
print(f'A_tx = {A_tx}')
print(f'B = {B}')
print(f'dot = {dot:0.2f}')
print(f'A_rx = {A_rx:0.2f}')
```

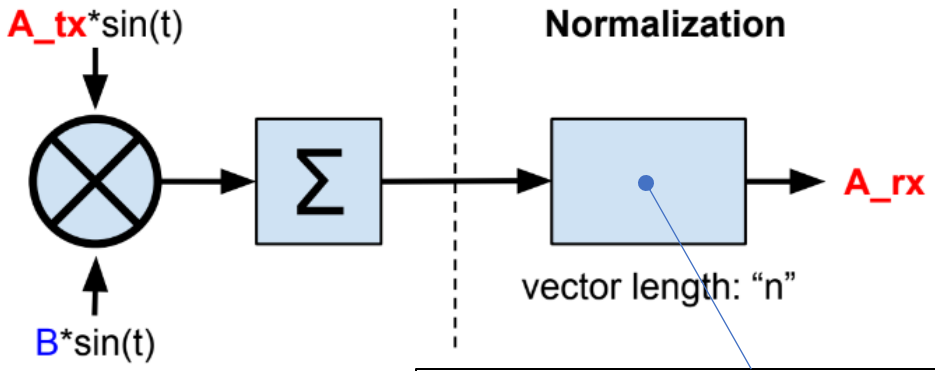
Expected results

B = 1
TIME_VECTOR_SIZE=**15**

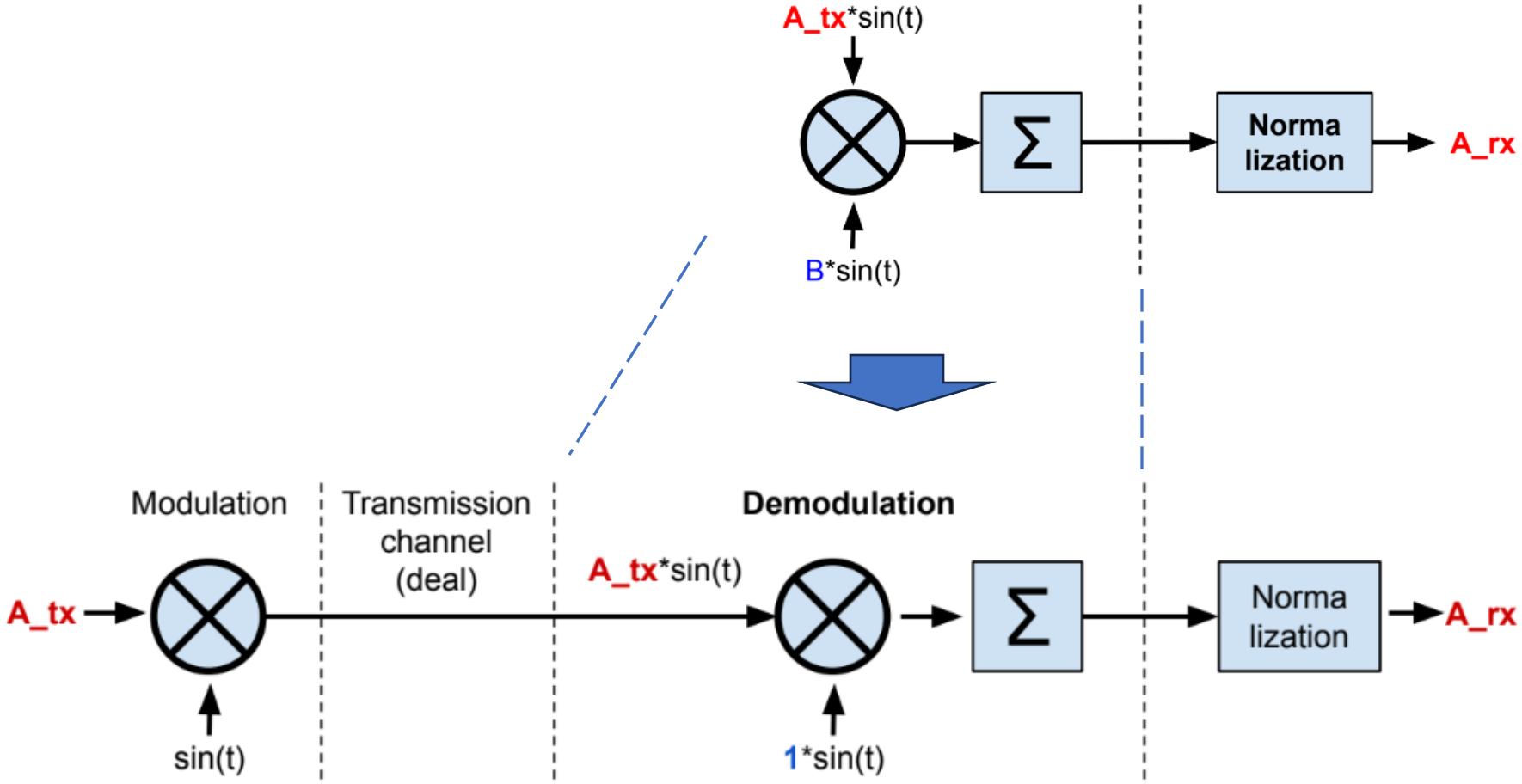
A_tx	A_rx
0	**0**
1	**1**
2	**2**

B= 1
TIME_VECTOR_SIZE=**30**

A_tx	A_rx
0	**0**
1	**1**
2	**2**



As seen from the previous slide dot product can be used for demodulation of amplitude modulated signalas



You are here

1. Sinewaves orthogonality
2. Orthogonal sinewaves amplitude modulation
3. Discrete Fourier Transformation

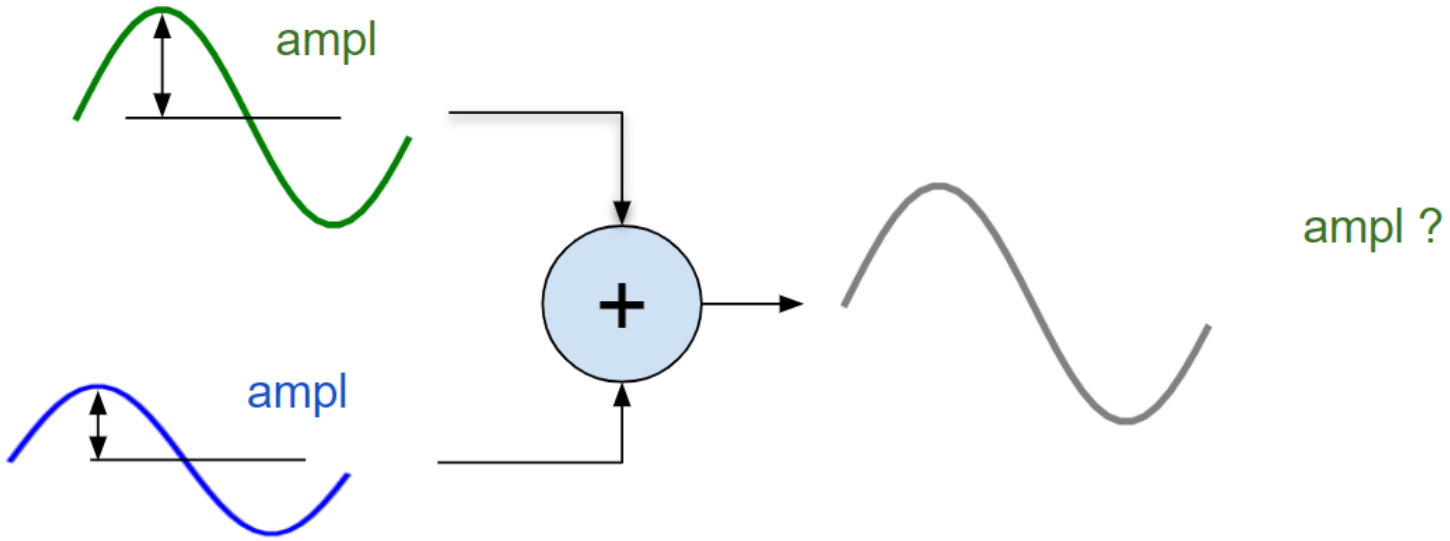
- Dot product of vectors
  - 2D vectors
    - Dot product formula
    - Orthogonal examples
    - Dot product vs. angle
  - Vector as a list of numbers
    - 2D example
    - 3D example
- Dot product of signals
  - Vector as a signal - signal as a vector
  - Dot product of signals
  - Signal's similarity
- Sum of periodical signals
- Dot product of sinusoids
  - Dot versus phase shift
  - Dot versus amplitudes
  - Interferences

NOKIA

Dot product of the same frequency sinewaves: **Interferencing**

Let's assume we have signal which is the **sum** of two sinusoids, **carrier and interferer**.

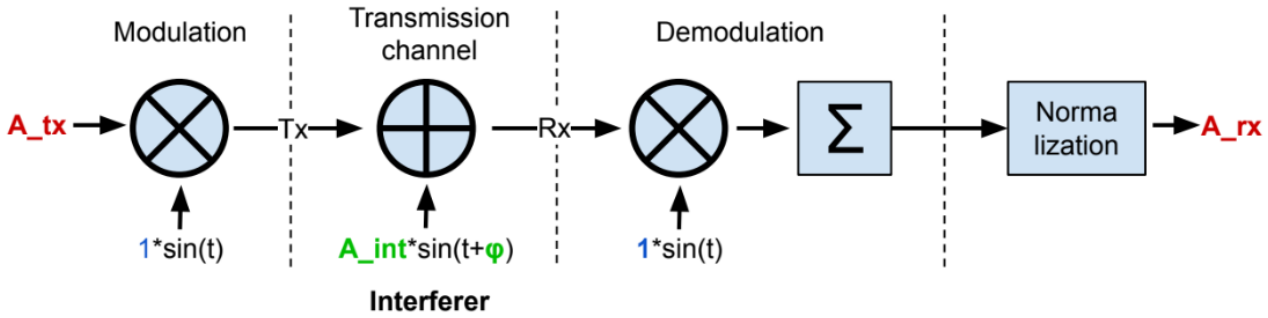
Will it be possible to **recover carrier amplitude** the spite of interferer presence.



Dot product of the same frequency sinewaves: **interferences, cont.**

Complete script to implement given scheme.

```python
import numpy as np
import matplotlib.pyplot as plt
pi = np.pi

# PARAMETERS
TIME_VECTOR_SIZE = 30
A_tx = 1
A_int = 2
A_int_phi = pi/4 + pi/2

# CALCULATION
t = np.linspace( start: 0, 2*pi,TIME_VECTOR_SIZE, endpoint=False)

# modulation
Tx   = A_tx  * np.sin(t)

#channel
Interfer = A_int * np.sin(t+A_int_phi)
Rx       = Tx + Interfer

# demodulation
Rx_mul_Sin = Rx * np.sin(t)
Rx_dot_Sin  = np.sum(Rx_mul_Sin) # Use Rx_mul_Sin please
A_rx = Rx_dot_Sin/TIME_VECTOR_SIZE*2

# PRESENTATION
plt.plot( *args: t,Tx,  '-' , label='Tx', color='green')
plt.plot( *args: t,Interfer, '-',  label='Interfer',color='blue')
plt.plot( *args: t,Rx,       '-' , label='Rx',      color='gray', linewidth=1))
plt.ylim( *args: -3.1, 3.1)
plt.legend()
plt.grid()
plt.axhline(y=0,color='black')
plt.show()
```



Use the script to fill the table

Φ = pi/4			
A_int = 1		A_tx=1	
A_tx	A_rx	A_int	A_rx
0.00	0.7	0.00	1
0.50	1.2	0.50	1.4
1.00	1.7	1.00	1.7
1.50	2.2	1.50	2.1
2.00	2.7	2.00	2.4

Φ = pi/2			
A_int = 1		A_tx=1	
A_tx	A_rx	A_int	A_rx
0.00	0	0.00	1
0.50	0.5	0.50	1
1.00	1.00	1.00	1
1.50	1.50	1.50	1
2.00	2.00	2.00	1

Φ = pi/2 + pi/4			
A_int = 1		A_tx=1	
A_tx	A_rx	A_int	A_rx
0.00	-0.7	0.00	1
0.50	-0.2	0.50	0.6
1.00	0.3	1.00	0.3
1.50	0.8	1.50	-0.1
2.00	1.3	2.00	-0.4

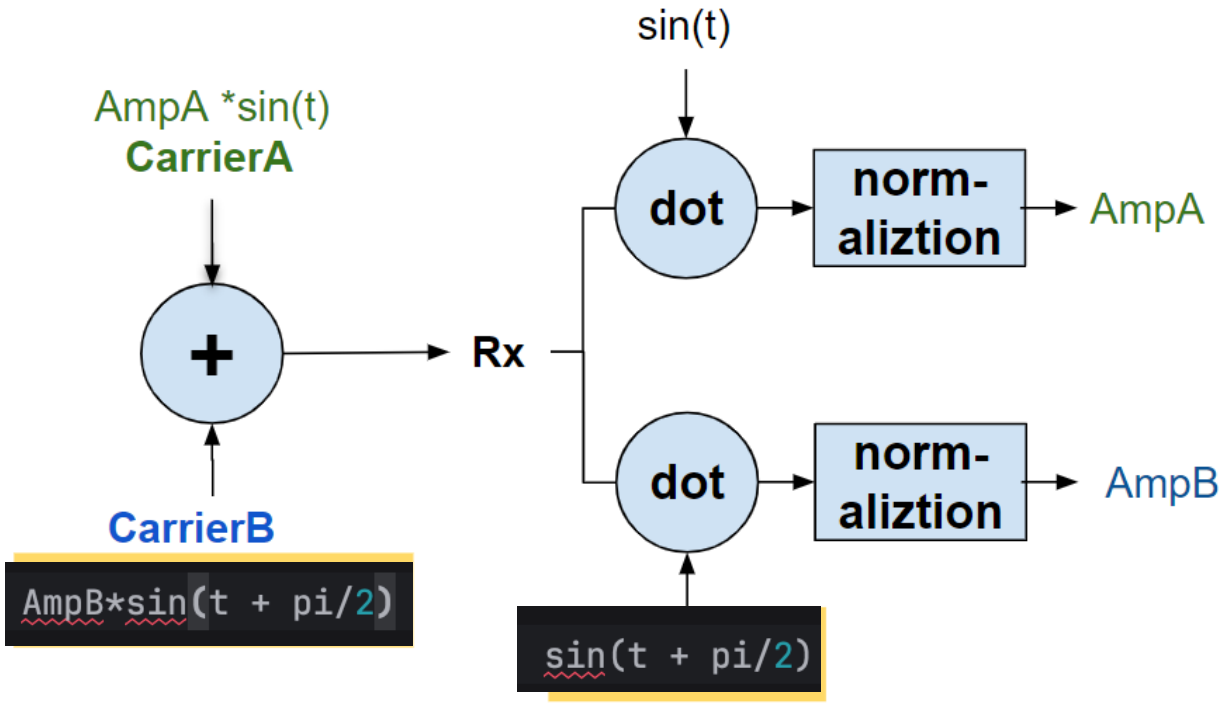
Dot product of the same frequency sinewaves: **interferences: conclusion**

When do two sinewaves of the same frequency do not disturb each other?

When they are orthogonal (phase pi/2)

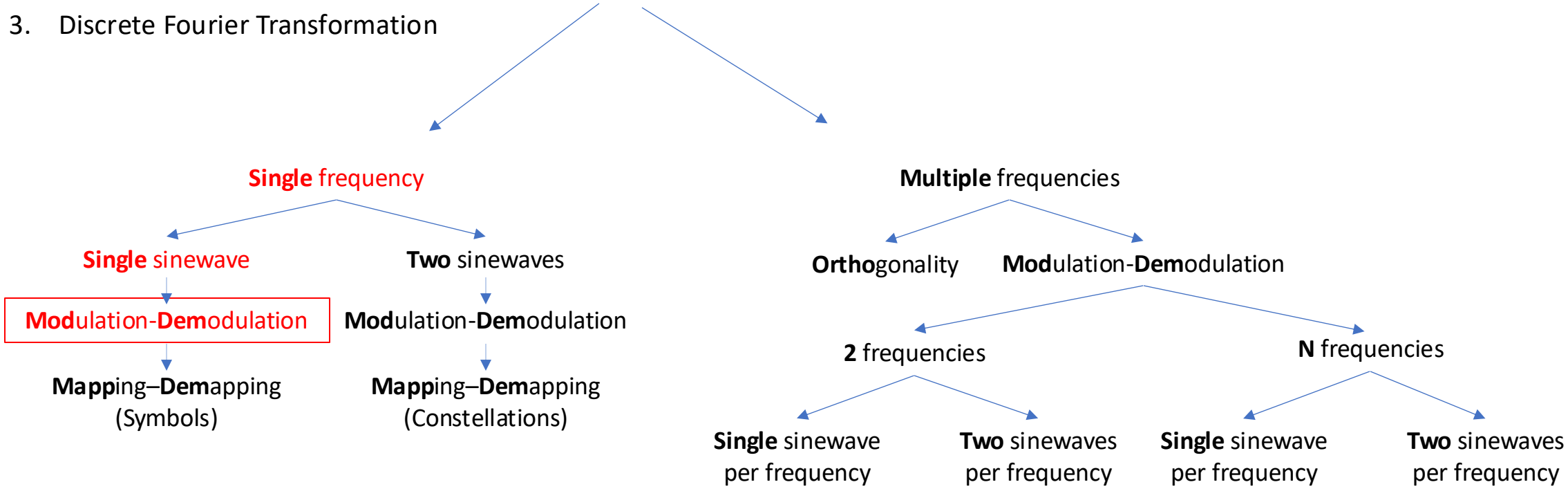
In this case sinewaves do not interfere each other because their dot product is 0.

Assuming above condition is satisfied, is it possible to “transfer amplitude” of two sinewaves via one signal?
Can we use “interferer” from previous slide as a second amplitude carier?
If yes, fulfill yellow boxes.



You are here

1. Sinewaves orthogonality
2. Orthogonal sinewaves amplitude modulation
3. Discrete Fourier Transformation

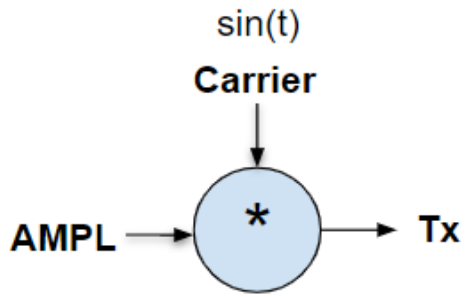


NOKIA

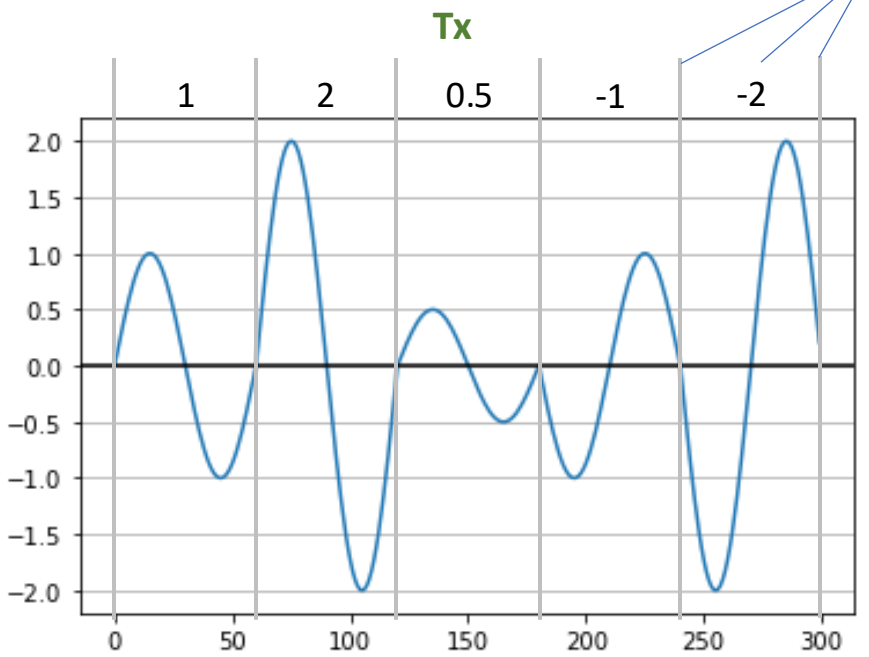
Single sinewave: **modulation**

Complete script to obtain modulated sinus as below

```
import numpy as np
import matplotlib.pyplot as plt
pi = np.pi

# PARAMETERS
TIME_VECTOR_SIZE = 60
AMPL_VECTOR = (1,2,0.5,-1,-2)

# CALCULATION
t = np.linspace( start: 0, 2*pi,TIME_VECTOR_SIZE, endpoint=False)

Carrier = np.sin(t)

Tx = np.array([])
for amp in AMPL_VECTOR:
    Tx = np.append(Tx, amp * Carrier)

# PRESENTATION
plt.plot(Tx)
plt.axhline(y=0,color='black')
plt.grid(axis='y')
plt.show()

# SAVING
np.save( file: 'TxSignal',Tx)
```



Do not try to obtain this lines and numbers with script
They were added manually



NOKIA

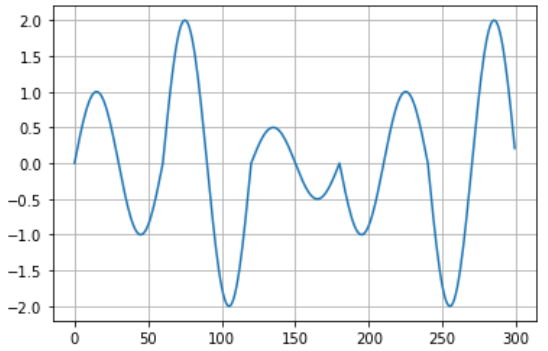
Single sinewave: **de**modulation: **sinewave periods separation**

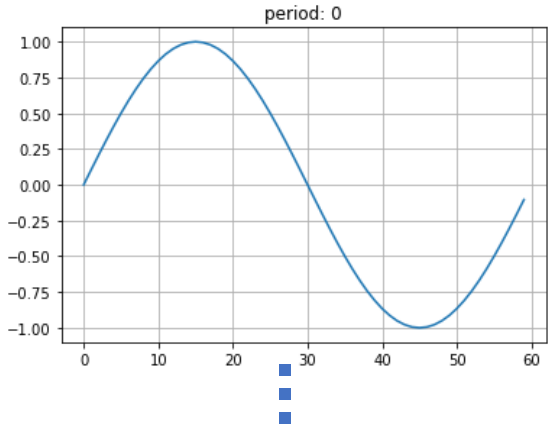
Below script is expected to:

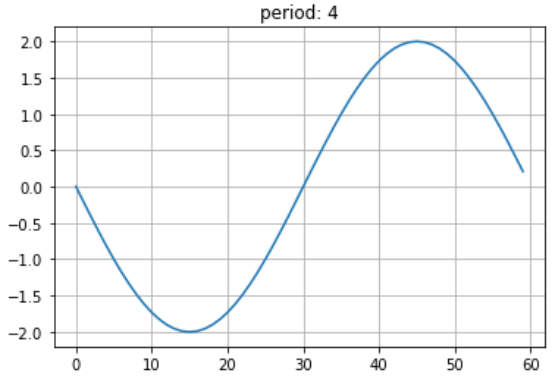
- loads and plots signal saved by previous script,
- Split signal to single periods,
- plots periods individually.

Complete the script to obtain plots as on the right.

```python
import numpy as np
import matplotlib.pyplot as plt

# parameters
PERIOD_VECTOR_SIZE = 60
PERIOD_NUMBER = 5

# loading Rx vector from file and..
Rx = np.load('TxSignal.npy')

# .. ploting it
plt.plot(Rx)
plt.grid()
plt.show()

# spliting vector into time slots coresponding to single periods
RxPeriods = np.reshape(Rx, shape: (PERIOD_NUMBER,PERIOD_VECTOR_SIZE))

# plotting periods one by one
for i, RxPeriod in enumerate(RxPeriods):
    plt.plot(RxPeriod)
    plt.title(f'period:{i}')
    plt.grid()
    plt.show()
```







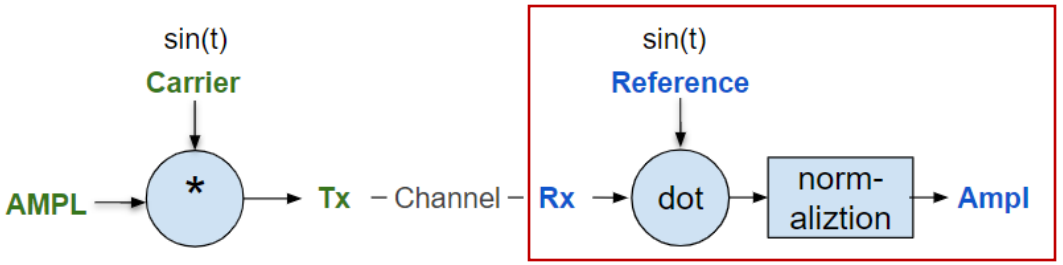
NOKIA

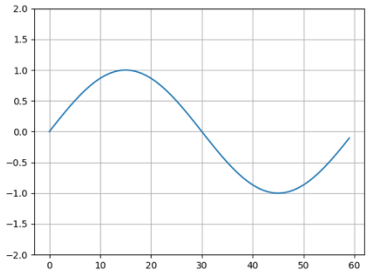
Single sinewave: **de**modulation: **dot product and normalization**

The previous script has been extended with amplitude decoding.

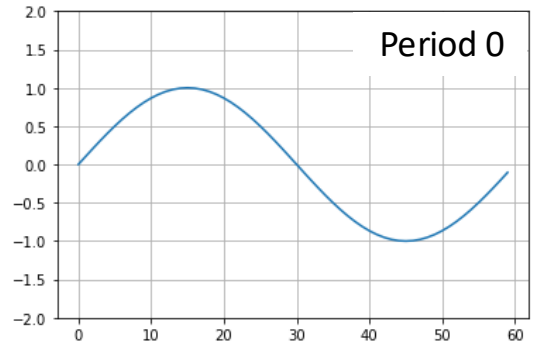
Complete the script to print amplitudes.

```
import ...
pi = np.pi

# parameters
PERIOD_VECTOR_SIZE = 60
PERIOD_NUMBER = 5

# loading Rx vector from file and..
Rx = np.load('TxSignal.npy')

# .. ploting it
plt.plot(Rx)
plt.grid()
plt.show()

# spliting vector into time slots coresponding to single periods
RxPeriods = np.reshape(Rx, newshape: (5,60))

# rx time
for RxPeriod in RxPeriods:
    # plotting
    plt.plot(RxPeriod)
    plt.ylim( *args: -2,2)
    plt.grid()
    plt.show()
    # decoding
    t = np.linspace( start: 0, 2*pi,PERIOD_VECTOR_SIZE, endpoint=False)
    Ref  = np.sin(t)
    dot  = np.dot(RxPeriod,Ref)
    ampl = dot/PERIOD_VECTOR_SIZE*2
    print(f'ampl = {ampl:0.2f}')
```

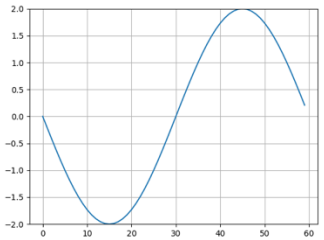




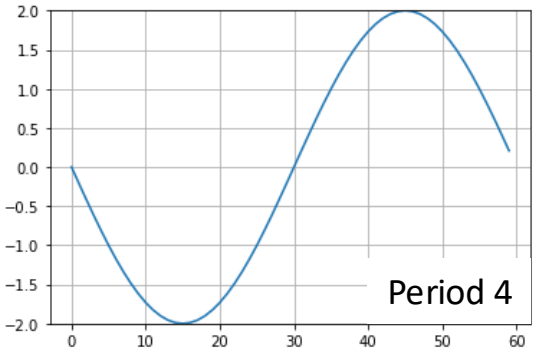
```
ampl = 1.00
```




```
ampl = 1.0000000000000002
```



```
ampl = -2.00
```




```
ampl = -2.0000000000000004
```


NOKIA

Single sinewave: Demodulation: Dot product and normalization, **cont.**

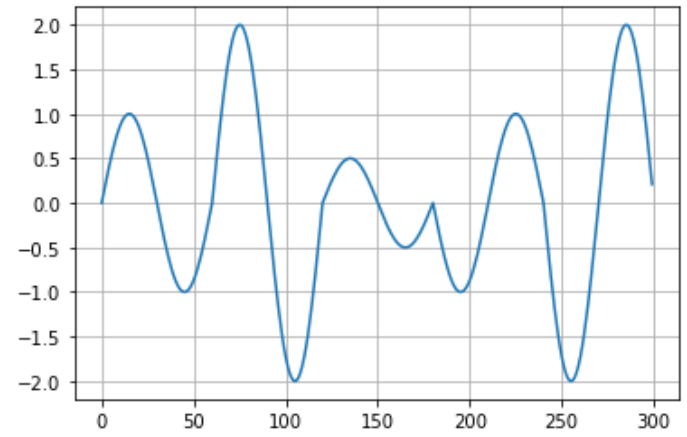
This script is a modified version of the script from previous slide.

It is expected to print amplitudes in a form of single vector (list).

Complete it to get result as shown below.

```python
import numpy as np
import matplotlib.pyplot as plt
pi = np.pi

# parameters
PERIOD_VECTOR_SIZE = 60
PERIOD_NUMBER = 5

# loading Rx vector from file and..
Rx = np.load('TxSignal.npy')

# .. ploting it
plt.plot(Rx)
plt.grid()
plt.show()

# spliting vector into time slots coresponding to single periods
RxPeriods = np.reshape(Rx, shape: (5,60))

# decoding amplitudes of Rx time slots

a_list = [] # create amplitude list
for RxPeriod in RxPeriods:
    # decode amplitude and append it to list
    # use as many code rows as you need
    t = np.linspace( start: 0, 2*pi,PERIOD_VECTOR_SIZE, endpoint=False)
    Ref = np.sin(t)
    dot = np.dot(RxPeriod, Ref)
    ampl = dot / np.dot(Ref, Ref)
    a_list.append(ampl)

np.set_printoptions(precision=16, legacy='1.13')  # set print options
print(a_list)  # print list
```

```
[1.0, 2.0, 0.5, -1.0, -2.0]
```



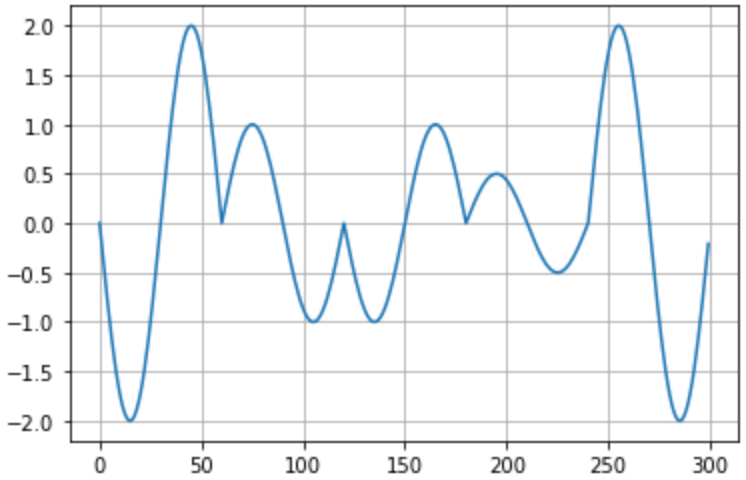

```
[1.0000000000000002, 2.0000000000000004, 0.5000000000000001, -1.0000000000000002, -2.0000000000000004]
```

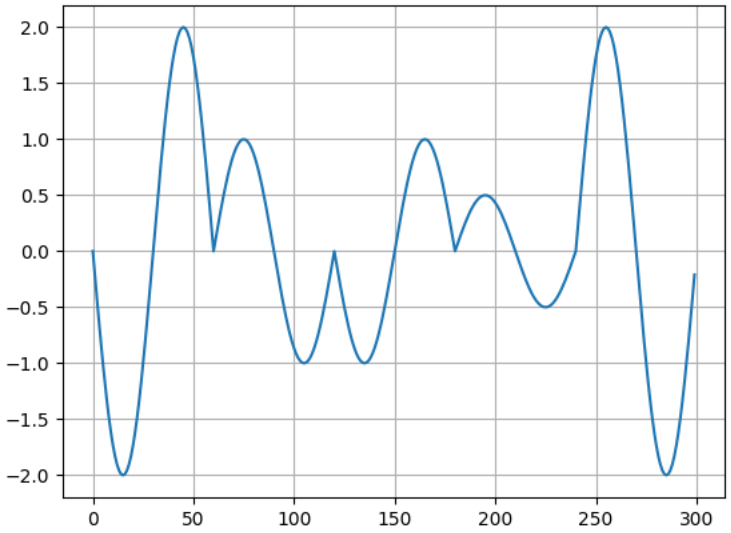

NOKIA

Single sinewave: Demodulation: Dot product and normalization, **cont.**

Use previous script to decode signal from “TxSignal_Exercise.npy” file.

Result should look as follow.





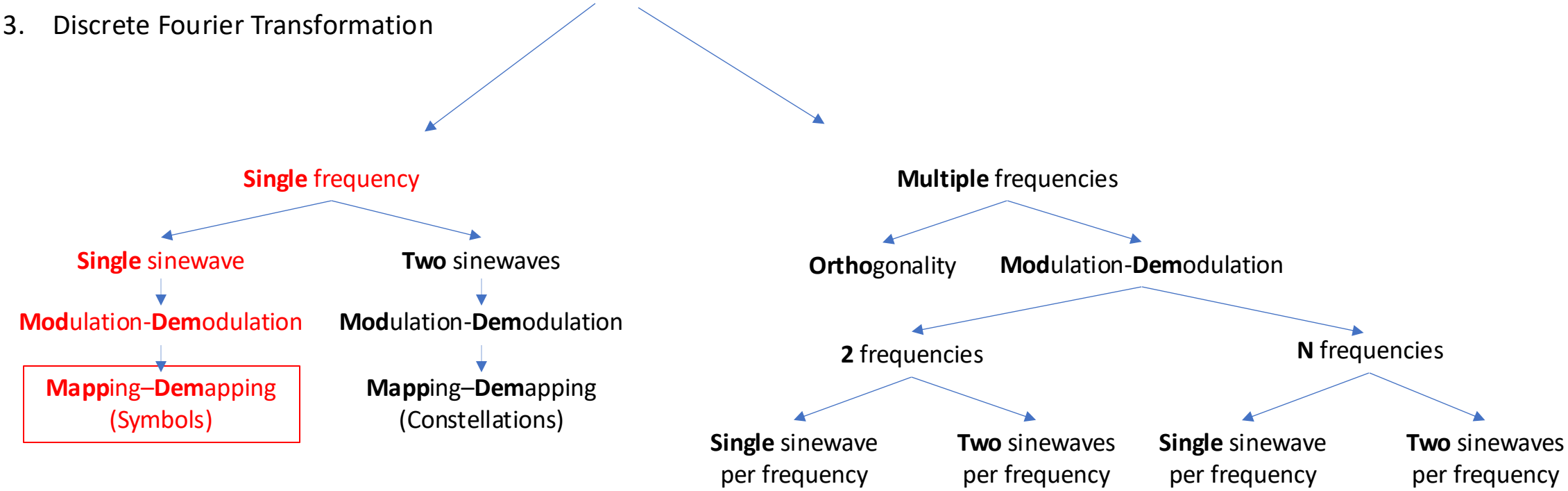

```
[-2.0, 1.0, -1.0, 0.5, 2.0]
```

```
[-2.0000000000000004, 1.0000000000000002, -1.0000000000000002, 0.5000000000000001, 2.0000000000000004]
```


NOKIA

You are here

1. Sinewaves orthogonality
2. Orthogonal sinewaves amplitude modulation
3. Discrete Fourier Transformation



NOKIA

Mapping–Demapping : Preparations : Files

Copy following scripts

`2_OrthogonalSinewavesAmplitudeModulation\1_SingleFrequency\1_SingleSineWave\`**`1_Modulation.py`**

`2_OrthogonalSinewavesAmplitudeModulation\1_SingleFrequency\1_SingleSineWave\`**`2_Demodulation_Step3_AmplitudeDecodingVector.py`**

to directory:

`2_OrthogonalSinewavesAmplitudeModulation\1_SingleFrequency\1_SingleSineWave\`**`3_MapingDemaping`**

and than rename copied file:

- `1_Modulation.py` -> **`1_Modulate.py`**
- `2_Demodulation_Step3_AmplitudeDecodingVector.py` -> **`2_Demodulte.py`**

Recall the functionalities above scripts.

Run `1_Modulate.py` and than `2_Demodulte.py`.

Check whether `2_Demodulte.py` prints AMPL_VECTOR from `1_Modulate.py`

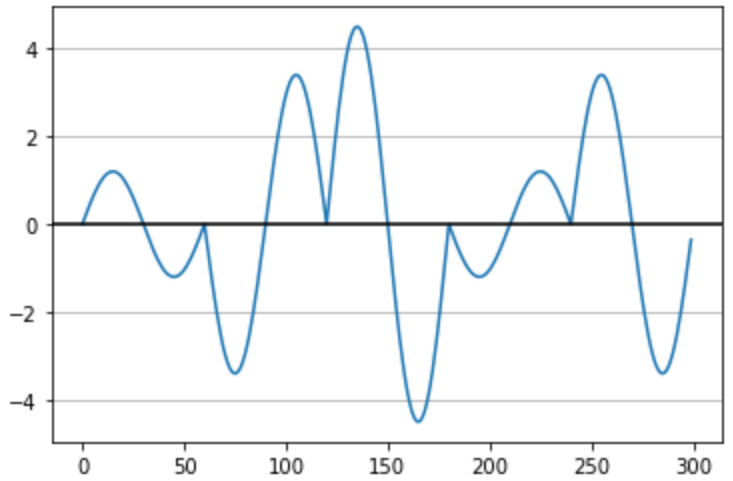
preparations : **mod-demod scripts merging**

Merge **`Modulate_1.py`** and **`Demodulate_2.py`** in single script **`3_ModDemod.py`**

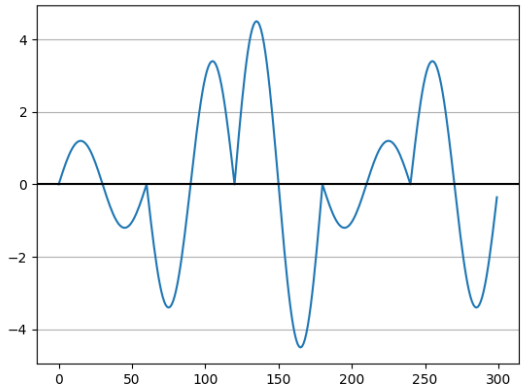
Remove ALL redundant code

```python
import numpy as np
import matplotlib.pyplot as plt
pi = np.pi
# PARAMETERS
TIME_VECTOR_SIZE = 60
AMPL_VECTOR = (1.2, -3.4, 4.5, -1.2, 3.4)
# CALCULATION
t = np.linspace( start: 0, 2*pi,TIME_VECTOR_SIZE, endpoint=False)
# modulation
Carrier = np.sin(t)
Tx = np.array([])
for amp in AMPL_VECTOR:
    Tx = np.append(Tx, amp * Carrier)
# channel
Rx=Tx # ideal one
# demodulation
RxPeriods = np.reshape(Rx, shape: (5,60))
amplitudes_l = [] # create amplitude list
for RxPeriod in RxPeriods:
    Ref = np.sin(t)
    dot = np.dot(RxPeriod, Ref)
    ampl = dot / np.dot(Ref, Ref)
    amplitudes_l.append(ampl)
# PRESENTATION
# Tx plot
plt.plot(Rx)
plt.axhline(y=0,color='black')
plt.grid(axis='y')
plt.show()
#
np.set_printoptions(precision=16, legacy='1.25')
print(f'received amplitudes: {amplitudes_l}')
```

dpoint=False)




```
received amplitudes: [1.2, -3.4, 4.5, -1.2, 3.4]
```




```
received amplitudes: [1.2, -3.4, 4.5, -1.2, 3.4]
```

NOKIA

preparations : **code refactoring**

Copy **`ModDemod.py`** to **`4_ModDemod_BySingleNumber.py`**

At present version, the script first modulates all five numbers and after that demodulates them.

Modify **`4_ModDemod_BySingleNumber.py`** to modulate and demodulate per single number (reduce loops to one)

Add perfect channel between modulation and demodulation.

Decrease transmitted numbers to 4

Plot sinewave corresponding to the last number only (plt.plot(Rx) after loop)

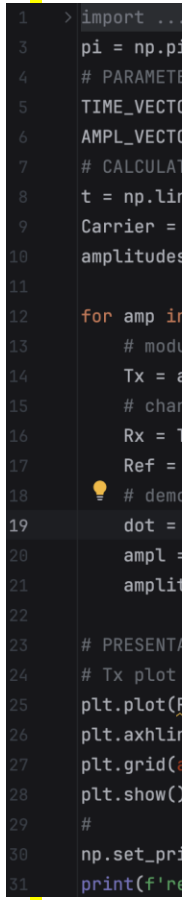
Remove ALL redundant code

Before refactoring

```
# modulation
for … all five numbers:
    …

# demodulation
for … all five numbers:
    …
```

After refactoring

```
for … all five numbers:

    # modulation
    …

    # perfect channel
    Rx = Tx

    # demodulation
    …
```



NOKIA

real channel : errors

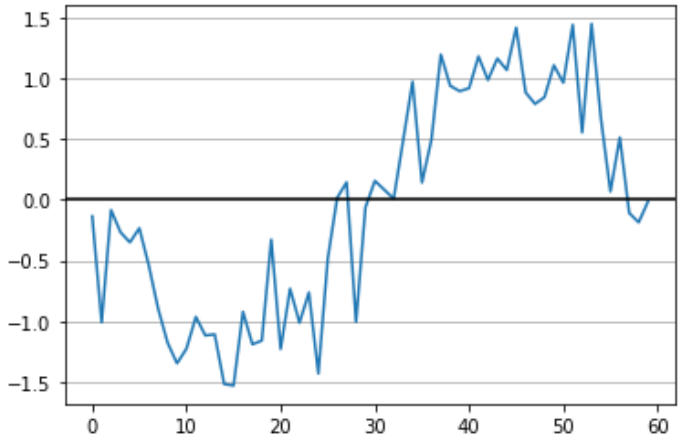
Copy **`4_ModDemod_BySingleNumber.py`** to **`5.1_ModDemod_RealChannel.py`**

Add normal distribution noise to channel (numpy.random.normal)

Set distribution center location at 0, deviation to 0.3 and size accordingly to Tx length

Remove redundand code

Run the script to see noise on the sine




```
import ...
pi = np.pi
# PARAMETERS
TIME_VECTOR_SIZE = 60
AMPL_VECTOR = (1.2, -3
# CALCULATION
t = np.linspace( start: 0
Carrier = np.sin(t)
amplitudes_l = []

for amp in AMPL_VECTOR
    # modulation
    Tx = amp*Carrier
    # channel
    Rx = Tx + np.rando
    # demodulation
    dot = np.dot(Rx, C
    ampl = dot / np.do
    amplitudes_l.appen

ampl1 = np.array(AMPL_
ampl2 = np.array(ampli
errors = ampl1 - ampl2

# PRESENTATION
# Tx plot
plt.plot(Rx)
plt.axhline(y=0,color=
plt.grid(axis='y')
plt.show()
#
np.set_printoptions(pr
print(f'received ampli
print(f'errors

print(f'errors
```

Add error calculation and printing.

```
received amplitudes: [1.2321601568330671, -3.4773478822319728, 4.447983206633805, -1.1685442767382594]
errors     : [-0.03216016  0.07734788  0.05201679 -0.03145572]
```

To do that calculate difference between send and received amplitudes

Hint: before subtraction convert list of received amplitudes to numpy array

NOKIA

real channel : **noise deviation**

Copy **5.1_ ModDemod_RealChannel.py** to **5.2_ ModDemod_Ranges.py**

Add parameter NOISE_DEVIATION to define “scale” argument of the “normal” function and set it to 1.

Increase number of transmission to 100 (single numbers)

Hint:

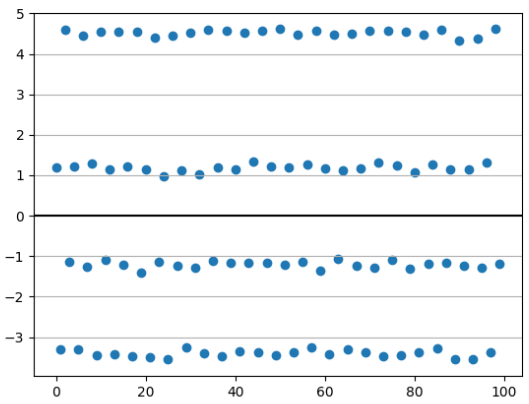
```
for i in range(TANSMISION_NR):
    amp = AMPL_VECTOR[i%4]
```

Plot received numbers versus time for the noise deviations: 0.5 and 2 and place below

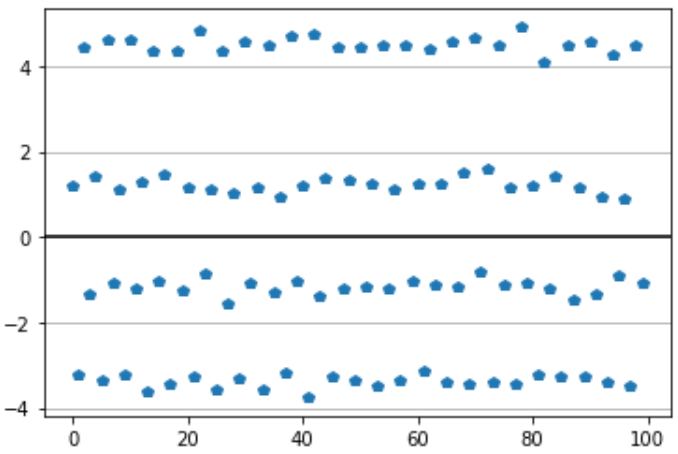
Remove redundant code (error calculation)

```
import ...
pi = np.pi
# PARAMETERS
TIME_VECTOR_SIZE = 60
AMPL_VECTOR = (1.2, -3.4, 4.5,
# CALCULATION
t = np.linspace( start: 0, 2*pi,
Carrier = np.sin(t)
amplitudes_l = []
NOISE_DEVIATION = 2
TRANSMISSION_NR = 100

for i in range(TRANSMISSION_NR
    amp = AMPL_VECTOR[i % 4]
    # modulation
    Tx = amp*Carrier
    # channel
    Rx = Tx + np.random.normal
    # demodulation
    dot = np.dot(Rx, Carrier)
    ampl = dot / np.dot(Carrie
    amplitudes_l.append(ampl)

# PRESENTATION
# Tx plot
plt.scatter(range(TRANSMISSION
plt.axhline(y=0,color='black')
plt.grid(axis='y')
plt.show()
#
np.set_printoptions(precision=
```

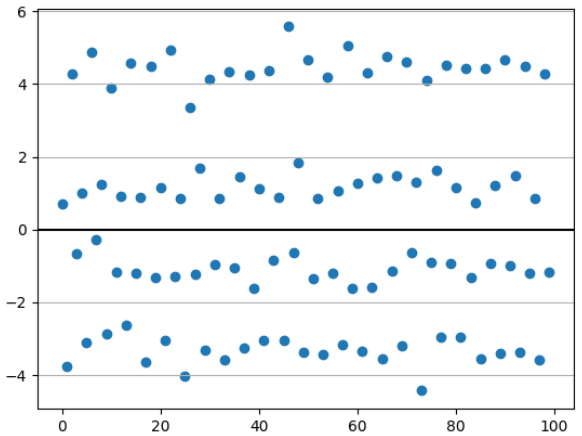
dev=**0.5**



dev=**1**



dev=**2**



time

noise deviation, **cont.**

Observations

In practice, one can not transfer numbers in reliable way directly as amplitudes of sinewave.

For a given noise deviation received amplitudes values are dispersed within a certain range.

So, one sends **specific values** but receives values in **specific range**.

Unlike the number of values, the **number of ranges is limited.**

Question

What is the relationship between noise deviation and number of nonoverlapping ranges?
Assume ranges are distributed evenly.
Give descriptive answer. The relationship between noise deviation and number of nonoverlapping ranges is that as the noise deviation increases , there is a higher possibility that the noisy amplitudes will overlap with neighbouring ranges. As a result, the number of non-overlapping ranges decrases.

noise deviation vs. amplitudes symbol number

Conclusion

We can send only **limited set of values** (amplitudes), and the number of values in the set depends on noise deviation.
The elements of that set are called **symbols**.

Code Refactoring

Copy **5.2_ModDemodRanges.py** to **5.3_ModDemodRangesColor.py**

Set `AMPL_VECTOR` to `(1, 2, 3, 4)`

Modify amplitudes plot to draw each symbol with unique color.
Use list indexing.

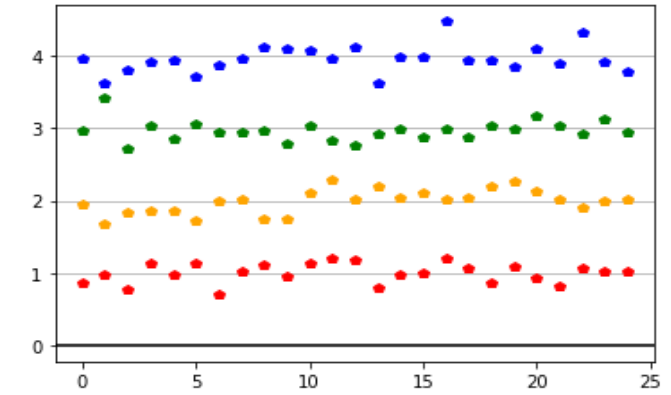
```
# Tx plot
plt.plot(amplitudes_l[...::...],'p', color='red')    # each first
plt.plot(amplitudes_l[...::...],'p', color='orange') # each seccond
plt.plot(amplitudes_l[...::...],'p', color='green')  # each thirdt
plt.plot(amplitudes_l[...::...],'p', color='blue')   # each fourth
```

Exercise

Use the script to estimate noise deviation acceptable for 8 symbols (amplitudes).

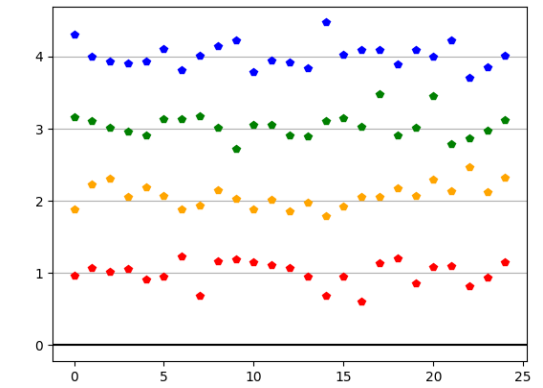
Assume we are limited to same amplitude range.

NOISE_DEVIATION = 1




```
import ...

pi = np.pi
# PARAMETERS
TIME_VECTOR_SIZE = 60
AMPL_VECTOR = (1, 2, 3,
# CALCULATION
t = np.linspace( start: 0,
Carrier = np.sin(t)
amplitudes_l = []
NOISE_DEVIATION = 1
TRANSMISSION_NR = 100

for i in range(TRANSMISS
    amp = AMPL_VECTOR[i
    # modulation
    Tx = amp*Carrier
    # channel
    Rx = Tx + np.random.
    # demodulation
    dot = np.dot(Rx, Car
    ampl = dot / np.dot(
    amplitudes_l.append(

# PRESENTATION
# Tx plot
plt.plot( *args: amplitude
plt.plot( *args: amplitude
plt.plot( *args: amplitude
plt.plot( *args: amplitude
plt.axhline(y=0,color='b
plt.grid(axis='y')
plt.show()
#
np.set_printoptions(prec
```

Estimation of acceptable noise dev.
0.5 both directions



NOKIA

symbol-amplitude mapping

Symbol-amplitude mapping

Usually, symbols have a form a set of bits.

Each combination is assigned to specific amplitude (Tx) and amplitude range (Rx).

Tables on the right show examples of symbol-amplitude mapping

for 2, 4 and 8 symbol transmission schemes for constant amplitude range (c.a. 0.0-5.0)

To transmit specific bitstream one must divide it to groups of bits

of the size appropriate for size of the symbol .

Example

Data to transfer: byte **00100111**: (lsb first)

Transmission scheme: 4 symbols

Symbol: Tx ampl. : Rx range

- 11: 2.0: 1.75-2.25
- 01: 1.0: 0.75-1.25
- 10: 1.5: 1.25-1.75
- 00: 0.5: 0.25-0.75

2 symbols

symbol	Tx amplitude	Rx amplitude range
0	1	0.5 - 1.5
1	2	1.5 - 2.5

4 symbols

symbol	Tx amplitude	Rx amplitude range
00	0.5	0.25 - 0.75
01	1.0	0.75 - 1.25
10	1.5	1.25 - 1.75
11	2.0	1.75 - 2.25

8 symbols

symbol	Tx amplitude	Rx amplitude range
000	0.25	0.125 - 0.375
001	0.50	0.375 - 0.625
010	0.75	0.625 – 0.875
011	1.00	0.875 – 1.125
100	1.25	1.125 – 1.375
101	1.50	1.375 – 1.625
110	1.75	1.625 – 1.875
111	2.00	1.875 – 2.125

symbol-amplitude mapping : **exercise**

Exercise

Fulfill table with symbols and amplitudes by using 8 symbol Tx scheme

Data to transfer: byte 00100111111100111111100111 : (lsb first)

symbol	Tx amplitude	Rx amplitude range
111	2.00	1.875 – 2.125
100	1.25	1.125 – 1.375
111	2.00	1.875 – 2.125
011	1.00	0.875 – 1.125
110	1.75	1.625 – 1.875
111	2.00	1.875 – 2.125
001	0.50	0.375 - 0.625
001	0.50	0.375 - 0.625

Cheat sheet

8 symbols

symbol	Tx amplitude	Rx amplitude range
000	0.25	0.125 - 0.375
001	0.50	0.375 - 0.625
010	0.75	0.625 – 0.875
011	1.00	0.875 – 1.125
100	1.25	1.125 – 1.375
101	1.50	1.375 – 1.625
110	1.75	1.625 – 1.875
111	2.00	1.875 – 2.125

NOKIA

Symbol-Amplitude mapping: **mapping functions verification : “symbol_to_amplitude”**

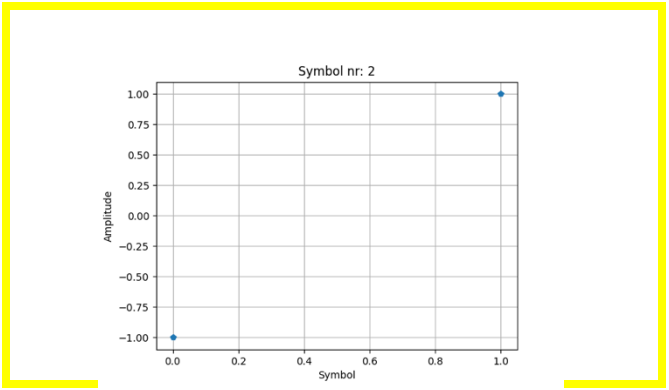
In the next exercises we will use readymade functions for “symbol-amplitude” mapping.
The functions make mapping between 2, 4 and 8 symbols amplitudes in range from -1 to 1.
They are collected in **`mapper_lib.py`** file so to take advantage of them you need to import them into you script.

Examine the content of **`mapper_lib.py`**

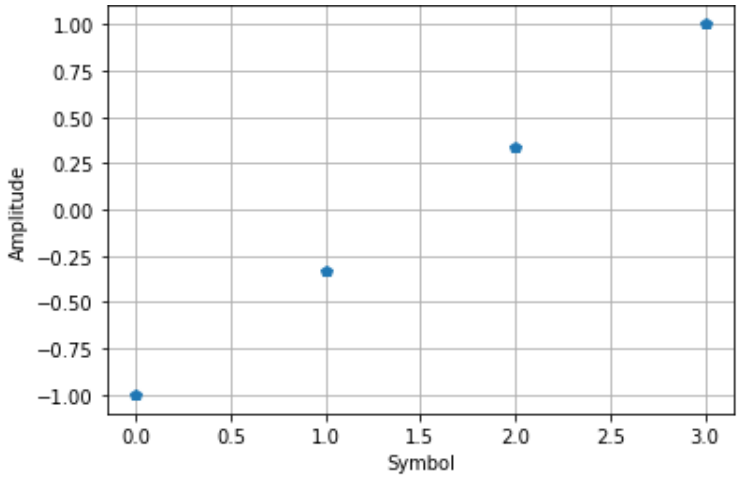
Below script plots mapping from symbol to amplitude for 4 symbols scheme.
Generate plots for remaining symbol numbers (put them on this slide).

```
import numpy as np
import matplotlib.pyplot as plt
from mapper_lib import symbol_to_ampl

SYMBOL_NR = 4

symbols_l = range(SYMBOL_NR)
ampl_l = list()
for symbol in symbols_l:
    ampl = symbol_to_ampl(SYMBOL_NR,symbol)
    ampl_l.append(ampl)
plt.plot(symbols_l,ampl_l,'-p')

plt.grid()
plt.title(f'Symbol nr: {SYMBOL_NR}')
plt.xlabel('Symbol')
plt.ylabel('Amplitude')
```

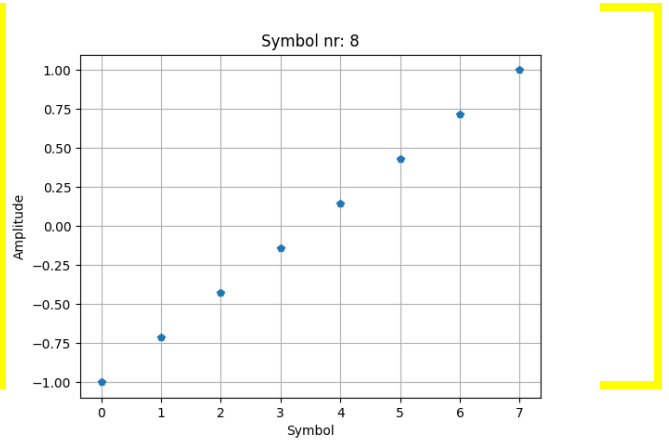
Symbol nr = **2**



Symbol nr = **4**



Symbol nr = **8**

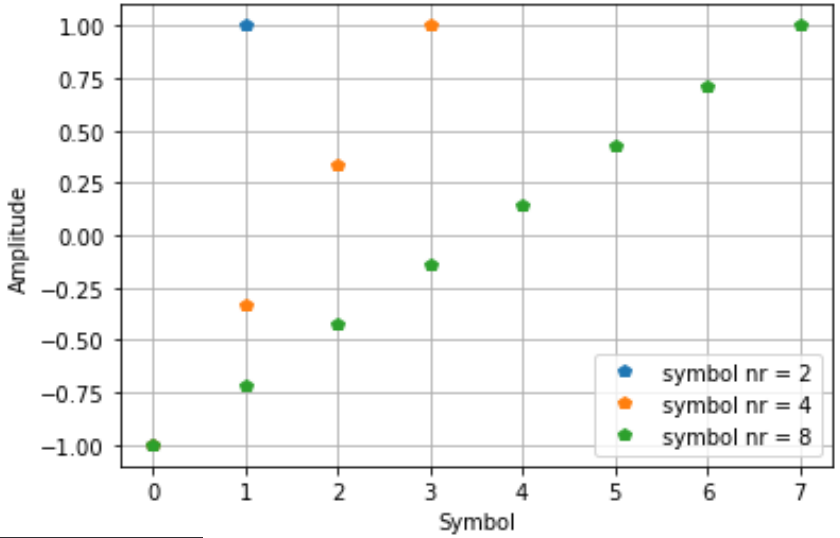


NOKIA

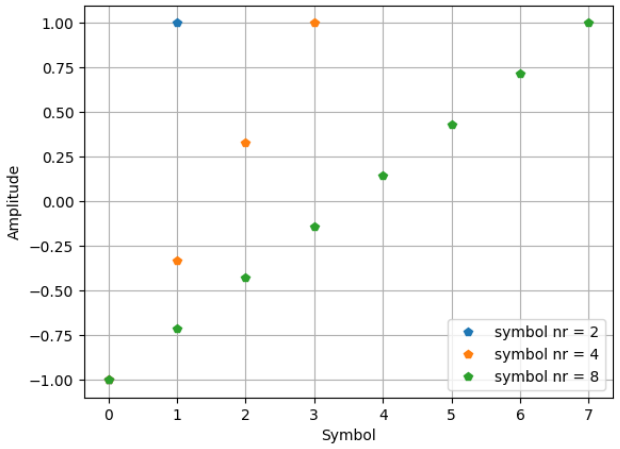
Symbol-Amplitude mapping: **mapping functions verification : “symbol_to_amplitude” cont.**

Complete below script to plot mapping from symbol to amplitude for transmission schemes with symbol number equals to 2, 4 and 8

```
for symbol_nr in 2,4,8:
    symbols_l = range(...)
    ampl_l = list()
    for symbol in symbols_l:
        ampl = symbol_to_ampl(symbol_nr,symbol)
        ampl_l.append(ampl)
    plt.plot(symbols_l,ampl_l,'p', label=...)

plt.legend()
plt.grid()
plt.xlabel('Symbol')
plt.ylabel('Amplitude')
```




```
import numpy as np
import matplotlib.pyplot as plt
from mapper_lib import symbol_to_ampl

for symbol_nr in 2,4,8:
    symbols_l = range(8)
    ampl_l = list()
    for symbol in symbols_l:
        ampl = symbol_to_ampl(symbol_nr,symbol)
        ampl_l.append(ampl)
    plt.plot( *args: symbols_l,ampl_l,'p', label=f'symbol nr = {symbol_nr}')

plt.legend()
plt.grid()
plt.xlabel('Symbol')
plt.ylabel('Amplitude')
plt.show()
```



NOKIA

Symbol-Amplitude mapping: **mapping functions verification : “amplitude_to_symbol”**

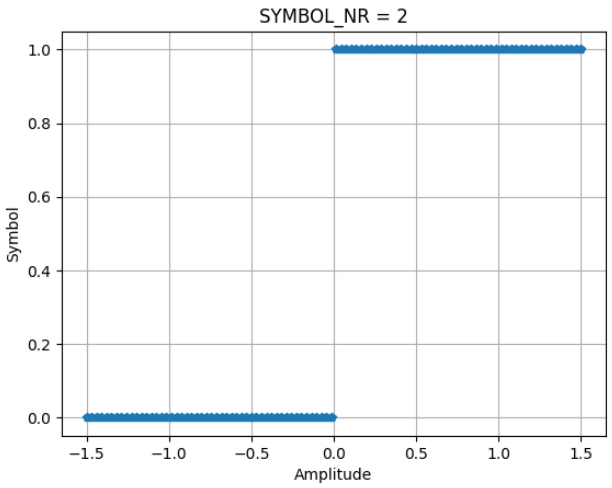
Base one of the previous script write script for plotting mapping form amplitude to symbol.

Name the script **`6.2.1_Maping_AmplToSymbol_Single.py`**.

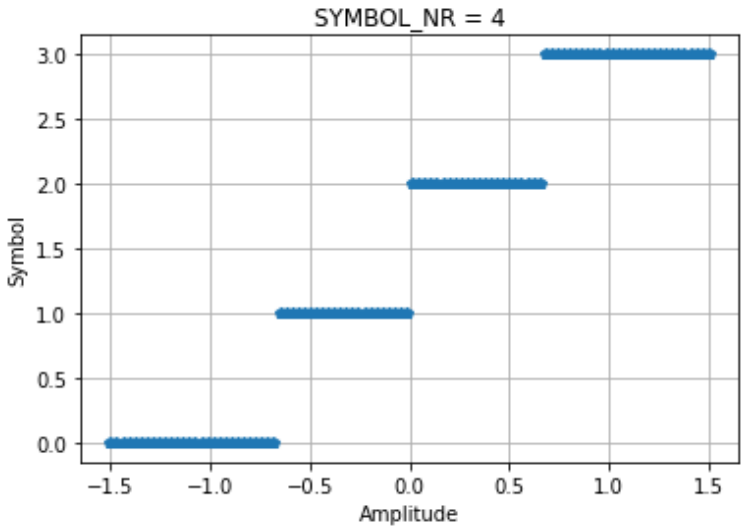
Parametrize transmission scheme: “SYMBOL_NR = …” in the beginning of script.

Use the script to add missing figures.

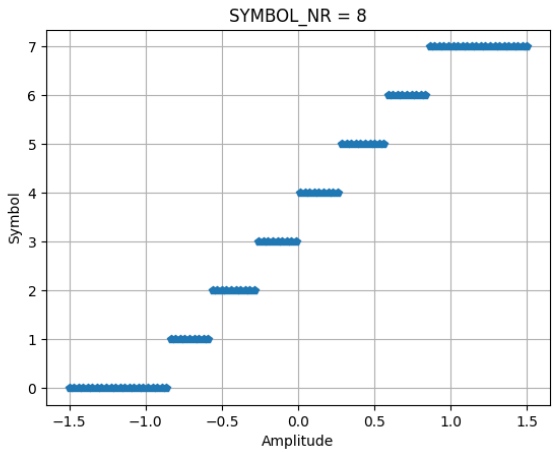
SYMBOL_NR = 2



SYMBOL_NR = 4



SYMBOL_NR = 8



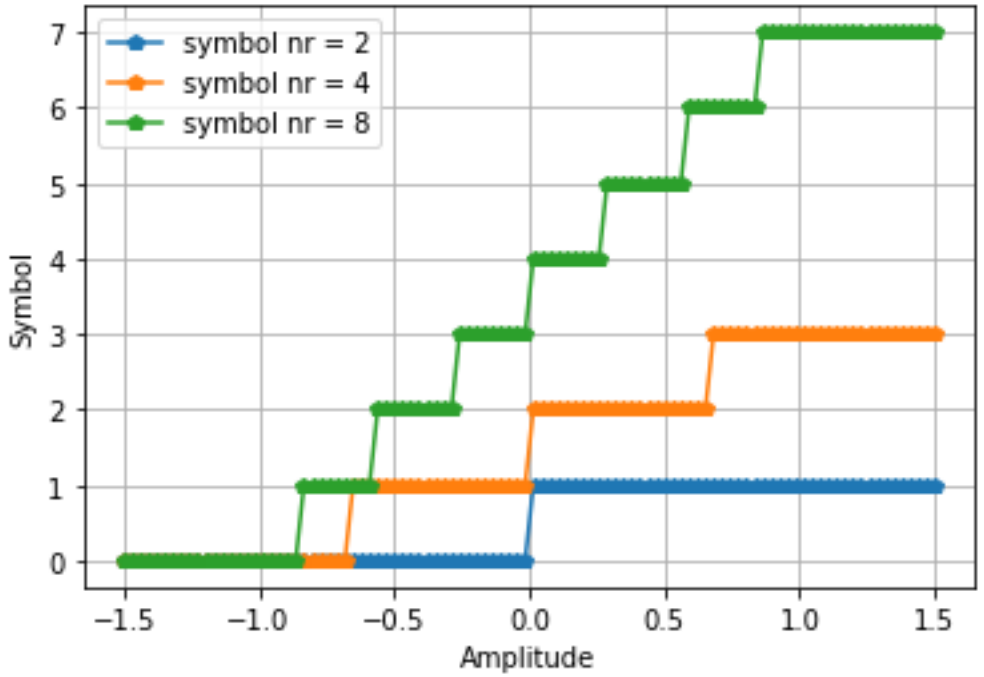

```
import numpy as
import matplotlib
from mapper_lib i

SYMBOL_NR = 2

symbols_l = list(
ampl_l = np.linsp
for ampl in ampl_
    symbol = ampl
    symbols_l.app
plt.plot( *args: a

plt.grid()
plt.title(f'SYMBO
plt.xlabel('Ampli
plt.ylabel('Symbo
plt.show()
```


NOKIA

Symbol-Amplitude mapping: **mapping functions verification : “amplitude_to_symbol” cont.**

Base one of the previous script write script to plot figure as below.

Name the script **`6.2.2_Maping_AmplToSymbol_Multiple.py`**.




```python
import numpy as np
import matplotlib.pyplot as plt
from mapper_lib import ampl_to_symbol

for symbol_nr in 2, 4, 8:
    ampl_l = np.linspace(-1.5, stop: 1.5, num: 100)
    symbols_l = list()
    for ampl in ampl_l:
        symbol = ampl_to_symbol(symbol_nr, ampl)
        symbols_l.append(symbol)
    plt.plot( *args: ampl_l, symbols_l, '-p', label=f'symbol nr = {symbol_nr}')

plt.legend()
plt.grid()
plt.xlabel('Symbol')
plt.ylabel('Amplitude')
plt.show()
```


NOKIA

Symbols transmission and reception

Complete below script to make given number of **transmissions\receptions** by using given number of **symbols** in the presence of given **noise** distribution

```python
import numpy as np
from mapper_lib import symbol_to_ampl
from mapper_lib import ampl_to_symbol

# PARAMETERS
TIME_VECTOR_SIZE = 60
TRANSMISIONS_NR = 10

NOISE_DEVIATION = 2.0
SYMBOL_NR = 4

t = np.linspace( start: 0, 2*np.pi,TIME_VECTOR_SIZE, endpoint=False)
Carrier = np.sin(t)
Ref     = Carrier

# TRANSMISION-RECEPTION
symbols_tx = np.random.randint( low: 0,SYMBOL_NR,TRANSMISIONS_NR)
symbols_rx = list()
for symbol in symbols_tx:
    # modulation
    ampl = symbol_to_ampl(SYMBOL_NR,symbol)
    Tx = ampl*Carrier
    # real channel
    Rx = Tx + np.random.normal( loc: 0,NOISE_DEVIATION,TIME_VECTOR_SIZE)
    # demodulation
    ampl = (np.dot(Rx,Ref)/TIME_VECTOR_SIZE)*2
    symbol = ampl_to_symbol(SYMBOL_NR, ampl)

    symbols_rx.append(symbol)

# PRESENTATION
symbols_rx = np.array(symbols_rx) # list to numpy array

# print('symbols_rx')
print('\n symbols_rx\n:', symbols_tx)
errors = symbols_rx != symbols_tx
print('\n errors:\n', errors)
print('\n errors nr:', sum(symbols_rx != symbols_tx))
```

Typical result

```
 symbols_rx
: [3 2 2 1 0 0 2 3 2 1]

 errors:
[False False False  True False False  True False  True False]

 errors nr: 3
```

Observe results while changing NOISE_DEVIATION and SYMBOL_NR

NOKIA

Symbols transmission and reception, **error number v.s. noise deviation**

Save previous script as **`7.2_ModDemodSymbols_ErrorRate_vs_Noise_Print.py`**.

Modify it to evaluate transmission error for given range of noise deviation (see exemplary result below).

Set TRANSMISIONS_NR to 1000

Use the script to add missing results below.

SYMBOL_NR = 2

```
noise dev : errors nr
0.00 : 0
0.22 : 0
0.44 : 0
0.67 : 0
0.89 : 0
1.11 : 0
1.33 : 0
1.56 : 0
1.78 : 0
2.00 : 4
```

SYMBOL_NR = 4

```
SYMBOL_NR: 4

noise dev : error nr
0.00: 0
0.22: 0
0.44: 0
0.67: 3
0.89: 36
1.11: 70
1.33: 141
1.56: 169
1.78: 238
2.00: 269
```

SYMBOL_NR = 8

```
noise dev : errors nr
0.00 : 0
0.22 : 1
0.44 : 58
0.67 : 188
0.89 : 339
1.11 : 435
1.33 : 474
1.56 : 547
1.78 : 572
2.00 : 601
```

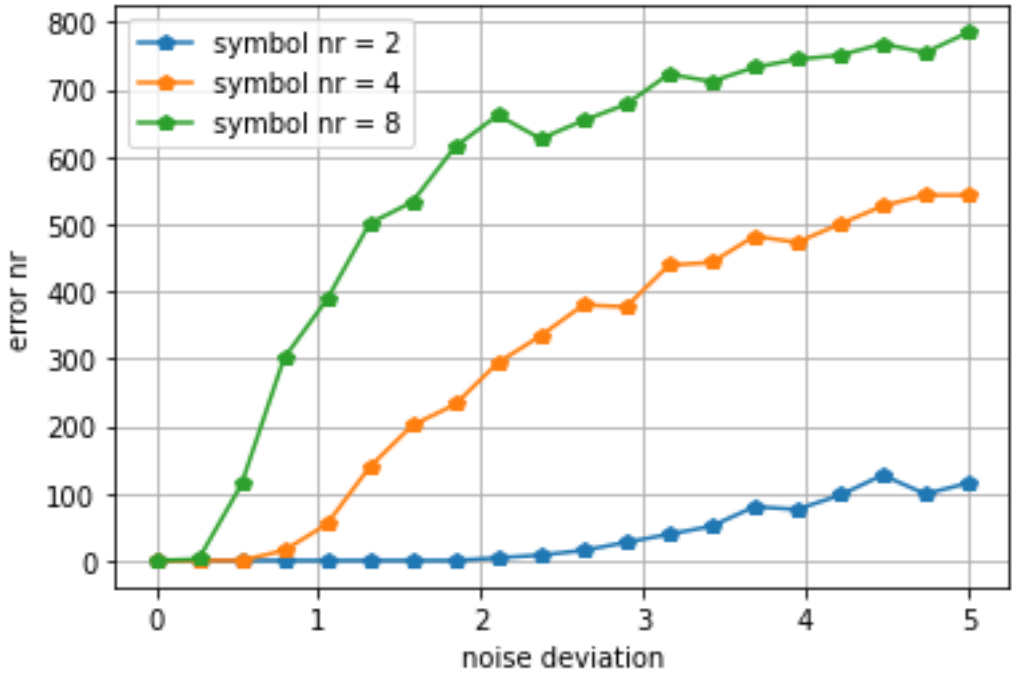
```python
import numpy as np
from mapper_lib import symbol_to_ampl
from mapper_lib import ampl_to_symbol

# PARAMETERS
TIME_VECTOR_SIZE = 60
TRANSMISIONS_NR = 1000
NOISE_DEVIATION = 2.0
SYMBOL_NR = 2
t = np.linspace( start: 0, 2 * np.pi, TIME_VECTOR_SIZE, endpoint=False)
Carrier = np.sin(t)
Ref = Carrier
noise_dev_list = list()
# TRANSMISION-RECEPTION
for noise_dev in np.linspace(0, NOISE_DEVIATION, 10):
    symbols_tx = np.random.randint( low: 0, SYMBOL_NR, TRANSMISIONS_NR)
    symbols_rx = list()
    error_nr = 0
    for symbol in symbols_tx:
        symbol1 = symbol
        # modulation
        ampl = symbol_to_ampl(SYMBOL_NR,symbol)
        Tx = ampl * Carrier
        # real channel
        Rx = Tx + np.random.normal( loc: 0, noise_dev, TIME_VECTOR_SIZE)
        # demodulation
        ampl = (np.dot(Rx, Ref) / TIME_VECTOR_SIZE) * 2
        symbol = ampl_to_symbol(SYMBOL_NR, ampl)
        symbols_rx.append(symbol)

        if symbol != symbol1:
            error_nr += 1

    noise_dev_list.append((noise_dev, error_nr))

# PRESENTATION
symbols_rx = np.array(symbols_rx)  # list to numpy array

print('SYMBOL_NR:', SYMBOL_NR)
print('noise dev' + ' : ' + 'errors nr')
for noise_dev, error_nr in noise_dev_list:
    print(f"{noise_dev:.2f} : {error_nr}")
```


NOKIA

Symbols transmission and reception, **error number v.s. noise deviation, cont.**

Save previous script as **`7.3_ModDemodSymbols_ErrorRate_vs_Noise_Plot.py`**.

Modify it to generate plots family as on the figure below




```python
import matplotlib.pyplot as plt
import numpy as np
from mapper_lib import symbol_to_ampl
from mapper_lib import ampl_to_symbol
# PARAMETERS
TIME_VECTOR_SIZE = 60
TRANSMISIONS_NR = 1000
NOISE_DEVIATION = 5
SYMBOL_NR = [2,4,8]
t = np.linspace( start: 0, 2 * np.pi, TIME_VECTOR_SIZE, endpoint=False)
Carrier = np.sin(t)
Ref = Carrier
# TRANSMISION-RECEPTION
for symbols in SYMBOL_NR:
    noise_dev_list = list()
    for noise_dev in np.linspace(0, NOISE_DEVIATION, 20):
        symbols_tx = np.random.randint( low: 0, symbols, TRANSMISIONS_NR)
        symbols_rx = list()
        error_nr = 0
        for symbol in symbols_tx:
            symbol1 = symbol
            # modulation
            ampl = symbol_to_ampl(symbols,symbol)
            Tx = ampl * Carrier
            # real channel
            Rx = Tx + np.random.normal( loc: 0, noise_dev, TIME_VECTOR_SIZE)
            # demodulation
            ampl = (np.dot(Rx, Ref) / TIME_VECTOR_SIZE) * 2
            symbol = ampl_to_symbol(symbols, ampl)
            symbols_rx.append(symbol)
            if symbol != symbol1:
                error_nr += 1
        noise_dev_list.append((noise_dev, error_nr))
    noise_dev_plot = [x[0] for x in noise_dev_list]
    error_nr_plot = [x[1] for x in noise_dev_list]
    plt.plot( *args: noise_dev_plot, error_nr_plot, '-p', label=f'symbol nr = {symbols}')
# PRESENTATION
symbols_rx = np.array(symbols_rx)  # list to numpy array
plt.grid()
plt.legend()
plt.xlabel('noise deviation')
plt.ylabel('error nr')
plt.show()
```


NOKIA

Symbols transmission and reception, error number v.s. noise deviation, **symbol nr selection**

There are methods to discover and correct errors in received data, but they work correctly below certain error rate.

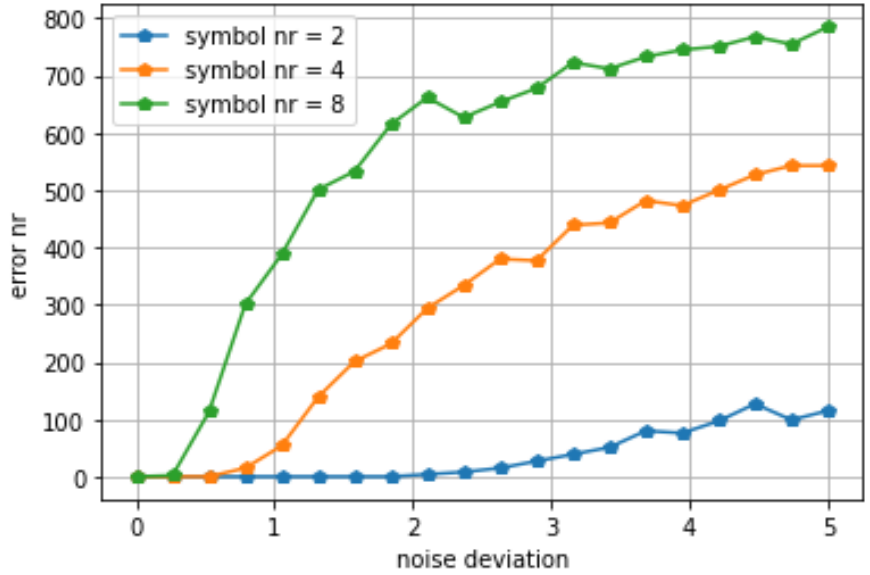
If error cannot be corrected data have to be retransmitted which is quiet time and power consuming.

Error rate depends on noise level which in turn depends on many factors like: distance to base station, weather conditions, buildings, and so on.

What's important, **noise level** (so error rate) can **change in time** so in order to guarantee **maximum transfer** rate at current conditions one has to use **symbol number** appropriate for current noise level.

If **symbol nr is too high** for current noise level (deviation) we get many errors and will have to make many retransmission which decrease effective transfer rate

If **symbol nr is too low** for current noise level (deviation) the number of errors will be low, but transfer will be also low so in practice w will be wasting possibilities given by channel.



NOKIA

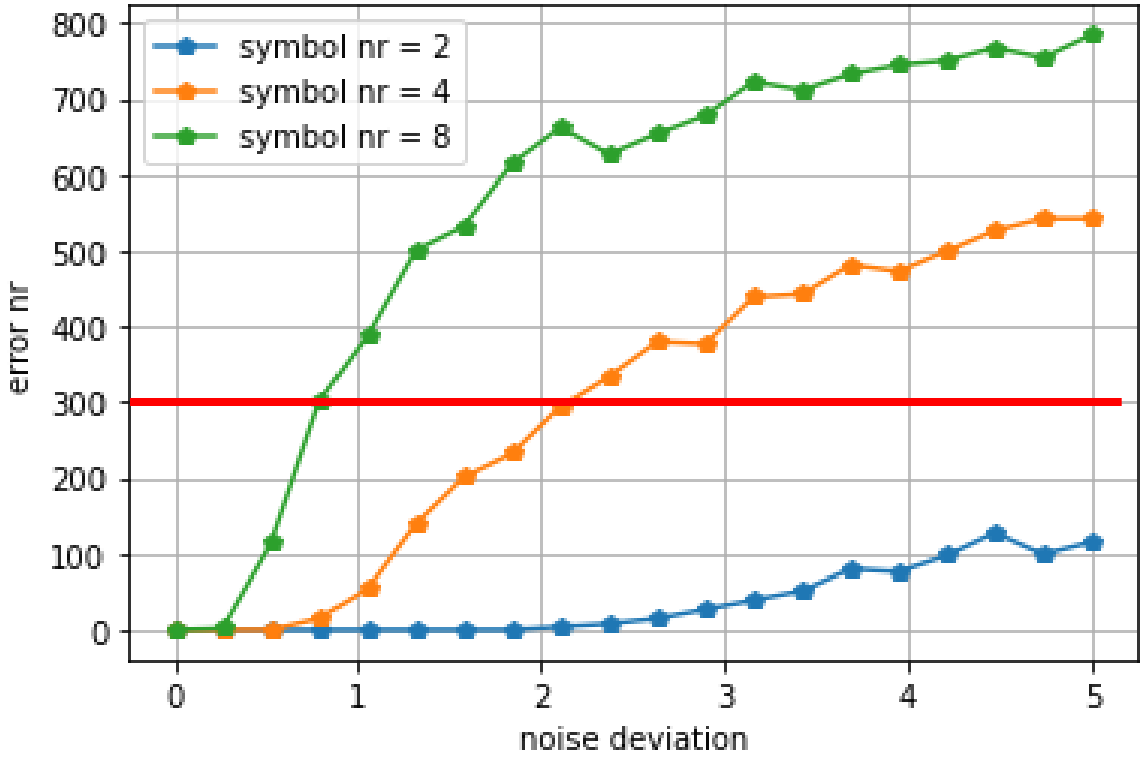
Symbols transmission and reception, error number v.s. noise deviation, **symbol nr selection - exercise**

As it was already said, to guarantee **maximum transfer** rate at current conditions one has to use **symbol number** appropriate for current noise level.

Base on figure below estimate acceptable and optimal symbol number for given:
- acceptable error number,
- noise deviation.

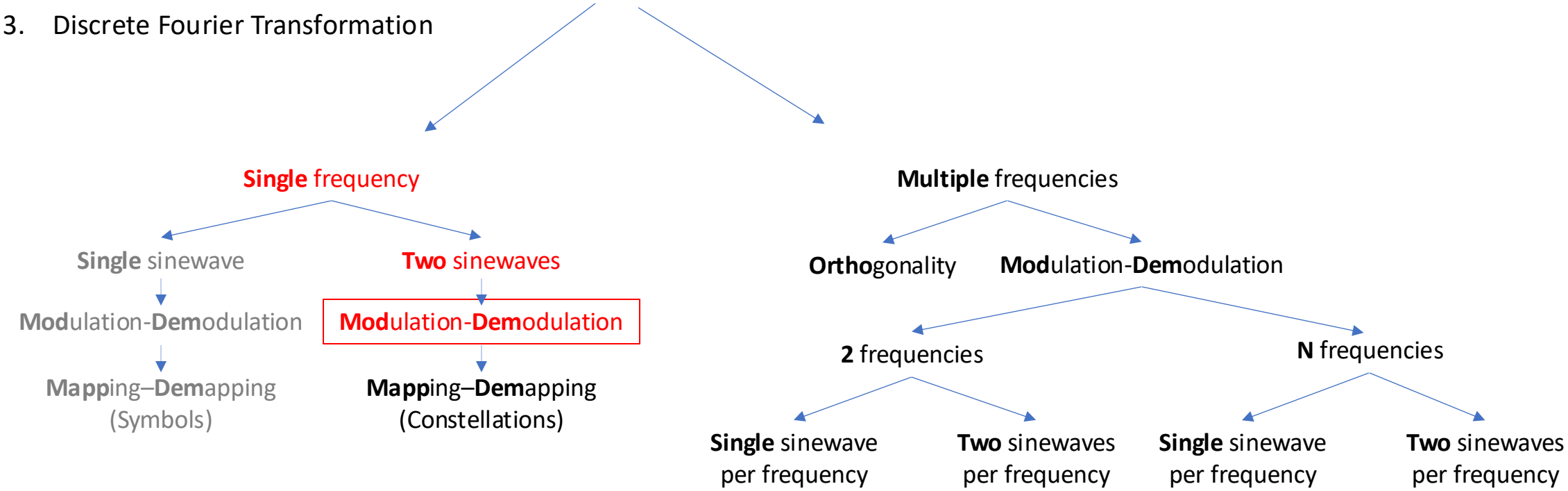
acceptable error number = **300**

noise	acceptable symbol nr(s)	optimal symbol nr
3.0	2	2
2.0	2,4	4
0.5	2,4,8	8



You are here

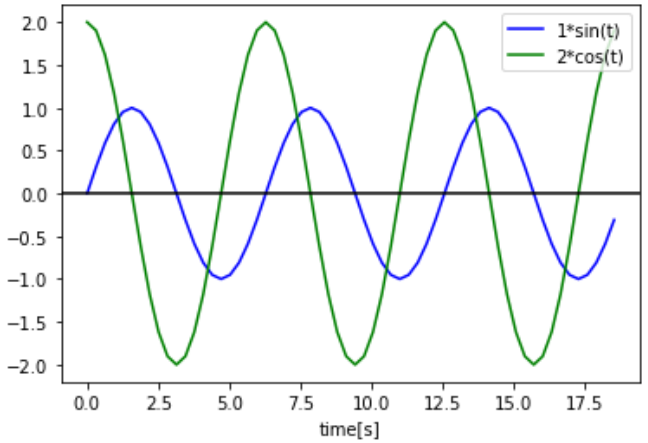
1. Sinewaves orthogonality
2. Orthogonal sinewaves amplitude modulation
3. Discrete Fourier Transformation

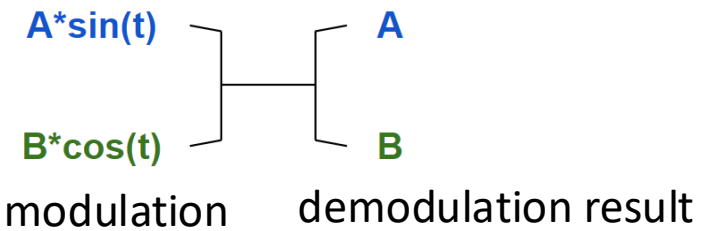


NOKIA

Quadrature modulation: idea

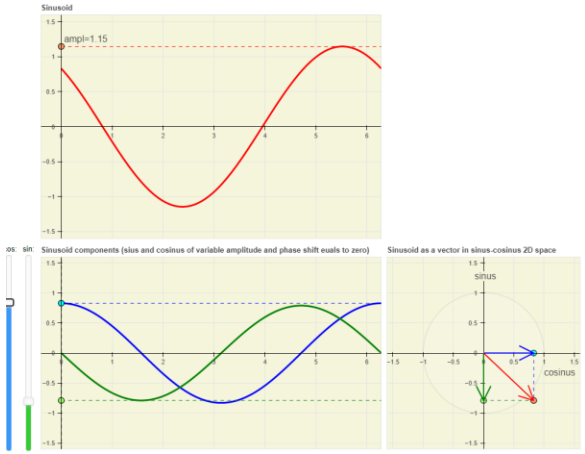
Wy know from previous slides that it is possible to transfer **two sinewaves** of the same frequency via **single channel** (signal) if sinewaves are **orthogonal.** It is the case for **pi/2 shifted** sinewaves (**sine** and **cosines** for example)



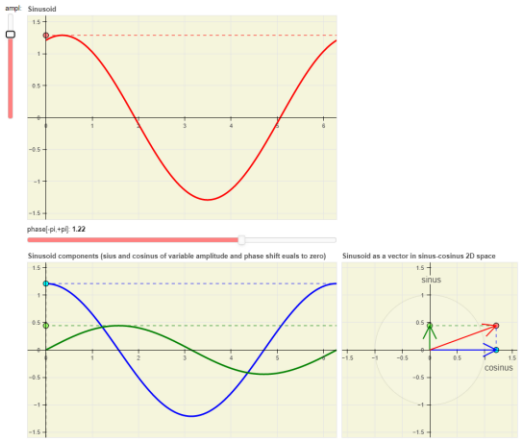


Such modulation of signal is called **quadrature modulation.** To see why please play with bellow interactive plots.
Do not hurry - take you time. 2_OrthogonalSinewavesAmplitudeModulation\1_SingleFrequency\2_TwoSineWaves\InteractivePresentations

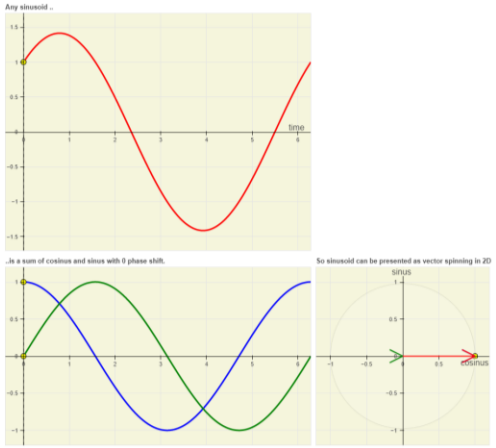
composition



decomposition



signal as a spinning vector



NOKIA

Quadrature modulation: idea

Use interactive plots from previous slide to answer following questions.

sinusoid_composition.html: find amplitude of sinus and cosinus to obtain sinusoid of amplitude 1.15 and phase 1.0
- ampl. sin: 0.97
- ampl. cos: 0.62

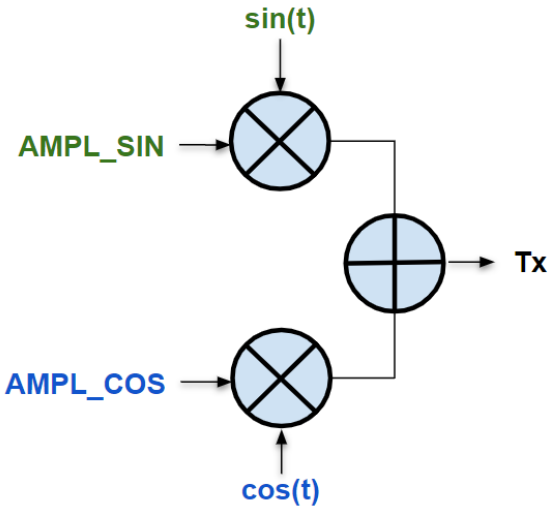
sinusoid_decomposition.html: find amplitude and phase of sinusoid of which is a sum of sinus of amplitude 0.5 and cosinus of amplidude 1.5
- sinusoid ampl.: 1.5811…
- sinusoid phase.: 1,249…

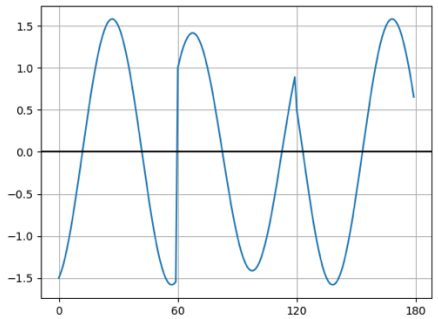
sinusoid_decomposition_spinning_vector.html: what are the vector angles at which sinus and cosinus have equal value
Pi/4, 3pi/4, 5pi/4, 7pi/4

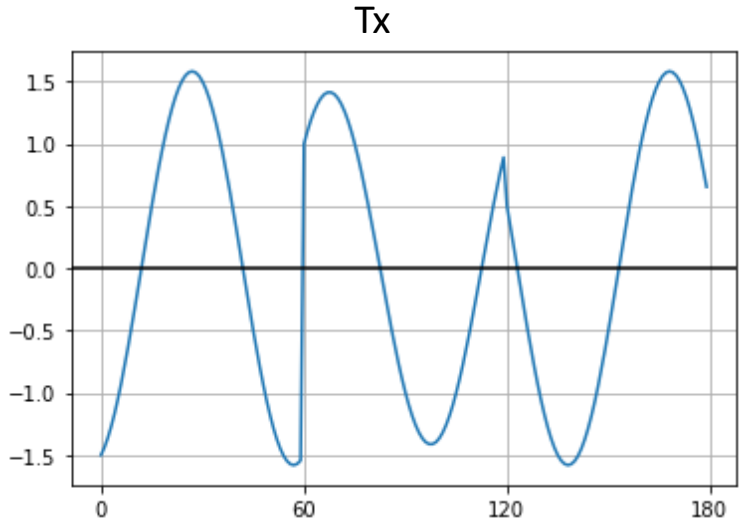
Quadrature modulation: **implementation**

Complete script to implement quadrature modulation. Result should look like on the waveform below.

```python
import numpy as np
import matplotlib.pyplot as plt
pi = np.pi

# PARAMETERS
TIME_VECTOR_SIZE = 60

AMPL_VECTOR_SIN = ( 0.5,  1.0,  -1.5)
AMPL_VECTOR_COS = (-1.5,  1.0,   0.5)

# CALCULATION
t = np.linspace( start: 0, 2*pi,TIME_VECTOR_SIZE, endpoint=False)

carrier_sin = np.sin(t)
carrier_cos = np.cos(t)

Tx = np.array([]) #empty time vector
for ampl_sin, ampl_cos in zip(AMPL_VECTOR_SIN, AMPL_VECTOR_COS):
    TxSinglePeriod = ampl_sin * carrier_sin  + ampl_cos * carrier_cos
    Tx = np.append(Tx, TxSinglePeriod)

# PRESENTATION
plt.plot(Tx)
plt.axhline(y=0,color='black')
plt.xticks((0,1*60,2*60,3*60))
plt.grid()

plt.show()

# SAVING
np.save( file: 'TxSignal',Tx)
```



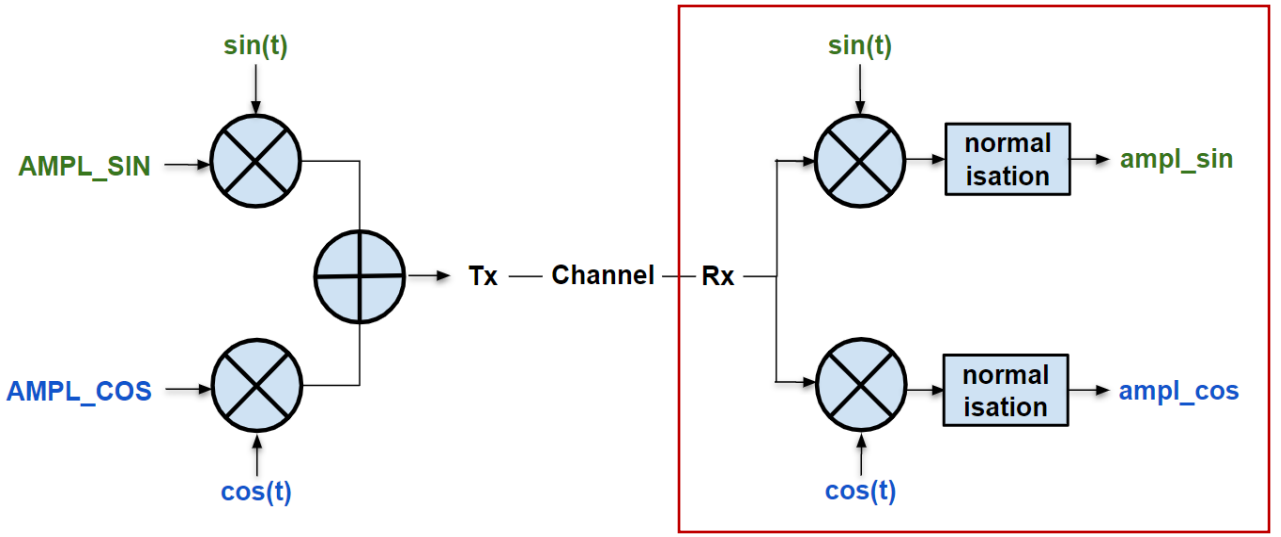




Quadrature **de**modulation: **implementation**

Complete script to demodulate signal from previous slide

```python
import ...
pi = np.pi

# parameters
PERIOD_VECTOR_SIZE = 60

# loading Rx vector from file and...
Rx = np.load('TxSignal.npy')

# .. ploting it
plt.plot(Rx)
plt.grid()
plt.show()

# spliting vector into time slots coresponding to single periods
# "-1" causes automatic evaluation array seccond dimension
RxPeriods = np.reshape(Rx, newshape: (-1,PERIOD_VECTOR_SIZE))

# lists for amplitudes
amplitudes_sin = []
amplitudes_cos = []

for RxPeriod in RxPeriods: # iterating periods
    t = np.linspace( start: 0, 2*pi,PERIOD_VECTOR_SIZE, endpoint=False)
    # sinus amplitude evaluation
    ref = np.sin(t)
    ampl = np.dot(RxPeriod,ref)/np.dot(ref,ref)
    amplitudes_sin.append(ampl)
    # cosinus amplitude evaluation
    ref = np.cos(t)
    ampl = np.dot(RxPeriod,ref)/np.dot(ref,ref)
    amplitudes_cos.append(ampl)

# presentation
amplitudes_sin = np.array(amplitudes_sin) # convert list to numpy 1D array
amplitudes_cos = np.array(amplitudes_cos) # ...
np.set_printoptions(precision=2)          # set numpy array print precision
print(f'amplitudes_sin = {amplitudes_sin}')
print(f'amplitudes_cos = {amplitudes_cos}')
```

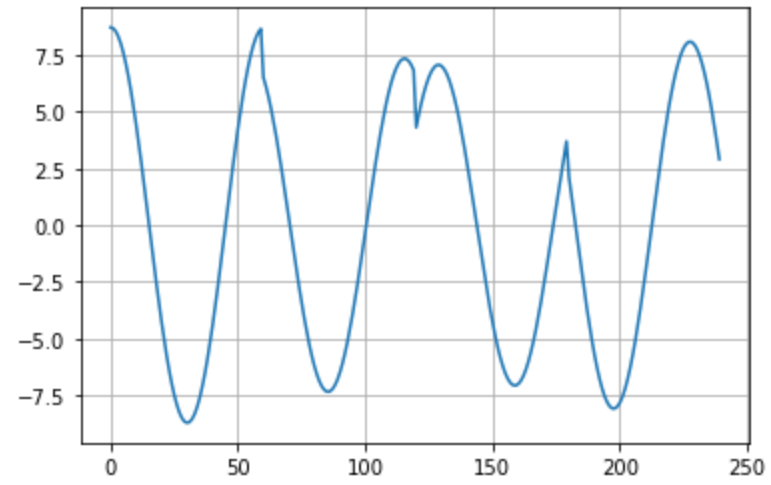



```
amplitudes_sin = [ 0.5  1.  -1.5]
amplitudes_cos = [-1.5  1.   0.5]
```

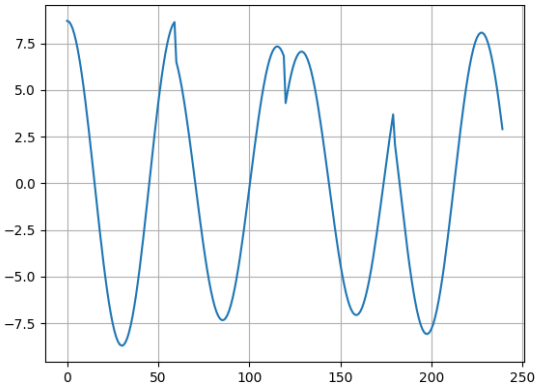

NOKIA

Quadrature **de**modulation: **implementation** cont.

Use previous script to decode signal from “Tx_Signal_Exercise.npy“ file.
Result should look as follow.




```
amplitudes_sin = [ 0.12 -3.4   5.6  -7.8 ]
amplitudes_cos = [8.7 6.5 4.3 2.1]
```



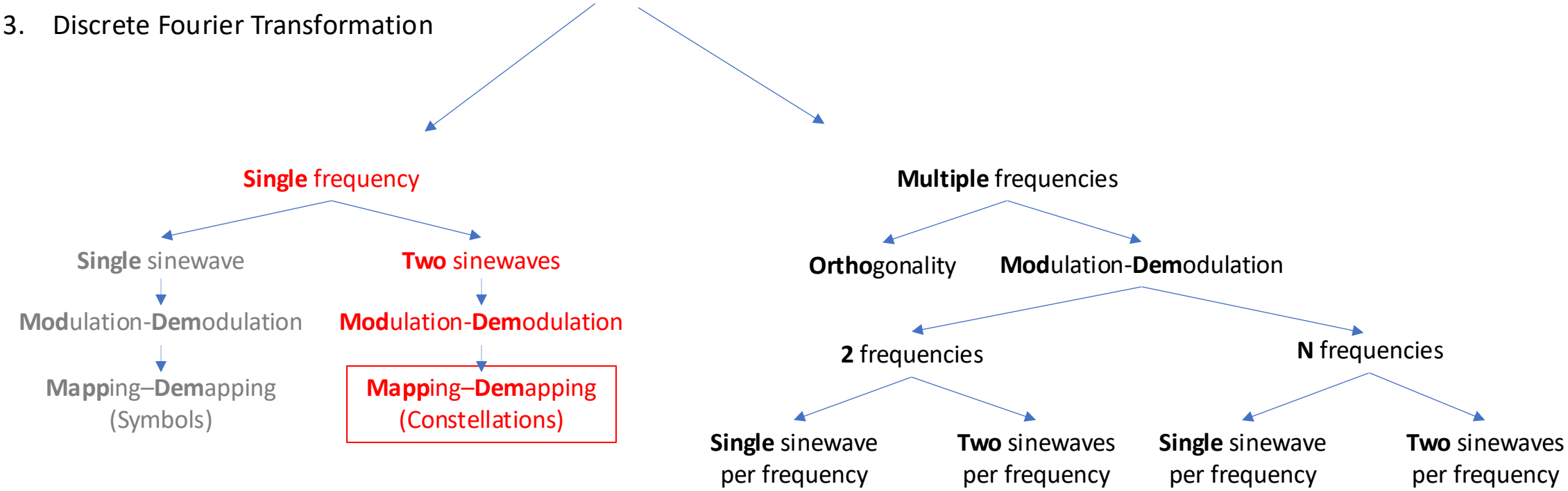

```
amplitudes_sin = [ 0.12 -3.4   5.6  -7.8 ]
amplitudes_cos = [8.7 6.5 4.3 2.1]
```

```python
import ...
pi = np.pi

# parameters
PERIOD_VECTOR_SIZE = 60

# loading Rx vector from file and...
Rx = np.load('TxSignal_exerc.npy')

# .. ploting it
plt.plot(Rx)
plt.grid()
plt.show()

# spliting vector into time slots coresponding to single periods
# "-1" causes automatic evaluation array seccond dimension
RxPeriods = np.reshape(Rx, newshape: (-1,PERIOD_VECTOR_SIZE))

# lists for amplitudes
amplitudes_sin = []
amplitudes_cos = []

for RxPeriod in RxPeriods: # iterating periods
    t = np.linspace( start: 0, 2*pi,PERIOD_VECTOR_SIZE, endpoint=False)
    # sinus amplitude evaluation
    ref = np.sin(t)
    ampl = np.dot(RxPeriod,ref)/np.dot(ref,ref)
    amplitudes_sin.append(ampl)
    # cosinus amplitude evaluation
    ref = np.cos(t)
    ampl = np.dot(RxPeriod,ref)/np.dot(ref,ref)
    amplitudes_cos.append(ampl)

# presentation
amplitudes_sin = np.array(amplitudes_sin) # convert list to numpy 1D array
amplitudes_cos = np.array(amplitudes_cos) # ...
np.set_printoptions(precision=2)          # set numpy array print precision
print(f'amplitudes_sin = {amplitudes_sin}')
print(f'amplitudes_cos = {amplitudes_cos}')
```


NOKIA

You are here

1. Sinewaves orthogonality
2. Orthogonal sinewaves amplitude modulation
3. Discrete Fourier Transformation



NOKIA

Maping-Demaping: Preparations : copying\renaming\verifying

Copy scripts `2_AmplitudeModulation\1_SingleFrequency\2_TwoSineWaves`:
- `1_Modulation.py`
- `2_Demodulation.py`

to: subfolder `3_MapingDemaping`

Rename them as below:
- `1.1_Modulation.py`
- `1.2_Demodulation.py`

Run `1.1_Modulation.py` and than `1.2_Demodulation.py`.
Check whether `1.2_Demodulation.py` prints amplitude vector from `1.1_Modulation.py`

```
AMPL_VECTOR_SIN = ( 0.12, -3.4, 5.6, -7.8)
AMPL_VECTOR_COS = ( 8.7,   6.5, 4.3,  2.1)
```

Maping-Demaping: Preparations : **merging**

Completing script on the right to **Merge** `1.1_Modulation.py` and 1.2_`Demodulation.py` into single `2.1_ModDemod.py`

Channel between modulation and demodulation should be ideal.

Amplitude vectors printed by the script (demodulation) should equal to amplitude vectors defined on the beginning of the script ().

```python
import ...
pi = np.pi

# PARAMETERS
TIME_VECTOR_SIZE = 60

AMPL_VECTOR_SIN = ( 0.12, -3.4, 5.6, -7.8)
AMPL_VECTOR_COS = ( 8.7,   6.5, 4.3,  2.1)

# CALCULATION
t = np.linspace( start: 0, 2*pi,TIME_VECTOR_SIZE, endpoint=False)

carrier_sin = ref_sin = np.sin(t)
carrier_cos = ref_cos = np.cos(t)

amplitudes_sin = list()
amplitudes_cos = list()

for ampl_sin, ampl_cos in zip(AMPL_VECTOR_SIN, AMPL_VECTOR_COS):

    # modulation
    Tx =  ampl_sin * carrier_sin + ampl_cos * carrier_cos

    # channel
    Rx = Tx # ideal

    # demodulation
    ampl = np.dot(Rx, ampl_sin)/TIME_VECTOR_SIZE*2
    amplitudes_sin.append(ampl)

    ampl = np.dot(Rx, ampl_cos)/TIME_VECTOR_SIZE*2
    amplitudes_cos.append(ampl)

# PRESENTATION
# presentation
amplitudes_sin = np.array(amplitudes_sin) # convert list to numpy 1D array
amplitudes_cos = np.array(amplitudes_cos) # ...
np.set_printoptions(precision=2)          # set numpy array print precision

print(f'amplitudes_sin = {amplitudes_sin}')
print(f'amplitudes_cos = {amplitudes_cos}')
```

real channel : errors

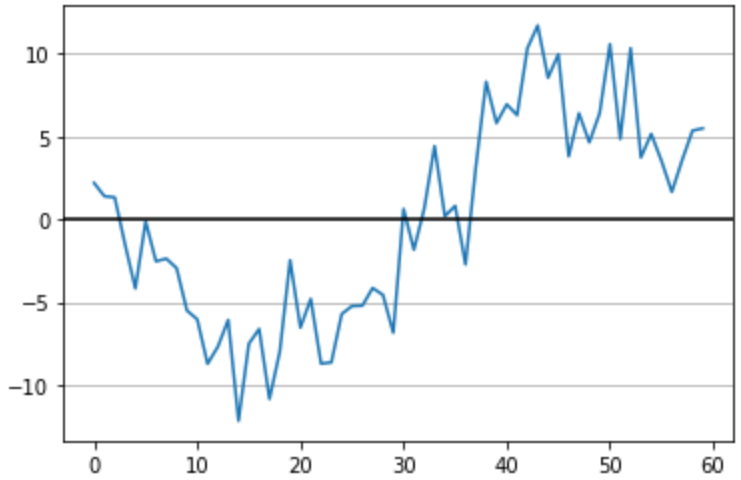
Copy `2.1_ModDemod.py` to `2.2_ModDemod_RealChannel.py`

1. **Add normal distribution noise** to channel (numpy.random.normal)

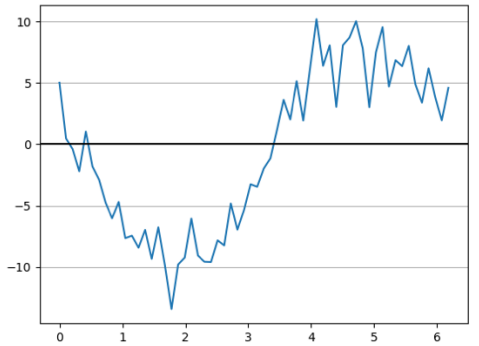
   Set distribution center location at 0, deviation to 2 and size accordingly to Tx length

   Plot Rx corresponding to last transmitted number (plot Rx after loop)

2. **Add error calculation and printing.**




```
amplitudes_sin = [ 0.19 -3.32  4.75 -7.81]
errors sin     : [-0.07 -0.08  0.85  0.01]

amplitudes_cos = [8.18 6.61 4.42 1.78]
errors cos     : [ 0.52 -0.11 -0.12  0.32]
```




```
amplitudes_sin = [ 0.62 -3.41  5.75 -8.57]
errors sin     :  [ 0.5  -0.01  0.15 -0.77]

amplitudes_cos = [8.08 6.94 4.55 2.16]
errors cos     :  [-0.62  0.44  0.25  0.06]
```

```python
import ...
from networkx.algorithms.bipartite.basic import color

pi = np.pi

# PARAMETERS
TIME_VECTOR_SIZE = 60
AMPL_VECTOR_SIN = ( 0.12, -3.4, 5.6, -7.8)
AMPL_VECTOR_COS = ( 8.7,   6.5, 4.3,  2.1)

# CALCULATION
t = np.linspace( start: 0, 2*pi,TIME_VECTOR_SIZE, endpoint=False)
carrier_sin = ref_sin = np.sin(t)
carrier_cos = ref_cos = np.cos(t)
amplitudes_sin = list()
amplitudes_cos = list()

for ampl_sin, ampl_cos in zip(AMPL_VECTOR_SIN, AMPL_VECTOR_COS):

    # modulation
    Tx = (ampl_sin*carrier_sin) + (ampl_cos*carrier_cos)
    # real channel
    Rx = Tx + np.random.normal( loc: 0,  scale: 2, len(Tx))
    # demodulation
    ampl = (np.dot(Rx,ref_sin)/TIME_VECTOR_SIZE)*2
    amplitudes_sin.append(ampl)

    ampl = (np.dot(Rx,ref_cos)/TIME_VECTOR_SIZE)*2
    amplitudes_cos.append(ampl)
# PRESENTATION
# PRESENTATION
# Rx plot
plt.plot( *args: t, Rx)
plt.grid(axis='y')
plt.axhline(y=0,color='black')
plt.show()
# amplitudes
amplitudes_sin = np.array(amplitudes_sin) # convert list to numpy 1D array
amplitudes_cos = np.array(amplitudes_cos) # ...
np.set_printoptions(precision=2)          # set numpy array print precision
print(f'amplitudes_sin = {amplitudes_sin}')
errors_sin = amplitudes_sin - AMPL_VECTOR_SIN
print(f'errors sin     :  {errors_sin}')
print()
```


NOKIA

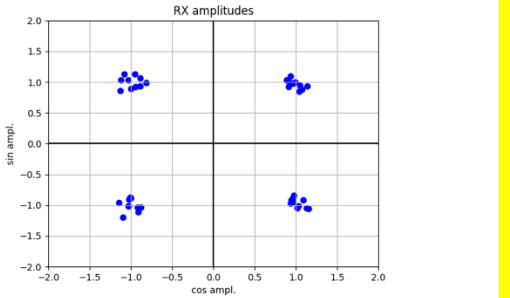
Constelations

1. Copy `2.2_ModDemod_RealChannel.py` to `2.3_ModDemod_RealChannel_Constelations.py`
2. Add parameter NOISE_DEVIATION to define “scale” argument of the “normal” function.
3. Set amplitudes to transmit as below
   AMPL_VECTOR_SIN = (1,-1, 1,-1)
   AMPL_VECTOR_COS = (1, 1,-1,-1)
4. Add parameter TANSMISION_NR=10 and add loop to repeat transmission TANSMISION_NR times

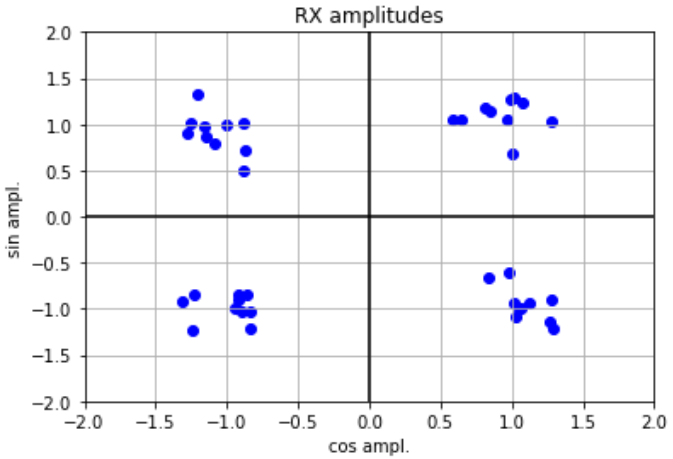
```
for i in range(TRANSMISON_NR):
    for ampl_sin, ampl_cos in zip(AMPL_VECTOR_SIN, AMPL_VECTOR_COS):
```

5. Use plt.scatter function to make plot as below. Do not forget about figure and axes titles, fixed x and y axis ranges, and horizontal and vertical lines at x and y equal 0 (black cross in the middle)
6. Generate missing figures (see below)

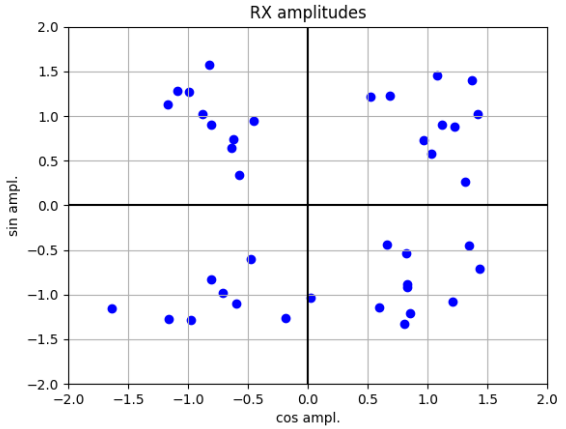
dev=**0.5**



dev=**1**



dev=**2**

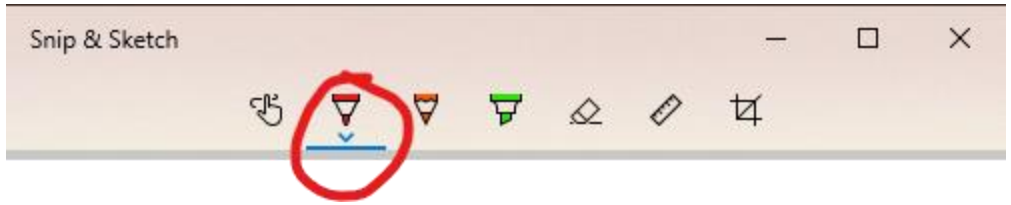



```
import ...
pi = np.pi
# PARAMETERS
TIME_VECTOR_SIZE
AMPL_VECTOR_SIN
AMPL_VECTOR_COS
# CALCULATION
t = np.linspace(
carrier_sin = re
carrier_cos = re
amplitudes_sin =
amplitudes_cos =
NOISE_DEVIATION
TRANSMISSION_NR
for i in range(T
    for ampl_sin
        # modula
        Tx = (am
        # real c
        Rx = Tx
        # demodu
        ampl = (
        amplitud
        ampl = (
        amplitud
# PRESENTATION
# Rx plot
plt.scatter(ampl
plt.axhline(y=0,
plt.axvline(x=0,
plt.title("RX am
plt.xlabel("cos
plt.ylabel("sin
plt.xlim( *args:
plt.ylim( *args:
plt.grid()
plt.show()
```

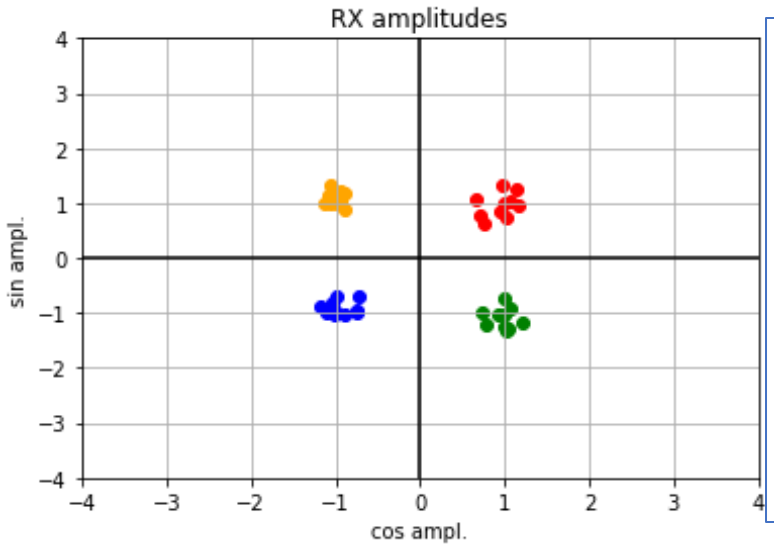

NOKIA

Constelations : **colors**

1. Copy *2.3_ModDemod_RealChannel_Constelations.py* to *2.4_ModDemod_RealChannel_Constelations_Colors.py*
2. Modify script to plot each amplitude with unique color (Hint: `plt.scatter(ampl_sin, ampl_cos, color=('red','orange','green','blue')[j%4])`)
3. Generate missing figures (see below) and count errors expecteded during Rx decoding. (use plots for counting), mark and caunt error symbols with windows screenshot tool (“shift-windows-s”)

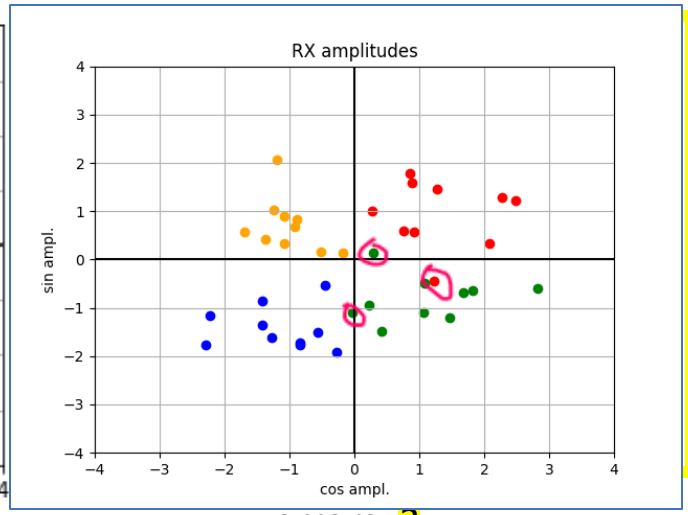


dev=**1**



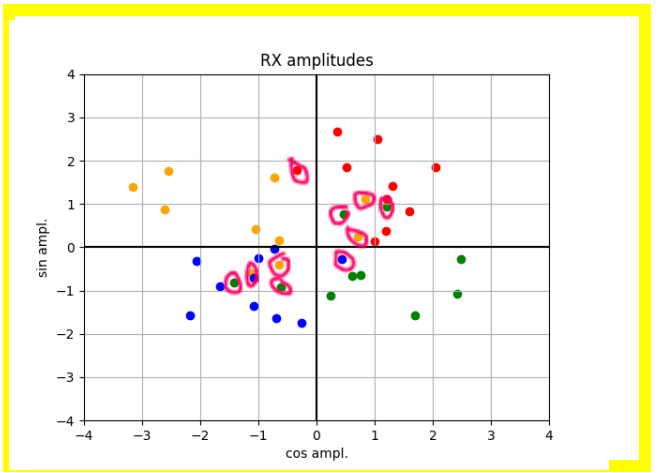
errors=0

dev=**4**



errors=3

dev=**6**



Errors = 10

```python
import ...
pi = np.pi
# PARAMETERS
TIME_VECTOR_SIZE = 60
AMPL_VECTOR_SIN = (1,-1,1,-1)
AMPL_VECTOR_COS = (1,1,-1,-1)
# CALCULATION
t = np.linspace( start: 0, 2*pi,TIME_VE
carrier_sin = ref_sin = np.sin(t)
carrier_cos = ref_cos = np.cos(t)
amplitudes_sin = list()
amplitudes_cos = list()
NOISE_DEVIATION = 6
TRANSMISSION_NR = 10
for i in range(TRANSMISSION_NR):
    for ampl_sin, ampl_cos in zip(AMP
        # modulation
        Tx = (ampl_sin*carrier_sin) +
        # real channel
        Rx = Tx + np.random.normal(0,
        # demodulation
        ampl1 = (np.dot(Rx,ref_sin)/T
        amplitudes_sin.append(ampl1)
        ampl2 = (np.dot(Rx,ref_cos)/T
        amplitudes_cos.append(ampl2)
# PRESENTATION
# Rx plot
for j in range(len(amplitudes_sin)):
    plt.scatter(amplitudes_sin[j], am
plt.axhline(y=0, color='black')
plt.axvline(x=0, color='black')
plt.title("RX amplitudes")
plt.xlabel("cos ampl.")
plt.ylabel("sin ampl.")
plt.xlim( *args: -4,4)
plt.ylim( *args: -4,4)
plt.grid()
```


NOKIA

Constelation : **transmission and reception**

Complete below script to make given number of **transmissions\receptions** by using given number of **symbols** in the presence of given **noise** distribution. Use amplitude maping\demaping functions from `mapper_lib.py`.

```python
import ...

# PARAMETERS
TIME_VECTOR_SIZE = 60
TRANSMISIONS_NR = 10

NOISE_DEVIATION = 2.0
SYMBOL_NR = 4

t = np.linspace( start: 0, 2*np.pi,TIME_VECTOR_SIZE, endpoint=False)
carrier_sin = ref_sin = np.sin(t)
carrier_cos = ref_cos = np.cos(t)

# TRANSMISION-RECEPTION
symbols_tx_sin = np.random.randint(0,SYMBOL_NR,TRANSMISIONS_NR)
symbols_tx_cos = np.random.randint(0,SYMBOL_NR,TRANSMISIONS_NR)

symbols_rx_sin = list()
symbols_rx_cos = list()

for symbol_sin, symbol_cos in zip(symbols_tx_sin,symbols_tx_cos):
    # modulation
    ampl_sin = symbol_to_ampl(SYMBOL_NR, symbol_sin)
    ampl_cos = symbol_to_ampl(SYMBOL_NR, symbol_cos)
    Tx = ampl_sin*carrier_sin+ampl_cos*carrier_cos
    # real channel
    Rx = Tx + np.random.normal(loc=0, scale=NOISE_DEVIATION, size=len(Tx))
    # demodulation
    ampl_sin = np.dot(ref_sin, Rx)/TIME_VECTOR_SIZE*2
    ampl_cos = np.dot(ref_cos, Rx)/TIME_VECTOR_SIZE*2

    symbol_sin = ampl_to_symbol(SYMBOL_NR, ampl_sin)
    symbol_cos = ampl_to_symbol(SYMBOL_NR, ampl_cos)

    symbols_rx_sin.append(symbol_sin)
    symbols_rx_cos.append(symbol_cos)
```

```python
# PRESENTATION
symbols_rx_sin = np.array(symbols_rx_sin) # list to numpy array
symbols_rx_cos = np.array(symbols_rx_cos)

print('SIN')
print('symbols_rx:', symbols_tx_sin)
errors = symbols_rx_sin != symbols_tx_sin
print('match:', ~errors)
print('errors nr:', sum(errors))

print('COS')
print('symbols_rx:', symbols_tx_cos)
errors = symbols_rx_cos != symbols_tx_cos
print('match:', ~errors)
print('errors nr:', sum(errors))
```

Typical result

```
SIN
symbols_rx: [1 3 1 2 2 1 3 3 3 2]
match: [False  True  True  True False  True  True  True False  True]
errors nr: 3
COS
symbols_rx: [2 2 2 3 2 1 0 0 2 2]
match: [False  True  True  True  True  True  True  True False  True]
errors nr: 2
```


NOKIA

Constelation: **error number v.s. noise deviation**

Save previous script as `7.2_ModDemodSymbols_ErrorRate_vs_Noise_Print.py`.
Modify it to evaluate transmission error for given range of noise deviation (see exemplary result below).
Set TRANSMISIONS_NR to 1000
Use the script to add complete results below.

```
SYMBOL_NR: 2
DEV:  SIN,  COS
0.00 : 0, 0
0.22 : 0, 0
0.44 : 0, 0
0.67 : 0, 0
0.89 : 0, 0
1.11 : 0, 0
1.33 : 0, 0
1.56 : 0, 1
1.78 : 2, 1
2.00 : 6, 0
```

```
SYMBOL_NR: 4

DEV: SIN, COS
0.00: 0, 0
0.22: 0, 0
0.44: 0, 0
0.67: 4, 2
0.89: 32, 29
1.11: 72, 66
1.33: 107, 130
1.56: 173, 169
1.78: 230, 204
2.00: 282, 267
```

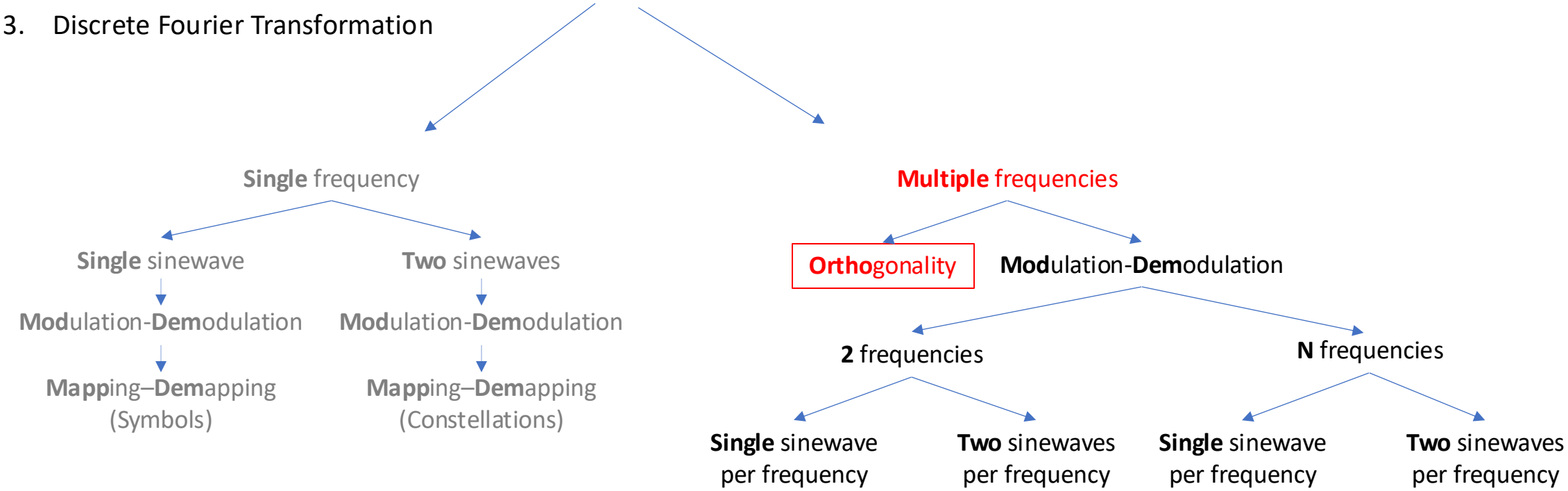
```
SYMBOL_NR: 8
DEV:  SIN,  COS
0.00 : 0, 0
0.22 : 1, 0
0.44 : 73, 76
0.67 : 235, 205
0.89 : 340, 339
1.11 : 415, 404
1.33 : 487, 510
1.56 : 531, 536
1.78 : 575, 585
2.00 : 604, 614
```

```python
import ...
# PARAMETERS
TIME_VECTOR_SIZE = 60
TRANSMISIONS_NR = 1000
NOISE_DEVIATION = 2.0
SYMBOL_NR = 4
t = np.linspace( start: 0, 2 * np.pi, TIME_VECTOR_SIZE, endpoint=False)
carrier_sin = ref_sin = np.sin(t)
carrier_cos = ref_cos = np.cos(t)
# TRANSMISION-RECEPTION
symbols_tx_sin = np.random.randint(0, SYMBOL_NR, TRANSMISIONS_NR)
symbols_tx_cos = np.random.randint(0, SYMBOL_NR, TRANSMISIONS_NR)
noise_dev_list = list()
for noise_dev in np.linspace(0, NOISE_DEVIATION, 10):
    symbols_rx_sin = list()
    symbols_rx_cos = list()
    for symbol_sin, symbol_cos in zip(symbols_tx_sin, symbols_tx_cos
        # modulation
        ampl_sin = symbol_to_ampl(SYMBOL_NR, symbol_sin)
        ampl_cos = symbol_to_ampl(SYMBOL_NR, symbol_cos)
        Tx = ampl_sin * carrier_sin + ampl_cos * carrier_cos
        # real channel
        Rx = Tx + np.random.normal(loc=0, scale=noise_dev, size=len(
        # demodulation
        ampl_sin = np.dot(ref_sin, Rx) / TIME_VECTOR_SIZE * 2
        ampl_cos = np.dot(ref_cos, Rx) / TIME_VECTOR_SIZE * 2
        symbol_sin = ampl_to_symbol(SYMBOL_NR, ampl_sin)
        symbol_cos = ampl_to_symbol(SYMBOL_NR, ampl_cos)
        symbols_rx_sin.append(symbol_sin)
        symbols_rx_cos.append(symbol_cos)
    symbols_rx_sin = np.array(symbols_rx_sin)
    symbols_rx_cos = np.array(symbols_rx_cos)
    errors_sin = sum(symbols_rx_sin != symbols_tx_sin)
    errors_cos = sum(symbols_rx_cos != symbols_tx_cos)
    noise_dev_list.append(((noise_dev, errors_sin, errors_cos)))
# PRESENTATION

print(f'SYMBOL_NR: {SYMBOL_NR}')

print(f'DEV:  SIN,  COS')
for noise_dev, errors_sin, errors_cos in noise_dev_list:
    print(f'{noise_dev:.2f} : {errors_sin}, {errors_cos}')
```

Assuming noise dev = 1.11 and acceptable error number = 300 what is an optimal symbol nr ?
: 4

You are here

1. Sinewaves orthogonality
2. Orthogonal sinewaves amplitude modulation
3. Discrete Fourier Transformation



NOKIA

Orthogonal frequencies: Introduction

Summary
In the last exercise we made data transmission we used two orthogonal sinewaves of the same frequency.
Amount of transmitted data was proportional to number of time slots (two number per slot, one for cosines second for sinus).
One time slot corresponded to single period of sinewave.

Observation
Above transition scheme based on observation that if two same frequency sinewaves are shifted of pi/2 they are orthogonal and in practice they do not interfere when placed in a single signal.

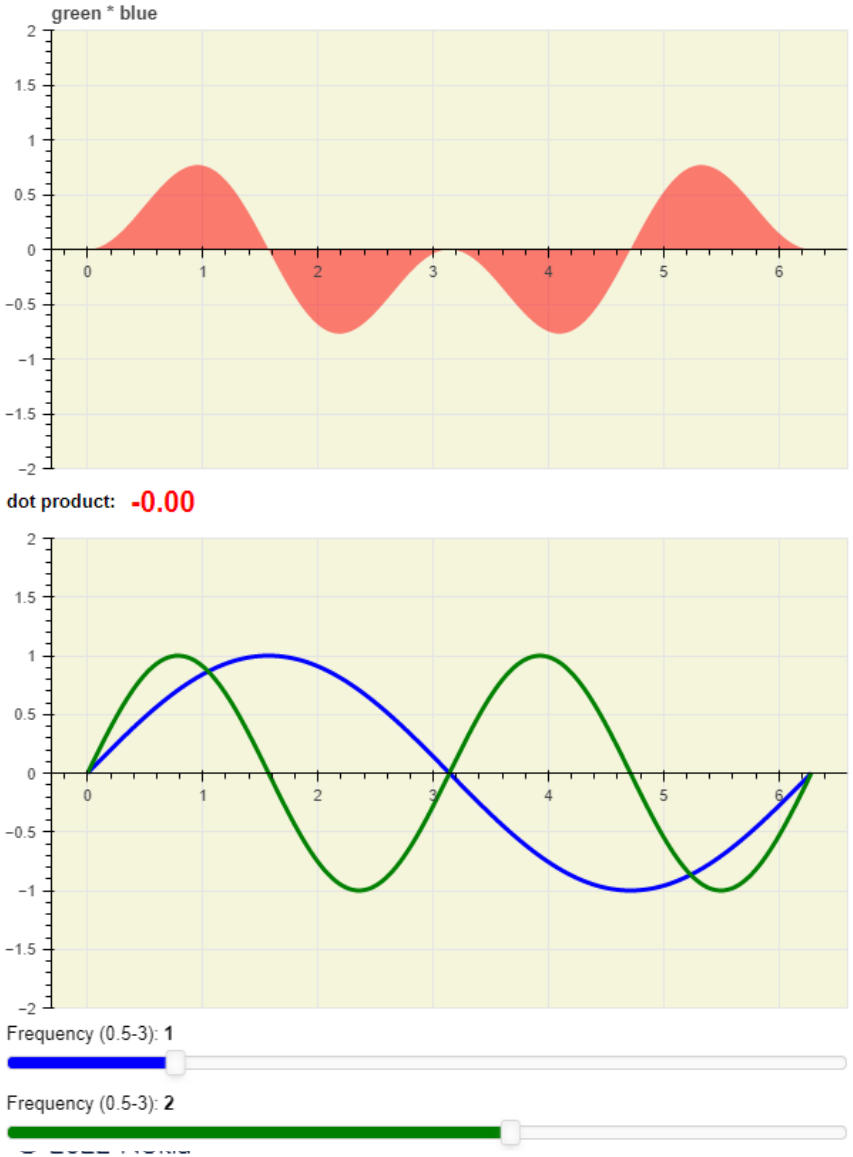
Motivation
To send more data via the same signal (channel)

Hypothesis
It I possible to transfer two or more sinewaves of different frequency via single signal in the way that hey do not interfere.

Experiments
Let's examinate orthogonality (dot product) of sinewaves of different frequencies

Frequencies orthogonality : **sinus**

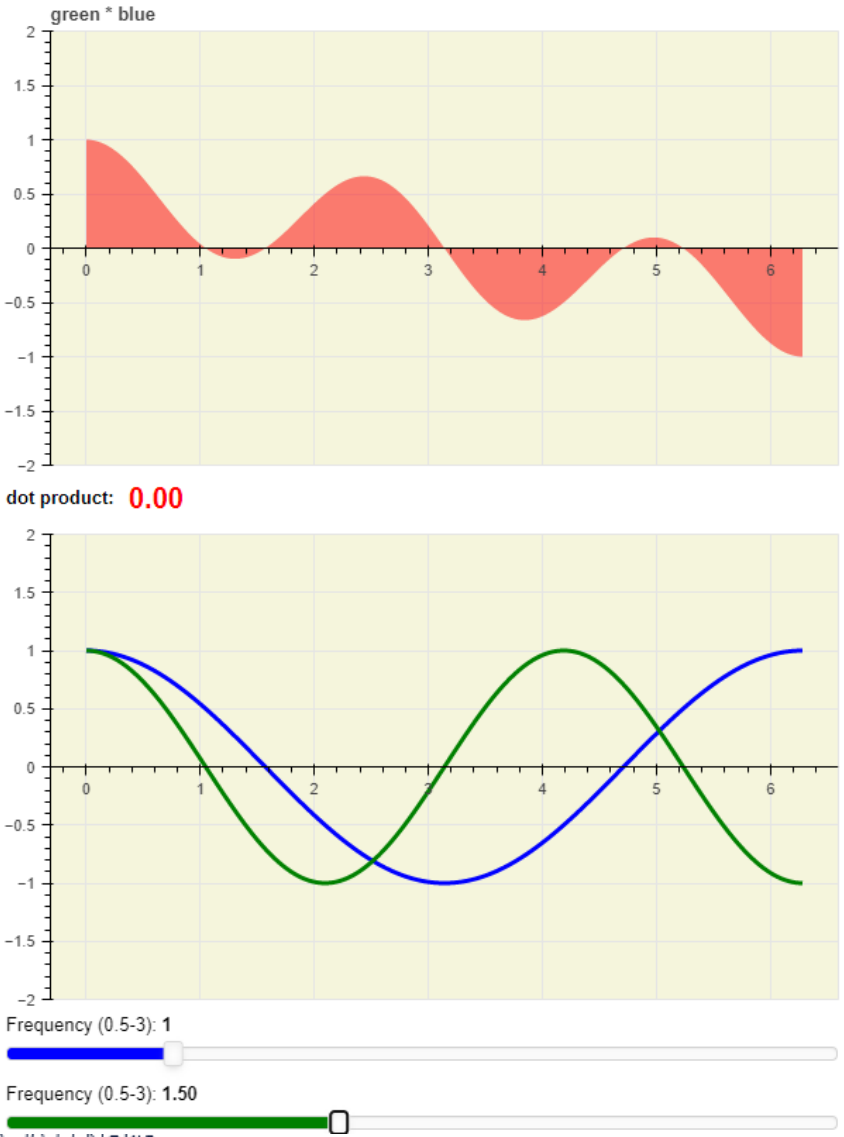
Use interactive presentation to find frequencies orthogonal to sin(1*t). (Fill the table)



Blue frequency = 1.0
Green frequency at which dot_product(blue, green)==0
0.5
1.5
2.0
2.5
3.0

Frequencies orthogonality : **co**sinus

Do the same for cosines



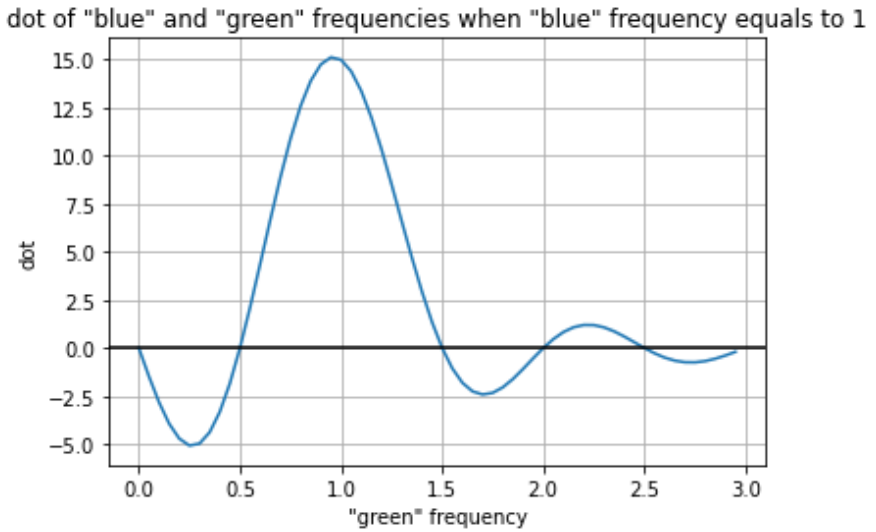
Blue frequency = 1.0
Green frequency at which dot_product(blue, green)==0
0.5
1.5
2.0
2.5
3.0

Orthogonal frequencies: estimation for **sine waves, scripting**

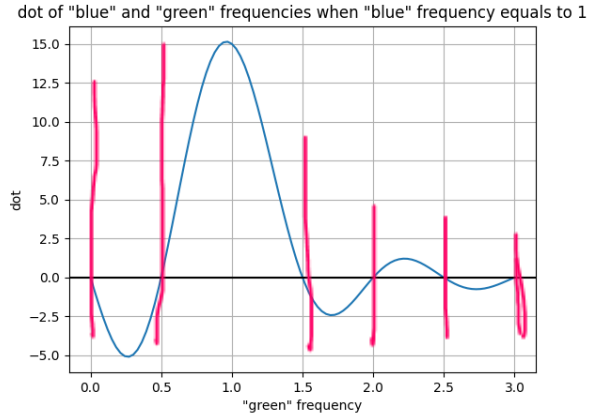
1. Complete script to plot relationship between frequency ratio of two sinusoids and dot product.

```
import ...

t = np.linspace( start: 0, 2*np.pi, num: 30, endpoint=False)
sin_blue  = np.sin(t)

dot_l = list()
freq_l = np.linspace( start: 0,  stop: 3,  num: 100)
for freq in freq_l:
    sin_green = np.sin(freq*t)
    dot_l.append(np.dot(sin_green, sin_blue))

plt.plot( *args: freq_l, dot_l)
plt.axhline(y=0,color='black')
plt.title('dot of "blue" and "green" frequencies when "blue" frequency equals to 1')
plt.xlabel('"green" frequency')
plt.ylabel('dot')
plt.grid()
plt.show()
```

Refence plot



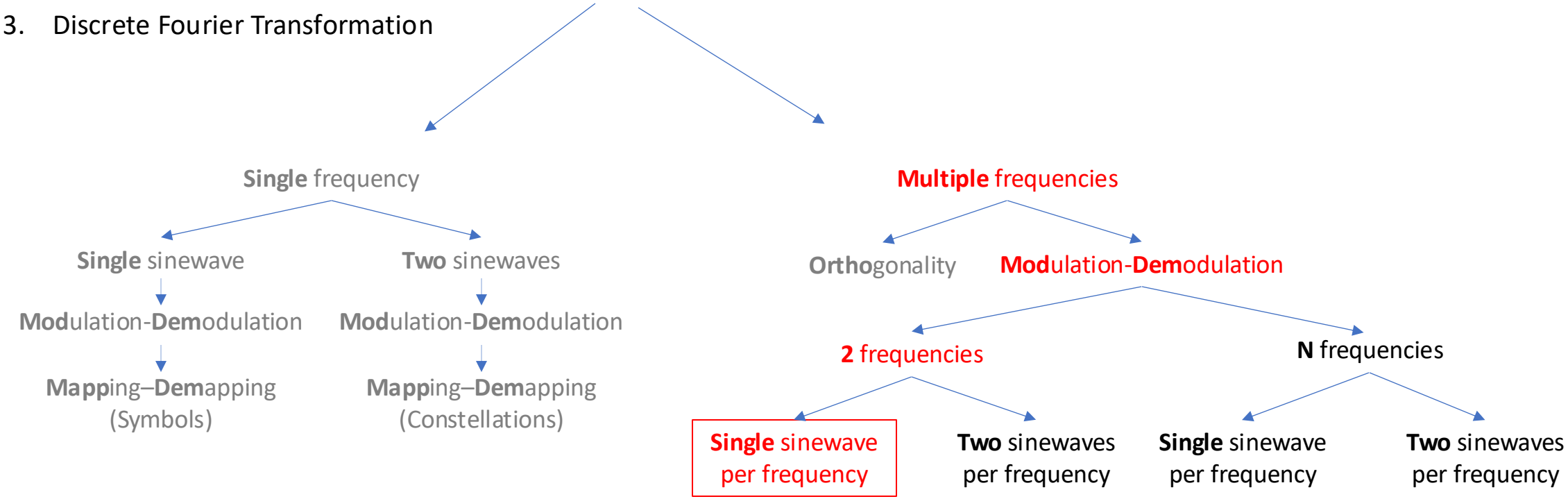
2. Mark orthogonal frequencies on x axis of the plot
Do it manually in PowerPoint by drawing red vertical lines on the plot



NOKIA

You are here

1. Sinewaves orthogonality
2. Orthogonal sinewaves amplitude modulation
3. Discrete Fourier Transformation



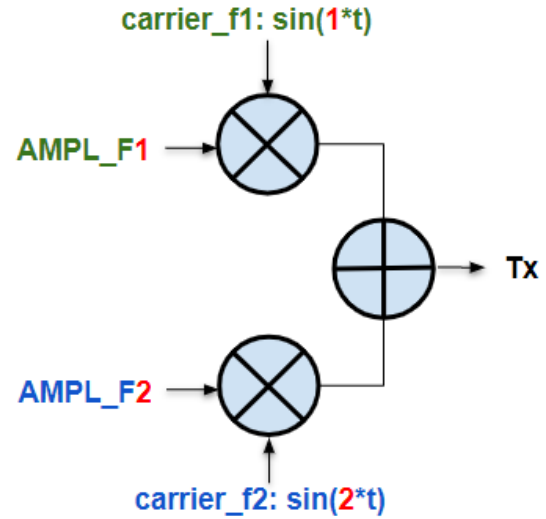
NOKIA

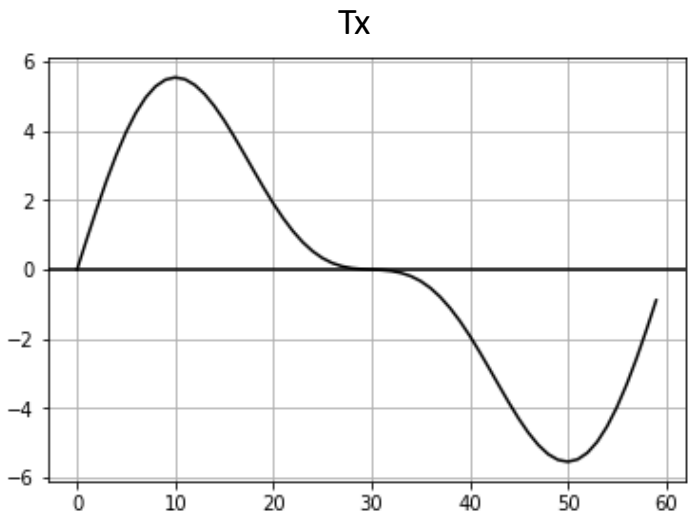
Two frequencies : Single sinewave per frequency : modulation

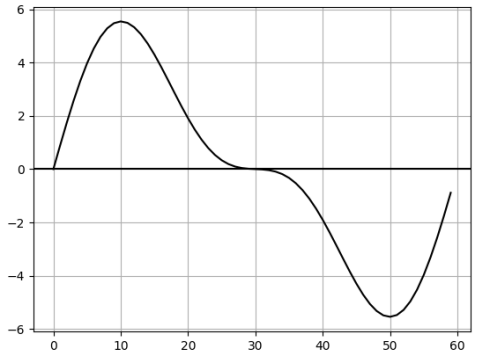
Complete script to obtain Tx signal composed of two modulated sinusoids of orthogonal frequencies

```python
import ...
pi = np.pi

# PARAMETERS
TIME_VECTOR_SIZE = 60

AMPL_F1 = 4.3
AMPL_F2 = 2.1

# CALCULATION
t = np.linspace( start: 0, 2*pi,TIME_VECTOR_SIZE, endpoint=False)

carrier_f1 = np.sin(1*t)
carrier_f2 = np.sin(2*t)

tx_f1 = AMPL_F1 * carrier_f1
tx_f2 = AMPL_F2 * carrier_f2

Tx = tx_f1 + tx_f2

# PRESENTATION
plt.plot( *args: Tx, color='black')
plt.axhline(y=0,color='black')
plt.grid()
plt.show()

# SAVING
np.save( file: 'TxSignal',Tx)
```

nt=False)





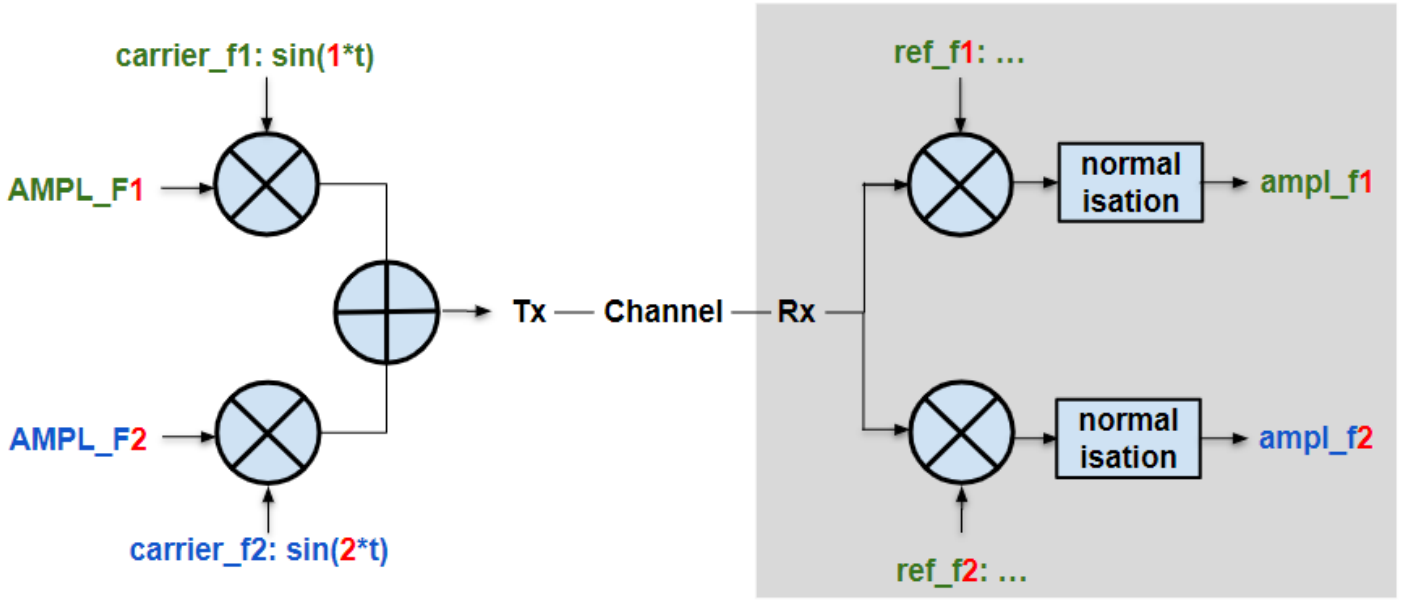


NOKIA

Two frequencies : Single sinewave per frequency : **Demodulation**

Complete script to decode amplitudes of sinusoids from Rx signal

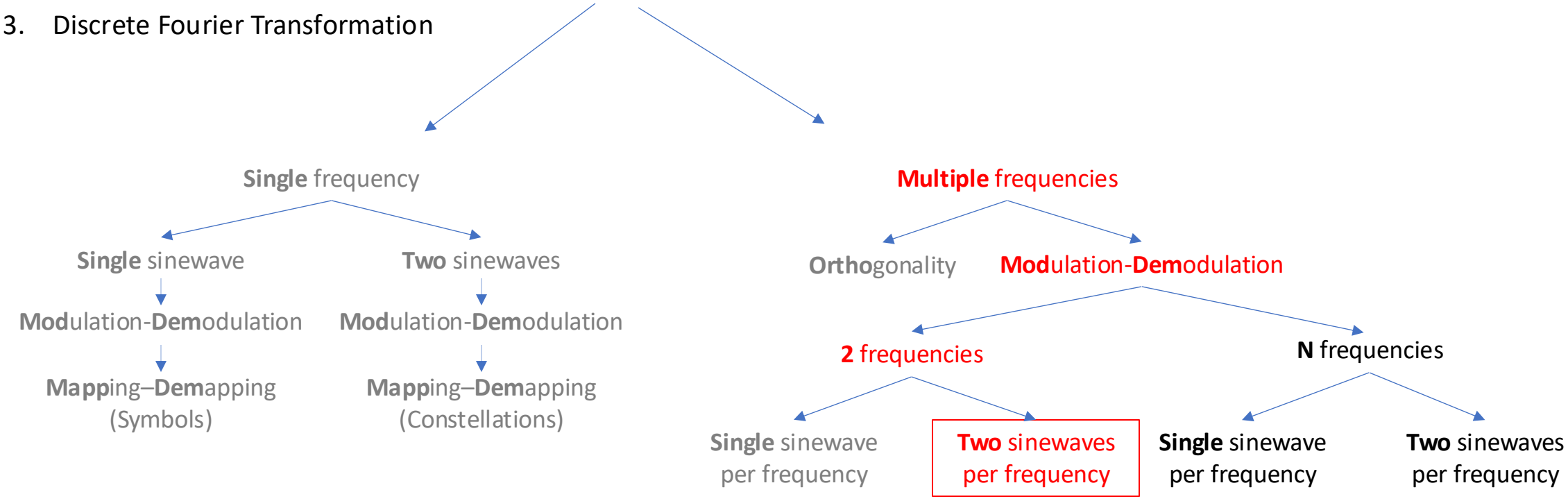
```python
import ...
pi = np.pi

# parameters
PERIOD_VECTOR_SIZE = 60

# loading Rx vector from file and..
Rx = np.load('TxSignal.npy')

# .. ploting it
plt.plot(Rx)
plt.axhline(y=0,color='black')
plt.grid()
plt.show()

# callculation
t = np.linspace( start: 0, 2*pi,PERIOD_VECTOR_SIZE, endpoint=False)

ref_f1 = np.sin(1*t)
ref_f2 = np.sin(2*t)

dot_f1 = np.dot(Rx,ref_f1)
dot_f2 = np.dot(Rx,ref_f2)

ampl_f1 = dot_f1/np.dot(ref_f1,ref_f1)
ampl_f2 = dot_f2/np.dot(ref_f2,ref_f2)

# presentation
print(f'ampl_f1 = {ampl_f1:0.2f}, ampl_f2 = {ampl_f2:0.2f}')
```




```
ampl_f1 = 4.30, ampl_f2 = 2.10
```

You are here

1. Sinewaves orthogonality
2. Orthogonal sinewaves amplitude modulation
3. Discrete Fourier Transformation

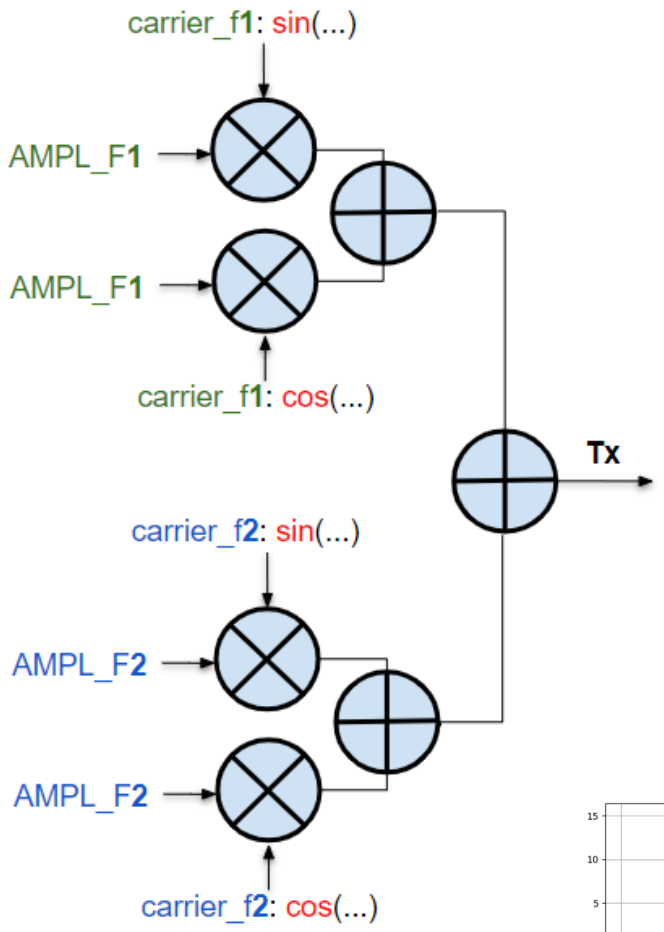


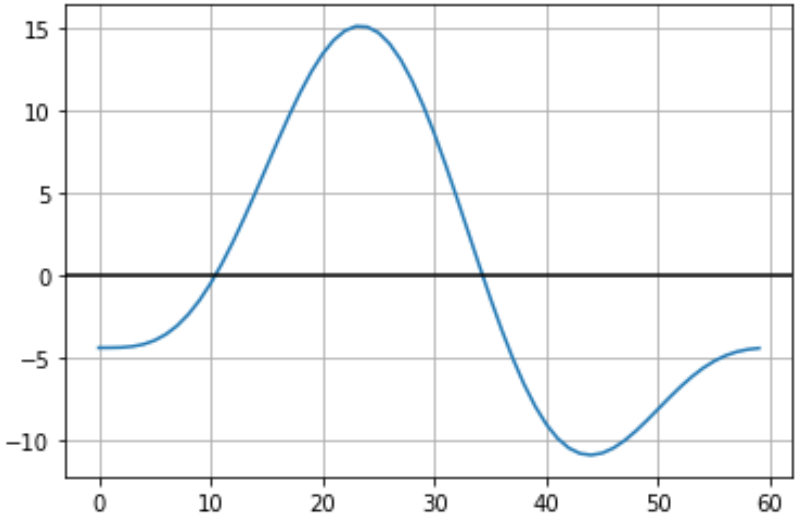
NOKIA

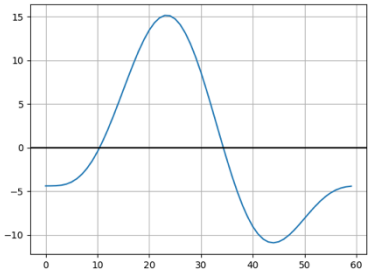
Two frequencies : **Two sinewaves** per frequency : **Modulation**

Complete script to make quadrature modulation of two orthogonal frequencies, i.e. 1 and 2

```python
import ...
pi = np.pi

# PARAMETERS
TIME_VECTOR_SIZE = 60

AMPL_F1_SIN =  8.7
AMPL_F1_COS = -6.5

AMPL_F2_SIN = -4.3
AMPL_F2_COS =  2.1

# CALCULATION
t = np.linspace( start: 0, 2*pi,TIME_VECTOR_SIZE, endpoint=False)

carrier_f1_sin = np.sin(1*t)
carrier_f1_cos = np.cos(1*t)

carrier_f2_sin = np.sin(2*t)
carrier_f2_cos = np.cos(2*t)

tx_f1_sin = AMPL_F1_SIN*carrier_f1_sin
tx_f1_cos = AMPL_F1_COS*carrier_f1_cos

tx_f2_sin = AMPL_F2_SIN*carrier_f2_sin
tx_f2_cos = AMPL_F2_COS*carrier_f2_cos

Tx = tx_f1_sin+tx_f1_cos+tx_f2_sin+tx_f2_cos

# PRESENTATION
plt.plot(Tx)
plt.axhline(y=0,color='black')
plt.grid()
plt.show()

# SAVING
np.save( file: 'TxSignal',Tx)
```







NOKIA

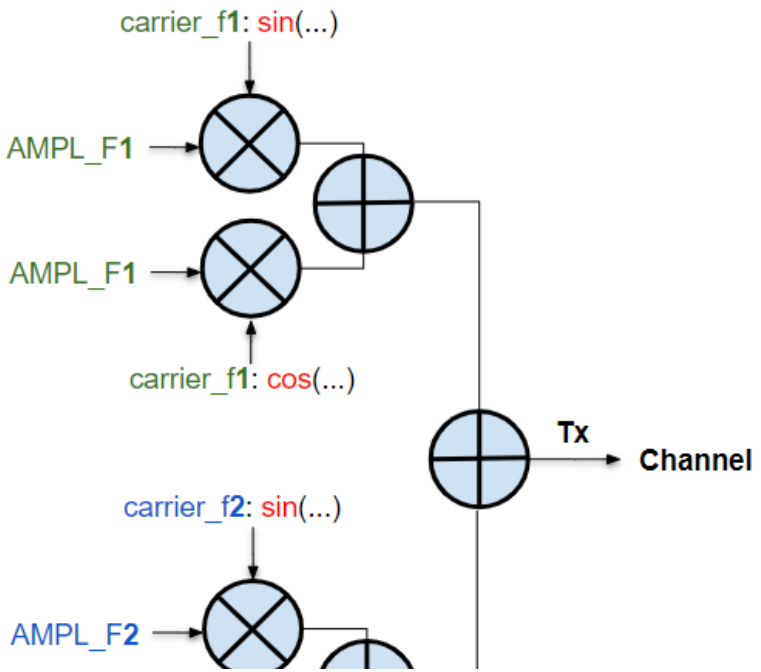
Two frequencies : **Two sinewaves** per frequency : **De**modulation

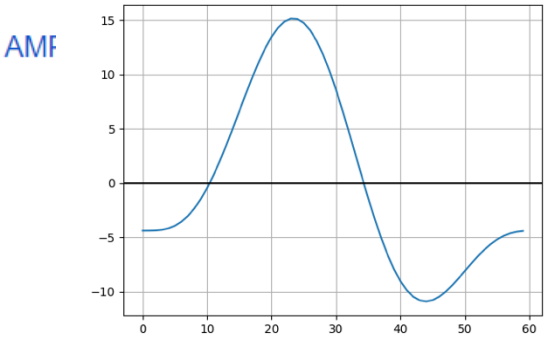
Complete script to demodulate signal which consists of two orthogonal quadrature modulated frequencies

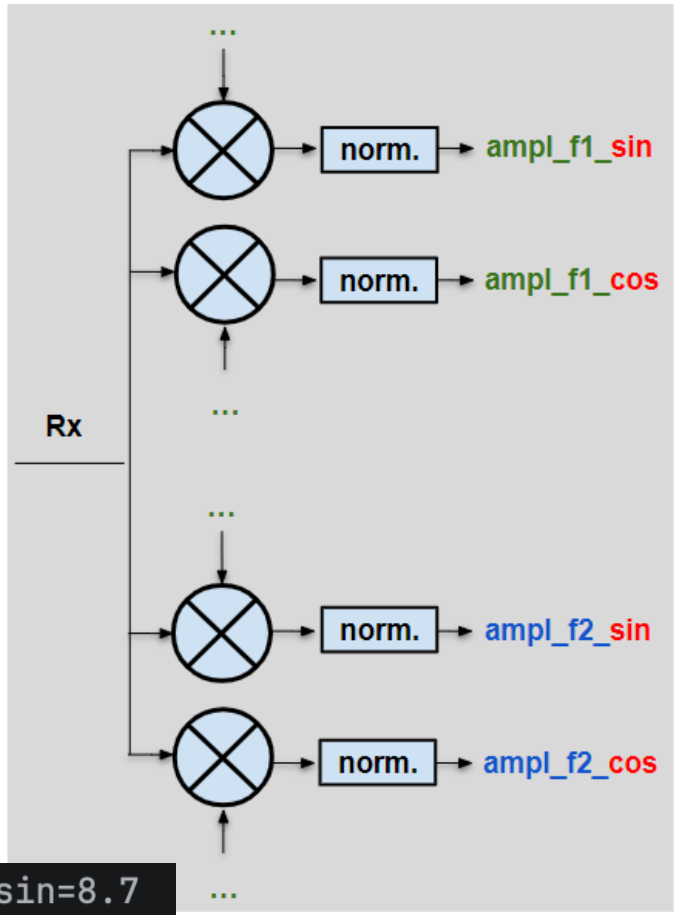
```python
import ...
pi = np.pi

# parameters
PERIOD_VECTOR_SIZE = 60

# loading Rx vector from file and..
Rx = np.load('TxSignal.npy')

# .. ploting it
plt.plot(Rx)
plt.axhline(y=0,color='black')
plt.grid()
plt.show()

# CALLCULATION
t = np.linspace( start: 0, 2*pi,PERIOD_VECTOR_SIZE, endpoint=False)
# references
ref_f1_sin = np.sin(1*t)
ref_f1_cos = np.cos(1*t)
ref_f2_sin = np.sin(2*t)
ref_f2_cos = np.cos(2*t)
# dot products
dot_f1_sin = np.dot(Rx, ref_f1_sin)
dot_f1_cos = np.dot(Rx, ref_f1_cos)
dot_f2_sin = np.dot(Rx, ref_f2_sin)
dot_f2_cos = np.dot(Rx, ref_f2_cos)
# amplitudes
amp_f1_sin = dot_f1_sin/np.dot(ref_f1_sin,ref_f1_sin)
amp_f1_cos = dot_f1_cos/np.dot(ref_f1_cos,ref_f1_cos)
amp_f2_sin = dot_f2_sin/np.dot(ref_f2_sin,ref_f2_sin)
amp_f2_cos = dot_f2_cos/np.dot(ref_f2_cos,ref_f2_cos)
# PRESENTATION
print(f'amp_f1_sin={amp_f1_sin:0.1f}')
print(f'amp_f1_cos={amp_f1_cos:0.1f}')
print(f'amp_f2_sin={amp_f2_sin:0.1f}')
print(f'amp_f2_cos={amp_f2_cos:0.1f}')
```

False)

print(f'amp_f1_sin={amp_f1_sin:0.1f}')



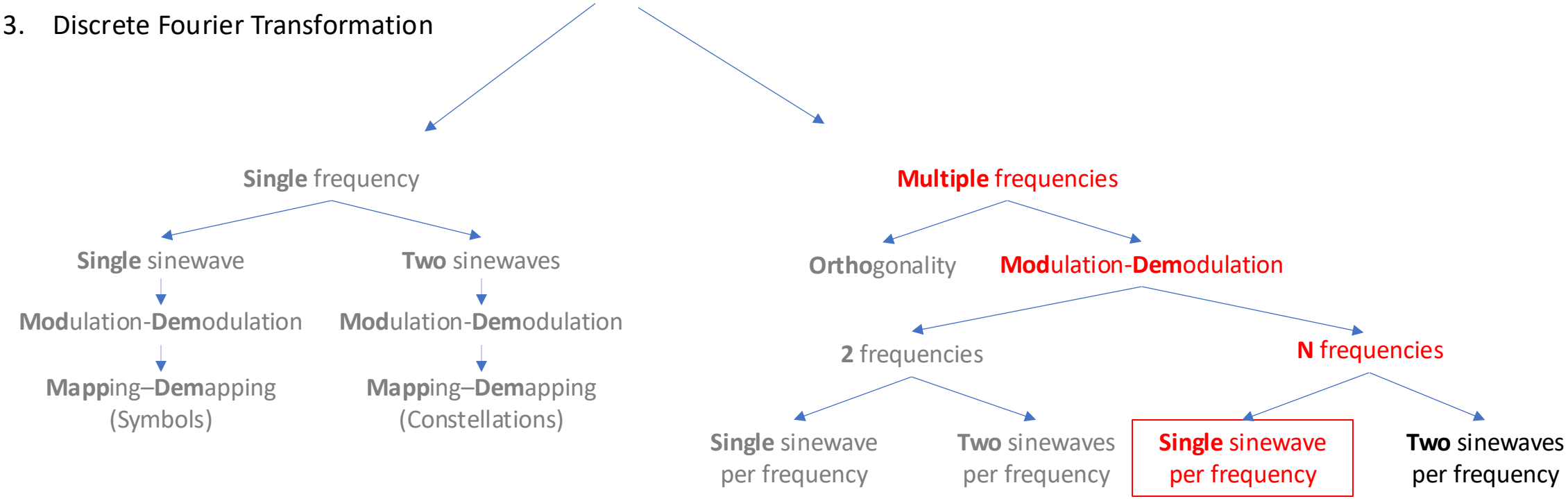





```
amp_f1_sin=8.7
amp_f1_cos=-6.5
amp_f2_sin=-4.3
amp_f2_cos=2.1
```

You are here

1. Sinewaves orthogonality
2. Orthogonal sinewaves amplitude modulation
3. Discrete Fourier Transformation

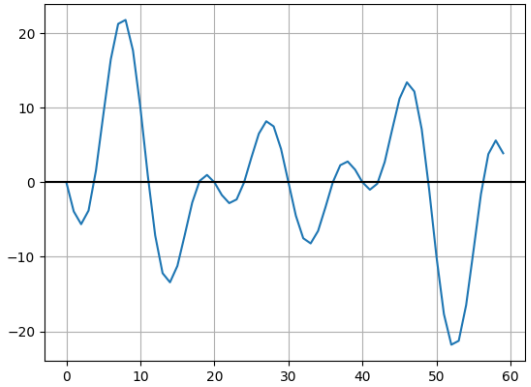


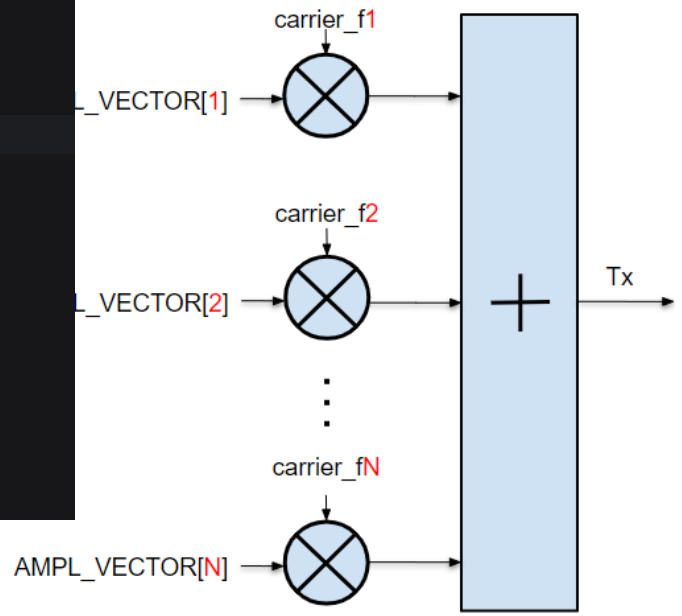
NOKIA

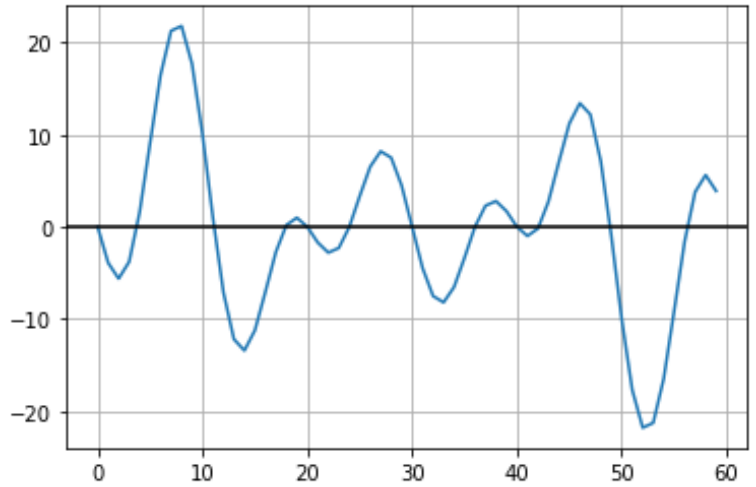
N frequencies : **Single** sinewave per frequency : **Modulation**

Complete script to obtain Tx signal composed of **N** amplitude modulated sinusoids of orthogonal frequencies

```python
import ...
pi = np.pi

# PARAMETERS
TIME_VECTOR_SIZE = 60
# amplitudes for frequencies [1,2,3,4,5,6]
AMPL_VECTOR = [1.23,4.56, 7.89,-1.23,-4.56, -7.89]

# CALCULATION
t = np.linspace( start: 0, 2*pi,TIME_VECTOR_SIZE, endpoint=False)

Tx = np.zeros(len(t))
for i, amp in enumerate(AMPL_VECTOR):
    f = i + 1
    Tx += amp*np.sin(f*t)

# PRESENTATION
plt.plot(Tx)
plt.axhline(y=0,color='black')
plt.grid()
plt.show()

# SAVING
np.save( file: 'TxSignal',Tx)
```





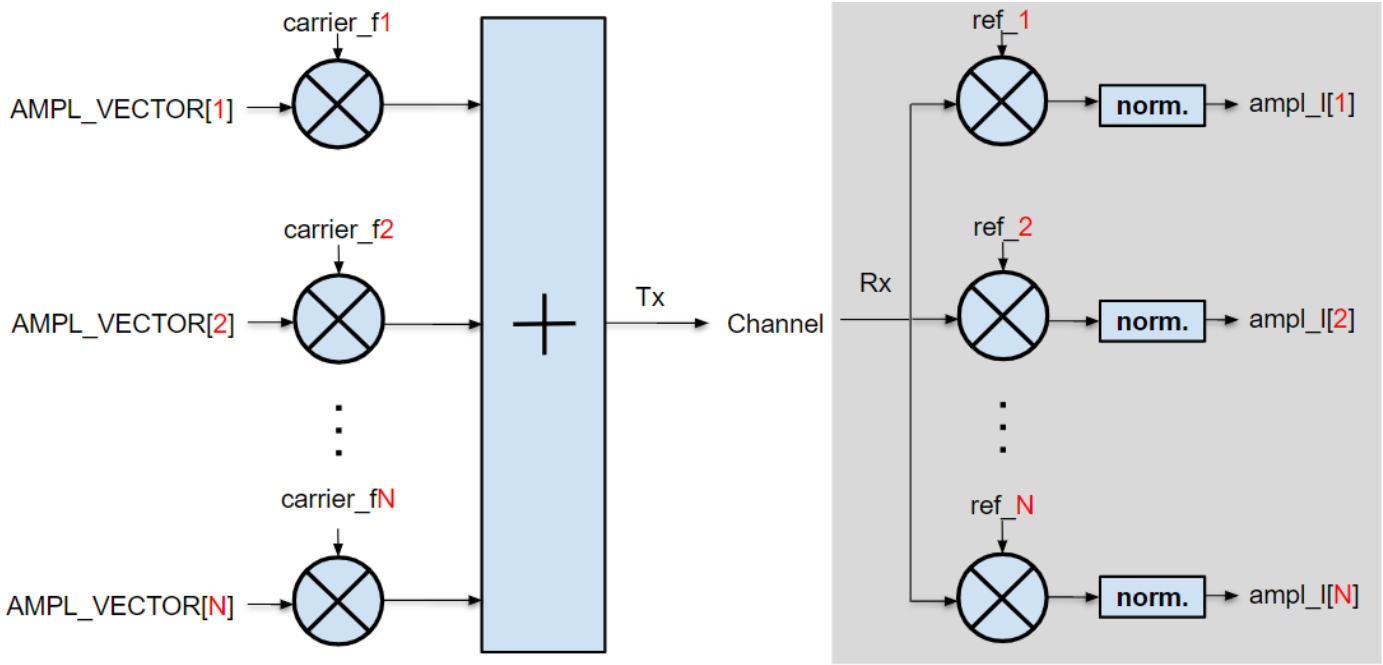


NOKIA

N frequencies : **Single** sinewave per frequency : **De**modulation

Complete script to **demodulate** signal which consists of N amplitude modulated sinusoids of orthogonal frequencies

```python
import ...
pi = np.pi

# loading Rx vector from file and..
Rx = np.load('TxSignal.npy')

# .. ploting it
plt.plot(Rx)
plt.axhline(y=0,color='black')
plt.grid()
plt.show()

# PARAMETERS
TIME_VECTOR_SIZE = 60
CARRIER_NR = 6

# CALCULATION
t = np.linspace( start: 0, 2*pi,TIME_VECTOR_SIZE, endpoint=False)

ampl_l = list()
for i in range(0, CARRIER_NR):
    f = i + 1
    dot  = np.dot(Rx, np.sin(f*t))
    ampl = dot/np.dot(np.sin(f*t), np.sin(f*t))
    ampl_l.append(ampl)

# PRESENTATION
for a in ampl_l:
    print(f'{a:0.2f}',end=', ')
```

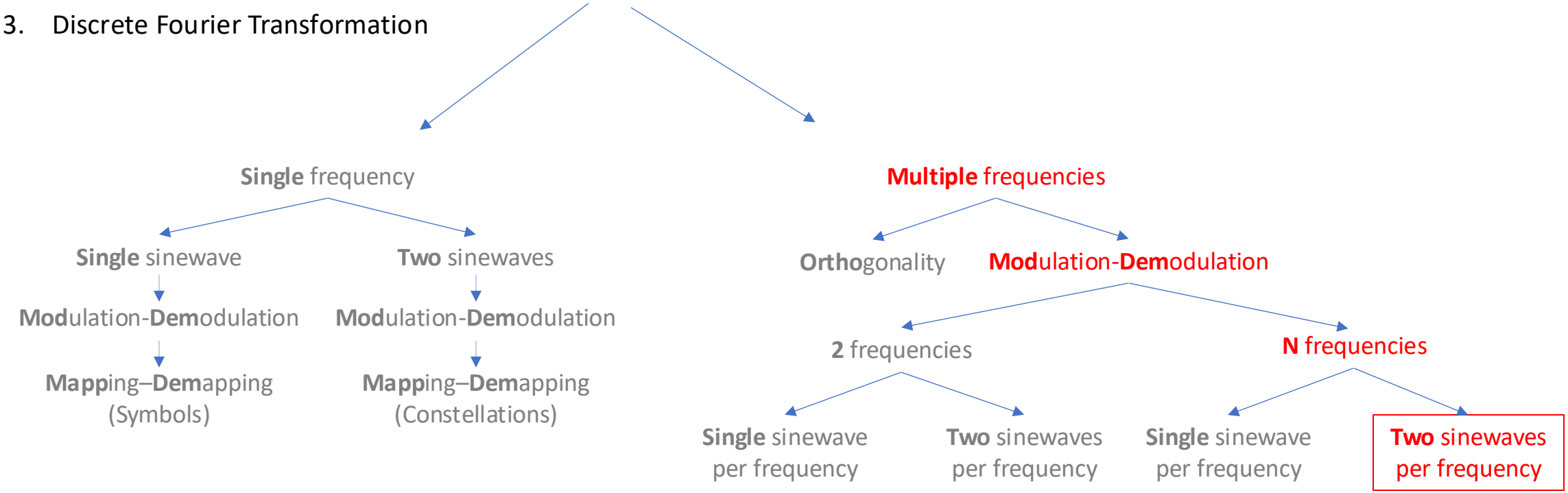



```
1.23, 4.56, 7.89, -1.23, -4.56, -7.89,
```


NOKIA

You are here

1. Sinewaves orthogonality
2. Orthogonal sinewaves amplitude modulation
3. Discrete Fourier Transformation



NOKIA

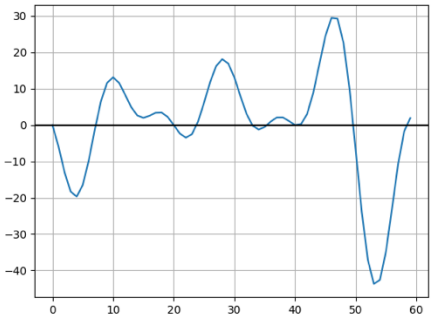
N frequencies : **Two** sinewaves per frequency : **Modulation**

Complete script to make **quadrature modulation** of **N** orthogonal frequencies

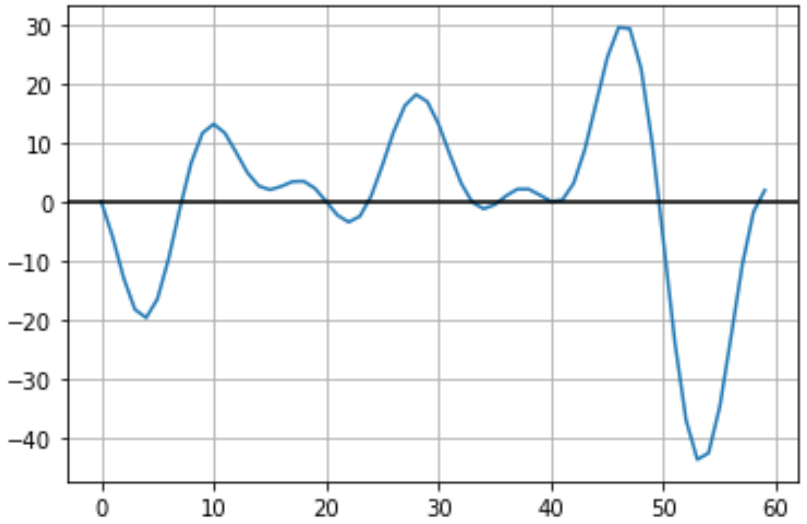
```python
import ...
pi = np.pi

# PARAMETERS
TIME_VECTOR_SIZE = 60

# amplitudes for frequency [1,2,3,4,5,6]
AMPL_VECTOR_SIN = [ 1.23,  4.56,  7.89, -1.23, -4.56, -7.89]
AMPL_VECTOR_COS = [-9.87, -6.54, -3.21,  9.87,  6.54,  3.21]

# CALCULATION
t = np.linspace( start: 0, 2*pi,TIME_VECTOR_SIZE, endpoint=False)

Tx = np.zeros(TIME_VECTOR_SIZE)
for i, (amp_sin,amp_cos) in enumerate(zip(AMPL_VECTOR_SIN,AMPL_VECTOR_COS)):
    f   =  i + 1
    Tx +=  amp_sin * np.sin(f*t) + amp_cos * np.cos(f*t)

# PRESENTATION
plt.plot(Tx)
plt.axhline(y=0,color='black')
plt.grid()
plt.show()

# SAVING
np.save( file: 'TxSignal',Tx)
```

```
56, -7.89]
54,  3.21]

=False)

TOR_SIN,AMPL_VECTOR_COS)):
```

```
np.save('TxSignal',Tx)
```



Tx



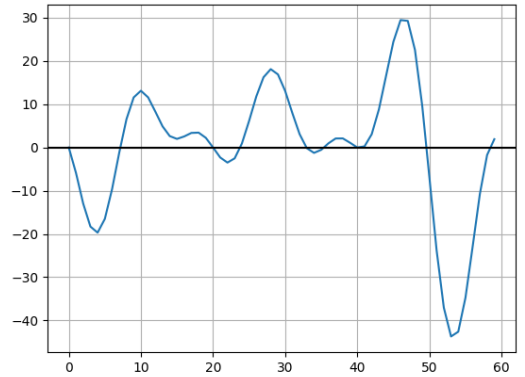
NOKIA

N frequencies : Two sinewaves per frequency : Demodulation

Complete script to **demodulate** signal which consists of **N quadrature modulated** frequencies

```
import ...
pi = np.pi

Rx = np.load('TxSignal.npy')

plt.plot(Rx)
plt.axhline(y=0,color='black')
plt.grid()
plt.show()

# PARAMETERS
TIME_VECTOR_SIZE = 60

# CALCULATION
t = np.linspace( start: 0, 2*pi,TIME_VECTOR_SIZE, endpoint=False)

ampl_sin_l = list()
ampl_cos_l = list()

for f in range(1,7):
    dot_sin = np.dot(Rx, np.sin(f*t))
    dot_cos = np.dot(Rx, np.cos(f*t))

    ampl_sin = dot_sin/np.dot(np.sin(f*t), np.sin(f*t))
    ampl_cos = dot_cos/np.dot(np.cos(f*t), np.cos(f*t))

    ampl_sin_l.append(ampl_sin)
    ampl_cos_l.append(ampl_cos)

for a in ampl_sin_l:
    print(f'{a:0.2f}',end=', ')
print ()
for a in ampl_cos_l:
    print(f'{a:0.2f}',end=', ')
```

nt=False)

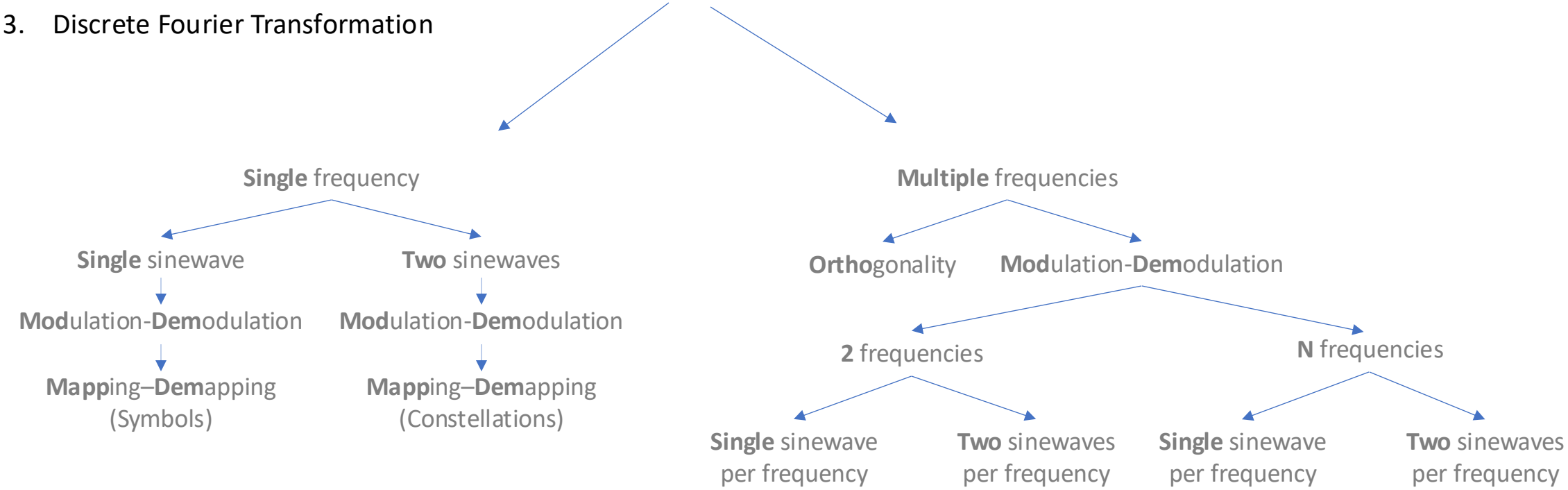



```
1.23, 4.56, 7.89, -1.23, -4.56, -7.89,
-9.87, -6.54, -3.21, 9.87, 6.54, 3.21,
```


NOKIA

You are here

1. Sinewaves orthogonality
2. Orthogonal sinewaves amplitude modulation - **DONE**
3. Discrete Fourier Transformation

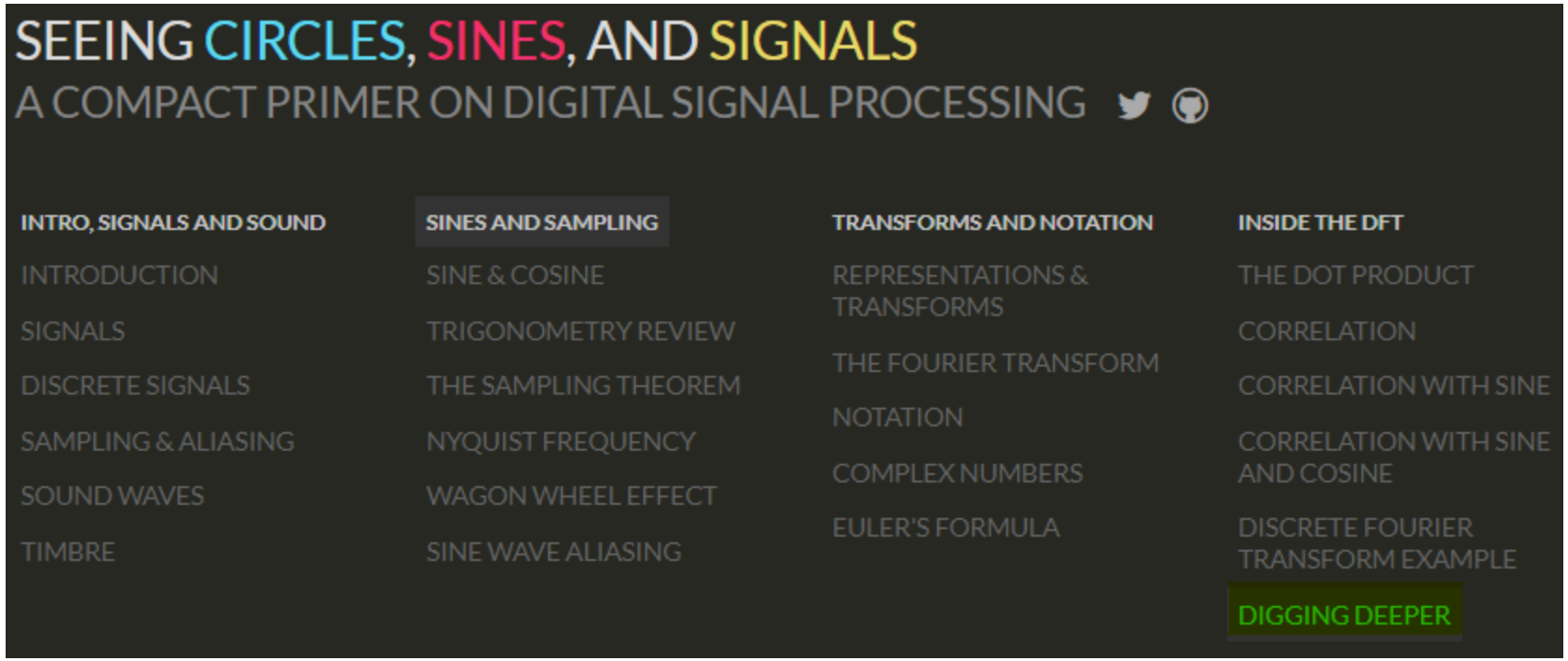


NOKIA

Discrete Fourier Transformation by Jack Schaedler

1. Sinewaves orthogonality
2. Orthogonal sinewaves amplitude modulation
3. Discrete Fourier Transformation (DFT)

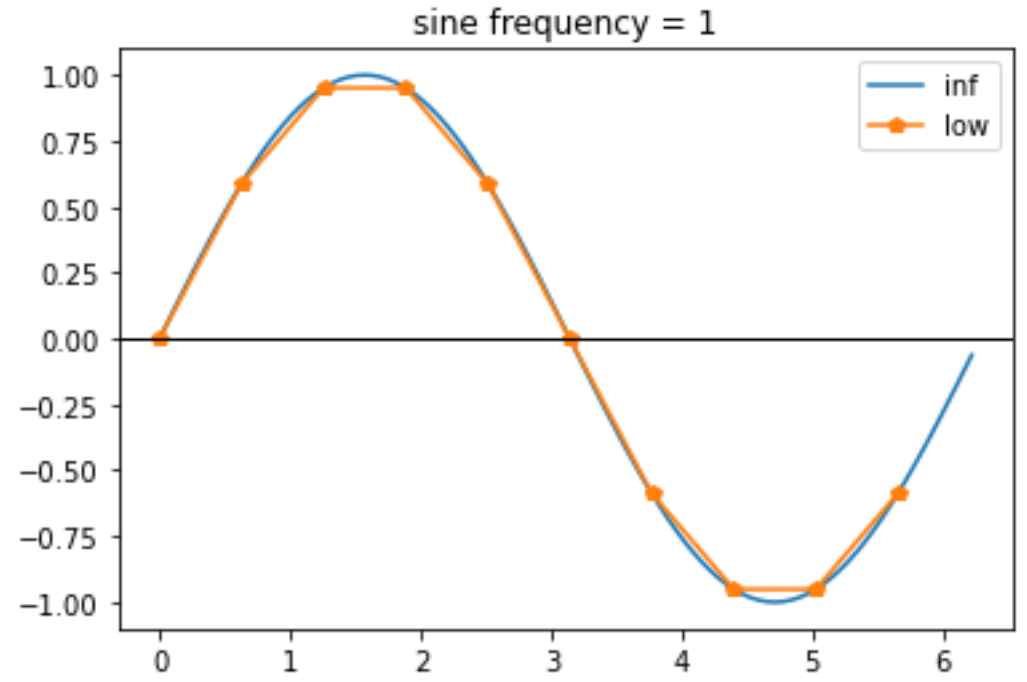
Familiar with below web page up to and including chapter “Digging deeper”.
https://jackschaedler.github.io/circles-sines-signals/index.html
(It is the best introduction to DFT known to tutor. It should take 3h)

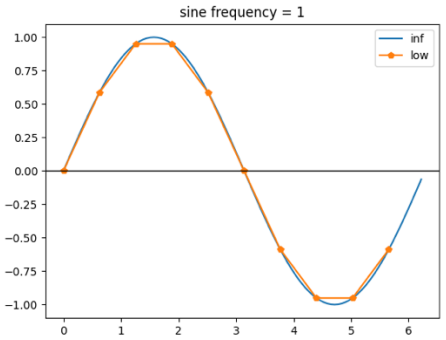


DFT, Aliazing (ambiguity)

Complete script to obtain given plot.

```python
import ...
from matplotlib.pyplot import legend

INFINIT_SAMPLE_RATE = 100 # infinity sample rate imitation (analog signal)
FINIT_SAMPLE_RATE = 10    # real (finit) sample rate

SIN_FREQ = 1

t_ideal    = np.linspace( start: 0, 2*np.pi, INFINIT_SAMPLE_RATE, endpoint=False)
t_sampled  = np.linspace( start: 0, 2*np.pi, FINIT_SAMPLE_RATE, endpoint=False)

sin_ideal   =  np.sin(SIN_FREQ*t_ideal)
sin_sampled =  np.sin(SIN_FREQ*t_sampled)

plt.plot( *args: t_ideal, sin_ideal, label='inf')
plt.plot( *args: t_sampled, sin_sampled, 'p-', label='low')
plt.axhline( y: 0,color='black', linewidth=1)
plt.title(f'sine frequency = {SIN_FREQ}')
plt.legend()
plt.show()
```

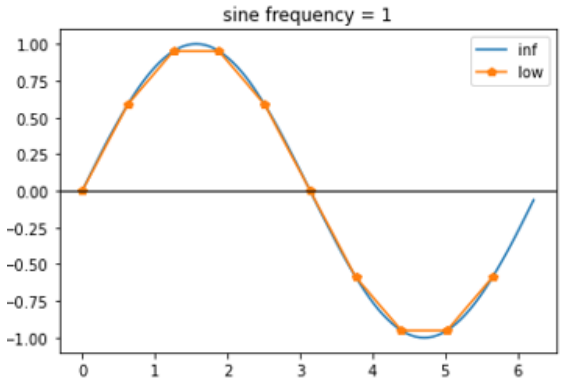


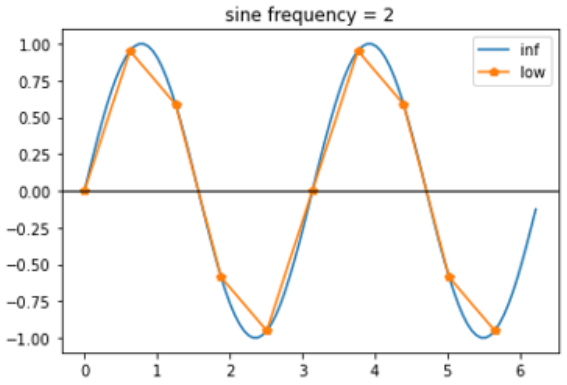


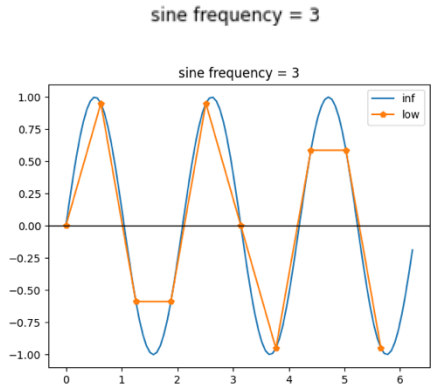
NOKIA

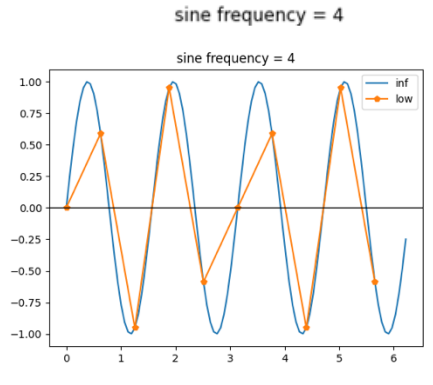
DFT, Aliazing cont

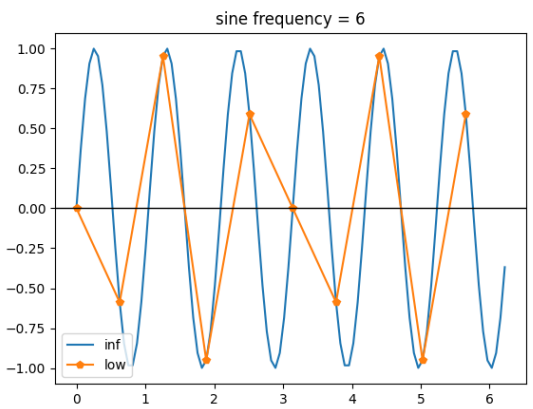
Use the script to complete missing plots

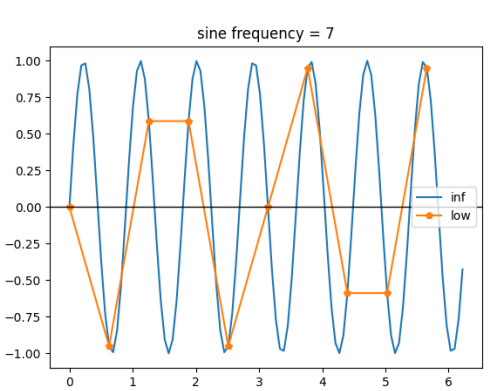


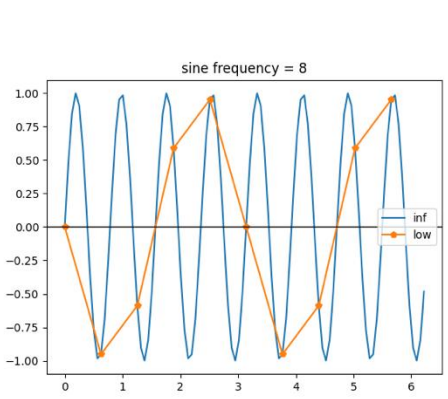


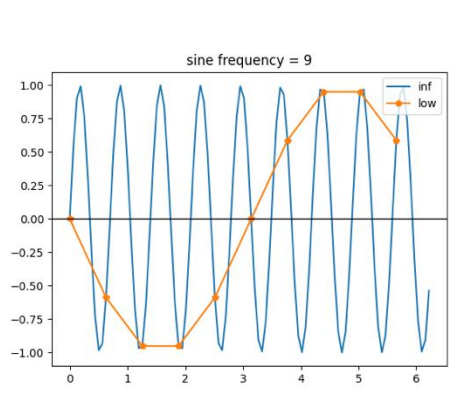










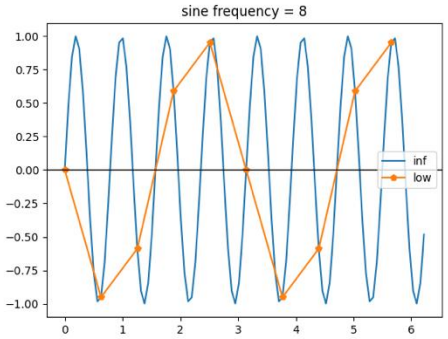


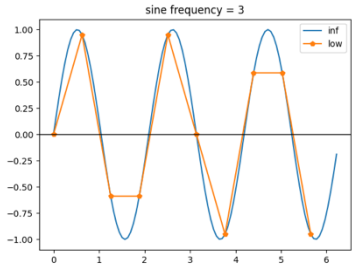
DFT Aliazing cont

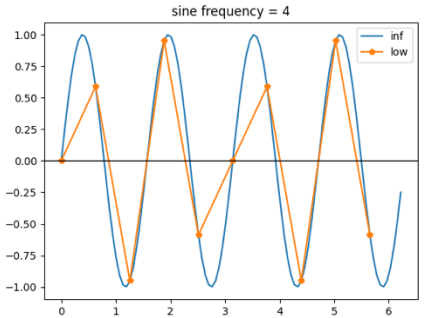
Copy all plots from previous slide to this slide **but**

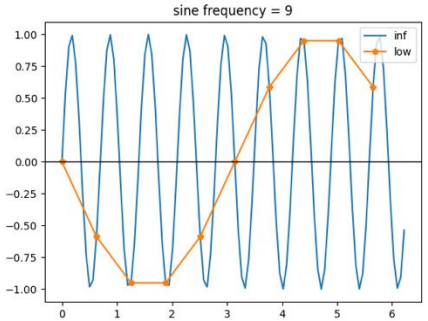
put the plots in lower row below similar (or “antisimilar”) plots from upper row.

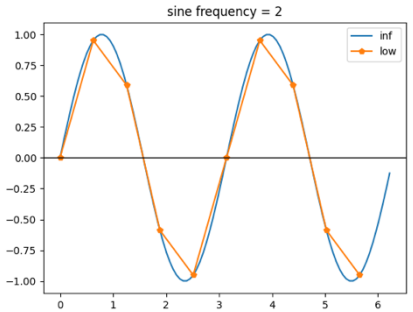
1. Which frequency is “axis of symmetry” ? 5
2. What is relation between it and sampling frequency? It’s half of real sample rate
3. Which sine is similar to sin(freq=3)? 7

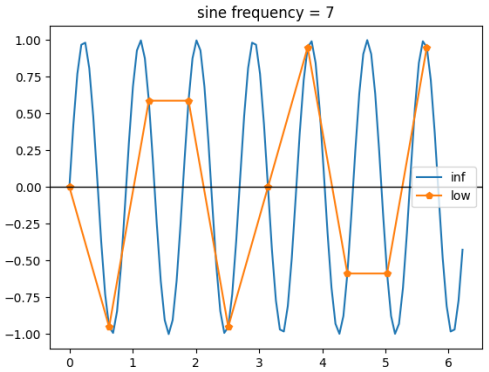


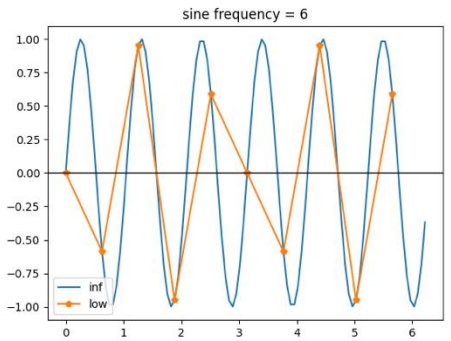


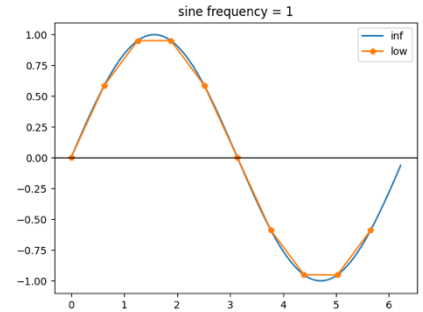












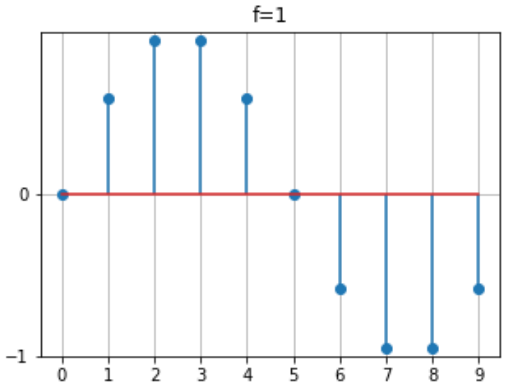
NOKIA

DFT, input

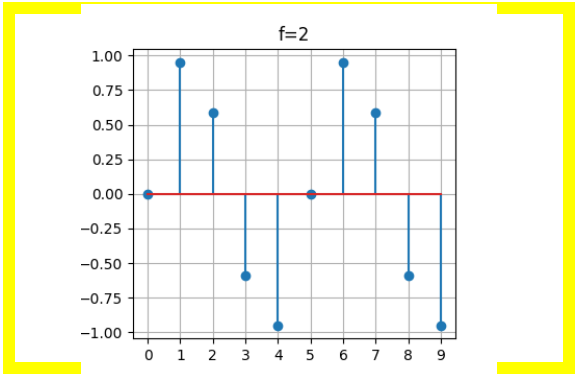
1. Use given script to complete plots
2. Move function “my_stem_plot” to library “mylib” and make script to run again

“f” is a “SIN_FREQ” parameter from script

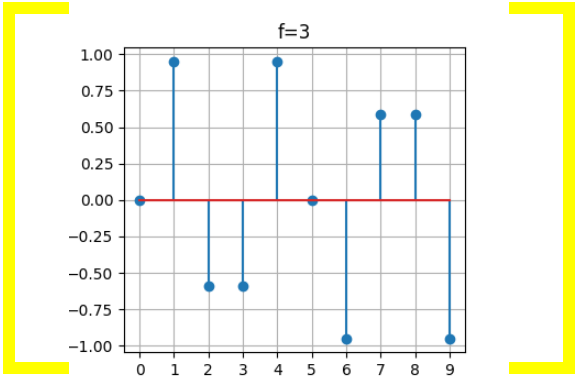
```
def my_stem_plot(y,title,y_range=None):
    x = np.arange(len(y))
    plt.stem(x,y,'-p')

    plt.xticks(x)

    if y_range:
        plt.ylim(y_range)
        plt.yticks(np.arange(*y_range))

    plt.grid()
    plt.title(title)
    fig = plt.gcf()
    fig.set_size_inches(4, 3.6)
    plt.show()

#---------------------------

SAMPLE_NR = 10
FREQ = 1

t = np.linspace(0, 2*np.pi, SAMPLE_NR, endpoint=False)

samples =  np.sin(t*FREQ)
my_stem_plot(samples,f'f={FREQ}')
```



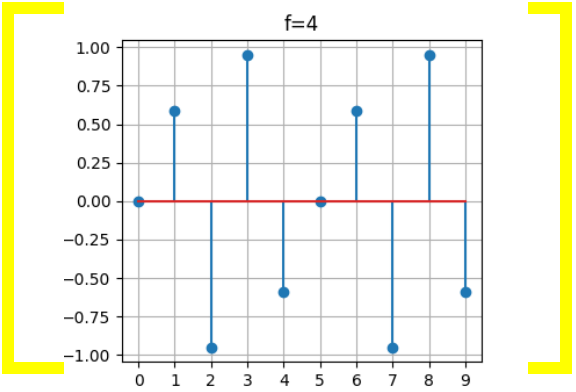
f=2



f=3



f=4



NOKIA

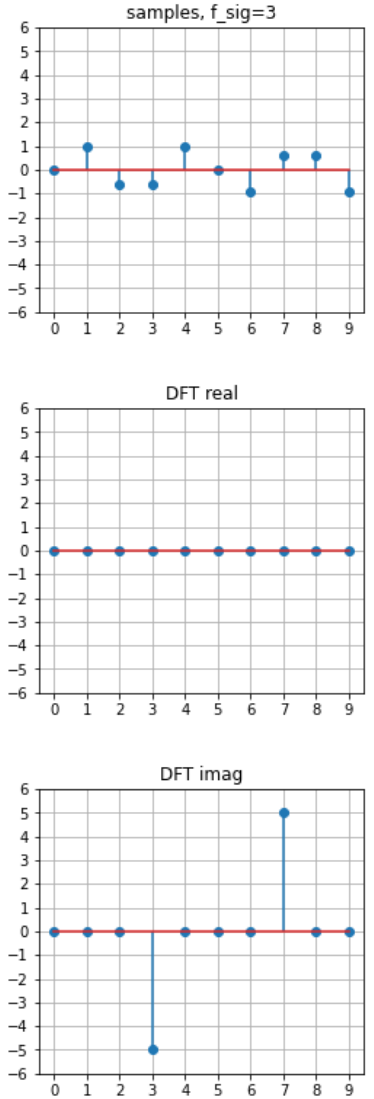
DFT, NumPy implementation (FFT)

Complete script to get given plot set:

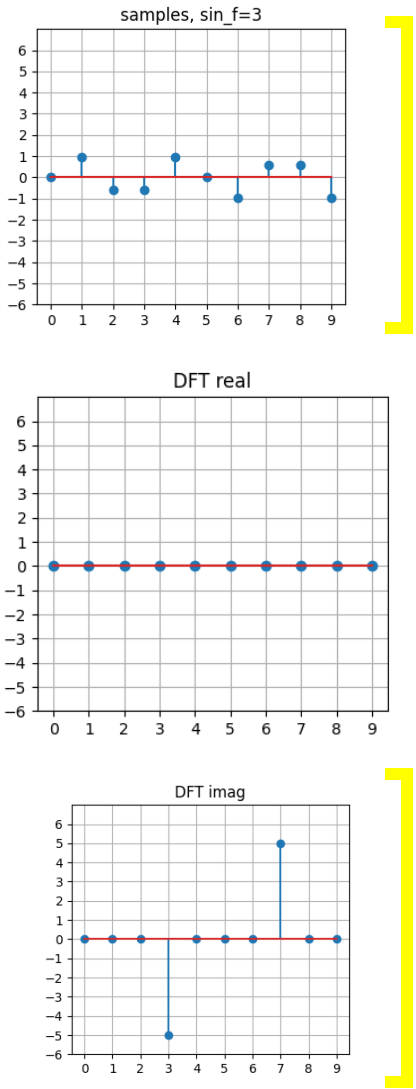
- import “my_stem_plot”
- modify “my_stem_plot” to get plots figures in the shape of square

```python
import numpy as np
import matplotlib.pyplot as plt
from mylib import my_stem_plot

SAMPLE_NR = 10
FREQ = 3

t = np.linspace( start: 0, 2*np.pi, SAMPLE_NR, endpoint=False)

samples =  np.sin(t*FREQ) #+ np.cos(t)
my_stem_plot(samples,  title: f'samples, sin_f={FREQ}',y_range=(-6,7))

fft = np.fft.fft(samples)
my_stem_plot(fft.real, title: 'DFT real',y_range=(-6,7))
my_stem_plot(fft.imag, title: 'DFT imag',y_range=(-6,7))
```

Reference plots



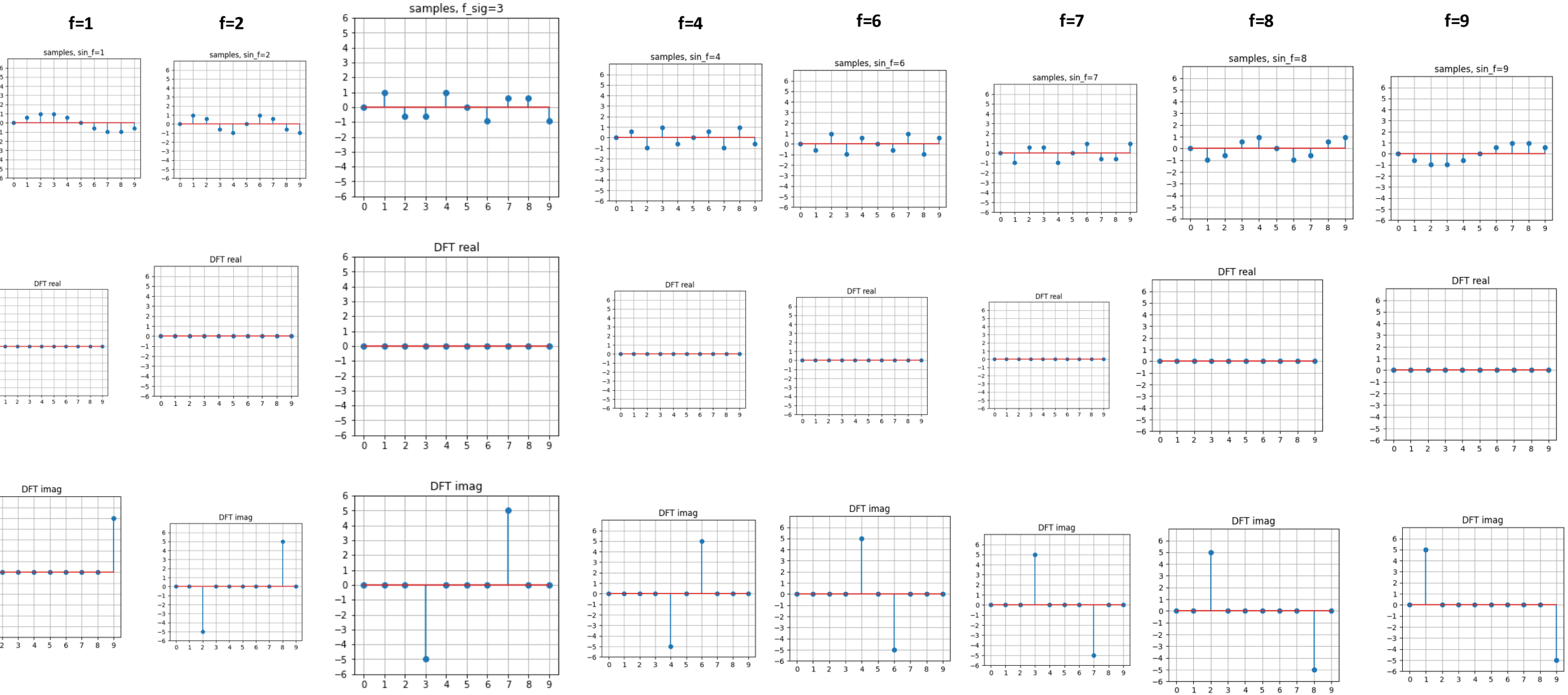
Your plots



NOKIA

DFT, numpy implementation (FFT)

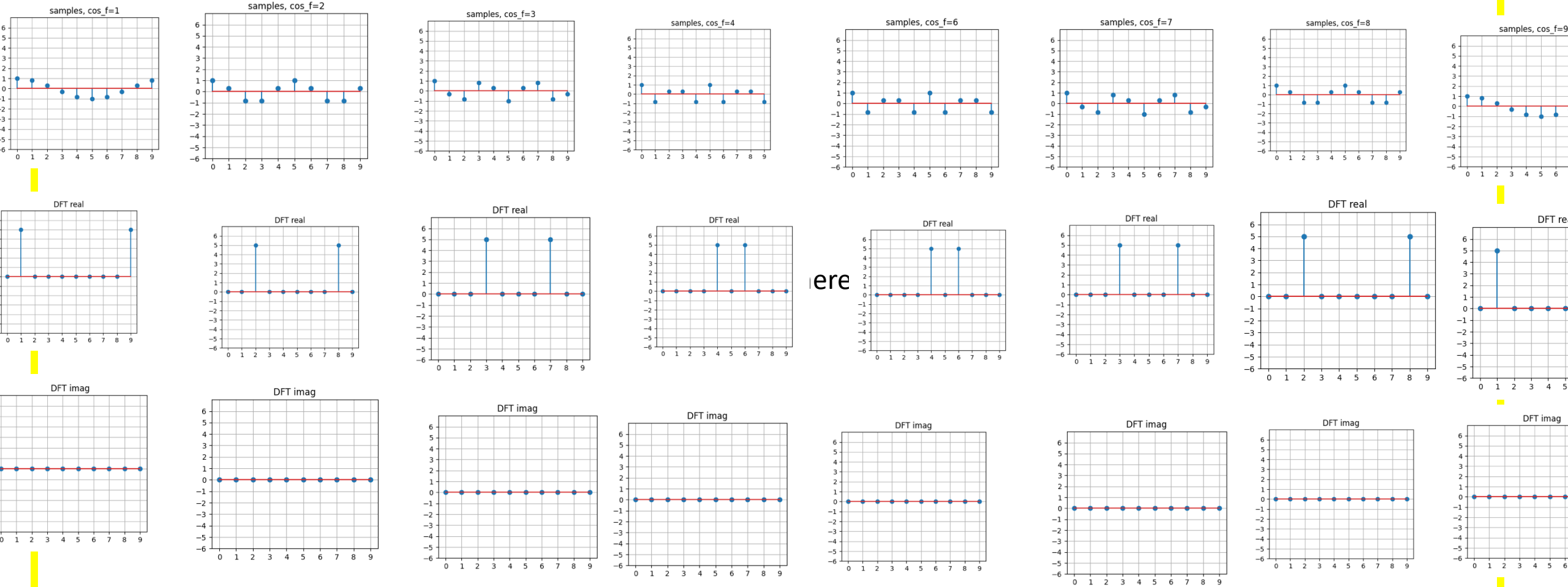
Complete given plot set



NOKIA

DFT: real component

Repeat previous slide for **cosine**

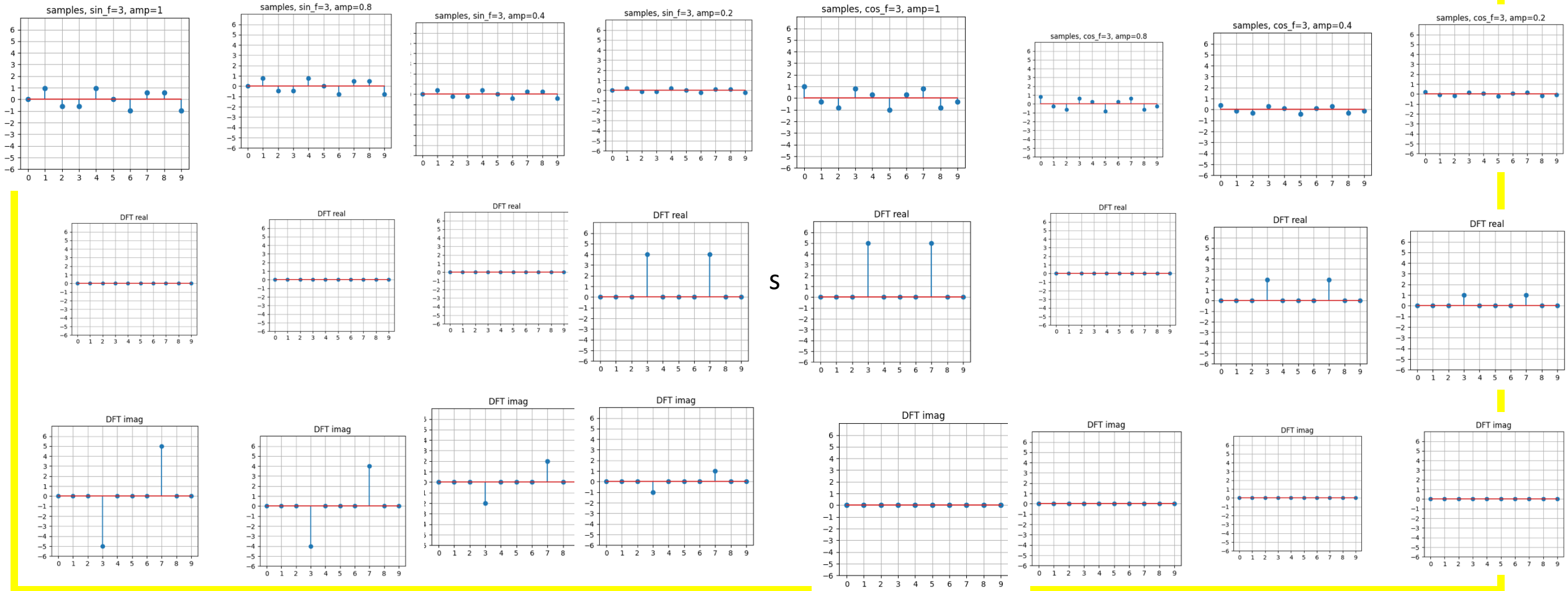


DFT, amplitude

Generate plot set from previous slide for sinus and csinus f=3 and signal amplitudes: 1, 4/5, 2/5, 1/5

Base on plots analizys write formula for DFT amplitude:

- RealAmp(cos_amp,SAMPLE_NR) = cos_amp*SAMPLE_NR/2
- ImagAmpl(sin_amp,SAMPLE_NR) = sin_amp*SAMPLE_NR/2



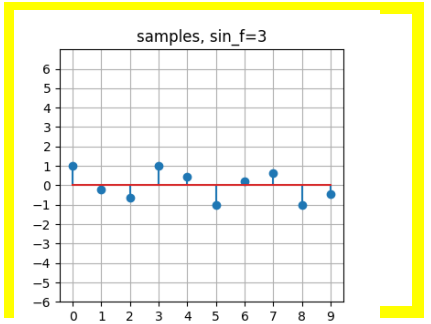
NOKIA

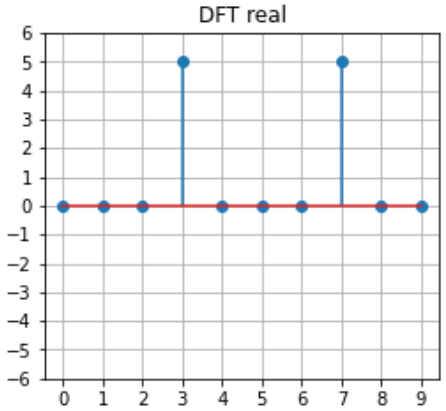
DFT, amplitude cont.

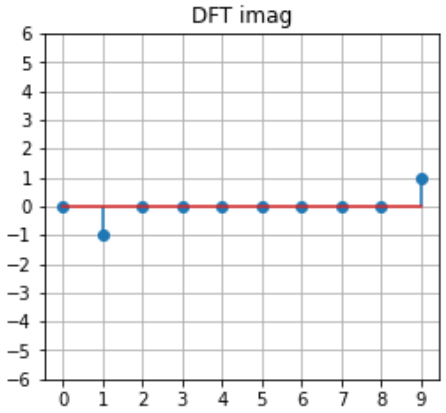
Find formula for signal which gives presented DFT

```
samples =  0.2*np.sin(t*FREQ/3) + np.cos(t*FREQ)
```

Place on the slide corresponding samples plot







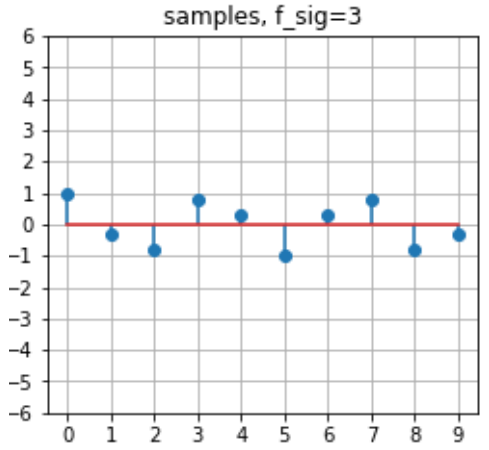
NOKIA

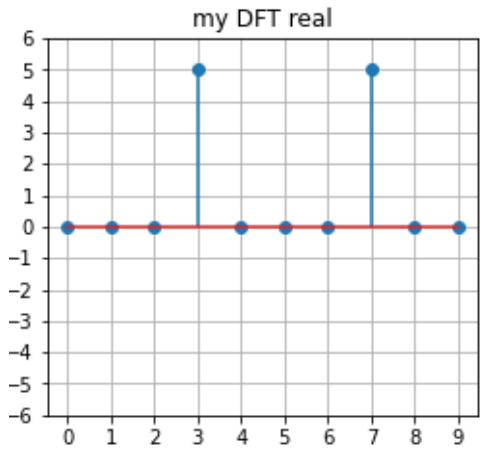
Your DFT

Complete the part of the script evaluating real part of DFT.
(Recall from previous chapters how to measure amplitude of sinus or cosinus in signal)

```python
import numpy as np
import matplotlib.pyplot as plt
from mylib import my_stem_plot

SAMPLE_NR = 10
SIN_FREQ = 3

t = np.linspace( start: 0, 2*np.pi, SAMPLE_NR, endpoint=False)

samples =  np.cos(t*SIN_FREQ)
my_stem_plot(samples, title: f'samples, f_sig={SIN_FREQ}')

real = list()
for f in range(SAMPLE_NR):
    real.append(np.dot(samples,np.cos(t*f)))

my_stem_plot(real, title: 'my DFT real',y_range=(-6,7))
```

Plots for SIN_FREQ =3 should look as following





NOKIA

Your DFT cont

1. Complete previous script with evaluation of imaginary part

2. Encapsulate DFT related code into “myDFT” function which should:
   - receive ne argument (samples)
   - not use SAMPLE_NUMER const.
   - use ne loop only
   - “`return real, image`”

3. Move the function to mylib and make the main script work

```python
import numpy as np
import matplotlib.pyplot as plt
from mylib import my_stem_plot
from mylib import myDFT

SAMPLE_NR = 10
SIN_FREQ = 3

t = np.linspace( start: 0, 2*np.pi, SAMPLE_NR, endpoint=False)

samples =  np.cos(t*SIN_FREQ)
my_stem_plot(samples, title: f'samples, f_sig={SIN_FREQ}')

real, imag = myDFT(samples)

my_stem_plot(real, title: 'my DFT real',y_range=(-6,7))
my_stem_plot(imag, title: 'my DFT imag',y_range=(-6,7))
```

```python
def myDFT(samples):  2 usages
    N = len(samples)
    real = list()
    imag = list()
    t = np.linspace(0, 2 * np.pi, N, endpoint=False)

    for f in range(N):
        real.append(np.dot(samples, np.cos(t * f)))
        imag.append(np.dot(samples, np.sin(t * f)))

    return real, imag
```

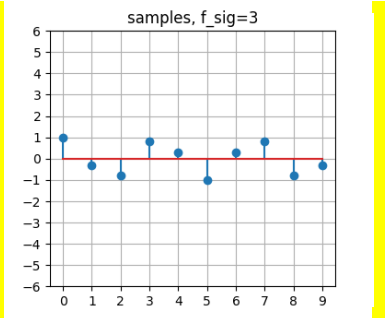

NOKIA

Your DFT

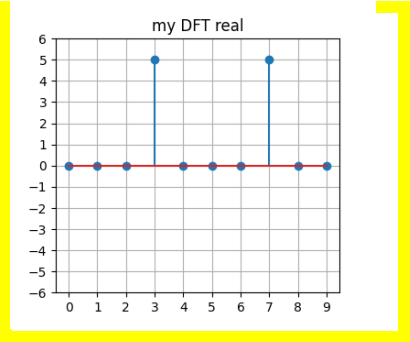
1. Check myDFT by using following script. It should generate plots as below.

```
SAMPLE_NR = 10

t = np.linspace(0, 2*np.pi, SAMPLE_NR, endpoint=False)

samples =  np.cos(t*1)*2/5+np.sin(t*4)*4/5
my_stem_plot(samples,f'samples')

real, imag = myDFT(samples)
my_stem_plot(real,'myDFT real')
my_stem_plot(imag,'myDFT imag')

fft = np.fft.fft(samples)
my_stem_plot(fft.real,'FFT real',y_range=(-6,7))
my_stem_plot(fft.imag,'FFT imag',y_range=(-6,7))
```

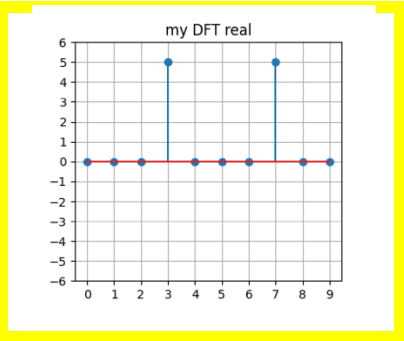
samples



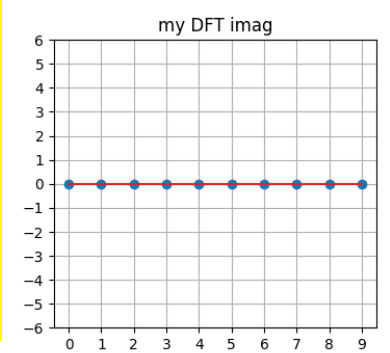
FFT real



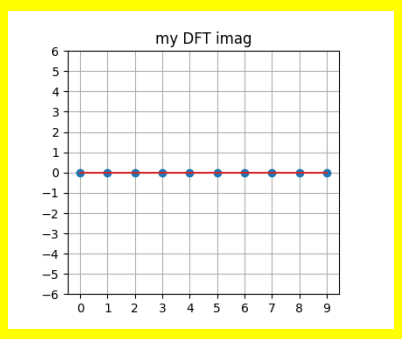
myDFT real



FFT imag



myDFT imag



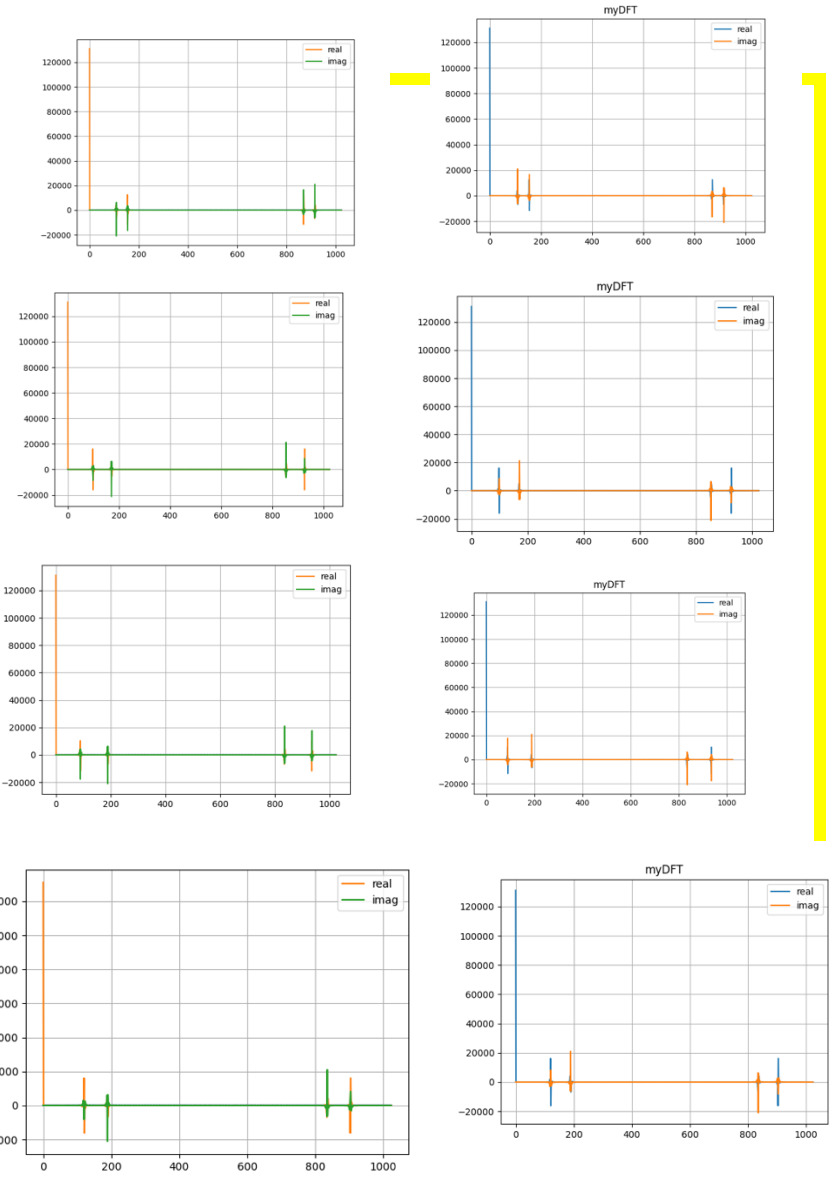
NOKIA

“myDFT–numpy DFT” comparison by using DTFM signal

Use following script for “myDFT–numpy DFT” comparison by using exemplary DTFM signal. (Read comments carefully)

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.io.wavfile import read

# loads samples from .wav file with exemplary DTFM signal
# adapt file path
samples = read(r'wav\a.wav')
samplig_freq = samples[0]
samples = samples[1]
samples = samples[:1024]
plt.plot(samples)

# use commenst to swith between myDTF and numpy DTF (FFT)
fft = np.fft.fft(samples)
real = fft.real
imag = fft.imag
# real, imag = myDFT(samples)

plt.plot(real,label='real')
plt.plot(imag,label='imag')
plt.grid()
plt.legend()

#culd be usefool for zooming
# plt.xlim(0,100)
# plt.ylim(-20_000,2_0000)
```

https://www.audiocheck.net/audiocheck_dtmf.php



Decoding DTFM signal by using DFT

Use following script from previous slide to decode following samples of DTFM signal (see Wikipedia).

Replace question mark with key.

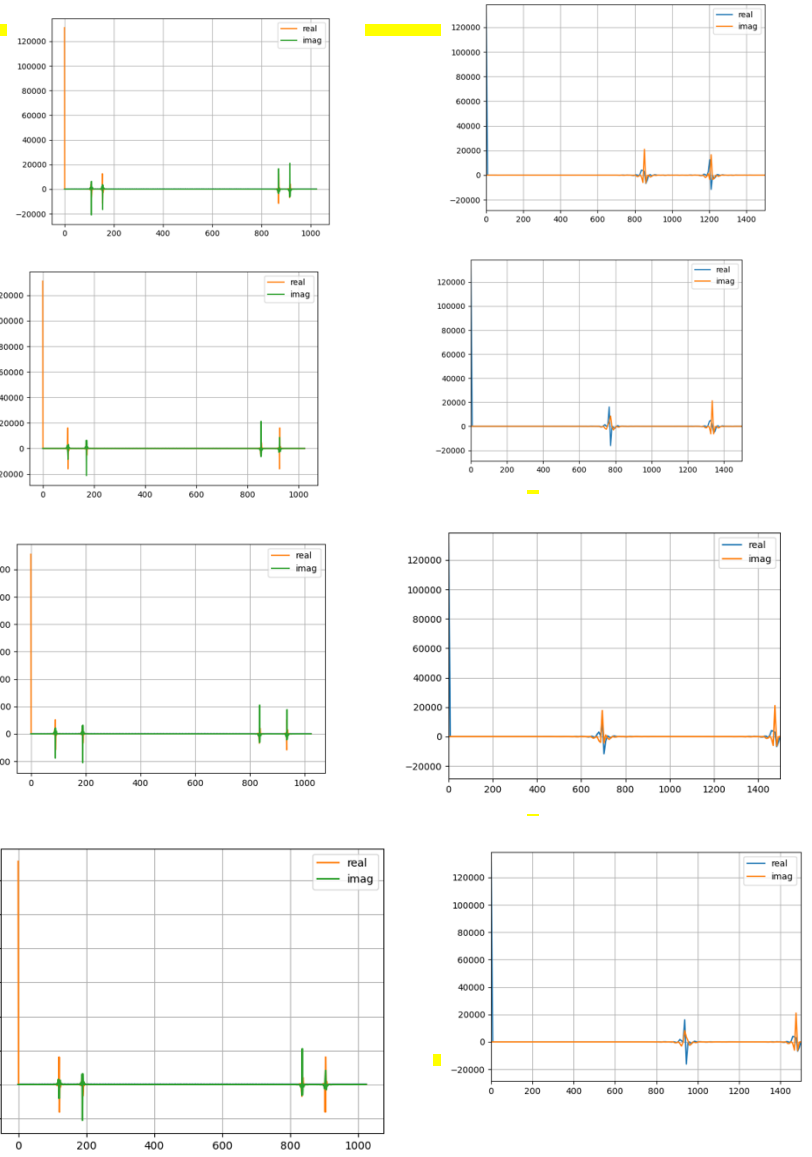
a.wav: 7

b.wav: 5

c.wav: 3

d.wav: #

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.io.wavfile import read
from mylib import myDFT, my_stem_plot
# loads samples from .wav file with exemplary DTFM signal
# adapt file path
samples = read(r'wav/d.wav')
samplig_freq = samples[0]
samples = samples[1]
samples = samples[:1024]
plt.plot(samples)
plt.show()
# use commenst to swith between myDTF and numpy DTF (FFT)
#fft = np.fft.fft(samples)
#real = fft.real
#imag = fft.imag
real, imag = myDFT(samples)

plt.title("myDFT")
plt.plot( *args: real,label='real')
plt.plot( *args: imag,label='imag')
plt.grid()
plt.legend()
plt.show()
#frequencies on x
plt.plot( *args: np.linspace( start: 0, samplig_freq,len(samples)), real, label='real')
plt.plot( *args: np.linspace( start: 0, samplig_freq, len(samples)), imag, label='imag')
plt.grid()
plt.show()
#culd be usefool for zooming
plt.xlim( *args: 0,100)
plt.ylim( *args: -20_000,20_000)
```



NOKIA