

[Help](#)

```
#include <stdlib.h>
#include "bs1d_doublim.h"
#include "error_msg.h"

static void polint(double xa[], double ya[], int n, double x, double *y, double
{
    int i, m, ns = 1;
    double den, dif, dift, ho, hp, w;
    double *c, *d;
    double *xb, *yb;

    c = (double *)malloc(sizeof(double) * (n + 1));
    d = (double *)malloc(sizeof(double) * (n + 1));
    xb = (double *)malloc(sizeof(double) * (n + 1));
    yb = (double *)malloc(sizeof(double) * (n + 1));

    for (i = 1; i <= n; i++)
    {
        xb[i] = xa[i - 1];
        yb[i] = ya[i - 1];
    }

    dif = fabs(x - xa[1]);

    for (i = 1; i <= n; i++)
    {
        if ((dift = fabs(x - xb[i])) < dif)
        {
            ns = i;
            dif = dift;
        }
        c[i] = yb[i];
        d[i] = yb[i];
    }
    *y = yb[ns--];
    for (m = 1; m < n; m++)
    {
        for (i = 1; i <= n - m; i++)
```

```

        {
            ho = xb[i] - x;
            hp = xb[i + m] - x;
            w = c[i + 1] - d[i];
            if ((den = ho - hp) == 0.0) printf("Error in routine polint\ n");
            den = w / den;
            d[i] = hp * den;
            c[i] = ho * den;
        }
        *y += (*dy = (2 * ns < (n - m) ? c[ns + 1] : d[ns--]));
    }

    free(c);
    free(d);
    free(xb);
    free(yb);
}

static double interpolation(double s0, double down, double up, double *lS0ld, in
{
    int j, index, z;
    double xa[10], ya[10], res, dres;

    /*First time step*/
    j = 0;
    while ((lS0ld[j] <= x) && (j <= k_old - 1))
    {
        j++;
    }

    if (j == k_old)
    {
        index = j - 2;
        for (z = 0; z < 3; z++)
        {
            xa[z] = s0 * exp(lS0ld[index + z]);
            ya[z] = P0ld[index + z];
        }
        polint(xa, ya, 3, s0 * exp(x), &res, &dres);
    }
}

```

```

else if (j == k_old - 1)
{
    index = j - 1;

    for (z = 0; z < 3; z++)
    {
        xa[z] = s0 * exp(lS0ld[index + z]);
        ya[z] = P0ld[index + z];
    }
    polint(xa, ya, 3, s0 * exp(x), &res, &dres);

}
else
{
    index = j - 1;
    for (z = 0; z < 4; z++)
    {
        xa[z] = s0 * exp(lS0ld[index + z]);
        ya[z] = P0ld[index + z];
    }
    polint(xa, ya, 4, s0 * exp(x), &res, &dres);
}
return res;
}

```

```

static int Tr_AGZ(int am, double s0, NumFunc_1 *L, NumFunc_1 *U, NumFunc_1 *Reba
{
    double downv[2], upv[2], tv[3];
    double rebate;
    double u, d, stock;
    int i, j, k, z, index;
    double q, qc;
    double dt;
    double **P, **lS;
    double down, up;
    int monit;
    double *lS0ld, *P0ld;
    double eps = 0.0001;
    double xa[10], ya[10], res, dres;
    double downn, upn;

```

```

int k_old;
int new_N;

/*Up and Down factors*/
upv[0] = (U->Compute)(U->Par, 0);
upv[1] = up1;
downv[0] = (L->Compute)(L->Par, 0);
downv[1] = down1;
rebate = (Rebate->Compute)(Rebate->Par, 0);
tv[0] = 0.;
tv[1] = t1;
tv[2] = t;

//Memory Allocation
P = (double **)malloc(sizeof(double *) * (N + 1000));
for (i = 0; i < N + 1000; i++)
    P[i] = (double *)malloc(sizeof(double) * (2 * (N + 10)));

lS = (double **)malloc(sizeof(double *) * (N + 1000));
for (i = 0; i < N + 1000; i++)
    lS[i] = (double *)malloc(sizeof(double) * (2 * (N + 10)));

lSOld = (double *)malloc(sizeof(double) * (2 * (N + 10)));
POld = (double *)malloc(sizeof(double) * (2 * (N + 10)));

for (monit = 1; monit >= 0; monit--)
{
    //Trasformed Up and down barrier
    down = log((downv[monit]) / s0);
    up = log((upv[monit]) / s0);

    /*Compute index k*/
    dt = (tv[monit + 1] - tv[monit]) / (double)N;
    k = ceil((up - down) / (2 * sigma * sqrt(dt)));
    dt = (up - down) / (2 * k * sigma) * (up - down) / (2 * k * sigma);
    new_N = floor((tv[monit + 1] - tv[monit]) / dt);

    /*Compute q*/
    u = exp(sigma * sqrt(dt));
    d = 1 / u;
    q = (exp((r - divid) * dt) - d) / (u - d);
}

```

```

qc = 1 - q;
q *= exp(-r * dt);
qc *= exp(-r * dt);

//Maturity Mesh
lS[new_N][0] = down;
for (j = 1; j <= k; j++)
    lS[new_N][j] = lS[new_N][j - 1] + 2 * sigma * sqrt(dt);

/*Backward Mesh*/
z = 1;
for (i = new_N - 1; i >= 0; i--)
{
    if (z % 2 == 0)
    {
        lS[i][0] = down;
        for (j = 1; j <= k; j++)
            lS[i][j] = lS[i][j - 1] + 2 * sigma * sqrt(dt);
    }
    else
    {
        lS[i][0] = down + sigma * sqrt(dt);
        for (j = 1; j <= k - 1; j++)
            lS[i][j] = lS[i][j - 1] + 2 * sigma * sqrt(dt);
    }
    z++;
}

//Maturity conditions
if (monit == 1)
{
    for (j = 0; j <= k; j++)
    {
        stock = s0 * exp(lS[new_N][j]);

        if ((j == 0) || (j == k))
        {
            if (am)
                P[new_N][j] = (p->Compute)(p->Par, stock);
            else
                P[new_N][j] = rebate;
        }
    }
}

```

```

        }
    else
        P[new_N][j] = (p->Compute)(p->Par, stock);
    }
}
else
{
    for (j = 0; j <= k; j++)
    {
        downn = log((downv[monit + 1]) / s0);
        upn = log((upv[monit + 1]) / s0);

        if ((lS[new_N][j] <= downn + eps) || (lS[new_N][j] >= upn - eps))
        {
            if (am)
                P[new_N][j] = (p->Compute)(p->Par, stock);
            else
                P[new_N][j] = rebate;
        }
        else
        {
            P[new_N][j] = interpolation(s0, downn, upn, lS0ld, k_old, P0ld);
        }
    }
}

/*Backward Resolution*/
z = 1;
for (i = new_N - 1; i >= 0; i--)
{
    if (z % 2 == 0)
    {
        //k_old=k;
        for (j = 0; j <= k; j++)
        {
            if (am)
            {
                stock = s0 * exp(lS[i][j]);
                if ((j == 0) || (j == k))
                    P[i][j] = (p->Compute)(p->Par, stock);
            }
        }
    }
}

```

```

        else
        {
            P[i][j] = qc * P[i + 1][j - 1] + q * P[i + 1][j];
            P[i][j] = MAX(P[i][j], (p->Compute)(p->Par, stock));
        }
    }
    else
    {
        if ((j == 0) || (j == k))
            P[i][j] = rebate;
        else
            P[i][j] = qc * P[i + 1][j - 1] + q * P[i + 1][j];
    }
}
}
else /*Price*/
{
    //k_old=k-1;
    for (j = 0; j < k; j++)
    {
        P[i][j] = qc * P[i + 1][j] + q * P[i + 1][j + 1];
        if (am)
            P[i][j] = MAX(P[i][j], (p->Compute)(p->Par, stock));
    }
}
z++;
}

for (j = 0; j <= k - 1; j++)
{
    lS0ld[j] = lS[0][j];
    k_old = k - 1;
    P0ld[j] = P[0][j];
}
} //end monit

/*First time step*/
j = 0;
while ((lS[0][j] < 0) && (j < k - 1)) j++;

//Interpolation at S0

```

```

    if (exp(lS[0][j - 1]) >= 1)
    {
        index = j;
        for (z = 0; z < 3; z++)
        {
            xa[z] = s0 * exp(lS[0][index + z]);
            ya[z] = P[0][index + z];
        }

        polint(xa, ya, 3, s0, &res, &dres);
    }
    else
    {
        index = j - 1;
        for (z = 0; z < 4; z++)
        {
            xa[z] = s0 * exp(lS[0][index + z]);
            ya[z] = P[0][index + z];
        }
        polint(xa, ya, 4, s0, &res, &dres);
    }

    *ptprice = res;

    for (i = 0; i < N + 1000; i++)
        free(P[i]);
    free(P);

    for (i = 0; i < N + 1000; i++)
        free(lS[i]);
    free(lS);

    free(POld);
    free(lSOld);

    return OK;
}

int CALC(TR_AGZ)(void *Opt, void *Mod, PricingMethod *Met)
{

```



```

TYPEOPT *ptOpt = (TYPEOPT *)Opt;
TYPEMOD *ptMod = (TYPEMOD *)Mod;
double r, divid;

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

return Tr_AGZ(ptOpt->EuOrAm.Val.V_BOOL, ptMod->S0.Val.V_PDOUBLE,
              ptOpt->LowerLimit.Val.V_NUMFUNC_1, ptOpt->UpperLimit.Val.V_NUMFU
              ptOpt->Maturity.Val.V_DATE - ptMod->T.Val.V_DATE, ptOpt->DateBet
              Met->Par[0].Val.V_INT2, &(Met->Res[0].Val.V_DOUBLE));
}

static int CHK_OPT(TR_AGZ)(void *Opt, void *Mod)
{
    if ((strcmp(((Option *)Opt)->Name, "TwoDoubleStepPutOutEuro") == 0) || (strcmp
        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 = 1600;
    }
    return OK;
}

PricingMethod MET(TR_AGZ) =
{
    "TR_AGZ",
    { {"StepNumber", INT2, {100}, ALLOW},
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CALC(TR_AGZ),

```

```
{ {"Price", DOUBLE, {100}, FORBID},  
  {" ", PREMIA_NULLTYPE, {0}, FORBID}  
},  
CHK_OPT(TR_AGZ),  
CHK_tree,  
MET(Init)  
};
```