Help

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

using namespace std;

#include "math/ImportanceSampling_jl/src/BasketOption.hpp"
#include "math/ImportanceSampling_jl/src/parser.hpp"
#include "pnl/pnl_vector.h"

//
// Basket options
//

double BasketOption::payoff(const PnlMat *path_val)
{
  double sum = 0.0;
  PnlVect final = pnl_vect_wrap_mat_row(path_val, path_val->m - 1);
  sum = pnl_vect_scalar_prod(lambda, &final);
  return max(sum - K, 0.0);
}

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


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

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

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

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

//
// Performance options
//

double PerformanceOption::payoff(const PnlMat *path_val)
{
  int i;
  double panier, panier_prev, sum;
  int nTimeSteps = path_val->m - 1;

  PnlVect S0 = pnl_vect_wrap_mat_row(path_val, 0);
  panier_prev = pnl_vect_scalar_prod(lambda, &S0);
  sum = 0.;
  for (i = 1 ; i <= nTimeSteps ; i++)
    {
      PnlVect St = pnl_vect_wrap_mat_row(path_val, i);
      panier = pnl_vect_scalar_prod(lambda, &St);
      sum += max(panier / panier_prev - 1., 0.);
      panier_prev = panier;
    }


  return 1. + sum;
}

PerformanceOption::PerformanceOption() { }
```

```
PerformanceOption::PerformanceOption(const Param &P) :
  BaseOption(P)
{
  label = "basket";
  K = 0.1;

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


void PerformanceOption::print() const
{
  cout << "**** Performance Option Characteristics ****" << endl;
  BaseOption::print();
  cout << " payoff coefficients : ";
  pnl_vect_print_asrow(lambda);
  cout << " strike : " << K << endl;
}

PerformanceOption::~PerformanceOption() { }

//
// Best Of options
//

double BestOfOption::payoff(const PnlMat *path_val)
{
  PnlVect final = pnl_vect_wrap_mat_row(path_val, path_val->m - 1);
  double Smax = 0.0;
  for (int i = 0 ; i < size ; i++)
    {
      double tmp = GET(lambda, i) * GET(&final, i);
      if (tmp > Smax)
        Smax = tmp;
    }

  return max(Smax - K, 0.0);
}
```

```
BestOfOption::BestOfOption() { }

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


void BestOfOption::print() const
{
  cout << "**** Best Of Option Characteristics ****" << endl;
  BaseOption::print();
  cout << " payoff coefficients : ";
  pnl_vect_print_asrow(lambda);
  cout << " strike : " << K << endl;
}

BestOfOption::~BestOfOption() { }


double WorstOfOption::payoff(const PnlMat *path_val)
{
  PnlVect final = pnl_vect_wrap_mat_row(path_val, path_val->m - 1);
  double Smin  = DBL_MAX;
  for (int i = 0 ; i < size ; i++)
    {
      double tmp = GET(lambda, i) * GET(&final, i);
      if (tmp < Smin)
        Smin = tmp;
    }

  return max(K - Smin, 0.0);
}


WorstOfOption::WorstOfOption() { }
```

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


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

WorstOfOption::~WorstOfOption() { }


//
// Geoemtric Basket options
//

double GeometricBasketOption::payoff(const PnlMat *path_val)
{
  PnlVect final = pnl_vect_wrap_mat_row(path_val, path_val->m - 1);
  double prod = pow(pnl_vect_prod(&final), 1. / size);
  if (is_call_)
    return max(prod - K_, 0.0);
  else
    return max(K_ - prod, 0.0);
}

GeometricBasketOption::GeometricBasketOption() { }


GeometricBasketOption::GeometricBasketOption(const Param &P, bool is_call) :
  BaseOption(P)
{
```

```
  label = "basket";
  is_call_ = is_call;
  P.extract("strike", K_);
}


void GeometricBasketOption::print() const
{
  cout << "**** GeometricBasket Option Characteristics ****" << endl;
  BaseOption::print();
  cout << " strike : " << K_ << endl;
}

GeometricBasketOption::~GeometricBasketOption() { }
```