

Help

```
#include "bs2d_std2d.h"
#include "error_msg.h"
#define PRECISION 1.0e-7 /*Precision for the localization of FD methods*/

static int restriction22(int l, double **d, double **u, double **f, double **aa)
{
    int i, j, nl;
    double **w;

    nl = pow(2, l + 1) - 1;

    w = (double **)calloc(nl + 2, sizeof(double *));
    if (w == NULL)
        return MEMORY_ALLOCATION_FAILURE;
    for (i = 0; i < nl + 2; i++)
    {
        w[i] = (double *)calloc(nl + 2, sizeof(double));
        if (w[i] == NULL)
            return MEMORY_ALLOCATION_FAILURE;
    }

    for (i = 1; i < nl + 1; i++)
        for (j = 1; j < nl + 1; j++)
            w[i][j] = u[i][j] * aa[i * N / (nl + 1)][j * N / (nl + 1)] + (u[i + 1][j] +

    for (i = 2; i < nl; i = i + 2)
        for (j = 2; j < nl; j = j + 2)
            d[i / 2][j / 2] = (((w[i - 1][j - 1] + w[i + 1][j - 1] + w[i - 1][j + 1] +

    for (i = 0; i < nl + 2; i++)
        free(w[i]);
    free(w);

    return OK;
}
```

```

static int substract_prolongation2(int l, double **u, double **v)
{
    int nl, nl1, i, j;
    double **w;

    nl = pow(2, l + 1) - 1;
    nl1 = pow(2, l) - 1;

    w = (double **)calloc(nl + 2, sizeof(double *));
    if (w == NULL)
        return MEMORY_ALLOCATION_FAILURE;
    for (i = 0; i < nl + 2; i++)
    {
        w[i] = (double *)calloc(nl + 2, sizeof(double));
        if (w[i] == NULL)
            return MEMORY_ALLOCATION_FAILURE;
    }

    for (i = 1; i < nl + 1; i = i + 2)
    {
        w[i][0] = w[0][i] = w[nl + 1][i] = w[i][nl + 1] = 0.0;
    }

    for (i = 0; i < nl1 + 2; i++)
        for (j = 0; j < nl1 + 2; j++)
            w[2 * i][2 * j] = v[i][j];

    for (i = 1; i < nl + 1; i = i + 2)
        for (j = 2; j < nl; j = j + 2)
            w[i][j] = (w[i - 1][j] + w[i + 1][j]) / 2.0;

    for (i = 1; i < nl + 1; i++)
        for (j = 1; j < nl + 1; j = j + 2)
            w[i][j] = (w[i][j - 1] + w[i][j + 1]) / 2.0;

    for (i = 1; i < nl + 1; i++)
        for (j = 1; j < nl + 1; j++)
            u[i][j] = u[i][j] - w[i][j];

    for (i = 0; i < nl + 2; i++)
        free(w[i]);
}

```

```

    free(w);

    return OK;
}

```

```

static int MGM22(int l, double **u, double **f, double **aa, double **bb, int N)
{
    int i, j, nl, ii, nl1;
    double **d, **v;

    nl = pow(2, l + 1) - 1;
    nl1 = pow(2, l) - 1;

    d = (double **)calloc(nl1 + 2, sizeof(double *));
    if (d == NULL)
        return MEMORY_ALLOCATION_FAILURE;
    for (i = 0; i < nl1 + 2; i++)
    {
        d[i] = (double *)calloc(nl1 + 2, sizeof(double));
        if (d[i] == NULL)
            return MEMORY_ALLOCATION_FAILURE;
    }

    v = (double **)calloc(nl1 + 2, sizeof(double *));
    if (v == NULL)
        return MEMORY_ALLOCATION_FAILURE;
    for (i = 0; i < nl1 + 2; i++)
    {
        v[i] = (double *)calloc(nl1 + 2, sizeof(double));
        if (v[i] == NULL)
            return MEMORY_ALLOCATION_FAILURE;
    }

    if (l == 0) u[1][1] = f[1][1] / (aa[N / 2][N / 2]);
    else
    {
        /* 2 iterations of Gauss-Seidel*/
        for (ii = 1; ii < 3; ii++)
            for (i = 1; i < nl + 1; i++)

```

```

        for (j = 1; j < nl + 1; j++)
            u[i][j] = ((-u[i + 1][j] - u[i - 1][j] - u[i][j + 1] - u[i][j - 1]))

restriction22(l, d, u, f, aa, bb, N);

for (i = 0; i <= nl1 + 1; i++)
    for (j = 0; j <= nl1 + 1; j++)
        v[i][j] = 0.0;

MGM22(l - 1, v, d, aa, bb, N);

subtract_prolongation2(l, u, v);

/* 2 iterations of Gauss-Seidel*/
for (ii = 1; ii < 3; ii++)
    for (i = 1; i < nl + 1; i++)
        for (j = 1; j < nl + 1; j++)
            u[i][j] = ((-u[i + 1][j] - u[i - 1][j] - u[i][j + 1] - u[i][j - 1]))
}

for (i = 0; i < nl1 + 2; i++)
    free(v[i]);
free(v);

for (i = 0; i < nl1 + 2; i++)
    free(d[i]);
free(d);

return OK;
}

static int mult_amer2(double s1, double s2, NumFunc_2 *p, double t, double r, double
{
    double h, x1, x2, k, limit, aa, bb, gg1, gg0, sigma11, sigma21, sigma22, m1, m2;
    double **P, **Obst, **A, **B, **G, **R;
    int Index, TimeIndex, i, j, jj, N;
    int **pp;

    /*Memory Allocation*/
    N = pow(2, l + 1) - 1 + 1;

```

```
P = (double **)calloc(N + 1, sizeof(double *));
if (P == NULL)
    return MEMORY_ALLOCATION_FAILURE;
for (i = 0; i < N + 1; i++)
{
    P[i] = (double *)calloc(N + 1, sizeof(double));
    if (P[i] == NULL)
        return MEMORY_ALLOCATION_FAILURE;
}

Obst = (double **)calloc(N + 1, sizeof(double *));
if (Obst == NULL)
    return MEMORY_ALLOCATION_FAILURE;
for (i = 0; i < N + 1; i++)
{
    Obst[i] = (double *)calloc(N + 1, sizeof(double));
    if (Obst[i] == NULL)
        return MEMORY_ALLOCATION_FAILURE;
}

pp = (int **)calloc(N + 1, sizeof(int *));
if (pp == NULL)
    return MEMORY_ALLOCATION_FAILURE;
for (i = 0; i < N + 1; i++)
{
    pp[i] = (int *)calloc(N + 1, sizeof(int));
    if (pp[i] == NULL)
        return MEMORY_ALLOCATION_FAILURE;
}

R = (double **)calloc(N + 1, sizeof(double *));
if (R == NULL)
    return MEMORY_ALLOCATION_FAILURE;
for (i = 0; i < N + 1; i++)
{
    R[i] = (double *)calloc(N + 1, sizeof(double));
    if (R[i] == NULL)
        return MEMORY_ALLOCATION_FAILURE;
}

G = (double **)calloc(N + 1, sizeof(double *));
```

```

if (G == NULL)
    return MEMORY_ALLOCATION_FAILURE;
for (i = 0; i < N + 1; i++)
{
    G[i] = (double *)calloc(N + 1, sizeof(double));
    if (G[i] == NULL)
        return MEMORY_ALLOCATION_FAILURE;
}

A = (double **)calloc(N + 1, sizeof(double *));
if (A == NULL)
    return MEMORY_ALLOCATION_FAILURE;
for (i = 0; i < N + 1; i++)
{
    A[i] = (double *)calloc(N + 1, sizeof(double));
    if (A[i] == NULL)
        return MEMORY_ALLOCATION_FAILURE;
}

B = (double **)calloc(N + 1, sizeof(double *));
if (B == NULL)
    return MEMORY_ALLOCATION_FAILURE;
for (i = 0; i < N + 1; i++)
{
    B[i] = (double *)calloc(N + 1, sizeof(double));
    if (B[i] == NULL)
        return MEMORY_ALLOCATION_FAILURE;
}

/*Covariance Matrix*/
sigma11 = sigma1;
//sigma12=0.0;
sigma21 = rho * sigma2;
sigma22 = sigma2 * sqrt(1.0 - SQR(rho));
m1 = (r - divid1) - SQR(sigma11) / 2.0;
m2 = (r - divid2) - (SQR(sigma21) + SQR(sigma22)) / 2.0;

/*Space Localisation*/
limit = sqrt(t) * sqrt(log(1 / PRECISION));
h = 2.*limit / (double)N;

```

```

/*Time Step*/
k = t / (double)M;

/*Terminal Values*/
x1 = log(s1);
x2 = log(s2);
trend1 = exp(x1 + m1 * t);
trend2 = exp(x2 + m2 * t);

for (j = 1; j < N; j++)
    for (i = 1; i < N; i++)
        P[i][j] = (p->Compute)(p->Par, trend1 * exp(sigma11 * (-limit + h * (double)j), trend2 * exp(sigma22 * (-limit + h * (double)i)));

/*Homegenous Dirichlet Conditions*/
for (i = 0; i <= N; i++)
{
    P[i][0] = 0.;
    P[i][N] = 0.;
    P[0][i] = 0.;
    P[N][i] = 0.;
}

aa = 1. + 2.*k / h / h + r * k;
bb = -1.*k / 2. / h / h;

/*Finite Difference Cycle*/
for (TimeIndex = 1; TimeIndex < M + 1; TimeIndex++)
{
    trend1 = exp(x1 + m1 * (t - TimeIndex * k));
    trend2 = exp(x2 + m2 * (t - TimeIndex * k));

    for (j = 1; j < N; j++)
        for (i = 1; i < N; i++)
            Obst[i][j] = (p->Compute)(p->Par, trend1 * exp(sigma11 * (-limit + h * (double)j), trend2 * exp(sigma22 * (-limit + h * (double)i)));

/*Init Control*/
for (i = 0; i <= N; i++)
    for (j = 0; j <= N; j++)
    {
        pp[i][j] = 0;
    }
}

```

```

        R[i][j] = -P[i][j];
    }

    /*Howard Cycle*/
    for (jj = 0; jj < 2; jj++)
    {
        for (i = 1; i < N; i++)
            for (j = 1; j < N; j++)
            {
                gg0 = P[i][j] * aa + (P[i + 1][j] + P[i - 1][j] + P[i][j + 1] +
                gg1 = P[i][j] - Obst[i][j];
                if (gg0 < gg1) pp[i][j] = 0;
                else pp[i][j] = 1;
            }

        for (i = 1; i < N; i++)
            for (j = 1; j < N; j++)
            {
                if (pp[i][j] == 0)
                {
                    G[i][j] = -R[i][j];
                    A[i][j] = aa;
                    B[i][j] = bb;
                }
                else
                {
                    G[i][j] = Obst[i][j];
                    A[i][j] = 1;
                    B[i][j] = 0;
                }
            }
    }

    /*Solve the system*/
    MGM22(1, P, G, A, B, N);
}
/*End Howard Cycle*/
}
/*End Finite Difference Cycle*/

Index = (int)((double)N / 2.0);

```



```

/*Price*/
*ptprice = P[Index][Index];

/*Deltas*/
*ptdelta2 = (P[Index - 1][Index] - P[Index + 1][Index]) / (2.*s2 * h * sigma22
*ptdelta1 = ((P[Index][Index + 1] - P[Index][Index - 1]) / (2.*s1 * h) - sigma

/*Memory desallocation*/
for (i = 0; i < N + 1; i++)
    free(P[i]);
free(P);

for (i = 0; i < N + 1; i++)
    free(Obst[i]);
free(Obst);

for (i = 0; i < N + 1; i++)
    free(A[i]);
free(A);

for (i = 0; i < N + 1; i++)
    free(B[i]);
free(B);

for (i = 0; i < N + 1; i++)
    free(R[i]);
free(R);

for (i = 0; i < N + 1; i++)
    free(G[i]);
free(G);

for (i = 0; i < N + 1; i++)
    free(pp[i]);
free(pp);

return OK;
}

int CALC(FD_FMGH)(void *Opt, void *Mod, PricingMethod *Met)

```

```

{
    TYPEOPT *ptOpt = (TYPEOPT *)Opt;
    TYPEMOD *ptMod = (TYPEMOD *)Mod;
    double r, divid1, divid2;

    r = log(1. + ptMod->R.Val.V_DOUBLE / 100.);
    divid1 = log(1. + ptMod->Divid1.Val.V_DOUBLE / 100.);
    divid2 = log(1. + ptMod->Divid2.Val.V_DOUBLE / 100.);

    return mult_amer2(ptMod->S01.Val.V_PDOUBLE, ptMod->S02.Val.V_PDOUBLE, ptOpt->P
        ptOpt->Maturity.Val.V_DATE - ptMod->T.Val.V_DATE, r, divid1,
        ptMod->Sigma1.Val.V_PDOUBLE, ptMod->Sigma2.Val.V_PDOUBLE, pt
        Met->Par[0].Val.V_INT, Met->Par[1].Val.V_INT, Met->Par[2].Va
        &(Met->Res[0].Val.V_DOUBLE), &(Met->Res[1].Val.V_DOUBLE), &(

}

static int CHK_OPT(FD_FMGH)(void *Opt, void *Mod)
{
    Option *ptOpt = (Option *)Opt;
    TYPEOPT *opt = (TYPEOPT *) (ptOpt->TypeOpt);

    if ((opt->EuOrAm). Val.V_BOOL == AMER)
        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 = 5;
        Met->Par[1].Val.V_INT2 = 100;
        Met->Par[2].Val.V_RGDOUBLE = 0.000001;

    }
}

```

```
    return OK;
}

PricingMethod MET(FD_FMGH) =
{
    "FD_FMGH2d",
    {"Number of grids", INT2, {100}, ALLOW}, {"TimeStep", INT2, {100}, ALLOW} , {
    CALC(FD_FMGH),
    { {"Price", DOUBLE, {100}, FORBID}, {"Delta1", DOUBLE, {100}, FORBID} , {"Delt
        {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CHK_OPT(FD_FMGH),
    CHK_ok,
    MET(Init)
};
```