Help

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

using namespace std;

#include "
href../../../../common/math/ImportanceSampling_jl/src/KouModel_h_src.pdfmath/Imp
#include "
href../../../../common/math/jlparser/include/jlparser/parser_h_src.pdfjlparser/p
#include "pnl/pnl_matrix.h"

KouModel::KouModel() : JumpModel() { }

KouModel::~KouModel()
{
  if (lambda_p) pnl_vect_free(&lambda_p);
  if (lambda_m) pnl_vect_free(&lambda_m);
  if (proba) pnl_vect_free(&proba);

}

KouModel::KouModel(const Param &P)
  : JumpModel(P)
{
  P.extract("lambda_p", lambda_p, poissonSize);
  P.extract("lambda_m", lambda_m, poissonSize);
  P.extract("positive jump probability", proba, poissonSize);
  /*
   * Compute the drift part of the jump diffusion process
   */
  for (int i = 0 ; i < size ; i++)
    {
      const double sigma_i = GET(sigma, i);
      const double l_p = GET(lambda_p, i);
      const double l_m = GET(lambda_m, i);
      const double p = GET(proba, i);
      LET(levyDrift, i) = interest - GET(dividend, i) - SQR(sigma_i) / 2.0
                          - GET(lambda, i) * (p * l_p / (l_p - 1) + (1 - p) * l
```

```
    }
  if (poissonSize == size + 1)
    {
      const double l_p = GET(lambda_p, size);
      const double l_m = GET(lambda_m, size);
      const double p = GET(proba, size);
      const double tmp = GET(lambda, size) * (p * l_p / (l_p - 1) + (1 - p) * l_
      pnl_vect_minus_double(levyDrift, tmp);
    }
}

/**
 * Computes one path of the model assuming the Brownian increments have
 * already been drawn in Gincr_drift
 * @param rng: PnlRng
 * @param mu is the vector of intensities. They are supposed to be constant
 * for every dimension
 */
void KouModel::pathMu_aux(PnlRng *rng, const PnlVect *mu)
{
  for (int i = 0 ; i < nTimeSteps ; i++)
    {
      for (int j = 0 ; j < poissonSize ; j++)
        {
          const double l_p = GET(lambda_p, j);
          const double l_m = GET(lambda_m, j);
          const double p = GET(proba, j);
          // Compute number of jumps
          int n = pnl_rng_poisson(GET(mu, j) * dt, rng);
          MLET(poissonMat, i, j) = n;
          // Compute jumps
          double jump = 1.;
          for (int k = 0 ; k < n ; ++k)
            {
              jump *= exp(pnl_rng_dblexp(l_p, l_m, p, rng));
            }

          MLET(jumpsMat, i, j) = jump;
        }
    }
  path();
```

```
}

/**
 * Auxiliary function.
 *
 * The Gaussian part has to be already drawn and available through
 * mod->Gincr_drift
 *
 * Compute a path of the KouModel model with piecewise constant intensities
 * (given by mu) for the poissonMat processes.
 *
 * @param rng a random number generator
 * @param mu jump intensity (one intensity per time time step)
 */
void KouModel::pathMuFull_aux(PnlRng *rng, const PnlVect *mu)
{
  PnlMat intensity = pnl_mat_wrap_array(mu->array, nTimeSteps, poissonSize);

  for (int i = 0 ; i < nTimeSteps ; i++)
    {
      for (int j = 0 ; j < poissonSize ; j++)
        {
          const double l_p = GET(lambda_p, j);
          const double l_m = GET(lambda_m, j);
          const double p = GET(proba, j);
          // Compute number of jumps
          int n = pnl_rng_poisson(MGET((&intensity), i, j), rng);
          MLET(poissonMat, i, j) = n;
          // Compute jumps
          double jump = 1.;
          for (int k = 0 ; k < n ; ++k)
            {
              jump *= exp(pnl_rng_dblexp(l_p, l_m, p, rng));
            }

          MLET(jumpsMat, i, j) = jump;
        }
    }
  path();
}
```

```
void KouModel::print() const
{
  cout << "**** KouModel Model Characteristics ****" << endl;
  cout << " lambda_plus : ";
  pnl_vect_print_asrow(lambda_p);
  cout << " lambda_minus : ";
  pnl_vect_print_asrow(lambda_m);
  cout << " proba of positive jumps : ";
  pnl_vect_print_asrow(proba);
  JumpModel::print();
}
```