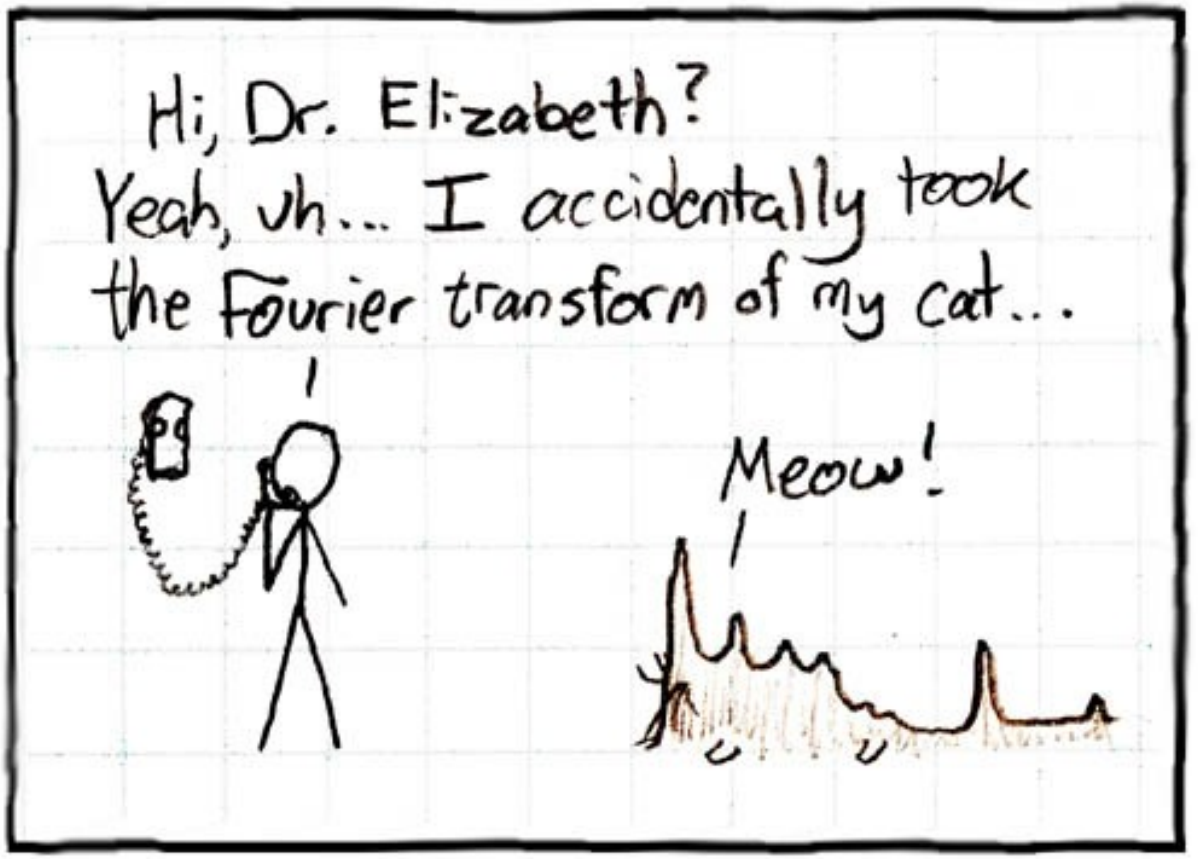
# Numerical Methods for Astrophysics:

## FOURIER TRANSFORMS

**Michela Mapelli**

# Fourier transforms. Concept

**Fourier transforms are ubiquitous in physics, astrophysics and in every-day life (e.g. the mp3 and jpg file format)**




**Very nice Fourier summary here (plus, you learn how to take the Fourier transform of your cat):**

**https://www.youtube.com/watch?v=UGRhqZ2ooIw**

# Fourier transforms. Concept

**A periodic function f(x)** defined on a finite interval 0, L
can be written as Fourier series provided that it is bounded and has at most a finite number of discontinuities and extrema

**EVEN (i.e. symmetric)** function about the midpoint x=L/2

$$f(x) = \sum_{k=0}^{\infty} \alpha_k \cos\left(\frac{2\,\pi\,k\,x}{L}\right)$$

**ODD (i.e. antisymmetric)** function about the midpoint

$$f(x) = \sum_{k=1}^{\infty} \beta_k \sin\left(\frac{2\,\pi\,k\,x}{L}\right)$$

**GENERAL PERIODIC FUNCTION**

$$f(x) = \sum_{k=0}^{\infty} \alpha_k \cos\left(\frac{2\,\pi\,k\,x}{L}\right) + \sum_{k=1}^{\infty} \beta_k \sin\left(\frac{2\,\pi\,k\,x}{L}\right)$$

# Fourier transforms. Concept

Using the complex notation:
(from Euler's formulas)

$$\cos\theta = \frac{1}{2}\left(\exp\left(-i\,\theta\right) + \exp\left(i\,\theta\right)\right)$$

$$\sin\theta = \frac{1}{2}\,i\,\left(\exp\left(-i\,\theta\right) - \exp\left(i\,\theta\right)\right)$$

we get

$$f(x) = \frac{1}{2}\sum_{k=0}^{\infty}\alpha_k\left[\exp\left(-i\,\frac{2\,\pi\,k\,x}{L}\right) + \exp\left(i\,\frac{2\,\pi\,k\,x}{L}\right)\right]$$
$$+\frac{i}{2}\sum_{k=1}^{\infty}\beta_k\left[\exp\left(-i\,\frac{2\,\pi\,k\,x}{L}\right) - \exp\left(i\,\frac{2\,\pi\,k\,x}{L}\right)\right]$$

Collecting terms and generalizing for whatever k

$$f(x) = \sum_{k=-\infty}^{\infty}\gamma_k\,\exp\left(i\,\frac{2\,\pi\,k\,x}{L}\right)$$

where

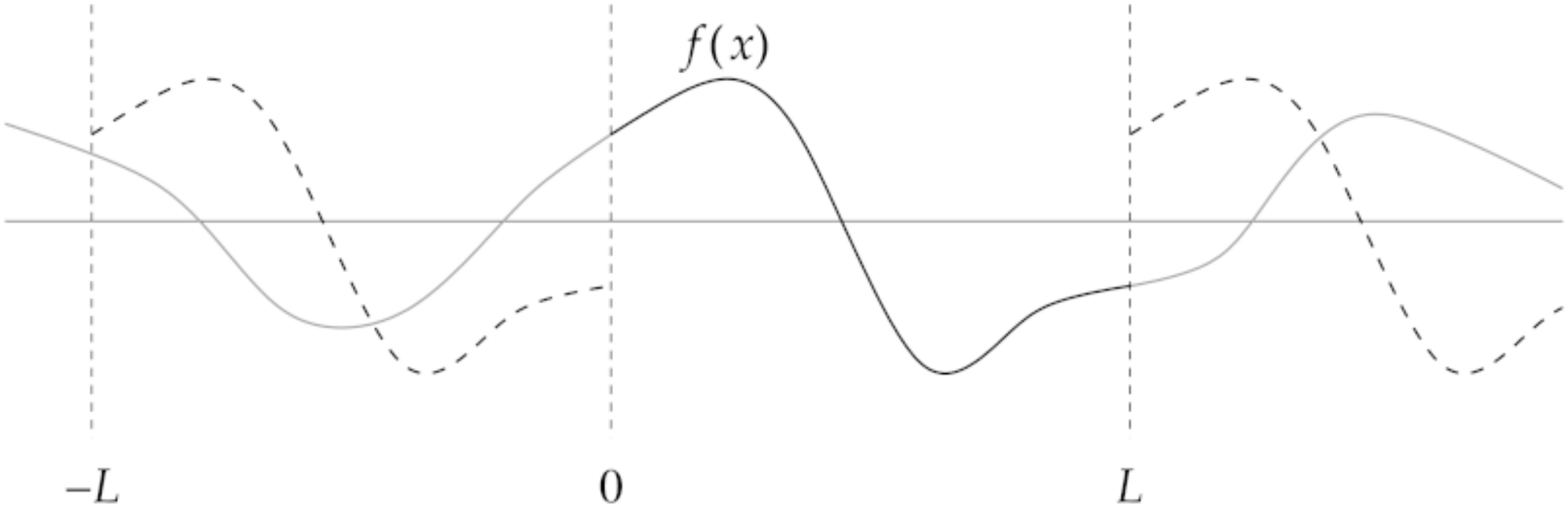
$$\gamma_k = \begin{cases} \frac{1}{2}\left(\alpha_{-k} + i\,\beta_{-k}\right) & \text{if } k < 0 \\ \alpha_0 & \text{if } k = 0 \\ \frac{1}{2}\left(\alpha_k - i\,\beta_k\right) & \text{if } k > 0 \end{cases}$$

# Fourier transforms. Concept

ONLY FOR PERIODIC FUNCTIONS

MOST FUNCTIONS are NOT PERIODIC

**If we are interested only in a portion of a non-periodic function in the interval 0, L, it is always possible to take that portion and just repeat it to create a periodic function**



# Fourier transforms. Concept

**The coefficients $\gamma_k$ are COMPLEX numbers**

**To calculate them** take $$f(x) = \sum_{k=-\infty}^{\infty} \gamma_k \exp\left(i \frac{2\,\pi\,k\,x}{L}\right)$$

and integrate the two terms with a “trick”

i.e. multiply both sides by $$\exp\left(-i\frac{2\,\pi k\,x}{L}\right)$$

$$\int_0^L f(x) \exp\left(-i \frac{2\,\pi\,k\,x}{L}\right) \mathrm{d}x = \sum_{k'=-\infty}^{\infty} \gamma_{k'} \int_0^L \exp\left(i\frac{2\,\pi\,(k'-k)\,x}{L}\right) \mathrm{d}x$$

if k == k’ then $$\int_0^L \exp\left(i\frac{2\,\pi\,(k'-k)\,x}{L}\right) \mathrm{d}x = \int_0^L e^0\,\mathrm{d}x = L$$

if k != k’ then $$\int_0^L \exp\left(i\frac{2\,\pi\,(k'-k)\,x}{L}\right) \mathrm{d}x = 0$$

# Fourier transforms. Concept

if k != k' then $$\int_0^L \exp\left(i\frac{2\,\pi\,(k'-k)\,x}{L}\right)\mathrm{d}x = 0$$

**WHY?**

You know from your analysis courses (otherwise ask me) that

$$\int e^{z\,x}\,\mathrm{d}x = \frac{e^{z\,x}}{z} \qquad\qquad z = a + i\,b$$

here $$z = \frac{i\,2\,\pi\,(k'-k)}{L}$$

Hence $$\int_0^L \exp\left(i\,\frac{2\,\pi\,(k'-k)\,x}{L}\right)\mathrm{d}x = \frac{L}{i\,2\,\pi\,(k'-k)}\left[\exp\left(i\,\frac{2\,\pi\,(k'-k)\,x}{L}\right)\right]_0^L =$$

$$\frac{L}{i\,2\,\pi\,(k'-k)}\left\{\exp\left[i\,2\,\pi\,(k'-k)\right]-1\right\} = 0$$

because $\exp\left(i\,2\,\pi n\right) = 1$ if *n* integer

# Fourier transforms. Concept

Hence the result of $$\int_0^L f(x)\,\exp\left(-i\,\frac{2\,\pi\,k\,x}{L}\right)\mathrm{d}x = \sum_{k'=-\infty}^{\infty}\gamma_{k'}\int_0^L \exp\left(i\frac{2\,\pi\,(k'-k)\,x}{L}\right)\mathrm{d}x$$

is $$\int_0^L f(x)\,\exp\left(-i\,\frac{2\,\pi\,k\,x}{L}\right)\mathrm{d}x = \gamma_k\,L$$

Or, re-writing in terms of $\gamma k$

$$\gamma_k = \frac{1}{L}\int_0^L f(x)\,\exp\left(-i\,\frac{2\,\pi\,k\,x}{L}\right)\mathrm{d}x$$

**Given f(x) we know the $\gamma k$ and**
**vice versa from the $\gamma k$ we can reconstruct f(x)**

# Fourier transforms. Discrete Fourier Transform (DFT)

**1. The integral** $\gamma_k = \frac{1}{L}\int_0^L f(x)\,\exp\left(-i\,\frac{2\,\pi\,k\,x}{L}\right)\mathrm{d}x$

**can be calculated analytically only for some f(x)**

**2. in many cases we do not know f(x) but just a sample $y_n = f(x_n)$**
**e.g. we have experimental data for $y_n$, not a functional form**

**→ it is important to calculate the $\gamma_k$ numerically**

**USE TRAPEZOIDAL RULE**

For $N$ slices of width $h = L/N$

$$\gamma_k = \frac{1}{L}\frac{L}{N}\left[\frac{1}{2}f(0) + \frac{1}{2}f(L) + \sum_{n=1}^{N-1} f(x_n)\,\exp\left(-i\,\frac{2\,\pi\,k\,x_n}{L}\right)\right]$$

where $x_n = \frac{n}{N}\,L$

Using f(0) = f(L) for periodicity, we simplify the sum

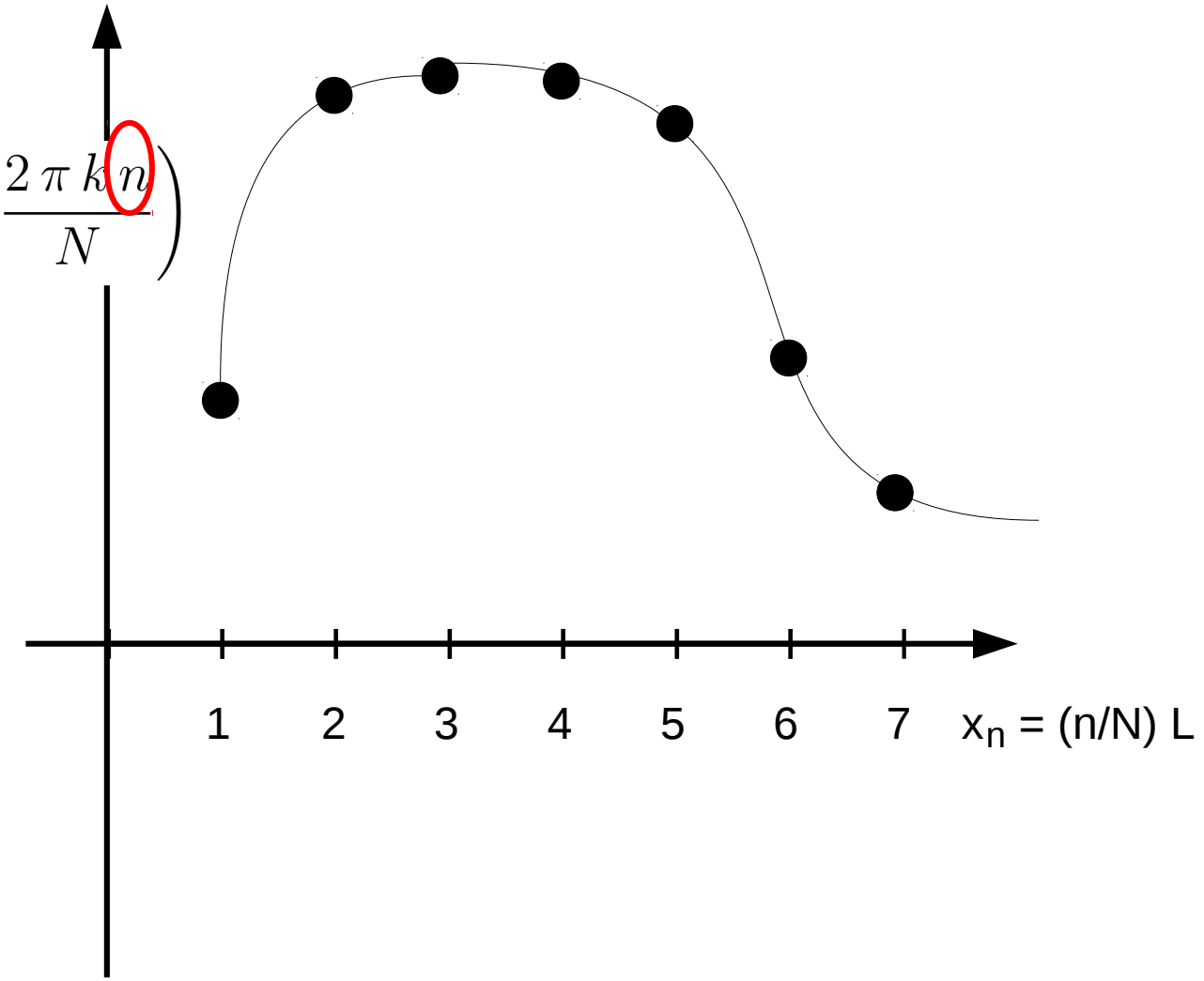
$$\gamma_k = \frac{1}{N}\sum_{n=0}^{N-1} f(x_n)\,\exp\left(-i\,\frac{2\,\pi\,k\,x_n}{L}\right)$$

# Fourier transforms. Discrete Fourier Transform (DFT)

$$\gamma_k = \frac{1}{N} \sum_{n=0}^{N-1} f(x_n) \exp\left(-i \frac{2\,\pi\,k\,x_n}{L}\right)$$

**If function f(x) is equally sampled over the interval,**
**i.e. the $x_n$ are equally spaced, we simplify $y_n = f(x_n)$ and $x_n = (n / N)\, L$**

$$\gamma_k = \frac{1}{N} \sum_{n=0}^{N-1} y_n \exp\left(-i \frac{2\,\pi\,k\,n}{N}\right)$$



# Fourier transforms. Discrete Fourier Transform (DFT)

$$\gamma_k = \frac{1}{N} \sum_{n=0}^{N-1} f(x_n) \exp\left(-i \frac{2\,\pi\,k\,x_n}{L}\right)$$

**If function f(x) is equally sampled over the interval,**
**i.e. the $x_n$ are equally spaced, we simplify $y_n = f(x_n)$ and $x_n = (n / N)\, L$**

$$\gamma_k = \frac{1}{N} \sum_{n=0}^{N-1} y_n \exp\left(-i \frac{2\,\pi\,k\,n}{N}\right)$$

**By convention, we use the following formula**

$$c_k = \sum_{n=0}^{N-1} y_n \exp\left(-i \frac{2\,\pi\,k\,n}{N}\right)$$

**where $c_k := \gamma_k N$**

**discrete Fourier transform (DFT) formula**

# Fourier transforms. Discrete Fourier Transform (DFT)

We derived this with trapezoidal rule: simplified method

However, with few mathematical steps we can show that
the DFT is, in a certain sense, exact:

with some math tricks we see that

$$y_n = \frac{1}{N} \sum_{k=0}^{N-1} c_k \, \exp\left( i \, \frac{2 \, \pi \, k \, n}{N} \right)$$

**Inverse discrete Fourier transform (inverse DFT)**

**given the coefficients $c_k$ we can recover the values of the samples $y_n$ that they come from exactly (except for rounding errors).**

# Fourier transforms. Discrete Fourier Transform (DFT)

**Even though Fourier coefficients are only approximate, they are actually exact in the sense that**
**we can completely recover the original samples from them**

**We can freely move back and fourth from the coefficients to the samples and vice versa WITHOUT LOSING ANY INFORMATION in our data by using**

$$c_k = \sum_{n=0}^{N-1} y_n \exp\left(-i \frac{2\pi k n}{N}\right)$$

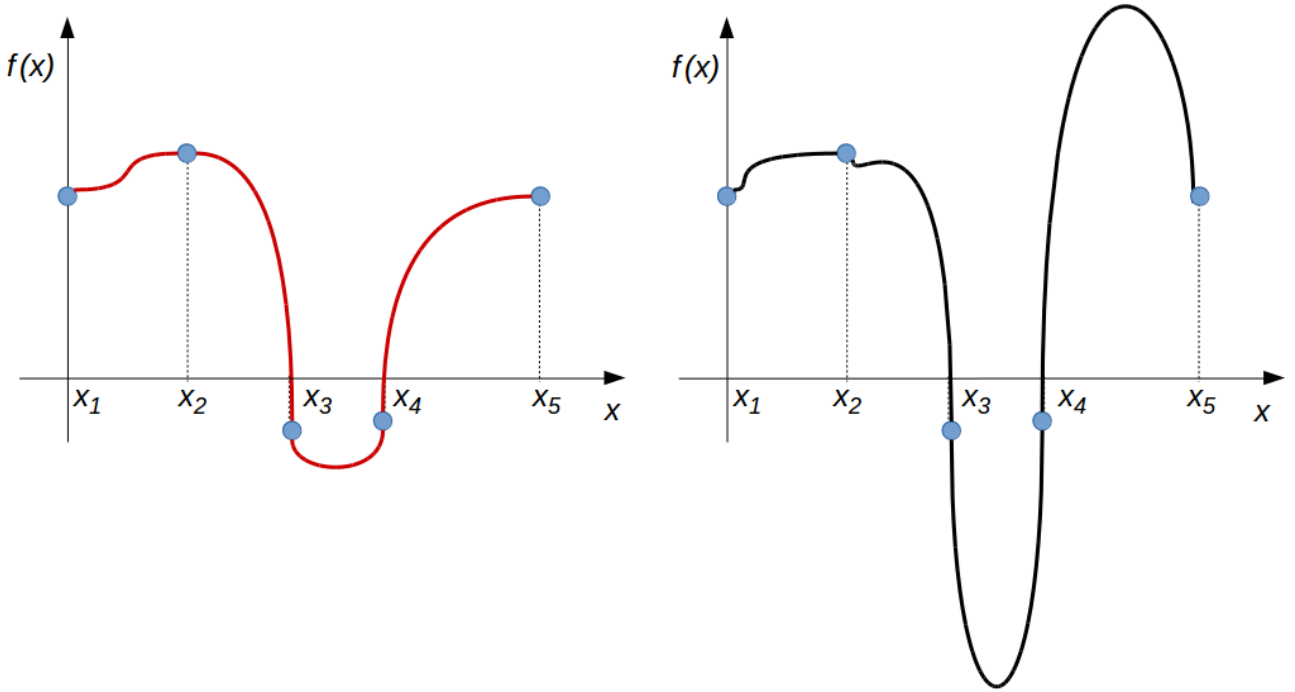
$$y_n = \frac{1}{N} \sum_{k=0}^{N-1} c_k \exp\left(i \frac{2\pi k n}{N}\right)$$

Notice that we only need the Fourier coefficients $c_k$ up to k = N – 1

to recover the samples, so 0<= k < N : no need to calculate an infinite sum

# Fourier transforms. Discrete Fourier Transform (DFT)

The inverse DFT equation only gives us the sample values $y_n = f(x_n)$:
it tells us nothing about the value of f(x) between sample points.




**Any two functions that have the same values $y_n$ at the sample points $x_n$ will have the same DFT,**
**no matter what they do between the sample points.**

**We cannot do better than this when we do not know the functional form of the function, but we have only a sample of discrete experimental data.**

# Fourier transforms. Discrete Fourier Transform (DFT)

**If REAL FUNCTIONS, further simplifications.**

Suppose all the $y_n$ are real and consider the value $c_k$ for *N/2 <k <N,*

which we write as *k = N – r* where *1<= r <= N/2*

$$c_{N-r} = \sum_{n=0}^{N-1} y_n \exp\left(-i\,\frac{2\,\pi\,(N-r)\,n}{N}\right)$$

$$= \sum_{n=0}^{N-1} y_n \exp\left(-i\,2\,\pi\,n\right) \exp\left(i\,\frac{2\,\pi\,r\,n}{N}\right)$$

$$= \sum_{n=0}^{N-1} y_n \exp\left(i\,\frac{2\,\pi\,r\,n}{N}\right) = c_r^*$$

where c* is the complex conjugate of c

This implies $c_{N-1} = c_1^*$

$c_{N-2} = c_2^*$

$c_{N-3} = c_3^*$

if f(x) is real, we calculate only the coefficients $c_k$ with *0 <= k <= N/2*

If N is even, we must calculate N/2+1 coefficients
if N is odd, we must calculate (N+1)/2

In python $N//2+1$

# Fourier transforms. Fast Fourier Transform (FFT)

**Computational cost of the DFT method: $N^2$,**
because we calculate *N/2+1* or (N+1)/2 coefficients and
for each coefficient we loop over *N* sampled points

→ **Scaling as $N\ (N//2+1) \sim N^2$**

**Fast Fourier transform (FFT) is faster**

Discovered in 1805 by Carl F. Gauss
Re-discovered independently in 1965 by James Cooley and John Tukey

**Assumes that the samples are a power of two, hence $N = 2^m$**
**with m integer.**

# Fourier transforms. Fast Fourier Transform (FFT)

**Divide the terms of the sum in the DFT formula into two equally sized groups.**

**- First group consists of the terms with *n* even, i.e. the terms with *n = 2 r* with *r = 0, 1,...N/2 – 1***

$$E_k = \sum_{r=0}^{\frac{N}{2}-1} y_{2\,r}\, \exp\left(-i\,\frac{2\,\pi k\,(2\,r)}{N}\right) = \sum_{r=0}^{\frac{N}{2}-1} y_{2\,r}\, \exp\left(-i\,\frac{2\,\pi k\,r}{\frac{1}{2}\,N}\right)$$

**is a Fourier transform of *N/2* samples**

**- Second group consists of the terms with *n* odd, i.e. the terms with *n = 2 r + 1***

$$\sum_{r=0}^{\frac{N}{2}-1} y_{2\,r+1}\, \exp\left(-i\,\frac{2\,\pi k\,(2\,r+1)}{N}\right) = \exp\left(-i\,\frac{2\,\pi k}{N}\right) \sum_{r=0}^{\frac{N}{2}-1} y_{2\,r+1}\, \exp\left(-i\,\frac{2\,\pi k\,r}{\frac{1}{2}\,N}\right) = \exp\left(-i\,\frac{2\,\pi k}{N}\right) O_k$$

**is another Fourier transform of *N/2* samples**

$$\longrightarrow \qquad c_k = E_k + \exp\left(-i\,\frac{2\,\pi k}{N}\right) O_k$$

# Fourier transforms. Fast Fourier Transform (FFT)

$$c_k = E_k + \exp\left(-i\,\frac{2\,\pi k}{N}\right) O_k$$

**The $c_k$ can be written as sum of $E_k$ and $O_k$,**
**both DFTs of f(x) but with half as many points**

**plus an extra-factor (TWIDDLE FACTOR):** $\exp\left(-i\,\frac{2\,\pi k}{N}\right)$

Repeat the splitting process onto each of the two Fourier transforms
Each of them can be divided in its even and its odd terms

**We repeat the splitting, until eventually we get to the point where each transform is the transform of just a single sample:**

$$c_0 = \sum_{n=0}^{0} y_n \, \exp\left(0\right) = y_0$$

# Fourier transforms. Fast Fourier Transform (FFT)

**The actual calculation of the FFT is the REVERSE of this reasoning:**

**We start from the individual samples and we combine them in pairs,**

**then we combine the pairs into fours,**

**the fours into eights, and so on,**

**creating larger and larger Fourier transforms, until we have reconstructed the full transform of the complete set of samples.**

## VERY FAST:

First round of the calculation: *N* samples,
Second round: *N/2* transforms with 2 coefficients each
Third round: *N/4* transforms with 4 coefficients each

→ ***N* operations per each level and *m* levels, where $m = \log_2(N)$,**

→ **FFT method scales as $N \log_2(N)$**

**DFT method scales as $N^2$**

# Fourier transforms. FFT in python

**module for FFTs in numpy: numpy.fft**

**1. numpy.fft.rfft:** FFT for real values

```
from numpy.fft import rfft
c=rfft(y)
```

returns N//2+1 values

**2. numpy.fft.irfft:** inverse FFT for real values

```
from numpy.fft import rfft, irfft
c=rfft(y)
y2 =ifft(c)
```

returns N values

**3. numpy.fft.rfft2:** FFT for real values in 2D

**4. numpy.fft.irfft2:** inverse FFT for real values in 2D

# Fourier transforms. FFT in python

**module for FFTs in numpy: numpy.fft**

**5. numpy.fft.fft:** FFT for complex values

**6. numpy.fft.ifft:** inverse FFT for complex values

**7. numpy.fft.fft2:** FFT for complex values in 2D

**8. numpy.fft.ifft2:** inverse FFT for complex values in 2D

# Fourier transforms. Physical interpretation

**Fourier transform represents a function via a set of real or complex sinusoidal waves.**

**Each term in the sum is a single wave with its own well-defined frequency $\nu = k/N$ and period $T = 1 / \nu$**

If f(x) is a function in space, then spatial frequencies,
if f(x) is a function in time, then temporal frequencies.

Saying that any function can be expressed as a Fourier transform is equivalent to saying that
**any function can be represented as a sum of waves of given frequencies**

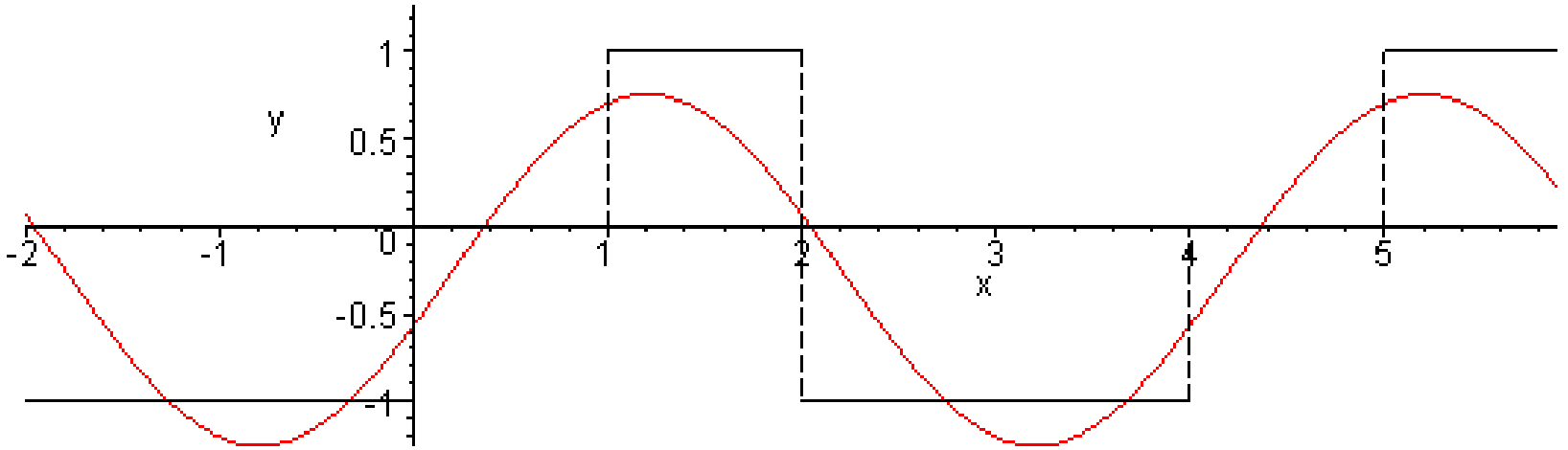
**and the coefficients of the Fourier transform tell us how much power is associated with each frequency, i.e. how big is the contribution of each frequency to the sum.**

# Fourier transforms. Physical interpretation

Saying that any function can be expressed as a Fourier transform
is equivalent to saying that
**any function can be represented as a sum of waves of given frequencies**

**and the coefficients of the Fourier transform tell us**
**how much power is associated with each frequency,**
**i.e. how big is the contribution of each frequency to the sum.**

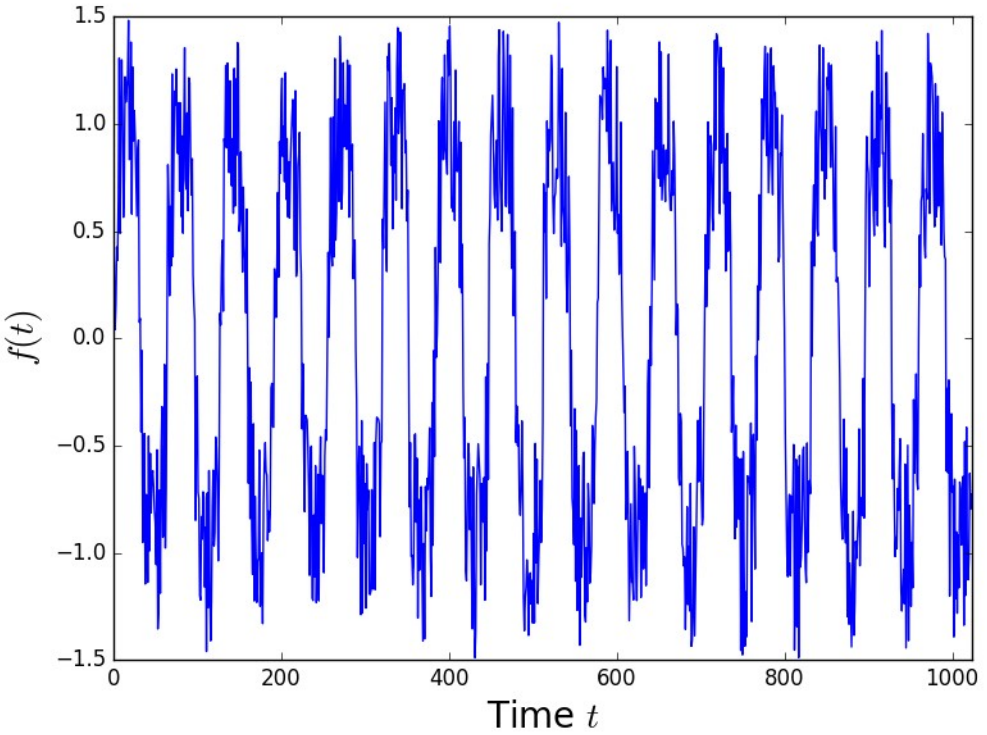
Example: Fourier representation of the piecewise function



# Fourier transforms. Physical interpretation

**Example:**
**examples/fourier/fft.py**

**signal consists of**
**a wave with**
**a well-defined frequency**
**and period,**
**but also some noise,**
**visible as smaller wiggles**
**in the line**




**We can calculate the Fourier transform of this signal as shown in examples/fourier/fft.py**

1. reads the file pitch.txt and plots
2. calculates the Fourier transform (numpy.fft.rfft)
3. plots the value of the absolute value of the coefficients $|c_k|$ versus k
4. calculates what is the frequency associated with the largest $|c_k|$

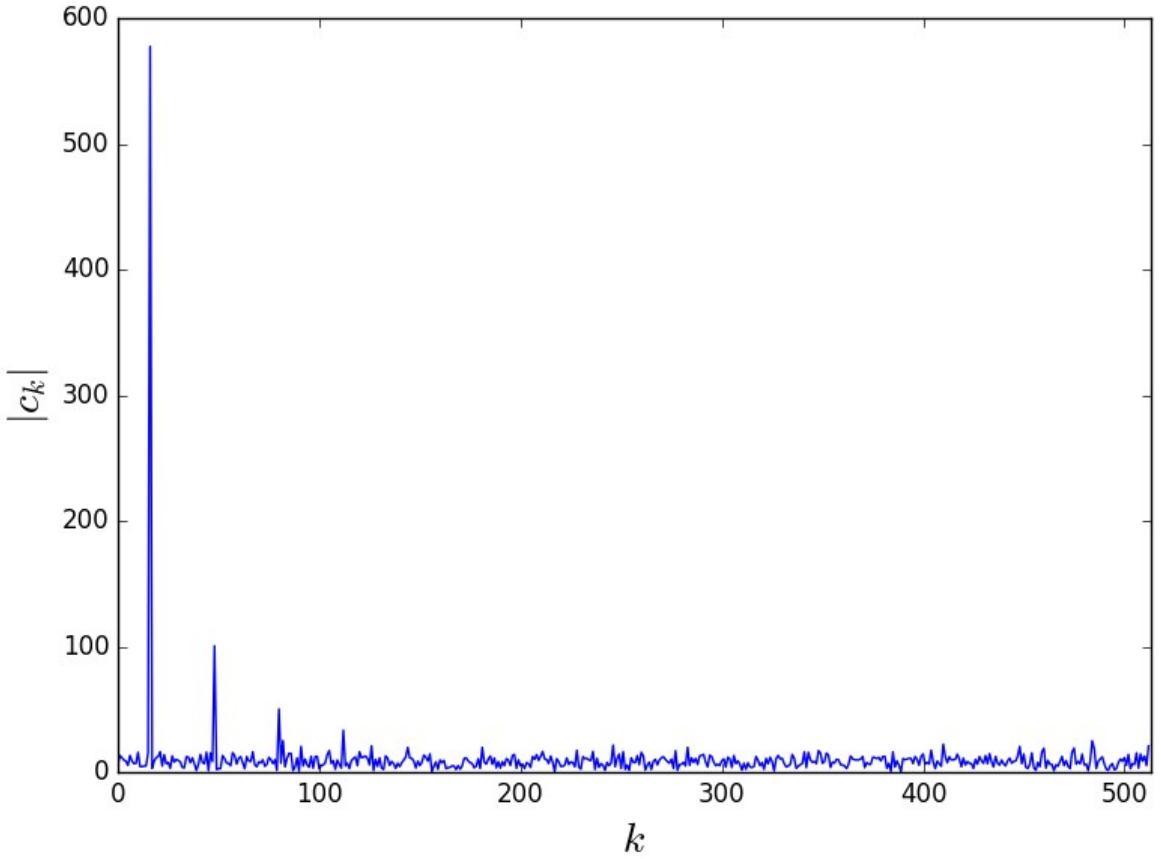
# Fourier transforms. Physical interpretation

**$|c_k|$ versus k**

**k proportional to the frequency of the waves $\nu = k/N$**

**Spikes in the plot**

**Main SPIKE**
**k = 16, $\nu \sim 0.0156$**
**$T = 1/\nu = 64$.**




**Main spike = main frequency and periodicity visible in Figure**

**Secondary spikes: HARMONICS**

**LOW-LEVEL BACKGROUND:** some small, random values of $|c_k|$.

Non-zero but very small $|c_k|$ are produced by some noise in the signal,

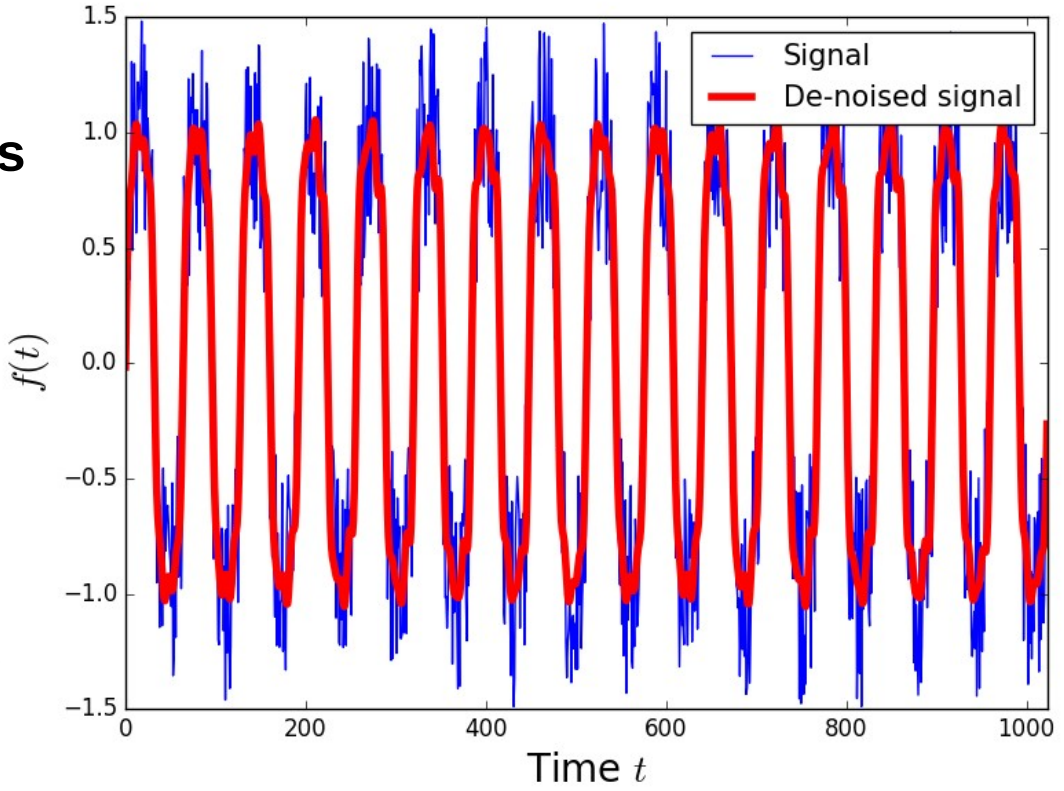
**WHITE NOISE (= does not show any dependence with k)**

# Fourier transforms. De-noising

**To get rid of the noise, reconstruct the initial samples without considering the smallest $|c_k|$ :**

cut the small $|c_k|$

**equivalent to applying a FILTER**




**Reconstruct original samples by doing the inverse FFT**
**after zeroing all $c_k$**
**with $|c_k| <= threshold \sim 25$**

Red: reconstructed signal after removing the noise-like coefficients.

Procedure works only if WHITE NOISE and
final result is sensitive to the choice of the threshold.

Not a tutorial on de-noising but very simplified example.

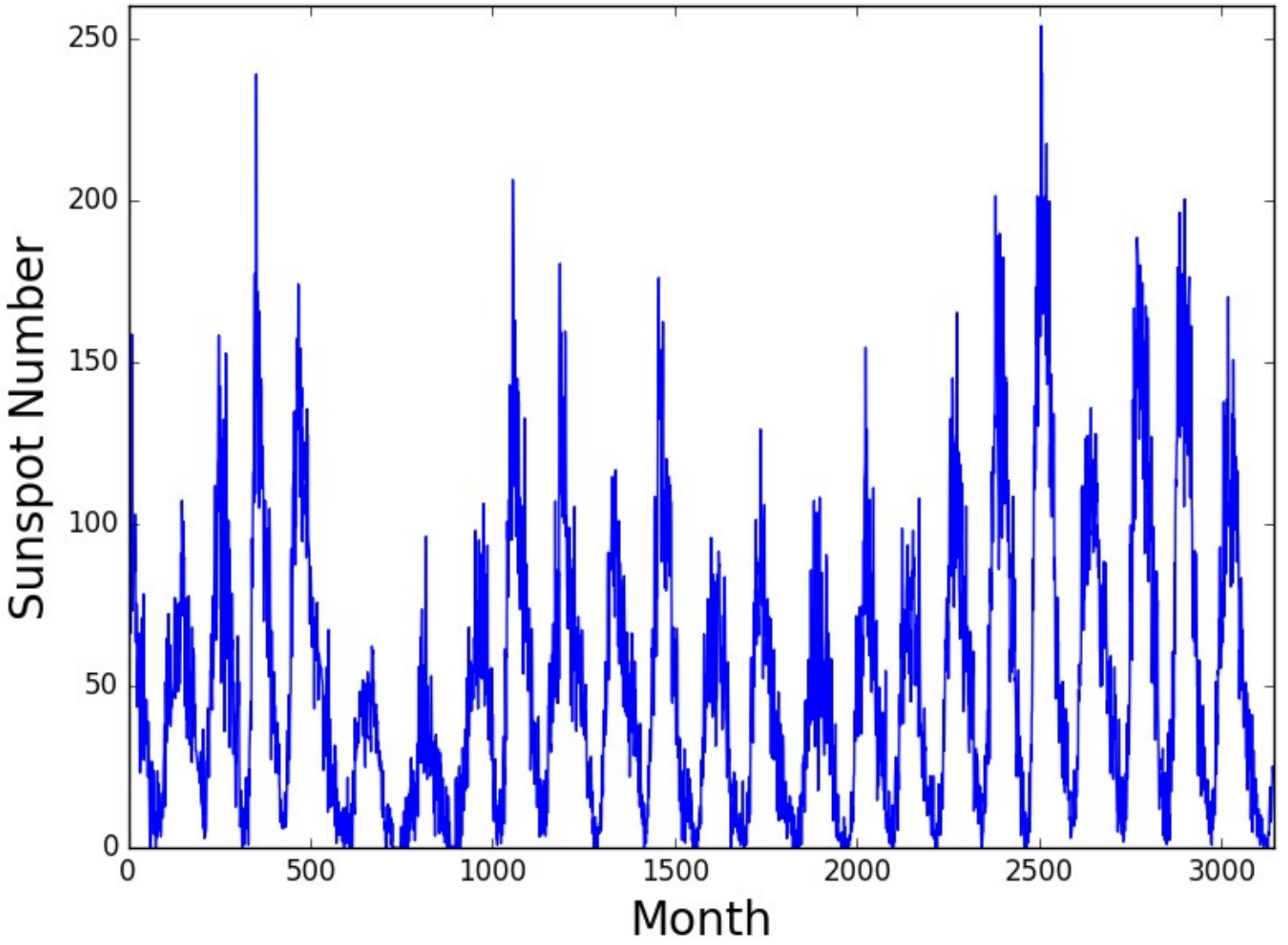
# Fourier transforms. Exercise

**EXERCISE:**

The file exercises/fourier/sunspots.txt (or, if you prefer to use gitlab: the file exercises/fourier/sunspots.txt) contains the observed number of sunspots on the Sun for each month since January 1749. The file contains two columns: column 0 is the month and column 1 is the number of sunspots per each month.

1. Plot the number of sunspots as a function of the month. The result should look like Figure 76.

# Fourier transforms. Exercise



# Fourier transforms. Exercise

**EXERCISE:**

The file exercises/fourier/sunspots.txt (or, if you prefer to use gitlab: the file exercises/fourier/sunspots.txt) contains the observed number of sunspots on the Sun for each month since January 1749. The file contains two columns: column 0 is the month and column 1 is the number of sunspots per each month.

1. Plot the number of sunspots as a function of the month. The result should look like Figure 76.
2. Write a script to perform the discrete Fourier transform (DFT) and the fast Fourier transform (FFT) of the number of sunspots as a function of the month. For the DFT, use equation 314. For the FFT, use the numpy.fft.rfft function. Compare the different performances. The DFT should be significantly slower than the FFT.

$$c_k = \sum_{n=0}^{N-1} y_n \exp\left(-i \frac{2\pi k n}{N}\right)$$

Note:

| | |
|---|---|
| Work with complex numbers | **from cmath import exp** |
| Define complex array | **ck =numpy.zeros(N//2+1,complex)** |
| *2 i* should be written as | **2j** |

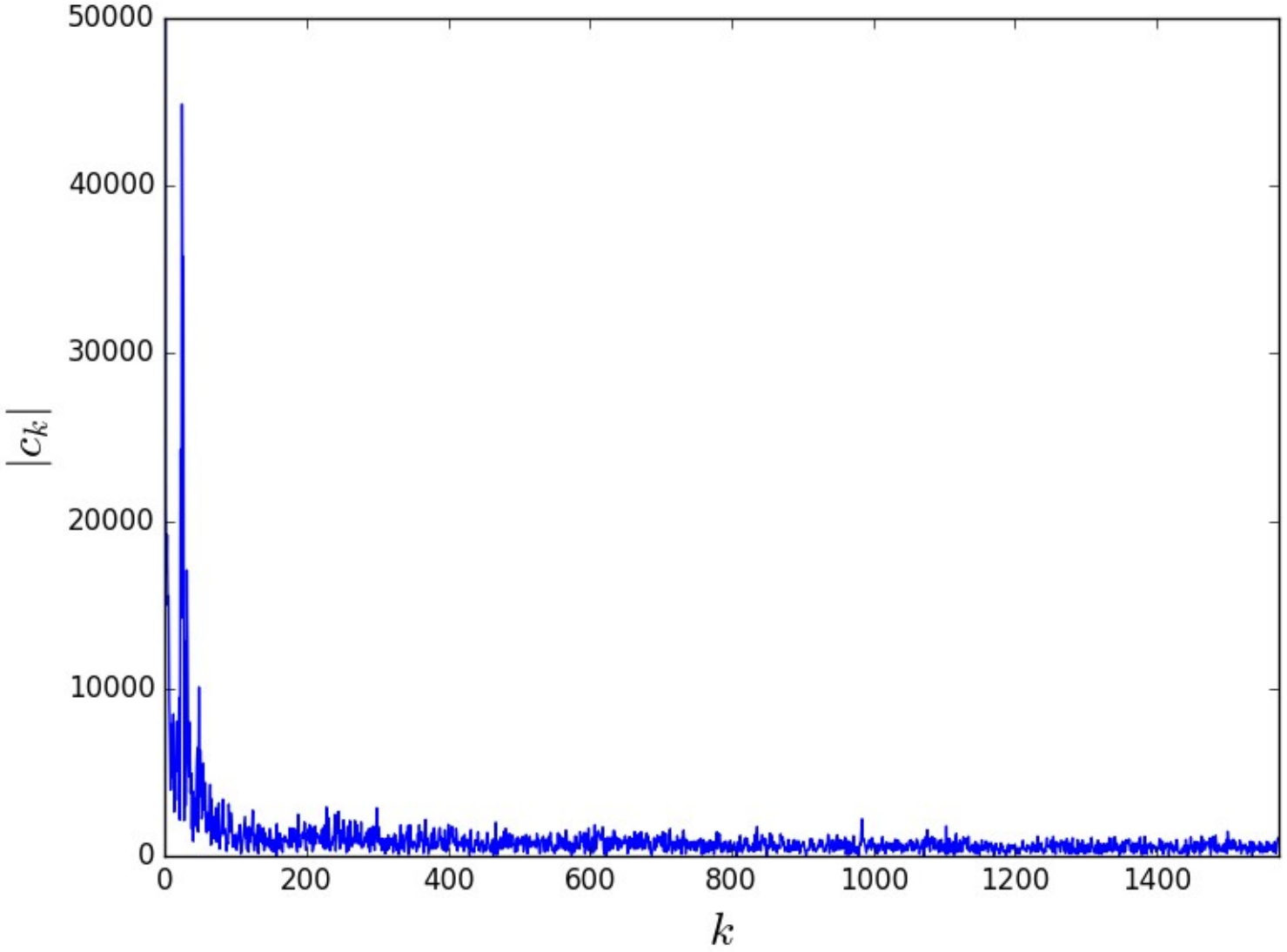
# Fourier transforms. Exercise

**EXERCISE:**

The file exercises/fourier/sunspots.txt (or, if you prefer to use gitlab: the file exercises/fourier/sunspots.txt) contains the observed number of sunspots on the Sun for each month since January 1749. The file contains two columns: column 0 is the month and column 1 is the number of sunspots per each month.

1. Plot the number of sunspots as a function of the month. The result should look like Figure 76.
2. Write a script to perform the discrete Fourier transform (DFT) and the fast Fourier transform (FFT) of the number of sunspots as a function of the month. For the DFT, use equation 314. For the FFT, use the numpy.fft.rfft function. Compare the different performances. The DFT should be significantly slower than the FFT.
3. Plot the Fourier coefficients $|c_k|$ as a function of $k$. The result should look like Figure 77.

# Fourier transforms. Exercise



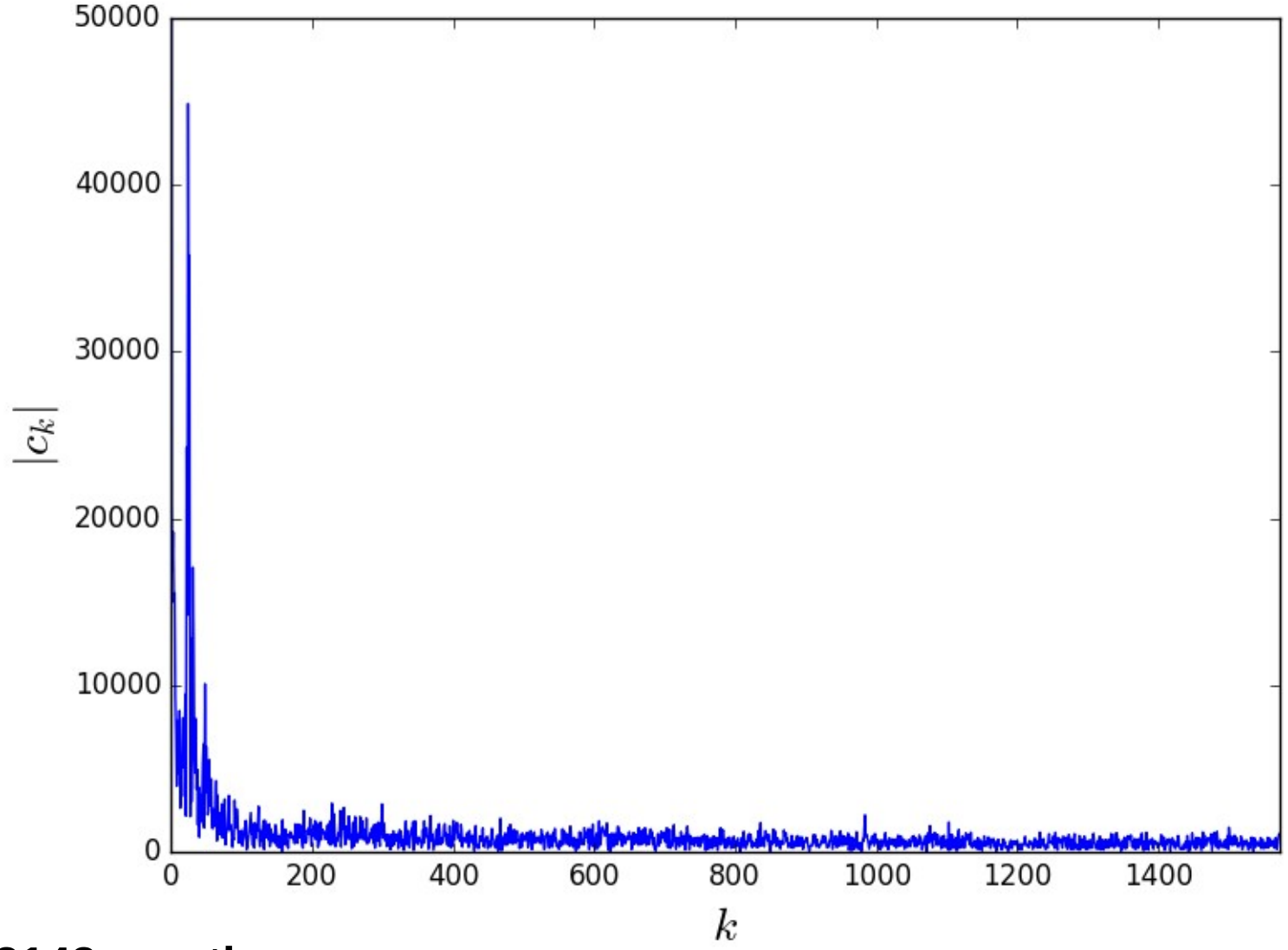
# Fourier transforms. Exercise

**EXERCISE:**

The file exercises/fourier/sunspots.txt (or, if you prefer to use gitlab: the file exercises/fourier/sunspots.txt) contains the observed number of sunspots on the Sun for each month since January 1749. The file contains two columns: column 0 is the month and column 1 is the number of sunspots per each month.

1. Plot the number of sunspots as a function of the month. The result should look like Figure 76.
2. Write a script to perform the discrete Fourier transform (DFT) and the fast Fourier transform (FFT) of the number of sunspots as a function of the month. For the DFT, use equation 314. For the FFT, use the numpy.fft.rfft function. Compare the different performances. The DFT should be significantly slower than the FFT.
3. Plot the Fourier coefficients $|c_k|$ as a function of $k$. The result should look like Figure 77.
4. Calculate the period $T = N/k$ associated with the largest $|c_k|$. This gives you the sunspot periodicity ($t \sim 10.9$ years).

# Fourier transforms. Exercise




**N = 3143 months**
**Frequency : k/N ~ 0.0076 months$^{-1}$**
**Period: T = N/K ~ 10.9 yr**

# Fourier transforms. Exercise

**EXERCISE:**

The file exercises/fourier/sunspots.txt (or, if you prefer to use gitlab: the file exercises/fourier/sunspots.txt) contains the observed number of sunspots on the Sun for each month since January 1749. The file contains two columns: column 0 is the month and column 1 is the number of sunspots per each month.

1. Plot the number of sunspots as a function of the month. The result should look like Figure 76.
2. Write a script to perform the discrete Fourier transform (DFT) and the fast Fourier transform (FFT) of the number of sunspots as a function of the month. For the DFT, use equation 314. For the FFT, use the numpy.fft.rfft function. Compare the different performances. The DFT should be significantly slower than the FFT.
3. Plot the Fourier coefficients $|c_k|$ as a function of $k$. The result should look like Figure 77.
4. Calculate the period $T = N/k$ associated with the largest $|c_k|$. This gives you the sunspot periodicity ($t \sim 10.9$ years).
5. Try to denoise the data assuming that there is just white noise. Plot the denoised signal. The result should look like Figure 78. Note: for the inverse Fourier transform it is sufficient that you use the corresponding numpy function (no need to write an indirect DFT).

# Fourier transforms. Exercise