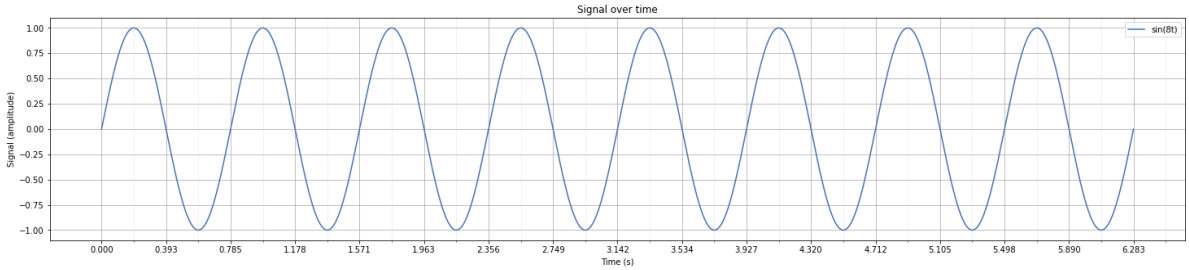
DSIP: Laboratorio 2

Andres Coronado

1. Signal Setting

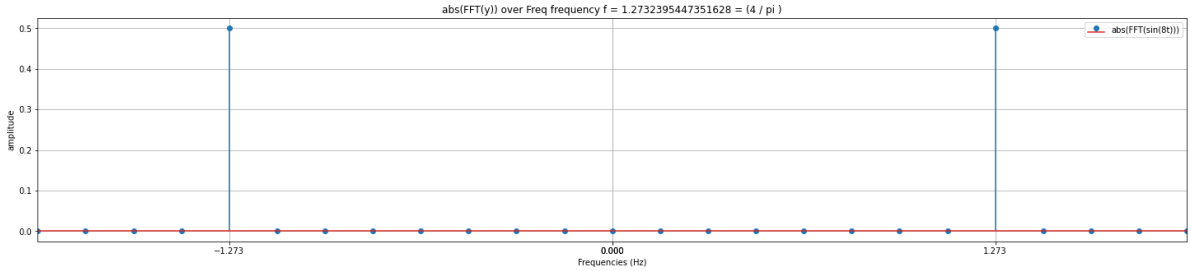
with $time_m = 2\pi$ and frequency $f = 4/\pi$ i set the signal $y = sin(\omega t)$ where ω is the angular frequency ($\omega = 2\pi \cdot f$) i built the following sinusoidal signal made of 6284 samples with a sampling rate of $1000Hz$:



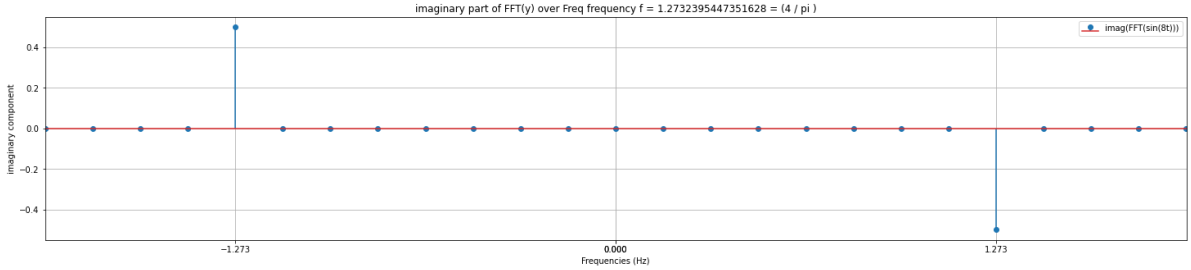
2. FFT

Now I'using the `numpy` library to compute the absolute value of the Fourier transform of the signal, and as expected I get a pick in $f = 4/\pi \approx 1.273Hz$ and in $-f$ (which corresponds to the complex conjugate as y is odd). The plot of the absolute value of $F(y(t))$

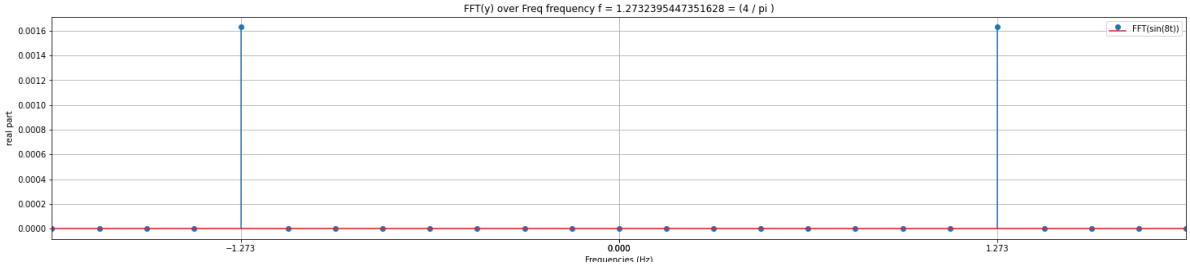
Using `fftfreq` to plot fft (normalized with the number of samples)



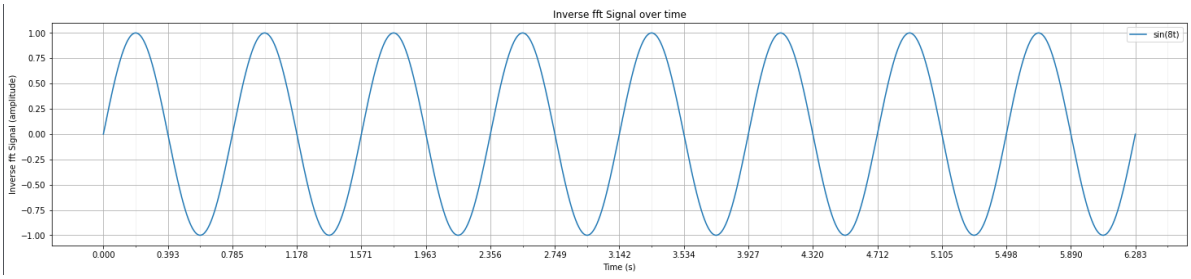
In the plot of the imaginary part I can clearly see the "oddity" of the signal and the subsequent complex conjugate behaviour:



The plot of the real part of the amplitudes shows peaks in the same positions :



Finally, once the signal is reconstructed using the inverse Fourier transform I got back the signal I started with

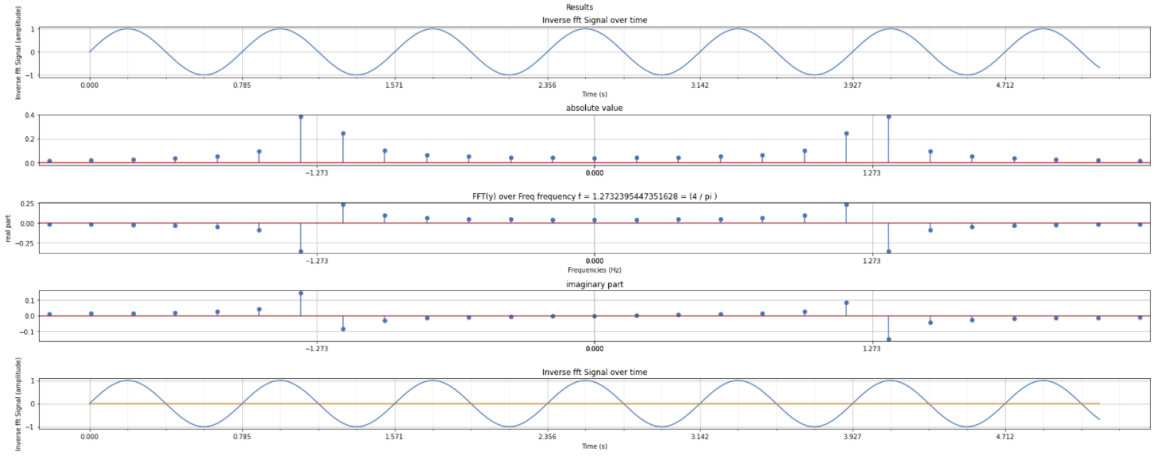


so far so good.

3. Playing around with Time

As I modified the $time_m$ value to be different of any multiple of π something went wrong and some aliasing effect started to show:

with $time_m = 5.2$ I had the following results, showing that the Fourier transform does not peek in the expected frequencies, but it rather have multiple peeks around f. Nevertheless the inverse transform still works fine (although it has a noisy imaginary part which value oscillates between small values around 0, even though the original signal was only real) again the fft results are nomalized over the number of samples on the y axis



4. The "Parsival test"

```
## parseval - norm of the function

## to evaluate the inner product we make use of the squared_norm function
norm_y = squared_norm(y)
norm_y
```
[113] ✓ 0.3s

```
2568.5562857969426
```

```
## parseval - sum of the squared coefficients of the series

## KEEP IN MIND THE NORMALIZATION IN THE PARSEVAL EQUALITY

norm_fft_y = squared_norm(fft_y)*( 1 / len(time))
norm_fft_y
```
[114] ✓ 0.3s

```
(2568.5562857969417+0j)
```

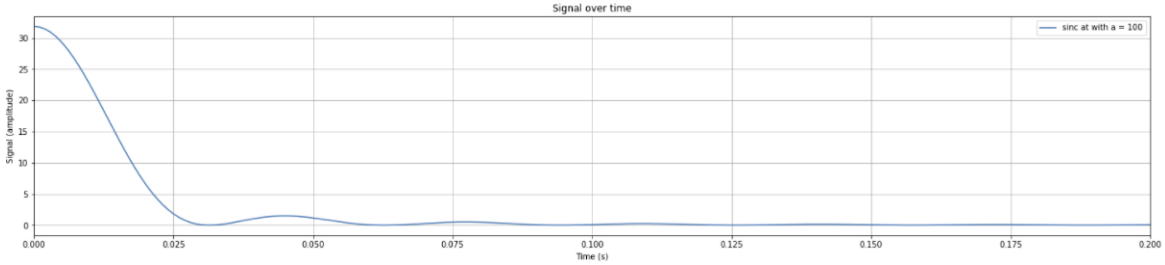
```
## sanity check - compute the norm of the signal obtained by using the inverse fourier transform
norm_y_est =  squared_norm(y_est)

norm_y_est
```
[115] ✓ 0.3s

```
(2568.556285796943+0j)
```

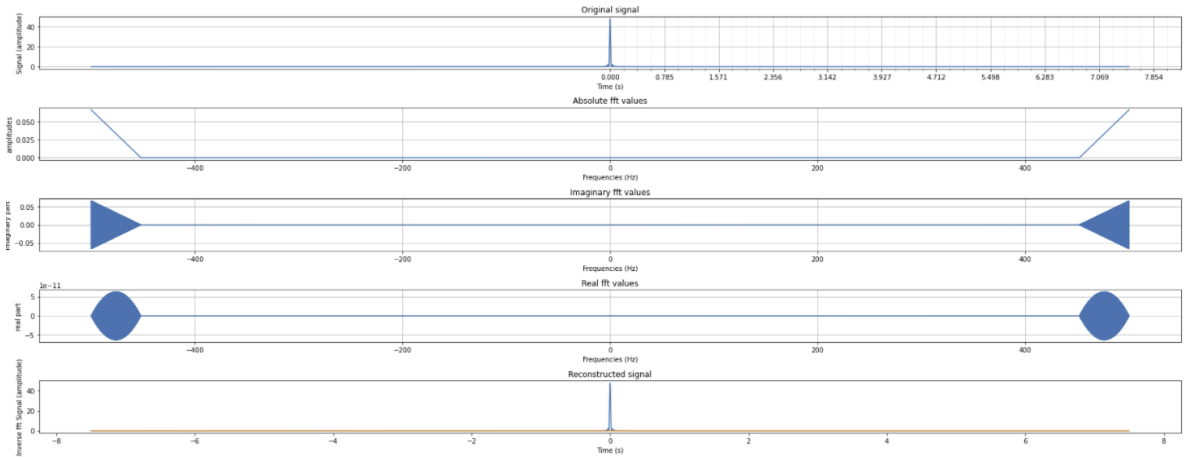
Using the scaling factor $FFT(y)$ as expected

- The norm of y and $FFT(y)$ are very close
- $||FFT(y)||^2$ and $||iFFT(FFT(y))||^2$ should be exactly the same, but is not and it is even closer to the original signal than FFT(y). If i try with a $time_m$ proportional to 2π it looks like these two values are in fact equal.

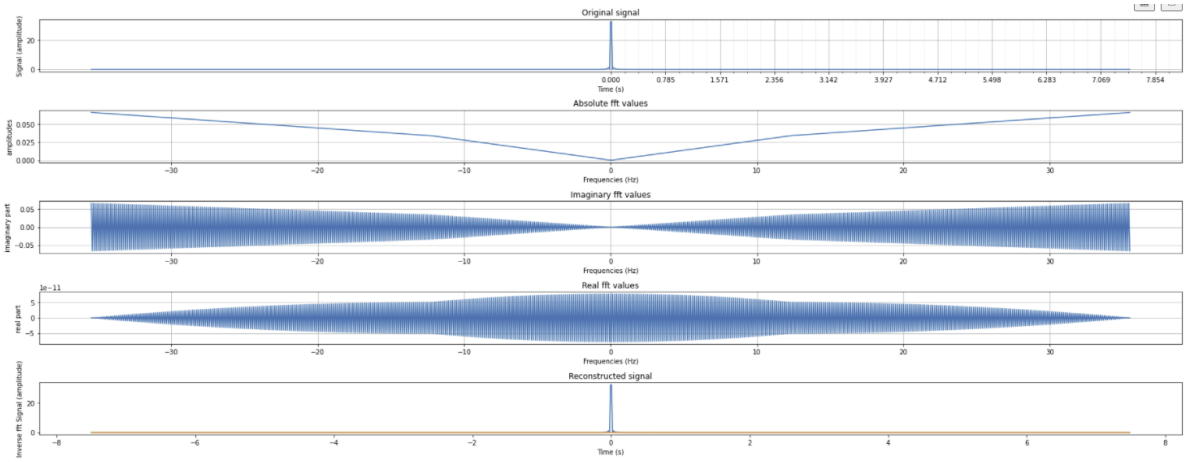
5. Some more fun



Using this $sinc^2$signal using $a = 150$ and plotting out the results I get the following (0 frequencies are in the middle) and starting a time $-time_m$:



lowering the number of samples I can clearly see aliasing :


Original signal
Signal (amplitude)
Time (s)
Absolute fft values
amplitudes
Frequencies (Hz)
Imaginary fft values
Imaginary part
Frequencies (Hz)
Real fft values
1e−11
real part
Frequencies (Hz)
Reconstructed signal
Inverse fft Signal (amplitude)
Time (s)