Help

```
#include <iostream>
#include <cmath>
#include <cstdlib>
#include <cstring>
#include <algorithm>

#include "
href../../../../common/math/mcam/src/basket_h_src.pdfbasket.hpp"
#include "
href../../../../common/math/jlparser/include/jlparser/parser_h_src.pdfjlparser/p
#include "pnl/pnl_vector.h"
#include "pnl/pnl_matrix.h"

//
// Basket options
//

double mcam::BasketOption::payoff(const PnlMat *S, int t)
{
    double sum = 0.0;
    PnlVect St = pnl_vect_wrap_mat_row(S, t);
    sum = pnl_vect_scalar_prod(lambda, &St);
    return std::max(sum - K, 0.0);
}

mcam::BasketOption::BasketOption ()
{
    lambda = NULL;
}

mcam::BasketOption::BasketOption (const Param &P)
    : Option(P)
{
    label = "basket";

    P.extract("payoff coefficients", lambda, size);
    P.extract("strike", K);
}
```

```
void mcam::BasketOption::print() const
{
    std::cout << std::endl;
    std::cout << "****************************************" << std::endl;
    std::cout << "**** Basket Option Characteristics ****" << std::endl;
    mcam::Option::print();
    std::cout << " payoff coefficients : ";
    pnl_vect_print_asrow(lambda);
    std::cout << " strike : " << K << std::endl;
    std::cout << "****************************************" << std::endl << std::
}

mcam::BasketOption::~BasketOption ()
{
    if (lambda)
        pnl_vect_free(&lambda);
}

//
// Geometric options
//

double mcam::GeometricPutOption::payoff(const PnlMat *S, int t)
{
    PnlVect St = pnl_vect_wrap_mat_row(S, t);
    double prod = pow(pnl_vect_prod(&St), 1. / size);
    return std::max(K - prod, 0.0);
}

mcam::GeometricPutOption::GeometricPutOption () : sigma(NULL) { }

mcam::GeometricPutOption::GeometricPutOption (const Param &P)
    : Option(P)
{
    label = "geometric";
    P.extract("strike", K);
    P.extract("model size", size);
    P.extract("volatility", sigma, size);
}

void mcam::GeometricPutOption::print() const
```

```
{
    std::cout << std::endl;
    std::cout << "*************************************" << std::endl;
    std::cout << "**** GeometricPut Option Characteristics ****" << std::endl;
    mcam::Option::print();
    std::cout << " strike : " << K << std::endl;
    std::cout << "*************************************" << std::endl << std::
}

mcam::GeometricPutOption::~GeometricPutOption ()
{
    if (sigma != NULL) pnl_vect_free(&sigma);
}

double mcam::GeometricCallOption::payoff(const PnlMat *S, int t)
{
    PnlVect St = pnl_vect_wrap_mat_row(S, t);
    double prod = pow(pnl_vect_prod(&St), 1. / size);
    return std::max(prod - K, 0.0);
}

mcam::GeometricCallOption::GeometricCallOption () : sigma(NULL) { }

mcam::GeometricCallOption::GeometricCallOption (const Param &P)
    : Option(P)
{
    label = "geometric";
    P.extract("strike", K);
    P.extract("model size", size);
    P.extract("volatility", sigma, size);
}

void mcam::GeometricCallOption::print() const
{
    std::cout << std::endl;
    std::cout << "*************************************" << std::endl;
    std::cout << "**** GeometricCall Option Characteristics ****" << std::endl;
    mcam::Option::print();
    std::cout << " strike : " << K << std::endl;
    std::cout << "*************************************" << std::endl << std::
}
```

```
mcam::GeometricCallOption::~GeometricCallOption ()
{
    if (sigma != NULL) pnl_vect_free(&sigma);
}

//
// Best Of options
//

double mcam::BestOfOption::payoff(const PnlMat *S, int t)
{
    PnlVect St = pnl_vect_wrap_mat_row(S, t);
    double Smax = 0.0;
    for (int i = 0; i < size; i++)
    {
        double tmp = GET(lambda, i) * GET(&St, i);
        if (tmp > Smax)
            Smax = tmp;
    }

    return std::max(Smax - K, 0.0);
}

mcam::BestOfOption::BestOfOption () { }

mcam::BestOfOption::BestOfOption (const Param &P)
    : Option(P)
{
    label = "bestof";
    P.extract("payoff coefficients", lambda, size);
    P.extract("strike", K);
}

void mcam::BestOfOption::print() const
{
    std::cout << std::endl;
    std::cout << "****************************************" << std::endl;
    std::cout << "**** Best Of Option Characteristics ****" << std::endl;
    mcam::Option::print();
    std::cout << " payoff coefficients : ";
```

```
    pnl_vect_print_asrow(lambda);
    std::cout << " strike : " << K << std::endl;
    std::cout << "**************************************" << std::endl << std::
}

mcam::BestOfOption::~BestOfOption () { }

double mcam::WorstOfOption::payoff(const PnlMat *S, int t)
{
    PnlVect St = pnl_vect_wrap_mat_row(S, t);
    double Smin  = DBL_MAX;
    for (int i = 0; i < size; i++)
    {
        double tmp = GET(lambda, i) * GET(&St, i);
        if (tmp < Smin)
            Smin = tmp;
    }

    return std::max(K - Smin, 0.0);
}

mcam::WorstOfOption::WorstOfOption () { }

mcam::WorstOfOption::WorstOfOption (const Param &P)
    : Option(P)
{
    label = "worstof";
    P.extract("payoff coefficients", lambda, size);
    P.extract("strike", K);
}

void mcam::WorstOfOption::print() const
{
    std::cout << std::endl;
    std::cout << "**************************************" << std::endl;
    std::cout << "**** worst Of Option Characteristics ****" << std::endl;
    mcam::Option::print();
    std::cout << " payoff coefficients : ";
    pnl_vect_print_asrow(lambda);
    std::cout << " strike : " << K << std::endl;
    std::cout << "**************************************" << std::endl << std::
```

```
}

mcam::WorstOfOption::~WorstOfOption () { }
```