SCRIPTING AND PROGRAMMING LABORATORY FOR DATA ANALYSIS

Lecture 7a – Introduction to scipy – part 2

Scipy

The *Scientific Python* *(SciPy)* package contains several tools dedicated to most part of the problems encountered in scientific research, like:

- Interpolation
- Integration
- Optimisation
- Special functions
- Linear algebra
- Fourier transform
- ...

Before implementing an algorithm by yourself, check the scipy doc
DO NOT REINVENT THE WHEEL!

As *Numpy*, also *Scipy* is largely written in Fortran or C, making it very computationally efficient and optimised.

Scipy - Linear algebra

Both NumPy and SciPy, have extensive tools for numerically solving problems in linear algebra. Some that you can find at most are

- solving a system of linear equations
- eigenvalue problems

Both rely on **`scipy.linalg`**

Scipy-Linear algebra

Both NumPy and SciPy, have extensive tools for numerically solving … bra. Some that you can find at most …

- solv…

```
import scipy.linalg
a = array([[-2, 3], [4, 5]])
a
array([[-2,  3],
       [ 4,  5]])

scipy.linalg.det(a)
-22.0
```

Compute the determinant

Invert a matrix

```
b = scipy.linalg.inv(a)

b
array([[-0.22727273,  0.13636364],
       [ 0.18181818,  0.09090909]])

dot(a,b)
array([[ 1.,  0.],
       [ 0.,  1.]])
```

Check the inversion

Scipy-Linear algebra

Both NumPy and SciPy, have extensive tools for numerically solving problems in linear algebra. Some that you can find at most are

- solving a system of linear equations

$$\begin{aligned} 2x_1 + 4x_2 + 6x_3 &= 4 \\ x_1 - 3x_2 - 9x_3 &= -11 \\ 8x_1 + 5x_2 - 7x_3 &= 1 \end{aligned}$$

$$A = \begin{pmatrix} 2 & 4 & 6 \\ 1 & -3 & -9 \\ 8 & 5 & -7 \end{pmatrix}, \quad \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} 4 \\ -11 \\ 1 \end{pmatrix}$$

The function ***linalg.solve*** *finds the solution of the linear system*

```
A = array([[2, 4, 6], [1, -3, -9], [8, 5, -7]])
b = array([4, -11, 2])
```

```
scipy.linalg.solve(A,b)
array([ -8.91304348,  10.2173913 ,  -3.17391304])
```

Scipy-Linear algebra

Both NumPy and SciPy, have extensive tools for numerically solving problems in linear algebra. Some that you can find at most are

- eigenvalue problems

One of the most common problems in science and engineering is the eigenvalue problem

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$$

where **A** is a square matrix, **x** is a column vector, and **lambda** is a scalar.

SCIPY-LINEAR ALGEBRA

Both NumPy and SciPy, have exte
solving problems in linear alge
at most are

- eigenvalue problems

One of the most common problems
is the eigenvalue problem

Ax

where **A** is a square matrix, **x**
is a scalar.

```
A = array([[2, 4, 6],[1, -3, -9],[8, 5, -7]])

In [15]: A
Out[15]: array([[ 2,  4,  6],
                [ 1, -3, -9],
                [ 8,  5, -7]])

In [16]: lam, evec = scipy.linalg.eig(A)

In [17]: lam
Out[17]: array([ 2.40995356+0.j, -8.03416016+0.j,
                -2.37579340+0.j])

In [18]: evec
Out[18]: array([[-0.77167559, -0.52633654,  0.57513303],
                [ 0.50360249,  0.76565448, -0.80920669],
                [-0.38846018,  0.36978786,  0.12002724]])
```

Note that eigenvalues are generally complex

*The function **linalg.eig** returns an array of eigenvalues and a matrix whose rows are the eigenvectors*

Scipy - Fourier transform

The Fourier Transform is a tool that breaks a waveform (a function or signal) into an alternate representation, characterized by the sine and cosine functions of varying frequencies and highlighting the frequencies that make up the signal.

The Fourier Transform shows that any waveform can be rewritten as the sum of sinusoids. (check out here for a simple intuitive explanation)

$$f(x) = \int_{-\infty}^{\infty} F(k)\, e^{2\pi i k x}\, dk$$ *Inverse transform (+i)*

$$F(k) = \int_{-\infty}^{\infty} f(x)\, e^{-2\pi i k x}\, dx.$$ *Forward transform (-i)*

Scipy - Fourier transform

Naive interpretation:

The complex exponential is an oscillatory function (i.e. cos +i sin). For a given **k**, only when **f(x)** oscillates in a “similar way” within an x interval, the integral evaluates in a non-zero number. Otherwise **F(k)** will be essentially null.

In this sense the fourier transform selects those frequencies at which **f(x)** oscillates

$$F(k) = \int_{-\infty}^{\infty} f(x)\, e^{-2\pi i k x}\, dx.$$

SciPy - Fourier transform

$$f(x) = \int_{-\infty}^{\infty} F(k)\, e^{2\pi i k x}\, dk$$

$$F(k) = \int_{-\infty}^{\infty} f(x)\, e^{-2\pi i k x}\, dx.$$

Being an integral, you may think to compute it with standard techniques of numerical integration (when no analytical solution exists).

Of course you can, but it will be quite inefficient!

You need to compute a lot of integrals, since **F(k)** is actually a function of **k**, i.e. “the frequencies”.

Moreover, if the function f(x) is not “continuous”, but a tabulated series, resorting to a discrete version of the fourier transform is much more convenient (and the only possible way on a computer with finite memory!).

Scipy-Fourier transform

We sample a time series with **N** uniformly spaced points

$$h_k \equiv h(t_k), \qquad t_k \equiv k\Delta, \qquad k = 0, 1, 2, \ldots, N-1$$

This sampling choice can give us information at **N** frequencies

$$f_n \equiv \frac{n}{N\Delta}, \qquad n = -\frac{N}{2}, \ldots, \frac{N}{2}$$

The continuous fourier transform becomes a **discrete fourier transform:**

$$H(f_n) = \int_{-\infty}^{\infty} h(t) e^{-2\pi i f_n t} dt \approx \sum_{k=0}^{N-1} h_k \, e^{-2\pi i f_n t_k} \Delta = \Delta \sum_{k=0}^{N-1} h_k \, e^{-2\pi i k n/N}$$

SciPy - Fourier transform

DFT can be computed in a very efficient way through the Fast Fourier Transform (FFT) algorithm (Cooley-Tuckey 1965)

1-D discrete Fourier transforms

Function samples

The FFT *y[k]* of length N of the length-N sequence *x[n]* is defined as

DFT, where k also ranges from 0 to N-1

$$y[k] = \sum_{n=0}^{N-1} e^{-2\pi j \frac{kn}{N}} x[n],$$

scipy.fft - forward transform

*2*pi*j*k * n/N*
N = len(data)

and the inverse transform is defined as follows

Normalisation!!!

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} e^{2\pi j \frac{kn}{N}} y[k].$$

scipy.ifft - inverse transform

Scipy - Fourier transform

DFT can be computed in a very efficient way through the Fast Fourier Transform (FFT) algorithm (Cooley-Tuckey 1965)

1-D discrete Fourier transforms

The FFT *y[k]* of length N of the length-N sequence *x[n]* is defined as

$$y[k] = \sum_{n=0}^{N-1} e^{-2\pi j \frac{kn}{N}} x[n] \,,$$

and the inverse transform is defined as follows

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} e^{2\pi j \frac{kn}{N}} y[k] \,.$$

Remind that the FFT is complex even if the original input was real! In this case the negative frequencies are simply the complex conjugate and generally are ignored!

$$y[0] = \sum_{n=0}^{N-1} x[n] \,.$$

Scipy - Fourier transform

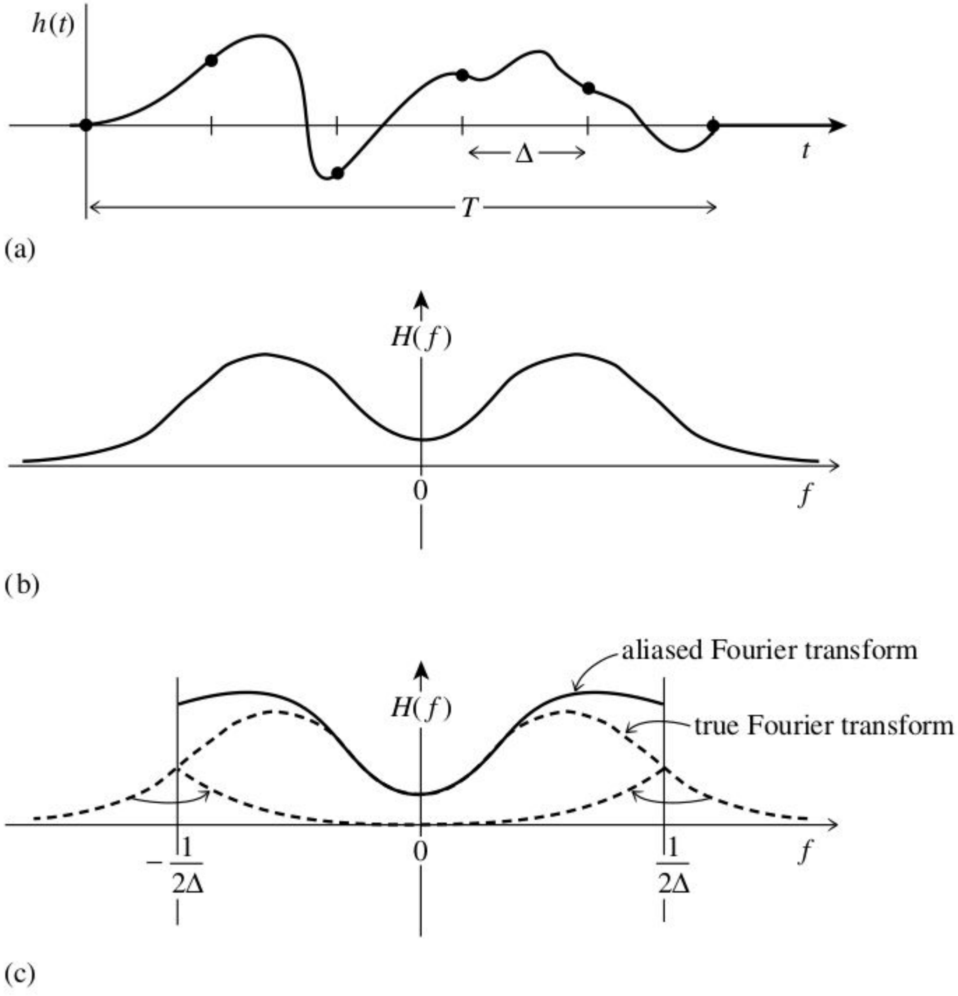
Subtleties:

- The maximum frequency you can survey depends on the sampling rate, the Nyquist frequency $$f_c \equiv \frac{1}{2\Delta}$$
- The frequency resolution depends on the total length **T** of your signal
- The FFT assumes that your sample is **N-periodic**

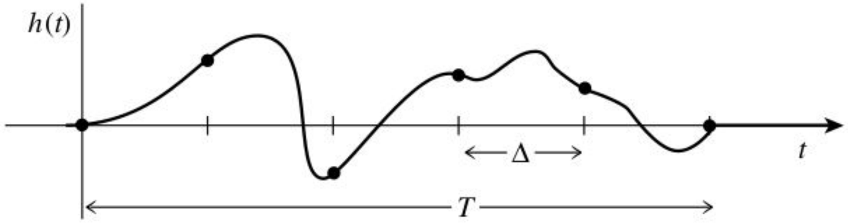
The above statements generate possible issues:

- **Aliasing:** when you sample rate is not high enough, you cannot get information about the high part of the frequency spectrum; power contained into high frequencies is “folded back” into your spectrum

Scipy-Fourier transform

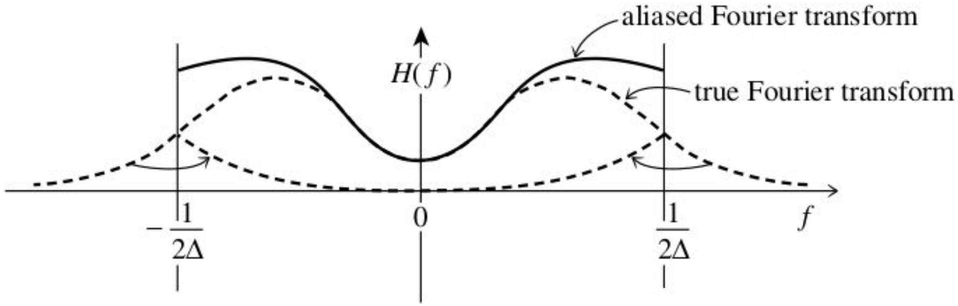


Scipy-Fourier transform



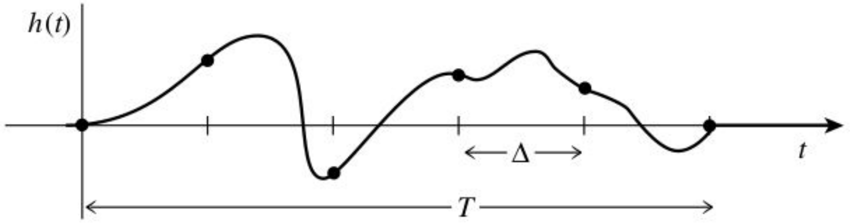
(a)

Figure 12.1.1. The continuous function shown in (a) is nonzero only for a finite interval of time T. It follows that its Fourier transform, whose modulus is shown schematically in (b), is not bandwidth limited but has finite amplitude for all frequencies. If the original function is sampled with a sampling interval Δ, as in (a), then the Fourier transform (c) is defined only between plus and minus the Nyquist critical frequency. Power outside that range is folded over or "aliased" into the range. The effect can be eliminated only by low-pass filtering the original function *before sampling*.



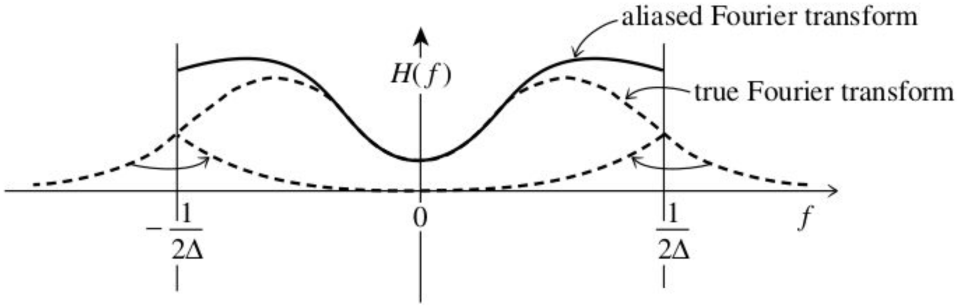
(c)

Scipy-Fourier transform



(a)

Figure 12.1.1. The continuous function shown in (a) is nonzero only for a finite interval of time T. It follows that its Fourier transform, whose modulus is shown schematically in (b), is not bandwidth limited but has finite amplitude for all frequencies. If the original function is sampled with a sampling interval Δ, as in (a), then the Fourier transform (c) is defined only between plus and minus the Nyquist critical frequency. Power outside that range is folded over or "aliased" into the range. The effect can be eliminated only by low-pass filtering the original function *before sampling*.



(c)

Possible solution:
increase the sampling rate

Scipy - Fourier transform

Subtleties:

- The maximum frequency you can survey depends on the sampling rate, the Nyquist frequency
- The frequency resolution depends on the total length **T** of your signal
- The FFT assumes that your sample is **N-periodic**

$$f_c \equiv \frac{1}{2\Delta}$$

The above statements generate possible issues:

- **Leakage:** because of the assumption of N-periodicity, if your time series “ends” at non-integer fraction of the period, then spurious power is added at several frequency in your spectrum

SCIPY - FOURIER TRANSFORM

Subtleties:

- The maximum frequency you can survey depends on the sampling rate, the Nyquist frequency
- The frequency resolution depends on the total length **T** of your signal
- The FFT assumes that your sample is **N-periodic**

$$f_c \equiv \frac{1}{2\Delta}$$

The above statements generate possible issues:

- **Leakage:** because of the assumption of N-periodicity, if your time series “ends” at non-integer fraction of the period, then spurious power is added at several frequency in your spectrum

Possible solution:

Use window functions (see numpy doc)

Scipy - Fourier transform

Basic usage:

```
t = np.linspace(0,T,N)
dt = t[1]-t[0]
samples = time_series(t)


FT = scipy.fft.fft(samples)
f = scipy.fft.fftfreq(N, dt)
```

Get a uniformly spaced time series with a specific dt

Perform the forward transformation

Get the frequencies based on the chosen sampling rate

SCRIPTING AND PROGRAMMING LABORATORY FOR DATA ANALYSIS

Lecture 7b – Brief intro to pandas

PANDAS

The **pandas** package is one of the most important tool at the disposal of Data Scientists.

It provides fast, flexible and expressive data structures designed to make working with **“relational”** or **“labeled”** data both easy and intuitive.

It aims to be the fundamental high-level building block for doing practical, real-world data analysis in Python.

PANDAS

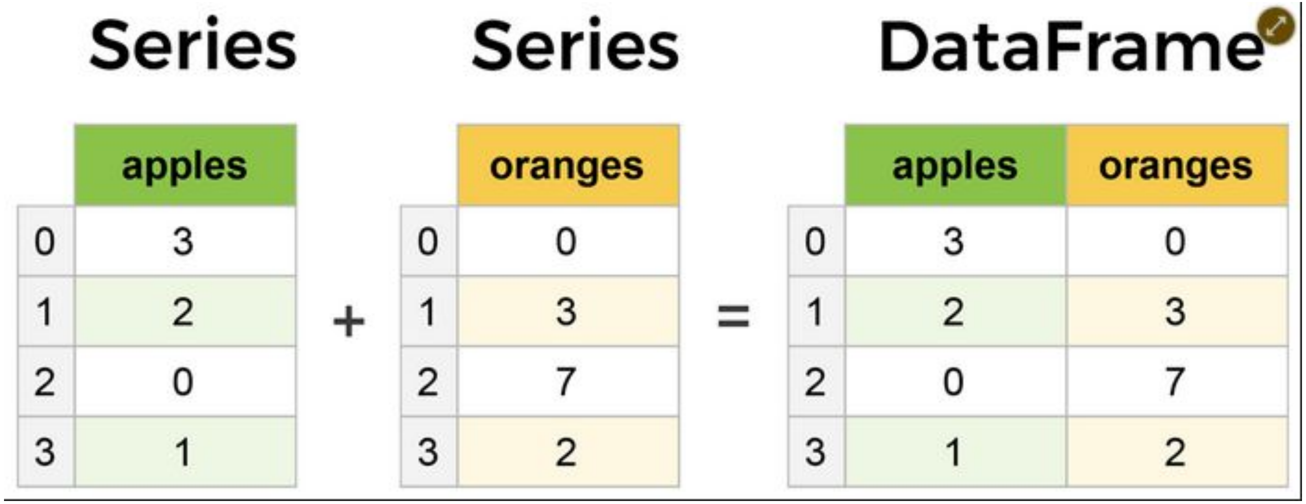
The **pandas** package is one of the most important tool at the disposal of Data Scientists pandas is well suited for many different kinds of data:

- Tabular data with heterogeneously-typed columns, as in an SQL table or Excel spreadsheet
- Ordered and unordered (not necessarily fixed-frequency) time series data.
- Arbitrary matrix data (homogeneously typed or heterogeneous) with row and column labels
- Any other form of observational / statistical data sets. The data need not be labeled at all to be placed into a pandas data structure

PANDAS

The primary two components of pandas are the **Series** and **DataFrame**.

A **Series** is essentially a column, and a **DataFrame** is a multi-dimensional table made up of a collection of Series.



PANDAS

There are many ways to create a DataFrame from scratch, but a great option is to just use a simple dict.

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```python
data = {
    'apples': [3, 2, 0, 1],
    'oranges': [0, 3, 7, 2]
}
```

```python
purchases = pd.DataFrame(data)
purchases
```

	apples	oranges
0	3	0
1	2	3
2	0	7
3	1	2

```python
purchases = pd.DataFrame(data, index=['June', 'Robert', 'Lily', 'David'])
purchases
```

	apples	oranges
June	3	0
Robert	2	3
Lily	0	7
David	1	2

We give a name to the row

PANDAS

There are many ways to create a DataFrame from scratch, but a great option is to just use a simple dict.

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```python
data = {
    'apples': [3, 2, 0, 1],
    'oranges': [0, 3, 7, 2]
}
```

```python
purchases = pd.DataFrame(data)
purchases
```

	apples	oranges
0	3	0

```python
purchases["oranges"]
```

```
June      0
Robert    3
Lily      7
David     2
Name: oranges, dtype: int64
```

```python
purchases = pd.DataFrame(data, index=['June', 'Robert', 'Lily', 'David'])
purchases
```

	apples	oranges
June	3	0
Robert	2	3
Lily	0	7
David	1	2

We give a name to the row

```python
purchases.loc['June']
```

```
apples     3
oranges    0
Name: June, dtype: int64
```

Dict like access

PANDAS

Usually you are going to read from a text file, e.g. CSV file (file that Excel can read: comma separated file)

With CSV files all you need is a single line to load in the data:

```
df = pd.read_csv('purchases.csv')
```

```
df
```

	apples	oranges
June	3	0
Robert	2	3
Lily	0	7
David	1	2

PANDAS

Choose explicitly which is the header

```
df = pd.read_csv("PS_2022.04.02_06.32.48.csv",header=93)
```

```
df.head()
```

	pl_name	hostname	default_flag	sy_snum	sy_pnum	discoverymethod	disc_year	disc_facility	soltype	pl_controv_flag	...
0	11 Com b	11 Com	1	2	1	Radial Velocity	2007	Xinglong Station	Published Confirmed	0	...
1	11 Com b	11 Com	0	2	1	Radial Velocity	2007	Xinglong Station	Published Confirmed	0	...
2	11 UMi b	11 UMi	0	1	1	Radial Velocity	2009	Thueringer Landessternwarte Tautenburg	Published Confirmed	0	...
3	11 UMi b	11 UMi	1	1	1	Radial Velocity	2009	Thueringer Landessternwarte Tautenburg	Published Confirmed	0	...
4	11 UMi b	11 UMi	0	1	1	Radial Velocity	2009	Thueringer Landessternwarte Tautenburg	Published Confirmed	0	...

5 rows × 87 columns

Shows the structure of your data from the top

df.tail does the same from bottom

PANDAS

Choose explicitly which is the header

```
df["banana"] = [0,2,4,7]
del df["apples"]
```

```
df
```

	oranges	banana
June	0	0
Robert	3	2
Lily	7	4
David	2	7

Add new column

Delete a column

```
df.iloc[0]
```

```
apples     3
oranges    0
Name: June, dtype: int64
```

```
df.loc["June"]
```

```
apples     3
oranges    0
Name: June, dtype: int64
```

Select rows by index or name

PANDAS

Look into your data

```
df.describe()
```

Statistical analysis

	apples	oranges
count	4.000000	4.00000
mean	1.500000	3.00000
std	1.290994	2.94392
min	0.000000	0.00000
25%	0.750000	1.50000
50%	1.500000	2.50000
75%	2.250000	4.00000
max	3.000000	7.00000

```
df.corr()
```

Correlate variables

	apples	oranges
apples	1.000000	-0.877058
oranges	-0.877058	1.000000

PANDAS

And many other things, check the documentation or ask google how can you do something with pandas. Also check here and here and here for tutorials!

In particular explore the connections with other packages like numpy and matplotlib.

Just practice by yourself, happy pandas!