Help

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

using namespace std;

#include "math/ImportanceSampling_jl/src/BNSModel.hpp"
#include "math/ImportanceSampling_jl/src/parser.hpp"
#include "pnl/pnl_matrix.h"


BNSModel::BNSModel() : JumpModel() { }

BNSModel::~BNSModel()
{
  if (kappa) pnl_vect_free(&kappa);
  if (beta) pnl_vect_free(&beta);
  if (leverage) pnl_vect_free(&leverage);
  if (Vol) pnl_vect_free(&Vol);
}

BNSModel::BNSModel(const Param &P)
  : JumpModel(P)
{
  P.extract("kappa", kappa, poissonSize);
  P.extract("jump size parameter", beta, poissonSize);
  P.extract("leverage parameter", leverage, poissonSize);
  Vol = pnl_vect_create(size);
  /*
   * The true jump intensity is kappa * lambda
   */
  pnl_vect_mult_vect_term(lambda, kappa);
  for (int i = 0 ; i < nTimeSteps ; i++)
    {
      PnlVect part = pnl_vect_wrap_subvect(lambdaFull, i * poissonSize, poissonS
      pnl_vect_clone(&part, lambda);
    }
  pnl_vect_mult_double(lambdaFull, maturity / nTimeSteps);
```

```
  /*
   * Compute the drift part of the jump diffusion process
   */
  for (int i = 0 ; i < size ; i++)
    {
      int i_poisson = (poissonSize == 1 ? 0 : i);
      LET(levyDrift, i) = interest - GET(dividend, i) - GET(lambda, i_poisson)
                          * GET(leverage, i) / (GET(beta, i_poisson) - GET(leve
    }
  if (poissonSize == size + 1)
    {
      pnl_vect_minus_double(levyDrift, GET(lambda, size) * GET(leverage, size)
                            / (GET(beta, size) - GET(leverage, size)));
    }
}

/**
 * Computes one path of the model
 */
void BNSModel::path()
{
  // Time 0
  pnl_mat_set_row(pathMatrix, init, 0);

  pnl_vect_clone(Vol, sigma);
  for (int j = 1 ; j <= nTimeSteps ; j++)
    {
      for (int i = 0 ; i < size ; i++)
        {
          int i_poisson = (poissonSize == 1 ? 0 : i);
          double delta_logX, deltaV;
          delta_logX = (GET(levyDrift, i) - GET(Vol, i) / 2) * dt + sqrt(GET(Vol
                       * MGET(Gincr_drift, j - 1, i) + GET(leverage, i) * MGET(j

          deltaV = (-GET(kappa, i) * GET(Vol, i)) * dt + MGET(jumpsMat, j - 1, i
          if (poissonSize == size + 1)
            {
              delta_logX = GET(leverage, i) * MGET(jumpsMat, j - 1, size);
              deltaV += MGET(jumpsMat, j - 1, size);
            }
```

```
          MLET(pathMatrix, j, i) = MGET(pathMatrix, j - 1, i) * exp(delta_logX);
          LET(Vol, i) += deltaV;
        }
    }
}

/**
 * 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 BNSModel::pathMu_aux(PnlRng *rng, const PnlVect *mu)
{
  for (int i = 0 ; i < nTimeSteps ; i++)
    {
      for (int j = 0 ; j < poissonSize ; j++)
        {
          const double beta_j = GET(beta, j);
          // Compute number of jumps
          int n = pnl_rng_poisson(GET(mu, j) * dt, rng);
          MLET(poissonMat, i, j) = n;
          // Compute jumps
          double jump = 0.;
          for (int k = 0 ; k < n ; k++)
            {
              jump += pnl_rng_exp(beta_j, 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 BNSModel 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 BNSModel::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 beta_j = GET(beta, j);
          // Compute number of jumps
          int n = pnl_rng_poisson(MGET(&intensity, i, j), rng);
          MLET(poissonMat, i, j) = n;
          // Compute jumps
          double jump = 0.;
          for (int k = 0 ; k < n ; k++)
            {
              jump += pnl_rng_exp(beta_j, rng);
            }

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

void BNSModel::print() const
{
  cout << "**** BNSModel Model Characteristics ****" << endl;
  cout << " jump size parameter : ";
  pnl_vect_print_asrow(beta);
  cout << " kappa : ";
  pnl_vect_print_asrow(kappa);
  cout << " leverage parameter : ";
  pnl_vect_print_asrow(leverage);
```

```
  JumpModel::print();
}
```