

Help

```

#include "temperedstable1d_vol.h"
#include "pnl/pnl_vector.h"
#include "pnl/pnl_random.h"
#include "pnl/pnl_specfun.h"
#include "pnl/pnl_mathtools.h"
#include "pnl/pnl_integration.h"
#include "pnl/pnl_laplace.h"
#include "pnl/pnl_fft.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2014+2) //The "#els
static int CHK_OPT(AP_KELLERRESSEL)(void *Opt, void *Mod)
{
    return NONACTIVE;
}
int CALC(AP_KELLERRESSEL)(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;
    //double VS;
    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;
p->mu = r - divid + pnl_sf_gamma(-ap) * cp * (exp(ap * log(lp)) - exp(ap * 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);

    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 + p->Cm * pnl_sf_gamma(-p->Ym)
}

```

```

double RealParts(double y, void *pp) ///Re(exp( psi (iZ sqrt(2u/T))/d ))exp(-z
{
    double z = y;
    double x;
    params *p = (params *) (pp);
    dcomplex aux, aux1, aux2;

    aux = Cpow_real(CRdiv(p->U, p->T), 0.5);
    x = aux.r;
    aux.r = -z * aux.i;
    aux.i = z * x;
}

```

```

    tslLaplace(p, aux, &aux1);
    aux1.r *= 1.0 / (double) p->N;
    aux1.i *= 1.0 / (double) p->N;
    aux2 = Cexp(aux1);

    return (aux2.r * exp(-z * z / 2.0)) / sqrt(2.0 * M_PI);

}

static double ImParts(double y, void *pp)    ///Im(exp( psi (iZ sqrt(2u/T))/d)
{
    dcomplex aux, aux1, aux2;
    double z = y;
    params *p = (params *) (pp);

    aux1 = Cpow_real(CRdiv(p->U, p->T), 0.5);

    aux.r = -z * aux1.i;
    aux.i = z * aux1.r;
    tslLaplace(p, aux, &aux1);
    aux1.r *= 1.0 / (double) p->N;
    aux1.i *= 1.0 / (double) p->N;
    aux2 = Cexp(aux1);

    return (aux2.i * exp(-z * z / 2.0)) / sqrt(2.0 * M_PI);

}

static dcomplex RealVarLaplace(params *p, dcomplex u) // E exp( psi (iZ sqrt(2u/T)
{
    dcomplex res ;
    PnlFunc Real, Im;
    int N;
    p->U = u;
    Real.F = &RealParts;
    Real.params = p;

    Im.F = &ImParts;
    Im.params = p;

```

```

    N = 500;

    res.r = pnl_integration(&Real, -5.0, 5.0, N , "simpson");
    res.i = pnl_integration(&Im, -5.0, 5.0, N , "simpson");

    return res;
}

static double P1(double u, params *p)
{
    dcomplex z, aux1, aux2;

    aux1.r = 1.0;
    aux1.i = u;
    z = Cpow_real(RealVarLaplace(p, CRmul(aux1, 2.0)), (double) p->N * p->T);

    aux2.r = p->K;
    aux2.i = p->K * u;
    aux2 = Cdiv(Cexp(aux2), Cpow_real(aux1, 2.0));

    return Creal(Cmul(aux2, z));
}

static double P1TSL(double u, void *pp)
{
    params *p = (params *) (pp);

    return P1(u, p) / M_PI ;
}

//-----
static int ap_kellerresel(double S0, double Strike, double T, double r, double d
{
    params p;
    PnlFunc Call;
    int N;
    dcomplex u;

```

```

double up;
int n;//Number of working days
double mval;
//double variance_swap_price;

u.r = 0.0;
u.i = 0.0;
n = 252;
Set_params(&p, r, divid, alphap, alpham, lambdap, lambdam, cp, cm, T, n, Strik

Call.F = &P1TSL;
Call.params = &p;
up = 100.0;
N = 400;

mval = pnl_sf_gamma(2 - alphap) * cp * exp((alphap - 2.0) * log(lambdap)) + pn

//variance_swap_price=sqrt(mval)*100,exp(-r*T)*(mval*10000-Strike*Strike);

*ptprice = pnl_integration(&Call, 0, up, N , "simpson") + (mval - p.K);

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

*ptprice = sqrt(*ptprice) * 100.0;

return OK;
}

int CALC(AP_KELLERRESSEL)(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_kellerresel(spot, strike, ptOpt->Maturity.Val.V_DATE - ptMod->T.Val.

}

static int CHK_OPT(AP_KELLERRESSEL)(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)
    {
        first = 0;
        Met->HelpFilenameHint = "ap_kellerresel";
    }
    return OK;
}

PricingMethod MET(AP_KELLERRESSEL) =
{
    "AP_KELLERRESSEL",
    { {" ", PREMIA_NULLTYPE, {0}, FORBID}},
    CALC(AP_KELLERRESSEL),
    { {"Price in 10000 variance points", DOUBLE, {100}, FORBID},
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CHK_OPT(AP_KELLERRESSEL),
    CHK_ok ,
    MET(Init)
} ;

```