Help

```
extern "C" {
#include "
href../../../mod/temperedstable1d/temperedstable1d_vol/temperedstable1d_vol_h_sr
#include "pnl/pnl_vector.h"
#include "pnl/pnl_specfun.h"
#include "pnl/pnl_mathtools.h"
#include "pnl/pnl_complex.h"
#include "pnl/pnl_fft.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2017+2) //The "#els
  static int CHK_OPT(AP_TSL_FFT)(void *Opt, void *Mod)
  {
    return NONACTIVE;
  }
  int CALC(AP_TSL_FFT)(void *Opt, void *Mod, PricingMethod *Met)
  {
    return AVAILABLE_IN_FULL_PREMIA;
  }

#else
}

typedef struct params
{
double Cm, Cp;
double G;
double M;
double Ym, Yp;
double mu;
double T;
int N;
double K;
dcomplex U;
} params;

static void Set_params(params *p, double r, double divid, double ap, double am,
{
p->Cp = cp;
p->Cm = cm;
```

```
p->G = lm;
p->M = lp;
p->Yp = ap;
p->Ym = am;
p->T = t;
p->N = n;
p->K = k * k / 10000.0;
p->U = u;
if (ap == 1)
{
p->mu = r - divid + cp * (lp * log(lp) - (lp-1) * log(lp - 1));
}
else{ p->mu = r - divid + pnl_sf_gamma(-ap) * cp * (exp(ap * log(lp)) - exp(ap *
if (am == 1)
{
p->mu = p->mu + cm * (lm * log(lm) - (lm+1) * log(lm + 1));
}
else{ p->mu = p->mu + pnl_sf_gamma(-am) * cm * (exp(am * log(lm)) - exp(am * log

}

static void tslLaplace(params *p, dcomplex u, dcomplex *res)//computation for ps
{
dcomplex aux1, aux2, aux3, aux4;
aux1.r = p->M - u.r;
aux1.i = -u.i;

aux2.r = p->G + u.r;
aux2.i = u.i;

aux3 = Cpow_real(aux1, p->Yp);
aux4 = Cpow_real(aux2, p->Ym);

if (p->Yp == 1)
{
aux3 = Cmul(Clog(aux1),aux3);
res->r = p->mu + p->Cp * (aux3.r - p->M * log(p->M));
res->i = p->Cp * aux3.i;
}
else
{
```

```
res->r = p->mu + p->Cp * pnl_sf_gamma(-p->Yp) * (aux3.r - exp(p->Yp * log(p->M))
res->i = p->Cp * pnl_sf_gamma(-p->Yp) * aux3.i;
}
if (p->Ym == 1)
{
aux4 = Cmul(Clog(aux2), aux4);
res->r = res->r + p->Cm * (aux4.r - p->G * log(p->G));
res->i = res->i + p->Cm * aux4.i;
}
else
{
res->r = res->r + p->Cm * pnl_sf_gamma(-p->Ym) * (aux4.r - exp(p->Ym * log(p->G)
res->i = res->i + p->Cm * pnl_sf_gamma(-p->Ym) * aux4.i;
}
}
```

```
  /*////////////////////////////////////////////////////*/
  int ap_tsl_fft(double S0, double Strike, double T, double r, double divid, dou
  {
    int L=2;
params p;
dcomplex u, aux, aux1;
int n;//Number of working days
double mval;//variance_swap_price;
long int Nx; // Number of space points for FFT
int Nz; // Number z points (z is an average sum of the squared increments)
double xmin, ximin, dx, dxi, zmin, zmax, dz;
PnlVectComplex *psi, *v;
PnlVect *x,*z, *Vz, *Vznew;
double Vmin, Vmax, V0,m0;

int id = -1;
int nT; // number of summands in RV
int ifCall = 1;
int k0;

u.r = 0.0;
u.i = 0.0;
```

```
n = 252;
Nx = pnl_pow_i(2.0, M);
Nz = 80;
Set_params(&p, r, divid, alphap, alpham, lambdap, lambdam, cp, cm, T, n, Strike,

   /////variance swap calculation
mval = 0;
if (alphap == 1)
{
mval = cp / lambdap;
m0 = p.mu -cp * (log(lambdap)+1);
}
else
{
mval = pnl_sf_gamma(2 - alphap) * cp * exp((alphap - 2.0) * log(lambdap));
m0 = p.mu + pnl_sf_gamma(1 - alphap) * cp * exp((alphap - 1.0) * log(lambdap));
}
if (alpham == 1)
{
mval = mval+cm / lambdam;
m0 = m0 + cm * (log(lambdam) + 1);
}
else
{
mval = mval+pnl_sf_gamma(2 - alpham) * cm * exp((alpham - 2.0) * log(lambdam));
m0 = m0 - pnl_sf_gamma(1 - alpham) * cm * exp((alpham - 1.0) * log(lambdam));
}
mval = mval + SQR(m0) / (double)n;


nT =  p.T*n; //// the number of the working days
zmin = 0.;
zmax = p.K / double(n);
dz = Kz*(zmax - zmin) / Nz; ///// uniform Z-grid
//dz = sqrt(8 * (zmax - zmin)) / Nz; //////// non-uniform Z-grid

x = pnl_vect_create(Nx); ///// space points
z = pnl_vect_create(Nz); ///// Z-points
Vz = pnl_vect_create(Nz); /////// prices at t^-_m
Vznew = pnl_vect_create(Nz);
```

```
psi = pnl_vect_complex_create(Nx); //// MGF psi(u)
v = pnl_vect_complex_create(Nx); /////// prices at t^+_m
//////////////////uniform Z-grid with a separate last Z point at 1.////////////
for (int k = 0; k < Nz-1; k++)
{
// LET(z, k) = SQR(k*dz);
LET(z, k) = k*dz;
}
LET(z, Nz - 1) =MAX(1.,Kz*zmax);
/////////////////////////////////end of Z-grid//////////////////

//////////////////////////X and Xi grids setup ///////////////////
//////////////////////////MGF computation//////////////////////
xmin = -L;
dx = -2 * xmin / Nx;

ximin = -M_PI / dx;
dxi = M_2PI / (dx*Nx);
for (long int i = 0; i <Nx; i++)
{
aux.r = 0.;
aux.i = ximin + i*dxi;
tslLaplace(&p, aux, &aux1);
aux1.r /= (double)p.N;
aux1.i /= (double)p.N;
LET_COMPLEX(psi, i) = Cexp(aux1);
LET(x,i) = xmin + i*dx;
}
////////////////end of the block////////////////////////
////////////////the first time step////////////////
for (int k = 0; k < Nz; k++)
{

id = -1;
for (long int i = 0; i < Nx; i++)
{
id *= -1;
aux.i = 0;
aux.r = MAX(0, p.K - ((nT - 1)*GET(z, k) + GET(x, i)*GET(x, i)) / (p.T));//((nT
LET_COMPLEX(v, i) = RCmul(id, aux);
}
```

```
pnl_fft_inplace(v);

for (long int i = 0; i < Nx; i++)
{
LET_COMPLEX(v, i) = Cmul(GET_COMPLEX(v, i), GET_COMPLEX(psi, i));
}
pnl_ifft_inplace(v);
for (long int i = 0; i < Nx; i++)
{
LET_COMPLEX(v, i) = RCmul(PNL_ALTERNATE(i), GET_COMPLEX(v, i));
}

LET(Vz, k) = GET_REAL(v, Nx / 2);
}
////////////////////end of the first time step/////////////////
if (nT > 1)
/////////////// time-stepping cycle//////////////////////////
{
for (int j = 2; j <= nT; j++)
{
for (int k = 0; k < Nz; k++)
{
id = -1;

for (long int i = Nx/2; i < Nx; i++)
{
id *= -1;
aux.i = 0;
aux.r = 0;
V0 = ((nT - j) *GET(z, k) + GET(x, i)*GET(x, i)) / (nT - j + 1);
k0 = 0;
while ((V0 >= GET(z, k0))&(V0 < GET(z, Nz-1)))
{
k0 = k0 + 1;
zmin = GET(z, k0-1);
zmax = GET(z, k0);
Vmin = GET(Vz, k0 - 1);
Vmax = GET(Vz, k0);
aux.r = Vmin + (Vmax - Vmin) / (zmax - zmin)*(V0 - zmin);
}
LET_COMPLEX(v, i) = RCmul(id, aux);
```

```
LET_COMPLEX(v, Nx-i) = GET_COMPLEX(v, i);

}
aux.r = 0;
LET_COMPLEX(v, 0) = aux;
pnl_fft_inplace(v);

for (long int i = 0; i < Nx; i++)
{
LET_COMPLEX(v, i) = Cmul(GET_COMPLEX(v, i), GET_COMPLEX(psi, i));
}
pnl_ifft_inplace(v);
for (long int i = 0; i < Nx; i++)
{
LET_COMPLEX(v, i) = RCmul(PNL_ALTERNATE(i), GET_COMPLEX(v, i));
}

LET(Vznew, k) = GET_REAL(v, Nx / 2);
}
for (int k = 0; k < Nz; k++)
{
LET(Vz, k) = GET(Vznew, k);
}
}
//////////////end of the time stepping cycle//////////////////////////
}
if (ifCall)//Call on RV
{
*ptprice = GET(Vz, 0) + (mval - p.K);

}
else   // Put on RV
{
*ptprice = GET(Vz, 0);
}

*ptprice = exp(-r * T) * (*ptprice);

*ptprice = *ptprice>0 ? sqrt(*ptprice)*100.0 : -1;

pnl_vect_free(&x);
```

```
pnl_vect_free(&z);
pnl_vect_free(&Vz);
pnl_vect_free(&Vznew);
pnl_vect_complex_free(&psi);
pnl_vect_complex_free(&v);


    return OK;
  }

extern "C" {
//-----------------------------------------------------------------------


  int CALC(AP_TSL_FFT)(void *Opt, void *Mod, PricingMethod *Met)
  {
    TYPEOPT *ptOpt = (TYPEOPT *)Opt;
    TYPEMOD *ptMod = (TYPEMOD *)Mod;
    double r, divid, strike,spot;
    NumFunc_1 *p;

    r = log(1. + ptMod->R.Val.V_DOUBLE / 100.);
    divid = log(1. + ptMod->Divid.Val.V_DOUBLE / 100.);
    p = ptOpt->PayOff.Val.V_NUMFUNC_1;
 strike = p->Par[0].Val.V_DOUBLE;
  spot = ptMod->S0.Val.V_DOUBLE;

  return ap_tsl_fft(spot, strike, ptOpt->Maturity.Val.V_DATE - ptMod->T.Val.V_DA

  }

  static int CHK_OPT(AP_TSL_FFT)(void *Opt, void *Mod)
  {
 if ((strcmp(((Option *)Opt)->Name, "CallRealVarEuro") == 0))
      return OK;

    return WRONG;
  }

#endif //PremiaCurrentVersion
  static int MET(Init)(PricingMethod *Met, Option *Opt)
```

```
  {
    static int first = 1;

    if (first)
      {
        Met->Par[0].Val.V_INT = 10;
        Met->Par[1].Val.V_INT=8;

        first = 0;
      }
    return OK;
  }

  PricingMethod MET(AP_TSL_FFT) =
  {
    "AP_TSL_FFT",
    { {"Number of space points (power of two)", INT, {10}, ALLOW    },
      {"Scale parameter in Z-grid", INT, {10}, ALLOW    },
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CALC(AP_TSL_FFT),
    { {"Price in 10000 variance points", DOUBLE, {100}, FORBID},
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CHK_OPT(AP_TSL_FFT),
    CHK_ok ,
    MET(Init)
  } ;

  /*////////////////////////////////////*/
}
```