“New” tools in HEP: my personal experience

Romain Madar

September 2018

Contents

1 Why did I look to other tools?

1.1 Fourier analysis

1.2 Principal Analysis Component

1.3 Change of air

2 The tools: python and jupyter notebooks

2.1 Quick tour of python tools

Python offers so (too?) much tools for data analysis (*non exhaustive* list)

It's true that python is a very dynamic language and offers many tools, in data analysis but not only. In the following part, I'll focus on packages which are relevant for HEP but there are much more to deal with in term of website creation, API for google maps or geographical data.

- data vizualization (interactive) matplolib, plotpy, seaborn, bokeh, . . .
- scientific, numeric and symbolic calculation scipy, numpy, simpy
- machine learning scikitlearn, kerras, tensorflow, pytorch, etc . . .
- data manipulation pandas
- **pure HEP**:
  - interfaced with ROOT in several ways pyROOT, rootpy, root_numpy, uproot, root_pandas
  - and few more hep-oriented libraries in scikit-hep (starting effort)

2.2 NumFocus

According the NumFOCUS website:

> The mission of NumFOCUS is to promote sustainable high-level programming languages, open code development, and reproducible scientific research. We accomplish this mission through our educational programs and events as well as through fiscal sponsorship of open source scientific computing projects. We aim to increase collaboration and communication within the data science and scientific computing community.

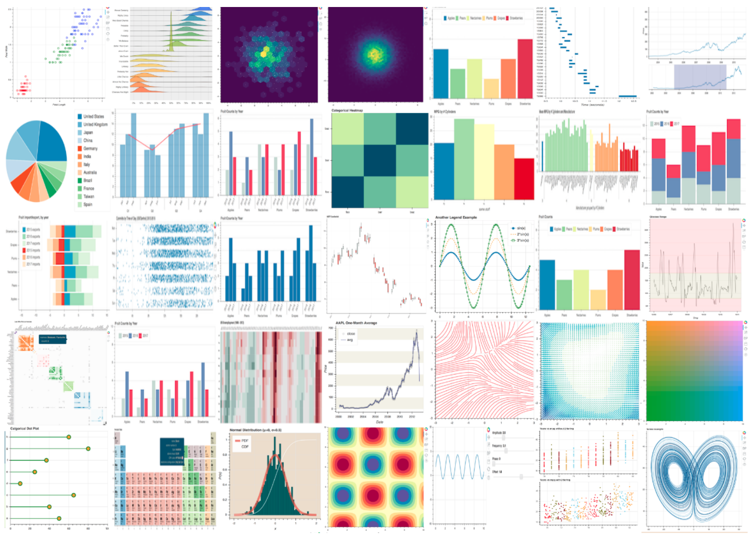
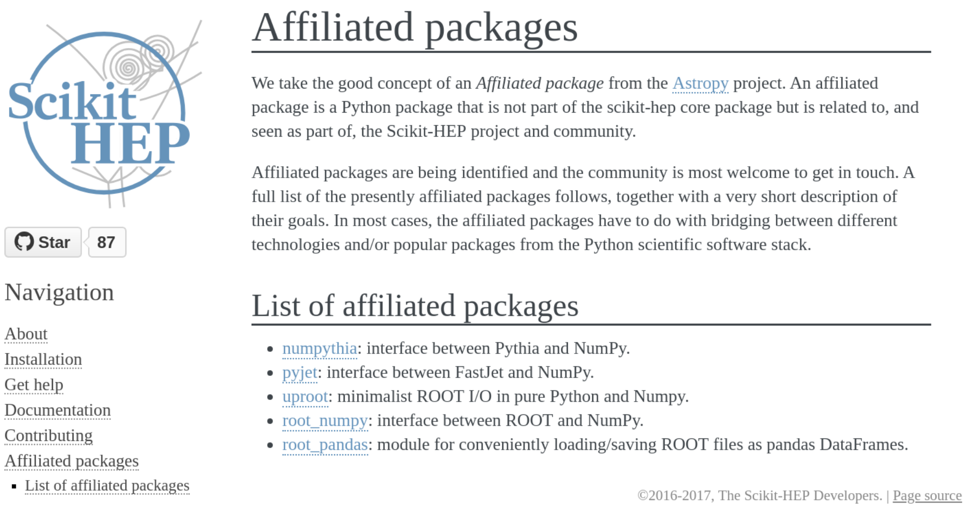


Figure 1: Gallery of bokeh with interactive plots, as in this example or this one



Affiliated packages

We take the good concept of an *Affiliated package* from the Astropy project. An affiliated package is a Python package that is not part of the scikit-hep core package but is related to, and seen as part of, the Scikit-HEP project and community.

Affiliated packages are being identified and the community is most welcome to get in touch. A full list of the presently affiliated packages follows, together with a very short description of their goals. In most cases, the affiliated packages have to do with bridging between different technologies and/or popular packages from the Python scientific software stack.

List of affiliated packages

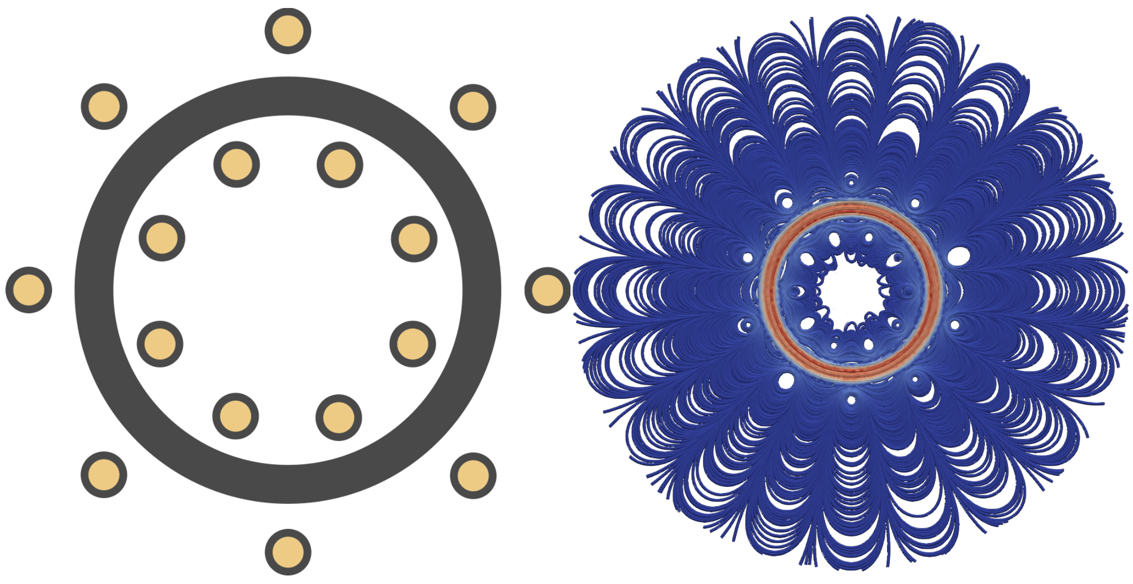
- numpythia: interface between Pythia and NumPy.
- pyjet: interface between FastJet and NumPy.
- uproot: minimalist ROOT I/O in pure Python and Numpy.
- root_numpy: interface between ROOT and NumPy.
- root_pandas: module for conveniently loading/saving ROOT files as pandas DataFrames.

Figure 2: Screenshot of the scikit-hep website showing the affiliated pacakges, on top of the actuall content of scikit-hep (pyjet, numpythia). Inspired by astropy

Figure 3: Supported projects

Projects cover data vizualization, astrophysics, thermodynamics, fluid mechanics, economy, data analysis, scientific computation, etc ... [1]

Example of electromagnetic problem solved with FEniCS (with a 92 lines code)



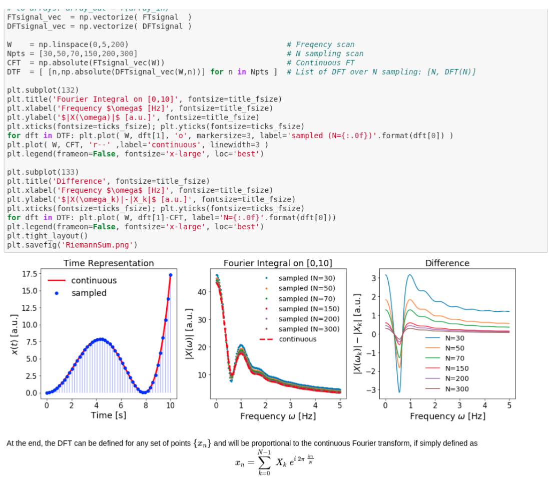
2.3 Quick tour of notebooks

Jupyter notebook environement allow to combine code, plots and notes in a friendly place.

Jupyter notebooks (or how to *try to* get back analysis repoductibility?)

- a single environment combining source code, plots and notes
- great for exploring data or learning new concepts and *document it*
- many nice features:
  - exportation (html, python, article, slides) – I'll come back on this
  - sharing with nbviewer & online execution with mybinder (beta)
  - SWAN online notebooks service at CERN (connected to CERNbox)
- jupyter project have **many** tutorials via nbviewer. E.g.:
  - signal processing tutorials: about 20 tutorials including filtering, markov chains, maximum likelihood approach, etc ...
  - probabilistic programing: 20 pages tutorial with code, plots and explanations.

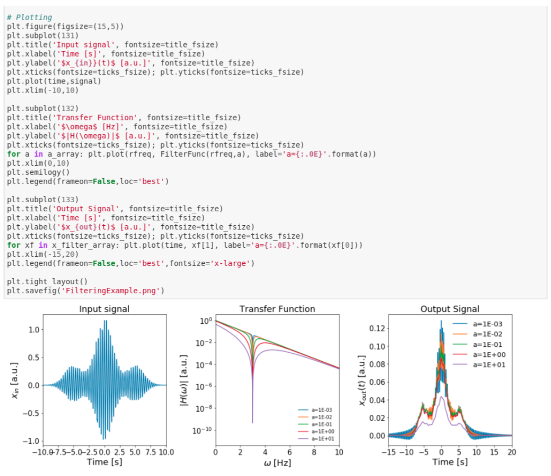
Example with Fourier analysis



```
FTsignal_vec  = np.vectorize( FTsignal  )
DFTsignal_vec = np.vectorize( DFTsignal )

W    = np.linspace(0,5,200)                                        # Freqency scan
Npts = [30,50,70,150,200,300]                                      # N sampling scan
CFT  = np.absolute(FTsignal_vec(W))                                # Continuous FT
DTF  = [ [n,np.absolute(DFTsignal_vec(W,n))] for n in Npts ]  # List of DFT over N sampling: [N, DFT(N)]

plt.subplot(132)
plt.title('Fourier Integral on [0,10]', fontsize=title_fsize)
plt.xlabel('Frequency $\omega$ [Hz]', fontsize=title_fsize)
plt.ylabel('$|X(\omega)|$ [a.u.]', fontsize=title_fsize)
plt.xticks(fontsize=ticks_fsize); plt.yticks(fontsize=ticks_fsize)
for dft in DTF: plt.plot( W, dft[1], 'o', markersize=3, label='sampled (N={:.0f})'.format(dft[0]) )
plt.plot( W, CFT, 'r--' ,label='continuous', linewidth=3 )
plt.legend(frameon=False, fontsize='x-large', loc='best')

plt.subplot(133)
plt.title('Difference', fontsize=title_fsize)
plt.xlabel('Frequency $\omega$ [Hz]', fontsize=title_fsize)
plt.ylabel('$|X(\omega_k)|-|X_k|$ [a.u.]', fontsize=title_fsize)
plt.xticks(fontsize=ticks_fsize); plt.yticks(fontsize=ticks_fsize)
for dft in DTF: plt.plot( W, dft[1]-CFT, label='N={:.0f}'.format(dft[0]))
plt.legend(frameon=False, fontsize='x-large', loc='best')
plt.tight_layout()
plt.savefig('RiemannSum.png')
```

At the end, the DFT can be defined for any set of points $\{x_n\}$ and will be proportional to the continuous Fourier transform, if simply defined as

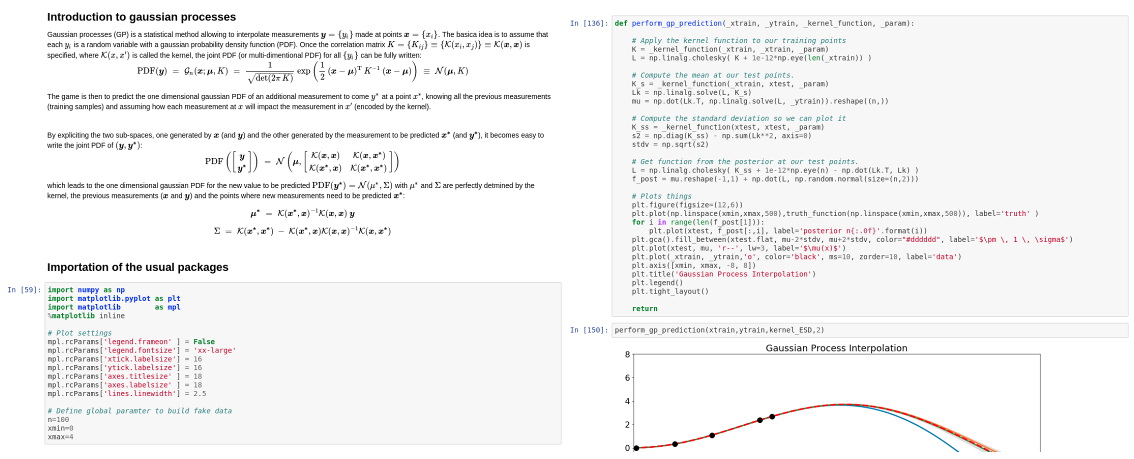
$$x_n = \sum_{k=0}^{N-1} X_k\, e^{i\,2\pi\,\frac{kn}{N}}$$



```
# Plotting
plt.figure(figsize=(15,5))
plt.subplot(131)
plt.title('Input signal', fontsize=title_fsize)
plt.xlabel('Time [s]', fontsize=title_fsize)
plt.ylabel('$x_{in}}(t)$ [a.u.]', fontsize=title_fsize)
plt.xticks(fontsize=ticks_fsize); plt.yticks(fontsize=ticks_fsize)
plt.plot(time,signal)
plt.xlim(-10,10)

plt.subplot(132)
plt.title('Transfer Function', fontsize=title_fsize)
plt.xlabel('$\omega$ [Hz]', fontsize=title_fsize)
plt.ylabel('$|H(\omega)|$ [a.u.]', fontsize=title_fsize)
plt.xticks(fontsize=ticks_fsize); plt.yticks(fontsize=ticks_fsize)
for a in a_array: plt.plot(rfreq, FilterFunc(rfreq,a), label='a={:.0E}'.format(a))
plt.xlim(0,10)
plt.semilogy()
plt.legend(frameon=False,loc='best')

plt.subplot(133)
plt.title('Output Signal', fontsize=title_fsize)
plt.xlabel('Time [s]', fontsize=title_fsize)
plt.ylabel('$x_{out}(t)$ [a.u.]', fontsize=title_fsize)
plt.xticks(fontsize=ticks_fsize); plt.yticks(fontsize=ticks_fsize)
for xf in x_filter_array: plt.plot(time, xf[1], label='a={:.0E}'.format(xf[0]))
plt.xlim(-15,20)
plt.legend(frameon=False,loc='best',fontsize='x-large')

plt.tight_layout()
plt.savefig('FilteringExample.png')
```

- view on nbviewer or execute on binder
- clone via github

Example with gaussian processes



Introduction to gaussian processes

Gaussian processes (GP) is a statistical method allowing to interpolate measurements $\boldsymbol{y} = \{y_i\}$ made at points $\boldsymbol{x} = \{x_i\}$. The basica idea is to assume that each y_i is a random variable with a gaussian probability density function (PDF). Once the correlation matrix $K = \{K_{ij}\} \equiv \{\mathcal{K}(x_i, x_j)\} \equiv \mathcal{K}(\boldsymbol{x}, \boldsymbol{x})$ is specified, where $\mathcal{K}(x, x')$ is called the kernel, the joint PDF (or multi-dimentional PDF) for all $\{y_i\}$ can be fully written:

$$\mathrm{PDF}(\boldsymbol{y}) \;=\; \mathcal{G}_n(\boldsymbol{x}; \boldsymbol{\mu}, K) \;=\; \frac{1}{\sqrt{\det(2\pi\,K)}} \exp\left(\frac{1}{2}\,(\boldsymbol{x}-\boldsymbol{\mu})^{\mathrm{T}}\,K^{-1}\,(\boldsymbol{x}-\boldsymbol{\mu})\right) \;\equiv\; \mathcal{N}(\boldsymbol{\mu}, K)$$

The game is then to predict the one dimensional gaussian PDF of an additional measurement to come y^* at a point x^*, knowing all the previous measurements (training samples) and assuming how each measurement at x will impact the measurement in x' (encoded by the kernel).

By expliciting the two sub-spaces, one generated by $\boldsymbol{x}$ (and $\boldsymbol{y}$) and the other generated by the measurement to be predicted $\boldsymbol{x}^*$ (and $\boldsymbol{y}^*$), it becomes easy to write the joint PDF of $(\boldsymbol{y}, \boldsymbol{y}^*)$:

$$\mathrm{PDF}\left(\begin{bmatrix} \boldsymbol{y} \\ \boldsymbol{y}^* \end{bmatrix}\right) \;=\; \mathcal{N}\left(\boldsymbol{\mu}, \begin{bmatrix} \mathcal{K}(\boldsymbol{x},\boldsymbol{x}) & \mathcal{K}(\boldsymbol{x},\boldsymbol{x}^*) \\ \mathcal{K}(\boldsymbol{x}^*,\boldsymbol{x}) & \mathcal{K}(\boldsymbol{x}^*,\boldsymbol{x}^*) \end{bmatrix}\right)$$

which leads to the one dimensional gaussian PDF for the new value to be predicted $\mathrm{PDF}(\boldsymbol{y}^*) = \mathcal{N}(\mu^*, \Sigma)$ with μ^* and Σ are perfectly detmined by the kernel, the previous measurements ($\boldsymbol{x}$ and $\boldsymbol{y}$) and the points where new measurements need to be predicted $\boldsymbol{x}^*$:

$$\boldsymbol{\mu}^* \;=\; \mathcal{K}(\boldsymbol{x}^*, \boldsymbol{x})^{-1}\mathcal{K}(\boldsymbol{x}, \boldsymbol{x})\,\boldsymbol{y}$$

$$\Sigma \;=\; \mathcal{K}(\boldsymbol{x}^*, \boldsymbol{x}^*) \;-\; \mathcal{K}(\boldsymbol{x}^*, \boldsymbol{x})\mathcal{K}(\boldsymbol{x}, \boldsymbol{x})^{-1}\mathcal{K}(\boldsymbol{x}, \boldsymbol{x}^*)$$

Importation of the usual packages

In [59]:
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib        as mpl
%matplotlib inline

# Plot settings
mpl.rcParams['legend.frameon' ] = False
mpl.rcParams['legend.fontsize'] = 'xx-large'
mpl.rcParams['xtick.labelsize'] = 16
mpl.rcParams['ytick.labelsize'] = 16
mpl.rcParams['axes.titlesize' ] = 18
mpl.rcParams['axes.labelsize' ] = 18
mpl.rcParams['lines.linewidth'] = 2.5

# Define global paramter to build fake data
n=100
xmin=0
xmax=4
```

In [136]:
```
def perform_gp_prediction(_xtrain, _ytrain, _kernel_function, _param):

    # Apply the kernel function to our training points
    K = _kernel_function(_xtrain, _xtrain, _param)
    L = np.linalg.cholesky( K + 1e-12*np.eye(len(_xtrain)) )

    # Compute the mean at our test points.
    K_s = _kernel_function(_xtrain, xtest, _param)
    Lk = np.linalg.solve(L, K_s)
    mu = np.dot(Lk.T, np.linalg.solve(L, _ytrain)).reshape((n,))

    # Compute the standard deviation so we can plot it
    K_ss = _kernel_function(xtest, xtest, _param)
    s2 = np.diag(K_ss) - np.sum(Lk**2, axis=0)
    stdv = np.sqrt(s2)

    # Get function from the posterior at our test points.
    L = np.linalg.cholesky( K_ss + 1e-12*np.eye(n) - np.dot(Lk.T, Lk) )
    f_post = mu.reshape(-1,1) + np.dot(L, np.random.normal(size=(n,2)))

    # Plots things
    plt.figure(figsize=(12,6))
    plt.plot(np.linspace(xmin,xmax,500),truth_function(np.linspace(xmin,xmax,500)), label='truth' )
    for i in range(len(f_post[1])):
        plt.plot(xtest, f_post[:,i], label='posterior n{:.0f}'.format(i))
    plt.gca().fill_between(xtest.flat, mu-2*stdv, mu+2*stdv, color="#dddddd", label='$\pm \, 1 \, \sigma$')
    plt.plot(xtest, mu, 'r--', lw=3, label='$\mu(x)$')
    plt.plot(_xtrain, _ytrain,'o', color='black', ms=10, zorder=10, label='data')
    plt.axis([xmin, xmax, -8, 8])
    plt.title('Gaussian Process Interpolation')
    plt.legend()
    plt.tight_layout()

    return
```

In [150]:
```
perform_gp_prediction(xtrain,ytrain,kernel_ESD,2)
```

- view on nbviewer or execute on binder
- clone via github

3 In practice: what is great and less great?

3.1 What's great about python

Python is nice because it's very fast to code!

Example 1: get all possible pairs

```
import itertools
mu_pt,el_pt = [23,42,55,137],[24,32,61,172]

# Get all pairs
all_pairs = list(itertools.product(mu_pt, el_pt))

# Print all pairs
print('all pairs: {}'.format(str(all_pairs)))

# Print every second pair
print('Every second pair: {}'.format(all_pairs[::2]))
```

```
all pairs: [(23, 24), (23, 32), (23, 61), (23, 172), (42, 24),
    (42, 32), (42, 61), (42, 172), (55, 24), (55, 32), (55, 61)
    , (55, 172), (137, 24), (137, 32), (137, 61), (137, 172)]
Every second pair: [(23, 24), (23, 61), (42, 24), (42, 61),
    (55, 24), (55, 61), (137, 24), (137, 61)]
```

Example 2: generate random binnings

```
def generate_bins(n,xmin,xmax,step=1.):
    import numpy as np
    xmin,xmax=xmin/step,xmax/step
    r = np.sort(np.random.random_integers(xmin,xmax,n))*step
    r = np.insert(r,0,xmin*step)
    r = np.insert(r,len(r),xmax*step)
    return r

bins = [generate_bins(10,0,500,5) for i in range(0,5)]
for b in bins: print(b)
```

```
[  0  20  70 105 170 215 310 335 385 440 480 500]
[  0  50  55 105 260 310 335 385 400 460 480 500]
[  0  25 185 205 230 245 290 315 355 410 450 500]
[  0  60 110 225 240 260 310 435 460 475 500 500]
[  0  80 115 115 190 260 315 350 385 450 470 500]
```

```
print('[test](if this is done in markdown)')
```

```
[test](if this is done in markdown)
```

3.2 What's is not so great about python

3.3 What is great about notebooks

3.4 What is not so great about notebooks

4 Concrete examples in ATLAS analysis

5 Side discovery: pandocs

References

[1] D. Abercrombie *et al.*, *Dark Matter Benchmark Models for Early LHC Run-2 Searches: Report of the ATLAS/CMS Dark Matter Forum*, 2015, https://arxiv.org/abs/1507.00966.