The NoisyFunOpt C++ Library

Francesco Calcavecchia

November 20, 2017

NoisyFunOpt is a C++ library that contains simple tools for optimising (minimize) a noisy function, e.g. an integral computed with the Monte Carlo technique. It currently supports only the Conjugate Gradient algorithm, with some minor modifications made to handle the noise. More specifically, two function values `a` and `b` with associated standard deviations `da` and `db` are considered to be:

- $\texttt{a} > \texttt{b}$ iff $\texttt{a} - 2\texttt{da} > \texttt{b} + 2\texttt{db}$;
- $\texttt{a} < \texttt{b}$ iff $\texttt{a} + 2\texttt{da} < \texttt{b} - 2\texttt{db}$;
- $\texttt{a} = \texttt{b}$ otherwise.

The code has been developed using the standard C++11.

In the following we will present the classes made available by the library. At the beginning we will report the necessary `#include` call and the prototype of the class. The comment `TO DO` indicates that the method needs to be implemented (as in the case of a pure virtual class).

1 NoisyFunction

```
\\ #include "NoisyFunction.hpp"
class NoisyFunction
{
   NoisyFunction(int);
   virtual ~NoisyFunction();

   int getNDim();

   virtual void f(const double *, double &, double &) = 0;  \\ TO DO
};
```

A `NoisyFunction` implements a function $f : \mathbb{R}^n \to \mathbb{R} \times \mathbb{R}$ which takes a multidimensional vector as input, and returns a scalar value with an associated error bar.

- `NoisyFunction(int ndim)`: The constructor. `ndim` must contains the value of n, i.e. the number of dimensions of the input;
- `~NoisyFunction()`: Virtual destructor. It does nothing;
- `getNDim()`: It returns the `ndim` provided with the constructor;
- `f(const double *x, double &val, double &dval)`: The implementation of f. It takes an array `x` as input and returns in `val` the value of the function and in `dval` the error bar associated with it.

2 NoisyFunctionWithGradient

```
\\ #include "NoisyFunction.hpp"
class NoisyFunctionWithGradient: public NoisyFunction
{
   NoisyFunctionWithGradient(int);
   virtual ~NoisyFunctionWithGradient();

   virtual void grad(const double *, double *, double *) = 0; // TO DO
};
```

`NoisyFunctionWithGradient` is a child class of `NoisyFunction` (therefore the implementation of the method `f` is required), and its aim is to generalize the class to the case in which the gradient of f is known.

- `NoisyFunctionWithGradient(int ndim)`: The constructor. As in `NoisyFunction`;
- `~NoisyFunctionWithGradient()`: The destructor. It does nothing;
- `grad(const double *x, double *grad, double *dgrad)`: The gradient of f. It takes `x` as input, and returns the gradient in `grad` and the corresponding errors in `dgrad`.

3 NFM

```
// #include "NoisyFunMin.hpp"
class NFM{
   NFM(NoisyFunction *);
   virtual ~NFM();

   void setX(const double *);
   void setGradientTargetFun(NoisyFunctionWithGradient *);
   void setEpsTargetFun(double &);
   void setEpsX(double &);
```

```
    int getNDim();
    double getX(const int &);
    void getX(double * x);
    double getF();
    double getDf();
    NoisyFunctionWithGradient* getGradientTargetFun();
    double getEpsTargetFun();
    double getEpsX();

    virtual void findMin() = 0; // TO DO
};
```

`NFM` is an interface for a generic minimisation method. Any actual implementation of this interface will have to specify the `findMin` method.

The minimisation process will start from a point `x`, set with the method `setX`, and wil continue until the minimum has been found. At the end of this process, `x` will have to be in this minimum.

- `NFM(NoisyFunction *f)`: The constructor. It takes as input a target function `f`;
- `~NFM()`: The destructor. It does nothing;
- `setX(const double *x)`: Set the starting point of the minimisator. By default it is set equal to 0;
- `setGradientTargetFun(NoisyFunctionWithGradient * grad)`: Set a target function with gradient;
- `setEpsTargetFun(double &eps)`: Set the value of the minimum change in the target function value that will make continue the minimisation process. This parameter might be irrelevant, depending on the actual implementation of the minimisation algorithm;
- `setEpsX(double &eps)`: Set the value of the minimum change in the minimum point that will make continue the minimisation process. This parameter might be irrelevant, depending on the actual implementation of the minimisation algorithm;
- `getNDim()`: It returns the number of dimensions of the space in which the function is minimised;
- `getX(const int &i)`: It returns the `i`-th coordinate of the actual position of `x`, which will coincide with the minimum after the minimisation process;

- `getF()`: It returns the value of f in `x`;
- `getDf()`: It returns the error bar associated with the value of f(`x`);
- `getGradientTargetFun()`: It returns the pointer to the target function with gradient, if there is one. Otherwise returns `0`;
- `getEpsTargetFun()`: It returns the value set with `setEpsTargetFun`, which is 0 by default;
- `getEpsX()`: It returns the value set with `setEpsX`, which is 0 by default;
- `findMin()`: The minimisation subroutine. At the end of this process, `x` is supposed to be in the minimum of the target function. Therefore, after calling this method, it will be possible to know where the minimum is by using the method `getX`, while its corresponding value and error bar will be directly accessible with `getF` and `getDf`.

4 ConjGrad

```
\\ #include "ConjGrad.hpp"
class ConjGrad: public NFM
{
   ConjGrad(NoisyFunctionWithGradient *);
   ~ConjGrad();
};
```

Implementation of the Conjugate Gradient algorithm, using the *Para-Gold Search* method for the one-dimensional minimisation (see https://publications.ub.uni-mainz.de/theses/volltexte/2014/3805/pdf/3805.pdf, page 61).

- `ConjGrad(NoisyFunctionWithGradient *f)`: The constructor. It takes as input the target function `f`;
- `~ConjGrad()`: The destructor. It does nothing;