**LIBRARY DEVELOPMENT FOR ANALYTICAL SIGNAL SIMULATION**

**Annotation**

This paper presents a software implementation of the analytical signal model in Python 3.6.9. The article describes the mathematical model of the signal under investigation: the structure of the analytical signal; Hilbert transform and its basic properties; signal parameters description:

• Amplitude

• Power

• Frequency

The block diagram of the implemented algorithms, the program code and an example of signal simulation.

**INTRODUCTION**

An analytical signal (complex signal) is a mathematical representation of an analog signal in the form of a complex analytical function of time used in signal processing theory. The usual, real signal x is in this case the real part of the analytic representation *s(t)*. The theory of analytical signals is a developed direction in the theory of control systems.

As a rule, ready-made models of the studied phenomenon are used to solve applied problems. In particular, if the solution of the problem is performed in any programming language, then ready-made libraries of functions and methods are used. This speeds up the development of a software solution and allows you to concentrate on solving the problem, bypassing the technical implementation of the subject. The layer of abstraction is also increased, allowing for a more flexible approach to the solution.

Therefore, when approaching the problem of modeling the analytical signal, such a method was chosen. Based on the already built mathematical models, an interface (library of functions) was developed for modeling the signal and its parameters. In connection with the often encountered problem of signal noise, the function of smoothing signals (including parameters) was also developed, implemented by the "simple moving average" method.

**MATHEMATICAL MODEL**

The formal definition of a complex signal is as follows:

*,* (1)

where *u(t)* is a real signal with a Fourier image; v (t) - conjugate signal by Hilbert transform method.

In this case, the conjugate signal is related to the real part using the Hilbert transform as follows:

. (2)

We list the most important properties of the Hilbert transform:

1. , where sgn(x) – sign function. (3)

2. . (4)

Formula (3) gives an idea of the amplitude-frequency characteristic of the Hilbert transform, (4) the formula shows how the inverse transformation is represented, that is, the return to the original signal.

In addition to the listed properties, it is also very important that the Hilbert transform is expressed through the Fourier transform. This fact plays a major role in the software implementation of the analytical signal model.

Let *U (iω) = F [u (t)]* be the Fourier image of the real signal. Since the Fourier image *s(t)* is:

. (5)

Then you can calculate *v(t)* complement by the formula:

, (6)

where *Im(z)* is the imaginary part of *z*; *F-1* - Inverse Fourier Transform.

Based on the complex signal model, it is also possible to give definitions of the main parameters of the signal, its amplitude *a(t)* or power *c(t)*:

, (7)

and phases:

(8)

and also frequencies:

, (9)

In this implementation of the signal model, the present smoothing method works according to the following principle. A certain area is taken before and after a certain point and, taking into account the numerical values ​​of the measurements included in this area, we calculate the average value. Formally, this can be described as follows.

Let there be a sample of *N* points *{f1, f2,…, fN}*. Then, to find the average in the vicinity of the selected point *i*, we take the arithmetic mean of *M* previous and subsequent points, including the point *i* itself. Then the new values ​​of the points, let *gi*, will be calculated by the formula:

(10)

**PROGRAM DESCRIPTION**

When developing a complex signal model in Python 3.6.9, the following functions were implemented, covering the construction of the conjugate signal by Hilbert transform method, the construction of an analytical signal, the calculation of the main signal parameters, as well as noise smoothing.

The work is carried out as follows: the library file is included in the program header; code is written using the functions described below; below is the code that displays the received data; launched through the OS command line; the user receives an image (graph) of the simulated signal. The library interface consists of the following functions:

• *hilbert (signal)* - the function takes an array of signal values, returns an array representing the values ​​of the Hilbert transform.

• *оrt\_signal (signal)* - the function takes as a parameter an array of values ​​of a valid signal and returns an array representing the values ​​of the conjugate signal.

• *analitic\_signal (signal)* - takes an array of signal values, returns an array of complex analytical signal values.

• *amp (signal)* - takes an array of analytical signal values, returns the amplitude of the analytical signal.

*• cap (signal)* - takes an array of analytical signal values, returns the analytical signal power.

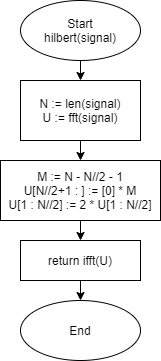
• *frq (signal, dx)* - takes an array of analytical signal values ​​(signal) and a signal sampling step (dx), returns the frequency of the analytical signal.

• *smoothing (signal, M)* - smooths the input signal using the moving averaging method, takes an array of values ​​of the actual signal (signal) and the degree of smoothing (M), returns a smoothed signal.

**ALGORITHMIC DESCRIPTION**

In this part of the work, the block diagrams of the above described functions are given with comments, with the exception of the functions amp (signal) and cap (signal), the implementation of which is trivial:

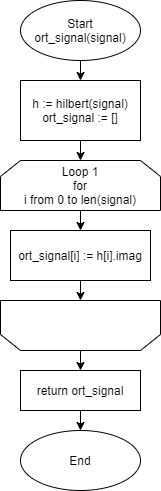
1.*hilbert ()*:



*Fig 1. Block diagram of the Hilbert transform function*

The *Start* block indicates the beginning of the program and contains the name of the function, as well as the accepted parameters. Further, in the description of subsequent functions, commenting is carried out starting from the second block. The second block: *N* - a variable containing the length (*len ()*) of the sample of signal values; *U* is the direct Fourier transform (*fft ()*) of the input signal. Third block: *M* - sample center. The next two lines calculate the Fourier image of the analytical signal, where, according to the theory, at negative frequencies *ω* the image is equal to zero, and at positive frequencies it is equal to the doubled value of the image of the real signal. Fourth block: Returns the inverse Fourier transform of a complex-valued signal. End of the algorithm.

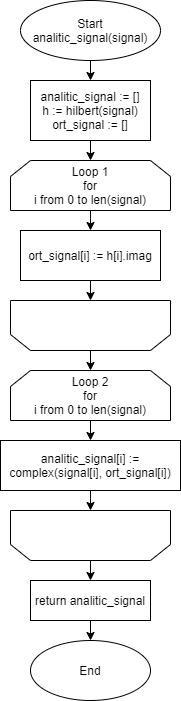
2.*ort\_signal ()*:



*Fig 2. Block diagram of the orthogonal signal function*

The second block: *h* - an array of values of the signal converted according to Hilbert; *ort\_signal* is an empty array for storing the values of the coupled signal. The third block: the loop of variable *i* passing from 0 to the end of the array, in the body of the loop, the imaginary part of the *h* array is assigned to the *ort\_signal* array. Fourth block: the function returns the array *ort\_signal*. End of the algorithm.

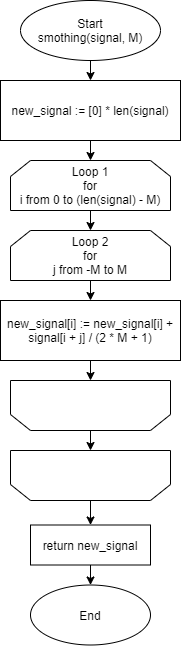
2.*analitic\_signal ()*:



*Fig 3. Block diagram of the analytical signal function*

The second block: *analitic\_signal* - an array for storing the final value; *h* and *ort\_signal* have the same meaning as in the previous algorithm. The third block: *Loop 1* - performs the same functions as in the previous algorithm. The fourth block: *Loop 2* - loop of variable *i* passing from 0 to the end of the array, complex values are assigned to the *analitic\_signal* array in the loop body, where the real part is the input signal, and the imaginary part is the conjugate signal. Fifth block: the function returns an array *analitic\_signal*. End of the algorithm.

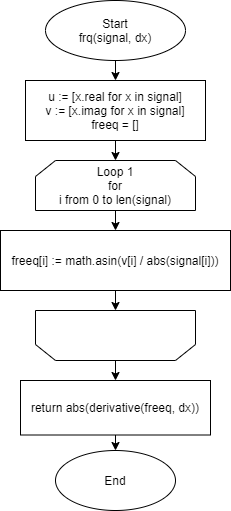
3.smoothing ():



*Fig 4. Block diagram of the function that implements signal smoothing*

Second block: *new\_signal* - an array for storing the final value. The third block: *Loop 1* - loop of the variable *i*, passing from 0 to the end of the array, except for the last *M* points. *Loop 1* contains a nested loop, *Loop 2*, which calculates new averages of the signal using the "simple moving average" method. Fourth block: the function returns the new\_signal array. End of the algorithm.

5.frq ():



*Fig 5. Block diagram of the analytical signal frequency function*

Second block: *u* - array of real values of the analytical signal; *v* - array of the imaginary part of the analytical signal; *freeq* is a variable for storing the total frequency values. The third block: *Loop 1* - the loop of variable *i* passing from 0 to the end of the signal sample. In the body of the loop, the phase values of the signal are calculated using the formula (8) with the arcsine. Fourth block: the function returns the time derivative of the phase, that is, the frequency value. End of the algorithm.

Full program code:

**import** matplotlib**.**pyplot **as** plt

**import** numpy **as** np

**import** math **as** m

**from** scipy**.**fftpack **import** **\***

**def** hilbert\_1**(**signal**):**

N **=** **len(**signal**)**

U **=** fft**(**signal**)**

M **=** N **-** N**//**2 **–** 1

U**[**N**//**2**+**1**:]** **=** **[**0**]** **\*** M

U**[**1**:**N**//**2**]** **=** 2 **\*** U**[**1**:**N**//**2**]**

v **=** ifft**(**U**)**

**return** v

**def** derivative**(**f**,** dx**):**

df **=** **[]**

df**.**append**((**f**[**1**]** **-** f**[**0**])** **/** dx**)**

**for** i **in** **range(**1**,** **len(**f**)):**

df**.**append**((**f**[**i**]** **-** f**[**i **-** 1**])** **/** dx**)**

**return** df

**def** ort\_signal**(**signal**):**

h **=** hilbert\_1**(**signal**)**

ort\_signal **=** **[]**

**for** i **in** **range(**0**,** **len(**signal**)):**

ort\_signal**.**append**(**h**[**i**].**imag**)**

**return** ort\_signal

**def** analitic\_signal**(**signal**):**

analitic\_signal **=** **[]**

h **=** hilbert\_1**(**signal**)**

ort\_signal **=** **[]**

**for** i **in** **range(**0**,** **len(**signal**)):**

ort\_signal**.**append**(**h**[**i**].**imag**)**

**for** i **in** **range(**0**,** **len(**signal**)):**

analitic\_signal**.**append**(complex(**signal**[**i**],** ort\_signal**[**i**]))**

**return** analitic\_signal

**def** smoothing**(**signal**,** K**):**

new\_signal **=** **[**0**]** **\*** **len(**signal**)**

**for** i **in** **range(**K**,** **len(**signal**)** **-** K**):**

**for** j **in** **range(-**K**,** K**):**

new\_signal**[**i**]** **+=** signal**[**i **+** j**]** **/** **(**2**\***K **+** 1**)**

**return** new\_signal

**def** amp**(**signal**):**

**return** np**.abs(**signal**)**

**def** cap**(**signal**):**

**return** np**.abs(**signal**)** **\*** np**.abs(**signal**)**

**def** frq**(**signal**,** dx**):**

u **=** **[**x**.**real **for** x **in** signal**]**

v **=** **[**x**.**imag **for** x **in** signal**]**

freeq **=** **[]**

**for** i **in** **range(**0**,** **len(**signal**)):**

freeq**.**append**(**m**.**asin**(**v**[**i**]** **/** np**.abs(**signal**[**i**])))**

**return** np**.abs(**derivative**(**freeq**,** dx**))**

**EXAMPLE OF SIGNAL CONSTRUCTION**

As an example of the program operation, the following code was written to simulate the conjugate signal for the *sin(x)* function. As a result, we have the following code:

**from** signal\_model **import** **\***

d **=** 10

discr **=** 0.0008

l **=** **int(**d **/** discr**)**

axis **=** **[(**x **\*** discr**)** **for** x **in** **range(**0**,** l**)]**

signal **=** **[**m**.**sin**(**x**)** **for** x **in** axis**]**

ort\_signal **=** ort\_signal**(**signal**)**

analitic\_signal **=** analitic\_signal**(**signal**)**

plt**.**axis**([**0**,** d**,** **-**2.5**,** 2.5**])**

plt**.**plot**(**axis**,** signal**)**

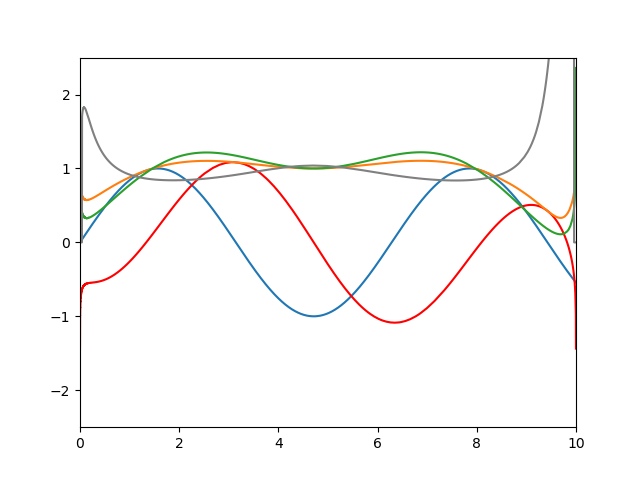
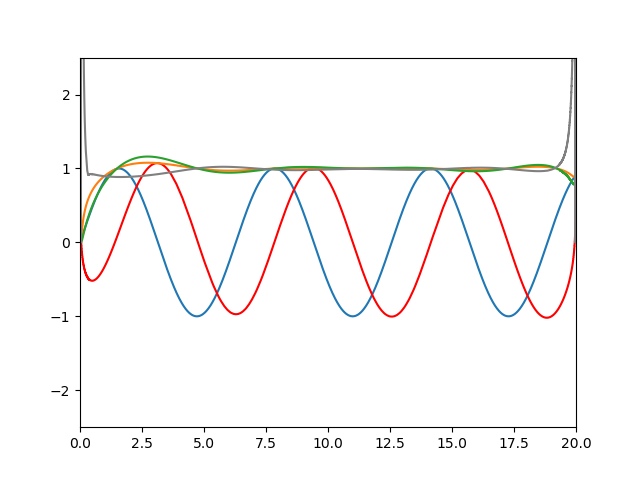
plt**.**plot**(**axis**,** ort\_signal**,** color **=** 'red'**)**

plt**.**plot**(**axis**,** amp**(**analitic\_signal**))**

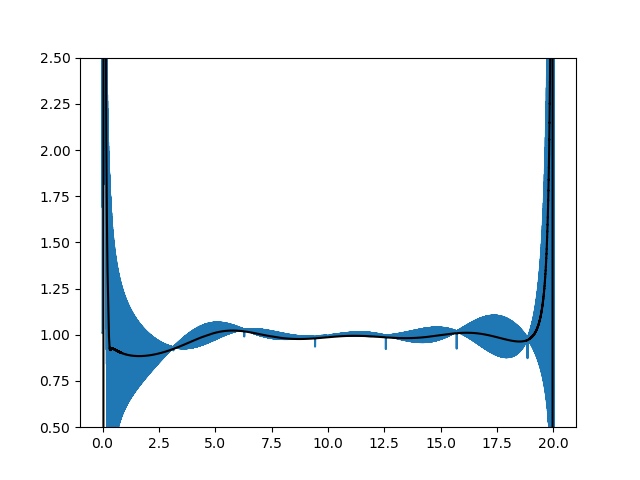
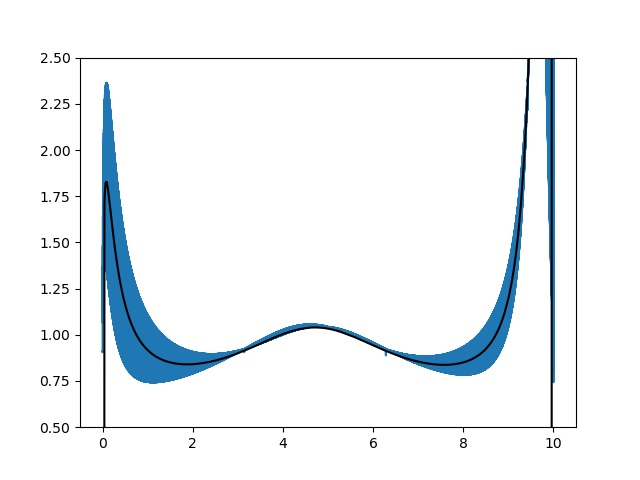
plt**.**plot**(**axis**,** cap**(**analitic\_signal**))**

plt**.**plot**(**axis**,** smoothing**(** frq**(**analitic\_signal**,** discr**),** 50**),** color **=** 'grey'**)**

plt**.**show**()**

The result of the program can be seen in the following figures:

*Figure 6 - 7. The result of the program. Blue - input valid signal; Red -* conjugate signal by Hilbert transform method*; Orange - amplitude; Green - power; Gray – frequency.*

 In particular, as mentioned earlier, sometimes when the signal is noisy, it becomes necessary to smooth it, which is demonstrated below:

*Fig. 8 - 9. An example of the smoothing function. Blue - noisy signal; Black - anti-aliased.*

**REFERENCES**

[1] https://ru.dsplib.org/content/hilbert/hilbert.html

[2] Faleev SP Calculation and modeling of signal processing devices of control systems. Tutorial. / Leningrad Order of Lenin Electrotechnical Institute named after V.I.Ulyanov (Lenin)

[3] Sato Yukio. Digital signal processing. / Yukio Sato: trans. with jap. Selina T. G. M.: Dodeka-XXI, 2010 .-- 176 p. : ill. - Add. Titus. l. jap. - ISBN 978-5-94120-251-5.