Help

```
#ifndef _MOD_H
#define _MOD_H

#include "
href../../../../common/math/jlparser/include/jlparser/parser_h_src.pdfjlparser/p
#include "pnl/pnl_matrix.h"
#include "pnl/pnl_random.h"

class BaseModel
{
public:
  double maturity; /*!< maturity of the option */
  int brownianSize; /*!< size of the Brownian motion, default is model size */
  int nTimeSteps;   /*!< step for time discretisation : default
                     * value 1*/
  double interest;     /*!< instantaneous interest rate */
  PnlVect *dividend;    /*!< instantaneous dividend rate */
  int size;    /*!< size of the model */
  double rho;   /*!< correlation of the model, assumed
                 * constant between the different dimensions */
  PnlMat *covChol;    /*!< Cholesky factorisation of the
                       * covariance matrix */
  PnlVect *init;   /*!< spot values */
  /**
   * the pathMatrix PnlMat * is used to store a path of the model
   *
   * number of rows = nb of time steps (= nTimeSteps + 1)
   *
   * number of columns = size of the model
   */
  PnlMat *pathMatrix;
  /**
   * the PnlMat of the standard normal r.v. used to build the Browniand
   * motion with no drift
   *
   * number of rows = nb of time steps (= nTimeSteps + 1)
   *
   * number of columns = size of the model
   */
```

```
  PnlMat *Gincr;
  /**
   * the PnlMat of the drifted standard normal r.v. used to build the Brownian
   * motion with drift
   *
   * number of rows = nb of time steps - 1 (=nTimeSteps)
   * number of columns = size of the model
   */
  PnlMat *Gincr_drift;
  /// Terminal value of the Brownian motion
  PnlVect *BT;
  double sqrt_nTimeSteps;
  double dt;
  double sqrt_dt;

public:
  BaseModel();
  BaseModel(const Param &ParamTab);
  void GenericConstructor(const Param &ParamTab);
  virtual ~BaseModel();

  int Size() const;
  double Rho() const;

  /**
   * Compute the Cholesky factorization of the correlation and store it into
   * covChol
   */
  virtual int computeCovChol();


  /**
   * Compute one path of the model.
   *
   * Draw the releveant random variables and call path()
   *
   * @param rng PnlRng
   */
  virtual void path(PnlRng *rng);
  /**
   * Compute one path of the model with a single drift vector for all the
```

```
 * time steps.
 *
 * @param rng : PnlRng
 * @param drift : the importance sampling vector
 *      (same drift for all time steps)
 */
virtual void path(PnlRng *rng, const PnlVect *drift);
/**
 * Compute one path of the model with one drift vector per time step
 *
 * @param rng : random number generator
 * @param drift : the full importance sampling vector  (one drift per time
 * step)
 */
virtual void pathFull(PnlRng *rng, const PnlVect *drift);

/**
 * Compute one path using Gincr_drift for the Brownian increments
 */
virtual void path() = 0;
/**
 * Compute one path of the model with a single drift vector for all the
 * time steps.
 *
 * @param drift : the importance sampling vector
 *      (same drift for all time steps)
 */
void path(const PnlVect *drift);
/**
 * Compute one path of the model
 *
 * @param drift : the full importance sampling vector  (one drift per time
 * step)
 */
void pathFull(const PnlVect *drift);

PnlMat* getSamplePath() { return pathMatrix; }

/**
 * Sets mod_path to a shifted path
 *
```

```
   * @param i index of the underlying o shift
   * @param h magnitude of the shift x <- x / (1 + h)
   */
  void shiftPath(int i, double h);
  /**
   * Sets mod_path to an unshifted path
   *
   * @param i index of the underlying o shift
   * @param h magnitude of the shift x <- x / (1 + h)
   */
  void unshiftPath(int i, double h);


  /** Prints a model description
   *  To be implemented in each class inherited from BaseModel
   */
  virtual void print() const = 0;

protected:
  PnlVect *workVector;   /*!< Workspace for intermediate computations */
  PnlMat *workMatrix;   /*!< Workspace for intermediate computations */
};

class StocVolModel : public BaseModel
{
protected:
  double gamma; /*!< Correlation between the vol and asset brownian motions (in
  PnlVect *sigma0; /*!< Initial volatilities */
  PnlVect *voVol; /*!< Volatility of Volatility (nu) */
  PnlVect *sigmaVector; /*!< Instantaneous volatility vector */

public:
  StocVolModel();
  StocVolModel(const Param&);
  ~StocVolModel();
  virtual int computeCovChol();
};

BaseModel *instantiate_model(const Param &P);

class JumpModel : public BaseModel
```

```
{
public:
  PnlVect *sigma; /*!< volatilty vector of size (size) */
  int poissonSize; /*!< dimension of the jump process = 1 if size == 1 and (size
  PnlVect *lambda; /*!< intensity vector of size poissonSize normalized to matur
  /**
   * the PnlMat of the jump heights
   *
   * number of rows = nb of time steps - 1 (=nTimeSteps)
   * number of columns = size of the model + 1 (common jump)
   */
  PnlMat *jumpsMat;
  /**
   * the PnlMat of the poisson increments
   *
   * number of rows = nb of time steps - 1 (=nTimeSteps)
   * number of columns = poissonSize
   */
  PnlMat *poissonMat;
  /**
   * To ease further computations, it is convenient to have at hand the
   * vector of intensities for all the poissonMat increments
   * size : Timestep x poissonSize
   */
  PnlVect *lambdaFull;

  JumpModel();
  JumpModel(const Param &P);
  virtual ~JumpModel();
  void print() const;

  /**
   * Compute one path of the model.
   *
   * Draw the releveant random variables and call path()
   *
   * @param rng PnlRng
   */
  virtual void path(PnlRng *rng);
  /**
   * Compute one path of the model with a single drift vector for all the
```

```
 * time steps.
 *
 * @param rng : PnlRng
 * @param drift : the importance sampling vector
 *      (same drift for all time steps)
 */
virtual void path(PnlRng *rng, const PnlVect *drift);
/**
 * Compute one path of the model with one drift vector per time step
 *
 * @param rng : random number generator
 * @param drift : the full importance sampling vector  (one drift per time
 * step)
 */
virtual void pathFull(PnlRng *rng, const PnlVect *drift);
/**
 * Compute a path of the JumpModel model with piecewise constant intensities
 * (given by mu) for the poissonMat processes.
 *
 * @param rng a random number generator
 * @param drift the Gaussian drift (one drift per time step)
 * @param mu jump intensity (one intensity per time time step)
 */
void pathMuThetaFull(PnlRng *rng, const PnlVect *drift, const PnlVect *mu);
/**
 * Compute a path of the JumpModel 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 pathMuFull(PnlRng *rng, const PnlVect *mu);
/**
 * Compute a path of the JumpModel model with constant intensities
 * (given by mu) for the poissonMat processes.
 *
 * @param rng a random number generator
 * @param drift the Gaussian drift (same drift for all time steps)
 * @param mu jump intensity (same intensity for all time steps)
 */
void pathMuTheta(PnlRng *rng, const PnlVect *drift, const PnlVect *mu);
```

```
  /**
   * Compute a path of the JumpModel model with constant intensities
   * (given by mu) for the poissonMat processes.
   *
   * @param rng a random number generator
   * @param mu jump intensity (same intensity for all time steps)
   */
  void pathMu(PnlRng *rng, const PnlVect *mu) ;
  virtual void path();

protected:
  void path_aux(PnlRng *rng);
  /**
   * 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
   */
  virtual void pathMu_aux(PnlRng *rng, const PnlVect *mu) = 0;
  /**
   * Auxiliary function.
   *
   * The Gaussian part has already been drawn in Gincr_drift
   *
   * Compute a path of the 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)
   */
  virtual void pathMuFull_aux(PnlRng *rng, const PnlVect *mu) = 0;
  PnlVect *levyDrift; /*!< drift part of the Levy process */
};

#endif
```