

Help

```

#include <stdlib.h>
#include "nig1d_pad.h"
#include "pnl/pnl_vector.h"
#include "pnl/pnl_matrix.h"
#include "pnl/pnl_complex.h"
#include "pnl/pnl_mathtools.h"
#include "pnl/pnl_fft.h"
#include "math/ap_fusai_levy/DiscreteAsianFMM.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion <
    (2012+2) //The "#else" part of the code will be freely av
    ailable after the (year of creation of this file + 2)
int CALC(AP_Aasian_FMMNIG)(void *Opt,void *Mod,Pricing
    Method *Met)
{
return AVAILABLE_IN_FULL_PREMIA;
}
#else
//-----
-----
static dcomplex cfNIG(double dt, dcomplex g, PnlVect *Para
    meters)
{
//-----
-----
// NIG Characteristic Function
//-----
-----
double alpha,beta,delta;
dcomplex term1, term2;
dcomplex charexp;

alpha=pnl_vect_get(Parameters,0);
beta=pnl_vect_get(Parameters,1);
delta=pnl_vect_get(Parameters,2)*dt;

term1=Cmul(g,Complex(0,1));
term1=Cadd(Complex(beta,0),term1);
term1=Cpow_real(term1,2);
term1=Csub(Complex(SQR(alpha),0),term1);

```

```

    term1=Csqrt(term1);
    term2=Complex(sqrt(SQR(alpha)-SQR(beta)),0);
    term1=Csub(term1,term2);
    term1=RCmul(-delta,term1);
    charexp=Cexp(term1);

    return charexp;
}
//-----
    -----
static double MomentsNIG(int moment, double rf, double dt,
    PnlVect *Parameters)
{
    //-----
    -----
    // compute moments of the NIG model
    //-----
    -----
    double alpha,beta,delta;
    double mom=0.;

    alpha=pnl_vect_get(Parameters,0);
    beta=pnl_vect_get(Parameters,1);
    delta=pnl_vect_get(Parameters,2);

    if(moment==1){
        mom=(dt*(beta*delta + (-SQR(alpha)+SQR(beta)+sqrt((
            SQR(alpha)-SQR(beta))*(SQR(alpha)-(1+beta)*(1+beta))))*delt
            a+sqrt(SQR(alpha)-SQR(beta))*rf))/sqrt(SQR(alpha)-SQR(beta)
            );
    }
    if(moment==2){
        mom=(SQR(beta)*delta*dt)/(POW(SQR(alpha)-SQR(beta),1
            .5))+(delta*dt)/sqrt(SQR(alpha)-SQR(beta)) + SQR(dt*((bet
            a*delta)/sqrt(SQR(alpha)-SQR(beta))-sqrt(SQR(alpha)-SQR(bet
            a))*delta+sqrt(SQR(alpha)-SQR(1+beta))*delta+rf));
    }

    return mom;
}
//-----

```

```

-----
static dcomplex charfunction(double r, double divid,
    double dt, dcomplex g, PnlVect *Parameters)
{ //-----
    -----
    // Levy Characteristic Function
    //-----
    -----

    double m;
    dcomplex result, mdtg, temp;

    temp=cfNIG(dt,Complex(0.,-1.), Parameters);
    m = Creal(Csub(Complex((r-divid)*dt, 0.), Clog(temp)));
    mdtg = Cmul(Complex(0., m), g);
    temp=cfNIG(dt, g, Parameters);
    result=Cmul(Cexp(mdtg), temp);

    return result;
}//-----
-----
static double BoundUpperTailLevy(double x, double rf,
    double divid, double dt, int maxmoment, PnlVect *Parameters)
{
    //-----
    -----
    // compute upper truncation
    //-----
    -----

    double minup,bound;
    int i;

    minup = 1.0;
    for(i = 1;i < maxmoment + 1; i++){
        bound = Creal(charfunction(rf, divid, dt, Complex(
            0,-i), Parameters))/exp(x*i);
        minup = MIN(minup, bound);
    }
    return minup;
}
//-----

```

```

-----
static double BoundLowerTailLevy(double x, double rf,
    double divid, double dt, int maxmoment, PnlVect *Parameters)
{
    //-----
    -----
    // compute lower truncation
    //-----
    -----
    double minlow, bound;
    int i;
    minlow = 1.0;

    for(i = 1; i < maxmoment + 1; i++){
        bound = Creal(charfunction(rf, divid, dt, Complex(
            0,i), Parameters))/exp(x*i);
        minlow = MIN(minlow, bound);
    }
    return minlow;
}
//-----
-----

static double findlowuplimit(double rf, double dt, PnlVect
    *Parameters)
{
    //-----
    -----
    // Truncate the transition density domain
    //-----
    -----
    double mom1, mom2, levylow, levyup, bound;
    int maxnummoments, lowfactor, upfactor;

    maxnummoments=10;
    lowfactor=5;
    upfactor=5;

    mom1=MomentsNIG(1, rf, dt, Parameters);
    mom2=MomentsNIG(2, rf, dt, Parameters);

```

```

levylow=mom1-lowfactor*POW(mom2-mom1*mom1,0.5);
    bound=BoundLowerTailLevy(-levylow, rf, 0., dt, maxnummoments, Parameters);
while(bound>POW(10.0,-8.0))
{
    lowfactor=lowfactor+1;
    levylow=mom1-lowfactor*POW(mom2-mom1*mom1,0.5);
    bound=BoundLowerTailLevy(-levylow, rf, 0.,dt, maxnummoments, Parameters);
}

levyup=mom1+upfactor*POW(mom2-mom1*mom1,0.5);
    bound=BoundUpperTailLevy(levyup, rf, 0., dt, maxnummoments, Parameters);
while(bound>POW(10.0,-8.0))
{
    upfactor=upfactor+1;
    levyup=mom1+upfactor*POW(mom2-mom1*mom1,0.5);
    bound=BoundUpperTailLevy(levyup, rf, 0., dt, maxnummoments, Parameters);
}

return MAX(ABS(levylow),levyup);
}
//-----
-----
static double truncate(double r, double divid, double dt,
    PnlVect *Parameters)
{
    //-----
    -----
    // find u for which cf(u)<10^-10
    //-----
    -----

    double abs_cf,step,umax;
    dcomplex cf;

    step = 1.5;
    umax = 5.0;
    cf = charfunction(r, divid,dt, Complex(umax, 0), Para

```

```

    meters);
abs_cf = sqrt(Creal(cf)*Creal(cf) + Cimag(cf)*Cimag(cf))
;

while (abs_cf > POW(10., -10.))
{
    umax = umax * step;
    cf = charfunction(r, divid, dt, Complex(umax, 0),
Parameters);
    abs_cf = sqrt(Creal(cf)*Creal(cf) + Cimag(cf)*Cimag(
cf)); //compute abs error
}

return (umax + umax / step) / 2;
}
//-----
-----
static void kernel(double r, double divid, double dt, lon
g N, double b, PnlVect *Parameters, PnlVect *inv, PnlVect *
logk)
{
    //-----
    -----
    // Compute the transition density function by using the
    Fractional Fourier Transform
    //-----
    -----
    int j;
    double wj,eta,alpha,dx,umax;
    dcomplex term1,ft,aa,a,cgyz;
    PnlVectComplex *y1, *y2;

    y1=pnl_vect_complex_create(2*N);
    y2=pnl_vect_complex_create(2*N);

    // bound of characteristic function grid
    umax = truncate(r, divid, dt, Parameters);

    // bound of the density function grid
    b=b*1.25;

```

```

// grids's steps
eta = umax / N;
dx = 2 * b / N;
alpha = eta * dx / (2 * M_PI);

for (j=0; j<=N-1; j++)
{
    // trapezoidal quadrature weights
    if( (j == 0) || j == (N - 1)){
        wj = 0.5*eta;
    }
    else {
        wj = eta;
    }
    a = Complex(cos(SQR(j) * alpha * M_PI), sin(SQR(j) *
alpha * M_PI));
    pnl_vect_complex_set(y2,j,a);
    aa = Complex(cos(SQR(N-j) * alpha * M_PI), sin(SQR(N-
j) * alpha * M_PI));
    pnl_vect_complex_set(y2,j+N,aa);

    ft = charfunction(r, divid, dt, Complex(j * eta, 0),
Parameters);
    term1 = Cexp(Complex(0,b*eta* j));
    ft = Cmul(term1, ft);
    pnl_vect_complex_set(y1,j,Cmul(RCdiv(wj,a),ft));
    pnl_vect_complex_set(y1,j+N,CZERO);
}
pnl_fft_inplace(y1);
pnl_fft_inplace(y2);
pnl_vect_complex_mult_vect_term(y1,y2);

//FFT inversion
pnl_ifft_inplace(y1);
for( j = 0; j<= N - 1;j++)
{
    a = Complex(cos(SQR(j) * alpha * M_PI), sin(SQR(j) *
alpha * M_PI));
    cgyz = Cdiv(pnl_vect_complex_get(y1,j),CRmul(a,M_PI));
    pnl_vect_set(logk,j,-b+j*dx);
    pnl_vect_set(inv,j,Creal(cgyz));
}

```

```

    }
    pnl_vect_complex_free(&y1);
    pnl_vect_complex_free(&y2);
}
//-----
-----

static int FMMNIG_Aasian(double pseudo_stock,double pseudo_
    strike,NumFunc_2 *po,double t,double r,double divid,double
    sigma,double theta,double kappa,int Mdates,int N,double *pt
    price,double *ptdelta)
{
    int flagCP=0,asian_type=0;
    //-----
    -----
    //Compute price and delta of an Asian call option under
    the NIG process
    // RECURSIVE PROCEDURE
    //-----
    -----
    //Recursive approach proposed in
    //Fusai, Marazzina, Marena, SIAM JOURNAL OF FINANCIAL
    MATHEMATICS, 2011
    //-----
    -----
    int c,i,j,Ni,start,count,flag,startcol,max_len_b;
    long nfft;
    double dt,low,up,b,x,y,xy,h,price,delta;
    double alphaNIG,betaNIG,deltaNIG,temp;
    PnlVect *CoeffLambda, *abscissa, *weights, *a_temp, *w_
        temp, *xdens, *dens, *Parameters, *vector, *vector1;
    PnlMat *Kmatrix;

    //-----
    -----
    //Call Fixed
    if((po->Compute)==&Call_OverSpot2)
    {flagCP=0;
        asian_type=0;
    }
    //Put Fixed

```



```

else if((po->Compute)==&Put_OverSpot2)
{
    flagCP=1;
    asian_type=0;
}
//Call Floating
else if((po->Compute)==&Call_StrikeSpot2)
{
    flagCP=0;
    asian_type=1;
}
//Put Floating
else if((po->Compute)==&Put_StrikeSpot2)
{
    flagCP=1;
    asian_type=1;
}
temp=1/(kappa*SQR(sigma));
betaNIG=theta*kappa*temp;
alphaNIG=sqrt(SQR(betaNIG)+temp);
deltaNIG=1/(kappa*sqrt(temp));
Parameters=pnl_vect_create_from_list(3,alphaNIG,betaNIG,
    deltaNIG);
// STEP 0: PREPARE GRID AND COEFFICIENTS
//-- payoff coefficients
if (asian_type==0) {
    CoeffLambda=pnl_vect_create_from_double(2,1./(Mdates+1));
    pnl_vect_set(CoeffLambda,0,pnl_vect_get(CoeffLambda,0)
        -pseudo_strike/pseudo_stock);
    c=0;
}
else{
    CoeffLambda=pnl_vect_create_from_double(2,-1./(Mdates+1));
    c=-1;
}

//-- price grid
low=-(3./2+30./Mdates); //lower bound
up=pnl_vect_get(CoeffLambda,1); //upper bound
//-- generate abscissa and weights for quadrature
if (asian_type==0){
    Ni=N; //number of nodes for x<0

```

```

N=Ni+N; //total number of nodes
abscissa=pnl_vect_create_from_zero(N);
weights=pnl_vect_create_from_zero(N);
a_temp=pnl_vect_create_from_zero(Ni);
w_temp=pnl_vect_create_from_zero(Ni);
gauleg_pn(low,0,a_temp,w_temp,Ni);
for (i=0;i<Ni;i++){
    x=pnl_vect_get(a_temp,i);
    y=pnl_vect_get(w_temp,i);
    pnl_vect_set(abscissa,i,x);
    pnl_vect_set(weights,i,y);
    //--- we consider the same number of nodes
    pnl_vect_set(abscissa,i+Ni,(x-low)*up/(-low));
    pnl_vect_set(weights,i+Ni, y*up/(-low));
}
pnl_vect_free(&a_temp);
pnl_vect_free(&w_temp);
}
else{
    abscissa=pnl_vect_create_from_zero(N);
    weights=pnl_vect_create_from_zero(N);
    gauleg_pn(low,up,abscissa,weights,N);
    Ni=N;
}

//-- time grid
dt=t/Mdates;

//-- compute the transition density
b=findlowuplimit(r,dt,Parameters);
nfft=32768;
dens=pnl_vect_create_from_zero(nfft);//contains the density
xdens=pnl_vect_create_from_zero(nfft);//contains the abscissa of the density
kernel(r,divid,dt,nfft,b,Parameters,dens,xdens);

// STEP 2: CREATE MATRICES AND VECTORS
vector=pnl_vect_create_from_zero(N);
for (j=0; j<=N-1; j++) {
    x=pnl_vect_get(abscissa,j)-(double)c;

```

```

    if (x>0.0)
        pnl_vect_set(vector,j,x);
}

// MATRIX
startcol=0; max_len_b=0; count=0;
Kmatrix=pnl_mat_create_from_double(Ni,N+2,0.0);
for (i=0; i<=Ni-1; i++){
    flag=0; start=0;
    x=pnl_vect_get(abscissa,i);
        for (j=startcol; j<=N-1; j++){
            y=pnl_vect_get(abscissa,j);
            xy=log(x/(y-pnl_vect_get(CoeffLambda,1)));
            if (ABS(xy)<=b){
                if (flag==0){
                    flag=1; // start to fill the row
                    pnl_mat_set(Kmatrix,i,0,j); // first element
of the band
                    startcol=j;
                    count=1;
                }
                count=count+1;
                xy=MAX(interp_lin(xy, nfft, &start, xdens, dens)
,0.0);
                pnl_mat_set(Kmatrix,i,count,-exp(-r*dt)*xy*(x/SQ
R(y-pnl_vect_get(CoeffLambda,1))));
                if (j==N-1){ // stop to fill the row (since it
is finished)
                    pnl_mat_set(Kmatrix,i,1,(N-1)-(int)pnl_mat_get(
Kmatrix,i,0)+1); // length of the band
                    max_len_b=MAX(max_len_b,(int)pnl_mat_get(Kmatr
ix,i,1));
                }
            }else if (flag==1){ // stop to fill the row
                pnl_mat_set(Kmatrix,i,1,(j-1)-(int)pnl_mat_get(
Kmatrix,i,0)+1); // length of the band
                max_len_b=MAX(max_len_b,(int)pnl_mat_get(Kmatrix
,i,1));
                break;
            }
        }
}

```

```
}

```

```
// STEP 3: RECURSIVE APPROACH
vector1=pnl_vect_create_from_zero(Ni);
if (asian_type==0){
    for(i=0;i<Mdates;i++){
        pnl_vect_mult_vect_term(vector,weights);
        bmat_mult_vect(Kmatrix,vector,vector1,Ni,N+2);
        for (j=0;j<Ni;j++){
            pnl_vect_set(vector,j,pnl_vect_get(vector1,j));
            xy=pnl_vect_get(CoeffLambda,1)*exp((r-divid)*dt)*(1
            -exp((i+1)*(r-divid)*dt))/(1-exp((r-divid)*dt));
            for (j=Ni;j<N;j++){
                pnl_vect_set(vector,j,exp(-r*(i+1)*dt)*(pnl_vect
                _get(abscissa,j)+xy));
            }
        }
    }
}else{
    for(i=0;i<Mdates;i++){
        pnl_vect_mult_vect_term(vector,weights);
        bmat_mult_vect(Kmatrix,vector,vector1,Ni,N+2);
        pnl_vect_clone(vector,vector1);
    }
}

xy=interp_lin1(pnl_vect_get(CoeffLambda,0),N,abscissa,vector);
price=pseudo_stock*xy;
if (asian_type==0){
    h=exp((up-low)/N);
    x=(pseudo_stock-h)*interp_lin1(pnl_vect_get(CoeffLambda,1)-pseudo_strike/(pseudo_stock-h),N,abscissa,vector);
    y=(pseudo_stock+h)*interp_lin1(pnl_vect_get(CoeffLambda,1)-pseudo_strike/(pseudo_stock+h),N,abscissa,vector);
    xy=(y-x)/(2*h);
}
delta=xy;

if (flagCP==1){
    price=price-pseudo_stock*(pnl_vect_get(CoeffLambda,0)*
    exp(-r*Mdates*dt)-c);
}
```

```

delta=delta-(pnl_vect_get(CoeffLambda,1)*exp(-r*Mdates*
    dt)-c);
for(i=0;i<Mdates;i++)
{
    price=price-pseudo_stock*pnl_vect_get(CoeffLambda,1)*
    exp(-r*i*dt);
    delta=delta-pnl_vect_get(CoeffLambda,1)*exp(-r*i*dt);
}
}

*ptprice=price;
*ptdelta=delta;
//-----DESTROY-----
-----
pnl_vect_free(&xdens);
pnl_vect_free(&dens);
pnl_vect_free(&Parameters);
pnl_vect_free(&vector);
pnl_vect_free(&vector1);
pnl_mat_free(&Kmatrix);
pnl_vect_free(&abscissa);
pnl_vect_free(&weights);
pnl_vect_free(&CoeffLambda);

return OK;
}

int CALC(AP_Aasian_FMMNIG)(void *Opt,void *Mod,Pricing
    Method *Met)
{
    TYPEOPT* ptOpt=(TYPEOPT*)Opt;
    TYPEMOD* ptMod=(TYPEMOD*)Mod;

    int return_value;
    double r,divid,time_spent,pseudo_spot,pseudo_strike;
    double t_0, T_0;

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

    T_0 = ptMod->T.Val.V_DATE;

```

```

t_0= (ptOpt->PathDep.Val.V_NUMFUNC_2)->Par[0].Val.V_PDOUB
LE;

if(T_0 < t_0)
{
    Fprintf(TOSCREEN,"T_0 < t_0, untreated case{n{n{n");
    return_value = WRONG;
}
/* Case t_0 <= T_0 */
else
{
    time_spent=(ptMod->T.Val.V_DATE-(ptOpt->PathDep.Val.
V_NUMFUNC_2)->Par[0].Val.V_PDOUBLE)/(ptOpt->Maturity.Val.V_
DATE-(ptOpt->PathDep.Val.V_NUMFUNC_2)->Par[0].Val.V_PDOUB
LE);
    pseudo_spot=(1.-time_spent)*ptMod->S0.Val.V_PDOUBLE;
    pseudo_strike=(ptOpt->PayOff.Val.V_NUMFUNC_2)->Par[0]
.Val.V_PDOUBLE-time_spent*(ptOpt->PathDep.Val.V_NUMFUNC_2)
->Par[4].Val.V_PDOUBLE;

return_value= FMMNIG_Aasian(pseudo_spot,pseudo_strike,pt
Opt->PayOff.Val.V_NUMFUNC_2,ptOpt->Maturity.Val.V_DATE-ptMod-
>T.Val.V_DATE,r,divid,ptMod->Sigma.Val.V_PDOUBLE,ptMod->Th
eta.Val.V_DOUBLE,ptMod->Kappa.Val.V_SPDOUBLE,Met->Par[0].
Val.V_INT2,Met->Par[1].Val.V_INT2,&(Met->Res[0].Val.V_
DOUBLE),&(Met->Res[1].Val.V_DOUBLE));
}

return return_value;
}
#endif //PremiaCurrentVersion

static int CHK_OPT(AP_Aasian_FMMNIG)(void *Opt, void *Mod)
{
    if ( (strcmp(((Option*)Opt)->Name,"AsianCallFixedEuro")
==0) || (strcmp( ((Option*)Opt)->Name,"AsianPutFixedEuro")
==0)|| (strcmp(((Option*)Opt)->Name,"    AsianCallFloatingEuro")==0) || (str
return OK;
return WRONG;
}

```

```

static int MET(Init)(PricingMethod *Met,Option *Opt)
{
    if ( Met->init == 0)
    {
        Met->init=1;
        Met->Par[0].Val.V_INT2=12;
        Met->Par[1].Val.V_INT2=3000;
    }
    return OK;
}

PricingMethod MET(AP_Asian_FMMNIG)=
{
    "AP_Asian_FMM_NIG",
    {"Nb.of Monitoring Dates",INT2,{2000},ALLOW },
    {"Nb.of Integration Points ",INT2,{1000},ALLOW},
    {" ",PREMIA_NULLTYPE,{0},FORBID}},
    CALC(AP_Asian_FMMNIG),
    {"Price",DOUBLE,{100},FORBID},{"Delta",DOUBLE,{100},FO
    RBID} ,{" ",PREMIA_NULLTYPE,{0},FORBID}},
    CHK_OPT(AP_Asian_FMMNIG),
    CHK_ok,
    MET(Init)
};

```

References