syncsweptsine

Release 0.2.0

Siegfried Gündert

Jun 12, 2023

CONTENTS:

1 Classes 3

2 Examples 5

3 Indices and tables 23

Python Module Index 25

Index 27

This module implements the Synchronized Swept Sine Method according to Nowak et al. 2015 as reusable python module.

It can be used for the system identification of linear and nonlinear systems. The identification results can be represented as Hammerstein models (Diagonal Volterra Series). Furthermore simple regularization is provided as optional feature.

CHAPTER ONE

CLASSES

High level classes:

- *SyncSweep*: defines the synchronized sweep model
- *HigherHarmonicImpulseResponse*: defines the Higher harmonic impulse response e.g. by deconvolution of the reference SyncSweep instance and the actual measured sweep signal at the output of the system under test.
- *HammersteinModel*: defines the generalized hammerstein model based on a list of kernels and corresponding nonlinearity orders.

Low level classes:

- *InvertedSyncSweepSpectrum*: defines the inverted spectrum of a synchronized sweep.
- *FrfFilterKernel*: defines a filter kernel based on a frequency response function.
- *IirFilterKernel*: defines a filter kernel based on IIR filter coefficients.

CHAPTER

TWO

EXAMPLES

Estimating the coefficients of a simple nonlinear system:

```
import numpy as np
from syncsweptsine import SyncSweep
from syncsweptsine import HigherHarmonicImpulseResponse
from syncsweptsine import HammersteinModel

sweep = SyncSweep(startfreq=16, stopfreq=16000, durationappr=10, samplerate=96000)

def nonlinear_system(sig):
    return 1.0 * sig + 0.25 * sig**2 + 0.125 * sig**3

outsweep = nonlinear_system(np.array(sweep))
hhir = HigherHarmonicImpulseResponse.from_sweeps(sweep, outsweep)
hm = HammersteinModel.from_higher_harmonic_impulse_response(
    hhir, 2048, orders=(1, 2, 3), delay=0)
for kernel, order in zip(hm.kernels, hm.orders):
    print('Coefficient estimate of nonlinear system:',
          np.round(np.percentile(abs(kernel.frf), 95), 3),
          'Order',
          order)

Out[7]:
Coefficient estimate of nonlinear system: 1.009 Order 1
Coefficient estimate of nonlinear system: 0.25 Order 2
Coefficient estimate of nonlinear system: 0.125 Order 3
```

Estimating the Hammerstein model of a theoretically created Hammerstein model usin IIR kernels:

```
from pylab import *

from syncsweptsine import IirFilterKernel
from syncsweptsine import HammersteinModel
from syncsweptsine import SyncSweep
from syncsweptsine import HigherHarmonicImpulseResponse


nfft = 1024
samplerate = 96000
# sweep params:
f1 = 1.2
```

(continues on next page)

(continued from previous page)

```
f2 = 16_000
dursec = 30

# Filter kernels for theoretical hammerstein model:
# the ARMA filters definition (ARMA order = 2, number of filters = N = 4)
A = [
    [1.0, -1.8996, 0.9025],
    [1.0, -1.9075, 0.9409],
    [1.0, -1.8471, 0.8649],
    ]
B = [
    [1.0, -1.9027, 0.9409],
    [1.0, -1.8959, 0.9025],
    [0.5, -0.9176, 0.4512],
    ]
orders = [1, 2, 3]
kernels_theo = [IirFilterKernel(*ba) for ba in zip(B, A)]
hm_theo = HammersteinModel(kernels_theo, orders)


# system identification of the theoretical system
sweep = SyncSweep(f1, f2, dursec, samplerate)
sweep_sig = sweep.get_windowed_signal(1024, 1024, pausestart=0, pausestop=512)
outsweep = hm_theo.filter(sweep_sig)
hhir = HigherHarmonicImpulseResponse.from_sweeps(sweep, outsweep)
hm_identified = HammersteinModel.from_higher_harmonic_impulse_response(
    hhir=hhir,
    length=nfft,
    orders=orders,
    delay=0,
)

# bode diagram of the theoretical and identification results
figure()
for theo, kernel, order in zip(hm_theo.kernels, hm_identified.kernels, orders):
    freq = kernel.freq
    G_kernel = kernel.frf
    freq_theo, G_kernel_theo = theo.freqz(nfft)

    ax = subplot(len(orders), 1, order )
    l0 = ax.semilogx(
        freq_theo/pi*samplerate/2,
        20*log10(abs(G_kernel_theo)),
        'b-',
        label=f'|H| Theor. (order={order})'
        )
    l1 = ax.semilogx(
        freq,
        20*log10(abs(G_kernel)),
        '--',
        color='skyblue',
        label=f'|H| Estimate (order={order})'
```

(continues on next page)

(continued from previous page)

```
        )
    xlim(4*f1, f2)
    ylim(-35, 35)
    ylabel('$|H|$ / dB')
    if order < max(orders): xticks([])
    grid()

    for ytlabel in ax.get_yticklabels(): ytlabel.set_color('b')

    ax2 = gca().twinx()
    ylim(-pi, pi)
    l2 = ax2.semilogx(
        freq_theo/pi*samplerate/2,
        unwrap(angle(G_kernel_theo)),
        'g-',
        label=f'$\phi$ Theor. (order={order})'
        )
    phi_theo = unwrap(angle(G_kernel*exp(-1j*freq*pi*nfft/hhir.samplerate)))
    l3 = ax2.semilogx(
        freq,
        phi_theo,
        '--',
        color='lightgreen',
        label=f'$\phi$ Estimate (order={order})'
        )
    for ytlabel in ax2.get_yticklabels(): ytlabel.set_color('g')
    ylabel('$\phi$ / rad')
    grid()
    lines = l0 + l1 + l2 + l3
    labels = [l.get_label() for l in lines]
    legend(lines, labels)
    xlabel('Frequency $f$ / Hz')
```

syncsweptsine.**hannramp**(*sig*, *left*, *right=None*)

Retruns faded signal faded with hanning flanks.

Returns

sigfaded: ndarray
Signal faded with hanning ramps.

class syncsweptsine.**SyncSweep**(*startfreq*, *stopfreq*, *durationappr*, *samplerate*)

Synchronized Swept Sine Signal Model

Parameters

startfreq
[scalar] Start frequency of sweep in Hz

stopfreq
[scalar] Stop frequency of sweep in Hz

durationappr
[scalar] Approximate duration in seconds

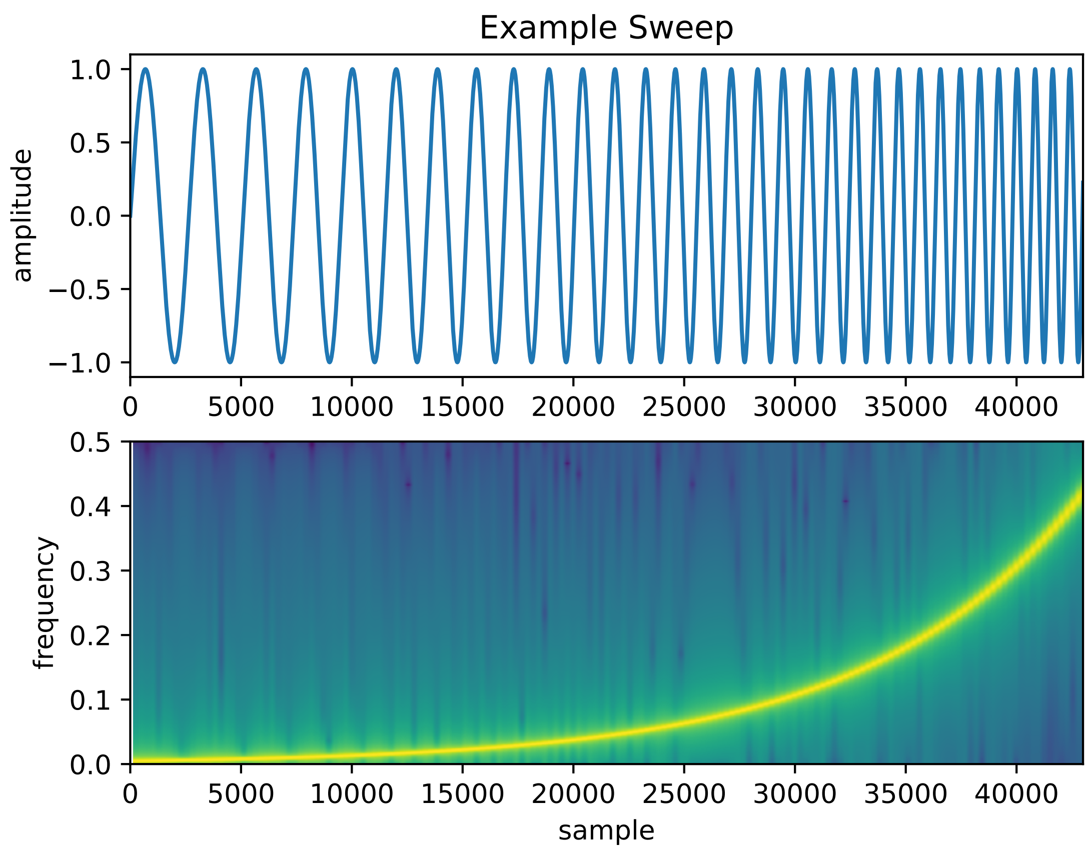
samplerate
[scalar] Samplerate of the signal in Hz.

Returns

sweep
[SyncSweep]

Examples

```
>>> sweep = SyncSweep(16, 16000, 5, 44100)
```



Attributes

duration
Actual duration of the sweep.

durationappr
Approximate/planned duration in seconds.

samplerate
Sample rate of the signal in Hz.

signal
Returns the sweep time signal.

startfreq
Start frequency in Hz

stopfreq
Stop frequency in Hz

sweepperiod
Returns the sweep period according to symbol L in the paper.

time
Time vector relating to given samplerate and actual duration.

Methods

get_windowed_signal(left, right[, ...])	Returns windowd sweep signal

property startfreq
Start frequency in Hz

property stopfreq
Stop frequency in Hz

property durationappr
Approximate/planned duration in seconds.

property samplerate
Sample rate of the signal in Hz.

property signal
Returns the sweep time signal.

property duration
Actual duration of the sweep.

property sweepperiod
Returns the sweep period according to symbol L in the paper.

property time
Time vector relating to given samplerate and actual duration.

get_windowed_signal(*left*, *right*, *pausestart=0*, *pausestop=0*, *amplitude=1*)
Returns windowd sweep signal

The sweep time signal will be faded in and out by hanning ramps.

Parameters

left
[int] Number of samples for fade in hanning ramp at start of the sweep.

right
[int] Number of samples for fade out hanning ramp at end of the sweep.

pausestart
[int] Number of samples for pause befor windowed sweep starts. default is 0.

pausestop
[int] Number of samples for pause after windowed sweep stopps. default is 0.

amplitude
[scalar] Cahnge the amplitude of the sweep. default is 1

`syncsweptsine.invert_spectrum_reg(spec, beta)`

Returns inverse spec with regularization by beta

Parameters

spec
[ndarray] Complex spectrum.

beta
[ndarray or scalar] Regularization parameter. Either of same size as spec or a scalar value.

Returns

invspec
[ndarray]

`syncsweptsine.spectrum_to_minimum_phase(spec)`

Returns a minimum-phase spectrum for given complex *spec*

Parameters

spec
[ndarray] Spectrum (must be twosided)

Returns

minphase
[ndarray]

`class syncsweptsine.InvertedSyncSweepSpectrum(samplerate, sweepperiod, startfreq, stopfreq, fftlen)`

Inverted Spectrum of Synchronized Swept Sine Signal Model Creates the analytical solution of the spectrum according to eq. 43.

Parameters

samplerate
[scalar] Sample rate of the sweep signal.

sweepperiod
[scalar] Sweep period of the sweep signal.

startfreq
[scalar] Start frequency of the sweep signal.

stopfreq
[scalar] Stop frequency of the sweep signal.

fftlen
[int] Number of spectral bins.

Returns

ispec
[InvertedSyncSweepSpectrum instance]

See also:

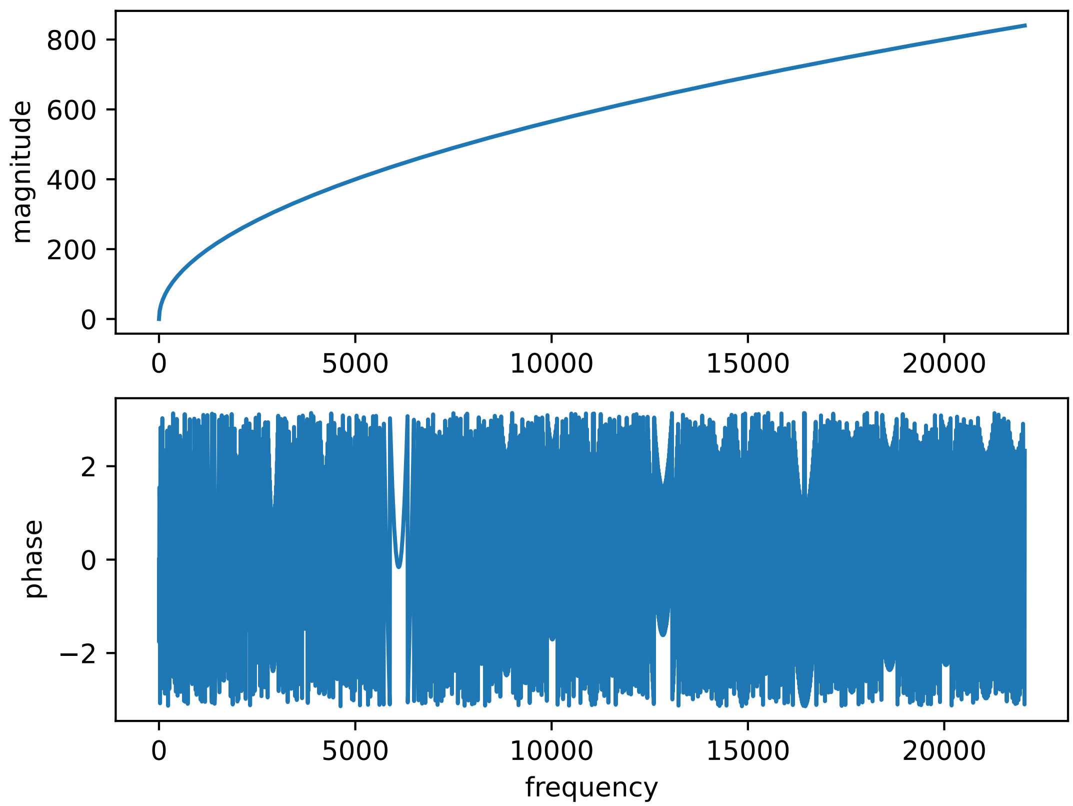
InvertedSyncSweepSpectrum.from_sweep()
SyncSweep

Notes

If you want to invert a SyncSweep instance use the *InvertedSyncSweepSpectrum.from_sweep()*.

Examples

```
>>> sweep = SyncSweep(16, 16000, 5, 44100)
>>> inv_sweep = InvertedSyncSweepSpectrum.from_sweep(sweep)
```



Attributes

fftlen
Number of fft bins.

freq
Frequency vector for the spectrum

spectrum
The inverted spectrum.

Methods

from_sweep(syncsweep, fftlen)	Returns a InvertedSyncSweepSpectrum instance for given syncsweep.

classmethod from_sweep(*syncsweep*, *fftlen*)

Returns a InvertedSyncSweepSpectrum instance for given syncsweep. Creates the analytical solution of the spectrum according to eq. 43.

Parameters

syncsweep
[SyncSweep] Instance of a SyncSweep.from_syncsweep

fftlen
[int] Length of fft for spectrum creation

property spectrum

The inverted spectrum.

property freq

Frequency vector for the spectrum

property fftlen

Number of fft bins.

class syncsweptsine.**HigherHarmonicImpulseResponse**(*hhir=None*, *hhfrf=None*, *sweepperiod=None*, *samplerate=None*)

Higher Harmonic Impulse Response Signal containing Impulse responsens for all harmonics.

To create a HigherHarmonicImpulseResponse from sweep input and output signals, use the HigherHarmonicImpulseResponse.from_sweeps() class method.

Parameters

hhir
[ndarray] Higher Harmonic Impulse Response array.

hhfrf
[ndarray] Higher Harmonic Frequency Response Function array. Optional. Will be available if .from_sweeps() method is used.

sweepperiod
[scalar] Sweep period of the used sweep. Needed for calculation of time position of harmonic impulse responses.

samplerate
[scalar]

Returns

hhir
[HigherHarmonicImpulseResponse]

See also:

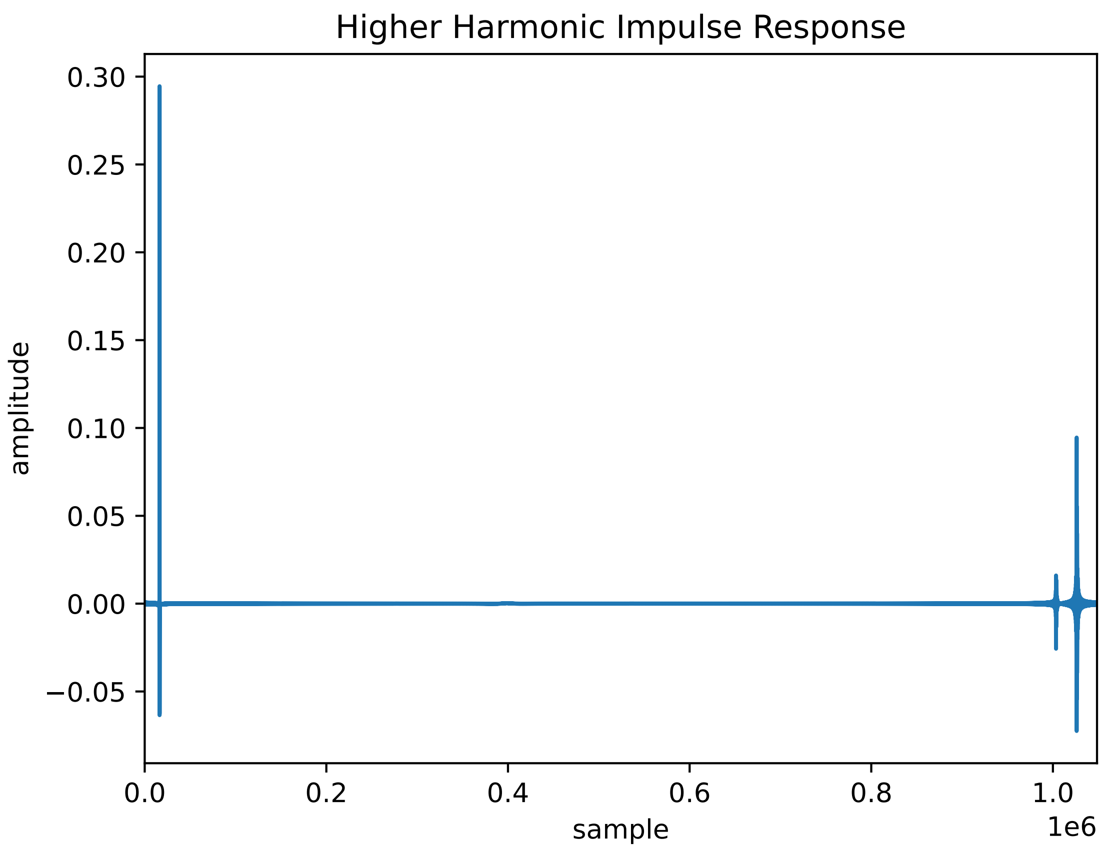
HigherHarmonicImpulseResponse.from_sweeps()
HigherHarmonicImpulseResponse.from_spectra()
SyncSweep
HammersteinModel

Notes

To create a HigherHarmonicImpulseResponse from sweep input and output signals, use the HigherHarmonicImpulseResponse.from_sweeps() class method.

Examples

```
>>> sweep = SyncSweep(16, 16000, 5, 44100)
>>> sig = sweep.get_windowed_signal(4096, 4096, 2*8192, 4*8192)
>>> measured = sig + 0.5*sig**2 + 0.25*sig**3
>>> hhir = HigherHarmonicImpulseResponse.from_sweeps(sweep, measured)
```



Attributes

samplerate
Returns the Samplerate of the impulse response.

Methods

from_spectra(rspec, rinvspec, sweepperiod, ...)	Returns Higher Harmonic Response instance
from_sweeps(syncsweep, measuredsweep[, ...])	Returns Higher Harmonic Impulse Response instance for given sweep signals.
harmonic_impulse_response(order[, length, ...])	Returns the harmonic impulse response of *order* and *length*
hir_index(order, length[, delay])	Returns the index the harmonic impulse response of *order* and *length*.
hir_sample_position(order)	Returns the sample delay for the harmonic impulse response of *order*.
hir_time_position(order)	Returns the time delay for the harmonic impulse response of *order*.
max_hir_length(order)	Returns the maximum length of mpulse responses for given orders.

property samplerate

Returns the Samplerate of the impulse response.

hir_time_position(*order*)

Returns the time delay for the harmonic impulse response of *order*.

hir_sample_position(*order*)

Returns the sample delay for the harmonic impulse response of *order*.

hir_index(*order*, *length*, *delay=0*)

Returns the index the harmonic impulse response of *order* and *length*.

Parameters

order
[int] Order of required harmonic impulse response.

length
[int] Length of required harmonic impulse response.

delay
[int] Delay of system under test the hhir was derived from.

max_hir_length(*order*)

Returns the maximum length of mpulse responses for given orders.

Parameters

order: int

Returns

maxlength
[int]

Notes

The HHIR contains all harmonic impulse responses (HIR). For slicing one specific HIR there is a maximum number of samples around this HIR. A bigger slice may contain parts of neighbouring HIRs. Depending on the highest order there is a maximum length.

`harmonic_impulse_response`(*order*, *length=None*, *delay=0*, *window=None*)

Returns the harmonic impulse response of *order* and *length*

Parameters

order
[int] Order of required harmonic impulse response.

length
[int] Length of required harmonic impulse response.

delay
[int] Delay of system under test the hhir was derived from.

`classmethod from_sweeps`(*syncsweep*, *measuredsweep*, *fftlen=None*, *regularize=1e-06*)

Returns Higher Harmonic Impulse Response instance for given sweep signals.

Parameters

syncsweep
[SyncSweep] A SyncSweep instance.

measuredsweep
[ndarray] Measured sweep. Must be the output signal of the system under test excited with the provided *syncsweep*. Besides it must be sampled at the same samplerate as the provided syncsweep.

fftlen
[int] Length of the calculated ffts. fftlen will be guessed from measuredsweep length if fftlen is None.

`classmethod from_spectra`(*rspec*, *rinvspec*, *sweepperiod*, *samplerate*)

Returns Higher Harmonic Response instance

Parameters

rspec
[ndarray] rfft spectrum from measured sweep.

rinvspec
[ndarray] rfft spectrum from inverted reference sweep.

sweepperiod
[scalar] The parameter L from the paper to calculate the time delays for hhir decomposition.

`class syncsweptsine.FrfFilterKernel`(*freq*, *frf*, *ir=None*)

Returns a FRF-FilterKernel

Parameters

freq
[ndarray] Frequency vector (positive frequencies)

frf
[ndarray] Frequency responce function (onesided spectrum)

ir
[ndarray] Impulse response (optional) If you just have an impulse response use the *FrfFilterKernel.from_ir()* classmethod.

See also:

HammersteinModel

Attributes

freq
Returns the frequency vector.

frf
Returns the frequency response function (FRF)

ir
Returns the impulse response (IR)

Methods

as_minimum_phase()	Returns a filter kernel with minimum phase response.
filter(x)	Returns the convolved signal *x*.

from_ir	

property freq
Returns the frequency vector.

property frf
Returns the frequency response function (FRF)

property ir
Returns the impulse response (IR)

filter(*x*)
Returns the convolved signal *x*.

as_minimum_phase()
Returns a filter kernel with minimum phase response.

class syncsweptsine.**IirFilterKernel**(*bcoeff*, *acoeff*)
Returns a IIR-FilterKernel

Parameters

bcoeff
[ndarray] Filter coefficients of the numerator.

acoeff
[ndarray] Filter coefficients of the denominator.

See also:

HammersteinModel

Methods

filter(x[, axis, zi])	Returns the filtered signal *x*.
freqz(nfft)	Returns the frequency response for the IIR Filter Kernel.
to_frf_filter_kernel(nfft)	Returns a FrfFilterKernel instance

filter(*x*, *axis=-1*, *zi=None*)

Returns the filtered signal *x*. For more info see help of *scipy.signal.lfilter*.

freqz(*nfft*)

Returns the frequency response for the IIR Filter Kernel.

Parameters

nfft
[int] Number of bins.

to_frf_filter_kernel(*nfft*)

Returns a FrfFilterKernel instance

Parameters

nfft
[int] Number of bins.

class syncsweptsine.**HammersteinModel**(*kernels*, *orders*)

Hammerstein Model

$$y = f(x) = \sum_{n}^{N} x^n * h_n$$

A Hammerstein model can be created from a *HigherHarmonicImpulseResponse* by using the method *HammersteinModel.from_higher_harmonic_impulse_response()*.

Parameters

kernels: iterable
Contains Kernels with a .filter() method.

orders: iterable
Denotes the nonlinearity order for the kernels. Must be of same length as *kernels*. The linear kernel order is 1 (x**1), the second order kernel is 2 (x**2) . . .

See also:

HigherHarmonicImpulseResponse
FrfFilterKernel
IirFilterKernel

Attributes

kernels
Returns the hammerstein kernels.

orders
Returns the orders for the hammerstein kernels.

Methods

create_kernel_to_hhfrf_transformation_matrix(orders)	Returns a transformation matrix for combining kernels to higher harmonic frequency response functions
filter(sig)	Returns nonlinear filtered signal by this hammerstein model cascade.
from_higher_harmonic_impulse_response(hhir, ...)	Returns a HammersteinModel for given HigherHarmonicImpulseResponse
from_sweeps(syncsweep, measuredsweep, orders)	Returns a HammersteinModel for given sweeps
gen_filtered_signal_cascade(sig)	Yields hammerstein cascade filtered signals.

classmethod from_sweeps(*syncsweep*, *measuredsweep*, *orders*, *delay=0*, *irlength=None*, *window=None*, *fftlen=None*, *regularize=1e-06*)

Returns a HammersteinModel for given sweeps

Parameters

syncsweep
[SyncSweep] A SyncSweep instance.

measuredsweep
[ndarray] Measured sweep. Must be the output signal of the system under test excited with the provided *syncsweep*. Besides it must be sampled at the same samplerate as the provided syncsweep.

orders
[iterable of int] The orders of hammerstein kernels to compute. Linear kernel is order 1 (x**1), quadratic kernel is order 2 (x**2), …

delay
[int] delay of the system under test, needed for correct slicing of harmonic impulse responses.

irlength
[int] length of the harmonic impulse response to compute the kernels from.

window
[bool, int or ndarray(length)] Linear kernel is order 1 (x**1), quadratic kernel is order 2 (x**2), …

fftlen
[int] Length of the calculated ffts. fftlen will be guessed from measuredsweep length if fftlen is None.

regularize
[scalar or False] Regularizes the system so if measuredsweep would be equal to the syncsweep signal, identity is ensured.

classmethod from_higher_harmonic_impulse_response(*hhir*, *length*, *orders*, *delay=0*, *window=None*)

Returns a HammersteinModel for given HigherHarmonicImpulseResponse

Parameters

hhir
[HigherHarmonicImpulseResponse]

length
[int] length of the harmonic impulse responses to compute hammerstein kernels from. The hammerstein kernels will have the same length.

orders
[iterable of int] The orders of hammerstein kernels to compute. Linear kernel is order 1 (x**1), quadratic kernel is order 2 (x**2), ...

delay
[int] delay of the system under test, needed for correct slicing of harmonic impulse responses.

window
[bool, int or ndarray(length)]

`property kernels`

Returns the hammerstein kernels.

`property orders`

Returns the orders for the hammerstein kernels.

`static create_kernel_to_hhfrf_transformation_matrix(`*orders*`)`

Returns a transformation matrix for combining kernels to higher harmonic frequency response functions

Parameters

orders
[int] Orders of the kernels.

Returns

transformation_matrix
[ndarray]

`gen_filtered_signal_cascade(`*sig*`)`

Yields hammerstein cascade filtered signals.

`filter(`*sig*`)`

Returns nonlinear filtered signal by this hammerstein model cascade.

`class syncsweptsine.LinearModel(`*kernel*`)`

Returns a LinearModel

Parameters

kernel
[FilterKernel] A kernel instance with a filter method

See also:

LinearModel.from_higher_harmonic_impulse_response()
LinearModel.from_hammerstein_model()

Attributes

kernel

Methods

`filter`(sig)	Returns linear filtered *sig*.
`from_hammerstein_model`(hmodel)	Returns a LinearModel of the given Hammerstein-Model.
`from_higher_harmonic_impulse_response`(hhir)	Returns a LinerModel for given HigherHarmonicImpulseResponse
`from_sweeps`(syncsweep, measuredsweep[, ...])	Returns a LinerModel for given sweeps

classmethod from_sweeps(*syncsweep: SyncSweep*, *measuredsweep*, *delay=0*, *irlength=None*, *window=None*, *fftlen=None*, *regularize=1e-06*, *bandpass=True*)

Returns a LinerModel for given sweeps

Parameters

syncsweep
[SyncSweep] A SyncSweep instance.

measuredsweep
[ndarray] Measured sweep. Must be the output signal of the system under test excited with the provided *syncsweep*. Besides it must be sampled at the same samplerate as the provided syncsweep.

delay
[int] delay of the system under test, needed for correct slicing of harmonic impulse responses.

irlength
[int] length of the harmonic impulse response to compute the kernel from.

window
[bool, int or ndarray(length)] Linear kernel is order 1 (x**1), quadratic kernel is order 2 (x**2), …

fftlen
[int] Length of the calculated ffts. fftlen will be guessed from measuredsweep length if fftlen is None.

regularize
[scalar or False] Regularizes the system so if measuredsweep would be equal to the syncsweep signal, identity is ensured.

classmethod from_higher_harmonic_impulse_response(*hhir: HigherHarmonicImpulseResponse*, *length=None*, *delay=0*, *window=None*, *startfreq=None*, *stopfreq=None*)

Returns a LinerModel for given HigherHarmonicImpulseResponse

Parameters

hhir
[HigherHarmonicImpulseResponse]

length
[int] length of the harmonic impulse compute the kernel from.

orders
[iterable of int] The orders of hammerstein kernels to compute. Linear kernel is order 1 (x**1), quadratic kernel is order 2 (x**2), …

delay
[int] delay of the system under test, needed for correct slicing of harmonic impulse responses.

window
[bool, int or ndarray(length)]

startfreq
[scalar or None] Frequency window in spectrum will be applied.

stopfreq
[scalar or None] Frequency window in spectrum will be applied.

`classmethod from_hammerstein_model`(*hmodel*)

Returns a LinearModel of the given HammersteinModel.

Parameters

hmodel
[HmmersteinModel]

Returns

lmodel
[LinearModel]

`filter`(*sig*)

Returns linear filtered *sig*.

CHAPTER

THREE

INDICES AND TABLES

- genindex
- search

PYTHON MODULE INDEX

S

syncsweptsine, 1

INDEX

A

as_minimum_phase() (*syncsweptsine.FrfFilterKernel method*), 16

C

create_kernel_to_hhfrf_transformation_matrix() (*syncsweptsine.HammersteinModel static method*), 19

D

duration (*syncsweptsine.SyncSweep property*), 9
durationappr (*syncsweptsine.SyncSweep property*), 9

F

fftlen (*syncsweptsine.InvertedSyncSweepSpectrum property*), 12
filter() (*syncsweptsine.FrfFilterKernel method*), 16
filter() (*syncsweptsine.HammersteinModel method*), 19
filter() (*syncsweptsine.IirFilterKernel method*), 17
filter() (*syncsweptsine.LinearModel method*), 21
freq (*syncsweptsine.FrfFilterKernel property*), 16
freq (*syncsweptsine.InvertedSyncSweepSpectrum property*), 12
freqz() (*syncsweptsine.IirFilterKernel method*), 17
frf (*syncsweptsine.FrfFilterKernel property*), 16
FrfFilterKernel (*class in syncsweptsine*), 15
from_hammerstein_model() (*syncsweptsine.LinearModel class method*), 21
from_higher_harmonic_impulse_response() (*syncsweptsine.HammersteinModel class method*), 18
from_higher_harmonic_impulse_response() (*syncsweptsine.LinearModel class method*), 20
from_spectra() (*syncsweptsine.HigherHarmonicImpulseResponse class method*), 15
from_sweep() (*syncsweptsine.InvertedSyncSweepSpectrum class method*), 12
from_sweeps() (*syncsweptsine.HammersteinModel class method*), 18
from_sweeps() (*syncsweptsine.HigherHarmonicImpulseResponse class method*), 15
from_sweeps() (*syncsweptsine.LinearModel class method*), 20

G

gen_filtered_signal_cascade() (*syncsweptsine.HammersteinModel method*), 19
get_windowed_signal() (*syncsweptsine.SyncSweep method*), 9

H

HammersteinModel (*class in syncsweptsine*), 17
hannramp() (*in module syncsweptsine*), 7
harmonic_impulse_response() (*syncsweptsine.HigherHarmonicImpulseResponse method*), 15
HigherHarmonicImpulseResponse (*class in syncsweptsine*), 12
hir_index() (*syncsweptsine.HigherHarmonicImpulseResponse method*), 14
hir_sample_position() (*syncsweptsine.HigherHarmonicImpulseResponse method*), 14
hir_time_position() (*syncsweptsine.HigherHarmonicImpulseResponse method*), 14

I

IirFilterKernel (*class in syncsweptsine*), 16
invert_spectrum_reg() (*in module syncsweptsine*), 10
InvertedSyncSweepSpectrum (*class in syncsweptsine*), 10
ir (*syncsweptsine.FrfFilterKernel property*), 16

K

kernels (*syncsweptsine.HammersteinModel property*), 19

L

LinearModel (*class in syncsweptsine*), 19

M

max_hir_length() (*syncsweptsine.HigherHarmonicImpulseResponse method*), 14
module
    syncsweptsine, 1

O

orders (*syncsweptsine.HammersteinModel property*), 19

S

samplerate (*syncsweptsine.HigherHarmonicImpulseResponse property*), 14
samplerate (*syncsweptsine.SyncSweep property*), 9
signal (*syncsweptsine.SyncSweep property*), 9
spectrum (*syncsweptsine.InvertedSyncSweepSpectrum property*), 12
spectrum_to_minimum_phase() (*in module syncsweptsine*), 10
startfreq (*syncsweptsine.SyncSweep property*), 9
stopfreq (*syncsweptsine.SyncSweep property*), 9
sweepperiod (*syncsweptsine.SyncSweep property*), 9
SyncSweep (*class in syncsweptsine*), 7
syncsweptsine
    module, 1

T

time (*syncsweptsine.SyncSweep property*), 9
to_frf_filter_kernel() (*syncsweptsine.IirFilterKernel method*), 17