

[Help](#)

```

#include <cmath>
#include <iostream>

extern "C" {
#include "bsnd_stdnd.h"
}
#include "math/bsnd_math/svd_bs_tools.h"

extern "C" {
#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2013+2) //The "#els
    static int CHK_OPT(FD_GreedySvd)(void *Opt, void *Mod)
    {
        return NONACTIVE;
    }
    int CALC(FD_GreedySvd)(void *Opt, void *Mod, PricingMethod *Met)
    {
        return AVAILABLE_IN_FULL_PREMIA;
    }
#else

    static double PrixPut_model_log(const PnlVect *step, const svd_bs::LogBS_Model
    {
        double res = contract.payoff(step, model.mu, model.r) * model.density(step,
        return res;
    }

    int GreedySvd(PnlVect *BS_Spot,
                  double strike,
                  double OP_Maturity,
                  double BS_Interest_Rate,
                  PnlVect *BS_Dividend_Rate,
                  PnlVect *BS_Volatility,
                  double rho,
                  int Npoints,
                  double *LowerPrice)

    {

```

```

//-----
// Numerical parameters
//-----
int dim = BS_Spot->size;
// We suppose the same discretisation at each dimension
int discretization = 1;

// Bndry is equal to 3 times the standard deviation of the stock returns
// (sqrt(T)*vol) when discretization = 1, if discretization=0 then Inf_Bndr
// Inf_Bndry = 0 in the case of the heat equation
double Inf_Bndry = -3 / (sqrt(OP_Maturity) * pnl_vect_max(BS_Volatility));
// Put Sup_Bndry=3/(sqrt(2)*0.1) when discretization=1 and Sup_Bndry=50*dim
// when discretization=0
// Sup_Bndry = 0 in the case of the heat equation
double Sup_Bndry = 3 / (sqrt(OP_Maturity) * pnl_vect_max(BS_Volatility));
// discretization = 0 if we work in the stock, and discretization = 1 if we
// work in the log(stock)

//-----
// Execution
//-----
// Pricing as a numerical integration of the svd form of the product
// payoff density
// Dans le fichier svd_decomposition.cpp, on doit tout commenter en
// tête sauf #define PRICING pour avoir les courbes de prix
PnlTridiagMat *M = svd_bs::initialise_mass_matrix(Npoints);
svd_bs::Option op(strike, OP_Maturity, BS_Spot, Inf_Bndry, Sup_Bndry);
svd_bs::LogBS_Model dynamic(BS_Interest_Rate, rho, BS_Volatility, OP_Maturit
svd_bs::non_linear_approximation svd(dim, Npoints, &PrixPut_model_log, dynam
svd.svd_decomposition(M);
*LowerPrice = svd.svd_integral();
pnl_tridiag_mat_free(&M);
return OK;
}

int CALC(FD_GreedySvd)(void *Opt, void *Mod, PricingMethod *Met)
{
    TYPEOPT *ptOpt = (TYPEOPT *)Opt;
    TYPEMOD *ptMod = (TYPEMOD *)Mod;
    double r, T, K;
    int i, res;

```

```

PnlVect *divid, *spot, *sig;

if (ptMod->Size.Val.V_PINT > 5)
{
    return MODEL_MAX_SIZE_EXCEEDED;
}

divid = pnl_vect_create(ptMod->Size.Val.V_PINT);
K = ptOpt->PayOff.Val.V_NUMFUNC_ND->Par[0].Val.V_DOUBLE;
T = ptOpt->Maturity.Val.V_DATE - ptMod->T.Val.V_DATE;
r = log(1. + ptMod->R.Val.V_DOUBLE / 100.);
spot = pnl_vect_compact_to_pnl_vect(ptMod->S0.Val.V_PNLVECTCOMPACT);
sig = pnl_vect_compact_to_pnl_vect(ptMod->Sigma.Val.V_PNLVECTCOMPACT);

for (i = 0; i < ptMod->Size.Val.V_PINT; i++)
    pnl_vect_set(divid, i,
        log(1. + pnl_vect_compact_get(ptMod->Divid.Val.V_PNLVECTCOMPACT, i)));

res = GreedySvd(spot, K, T, r, divid, sig,
    ptMod->Rho.Val.V_DOUBLE,
    Met->Par[0].Val.V_PINT,
    &(Met->Res[0].Val.V_DOUBLE));

if ((ptOpt->PayOff.Val.V_NUMFUNC_ND->Compute) == &CallBasket_nd)
{
    Met->Res[0].Val.V_DOUBLE = (pnl_vect_sum(spot) / spot->size - K * exp(-r * T));
}
pnl_vect_free(&divid);
pnl_vect_free(&spot);
pnl_vect_free(&sig);

return res;
}

static int CHK_OPT(FD_GreedySvd)(void *Opt, void *Mod)
{
    Option *ptOpt = (Option *)Opt;
    if ((strcmp(ptOpt->Name, "CallBasketEuro_nd") == 0) ||
        (strcmp(ptOpt->Name, "PutBasketEuro_nd") == 0))
        return OK;
}

```

```
        return WRONG;
    }

#endif //PremiaCurrentVersion
    static int MET(Init)(PricingMethod *Met, Option *Opt)
    {
        if (Met->init == 0)
        {
            Met->init = 1;
            Met->HelpFilenameHint = "FD_svd_bs";
        }

        return OK;
    }

PricingMethod MET(FD_GreedySvd) =
{
    "FD_Greedy",
    {{{"Discretization points per dimension", PINT, {51}, ALLOW}, {" ", PREMIA_NULLTYPE, {0}, FORBID}},
    CALC(FD_GreedySvd),
    {{{"Price", DOUBLE, {100}, FORBID}, {" ", PREMIA_NULLTYPE, {0}, FORBID}}},
    CHK_OPT(FD_GreedySvd),
    CHK_split,
    MET(Init)
};

}
```