

[Help](#)

```
#include <stdlib.h>
#include "
href../../mod/cir1d/cir1d_std/cir1d_std_h_src.pdfcir1d_std.h"

/*Product*/
static double dt, dr, r_min, r_max;
static double *r_vect, *disc, * *Ps, * *Option_Price;
static double *pu, *pm, *pd;
static long Ns;

/* static int j_max;*/

/*Memory Allocation*/
static void memory_allocation(long Nt)
{
    int i;

    if ((r_vect = malloc(sizeof(double) * (Ns + 1))) == NULL)
    {
        printf("Allocation error");
        exit(1);
    }
    if ((disc = malloc(sizeof(double) * (Ns + 1))) == NULL)
    {
        printf("Allocation error");
        exit(1);
    }
    if ((pu = malloc(sizeof(double) * (Ns + 1))) == NULL)
    {
        printf("Allocation error");
        exit(1);
    }
    if ((pm = malloc(sizeof(double) * (Ns + 1))) == NULL)
    {
        printf("Allocation error");
        exit(1);
    }
    if ((pd = malloc(sizeof(double) * (Ns + 1))) == NULL)
    {
```

```

        printf("Allocation error");
        exit(1);
    }
    if ((Ps = malloc(sizeof(double *) * (Nt + 1))) == NULL)
    {
        printf("Allocation error");
        exit(1);
    }
    for (i = 0; i <= Nt; i++)
    {
        Ps[i] = malloc(sizeof(double) * (Ns + 1));
    }
    if ((Option_Price = malloc(sizeof(double *) * (Nt + 1))) == NULL)
    {
        printf("Allocation error");
        exit(1);
    }
    for (i = 0; i <= Nt; i++)
    {
        Option_Price[i] = malloc(sizeof(double) * (Ns + 1));
    }

    return;
}

/*Memory Desallocation*/
static void free_memory(long Nt)
{
    int i;

    free(r_vect);
    free(pu);
    free(pm);
    free(pd);
    free(disc);

    for (i = 0; i < Nt + 1; i++)
        free(Ps[i]);
    free(Ps);

    for (i = 0; i < Nt + 1; i++)

```

```

    free(Option_Price[i]);
free(Option_Price);

return;
}

/*Compute probabilities*/
static int init_prob(double k, double sigma, double theta, double T, double t0,
{
    double df;
    int j;
    double beta, alpha1, alpha2;

    /*Time and Space Step*/
    dt = (T - t0) / (double)Nt;
    dr = sigma * sqrt(3. / 4.*dt);

    /*Localization*/
    alpha1 = (4.*k * theta - SQR(sigma)) / 8.;
    alpha2 = k / 2.;
    beta = dr / (2.*dt);
    r_min = (-beta + sqrt(SQR(beta) + 4.*alpha1 * alpha2)) / (2.*alpha2);
    r_max = (beta + sqrt(SQR(beta) + 4.*alpha1 * alpha2)) / (2.*alpha2);
    Ns = (int)ceil((r_max - r_min) / dr);
    memory_allocation(Nt);

    /*Compute probabilities*/
    for (j = 0; j <= Ns; j++)
    {
        r_vect[j] = r_min + (double)j * dr;
        disc[j] = exp(-SQR(r_vect[j]) * dt);
        df = ((4.*k * theta - SQR(sigma)) / (8.*r_vect[j]) - r_vect[j] * k / 2.) *

        /*Boundary*/
        if (j == 0)
        {
            pu[j] = 1. / 6. + (SQR(df) - df) / 2.;
            pm[j] = df - 2.*pu[j];
            pd[j] = 1. - pu[j] - pm[j];
        }
        else if (j == Ns)

```

```

        {
            pd[j] = 1. / 6. + (SQR(df) + df) / 2.;
            pm[j] = -df - 2.*pd[j];
            pu[j] = 1. - pd[j] - pm[j];

        }
        /*Not Boundary*/
        else
        {
            pu[j] = 1. / 6. + (SQR(df) + df) / 2.;
            pd[j] = pu[j] - df;
            pm[j] = 1. - pu[j] - pd[j];
        }
    }

    return OK;
}

/*Zero Coupon Bond*/
static int zcb_cir(long Nt)
{
    int i, j;

    /*Maturity conditions for pure discount Bond*/
    for (j = 0; j <= Ns; j++)
        Ps[Nt][j] = 1.;

    /*Dynamic Programming*/
    for (i = Nt - 1; i >= 0; i--)
        for (j = 0; j <= Ns; j++)
        {
            if (j == 0)
                Ps[i][j] = disc[j] * (pu[j] * Ps[i + 1][j + 2] + pm[j] * Ps[i + 1][j + 1] + pu[j] * Ps[i + 1][j]);
            else if (j == Ns)
                Ps[i][j] = disc[j] * (pd[j] * Ps[i + 1][j - 2] + pm[j] * Ps[i + 1][j - 1] + pu[j] * Ps[i + 1][j]);
            else
                Ps[i][j] = disc[j] * (pu[j] * Ps[i + 1][j + 1] + pm[j] * Ps[i + 1][j] + pd[j] * Ps[i + 1][j - 1]);
        }

    return 1.;
}

```

```

/*Cap,Floor=Portfolio of zero-bond options*/
static int capfloor_cir1d(double r0, double k, double t0, double sigma, double t
{
    int i, j, z, Nt, NsY, Nt0, nb_payement;
    double val, val1, tmp, sum;

    /*Number of maximal steps*/
    Nt = NtY * (long)((T - t0) / periodicity);

    /*Compute probabilities*/
    init_prob(k, sigma, theta, T, t0, Nt);

    /*Compute Cap or Floor*/
    nb_payement = (int)((T - first_payement) / periodicity);
    sum = 0.;
    NsY = Nt;
    tmp = p->Par[0].Val.V_DOUBLE;
    p->Par[0].Val.V_DOUBLE = 1. / (1. + K * periodicity);
    for (z = nb_payement; z > 0; z--)
    {
        /*Number of steps for generic caplet/flooret*/
        NsY = Nt - (nb_payement - z) * NtY;

        /*Compute Zero Coupon Prices*/
        zcb_cir(NsY);

        /*Compute Caplet or Flooret*/
        /*Maturity conditions*/
        Nt0 = NsY - NtY;
        for (j = 0; j <= Ns; j++)
            Option_Price[Nt0][j] = (p->Compute)(p->Par, Ps[Nt0][j]);

        /*Explicit Finite Difference Cycle*/
        for (i = Nt0 - 1; i >= 0; i--)
            for (j = 0; j <= Ns; j++)
            {
                /*Boundary*/
                if (j == 0)
                    Option_Price[i][j] = disc[j] * (pu[j] * Option_Price[i + 1][j + 2]
                else if (j == Ns)

```

```

        Option_Price[i][j] = disc[j] * (pd[j] * Option_Price[i + 1][j - 2]
/*Not Boundary*/
    else
        Option_Price[i][j] = disc[j] * (pu[j] * Option_Price[i + 1][j + 1]
    }

/*Linear Interpolation*/
j = 0;
while (SQR(r_vect[j]) < r0)
    j++;
val = Option_Price[0][j];
val1 = Option_Price[0][j - 1];

/*Sum*/
sum += (1. + K * periodicity) * (val + (val - val1) * (r0 - SQR(r_vect[j]))
}

/*Price*/
*price = Nominal * sum;

/*Memory Disallocation*/
p->Par[0].Val.V_DOUBLE = tmp;
free_memory(Nt);

return OK;
}

int CALC(FD_CAPFLOOR)(void *Opt, void *Mod, PricingMethod *Met)
{
    TYPEOPT *ptOpt = (TYPEOPT *)Opt;
    TYPEMOD *ptMod = (TYPEMOD *)Mod;

    return capfloor_cir1d(ptMod->r0.Val.V_PDOUBLE, ptMod->k.Val.V_DOUBLE, ptMod->T
        ptMod->theta.Val.V_PDOUBLE, ptOpt->FirstResetDate.Val.V_
}

static int CHK_OPT(FD_CAPFLOOR)(void *Opt, void *Mod)

```

```

{
    if ((strcmp(((Option *)Opt)->Name, "Cap") == 0) || (strcmp(((Option *)Opt)->Na
        return OK;
    else
        return WRONG;
}

static int MET(Init)(PricingMethod *Met, Option *Opt)
{
    if (Met->init == 0)
    {
        Met->init = 1;

        Met->Par[0].Val.V_LONG = 10;

    }
    return OK;
}

PricingMethod MET(FD_CAPFLOOR) =
{
    "FD_Explicit_Cir1d_CapFloor",
    { {"TimeStepNumber for Period", LONG, {100}, ALLOW},
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CALC(FD_CAPFLOOR),
    {"Price", DOUBLE, {100}, FORBID}, {" ", PREMIA_NULLTYPE, {0}, FORBID}},
    CHK_OPT(FD_CAPFLOOR),
    CHK_ok,
    MET(Init)
} ;

```