PyROOT in the Lab

Jean-François Caron

Queen's University

May 12, 2020

What is PyROOT?

PyROOT is a bridge allowing you to call C++ ROOT functions from a python program. It is automatically generated from the ROOT source code, so the classes and functions are all equivalent.

Pros

- Few new interfaces to learn.
- High-performance with built-in ROOT objects.
- Flexibility and scope of Python language and standard library.
- Can add-in 3rd-party numerical python libraries.

Cons

- Python-side performance can be bad.
- Need to code-switch between Python and C++.
- More libraries to install.
- Sometimes need workarounds for ROOT weirdness.

My strategy: use compiled C++ code with ROOT libraries for heavy number-crunching, but use PyROOT for exploration, interactive use, and plotting.

Your Primary Tool: TGraph

The ROOT TGraph is a basic 2D graph of X vs Y.

```
# python3 -i demo_tgraph.py
import ROOT, array

x = array.array("d",range(10)) # "d" for double-precision floating-point.
y = array.array("d",[0]*len(x))# array of ten zeros.

for i,xi in enumerate(x):
    y[i] = xi**2

g = ROOT.TGraph(len(x),x,y)
g.Draw("AL")

# https://docs.python.org/3/library/array.html             Python array module
# https://root.cern.ch/doc/master/classTGraph.html         TGraph Documentation
# https://root.cern.ch/doc/master/classTGraphPainter.html  TGraph Draw Options
# https://root.cern.ch/doc/master/classTMath.html          TMath Documentation
```

Numpy arrays can also be used instead of array.array.
You can also create an empty `ROOT.TGraph(N)` and fill the points in one by one using `g.SetPoint(i,a,b)`.
Exercises (3 minutes):

1. Try plotting `ROOT.TMath.Sin` or your favourite function.
2. Try `g.GetXaxis().SetTitle("foo")` and `g.SetTitle("bar")`.

Histograms: TH1D

Histograms are frequently used in particle physics. In ROOT they play a more central role than even TGraphs. You create them with a certain range of bins and fill them with values.

```python
# python3 -i demo_th1d.py
import ROOT

rng = ROOT.TRandom3(1234) # Random number generator object
# Parameters are: Number of bins, lowest edge, highest edge
h = ROOT.TH1D("h","Histogram Title", 10, 0, 10)
for i in range(500):
    value = rng.Gaus(5,1)
    h.Fill(value)
h.Draw()
# https://root.cern.ch/doc/master/classTH1.html       TH1* Documentation
# https://root.cern/doc/master/classTHistPainter.html TH1* Draw Options
# https://root.cern.ch/doc/master/classTRandom3.html  TRandom3 Documentation
```

The TH1D class is full-featured: you can set variable bin widths, fill with different weights, change bin statistics, interface to fitting, etc.

Note: there is no reason to use the other TH1* types.

Reading Files 0: Fake Data

So you can make graphs, but how do you get the data from a file into the program? First, let's generate an example text file to work with.

```
# python3 demo_generate_data.py
import ROOT, csv

outfilename = "demo_data_file.csv"
n_lines = 10000
random_seed = 1337                # Fixed seed for reproducibility
rng = ROOT.TRandom3(random_seed) # Random number generator object
t, tau = 0, 500 # Average time interval for simulated Poisson process.

outfile = open(outfilename,"w") # "write" mode.
outfile.write("# Generated using ROOT.TRandom3 with seed %d\r\n" % random_seed)
outfile.write("# tau = %d\r\n" % tau)
outfile.write("# time, binomial, gaussian\r\n")
writer = csv.writer(outfile)

for i in range(n_lines):
    t += rng.Exp(tau)           # Exponential distribution with tau
    var2 = rng.Binomial(20,0.2) # 20 trials, 0.2 chance of success
    var3 = rng.Gaus(0,1)        # central value 0, width 1
    writer.writerow([t,var2,var3])
outfile.close()

# References
# https://root.cern.ch/doc/master/classTRandom3.html
# https://docs.python.org/3/library/csv.html
```

Exercise (1 minute): Examine the output file with `wc` and `less`.

Reading Files 1: Manually Filling a TTree

```python
import ROOT, array, csv

infilename = "demo_data_file.csv"
outfilename = infilename.replace(".csv",".root")
treename = infilename.replace(".csv","")
outfile = ROOT.TFile(outfilename,"RECREATE") # Erases any existing file.

# Create a one-element python array to hold the value (could also use numpy).
time_arr = array.array("d",[0])
binomial_arr = array.array("l",[0]) # And so on for every variable...

# Create the tree and the branch manually.
t = ROOT.TTree(treename,"tree title")
t.Branch("time", time_arr, "time/D")              # D for doubles
t.Branch("binomial", binomial_arr, "binomial/L") # L for integers

# Now loop over the file manually:
with open(infilename,"r") as csvfile:
    reader = csv.reader(csvfile)
    for row in reader:
        if row[0].startswith("#"): # skip comment lines
            continue
        time_arr[0] = float(row[0])  # Important: change the CONTENT of the arrays.
        binomial_arr[0] = int(row[1])
        t.Fill()

# Write the data from the TFile to the actual file on disk.
outfile.Write()
outfile.Close()

# Reference: https://root.cern.ch/how/how-write-ttree-python
#            https://docs.python.org/3/library/array.html
```

Reading Files 2: TTree.ReadFile

```python
# python3 demo2_ttree_readfile.py
import ROOT

infilename = "demo_data_file.csv"
outfilename = infilename.replace(".csv",".root")
treename = infilename.replace(".csv","") # Give the tree the same name as the file.

# Create the ROOT TFile.
outfile = ROOT.TFile(outfilename,"RECREATE") # Erases any existing file.

# Create the tree.  TTrees are automatically added to the current TFile, if any.
t = ROOT.TTree(treename,treename)
# Define branches of a TTree
# The syntax is branchname/typecode:branchname/typecode...
branches = "time/D:binomial/L:gaussian/D" # L for integers, D for floats

t.ReadFile(infilename,branches)

# Write the data from the TFile to the actual file on disk.
outfile.Write()
outfile.Close()
# References:
# https://root.cern.ch/doc/master/classTFile.html
# https://root.cern.ch/doc/master/classTTree.html#a9c8da1fbc68221b31c21e55bddf72ce7
```

Working with TTrees

Exercise (5 minutes): enter these commands interactively in python.

```
# python3 -i demo7_work_trees.py
import ROOT

infile = ROOT.TFile("demo_data_file.root","READ")

# Check the contents of the file
infile.ls()

# Get the tree out of the file
tree = infile.Get("demo_data_file")
# NOTE: a failed "Get" returns <ROOT.TObject object at 0x(nil)>

# Show the contents of the 0th entry and number of entries
tree.Show(0)
N = tree.GetEntries()
print(N)

# Shows a summary of the contents of the whole tree.
tree.Print()

# You can also get the list of branches programmatically:
for b in tree.GetListOfBranches():
    print("branch:",b.GetName())

# Once "got", each branch can be accessed as a data member of the tree.
for i in range(5):
    tree.GetEntry(i)
    print(tree.time,tree.gaussian)
```

Making Plots

```
# python3 -i demo8_ttree_draw.py
import ROOT

infile = ROOT.TFile("demo_data_file.root","READ")
tree = infile.Get("demo_data_file")

tree.Draw("binomial") # ROOT has an algorithm to guess decent histogram boundaries.
htemp = ROOT.gROOT.FindObject("htemp") # Temporary histograms are called "htemp"
# NOTE: a failed FindObject returns <ROOT.TObject object at 0x(nil)>
h1 = htemp.Clone("h1") # Clone to make sure it doesn't get deleted.
input("Press Enter to continue.")

# You can give your histogram an explicit name and binning:
tree.Draw("binomial >> h2(10,0,15)")
h2 = ROOT.gROOT.FindObject("h2") # Bring "h2" over to the python side.
input("Press Enter to continue.")

tree.Draw("gaussian:time") # Unbinned 2D scatter plot TGraphs are made with y:x
g1 = ROOT.gROOT.FindObject("Graph").Clone("g1") # Temporary is called "Graph"
# NOTE: if you use the >>name notation, it instead makes a BINNED TH2*!
g1.Draw("AP")       # Remember to draw the clone before working on it further.
g1.SetTitle("Noise")

# References:
# https://root.cern.ch/doc/master/classTTree.html#a73450649dc6e54b5b94516c468523e45
```

Exercise (3 minutes):

1. Draw a histogram or graph of a quantity of your choosing, with proper axis labels.

Saving and Exporting

To properly save figures, you need to save the TCanvas, not the TGraph or TH1. The active canvas can be saved with `ROOT.gPad.SaveAs("foo.pdf")` or:

```
# python3 -i demo9_saving.py
import ROOT

infilename = "demo_data_file.root"
infile = ROOT.TFile(infilename,"READ")
treename = infilename.replace(".root","")
tree = infile.Get(treename)

# Create a TCanvas.  New canvases are automatically set to the active one.
c1 = ROOT.TCanvas("c1")

# Draw your thing.
tree.Draw("gaussian:time")
c1.SaveAs("plots/demo9_binomial.png") # png makes lightweight figures, but scale badly.
c1.SaveAs("plots/demo9_binomial.pdf") # scales well, but can be huge with lots of points.
c1.SaveAs("plots/demo9_binomial.tex") # Figure for inclusion in a LaTeX document.
c1.SaveAs("plots/demo9_binomial.C")   # Useful format to re-load into ROOT later.
```

Exercise (2 minutes):

1. Save the figure from the last exercise in the four formats shown.
2. Try to view the ouput files (as text or as figures) (`evince` for pdf, `eog` for png).
3. Look at the size difference in the output files with `ls -lh plots`.

Aside: The Power of PyROOT

You can add all sorts of functionality in Python. Here is a function I made to automatically timestamp and move a file. I use it before saving figures.

```
def ArchiveExisting(fname):
    """This function takes a filename (relative or absolute) and checks to see
    if such a file already exists.  If it doesn't, nothing is done.  If a file already
    exists, then it moves the existing file into a directory "old" in the same final
    directory as the file, and appends a timestamp to the filename of the moved file.
    If "old" does not exist, it is created."""
    import os, datetime
    if not os.path.exists(fname):
        return
    head,tail = os.path.split(fname)
    olddir = os.path.join(head,"old")
    if not os.path.exists(olddir):
        os.mkdir(olddir)
    elif not os.path.isdir(olddir):
        raise RuntimeError("Need to create directory "+
                           olddir+" but file already exists with that name")
    timestamp = datetime.datetime.now().strftime("_%Y%m%d%H%M%S")
    barename,ext = os.path.splitext(tail)
    archivename = os.path.join(olddir,barename+timestamp+ext)
    os.rename(fname,archivename)
    return
```

You could write this in C++ too, but in Python it's way easier!
Homework: Use the python documentation to figure out exactly how this works.

Advanced Drawing

```
# python3 -i demo9b_advanced_drawing.py
import ROOT

infile = ROOT.TFile("demo_data_file.root","READ")
tree = infile.Get("demo_data_file")

tree.GetEntry(0)
t0 = tree.time
drawstring = "TMath::Power(time/(1.0*Entry$)-500,2):((time-%g)/1e3)" % t0
tree.Draw(drawstring)

g1 = ROOT.gROOT.FindObject("Graph").Clone("g1")
g1.Draw("ALP")
g1.SetMaximum(2500) # Setting the range in Y is different than in X.
g1.SetMinimum(0)
g1.GetXaxis().SetRangeUser(0,1000) # "User" coordinates means in the graph units.

# Arbitrary C++-style expressions are allowed with the names in the TTree.
# The C++ ternary operator (A ? B : C) is available, so you can do anything!
# Special names are also available: Entry$, Entries$, Sum$, etc.
# Also functions from ROOT.TMath:: and the C++ std::cmath modules. NOTE: "::"
# Reference: https://root.cern.ch/doc/master/classTFormula.html
#            https://root.cern.ch/root/html524/TMath.html
#            https://www.cplusplus.com/reference/cmath/
# https://root.cern.ch/doc/master/classTTree.html#a73450649dc6e54b5b94516c468523e45
# https://root.cern.ch/root/htmldoc/guides/users-guide/ROOTUsersGuide.html Sec. 9.3.3
```

This method can take you surprisingly far, and it's very fast because the looping happens outside of Python.

Stacking Histograms

```
# python3 -i demo10_stacks.py
import ROOT

infile = ROOT.TFile("demo_data_file.root","READ")
tree = infile.Get("demo_data_file")

c1 = ROOT.TCanvas()
tree.Draw("gaussian >> h1")
h1 = ROOT.gROOT.FindObject("h1")
tree.Draw("gaussian/(1.0*binomial) >> h2")
h2 = ROOT.gROOT.FindObject("h2")
h2.SetLineColor(ROOT.kRed)

hs = ROOT.THStack("hs","THStack")
hs.Add(h1)
hs.Add(h2)
hs.Draw("NOSTACK") # Default draw stacks 'em vertically

# 2D coordinates go X1,Y1,X2,Y2, (0,0) is at bottom left
# NDC means Normalized Device Coordinates
tl = ROOT.TLegend(0.6,0.6,0.9,0.9,"Header Text","NDC")
tl.AddEntry(h1,"Gaussian")
tl.AddEntry(h2,"Gaus/Binom")
tl.Draw()

# Reference:
# https://root.cern.ch/doc/master/classTHStack.html
# https://root.cern.ch/doc/master/classTLegend.html
```

Stacking Graphs

```python
# python3 -i demo11_stacks.py
import ROOT

infile = ROOT.TFile("demo_data_file.root","READ")
tree = infile.Get("demo_data_file")

c1 = ROOT.TCanvas()
tree.Draw("gaussian:time")
g1 = ROOT.gROOT.FindObject("Graph").Clone("g1")
tree.Draw("gaussian/binomial:time")
g2 = ROOT.gROOT.FindObject("Graph").Clone("g2")
g2.SetLineColor(ROOT.kRed)

mg = ROOT.TMultiGraph("mg","TMultiGraph")
mg.Add(g1)
mg.Add(g2)
mg.Draw("AL") # Don't need NOSTACK

# 2D coordinates go X1,Y1,X2,Y2, (0,0) is at bottom left
tl = ROOT.TLegend(0.6,0.1,0.9,0.3,"","NDC")
tl.AddEntry(g1,"Gaussian")
tl.AddEntry(g2,"Gaus/Binom")
tl.Draw()

# Reference:
# https://root.cern.ch/doc/master/classTMultiGraph.html
# https://root.cern.ch/doc/master/classTLegend.html
```

Error Bars

Use TGraphErrors, TGraphAsymmErrors, etc.

```
# python3 -i demo12_errors.py
import ROOT, array

x = array.array("d",range(10))
y = array.array("d",[0, 1, 4, 9, 16, 25, 36, 49, 64, 81])
x_errors = ROOT.nullptr # Use ROOT.nullptr where you'd otherwise send 0 or NULL.

y_errors = array.array("d",[0]*len(y))
for i,yi in enumerate(y):
    y_errors[i] = ROOT.TMath.Sqrt(yi)

g = ROOT.TGraphErrors(len(x), x, y, x_errors, y_errors)
g.Draw("AP")

# Reference: https://root.cern.ch/doc/master/classTGraphErrors.html
#            https://root.cern.ch/doc/master/classTGraphAsymmErrors.html
```

Unfortunately you cannot create TGraphErrors directly with TTree.Draw.

Aside: The Power of PyROOT 2

```python
import numbers, array, ROOT

def get_error(e,N):
    if e is None:
        error = ROOT.nullptr
    elif isinstance(e, numbers.Number):
        error = array.array('d',[e]*N)
    else:
        assert len(e) == N
        error = array.array('d',e)
    return error

def AddErrors(g,ex = None,ey = None):
    """Takes a TGraph and turns it into a TGraphErrors with either
    fixed or array errors."""
    N = g.GetN()
    xbuf = g.GetX() # Returns a "read-write buffer" which is a dumb array.
    xbuf.SetSize(N) # So we have to tell it what size it is.
    x = array.array('d',xbuf)
    ybuf = g.GetY()
    ybuf.SetSize(N)
    y = array.array('d',ybuf)

    xerror = get_error(ex,N)
    yerror = get_error(ey,N)

    ge = ROOT.TGraphErrors(N,x,y,xerror,yerror)
    ge.SetName(g.GetName()+"_e")
    return ge
```

Basic Fitting 1

ROOT has too many ways to fit things. This is just one way.

```
# python3 -i demo13_fit.py
import ROOT

infile = ROOT.TFile("demo_data_file.root","READ")
tree = infile.Get("demo_data_file")

tree.Draw("gaussian >> h1")
h1 = ROOT.gROOT.FindObject("h1")

ftr_p = h1.Fit("gaus","S") # gaus, expo, pol0, pol1...polN are shortcuts.
# Fit normally returns an empty TFitResultPtr, option "S" makes it Store the results.
ftr = ftr_p.Get() # You have to Get the TFitResult from the pointer.  It's dumb.

central_value = ftr.Parameter(1)
width = ftr.Parameter(2)

# Reference:
# https://root.cern.ch/doc/master/classTF1.html
# https://root.cern.ch/doc/master/classTGraph.html#a61269bcd47a57296f0f1d57ceff8feeb
# https://root.cern.ch/doc/master/classTGraph.html#aa978c8ee0162e661eae795f6f3a35589
```

Note the same process works for TGraphs.

Basic Fitting 2

You can define your own function with a TF1 or with a C++ function.

```
# python3 -i demo14_fit2.py
import ROOT

infile = ROOT.TFile("demo_data_file.root","READ")
tree = infile.Get("demo_data_file")

c1 = ROOT.TCanvas()
# The first empty "" is a TCut string the second is a draw option.
tree.Draw("sin(time/1000.0):time/1000.0","","",100,0) # Draw 100 entries starting at 0.
g1 = ROOT.gROOT.FindObject("Graph").Clone("g1")
g1.Draw("ALP")
input("Press Enter to continue.") # Pause

f1 = ROOT.TF1("f1","[0]*sin(x/[1] + [2]) + [3]",0,50) # Generic sine function.
f1.SetParNames("scale","period","phase","offset") # Optional.
f1.SetParameters(1,1,0,0) # Set initial parameter guesses/
# f1.Draw("same") # To see if our initial guess is close.
ftr_p = g1.Fit(f1,"S")

# Reference:
# https://root.cern.ch/doc/master/classTF1.html
# https://root.cern.ch/doc/master/classTGraph.html#a61269bcd47a57296f0f1d57ceff8feeb
# https://root.cern.ch/doc/master/classTGraph.html#aa978c8ee0162e661eae795f6f3a35589
```

Neglected Topics

1. "Collection" objects in TTrees
2. 3D and higher plots, profile plots
3. TDataFrame
4. "Out parameters"
5. Including your own compiled C++ ROOT code in PyROOT
6. TTree.Scan

Getting Help

1. The ROOT forum is very active: https://root-forum.cern.ch/
2. ROOT User's Guide: https://root.cern.ch/root/htmldoc/guides/users-guide/ROOTUsersGuide.html
3. ROOT Reference Guide: https://root.cern/doc/master/
4. PyROOT-specific tutorials: https://root.cern/doc/master/group__tutorial__pyroot.html
5. The FreeNode IRC channels #python and #c++-basic are helpful.
6. Python official documentation: https://docs.python.org/3/

Major Exercise/Homework

You will probably not finish (or start?) this during the workstop time.

1. Generate a new data set (as on slide 5) with the binomial distribution for N = 20, 200, and 2000 trials.
2. Convert this data set to a ROOT TFile with a TTree in it (as on slide 7).
3. Plot the distributions in TH1Ds and put them together in a THStack (as on slide 13).
4. Fit each of the distributions with a Gaussian function, note the Chi2/NDf (as on slide 17).
5. Save the produced plot in your favourite format (as on slide 10).
6. Bonus: put the fit results in a TPaveText on top of the THStack plot.