

[Help](#)

```
#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2007+2) //The "#els
#else

#include <stdlib.h>
#include "
href../../mod/hk1d/hk1d_std/hktree_h_src.pdfhktree.h"
#include "
href../../mod/hk1d/hk1d_std/mathsb_h_src.pdfmathsb.h"
#include "pnl/pnl_mathtools.h"

double CondExp(double t, double s, double x);
/* returns  $E(x_t | x_s=x)$  */

double CondVar(double t, double s, double a0, double sigma0);
/* returns  $\text{Var}(x_t | x_s=x)$  which must be independent of  $x$  */

void SetHKtree(struct Tree *Meth, double a0, double sigma0)
{
    int jmin, jmax, jminprev, jmaxprev;
    int h, i, j, k, nv;
    double M, mujk, Mij, dx;

    /* SetTimegrid(Meth, Tf, Ngrid);*/

    if (Meth->t == NULL) printf("FATAL ERROR IN SetTree(), SetTimegrid must be use

    /* Allocation of the tree variables */
    Meth->pLRij = malloc((Meth->Ngrid + 1) * sizeof(double *));
    Meth->pLQij = malloc((Meth->Ngrid + 1) * sizeof(double *));
    Meth->pLPDo = malloc((Meth->Ngrid) * sizeof(double *));
    Meth->pLPMi = malloc((Meth->Ngrid) * sizeof(double *));
    Meth->pLPUp = malloc((Meth->Ngrid) * sizeof(double *));
    Meth->pLRef = malloc((Meth->Ngrid) * sizeof(int *));
    Meth->TSize = malloc((Meth->Ngrid + 1) * sizeof(int));
```

```

/*
// initialization at t[0]
jmin=0;
jmax=0;
nv=1;
Meth->pLRij[0] = malloc(sizeof(double));
Meth->pLQij[0] = malloc(sizeof(double));
Meth->pLRij[0][0]=xi;
Meth->>TSize[0]=1;
*/

/* initialization at t[0] [there are 3 points at t[0] (for the delta computat
{
    dx = sqrt(3.) * sqrt(CondVar(Meth->t[1], Meth->t[0], a0, sigma0));
    /* this corresponds to: Delta_x_i+1 = sqrt(3) * V_i [Brigo/Mercurio, p. 489
    jmin = -1;
    jmax = +1;
    nv = 3;
    Meth->pLRij[0] = malloc(3 * sizeof(double));
    Meth->pLQij[0] = malloc(3 * sizeof(double));
    Meth->pLRij[0][0] = -dx;
    Meth->pLRij[0][1] = 0;
    Meth->pLRij[0][2] = +dx;
    Meth->>TSize[0] = 3;
}

/* iteration on the time step */
for (i = 1; i <= Meth->Ngrid; i++)
{
    dx = sqrt(3.) * sqrt(CondVar(Meth->t[i], Meth->t[i - 1], a0, sigma0));
    /* this corresponds to: Delta_x_i+1 = sqrt(3) * V_i [Brigo/Mercurio, p. 4
    jminprev = jmin;
    jmaxprev = jmax;

    M = CondExp(Meth->t[i], Meth->t[i - 1], Meth->pLRij[i - 1][0]);
    jmin = pnl_iround(M / dx) - 1;
    M = CondExp(Meth->t[i], Meth->t[i - 1], Meth->pLRij[i - 1][nv - 1]);

```

```

jmax = pnl_iround(M / dx) + 1;

Meth->pLPDo[i - 1] = malloc(nv * sizeof(double));
Meth->pLPMi[i - 1] = malloc(nv * sizeof(double));
Meth->pLPUp[i - 1] = malloc(nv * sizeof(double));
Meth->pLRef[i - 1] = malloc(nv * sizeof(int));

nv = jmax - jmin + 1;
Meth->TSize[i] = nv;

Meth->pLRij[i] = malloc(nv * sizeof(double));

Meth->pLQij[i] = malloc(nv * sizeof(double));
/* for HW-trees, the alloc. of Meth->pLQij is done in TranslateTree !*/

for (k = jmin; k <= jmax; k++)
{
    j = k - jmin;
    Meth->pLRij[i][j] = k * dx;
}
/* Remark: Meth->pLRij[i] is an increasing sequence !*/

for (k = jminprev; k <= jmaxprev; k++)
{
    j = k - jminprev;
    Mij = CondExp(Meth->t[i], Meth->t[i - 1], Meth->pLRij[i - 1][j]);
    h = pnl_iround(Mij / dx);

    mujk = Mij - h * dx;

    Meth->pLPUp[i - 1][j] = 1. / 6. + SQR(mujk / dx) / 2. + mujk / (2.*dx)
    Meth->pLPMi[i - 1][j] = 2. / 3. - SQR(mujk / dx);
    Meth->pLPDo[i - 1][j] = 1. / 6. + SQR(mujk / dx) / 2. - mujk / (2.*dx)
    Meth->pLRef[i - 1][j] = h - jmin;

    if (h <= jmin) printf("ERROR FATAL JMIN JMAX IN SetTree(), ExpectCond_
    if (h >= jmax) printf("ERROR FATAL JMIN JMAX IN SetTree(), ExpectCond_

```

```

    }

} /* end of i-loop */

}

```

```

/*//////////////////////////////////////
// returns E( x_t | x_s=x ) which is x
// recall: x_t = int_0^t sigma*exp(a*u) dW_u
////////////////////////////////////*/
double CondExp(double t, double s, double x)
{
    return x;
}

```

```

/*//////////////////////////////////////
// returns Var( x_t | x_s=x ) which is SQR(sigma0) * (exp(2*a0*t) - exp(2*a0*s))
// recall: x_t = int_0^t sigma0*exp(a0*u) dW_u
////////////////////////////////////*/
double CondVar(double t, double s, double a0, double sigma0)
{
    return SQR(sigma0) * (exp(2 * a0 * t) - exp(2 * a0 * s)) / (2 * a0);
}

```

```

void SetTimegrid(struct Tree *Meth, double Tf, int Ngrid)
{
    int i;

```

```

Meth->Ngrid = Ngrid;
Meth->Tf = Tf;

Meth->t = malloc((Meth->Ngrid + 1) * sizeof(double));

for (i = 0; i <= Meth->Ngrid; i++)
{
    Meth->t[i] = i * Meth->Tf / Meth->Ngrid;
}

}

int indiceTime(struct Tree *Meth, double s)
{
    int i = 0;

    if (Meth->t == NULL)
    {
        printf("FATALE ERREUR, PAS DE GRILLE DE TEMPS !");
    }
    else
    {
        while (i <= Meth->Ngrid && Meth->t[i] <= s)
        {
            i++;
        }
    }
    return i - 1;
}

```

```

void Computepayoff1(struct Tree *Meth, double s)
{
    double ht;
    int i, j, n;

```

```

n = indiceTime(Meth, s);

if (Meth->t == NULL) printf("FATAL ERROR IN Computepayoff(), SetTimegrid() and
if (Meth->pLRij == NULL) printf("FATAL ERROR IN Computepayoff(), SetTimegrid()

if (Meth->Payofffunc == NULL)
{
    initPayoff1(Meth, Meth->Tf);
    printf("DEFAULT PAYOFF 1\ n"); /*Payoff 1 par default.*/
}

/*/ initialization: Meth->pLQij[n] = Meth->Payofffunc[n], where n=indiceTime(Me
for (j = 0; j < Meth->TSize[n]; j++)
{
    Meth->pLQij[n][j] = Meth->Payofffunc[n][j];
}

/* AMERICAN backward iteration of Meth->pLQij[i] from i=n-1 to i=0
// here AMERICAN means: taking the max with Meth->Payofffunc[i]*/
for (i = n - 1; i >= 0; i--)
{
    for (j = 0; j < Meth->TSize[i]; j++)
    {
        ht =    Meth->pLPDo[i][j] * Meth->pLQij[i + 1][ Meth->pLRef[i][j] - 1
                + Meth->pLPMi[i][j] * Meth->pLQij[i + 1][ Meth->pLRef[i][j] ]
                + Meth->pLPUp[i][j] * Meth->pLQij[i + 1][ Meth->pLRef[i][j] +

        /* american !!*/
        if (ht < Meth->Payofffunc[i][j]) ht = Meth->Payofffunc[i][j];

        Meth->pLQij[i][j] = ht;
    }
}
}

```

```

double OPTIONr(struct Tree *Meth, double r, double s)
{
    double  theta, R_T;
    int j, Ns, Nr;

    Ns = indiceTime(Meth, s);
    j = 0;

    while (Meth->pLRij[Ns][j] < r && j < Meth->TSize[Ns] - 1)
    {
        j++;
    }
    if (j == 0)
    {
        theta = 0;
    }
    else
    {
        theta = (r - Meth->pLRij[Ns][j - 1]) / (Meth->pLRij[Ns][j] - Meth->pLRij[Ns][j - 1]);
    }
    if (theta > 1)
    {
        theta = 1;
        j = j + 1;
    }

    Nr = j - 1;

    if (Nr < 0)
    {
        Nr = 0;
    }
    if (j > Meth->TSize[Ns] - 2)
    {
        printf("WARNING : Instantaneous futur spot rate is out of tree\ n");
    }
    if (Nr == 0)

```

```

    {
        printf("WARNING : Instantaneous futur spot rate is out of tree\ n");
    }

    R_T = theta * Meth->pLQij[Ns][Nr + 1] + (1 - theta) * Meth->pLQij[Ns][Nr];

    return R_T;
}

```

```

double OPTION(struct Tree *Meth)
{
    return Meth->pLQij[0][1];
}

```

```

int DeleteTree(struct Tree *Meth)
{
    int i;

    for (i = 0; i < Meth->Ngrid + 1; i++)
    {
        free(Meth->pLRij[i]);
    }
    for (i = 0; i < Meth->Ngrid + 1; i++)
    {
        free(Meth->pLQij[i]);
    }
    for (i = 0; i < Meth->Ngrid; i++)
    {
        free(Meth->pLPDo[i]);
    }
    for (i = 0; i < Meth->Ngrid; i++)
    {
        free(Meth->pLPMi[i]);
    }
}

```



```

    }
    for (i = 0; i < Meth->Ngrid; i++)
    {
        free(Meth->pLPUp[i]);
    }
    for (i = 0; i < Meth->Ngrid; i++)
    {
        free(Meth->pLRef[i]);
    }

    free(Meth->pLRij);
    free(Meth->pLQij);
    free(Meth->pLPDo);
    free(Meth->pLPMi);
    free(Meth->pLPUp);
    free(Meth->pLRef);
    free(Meth->TSize);
    free(Meth->t);

    free(Meth->Payofffunc);
    return 1;
}

```

```

void initPayoff1(struct Tree *Meth, double T0)
{
    int i, j, n;

    n = indiceTime(Meth, T0);
    /*T0 must be <= Tf, the final time of the tree !! */

    /* Allocation of Meth->Payofffunc[0...n] */
    Meth->Payofffunc = malloc((n + 1) * sizeof(double *));
    for (i = 0; i <= n; i++)
    {
        Meth->Payofffunc[i] = malloc((Meth->TSize[i]) * sizeof(double));
    }
}

```

```

/* Set the Payofffunc[n][j]=1 */
for (j = 0; j < Meth->TSize[n]; j++) Meth->Payofffunc[n][j] = 1;

/*Set the Payofffunc[i][j]=0, where i<n */
for (i = n - 1; i >= 0; i--)
{
    for (j = 0; j < Meth->TSize[i]; j++)
    {
        Meth->Payofffunc[i][j] = 0;
    }
}
}

```

```

int AddTime(struct Tree *Meth, double T)
{
    int i, j;
    double *tmp;

    if (T < 0)
    {
        printf("Error: I can't add negative times to a tree !\ n");
        return -1;
    }

    if ((Meth->t == NULL) && (T == 0))
    {
        Meth->t = malloc(sizeof(double));
        Meth->t[0] = 0;
        Meth->Ngrid = 0;
        Meth->Tf = 0;
        return 0;
    }
}

```

```

if ((Meth->t == NULL) && (T > 0))
{
    Meth->t = malloc(2 * sizeof(double));
    Meth->t[0] = 0;
    Meth->t[1] = T;
    Meth->Ngrid = 1;
    Meth->Tf = T;
    return 1;
}

i = 0;
while ((i < Meth->Ngrid) && (Meth->t[i] < T))
{
    i++;
    if (Meth->t[i] == T) return i;
}

/* we know that t[i]!=T and (i=Ngrid or t[i]>=T) */
if (Meth->t[i] < T) i = Meth->Ngrid + 1;
tmp = malloc((Meth->Ngrid + 1) * sizeof(double));
for (j = 0; j <= Meth->Ngrid; j++) tmp[j] = Meth->t[j];

Meth->Ngrid = Meth->Ngrid + 1;
free(Meth->t);
Meth->t = malloc((Meth->Ngrid + 1) * sizeof(double));
for (j = 0; j < i; j++) Meth->t[j] = tmp[j];
Meth->t[i] = T;
for (j = i + 1; j <= Meth->Ngrid; j++) Meth->t[j] = tmp[j - 1];

free(tmp);
return i;
}

```

```

void DeletePayoff1(struct Tree *Meth, double T0)

```

```
{
    int i, n;
    n = indiceTime(Meth, T0);
    for (i = 0; i < n + 1; i++)
        free(Meth->Payofffunc[i]);
}

#endif //PremiaCurrentVersion
```