JUAS and MADX in python

For this course a **basic knowledge of Python is assumed**, therefore if you are not familiar with it you can find, in the following sections, few resources to fill the gap. During the course we will use Python3 in a Jupyter notebook and, mostly, the *numpy*, *matplotlib*, *pandas*, *sympy*, *PyNAFF* and *cpymad* packages. We will explain in the following sections how to install this software on your laptops.

After a short introduction where we provided some useful links to get familiar with Python, we will focus on the software setup. Depending on your operating systems (we will consider OSX, Windows and UNIX) you have different procedures to follow.

For **OSX** users, please follow the instructions in the *OSX: Install and run a Docker image* section.

For the **Windows** users, please follow the instructions in the *Windows: Docker Toolbox* section.

For **UNIX** users, please follow the instructions in the instructions in the *UNIX: Anaconda + cpymad* section.

A (very) short introduction to Python

Test Python on a web page

If you are not familiar with Python and you have not it installed on your laptop, you can start playing with simple python snippets on the web: without installing any special software you can connect, e.g., to

https://www.pythonanywhere.com/try-ipython/ (https://www.pythonanywhere.com/try-ipython/)

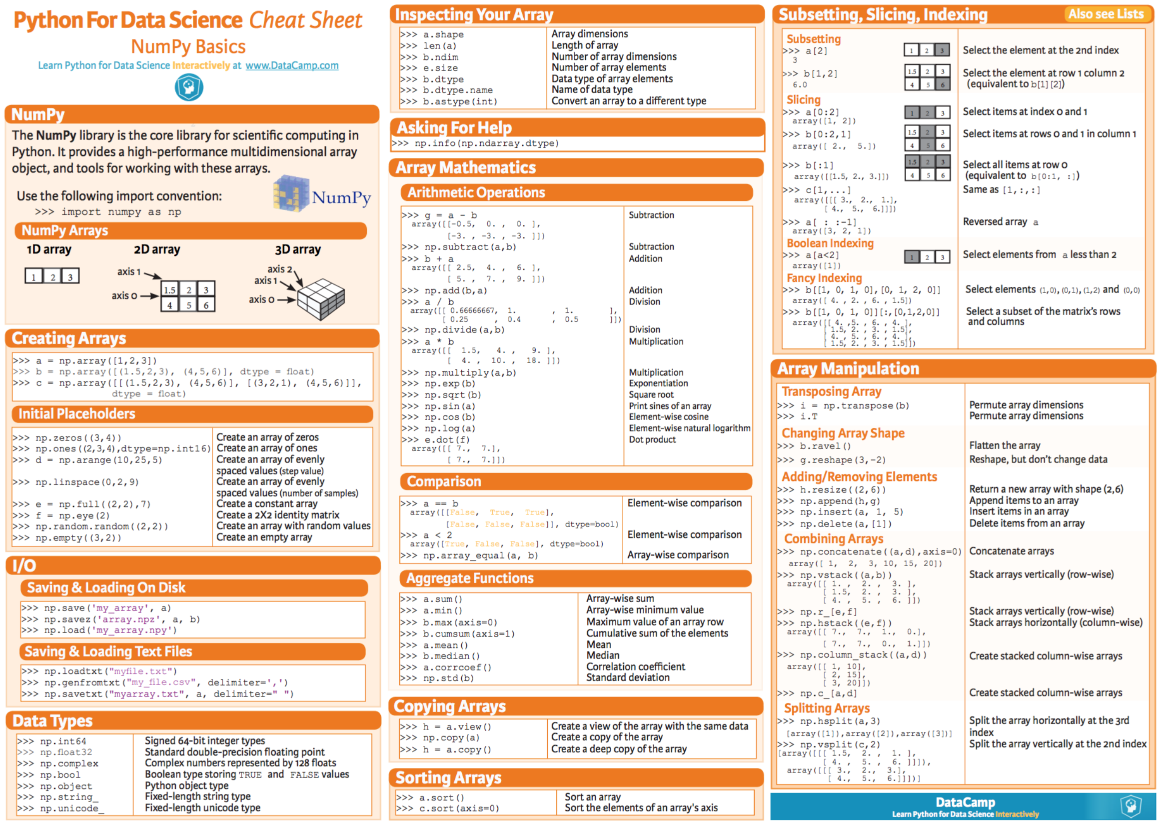
and test the following commands

```
import numpy as np
# Matrix definition
Omega=np.array([[0, 1],[-1,0]])
M=np.array([[1, 0],[1,1]])

# Sum and multiplication of matrices
Omega - M.T @ Omega @ M
# M.T means the "traspose of M".

# Function definition
def Q(f=1):
    return np.array([[1, 0],[-1/f,1]])

#Eigenvalues and eigenvectors
np.linalg.eig(M)
```

You can compare and check your output with the ones here (https://cernbox.cern.ch/index.php/s/xipyXzX7V9KBJbl).

The *numpy* package

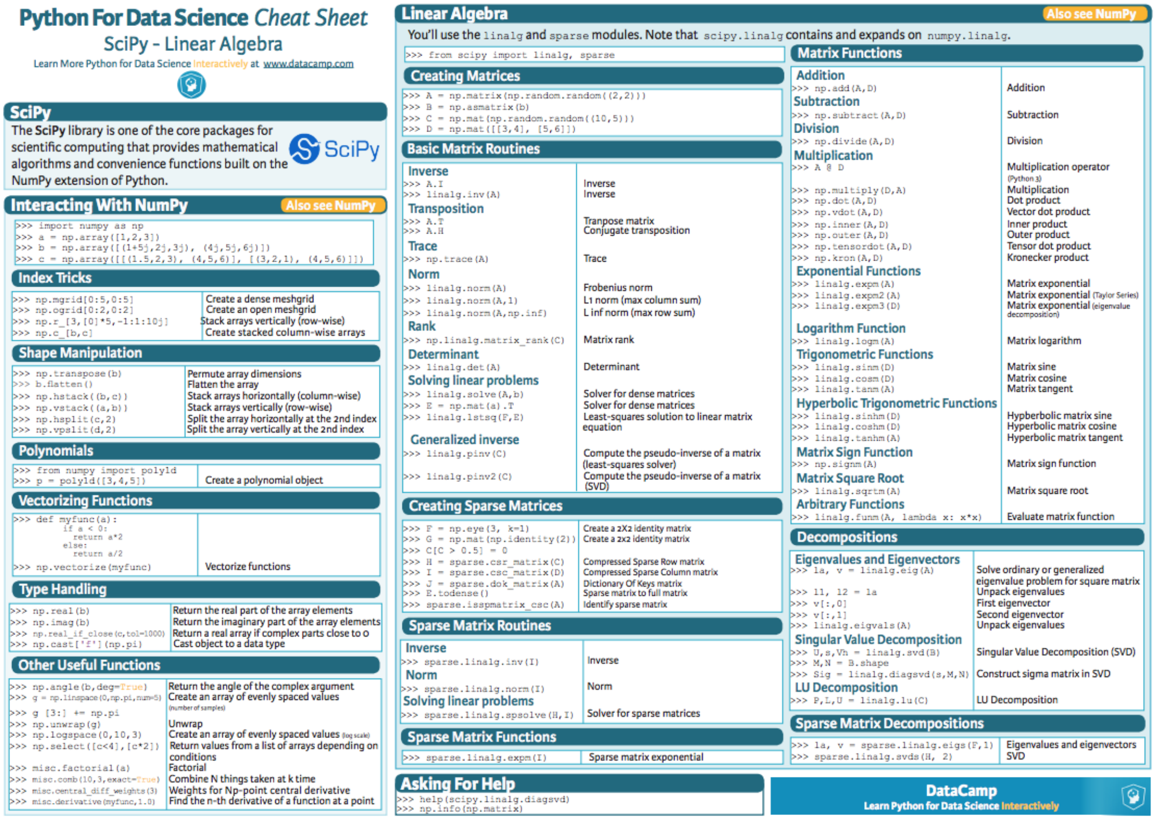
To get familiar with the *numpy* package have a look at the following summary poster (https://s3.amazonaws.com/assets.datacamp.com/blog_assets/Numpy_Python_Cheat_Sheet.pdf).



You can google many other resources, but the one presented of the poster cover the set of instructions you should familiar with.

The linalg module

To get familiar with the Linear Algebra (linalg) module have a look at the following summary poster (https://s3.amazonaws.com/assets.datacamp.com/blog_assets/Python_SciPy_Cheat_Sheet_Linear_Algebra.pdf).



Python For Data Science Cheat Sheet

SciPy - Linear Algebra

Learn More Python for Data Science Interactively at www.datacamp.com

SciPy

The SciPy library is one of the core packages for scientific computing that provides mathematical algorithms and convenience functions built on the NumPy extension of Python.

Interacting With NumPy

Also see NumPy

```
>>> import numpy as np
>>> a = np.array([1,2,3])
>>> b = np.array([(1+5j,2j,3j), (4j,5j,6j)])
>>> c = np.array([[(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)]])
```

Index Tricks

`>>> np.mgrid[0:5,0:5]`	Create a dense meshgrid
`>>> np.ogrid[0:2,0:2]`	Create an open meshgrid
`>>> np.r_[3,[0]*5,-1:1:10j]`	Stack arrays vertically (row-wise)
`>>> np.c_[b,c]`	Create stacked column-wise arrays

Shape Manipulation

`>>> np.transpose(b)`	Permute array dimensions
`>>> b.flatten()`	Flatten the array
`>>> np.hstack((b,c))`	Stack arrays horizontally (column-wise)
`>>> np.vstack((a,b))`	Stack arrays vertically (row-wise)
`>>> np.hsplit(c,2)`	Split the array horizontally at the 2nd index
`>>> np.vpslit(d,2)`	Split the array vertically at the 2nd index

Polynomials

`>>> from numpy import poly1d`	
`>>> p = poly1d([3,4,5])`	Create a polynomial object

Vectorizing Functions

```
>>> def myfunc(a):
        if a < 0:
          return a*2
        else:
          return a/2
>>> np.vectorize(myfunc)
```

Vectorize functions

Type Handling

`>>> np.real(b)`	Return the real part of the array elements
`>>> np.imag(b)`	Return the imaginary part of the array elements
`>>> np.real_if_close(c,tol=1000)`	Return a real array if complex parts close to 0
`>>> np.cast['f'](np.pi)`	Cast object to a data type

Other Useful Functions

`>>> np.angle(b,deg=True)`	Return the angle of the complex argument
`>>> g = np.linspace(0,np.pi,num=5)`	Create an array of evenly spaced values (number of samples)
`>>> g [3:] += np.pi`	
`>>> np.unwrap(g)`	Unwrap
`>>> np.logspace(0,10,3)`	Create an array of evenly spaced values (log scale)
`>>> np.select([c<4],[c*2])`	Return values from a list of arrays depending on conditions
`>>> misc.factorial(a)`	Factorial
`>>> misc.comb(10,3,exact=True)`	Combine N things taken at k time
`>>> misc.central_diff_weights(3)`	Weights for Np-point central derivative
`>>> misc.derivative(myfunc,1.0)`	Find the n-th derivative of a function at a point

Linear Algebra

Also see NumPy

You'll use the `linalg` and `sparse` modules. Note that `scipy.linalg` contains and expands on `numpy.linalg`.

```
>>> from scipy import linalg, sparse
```

Creating Matrices

```
>>> A = np.matrix(np.random.random((2,2)))
>>> B = np.asmatrix(b)
>>> C = np.mat(np.random.random((10,5)))
>>> D = np.mat([[3,4], [5,6]])
```

Basic Matrix Routines

Inverse	
`>>> A.I`	Inverse
`>>> linalg.inv(A)`	Inverse
Transposition	
`>>> A.T`	Tranpose matrix
`>>> A.H`	Conjugate transposition
Trace	
`>>> np.trace(A)`	Trace
Norm	
`>>> linalg.norm(A)`	Frobenius norm
`>>> linalg.norm(A,1)`	L1 norm (max column sum)
`>>> linalg.norm(A,np.inf)`	L inf norm (max row sum)
Rank	
`>>> np.linalg.matrix_rank(C)`	Matrix rank
Determinant	
`>>> linalg.det(A)`	Determinant
Solving linear problems	
`>>> linalg.solve(A,b)`	Solver for dense matrices
`>>> E = np.mat(a).T`	Solver for dense matrices
`>>> linalg.lstsq(F,E)`	Least-squares solution to linear matrix equation
Generalized inverse	
`>>> linalg.pinv(C)`	Compute the pseudo-inverse of a matrix (least-squares solver)
`>>> linalg.pinv2(C)`	Compute the pseudo-inverse of a matrix (SVD)

Creating Sparse Matrices

`>>> F = np.eye(3, k=1)`	Create a 2X2 identity matrix
`>>> G = np.mat(np.identity(2))`	Create a 2x2 identity matrix
`>>> C[C > 0.5] = 0`	
`>>> H = sparse.csr_matrix(C)`	Compressed Sparse Row matrix
`>>> I = sparse.csc_matrix(D)`	Compressed Sparse Column matrix
`>>> J = sparse.dok_matrix(A)`	Dictionary Of Keys matrix
`>>> E.todense()`	Sparse matrix to full matrix
`>>> sparse.isspmatrix_csc(A)`	Identify sparse matrix

Sparse Matrix Routines

Inverse	
`>>> sparse.linalg.inv(I)`	Inverse
Norm	
`>>> sparse.linalg.norm(I)`	Norm
Solving linear problems	
`>>> sparse.linalg.spsolve(H,I)`	Solver for sparse matrices

Sparse Matrix Functions

`>>> sparse.linalg.expm(I)`	Sparse matrix exponential

Asking For Help

```
>>> help(scipy.linalg.diagsvd)
>>> np.info(np.matrix)
```

Matrix Functions

Addition	
`>>> np.add(A,D)`	Addition
Subtraction	
`>>> np.subtract(A,D)`	Subtraction
Division	
`>>> np.divide(A,D)`	Division
Multiplication	
`>>> A @ D`	Multiplication operator (Python 3)
`>>> np.multiply(D,A)`	Multiplication
`>>> np.dot(A,D)`	Dot product
`>>> np.vdot(A,D)`	Vector dot product
`>>> np.inner(A,D)`	Inner product
`>>> np.outer(A,D)`	Outer product
`>>> np.tensordot(A,D)`	Tensor dot product
`>>> np.kron(A,D)`	Kronecker product
Exponential Functions	
`>>> linalg.expm(A)`	Matrix exponential
`>>> linalg.expm2(A)`	Matrix exponential (Taylor Series)
`>>> linalg.expm3(D)`	Matrix exponential (eigenvalue decomposition)
Logarithm Function	
`>>> linalg.logm(A)`	Matrix logarithm
Trigonometric Functions	
`>>> linalg.sinm(D)`	Matrix sine
`>>> linalg.cosm(D)`	Matrix cosine
`>>> linalg.tanm(A)`	Matrix tangent
Hyperbolic Trigonometric Functions	
`>>> linalg.sinhm(D)`	Hypberbolic matrix sine
`>>> linalg.coshm(D)`	Hyperbolic matrix cosine
`>>> linalg.tanhm(A)`	Hyperbolic matrix tangent
Matrix Sign Function	
`>>> np.signm(A)`	Matrix sign function
Matrix Square Root	
`>>> linalg.sqrtm(A)`	Matrix square root
Arbitrary Functions	
`>>> linalg.funm(A, lambda x: x*x)`	Evaluate matrix function

Decompositions

Eigenvalues and Eigenvectors	
`>>> la, v = linalg.eig(A)`	Solve ordinary or generalized eigenvalue problem for square matrix
`>>> l1, l2 = la`	Unpack eigenvalues
`>>> v[:,0]`	First eigenvector
`>>> v[:,1]`	Second eigenvector
`>>> linalg.eigvals(A)`	Unpack eigenvalues
Singular Value Decomposition	
`>>> U,s,Vh = linalg.svd(B)`	Singular Value Decomposition (SVD)
`>>> M,N = B.shape`	
`>>> Sig = linalg.diagsvd(s,M,N)`	Construct sigma matrix in SVD
LU Decomposition	
`>>> P,L,U = linalg.lu(C)`	LU Decomposition

Sparse Matrix Decompositions

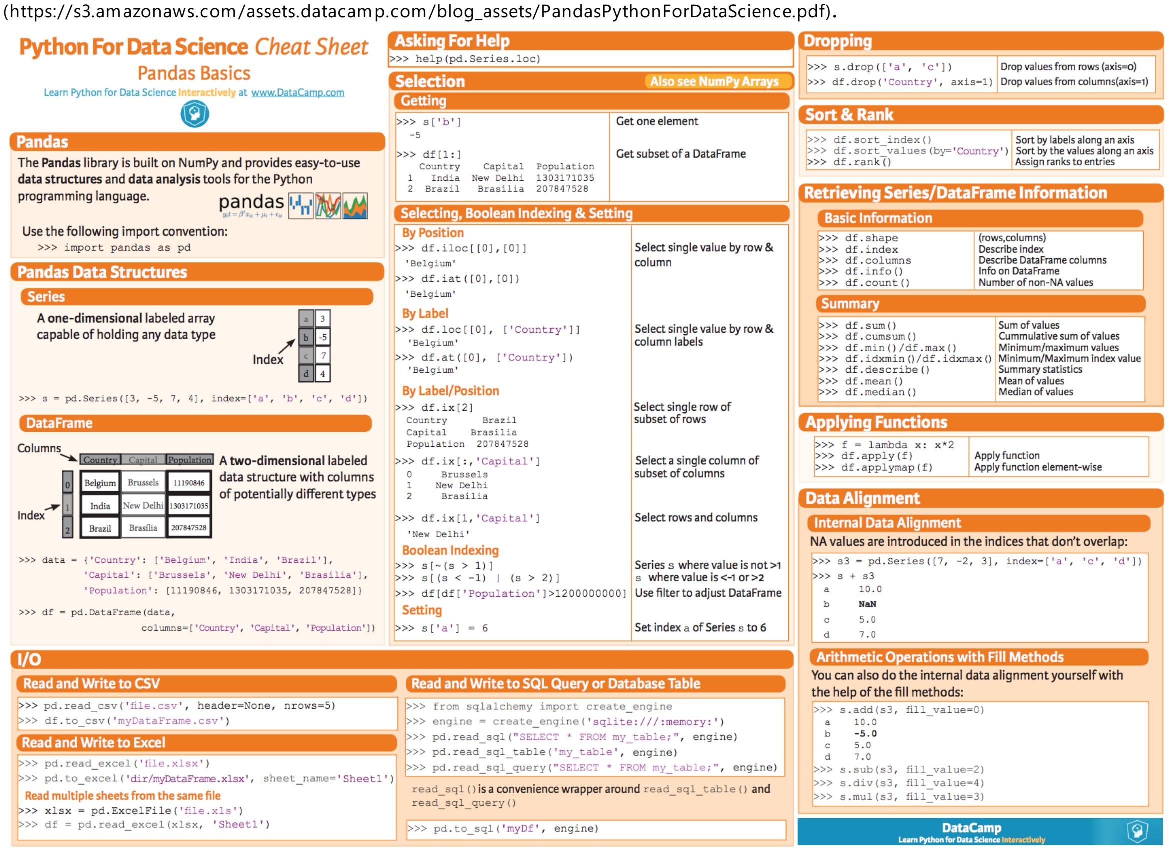
`>>> la, v = sparse.linalg.eigs(F,1)`	Eigenvalues and eigenvectors
`>>> sparse.linalg.svds(H, 2)`	SVD

DataCamp

Learn Python for Data Science Interactively

The *pandas* package

To get familiar with the *pandas* package have a look at the following summary poster (https://s3.amazonaws.com/assets.datacamp.com/blog_assets/PandasPythonForDataScience.pdf).



Python For Data Science Cheat Sheet

Pandas Basics

Learn Python for Data Science Interactively at www.DataCamp.com

Pandas

The **Pandas** library is built on NumPy and provides easy-to-use **data structures** and **data analysis** tools for the Python programming language.

Use the following import convention:

```
>>> import pandas as pd
```

Pandas Data Structures

Series

A one-dimensional labeled array capable of holding any data type

Index

a	3
b	-5
c	7
d	4

```
>>> s = pd.Series([3, -5, 7, 4], index=['a', 'b', 'c', 'd'])
```

DataFrame

Columns

A two-dimensional labeled data structure with columns of potentially different types

Index

	Country	Capital	Population
0	Belgium	Brussels	11190846
1	India	New Delhi	1303171035
2	Brazil	Brasília	207847528

```
>>> data = {'Country': ['Belgium', 'India', 'Brazil'],
            'Capital': ['Brussels', 'New Delhi', 'Brasília'],
            'Population': [11190846, 1303171035, 207847528]}

>>> df = pd.DataFrame(data,
                      columns=['Country', 'Capital', 'Population'])
```

Asking For Help

```
>>> help(pd.Series.loc)
```

Selection

Also see NumPy Arrays

Getting

`>>> s['b']` `-5`	Get one element

`>>> df[1:]`
 `Country Capital Population`
 `1 India New Delhi 1303171035`
 `2 Brazil Brasília 207847528`
	Get subset of a DataFrame

Selecting, Boolean Indexing & Setting

By Position	
`>>> df.iloc[[0],[0]]` `'Belgium'`	Select single value by row & column
`>>> df.iat([0],[0])` `'Belgium'`	
By Label	
`>>> df.loc[[0], ['Country']]` `'Belgium'`	Select single value by row & column labels
`>>> df.at([0], ['Country'])` `'Belgium'`	
By Label/Position	

`>>> df.ix[2]`
 `Country Brazil`
 `Capital Brasília`
 `Population 207847528`
	Select single row of subset of rows

`>>> df.ix[:,'Capital']`
 `0 Brussels`
 `1 New Delhi`
 `2 Brasília`
	Select a single column of subset of columns

`>>> df.ix[1,'Capital']` `'New Delhi'`	Select rows and columns
Boolean Indexing	
`>>> s[~(s > 1)]`	Series s where value is not >1
`>>> s[(s < -1) | (s > 2)]`	s where value is <-1 or >2
`>>> df[df['Population']>1200000000]`	Use filter to adjust DataFrame
Setting	
`>>> s['a'] = 6`	Set index a of Series s to 6

Dropping

`>>> s.drop(['a', 'c'])`	Drop values from rows (axis=0)
`>>> df.drop('Country', axis=1)`	Drop values from columns(axis=1)

Sort & Rank

`>>> df.sort_index()`	Sort by labels along an axis
`>>> df.sort_values(by='Country')`	Sort by the values along an axis
`>>> df.rank()`	Assign ranks to entries

Retrieving Series/DataFrame Information

Basic Information

`>>> df.shape`	(rows,columns)
`>>> df.index`	Describe index
`>>> df.columns`	Describe DataFrame columns
`>>> df.info()`	Info on DataFrame
`>>> df.count()`	Number of non-NA values

Summary

`>>> df.sum()`	Sum of values
`>>> df.cumsum()`	Cummulative sum of values
`>>> df.min()/df.max()`	Minimum/maximum values
`>>> df.idxmin()/df.idxmax()`	Minimum/Maximum index value
`>>> df.describe()`	Summary statistics
`>>> df.mean()`	Mean of values
`>>> df.median()`	Median of values

Applying Functions

`>>> f = lambda x: x*2`	
`>>> df.apply(f)`	Apply function
`>>> df.applymap(f)`	Apply function element-wise

Data Alignment

Internal Data Alignment

NA values are introduced in the indices that don't overlap:

```
>>> s3 = pd.Series([7, -2, 3], index=['a', 'c', 'd'])
>>> s + s3
  a    10.0
  b     NaN
  c     5.0
  d     7.0
```

Arithmetic Operations with Fill Methods

You can also do the internal data alignment yourself with the help of the fill methods:

```
>>> s.add(s3, fill_value=0)
  a    10.0
  b    -5.0
  c     5.0
  d     7.0
>>> s.sub(s3, fill_value=2)
>>> s.div(s3, fill_value=4)
>>> s.mul(s3, fill_value=3)
```

I/O

Read and Write to CSV

```
>>> pd.read_csv('file.csv', header=None, nrows=5)
>>> df.to_csv('myDataFrame.csv')
```

Read and Write to Excel

```
>>> pd.read_excel('file.xlsx')
>>> pd.to_excel('dir/myDataFrame.xlsx', sheet_name='Sheet1')
```

Read multiple sheets from the same file

```
>>> xlsx = pd.ExcelFile('file.xls')
>>> df = pd.read_excel(xlsx, 'Sheet1')
```

Read and Write to SQL Query or Database Table

```
>>> from sqlalchemy import create_engine
>>> engine = create_engine('sqlite:///:memory:')
>>> pd.read_sql("SELECT * FROM my_table;", engine)
>>> pd.read_sql_table('my_table', engine)
>>> pd.read_sql_query("SELECT * FROM my_table;", engine)
```

`read_sql()` is a convenience wrapper around `read_sql_table()` and `read_sql_query()`

```
>>> pd.to_sql('myDf', engine)
```

DataCamp

Learn Python for Data Science Interactively

JupyterLab

JupyterLab is a user-friendly environment to work with Python.

You can find an overview on JupyterLab here (https://jupyterlab.readthedocs.io/en/stable/).

In the following section we will explain how to install a Python on your laptop: we propose three different approaches for OSX, Windows and UNIX systems, respectively.

OSX: install and run a Docker image

In order to ease the installation procedure, we prepared a virtual environment that launch a Python3 Jupyter server (the installation on *cpymad* on OSX can be tricky, so we suggest to use the Docker image).

STEP 1: install the Docker Desktop

Please install the Docker Desktop (https://www.docker.com/products/docker-desktop).

> This is available for MAC and Windows 10 Enterprise and Professional (but not the 'Home Edition'). If you have Windows Home edition please refer to the *Windows: Docker Toolbox* section.

STEP 2: run the Docker image

Once the Docker Desktop is installed and running, open a terminal, move to a folder you want to use for CAS exercises and run the instruction

```
>> docker run -p 8888:8888 -v "$PWD":/cas sterbini/cas_aap_2019
```

This will download the image (~5GB): an internet connection is needed **only for the first time**, afterwords you can work offline.

> It is very important to have an offline working solution. Our experience with the previous schools showed that the standard WiFi infrastructure does not always meet the needed bandwidth performance. So, even if you have a working online Python environment (e.g. https://swan.web.cern.ch/ (https://swan.web.cern.ch/)) we strongly encourage to use an offline Python distribution.

You should get something as

```
MACBE16107:Tutorials sterbini$ docker run -p 8888:8888 -v "$PWD":/cas sterbini/cas_aap_20
[I 08:37:31.108 LabApp] Writing notebook server cookie secret to /cas/.local/share/jupyte
[I 08:37:31.341 LabApp] JupyterLab extension loaded from /opt/conda/lib/python3.6/site-pa
[I 08:37:31.341 LabApp] JupyterLab application directory is /opt/conda/share/jupyter/lab
[W 08:37:31.343 LabApp] JupyterLab server extension not enabled, manually loading...
[I 08:37:31.353 LabApp] JupyterLab extension loaded from /opt/conda/lib/python3.6/site-pa
[I 08:37:31.353 LabApp] JupyterLab application directory is /opt/conda/share/jupyter/lab
[I 08:37:31.354 LabApp] Serving notebooks from local directory: /cas
[I 08:37:31.354 LabApp] The Jupyter Notebook is running at:
[I 08:37:31.354 LabApp] http://(4bd247dca0ba or 127.0.0.1):8888/?token=ea65f062bfce037fd7
[I 08:37:31.355 LabApp] Use Control-C to stop this server and shut down all kernels (twic
[C 08:37:31.364 LabApp]

    Copy/paste this URL into your browser when you connect for the first time,
    to login with a token:
        http://(4bd247dca0ba or 127.0.0.1):8888/?token=ea65f062bfce037fd7a3b47926393a0d5d
```

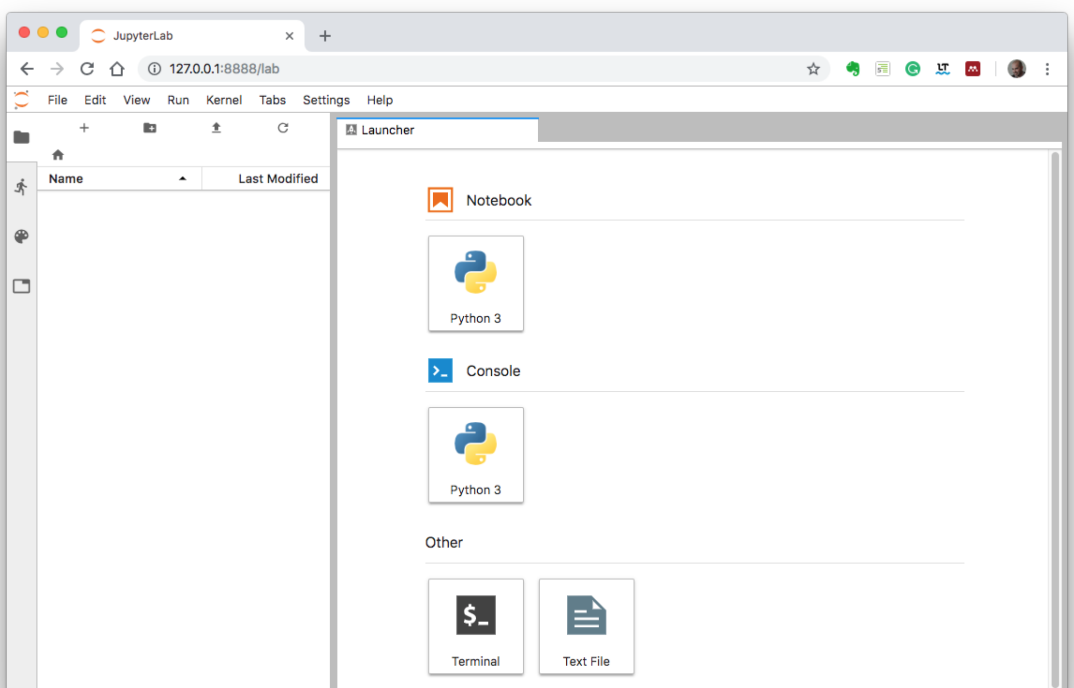
The last line is the most important one.

STEP 3: open JupyterLab from a browser

Open a web browser and connect to the python server at (**in this case**, check the last line) http://127.0.0.1:8888/?token=ea65f062bfce037fd7a3b47926393a0d5ded381785b0136b (http://127.0.0.1:8888/?token=ea65f062bfce037fd7a3b47926393a0d5ded381785b0136b)

You have to copy, paste and **edit** the last line on the address field of your browser.

You should see something as



This JupyterLab environment is setup with the software needed for the Optics course.

The *Laucher* tab allows you to open a notebook or console or some basic terminal/editing environment.

You can clic on the *Python 3 Notebook* icon in the *Launcher* tab and test the code example at the end of this document to verify that everything is working as expected.

UNIX: Anaconda distribution

For UNIX system the simplest way to install Python environment on your laptop is to setup manually your environment by installing *anaconda* (http://docs.continuum.io/_downloads/9ee215ff15fde24bf01791d719084950/Anaconda-Starter-Guide.pdf) from http://docs.continuum.io/anaconda/ (http://docs.continuum.io/anaconda/)

STEP 1: Anaconda installation

We suggest to install the Python 3.7 version (2019.03)
https://www.anaconda.com/distribution/ (https://www.anaconda.com/distribution/)
List the packages that are installed with

```
>> conda list
```

and verify that you have *matplotlib*, *numpy*, *scipy*, *pandas*, *sympy*.
If some of them are missing, please install them with, e.g.,

```
>> conda install -c conda-forge matplotlib
```

STEP 2: *PyNAFF* installation

You can find information on https://pypi.org/project/PyNAFF/ (https://pypi.org/project/PyNAFF/).
In most of the case it is enough to open a terminal and do

```
>> pip install PyNAFF
```

STEP 3: *cpymad* installation

The standard *anaconda* distribution comes with most of the needed packages but *cpymad*. You can install it following the instructions from https://github.com/hibtc/cpymad (https://github.com/hibtc/cpymad). In most of the case it is enough to open a terminal and do

```
>> pip install cpymad
```

STEP 4: JupyterLab

Now you can launch from a terminal JupyterLab

```
>> jupyter-lab
```

If JupyterLab is not installed in your system you can install it with

```
>> conda install -c conda-forge jupyterlab
```

The *jupyter-lab* command will open a browser.

The Laucher tab allows you to open a notebook or console or some basic terminal/editing environment.

You can clic on the Python 3 Notebook icon in the Launcher tab and test the code example at the end of this document to verify that everything is working as expected.

Windows: Docker Toolbox

If you have Windows 10 Professional or Enterprise you can follow the instructions given for OSX. The other Windows versions are not compatible with **Docker Desktop**.
In alternative to **Docker Desktop**, there is a legacy software caller **Docker Toolbox** (we tested it on Windons 10 Home). This comes with an Oracle Virtual Box where you will run the Docker Image.

STEP 1: Docker Toolbox installation

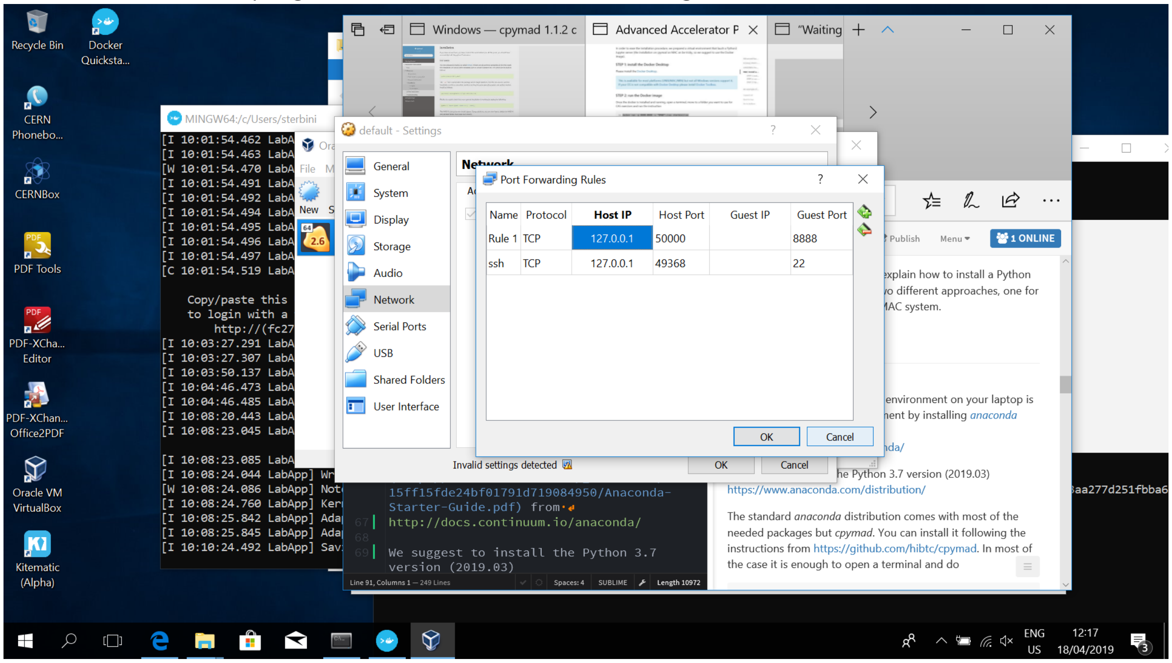
Download **Docker Toolbox** from
https://github.com/docker/toolbox/releases/download/v18.09.3/DockerToolbox-18.09.3.exe
(https://github.com/docker/toolbox/releases/download/v18.09.3/DockerToolbox-18.09.3.exe)

Install it (using custom configuration).

STEP 2: redirect the port 8888

You have to redirect the port 8888 of the virtual machine to the localhost port 50000 (you can use another free port if you like or if port the 50000 is already in use). This can be done from the Oracle VM VirtualBox (icon on your desktop) accessing the menu of the default Virtual machine:

‘Settings’->-‘Network’->-‘Advanced’->-‘Port Forwarding’.
Add (‘+’ icon on the top/right) a rule as shown in the following screenshot



This means that you can access the port 8888 of the virtual machine from the port 50000 of the localhost (127.0.0.1).

STEP 3: install and launch the docker image

Now open the Docker Quickstart (icon on the Desktop). It will take some time to start the virtual machine. Then move to your home folder typing

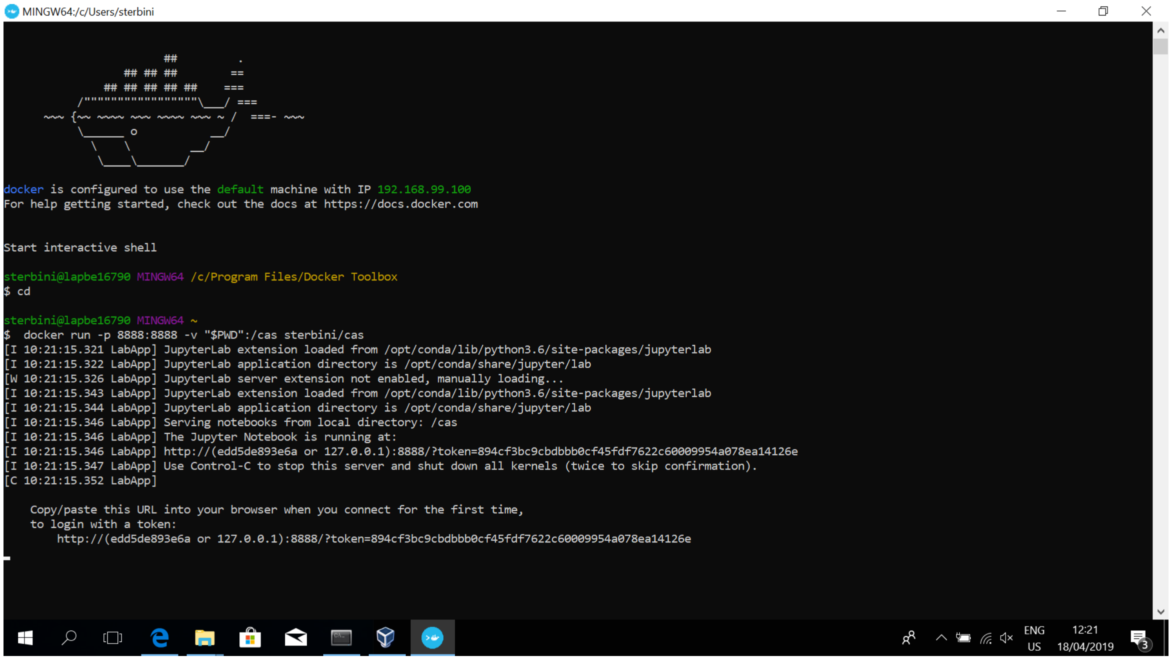
```
>> cd
```

Then type

```
>> docker run -p 8888:8888 -v "$PWD":/cas sterbini/cas_aap_2019
```

This download the docker image (only for the first time, ~5 GB) and run it.

You will get something like



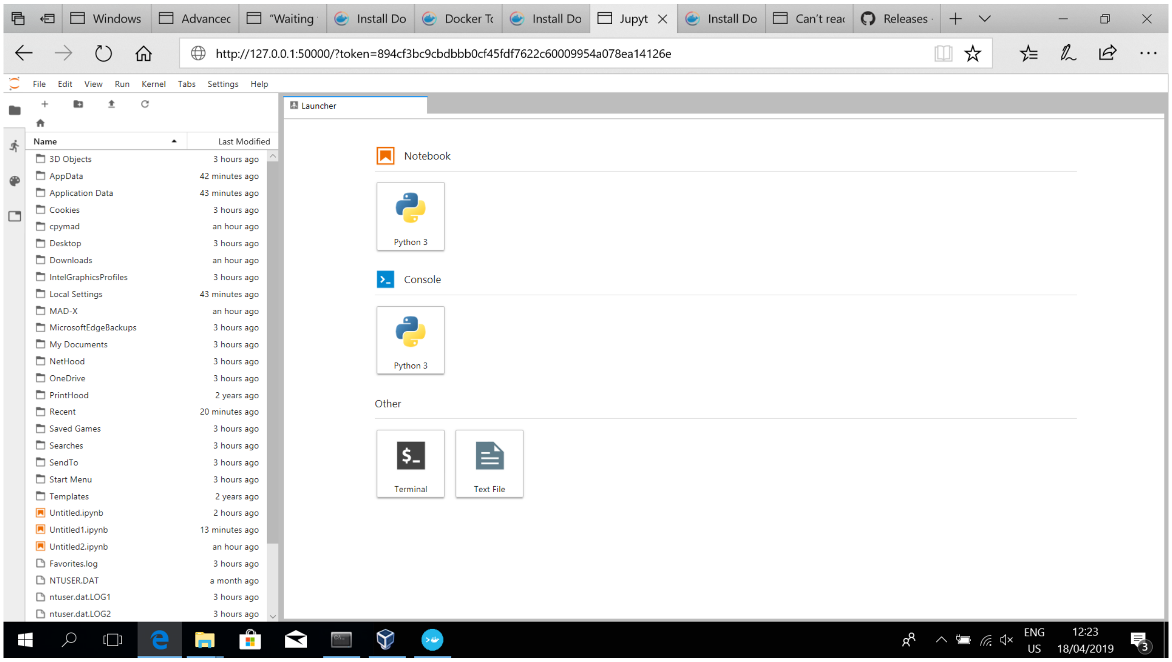
The last line is the most important one.

STEP 4: open JupyterLab from a browser

Open a web browser and connect to the python server at (**in this case**, see the last line of the previous screenshot)
http://127.0.0.1:50000/?token=HereGoesYourAlphanumericToken (http://127.0.0.1:50000/?token=HereGoesYourAlphanumericToken)

You have to copy, paste and **edit** the last line on the address field of your browser. **Please remember to replace the port 8888 with the port 50000.**

It will open a browser and you should get something as



The Laucher tab allows you to open a notebook or console or some basic terminal/editing environment.

You can clic on the Python 3 Notebook icon in the Launcher tab and test the code example at the end of this document to verify that everything is working as expected.

An example of Python3 notebook in Jupyter Lab.

From the *Launcher* of JupyterLab select the Python 3 Notebook.
You can import the python library of MAD-X (*cpymad*) with the following command.

```
from cpymad.madx import Madx
from matplotlib import pyplot as plt
myMad = Madx()
```

and now you can twiss a simple FODO cell (more details during the course) with the following code

```
myString='''
! *********************************************************************
! Second part
! *********************************************************************

! *********************************************************************
! Definition of parameters
! *********************************************************************

l_cell=100;
quadrupoleLenght=5;
f=200;
myK:=1/f/quadrupoleLenght;// m^-2

! *********************************************************************
! Definition of magnet
! *********************************************************************
QF: quadrupole, L=quadrupoleLenght, K1:=myK;
QD: quadrupole, L=quadrupoleLenght, K1:=-myK;


! *********************************************************************
! Definition of sequence
! *********************************************************************
myCell:sequence, refer=entry, L=L_CELL;
quadrupole1: QF, at=0;
marker1: marker, at=25;
quadrupole2: QD, at=50;
endsequence;

! *********************************************************************
! Definition of beam
! *********************************************************************
beam, particle=proton, energy=2;

! *********************************************************************
! Use of the sequence
! *********************************************************************
use, sequence=myCell;

! *********************************************************************
! TWISS
! *********************************************************************
title, 'My first twiss';
twiss, file=myFirstTwiss.twiss;
'''
myMad.input(myString);
```

You can see the *Q1* and *betymax* parameters by executing this cell

```
myString='''
value, table(SUMM,Q1);
value, table(SUMM,betymax);
'''
myMad.input(myString);
```

and you can get a pandas dataframe with the information of the lattice by

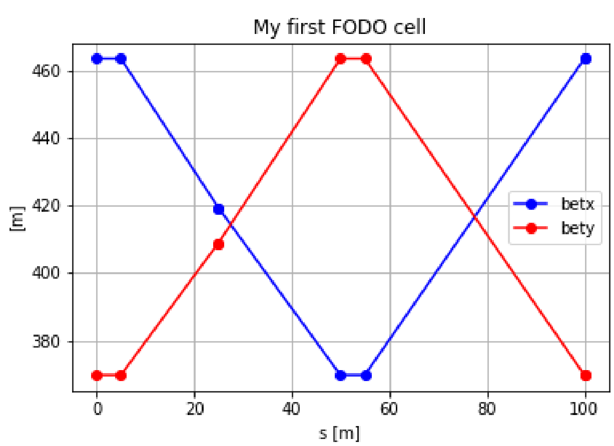
```
myDF=myMad.table.twiss.dframe()
myDF[['name','s','betx','bety']].head()
```

You should get something like

	name	s	betx	bety
#s	mycell$start:1	0.0	463.623288	369.779162
quadrupole1	quadrupole1:1	5.0	463.623288	369.779162
drift_0[0]	drift_0:0	25.0	419.394867	408.967742
marker1	marker1:1	25.0	419.394867	408.967742
drift_1[0]	drift_1:0	50.0	369.779162	463.623288

To plot some data of the *twiss* table you can execute

```
plt.plot(myDF['s'],myDF['betx'],'ob-')
plt.plot(myDF['s'],myDF['bety'],'or-')
plt.legend()
plt.grid()
plt.xlabel('s [m]')
plt.ylabel('[m]')
plt.title('My first FODO cell')
```



You can do also a bit of symbolic computation with

```
import sympy as sy
import numpy as np
from sympy import init_session
init_session()
la=np.linalg
L_cell=sy.Symbol('L_cell', positive=True);
f_1=sy.Symbol('f_1', positive=True);
f_2=sy.Symbol('f_2', positive=True);
f=sy.Symbol('f', positive=True);


QF=sy.Matrix([[1,0], [-1/f,1]])
DRIFT=sy.Matrix([[1,L_cell/2], [0,1]])
QD=sy.Matrix([[1,0], [1/f,1]])
# This is the OTM
M=DRIFT@QD@DRIFT@QF
M=sy.simplify(M)
M
```

or FTT analysis using PyNAFF, e.g.,

```
import PyNAFF as pnf
import numpy as np

t = np.linspace(1, 3000, num=3000, endpoint=True)
Q = 0.12345
signal = np.sin(2.0*np.pi*Q*t)
pnf.naff(signal, 500, 1, 0 , False, window=1)
# outputs an array of arrays for each frequency. Each sub-array includes:
# [order of harmonic, frequency, Amplitude, Re{Amplitude}, Im{Amplitude]

# My frequency is simply
pnf.naff(signal, 500, 1, 0 , False)[0][1]
```

An example of test ipython notebook is shown in

https://cernbox.cern.ch/index.php/s/JJCu7KRPAjuitVF (https://cernbox.cern.ch/index.php/s/JJCu7KRPAjuitVF)