Mantid fitting code

`IFittingAlgorithm.h`

Base class for main fitting algorithms Takes care of workspace and function then creates domain handles.

Domains:

Fitting algorithm creates `Domain creators` which create the domain once they have all the information.

The domain requires the workspace and function types to be known. This means we create the domain once we are about run a fit.

The most common domain is `FitMW.h`, which is used for matrix workspaces and most standard functions.

4 methods in domain needs an override:

1) `declareDatasetProperties`
2) `createDomain` sets up domain values and also weights (which are typically 1/error)
3) `createOutputWorkspace` workspace created after the fit, containing the fit curves and errors.
4) `initFunction` Initializes the function and sets its workspace

Minimizer

Minimizer actually performs the fitting. We have:

1) Levenberg–Marquardt. THis is the main fitting minimizer. Its a mix of gradient descent and newtons method. We have the GSL version and our version with MD appended on the end. It is a trust region method which self regulates itself using a damping parameter. When the fit is far away from the minimum it behaves as a gradient descent algorithm, taking large steps in the direction of the gradient. When the fit is closer to a minimum it behaves as a Newtons method, leading to quadratic convergence. In the MD version we chunk the dataset to help us solve very large problems.

- Other than that we have lots of other GSL minimizers, e.g simplex (non gradient method) and various other gradient methods (conjugate gradient).

To interact with the GSL minimizers we have GSLFunction.h

Minimizer only knows about cost function and not the function.

Cost function:

Cost functions are a function of the parameters There are a few cost functions, but we usually use least squares (chi squared when we have Gaussian errors).

Aside from that we have:

- Unweighted least squares
- RWP - for exponential data

- Poisson

The cost functions have a base class in costFunction.h

Cost function class should be implemented such that we can find the cost function value, derivatives and hessian.

In cost functions we also handle constraints by adding a penalty factor to the cost function if a constraint is broken.

A few issues with penalty constraints (in my opinion):

- Gradient algorithm searches in regions that should be impossible according to the constraint. Can lead to difficulty converging.
- Constraint is approximate as its controlled by a penalty factor. If the penalty factor is too large the problem won't coverage.
- This is definitely an area for improvement. . . ..

My opinion: Don't use constraints too much. LMFit says the same thing it its manual.

Alternatively, we can use Lagrange constraints, but it leads to a saddle point problem, meaning we may not converge if we are using a gradient based minimizer.

A brief opinion on the minimizers:

- Levenberg - Probably the most reliable - quadratic converge if we are close to a minimum. Forming Hessian is costly for large problems.
- Simplex non gradient method, pretty robust but pretty slow as it just moves the solution about using geometric properties.
- Others probably not as tested, so read the GSL manual if you need to know about them.
- FABADA (no idea its some bayesian minimizer. I think this a Indirect specific thing.

IFunction

IFunction is the interface for all functions. This is a really big interface (far too big). Its aim is to give you function values if you supply it a domain. It has:

- Attributes - non fitting parameters, this is stored as variant as the value stored can be boolean, string, double ect. An example is `n` in a polynomial.
- function() returns values
- functionDeriv() optionally calculate the derivative. Usually people just use the numerical derivative. Neither are very "modern" as most other fitting libraries use the concept of dual numbers to calculate exact derivatives. See ceres-solver jets.
- Activate parameters lets us use different parameters in the cost function. This is used with the sigma parameter in the Gaussian function.

Outside of that you can look in the IFunction header for function declarations and descriptions.

There are no parameters in IFunction. These are stored in ParamFunction.

CompositeFunctions give us a way to add multiple IFunctions together.

Jacobian is the derivative of the function with respect to its parameters.

Convolution: Allows you to convolute functions together, this is used in Indirect to convolve peaks with a resolution function.

Convolution uses the GSL fast fourier transform approach, but also has a direct calculation approach if non-symmetric bounds, i.e for the integration range [A,B] if |A| and |B| are not equal (or even approximately equal) we use the direct method.

Resolution function is a tabulate function of the resolution. You just need to set X and Y.

MultiDomainFunction is a type of composite, where at each index (I) we have a function and an associated domain. This is how we do sequential and simultaneous fitting.

IFunctionWithLocation - splits background and peak functions.

Crystalfield

Defines a type of function. These differ from normal function as they use special parameters. Primarily the differences are: - Uses eigenvalue, - Number of parameters depends on other parameters. - uses multidomain functions as it fits to multiple spectra. - fits the physical properties - Uses lots of attributes (e.g temperature)

Its probably best to discuss any issues with Duc Le.

Fit Algorithm

This is the main fitting algorithm. Requires a function and an input workspace, the rest of the properties can be defaulted.

Function factory

Creates functions from an input string. An example use in python is

```
FunctionFactory.Instance().createInitialized("name=LinearBackground")
```

This is a singleton instance, like all other factories.

CalculateChiSquared

Calculates ChiSquared for input function and domain.

ProfileChiSquared1D

Makes slices in the chi squared profile to estimate the parameter probability distributions, e.g we can find the 1-sigma, 2-sigma and 3-sigma bounds. More accurate than covariance matrix, but takes a bit longer. Good for non-linear fits. It also will find asymmetric errors unlike the covariance matrix.

Good references:

- Numerical Recipes in C: The Art of Scientific Computing - https://www.cec.uchile.cl/cinetica/pcordero/M - CHAPTER 15
- Ceres-solver documentation - http://ceres-solver.org
- LMFit documentation - https://lmfit.github.io/lmfit-py/
- Minuint ROOT documentation (The fitting program used at Cern) https://root.cern.ch/download/minuit.pdf - This really goes into detail on obtaining errors on fit parameters. This is good documentation.
- Lecture notes on statistical data analysis - http://www.pp.rhul.ac.uk/~cowan/stat_course.html
- This book https://www.amazon.co.uk/Statistical-Analysis-Oxford-Science-Publications/dp/0198501552