Mathematica in HEP - Large Scale Data Processing and ROOT Interoperability

Manuel Guth - University of Freiburg
Sebastian White - CERN/U. Virginia

HEP Diana 30.10.2017

Structure

- Motivation
- Mathematica calls ROOT
- ROOT calls Mathematica
- Wolfram Cloud
- Summary

Motivation

- Mathematica widely used in theoretical physics
  - Possibility to interact with experimental side?
- How to make use of analytical Mathematica tools within ROOT?
- Is there a quick tool to check data quality e.g. at test beam?
- What about direct analysis of big data sets from experiment?

Mathematica Importer for ROOTFiles

- Taking advantage of nice and simple plotting features of Mathematica
- Simple compared to other programming languages
- Allow theorists to directly use data, provided from experiments
- Detailed description:

  http://library.wolfram.com/infocenter/Articles/7793/

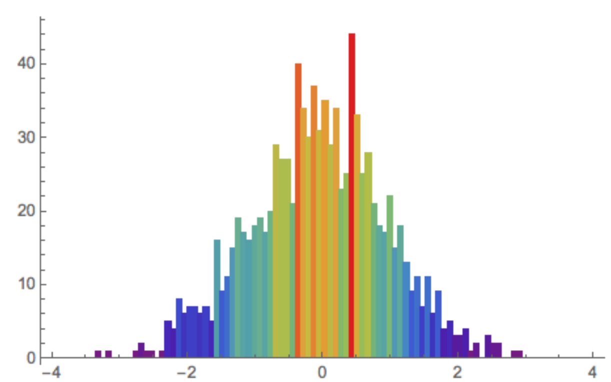
```
(* This imports the histogram data of a given TH1F object. *)
histdata = Import["demo.root", {"ROOT", "TH1FData", "h7"}];
```

The data is of the form:
{ {x1, Δx1, count1, error1}, {x2, Δx2, count2, error2}, ... }

```
(* show first 10 entries in a grid *)
head = {"x", "Δx", "count", "Δcount"};
Grid[Join[{head}, Take[histdata, 10]], Frame → All]
```

x	Δx	count	Δcount
-4.	0.08	0.	0.
-3.92	0.08	0.	0.
-3.84	0.08	0.	0.
-3.76	0.08	0.	0.
-3.68	0.08	0.	0.
-3.6	0.08	0.	0.
-3.52	0.08	0.	0.
-3.44	0.08	0.	0.
-3.36	0.08	1.	1.
-3.28	0.08	0.	0.

```
(* Options available to Histogram[] can be passed directly. *)
graphics2 = Import["demo.root", {"ROOT", "TH1FGraphics", "h7"},
  ColorFunction → Function[{height}, ColorData["Rainbow"][height]]]
```

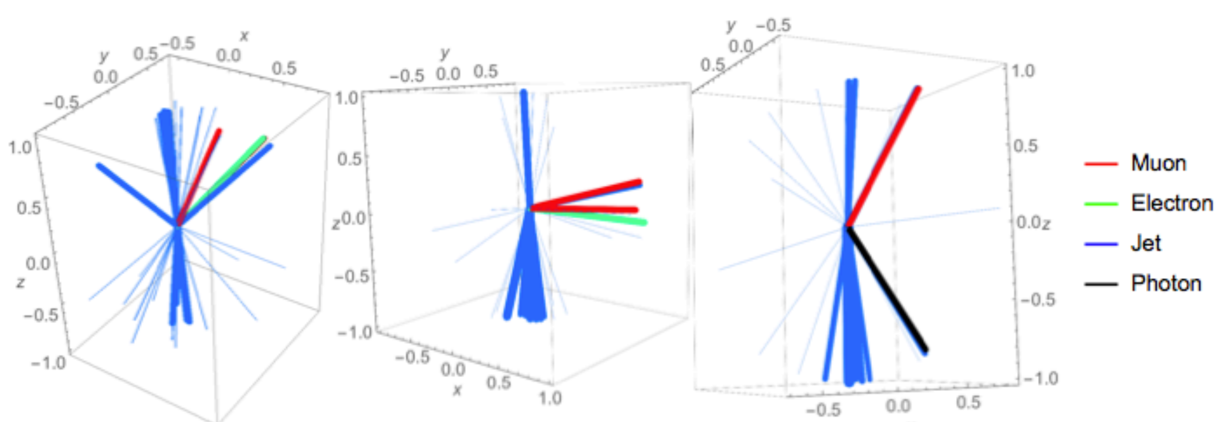


Mathematica Importer for ROOTFiles

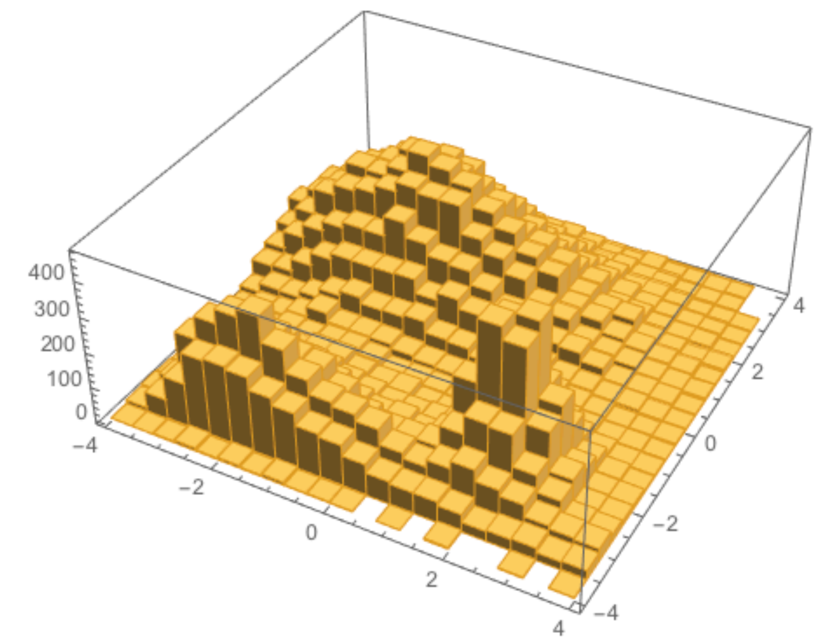
- LHC Data Exploration from Natalia Kovalchuk: http://community.wolfram.com/groups/-/m/t/1137265

```
momMuons = threeMomentum[nEv, "Particles", "Muons", All];
momJets = threeMomentum[nEv, "Particles", "Jets", All];
momElectrons = threeMomentum[nEv, "Particles", "Electrons", All];
momPhotons = threeMomentum[nEv, "Particles", "Photons", All];

Legended[Graphics3D[{Style[Line[{{0, 0, 0}, Normalize[#]}], Hue[.99],
      Thickness[Log@Norm[#]/200]] & /@ momMuons,
   Style[Line[{{0, 0, 0}, Normalize[#]}], Hue[.6],
      Thickness[Log@Norm[#]/300]] & /@ momJets,
   Style[Line[{{0, 0, 0}, Normalize[#]}], Hue[.43],
      Thickness[Log@Norm[#]/200]] & /@ momElectrons,
   Style[Line[{{0, 0, 0}, Normalize[#]}],
      Thickness[Log@Norm[#]/200]] & /@ momPhotons}, Axes -> True,
  AxesLabel -> {x, y, z} ],
 LineLegend[{Red, Green, Blue, Black}, {"Muon", "Jets", "Electron",
   "Photon"}]]
```




```
(* Import the histogram directly as a Graphics *)
graphics3D = Import["th2f.root", {"ROOT", "TH2FGraphics", "h2"}]
```



- Nice starting point for undergraduated students

Mathematica Importer for ROOTFiles

- Feedback
  - Marvin Johnson (DUNE)
    - “Mathematica’s rich set of built in functions eliminates the need to generate my own code either in ROOT or Python. [..] allows real time analysis of data from test beams”
    - Missing Subfolder support from ROOTFile importer
  - Natalia Kovalchuk
    - Would need expansion of data type
  - Since Mathematica Importer for ROOTFiles is back working (minor changes were needed) -> new features can be implemented (need feedback from community)

ROOT calls Mathematica

- ROOT can interact via MathLink with Mathematica
- Mathlink is included in the Mathematica installation
- Supported for Linux and macOS
- https://root.cern.ch/how/how-use-mathematica-root

```
root [0] startMathematica()
Started Mathematica Engine...
(int)0
root [1] callMathematicaGamma(5.0)
(double)2.400000000000000e+01
root [2] callMathematicaGamma(5.45)
(double)4.83037558000228415e+01
root [3] stopMathematica()
root [4] .q
```

ROOT calls Mathematica

- If there is no big interchange necessary
    - Parser from ROOT to wolframscript would be easy
    - Interplay only possible via strings
    - For small applications maybe more useful e.g. calculating an analytical Integral

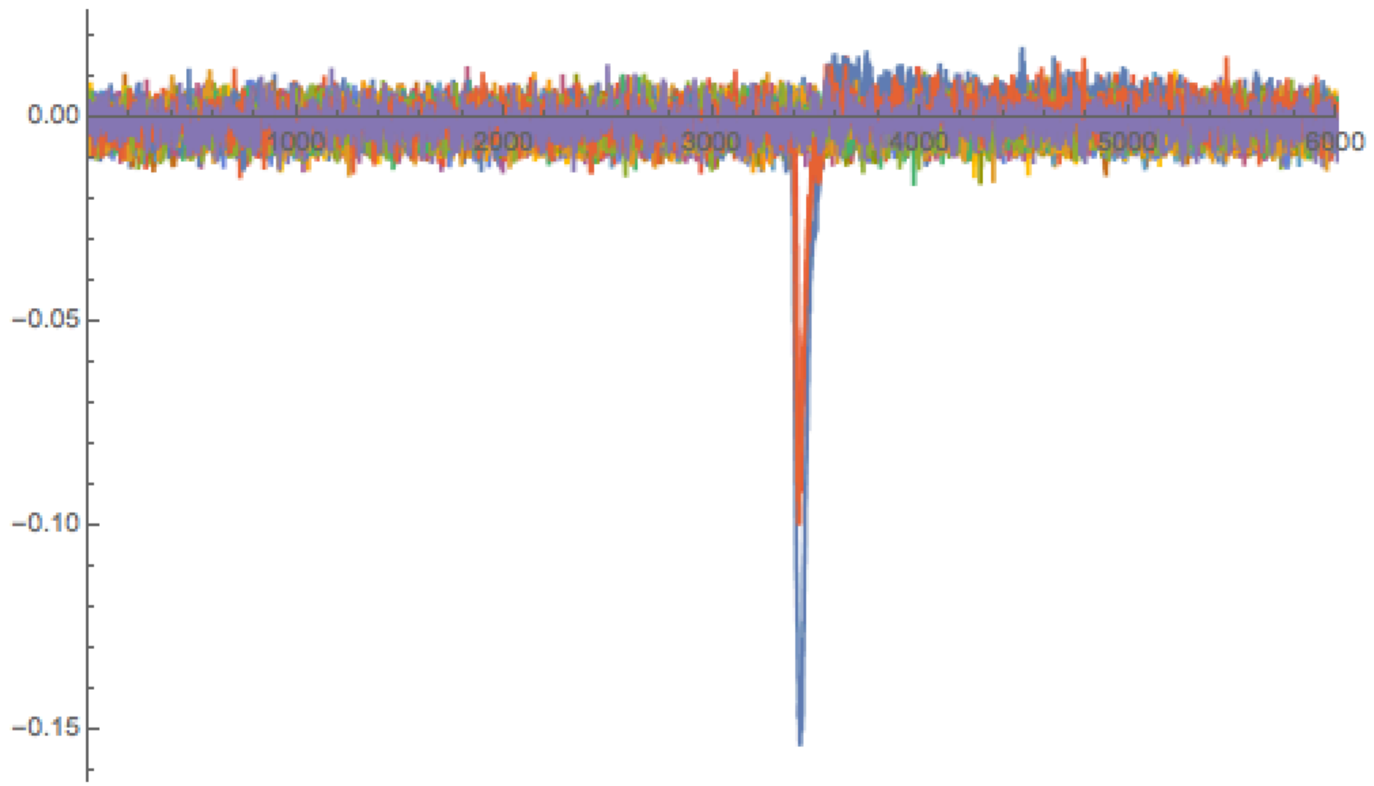
```
root [0] TString inte = (TString)gSystem -> GetFromPipe("wolframscript -code 'Integrate[Sin[2x],{x,0,4}]'")
(TString &) "Sin[4]^2"[8]
root [1] TString inteN = (TString)gSystem -> GetFromPipe("wolframscript -code 'N[Integrate[Sin[2x],{x,0,4}]]'")
(TString &) "0.5727500169043067"[18]
```

```
root [0] TString inteN = (TString)gSystem -> GetFromPipe("wolframscript -code 'N[Integrate[Sin[2x],{x,0,4}]]'")
(TString &) "0.5727500169043067"[18]
root [1] Double_t d_inteN = inteN.Atof()
(double) 0.572750
root [2]
```

Conversion of Lecroy Binaries in Mathematica

- New plugin was written to convert binaries from Lecroy Oscilloscopes within Mathematica

- No need for intermediate step (C++, python, matlab)



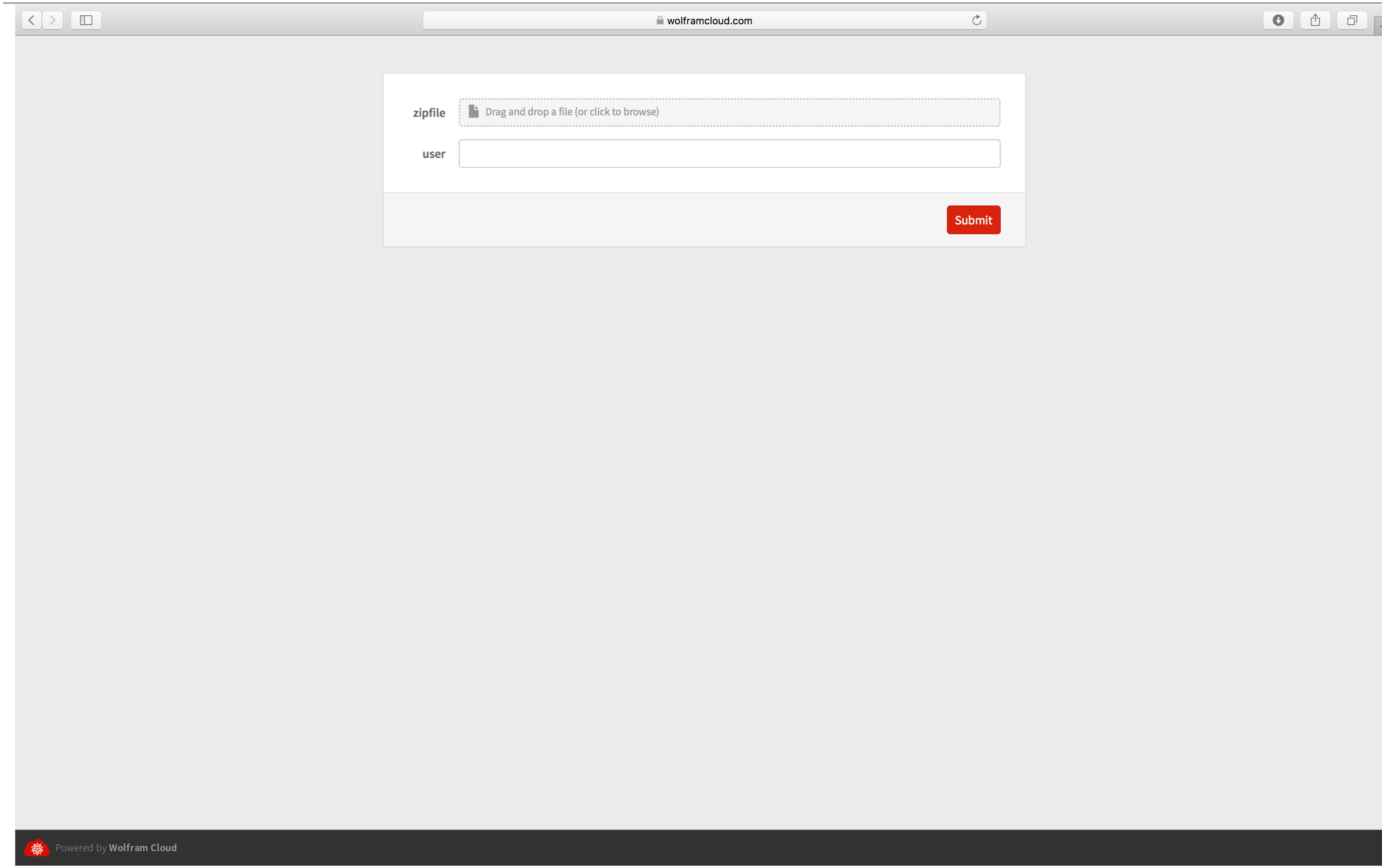
Reading Binary files

```
ReadFile[Infile_] := Module[{file = Infile},
str = OpenRead[file, BinaryFormat → True];
readHeader[str_] := Module[{struc},
    struc = {
      {"descriptorName", "UnsignedInteger8", 16},
      {"templateName", "UnsignedInteger8", 16},
      {"commType1", "Integer8", 1},
      {"commType2", "Integer8", 1},
      {"commOrder1", "Integer8", 1},
      {"commOrder2", "Integer8", 1},
      {"waveDescriptor", "Integer32", 1},
      {"userText", "Integer32", 1},
      {"regDesc1", "Integer32", 1},
      {"trigtimeArray", "Integer32", 1},
      {"risTimeArray", "Integer32", 1},
      {"resArray1", "Integer32", 1},
      {"waveArray1", "Integer32", 1},
      {"waveArray2", "Integer32", 1},
      {"resArray2", "Integer32", 1},
      {"resArray3", "Integer32", 1},
      {"instrumentName", "UnsignedInteger8", 16},
      {"instrumentNumber", "Integer32", 1},
      {"traceLabel", "UnsignedInteger8", 16},
      {"reserved1", "Integer16", 1},
      {"reserved2", "Integer16", 1},
      {"waveArrayCount", "Integer32", 1},
      {"pointsPerScreen", "Integer32", 1},
      {"firstValidPoint", "Integer32", 1},
      {"lastValidPoint", "Integer32", 1},
      {"firstPoint", "Integer32", 1},
      {"sparsingFactor", "Integer32", 1},
      {"segmentIndex", "Integer32", 1},
      {"subArrayCount", "Integer32", 1},
      {"sweepsPerAcq", "Integer32", 1},
      {"pointsPerPair", "Integer16", 1},
      {"pairOffset", "Integer16", 1},
      {"verticalGain", "Real32", 1},
      {"verticalOffset", "Real32", 1},
      {"maxValue", "Real32", 1},
      {"minValue", "Real32", 1},
      {"nominalBits", "Integer16", 1},
      {"nominalSubArrayCount", "Integer16", 1},
      {"horizontalInterval", "Real32", 1},
      {"horizontalOffset", "Real64", 1},
      {"pixelOffset", "Real64", 1},
      {"verticalUnit", "UnsignedInteger8", 48},
      {"horizontalUnit", "UnsignedInteger8", 48},
      {"horizontalUncertainty", "Real32", 1},
      {"triggerTimeSec", "Real64", 1},
      {"triggerTimeMin", "Integer8", 1},
      {"triggerTimeHour", "Integer8", 1},
      {"triggerTimeDay", "Integer8", 1},
      {"triggerTimeMonth", "Integer8", 1},
      {"triggerTimeYear", "Integer16", 1},
      {"triggerTimeR", "Integer16", 1},
      {"aqcDuration", "Real32", 1},
      {"recordType", "Integer16", 1},
      {"processingDone", "Integer16", 1},
      {"reserved5", "Integer16", 1},
      {"risSweeps", "Integer16", 1},
      {"timebase", "Integer16", 1},
      {"vertCoupling", "Integer16", 1},
      {"probeAtt", "Real32", 1},
      {"fixedVertGain", "Integer16", 1},
      {"bandwidthLimit", "Integer16", 1},
      {"verticalVernier", "Real32", 1},
      {"aqcVertOffset", "Real32", 1},
      {"waveSource", "Integer16", 1}
     };
    Map[<|First@# → BinaryReadList[str, Sequence @@ Rest[#]]|> &, struc, 1]];
rewind[str_, n_] := SetStreamPosition[str, StreamPosition[str] - n];
$HistoryLength = 0;
SetStreamPosition[str, 0];
  While[! StringMatchQ[FromCharacterCode@BinaryReadList[str, "UnsignedInteger8", 8], "WAVEDESC"], rewind[str, 7]];
  rewind[str, 8];
  header = Map[If[Length[#] > 1, FromCharacterCode[#], First@#] &, Join @@ readHeader[str]];
  rewind[str, -header@"userText"];
  trigTimeArray = Partition[BinaryReadList[str, "Real64", 2 header@"subArrayCount", ByteOrdering → -1], 2];
  rewind[str, -header@"risTimeArray"];
  waveArray = BinaryReadList[str, "Integer16", header@"waveArrayCount", ByteOrdering → -1];
  waves = MapIndexed[{(#2[[2]] - 1) * header@"horizontalInterval" + trigTimeArray[[#2[[1]], 2]], #1 * header@"verticalGain" - hea
```

Cloud Deploy

- General Idea:
  - Having a simple tool in the browser to perform quick data check
  - Idea came up during PICOSEC test beam (RD51 project, more details here) to check quality of time series data
- Principle:
  - Upload data in zip format to web page
  - After computation, getting mail alert
  - Picking up the results from an specific URL
- Realised in great collaboration with Jesus Hernandez from Wolfram

Cloud Deploy



Cloud Deploy

1. Drag and drop .zip file (or browse)

2. Enter Mail

3. Press “Submit”

4. Pick up results

Manuel Guth

Inbox -

Job is complete

To: Manuel Guth

Pick up results here:
https://www.wolframcloud.com/objects/user-8e9eeba4-65a4-435d-9ef4-328cdc4469e3/Results/Manuel.guth.zip

Data Format similar to hdf5/ pandas

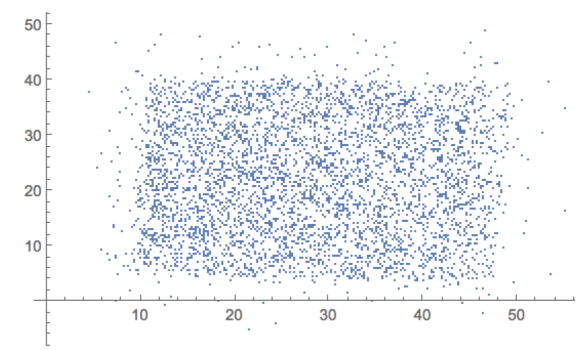
- Data format similar as in python (e.g. pandas)
- Quick displaying tools with cuts etc. (DensityHistogram, Histogram3D, SmoothDensityHistogram, Query[Histogram, “Y”], …)
- Easy and fast handling (no need to mess with indices)

```
runs970ds = Module[{raw},
  raw = Import["C:\\Users\\jhernandez\\ownCloud\\Data\\CERN\\970.txt", "Table"];
  Dataset[AssociationThread[First[raw] → #] & /@ Rest[raw]]]
```

SRSCount	Tracks	TrackNr	X	Y
3	1	0	22.0035	23.6859
4	1	0	23.738	11.68
5	1	0	18.0315	30.0048

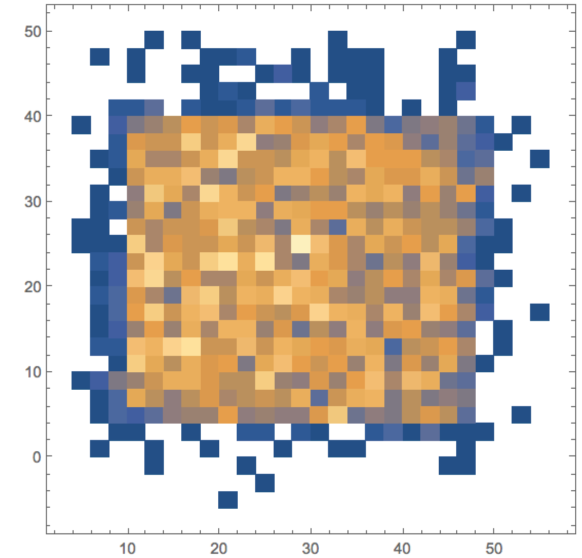
```
eventsFromCuts = Flatten[allwaves[[All, 1, "waveform", All, "srsEventNum"]]];

runs970ds[Select[MemberQ[eventsFromCuts, #SRSCount] &], {"X", "Y"}] // ListPlot
```



Another few ways to look at the data

```
runs970ds[Select[MemberQ[eventsFromCuts, #SRSCount] &], {"X", "Y"}] // DensityHistogram
```



Data Format similar to hdf5/ pandas

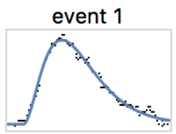
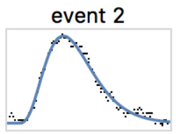
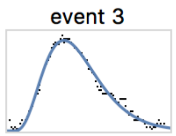
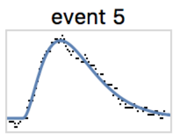
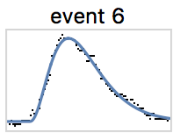
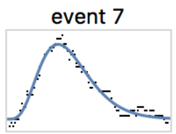
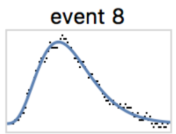
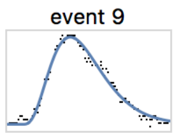
- Performed fits can be directly stored in well accessible table format
- Also after cuts, e.g. event number stays associated
- Fast and simple accessible
- Sample data/fit from HyperFast Silicon (HFS) data within 2017 PICOSEC

Map function across waves

Here I use MapIndexed (this allows me to use the position as an argument). Dataset groups the results together.

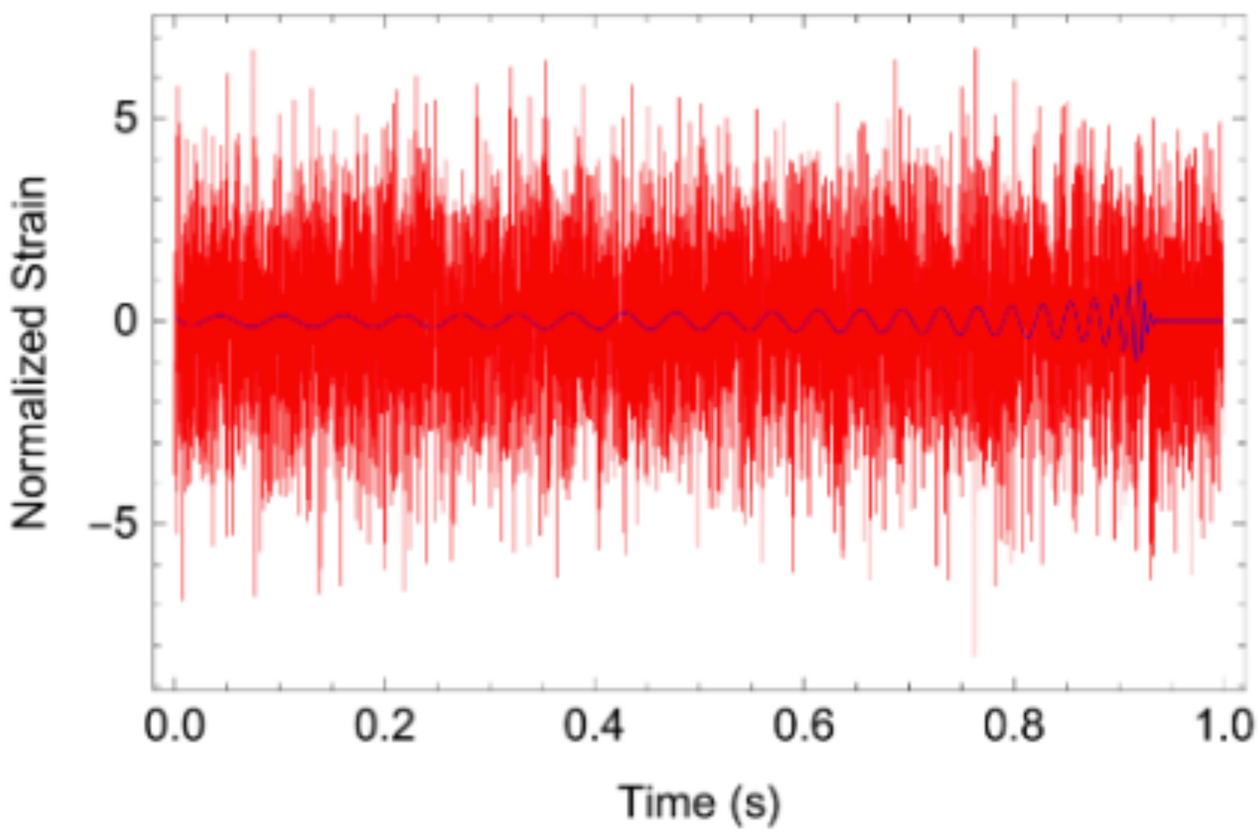
```
ds = Dataset[MapIndexed[fit[#1, #2⟦1⟧] &, wave4⟦1 ;; 100⟧]]
```

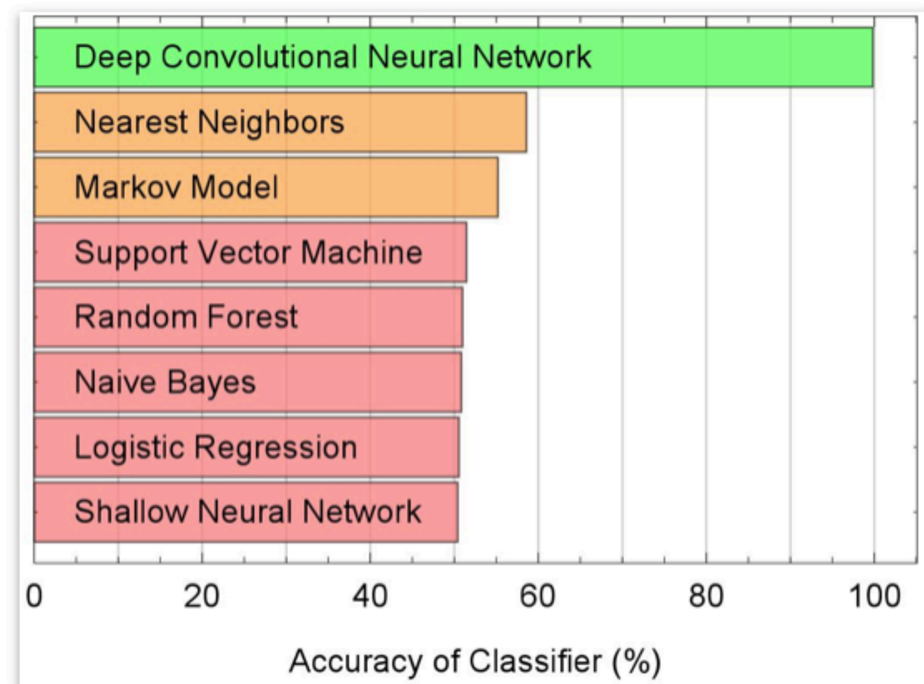
NonlinearModelFit: The step size in the search has become less than the tolerance prescribed by the PrecisionGoal option, but the

event	bestFitParameters	adjustedRSquared	plot
1	{A → 0.119864, n → 2.11306, to → 0.592996, toff → 6.41963}	0.994857	
2	{A → 0.0962981, n → 3.7208, to → 0.401652, toff → 11.3142}	0.992228	
3	{A → 0.11766, n → 3.70992, to → 0.454327, toff → 4.29665}	0.994448	
4	NonlinearModelFit::sszero	—	
5	{A → 0.0926168, n → 2.05265, to → 0.595536, toff → 7.40185}	0.991077	
6	{A → 0.11257, n → 2.50197, to → 0.506459, toff → 17.7226}	0.9939	
7	{A → 0.0667517, n → 4.39367, to → 0.377799, toff → 27.448}	0.986334	
8	{A → 0.0815095, n → 4.40926, to → 0.407061, toff → 25.6584}	0.993418	
9	{A → 0.0902037, n → 3.41859, to → 0.454174, toff → 21.8924}	0.992225	

Outreach - LIGO Real Time Signal Detection

- LIGO collaboration used deep neural network in Mathematica for real time signal detection and parameter estimation





- https://gravity.ncsa.illinois.edu/research/deep-learning/real-time-detection-and-parameter-estimation/
- Paper manuscript ready — "Deep Filtering: A Deep Neural Network Framework for Real-time Multimessenger Astrophysics"
  - ➡ will be published soon

Summary

- Interaction of theorists with experimental side?
  - ✓ Mathematica importer for ROOT files
- How to make use of analytical Mathematica tools within ROOT?
  - ✓ Calling Mathematica from ROOT
- Is there a quick tool to check data quality e.g. at test beam?
  - ✓ Wolfram cloud deploy
- What about direct analysis of big data sets from experiment?
  - ✓ Wolframscript (on lxplus?) or cloud computing

Backup

Outreach - LIGO Real Time Signal Detection