Help

```
#ifndef _OPTIM_HPP
#define _OPTIM_HPP

#include <string>
#include "pnl/pnl_matrix.h"
#include "pnl/pnl_vector.h"

#include "
href../../../../common/math/mcam/src/Martingale_h_src.pdfMartingale.hpp"
#include "
href../../../../common/math/mcam/src/chaos_h_src.pdfchaos.hpp"
#include "
href../../../../common/math/mcam/src/Option_h_src.pdfOption.hpp"
#include "
href../../../../common/math/mcam/src/DupireModel_h_src.pdfModel.hpp"

namespace mcam {

//! Return signal for the step method
enum {Accept = 0, Reject = 1, Stop = 2};

/// @class StepMethod
/// @brief Describe how to go from one iterate to the next
class StepMethod
{

public:
    virtual int Step(double cost, double &step, double &best_cost) = 0;
    virtual ~StepMethod() { }
};

class LineSearchStep: public StepMethod
{
public:
    virtual int Step(double cost, double &step, double &best_cost);
};

class ExploreStep: public StepMethod
{
```

```
public:
    virtual int Step(double cost, double &step, double &best_cost);
};

class Optim
{
public:
    Model *mod;
    Option *opt;
    Martingale *mart;
    int iterationsSA;
    int iterationsSAA;
    int iterationsMC;
    int block_SA;
    double gamma;

    Optim();
    Optim(Model *mod, Option *opt, Martingale *mart, const Param &P);
    virtual ~Optim() { };
    virtual void print() const ;
    virtual double dfcost(bool ComputeDf = true) = 0;
    int saGainVector(PnlRng_Workspace &rng, PnlVect *alpha, FILE *outfile = NULL
    int sa(PnlRng_Workspace &rng, PnlVect *alpha, FILE *outfile = NULL, bool wit
    void saa(PnlRng_Workspace &rng, PnlVect *alpha, double &prix, double &var, S
    void saa_average(PnlRng_Workspace &rng, PnlVect *alpha, StepMethod *stepMeth
    void EGradCost(PnlRng_Workspace &rng, const PnlVect *alpha, double &E, doubl
    double computeEuropenPrice(PnlRng_Workspace &rng);


protected:
    bool useMPI;
    PnlVect_Workspace grad_m;
    std::string type_str;
    virtual void truncate(PnlVect *alpha, double &diameter, int &ntrunc) { };
    double gain_sa(int l, double gamma, bool with_averaging) const;
    void computeFullGradient(PnlVect *p_grad, double & p_cost, double &p_var, Pn
    void computeFullGradientMPI(PnlVect *p_grad, double & p_cost, double &p_var,
    int getIterationsSAA() const;
    double computeMoneyness(PnlRng_Workspace &rng);
    PnlMat** sampleBrownianIncrementsInTheMoney(int iterations, PnlRng_Workspace
};
```

```
Optim *instantiate_optim(Model *mod, Option *opt, Martingale *mart, const Param

class MaxCost: public Optim
{
public:
    MaxCost();
    MaxCost(Model *mod, Option *opt, Martingale *mart, const Param &P);
    virtual ~MaxCost();
    virtual double dfcost(bool ComputeDf = true);
    virtual void print() const;

protected:
    virtual void truncate(PnlVect *alpha, double &diameter, int &ntrunc);
};

}
#endif /* _OPTIM_HPP */
```