Help

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "pnl/pnl_random.h"
#include "pnl/pnl_vector.h"
#include "pnl/pnl_linalgsolver.h"
#include "pnl/pnl_finance.h"
#include "math/equity_pricer/implied_bs.h"
#include "pnl/pnl_mathtools.h"
#include "math/equity_pricer/gridsparse_functions.h"
#include "gridsparse_functions_varswap.h"

#define DEFAULT_VALUE_VARSWAP3D_MOD_DOMAIN_SIZE 4
#define SOLVER_PRECISION 1e-5
#define SOLVER_MAX_ITER  200
#define SOLVER_GMRES_RESTART  20


#define SPARSE_H2N(Vout,Vin,index,father)  LET(Vout,index)=
    GET(Vin,index)+0.5*( GET(Vout,(*father))+GET(Vout,*(father+1
    )))
#define SPARSE_N2H(Vout,Vin,index,father)  LET(Vout,index)=
    GET(Vin,index)-0.5*( GET(Vin,(*father))+GET(Vin,*(father+1))
    )
#define SPARSE_N2H_FUNC(Vin,index,father)  GET(Vin,index)-0
    .5*( GET(Vin,(*father))+GET(Vin,*(father+1)))

/////////////////////////////////////////////////////////
// Stochastic Variance Swap Model Operator on Sparse Grid
/////////////////////////////////////////////////////////

// store the volatility on point of index i
/*
 * @param Op a SVSSparseOp contains data for abstract matr
    ix-vector multiplication A Vin
 * store ${f exp(0.5*sum_{d=2}^{dim} x_d^i ${f where ${f x^
    i_t ${f is
 * Orstein Ulhembeck process in vector Op->V_volatility
 */
void Get_Local_Volatility_init(SVSSparseOp * Op)
```

```
{
  int i,d;
  double sum=0.0;
  LET(Op->V_volatility,0)=0;
  for(i=1;i<Op->V_volatility->size;i++)
    {
      sum=0.0;
      for(d=1;d<Op->G->dim;d++)
    sum+=GET(Op->Model->Beta,d-1)/GET(Op->Model->SqrtMeanRe    version,d-1)*Grid
      LET(Op->V_volatility,i)=MIN(exp(0.5*sum),10);
    }
}

void Get_Local_Volatility(SVSSparseOp * Op,int i,double *     vol,double * sqr_v
{*vol=GET(Op->V_volatility,i);*sqr_vol=(*vol)*(*vol);}

/**
 * initilisation of the sparse operator for diffusion equa
    tion associated to
 * stochastic variance swap model
 *
 * @param pointer on SVSSparseOp
 */
void initialise_svs_sparse_operator(SVSSparseOp * Op)
{
  Op->PC=pnl_vect_create_from_zero(Op->G->size);
  Op->V_tmp0=pnl_vect_create_from_zero(Op->G->size);
  Op->V_tmp1=pnl_vect_create_from_zero(Op->G->size);
  Op->V_tmp2=pnl_vect_create_from_zero(Op->G->size);
  Op->V_tmp3=pnl_vect_create_from_zero(Op->G->size);
  Op->V_tmp4=pnl_vect_create_from_zero(Op->G->size);
  Op->V_tmp5=pnl_vect_create_from_zero(Op->G->size);
  Op->V_tmp6=pnl_vect_create_from_zero(Op->G->size);
  Op->V_tmp7=pnl_vect_create_from_zero(Op->G->size);
  Op->V_volatility=pnl_vect_create_from_zero(Op->G->size);
  Get_Local_Volatility_init(Op);
}



/**
```

```
 * Create the sparse operator for diffusion equation assoc
    iated to
 * stochastic variance swap model
 *
 * @param lev int level of sparse grid
 * @param dim int dimension of model dim = n+1
 with n number of factors
 * @return pointer on SVSSparseOp
 */
SVSSparseOp * svs_sparse_operator_create(int lev,int N_T,
    VARSWAP3D_MOD * M)
{
  int d,dim;
  PnlVect * X0,*X1;
  double sum =0.0;
  SVSSparseOp * Op = malloc(sizeof(SVSSparseOp));
  Op->Model=M;
  // Create space discretization grid
  dim=Op->Model->Nb_factor+1;
  X0=pnl_vect_create(dim);
  X1=pnl_vect_create(dim);
  for(d=1;d<dim;d++)
    {
      LET(X1,d)=sqrt(1-exp(-2* GET(Op->Model->MeanRe    version,d-1)*Op->Model->
    ZE;
      LET(X0,d)=-1.0*GET(X1,d);
      sum+=GET(Op->Model->Beta,d-1)/GET(Op->Model->SqrtMea
    nReversion,d-1)*GET(X1,d);
    }
  svs_sigma_time(Op->Model, Op->Model->T);
  LET(X1,0)=2;//Op->Model->V0*exp(0.5*sum)*sqrt(Op->Model->
    T)*2;
  LET(X0,0)=-1.0*GET(X1,0);
  Op->G=grid_sparse_create(X0,X1,lev);
  pnl_vect_free(&X0);
  pnl_vect_free(&X1);
  // Create time grid - Here Uniform grid
  Op->TG=premia_pde_time_homogen_grid(Op->Model->T,N_T);
  // Initialise size of temporary vectors
  initialise_svs_sparse_operator(Op);
  return Op;
```

```
}

/**
 * desaloccation of sparse operator for diffusion equation
    associated to
 * stochastic variance swap model not free variance swap model
 *
 * @param pointer on SVSSparseOp
 */
void svs_sparse_operator_free(SVSSparseOp ** Op)
{
  premia_pde_time_grid_free(&(*Op)->TG);
  pnl_vect_free(&(*Op)->PC);
  pnl_vect_free(&(*Op)->V_tmp0);
  pnl_vect_free(&(*Op)->V_tmp1);
  pnl_vect_free(&(*Op)->V_tmp2);
  pnl_vect_free(&(*Op)->V_tmp3);
  pnl_vect_free(&(*Op)->V_tmp4);
  pnl_vect_free(&(*Op)->V_tmp5);
  pnl_vect_free(&(*Op)->V_tmp6);
  pnl_vect_free(&(*Op)->V_tmp7);
  pnl_vect_free(&(*Op)->V_volatility);
  GridSparse_free(&(*Op)->G);
  free(*Op);
  *Op=NULL;
}



/*
 * WARNING: with the convention A is matrix operator, we
    compute
 * V_out = a * PC V_in + b Vout
 *
 * @param Op a SVSSparseOp contains data for abstract matr
    ix-vector multiplication PC Vin
 * @param Vin a PnlVec, input parameters
 * @param a a double
 * @param b a double
 * @param Vout a PnlVec, the output
 */
```

```
void GridSparse_apply_svs_PC(SVSSparseOp * Op,
                 const PnlVect * Vin,
                             const double a,
                             const double b,
                             PnlVect * Vout)
{pnl_vect_clone(Vout,Vin); pnl_vect_mult_vect_term(Vout,Op-
    >PC);}

int Index[3]; //Index[0]=dir, Index[1]= Position Index[2]=
    left or right

void GridSparse_add_rhs_svs(SVSSparseOp * Op,
                            PnlVect * V_rhs)
{
  double Forward,sqr_vol,vol;
  int i;
  // No optimisation,
  // Good way : compute & store term in x_1 on uniforme
    fine grid and
  // use it in this loop in place of evaluate premia_bs_s_
    square_gamma ...
  // Here we suppose r=0 - bond = 1.
  PnlVect *VT_rhs=Op->V_tmp0;
  for(i=1;i<VT_rhs->size;i++)
    {
      Forward=Op->Model->S0*exp(GridSparse_real_value_at_po
    ints(Op->G,0,i));
      Get_Local_Volatility(Op,i,&vol,&sqr_vol);
      LET(VT_rhs,i)=(sqr_vol-1.0)*pnl_bs_impli_s_square_gam
    ma (Op->Model->V0_time,1,
                             Forward,
                             Op->Model->Strike,
                             premia_pde_time_grid_time(Op->
    TG)
                             +premia_pde_time_grid_step(Op-
    >TG));
    }
  Nodal_to_Hier(VT_rhs,Op->G);
  pnl_vect_axpby( 0.5*premia_pde_time_grid_step(Op->TG)*
         Op->Model->V0_sqr,VT_rhs,1.0,V_rhs);
}
```

```
void GridSparse_preconditioner_svs_init (SVSSparseOp * Op)
{
  int i,d;
  double vol,sqr_vol,jacobi;
  i=1;
  do
    {
      jacobi=0;
      for(d=1;d<Op->G->dim;d++)
        {
      jacobi+=//0.5*GET(Op->Model->Beta,d-1)*GET(Op->Model-
    >Beta,d-1)*
        GET(Op->Model->MeanReversion,d-1)*(2<<((log2int(pn
    l_mat_int_get(Op->G->Points,i,d))+1)));
        }
      Get_Local_Volatility(Op,i,&vol,&sqr_vol);
      jacobi+=0.5*sqr_vol*Op->Model->V0_sqr*
          (2<<((log2int(pnl_mat_int_get(Op->G->Points,i,0))
    +1)));
      LET(Op->PC,i)=1.0/sqrt(1.0+premia_pde_time_grid_step(
    Op->TG)*jacobi);
      i++;
    }while(i<Op->G->size);
}

int Operator_Initialisation_Log1F(const PnlVect * Vin,SVS
    SparseOp * Op)
{
  // compute : = u + theta L(u) with L :=BS operator
  PnlVect * Drift_Price, * Vol_Price,* Drift_Price_T, *     Vol_Price_T;
  int Index[3]; //Index[0]=dir, Index[1]= Position Index[2]
    = left or right
  int * father;
  double vol,sqr_vol;
  Drift_Price=Op->V_tmp0;
  Vol_Price=Op->V_tmp1;
  Drift_Price_T=Op->V_tmp2;
  Vol_Price_T=Op->V_tmp3;
```

```
  Index[2]=0;
  Index[0]=1;
  //>> on y_1 = x_2
  Index[1]=1;
  father = pnl_hmat_int_lget(Op->G->Ind_Father,Index);
  do
    {
      SPARSE_H2N(Drift_Price,Vin,Index[1],father);
      Index[1]++;father+=2;
    }while(Index[1]<Op->G->size);
  Index[1]=1;
  father = pnl_hmat_int_lget(Op->G->Ind_Father,Index);
  do
    {
      Get_Local_Volatility(Op,Index[1],&vol,&sqr_vol);
      LET(Vol_Price_T,Index[1])=sqr_vol*GET(Drift_Price,Ind
    ex[1]);
      LET(Drift_Price_T,Index[1])=vol*GET(Drift_Price,Ind
    ex[1]);
      SPARSE_N2H(Drift_Price,Drift_Price_T,Index[1],father)
    ;
      SPARSE_N2H(Vol_Price,Vol_Price_T,Index[1],father);
      Index[1]++;father+=2;
    }while(Index[1]<Op->G->size);
  return OK;
}

/*
 * Compute {cL u ... the discret operator
 */
int Operator_SVS_X1F(const PnlVect * Vin,PnlVect * Vout,SVS
    SparseOp * Op,const double a)
{
  PnlVect * Drift_Price,* Vol_Price,*Correl_Price,*dYY,*dS
    Y,*dS,*Price_Dir;
  int Index[3];
  //>> Index[0]=dir, Index[1]= Position Index[2]= left or
    right
  int * father;
  int * neig;
  double coeff, alpha,beta_sqr;
```

```
double sigma =0.5*Op->Model->V0_sqr;
double coeff_2=Op->Model->Rho*Op->Model->V0_time;
alpha=a*Op->theta_time_scheme*premia_pde_time_grid_step(
  Op->TG);
if(Op->G->dim==2)
  Operator_Initialisation_Log1F(Vin,Op);
else
  return WRONG;
Drift_Price=Op->V_tmp0;
Vol_Price=Op->V_tmp1;
Correl_Price=Op->V_tmp2;
dYY=Op->V_tmp4;
dSY=Op->V_tmp5;
dS=Op->V_tmp6;
Price_Dir=Op->V_tmp7;
pnl_vect_set_zero(Price_Dir);
pnl_vect_set_zero(dSY);
pnl_vect_set_zero(dYY);
pnl_vect_set_zero(dS);
//>>compute nodale representation in dimension Dim
Index[2]=0;
//>> Log spot variables
Index[0]=0;
//>> Left father
if (Index[0]>Op->G->dim) {printf("error in dimension");ab
  ort();}
Index[1]=1;//First point on map
father = pnl_hmat_int_lget(Op->G->Ind_Father,Index);
do
  {
    // SPARSE_H2N(Price_Dir,Vin,Index[1],father);
    // Not need without interest rate
    SPARSE_H2N(dSY,Drift_Price,Index[1],father);
    SPARSE_H2N(dYY,Vol_Price,Index[1],father);
    Index[1]++;father+=2;
  }while(Index[1]<Op->G->size);
Index[1]= 1;
father  = pnl_hmat_int_lget(Op->G->Ind_Father,Index);
neig    = pnl_hmat_int_lget(Op->G->Ind_Neigh,Index);
coeff=0.5*Op->Model->Rho*Op->Model->V0_time*Op->Model->Su
  m_Beta;
```

```
do
  {
    // >> Term : sigma^2/2 d2 P/d2x
    LET(Vol_Price,Index[1])=sigma*FD_Lap_Stencil_Center(
  Index[1],Index[0],dYY,Op->G,*(neig),*(neig+1));

    // >> Term : - sigma^2/2 d P/dx
    // >> Term : - sigma / 2  rho Sum_Beta dP/dx

    //>> Centered Scheme
    //LET(Vol_Price,Index[1])-=sigma*FD_Conv_Stencil_Cent
  er(Index[1],Index[0],dYY,Op->G,*(neig),*(neig+1));
    //LET(Vol_Price,Index[1])-=coeff*FD_Conv_Stencil_Cent
  er(Index[1],Index[0],dSY,Op->G,*neig,*(neig+1));

    //>> Decentered Scheme
    LET(Vol_Price,Index[1])-=sigma*FD_Conv_Stencil_DeCent
  er(Index[1],Index[0],dYY,Op->G,*(neig));
    LET(Vol_Price,Index[1])-=coeff*((coeff>0)?FD_Conv_
  Stencil_DeCenter(Index[1],Index[0],dSY,Op->G,*neig):
                    (-1.)*FD_Conv_Stencil_DeCenter(Index[
  1],Index[0],dSY,Op->G,*(neig+1)));

    // >> correlation Term sigma^2/2 d P/dx
    LET(dS,Index[1])=coeff_2*FD_Conv_Stencil_Center(Ind
  ex[1],Index[0],dSY,Op->G,*(neig),*(neig+1));
    SPARSE_N2H(Correl_Price,dS,Index[1],father);
    LET(Vout,Index[1])+=alpha*(SPARSE_N2H_FUNC(Vol_Price,
  Index[1],father));
    Index[1]++;father+=2;neig+=2;
  }while(Index[1]<Op->G->size);

Index[0]++;
// variable x_2 = y_1
if (Index[0]>Op->G->dim) {printf("error in dimension");ab
  ort();}
Index[1]=1;
father = pnl_hmat_int_lget(Op->G->Ind_Father,Index);
do
  {
    SPARSE_H2N(Price_Dir,Vin,Index[1],father);
```

```
      SPARSE_H2N(dSY,Correl_Price,Index[1],father);
      Index[1]++;father+=2;
    }while(Index[1]<Op->G->size);
  Hier_to_Nodal_in_dir(Index[0],Vin,Price_Dir,Op->G);
  Index[1]= 1;
  father  = pnl_hmat_int_lget(Op->G->Ind_Father,Index);
  neig    = pnl_hmat_int_lget(Op->G->Ind_Neigh,Index);
  beta_sqr=GET(Op->Model->MeanReversion,Index[0]-1);
  do
    {
      coeff=GridSparse_real_value_at_points(Op->G,Index[0],
    Index[1]);
      LET(Vol_Price,Index[1])=beta_sqr*FD_Lap_Stencil_Cent
    er(Index[1],Index[0],Price_Dir,Op->G,*neig,*(neig+1));
      LET(Vol_Price,Index[1])-=GET(Op->Model->MeanRe    version,Index[0]-1)*coef
      ((coeff>0)? FD_Conv_Stencil_DeCenter(Index[1],Index[0
    ],Price_Dir,Op->G,*neig):
         (-1.0)*FD_Conv_Stencil_DeCenter(Index[1],Index[0],
    Price_Dir,Op->G,*(neig+1)));

    //FD_Conv_Stencil_Center(Index[1],Index[0],Price_Dir,
    Op->G,*neig,*(neig+1));

      LET(Vol_Price,Index[1])+=GET(Op->Model->SqrtMeanRe    version,Index[0]-1)*
    FD_Conv_Stencil_Center(Index[1],Index[0],dSY,Op->G,*(ne
    ig),*(neig+1));
      //>> Correl x y_1 term
      //>> Back to hierarchical representation
      LET(Vout,Index[1])+=alpha*(SPARSE_N2H_FUNC(Vol_Price,
    Index[1],father));
      Index[1]++;father+=2;neig+=2;
    }while(Index[1]<Op->G->size);
  return OK;
}

int Operator_Initialisation_Log3F(const PnlVect * Vin,SVS
    SparseOp * Op)
{
  // Pointwize multiplication
  // Use for computation of u + theta L(u)
  PnlVect * Drift_Price, * Vol_Price,* Drift_Price_T, *     Vol_Price_T;
```

```
// >> Index[0]=dir, Index[1]= Position Index[2]= left or
  right
int Index[3];
int * father;
double vol,sqr_vol;
Drift_Price=Op->V_tmp0;
Vol_Price=Op->V_tmp1;
Drift_Price_T=Op->V_tmp2;
Vol_Price_T=Op->V_tmp3;
// >> on y_1 = x_2
Index[2]=0;Index[0]=1;Index[1]=1;
father = pnl_hmat_int_lget(Op->G->Ind_Father,Index);
do
  {
    SPARSE_H2N(Drift_Price,Vin,Index[1],father);
    Index[1]++;father+=2;
  }while(Index[1]<Op->G->size);
// >> on y_2 = x_3
Index[0]++;Index[1]=1;
father = pnl_hmat_int_lget(Op->G->Ind_Father,Index);
do{
  SPARSE_H2N(Drift_Price_T,Drift_Price,Index[1],father);
  Index[1]++;father+=2;
}while(Index[1]<Op->G->size);
// >> on y_3 = x_4
Index[0]++;Index[1]=1;
father = pnl_hmat_int_lget(Op->G->Ind_Father,Index);
do {
  SPARSE_H2N(Drift_Price,Drift_Price_T,Index[1],father);
  Index[1]++;father+=2;
}while(Index[1]<Op->G->size);
Index[1]=1;
father = pnl_hmat_int_lget(Op->G->Ind_Father,Index);
do
  {
    Get_Local_Volatility(Op,Index[1],&vol,&sqr_vol);
    LET(Vol_Price_T,Index[1])=sqr_vol*GET(Drift_Price,Ind
  ex[1]);
    LET(Drift_Price_T,Index[1])=vol*GET(Drift_Price,Ind
  ex[1]);
    SPARSE_N2H(Drift_Price,Drift_Price_T,Index[1],father)
```

```
    ;
      SPARSE_N2H(Vol_Price,Vol_Price_T,Index[1],father);
      Index[1]++;father+=2;
    }while(Index[1]<Op->G->size);
  // >> Come back to Hierarchic representation
  Index[0]--;Index[1]=1;
  father = pnl_hmat_int_lget(Op->G->Ind_Father,Index);
  do
    {
      SPARSE_N2H(Drift_Price_T,Drift_Price,Index[1],father)
    ;
      SPARSE_N2H(Vol_Price_T,Vol_Price,Index[1],father);
      Index[1]++;father+=2;
    }while(Index[1]<Op->G->size);
  Index[0]--;Index[1]=1;
  father = pnl_hmat_int_lget(Op->G->Ind_Father,Index);
  do
    {
      SPARSE_N2H(Drift_Price,Drift_Price_T,Index[1],father)
    ;
      SPARSE_N2H(Vol_Price,Vol_Price_T,Index[1],father);
      Index[1]++;father+=2;
    }while(Index[1]<Op->G->size);
  return OK;
}

/*
 * Compute {cL u ... the discret operator
 */
int Operator_SVS_X3F(const PnlVect * Vin,PnlVect * Vout,SVS
    SparseOp * Op,const double a)
{
  PnlVect * Drift_Price,* Vol_Price,*Correl_Price,*dYY,*dS
    Y,*dS,*Price_Dir;
  int Index[3];
  //Index[0]=dir, Index[1]= Position Index[2]= left or right
  int * father, * neig;
  double coeff,alpha,sigma,coeff_2,beta_sqr;
  alpha=a*Op->theta_time_scheme*premia_pde_time_grid_step(
    Op->TG);
  if(Op->G->dim==4)
```

```
  Operator_Initialisation_Log3F(Vin,Op);
else
  return WRONG;
// Rename data vectors of SVSSparseOp use for store compu
  tation
Drift_Price=Op->V_tmp0;
Vol_Price=Op->V_tmp1;
Correl_Price=Op->V_tmp2;
dYY=Op->V_tmp4;
dSY=Op->V_tmp5;
dS=Op->V_tmp6;
Price_Dir=Op->V_tmp7;
// >>compute nodale representation in dimension Dim :
Index[2]=0;// >> Left father
Index[0]=0;// >> First dimension, here Log spot variables
if (Index[0]>Op->G->dim) {printf("error in dimension");ab
  ort();}
Index[1]=1;// >> First point on map
father = pnl_hmat_int_lget(Op->G->Ind_Father,Index);
do
  {
    SPARSE_H2N(dSY,Drift_Price,Index[1],father);
    SPARSE_H2N(dYY,Vol_Price,Index[1],father);
    Index[1]++;father+=2;
  }while(Index[1]<Op->G->size);
Index[1]= 1;
father  = pnl_hmat_int_lget(Op->G->Ind_Father,Index);
neig    = pnl_hmat_int_lget(Op->G->Ind_Neigh,Index);
sigma =0.5*Op->Model->V0_sqr;
coeff=0.5*Op->Model->Rho*Op->Model->V0_time*Op->Model->Su
  m_Beta;
coeff_2=Op->Model->Rho*Op->Model->V0_time;
do
  {
    // >> Term : sigma^2/2 d2 P/d2x
    LET(Vol_Price,Index[1])=sigma*FD_Lap_Stencil_Center(
  Index[1],Index[0],dYY,Op->G,*(neig),*(neig+1));

    // >> Term : - sigma^2/2 d P/dx
    // >> Term : - sigma / 2  rho Sum_Beta dP/dx
```

```
    //>> Centered Scheme
    //LET(Vol_Price,Index[1])-=sigma*FD_Conv_Stencil_Cent
  er(Index[1],Index[0],dYY,Op->G,*(neig),*(neig+1));
    //LET(Vol_Price,Index[1])-=coeff*FD_Conv_Stencil_Cent
  er(Index[1],Index[0],dSY,Op->G,*neig,*(neig+1));

    //>> Decentered Scheme
    LET(Vol_Price,Index[1])-=sigma*FD_Conv_Stencil_DeCent
  er(Index[1],Index[0],dYY,Op->G,*(neig));
    LET(Vol_Price,Index[1])-=coeff*((coeff>0)?FD_Conv_
  Stencil_DeCenter(Index[1],Index[0],dSY,Op->G,*neig):
                        (-1.)*FD_Conv_Stencil_DeCenter(
  Index[1],Index[0],dSY,Op->G,*(neig+1)));

    // >> correlation Term sigma^2/2 d P/dx
    LET(dS,Index[1])=coeff_2*FD_Conv_Stencil_Center(Ind
  ex[1],Index[0],dSY,Op->G,*(neig),*(neig+1));
    // Without centered scheme
    //LET(dS,Index[1])=coeff_2*((coeff_2<0)?FD_Conv_Sten
  cil_DeCenter(Index[1],Index[0],dSY,Op->G,*neig):
    //                (-1.0)*FD_Conv_Stencil_DeCenter(Ind
  ex[1],Index[0],dSY,Op->G,*(neig+1)));
    SPARSE_N2H(Correl_Price,dS,Index[1],father);
    LET(Vout,Index[1])+=alpha*(SPARSE_N2H_FUNC(Vol_Price,
  Index[1],father));
    Index[1]++;father+=2;neig+=2;
  }while(Index[1]<Op->G->size);
Index[0]++;// variable x_2 = y_1
if (Index[0]>Op->G->dim) {printf("error in dimension");ab
  ort();}
Index[1]=1;
father = pnl_hmat_int_lget(Op->G->Ind_Father,Index);
do
  {
    SPARSE_H2N(Price_Dir,Vin,Index[1],father);
    SPARSE_H2N(dSY,Correl_Price,Index[1],father);
    Index[1]++;father+=2;
  }while(Index[1]<Op->G->size);
Hier_to_Nodal_in_dir(Index[0],Vin,Price_Dir,Op->G);
Index[1]=1;
father = pnl_hmat_int_lget(Op->G->Ind_Father,Index);
```

```
neig   = pnl_hmat_int_lget(Op->G->Ind_Neigh,Index);
beta_sqr=GET(Op->Model->MeanReversion,Index[0]-1);
do
  {
    coeff=GridSparse_real_value_at_points(Op->G,Index[0],
  Index[1]);
    LET(Vol_Price,Index[1])=beta_sqr*FD_Lap_Stencil_Cent
  er(Index[1],Index[0],Price_Dir,Op->G,*neig,*(neig+1));
    LET(Vol_Price,Index[1])-=GET(Op->Model->MeanRe    version,Index[0]-1)*coef
  ((coeff>0)? FD_Conv_Stencil_DeCenter(Index[1],Index[0],
  Price_Dir,Op->G,*neig):
      (-1.0)*FD_Conv_Stencil_DeCenter(Index[1],Index[0],
  Price_Dir,Op->G,*(neig+1)));
    // spot vol correlation term
    LET(Vol_Price,Index[1])+=GET(Op->Model->SqrtMeanRe    version,Index[0]-1)*
  FD_Conv_Stencil_Center(Index[1],Index[0],dSY,Op->G,*ne
  ig,*(neig+1));
    //>> vol vol correlation term, for next correlation
  term ...
    LET(Drift_Price,Index[1])=GET(Op->Model->SqrtMeanRe    version,Index[0]-1)
  *FD_Conv_Stencil_Center(Index[1],Index[0],Price_Dir,Op-
  >G,*neig,*(neig+1));
    //>> Back to hierarchical representation
    LET(dS,Index[1])=SPARSE_N2H_FUNC(Drift_Price,Index[1]
  ,father);
    //>> Back to hierarchical representation
    LET(Vout,Index[1])+=alpha*(SPARSE_N2H_FUNC(Vol_Price,
  Index[1],father));
    Index[1]++;father+=2;neig+=2;
  }while(Index[1]<Op->G->size);

Index[0]++;//>>Variable x_3=y_2
if (Index[0]>Op->G->dim) {printf("error in dimension");ab
  ort();}
Index[1]=1;
father = pnl_hmat_int_lget(Op->G->Ind_Father,Index);
do
  {

    SPARSE_H2N(Price_Dir,Vin,Index[1],father);
    SPARSE_H2N(dSY,Correl_Price,Index[1],father);
```

```
    SPARSE_H2N(dYY,dS,Index[1],father);
    Index[1]++;father+=2;
  }while(Index[1]<Op->G->size);
Index[1]=1;
father = pnl_hmat_int_lget(Op->G->Ind_Father,Index);
neig  = pnl_hmat_int_lget(Op->G->Ind_Neigh,Index);
beta_sqr=GET(Op->Model->MeanReversion,Index[0]-1);
do
  {
    coeff=GridSparse_real_value_at_points(Op->G,Index[0],
  Index[1]);
    LET(Vol_Price,Index[1])=beta_sqr*FD_Lap_Stencil_Cent
  er(Index[1],Index[0],Price_Dir,Op->G,*neig,*(neig+1));
    LET(Vol_Price,Index[1])-=GET(Op->Model->MeanRe    version,Index[0]-1)*coef
  ((coeff>0)? FD_Conv_Stencil_DeCenter(Index[1],Index[0],
  Price_Dir,Op->G,*neig):
      (-1.0)*FD_Conv_Stencil_DeCenter(Index[1],Index[0],
  Price_Dir,Op->G,*(neig+1)));
    // spot vol correlation term
    LET(Vol_Price,Index[1])+=GET(Op->Model->SqrtMeanRe    version,Index[0]-1)*
  FD_Conv_Stencil_Center(Index[1],Index[0],dSY,Op->G,*ne
  ig,*(neig+1));
    //>> Correl y_1 y_2 Term
    LET(Vol_Price,Index[1])+=GET(Op->Model->SqrtMeanRe    version,Index[0]-1)*
  FD_Conv_Stencil_Center(Index[1],Index[0],dYY,Op->G,*ne
  ig,*(neig+1));
    //>> vol vol correlation term, for next correlation
  term ...
    LET(Drift_Price,Index[1])=GET(Op->Model->SqrtMeanRe    version,Index[0]-1)
  *FD_Conv_Stencil_Center(Index[1],Index[0],Price_Dir,Op-
  >G,*neig,*(neig+1));
    //>> Back to hierarchical representation
    LET(dS,Index[1])+=SPARSE_N2H_FUNC(Drift_Price,Index[1
  ],father);
    LET(Vout,Index[1])+=alpha*(SPARSE_N2H_FUNC(Vol_Price,
  Index[1],father));
    Index[1]++;father+=2;neig+=2;
  }while(Index[1]<Op->G->size);
Index[0]++;//>>Variable x_4=y_3
if (Index[0]>Op->G->dim) {printf("error in dimension");ab
  ort();}
```

```
  Index[1]=1;
  father = pnl_hmat_int_lget(Op->G->Ind_Father,Index);
  do
    {
      SPARSE_H2N(Price_Dir,Vin,Index[1],father);
      SPARSE_H2N(dSY,Correl_Price,Index[1],father);
      SPARSE_H2N(dYY,dS,Index[1],father);
      Index[1]++;father+=2;
    }while(Index[1]<Op->G->size);
  Index[1]=1;
  father = pnl_hmat_int_lget(Op->G->Ind_Father,Index);
  neig  = pnl_hmat_int_lget(Op->G->Ind_Neigh,Index);
  beta_sqr=GET(Op->Model->MeanReversion,Index[0]-1);
  do
    {
      coeff=GridSparse_real_value_at_points(Op->G,Index[0],
    Index[1]);
      LET(Vol_Price,Index[1])=beta_sqr*FD_Lap_Stencil_Cent
    er(Index[1],Index[0],Price_Dir,Op->G,*neig,*(neig+1));
      LET(Vol_Price,Index[1])-=GET(Op->Model->MeanRe    version,Index[0]-1)*coef
    ((coeff>0)? FD_Conv_Stencil_DeCenter(Index[1],Index[0],
    Price_Dir,Op->G,*neig):
         (-1.0)*FD_Conv_Stencil_DeCenter(Index[1],Index[0],
    Price_Dir,Op->G,*(neig+1)));
      // spot vol correlation term
      LET(Vol_Price,Index[1])+=GET(Op->Model->SqrtMeanRe    version,Index[0]-1)*
    FD_Conv_Stencil_Center(Index[1],Index[0],dSY,Op->G,*ne
    ig,*(neig+1));
      //>> Correl y_3 y_2 + y_1 Term
      LET(Vol_Price,Index[1])+= GET(Op->Model->SqrtMeanRe    version,Index[0]-1)
    FD_Conv_Stencil_Center(Index[1],Index[0],dYY,Op->G,*ne
    ig,*(neig+1));
      //>> Back to hierarchical representation
      LET(Vout,Index[1])+=alpha*(SPARSE_N2H_FUNC(Vol_Price,
    Index[1],father));
      Index[1]++;father+=2;neig+=2;
    }while(Index[1]<Op->G->size);
  return OK;
}
```

```
/*
 * WARNING: with the convention A is matrix operator, we
   compute
 * V_out = a * A V_in + b Vout
 * Hera A = (Mass + sign Delta_t SVS_FD_Operator )
 *
 * @param Op a SVSSparseOp contains data for abstract matr
   ix-vector multiplication A Vin
 * @param Vin a PnlVec, input parameters
 * @param a a double
 * @param b a double
 * @param Vout a PnlVec, the output
 */
void GridSparse_apply_svs(SVSSparseOp * Op,
                          const PnlVect * Vin,
                          const double a,
                          const double b,
                          PnlVect * Vout)
{
  //>> Do V_out = a  Mass     V_in   +  b  Vout
  pnl_vect_axpby(a,Vin,b,Vout);
  //>> Do V_out += a Rigidity V_in
  if(Op->G->dim<=2)
    Operator_SVS_X1F(Vin,Vout,Op,a);
  else
    Operator_SVS_X3F(Vin,Vout,Op,a);
}



/*
 * Solve FD discret Sparse version of for diffusion equatio
   n associated to
 * stochastic variance swap model with theta-scheme in time
 * Vout - theta delta_t SVS_Operator Vout
 * = Vin +(1- theta) delta_t SVS_Operator  Vin + delta_t
   Source_term ,
 *
 * @param Op a SVSSparseOp contains data for abstract matr
   ix-vector multiplication PC Vin
 * @param Vin a PnlVec, the RHS
```

```
 * @param Vout a PnlVec, the output
 */
void GridSparse_Solve_svs(SVSSparseOp * Op,PnlVect * Vres)
{
  int *neig;
  PnlVect* V_rhs;
  PnlBicgSolver* Solver;
  //PnlGmresSolver* Solver;
  double theta=0.0;// Euler implicit
  int Index[3]={0,1,0};
    V_rhs=pnl_vect_create_from_zero(Vres->size);
  Op->theta_time_scheme=-1.0+theta;
  GridSparse_preconditioner_svs_init(Op);
  Solver=pnl_bicg_solver_create(Vres->size,SOLVER_MAX_ITER,
    SOLVER_PRECISION);
  //Solver=pnl_gmres_solver_create(Vin->size,SOLVER_MAX_
    ITER,SOLVER_GMRES_RESTART,SOLVER_PRECISION);
  premia_pde_time_start(Op->TG);
  svs_sigma_time(Op->Model,0.0);
  do
    {
      Op->theta_time_scheme=theta;
      if (theta!=0.0)
    GridSparse_apply_svs(Op,Vres,1.0,0.0,V_rhs);
      else
    pnl_vect_clone(V_rhs,Vres);

      GridSparse_add_rhs_svs(Op,V_rhs);
      Op->theta_time_scheme=-1.0+theta;
      //pnl_gmres_solver_solve((void*)GridSparse_apply_svs,
      pnl_bicg_solver_solve((void*)GridSparse_apply_svs,
                            Op,
                            (void*)GridSparse_apply_svs_PC,
                            Op,
                            Vres,
                            V_rhs,
                            Solver);
     svs_sigma_time(Op->Model,premia_pde_time_grid_time(Op-
    >TG));
     }while(premia_pde_time_grid_increase(Op->TG));
  pnl_bicg_solver_free(&Solver);
```

```
  //pnl_gmres_solver_free(&Solver);
  pnl_vect_mult_double(Vres,Op->Model->Bond);
  pnl_vect_free(&V_rhs);
  Hier_to_Nodal(Vres,Op->G);
  // Here we put delta on Vres[0] ...
  neig  = pnl_hmat_int_lget(Op->G->Ind_Neigh,Index);
  LET(Vres,0)=FD_Conv_Stencil_Center(Index[1],Index[0],Vres
    ,Op->G,*(neig),*(neig+1));
}
```

References