

## [Help](#)

```
/* Broadie & Glasserman algorithm (stochastic mesh)*/
#include <stdlib.h>
#include <stdio.h>
#include <
href../../../../common/math/cdo/cdo_math_h_src.pdfmath.h>

#include "
href../../../../mod/bsnd/bsnd_stdnd/bsnd_stdnd_h_src.pdfbsnd_stdnd.h"
#include "
href../../../../common/math/linsys_h_src.pdfmath/linsys.h"
#include "pnl/pnl_basis.h"
#include "
href../../../../mod/bsnd/bsnd_stdnd/black_h_src.pdfblack.h"
#include "
href../../../../common/optype_h_src.pdfoptype.h"
#include "
href../../../../common/enums_h_src.pdfenums.h"
#include "
href../../../../common/var_h_src.pdfvar.h"
#include "pnl/pnl_random.h"
#include "pnl/pnl_matrix.h"

static double *Mesh = NULL, *Path = NULL, *Price = NULL, *VectInvMeshDensity = N

static int BrGl_Allocation(long AL_Mesh_Size,
                           int OP_Exercise_Dates, int BS_Dimension)
{
    if (Mesh == NULL) Mesh = (double *)malloc(AL_Mesh_Size * OP_Exercise_Dates * B
    if (Mesh == NULL) return MEMORY_ALLOCATION_FAILURE;
    if (Price == NULL) Price = (double *)malloc(AL_Mesh_Size * OP_Exercise_Dates
    if (Price == NULL) return MEMORY_ALLOCATION_FAILURE;
    if (Path == NULL) Path = (double *)malloc(OP_Exercise_Dates * BS_Dimension *
    if (Path == NULL) return MEMORY_ALLOCATION_FAILURE;
    if (VectInvMeshDensity == NULL) VectInvMeshDensity = (double *)malloc(OP_Exerc
    if (VectInvMeshDensity == NULL) return MEMORY_ALLOCATION_FAILURE;
    return OK;
}

static void Brod_Liberation()
```

```

{
    if (Mesh != NULL)
    {
        free(Mesh);
        Mesh = NULL;
    }
    if (Price != NULL)
    {
        free(Price);
        Price = NULL;
    }
    if (Path != NULL)
    {
        free(Path);
        Path = NULL;
    }
    if (VectInvMeshDensity != NULL)
    {
        free(VectInvMeshDensity);
        VectInvMeshDensity = NULL;
    }
}

static double MeshDensity(int Time, double *Stock, int OP_Exercise_Dates, int AL
                           int BS_Dimension, double *BS_Spot, double Step)
{
    long k;
    double aux = 0;
    /*density function of the mesh law generator */
    if (Time > 1)
    {
        for (k = 0; k < AL_Mesh_Size; k++)
            aux += BS_TD(Mesh + k * OP_Exercise_Dates * BS_Dimension + (Time - 1) *
                return aux / (double)AL_Mesh_Size;
    }
    else
    {
        return BS_TD(BS_Spot, Stock, BS_Dimension, Step);
    }
}

```

```

static double Weight(int Time, double *iStock, double *jStock, int j, int BS_Dim
                    double Step, int AL_Mesh_Size)
{
    /*computation of the weight between the vectors iStock and jStock*/
    if (Time > 0)
        return BS_TD(iStock, jStock, BS_Dimension, Step) * VectInvMeshDensity[Time *
    else
        return 1.;
}

static void InitMesh(int AL_Mesh_Size, int BS_Dimension, double *BS_Spot,
                    int OP_Exercise_Dates, double Step, double Sqrt_Step,
                    int generator)
{
    int j, k, aux;

    /*mesh initialization; see the documentation*/
    for (k = 0; k < AL_Mesh_Size; k++)
        BS_Forward_Step(Mesh + k * OP_Exercise_Dates * BS_Dimension + BS_Dimension,

    for (j = 2; j < OP_Exercise_Dates; j++)
    {
        for (k = 0; k < AL_Mesh_Size; k++)
        {
            aux = (int)(pnl_rand_uni(generator) * AL_Mesh_Size);
            BS_Forward_Step(Mesh + k * OP_Exercise_Dates * BS_Dimension + j * BS_D
        }
    }
}

static void Close()
{
    /*memory liberation*/
    Brod_Liberation();
    BS_Transition_Liberation();
    End_BS();
}

/*see the documentation for the parameters meaning*/

```

```

static int BrGl(PnlVect *BS_Spot,
               NumFunc_nd *p,
               double OP_Maturity,
               double BS_Interest_Rate,
               PnlVect *BS_Dividend_Rate,
               PnlVect *BS_Volatility,
               double *BS_Correlation,
               long AL_MonteCarlo_Iterations,
               int generator,
               int AL_Mesh_Size,
               int OP_Exercise_Dates,
               double *AL_FPrice,
               double *AL_BPrice)
{
    double aux, Step, Sqrt_Step, DiscountStep;
    long i, j, k, init_mc;
    int l;
    /* double AL_FPrice, AL_BPrice; */
    int BS_Dimension = BS_Spot->size;
    PnlVect VMesh;
    VMesh.size = BS_Dimension;

    /* MC sampling */
    init_mc = pnl_rand_init(generator, BS_Dimension, AL_MonteCarlo_Iterations);
    /* Test after initialization for the generator */
    if (init_mc != OK) return init_mc;

    /*time step*/
    Step = OP_Maturity / (double)(OP_Exercise_Dates - 1);
    Sqrt_Step = sqrt(Step);
    /*discounting factor for a time step*/
    DiscountStep = exp(-BS_Interest_Rate * Step);

    Init_BS(BS_Dimension, BS_Volatility->array,
            BS_Correlation, BS_Interest_Rate, BS_Dividend_Rate->array);
    /*memory allocation of the BlackScholes variables*/
    BS_Transition_Allocation(BS_Dimension, Step);
    /*memory allocation of the algorithm's variables*/
    BrGl_Allocation(AL_Mesh_Size, OP_Exercise_Dates, BS_Dimension);

    /*initialization of the mesh*/

```

```

InitMesh(AL_Mesh_Size, BS_Dimension, BS_Spot->array, OP_Exercise_Dates, Step,

/* Backward Price */
/*partial computation of the weights*/
for (j = OP_Exercise_Dates - 2; j >= 1; j--)
{
    for (i = 0; i < AL_Mesh_Size; i++)
    {
        VectInvMeshDensity[j * AL_Mesh_Size + i] = 1. / MeshDensity(j + 1, Mes
    }
}

/*initialization of the mesh prices at the maturity*/
for (i = 0; i < AL_Mesh_Size; i++)
{
    VMesh.array = Mesh + i * OP_Exercise_Dates * BS_Dimension + (OP_Exercise_D
    Price[i * OP_Exercise_Dates + OP_Exercise_Dates - 1] = p->Compute(p->Par,
}

/*dynamical programming algorithm*/
for (j = OP_Exercise_Dates - 2; j >= 1; j--)
{
    for (i = 0; i < AL_Mesh_Size; i++)
    {
        aux = 0;
        /*approximation of the conditionnal expectation*/
        for (k = 0; k < AL_Mesh_Size; k++)
        {
            aux += Price[k * OP_Exercise_Dates + j + 1] * Weight(j, Mesh + i *
        }
        aux *= DiscountStep / (double)AL_Mesh_Size;
        /*exercise decision*/
        VMesh.array = Mesh + i * OP_Exercise_Dates * BS_Dimension + j * BS_Dim
        Price[i * OP_Exercise_Dates + j] = MAX(p->Compute(p->Par, &VMesh), aux
    }
}

aux = 0;
for (i = 0; i < AL_Mesh_Size; i++)
{
    aux += Price[i * OP_Exercise_Dates + 1];
}

```

```

/*output backward price*/
*AL_BPrice = MAX(p->Compute(p->Par, BS_Spot), DiscountStep * aux / (double)AL_

/* Forward Price */
if (*AL_BPrice == p->Compute(p->Par, BS_Spot))
    *AL_FPrice = *AL_BPrice;
else
{
    *AL_FPrice = 0.;
    for (i = 0; i < AL_MonteCarlo_Iterations; i++)
    {
        /*BlackScholes spot*/
        for (l = 0; l < BS_Dimension; l++)
            Path[l] = BS_Spot->array[l];

        j = 0;
        /*optimal stopping of a BlackScholes path*/
        do
        {
            aux = 0;
            /*approxiamtion of the continuation value*/
            for (k = 0; k < AL_Mesh_Size; k++)
            {
                aux += (Price[k * OP_Exercise_Dates + j + 1]) * Weight(j, Path
            }
            aux *= DiscountStep / (double)AL_Mesh_Size;
            VMesh.array = Path + j * BS_Dimension;
            aux -= p->Compute(p->Par, &VMesh);
            /*BlackScholes stock increment*/
            BS_Forward_Step(Path + (j + 1) * BS_Dimension, Path + j*BS_Dimensi
            j++;
        }
        while ((0 < aux) && (j < OP_Exercise_Dates - 1));
        /*MonteCarlo formulae for the forward price*/
        VMesh.array = Path + j * BS_Dimension;
        *AL_FPrice += Discount((double)(j) * Step, BS_Interest_Rate) * p->Comp
    }
}
/*output forward price*/
*AL_FPrice /= (double)AL_MonteCarlo_Iterations;
}
Close();

```

```

    return OK;
}

int CALC(MC_BroadieGlassermannND)(void *Opt, void *Mod, PricingMethod *Met)
{
    TYPEOPT *ptOpt = (TYPEOPT *)Opt;
    TYPEMOD *ptMod = (TYPEMOD *)Mod;
    double r;
    double *BS_cor;
    int i, res;
    PnlVect *divid = pnl_vect_create(ptMod->Size.Val.V_PINT);
    PnlVect *spot, *sig;

    spot = ptMod->S0.Val.V_PNLVECT;
    sig = ptMod->Sigma.Val.V_PNLVECT;

    for (i = 0; i < ptMod->Size.Val.V_PINT; i++)
        pnl_vect_set(divid, i,
                     log(1. + GET(ptMod->Divid.Val.V_PNLVECT, i) / 100.));

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

    if ((BS_cor = malloc(ptMod->Size.Val.V_PINT * ptMod->Size.Val.V_PINT * sizeof(
        return MEMORY_ALLOCATION_FAILURE;
    for (i = 0; i < ptMod->Size.Val.V_PINT * ptMod->Size.Val.V_PINT; i++)
        BS_cor[i] = ptMod->Rho.Val.V_DOUBLE;
    for (i = 0; i < ptMod->Size.Val.V_PINT; i++)
        BS_cor[i * ptMod->Size.Val.V_PINT + i] = 1.0;

    res = BrGl(spot,
               ptOpt->PayOff.Val.V_NUMFUNC_ND,
               ptOpt->Maturity.Val.V_DATE - ptMod->T.Val.V_DATE,
               r, divid, sig,
               BS_cor,
               Met->Par[0].Val.V_LONG,
               Met->Par[1].Val.V_ENUM.value,
               Met->Par[2].Val.V_INT,
               Met->Par[3].Val.V_INT,
               &(Met->Res[0].Val.V_DOUBLE),
               &(Met->Res[1].Val.V_DOUBLE));
    pnl_vect_free(&divid);
}

```

```

    free(BS_cor);

    return res;
}

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

    if ((opt->EuOrAm).Val.V_BOOL == AMER)
        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 = 10000;
        Met->Par[1].Val.V_ENUM.value = 0;
        Met->Par[1].Val.V_ENUM.members = &PremiaEnumMCRNGs;
        Met->Par[2].Val.V_INT = 200;
        Met->Par[3].Val.V_INT = 20;
    }

    return OK;
}

PricingMethod MET(MC_BroadieGlassermannND) =
{
    "MC_BroadieGlassermann_ND",
    { {"N iterations", LONG, {100}, ALLOW},
      {"RandomGenerator", ENUM, {0}, ALLOW},
      {"Mesh Size", INT, {100}, ALLOW},
      {"Number of Exercise Dates", INT, {100}, ALLOW},
    }
}

```



```

    {" ", PREMIA_NULLTYPE, {0}, FORBID}
},
CALC(MC_BroadieGlassermannND),
{ {"Price Forward", DOUBLE, {100}, FORBID}, {"Price Backward", DOUBLE, {100},
  {" ", PREMIA_NULLTYPE, {0}, FORBID}
},
CHK_OPT(MC_BroadieGlassermannND),
CHK_mc,
MET(Init)
};

```