Help

```
// Compute the price of a call and put option on a single asset whose
// volatility follows a Wishart process
// Céline Labart, Sept. 2013


#include "
href../../../mod/hes1d_multifactor/hes1d_multifactor_std/hes1d_multifactor_std_h
#include "pnl/pnl_random.h"
#include "pnl/pnl_basis.h"
#include "pnl/pnl_vector.h"
#include "pnl/pnl_cdf.h"
#include "pnl/pnl_complex.h"
#include "pnl/pnl_fft.h"
#include "pnl/pnl_finance.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2014+2) //The "#els
static int CHK_OPT(AP_FFT_Wishart)(void *Opt, void *Mod)
{
  return NONACTIVE;
}
int CALC(AP_FFT_Wishart)(void *Opt, void *Mod, PricingMethod *Met)
{
  return AVAILABLE_IN_FULL_PREMIA;
}
#else

/**
 * Characteristics of a product
 */
typedef struct _Product  Product;
struct _Product
{
  double r; /*!< interest rate */
  double divid; /*!< interest rate */
  double S0; /*!< spot */
  double T; /*!< maturity */
  double beta; /*!< volatility */
  double K; /*!< strike */
  int d;//dimension de M, Q et R
```

```
  PnlMatComplex *M;
  PnlMatComplex *Q;
  PnlMatComplex *R;
  PnlMatComplex *Sigma0;
};


//compute (tM+2gamma R Q)
static PnlMatComplex *fonction_mat_inter(dcomplex gamma, const Product *P)
{
  PnlMatComplex *tM;
  PnlMatComplex *res;
  tM = pnl_mat_complex_transpose(P->M); //tM
  res = pnl_mat_complex_mult_mat(P->R, P->Q); //res=R*Q
  pnl_mat_complex_mult_dcomplex(res, RCmul(2, gamma)); //res=2gamma*(RQ)
  pnl_mat_complex_plus_mat(res, tM); //res=tM+2gamma*(RQ)

  pnl_mat_complex_free(&tM);
  return res;
}

//Compute the exponential matrix (14) of the paper
static PnlMatComplex *fonction_expE(dcomplex gamma, const Product *P, PnlMatComp
{
  PnlMatComplex *E11 = pnl_mat_complex_create(P->d, P->d);
  PnlMatComplex *E12;
  PnlMatComplex *E21;
  PnlMatComplex *E22 = pnl_mat_complex_create(P->d, P->d);
  PnlMatComplex *tQ;

  PnlVectComplex *V = pnl_vect_complex_create(P->d);
  PnlMatComplex *E = pnl_mat_complex_create(2 * P->d, 2 * P->d);
  PnlMatComplex *expE = pnl_mat_complex_create(2 * P->d, 2 * P->d);
  dcomplex gg;
  dcomplex _deuxT;
  dcomplex Tcomplex;
  int i;

  Tcomplex = Complex(P->T, 0);

  //création de E11
```

```
pnl_mat_complex_clone(E11, P->M);
pnl_mat_complex_mult_dcomplex(E11, Tcomplex);

//creation de E21
_deuxT = Complex(-2 * P->T, 0);

gg = Cmul(gamma, RCadd(-1, gamma)); //gg=gamma*(gamma-1)
gg = RCmul(0.5, gg); //gg=gamma(gamma-1)/2
for (i = 0; i < P->d; i++)
  {
    pnl_vect_complex_set_real(V, i, Creal(gg));
    pnl_vect_complex_set_imag(V, i, Cimag(gg));
  }
E21 = pnl_mat_complex_create_diag(V);
pnl_mat_complex_mult_dcomplex(E21, Tcomplex);

//creation de E12
tQ = pnl_mat_complex_transpose(P->Q); //tQ
E12 = pnl_mat_complex_mult_mat(tQ, P->Q); //tQQ=tQ*Q
pnl_mat_complex_mult_dcomplex(E12, _deuxT);

//creation de E22
pnl_mat_complex_clone(E22, mat_inter);
pnl_mat_complex_mult_dcomplex(E22, RCmul(-1, Tcomplex));


//creation de la matrice E=(E11 E12:E21 E22)
pnl_mat_complex_set_subblock(E, E11, 0, 0);
pnl_mat_complex_set_subblock(E, E12, 0, P->d);
pnl_mat_complex_set_subblock(E, E21, P->d, 0);
pnl_mat_complex_set_subblock(E, E22, P->d, P->d);


pnl_mat_complex_exp(expE, E);

//pnl_mat_complex_free(&tQQ);
pnl_mat_complex_free(&tQ);
pnl_mat_complex_free(&E11);
pnl_mat_complex_free(&E12);
pnl_mat_complex_free(&E21);
pnl_mat_complex_free(&E22);
```

```
  pnl_mat_complex_free(&E);
  pnl_vect_complex_free(&V);
  return expE;
}

//compute the matrix A(T)
static PnlMatComplex *fonction_A(dcomplex gamma, const Product *P, PnlMatComplex
{
  PnlMatComplex *res;
  PnlMatComplex *A21 = pnl_mat_complex_create(P->d, P->d);
  PnlMatComplex *A22 = pnl_mat_complex_create(P->d, P->d);
  PnlMatComplex *A22_inv = pnl_mat_complex_create(P->d, P->d);
  pnl_mat_complex_extract_subblock(A21, expE, P->d, P->d, 0, P->d);
  pnl_mat_complex_extract_subblock(A22, expE, P->d, P->d, P->d, P->d);
  pnl_mat_complex_inverse(A22_inv, A22);
  res = pnl_mat_complex_mult_mat(A22_inv, A21);
  pnl_mat_complex_free(&A21);
  pnl_mat_complex_free(&A22);
  pnl_mat_complex_free(&A22_inv);
  return res;
}

//compute the value c(T)
static dcomplex fonction_c(dcomplex gamma, const Product *P, PnlMatComplex *expE
{
  dcomplex tmp, Tcomplex;
  PnlMatComplex *mat_tmp = pnl_mat_complex_create(P->d, P->d);
  PnlMatComplex *A22 = pnl_mat_complex_create(P->d, P->d);
  PnlMatComplex *logA22 = pnl_mat_complex_create(P->d, P->d);
  Tcomplex = Complex(P->T, 0);

  pnl_mat_complex_extract_subblock(A22, expE, P->d, P->d, P->d, P->d); //A22

  pnl_mat_complex_log(logA22, A22); //log(A22)

  pnl_mat_complex_clone(mat_tmp, mat_inter); //mat_tmp=(tM+2gamma(RQ))
  pnl_mat_complex_mult_dcomplex(mat_tmp, Tcomplex); //mat_tmp=(tM+2gamma(RQ))T
  pnl_mat_complex_plus_mat(mat_tmp, logA22); //mat_tmp=log(A22)+(tM+2gamma(RQ))T
  tmp = pnl_mat_complex_trace(mat_tmp); //tmp=Tr(log(A22)+(tM+2gamma(RQ))T);

  tmp = RCmul(-P->beta / 2, tmp); //tmp=-beta/2(Tr(log(A22)+(tM+2gamma(RQ))T))
```

```
  tmp = Cadd(tmp, RCmul(P->r * P->T, gamma)); //tmp==-beta/2(Tr(log(A22)+(tM+2ga

  pnl_mat_complex_free(&A22);
  pnl_mat_complex_free(&logA22);
  pnl_mat_complex_free(&mat_tmp);

  return tmp;
}


//compute the characteristic function of Y_T E(e^{i gamma Y_T})
static dcomplex carac(dcomplex gamma, const Product *P)
{
  dcomplex a;
  dcomplex b;
  dcomplex c;
  dcomplex tmp;
  PnlMatComplex *expE;
  PnlMatComplex *mat_inter;
  PnlMatComplex *A, *A_Sigma;
  gamma = Cmul(gamma, Complex(0, 1));

  mat_inter = fonction_mat_inter(gamma, P);
  expE = fonction_expE(gamma, P, mat_inter);
  A = fonction_A(gamma, P, expE);
  A_Sigma = pnl_mat_complex_mult_mat(A, P->Sigma0);
  a = pnl_mat_complex_trace(A_Sigma); //a=Tr(A(T)Sigma0)
  b = RCmul(log(P->S0), gamma); //b=b(T)*ln(S0)
  c = fonction_c(gamma, P, expE, mat_inter); //c(T)
  tmp = Cadd(b, c);
  tmp = Cadd(tmp, a); //tmp=a+b+c

  pnl_mat_complex_free(&A);
  pnl_mat_complex_free(&mat_inter);
  pnl_mat_complex_free(&expE);
  pnl_mat_complex_free(&A_Sigma);
  return Cexp(tmp);
}

//compute the psi function
static dcomplex psi(double v, const Product *P, double alpha)
```

```
{
  dcomplex denom = Complex(alpha * alpha + alpha - v * v, (2 * alpha + 1) * v);
  dcomplex phi = carac(RCsub(v, Complex(0, alpha + 1)), P);

  return Cdiv(RCmul(exp(-P->r * P->T), phi), denom);
}


/* compute the price of the option by FFT */
static double AP_FFT_Wishart(int iscall, const Product *P, int N, double lambda,
{
  dcomplex temp = Complex(0, 0);
  int j;
  int u;
  double prix;
  PnlVectComplex *in = pnl_vect_complex_create_from_zero(N + 1);
  PnlVectComplex *out = pnl_vect_complex_create_from_zero(N + 1);

  for (j = 0; j < N + 1; j++)
    {
      temp = CRmul(Cmul(CIexp(M_PI * (j)), psi(eta * (j), P, alpha)), eta / 3);


      if (j == 0)
        {
          temp = CRmul(temp, 1);
        }
      else if (j == N)
        {
          temp = CRmul(temp, 1);
        }
      else
        {
          temp = CRmul(temp, 3 + pow(-1.0, (double)j + 1));
        }
      pnl_vect_complex_set(in, j, temp);
    }
  pnl_fft(in, out);
  u = 1;
  while (log(P->K) > -M_PI / eta + 2 * M_PI * (u - 1) / (N * eta))
    {
```

```
      u++;
    }
  prix = Creal(CRmul(pnl_vect_complex_get(out, u - 1), exp(-alpha * (-M_PI * (N

  pnl_vect_complex_free(&in);
  pnl_vect_complex_free(&out);
  if (iscall == 1) return prix;
  else return prix + exp(-P->r * P->T) * P->K - P->S0;
}


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

  Product P;
  //paramètres FFT
  int N = 4096;
  double lambda = 0.01;
  double eta = 2 * M_PI / (N * lambda);
  double alpha = 1.1;
  int iscall = (ptOpt->PayOff.Val.V_NUMFUNC_1->Compute == &Call);

  P.r = log(1. + ptMod->R.Val.V_DOUBLE / 100.);
  P.divid = log(1. + ptMod->Divid.Val.V_DOUBLE / 100.);
  P.T = ptOpt->Maturity.Val.V_DATE - ptMod->T.Val.V_DATE;
  P.d = ptMod->size.Val.V_PINT;
  P.beta = ptMod->beta.Val.V_DOUBLE;
  P.S0 = ptMod->S0.Val.V_PDOUBLE;
  P.K = ptOpt->PayOff.Val.V_NUMFUNC_1->Par[0].Val.V_PDOUBLE;
  P.M = pnl_mat_complex_create_from_mat(ptMod->M.Val.V_PNLMAT);
  P.Q = pnl_mat_complex_create_from_mat(ptMod->Q.Val.V_PNLMAT);
  P.R = pnl_mat_complex_create_from_mat(ptMod->Correl.Val.V_PNLMAT);
  P.Sigma0 = pnl_mat_complex_create_from_mat(ptMod->Sigma0.Val.V_PNLMAT);

  Met->Res[0].Val.V_DOUBLE = AP_FFT_Wishart(iscall, &P, N, lambda, eta, alpha);

  pnl_mat_complex_free(&P.M);
  pnl_mat_complex_free(&P.R);
  pnl_mat_complex_free(&P.Q);
```

```
  pnl_mat_complex_free(&P.Sigma0);
  return OK;
}

static int CHK_OPT(AP_FFT_Wishart)(void *Opt, void *Mod)
{
  if ((strcmp(((Option *)Opt)->Name, "CallEuro") == 0) || (strcmp(((Option *)Opt

    return OK;
  return WRONG;
}

#endif //PremiaCurrentVersion
static int MET(Init)(PricingMethod *Met, Option *Opt)
{
  Met->init = 1;
  Met->HelpFilenameHint = "ap_fft";
  return OK;
}

PricingMethod MET(AP_FFT_Wishart) =
{
  "AP_FFT",
  { {" ", PREMIA_NULLTYPE, {0}, FORBID}},
  CALC(AP_FFT_Wishart),
  { {"Price", DOUBLE, {100}, FORBID},
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CHK_OPT(AP_FFT_Wishart),
  CHK_mc,
  MET(Init)
};
```