

Help

```

#include "nig1d_std.h"
#include "pnl/pnl_cdf.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion <
    (2011+2) //The "#else" part of the code will be freely av
    ailable after the (year of creation of this file + 2)
static int CHK_OPT(AP_spmNIG)(void *Opt, void *Mod)
{
    return NONACTIVE;
}
int CALC(AP_spmNIG)(void*Opt,void *Mod,PricingMethod *Met)
{
    return AVAILABLE_IN_FULL_PREMIA;
}
#else

//static double alpha, beta, delta, mu, r, divid, T, S0,
//    strike;
//static double uu, vv, ww, rho1, rho2;
//static long int points;

// ===== SPM approx =====
// =====
static double mgf_exp(int ind, double T, double r, double
    logs, double alpha, double beta, double delta, double mu,
    double xi)
{
    double ans;

    ans = (logs+T*mu)*xi - delta*T*( sqrt( alpha*alpha - (bet
        a+xi)*(beta+xi) ) - sqrt(alpha*alpha - beta*beta ) ) - ind*
        ( logs+r*T );

    return ans;
}
// -----
// -----
static double secderiv(double T, double alpha, double beta,
    double delta, double mu, double x)

```

```

{
    double a2, b2;
    a2 = alpha*alpha;
    b2 = (beta+x)*(beta+x);
    return delta*T*a2 / ( (a2-b2)*sqrt(a2-b2) );
}
// -----
-----
static double thirdderiv(double T, double alpha, double bet
    a, double delta, double mu, double x)
{
    double a2, b2;
    a2 = alpha*alpha;
    b2 = (beta+x)*(beta+x);
    return 3.*delta*T*(beta+x)*a2 /( (a2-b2)*(a2-b2)*sqrt(a2
        -b2) );
}
// -----
-----
static double fourthderiv(double T, double alpha, double
    beta, double delta, double mu, double x)
{
    double a2, b2, ab;
    a2 = alpha*alpha;
    b2 = (beta+x)*(beta+x);
    ab = a2 - b2;
    return 3.*delta*T*a2*(a2+4.*b2) /( ab*ab*ab*sqrt(ab) );
}

// -----
-----
static double spmapprox(int ind, double spot, double strk,
    double ti, double ri, double dividi, double alpha, double beta,
    double delta, double mu, double xi)
{
    double mgval, logs, logk;
    double uu, ww, deriv2, deriv3,prob;

    logs = log(spot);
    logk = log(strk);

```

```

// s-p for K
uu= xi-ind;
if(uu==0.)
{
    deriv2 = secderiv(ti, alpha, beta, delta, mu, xi);
    deriv3 = thirdderiv(ti, alpha, beta, delta, mu, xi);
    prob = 0.5 - deriv3/( 6.*deriv2*sqrt(2.*M_PI*deriv2)
    );
    return prob;
}
// s-p for Ko
mgval = mgf_exp(ind, ti, ri, logs, alpha, beta, delta,
    mu, xi);

ww= 2.0*(uu*logk - mgval);

if(ww>0.) { ww = sqrt( ww );}
else { printf("Cannot compute: ww<0!\n"); ww= 1.; }
if(uu<0.){ww *= -1.;}

deriv2 = secderiv(ti, alpha, beta, delta, mu, xi);
deriv3 = thirdderiv(ti, alpha, beta, delta, mu, xi);
//deriv4 = fourthderiv(ti, alpha, beta, delta, mu, xi);

// probability approx Luganini-Rice formula
if(deriv2>0.) {
    //znam = uu*sqrt(deriv2);
    //addterm = (deriv4/(deriv2*deriv2)/8. - 5.*deriv3*de
    riv3/(deriv2*deriv2*deriv2)/24.)/znam - deriv3/(deriv2*sqrt(
    deriv2))/4./(znam*znam) - 1./(znam*znam*znam) + 1./(ww*ww*
    ww) ;
    prob = 1. - cdf_nor(ww) + pnl_normal_density(ww)*( 1.
    0/(uu*sqrt(deriv2)) - 1.0/ww /*+ addterm*/);
}
else {printf("Cannot compute: second derivative is negat
    ive! %e\n", deriv2); prob=0.;}

return prob;
}
//-----
-----

```

```

static double optimalpar_nig(double alpha, double beta,
    double delta, double mu, double ti, double logk)
{
    double lkt = logk/ti - mu;
    return alpha*lkt / sqrt(delta*delta + lkt*lkt) - beta;
}
// -----
-----
static int spmcall(int ifCall, double ti, double ri,
    double dividi, double spot, double strk, double sigma, double th
    eta, double kappa, double *ptprice)
{
    double alpha, beta, delta, mu, optimalsigma, prob0, prob
    1, spmprice;
    double logs, logk;
    double sig2=sigma*sigma;

    alpha=sqrt(theta*theta+sig2/kappa)/sig2;
    beta=theta/sig2;
    delta=sigma/sqrt(kappa);
    logs = log(spot);
    logk = log(strk);

    mu = (ri - dividi) + delta*( sqrt(alpha*alpha-(beta+1.)
        *(beta+1.)) - sqrt(alpha*alpha - beta*beta) );

    optimalsigma = optimalpar_nig(alpha, beta, delta, mu,
        ti, logk-logs);

    prob0=spmapprox(0, spot, strk, ti, ri, dividi, alpha,
        beta, delta, mu, optimalsigma );
    prob1=spmapprox(1, spot, strk, ti, ri, dividi, alpha,
        beta, delta, mu, optimalsigma );
    spmprice = -(exp(-ri*ti)*strk*prob0 - spot*prob1);
    if (!ifCall)
    {
        spmprice = spmprice-spot+strk*exp(-ri*ti);
    }
    *ptprice = spmprice;

    return OK;
}

```

```

}

int CALC(AP_spmNIG)(void*Opt,void *Mod,PricingMethod *
    Met)
{
    TYPEOPT* ptOpt=(TYPEOPT*)Opt;
    TYPEMOD* ptMod=(TYPEMOD*)Mod;
    NumFunc_1 *p;
    double r,divid, tt, s0, strk, sigma, theta, kappa;
    int res;
    int ifCall;

    r=log(1.+ptMod->R.Val.V_DOUBLE/100.);
    divid=log(1.+ptMod->Divid.Val.V_DOUBLE/100.);
    tt=ptOpt->Maturity.Val.V_DATE-ptMod->T.Val.V_DATE;
    s0 = ptMod->S0.Val.V_PDOUBLE;
    p= ptOpt->PayOff.Val.V_NUMFUNC_1;
    ifCall=((p->Compute)==&Call);
    strk= p->Par[0].Val.V_DOUBLE;
    sigma=ptMod->Sigma.Val.V_PDOUBLE;
    theta=ptMod->Theta.Val.V_DOUBLE;
    kappa= ptMod->Kappa.Val.V_SPDOUBLE;

    res = spmcall(ifCall, tt, r, divid, s0, strk, sigma, th
        eta, kappa, &(Met->Res[0].Val.V_DOUBLE));

    return res;
}

static int CHK_OPT(AP_spmNIG)(void *Opt, void *Mod)
{
    if ( (strcmp( ((Option*)Opt)->Name,"CallEuro")==0) || (
        strcmp( ((Option*)Opt)->Name,"PutEuro")==0) )
        return OK;

    return WRONG;
}

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

```

```
{
    if ( Met->init == 0)
    {
        Met->init=1;
        Met->HelpFilenameHint = "ap_spm_nig";
    }

    return OK;

}

PricingMethod MET(AP_spmNIG)=
{
    "AP_SaddlePoint_NIG",
    {{ " ",PREMIA_NULLTYPE,{0},FORBID}},
    CALC(AP_spmNIG),
    {{ "Price",DOUBLE,{100},FORBID},{ " ",PREMIA_NULLTYPE,{0}
    ,FORBID}},
    CHK_OPT(AP_spmNIG),
    CHK_ok,
    MET(Init)
} ;
```

References