QSigEx Manual

Pierre-Luc Drouin
Alain Bellerive

Carleton University

November 26, 2007

Preface

As a summer student working on the Sudbury Neutrino Observatory (SNO) experiment at Carleton University, I have been asked in the second half of summer 2002 to write a C++ package that would allow to extract neutrino fluxes from the detector data. The package had to be designed such that it could be expanded by the users and that its structure would be flexible enough to allow analysis changes without code modifications. Under the supervision of Alain Bellerive, I have written what became QSigEx 1.00. The package met the first design rule. However, due to the limited time we had to write some important blocks of code, the program was not as flexible and as user-friendly as it should.

In summer 2003, I decided to redesign the package. I wanted to increase the flexibility of QSigEx, but also to incorporate more the ROOT classes into the code. I've decided to store the information produced by the package into a ROOT `TDirectory` structure, that would give to QSigEx a very user-friendly interface in comparison to C-style arrays used in version 1.00. Finally, after three months of work, I got a package that satisfied our objective.

I hope QSigEx will suit your needs!

Pierre-Luc Drouin
Ottawa, August 2003.

WEB

`www.physics.carleton.ca/research/sno/anal/software/qsigex.html`

Authors and Contributors

Alain Bellerive: Supervisor, librarian (2004), gaussian correlations

Mark G. Boulay: Contribution to QSigEx behaviour

Pierre-Luc Drouin: Code design and implementation

Darren Grant: Librarian (2002,2003), extended likelihood function coding

Kathryn Miknaitis: Librarian (2002,2003), contribution to `QSigExFit` class coding

Osama Moussa: Contribution to `QSigExTHOps` class coding

Ryan MacLellan: Contribution to `QSigExDis` and `QSigExIO` behaviour

Contents

1 Introduction

QSigEx is a package that can be used to estimate population parameters using a sample of events. In particular, it estimates in which proportions some event groups are present in the population. This situation arises often in particle physics in what is called signal extraction, from which QSigEx borrows its name.

Let a population be composed of n groups. Each entry in the population has some measurable characteristics, such that this entry can be described by a vector $\vec{x}$, where each x_i coordinate $(i = 1, 2, \ldots, m)$ corresponds to a specific observable. The population entries of a given group have characteristic $\vec{x}$ values that allow to differenciate the group from another one. Then, the entries in a group have a particular distribution. In some cases, it is possible to predict the distribution associated which each group (using Monte Carlo method for example). Provided these distributions, the probability density of an event member of group G_j to have coordinates $\vec{x}$ is given by $f(\vec{x}|G_j)$. In QSigEx, the function $f(\vec{x}|G_j)$ is called the *joint probability density function* for the group G_j, since it gives the probability density of an event considering all its characteristics ($\vec{x}$).

In the case where all the groups are mutually exclusive (an entry member of group G_j cannot be member of group G_k at the same time, for all j and k, $j \neq k$), the probability density of an entry with coordinates $\vec{x}$ to be a member of any group is given by:

$$f(\vec{x}) = \sum_{j=1}^{n} f(\vec{x} \cap G_j) = \sum_{j=1}^{n} p(G_j) f(\vec{x}|G_j) \tag{1.1}$$

where $p(G_j)$ is the probability of an event to be member of group G_j. The set of probabilities $p(G_1),\ p(G_2),\ \ldots\ ,\ p(G_n)$, or some values function of these probabilities (depending on the minimization function that is defined by the user) are the population parameters that are estimated by QSigEx. They can be viewed as the fraction of events which belong to a given group.

In a given group, the variables x_i can be independent or not. When the variables are considered to be independent, the function $f(\vec{x}|G_j)$ $(j = 1, 2, \ldots, n)$ can be expressed using the *marginal probability density functions* $f_i(x_i|G_j)$ for each of these variables:

$$f(\vec{x}|G_j) = \prod_{i=1}^{m} f_i(x_i|G_j) \tag{1.2}$$

In this expression, m is the number of dimensions of vector $\vec{x}$ used to fit the parameters.

Sometimes, only a pair of variables are significantly correlated. In this situation, the user can choose to produce, using a method of his choice, a probability density function (PDF) $f_{k,\,l}\,((x_k, x_l)|G_j)$ $(k,\ l\ \ \in \{1, 2, \ldots, m\})$ that would be equivalent to $f_k(x_k|G_j) f_l(x_l|G_j)$ if the variables were not correlated. In QSigEx, this is called a *bidimensional marginal probability density function*, since it doesn't take into account

all the variables. The *joint probability density function* $f(\vec{x}|G_j)$ is then given by:

$$f(\vec{x}|G_j) = f_{k,\,l}\left((x_k, x_l)|G_j\right) \prod_{\substack{i=1 \\ i\neq k,\; i\neq l}}^{m} f_i(x_i|G_j) \tag{1.3}$$

The procedure can be repeated with other pairs of variables, if these pairs are not correlated to the other variables and also with triplet of variables if necessary.

When the correlations between the variables are generally strong enough to be considered, the information provided by the functions $f_i(x_i|G_j)$ is not sufficient and the joint probability density functions $f(\vec{x}|G_j)$ must be defined using more sophisticated methods. QSigEx can handle uncorrelated or correlated variables [3]. It can compute joint probability densities using either unidimensional or multidimensional marginal probability density functions.

2 Getting Started

2.1 QSigEx and ROOT

QSigEx relies on ROOT [4], the object-oriented data analysis framework developed at CERN. Not only the functions used to generate the PDFs and the object containing the sample data are ROOT objects; the QSigEx classes are themselves derived from ROOT classes and their outputs are ROOT objects as well. It is important that QSigEx users be familiar with ROOT before using QSigEx, considering how QSigEx is based on this program.

2.2 Main Modules

As a QSigEx user, you will have to use a reduced number of classes. Among these classes are the QSigEx main modules. Each of these modules accomplishes a specific task in the parameters fitting process. Here are the steps that are followed in order to evaluate the population parameters as they are divided via the main modules.

Module:

1. Since the variables x_i of the population entries generally have values that are contained in a certain range and since it is often impossible in practice to predict the shape of their distribution for an infinite range of values, it's usually needed to define *cuts* that define regions of valid variables values.
2. A *data sample* must be loaded in QSigEx. The defined cuts are applied on this sample to get a *clean data sample*.
3. The *marginal PDFs* are produced by QSigEx using the ROOT objects provided by the user. These objects are usually generated using an analytical or Monte Carlo method. The marginal PDFs are normalized by QSigEx according to the defined cuts.
4. If the variables x_i cannot be considered independent, information related to their correlations is loaded by QSigEx.
5. Using the clean data sample and the marginal PDFs, the probability density of each data event to have a value of x_i for its coordinate i if the event belongs to group G_j is computed for each species.
6. Using the probability densities computed at the preceding step, the joint probability density of each data event to have coordinates $\vec{x}$ if the event belongs to species G_j is computed for each species. In the simple case where the variables x_i are not correlated, this is simply the product of all marginal probability densities of a given group for each of these species, as explained in section 1.

```
TFile f1("qsigex.root","NEW");

QSigExCuts cuts(&f1,"cardfile.dat");
cuts.Get();

QSigExCleanData cleandata(&f1,"cardfile.dat");
cleandata.Get();

QSigExTTreePDF pdfs(&f1,"cardfile.dat");
pdfs.Get();

QSigExGaussCor gcor(&f1);
gcor.Get();

QSigExProbs<Float_t> probs(&f1);
probs.Get();

QSigExGCJointProbs gcjprobs(&f1);
gcjprobs.Get();

QSigExFit fitter(&f8,QExtendedLikelihood,"cardfile.dat");
fitter.Get();

f1.Write();
f1.Close();
```

Figure 2.1: Example of QSigEx usage

7. Finally, the parameters are evaluated, using a minimization function defined by the user and the joint probability densities computed at step 6.

Figure 2.1 shows an example of a standard run of QSigEx for correlated variables. You can see the seven QSigEx classes instances that are created and for which a `Get()` member function is called. The interface of QSigEx will be explained in more details in the next section.

3 General Interface of QSigEx

3.1 Overview

As it has been explained in preceeding sections, QSigEx is divided in main modules that each accomplish one of the six (or seven) steps needed to evaluate the parameters. To perform this task, the modules need to share some information. To improve the flexibility of the code and to allow the users to access the infomation easily, it is done using a `TDirectory` structure. `TDirectory` is an important ROOT class that is, among others, the class from which is derived `TFile`.

In QSigEx, a given main module doesn't depend directly on another one. Instead, each module reads the needed information and writes its results in the `TDirectory` structure. It allows the developer to define and use a new module without having to modify the others. Moreover, since all the main classes of QSigEx are derived from a common abstract base class, the member functions of a module can be called using a pointer to its base class, increasing the flexibility of the code.

All the information produced by the main modules is added to the `TDirectory` structure, as standard ROOT objects (`TTree`, `TF1`, `TNamed`, etc.) or as QSigEx objects derived from `TObject` (example: `QSigExDis` derived classes). In all cases, almost all the information can be easily accessed using a ROOT `TBrowser` object. For example, the parameters values are written as `TNamed` objects, which title is the fit value. Also, all the marginal PDFs, data marginal probability densities and joint probability densities can be plotted by a simple mouse click in a `TBrowser`.

Since some initial information has to be provided to the main modules, the user has to write configuration parameters in a file before to run QSigEx.

3.2 `QSigExDirHandler`: A Common Interface

`QSigExDirHandler` is an abstract base class for the main QSigEx classes. It provides a common interface for all these classes, such that the user already knows the basic behaviour of the member functions of a main module before starting using it. Another advantage of `QSigExDirHandler` is that it allows to hold a pointer to a main class instance that has one of the six (or seven) roles described earlier without hardcoding it's class name. The following subsections describe the behaviour of the pure virtual member functions declared in the `QSigExDirHandler` class, but also some of the public member functions implemented in this class.

3.2.1 `void SetDir(TDirectory* folder)`

This function sets the address of the `TDirectory` instance that is used to hold the information of QSigEx. This `TDirectory` is the root directory of the QSigEx structure. A `TFile` pointer, or a pointer to an instance of any class derived from `TDirectory` can be passed.

3.2.2 `TDirectory* GetDir()`

This function returns a pointer to the `TDirectory` instance previously set using `QSigExDirHandler::SetDir`.

3.2.3 `void LoadCardFile(const Char_t* cardfilename)`

This pure virtual member funcion reads the content of the configuration file which filename is `cardfilename` and stores the information related to the module into its internal member variables.

3.2.4 `void ClearCardBuf()`

This pure virtual member function erases the information held by the internal member variables of the module where is stored the information from the configuration file.

3.2.5 `Int_t Get()`

This pure virtual member function uses the information stored in the internal variables of the module and from the `TDirectory` structure to accomplish the task assigned to the module. The results of this task are stored in the `TDirectory` structure.

3.2.6 `void CleanDir()`

This pure virtual member function removes all the previous results associated with the current instance from the `TDirectory` structure. It also deletes the `TDirectory` objects created by the instance.

```
#               ROOT file                 obj name
DATA_FILE       SaltRunsApr25_Blind.root  treeobj

#       name    condition
cut     rmax    r<=550
cut     temin   teff>=5.5
cut     itrmin  itr>0.55
cut     b14min  b14>-0.12
cut     b14max  b14<0.95
cut     ctmin   cossun>=-1
cut     ctmax   cossun<=1

#     type  group  s.group  obj name  ROOT file   inputs
pdf 1  TH1F   cc       cossun  cc_cossun pdfs.root  cossun
pdf 1  TH2F   cc       b142d   cc_b142d  pdfs.root  b14  teff

pdf 1  TH2F   es       b142d   es_b142d  pdfs.root  b14  teff
pdf 1  TH1F   es       cossun  es_cossun pdfs.root  cossun

pdf 1  TH2F   nc       b142d   nc_b142d  pdfs.root  b14  teff
pdf 1  TH1F   nc       cossun  nc_cossun pdfs.root  cossun

#     parameter name  active  start  min   max    step
flux    alpha_cc        1      500    0  1000000  0.01
flux    alpha_es        1      500    0  1000000  0.01
flux    alpha_nc        1      500    0  1000000  0.01

#         Minuit algorithm  "UP" value
minimizer      MIGrad           1
```

Figure 3.1: An example of a QSigEx configuration file

3.3 The Configuration File

To configure QSigEx, the user has to create a configuration file (card file). Some of the main modules need to read parameters from this file and some other do not. The configuration file entries format depends also on the main module. The user should look at the QSigEx autogenerated HTML [2] documentation produced with ROOT to learn in details the card file syntax. Figure 3.1 shows a configuration file example.

3.4 The `TDirectory` Structure

The main modules produce results that are stored in the QSigEx `TDirectory` structure. A pointer to the root of this structure has to be passed to initialize the classes derived from `QSigExDirHandler`. Each one of these classes modifies the structure, by adding subdirectories and other types of objects. Even if the `TDirectory` structure can differ depending on the specific classes that are used to get the parameters values, there are some subdirectories of this structure that are created independently of the main modules combination that is used. For more details on the specific interface of a given class, please refer to the QSigEx autogenerated HTML [2] documentation. A description of the common QSigEx directories is provided in this section. The module numbers are the ones described in section 2.2.

3.4.1 "Cuts" `TDirectory`

The "Cuts" `TDirectory` is created by a module 1. It contains the subdirectories "Equivalences" and "Cuts Expressions" that are both used by modules 2 and 3 to apply cuts. The first subdirectory contains a set of formulæ that allow to simplify the cuts expressions. One equivalence, for example, can combine the variables x, y and z to express the radius r as $\sqrt{x^2+y^2+z^2}$. A cut, defined in the "Cuts Expressions" `TDirectory`, can use this equivalence to express limits on the radius values.

3.4.2 "Event Info" `TDirectory`

The "Event Info" `TDirectory` contains the information related to the data sample used in the fit. Only the events within the predefined cuts are written in this directory. The information is written by a module 2 as a `TTree` which branches contain Float_t values.

3.4.3 "PDFs" `TDirectory`

This `TDirectory` is used to save the information related to the population probability density functions $f(\vec{x}|G_j)$. "PDFs" is subdivided in directories that represent the population groups. Each one of these directories contains `TDirectory` objects that are named according to the systematic groups of the population. These groups are not used in the fitting process of QSigEx, but are useful if one wants to compute systematic errors on the fitted parameters. Inside these folders are contained the marginal PDFs directories. A PDF `TDirectory` is basically composed of an object derived from `QSigExDis` (the PDF itself) and an "Inputs" directory where are listed the PDF coordinates names. Other objects can be written in one of the described directory levels if, for example, correlations have to be computed between the variables. Modules 3 and if needed a module 4 add content to "PDFs" `TDirectory`.

3.4.4 "Probs" `TDirectory`

The "Probs" `TDirectory` is divided into two subdirectories: "PDFsProbs" and "JointPDFsProbs". These folders contain the marginal probability densities and the joint probability densities respectively. Both contain `TTree` objects which branches are composed of Double_t values. Modules 5 and 6 contribute to the "Probs" `TDirectory`.

3.4.5 "Fit" `TDirectory`

The modules 7 write their results and the parameters configuration in the "Fit" `TDirectory`. The user should refer to the HTML [2] documentation for more details.

Bibliography

[1] Glen Cowan. *Statistical Data Analysis*. Oxford Science Publications, Clarendon Press, Oxford, 1998.

[2] Pierre-Luc Drouin. *QSigEx Classes and Members Reference Guide*. http://www.physics.carleton.ca/research/sno/anal/software/qsigex.html, Carleton University, Ottawa.

[3] Dean Karlen. Using projections and correlations to approximate probability distributions. *Computers in Physics*, 1998.

[4] R. Brun & F. Rademakers. *ROOT*. http://root.cern.ch.