

## [Help](#)

```
//Pricing Discrete Asian Options
//according to Fusai-Meucci

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2008+2) //The "#els
#else

#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <
href../../../../common/math/cdo/cdo_math_h_src.pdfmath.h>
#include <ctype.h>
#include <time.h>

#include "
href../../../../common/math/ap_fusai_levy/nrutil_h_src.pdfnrutil.h"
#include "pnl/pnl_complex.h"
#include "../moments.h"
#include "
href../../../../common/math/ap_fusai_levy/fractional_fft_h_src.pdffractional_fft.h"
#include "
href../../../../common/math/ap_fusai_levy/spline_h_src.pdfspline.h"
#include "
href../../../../common/math/ap_fusai_levy/linalg_h_src.pdflinalg.h"
#include "
href../../../../common/math/ap_fusai_levy/rncf_h_src.pdfrncf.h"
#include "pnl/pnl_mathtools.h"

double DiscreteAsian(int model,//modello
                    double spot,
                    double strike,
                    double rf,
                    double dt,
                    int ndates,
                    double lowlim,
                    double uplim,
                    int npoints,//n. of quadrature points
                    long nfft,//n. of points for the fft inversion
                    double ModelParameters[], //the parameters of the model
```

```

        double price[],
        double solution[], double *delta) //OUTPUT: Contains the so
{

    int i, j, k;
    double inter_dens = 0.;
    double upperdens, b;
    //parameters for spline interpolation
    int ier;
    double dfb, ddfb, arg;
    //vectors where to store the outputs of the FFT inversion
    double *inv, *logk;
    //abscissa and weights for Gaussian quadrature with npoints
    double *abscissa, *weights, *temp;
    double **c, ** kernelmatrix;
    double optprice, optdelta;
    double gamma_price;

    inv = dvector(0, nfft - 1); //contains the density
    logk = dvector(0, nfft - 1); //contains the abscissa of the density

    c = dmatrix(0, nfft - 1, 0, 2);
    kernelmatrix = dmatrix(0, npoints - 1, 0, npoints - 1);

    abscissa = dvector(1, npoints);
    weights = dvector(1, npoints);
    temp = dvector(0, npoints - 1);

    //Generate abscissa and weights for quadrature
    gauleg(lowlim, uplim, abscissa, weights, npoints);

    b = MAX(fabs(lowlim - log(exp(uplim) + 1)), fabs(uplim - log(exp(lowlim) + 1)));
    b = b * 1.1;
    upperdens = b;
    TableIFRT(1, model, rf, dt, nfft, b, 1.5, ModelParameters, inv, logk);

    //spline interpolation
    i = spline(logk, inv, nfft, c);
    if (i > 100) return i;

```

```

//construct the kernel matrix
for (i = 1; i <= npoints; i++)
{
    for (j = 1; j <= npoints; j++)
    {
// argument of the density
        arg = abscissa[i] - log(exp(abscissa[j]) + 1);
        if (arg > -upperdens)
        {
            if (arg < upperdens)
            {
                inter_dens = MAX(splevl(arg, nfft, logk, inv, c, &dfb, &ddfb,
            }
        }

        if (arg < -upperdens) inter_dens = 0.0;

        if (arg > upperdens) inter_dens = 0.0;
//construct the kernel
        kernelmatrix[i - 1][ j - 1] = weights[j] * MAX(inter_dens, 0.0);
        /*printf("%d %d %d %f\ n",npoints,i-1,j-1,kernelmatrix[i-1][ j-1]);*/
    }
}
//construct the initial condition

    arg = abscissa[i];
    if (arg > -upperdens)
    {
        if (arg < upperdens)
        {
            solution[i - 1] = MAX(splevl(abscissa[i], nfft, logk, inv, c, &dfb
        }
    }

    if (arg < -upperdens) solution[i - 1] = 0.0;

    if (arg > upperdens) solution[i - 1] = 0.0;
}
//iterations over the monitoring dates
for (k = 1; k < ndates; k++)
{

```

```

        //compute K*v_n
        matvec(kernelmatrix, npoints, npoints, solution, temp);
        //update v_n+1
        for (i = 0; i <= npoints - 1; i++)
        {
            solution[i] = temp[i];
        }
    }

    optprice = 0.0, optdelta = 0.0;
    gamma_price = log(strike * ((double)(ndates + 1)) / spot - 1.);
    for (i = 0; i <= npoints - 1; i++)
    {
        //spot price
        price[i] = spot * exp(abscissa[i + 1]);
        ///option price
        optprice = optprice + weights[i + 1] * MAX(spot * (1 + exp(abscissa[i + 1]
            if (abscissa[i + 1] > gamma_price)
                optdelta = optdelta + weights[i + 1] * MAX((1 + exp(abscissa[i + 1])), 0
        }
    }
    *delta = optdelta;

    free_dvector(abscissa, 1, npoints);
    free_dvector(weights, 1, npoints);
    free_dvector(temp, 0, npoints - 1);
    free_dvector(inv, 0, nfft - 1);
    free_dvector(logk, 0, nfft - 1);
    free_dmatrix(c, 0, nfft - 1, 0, 2);
    free_dmatrix(kernelmatrix, 0, npoints - 1, 0, npoints - 1);

    return optprice;
}

void newmomentsAM(int model, double rf, double dt, int maxmoment,
    int ndates, double parameters[], double **momtable)
{
    int i, ii, k;
    double sum;

    for (i = 1; i < maxmoment + 1; i++)

```

```

    {
        momtable[1][i] = Creal(cfrn(model, rf, dt, Complex(0, -i), parameters));
    }

for (ii = 2; ii < ndates + 1; ii++)
{
    for (i = 1; i < maxmoment + 1; i++)
    {
        sum = 0;
        for (k = 1; k <= i; k++)
        {
            sum = sum + momtable[ii - 1][k] * bico(i, i - k);
        }

        momtable[ii][i] = momtable[1][i] * (1 + sum);
    }
}

void newmomentsArithM(int ndates, double Lmoments[], double *AvgMoments)
{
    AvgMoments[1] = (1.0 + Lmoments[1]) / (1 + ndates);
    AvgMoments[2] = (1.0 + 2.0 * Lmoments[1] + Lmoments[2]) / POW((1 + ndates), 2);
    AvgMoments[3] = (1.0 + 3.0 * Lmoments[1] + 3.0 * Lmoments[2] + Lmoments[3]) /
    AvgMoments[4] = (1.0 + 4.0 * Lmoments[1] + 6.0 * Lmoments[2] + 4.0 * Lmoments[3] + Lmoments[4]) /
    AvgMoments[5] = (1.0 + 5.0 * Lmoments[1] + 10.0 * Lmoments[2] + 10.0 * Lmoments[3] + 5.0 * Lmoments[4] + Lmoments[5]) /
    POW((1 + ndates), 5);

}

double boundAM(int model, double bound, double rf, double dt, int maxmoment,
               int ndates, double parameters[], double moments[])
{
    int i;
    double min = 1.0, ratio;

    for (i = 1; i < maxmoment + 1; i++)
    {
        ratio = moments[i] / exp(bound * i);
        if (ratio < min)
            min = ratio;
    }
}

```

```

        if (ratio < min) min = ratio;

        ///printf("bound%.5f R %.9f,min %.7f \ n",bound, ratio,min);
    }

    return min;
}

//We find in an automatic way the extremes of integration
int findlowuplimit(int model, double rf, double dt, int maxnummoments,
                  int ndates, double lowfactor, double upfactor,
                  double parameters[], double extremes[])
{
    int kk;
    double **momtable;
    double *moments;
    double *ArAvmoments;
    double mu1, mu2, mom1, mom2;
    double levylow, levyup, lowlim;
    double uplim;
    double bound;

    ArAvmoments = dvector(1, 5);
    momtable = dmatrix(1, ndates, 1, maxnummoments);
    moments = dvector(1, maxnummoments);
    //vectors where to store the outputs of the FFT inversion

    //compute the moments of the arithmetic average
    newmomentsAM(model, rf, dt, maxnummoments, ndates, parameters, momtable);

    for (kk = 1; kk < maxnummoments + 1; kk++)
    {
        moments[kk] = momtable[ndates][kk];
    }
    mu1 = momtable[ndates][1];
    mu2 = momtable[ndates][2];
    // printf("\ nMOMENTS SUM\ nm1=%.12f m2=%.12f\ n", mu1,mu2);

    uplim = log(mu1 + upfactor * POW(mu2 - mu1 * mu1, 0.5));

```

```

bound = boundAM(model, uplim, rf, dt, maxnummoments, ndates, parameters, momen

while (bound > POW(10.0, -5.0))
{
    uplim = uplim + 0.15;
    bound = boundAM(model, uplim, rf, dt, maxnummoments, ndates, parameters, m
    // printf("test: low%.12f\ t up %.12f \ tbound %.15f\ n",lowlim,uplim,boun
}

mom1 = MomentsLevy(model, rf, 1, dt, parameters);
mom2 = MomentsLevy(model, rf, 2, dt, parameters);
//printf("\ nMOMENTS Levy\ nm1=%.12f m2=%.12f\ n",
//mom1,mom2);

levylow = mom1 - lowfactor * POW(mom2 - mom1 * mom1, 0.5);
bound = BoundLowerTailLevy(model, -levylow, rf, dt, maxnummoments, parameters)

while (bound > POW(10.0, -8.0))
{
    levylow = levylow - 0.05;

    bound = BoundLowerTailLevy(model, -levylow, rf, dt, maxnummoments, paramet
}

levyup = mom1 + lowfactor * POW(mom2 - mom1 * mom1, 0.5);
bound = BoundUpperTailLevy(model, levyup, rf, dt, maxnummoments, parameters);

while (bound > POW(10.0, -8.0))
{
    levyup = levyup + 0.05;

    bound = BoundUpperTailLevy(model, levyup, rf, dt, maxnummoments, parameter
}

lowlim = -MAX(fabs(levylow), levyup);
//impliedfactor=(mom1-lowlim)/POW(mom2-mom1*mom1,0.5);

extremes[1] = lowlim;

```

```

    extremes[2] = uplim;

    free_dvector(moments, 1, maxnummoments);
    free_dvector(ArAvmoments, 1, 5);
    free_dmatrix(momtable, 1, ndates, 1, maxnummoments);

    return 1;
}

/*****
PRICING MODELS
*****/
/*****
BLACK SCHOLES MODEL FOR
DISCRETE ASIAN OPTIONS
*****/
double Asian_BS_FusaiMeucci(double spot,
                             double strike,
                             double maturity,
                             double rf,
                             double dividend,
                             double sigmaBS,
                             int nmonitoringdates,
                             double lowlim,
                             double uplim,
                             int nquadpoints, //n. of quadrature points
                             long nfft,
                             double price[],
                             double solution[], double *delta) //OUTPUT: Contains
{
    double asiabs;
    double dt = maturity / (nmonitoringdates);
    double *BSPParameters;
    int maxnummoments = 10;
    double lowfactor = 10;
    double upfactor = 10;
    double *extremes;

    // double *solution;
    BSPParameters = dvector(1, 1);
    BSPParameters[1] = sigmaBS;

```

```

    extremes = dvector(1, 2);

    findlowuplimit(1, rf, dt, maxnummoments,
                  nmonitoringdates, lowfactor, upfactor,
                  BSParameters, extremes);

    asiabs = DiscreteAsian(1, spot, strike, rf, dt,
                          nmonitoringdates, extremes[1], extremes[2],
                          nquadpoints, nfft, //n. of points for the fft inversion
                          BSParameters, //the parameters of the model
                          price,
                          solution, delta);
    free_dvector(extremes, 1, 2);
    free_dvector(BSParameters, 1, 1);

    return asiabs;
}
/*****
NIG MODEL FOR
DISCRETE ASIAN OPTIONS
*****/
double Asian_NIG_FusaiMeucci(double spot,
                             double strike,
                             double maturity,
                             double rf,
                             double dividend,
                             double alphaNIG, double betaNIG, double deltaNIG,
                             int nmonitoringdates,
                             double lowlim,
                             double uplim,
                             int nquadpoints, //n. of quadrature points
                             long nfft,
                             double price[],
                             double solution[], double *delta) //OUTPUT: Containin
{
    double asianig;
    double dt = maturity / (nmonitoringdates);
    double *NIGParameters;
    int maxnummoments = 10;
    double lowfactor = 10;

```

```

double upfactor = 10;

double *extremes;

// double *solution;
NIGParameters = dvector(1, 3);
NIGParameters[1] = alphaNIG;
NIGParameters[2] = betaNIG;
NIGParameters[3] = deltaNIG;

extremes = dvector(1, 2);

findlowuplimit(2, rf, dt, maxnummoments,
               nmonitoringdates, lowfactor, upfactor,
               NIGParameters, extremes);

asianig = DiscreteAsian(2, spot, strike, rf, dt,
                       nmonitoringdates, extremes[1], extremes[2],
                       nquadpoints, nfft, //n. of points for the fft inversion
                       NIGParameters, //the parameters of the model
                       price,
                       solution, delta);
free_dvector(extremes, 1, 2);
free_dvector(NIGParameters, 1, 3);

return asianig;
}

/*****
MERTON MODEL FOR
DISCRETE ASIAN OPTIONS
*****/
double Asian_MERTON_FusaiMeucci(double spot,
                                double strike,
                                double maturity,
                                double rf,
                                double dividend,
                                double sgMerton, double alphaMerton, double lambda,
                                int nmonitoringdates,

```

```

double lowlim,
double uplim,
int nquadpoints,//n. of quadrature points
long nfft,
double price[],
double solution[], double *delta)

//OUTPUT: Contains the solution
{
double asiamerton;
double dt = maturity / (nmonitoringdates);
double *MertonParameters;
int maxnummoments = 10;
double lowfactor = 10;
double upfactor = 10;

double *extremes;

// double *solution;
MertonParameters = dvector(1, 4);
MertonParameters[1] = sgMerton;
MertonParameters[2] = alphaMerton;
MertonParameters[3] = lambdaMerton;
MertonParameters[4] = deltaMerton;

extremes = dvector(1, 2);

findlowuplimit(7, rf, dt, maxnummoments,
               nmonitoringdates, lowfactor, upfactor,
               MertonParameters, extremes);

asiamerton = DiscreteAsian(7, spot, strike, rf, dt,
                           nmonitoringdates, extremes[1], extremes[2],
                           nquadpoints, nfft,//n. of points for the fft invers
                           MertonParameters, //the parameters of the model
                           price,
                           solution, delta);

free_dvector(extremes, 1, 2);
free_dvector(MertonParameters, 1, 4);
return asiamerton;

```

```

}

/*****
CGMY MODEL FOR
DISCRETE ASIAN OPTIONS
*****/
double Asian_CGMY_FusaiMeucci(double spot,
                                double strike,
                                double maturity,
                                double rf,
                                double dividend,
                                double CCGMY, double GCGMY, double MCGMY, double YCGMY,
                                int nmonitoringdates,
                                double lowlim,
                                double uplim,
                                int nquadpoints, //n. of quadrature points
                                long nfft,
                                double price[],
                                double solution[], double *delta) //OUTPUT: Contain
{
    double asiacgmy;
    double dt = maturity / (nmonitoringdates);
    double *CGMYParameters;
    int maxnummoments = 10;
    double lowfactor = 10;
    double upfactor = 10;
    double *extremes;

    // double *solution;
    CGMYParameters = dvector(1, 4);
    CGMYParameters[1] = CCGMY; ///C
    CGMYParameters[2] = GCGMY; ///G
    CGMYParameters[3] = MCGMY; ///M
    CGMYParameters[4] = YCGMY; ///Y

    extremes = dvector(1, 2);

    findlowuplimit(5, rf, dt, maxnummoments,
                   nmonitoringdates, lowfactor, upfactor,
                   CGMYParameters, extremes);
}

```

```

    asiacgmy = DiscreteAsian(5, spot, strike, rf, dt,
                             nmonitoringdates, extremes[1], extremes[2],
                             nquadpoints, nfft, //n. of points for the fft inversio
                             CGMYParameters, //the parameters of the model
                             price,
                             solution, delta);

    free_dvector(extremes, 1, 2);
    free_dvector(CGMYParameters, 1, 4);
    return asiacgmy;
}

/*****
KOU DE MODEL FOR
DISCRETE ASIAN OPTIONS
*****/
double Asian_DE_FusaiMeucci(double spot,
                             double strike,
                             double maturity,
                             double rf,
                             double dividend,
                             double sgDE, double lambdaDE, double pDE, double eta
                             int nmonitoringdates,
                             double lowlim,
                             double uplim,
                             int nquadpoints, //n. of quadrature points
                             long nfft,
                             double price[],
                             double solution[], double *delta) //OUTPUT: Contains
{
    double asiade;
    double dt = maturity / (nmonitoringdates);
    double *DEParameters;
    int maxnummoments = 10;
    double lowfactor = 10;
    double upfactor = 10;
    double *extremes;

    // double *solution;
    DEParameters = dvector(1, 5);
    DEParameters[1] = sgDE;

```

```

DEParameters[2] = lambdaDE;
DEParameters[3] = pDE;
DEParameters[4] = eta1DE;
DEParameters[5] = eta2DE;

extremes = dvector(1, 2);

findlowuplimit(6, rf, dt, maxnummoments,
               nmonitoringdates, lowfactor, upfactor,
               DEParameters, extremes);

asiade = DiscreteAsian(6, spot, strike, rf, dt,
                      nmonitoringdates, extremes[1], extremes[2],
                      nquadpoints, nfft, //n. of points for the fft inversion
                      DEParameters, //the parameters of the model
                      price,
                      solution, delta);

free_dvector(extremes, 1, 2);
free_dvector(DEParameters, 1, 5);

return asiade;
}
#endif //PremiaCurrentVersion

```