

Help

```
#include <stdio.h>
#include <stdlib.h>
#include <
href../../common/math/cdo/cdo_math_h_src.pdfmath.h>
#include "pnl/pnl_linalgsolver.h"
#include "pnl/pnl_vector.h"
#include "pnl/pnl_matrix.h"
#include "pnl/pnl_tridiag_matrix.h"

double sigma_sqr(double t, double S, double sig, double alpha)
{
    return sig * sig * pow(S, 2 * (alpha - 1));
}

void quadratic_interpolation(double Fm1,
                             double F0,
                             double Fp1,
                             double Xm1,
                             double X0,
                             double Xp1,
                             double X,
                             double *FX,
                             double *dFX)
{
    //quadratic interpolation
    double A = Fm1;
    double B = (F0 - Fm1) / (X0 - Xm1);
    double C = (Fp1 - A - B * (Xp1 - Xm1)) / ((Xp1 - Xm1) * (Xp1 - X0));
    (*FX) = A + B * (X - Xm1) + C * (X - Xm1) * (X - X0);
    (*dFX) = B + C * (2 * X - Xm1 - X0);
}

typedef struct Parisian_Option
{
    int product;
    // 1 - Call ; 2 - Put;
    int product_type;
    // 1- Vanilla; 2 Up-and-Out ;3 Down-and-Out ; 4 Double barrier ;5
```

```

// parisian ...
double S0;
double K;
double T;
double rebate;
double barrier;
double duration;
double price;
double delta;
double implied_vol;
} Parisian_Option;

Parisian_Option *parisian_option_create(int product_, int product_type_,
                                         double S0_, double K_, double T_, double
{
    Parisian_Option *op = malloc(sizeof(Parisian_Option));
    op->product = product_;
    op->product_type = product_type_;
    op->S0 = S0_;
    op->K = K_;
    op->T = T_;
    op->rebate = rebate_;
    op->barrier = barrier_;
    op->duration = duration_;
    op->price = 0;
    op->delta = 0;
    op->implied_vol = 0;
    return op;
}

void parisian_option_compute_rigidity_matrix(PnlTridiagMat *A_rhs,
      PnlTridiagMat *A_lhs,
      PnlTridiagMat *B_rhs,
      PnlTridiagMat *B_lhs,
      double t,
      double dt,
      double h,
      double S0,
      double Bnd,
      double vol,

```

```

double coefcev,
double r,
double (*sigma_sqr)(double t, double S, double sig, double alpha),
double theta,
int Index_Barrier,
int N)
{

int i;
double a, b, alpha, beta, gama;

pnl_tridiag_mat_resize(A_rhs, Index_Barrier);
pnl_tridiag_mat_resize(A_lhs, Index_Barrier);
pnl_tridiag_mat_resize(B_rhs, N - Index_Barrier);
pnl_tridiag_mat_resize(B_lhs, N - Index_Barrier);

//printf("%lf %lf %lf %lf %lf %lf %d\ n",alpha,beta,gama,dt,theta,S0*exp(-1+In

for (i = 0; i < Index_Barrier; i++)
{
a = sigma_sqr(t, S0 * exp(-Bnd + (i + 1) * h), vol, coefcev) / (2 * h * h)
b = (r - sigma_sqr(t, S0 * exp(-Bnd + (i + 1) * h), vol, coefcev) / 2) / (
alpha = a - b;
beta = -2 * a - r;
gama = a + b;

pnl_tridiag_mat_set(A_rhs, i, 0, 1 + beta * dt * (1 - theta));
pnl_tridiag_mat_set(A_rhs, i, 1, gama * dt * (1 - theta));
pnl_tridiag_mat_set(A_lhs, i, 0, 1 - beta * dt * (theta));
pnl_tridiag_mat_set(A_lhs, i, 1, -gama * dt * (theta));

pnl_tridiag_mat_set(A_rhs, i, -1, alpha * dt * (1 - theta));
pnl_tridiag_mat_set(A_lhs, i, -1, -alpha * dt * theta);
pnl_tridiag_mat_set(A_rhs, i, 1, gama * dt * (1 - theta));
pnl_tridiag_mat_set(A_lhs, i, 1, -gama * dt * (theta));
}

pnl_tridiag_mat_set(A_rhs, 0, -1, 0);
pnl_tridiag_mat_set(A_lhs, 0, -1, 0);
pnl_tridiag_mat_set(A_rhs, Index_Barrier - 1, 1, 0);
pnl_tridiag_mat_set(A_lhs, Index_Barrier - 1, 1, 0);

```

```

for (i = 0; i < N - Index_Barrier; i++)
{
    a = sigma_sqr(t, S0 * exp(-Bnd + (i + 1 + Index_Barrier) * h), vol, coefce
    b = (r - sigma_sqr(t, S0 * exp(-Bnd + (i + 1 + Index_Barrier) * h), vol, c
    alpha = a - b;
    beta = -2 * a - r;
    gama = a + b;
    if (i == 0)
    {
        pnl_tridiag_mat_set(B_rhs, i, 0, 1 + beta * dt * (1 - theta));
        pnl_tridiag_mat_set(B_rhs, i, 1, gama * dt * (1 - theta));
        pnl_tridiag_mat_set(B_lhs, i, 0, 1 - beta * dt * (theta));
        pnl_tridiag_mat_set(B_lhs, i, 1, -gama * dt * (theta));
    }
    if ((i > 0) && (i < N - Index_Barrier - 1))
    {
        pnl_tridiag_mat_set(B_rhs, i, -1, alpha * dt * (1 - theta));
        pnl_tridiag_mat_set(B_rhs, i, 0, 1 + beta * dt * (1 - theta));
        pnl_tridiag_mat_set(B_rhs, i, 1, gama * dt * (1 - theta));
        pnl_tridiag_mat_set(B_lhs, i, -1, -alpha * dt * (theta));
        pnl_tridiag_mat_set(B_lhs, i, 0, 1 - beta * dt * (theta));
        pnl_tridiag_mat_set(B_lhs, i, 1, -gama * dt * (theta));
    }
    if (i == N - Index_Barrier - 1)
    {
        pnl_tridiag_mat_set(B_rhs, i, -1, alpha * dt * (1 - theta));
        pnl_tridiag_mat_set(B_rhs, i, 0, 1 + beta * dt * (1 - theta));
        pnl_tridiag_mat_set(B_lhs, i, -1, -alpha * dt * (theta));
        pnl_tridiag_mat_set(B_lhs, i, 0, 1 - beta * dt * (theta));
    }
}

// pnl_tridiag_mat_print(A_rhs);
//pnl_tridiag_mat_print(A_lhs);
//pnl_tridiag_mat_print(B_rhs);
//pnl_tridiag_mat_print(B_lhs);
//printf("ok for tri\ n");

}

```

```

int main()
{

    double S0;//=100;
    double T;//=1;
    double Y;//=140;
    double D;//=0.5;
    double K;//=100;
    double r;//=0.05;
    double vol;//=0.2;
    double price, price_1, price1, price_2, price2;
    double delta, delta_1, delta1;
    double gamma;
    double Teta;
    PnlTridiagMat *A_rhs, *A_lhs, *B_rhs, *B_lhs;
    PnlVect Vtmp, Vtmp2;
    PnlVect *VaA, *VbA;
    PnlMat *VaB, *VbB;
    PnlVect *ResAtmp, *ResBtmp;
    double Bnd, coefcev, dt, h, theta;
    int i, j, k, J, Index_Barrier;
    double t;
    double a, b, gama, alpha;
    int M, N;

    M = 3;
    N = 10; //500

    A_rhs = pnl_tridiag_mat_create(0);
    A_lhs = pnl_tridiag_mat_create(0);
    B_rhs = pnl_tridiag_mat_create(0);
    B_lhs = pnl_tridiag_mat_create(0);

    t = 0; // Sigma not depend of t
    /*
    printf("Please input the parameters according to the following format\ n");
    printf("S0 Vol R K Delay Barrier Maturity\ n");
    scanf("%lf %lf %lf %lf %lf %lf %lf", &S0, &vol, &r, &K, &D, &Y, &T);
    printf("Your parameters are as follows\ t\ t\ n");
    printf("S0\ t %f\ n", S0);

```

```

    printf("Vol\ t %f\ n", vol);
    printf("R\ t %f\ n", r);
    printf("K\ t %f\ n", K);
    printf("D\ t %f\ n", D);
    printf("Barrier\ t %f\ n", Y);
    printf("T\ t %f\ n", T);
*/
S0 = 100;
T = 1;
Y = 140;
D = 0.5;
K = 100;
r = 0.05;
vol = 2.0;

coefcev = 0.5;
Bnd = vol * sqrt(T) * 5 * pow(S0, (coefcev - 1));
//Bnd=MAX(Bnd,2*log(Y/So));
dt = T / M;
h = 2 * Bnd / N;
theta = 0.6;
J = floor(D / dt) + 1;
Index_Barrier = floor((log(Y / S0) + Bnd) / h);

parisian_option_compute_rigidity_matrix(A_rhs, A_lhs, B_rhs, B_lhs,
                                         t, dt, h, S0, Bnd, vol, coefcev, r,
                                         &sigma_sqr, theta, Index_Barrier, N);

VaA = pnl_vect_create_from_double(Index_Barrier, 0.);
VbA = pnl_vect_create_from_double(Index_Barrier, 0.);
VaB = pnl_mat_create_from_double(J, N - Index_Barrier, 0.);
VbB = pnl_mat_create_from_double(J, N - Index_Barrier, 0.);

ResAtmp = pnl_vect_create_from_double(Index_Barrier, 0.);
ResBtmp = pnl_vect_create_from_double(N - Index_Barrier, 0.);

for (i = 0; i < Index_Barrier; i++)

```

```

{
    pnl_vect_set(VaA, i, MAX(S0 * exp(-Bnd + (i + 1)*h) - K, 0));
}

Vtmp = pnl_mat_wrap_row(VaB, 1);
for (i = 0; i < N - Index_Barrier; i++)
    pnl_vect_set(&Vtmp, i, MAX(S0 * exp(-Bnd + (i + Index_Barrier + 1)*h) - K, 0)
for (j = 2; j < J; j++)
{
    Vtmp2 = pnl_mat_wrap_row(VaB, j);
    pnl_vect_clone(&Vtmp2, &Vtmp);
}

//pnl_vect_print(VaA);
//pnl_mat_print(VaB);
//printf("ok for Va initial \ n");

for (k = 1; k <= M; k++)
{
    a = sigma_sqr(t, S0 * exp(-Bnd + Index_Barrier * h), vol, coefcev) / (2 *
    b = (r - sigma_sqr(t, S0 * exp(-Bnd + Index_Barrier * h), vol, coefcev) /
    gama = a + b;
    //printf(">>> %d \ n ", (int)((double)k/(double)M*100));
    pnl_tridiag_mat_mult_vect_inplace(ResAtmp, A_rhs, VaA);
    pnl_vect_set(ResAtmp, Index_Barrier - 1, pnl_vect_get(ResAtmp, Index_Barrier
        + dt * gama * pnl_mat_get(VaB, J - 2, 0));
    //pnl_vect_print(ResAtmp);
    pnl_tridiag_mat_lu_syslin(VbA, A_lhs, ResAtmp);
    //pnl_vect_print(VbA);
    a = sigma_sqr(t, S0 * exp(-Bnd + (Index_Barrier + 1) * h), vol, coefcev) /
    b = (r - sigma_sqr(t, S0 * exp(-Bnd + (Index_Barrier + 1) * h), vol, coefcev) /
    alpha = a - b;
    for (j = 0; j < J - 1; j++)
    {
        Vtmp = pnl_mat_wrap_row(VaB, j);
        pnl_tridiag_mat_mult_vect_inplace(ResBtmp, B_rhs, &Vtmp);
        pnl_vect_set(ResBtmp, 0, pnl_vect_get(ResBtmp, 0) + (1 - theta)*dt * a
        Vtmp = pnl_mat_wrap_row(VbB, j + 1);
        pnl_tridiag_mat_lu_syslin(&Vtmp, B_lhs, ResBtmp);
    }
}

```

```

        //pnl_vect_print(VbA);
        //pnl_mat_print(VbB);
        if (k == M)
        {
            if (N / 2 <= Index_Barrier)
                Teta = (pnl_vect_get(VaA, N / 2 - 1) - pnl_vect_get(VbA, N / 2 - 1))
            else
                Teta = (pnl_mat_get(VaB, J - 1, N / 2 - 1 - Index_Barrier) - pnl_mat
            }
            pnl_vect_clone(VaA, VbA);
            pnl_mat_clone(VaB, VbB);
        }

        if ((N / 2 - 2) <= Index_Barrier)
            price_2 = pnl_vect_get(VaA, N / 2 - 3);
        else
            price_2 = pnl_mat_get(VaB, J - 1, N / 2 - 3 - Index_Barrier);

        if ((N / 2 - 1) <= Index_Barrier)
            price_1 = pnl_vect_get(VaA, N / 2 - 2);
        else
            price_1 = pnl_mat_get(VaB, J - 1, N / 2 - 2 - Index_Barrier);

        if (N / 2 <= Index_Barrier)
            price = pnl_vect_get(VaA, N / 2 - 1);
        else
            price = pnl_mat_get(VaB, J - 1, N / 2 - 1 - Index_Barrier);

        if ((N / 2 + 1) <= Index_Barrier)
            price1 = pnl_vect_get(VaA, N / 2);
        else
            price1 = pnl_mat_get(VaB, J - 1, N / 2 - Index_Barrier);

        if ((N / 2 + 2) <= Index_Barrier)
            price2 = pnl_vect_get(VaA, N / 2 + 1);
        else
            price2 = pnl_mat_get(VaB, J - 1, N / 2 + 1 - Index_Barrier);

        delta_1 = 0.5 * ((price_1 - price_2) / (S0 * exp(-h) - S0 * exp(-2 * h)) + (pr

```

```

delta = 0.5 * ((price - price_1) / (S0 - S0 * exp(-h)) + (price1 - price) / (S
delta1 = 0.5 * ((price1 - price) / (S0 * exp(h) - S0) + (price2 - price1) / (S
gamma = 0.5 * ((delta - delta_1) / (S0 - S0 * exp(-h)) + (delta1 - delta) / (S

printf("\ nprice is %lf\ n", price);
printf("delta is %lf\ n", delta);
printf("gamma is %lf\ n", gamma);
printf("theta is %lf\ n", Teta);

pnl_tridiag_mat_free(&A_rhs);
pnl_tridiag_mat_free(&A_lhs);
pnl_tridiag_mat_free(&B_rhs);
pnl_tridiag_mat_free(&B_lhs);
pnl_vect_free(&ResAtmp);
pnl_vect_free(&ResBtmp);
pnl_mat_free(&VaB);
pnl_mat_free(&VbB);
pnl_vect_free(&VaA);
pnl_vect_free(&VbA);

return 0;
}

```