ROOT – Python Interface

ROOT hat ein Python binding (PyROOT) und vereinigt damit kompilierte C++ Bibliotheken mit der Flexibilität von Python. Die Bindung wird automatisch aufgesetzt. Für die aktuelle Version 6.26.06 wurde PyROOT erheblich verbessert und stabilisiert. ROOT und pyROOT können entweder direkt kompiliert, in einer virtuellen Python Umgebung wie `anaconda3` installiert oder im CIP pool in der jupyter Lab Installation verwendet werden.

- Installation von PyROOT beim schon diskutierten ROOT Kompilieren
  - Wenn python include files vorhanden sind, wird PyROOT automatisch kompiliert.
  - Einige Tipps:
    Entscheiden Sie sich für eine Python Version, z.B. python3 und installieren Sie keine Mischung von python2 und python3.
    Installieren Sie `header files` der gewünschten Pakete (falls vorhanden).

Befehle für OpenSuSE als superuser

```
>$ zypper python3 python3-devel python3-numpy
          python3-numpy-devel jupyter metakernel
```

Achten Sie beim Konfigurieren mit cmake (siehe obigen link und das entspechende Text file) darauf, ob und welche Version für python gefunden wurde.

ROOT – Python Interface

- Installation von PyROOT in `anaconda3`
  - `anaconda3` Installation: Herunterladen von Anaconda3-2020.11-Linux-x86_64.sh
    Installation der erforderlichen Pakete entsprechend Ihrem Betriebssystem als `superuser`, aber `anaconda3` installieren Sie als `user`

    ```
    >$ bash  Anaconda3-2020.11-Linux-x86_64.sh
    ```

    Beantworten Sie die Frage `Soll das environment ausgeführt werden` mit `Yes` und stellen Sie sicher, das es ausgeführt ist. Am besten kopieren Sie die Zeilen, die dadurch in `.bashrc` hinzugefügt wurden in ein separates file `SetPathAnaconda.sh` und führen es mit `sh SetPathAnaconda.sh` aus, wenn `anaconda3` verwendet werden soll.
  - Benutzung:
- `>$ conda create -n myFirstEnv python=3.6 ; conda myFirstEnv activate`

  Installation von zu verwendenden Paketen mit `conda oder pip`

  ```
  (myFirstEnv) >$ conda install matplotlib scikit-learn
  (myFirstEnv) >$ pip install numpy wheel ….
  ```

  ROOT installation aus conda-forge

  ```
  (myFirstEnv) >$ conda install -c conda-forge root
  ```

  Verlassen von conda

  ```
  (myFirstEnv) >$ conda deactivate
  ```

In der conda Umgebung `(myFirstEnv)`

ROOT – Python Interface

PyROOT kann auch in der jupyter Lab Installation des CIP Pools unter der folgenden Adresse verwendet werden
https://jupyter3.kip.uni-heidelberg.de/

- Benutzung
  - Wählen Sie unter “Notebook“ oder “Console“ im Launcher “Python3“ aus Sie erhalten Zugang zu einem File in das python code eingegeben werden kann. In einem Notebook können Ihre Eingaben als iPython Notebook Files gespeichert und ausgeführt werden (files *.ipynb).
  - Im Notebook steht pyROOT der ROOT version 6.22.02 zur Verfügung, andere nützliche installierte Pakete sind scikit-learn, numpy, matplotlib, iminuit, ……
  - Die Bedienung ist sehr intuitiv, die Dokumentation ist unter https://jupyterlab.readthedocs.io/en/stable/ zu finden.

ROOT – Python Interface

Eine kurze python Einführung finden Sie unter
https://www.physi.uni-heidelberg.de/~marks/root_einfuehrung/Folien/01_intro_python.pdf

ROOT erlaubt einen interaktiven Zugang zum Python Interface entweder mit dem python Interpreter oder mit Hilfe von Web basierenden jupyter notebooks

- Python shell
- - python shell in einem Terminal `python3` oder `ipython3`

  - python Macros mit ROOT code `python3 -i HistLandau.py`
    `ipython3 -i HistLandau.py`

HistLandau.py

- Start eines Jupyter notebooks `root --notebook`
  Der Befehl öffnet einen Web Browser mit einer Jupyter Session in der `.ipynb` notebooks ausgeführt werden können.
- Beispiel:

HistLandau.ipynb

```
import ROOT
%jsroot on
c = ROOT.TCanvas()
f1 = ROOT.TF1("func1", "sin(x)", 0, 10)
f1.Draw()
c.Draw() #  show graphics!
```

PyROOT

Erste Schritte in PyROOT: Fast alle bisher betrachteten ROOT Beispiele lassen sich mehr oder weniger einfach auf python übertragen. Beispiele sind auch im tutorial Bereich jeder ROOT Installation zu finden oder in https://root.cern/manual/python/

- Füllen eines Histogramms mit einer Gauss-Verteilung.

```
>$ python3
Python 3.6.12 (default, Dec 02 2020, 09:44:23) [GCC] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>> import ROOT
>>> h = ROOT.TH1F("myHist", "myTitle", 64, -4, 4)
>>> h.FillRandom("gaus")
>>> h.Draw()
```

- Zeichnen einer Funktion

```
>>> f = ROOT.TF1("f1", "sin(x)/x", 0., 10.)
>>> f.Draw()
```

Direkt in Linux - oder in ancondo3 python öffnen die obigen Befehle ein TCanvas. In einem jupyter notebook muss das TCanvas instanziert, gezeichnet werden, es enthält dann die zuvor hinzugefügten Elemente.

```
>>> c = TCanvas('c', 'My Plot',700, 500 )
>>> c.Draw()
```

PyROOT

Wie schon im ROOT interpreter können python Befehle in “python macros“ gespeichert werden (files *.py). Damit ROOT Canvases sichtbar bleiben, sollten PyROOT Anwendungen in der Linux oder anaconda3 shell interaktiv gestartet werden.

```
>$ python3 -i myMacro.py
```

- Ähnliche Eigenschaften in python nutzbar wie in C++, e.g.

```
>>> import ROOT
>>> import sys
>>> if len( sys . argv ) != 3:
>>>         print "USAGE:%s <input file> <output file>"
                                          %(sys . argv [0])
>>>         sys . exit (1)
>>> inFileName = sys . argv [1]
>>> outFileName = sys . argv [2]
>>> print "Reading from ",inFileName,"and writing to",
                                                 outFileName
```

- Up to factor 10 more CPU time same task → C++
+ Impression of more simple usage → python

ReadParameter.py

Aufrufen des python scripts in der shell

```
>$ python ReadParameter.py Data.root hist.root
```

PyROOT

- Input/ Output → Important: get the physics data into the python environment
  - `text` format and `root` format including objects
  - text → use numpy, z.B. die Daten werden in numpy array kopiert, es gibt Optione, die auch komplexe Strukturen unterstützen

```
>>> import numpy as np
>>> data = np.genfromtxt('D0Mass.txt', dtype='d')
```

  - root → benutze PyROOT auch um python oder numpy arrays zu füllen

```
>>> file = TFile("FitTestTTree.root",'update')
>>> tree = file.Get("FitTest")
>>> sig = TH1D( 'signal', 'Signaldistribution', 100, 0., 250.)

>>> N = tree.GetEntries()
>>> x = np.zeros(N)
>>> for i in range(N):
>>>    tree.GetEntry(i)
>>>    x[i] = tree.signal_1
>>>    sig.Fill(tree.signal_1)
```

  - Output

```
>>> myfile = TFile( 'py-fillFile.root', 'RECREATE' )
>>> myHisto.Write()
>>> myfile.Close()
```

PyROOT - Objekte

- Bei der Instanzierung einer C++ Klasse in pyROOT wird ein C++ Objekt und ein python Proxy Objekt erzeugt. Der python Aufruf wird an das C++ Objekt weitergeleitet. Es gibt Funktionen, die den Zugriff auf den python Proxy mit `cppyy` erlauben.

- obtain the address of an internal C++ object from its Python proxy

```
>>> import ROOT
>>> ROOT.gInterpreter.Declare("struct Particle{int
        charge = 1; double mass = 30 ; double
                                    energy=400 ;};")

>>> pion = ROOT.Particle()
>>> print("Address of pion:",ROOT.addressof(pion))
>>> print("Mass of pion:",pion.mass)
>>> pion.mass = 55
>>> print("Address of mass in
                  pion:",ROOT.addressof(pion,'mass'))
>>> print("Mass of pion:",pion.mass)
```

PyROOT - Objekte

- Bei der Instanzierung einer C++ Klasse in pyROOT wird ein C++ Objekt und ein python Proxy Objekt erzeugt. Der python Aufruf wird an das C++ Objekt weitergeleitet. Es gibt Funktionen, die den Zugriff auf den python Proxy mit `cppyy` erlauben.

- Create a C++ templated class instance

```
>>> import ROOT
>>> vInt = ROOT.std.vector[int]()
>>> for i in range(0, 10):
        vInt.push_back(i)
>>> for i in vInt:
        print(i, end=' ')
```

pyObjects.py

PyROOT - Funktionen

- TF{1,2,3} Konstruktoren benötigen als Argument Python Funktionen oder Klassen, die Operationen wie plotting and fitting erlauben. Typischerweise werden ein Datenarray mit {1, 2, 3} Dimensionen und ein Parameterarray zur Verfügung gestellt. Die `SetParameter` Methoden setzen Funktionsparameter nach der Instanzierung der TF Objekte.

  - Use a function instancianting a TF object

```
>>> import ROOT
>>> def myFunc(arr, par):
        return par[0]+par[1]*arr[0]+par[2]*arr[0]*arr[0]
>>> f = ROOT.TF1('pyfunc', myFunc, -5., 5.,3)
>>> f.SetParameters(5., 2., 0.5)
>>> c = ROOT.TCanvas()
>>> f.Draw()
```

Funktionsdefinition mit Daten- und Parameterarray

Parameter

pyFunctions.py

PyROOT - Funktionen

- TF{1,2,3} Konstruktoren benötigen als Argument Python Funktionen oder Klassen, die Operationen wie plotting and fitting erlauben. Typischerweise werden ein Datenarray mit {1, 2, 3} Dimensionen und ein Parameterarray zur Verfügung gestellt. Die `SetParameter` Methoden setzen Funktionsparameter nach der Instanzierung der TF Objekte.

  - Use a class instancianting a TF object

```
>>> import ROOT
>>> class polyFunc:
      def __call__(self, arr, par):
        return par[0]+par[1]*arr[0]+par[2]*arr[0]*arr[0]
>>> myPol = PolyFunc()
>>> f = ROOT.TF1('pyfunc', myPol, -5., 5.,3)
>>> f.SetParameters(5., 2., 0.5)
>>> c = ROOT.TCanvas()
>>> f.Draw()
```

Klassendefinition mit Daten- und Parameterarray

Parameter

pyClass.py

PyROOT - TTree

- Schreiben eines TTree Objektes:

```
>>> from ROOT import TFile, TTree, gRandom
>>> from array import array

>>> f = TFile("myTree.root", 'recreate')
>>> tree = TTree("myTree", "My first Tree in python")

>>> sig = array('d',[0])
>>> tree.Branch("sig",  sig,  'sig/D')

>>> for i in range(10000):
            sig = gRandom.Gaus(5,1)
            tree.Fill()
>>> f.Write()
>>> f.Close()
```

serves as pointer

create branch

- Lesen eines TTree Objektes:

```
>>> N = tree.GetEntries()

>>> sig = array('d',[0])
>>> tree.Branch("sig",  sig,  'sig/D')

>>> for i in range(N):
            tree.GetEntry(i)
            print(tree.sig)
```

PyROOT - TTree

- Schreiben eines TTree Objektes:

```
>>> from ROOT import TFile, TTree, gRandom
>>> from array import array

>>> f = TFile("myTree.root", 'recreate')
>>> tree = TTree("myTree", "My first Tree in python")

>>> sig = array('d',[0])
>>> tree.Branch("sig",  sig,  'sig/D')
```

serves as pointer

create branch

Ein komplexeres Beispiel zum Schreiben eines ROOT trees ist die im File Root.pdf Seite 80 beschriebene Simulation eines π^0 Zerfalls.

pi0DecayCalo.py

- Lesen eines TTree Objektes:

```
>>> N = tree.GetEntries()

>>> sig = array('d',[0])
>>> tree.Branch("sig",  sig,  'sig/D')

>>> for i in range(N):
        tree.GetEntry(i)
        print(tree.sig)
```

PyROOT - Beispiele

- PyROOT Beispiele sind in der tutorial section im Web und dem tutorial directory der ROOT Installation
  https://root.cern.ch/doc/master/group__tutorial__pyroot.html

- Einige an unseren Kurs angelehnte Beispiele sind im Beispielbereich in form von iPython files zu finden.
  - reading text file and gaussian fit → histFit.ipynb
  - LSQ fit of a polynomial to numpy array in Minuit 2 with iminuit and plot → iminuitFit.ipynb
  - Graph fitting with options using a python function with confidence level plotting → fitGraph.ipynb
  - Read root tree and fit gaus + exponential defining the fit function in python and as TFormula → fitTestAnalyse.ipynb
  - rooFit example of fitting extended gaussian PDF and plotting residuals → rooSimple.ipynb

Zusammenfassung

- PyROOT ist sicher eine extrem nützliche Ergänzung zum ROOT Datenanalyse Environment → das Konvertieren unserer Kurs Beispiele in python code ist einfach.
- Insbesondere gibt es Zugang von in ROOT gespeicherten Daten zu Machine Learning Umgebungen wie scikit-learn und TensorFlow mit dem einfach zu bedienendem Keras.
- Ein anderes Tools zum Einlesen von in ROOT gespeicherten Daten in numpy und awkward arrays ist uproot (ohne ROOT Installation nutzbar)