RapidFit
A beginners Guide, Talk 2 of ?

Robert Currie, Edinburgh University

Introduction

- Tutorials
- More on XML Options *(A LOT more)*
- RapidFit C++ Objects you see in your PDF
- Q & A
- Plan of next talk

Introduction cont

Disclaimer!

I do **NOT** intend everyone to follow everything in this tutorial.
I do **NOT** intend to try and stop and explain every little sordid detail about the inner workings of what is not important.
This document **IS** trying to be definitive.

This 'talk' is dry and technical. **Please** shout if you're falling asleep so I can speed things up.

I **will** answer questions and intend to give people a view of what can can't be done in the XML.

I will **NOT** cover constructors/operators/object scope/passing by reference/memory management/polymorphism/inheritance/templates/interfaces/any other C++ idioms.

Introduction cont

I asked for feedback and have so far only been asked to address 2 things:
Blinding and Running on Batch Systems.

I can and will be covering blinding in this talk but will not be covering running on Batch Systems till next week. (Technical reasons combined with the fact that it's a wide topic)

If I don't get other questions I will spend next week on Runtime arguments and batch systems.

The Week After I will discuss writing your own study in RapidFit / Modifying the Toy Study. As well as how to easily intercept the Input/Output at the various stages of RapidFit.

Tutorials and SVN commits

The tutorial XMLs and pdfs are in HEAD, checked, working and verified as of 15/10/2012.

I won't give everyone the long turgid details but the examples I wrote last week didn't 'just work'.

But: RapidFit now works extremely well 'Out of the Box'.

I have committed and tested:

- SimpleGauss, SimpleGauss2D, SimpleGauss3D
- SimpleDoubleGauss, OptimisedDoubleGauss

I will run an example fit or 2 at the end to show these off.

Tutorials

I have added XML into a new directory '/path/to/RapidFit/tutorial'.

I've added some sensibly named XML which demonstrate the basics.

I encourage everyone to perform 'svn up' and run on the latest code. There are a LOT of cumulative bugfixes/improvements and such in the codebase.

I am going to cover a lot of 'other Options' in the XML and will add some basic XMLs showing how to use each of these.

Tutorials cont

I have added new XML and talks to svn in the new tutorials folder.

Specifically I recommend looking at:

SimpleGauss PDF.
(Simplest PDF I ever plan to write)

SimpleDoubleGauss PDF.
A simple way to implement Components in your PDF without the code associated with everything else everyone normally does.

OptimisedDoubleGauss PDF.
An Optimised form of the DoubleGauss PDF which cuts >50% of the CPU out from fitting.

I cannot go through everything that everyone will ever need to do. But, OptimisedDoubleGauss is well worth reading and understanding and comparing to SimpleDoubleGauss if you intend to use more complex features in your PDF.

Miscellaneous Stuff...

RapidFit uses it's own XML parser.
This allows us to do things such as comment out a line of XML by making the first non-whitespace character '#'.
This is NOT standards compliant, but is fast and convenient compared to wrapping code inside <!– –!> which is a pain when you want to comment out the 1 line in many places in the same file. (It happens more often than you think!)

RapidFit is doing more and more for the user as time goes on. It doesn't remove the need to you to think for yourself. But you no longer have to worry about strange quirks or not being able to do certain things in the code.

Miscellaneous cont

<Seed>
This technically falls under miscellaneous because I don't know where else to put it.
RapidFit makes use of the TRandom3 object wherever we require random numbers. This means we have chosen a good random sampling and that we can choose the start seed of our random number chains.
I have modified RapidFit to use a consistent set of Random number generators throughout the codebase. This means that you can select a random number in the XML, and, provided nothing changes that can effect the random number distributions you can get exactly the same results back out.

<Repeats>
This XML tag allows us to control the size of a Toy Study within the XML.

XML

I plan to go through a lof of the 'hidden'/'undocumented' features in the XML that people use day to day in fitting. There are probably some half dozen I have forgotten about or don't care for.

If you want to know what XML options you can use, read XMLConfigurationReader.cpp

If you want to know what Runtime arguments you can use, read ParseCommandLine.cpp

I cannot sit down and explain everything in these files, and in some exotic cases I can't tell you what the code does because I simply don't know.

<ParameterSet>

A <ParameterSet> is composed of <PhysicsParameter>-s. Nothing else.

A <PhysicsParameter> MUST have a <Name> and a <Value> tags.
This is where a Parameter starts from in the Fit and MUST be defined!

```
<ParameterSet>
  <PhysicsParameter>
    <Name>sigma</Name>
    <Value>2.</Value>
  </PhysicsParameter>
  <PhysicsParameter>
    <Name>sigma2</Name>
    <Minimum>0.</Minimum>
    <Maximum>20.</Maximum>
    <Type>Free</Type>
    <Value>5.</Value>
    <BlindString>FooBar</BlindString>
    <BlindScale>1.</BlindScale>
    <StepSize>0.01</StepSize>
  </PhysicsParameter>
</ParameterSet>
```

<ParameterSet> cont

<PhysicsParameter> may have <Maximum> & <Minimum> tags.
These represent physical ranges of Parameters. i.e. $0. < \text{fraction} < 1$.
This may alternately be a way of constraining an unstable fit.
Eg: $-2\pi < \delta_i < 2\pi$.

<PhysicsParameter> may also have a <StepSize> tag.
This is the first guess at the Parameter's Error.

<Type> This allows you to fix/free Parameters in the fit. Fixed Parameters are not exposed to Minuit but are ALWAYS passed to your PDF.
In order to fix a parameter the value 'Fixed' must be used.

By default a <PhysicsParameter> has a Name a starting value, the assumed error of 0.1 and the range of $\pm$inf in the fit.

At the moment options passed to these sections of the XML are evaluated as raw numbers/string as appropriate. I would like someone to add the ability to accept TF1-s as the arguments to numerical numbers to allow the use of constants and such.

<ParameterSet> cont + Blinding

<PhysicsParameter> may also include <BlindString> and <BlindScale> tags. These tags are important as they control blinding in your fits.

The value of the Blinding Offset in PhysicsParameter is *internally* calculated in RapidFit and **NOT** exposed to the user!

PhysicsParameter objects in C++ have 2 sets of Functions: GetBlindedValue/SetBlindedValue and GetValue/SetValue.

Your PDF is exposed to the ***True*** Value of the PhysicsParameter through GetValue/GetTrueValue.

ALWAYS use GetBlindedValue for your PDF debugging Output!

Minuit and **EVERYTHING ELSE** you see in the Output to screen & file in RapidFit is hidden behind the GetBlindedValue method.
More on this later under the C++ section.

<Minimiser>

The simplest Minimiser is <Minimiser>Minuit</Minimiser>. This hides a lot of things from the user. 90% of the time you don't want/need more than this.

```
<Minimiser>
  <MinimiserName>Minuit</MinimiserName>
  <MaxSteps>100000</MaxSteps>
  <GradTolerance>0.001</GradTolerance>
  <Quality>1</Quality>
  <ConfigureMinimiser>HesseFirst</ConfigureMinimiser>
</Minimiser>
```

<MaxSteps> is the Maximum number of calls Minuit can make before it gives up.

<GradTolerance> is 1000*(DistanceToMinimim) of whole LL function.

<Quality> is how much Minuit should care about the error matrix on 1st pass.

0: it doesn't, 1: it does a bit, 2: error matrix must be correct before Hesse.

<ConfigureMinimiser> special arguments passed to the Minimiser class.

Specialist Functions, only need to know/care once you've done a LOT of work.

<Minimiser> cont

The only officially supported and sanction Minimiser is Minuit. Minuit2 and Fumili are available but mileage will vary.

Minuit also is the only Minimiser which fully supports <ConfigureMinimiser>.

The Available Options are:

- '*HesseFirst*' This runs Hesse first before Migrad
- '*SeekFirst*' This runs Seek first before Migrad
- '*SimplexFirst*' This runs Simplex first before Migrad
- '*NoHesse*' This Doesn't run Hesse after finishing Migrad
- '*MinosErrors*' This runs Minos after running Hesse
- '*RooFitErrors*' Use Correlation Matricies to correct parameter errors for non unitary event weighting

<FitFunction>

The simplest Minimiser is <FitFunction>NegativeLogLikelihood</FitFunction>.
This hides a lot of things from the user.
99% of the time you don't want/need more than this.

The <FitFunction> tag is quite powerful so I will spend 2/3 slides discussing this.

A more complete example of the options available through the <FitFunction> tag are:

```
<FitFunction>
  <FunctionName>NegativeLogLikelihoodThreaded</FuntionName>
  <Threads>4</Threads>
  <UseGSLNumericalIntegration>True</UseGSLNumericalIntegration>
  <WeightName>sWeight</WeightName>
  <NormaliseWeights>True</NormaliseWeights>
  <Trace>FileName.root</Trace>
  <Strategy>SomeStrategy</Strategy>
  <SetIntegratorTest>False</SetIntegratorTest>
</FitFunction>
```

<FunctionName> is either **NegativeLogLikelihood**, **NegativeLogLikelihoodThreaded**, **NegativeLogLikelihoodNumerical** or whatever you code up.
NegativeLogLikelihood is the normal class, Threaded is faster and makes use of multi-threading across multiple cores, Numerical is even more optimal for analyses relying on Numerical Normalisation.

<Threads> is number of simultaneous threads to use when fitting.

<FitFunction> cont

<UseGSLNumericalIntegration> is a new flag that has just been introduced but allows you to try using a GSL function for numerical integration.
(I have had some success with this speeding up Projections)

<WeightName> This is the XML tag that turns your fit into an sFit.
The argument passed to this XMLTag is the name of the Observable you wish to use as the per-event weight in your fit.
NB: This Observable MUST be in your PhaseSpaceBoundary!

<NormaliseWeights> This flag allows you to use RapidFit to apply a statistical correction to your DataSet in order to get the correct Errors from your Fit.
This is correct to within 90% and is the easiest way to apply this correction factor for an nTuple.
BUT BE CAREFUL, THIS DOES NOT MAGICALLY SOLVE ALL YOUR WOES!

<FitFunction> cont

<Trace> This is a Special Tag.
This allows you to 'trace' everything that happened within the fit.
This takes the argument of a **fileName.root** and this is the file where the result of every single Minuit call is saved.
This is extremely useful for satis-
fying your curiosity or for performing diagnostics to watch Minuit diverge or mis-behave.

<SetIntegratorTest> This allows you to turn off the Integrator test at the start of RapidFit which compares the Numerical and Analytical Integrals.
Numerical Integration uses the PDF::Evaluate function and Analytical is PDF::Normalise so it's a self consistency test of the PDF.

<Strategy> This allows you to employ a 'Fit Strategy' to fit a dataset or to perform an LL scan. This is another specialist feature but allows you to overcome the limitations of Minuit to find the correct minima.

<ToFit>

This is the most complex part of the XML and will take a while to fully cover all the available options.

```
<ToFit>
  <PDF>
      <Name>SimpleGauss</Name>
  </PDF>
  <DataSet>
    <Source>Foam</Source>
    <NumberEntries>1000</NumberEntries>
    <PhaseSpaceBoundary>
      <Observable>
        <Name>x</Name>
        <Minimum>-10.</Minimum>
        <Maximum>10.</Maximum>
        <Unit>someUnit</Unit>
      </Observable>
    </PhaseSpaceBoundary>
  </DataSet>
</ToFit>
<ToFit>
  <ConstraintFunction>
    <ExternalConstraint>
      <Name>deltaM</Name>
      <Value>17.63</Value>
      <Error>0.11</Error>
    </ExternalConstraint>
  </ConstraintFunction>
</ToFit>
```

<ToFit> cont

<PDF> This Section of the ToFit MUST contain at least a <Name> tag.
<CommonPDF> May also be used here.
This Section May also contain:
<ParameterSubstitution> and/or <AppendParameterNames> *(same effect)*
as well as <ConfigurationParameter>.
This allows you to configure the PDF at runtime to have *(subtley)* different behaviour for different XMLs.

<ConstraintFunction> This section allows you to use an ExternalConstraint which is effectively constraining free PhysicsParameter(s) in the fit.
You will likely know of this only when you need to.

<DataSet> This section Contains everything you will need to select your dataset and is a complex topic to cover.

<ToFit> cont <DataSet>

<DataSet> MUST contain:
<Source>
This is either File/Foam/FoamFile
<NumberEntries>
This is the (Maximum)Number of Events to Read in
<PhaseSpaceBoundary>
This is the possible PhaseSpaceBoundary for all Data.

When the <Source> is 'File' the XML tag <FileName> should be added.
RapidFit by default will pick up the first nTuple in the file it finds.
The XML tag <NTuplePath> can also be added which allows you to pick a specific nTuple within a file when there is more than 1.

When the <Source> is 'Foam' the PDF::Evaluate function is used to generate a dataset.
When the <Source> is 'FoamFile' is used. Foam is used to produce an nTuple and the nTuple is saved and read back in from disk. (reason why is probably not important to you.)

<ToFit> cont <PhaseSpaceBoundary>

<PhaseSpaceBoundary> defines the contents of the DataSet in memory and the region the data can populate.

The PhaseSpaceBoundary is populated by <Observable> objects.

<Observable> tags MUST contain a <Name> and <Unit> tag.

The <Observable> tag must contain either a Continuous or Discrete Range that the Data can populate.
A Continuous Constraint is defined with <Maximum> & <Minimum> tags.
A Discrete Constraint is defined as a set of <Value> tags.

In RapidFit a Discrete and Continuous Observable are different. A PDF can be addressed with any value within a continuous Constraint that can vary by any amount whilst it has one Discrete value at a time.

<ToFit> cont <PhaseSpaceBoundary> cont

<Observable> allow the user to read in data from an nTuple.

RapidFit can cope with nTuples having different structures really easily.

When reading in an Observable from an nTuple each Observable in RapidFit contains the result of evaluating a <TF1> tag in the Observable structure.
If none is provided then RapidFit evaluates the **Name** of the Observble as the object to read in.

The use of <TF1> tags allows for RapidFit to read in and perform a complex transform 'on the fly' which means that the PDF can be coded up in such a way that it doesn't depend *too strongly* on the contents of the nTuple.

<ToFit> cont

Many <ToFit> segments can be used in parallel. The more complex the analysis the more <ToFit> segments you're likely going to need.
To try and combat this increased complexity I recently introduced <CommonPDF> and <CommonPhaseSpace> which can be used as such:

```
<CommonPDF>
  <PDF>
    <Name>SomePDF</Name>
  </PDF>
</CommonPDF>
<CommonPhaseSpace>
  ...
</CommonPhaseSpace>
<ToFit>
  <CommonPDF>True</CommonPDF>
  <DataSet>
    <Source>File</Source>
    <FileName>File1.root</FileName>
    <CommonPhaseSpace>True</CommonPhaseSpace>
  </DataSet>
</ToFit>
<ToFit>
  <CommonPDF>True</CommonPDF>
  <DataSet>
    <Source>File</Source>
    <FileName>File2.root</FileName>
    <CommonPhaseSpace>True</CommonPhaseSpace>
  </DataSet>
</ToFit>
```

<Output>

There is a lot of different output that can be extracted from the RapidFit tool. This is generally broken down into Parameter scans or Observable Projections.

```
<Output>
  <Projection>x</Projection>
  <ComponentProjection>y</ComponentProjection>
  <Scan>
    <Name>sigma</Name>
    <Sigma>3</Sigma>
    <Points>100</Points>
  </Scan>
  <TwoDScan>
    <X_Param>
      <Name>sigma</Name>
      <Sigma>3</Sigma>
      <Points>30</Points>
    </X_Param>
    <Y_Param>
      <Name>sigma2</Name>
      <Minium>3.</Minimum>
      <Maximum>5.<Maximum>
      <Points>30</Points>
    </Y_Param>
  </TwoDScan>
</Output>
```

<ComponentProjection>

For complex full analyses RapidFit can perform projections but they will likely take a long time to run.
For simpler analyses the projections don't take too long and are highly configurable.

```
<Output>
  <ComponentProjection>
    <Name>mass</Name>
    <DataBins>256</DataBins>
    <PDFpoints>1024</PDFpoints>
    <LogY>True</LogY>
    <ColorKey>0:2:3:4:6:7:8:9:11</ColorKey>
    <StyleKey>1:2:3:4:5:6:7:8:9</StyleKey>
    <WidthKey>0:3:3:3:3:3:3:3:3</WidthKey>
    <YTitle>Events</YTitle>
    <XTitle>Mass</XTitle>
    <XMin>5000</XMin>
    <XMax>6000</XMax>
    <YMin>100</YMin>
    <YMax>20000</YMax>
    <DrawPull>True</DrawPull>
    <CalcChi2>True</CalcChi2>
  </ComponentProjection>
</Output>
```

<ComponentProjection> cont

We always need to know the <Name> of what is being projected.

The Number of Bins in the histogram for Data <DataBins> can be varied.
The Number of Points PDF is Numerically Integrated at <PDFpoints> can be varied.
Setting Log on the Y axis can be turned on/off <LogY>.

The ROOT Colour, Style and Line Width can be set according to he keys.
(0 width means a line is not actually drawn but is saved to the .root file)

The Ranges of the X and Y axis can be altered.

The Total χ^2/NDOF can be calculated from your PDF and data in a projection <CalcChi2>.

The Residuals (pull per-bin) can be calculated and drawn <DrawPull>. (Alongside Projection and Separately).

Projections used to be something RapidFit was extremely poor at. The tool does not yet do as much hand-holding as some other tools (RooFit) already out there do, but it's improving in leaps and bounds.

<PreCalculator>

This is the section of RapidFit that allows us to calculate per-event weights and store then in an nTuple.

```
<Precalculator>
  <Name>SWeightPrecalculator</Name>
  <WeightName>sWeight</WeightName>
  <OutputFile>1fbUB-6binSig.root</OutputFile>
  <Config>1</Config>
  <UseAlpha>True</UseAlpha>
</Precalculator>
```

The <PreCalculator> section of the XML contains a lot of Info:
<Name> Name of the Weighting tool to use.
<WeightName> The Name of the new branch you wish to save the weight as in the new nTuple.
<OutputFile> The Name of the new Weighted nTuple that is to be produced.
<Config> This changes whether to use the first or second PDF in your Sum/NormalisedSum PDF as the signal component.

<UseAlpha> This applies the statistical correction factor directly to the weights as they're written to disk.
This is important if you plan to get the errors consistent and correct, but is NOT recommended if you plan to read yields from the nTuple directly.

C++ Objects in RapidFit

In RapidFit you need to be aware of at MOST about 12 classes from the perspective of doing physics and getting something working.
The most obvious are:

- DataPoint
- ObservableRef
- ComponentRef
- YourPDF
- PhaseSpaceBoundary
- IConstraint
- Observable
- ParameterSet
- PhysicsParameter

DataPoint and Observable

Each DataPoint object in C++ in RapidFit corresponds to an event on the nTuple which has passed all of the cuts and been read into memory.

A DataPoint exists as the object which manages the Observables for each Event.

Each Observable can be Requested from the DataPoint with DataPoint::GetObservable(string)

Each DataPoint may know what PhaseSpaceBoundary it lies in.

Each Observable contains a Value and a Unit and these may be retrieved with Observable::GetValue() and Observable::GetUnit() accordingly.

ParameterSet and PhysicsParameter

These objects correspond to the objects in your XML on a 1-to-1 comparison.

The ParameterSet exists as an object which Manages the PhysicsParameters for a given PDF.

When using a PhysicsParameter in your PDF to do calculations always request the PhysicsParameter to return it's true value.

This is done with PhysicsParameter::GetValue() or PhysicsParameter::GetTrueValue(). (These are identical, but GetTrueValue is more explicit!)

WARNING: IF YOU PRINT A VALUE TO SCREEN FROM YOUR PDF MAKE SURE IT'S FROM:
PhysicsParameter::GetBlindedValue()

Note: We use the RooFit blinding numbers to be compatible for cross-checks.

ObservableRef

All PDFs now use ObservableRef objects to store Observable/PhysicsParameter names.
(The name of this class is historically unfortunate)

Each ObservableRef object contains knowledge of the string inside itself and where that string is in a lookup table.

Keeping this information in the same place means that I don't have to repeat lookup operations.
(These are slow and inefficient in terms of CPU)

These are used for:
ParameterSet::GetPhysicsParameter(ObservableRef)
DataPoint::GetObservable(ObservableRef)

PhaseSpaceBoundary

A PhaseSpaceBoundary in RapidFit is the class which Manages the Constraints (IConstraint) placed on each Observable in RapidFit.

In RapidFit there are 2 types of constraints, Continuous and Discrete.

Discrete constraints return true from IConstraint::IsDiscrete(). Discrete Constraints contain a list of possible values which can be requested by: IConstraint::GetValues()

Continuous Constraints contain a range of continuous values. The maximum and minimum can be requested by: IConstraint::GetMaximum() and IConstraint::GetMinimum()

C++ Component Plotting

Your PDF can produce Component Plots if you set it up to.

The Simplest example is shown in the SimpleDoubleGauss PDF.

This PDF adds the functions:
SimpleDoubleGauss::PDFComponents() and
SimpleDoubleGauss::EvaluateComponent(DataPoint*, ComponentRef*).
The ComponentRef class has the ability to store the Component Name and Reference Number in the one place to reduce the cost of lookups in your PDF (which are slow and stupid).

I've shown an example in SimpleDoubleGauss how to implement ComponentPlotting in a way that it runs very quickly.

Although I recommend understanding OptimisedDoubleGauss as it implements ComponentPlotting in a way that is much better.

Conclusion

We have Covered **A LOT** of information in this talk.

This is me giving people a guided
tour of the kingdom and then letting you go home and study the map.

I don't expect you to know what everything is or be able to use it all but it's available should you need/want to use it.

Apologies for the lack of running examples, but there are some new XML and highly documented pdfs to try in HEAD.

Next Week

Next week I plan to give a talk on 2 things:

- Runtime Arguments for RapidFit
- Running on a batch system in RapidFit

I will be working on making XML/scripts/PDFs available on this and will encourage people's feedback and suggestions as appropriate.