Gsim – Programming custom processing

07/10/09

Concepts

Gsim is C++ program which is based on Qt library and some other libraries for data analysis.

Each processing function is represented as an C++ object in the code. By default all processing functions are described in the files processing.h and processing.cpp.

The main features of the processing class:

- Each processing class should be derived from class BaseProcessingFunc
- The minimal implementation consists of implementation for virtual apply() and doInteractive() class members.
- doInteractive() function is responsible for the interactive input from the user inetrface and creates an 'options' string which contains all options provided by the user.
- apply() function should analyse the options string and apply the processing mathematics on the dataset.
- Data is available via 'data' pointer of DataStore class (described in base.h) within the BaseProcessingFunc derived class. The programmer also has an access to the plots ('plot2D' and 'plot3D' pointers) and even to the main program window via 'mainwindow' pointer.
- Once the class is created the processing object should be created and 'registered' in Gsim interface – it should be added to list 'availableProcList' in MainForm::initAvailableProcList member (file mainformsignals.cpp).

Quick step-by-step creation of a new processing function

Declaring a new class.

This part will show the steps needed to create 'addNoise' function.

First a new class must be declared in the 'processing.h' file:

```
class addNoiseProc: public BaseProcessingFunc
{
public:
addNoiseProc(DataStore* d, QString n=tr("addNoise"), QString o=tr("1.0"), QString c=tr("noise")) :
BaseProcessingFunc(d, n, o, c) { };
virtual void apply(int);
void doInteractive();
void plotsUpdate();
}
```

First line declares a new class 'addNoiseProc' which derived from the class BaseProcessingFunc.

The string:

```
addNoiseProc(DataStore* d, QString n=tr("addNoise"), QString o=tr("1.0"), QString c=tr("noise")) :
BaseProcessingFunc(d, n, o, c) { };
```

describes the constructor. The arguments of the constructor should be:

1. Pointer for the DataStore object.

2. QString which specifies the name, visible to User. In this case it's “addNoise”.
3. A default options string value. In our case “1.0”.
4. A command name for the command line interface. In this case 'noise'.

Then three public members are declared:

apply, do Interactive and plotsUpdate. The meaning of these functions will be described below.

Implementing 'apply' function.

All implementations described here and below can be found in the 'processing.cpp' file.

The 'apply' member is the main member for a new class – it provides the main mathematical operations on the dataset – either FID or spectrum.

For apply function it is:

```
void addNoiseProc::apply(int k)
{
bool ok;
double level=getDouble(0, ok);
if (!ok)
        return;
size_t np=data->get_odata(k)->cols();
size_t ni=data->get_odata(k)->rows();
if (!ni)
        ni=1;
List<double> realpart;
for (size_t i=0; i<ni; i++)
        for (size_t j=0; j<np; j++)
                realpart.push_back(data->get_odata(k,i,j).re);
double maxval=max(realpart);
double absNoise=maxval*level/100.0;
for (size_t i=0; i<ni; i++)
        for (size_t j=0; j<np; j++) {
                complex old= data->get_odata(k,i,j);
                old.re+=random(absNoise);
                old.im+=random(absNoise);
        data->set_odata(k,i,j,old);
        }
}
```

First string is a header:

```
void addNoiseProc::apply(int k)
```

It takes integer 'k' – an index of the data (spectrum or FID) with the DataStore object. In other words, it's a number of spectrum/FID in Gsim.

```
bool ok;
double level=getDouble(0, ok);
if (!ok)
        return;
```

This implementation uses 'getDouble' function to get the value for the 'noise level' provided by user. The first argument is a position of the requested option in the option string (it is the first option in the options string in this case). The second argument is true if extraction is successful. The function is doing nothing if option extraction has failed.

```
size_t np=data->get_odata(k)->cols();
size_t ni=data->get_odata(k)->rows();
if (!ni)
        ni=1;
```

This part determines the dimensionality of the data. Command data->get_odata(k) returns 'cmatrix' object (as described in Libcmatrix interface). The cols() and rows() members gives the dimensionality in direct and indirect dimensions respectively.

Extra care is taken to ensure that 'ni' is not equal to zero.

```
List<double> realpart;
for (size_t i=0; i<ni; i++)
	for (size_t j=0; j<np; j++)
		realpart.push_back(data->get_odata(k,i,j).re);
```

This part creates a list of doubles 'realpart' and fills it with the real part of the original data. The function data->get_odata(k,i,j) returns a complex (i,j) point of the matrix which corresponds to the data with index 'k' within DataStore. The list 'realpart' is needed to take the maximum value in the next line:

```
double maxval=max(realpart);
```

This 'maxval' is used to find the noise level expressed in absolute units:

```
double absNoise=maxval*level/100.0;
```

After that the noise can be added to every (complex) point of the original dataset within the cycle:

```
for (size_t i=0; i<ni; i++)
	for (size_t j=0; j<np; j++) {
		complex old= data->get_odata(k,i,j);
		old.re+=random(absNoise);
		old.im+=random(absNoise);
		data->set_odata(k,i,j,old);
}
```

The last command data->set_odata(k,i,j,old) sets the value 'old' for the datapoint (i,j) in the cmatrix with index 'k'.

Creating 'plotsUpdate'

This step is optional – BaseProcessingFunc has a default implementation which should be enough in many cases. For 'addNoise' functions it simply resets the window – it's either 'plot2D' or 'plot3D' depending on the dimensionality of the active dataset.

```
void addNoiseProc::plotsUpdate()
{
if (data->get_odata(data->getActiveCurve())->rows()<2)
	plot2D->cold_restart(data);
else
	plot3D->cold_restart(data);
}
```

The function data->getActiveCurve() returns the index of the data (FID or spectrum) which is active in the user inerface.

Creating 'doInteractive()'

```
void addNoiseProc::doInteractive()
{
bool ok;
double level = QInputDialog::getDouble(0, tr("Add noise"),tr("Noise level"),
1.0, 0, 99999999.0, 1, &ok);
if (!ok)
	throw Failed("Non-numerical value");
optionString=QString("%1").arg(level);
setOptions(optionString);
}
```

This function is responsible for the interactive input from the user. The parent class 'BaseProcessingFunc' has a default implementation which simply shows a window with 'No interactive input available' message.

For the 'addNoiseProc' the standard Qt dialog 'QInputDialog' is used. It allows to input a single float number. See a description in Qt documentation. Qt has a nice documentaion browser which is called Qt Assistant ('assistant' command in the Unix shell).

The value provide by the user than transferred into optionString:

```
optionString=QString("%1").arg(level);
```

And then processed for further access in

```
setOptions(optionString);
```

Registering a new processing object.

A new object should be created and 'registered' in 'availableProcList'. That makes it 'visible' for Gsim interface. To do that a following line should be added in MainForm::initAvailableProcList() member (file mainformsignals.cpp):

```
availableProcList.push_back(new addNoiseProc(data));
```

This causes several actions including:

1. Command line will accept 'noise' command.
2. 'Processing' menu will have a new 'addNoise' entry
3. An automatic icon will be generated which can be added to the toolbar using 'Edit processing list' dialog.

The 'addNoise' function is possibly an easiest example of the processing functions. Look into files 'processing.h' and 'processing.cpp' for more complex cases.

Accessing data

The spectral data is available for the processing class via 'data' pointer. This pointer has a type DataStore. It is declared in 'base.h' file.

Will be described later in detail...

Accessing plot windows

The plot windows are accessible via 'plot2D' (for 1D spectra/FID) or 'plot3D' pointers. They are declared in 'plot2dwidget.h' and 'plot3dwidget.h'.

Will be described later in detail...

Tools available within BaseProcessingFunc-derived class

QStringList 'option' contains a list of options already split by setOption command. To get a first option as a string do:

```
QString my_string=option.at(0);
```

```
optionError(QString error)
```

calls a pop-up windows with the error message 'error'.

```
double getDouble(size_t i, bool& ok);
```

Tries to get an option 'i' and convert it to double. Booling 'ok' contains equal to 'true' if succesful.

```
int getInt(size_t, bool&);
```

Similar to getDouble but for ineteger numbers. It also can convert a value like '2k' into 2048.