Help

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

using namespace std;

#include "
href../../../../common/math/mcam/src/HestonModel_h_src.pdfHestonModel.hpp"
#include "
href../../../../common/math/jlparser/include/jlparser/parser_h_src.pdfjlparser/p

mcam::HestonModel::HestonModel()
    : reverting_rate(NULL)
    , long_run_var(NULL)
{
}

mcam::HestonModel::~HestonModel()
{
    pnl_vect_free(&voVol);
    pnl_vect_free(&sigma0);
    pnl_vect_free(&sigmaVector);
    pnl_vect_free(&reverting_rate);
    pnl_vect_free(&long_run_var);
    pnl_mat_free(&covChol);
}

// Do not call the Model constructor because the brownian size is twice the
// size of the model and many objects have to be updated
mcam::HestonModel::HestonModel(const Param& P)
    : Model(P)
{
     P.extract("initial volatility", sigma0, size);
     P.extract("volatility of volatility", voVol, size);
     P.extract("asset vol correlation", gamma);
     brownianSize = 2 * size;
     sigmaVector = pnl_vect_new();
     P.extract("reverting rate", reverting_rate, size);
     P.extract("long run variance", long_run_var, size);
```

```
    P.extract("correlation", rho);
    /* test the value of the correlation */
    if (rho > 1 || rho < -1.0 / (size - 1))
    {
        perror("correlation out of range");
        exit(1);
    }
    computeCovChol();
}

int mcam::HestonModel::computeCovChol()
{
  const double cross_corel = gamma * rho;
  const double correl_W = gamma * gamma * rho;
  covChol = pnl_mat_create(brownianSize, brownianSize);
  // Compute the overall correlation matrix
  // First diagonal block
  for (int i = 0; i < size; i++)
    {
      MLET(covChol, i, i) = 1.;
      for (int j = 0; j < i ; j++)
        {
          MLET(covChol, i, j) = rho;
          MLET(covChol, j, i) = rho;
        }
    }
  // Second diagonal block
  for (int i = size; i < brownianSize; i++)
    {
      MLET(covChol, i, i) = 1.;
      for (int j = size; j < i; j++)
        {
          MLET(covChol, i, j) = correl_W;
          MLET(covChol, j, i) = correl_W;
        }
    }
  // Extra diagonal block
  for (int i = size; i < brownianSize; i++)
    {
      for (int j = 0; j < size; j++)
        {
```

```
          MLET(covChol, i, j) = cross_corel;
          MLET(covChol, j, i) = cross_corel;
        }
    }

  // Set the diagnonal terms of the extra diagonal blocks
  for (int i = 0; i < size; i++)
    {
      MLET(covChol, i, i + size) = gamma;
      MLET(covChol, i + size, i) = gamma;
    }
  return pnl_mat_chol(covChol);
}
void mcam::HestonModel::path(const PnlMat *G)
{
    PnlMat *S = path_m();
    PnlMat *regressor = regressor_m();
    PnlMat *w = work();
    // Time 0
    pnl_mat_resize(S, subdates + 1, size);
    pnl_mat_resize(regressor, subdates + 1, 2 * size);
    pnl_mat_set_row(S, spot, 0);
    pnl_mat_resize(w, subdates, size);
    pnl_mat_dgemm('N', 'T', sqrt_dt, G, covChol, 0., w);
    pnl_vect_clone(sigmaVector, sigma0);

    int j;
    double tj;
    for (j = 1, tj = dt; j <= subdates; j++, tj += dt)
    {
        for (int i = 0; i < size; i++)
        {
            double sigma_i_plus = max(GET(sigmaVector, i), 0.);
            MLET(S, j, i) = MGET(S, j - 1, i) * (1. + (r - GET(dividend, i)) * d
            LET(sigmaVector, i) += GET(reverting_rate, i) * (GET(long_run_var, i
            MLET(regressor, j, i) = MGET(S, j, i);
            MLET(regressor, j, size + i) = GET(sigmaVector, i);
        }
    }
}
```

```
PnlMat* mcam::HestonModel::getRegressor() const
{
    return this->regressor_m();
}

PnlVect * mcam::HestonModel::getMin() const
{
    PnlVect *min_v = pnl_vect_create (2 * size);
    for (int i = 0; i < size; i++)
    {
        double sig = 2 * sqrt(GET(sigma0, i));
        LET(min_v, i) = GET(spot,i) * exp (MIN(r - sig*sig/2, 0) * T - sig * 4.0
        LET(min_v, size + i) = 0;
    }
    return min_v;
}

PnlVect * mcam::HestonModel::getMax() const
{
    PnlVect *max_v = pnl_vect_create (2 * size);
    for (int i = 0; i < size; i++)
    {
        double sig = 2 * sqrt(GET(sigma0, i));
        LET(max_v, i) = GET(spot,i) * exp (MAX(r - sig*sig/2, 0) * T + sig * 4.0
        LET(max_v, size + i) = 1;
    }
    return max_v;
}
void mcam::HestonModel::print() const
{
    cout << endl;
    cout << "**** Heston Model Characteristics ****" << endl;
    cout << " initial volatility ";
    pnl_vect_print_asrow(sigma0);
    cout << " volatility of volatility ";
    pnl_vect_print_asrow(voVol);
    cout << " reverting rate ";
    pnl_vect_print_asrow(reverting_rate);
    cout << " long run variance ";
    pnl_vect_print_asrow(long_run_var);
    cout << " asset vol correlation " << gamma << endl;
```

```
    mcam::Model::print();
}
```