```
   Help
// vim:ft=cpp

#ifndef SVD_BS_H
#define SVD_BS_H

#include "pnl/pnl_complex.h"
#include "pnl/pnl_vector.h"
#include "pnl/pnl_matrix.h"
#include <vector>
#include "pnl/pnl_tridiag_matrix.h"

namespace svd_bs
{

class LogBS_Model
{
public :
  double r;
  PnlVect *mu;
  PnlVect *tmp_y;
  PnlVect *tmp_x;
  double Inf_Bndry;
  double Sup_Bndry;
  double rho;
  PnlVect *vol;
  PnlMat  *Rho;
  double det_sigma;

  LogBS_Model();
  LogBS_Model(double r_, double rho_, const PnlVect *vol_,
    double T, double Inf_Bndry_, double Sup_Bndry_);
  ~LogBS_Model();
  void print() const;
  double density(const PnlVect *x, const double T) const;
  //transform [0,1] to [-Bndry,Bndry]
  void grid_x(PnlVect *pos, const PnlVect *y, double T)
    const;
  double jacobian(double position, double T) const;
};
```

```
class Option
{
public:
  double Strike;
  double T;
  PnlVect *Spot;
  double Inf_Bndry;
  double Sup_Bndry;
public:
  Option();
  Option(double strike_, double matu_, PnlVect *Today_spot,
     double Inf_Bndry_, double Sup_Bndry_);
  Option(const Option &op);
  ~Option();
  void print() const;
  double payoff(const PnlVect *x, PnlVect *mu, double r)
    const;
};

class non_linear_approximation : public std::vector<PnlMat
    *>
{
public:
  int dim;
  int N;
  double h;
  int discretization;
  const LogBS_Model *model;
  const Option *contract;
  PnlMat *Grid_mat;
  PnlVectInt *tmp_pos;
  PnlVect *tmp_step;
  PnlVect *tmp_r_step;
  //We stock the values of the function in a matrix to impr
    ove the running
  //time.This matrix is used when we need to evaluate the
    function.
  //It contains only the points where the function is diffe
    rent to zero. If we consider a situation in a very big dim
    ension (>9) we will have problems to stock this matrix.
  //This function fill in the matrix. See the function de
```

```
    finition to obtain more details.
  //Only if we use the maj function, we will use the next
    line
  //precompute_mat_funct(Funct_mat);
  //New modifications

  //New modifications
  PnlMatInt *Index;/* */
  PnlVect *Value; /* */
  //PnlMat* Mr;/* */
  PnlMat *Ms;/* */
  PnlVect *p_Ms;/* */

  double (*apply)(const PnlVect *step, const LogBS_Model &
    model, const Option &contract);

public :
  non_linear_approximation();
  non_linear_approximation(int dim_, int N_, double (*f)(
    const PnlVect *step, const LogBS_Model &model, const Option &
    contract), const LogBS_Model &model_, const Option &contract_,
     int discretization_);
  ~non_linear_approximation();

  //non_linear_approximation & operator = (cons non_linear_
    approximation& svd_);

  double step_h() const;
  double Grid(int i, int p) const;


  void Ms_init(PnlMat *R, const PnlTridiagMat *M);
  void update_Ms(PnlMat *R, int n, const PnlTridiagMat *M);
  void compute_rhs_product_term_in_dim(int n);


  void add_element(const PnlMat *M);

  void update(PnlMat *R, int n, const PnlTridiagMat *M, Pn
    lVect *rhs);
```

```
  void svd_decomposition(const PnlTridiagMat *M);

  // Print Functions
  void print_svd();
  void print_2d();

  //Error Functions
  /*
     Check compatibility with current version or delete
     */
  double error_frob(const PnlMat *R);
  double error_frob_basket(const PnlMat *R, const PnlTridia
    gMat *M);
  double error_l2(const PnlMat *R, const PnlTridiagMat *M);
  double error_l2_basket(const PnlMat *R, const PnlTridia
    gMat *M);
  double error_l_infinity();

  double svd_integral() const;
  double svd_evaluation(PnlVect *x) const;
  void compute_Index_Value();

};

double integral(int i, int N);
double hat_function(int i, double x, int N);
/*The evaluation of the Grid*/
void Grid_logS(PnlVect *pos, PnlVect *step, const LogBS_
    Model &model, const Option &contract);
void Grid_S(PnlVect *pos, PnlVect *step, const LogBS_Model
    &model, const Option &contract);
PnlMat *construct_grid_mat(int dim, int N, const LogBS_
    Model &model, const Option &contract,  double h, int discretiz
    ation);
int general_initialization(PnlVectInt *pos, const double &
    go_out, PnlVect *step, PnlMat *Grid_mat, int n);
int general_increment(PnlVectInt *pos, PnlVect *step, cons
    t double &go_out, PnlMat *Grid_mat, int n, int N);
PnlTridiagMat *initialise_mass_matrix(int N);

};
```

```
#endif
```

References