Table of Contents

The Ostap tutorials build passing

Ostap is a set of extensions/decorators and utilities over the basic `PyROOT` functionality (python wrapper for `ROOT` framework). These utilities greatly simplify the interactive manipulations with `ROOT` classes through python. The main ingredients of `Ostap` are

- preconfigured ipython script `ostap`, that can be invoked from the command line.
- *decoration* of the basic `ROOT` objects, like histograms, graphs etc.
  - operations and operators
  - iteration, element access, etc
  - extended functionality
- *decoration* of many basic `ROOT.RooFit` objects
- set of new useful fit models, components and operations
- other useful analysis utilities

Getting started

The main ingredients of `Ostap` are

- preconfigured ipython script `ostap` , that can be invoked from the command line.

  ```
  ostap
  ```

  > ### Challenge
  >
  > Invoke the script with `-h` option to get the whole list of all command line options and keys

  Optionally one can specify the list of python files to be executed before appearance of the interactive command prompt:

  ```
  ostap a.py b.py c.py d.py
  ```

  The list of optional arguments can include also root-files, in this case the files will be opened and their handlers will be available via local list `root_files`

  ```
  ostap a.py b.py c.py d.py file1.root file2.root e.py file3.root
  ```

  Also `ROOT` macros can be specified on the command line

  ```
  ostap a.py b.py c.py d.py file1.root q1.C file2.root q2.C e.py file3.root q4.C
  ```

The script automatically opens `TCanvas` window (unless `--no-canvas` option is specified) with (a little bit modified) LHCb style. It also loads necessary decorators for `ROOT` classes. At last it executes the python scripts and opens root-files, specified as command line arguments.

Values with uncertanties: `ValueWithError`

One of the central object in `ostap` is C++ class `Gaudi::Math::ValutWithError`, accessible in python via shortcut `VE`. This class stands fot r a combination of the value with uncertainties:

```
from Ostap.Core import VE
a = VE( 10 , 10 ) ## the value & squared uncertainty - 'variance'
b = VE( 20 , 20 ) ## the value & squared uncertainty - 'variance'
print "a=%s" % a
print "b=%s" % b
print 'Value    of a is %s'  % a.value()
print 'Effor    of b is %s'  % b.error()
print 'Variance of b is %s'  % b.cov2 ()
```

A lot of math operations are predefined for `VE`-objects.

Challenge

Make a try with all binary operations (`+,-,*,/,**`) for the pair of `VE` objects and combinations of `VE`-objects with numbers, e.g.

```
a + b
a + 1
1 - b
2 ** a
a +=1
b += a
```

Compare the difference for following expresssions:

```
a/a      ## <--- HERE
a/VE(a)  ## <--- HERE
a-a      ## <--- HERE
a-VE(a)  ## <--- HERE
```

Note that for trivial cases the correlations are propertly taken into account

Additionally many math-functions are provided, carefully takes care on uncertainties

```
from LHCbMath.math_ve import *
sin(a)+cos(b)/tanh(b)
atan2(a,b)/log(a)
```

Simple operations with histograms

Historgam content

`Ostap.PyRoUts` module provides two ways to access the histogram content

- by bin index, using operator `[]` : for 1D historgam index is a simple integer number, for 2D and 3D-histograms the bin index is a 2 or 3-element tuple
- using *functional* interface with operator `()` .

  ```
  histo = ...
  print histo[2]    ## print the value/error associated with the 2nd bin
  print histo(2.21) ## print the value/error at x=2.21
  ```

  Note that the result in both cases is of type `VE` , *value+/-uncertainty*, and the interpolation is involved in the second case. The interpolation can be controlled using `interpolation` argument

  ```
  print histo ( 2.1 , interpolation = 0 ) ## no interpolation
  print histo ( 2.1 , interpolation = 1 ) ## linear interpolation
  print histo ( 2.1 , interpolation = 2 ) ## parabolic interpolation
  print histo ( 2.1 , interpolation = 3 ) ## cubic interpolation
  ```

  Similarly for 2D and 3D cases, `interpolation` parameter is 2 or 3-element tuple, e.g. `(2,2)` `(3,2,2)` , `(3,0,0)` , ...

Set bin content

```
histo[1] = VE(10,10)
histo[2] = VE(20,20)
```

Loops over the histogram content:

```
for i in histo :
    print 'Bin# %s, the content%s' % ( i, histo[i] )
for entry in histo.iteritems() :
    print 'item  ', entry
```

Histogram slicing

The slicing of 1D-historgam can be done easily using native `slice` in python

```
h1 = h[3:8]
```

For 2D and 3D-casss the slicing is less trivial, but still simple

```
histo2D = ...
h1 = histo2D.sliceX ( 1 )
h2 = histo2D.sliceY ( [1,3,5] )
h3 = histo2D.sliceY ( 3 )
h4 = histo2D.sliceY ( [3,4,5] )
```

Operators and operations

A lot of operators and operations are defined for histograms.

```
histo += 1
histo /= 10
histo  = 1     + histo      ## operations with constants
histo  = histo + math.cos   ## operations with functions
histo /= lambda x : 1 + x   ## lambdas are also functions
```

Also binary operations are defined

```
h1 = ...
h2 = ...
h3 = h1 + h2
h4 = h1 / h2
h5 = h1 * h2
h6 = h1 - h2
```

For the binary operations the action is defiened accordinh to the rule

- the type of the result is defined by the first operand (type, and binning)
- for each bin `i` the result is estimated as `a oper b`, where:
  - `oper` stands for corresponding operator (`+,-,*,/,**)`
  - `a = h1[i]` is a value of the first operand at bin `i`
  - `b = h2(x)`, where `x` is a bin-center of bin `i`

More operations

There are many other useful opetations:

- `abs` : apply `abs` function bin-by-bin
- `asym` : equivalent to `(h1-h2)/(h1+h2)` with correct treatment of correlated uncertainties
- `frac` : equivalent to `(h1)/(h1+h2)` with correct treatment of correlated uncertainties
- `average` : make an average of two historgam
- `chi2` : bin-by-bin chi2-tension between two historgams
- ... and many more

Transformations

```
h1 = histo.transform ( lambda x,y : y    ) ## identical transformation (copy)
h2 = histo.transform ( lambda x,y : y**3 ) ## get the third power of the histogram content
h3 = histo.transform ( lambda x,y : y/x  ) ## less trivial functional transformation
```

Efficiencies

There are several special cases to get the efficiency-historgams

```
accepted = ... ## historgam with accepted sample
rejected = ... ## historgam with rejected sample
total    = ... ## historgam with total sample

eff1 = accepted/total          ## value is correct, uncertainties are *NOT* correct
eff2 = 1/(1+rejected/accepted) ## everything is correct (binomial)
eff3 = accepted %  total       ## everything is correct (binomial)
eff4 = accepted // total       ## correct binomial, if both histograms are "natural"
```

Running sums and the efficiencies of cuts

```
h1 = histo.sumv ()        ## increasing order: sum(first,x)
h2 = histo.sumv ( False ) ## decreasing order: sum(x,last )
```

Such functionality immediately allows to calculate efficiency historgrams using `effic` method:

```
h1 = histo.effic ()        ## efficiency of var<x cut
h2 = histo.effic ( False ) ## efficiency of var>x cut
```

Math functions

The standard math-functions can be applied to the histoigram (bin-by-bin):

```
from LHCbMath.math_ve import *
h1 = sin ( histo )
h2 = exp ( histo )
h3 = exp ( abs ( histo ) )
...
```

Sampling

There is an easy way to sample the histograms according to their content, e.g. for toy-experiments:

```
h1 = histo.sample() ## make a random histogram with content sampled according to bin+-error in original histo
h2 = histo.sample( accept = lambda s : s > 0 ) ##sample but require that sampled values are positive

#### Smearing/convolution with gaussian
It is very easy to smear 1D histogram according to gaussian resolution
```python
h1 = histo.smear ( 0.015 ) ## apply "smearing" with sigma = 0.015
```

Rebin

```
original = ... ## the original historgam to be rebinned
template = ... ## historgams that deifned new binning  scheme
rebin1   = original.rebinNumbers  ( template ) ## compare it!
rebin2   = original.rebinFunction ( template ) ## compare it!
`
```
```

Contributing

ostap-tutorials is an open source project, and we welcome contributions of all kinds:

- New lessons;
- Fixes to existing material;
- Bug reports; and
- Reviews of proposed changes.

By contributing, you are agreeing that we may redistribute your work under these licenses. You also agree to abide by our contributor code of conduct.

Getting Started

1. We use the fork and pull model to manage changes. More information about forking a repository and making a Pull Request.
2. To build the lessons please install the dependencies.
3. For our lessons, you should branch from and submit pull requests against the `master` branch.
4. When editing lesson pages, you need only commit changes to the Markdown source files.
5. If you're looking for things to work on, please see the list of issues for this repository. Comments on issues and reviews of pull requests are equally welcome.

Dependencies

To build the lessons locally, install the following:

1. Gitbook

Install the Gitbook plugins:

```
$ gitbook install
```

Then (from the ostap-tutorials directory) build the pages and start a web server to host them:

```
$ gitbook serve
```

You can see your local version by using a web-browser to navigate to `http://localhost:4000` or wherever it says it's serving the book.

The title

Learning Objectives

- The starterkit lessons all start with objectives about the lesson
- Objective 2 with some *formatted* **text** ~~like~~ `this`

Basic formatting

You can make **bold**, *italic* and ~~strikethrough~~ text. Add relative links like this one and absolute links in a couple of different ways.

Have bulleted lists:

- Point 1
- Point 2
    - Sub point
        - Sub point
    - Sub point
- Point 2

Use numbered lists:

1. First
2. Second
    i. Second first
        i. Second first first
    ii. Second second
3. Third

LaTeX

You can use inline LaTeX maths such as talking about the decay $$D^{* +} \rightarrow \left(D^0 \rightarrow K^{-} \pi^{+} \right)$$.

Code highlighting

And have small lines of code inline like saying `print("Hello world")` or have multiple lines with syntax highlighting for python:

```python
import sys

def stderr_print(string):
    sys.stderr.write(string)

stderr_print("Hello world")
```

bash:

```bash
lb-run Bender/latest $SHELL
dst_dump -f -n 100 my_file.dst 2>&1 | tee log.log
```

and more!

Callouts

Prequisites

- Prequisite 1
- Prequisite 2

Objectives

- Objective 1
- Objective 2

Challenge

Set a challenge here, and the solution will remain hidden until it's clicked

- How to print?

Solution

The answer is:

```
print("Hello world")
```

Extra details that are hidden by default

Some extra details

Keypoints

- Summary point 1

- Summary point 2

Quotes

> This was said by someone

Tables

Simple tables are possible

First Header	Second Header
Content from cell 1	Content from cell 2
Content in the first column	Content in the second column

Images

Section types

This is a section

Subsections

And a subsection

Subsubsections

And a subsubsection