Help

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

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

//
// Asian options
//

/*
 * (lambda . 1/t \ int_0^t S_u du - K)_+
 *
 * We use a trapezoidal rule to approximate the integral
 */
double mcam::AsianOption::payoff(const PnlMat *S, int t)
{
    if (t == 0)
    {
        PnlVect S0 = pnl_vect_wrap_mat_row(S, 0);
        return fmax(pnl_vect_scalar_prod(&S0, lambda) - K, 0.);
    }
    PnlMat subS = pnl_mat_wrap_mat_rows(S, 0, t);
    PnlVect subOnes = pnl_vect_wrap_subvect(ones, 0, t + 1);
    double sum = pnl_mat_scalar_prod(&subS, &subOnes, lambda);
    PnlVect S0 = pnl_vect_wrap_mat_row(S, 0);
    PnlVect St = pnl_vect_wrap_mat_row(S, t);
    sum -= 0.5 * (pnl_vect_scalar_prod(&S0, lambda) + pnl_vect_scalar_prod(&St,
    sum /= t;
    return fmax(sum - K, 0.0);
}
```

```
mcam::AsianOption::AsianOption ()
{
    lambda = NULL;
    ones = NULL;
}

mcam::AsianOption::AsianOption (const Param &P)
    : Option(P)
{
    label = "asian";

    P.extract("payoff coefficients", lambda, size);
    P.extract("strike", K);
    ones = pnl_vect_create_from_scalar( subdates + 1, 1.);
}

void mcam::AsianOption::print() const
{
    std::cout << std::endl;
    std::cout << "**************************************" << std::endl;
    std::cout << "**** Asian 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::AsianOption::~AsianOption ()
{
    if (lambda)
        pnl_vect_free(&lambda);
    if (ones)
        pnl_vect_free(&ones);
}

//
// Moving average options
//

/*
```

```
 * (lambda . 1/t \ int_0^t S_u du - K)_+
 *
 * We use a trapezoidal rule to approximate the integral
 */
double mcam::MovingAverageOption::payoff(const PnlMat *S, int t)
{
    if (t < window + delay) return 0;
    PnlMat subS = pnl_mat_wrap_mat_rows(S, t - window - delay + 1, t - delay);
    double sum = pnl_mat_scalar_prod(&subS, ones, lambda);
    PnlVect Send = pnl_vect_wrap_mat_row(S, t);
    double Sfin = pnl_vect_scalar_prod(&Send, lambda);
    sum /= window;
    return fmax(Sfin - sum, 0.0);
}

mcam::MovingAverageOption::MovingAverageOption ()
{
    lambda = NULL;
    ones = NULL;
    window = 0;
    delay = 0;
}

mcam::MovingAverageOption::MovingAverageOption (const Param &P)
    : Option(P)
{
    label = "moving average";
    delay = 0;

    P.extract("payoff coefficients", lambda, size);
    P.extract("window", window);
    P.extract("delay", delay, true);
    ones = pnl_vect_create_from_scalar(window , 1.);
}

void mcam::MovingAverageOption::print() const
{
    std::cout << std::endl;
    std::cout << "************************************" << std::endl;
    std::cout << "**** MovingAverage Option Characteristics ****" << std::endl;
    mcam::Option::print();
```

```
    std::cout << " payoff coefficients : ";
    pnl_vect_print_asrow(lambda);
    std::cout << " window : " << window << std::endl;
    std::cout << " delay : " << delay << std::endl;
    std::cout << "************************************" << std::endl << std::
}

mcam::MovingAverageOption::~MovingAverageOption ()
{
    if (lambda)
        pnl_vect_free(&lambda);
    if (ones)
        pnl_vect_free(&ones);
}
```