dsplab Documentation

Release 0.39.0

Aleksandr Popov, Kirill Butin

Jan 27, 2021

Contents

Digital signal processing tools

CHAPTER 1

Requirements

- numpy
- scipy

CHAPTER 2

Modules

2.1 DSP procedures

2.1.1 modulation

Examples

Harmonic

```
"""Harmonics with noise."""
import os
import sys
import random
import matplotlib.pyplot as plt

sys.path.insert(0, os.path.abspath('.'))
from dsplab.modulation import harm


def noise(t):
    """Return random value."""
    return random.normalvariate(0, 0.1)


def main():
    """Run example."""
    T = 10
    fs = 100
    x, t = harm(
        length=T, sample_rate=fs,
        amp=1, freq=1,
    )
```

(continues on next page)

(continued from previous page)

```
    plt.plot(t, x)
    x, t = harm(
        length=T, sample_rate=fs,
        amp=2, freq=1,
        noise_a=noise
    )
    plt.plot(t, x)
    x, t = harm(
        length=T, sample_rate=fs,
        amp=2, freq=1,
        noise_f=noise
    )
    plt.plot(t, x)
    plt.show()


if __name__ == "__main__":
    main()
```

Amplitude

```
"""Example of amplitude modulation."""
import os
import sys
import matplotlib.pyplot as plt

sys.path.insert(0, os.path.abspath('.'))
from dsplab.modulation import amp_mod


def modulator(t):
    """Return amplitude value."""
    if t < 5:
        return 1
    return 2


def main():
    """Run example."""
    T = 10
    fs = 100
    f = 1
    x, t = amp_mod(
        length=T,
        sample_rate=fs,
        freq=f,
        func=modulator,
    )
    plt.plot(t, x)
    plt.show()


if __name__ == "__main__":
    main()
```

Frequency

```
"""Example of frequency modulation."""
import os
import sys
import matplotlib.pyplot as plt

sys.path.insert(0, os.path.abspath('.'))
from dsplab.modulation import freq_mod


def modulator(t):
    """Return frequency value."""
    return 0.05*t + 0.5


def main():
    """Run example."""
    T = 10
    fs = 100
    amp = 1
    xs, phases, ts = freq_mod(
        length=T,
        sample_rate=fs,
        amp=amp,
        func=modulator,
    )
    plt.subplot(211)
    plt.plot(ts, xs)
    plt.subplot(212)
    plt.plot(ts, phases)
    plt.show()


if __name__ == "__main__":
    main()
```

Members

Functions for modulation and demodulation.

dsplab.modulation.**amp_mod**(*length*, *sample_rate*, *func*, *freq*, *phi=0*, *noise_f=None*, *noise_a=None*)

Amplitude modulation.

Parameters

- **length** (*float*) – Length pf signal (sec).
- **sample_rate** (*float*) – Sampling frequency (Hz).
- **freq** (*float*) – Frequency of signal (Hz).
- **phi** (*float*) – Initial phase (radians).
- **func** (*Object*) – Function that returns amplitude value depending on time.
- **noise_f** (*Object*) – Function that returns noise value added to frequency.
- **noise_a** (*Object*) – Function that returns noise value added to amplitude.

Returns

- *np.array* – Signal values.
- *np.array* – Time values.

dsplab.modulation.**digital_hilbert_filter**(*ntaps=101*, *window='hamming'*)
Calculate digital hilbert filter.

Parameters

- **ntaps** (*integer*) – Length of filter.
- **window** (*str*) – Window. Default is 'hamming'.

Returns Filter.

Return type np.array

dsplab.modulation.**envelope_by_extremums**(*xdata*, *sample_rate=1*, *tdata=None*)
Calculate envelope by local extremums of signals.

Parameters

- **xdata** (*array_like*) – Signal values.
- **sample_rate** (*float*) – Sampling frequency.
- **tdata** (*array_like*) – Time values. Use it for unregular discretized input signal.

Returns

- *np.array* – Damping values.
- *np.array* – Time values.

dsplab.modulation.**freq_amp_mod**(*length*, *sample_rate*, *a_func*, *f_func*, *phi=0*)
Simultaneous frequency and amplitude modulation.

Parameters

- **length** (*float*) – Length pf signal (sec).
- **sample_rate** (*float*) – Sampling frequency (Hz).
- **a_func** (*Object*) – Function that returns amplitude value depending on time.
- **f_func** (*Object*) – Function that returns frequency values (in Hz) depending on time.
- **phi** (*float*) – Initial phase (radians).

Returns

- *np.array* – Signal values.
- *np.array* – Full phase values.
- *np.array* – Time values.

dsplab.modulation.**freq_by_extremums**(*xdata*, *sample_rate*)
Calculate frequency of oscillating signal by extremums.

Parameters

- **xdata** (*array_like*) – Values of input signals.
- **sample_rate** (*float*) – Sampling frequency (Hz).

Returns Frequency.

Return type float

dsplab.modulation.**freq_by_zeros**(*xdata*, *sample_rate*)
Calculate average frequency of detrended oscillating signal by counting zeros.

dsplab.modulation.**freq_mod**(*length*, *sample_rate*, *amp*, *func*, *phi=0*, *noise_f=None*, *noise_a=None*)
Amplitude modulation.

Parameters

- **length** (*float*) – Length pf signal (sec).
- **sample_rate** (*float*) – Sampling frequency (Hz).
- **amp** (*float*) – Amplitude of signal.
- **phi** (*float*) – Initial phase (radians).
- **func** (*Object*) – Function that returns frequency values (in Hz) depending on time.
- **noise_f** (*Object*) – Function that returns noise value added to frequency.
- **noise_a** (*Object*) – Function that returns noise value added to amplitude.

Returns

- *np.array* – Signal values.
- *np.array* – Full phase values.
- *np.array* – Time values.

dsplab.modulation.**freqs_by_wave_len**(*xdata*, *tdata*, *cut_nans=True*)
Calculate frequencies using lenghs of waves and linear interpolation.

Parameters

- **xdata** (*np.ndarray*) – Signal values.
- **tdata** (*np.ndarray*) – Time values.
- **cut_nans** (*boolean*) – If True, the nan values at the ends of the of the produced array will removed.

Returns Freqs values.

Return type np.ndarray

dsplab.modulation.**harm**(*length*, *sample_rate*, *amp*, *freq*, *phi=0*, *noise_a=None*, *noise_f=None*)
Generate harmonic signal.

Parameters

- **length** (*float*) – Length pf signal (sec).
- **sample_rate** (*float*) – Sampling frequency (Hz).
- **amp** (*float*) – Amplitude of signal.
- **freq** (*Object*) – Frequency of signal (Hz).
- **phi** (*float*) – Initial phase (radians).
- **noise_a** (*callable*) – Returns noise value added to amplitude.
- **noise_f** (*callable*) – Returns noise value added to frequency.

Returns

- *np.array* – Signal values.

- *np.array* – Time values.

dsplab.modulation.**iq_demod**(*xdata*, *tdata*, *f_central*, *a_coeffs*, *b_coeffs*)
Return instantaneous frequency of modulated signal using IQ processign.

Parameters

- **xdata** (*array_like*) – Signal values.
- **tdata** (*array_like*) – Time values.
- **f_central** (*float*) – Carrier frequency.
- **a_coeffs** (*array_like*) – a values of filter.
- **b_coeffs** (*array_like*) – b values of filter.

Returns

- *np.ndarray of floats* – Instantaneous frequency values.
- *np.ndarray* – Time values.

dsplab.modulation.**linint**(*xdata*, *tdata*, *ts_new*)
Find values of xdata in ts_new points.

Parameters

- **xdata** (*np.ndarray*) – Signal values.
- **tdata** (*np.ndarray*) – Time values.
- **ts_new** (*np.ndarray*) – New time values.

Returns New signal values.

Return type np.ndarray

dsplab.modulation.**phase_mod**(*length*, *sample_rate*, *amp*, *freq*, *func*, *noise_f=None*, *noise_a=None*)
Phase modulation.

Parameters

- **length** (*float*) – Length pf signal (sec).
- **sample_rate** (*float*) – Sampling frequency (Hz).
- **amp** (*float*) – Amplitude of signal.
- **freq** (*float*) – Frequency of signal (Hz).
- **func** (*Object*) – Function that returns phase values (in radians) depending on time.
- **noise_f** (*Object*) – Function that returns noise value added to frequency.
- **noise_a** (*Object*) – Function that returns noise value added to amplitude.

Returns

- *np.array* – Signal values.
- *np.array* – Time values.

dsplab.modulation.**wave_lens**(*xdata*, *tdata*)
Calculate wave lengths of signal by space between zeros.

Parameters

- **xdata** (*np.ndarray*) – Signal values.

- **tdata** (*np.ndarray*) – Time values.

Returns

- *np.ndarray* – Wave lengths.
- *np.ndarray* – Time values.

2.1.2 filtration

Filtration of signals.

dsplab.filtration.**butter_filter**(*xdata*, *sample_rate*, *freqs*, *order*, *btype='band'*)

Butterworth filter.

Parameters

- **xdata** (*array_like*) – Signal values.
- **sample_rate** (*float*) – Sampling frequency (Hz).
- **freqs** (*array_like*) – One or two frequencies.
- **order** (*integer*) – Order of filter.
- **btype** (*str ('band' | 'lowpass')*) – Type of filter.

Returns filtered signal.

Return type np.array

dsplab.filtration.**find_butt_bandpass_order**(*band*, *sample_rate*)

Calculate the order of Butterworth bandpass filter using minimization of metric between ideal and real frequency response.

Parameters

- **band** (*array_like*) – Pair of frequencies. Bounds of bandpass (Hz).
- **sample_rate** (*float*) – Sample rate (Hz).

Returns Order of filter.

Return type integer

dsplab.filtration.**haar_one_step**(*xdata*, *tdata*, *denominator=2*)

One cascade of Haar transform.

Parameters

- **xdata** (*array_like*) – Signal values.
- **tdata** (*array_like*) – Time values.
- **denominator** (*integer*) – Denominator used in Haar transform (default is 2).

Returns

- *np.array* – Scaled signal values.
- *np.array.* – Details of x
- *np.array.* – Decimated time values

dsplab.filtration.**haar_scaling**(*xdata*, *tdata*, *steps_number*)

Scaling with Haar transform.

Parameters

- **xdata** (*array_like*) – Signal values.
- **tdata** (*array_like*) – Time values.
- **steps_number** (*integer*) – Number of cascades.

Returns

- *np.array* – Scaled signal values.
- *np.array* – Decimated time values.

dsplab.filtration.**smooth**(*xdata*, *ntaps=3*, *cut=True*)
Smooth signal with Hamming window.

dsplab.filtration.**stupid_bandpass_filter**(*xdata*, *sample_rate*, *bandpass*)
Return low-pass filtered signal.

Parameters

- **xdata** (*array_like*) – Signal values.
- **sample_rate** (*float*) – Sampling frequency.
- **bandpass** (*np.array of 2 floats*) – Bounds of bandpass (Hz).

Returns Filteres signal.

Return type np.array

dsplab.filtration.**stupid_lowpass_filter**(*xdata*, *sample_rate*, *cutoff*)
Return low-pass filtered signal.

Parameters

- **xdata** (*array_like*) – Signal values.
- **sample_rate** (*float*) – Sampling frequency.
- **cutoff** (*float*) – Cutoff frequency.

Returns Filteres signal.

Return type np.array

dsplab.filtration.**trend_smooth**(*xdata*, *sample_rate=1*, *tdata=None*, *cut_off=0.5*)
Calculate trend of signal using smoothing filter.

Parameters

- **xdata** (*array_like*) – Signal values.
- **tdata** (*array_like*) – Time values.
- **cut_off** (*float*) – The frequencies lower than this are trend's frequencies.

Returns

- *np.array* – Trend values.
- *np.array* – Time values.

2.1.3 prony

This module implements Prony decomposition of signal.

dsplab.prony.**prony_decomp**(*xdata*, *ncomp*)
Prony decomposition of signal.

Parameters

- **xdata** (*array_like*) – Signal values.
- **ncomp** (*integer*) – Number of components. 2*ncomp must be less tham length of xdata.

Returns

- *np.array* – Mu-values.
- *np.array* – C-values.
- *np.array* – Components.

2.1.4 spectran

Some functions for spectral analysis.

dsplab.spectran.**calc_specgram**(*xdata*, *sample_rate=1*, *tdata=None*, *nseg=256*, *nstep=None*, *freq_bounds=None*, *extra_len=None*)

Return spectrogram data prepared to further plotting.

Parameters

- **xdata** (*array_like*) – Signal values
- **sample_rate** (*float*) – Sampling frequency (Hz)
- **tdata** (*array_like*) – Time values (sec)
- **nseg** (*integer*) – Length of window (number of samples)
- **nstep** (*integer*) – Length of step between Fourier transforms
- **freq_bounds** (*tuple of 2 float*) – Bounds of showed band
- **extra_len** (*integer*) – Number of values using for fft

Returns

- *np.ndarray* – Array of spectrums
- *np.ndarray* – Time values

dsplab.spectran.**spectrum**(*xdata*, *sample_rate=1*, *window='hamming'*, *one_side=False*, *return_amplitude=True*, *extra_len=None*, *save_energy=False*)

Return the Fourier spectrum of signal.

Parameters

- **xdata** (*array_like*) – Signal values
- **sample_rate** (*float*) – Sampling frequency (Hz)
- **window** (*str*) – Window.
- **one_side** (*boolean*) – If True, the one-side spectrum is calculated (default value is False)
- **return_amplitude** (*boolean*) – If True, the amplitude spectrum is calculated
- **extra_len** (*int*) – If the value is set, the signal is padded with zeros to the extra_len value.
- **save_energy** (*boolean*) – If True, the result of FFT has the same energy as signal. If False, the X (spectrum) is multiplied to 2/len(xdata). Use False if you want to see the correct amplitude of components in spectrum.

Returns

- *np.ndarray of complex numbers* – Spectrum
- *np.ndarray of floats* – Frequency values (Hz)

dsplab.spectran.**stft**(*xdata*, *sample_rate=1*, *nseg=256*, *nstep=None*, *window='hamming'*, *nfft=None*, *padded=False*)

Return result of short-time fourier transform.

Parameters

- **xdata** (*numpy.ndarray*) – Signal.
- **sample_rate** (*float*) – Sampling frequency (Hz).
- **nseg** (*int*) – Length of segment (in samples).
- **nstep** (*int*) – Optional. Length of step (in samples). If not setted then equal to nseg//2.
- **window** (*str*) – Window.
- **nfft** (*int*) – Length of the FFT. If None or less than nseg, the FFT length is nseg.

Returns Result of STFT, two-side spectrums.

Return type numpy.ndarray

2.2 Organization of calculations

2.2.1 activity

This module implements the base classes for Activities.

class dsplab.flow.activity.**Activity**

Bases: object

Any activity is the something that may be called and can provide the information about itself. To get working activity the __call__ method must be implemented.

info(*as_string=None*)

Deprecated.

set_descr(*descr*)

Deprecated.

class dsplab.flow.activity.**ActivityMeta**(*name*, *bases*, *attrs*)

Bases: type

Metaclass for Activity.

class_info()

Return the information about activity.

Returns Information about class of activity.

Return type dict

class dsplab.flow.activity.**Work**(*descr=None*, *worker=None*)

Bases: *dsplab.flow.activity.Activity*

Work is data processing that can be done in a variety of ways.

descr
Description of work

get_descr()
Return description.

set_descr(*descr*)
Set description.

set_worker(*act*)
Set worker for doing work. Worker must be callable.

class dsplab.flow.activity.**Worker**
Bases: *dsplab.flow.activity.Activity*

Deprecated.

add_param(*name*, *value=None*)
Deprecated.

dsplab.flow.activity.**get_work_from_dict**(*settings*, *params=None*)
Create and return Work instance described in dictionary.

2.2.2 online

This module implements the base class for online filters.

class dsplab.flow.online.**And**
Bases: *dsplab.flow.activity.Activity*

And operation.

class dsplab.flow.online.**Delayer**(*ntaps*, *fill_with=0*)
Bases: *dsplab.flow.online.QueueFilter*

Provide delay in online processing.

proc_queue()
Process queue.

class dsplab.flow.online.**OnlineFilter**(*ntaps=None*, *smooth_ntaps=None*, *fill_with=0*, *step=1*)
Bases: *dsplab.flow.activity.Activity*

Universal online filter.

Parameters

- **ntaps** (*int*) – Length of internal queue using for accumulation of input samples. Default is None.
- **smooth_ntaps** (*int*) – Length of queue using for smoothing output values. Default id None.
- **fill_with** (*object*) – Initial value of every element of queues.
- **step** (*int*) – Step. Must be positive.

proc_queue()
Process queue.

Returns Ouput value.

Return type object

proc_sample(*sample*)
Process sample.

Parameters **sample** (*object*) – Input sample.

Returns Output value.

Return type object

class dsplab.flow.online.**Or**
Bases: *dsplab.flow.activity.Activity*

Or operation.

class dsplab.flow.online.**QueueFilter**(*ntaps*, *fill_with=0*)
Bases: *dsplab.flow.activity.Activity*

Online filter with queue.

Parameters

- **ntaps** (*int*) – Lenght of filter.
- **fill_with** (*object*) – Initial value of every element of queue.

proc_queue()
Process queue.

dsplab.flow.online.**unwrap_point**(*phi*)
Unwrap angle (for signle value).

2.2.3 plan

Examples

Helper module with workers

```
"""Workers for examples."""
import os
import sys
sys.path.insert(0, os.path.abspath('.'))
from dsplab.flow.activity import Activity


class Linear(Activity):
    """Linear transformation: y = k*x + b."""
    def __init__(self, k, b):
        super().__init__()
        self.k = k
        self.b = b

    def __call__(self, x):
        y = x*self.k + self.b
        return y


class Sum(Activity):
    """Sum."""
    def __call__(self, *xs):
```

(continues on next page)

(continued from previous page)

```
        y = sum(xs)
        return y


class Inc(Activity):
    """Add 1 to value."""
    def __init__(self):
        super().__init__()

    def __call__(self, x):
        y = x + 1
        return y


class DoNothing(Activity):
    """Just pass input to output."""
    def __init__(self):
        super().__init__()

    def __call__(self, x):
        return x
```

Basic usage

```
"""Basic usage of plan."""
import os
import sys
sys.path.insert(0, os.path.abspath('.'))
from dsplab.flow.activity import Work
from dsplab.flow.plan import WorkNode, Plan
from workers import Linear


def main():
    """Run example."""
    plan = Plan()
    node_a = WorkNode(work=Work("Linear transformation", worker=Linear(1, 1)))
    node_b = WorkNode(work=Work("Linear transformation", worker=Linear(2, 2)))
    node_c = WorkNode(work=Work("Linear transformation", worker=Linear(3, 3)))
    plan.add_node(node_a)
    plan.add_node(node_b, inputs=[node_a])
    plan.add_node(node_c, inputs=[node_b])
    plan.inputs = [node_a]
    plan.outputs = [node_c, node_b]

    print(plan([5]))


if __name__ == "__main__":
    main()
```

Using of start and stop hooks

```
"""Start and stop hooks."""
import os
import sys
sys.path.insert(0, os.path.abspath('.'))
from dsplab.flow.activity import Work
from dsplab.flow.plan import WorkNode, Plan


def func(x):
    """Worker."""
    return x + 1


def start_handler(node):
    """Node start handler."""
    print("'{}' started".format(node.work.descr))


def stop_handler(node):
    """Node stop handler."""
    print("'{}' finished".format(node.work.descr))


def progress_handler():
    """Progress handler."""
    print("Calculated one node.")


def main():
    """Entry point."""
    print(__doc__)
    node = WorkNode(work=Work("Increment", worker=func))
    node.set_start_hook(start_handler, node)
    node.set_stop_hook(stop_handler, node)
    plan = Plan()
    plan.add_node(node)
    plan.set_progress_hook(progress_handler)
    plan.inputs = [node]
    plan.outputs = [node]
    plan([5])


if __name__ == "__main__":
    main()
```

MapNode (applying work for iterable input)

```
"""Mapping."""
import os
import sys
sys.path.insert(0, os.path.abspath('.'))
from dsplab.flow.activity import Work
from dsplab.flow.plan import MapNode, WorkNode, Plan
```

(continues on next page)

(continued from previous page)

```
from workers import Sum, DoNothing


def main():
    """Run example."""
    plan = Plan()

    pass_node_1 = WorkNode(
        Work("Pass", worker=DoNothing())
    )
    pass_node_2 = WorkNode(
        Work("Pass", worker=DoNothing())
    )

    map_node = MapNode(
        work=Work("Transformation", worker=Sum()),
        inputs=[pass_node_1, pass_node_2]
    )

    plan.add_node(pass_node_1)
    plan.add_node(pass_node_2)
    plan.add_node(map_node)
    plan.inputs = [pass_node_1, pass_node_2]
    plan.outputs = [map_node]

    res = plan([
        [1, 1, 1],
        [2, 2, 2],
    ])
    print("Outputs:", res)


if __name__ == "__main__":
    main()
```

PackNode (pack inputs to list)

```
"""Pack inputs to list."""
import os
import sys
sys.path.insert(0, os.path.abspath('.'))
from dsplab.flow.activity import Work
from dsplab.flow.plan import WorkNode, PackNode, Plan
from workers import DoNothing


def main():
    """Run example."""
    print(__doc__)
    plan = Plan()
    node_1 = WorkNode(Work("Pass", worker=DoNothing()))
    node_2 = WorkNode(Work("Pass", worker=DoNothing()))
    node_3 = WorkNode(Work("Pass", worker=DoNothing()))
    pack_node_1 = PackNode()
```

(continues on next page)

(continued from previous page)

```
    pack_node_2 = PackNode()

    plan.add_node(node_1)
    plan.add_node(node_2)
    plan.add_node(node_3)
    plan.add_node(pack_node_1, inputs=[node_1, node_2])
    plan.add_node(pack_node_2, inputs=[node_2, node_3])

    plan.inputs = [node_1, node_2, node_3]
    plan.outputs = [pack_node_1, pack_node_2]

    print(plan([1, 2, 3]))


if __name__ == "__main__":
    main()
```

SelectNode

```
"""Using of SelectNode with multiple input."""
import os
import sys
sys.path.insert(0, os.path.abspath('.'))
from dsplab.flow.activity import Work
from dsplab.flow.plan import SelectNode, WorkNode, Plan
from workers import DoNothing


def main():
    """Run example."""
    plan = Plan()

    pass_node_1 = WorkNode(Work(descr="Pass", worker=DoNothing()))
    pass_node_2 = WorkNode(Work(descr="Pass", worker=DoNothing()))
    select_node_m = SelectNode(index=0)
    select_node_s = SelectNode(index=0)

    plan.add_node(pass_node_1)
    plan.add_node(pass_node_2)
    plan.add_node(select_node_m,
                  inputs=[pass_node_1, pass_node_2])
    plan.add_node(select_node_s,
                  inputs=[pass_node_1])
    plan.inputs = [pass_node_1, pass_node_2]
    plan.outputs = [select_node_m, select_node_s]

    res = plan([
        [1, 2, 3],
        [2, 3, 4]
    ])
    print("Outputs: ", res)


if __name__ == "__main__":
    main()
```

Node-generator

'Node-generator' means the no input node with no inputs.

```
"""Node may not have inputs."""
import os
import sys
from random import randint

sys.path.insert(0, os.path.abspath('.'))
from dsplab.flow.activity import Work
from dsplab.flow.plan import WorkNode, Plan


def gen():
    """Generate random number."""
    y = randint(1, 10)
    print("gen -> {}".format(y))
    return y


def inc(x):
    """Increment."""
    y = x + 1
    print("{} -> inc -> {}".format(x, y))
    return y


def plus(x1, x2):
    """Sum of two numbers."""
    y = x1 + x2
    print("{}, {} -> plus -> {}".format(x1, x2, y))
    return y


def main():
    """Run example."""
    p = Plan()
    g = WorkNode(Work("Generate random number", gen))
    a = WorkNode(Work("Add 1", inc))
    b = WorkNode(Work("Summation", plus))
    p.add_node(g)
    p.add_node(a)
    p.add_node(b, inputs=[g, a])
    p.inputs = [a]
    p.outputs = [b]

    x = [1]
    print(x)
    y = p(x)
    print(y)


if __name__ == "__main__":
    main()
```

Get Plan instance from dict

```
"""Get plan from dictionary."""
import os
import sys
sys.path.insert(0, os.path.abspath('.'))
from dsplab.flow.plan import get_plan_from_dict


SETTINGS = {
    'nodes': [
        {
            'id': 'a',
            'class': 'WorkNode',
            'work': {
                'descr': "First step",
                'worker': {
                    'class': "workers.Linear",
                    'params': {
                        'k': 1,
                        'b': 1,
                    }
                }
            }
        },

        {
            'id': 'b',
            'class': 'WorkNode',
            'work': {
                'descr': "Second step",
                'worker': {
                    'function': "numpy.exp"
                }
            },
            'inputs': ['a'],
        },

        {
            'id': 'c',
            'class': 'WorkNode',
            'work': {
                'descr': "Third step",
                'worker': {
                    'class': "workers.Inc"
                }
            },
            'inputs': ['b'],
        },

        {
            'id': 'd',
            'class': 'PackNode',
            'inputs': ['b', 'c'],
            'result': 'Result value'
        }
    ],
```

(continues on next page)

(continued from previous page)

```
    'inputs': ['a'],
    'outputs': ['d'],
}


def main():
    """Run example."""
    plan = get_plan_from_dict(SETTINGS)
    x = 1
    y = plan([x])
    print(y)


if __name__ == "__main__":
    main()
```

Quick plan for on-line processing

```
"""Online plan."""
import os
import sys
sys.path.insert(0, os.path.abspath('.'))
from dsplab.flow.plan import Plan, WorkNode
from dsplab.flow.activity import Work
from workers import Inc


def main():
    """Run example."""
    node_1 = WorkNode(work=Work("Step 1", worker=Inc()))
    node_2 = WorkNode(work=Work("Step 2", worker=Inc()))
    node_3 = WorkNode(work=Work("Step 3", worker=Inc()))
    plan = Plan(quick=True)
    plan.add_node(node_1)
    plan.add_node(node_2, inputs=[node_1])
    plan.add_node(node_3, inputs=[node_2])
    plan.inputs = [node_1]
    plan.outputs = [node_3]

    plan.reduce_calls()
    xs = [1, 2, 3, 4, 5]
    for x in xs:
        y = plan([x])[0]
        print("{} -> {}".format(x, y))


if __name__ == "__main__":
    main()
```

Members

Verification of plan

2.3 Online processing

2.3.1 player

Examples

Random numbers

```
"""Example of playing random signal."""
import os
import sys
sys.path.insert(0, os.path.abspath('.'))
from dsplab.player import RandomDataProducer, SignalPlayer


def main():
    """Entry point."""
    data_producer = RandomDataProducer(interval=(1, 100))
    player = SignalPlayer(interval=0.02)
    player.set_data_producer(data_producer)
    player.start()
    while True:
        try:
            sample = player.get_sample()
            print(sample)
        except KeyboardInterrupt:
            break
    player.stop()


if __name__ == "__main__":
    main()
```

CSV-file

```
"""Example of playing signal from CSV file."""
import os
import sys
sys.path.insert(0, os.path.abspath('.'))
from dsplab.player import CsvDataProducer, SignalPlayer


def main():
    """Play CSV file."""
    data_producer = CsvDataProducer(file_name="demo/player/test.csv", delimiter=',')
    data_producer.select_columns(['F', 'Ua1'])
    # data_producer.select_columns([0, 1]) # <--- It's all right too
    player = SignalPlayer(interval=0.02)
    player.set_data_producer(data_producer)
    player.start()
    while True:
```

(continues on next page)

(continued from previous page)

```
        try:
            sample = player.get_sample()
            print(sample)
        except KeyboardInterrupt:
            break
    player.stop()


if __name__ == "__main__":
    main()
```

Members

Playing signal from file. Online mode for offline data.

class dsplab.player.**CsvDataProducer**(*file_name=None*, *delimiter=';'*, *encoding='utf-8'*, *columns=None*)

Bases: *dsplab.player.DataProducer*

Produces sample from headered CSV file.

delimiter
delimiter in CSV file.

get_delimiter()
Return delimiter.

get_sample()
Return sample.

select_columns(*keys*)
Select returned columns. Numbers or names of columns can be used.

set_delimiter(*delimiter*)
Set delimiter.

set_file(*file_name*, *delimiter=None*, *encoding='utf-8'*)
Set file for reading.

start()
Init reader.

class dsplab.player.**DataProducer**

Bases: object

Base class for adapters for data producer.

get_sample()
Return sample.

start()
Do some operations in producer when player starts.

stop()
Do some operations in producer when player stops.

class dsplab.player.**RandomDataProducer**(*interval*)

Bases: *dsplab.player.DataProducer*

Data producer with random values on output.

get_sample()
Return sample.

class dsplab.player.**RepeatedTimer**(*interval*, *function*, **args*, ***kwargs*)
Bases: object

Timer.

get_interval()
Get interval.

interval
Timeout interval (sec)

set_interval(*interval*)
Set interval.

start()
Start timer.

stop()
Stop timer.

class dsplab.player.**SignalPlayer**(*interval*)
Bases: object

Class for playing text file as stream.

get_sample()
Return sample.

set_data_producer(*data_producer*)
Set adapter with get_sample() method.

start()
Start player.

stop()
Stop player.

CHAPTER 3

Project

3.1 History

3.1.1 0.39

- Plan, activity, node and online were replaced to new subpackage **flow**
- Verification of plan is implemented

3.1.2 0.38

- Activity methods **info()** and **set_descr()** have been marked as deprecated
- **Worker** class has been marked as deprecated
- New Node's method **get_result_info()**
- New Plan's property **descr**
- **reset()** method of the Node has been renamed to **clear_result()**

3.1.3 0.37

- Possibility to use parameters as "$name" in JSON with plan
- New method of Plan: set_quick()
- Examples have been brushed (style and flakes)

3.1.4 0.36

- Lightweight QueueFilter class has been added.

- Function for unwrapping point has been updated.
- Online delayer has been added.
- Online logic classes have been refactored.

3.1.5 0.35

- PassNode has been added.
- Deprecated stuff in player has been deleted.

3.1.6 0.34

- The possibility of describing of the resulting data of node was added.
- The code in info() was refactored.
- Examples of using the Plan were updated.

3.1.7 0.33

- Adjust Plan for quick on-line execution
- Add verify() method to Plan
- Add clear() method to Plan
- Update examples of Plan

3.1.8 0.32

- Add new types of nodes: WorkNode, MapNode, PackNode and SelectNode
- Update get_plan_from_dict() to support all types of nodes
- Update examples
- Add get_nodes() to Plan
- Support progress handling in plan
- Add arguments to hooks of nodes
- Refactoring of plan.py

3.1.9 0.31

- Add smooth function to filtration
- Brush filtration module
- Merge frequency unit to modulation one
- Split activity unit to activity and online
- Refactoring trend_smooth()
- Refactoring find_butt_bandpass_order()

- Refactoring haar_one_step()
- Refactoring haar_scaling()
- Unify arguments names

3.1.10 0.30

- Add descr option to __init__ of Plan
- Add function for simultaneous amp and freq modulation
- Brush modulation module
- Rename functions names and arguments in modulation unit
- Set default phi=0 in modulation
- Merge envelop module to modulation one
- Update envelop examples
- Update modulation examples
- Refactoring iq_demod()

3.1.11 0.29

- Add worker subclass of activity class
- Update examples

3.1.12 0.28

- Add class info of Activity in metaclass
- Add method for getting information about class of Activity without creating instance
- Add method for adding and documenting parameters of activity
- Brush activity module with pylint
- Some fixes

3.1.13 0.27

- Add tool for playing of archive data as if they are being get in online mode.
- Add adapter for producing data for player.
- Add CSV data producer.
- Add Random data producer.
- Add examples of playing signals.

3.1.14 0.26

- Modulation: Return angles from fm()
- Activity: Add a function for setup work from dictionary
- Plan: Add a function for setup plan from dictionary
- Plan: Deprecate setup_plan()
- Plan: Update demo
- Modulation: Add a function for generate the harmonic signal (with constant amplitude, frequency and phase)

3.1.15 0.25

- Modulation: Add noise_a and noise_f parameters to am, fm, phm
- Plan: Add the key 'function' to explicit description of worker in node settings
- Plan: Add a worker with no init args to the example of setup_plan
- Docs: Add more examples
- Some bugs fixed

3.1.16 0.24

- Plan: Provide auto and manual terminals without auto_terminals option
- Plan: Support the inputs and outputs in the function for setup plan from dict
- Plan: Rename Translator to Transmitter
- Modulation: Add a function for phase modulation

3.1.17 0.23

- Activity: Remove Strategy and subclasses
- Activity: Use the docstring for description in _info
- Plan: Add docstrings to Plan.outputs property
- Plan: Add remove node method and demo
- Plan: Remove detection of terminals from call
- Plan: Add auto_terminals option to init
- Demo: Replace plan examples to plan/ folder from activity/
- Add link to docs to README

3.1.18 0.22

- Add function am to modulation unit
- Add function fm to modulation unit
- Add demo for am

- Add demo for fm

3.1.19 0.21

- The possibility of specifying outputs is supported.
- The Translator node is added for constructing more flexible input of plan.
- More examples of using plans are added.
- The hooks for starting and finishing calculations in node are added.
- A small refactoring is performed.

3.1.20 0.20

- The function for setup plan. The settings are taken from list of dictionaries.
- Refactoring.

3.1.21 0.19

- Activity module redesigned.
- Info stuff of activities redesigned.
- Work class added. Work is the activity that can be done by different ways. Work has worker. Worker is the activity.
- Added tools for constructing the plans of works. Plan is the number of linked nodes and every node is the 'work place' for some worker.

3.1.22 0.18

- The module activity containing base classes for different processing tools added.

3.1.23 0.17

- The base class for online filters was added

3.1.24 0.16

- Add digital_hilbert_filter function to envelope and deprecate hilbert_filter
- Add example for IQ demodulation

3.1.25 0.15

- More universal function for IQ-processing was added.

3.1.26 0.14

- Window parameter was added to spectrum and stft.
- Some code in spectran enhanced.

3.1.27 0.13

- Function for calculation of frequency using wave lengths was added.
- Fixed errors in spectrogram calculation.

3.1.28 0.12

- Function for calculation of instantaneous frequency with phasor was added to new module called modulation.
- Function for calculation of spectrogram was added.
- Function for finding the trend with smoothing filtration was added.
- Stupid filters (FFT and back) were added.
- Spectrum function was rewritten.
- Some code was cleaned.
- More tests were added.

3.1.29 0.11

- Function for calculation of order of Butterworth bandpass filter was added.
- Some docs were added.

3.1.30 0.10

- Tools for spectral analysis were added
- Haar transform was added
- More demo were added
- Some bugs were fixed

3.1.31 0.9

- Function for calculation digital Hilbert filter was added
- Demo for digital Hilbert filter was added

3.1.32 0.8

- Specific module damping was removed
- Function for read signal from CSV was added
- More tests were added

3.1.33 0.7

- Envelope by maximums replaced to envelope by extremums.
- Demo added.
- More tests added.

3.1.34 0.6

- Prony's decomposition of signal is added.

3.1.35 0.5

- Stupid procedure for calculation damping time is added.

3.2 Demo

If dsplab is not installed you can go to dsplab root folder and run examples such way:

```
path/to/dsplab$ python3 demo/prony/demo.prony
```

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

d

Index