

Adding a Custom Chain Proposal Algorithm

When running a Monte Carlo Markov Chain with the **chain** command, XSPEC provides several built-in proposal options from which to draw trial parameter values for the next step in the chain. A **built-in** proposal is selected prior to the chain run with the command:

```
chain proposal <distribution> <source>
```

where <distribution> is the statistical distribution used to randomize the parameter values (e.g. gaussian, cauchy), and <source> refers to the source of the applied covariance information (see the **chain** command for details).

It is also possible for the user to create an arbitrary new proposal scheme and add it to the options available under the chain proposal command. This is done in a way similar to the adding of local models described in Appendix C, though in this case the code can only be written in C++. Essentially three steps are involved, each described in greater detail below:

- Create a small text file named `randomize.dat`.
- Write a class which inherits from XSPEC's abstract base class `RandomizerBase`.
- Run XSPEC's **initpackage** and **lmod** commands to build and load the shared library containing the new proposal class(es).

The `randomize.dat` Initialization File

This file must be placed in the same directory as the user's proposal code files, and plays a role similar to the local models' `.dat` initialization files, though it has a much simpler structure. It also **MUST** be named `randomize.dat`, for this name is the only clue **initpackage** has to distinguish between creating a chain proposal or local models library.

All that needs to be entered into this file is a line of the form:

```
<class name> [<opt string arg1> <opt string arg2> ... <opt string argN>]
```

for each proposal class that will go in the library. <class name> must be a case-sensitive match to the actual C++ class name, and is the only required entry on the line. The class should also be stored in code files <class name>.h and <class name>.cxx.

Any additional arguments on the line will be placed in a single C++ string (including any separating whitespace), and passed to the class constructor. This is to allow the option of setting initialization parameters at the class construction stage. Therefore if optional arguments are included, the class must have a constructor which takes a single string argument. Otherwise, the constructor should contain no arguments.

For example, a `randomize.dat` file declaring two classes might contain:

```
MyProposal1  
MyProposal2    1.4773    on    false
```

In the code files, the constructor declarations corresponding to this would then be:

MyProposal1.h

```
class MyProposal1 : public RandomizerBase
{
    public:
        // ...
        MyProposal1();
        // ...
};
```

MyProposal2.h

```
class MyProposal2 : public RandomizerBase
{
    public:
        // ...
        MyProposal2(const string& initArgs);
        // ...
};
```

Writing a Chain Proposal Class

All user proposal classes must inherit from XSPEC's abstract class RandomizerBase, whose interface is defined in the file:

headas-<version>/Xspec/src/XSFit/Randomizer/RandomizerBase.h.

The proposal class must declare a constructor as described in the previous section, and which explicitly calls the RandomizerBase constructor, passing it a lower-case name string. This name will become the proposal identifier when making a selection using the `chain proposal` option during an XSPEC session. For example:

MyProposal1.cxx

```
MyProposal1::MyProposal1()
    : RandomizerBase("myprop1")
{
}
```

In XSPEC:

```
XSPEC12> chain proposal myprop1 [<optional initializing args>]
```

The RandomizerBase class contains 5 private virtual functions: `doRandomize`, `doInitializeLoad`, `doInitializeRun`, `doAcceptedRejected`, and `getCovariance`:

`doRandomize` is the only pure virtual function and therefore is the only one which **must** be overridden in the inheriting class. Its signature is:

```
virtual void doRandomize(RealArray& parameterValues, const Fit* fit)
```

where RealArray is a typedef for `std::valarray<double>` and is defined in `src/main/xsTypes.h`. This function is called by XSPEC for each chain iteration, and XSPEC passes in the current variable

model parameter values. The overridden `doRandomize` function performs the necessary parameter modifications and sends them back in the same array.

The function's second argument is a const pointer to XSPEC's global `Fit` class object. For those willing to further explore XSPEC's internals, this pointer provides access to various fit and chain information (such as covariance matrices), which may be necessary for the user's proposal scheme.

`doInitializeLoad` and `doInitializeRun` may be optionally overridden to perform initialization tasks at different stages during runtime. The default versions of these functions in `RandomizerBase` do nothing. `doInitializeLoad` is called by XSPEC immediately after the proposal is selected with the `chain proposal` command. Therefore one may find it useful to have this function process any additional arguments which may be entered on the command line:

```
chain proposal myprop [<optional initializing args>]
```

XSPEC automatically bundles [`<optional initializing args>`] into a single string and places it in the `m_initString` data member of `RandomizerBase`, to which the inheriting class has access. `doInitializeRun` is called once at the start of a chain run, and is useful for any tasks which must be performed one time immediately after the `chain run` command is entered.

`doAcceptedRejected` is called after each iteration in the chain. Its first argument is an array filled with the most recently attempted model parameter values, and its second argument is a boolean true or false indicating whether the attempt was accepted or rejected. The base class function does nothing with this, but an inherited class may want to use this information in an overridden function.

In its simplest form, a proposal class may be declared and defined as in the following example. This doesn't actually do anything since the `doRandomize` function is empty and the `parameterValues` array is left unchanged.

MyProposal.h

```
#ifndef MYPROPOSAL_H
#define MYPROPOSAL_H

#include <xsTypes.h>
#include <XSFit/Randomizer/RandomizerBase.h>

class Fit; // only a forward declaration is required for Fit

class MyProposal : public RandomizerBase
{
public:
    MyProposal();
    virtual ~MyProposal();
private:
    virtual void doRandomize(RealArray& parameterValues, const Fit* fit);
};

#endif
```

MyProposal.cxx

```
#include "MyProposal.h"
#include <XSFit/Fit/Fit.h>

MyProposal::MyProposal()
    : RandomizerBase("myprop")
{
}

MyProposal::~MyProposal()
{
}

void MyProposal::doRandomize(RealArray& parameterValues, const Fit* fit)
{
    // This is where the proposal algorithm should modify the variable
    // model parameters in the parameterValues array.
}
```

Building and Loading the Proposal Class Library

Once the randomize.dat file and the class(es) have been written, the library can be built and loaded during an XSPEC session using the same **initpackage** and **lmod** sequence that is used for local model libraries. To create a Makefile and build the library:

```
XSPEC12> initpackage <name> randomize.dat <directory>
```

To load the new proposal(s) into XSPEC:

```
XSPEC12> lmod <name> <directory>
```

where <name> is the name you choose for the package collection of proposal classes. It will also become the library file name. The only differences from the local models case are that here the initializer file **MUST** be named randomize.dat, and that the directory path to the proposal classes (either relative or absolute) must be provided on the command line. If this is left off XSPEC will default to looking in the directory set by LOCAL_MODEL_DIRECTORY, and these classes should **NOT** be stored in the same directory as local models. If the building and loading has successfully completed, you should see the proposal name (the same name string that was passed to the RandomizerBase constructor) appear in the chain proposal list displayed by typing `chain proposal` with no other arguments.