

## [Help](#)

```
extern "C" {
#include "
href../../../../mod/rstemperedstable1d/rstemperedstable1d_std/rstemperedstable1d_st
}
#include "
href../../../../common/math/numerics_h_src.pdfmath/numerics.h"
#include "
href../../../../common/math/fft_h_src.pdfmath/fft.h"

extern "C" {

    static const int step = 1;
    static const double xmax = 1.6;
    static const double xmin = -1.2;
    static const double eps = 1e-004;
    //static const double er=1e-008;
    //static const double minaccur=1e-008;
    static const double hh = 0.002;
    /*static char *pfname="iprices.dat";
    static char *ebfname="iearly.dat";
    static char *pftitle="Option prices \ n Spot \ t Option Price\ n";
    static char *ebftitle="Early exercise boundaries \ n Time \ t Boundary\ n";*/

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2008+2) //The "#els
    static int CHK_OPT(FD_KLZ)(void *Opt, void *Mod)
    {
        return NONACTIVE;
    }
    int CALC(FD_KLZ)(void *Opt, void *Mod, PricingMethod *Met)
    {
        return AVAILABLE_IN_FULL_PREMIA;
    }
#else

    static double intlpl(long k, double lp, double h, double nu,
                        double c, double er);
    static double intlpl0(long k, double lp, double h, double nu,
                        double c, double er);
    static double intlpl1(long k, double lp, double h, double nu,
```

```

        double c, double er);
static double intlpp(long k, double lp, double h, double nu,
        double c, double er);
static void fillarray(double *v1, double *v2, long int N);
static void confft(double *creal, double *cimage, double *v,
        double *vreal, double *vimage, double *res,
        double *resimage, long int n, long int m, long int Nbin, in
static void strike_correct(double strike, double *zz,
        long int N, int islog);
/*static void printoutDA(PnlVect *ptDA1, PnlVect *ptDA2,
        char *foutname, char *strtitle);*/

/*////////////////////////////////////////*/
static int fds_ts_amerput(double lm, double lp,
        double alpha_plus, double alpha_minus, double c_plus,
        double c_minus,
        double r, double divid,
        double T, double h, double Strike,
        double Spot,
        double eps, long errr, int step,
        double *Price/*,
        PnlVect **TimePoints,
PnlVect **EEBoundaries,
PnlVect **SpacePoints,
        PnlVect **Prices*/)
{
    double *t, *y, *HH, *v1, *payoff;
    long int N1, Nx, Ns, Nf, Nbin;
    double *v2;
    double *v3;
    double *tmp, *ureal, *uimage, *zreal, *zimage;
    double *vreal, *vimage, *tmp2, *tmpimage;

    PnlVect *TP, *EEB, *SP, *PP;

    double *cp, *cm, *p, *cmm, *cpp, *ccm, *ccp, *alp, *alm;
    long int *Lm;
    double a2 = 0.;
    double er, minaccr;

```

```

double logSpot;

double lpnu = exp(alpha_plus * log(lp));
double lmnu = exp(alpha_minus * log(lm));
double gamma_plus = pnl_tgamma(-alpha_plus);
double gamma_minus = pnl_tgamma(-alpha_minus);

double Am = -log(eps / c_minus / lmnu) / lm;
double Ap = -log(eps / c_plus / lpnu) / lp;
double mA = Ap > Am ? Ap : Am;

long int j, L;
long int kmax = (long int)ceil(mA / h);

//!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
double mu = r - divid + c_minus * gamma_minus * (lmnu - exp(alpha_minus * lo

//!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

Ns = (long int)ceil((xmax - xmin) / h);
long int Nmax = Ns + kmax;
Nf = Nmax + kmax;
long int N0 = (long int)ceil(-xmin / h) + kmax;
Nbin = 2;
while (Nbin < Nf) Nbin *= 2;
long int k;
double ereq;
double accur;
double dt;
double sum_m;
double sum_p;
double cc00;

Nx = Nmax; /*number of space points*/

/*Memory allocation for space grid*/

cp = (double *)calloc(kmax + 1, sizeof(double));
if (cp == NULL)

```

```

    return MEMORY_ALLOCATION_FAILURE;
cm = (double *)calloc(kmax + 1, sizeof(double));
if (cm == NULL)
    return MEMORY_ALLOCATION_FAILURE;
cpp = (double *)calloc(kmax + 1, sizeof(double));
if (cpp == NULL)
    return MEMORY_ALLOCATION_FAILURE;
cmm = (double *)calloc(kmax + 1, sizeof(double));
if (cmm == NULL)
    return MEMORY_ALLOCATION_FAILURE;
ccp = (double *)calloc(kmax + 1, sizeof(double));
if (ccp == NULL)
    return MEMORY_ALLOCATION_FAILURE;
ccm = (double *)calloc(kmax + 1, sizeof(double));
if (ccm == NULL)
    return MEMORY_ALLOCATION_FAILURE;
alp = (double *)calloc(kmax + 1, sizeof(double));
if (alp == NULL)
    return MEMORY_ALLOCATION_FAILURE;
alm = (double *)calloc(kmax + 1, sizeof(double));
if (alm == NULL)
    return MEMORY_ALLOCATION_FAILURE;
ureal = (double *)calloc(Nbin, sizeof(double));
if (ureal == NULL)
    return MEMORY_ALLOCATION_FAILURE;
uimage = (double *)calloc(Nbin, sizeof(double));
if (uimage == NULL)
    return MEMORY_ALLOCATION_FAILURE;
zreal = (double *)calloc(Nbin, sizeof(double));
if (zreal == NULL)
    return MEMORY_ALLOCATION_FAILURE;
zimage = (double *)calloc(Nbin, sizeof(double));
if (zimage == NULL)
    return MEMORY_ALLOCATION_FAILURE;
y = (double *)calloc(Nmax + 1, sizeof(double)); /*space grid points*/
if (y == NULL)
    return MEMORY_ALLOCATION_FAILURE;
payoff = (double *)calloc(Nmax + 1, sizeof(double));
if (payoff == NULL)
    return MEMORY_ALLOCATION_FAILURE;
v1 = (double *)calloc(Nmax + 1, sizeof(double)); /*prices at previous time s

```

```

if (v1 == NULL)
    return MEMORY_ALLOCATION_FAILURE;
v2 = (double *)calloc(Nmax + 1, sizeof(double)); /*current price*/
if (v2 == NULL)
    return MEMORY_ALLOCATION_FAILURE;
v3 = (double *)calloc(Nmax + 1, sizeof(double)); /*previous iteration form c
if (v3 == NULL)
    return MEMORY_ALLOCATION_FAILURE;
vreal = (double *)calloc(Nbin, sizeof(double));
if (vreal == NULL)
    return MEMORY_ALLOCATION_FAILURE;
vimage = (double *)calloc(Nbin, sizeof(double));
if (vimage == NULL)
    return MEMORY_ALLOCATION_FAILURE;
tmpimage = (double *)calloc(Nbin, sizeof(double));
if (tmpimage == NULL)
    return MEMORY_ALLOCATION_FAILURE;
tmp2 = (double *)calloc(Nbin, sizeof(double));
if (tmp2 == NULL)
    return MEMORY_ALLOCATION_FAILURE;
tmp = (double *)calloc(Nbin, sizeof(double)); /*previous iteration form curr
if (tmp == NULL)
    return MEMORY_ALLOCATION_FAILURE;
TP = (PnlVect *)calloc(1, sizeof(PnlVect)); /*time grid points*/
if (TP == NULL)
    return MEMORY_ALLOCATION_FAILURE;
EEB = (PnlVect *)calloc(1, sizeof(PnlVect)); /*early exercise boundaries*/
if (EEB == NULL)
    return MEMORY_ALLOCATION_FAILURE;
SP = (PnlVect *)calloc(1, sizeof(PnlVect)); /*space grid points*/
if (SP == NULL)
    return MEMORY_ALLOCATION_FAILURE;
PP = (PnlVect *)calloc(1, sizeof(PnlVect)); /*option prices*/
if (PP == NULL)
    return MEMORY_ALLOCATION_FAILURE;

/*Computation of coefficients*/

k = 1;

er = eps / (Nmax + 1) / 3;

```

```

while (k < kmax)
{
    k++;
    cp[k] = intlpl(k - 1, lp, h, alpha_plus, c_plus, er); /* coefficients for
    cm[k] = intlpl(k - 1, lm, h, alpha_minus, c_minus, er); /* coefficients f
}
k = 0;
while (k < kmax - 1)
{
    k++;
    ccp[k] = intlpp(k, lp, h, alpha_plus, c_plus, er); /* coefficients for i
    ccm[k] = intlpp(k, lm, h, alpha_minus, c_minus, er); /* coefficients for
}

cp[1] = intlpl(0, lp, h, alpha_plus, c_plus, er);
cm[1] = intlpl(0, lm, h, alpha_minus, c_minus, er);

//!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

sum_m = intlpl1(kmax, lm, h, alpha_minus, c_minus, er);
sum_p = intlpl1(kmax, lp, h, alpha_plus, c_plus, er);

cc00 = (cp[1] + sum_p) / pow(h, alpha_plus) + (cm[1] + sum_m) / pow(h, alpha

/*number of time steps*/
N1 = step * (1 + (long int)ceil(3 * T * (fabs(mu / h) + cc00) / 2.0));
//!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

/*Memory allocation for time grid*/
Lm = (long int *)calloc(N1 + 2, sizeof(long int));
if (Lm == NULL)
    return MEMORY_ALLOCATION_FAILURE;
t = (double *)calloc(N1 + 2, sizeof(double)); /*time points*/
if (t == NULL)
    return MEMORY_ALLOCATION_FAILURE;
HH = (double *)calloc(N1 + 2, sizeof(double)); /*early exercise boundaries*/
if (HH == NULL)

```

```

    return MEMORY_ALLOCATION_FAILURE;

HH[1] = N1;
Lm[1] = N0;

/*Time step*/
dt = T / N1;
t[1] = 0;
t[2] = dt;

//!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!1
if (mu > 0)
{
    a2 = 1 + dt * (mu / h + cc00 + r - divid);
}
if (mu <= 0)
{
    a2 = 1 + dt * (-mu / h + cc00 + r - divid);
}
//!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!1

k = 0;
while (k < kmax - 1)
{
    k++;
    alp[k] = dt * (ccp[k] + cp[k]) / pow(h, alpha_plus) / a2; /* coefficient
    alm[k] = dt * (ccm[k] + cm[k]) / pow(h, alpha_minus) / a2; /* coefficient

}

if (mu > 0)
{
    alp[1] = alp[1] + dt * mu / h / a2;
}

if (mu <= 0)
{
    alm[1] = alm[1] - dt * mu / h / a2;
}
alp[kmax] = dt * cp[kmax] / pow(h, alpha_plus) / a2;

```

```

alm[kmax] = dt * cm[kmax] / pow(h, alpha_minus) / a2;

for (j = 0; j < kmax; j++)
{
    ureal[j] = alp[kmax - j];
    uimage[j] = 0;
    zreal[j] = alm[j + 1];
    zimage[j] = 0;
}
for (j = kmax; j < Nbin; j++)
{
    ureal[j] = 0;
    uimage[j] = 0;
    zreal[j] = 0;
    zimage[j] = 0;
}
fft1d(ureal, uimage, Nbin, -1);

fft1d(zreal, zimage, Nbin, -1);


k = 0;

/*Put Pay-off function*/
for (j = 0; j < N0; j++)
{
    y[j] = (j - N0) * h;
    payoff[j] = 1 - exp(y[j]);
}
for (j = N0; j <= Nx; j++)
{
    y[j] = (j - N0) * h;
    payoff[j] = 0;
}

fillarray(v1, payoff, Nmax + 1);
fillarray(v2, payoff, Nmax + 1);

minaccur = eps / (N1 + 1) / 3;

```



```

/*Main Loop on time grid*/
for (L = 2; L <= N1 + 1; L++)
{
    t[L] = t[L - 1] + dt;
    j = Lm[L - 1] + 2; /*early exercise boundary */
    ereq = 1;
    /* first approximation*/

    confft(ureal, uimage, v1, vreal, vimage, tmp, tmpimage, Ns, kmax, Nbin,

    confft(zreal, zimage, v1, vreal, vimage, tmp2, tmpimage, Ns, kmax, Nbin,

    while (ereq > 0)
    {
        v2[j] = v1[j] / a2 + tmp[j - 1] + tmp2[j - 2];
        if ((v2[j] < payoff[j]) || (j == 1))
        {
            ereq = -1;
            Lm[L] = j;
            v2[j] = payoff[j];
        }
        j--;
    }
    if (j == 1) v2[1] = v1[1] / a2 + tmp[0] + tmp2[Nbin - 1];
    for (j = Lm[L]; j <= Nmax; j++)
    {
        v2[j] = v1[j] / a2 + tmp[j - 1] + tmp2[j - 2];
    }

    fillarray(v3, v2, Nmax + 1);
    j = Lm[L];
    HH[L] = exp(y[j]);

    /*Iterative solution for prices*/
    accur = 0;

    /*computation of error*/
    for (j = Lm[L]; j <= Nmax; j++)
    {

```

```

        if (fabs(v1[j] - v2[j]) > accur)
        {
            accur = fabs(v1[j] - v2[j]);
        }
    }

    /* iterative computation of price */

    while (accur > minaccur)
    {
        accur = 0;
        j = Lm[L - 1] + 2;
        ereq = 1;

        conflft(ureal, uimage, v3, vreal, vimage, tmp, tmpimage, Ns, kmax, Nb)

        conflft(zreal, zimage, v3, vreal, vimage, tmp2, tmpimage, Ns, kmax, N)

        while (ereq > 0)
        {
            v2[j] = v1[j] / a2 + tmp[j - 1] + tmp2[j - 2];
            if ((v2[j] < payoff[j]) || (j == 1))
            {
                ereq = -1;
                Lm[L] = j;
                v2[j] = payoff[j];
            }
            j--;
        }
        if (j == 1) v2[1] = v1[1] / a2 + tmp[0] + tmp2[Nbin - 1];
        for (j = Lm[L]; j <= Nmax; j++)
        {
            v2[j] = v1[j] / a2 + tmp[j - 1] + tmp2[j - 2];
            if (fabs(v2[j] - v3[j]) > accur)
            {
                accur = fabs(v2[j] - v3[j]);
            }
        }

        j = Lm[L];
    }

```

```

        HH[L] = exp(y[j]);

        fillarray(v3, v2, Nmax + 1);

    }
    p = v1;
    v1 = v2;
    v2 = p;
}

/*Memory desallocation*/
free(cp);
free(cm);
free(cpp);
free(cmm);
free(ccp);
free(ccm);
free(alp);
free(alm);
free(v2);
free(v3);
free(Lm);
free(vreal);
free(vimage);
free(tmp);
free(tmp2);
free(tmpimage);
free(ureal);
free(uimage);
free(zreal);
free(zimage);
free(payoff);
free(TP);
free(PP);
free(EEB);
free(SP);
free(t);

logSpot = log(Spot / Strike);
j = (long int)ceil((logSpot - xmin) / h) + kmax;

```

```

strike_correct(Strike, y, Nx + 1, 1);
strike_correct(Strike, v1, Nx + 1, 0);
strike_correct(Strike, HH, N1 + 1, 0);

*Price = (Spot - y[j]) / (y[j + 1] - y[j]) * (v1[j + 1] - v1[j]) + v1[j];

free(y);
free(v1);
free(HH);
return OK;
}

/*////////////////////////////////////////*/
static void conffft(double *creal, double *cimage, double *v,
                   double *vreal, double *vimage, double *res,
                   double *resimage, long int n, long int m, long int Nbin, in
{

    long int Nz = Nbin - n - m - m;
    long int j;

    if (d > 0)
    {
        for (j = 0; j < n; j++)
        {
            vreal[j] = v[m + j + 1];
            vimage[j] = 0;
        }
        for (j = n; j < n + m + Nz; j++)
        {
            vreal[j] = 0;
            vimage[j] = 0;
        }
        for (j = 1; j < m + 1; j++)
        {
            vreal[n + m + Nz - 1 + j] = v[j];
            vimage[n + m + Nz - 1 + j] = 0;
        }
    }
}

```

```

else
{
    for (j = 0; j < n + m; j++)
    {
        vreal[j] = v[j + 1];
        vimage[j] = 0;
    }
    for (j = n + m; j < Nbin; j++)
    {
        vreal[j] = 0;
        vimage[j] = 0;
    }
}

fft1d(vreal, vimage, Nbin, -1);

for (j = 0; j < Nbin; j++)
{
    res[j] = creal[j] * vreal[j] - cimage[j] * vimage[j];
    resimage[j] = cimage[j] * vreal[j] + creal[j] * vimage[j];
}
fft1d(res, resimage, Nbin, 1);
}

/*////////////////////////////////////////*/
static void fillarray(double *v1, double *v2, long int N)
{
    long int j;
    for (j = 0; j < N; j++)
        v1[j] = v2[j];
}

/*////////////////////////////////////////*/
static void strike_correct(double strike, double *zz, long int N, int islog)
{
    long int j;
    if (islog)
        for (j = 0; j < N; j++)
            zz[j] = strike * exp(zz[j]);
    else
        for (j = 0; j < N; j++)

```

```

        zz[j] = strike * zz[j];

    }

    /*////////////////////////////////////////*/
    static double intlpl(long k, double lp, double h, double nu,
                        double c, double er)
    {
        double err = 1;
        long int j, n = 1;
        double st = 0.5;
        double w, s1, s2, v1, v2, res;
        s1 = exp(-lp * (k + 1) * h) * pow(k + 1, -1 - nu);
        s2 = exp(-lp * (k + st) * h) * pow(k + st, -1 - nu) * st;
        v2 = st * (s1 + 4.0 * s2) / 3.0;
        v1 = 0;

        n = 2;

        while (err > er)
        {
            v1 = v2;
            s1 += 2.0 * s2;
            s2 = 0;
            w = k + st / 2.0;
            for (j = 1; j <= n; j++)
            {
                s2 += exp(-lp * w * h) * pow(w, -1 - nu) * (w - k);
                w += st;
            }
            st = st / 2.0;
            n = n * 2;
            v2 = st * (s1 + 4.0 * s2) / 3.0;
            err = v2 > 0 ? fabs((v1 - v2) / v2) : 1;
            if (n > 1200000) err = er / 2.0;

        }

        res = c * v2;
    }

```

```

    return res;
}

/*////////////////////////////////////*/
static double intl0(long k, double lp, double h, double nu,
                    double c, double er)
{
    double err = 1;
    long int j, n = 1;
    double st = 0.5;
    double w, s1, s2, v1, v2, res;
    s1 = exp(-lp * h);
    s2 = exp(-lp * st * h) * pow(st, 3 - nu);
    v2 = st * (s1 + 4.0 * s2) / 3.0;
    v1 = 0;

    n = 2;

    while (err > er)
    {
        v1 = v2;
        s1 += 2.0 * s2;
        s2 = 0;
        w = st / 2.0;
        for (j = 1; j <= n; j++)
        {
            s2 += exp(-lp * w * h) * pow(w, 3 - nu);
            w += st;
        }
        st = st / 2.0;
        n = n * 2;
        v2 = st * (s1 + 4.0 * s2) / 3.0;
        err = v2 > 0 ? fabs((v1 - v2) / v2) : 1;
        if (n > 1200000) err = er / 2.0;

    }

    res = c * (v2 * pow(lp * h, 3) / (2 - nu) / (3 - nu) / (1 - nu) + exp(-lp *
return res;

```

```

}
/*////////////////////////////////////*/
static double intlpp(long k, double lp, double h, double nu,
                    double c, double er)
{
    double err = 1;
    long int j, n = 1;
    double st = 0.5;
    double w, s1, s2, v1, v2, res;
    s1 = exp(-lp * k * h) * pow(k, -1 - nu);
    s2 = exp(-lp * (k + st) * h) * pow(k + st, -1 - nu) * 0.5;
    v2 = st * (s1 + 4.0 * s2) / 3.0;
    v1 = 0;

    n = 2;

    while (err > er)
    {
        v1 = v2;
        s1 += 2.0 * s2;
        s2 = 0;
        w = k + st / 2.0;
        for (j = 1; j <= n; j++)
        {
            s2 += exp(-lp * w * h) * pow(w, -1 - nu) * (k + 1 - w);
            w += st;
        }
        st = st / 2.0;
        n = n * 2;
        v2 = st * (s1 + 4.0 * s2) / 3.0;
        err = v2 > 0 ? fabs((v1 - v2) / v2) : 1;
        if (n > 1200000) err = er / 2.0;

    }

    res = c * v2;

    return res;
}

```



```

/*//////////////////////////////////////////////////////////////
static double intlpl1(long k, double lp, double h, double nu,
                      double c, double er)
{
    double err = 1;
    long int j, n = 1;
    double st = (k - 1) * 0.5;
    double w, s1, s2, v1, v2, res;
    s1 = exp(-lp * h) + exp(-lp * k * h) * pow(k, -1 - nu);
    s2 = exp(-lp * (1 + st) * h) * pow(1 + st, -1 - nu);
    v2 = st * (s1 + 4.0 * s2) / 3.0;
    v1 = 0;

    n = 2;

    while (err > er)
    {
        v1 = v2;
        s1 += 2.0 * s2;
        s2 = 0;
        w = 1 + st / 2.0;
        for (j = 1; j <= n; j++)
        {
            s2 += exp(-lp * w * h) * pow(w, -1 - nu);
            w += st;
        }
        st = st / 2.0;
        n = n * 2;
        v2 = st * (s1 + 4.0 * s2) / 3.0;
        err = v2 > 0 ? fabs((v1 - v2) / v2) : 1;
        if (n > 1200000) err = er / 2.0;

    }

    res = c * v2;

    return res;
}
//////////////////////////////////////////////////////////////

```

```

/*static void printoutDA(PnlVect *ptDA1, PnlVect *ptDA2, char *foutname, char
{

FILE *fic;
long int i, nn;
double *ptd1, *ptd2;

if((fic = fopen(foutname,"w")) == NULL)
{
printf("Unable to open output File %s\ n",foutname);
return;
}

nn=ptDA1->size;
ptd1=ptDA1->array;
ptd2=ptDA2->array;
fprintf(fic, "%s", strttitle);
i=2;
do
{
fprintf(fic, "%f \ t%f \ n",ptd1[i], ptd2[i]);
i++;
}while(i<nn);

fclose(fic);

}*/
/*////////////////////////////////////////*/

int CALC(FD_KLZ)(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 fds_ts_amerput(
    ptMod->LambdaMinus.Val.V_DOUBLE, ptMod->LambdaPlus.Val.V_DOUBLE,
    ptMod->AlphaPlus.Val.V_RGDOUBLE, ptMod->AlphaMinus.Val.V_RGDOUBLE,
    , r, divid,
    ptOpt->Maturity.Val.V_DATE - ptMod->T.Val.V_DATE,
    Met->Par[0].Val.V_RGDOUBLE/*xstep*/, strike,
    spot, Met->Par[1].Val.V_RGDOUBLE, 1, step,/*multiplier*/
    & (Met->Res[0].Val.V_DOUBLE)
    /*,&(Met->Res[1].Val.V_PNLVECT),
    &(Met->Res[2].Val.V_PNLVECT),
    &(Met->Res[3].Val.V_PNLVECT),
    &(Met->Res[4].Val.V_PNLVECT)*/);
}

static int CHK_OPT(FD_KLZ)(void *Opt, void *Mod)
{
    if ((strcmp(((Option *)Opt)->Name, "PutAmer") == 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_RGDOUBLE = hh;
        Met->Par[1].Val.V_RGDOUBLE = eps;

        first = 0;
    }

    return OK;
}

```

```

PricingMethod MET(FD_KLZ) =
{
    "FD_KLZ",
    { {"SpaceStep", RGDOUBLE, {100}, ALLOW    },
      {"Accuracy: ", RGDOUBLE, {100}, ALLOW    },
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CALC(FD_KLZ),
    { {"Price", DOUBLE, {100}, FORBID}
      /*{"Time Points",PNLVECT,{100},FORBID},
      {"Early Exercise Boundaries",PNLVECT,{100},FORBID} ,
      {"Space Points",PNLVECT,{100},FORBID} ,
      {"Prices",PNLVECT,{100},FORBID}*/ ,
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CHK_OPT(FD_KLZ),
    CHK_ok ,
    MET(Init)
} ;

/*////////////////////////////////////////*/

}

```