SFrame Analysis on D3PD

Marcello Barisonzi

DESY

Introduction to SFrame

- SFrame is a ROOT-based analysis package
- Designed with SUSY physics in mind: multiple analysis cycles that reduce amount of data at each passage
- Easily adaptable to single-cycle analyses
- Advantages over ROOT: faster (as branches connection customizable), easier histogram management, sample mixing with luminosity scaling, dedicated Ganga plugin
- SFrame reference page:
  https://twiki.cern.ch/twiki/bin/view/Main/SFramePage

Installation

- To install Sframe on your local directory do the following.
- Set the ATLAS CVS root in your environment

```
export CVSROOT = :kserver:atlas-sw.cern.ch:/atlascvs
```

- Get a kerberos ticket at CERN, then check out:

```
cvs co -r SFrame-02-01-05 -d SFrame groups/sframe/SFrame
```

- Initialize ROOT at NAF with `ini root`
- Get into the SFrame directory and compile:

```
source setup.sh
gmake
```

Package structure

- The package contains three subdirs:
  - SFrame/core. Main classes of the framework. Stay away from it, unless you know what to do.
  - SFrame/plug-ins. Shared classes that can be useful in physics analyses.
  - SFrame/user. This is your home, where you will develop your analysis code.

Creating your code

- Change to the SFrame/user directory, and type:

  `sframe_create_cycle.py -n MyNAFAnalysis`

- This script creates three new skeleton files:
  - `./include/MyNAFAnalysis.h`
  - `./src/MyNAFAnalysis.cxx`
  - `./config/MyNAFAnalysis_config.xml`
- We will edit these (almost empty) files to make an analysis

MyNAFAnalysis.cxx

- BeginCycle(): Function called once at the beginning of execution. You can use it to perform initial configuration of the cycle and all the code that it uses.
- BeginInputData(): Function called once before processing each of the input data types. SFrame creates one output file per input data type. Initialisation of output objects and declaration of the output variables to be done here.
- BeginInputFile(): For each new input file the user has to connect his input variables.
- ExecuteEvent(): This is the main analysis function that is called for each event. It receives the weight of the event, as it is calculated by the framework from the luminosities and generator cuts defined in the XML configuration.
- EndInputData(): Function called as last before the processing of the input data type is finished, and the output file is closed. Any histogram finalisation should be performed here.
- EndCycle(): Function called once at the end of the cycle execution. Any finalisation steps should be done here. (e.g. closing helper files.)

Exercise I: How to plot a pT spectrum

- Suppose you want to plot the pT spectrum of Jets that are not overlapping with Electrons, as extracted from a D3PD How would you do that in SFrame?
- You need the variable names from the D3PD file, you need to tell SFrame to retrieve that specific variables from the Tree that contains them, and then you need to book and fill a histogram for all Jets that do not overlap with an Electron
- Let's follow the procedure step by step.

What variables do I need?

- D3PD Ntuples are generated by EV-like tools, which are very flexible
- Documentation helps, but maybe the name of the variables have been changed, better to check directly on the D3PD file.
- Browse the file in ROOT and look for the FullReco0 tree (that's where the reconstructed objects are)
- You should find three interestering variables: PJet_N, PJet_p_T and PJet_Electron_Overlap

Modifying the include file

- Now edit `include/MyNAFAnalysis.h` to declare the new variables. Type just below `private:`

```
std::string m_RecoTreeName;
int m_PJet_N;
std::vector<double>* m_PJet_p_T;
std::vector<int>* m_PJet_Electron_Overlap;
```

- The string will be used to define the tree name, while the tree other variables should be obvious.
- The variable names do not need to be the same as the DPD file's. However this way is easier to memorize

Source Code I: The Constructor

- Now that the include file is ready, let's edit `src/MyNAFAnalysis.cxx`
- The first thing to do is to modify the constructor `MyNAFAnalysis::MyNAFAnalysis()`
- Remember we defined two vectors in the include file: they have to be initialized to zero, otherwise ROOT will crash
- So your constructor should look like:

```
: SCycleBase(),
  m_PJet_p_T(0),
  m_PJet_Electron_Overlap(0)
```

Source Code II: Trees and Variables

- While we are still in the constructor, let's add this line:

```
DeclareProperty( "RecoTreeString", m_RecoTreeName );
```

this tells SFrame to read the name of the reco tree from a config xml file.

- Then, to make SFrame retrieve the variables we want, add these lines in `MyNAFAnalysis::BeginInputFile()`:

```
ConnectVariable( m_RecoTreeName.c_str(), "PJet_N",  m_PJet_N );
ConnectVariable( m_RecoTreeName.c_str(), "PJet_p_T", m_PJet_p_T );
ConnectVariable( m_RecoTreeName.c_str(), "PJet_Electron_Overlap",
m_PJet_Electron_Overlap );
```

Source Code III: executing the event

- Finally, edit `MyNAFAnalysis::ExecuteEvent()`:

```
for(Int_t i = 0; i <  m_PJet_N; i++) {
    if( m_PJet_Electron_Overlap->at(i) == 0) {
                    Double_t pt = (*m_PJet_p_T)[i];
                    Book( TH1F( "h1", "good jet pt",
                          100, 0.0, 1e6 ) )->Fill(pt, 1.);
        }
    }
```

- Try to compile, it should work!
- Please note:
  - How I used two different ways to dereference the vector of pointers; you can pick the one you like best
  - How the `Book()` utility function lets you declare and fill your histograms in one go!

XML Configuration

- The last step left is to edit the configuration file

  `config/MyNAFAnalysis_config.xml`

- Search and replace the following lines:

```
<Library Name="libSFrameUser" />
<In FileName="YourInputFileComesHere"/>
<Item Name="RecoTreeString" Value="FullReco0" />
```

- Change the `<InputData Lumi>` to 1.0 and add inbetween `<InputData></InputData>`:

```
<InputTree Name="FullReco0" />
```

- Now you are ready to run:

```
sframe_main config/MyNAFAnalysis_config.xml
```

Exercise II

- We plotted the pT of the jets in a histogram, but what if we want to store it in an output tree?
- Declare new vector in include file:

```
std::vector<double> m_GoodJet_p_T;
```

- Declare the output variable in `BeginInputData()`:

```
DeclareVariable( m_GoodJet_p_T, "GoodJet_p_T" );
```

- Once you have computed the pt of the jet, push the value in the vector, and don't forget to clear the vector at the beginning of each event!
- Add this to the XML file: `<OutputTree Name="MyNAFAnalysisTree" />`
- Now compile and run again, and check that your histogram variable and your ntuple variable agree!

Further Exercises

- Now you are on your own, feel free to try things out
- Add more variables from the Reco tree to improve your analysis (muon overlap?)
- Add variables from a DIFFERENT tree (Truth?)
- Use output of 1st cycle as input for 2nd cycle
- If you feel adventurous, try to run your job with Ganga (see wiki page for info)