

---

**Satis**

**Team COOP**

**Dec 17, 2021**



## CONTENTS:

<b>1</b>	<b>Welcome to Satis</b>	<b>1</b>
1.1	Installation . . . . .	2
1.2	How to use it . . . . .	2
1.3	Satis as a package . . . . .	6
<b>2</b>	<b>Acknowledgements</b>	<b>9</b>
<b>3</b>	<b>Analyze temporal signals with Satis</b>	<b>11</b>
3.1	Introduction . . . . .	11
3.2	Temporal_analysis_tool . . . . .	13
<b>4</b>	<b>satis package</b>	<b>15</b>
4.1	Submodules . . . . .	15
4.2	satis.avbp2global module . . . . .	15
4.3	satis.avbp2global_compute_fft module . . . . .	15
4.4	satis.avbp2global_diagnostics module . . . . .	15
4.5	satis.avbp2global_read_inputs module . . . . .	15
4.6	satis.cerfacs_plot module . . . . .	15
4.7	satis.cli module . . . . .	15
4.8	satis.io module . . . . .	15
4.9	satis.satis_library module . . . . .	15
4.10	satis.temporal_analysis_tool module . . . . .	15
<b>5</b>	<b>Indices and tables</b>	<b>17</b>



WELCOME TO SATIS



logo

*Spectral Analysis for Times Signals.*

Satis is a python3 / scipy implementation of the Fourier Spectrums in Amplitude and Power Spectral Density. It is particularly suited for CFD signals with the following characteristics:

- Short sampling time,
- Potentially short recording time,
- Low signal-to-noise ratio,
- Multiple measures available.

## 1.1 Installation

The package is available on [PyPI](#) so you can install it using pip:

```
pip install satis
```

## 1.2 How to use it

```
(my_env)rossi@pluto:~>satis

Usage: satis [OPTIONS] COMMAND [ARGS]...

----- SATIS -----

You are now using the Command line interface of Satis, a simple tool for
spectral analysis optimized to signals from CFD, created at CERFACS
(https://cerfacs.fr).

This is a python package currently installed in your python environment.

Options:
  --help  Show this message and exit.

Commands:
  datasetforbeginners  Copy a set of signals to train using Satis.
  fourierconvergence   Plot discrete Fourier transform of the complete...
  fouriervariability   Plot the Fourier variability diagnostic results.
  psdconvergence       Plot the PSD convergence diagnostic results.
  psdvariability       Plot the spectral energy at the target frequency.
  time                Plot the temporal signal and its time-average.
```

Several command lines are available on satis. You can display them running the command `satis --help`.

### 1.2.1 Dataset for beginners

```
satis datasetforbeginners
```

With this command , you can copy in your local directory a file `my_first_dataset.dat` to start using satis. It contains several signals of a CFD simulation. These signals have been recorded at different locations to create an average signal less sensitive to noise. For your first time with satis, we recommend to do the following diagnostics in the order with `my_first_dataset.dat`.

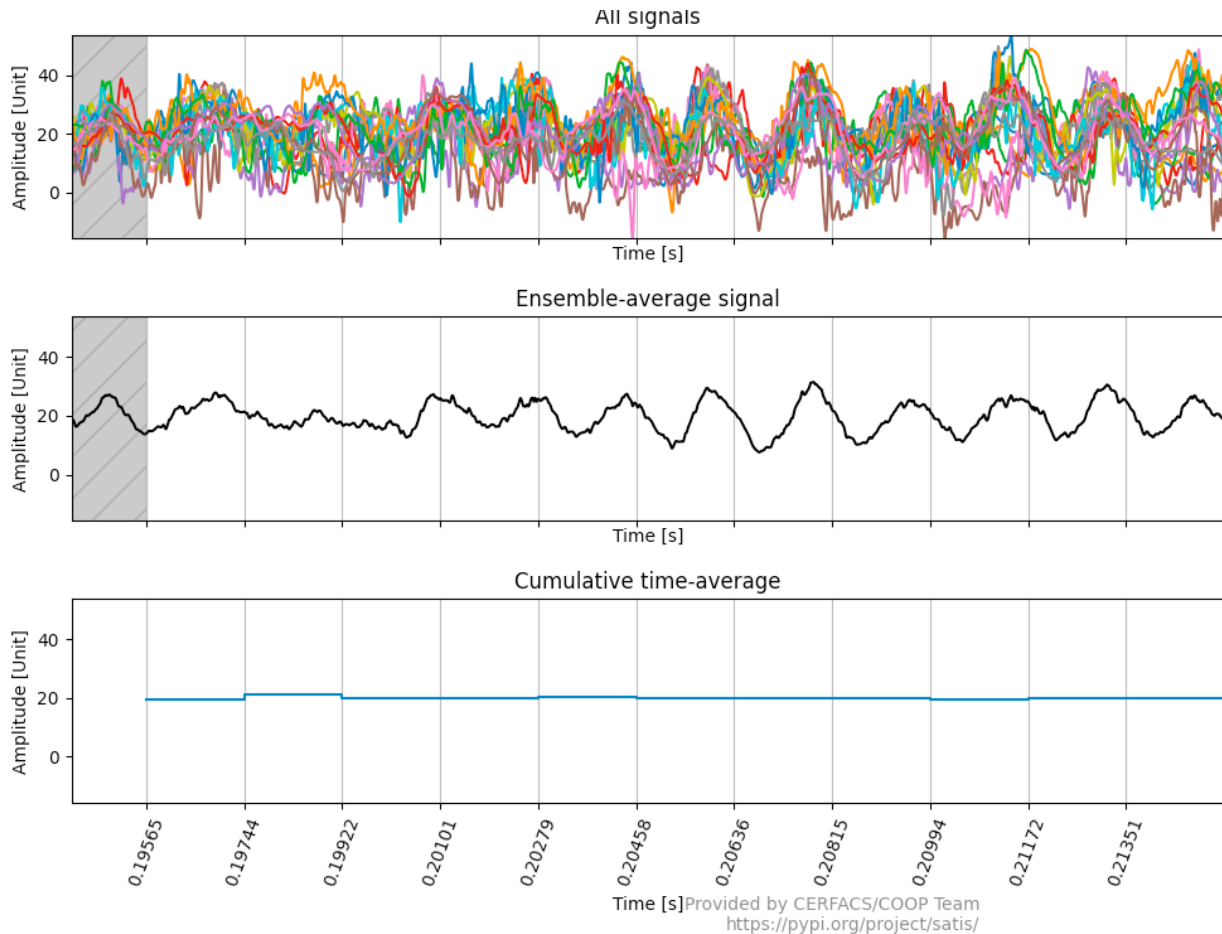
## 1.2.2 Time

```
satis time my_first_dataset.dat
```

This diagnostic plots a time graph of your signals. This plot aims at showing you if the average signal is well converged or if there is a transient behavior. To delete a transient behavior, you can add at the end of the diagnostic command `-t *starting_time*` to declare the beginning of the converged behavior.

If a periodic pattern is visible, you should calculate its frequency and declare it with `-f *calculated_frequency*`

There is also a cumulative time-average. If this curve is not almost flat, you did probably not remove enough transient behavior.



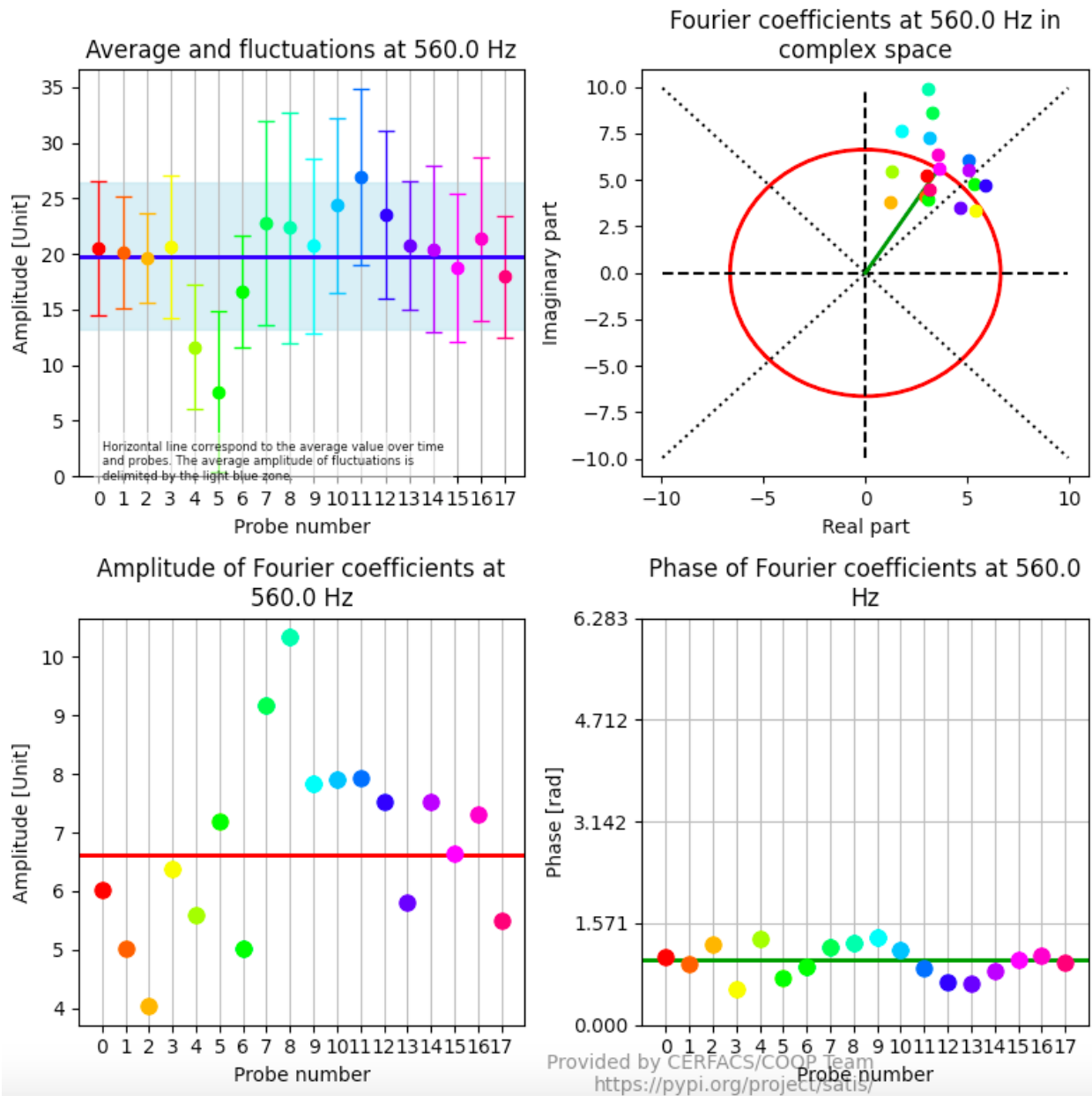
diagnostic

time

## 1.2.3 Fourier variability

```
satis fourieviability my_first_dataset.dat -t 0.201 -f 560
```

In this diagnostic, the Fourier coefficients of each signal at the specified frequency is plotted so that you can check the signals are equivalent. If a signal seems have different characteristics to the others, you should think about removing it. The average signal would be cleaner. To do so, declare the subset of signals you want to use with: `--subset 1 3 14 ...`



fourier

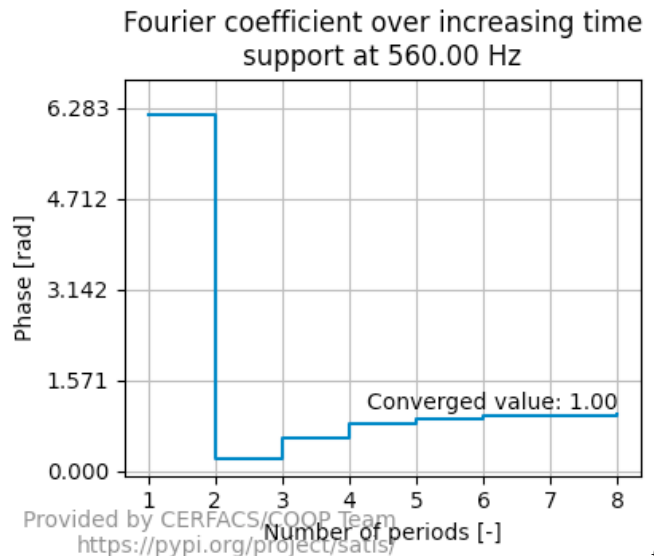
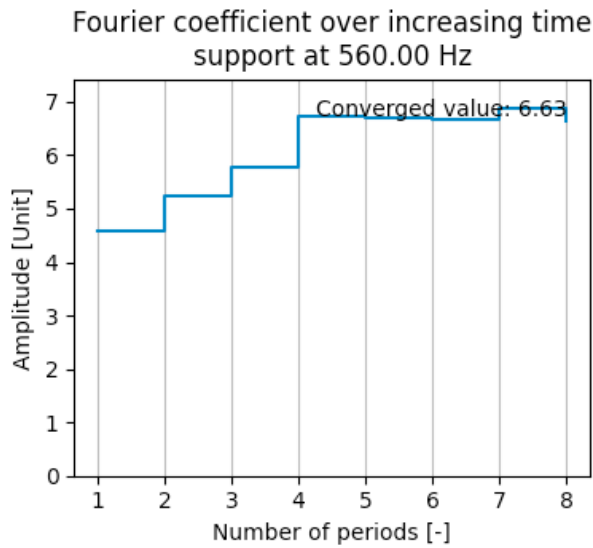
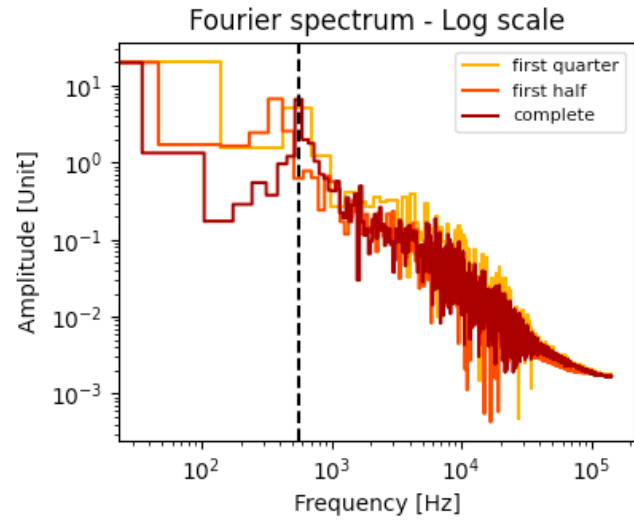
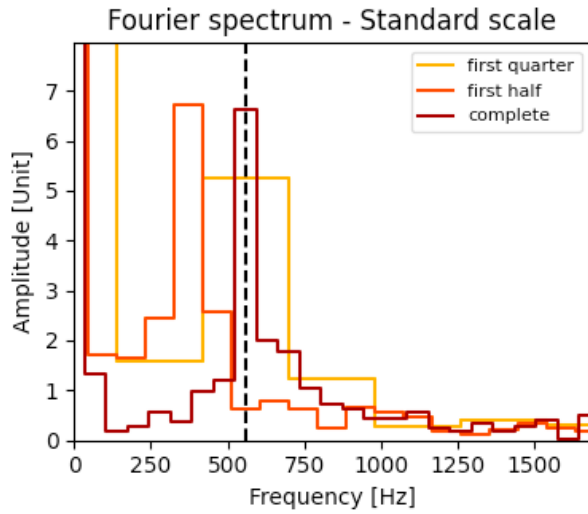
variability diagnostic

### 1.2.4 Fourier convergence

```
satis fourierconvergence my_first_dataset.dat -t 0.201 -f 560
```

Since this diagnostic is based on the average signal, the user should have checked beforehand that all input signals are equivalent thanks to the `fouriervariability` diagnostic. The top plots show the amplitude of the Discrete Fourier Transform performed on the complete average signal, the last “half” of the signal and the last “quarter” of the signal. The bottom plots show the convergence over increasing time of the amplitude and phase of the signal at the specified frequency.





Provided by CERFACS/COOP Team  
<https://pypi.org/project/satis/>

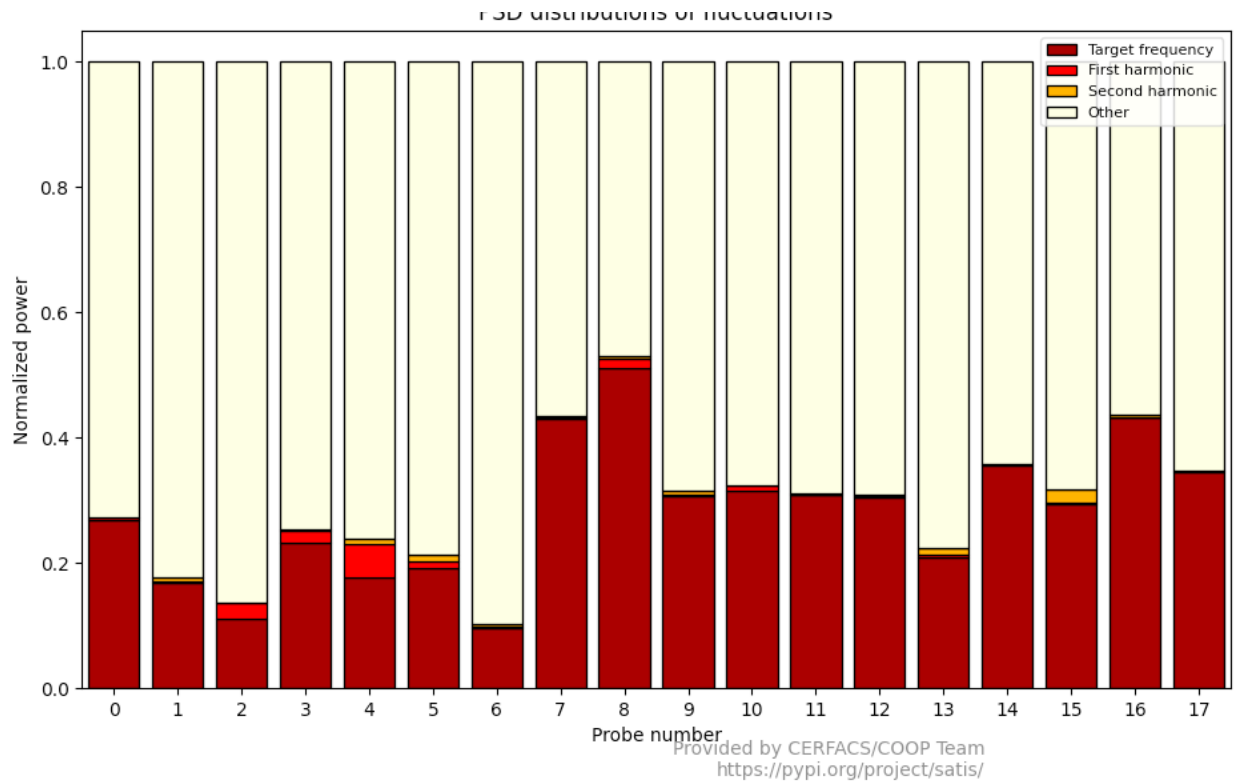
fourier

convergence diagnostic

### 1.2.5 PSD variability

```
satis psdvariability my_first_dataset.dat -t 0.201 -f 560
```

This diagnostic shows the distribution of the spectral energy of fluctuations on the target frequency, its first and second harmonic and the rest of the frequencies. Note that this distribution is related to the fluctuations and that the time-average has been removed from the signal.



psd

variability diagnostic

## 1.2.6 PSD convergence

```
satis psdconvergence my_first_dataset.dat -t 0.201 -f 560
```

Just as the Fourier convergence, the PSD convergence diagnostic shows the Power Spectral Density obtained on the complete signal, the last half and the last quarter. The left uses a standard linear scale while the right plot shows the same result with log scales.

![psd convergence diagnostic]<https://cerfacs.fr/coop/images/satis/psdconvergence.png>

## 1.3 Satis as a package

Of course, you can use satis in your own project importing it as a package:

```
import os
import glob
import satis
import matplotlib.pyplot as plt

*you awesome code*

time, signals = satis.read_signal('your_dataset.dat')
clean_time = satis.define_good_time_array(time, signals)
clean_signals = satis.interpolate_signals(time, signals, clean_time)
```

(continues on next page)

(continued from previous page)

```
new_time, new_signals = satis.get_clean_signals(clean_time, signals,  
↪ frequency)  
plt.plot(new_time, new_signals)  
fourier = satis.get_coeff_fourier(new_time, new_signals,  
↪ frequency)  
*your awesome code
```

calcula

calculated\_



## ACKNOWLEDGEMENTS

This package is the result of work done at Cerfac's COOP Team. The contributors of this project are:

- Franchine Ni
- Antoine Dauplain
- Tamon Nakano
- Matthieu Rossi



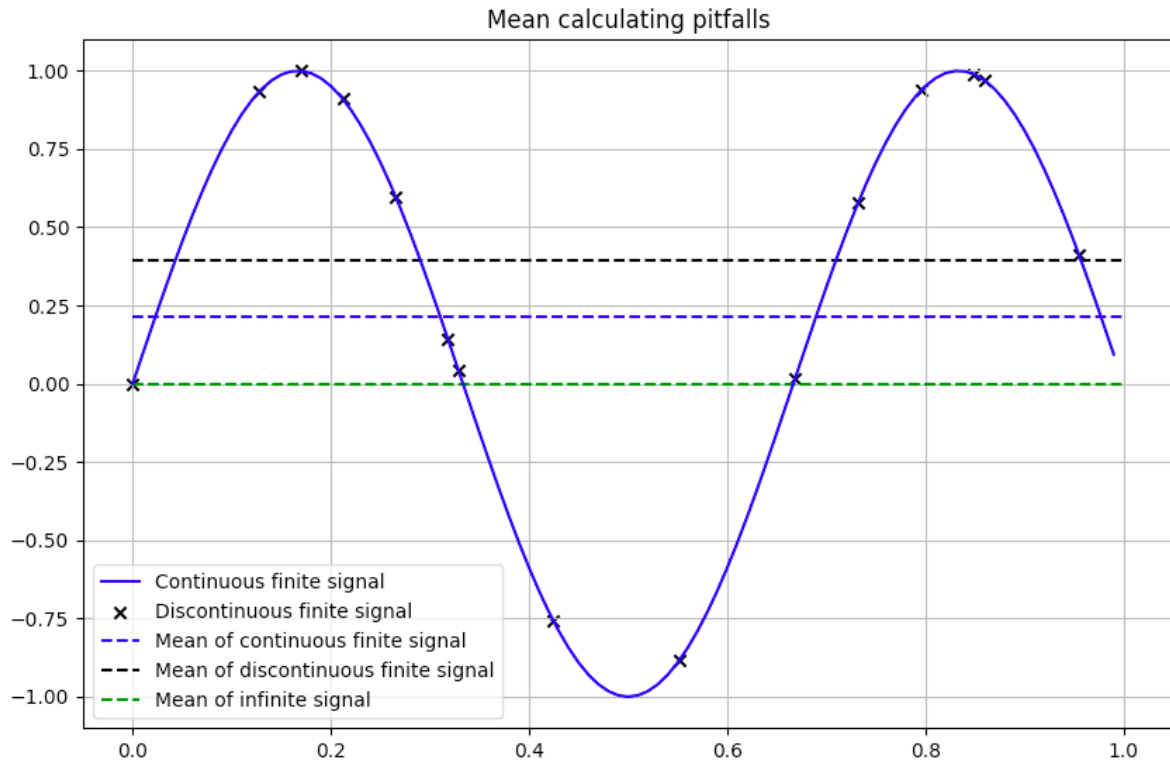
## ANALYZE TEMPORAL SIGNALS WITH SATIS

### 3.1 Introduction

Analyzing a temporal signal may seem a straightforward task. However, when you dive into this topic, several questions may arise:

- What is the mean value of my signal?
- Does my signal capture all the frequencies of interest?
- If this signal is the result of a simulation,
  - is my signal well discretized?
  - is my signal periodic?
  - what is the spectral power of my signal?
  - ...

A canonical example of what we want to discuss here should be the following one:



Mean

pitfalls

In this graph we have a sinus-shaped signal (blue curve). What is the mean value of this signal?

Well, it seems to be 0...

Ok, let's keep this value in mind.

Imagine the signal is the result of a very complex numerical simulation and your time record makes the signal has been discretized at different time steps (black crosses). If we want to calculate the mean value of our signal, the ingenious mind would calculate:

$$\bar{m} = \frac{1}{n} \sum_n x_n$$

Classical mean

Doing so, one would get a mean value of approximately 0.35.

Yes, but obviously the signal is poorly discretized!

Alright, if one had the same finite signal discretized at high frequency, averaging over the time record the value of the signal (to get close to the integral of the blue curve), one would have get a mean value of approx. 0.22.

...

The point of this example is to highlight the need to be aware of some traps in signal analysis. Cleaning up the signal before doing any calculation is a good practice. Hopefully, Satis provides a module named `temporal_analysis_tool` full of functions to help you avoiding these traps (see more information about Satis [here](#)).



## 3.2 Temporal\_analysis\_tool

Hereafter are detailed the functions you can find in the module `temporal_analysis_tool` plus how to use them. Generally, you will have to get your signal and put it in a time array and signal array:

```
record = numpy.genfromtxt("my_record.dat")
time = record[:,0]
signal = record[:,1]
```

### 3.2.1 resample\_signal

Resample the initial signal at a constant time interval.

```
rescaled_time, rescaled_signal = resample_signal(time,
↪ signal,
↪ dtime=0.01)
```

**NB:** If a `dtime` is given, the interpolation is made to have a signal with a time interval of `dtime`. Else, the `dt` is the smallest time interval between two values of the signal.

### 3.2.2 calc\_autocorrelation\_time

Estimate the autocorrelation time at a given threshold.

The idea is: if you have a finite time record, you can always refine your signal, but at some point, you will not add any further information getting more points. This function returns the first time at which the signal is correlated **under** the threshold.

```
auto_time = calc_autocorrelation_time(time, signal, threshold=0.2)
```

### 3.2.3 sort\_spectral\_power

Determine the harmonic power contribution of the signal.

It calculates the Power Spectral Density (PSD) of the complete signal and of a downsampled version of the signal. The difference of the two PSD contains only harmonic components.

```
harmonic_power, total_power = sort_spectral_power(time, signal)
```

### 3.2.4 power\_representative\_frequency

Calculate the frequency that captures a level of spectral power.

It calculates the cumulative power spectral density and returns the frequency that reaches the threshold in percent of spectral power.

```
threshold_frequency = power_representative_frequency(time,  
↪ signal,  
↪ threshold=0.8)
```

### 3.2.5 duration\_for\_uncertainty

Give a suggestion of simulation duration to get an confidence interval range (around the mean) of the target value with a certain level of confidence.

This calculation is based on the Central Limit Theorem (CLT). It supposes several assumptions detailed in this [notebook](#).

```
duration = duration_for_uncertainty(time,  
                                     signal,  
                                     target=10,  
                                     confidence=0.95)
```

### 3.2.6 uncertainty\_from\_duration

Give a confidence interval range given the time step, the standard deviation, the duration and the level of confidence.

```
range = uncertainty_from_duration(dtime,  
                                   sigma,  
                                   duration,  
                                   confidence=0.95)
```

### 3.2.7 calculate\_std

Give the standard deviation of a signal at a given frequency.

```
std = calculate_std(time, signal, frequency)
```

### 3.2.8 power\_spectral\_density

Automate the computation of the Power Spectral Density of a signal.

```
frequency, psd = power_spectral_density(time, signal)
```

## SATIS PACKAGE

### 4.1 Submodules

#### 4.2 `satis.avbp2global` module

#### 4.3 `satis.avbp2global_compute_ftf` module

#### 4.4 `satis.avbp2global_diagnostics` module

#### 4.5 `satis.avbp2global_read_inputs` module

#### 4.6 `satis.cerfac_plot` module

#### 4.7 `satis.cli` module

#### 4.8 `satis.io` module

#### 4.9 `satis.satis_library` module

#### 4.10 `satis.temporal_analysis_tool` module



## INDICES AND TABLES

- genindex
- modindex
- search