AFit
an introduction

http://pprc.qmul.ac.uk/~bevan/afit/

Adrian Bevan

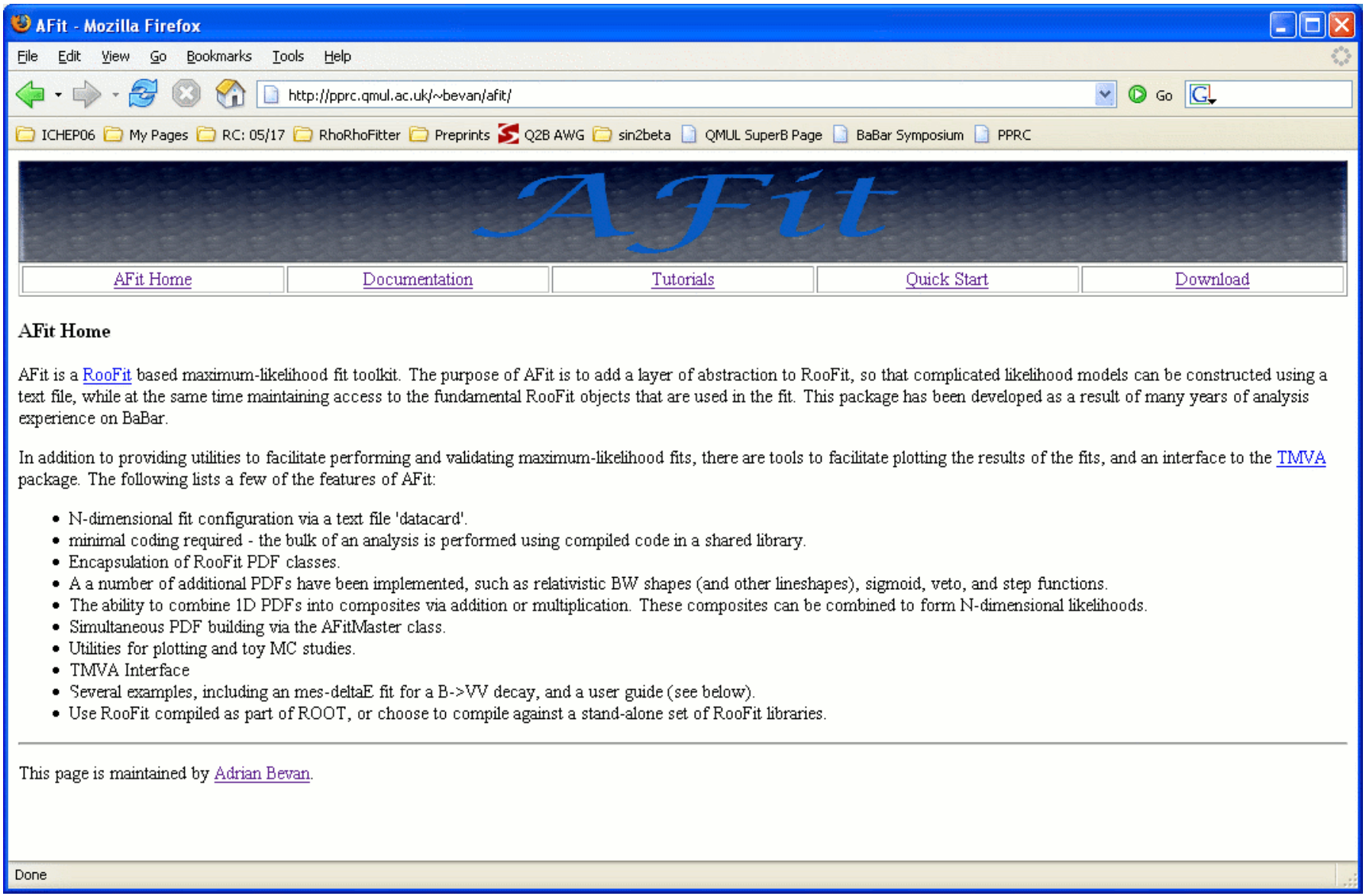
Overview

- What is *AFit*?
- AFit Home Page
- Making a PDF
  - PDF Builders
  - Factory classes: AFitPdfFactory, and AFitComponentFactory
  - The AFitMaster
- Where to find out more information

What is *AFit*?

- *AFit* is an interface to RooFit and TMVA
  - RooFit:
    - Simplify the process of defining and implementing the code required to construct a likelihood.
    - Provide the option to delegate the likelihood configuration completely to an ASCII file.
    - Provide utilities to simplify validation and interpretation of the likelihood.
      - Plotting, pure and embedded toy Monte Carlo simulations, correlations between discriminating variables etc.
  - TMVA:
    - Simplify the process of defining which classifiers to train and evaluate.
    - Provide an interface to append classifiers to your RooDataSet(s).

AFit Home Page



Making a PDF

- PDFs are accessed via a ‘PDF builder’.
  - This is derived from the AFitAbsPdfBuilder class.
- There are three ways to make a PDF in *AFit*:

1. Use an AFitAbsPdfBuilder derived object directly.
2. Use a Factory class to make the builder.
3. Use the AFitMaster.

PDF builders

- What is a builder?
  - Each AFit PDF is made using a PDF Builder.
  - Has an instance of all RooAbsReal and RooCategory types that define the shape of the PDF.
  - Contains a RooArgSet of these variables called `'varSet'`.
  - Makes a RooAbsPdf by calling the `getPdf()` function.

e.g. consider the AFitArgus builder as an example

```
// the discriminating variable x is the beam constrained B meson mass.
RooRealVar x(''x'', '''', 5.25, 5.29);

// The AUGUS PDF
AFitArgus argus(x, ''arguspdf'');
RooAbsPdf * argusPDF = argus.getPdf();
```

Discriminating variable

AFit pdf Builder

Unique name of PDF

RooAbsPdf object to use for fitting

PDF builders

- The AFitArgus builder has data members:
  - xi (ξ).
  - endpt (m_0).

- These are both objects of type RooRealVar.
- Have names derived from the PDF builder name. e.g. for a builder with name “arguspdf”:
  - The name of xi will be “arguspdfxi”.
  - The name of endpoint will be “arguspdfendpt”.
  - Parameter suffixes match the corresponding RooFit PDF variable names.

- The functional form of this PDF is

$$\mathcal{P}(m\,;m_0,\xi) = \frac{1}{N}\cdot m\sqrt{1-(m/m_0)^2}\cdot\exp(\xi\,(1-(m/m_0)^2))\cdot\theta(m<m_0)$$

$$\text{where } \theta(m<m_0)=1 \text{ and } \theta(m>m_0)=0.$$

- The parameters can be read from a configuration file specified by using the `AFitAbsPdfBuilder::setDataCard(const char *)` member function, before calling the `getPdf()` function to instantiate the RooAbsPdf.

PDF builders

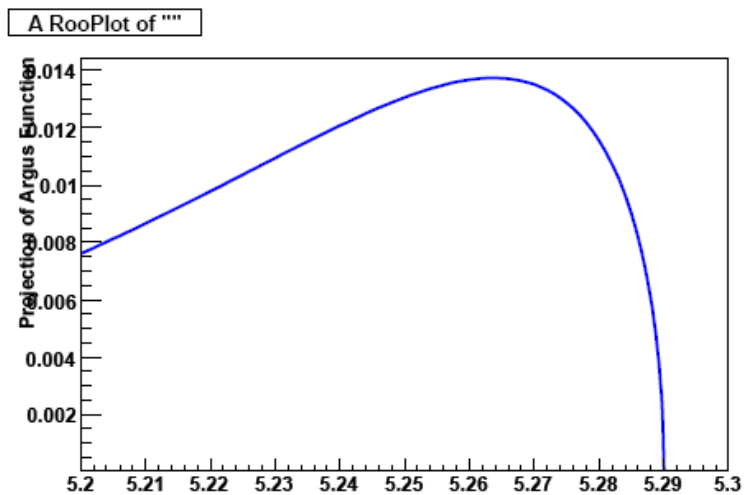
- The configuration file used to specify the values (and fit ranges) of ξ and m_0 looks like the following example:

```
[arguspdf]
arguspdfxi =  -20.0 L(-100 - 10)
arguspdfendpt =  5.29 +/- 0.001 C L(5.25 – 5.30)
```

- The `getPdf()` function returns an Argus PDF with the specified parameters. As this is a RooAbsPdf, it can be manipulated in the usual way.

- e.g. the PDF can be plotted using

```
RooPlot * frame = x.frame();
argusPDF->plotOn(frame);
frame->Draw();
```

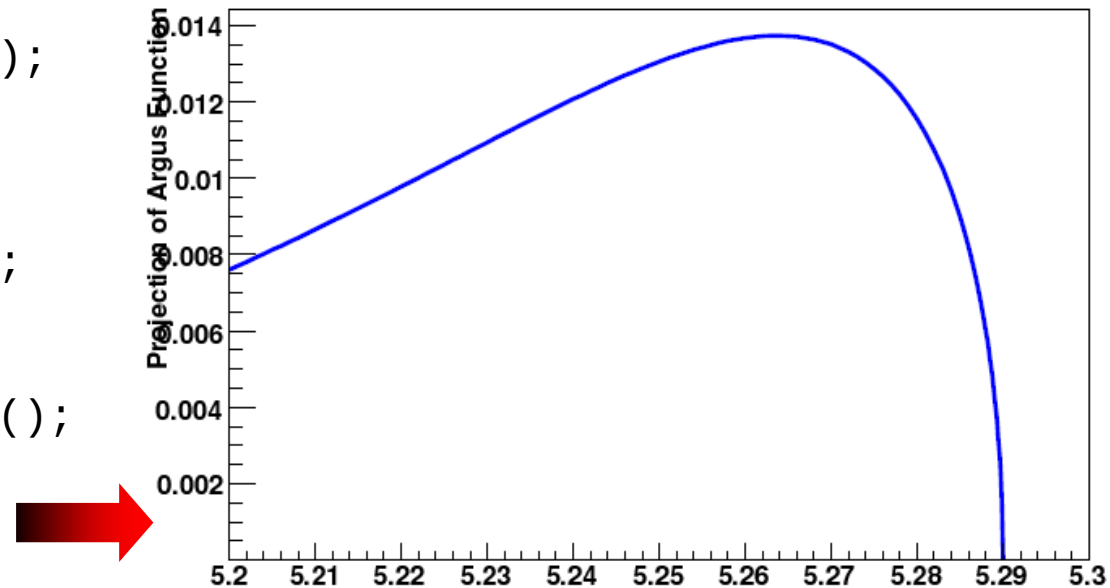


PDF builders

- You don't have to use a configuration file: each PDF builder has an instance of its variables.

- PDF parameters can be set by hand in a macro:

```
RooRealVar x("x", "M_{ES}", 5.25, 5.29);

AFitArgus argus(x, "arguspdf");

argus.endpt->setVal(5.289);
argus.xi->setVal(-21.0);

RooAbsPdf * argusPDF(0);
argusPDF = argus.getPdf();


RooPlot * frame = x.frame();
argusPDF->plotOn(frame);
frame.Draw();
```



PDF builders

- If you have several similar PDFs for different components, it can useful to ensure that these have some common parameters.
  - e.g. Argus endpoint for continuum and charmbackground.
  - Need to use the same variable for this.
  - Simple to implement this for any PDF:

```
[arguspdf]
arguspdfxi =  -20.0 L(-100 - 10)
arguspdfendpt =  5.29 +/- 0.001 C L(5.25 – 5.30)
argusvariablesToReplace = argusendpt:continuumendpt
```

  - Can also do this in a macro [before calling `getPdf()`]

```
argus2.variablesToReplace.setVal("oldVar:newVar");
```

  - `newVar` must have been created before you try and replace the existing parameter for this PDF (`oldVar`) with it.

PDFs with multiple components (+)

- AFitAddPdf can be used to easily combine several PDF components together.

$$\mathcal{P}(x) = f_1\mathcal{P}_1(x) + f_2\mathcal{P}_2(x) + \ldots + \left(1 - \sum_i f_i\right)\mathcal{P}_n(x)$$

- Easy to set up a two Gaussaian + argus PDF.

```
RooRealVar x("x", "", 0, 3);
AFitAbsPdfBuilder::setDataCard("datacardAddPdf.txt");
AFitAddPdf pdf(x, “DGA", "gaussian,gaussian,argus");
RooAbsPdf * PDF = pdf.getPdf();
```

- Easy to configure using a data card.

```
[DGA]
DGA_frac0 = 0.3 +/- 1.0
DGA_frac1 = 0.3 +/- 1.0

[DGA_0]
DGA_0mean = 1.0 +/- 0.0
DGA_0width = 1.0 +/- 0.0

[DGA_1]
DGA_1mean = 0.0 +/- 0.0
DGA_1width = 0.2 +/- 0.0

[DGA_2]
DGA_2endpt = 3.0 +/- 0.0
DGA_2xi = -20 +/- 0.0
DGA_2power = 0.5 +/- 0.0
```

Specify the fractions for the PDF under [DGA]

Specify PDF parameters for the three PDFs (the order specified in constructor is preserved in the numbering of sub-component PDFs)

PDFs with multiple components (+)

- AFitAddPdf can be used to easily combine several PDF components together.

$$\mathcal{P}(x) = f_1\mathcal{P}_1(x) + f_2\mathcal{P}_2(x) + \ldots + \left(1 - \sum_i f_i\right)\mathcal{P}_n(x)$$

- Easy to set up a two Gaussaian + argus PDF.

```
RooRealVar x("x", "", 0, 3);
AFitAbsPdfBuilder::setDataCard("datacardAddPdf.txt");
AFitAddPdf pdf(x, "DGA", "gaussian,gaussian,argus");
RooAbsPdf * PDF = pdf.getPdf();
```

- Easy to configure using a data card.

```
[DGA]
DGA_frac0 = 0.3 +/- 1.0
DGA_frac1 = 0.3 +/- 1.0

[DGA_0]
DGA_0mean = 1.0 +/- 0.0
DGA_0width = 1.0 +/- 0.0

[DGA_1]
DGA_1mean = 0.0 +/- 0.0
DGA_1width = 0.2 +/- 0.0

[DGA_2]
DGA_2endpt = 3.0 +/- 0.0
DGA_2xi = -20 +/- 0.0
DGA_2power = 0.5 +/- 0.0
```

Specify the fractions for the PDF under [DGA]

Specify PDF parameters for the three PDFs (the order specified in constructor is preserved in the numbering of sub-component PDFs)

PDFs with multiple components (×)

- AFitMultiplyPdf can be used to easily combine several PDF components together (e.g. efficiency functions):

$$\mathcal{P}(x) = \mathcal{P}_1(x) \times \mathcal{P}_2(x) \times \ldots$$

- Easy to instantiate this PDF:

```
RooRealVar x("x", "", -1.0, 1.0);
AFitAbsPdfBuilder::setDataCard("macros/testMultiplyPdf.txt");
AFitMultiplyPdf pdf(x, "helicity_pdf", "poly2,helicity");
RooAbsPdf * PDF = pdf.getPdf();
```

- Easy to configure using a data card.

```
[helicity_pdf]
types = poly2,helicity

[helicity_pdf_0]
helicity_pdf_0_p0 = 0.2 +/- 0.0 C L(-1.0 - 1.0)
helicity_pdf_0_p1 = 0.2 +/- 0.0 C L(-1.0 - 1.0)

[helicity_pdf_1]
helicity_pdf_1_type = xsqr
```

Specify the PDFs to multiply

Specify the PDF parameters

PDFs with multiple components (×)

- AFitMultiplyPdf can be used to easily combine several PDF components together (e.g. efficiency functions):

$$\mathcal{P}(x) = \mathcal{P}_1(x) \times \mathcal{P}_2(x) \times \ldots$$

- Easy to instantiate this PDF:

```
RooRealVar x("x", "", -1.0, 1.0);
AFitAbsPdfBuilder::setDataCard("macros/testMultiplyPdf.txt");
AFitMultiplyPdf pdf(x, "helicity_pdf", "poly2,helicity");
RooAbsPdf * PDF = pdf.getPdf();
```

- Easy to configure using a data card.

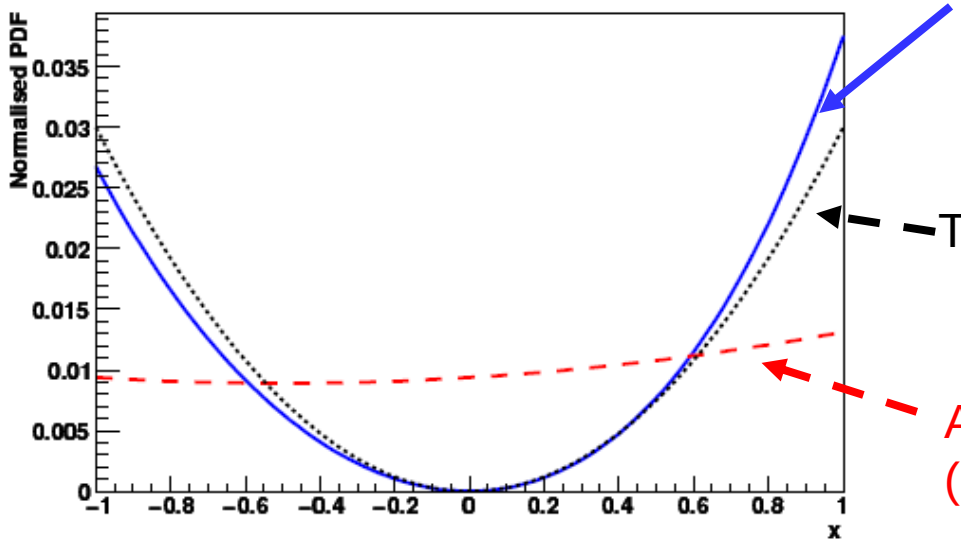
```
[helicity_pdf]
types = poly2,helicity

[helicity_pdf_0]
helicity_pdf_0_p0 = 0.2 +/- 0.0 C L(-1.0 - 1.0)
helicity_pdf_0_p1 = 0.2 +/- 0.0 C L(-1.0 - 1.0)

[helicity_pdf_1]
helicity_pdf_1_type = xsqr
```

Specify the PDFs to multiply

Specify the PDF parameters

PDFs with multiple components (×)

- this helicity×poly2 PDF looks like:



The composite PDF

The symmetric 'helicity' PDF

Acceptance function 'poly2'
(defined using a polynomial)

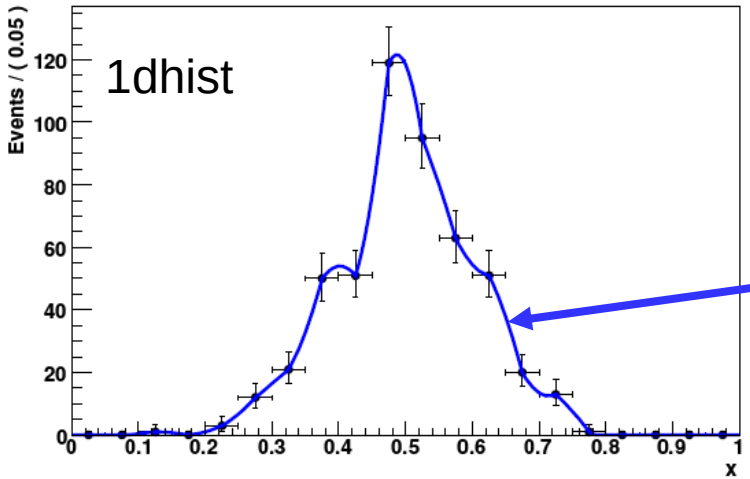
$$\mathcal{P}(x) = x^2\left(p_1 x + p_2 x^2\right)$$

- Where p1 and p_2 are the parameters of the PDF.
- In a similar way one can compute a composite sum of PDFs.

Non-parametric PDFs

- Types available: Histogram “1dhist”

  KEYS “1dkeys”



Data Points = histogram (read from file)

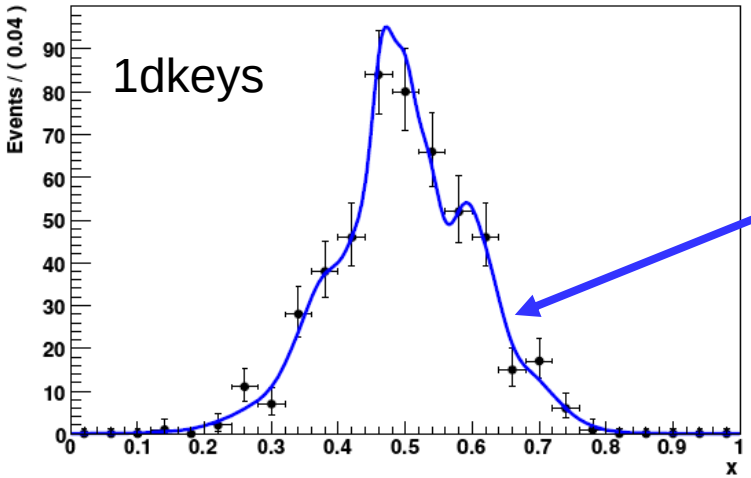
Curve = smoothed histogram, smoothing option = 2.

```
RooRealVar x("x", "x", 0.0, 1.0);
AFitHist pdfbld(x, "pdf");

pdfbld.datafile.setVal("macros/testHistPdf.root");
pdfbld.histname.setVal("hist");
pdfbld.order.setVal(2);
RooAbsPdf * pdf = pdfbld.getPdf();
```

Non-parametric PDFs

- Types available: Histogram “1dhist”

KEYS “1dkeys”



Data Points = histogram (read from file)

Curve = smoothed histogram, KEYS algorithm (sum of Gaussian kernels).

rho = smoothing parameter.

```
RooRealVar x("x", "x", 0.0, 1.0);
x.setBins(25);
AFitKeys pdfbld(x, "pdf");

pdfbld.datafile.setVal("macros/testKeysPdf.root");
pdfbld.treename.setVal("data");
pdfbld.rho.setVal(1);
RooAbsPdf * pdf = pdfbld.getPdf();
```

Available PDFs

PDF	Pdf Factory Label
Argus	argus
Breit-Wigner	breitwigner
Relativistic Breit-Wigner	relbreitwigner
Bukin	bukin
Chebychev Polynomial	chebyN
Crystal Ball	cbshape
Decay	decay
BCPDecay	cpdecay
BDecay	bdecay
Exponential	exponential
Gaussian	gaussian
Asymmetric Gaussian	agaussian
Generic PDF	generic
Gounaris-Sakurai	gounarissakurai
Helicity	helicity
Histogram	1dhist
KEYS	1dkeys
Landau	landau
Novosibirsk	novosibirsk
Polynomial	polyN
PSF	psf
Resolution‡	resolution
Sigmoid	sigmoid
Step/Veto	step
Voigtian	voigtian
Composite Add PDF	add:x,y,...
Composite Multiply PDF	multiply:x,y,...
Multi-dimensional PDFs	–

- PDF library includes everything you'd expect
  - RooFit PDFs [only 2D missing]
  - Common line-shapes.
  - Sigmoid
  - Veto/step
  - Resolution models
  - Decay models for CP fitting
- Add PDFs together in 1D.
- Multiply PDF by an 'efficiency function' (e.g. helicity distribution).
- Multiply PDFs together to make ND PDFs.

Factory Classes

- There are two types of Factory classes:
  - AFitPdfFactory:
    - Construct one of the PDF types known to AFit (or a composite of one of these types). The PDF can either be one-dimensional or N-dimensional.
  - AFitComponentFactory:
    - Construct a fit component; e.g. and N-dimensional signal or background PDF.
    - Components are:
      - default = a product of one-dimensional PDFs
      - composite = the sum of a set of PDFs, each being the product of one-dimensional PDFs.
      - vvpolarization = a longitudinal+transverse spin 0→11 model

AFitPdfFactory

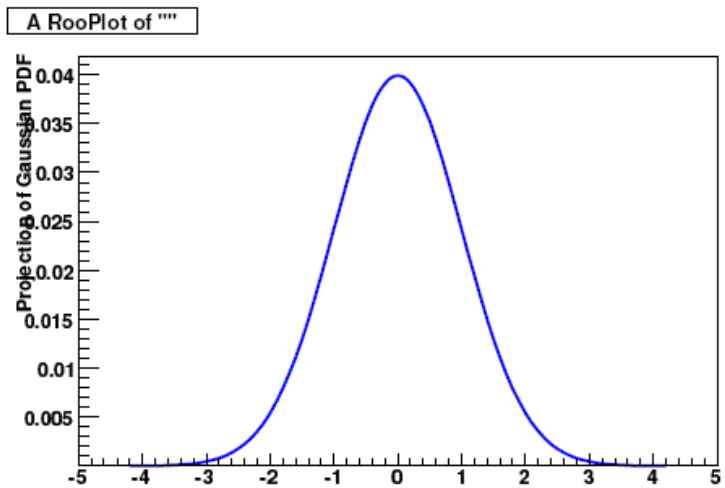
- Make an instance of the factory, then call a `makePdf` function to instantiate a PDF builder:

```
AFitAbsPdfBuilder::setDataCard("datacard.txt");

RooRealVar x("x", "x", 0.0, -5, 5);
AFitPdfFactory fact;
```

- Use the factory to make PDFs of different types

```
AFitAbsPdfBuilder * bld =
      (AFitAbsPdfBuilder *)fact.makePdf("G", "gaussian", x);

RooAbsPdf * G = bld->getPdf();
```

- Can be useful in setting up 1D fits… but not optimal for more complicated problems.



AFitPdfFactory

- The PDF Factory uses a unique 'PDF Factory Label' to identify the type of PDF to be built.
  - These are listed in the user guide and in the table back on page 11.

PDF	Pdf Factory Label	Variable Names
Argus	argus	endpt, xi, power
Breit-Wigner	breitwigner	m0, Gamma
Relativistic Breit-Wigner	relbreitwigner	mass, width, radius, spin, mass_a, mass_b
Bukin	bukin	Xp, sigp, xi, rho1, rho2
Chebychev Polynomial	chebyN	_p*i*
Crystal Ball	cbshape	mean, width, alpha, n
Decay	decay	tau
BCPDecay	cpdecay	tau, dm, S, C, avgMistag, delMistag, mu
BDecay	bdecay	tau, dm, f0, f1, f2, f3, dgamma
Exponential	exponential	constant
Gaussian	gaussian	mean, width
.		
.		
.		

- Some PDFs may depend on conditional variables. For example, the resolution function.
  - These can be made by calling the `makeConditionalPdf` function to instantiate a PDF builder.

AFitPdfFactory

- Multi-dimensional PDFs can also be made using call to `makePdf`:

```
AFitAbsPdfBuilder * AFitPdfFactory::makePdf(TString name,
                                            RooArgList &discVarList)
```

- This actually makes an `AFitProdPdf` object.

```
RooRealVar x("x", "", -1, 3);
RooRealVar y("y", "", 0, 3);
RooRealVar q("q", "", -2, 3);
AFitAbsPdfBuilder::setDataCard("datacardProdPdf.txt");
RooArgList discVars;
discVars.add(x);
discVars.add(y);
discVars.add(q);

AFitPdfFactory fact;
AFitAbsPdfBuilder * componentPDF = fact.makePdf("component", discVars);

RooAbsPdf * PDF = componentPDF.getPdf();
```

The discriminating variables

The data card specifying the PDF configuration and parameters.

The 3D RooAbsPdf model $\mathcal{P}(x, y, z)$.

This brings us onto the topic of fit components

AFitComponentFactory

- The different PDF components implemented are:
  - default:

$$PDF(\underline{x}) = PDF_1(x_1) \times PDF_2(x_2) \times \ldots \times PDF_N(x_N)$$

  - composite:

$$PDF(\underline{x}) = (1 - f_1 - \ldots - f_n)\mathcal{C}_0(\underline{x}) + f_1\mathcal{C}_1(\underline{x}) + \ldots + f_n\mathcal{C}_n(\underline{x})$$

    The C_i are themselves components, and they are combined with relative fractional weighting f_i.

  - vvpolarisation:

$$\mathcal{P}(\underline{x}) = f_L^{eff}\mathcal{P}_L(\underline{x}) + (1 - f_L^{eff})\mathcal{P}_T(\underline{x})$$

$$f_L^{eff} = \frac{f_L}{(1 - f_L)\epsilon_T/\epsilon_L + f_L}$$

    L = longitudinal, T=transverse, and ε are efficiencies.

    f_L is the fraction of longitudinal events, and f_L^{eff} is an effective parameter related to f_L.

AFitComponentFactory

- Use is similar to the PDF Factory:
  - Define discriminating variable(s).

```
RooRealVar x("x", "", -1, 3);
RooRealVar y("y", "", 0, 3);
RooRealVar q("q", "", -2, 3);
RooArgList ll(x, y, q);
```

List of discriminating variables.

  - Specify datacard with fit configuration information.

```
AFitAbsPdfBuilder::setDataCard("datacardComponentModel.txt");
```

  - Make the PDFs:

```
AFitComponentFactory fact;
RooAbsPdf * signalPdf   = fact.makeComp("signal", "default", ll);
RooAbsPdf * bgPdf       = fact.makeComp("bg", "composite", ll);
RooAbsPdf * vvsignalPdf = fact.makeComp("vvsignal", "vvpolarisation", ll);
```

PDF name

component type

- There is also a global PDF builder class that can make the whole fit model for you.

The AFitMaster

- The AFitMaster can
  - Build a multi-dimensional PDF with many fit components.
  - Build a simultaneous PDF.
  - Fit data samples to define PDF parameters.
  - Write out a template datacard given an initial fit configuration.
- This class helps manage the construction of your fit.

Using AFitMaster to build a model

- Build a complicated model with a 2 line ROOT macro:

```
AFitMaster master(‘‘mydatacard.txt’’);
RooAbsPdf * pdf = master.getPdf();
```

- Have to specify fit configuration in text file (datacard):

```
[FitConfiguration]
// specify the variables to use in the fit
variables = bMes,bDeltaE
// specify the names of the signal and background components
components = signal,continuum,Bbg0
fitOptions = etrmh

// set the limits and initial values of the variables used
bMes = 5.2700 +/- 0 L(5.25 - 5.29) B(30)
bDeltaE = 0.0000 +/- 0 L(-0.3 - 0.3) B(30)

// set the component types
signal = default
continuum = default
Bbg0 = default

// give initial values for the yields of each component
signalYield = 500.00 +/- 10.000 L(-100 - 10000)
continuumYield = 2000.00 +/- 10.000 L(-100 - 10000)
Bbg0Yield = 50.000 +/- 10.000 L(-100 - 10000)
```

You define the:

variables to use

fit components

component types

fit yields (assumes you want to do an extended-unbinned ML fit.

Using AFitMaster to build a model

- Build a complicated model with a 2 line ROOT macro:

```
AFitMaster master(‘‘mydatacard.txt’’);
RooAbsPdf * pdf = master.getPdf();
```

- Have to specify fit configuration in text file (datacard):

```
[FitConfiguration]
// specify the variables to use in the fit
variables = bMes,bDeltaE
// specify the names of the signal and background components
components = signal,continuum,Bbg0
fitOptions = etrmh

// set the limits and initial values of the variables used
bMes = 5.2700 +/- 0 L(5.25 - 5.29) B(30)
bDeltaE = 0.0000 +/- 0 L(-0.3 - 0.3) B(30)

// set the component types
signal = default
continuum = default
Bbg0 = default
// give initial values for the yields of each component
signalYield = 500.00 +/- 10.000 L(-100 - 10000)
continuumYield = 2000.00 +/- 10.000 L(-100 - 10000)
BbgOYield = 50.000 +/- 10.000 L(-100 - 10000)
```

Component name + “Yield”

You define the:

variables to use

fit components

component types

fit yields (assumes you want to do an extended-unbinned ML fit.

Using AFitMaster to build a model

```
// define the shapes used for the signal M_ES and ΔE PDFs
[signal]
signal_bMes_type = gaussian
signal_bDeltaE_type = landau

// define the shapes used for background M_ES and ΔE PDFs
[continuum]
continuum_bMes_type = argus
continuum_bDeltaE_type = poly2

// define the shapes used for background M_ES and ΔE PDFs
[Bbg0]
Bbg0_bMes_type = argus
Bbg0_bDeltaE_type = poly2
```

You define the:

pdf types for each component

- The rest of the configuration file is used to specify the shape parameters.
  - By default all parameters are allowed to float, and have a dummy range.
  - Interface to fit to MC/data control samples (see user guide for details).
  - Can also build a RooSimultaneous using `getSimPdf()`.

Using AFitMaster to build a model

```
// define the shapes used for the signal M_ES and ΔE PDFs
[signal]
signal_bMes_type = gaussian
signal_bDeltaE_type = landau

// define the shapes used for background M_ES and ΔE PDFs
[continuum]
continuum_bMes_type = argus
continuum_bDeltaE_type = poly2

// define the shapes used for background M_ES and ΔE PDFs
[Bbg0]
Bbg0_bMes_type = argus
Bbg0_bDeltaE_type = poly2
```

You define the:

pdf types for each component

Component name used as block name, and used as prefix for all variable names.

- The rest of the configuration file is used to specify the shape parameters.
  - By default all parameters are allowed to float, and have a dummy range.
  - Interface to fit to MC/data control samples (see user guide for details).
  - Can also build a RooSimultaneous using `getSimPdf()`.

Using AFitMaster to build a model

- Defining the signal PDF parameters:

```
[signal]
signal_bMes_type = gaussian
signal_bDeltaE_type = landau
```

The type specified defines the variables are used in the PDF, and the variables that should be specified in the datacard.

Signal m_{ES} distribution is a Gaussian:

```
[signal_bMes]
signal_bMesmean =  5.28 +/- 0.01 C L(5.25 - 5.29)
signal_bMeswidth =  0.003 C L(0 - 0.04)
signal_bMeswidthL =  1.0000 L(0 - 0.04)
signal_bMeswidthR =  1.0000 L(0 - 0.04)
signal_bMesasymmetricWidth = false
```

If asymmetricWidth=true, then the L and R widths are used instead of the symmetry one.

Signal ΔE distribution is a Landau:

```
[signal_bDeltaE]
signal_bDeltaEmean =  0.0000 C L(-0.3 - 0.3)
signal_bDeltaEsigma =  0.01 C L(0 - 0.6)
```

Using AFitMaster to build a model

- Defining the signal PDF parameters:

```
[signal]
signal_bMes_type = gaussian
signal_bDeltaE_type = landau
```

The type specified defines the variables are used in the PDF, and the variables that should be specified in the datacard.

Signal m_{ES} distribution is a Gaussian:

```
[signal_bMes]
signal_bMesmean =  5.28 +/- 0.01 C L(5.25 - 5.29)
signal_bMeswidth =  0.003 C L(0 - 0.04)
signal_bMeswidthL =  1.0000 L(0 - 0.04)
signal_bMeswidthR =  1.0000 L(0 - 0.04)
signal_bMesasymmetricWidth = false
```

If asymmetricWidth=true, then the L and R widths are used instead of the symmetry one.

Signal ΔE distribution is a Landau:

```
[signal_bDeltaE]
signal_bDeltaEmean =  0.0000 C L(-0.3 - 0.3)
signal_bDeltaEsigma =  0.01 C L(0 - 0.6)
```

L(x – y) sets the allowed range for parameters floating in the fit.

C means parameter is constant

Using AFitMaster to build a model

- Defining the continuum PDF parameters:

```
[continuum]
continuum_bMes_type = argus
continuum_bDeltaE_type = poly2
```

The type specified defines the variables are used in the PDF, and the variables that should be specified in the datacard.

Continuum m_{ES} distribution is an argus:

```
[continuum_bMes]
continuum_bMesendpt =  5.2900 C L(-INF - +INF) // [GeV/c2]
continuum_bMesxi = -20.0000 C L(-200 - 0)
continuum_bMespower =  0.50000 C L(-INF - +INF)
```

Continuum ΔE distribution is a polynomial:

```
[continuum_bDeltaE]
continuum_bDeltaE_p0 =  0.1000 +/- 0.05 L(-1 - 1)
continuum_bDeltaE_p1 =  0.2000  +/- 0.05 L(-1 - 1)
```

Using AFitMaster to build a model

- Defining the BBg0 PDF parameters:

```
[Bbg0]
Bbg0_bMes_type = argus
Bbg0_bDeltaE_type = poly2
```

The type specified defines the variables are used in the PDF, and the variables that should be specified in the datacard.

B background m_{ES} distribution is an argus:

```
[Bbg0_bMes]
Bbg0_bMesendpt =  5.2900 C L(-INF - +INF) // [GeV/c2]
Bbg0_bMesxi = -30.0000 C L(-200 - 0)
Bbg0_bMespower =  0.50000 C L(-INF - +INF)
```

B background ΔE distribution is a polynomial:

```
[Bbg0_bDeltaE]
Bbg0_bDeltaE_p0 =  1.0000 C L(-1 - 1)
Bbg0_bDeltaE_p1 =  0.1000 C L(-1 - 1)
```

Using AFitMaster to build a model

- How do I know what needs to be configured in the datacard?
  - Each PDF has a unique name given by:

    <component name>_<discriminating variable name>
  - This PDF name us used in square brackets to start a 'block' of the datacard. Within this block, PDF variables are defined.
  - The names of these variables can be found by reading the source code for the class. They all start with the PDF name.

- e.g. a 1dhist PDF is created using AFitHist. If you read the code in the constructor function of the class (in AFitHist.cc) you'll see the following lines of code:

```
datafile(name+"_file", "File name", "null"),
histname(name+"_hist", "Histogram name", "null"),
order(name+"_order", "Interpolation order", 0)
```

Using AFitMaster to build a model

- e.g. a 1dhist PDF contd.

```
datafile(name+"_file", "File name", "null"),
histname(name+"_hist", "Histogram name", "null"),
order(name+"_order", "Interpolation order", 0)
```

The variable name rule

description of the variable

initial value (&range)
only values are appropriate
for this class.

- So if I want to configure a 1dhist PDF for component the discriminating variable x of the component signal then the block in the datacard would look like:

```
[signal_x]
signal_x_file = myHistRootFile.root
signal_x_hist = theHistogramName
signal_x_order = 1 C
```

Note: For the non-parametric PDFs, you have to make sure that the ROOT files you specify exist, and contain the data (e.g. histogram) prior to making the PDF

Simultaneous PDFs

- There is a `getSimPdf()` function that can be used to build simultaneous PDFs:

```
AFitMaster master("AFit/example/cpfit_tagging.txt");
RooAbsPdf * pdf = master.getSimPdf();
```

- The datacard needs to specify the categorie(s) and splitting rules for the underlying RooSimPdfBuilder.

```
catVars = signal_deltat_cat

signal_deltat_cat_categories = Lepton,Kaon1,Kaon2,KaonPion,Pion,Other,NoTag

signal_deltat_cat_splitrule = signal_deltat_cat :
                  signal_deltat_avgMistag,signal_deltat_delMistag,signal_tageff
```

The category used to split the PDF

Labels for the category

List of variables to split by category labels

- This problem is covered in one of the AFit examples.

Blinding

- By default all yields are unblind.
- It is trivial to blind a specific yield:
  - Each component coefficient has a variable called

    `<coefficient name>_BlindingType`
  - which can have one of two values:

    ```
    unblind              [this is the default]
    blind                [use this value to blind parameter]
    ```
  - For example to blind the signal yield, add

    `signalYeild_BlindingType = blind`

    to the `[FitConfiguration]` block of your datacard.
- A RooUnblindOffset is used to blind parameters.

Blinding

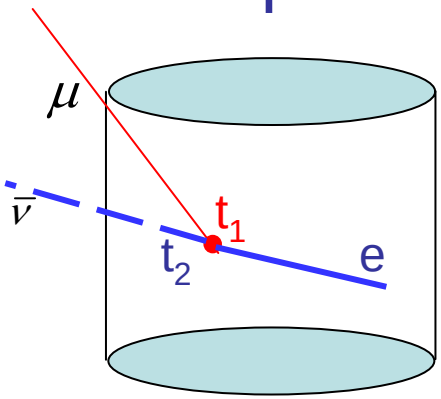
- In addition to blinding yields using the AFitMaster, you can blind CP asymmetries:
  - AFitBCPGenDecay has the option to blind S and C.
  - You can blind S and C by adding

    ```
    <pdf name>blindingState = blind
    ```

    to the datacard block that defines the parameters for this PDF.
  - When fitting several components with a TDCPV model for Δt, the blinding of S and C for each component is independent.

Example: τ_μ

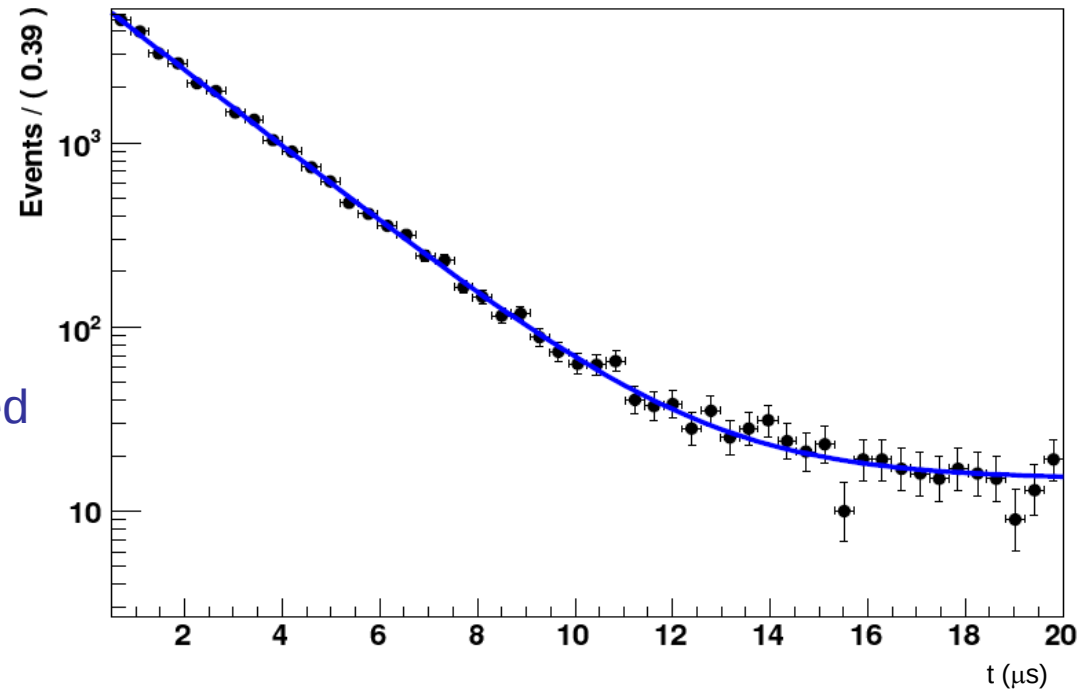
- Simple undergraduate experiment



Neglecting detector resolution effects, $t = t_2 - t_1$ has the form of an exponential (signal) + constant (background) distribution:

$$\mathcal{P} = N_{\text{signal}} e^{-t/\tau_\mu} + N_{\text{background}}$$

- Straightforward to write a data card and use the AFitMaster to fit such a distribution.
- This example is also documented in the user guide.



Example: τ_μ

- The macro:

Specify the data card and make the pdf.

```
AFitMaster master("AFit/example/MuonLifetime.txt");
RooAbsPdf * pdf = master.getPdf();

TFile f("AFit/example/MuonLifetime.root");
TTree * tree = (TTree*)f.Get("data");

RooArgSet * compSet = pdf->getComponents();
RooArgSet * parSet  = pdf->getParameters(compSet);

RooRealVar * t = (RooRealVar*)parSet->find("t");

RooDataSet  data("data", "", tree, RooArgSet(*t));
data.Print("v");

pdf->fitTo(data, "etrm");
```

Prepare a RooDataSet for fitting

Fit the data

- These are all the steps required in a ROOT macro to prepare a PDF for fitting.

Example: τ_μ

- The data card.

```
[FitConfiguration]
// specify the variables and components to use in the fit
variables = t
components = signal,background
fitOptions = etrmh

t = 0.5 +/- 0.01 L(0.5 - 20.0) B(50)

signal = default
background = default

signalYield =  30000 +/- 10.000 L(-100 - 1e6)
backgroundYield =  100 +/- 100.000 L(-1000 - 1e4)

[signal]
signal_t_type = exp

[background]
background_t_type = poly1

[signal_t]
signal_tconstant = 2.2 +/- 0.1 L(-4.0 - 4.0)
signal_tfitLifetime = true

[background_t]
background_t_p0 =  0.0000 C L(-1 - 1)
```

Specify the discriminating variables and pdf componets

Define discriminating variable ranges

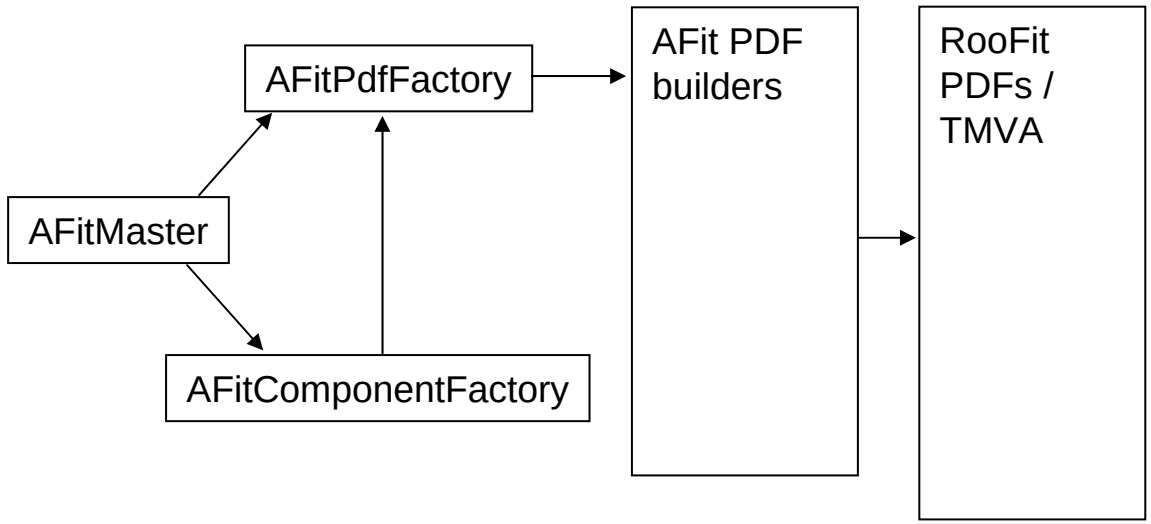
Specify the fit component types and yields.

Specify the PDF types for the components

Specify the parameters for the PDFs.

Overview of the package

Can use the AFitMaster, Factories or builders to make RooAbsPdf objects. These can be used as is or combined with other RooAbsPdfs using RooFit directly. Alternatively they can be used with one of the AFit utility classes.



Once the model has been instantiated as a RooAbsPdf, the utilitiy classes can be used to validate and develop the model, fit data, and inspect the results. The utilities are independent of the AFit builders and can be used with any RooAbsPdf.

Utility classes:
Plotting
Statistics
Toy MC studies
TMVA Interface

Where to find out more information

- *AFit* is available from:

  http://pprc.qmul.ac.uk/~bevan/afit/

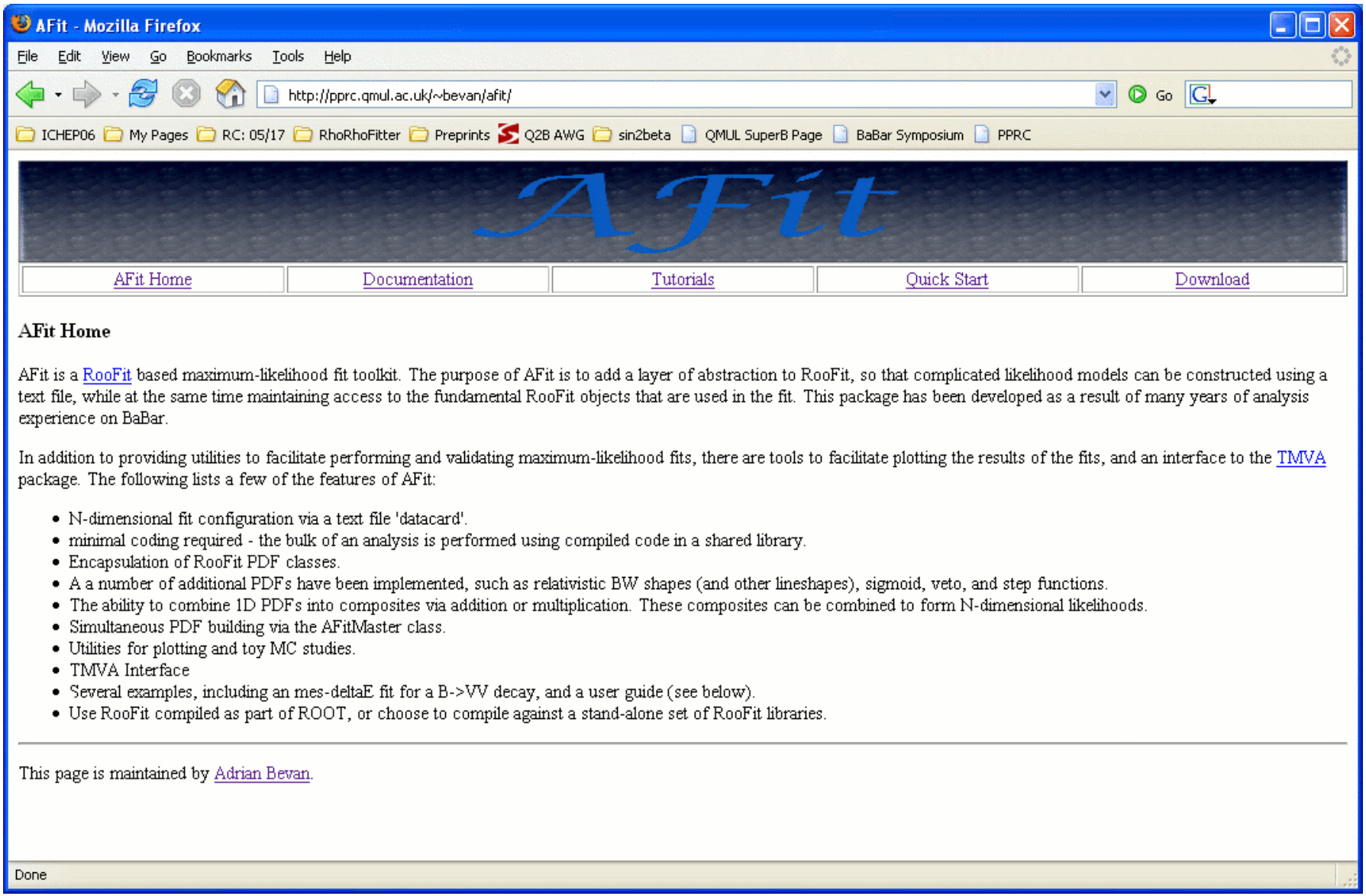
- Available for down load:
  - User guide
  - Source code
  - Examples available (sub-directory of source code):
    - Exponential decay fit for lifetime
    - M_{ES}-ΔE fit configuration for signal + continuum + B background
    - Simultaneous PDF
    - How to set up a simple time-dependent CP fit

- Web page also has quick start compile instructions.

- Requirements: ROOT v5.20 or v5.22.

- Need to pre-compile ROOT with RooFit, or compile RooFit before compiling AFit (requires modification of GNUMakefile.standalone).

AFit Home Page



Documentation available:

AFit User Guide

Abstract

AFit provides a high level user interface to RooFit and TMVA. Complicated maximum-likelihood fits can be set up using a text file (without the need to write a lot of code), and modified quickly to refine an analysis. There are a number of utilities to facilitate further analysis of the likelihood fit, such as toy MC, and plotting interfaces. The TMVAInterface provided allows a user to configure which classifiers to run with which variables. It is then possible to append RooDataSets with the output classifier MVAs for inclusion in a fit to the data. A number of examples are provded in the last section of this user guide.

Contents

1

2

1 Introduction to AFit

This package has been developed in order to add a layer of abstraction on top of RooFit [1] and simplify complex maximum likelihood fit based data analysis. The underlying function minimiser is MINUIT [2]. The aim of AFit is to provide a general fit framework that can be used in order to analyse data, but without having to write code in order to set the fit up. In order to do set up a general fit, you can configure AFit using a 'datacard'. In addition to this high level abstraction, it is also possible to use the individual wrapper classes to the RooFit Probability Density Functions (PDFs) when addressing simple problems.

The default version of RooFit to use with AFit is the version bundled with ROOT [3]. If this is not suitable for your needs, it is possible to uncomment the line `#define __BABARROOFIT__` in `AFit/GlobalInfo.hh` in order to compile against a version of RooFit that has been checked out and compiled in the same base directory as AFit.

AFit provides a higher level interface to RooFit that facilitates using a text configuration

3