Help

```
#include  "
href../../../mod/wishart2d/wishart2d_std2d/wishart2d_std2d_h_src.pdfwishart2d_st
#include "pnl/pnl_random.h"
#include "pnl/pnl_mathtools.h"
#include "pnl/pnl_vector.h"
#include "pnl/pnl_matrix.h"
#include "pnl/pnl_cdf.h"
#include "
href../../../common/enums_h_src.pdfenums.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2010+2) //The "#els
static int CHK_OPT(MC_WISHART)(void *Opt, void *Mod)
{
  return NONACTIVE;
}
int CALC(MC_WISHART)(void *Opt, void *Mod, PricingMethod *Met)
{
  return AVAILABLE_IN_FULL_PREMIA;
}
#else

static double DiscLawMatch5(int generator)
{
  double u = pnl_rand_uni(generator);
  if (u < 1. / 6.) return -sqrt(3);
  if (u < 1. / 3.) return sqrt(3);
  return 0;
}

static void Mat_inverse_inplace(PnlMat *a, int dim)
{
  //----------------Declaration of variables
  int i;
  PnlMat *tmp;
  PnlVect *tmp1;
  PnlVect *tmp2;
  //---------------
  tmp = pnl_mat_copy(a);
  tmp1 = pnl_vect_create_from_double(dim, 0.);
```

```
  tmp2 = pnl_vect_create_from_double(dim, 0.);

  pnl_vect_set(tmp2, 0, 1.);
  pnl_mat_syslin(tmp1, tmp, tmp2);
  pnl_mat_set_col(a, tmp1, 0);

  for (i = 1; i < dim; i++)
    {

      pnl_vect_set(tmp2, i - 1, 0.);
      pnl_vect_set(tmp2, i, 1.);
      pnl_mat_syslin(tmp1, tmp, tmp2);
      pnl_mat_set_col(a, tmp1, i);


    }
  //---------------desallocation memory
  pnl_mat_free(&tmp);
  pnl_vect_free(&tmp1);
  pnl_vect_free(&tmp2);

}


//----------Premia ne dipose pas d'une fonction faisant le swap entre deux colon
//----------Cette fonction effectue cette operation à partir d'un rang first
static void Swap_cols_Matrix(PnlMat *a, int i, int j, int dim, int first)
{

  double tmp;
  int k;
  for (k = first; k < dim; k++)
    {
      tmp = pnl_mat_get(a, k, i);
      pnl_mat_set(a, k, i, pnl_mat_get(a, k, j));
      pnl_mat_set(a, k, j, tmp);
    }

  return;
}
//----------Premia ne dipose pas d'une fonction faisant le swap entre deux ligne
```

```
//----------Cette fonction effectue cette operation à partir d'un rang first
static void Swap_rows_Matrix(PnlMat *a, int i, int j, int dim, int first)
{

  double tmp;
  int k;
  for (k = first; k < dim; k++)
    {
      tmp = pnl_mat_get(a, i, k);
      pnl_mat_set(a, i, k, pnl_mat_get(a, j, k));
      pnl_mat_set(a, j, k, tmp);
    }

  return;
}
//------------Permutaion des lignes pour la matrice de passage
static void Permutation_Swap_Matrix(PnlMat *a, int i, int j)
{
  pnl_mat_swap_rows(a, i, j);
  return;
}
//------------Factorization de chlosky avec le resultat, page 145, Matrix Comput
//------------Input t la matrice à decomposer
//------------Output G      : la portion de t inferieur,
//------------       permute: la permutation associe,
//------------       retour : dimension de la partie inversible
//------------la matrice peut etre juste positive
static double  Chol_Factorization_MatHigham(PnlMat *t, PnlMat *G, PnlMat *permut
{
  //---------Parametre temporaire
  //PnlMat* res;
  //---------Initialisation les parameteres de la boucle

  //int ttt;
  int r = 0;
  double tmp = 0;
  double tmp1 = 0;
  int rank = 0;
  int i, j, k;

  //--------l'algorithme Outer Product pour la decomposition de Cholesky
```

```
//--------page 145. Matrix computations Golub

for (k = 0; k < n; k++)
  {
    //---recherche de la valeur maximale


    rank = k;
    tmp = 0;

    for (i = k; i < n; i++)
      {
        //scanf("%d",ttt);
        tmp1 = pnl_mat_get(t, i, i);
        if (tmp < tmp1)
          {
            tmp = tmp1;
            rank = i;
          }

      }

    //---le calcul de cholesky apres la permutation
    if (tmp > 0)
      {
        r += 1;
        Swap_rows_Matrix(t, k, rank, n, 0);
        //Mat_afficher(t);
        Swap_cols_Matrix(t, k, rank, n, 0);
        //Mat_afficher(t);
        Permutation_Swap_Matrix(permute, k, rank);
        pnl_mat_set(t, k, k, sqrt(tmp));

        for (j = k + 1; j < n; j++)
          {
            pnl_mat_set(t, j, k, pnl_mat_get(t, j, k) / sqrt(tmp));
          }
        for (j = k + 1; j < n; j++)
          {
            for (i = k + 1; i < n; i++)
```

```
            pnl_mat_set(t, i, j, pnl_mat_get(t, i, j) - pnl_mat_get(t, i, k)
          }
      }
    else
      {
        break;
      }

  }


//--- traitement de la partie non inversible si il 'existe
//--- le trigger de l'evenement sera la valeur du retour de la fonction
for (k = 0; k < n; k++)
  {
    for (i = k + 1; i < n; i++)
      {
        pnl_mat_set(t, k, i, 0.0);
      }
  }

if (r < n)
  {
    for (k = r; k < n; k++)
      {
        pnl_mat_set(t, k, k, 1.0);
        if (r < n - 1)
          {
            for (i = r - 1; i < k; i++)
              {
                pnl_mat_set(t, k, i, 0.0);
              }
          }

      }

  }

//-------Initialisation de la matrice effective G
if (r > 0)
  {
```

```
      pnl_mat_resize(G, r, r);

      for (k = 0; k < r; k++)
        {
          for (i = 0; i < r; i++)
            {
              if (i > k)
                pnl_mat_set(G, k, i, 0.0);
              else
                pnl_mat_set(G, k, i, pnl_mat_get(t, k, i));
            }
        }
    }

  return r;
  //-------Gestion de memoire
}
static void SquarePo(PnlMat *t)
{
  //--------------Decalration of variable
  double delta;
  double a, b, d;
  PnlVect *diag;
  PnlMat *Matmp;
  PnlMat *Matmp2;
  PnlMat *Matmp3;
  //------------------------------------
  Matmp2 = pnl_mat_create_from_double(2, 2, 0.);
  Matmp3 = pnl_mat_create_from_double(2, 2, 0.);
  diag = pnl_vect_create_from_double(2, 0.);

  if (pnl_mat_get(t, 1, 0) != 0)
    {
      //-----------une matrice symetrique non diagonale

      //-----------Initialisation des parametres
      a = pnl_mat_get(t, 0, 0);
      d = pnl_mat_get(t, 1, 1);
      b = pnl_mat_get(t, 0, 1);
```

```
    //----------redimensionner les objets
    pnl_mat_resize(t, 2, 2);

    //---------Calcul des valeurs propres
    delta = (a - d) * (a - d) + 4.*b * b;
    delta = sqrt(delta);
    pnl_vect_set(diag, 0, 0.5 * (a + d + delta));
    pnl_vect_set(diag, 1, 0.5 * (a + d - delta));
    //--------Calcul des vecteurs propres
    delta = b * b + (a - pnl_vect_get(diag, 0)) * (a - pnl_vect_get(diag, 0));
    delta = 1. / sqrt(delta);
    pnl_mat_set(Matmp2, 0, 0, b * delta);
    pnl_mat_set(Matmp2, 1, 1, b * delta);
    pnl_mat_set(Matmp2, 0, 1, (a - pnl_vect_get(diag, 0))*delta);
    pnl_mat_set(Matmp2, 1, 0, (d - pnl_vect_get(diag, 1))*delta);

  }
else
  {
    //------------une matrice deja digonale

    pnl_vect_set(diag, 0, pnl_mat_get(t, 0, 0));
    pnl_vect_set(diag, 1, pnl_mat_get(t, 1, 1));
    pnl_mat_set_id(Matmp2);

  }

//---------Annuler les valeurs negatives
if (pnl_vect_get(diag, 0) < 0)
  pnl_vect_set(diag, 0, 0.);
else
  pnl_vect_set(diag, 0, sqrt(pnl_vect_get(diag, 0)));

if (pnl_vect_get(diag, 1) < 0)
  pnl_vect_set(diag, 1, 0.);
else
  pnl_vect_set(diag, 1, sqrt(pnl_vect_get(diag, 1)));

//---------Caclul de la nouvelle matrice
```

```
  Matmp = pnl_mat_create_diag(diag);
  pnl_mat_mult_mat_inplace(Matmp3, Matmp2, Matmp);
  pnl_mat_sq_transpose(Matmp2);
  pnl_mat_mult_mat_inplace(Matmp, Matmp3, Matmp2);
  pnl_mat_clone(t, Matmp);

  //-------------------Desallocation de memoire
  pnl_vect_free(&diag);
  pnl_mat_free(&Matmp);
  pnl_mat_free(&Matmp2);
  pnl_mat_free(&Matmp3);

}

//------------------ Bound computation
//------------------ computation in one time time all variables that can be used
static void Compute_Tmp_V(double t, double a, PnlVect *diag, int size_d, PnlVect
{
  double u = 0.;
  int i, j;

  pnl_vect_resize_from_double(L_tmp, size_d, 0.0);
  pnl_vect_resize_from_double(phi_tmp, size_d, 0.0);

  pnl_mat_resize(KK_tmp, size_d, size_d);
  pnl_mat_set_double(KK_tmp, 0.);

  for (i = 0; i < size_d; i++)
    {
      if (-pnl_vect_get(diag, i) > 0)
        {

          u = (1. - exp(pnl_vect_get(diag, i) * t)) / (-2.*pnl_vect_get(diag, i)
          pnl_vect_set(L_tmp, i, (1. - exp(2.*pnl_vect_get(diag, i)*t)) / (-2.*p

        }
      else
        {
          u = t * 0.5;
          pnl_vect_set(L_tmp, i, t);
```

```
        }

      pnl_vect_set(phi_tmp, i, u);


    }

  for (i = 0; i < size_d; i++)
    {
      for (j = 0; j < size_d; j++)
        {
          if (a - j >= 1)
            pnl_mat_set(KK_tmp, i, j, 0.);
          else
            {

              u = sqrt(exp(-t * pnl_vect_get(diag, i)) * (1. - (a - (double)j))
              u = exp(-pnl_vect_get(diag, i) * t) * ((1. - (a - (double)j)) * pn
              pnl_mat_set(KK_tmp, i, j, u);
              u = 0.;
            }


        }

    }
  return;
}

//-----------------CIR Exact Simulation
/*
 *      Bessel Exact Simultion Glasserman
 *      dr(t) = (alpha - kr(t)) dt + 2 * sqrt(r(t)) * dWt
 */

//-----------------CIR discretization New L1
static double CIR_NVA_L1(double t, double x, double a, double k, int generator,
{

  double w = 0;
  double tmp = 0;
```

```
double ksi = 0;


w = pnl_rand_uni(generator);

if (x >= KK)
  {
    //N Victoir discretization paper 2006
    if (w < 1. / 6.)
      w = -1.0;
    else
      {
        if (w < 1. / 3.)
          w = 1.0;
        else
          w = 0.0;
      }

    w = sqrt(3.*t) * w;
    tmp = sqrt((a - 1.) * phii + exp(-k * t * 0.5) * x) + w;
    return tmp * tmp * exp(-k * t * 0.5) + (a - 1.) * phii;
  }
else
  {
    //Matching moment Aurelien Alfonsi paper 2008
    if (k != 0)
      ksi = x * k / ((exp(k * t) - 1.));
    else
      ksi = x / t;


    tmp = LL * LL * (2 * (a + 2.*ksi)) + (a + ksi) * LL * (a + ksi) * LL;
    ksi = (a + ksi) * LL;
    tmp = 0.5 * (1. - sqrt(1. - ksi * ksi / tmp));
    if (w < tmp)
      {
        return  0.5 * ksi / tmp;
      }
    else
      {
        return  0.5 * ksi / (1. - tmp);
```

```
        }
    }

}

//-----------------CIR Exact Simulation
/*
 *      Bessel Exact Simultion Glasserman
 *      dr(t) = (alpha - kr(t)) dt + 2 * sqrt(r(t)) * dWt
 */
static double BesselExactRandom(double t, double alpha, double k, double initial
{
  //  double sigma =2.0;
  //double pathtime = L*exp(-k*t);
  double d = alpha;
  double c = L;//sigma*sigma*pathtime/(4.0);
  double lambda = (initialValue * exp(-k * t)) / c;
  double z, x;
  double tmp = 0.0;
  if (d > 1.0)
    {
      z = pnl_rand_normal(ge);
      x = 2.*pnl_rand_gamma((d - 1.) * 0.5, 1.0, ge); //ran_chisq(d-1.0,ge);//
      tmp = z + sqrt(lambda);
      tmp = c * (tmp * tmp + x);
      return tmp;
    }
  else
    {
      z = pnl_rand_poisson(lambda * 0.5, ge);
      x = 2.*pnl_rand_gamma((d + 2.*z) * 0.5, 1., ge); //ran_chisq(d + 2.0*z,ge)
      return c * x;
    }
}

//----------------Weak order scheme for the operator L witht drift, and a sigma
static void Wishart_Disc(PnlMat *F, int dim, double t, double a, PnlMat *driftEx
{
  //--------------------Declaration of variable
  PnlMat  *K;
  PnlMat  *G;
```

```
PnlMat *permute;
PnlMat *Matmp;

PnlVect *U;
PnlVect *X;
PnlVect *Xtmp;
PnlVect *Vtmp;
double tmp, tmp2;
int i, j, k, h;
int rank;
double D;
int orderoperator;
//-----------------------


//----------Matrix Initialization
Matmp = pnl_mat_create_from_double(dim, dim, 0.);
//---------first step calculus
pnl_mat_mult_mat_inplace(Matmp, F, driftExpT);
pnl_mat_mult_mat_inplace(F, driftExp, Matmp);

//---------Wishart discretization
K = pnl_mat_create_from_double(dim - 1, dim - 1, 0.0);
G = pnl_mat_create_from_double(1, 1, 0.0);
U = pnl_vect_create_from_double(1, 0.0);
X = pnl_vect_create_from_double(dim - 1, 0.0);
Vtmp = pnl_vect_create_from_double(dim - 1, 0.0);
Xtmp = pnl_vect_create_from_double(1, 0.0);
permute = pnl_mat_create_from_double(dim - 1, dim - 1, 0.);
pnl_mat_set_id(permute);


for (h = 0; h < dim; h++)
  {
    //----------Matrix Initialization within the loop
    pnl_mat_set_double(K, 0.0);
    pnl_mat_set_double(G, 0.0);
    pnl_vect_set_double(U, 0.);
    pnl_vect_set_double(X, 0.);
    pnl_vect_set_double(Xtmp, 0.);
    pnl_vect_set_double(Vtmp, 0.);
```

```
pnl_mat_set_id(permute);
D = pnl_vect_get(diagonal, h);
orderoperator = h;

//-----------Permutation for the operator L_h
Swap_rows_Matrix(F, 0, h, dim, 0);
Swap_cols_Matrix(F, 0, h, dim, 0);


for (i = 1; i < dim; i++)
  for (j = 1; j < dim; j++)
    pnl_mat_set(K, i - 1, j - 1, pnl_mat_get(F, i, j));
for (i = 1; i < dim; i++)
  pnl_vect_set(X, i - 1, pnl_mat_get(F, 0, i));
//----------Chloesky Factorization modified
rank = Chol_Factorization_MatHigham(K, G, permute, dim - 1);

if (rank != 0)
  {
    //------If the matrix is not null
    pnl_mat_mult_vect_inplace(Vtmp, permute, X);
    pnl_vect_clone(X, Vtmp);
    k = G->m;
    pnl_vect_resize(Xtmp, k);
    pnl_vect_resize(U, k);
    for (i = 0; i < k; i++)
      pnl_vect_set(Xtmp, i, pnl_vect_get(X, i));


    pnl_mat_lower_syslin(U, G, Xtmp);

    //pnl_vect_rand_normal(Xtmp,k,generator);

    for (i = 0; i < k; i++)
      {
        pnl_vect_set(Xtmp, i, DiscLawMatch5(generator));

      }

    pnl_vect_mult_double(Xtmp, sqrt(pnl_vect_get(L_tmp, orderoperator)));
```

```
    tmp = pnl_vect_norm_two(U);


    tmp = tmp * tmp;
    pnl_vect_mult_double(U, exp(t * D));
    pnl_vect_plus_vect(Xtmp, U);

    tmp = CIR_NVA_L1(t, pnl_mat_get(F, 0, 0) - tmp, a - (k), -2.*D, genera

    tmp2 = pnl_vect_norm_two(Xtmp);
    tmp2 = tmp2 * tmp2;
    pnl_mat_set(F, 0, 0, tmp + tmp2);


    for (i = 0; i < k; i++)
      pnl_vect_set(X, i, pnl_vect_get(Xtmp, i));
    for (i = k + 1; i < dim - 1; i++)
      pnl_vect_set(X, i, 0.);

    //---------Original vector C computation in t
    pnl_mat_mult_vect_inplace(Vtmp, K, X);
    pnl_vect_clone(X, Vtmp);

    pnl_mat_mult_vect_transpose_inplace(Vtmp, permute, X);
    pnl_vect_clone(X, Vtmp);

    for (i = 1; i < dim; i++)
      {
        tmp = pnl_vect_get(X, i - 1);

        pnl_mat_set(F, 0, i, tmp);
        pnl_mat_set(F, i, 0, tmp);
      }
  }
else
  {
    //-----If the submatrix is null

    tmp = CIR_NVA_L1(t, pnl_mat_get(F, 0, 0), a, -2.*D, generator, pnl_vec

    pnl_mat_set(F, 0, 0, tmp);
```

```
      }

      //-----Back to the original matrix
      Swap_rows_Matrix(F, 0, h, dim, 0);
      Swap_cols_Matrix(F, 0, h, dim, 0);
    }
  //----------Last step of discretization
  pnl_mat_mult_mat_inplace(Matmp, F, driftExpT);
  pnl_mat_mult_mat_inplace(F, driftExp, Matmp);

  //----------Memory desallocation
  pnl_mat_free(&K);
  pnl_mat_free(&G);
  pnl_mat_free(&permute);
  pnl_mat_free(&Matmp);

  pnl_vect_free(&Vtmp);
  pnl_vect_free(&U);
  pnl_vect_free(&X);
  pnl_vect_free(&Xtmp);

  return;

}


static void Wishart_Disc_E(PnlMat *F, int dim, double t, double a, PnlMat *drift
{
  //--------------------Declaration of variable
  PnlMat  *K;
  PnlMat  *G;
  PnlMat *permute;
  PnlMat *Matmp;

  PnlVect *U;
  PnlVect *X;
  PnlVect *Xtmp;
  PnlVect *Vtmp;
  double tmp, tmp2;
  int i, j, k, h;
```

```
int rank;
double D;
int orderoperator;
//-----------------------


//----------Matrix Initialization
Matmp = pnl_mat_create_from_double(dim, dim, 0.);
//---------first step calculus
pnl_mat_mult_mat_inplace(Matmp, F, driftExpT);
pnl_mat_mult_mat_inplace(F, driftExp, Matmp);

//---------Wishart discretization
K = pnl_mat_create_from_double(dim - 1, dim - 1, 0.0);
G = pnl_mat_create_from_double(1, 1, 0.0);
U = pnl_vect_create_from_double(1, 0.0);
X = pnl_vect_create_from_double(dim - 1, 0.0);
Vtmp = pnl_vect_create_from_double(dim - 1, 0.0);
Xtmp = pnl_vect_create_from_double(1, 0.0);
permute = pnl_mat_create_from_double(dim - 1, dim - 1, 0.);
pnl_mat_set_id(permute);


for (h = 0; h < dim; h++)
  {
    //----------Matrix Initialization within the loop
    pnl_mat_set_double(K, 0.0);
    pnl_mat_set_double(G, 0.0);
    pnl_vect_set_double(U, 0.);
    pnl_vect_set_double(X, 0.);
    pnl_vect_set_double(Xtmp, 0.);
    pnl_vect_set_double(Vtmp, 0.);
    pnl_mat_set_id(permute);
    D = pnl_vect_get(diagonal, h);
    orderoperator = h;

    //-----------Permutation for the operator L_h
    Swap_rows_Matrix(F, 0, h, dim, 0);
    Swap_cols_Matrix(F, 0, h, dim, 0);
```

```
for (i = 1; i < dim; i++)
  for (j = 1; j < dim; j++)
    pnl_mat_set(K, i - 1, j - 1, pnl_mat_get(F, i, j));
for (i = 1; i < dim; i++)
  pnl_vect_set(X, i - 1, pnl_mat_get(F, 0, i));
//----------Chloesky Factorization modified
rank = Chol_Factorization_MatHigham(K, G, permute, dim - 1);

if (rank != 0)
  {
    //------If the matrix is not null
    pnl_mat_mult_vect_inplace(Vtmp, permute, X);
    pnl_vect_clone(X, Vtmp);
    k = G->m;
    pnl_vect_resize(Xtmp, k);
    pnl_vect_resize(U, k);
    for (i = 0; i < k; i++)
      pnl_vect_set(Xtmp, i, pnl_vect_get(X, i));


    pnl_mat_lower_syslin(U, G, Xtmp);

    pnl_vect_rand_normal(Xtmp, k, generator);

    pnl_vect_mult_double(Xtmp, sqrt(pnl_vect_get(L_tmp, orderoperator)));

    tmp = pnl_vect_norm_two(U);


    tmp = tmp * tmp;
    pnl_vect_mult_double(U, exp(t * D));
    pnl_vect_plus_vect(Xtmp, U);


    tmp = BesselExactRandom(t, a - k, -2.*D, pnl_mat_get(F, 0, 0) - tmp, g
    tmp2 = pnl_vect_norm_two(Xtmp);
    tmp2 = tmp2 * tmp2;
    pnl_mat_set(F, 0, 0, tmp + tmp2);


    for (i = 0; i < k; i++)
```

```
          pnl_vect_set(X, i, pnl_vect_get(Xtmp, i));
        for (i = k + 1; i < dim - 1; i++)
          pnl_vect_set(X, i, 0.);

        //---------Original vector C computation in t
        pnl_mat_mult_vect_inplace(Vtmp, K, X);
        pnl_vect_clone(X, Vtmp);

        pnl_mat_mult_vect_transpose_inplace(Vtmp, permute, X);
        pnl_vect_clone(X, Vtmp);

        for (i = 1; i < dim; i++)
          {
            tmp = pnl_vect_get(X, i - 1);

            pnl_mat_set(F, 0, i, tmp);
            pnl_mat_set(F, i, 0, tmp);
          }
      }
    else
      {
        //-----If the submatrix is null
        tmp = BesselExactRandom(t, a, -2.*D, pnl_mat_get(F, 0, 0), generator,
        //CIR_NVA_L1(t,pnl_mat_get(F,0,0), a,-2.*D,generator, pnl_vect_get(phi

        pnl_mat_set(F, 0, 0, tmp);

      }

    //-----Back to the original matrix
    Swap_rows_Matrix(F, 0, h, dim, 0);
    Swap_cols_Matrix(F, 0, h, dim, 0);
  }
//----------Last step of discretization
pnl_mat_mult_mat_inplace(Matmp, F, driftExpT);
pnl_mat_mult_mat_inplace(F, driftExp, Matmp);

//----------Memory desallocation
pnl_mat_free(&K);
pnl_mat_free(&G);
pnl_mat_free(&permute);
```

```
  pnl_mat_free(&Matmp);

  pnl_vect_free(&Vtmp);
  pnl_vect_free(&U);
  pnl_vect_free(&X);
  pnl_vect_free(&Xtmp);

  return;

}

//----------------weak order of the first scheme in modelpresented by Gourieroux
static void HW(PnlVect *S, PnlMat *F, PnlMat *Y, int dim, double t, double a, Pn
{

  //----------------------Declaration of variable
  int i, j;
  PnlMat *tmp1;
  PnlMat *tmp2;
  double tmp;
  //----------------------


  tmp1 = pnl_mat_copy(F);
  tmp2 = pnl_mat_create_from_double(dim, dim, 0.);
  //----------discretization of the Wishart matrix Xt
  Wishart_Disc(F, dim, t, a, driftExp, driftExpT, diagonal, generator, phi_tmp,
  //----------discretization of the median matrix  Yt
  pnl_mat_plus_mat(tmp1, F);
  pnl_mat_mult_double(tmp1, 0.5 * t);
  pnl_mat_plus_mat(Y, tmp1);
  //----------discrteization of the stock St
  //pnl_mat_minus_mat(tmp2,Y);
  for (i = 0; i < dim; i++)
    {

      pnl_mat_mult_mat_inplace(tmp2, tmp1, D[i]);
      tmp = 0;
      for (j = 0; j < dim; j++)
        {
          tmp += pnl_mat_get(tmp2, j, j);
```

```
      }
    tmp += pnl_vect_get(mu, i) * t;
    tmp = pnl_vect_get(S, i) * exp(tmp);
    pnl_vect_set(S, i, (tmp));
  }


//----------------------Desallocation memory
pnl_mat_free(&tmp1);
pnl_mat_free(&tmp2);
return ;
}

//Exact Scheme Wishart
static void HW_E(PnlVect *S, PnlMat *F, PnlMat *Y, int dim, double t, double a,
{

//----------------------Declaration of variable
int i, j;
PnlMat *tmp1;
PnlMat *tmp2;
double tmp;
//----------------------


tmp1 = pnl_mat_copy(F);
tmp2 = pnl_mat_create_from_double(dim, dim, 0.);
//----------discretization of the Wishart matrix Xt
Wishart_Disc_E(F, dim, t, a, driftExp, driftExpT, diagonal, generator, phi_tmp


//----------discretization of the median matrix  Yt
pnl_mat_plus_mat(tmp1, F);
pnl_mat_mult_double(tmp1, 0.5 * t);
pnl_mat_plus_mat(Y, tmp1);
//----------discrteization of the stock St
//pnl_mat_minus_mat(tmp2,Y);
for (i = 0; i < dim; i++)
  {
```

```
      pnl_mat_mult_mat_inplace(tmp2, tmp1, D[i]);
      tmp = 0;
      for (j = 0; j < dim; j++)
        {
          tmp += pnl_mat_get(tmp2, j, j);

        }
      tmp += pnl_vect_get(mu, i) * t;
      tmp = pnl_vect_get(S, i) * exp(tmp);
      pnl_vect_set(S, i, (tmp));
    }


  //---------------------Desallocation memory
  pnl_mat_free(&tmp1);
  pnl_mat_free(&tmp2);
  return ;

}

static void HZZ(PnlVect *S, PnlMat *F, PnlMat *Y, PnlMat *aT, int dim, double t,
{
  //-----------------------Declaration of variable
  PnlMat *tmp1;
  PnlVect *tmp2;
  PnlVect *tmp3;
  double tmp;
  int i;
  //-----------------------
  tmp1 = pnl_mat_copy(F);
  tmp2 = pnl_vect_create_from_double(dim, 0.);
  tmp3 = pnl_vect_create_from_double(dim, 0.);
  tmp  = 0.;


  SquarePo(tmp1);
  pnl_vect_rand_normal(tmp2, dim, generator);
  tmp = sqrt(t);
  pnl_vect_mult_double(tmp2, tmp);
  pnl_mat_mult_vect_inplace(tmp3, tmp1, tmp2);
  pnl_mat_mult_vect_inplace(tmp2, aT, tmp3);
```

```
  for (i = 0; i < dim; i++)
    {
      tmp = pnl_vect_get(S, i) * exp(pnl_vect_get(tmp2, i));
      pnl_vect_set(S, i, (tmp));
    }

  //-----------------------Desallocation of memory
  pnl_mat_free(&tmp1);
  pnl_vect_free(&tmp2);
  pnl_vect_free(&tmp3);
  return;
}


//----------------the scheme for the stock in christian Gourieroux paper derivat
//----------------volatility Christian gourieroux and Razvan Sufana
//----------------dlog(St) = mu.dt + trace(D1 x Ft).dt + aT x sqrt(Ft) x dWt ; w
//----------------dFt = a.dt + (b1 x Ft + Ft x b1T).dt + sqrt(Ft) x dZt + dZtT x
//----------------Zt and Wt are independent B.Ms
static int mc_wishart2d(PnlVect *S0, NumFunc_2 *p, double t, double r, PnlVect *
{
  int dim = 2;
  double alphaa, z_alpha;
  double  price_sample, mean_price, var_price;
  double S_T1, S_T2;
  //----------------------Declaration of variable
  double DT;
  int i, j;
  int init_mc;
  PnlVect  *phi_tmp;
  PnlVect  *L_tmp;
  PnlVect  *diagonal;
  PnlVect  *Stmp;
  PnlVect *mu;

  PnlMat   *Ftmp;
  PnlMat   *Ytmp;
  PnlMat   *KK_tmp;
  PnlMat   *btmp;
  PnlMat   *atmp;
```

```
PnlMat   *tmp1;
PnlMat *F0, *Q, *b;
PnlMat **MatDrift;

//----------------------Initialisation of variable

F0 = pnl_mat_create_from_double(dim, dim, 0.0);
Q = pnl_mat_create_from_double(dim, dim, 0.0);
b = pnl_mat_create_from_double(dim, dim, 0.0);
mu = pnl_vect_create_from_double(dim, 0.);
pnl_vect_set(mu, 0, r + pnl_vect_get(divid, 0));
pnl_vect_set(mu, 1, r + pnl_vect_get(divid, 1));

MLET(F0, 0, 0) = GET(Sigma0V, 0);
MLET(F0, 0, 1) = GET(Sigma0V, 1);
MLET(F0, 1, 0) = GET(Sigma0V, 2);
MLET(F0, 1, 1) = GET(Sigma0V, 3);

MLET(Q, 0, 0) = GET(QP, 0);
MLET(Q, 0, 1) = GET(QP, 1);
MLET(Q, 1, 0) = GET(QP, 2);
MLET(Q, 1, 1) = GET(QP, 3);


MLET(b, 0, 0) = GET(bP, 0);
MLET(b, 0, 1) = GET(bP, 1);
MLET(b, 1, 0) = GET(bP, 2);
MLET(b, 1, 1) = GET(bP, 3);

/* Value to construct the confidence interval */
alphaa = (1. - confidence) / 2.;
z_alpha = pnl_inv_cdfnor(1. - alphaa);


mean_price = 0.;
var_price  = 0.;
price_sample = 0.;

diagonal = pnl_vect_create_from_double(dim, 0.);
Ftmp     = pnl_mat_copy(F0);
Ytmp     = pnl_mat_create_from_double(dim, dim, 0.);
```

```
KK_tmp   = pnl_mat_create_from_double(dim, dim, 0.);
Stmp     = pnl_vect_copy(S0);
L_tmp    = pnl_vect_create_from_double(dim, 0.);
phi_tmp  = pnl_vect_create_from_double(dim, 0.);
MatDrift = (PnlMat **) malloc(dim * sizeof(PnlMat *));
DT       = (double)(t / ((double)(M)));
Compute_Tmp_V(DT, alpha, diagonal, dim , phi_tmp, L_tmp, KK_tmp);
btmp     = pnl_mat_create_from_double(dim, dim, 0.);
init_mc  = 1;
atmp = pnl_mat_copy(Q);
pnl_mat_sq_transpose(atmp);

//Exact Scheme for Wishart and Weak for the stock
if (flag_scheme == 1)
  {

    for (i = 0; i < dim; i++)
      {
        MatDrift[i] = pnl_mat_create_from_double(dim, dim, 0.);
        pnl_mat_set(MatDrift[i], i, i, -0.5);
        pnl_mat_mult_mat_inplace(btmp, MatDrift[i], atmp);
        pnl_mat_mult_mat_inplace(MatDrift[i], Q, btmp);

      }

    pnl_mat_clone(btmp, b);
    tmp1 = pnl_mat_copy(atmp);
    Mat_inverse_inplace(tmp1, dim);
    pnl_mat_mult_mat_inplace(atmp, tmp1, btmp);
    pnl_mat_clone(tmp1, Q);
    pnl_mat_sq_transpose(tmp1);
    pnl_mat_mult_mat_inplace(btmp, atmp, tmp1);

    pnl_mat_mult_double(btmp, 0.5 * DT);
    pnl_mat_exp(atmp, btmp);
    pnl_mat_clone(btmp, atmp);
    pnl_mat_sq_transpose(atmp);

    init_mc = pnl_rand_init(generator, 1, (long) M * N * dim * (dim + 1));

    for (i = 1; i <= N; i++)
```

```
      {
        pnl_mat_clone(Ftmp, F0);
        pnl_vect_clone(Stmp, S0);
        pnl_mat_set_double(Ytmp, 0.);

        for (j = 1; j <= M; j++)
          {

            //----------Scheme discretization

            HZZ(Stmp, Ftmp, Ytmp, tmp1, dim, DT * 0.5, generator);

            HW_E(Stmp, Ftmp, Ytmp, dim, DT, alpha, btmp, atmp, diagonal, gener

            HZZ(Stmp, Ftmp, Ytmp, tmp1, dim, DT * 0.5, generator);

          }
        //------------Payoff calculus
        S_T1 = pnl_vect_get(Stmp, 0);
        S_T2 = pnl_vect_get(Stmp, 1);
        price_sample = (p->Compute)(p->Par, S_T1, S_T2);

        /* Sum */
        mean_price += price_sample;
        /* Sum of squares */
        var_price  += price_sample * price_sample;

      }
  }
//Weak Schemes for Wishart and stock
if (flag_scheme == 2)
  {


    for (i = 0; i < dim; i++)
      {
        MatDrift[i] = pnl_mat_create_from_double(dim, dim, 0.);
        pnl_mat_set(MatDrift[i], i, i, -0.5);
        pnl_mat_mult_mat_inplace(btmp, MatDrift[i], atmp);
        pnl_mat_mult_mat_inplace(MatDrift[i], Q, btmp);
```

```
  }

pnl_mat_clone(btmp, b);
tmp1 = pnl_mat_copy(atmp);
Mat_inverse_inplace(tmp1, dim);
pnl_mat_mult_mat_inplace(atmp, tmp1, btmp);
pnl_mat_clone(tmp1, Q);
pnl_mat_sq_transpose(tmp1);
pnl_mat_mult_mat_inplace(btmp, atmp, tmp1);

pnl_mat_mult_double(btmp, 0.5 * DT);
pnl_mat_exp(atmp, btmp);
pnl_mat_clone(btmp, atmp);
pnl_mat_sq_transpose(atmp);

init_mc = pnl_rand_init(generator, 1, (long) M * N * dim * (dim + 1));

for (i = 1; i <= N; i++)
  {
    pnl_mat_clone(Ftmp, F0);
    pnl_vect_clone(Stmp, S0);
    pnl_mat_set_double(Ytmp, 0.);

    for (j = 1; j <= M; j++)
      {

        //----------Scheme discretization

        HZZ(Stmp, Ftmp, Ytmp, tmp1, dim, DT * 0.5, generator);

        HW(Stmp, Ftmp, Ytmp, dim, DT, alpha, btmp, atmp, diagonal, generat

        HZZ(Stmp, Ftmp, Ytmp, tmp1, dim, DT * 0.5, generator);

      }
    //------------Payoff calculus
    S_T1 = pnl_vect_get(Stmp, 0);
    S_T2 = pnl_vect_get(Stmp, 1);
    price_sample = (p->Compute)(p->Par, S_T1, S_T2);

    /* Sum */
```

```
        mean_price += price_sample;
        /* Sum of squares */
        var_price  += price_sample * price_sample;

      }

  }

//-------------final Value

/*----Price Estimator------*/
*ptprice = (mean_price / (double)N);
*pterror_price = sqrt(var_price / (double)N - (*ptprice) * (*ptprice)) / sqrt(
/* Price Confidence Interval */
*inf_price = *ptprice - z_alpha * (*pterror_price);
*sup_price = *ptprice + z_alpha * (*pterror_price);


///--------------------Desallocation memory
pnl_mat_free(&KK_tmp);
pnl_mat_free(&btmp);
pnl_mat_free(&atmp);
pnl_mat_free(&tmp1);
pnl_mat_free(&Ytmp);
pnl_mat_free(&Ftmp);
pnl_mat_free(&Q);
pnl_mat_free(&F0);
pnl_mat_free(&b);

pnl_vect_free(&Stmp);
pnl_vect_free(&phi_tmp);
pnl_vect_free(&L_tmp);
pnl_vect_free(&diagonal);
pnl_vect_free(&mu);
for (i = 0; i < dim; i++)
  {
    pnl_mat_free(&MatDrift[i]);

  }
free(MatDrift);
return init_mc;
```

```
}

int CALC(MC_WISHART)(void *Opt, void *Mod, PricingMethod *Met)
{
  TYPEOPT *ptOpt = (TYPEOPT *)Opt;
  TYPEMOD *ptMod = (TYPEMOD *)Mod;

  double r;

  r = log(1. + ptMod->R.Val.V_DOUBLE / 100.);

  return mc_wishart2d(ptMod->S0.Val.V_PNLVECT,
                      ptOpt->PayOff.Val.V_NUMFUNC_2,
                      ptOpt->Maturity.Val.V_DATE - ptMod->T.Val.V_DATE,
                      r,
                      ptMod->Divid.Val.V_PNLVECT, ptMod->alpha.Val.V_PDOUBLE,
                      ptMod->b.Val.V_PNLVECT,
                      ptMod->Sigma0.Val.V_PNLVECT,
                      ptMod->Q.Val.V_PNLVECT,
                      Met->Par[0].Val.V_LONG,
                      Met->Par[1].Val.V_INT,
                      Met->Par[2].Val.V_ENUM.value,
                      Met->Par[3].Val.V_ENUM.value,
                      Met->Par[4].Val.V_PDOUBLE,
                      &(Met->Res[0].Val.V_DOUBLE),
                      &(Met->Res[1].Val.V_DOUBLE),
                      &(Met->Res[2].Val.V_DOUBLE),
                      &(Met->Res[3].Val.V_DOUBLE)
                     );
}

static int CHK_OPT(MC_WISHART)(void *Opt, void *Mod)
{
  if ((strcmp(((Option *)Opt)->Name, "CallMaximumEuro") == 0) || (strcmp(((Optio
    return OK;
  return  WRONG;
}

#endif //PremiaCurrentVersion
```

```
static int MET(Init)(PricingMethod *Met, Option *Opt)
{
  //int type_generator;
  if (Met->init == 0)
    {
      Met->init = 1;

      Met->Par[0].Val.V_LONG = 15000;
      Met->Par[1].Val.V_INT = 10;
      Met->Par[2].Val.V_ENUM.value = 2;
      Met->Par[2].Val.V_ENUM.members = &PremiaEnumDiscretizationScheme;
      Met->Par[3].Val.V_ENUM.value = 0;
      Met->Par[3].Val.V_ENUM.members = &PremiaEnumMCRNGs;
      Met->Par[4].Val.V_DOUBLE = 0.95;

    }

  return OK;
}

PricingMethod MET(MC_WISHART) =
{
  "MC_Wishart2D",
  { {"N iterations", LONG, {100}, ALLOW},
    {"TimeStepNumber", LONG, {100}, ALLOW},
    {"Discretization Scheme", ENUM, {100}, ALLOW},
    {"RandomGenerator", ENUM, {100}, ALLOW},
    {"Confidence Value", DOUBLE, {100}, ALLOW},
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CALC(MC_WISHART),
  { {"Price", DOUBLE, {100}, FORBID},
    {"Error Price", DOUBLE, {100}, FORBID},
    {"Inf Price", DOUBLE, {100}, FORBID},
    {"Sup Price", DOUBLE, {100}, FORBID} ,
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CHK_OPT(MC_WISHART),
  CHK_mc,
  MET(Init)
};
```