

[Help](#)

```

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2008+2) //The "#els
#else

#include "functions.h"
#include "common.h"
#include<iostream>
#include <iostream>
#include<cmath>
extern "C" {
#include "pnl/pnl_mathtools.h"
}
using namespace std;

long double inverselaplace::InvLF(const long double &T)const
{
    int n = 10, k;

    std::vector<long double> v;
    long double deux = 2.0;
    v.resize(n + 1);
    v[1] = 0.0833333333;
    v[2] = -32.08333333;
    v[3] = 1279.000076;
    v[4] = -15623.66689;
    v[5] = 84244.16946;
    v[6] = -236957.5129;
    v[7] = 375911.6923;
    v[8] = -340071.6923;
    v[9] = 164062.5128;
    v[10] = -32812.50256;
    long double s = 0.0;
    for (k = 1; k <= n; k++)
    {
        s = s + v[k] * (*LF).f(k * log(deux) / T);
    }

    return s * log(deux) / T;
}
std::vector<long double> newton::racine()const

```

```
{
    long double x = start, x0 = start, temp;
    std::vector<long double> sol(1);
    int n = 1;
    do
    {
        temp = x;
        x = x0 - (*F).f(x0) / (*F).Df(x0);
        x0 = temp;
        n++;
    }
    while ((*F).f(x) > accuracy && n <= 1000);
    sol[0] = x;
    return sol;
}

std::vector<long double> secante::racine()const
{
    long double x = start1, x0 = start0, x1 = start1, temp;
    std::vector<long double> sol(1);
    int n = 1;
    do
    {
        temp = x;
        x = x1 - (*F).f(x1) / (((*F).f(x1) - (*F).f(x0)) / (x1 - x0));
        x0 = temp;
        x1 = x;
        n++;
    }
    while ((*F).f(x) > accuracy && n <= 1000);
    sol[0] = x;
    return sol;
}

std::vector<long double> dichotomie::racine()const
{
    long double x0 = a, x1 = b, x, fx0 = (*F).f(a), fx1 = (*F).f(b), fx;
    std::vector<long double> sol(1);

    x = (x0 + x1) / 2;
    fx = (*F).f(x);
    sol[0] = x0;
    if (abs(fx0) <= accuracy)
```

```

        return sol;
    sol[0] = x1;
    if (abs(fx1) <= accuracy)
        return sol;
    sol[0] = x;
    if (abs(fx) <= accuracy)
        return sol;
    while (abs(fx) > accuracy)
    {
        if (fx * fx0 < 0)
        {
            x1 = x;
            fx1 = fx;
        }
        else
        {
            x0 = x;
            fx0 = fx;
        }
        x = (x0 + x1) / 2;
        fx = (*F).f(x);
    }
    sol[0] = x;
    return sol;
}

long double KCE::f(const long double &x)const
{
    double
    nu = param[0],
    sigma = param[1],
    lambda = param[2],
    p = param[3],
    eta1 = param[4],
    eta2 = param[5],
    alpha = param[6],
    q = param[7];
    return nu * x + sigma * sigma * x * x / 2 + lambda * (p * eta1 / (eta1 - x) +
}

long double KCE::Df(const long double &x)const
{
    double

```

```

    nu = param[0],
    sigma = param[1],
    lambda = param[2],
    p = param[3],
    eta1 = param[4],
    eta2 = param[5],
    q = param[7];
    return nu + sigma * sigma * x + lambda * (p * eta1 / ((eta1 - x) * (eta1 - x))
}
long double LPLB::f(const long double &alpha) const
{
    long double
    nu = param[0],
    sigma = param[1],
    lambda = param[2],
    p = param[3],
    eta1 = param[4],
    eta2 = param[5],
    S0 = param[6],
    r = param[7],
    M = param[9],
    delta = param[10],
    eps = 0.000000000000000001;
    long double beta1, beta2, x, x0;
    long double y[8] = {nu, sigma, lambda, p, eta1, eta2, alpha + r, 1. - p};
    KCE G(y);
    x = eta1 - 0.1;
    while (G.f(x) < 0)
        x = x + 1e-2;
    dichotomie solG(G, eps, x, 0.00001);
    beta1 = solG.racine()[0];
    newton solG1(G, beta1, eps);
    beta1 = solG1.racine()[0];

    x = eta1 + 10;
    x0 = eta1 + 0.1;
    while (G.f(x0) > 0)
        x0 = x0 - 1e-2;
    while (G.f(x) < 0)
        x = x + 10;
    dichotomie solG2(G, x0, x, 0.00001);

```

```

    beta2 = solG2.racine()[0];
    newton solG3(G, beta2, eps);
    beta2 = solG3.racine()[0];

    long double A = (eta1 - beta1) * beta2 / (beta1 - 1);
    long double B = (beta2 - eta1) * beta1 / (beta2 - 1);
    long double C = (alpha + r) * eta1 * (beta2 - beta1);

    return S0 * A / C * pow(S0 / M, beta1 - 1) + S0 * B / C * pow(S0 / M, beta2 - 1);
}
long double DLPLB::f(const long double &alpha) const
{
    long double
    nu = param[0],
    sigma = param[1],
    lambda = param[2],
    p = param[3],
    eta1 = param[4],
    eta2 = param[5],
    S0 = param[6],
    r = param[7],
    M = param[9],
    delta = param[10],
    eps = 0.000000000000000001;
    long double beta1, beta2, x, x0;
    long double y[8] = {nu, sigma, lambda, p, eta1, eta2, alpha + r, 1. - p};
    KCE G(y);
    x = eta1 - 0.1;
    while (G.f(x) < 0)
        x = x + 1e-2;
    dichotomie solG(G, eps, x, 0.00001);
    beta1 = solG.racine()[0];
    newton solG1(G, beta1, eps);
    beta1 = solG1.racine()[0];

    x = eta1 + 10;
    x0 = eta1 + 0.1;
    while (G.f(x0) > 0)
        x0 = x0 - 1e-2;
    while (G.f(x) < 0)
        x = x + 10;
}

```

```

dichotomie solG2(G, x0, x, 0.00001);
beta2 = solG2.racine()[0];
newton solG3(G, beta2, eps);
beta2 = solG3.racine()[0];

long double A = (eta1 - beta1) * beta2 / (beta1 - 1);
long double B = (beta2 - eta1) * beta1 / (beta2 - 1);
long double C = (alpha + r) * eta1 * (beta2 - beta1);

return beta1 * A / C * pow(S0 / M, beta1 - 1) + beta2 * B / C * pow(S0 / M, beta2 - 1);
}

long double LCLB::f(const long double &alpha) const
{
    long double
    nu = param[0],
    sigma = param[1],
    lambda = param[2],
    p = param[3],
    eta1 = param[4],
    eta2 = param[5],
    S0 = param[6],
    r = param[7],
    m = param[9],
    delta = param[10],
    eps = 1e-16;
    long double beta1, beta2, x, x0;
    long double y[8] = {nu, sigma, lambda, p, eta1, eta2, alpha + r, 1. - p};
    KCE G(y);
    x = eta1 - 0.1;
    while (G.f(x) < 0)
        x = x + 1e-2;
    dichotomie solG(G, eps, x, 1e-5);
    beta1 = solG.racine()[0];
    newton solG1(G, beta1, eps);
    beta1 = solG1.racine()[0];

    x = eta1 + 10;
    x0 = eta1 + 0.1;
    while (G.f(x0) > 0)
        x0 = x0 - 1e-2;

```

```

while (G.f(x) < 0)
    x = x + 10;
dichotomie solG2(G, x0, x, 1e-5);
beta2 = solG2.racine()[0];
newton solG3(G, beta2, eps);
beta2 = solG3.racine()[0];

long double A = (eta1 - beta1) * beta2 / (beta1 + 1);
long double B = (beta2 - eta1) * beta1 / (beta2 + 1);
long double C = (alpha + r) * eta1 * (beta2 - beta1);

return S0 * A / C * pow(m / S0, beta1 + 1) + S0 * B / C * pow(m / S0, beta2 +
}
long double DLCLB::f(const long double &alpha) const
{
    long double
    nu = param[0],
    sigma = param[1],
    lambda = param[2],
    p = param[3],
    eta1 = param[4],
    eta2 = param[5],
    S0 = param[6],
    r = param[7],
    m = param[9],
    delta = param[10],
    eps = 1e-16;
    long double beta1, beta2, x, x0;
    long double y[8] = {nu, sigma, lambda, p, eta1, eta2, alpha + r, 1. - p};
    KCE G(y);
    x = eta1 - 0.1;
    while (G.f(x) < 0)
        x = x + 1e-2;
    dichotomie solG(G, eps, x, 1e-5);
    beta1 = solG.racine()[0];
    newton solG1(G, beta1, eps);
    beta1 = solG1.racine()[0];

    x = eta1 + 10;
    x0 = eta1 + 0.1;
    while (G.f(x0) > 0)

```

```

    x0 = x0 - 1e-2;
while (G.f(x) < 0)
    x = x + 10;
dichotomie solG2(G, x0, x, 1e-5);
beta2 = solG2.racine()[0];
newton solG3(G, beta2, eps);
beta2 = solG3.racine()[0];

long double A = (eta1 - beta1) * beta2 / (beta1 + 1);
long double B = (beta2 - eta1) * beta1 / (beta2 + 1);
long double C = (alpha + r) * eta1 * (beta2 - beta1);

return -beta1 * A / C * pow(m / S0, beta1 + 1) - beta2 * B / C * pow(m / S0, b
}
long double LPsiM::f(const long double &alpha)const
{
    long double
    nu = param[0],
    sigma = param[1],
    lambda = param[2],
    p = param[3],
    eta1 = param[4],
    eta2 = param[5],
    b = param[7],
    eps = 1e-16;
    long double beta1, beta2, x, x0;
    long double y[8] = {nu, sigma, lambda, p, eta1, eta2, alpha, 1 - p};
    KCE G(y);
    x = eta1 - 0.1;
    while (G.f(x) < 0)
        x = x + 1e-2;
    dichotomie solG(G, eps, x, 1e-5);
    beta1 = solG.racine()[0];
    newton solG1(G, beta1, eps);
    beta1 = solG1.racine()[0];

    x = eta1 + 10;
    x0 = eta1 + 0.1;
    while (G.f(x0) > 0)
        x0 = x0 - 1e-2;
    while (G.f(x) < 0)

```



```

    x = x + 10;
    dichotomie solG2(G, x0, x, 1e-5);
    beta2 = solG2.racine()[0];
    newton solG3(G, beta2, eps);
    beta2 = solG3.racine()[0];

    return ((eta1 - beta1) * beta2 * expl(-b * beta1) + (beta2 - eta1) * beta1 * e
}
long double LdPsiM::f(const long double &alpha)const
{
    long double
    nu = param[0],
    sigma = param[1],
    lambda = param[2],
    p = param[3],
    eta1 = param[4],
    eta2 = param[5],
    b = param[7],
    eps = 1e-16;
    long double beta1, beta2, x, x0;
    long double y[8] = {nu, sigma, lambda, p, eta1, eta2, alpha, 1 - p};
    KCE G(y);
    x = eta1 - 0.1;
    while (G.f(x) < 0)
        x = x + 1e-2;
    dichotomie solG(G, eps, x, 1e-5);
    beta1 = solG.racine()[0];
    newton solG1(G, beta1, eps);
    beta1 = solG1.racine()[0];

    x = eta1 + 10;
    x0 = eta1 + 0.1;
    while (G.f(x0) > 0)
        x0 = x0 - 1e-2;
    while (G.f(x) < 0)
        x = x + 10;
    dichotomie solG2(G, x0, x, 1e-5);
    beta2 = solG2.racine()[0];
    newton solG3(G, beta2, eps);
    beta2 = solG3.racine()[0];

```

```

    return ((eta1 - beta1) * beta2 * expl(-b * beta1) + (beta2 - eta1) * beta1 * e
}

```

```

long double LPSiMA::f(const long double &alpha)const
{
    long double
    nu = param[0],
    sigma = param[1],
    lambda = param[2],
    p = param[3],
    eta1 = param[4],
    eta2 = param[5],
    b = param[7],
    eps = 1e-16;
    long double beta1, beta2, x, x0;
    long double y[8] = {nu, sigma, lambda, p, eta1, eta2, alpha, 1 - p};
    KCE G(y);
    x = eta1 - 0.1;
    while (G.f(x) < 0)
        x = x + 1e-2;
    dichotomie solG(G, eps, x, 1e-5);
    beta1 = solG.racine()[0];
    newton solG1(G, beta1, eps);
    beta1 = solG1.racine()[0];

    x = eta1 + 10;
    x0 = eta1 + 0.1;
    while (G.f(x0) > 0)
        x0 = x0 - 1e-2;
    while (G.f(x) < 0)
        x = x + 10;
    dichotomie solG2(G, x0, x, 1e-5);
    beta2 = solG2.racine()[0];
    newton solG3(G, beta2, eps);
    beta2 = solG3.racine()[0];

    return ((eta1 - beta1) * expl(-b * beta1) + (beta2 - eta1) * expl(-b * beta2))
}
long double LPSiMB::f(const long double &alpha)const
{
    long double

```

```

nu = param[0],
sigma = param[1],
lambda = param[2],
p = param[3],
eta1 = param[4],
eta2 = param[5],
b = param[7],
eps = 1e-16;
long double beta1, beta2, x, x0;
long double y[8] = {nu, sigma, lambda, p, eta1, eta2, alpha, 1 - p};
KCE G(y);
x = eta1 - 0.1;
while (G.f(x) < 0)
    x = x + 1e-2;
dichotomie solG(G, eps, x, 1e-5);
beta1 = solG.racine()[0];
newton solG1(G, beta1, eps);
beta1 = solG1.racine()[0];

x = eta1 + 10;
x0 = eta1 + 0.1;
while (G.f(x0) > 0)
    x0 = x0 - 1e-2;
while (G.f(x) < 0)
    x = x + 10;
dichotomie solG2(G, x0, x, 1e-5);
beta2 = solG2.racine()[0];
newton solG3(G, beta2, eps);
beta2 = solG3.racine()[0];

return ((eta1 - beta1) * (beta2 - eta1) * (expl(-b * beta1) - expl(-b * beta2))
}
long double amer_eq::f(const long double &v) const
{
    long double
ksi = param[2] * param[3] / (param[3] - 1) + (1 - param[2]) * param[4] / (para
nu = (param[7] - param[9]) - param[0] * param[0] / 2 - param[1] * ksi,

sigma = param[0],
lambda = param[1],
p = param[2],

```

```

eta1 = param[3],
eta2 = param[4],
K = param[6],
r = param[7],
delta = param[9],
T = param[8],
C = param[10],
D = param[11];

long double x[8] = {nu, sigma, lambda, p, eta1, eta2, log(K / v), T};
long double cst1 = psiVN(x, 7);
x[0] = r - delta + sigma * sigma / 2 - lambda * ksi;
x[2] = lambda * (ksi + 1);
x[3] = p * eta1 / ((1 + ksi) * (eta1 - 1));
x[4] = eta1 - 1;
x[5] = eta2 + 1;

long double cst2 = psiVN(x, 7);
long double EuP = K * exp(-r * T) * (1 - cst1) - v * exp(-delta * T) * (1 - cst2);
long double proba = 1 - cst1;

return C * K - D * (v * exp(-delta * T) + EuP) - (C - D) * K * exp(-r * T) * p
}
////////////////////////////////////
//fonction psi=P[ZT>=a] ou Z levy poisson compose long
//double exponentielle
long double psiVN(long double *x, int Nb)
{
    long double
    nu = x[0],
    sigma = x[1],
    lambda = x[2],
    p = x[3],
    eta1 = x[4],
    eta2 = x[5],
    a = x[6],
    T = x[7];

    if (T < 1e-16)
        return (a > 0 ? 0.0 : 1.0);
    std::vector<std::vector<long double>> > pm, qm;

```

```

std::vector<long double> pi;
long double zr = 0.0;
int i, k, n;

pm.resize(Nb + 1);
for (i = 0; i <= Nb; i++)
    pm[i].resize(i + 1);
qm.resize(Nb + 1);
for (i = 0; i <= Nb; i++)
    qm[i].resize(i + 1);
pi.resize(Nb + 1);
pi[0] = expl(-x[2] * x[7]);
for (n = 1; n <= Nb; n++)
{
    pi[n] = expl(-lambda * T) * powl(lambda * T, n) / (fact_dia(n));
    pm[n][n] = powl(p, n);
    qm[n][n] = powl(1 - p, n);
    for (k = 1; k < n; k++)
    {
        pm[n][k] = zr;
        qm[n][k] = zr;
        for (i = k; i < n; i++)
        {
            pm[n][k] = pm[n][k] + bin_dia(n, i) * powl(x[3], i) * powl(x[4] /
            qm[n][k] = qm[n][k] + bin_dia(n, i) * powl(1 - x[3], i) * powl(x[3]
        }
    }
}

long double s1 = zr, s2 = zr;
long double c1 = expl(sigma * eta1 * sigma * eta1 * T / 2) / (sigma * sqrtl(2
long double c2 = expl(sigma * eta2 * sigma * eta2 * T / 2) / (sigma * sqrtl(2

long double c3, c4;
std::vector<long double> Inv1(Nb), Inv2(Nb);

Inv1 = In(a - nu * T, -eta1, -1. / (sigma * sqrtl(T)), -sigma * eta1 * sqrtl(T)
Inv2 = In(a - nu * T, eta2, 1. / (sigma * sqrtl(T)), -sigma * eta2 * sqrtl(T),
c3 = sigma * sqrtl(T) * eta1;

```

```

c4 = sigma * sqrtl(T) * eta2;
for (n = 1; n <= Nb; n++)
{
    for (k = 1; k <= n; k++)
    {
        s1 = s1 + pi[n] * pm[n][k] * powl(c3, k) * Inv1[k - 1];
        s2 = s2 + pi[n] * qm[n][k] * powl(c4, k) * Inv2[k - 1];
    }
}
if (c1 > 1e+100)
{
    c1 = 1.;
    s1 = (1 - pi[0]) * p * Hh0((a - nu * T) / (sigma * sqrtl(T))) / sqrtl(2 *
}
if (c2 > 1e+100)
{
    c2 = 1.;
    s2 = (1 - pi[0]) * (1 - p) * Hh0((a - nu * T) / (sigma * sqrtl(T))) / sqrt
}
return c1 * s1 + c2 * s2 + pi[0] * Hh0((a - nu * T) / (sigma * sqrtl(T))) / sq
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//derivÃ de psi=P[ZT>=a] % Ã a
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//derivÃÃ de psi=P[ZT>=a] % ÃÃ a
long double dpsivN(long double *x, int Nb)
{
    long double
    nu = x[0],
    sigma = x[1],
    lambda = x[2],
    p = x[3],
    eta1 = x[4],
    eta2 = x[5],
    a = x[6],
    T = x[7];

    std::vector<std::vector<long double> > pm, qm;
    std::vector<long double> pi;
    long double zr = 0;
    int i, k, N;

```

```

pm.resize(Nb + 1);
for (i = 0; i <= Nb; i++)
    pm[i].resize(i + 1);
qm.resize(Nb + 1);
for (i = 0; i <= Nb; i++)
    qm[i].resize(i + 1);
pi.resize(Nb + 1);
pi[0] = exp(-x[2] * x[7]);
for (N = 1; N <= Nb; N++)
{
    pi[N] = exp(-lambda * T) * pow(lambda * T, N) / fact_dia(N);
    pm[N][N] = pow(p, N);
    qm[N][N] = pow(1 - p, N);
    for (k = 1; k < N; k++)
    {
        pm[N][k] = zr;
        qm[N][k] = zr;
        for (i = k; i < N; i++)
        {
            pm[N][k] = pm[N][k] + bin_dia(N, i) * pow(x[3], i) * pow(x[4] / (x[3] - 1), N - i);
            qm[N][k] = qm[N][k] + bin_dia(N, i) * pow(1 - x[3], i) * pow(x[3], N - i);
        }
    }
}

long double s1 = zr, s2 = zr;
long double c1 = exp(sigma * eta1 * sigma * eta1 * T / 2) / (sigma * sqrt(2 * sigma * T));
long double c2 = exp(sigma * eta2 * sigma * eta2 * T / 2) / (sigma * sqrt(2 * sigma * T));
long double c3, c4;
std::vector<long double> Inv1(Nb), Inv2(Nb);
Inv1 = dIn(a - nu * T, -eta1, -1 / (sigma * sqrt(T)), -sigma * eta1 * sqrt(T), N);
Inv2 = dIn(a - nu * T, eta2, 1 / (sigma * sqrt(T)), -sigma * eta2 * sqrt(T), N);
for (N = 1; N <= Nb; N++)
{
    c3 = sigma * sqrt(T) * eta1;
    c4 = sigma * sqrt(T) * eta2;
    for (k = 1; k <= N; k++)
    {
        s1 = s1 + pi[N] * pm[N][k] * c3 * Inv1[k - 1];
    }
}

```

```

        s2 = s2 + pi[N] * qm[N][k] * c4 * Inv2[k - 1];
        c3 = c3 * sigma * sqrt(T) * eta1;
        c4 = c4 * sigma * sqrt(T) * eta2;
    }
}
if (c1 > 1e+100)
{
    c1 = 1.;
    s1 = -(1 - pi[0]) * p * dnorm(-(a - nu * T) / (sigma * sqrt(T))) / (sigma
}
if (c2 > 1e+100)
{
    c2 = 1.;
    s2 = -(1 - pi[0]) * (1 - p) * dnorm(-(a - nu * T) / (sigma * sqrt(T))) / (
}
return c1 * s1 + c2 * s2 - pi[0] * dnorm(-(a - nu * T) / (sigma * sqrt(T))) /
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//fonction psi=P[ZT+ksi+>=a] ou Z levy poisson compose long
//double exponentielle
std::vector<long double> psiVNB(long double *x, int Nb)
{
    long double
    nu = x[0],
    sigma = x[1],
    lambda = x[2],
    p = x[3],
    eta1 = x[4],
    eta2 = x[5],
    a = x[6],
    T = x[7];
    std::vector<long double> pi, y;
    y.resize(2);
    std::vector<std::vector<long double> > pm, qm;
    long double zr = 0.0;
    int i, k, n, j;

    if (T < 1e-16)
    {
        if (a > 0)
        {

```



```

        y[0] = 0.0;
        y[1] = expl(-eta1 * a);
    }
    else
    {
        y[0] = 1.0;
        y[1] = 1.0;
    }

    return y;
}

pm.resize(Nb + 1);
for (i = 0; i <= Nb; i++)
    pm[i].resize(i + 1);
qm.resize(Nb + 1);
for (i = 0; i <= Nb; i++)
    qm[i].resize(i + 1);
pi.resize(Nb + 1);
pi[0] = expl(-x[2] * x[7]);
for (n = 1; n <= Nb; n++)
{
    pi[n] = expl(-lambda * T) * powl(lambda * T, n) / (fact_dia(n));
    pm[n][n] = powl(p, n);
    qm[n][n] = powl(1 - p, n);
    for (k = 1; k < n; k++)
    {
        pm[n][k] = zr;
        qm[n][k] = zr;
        for (i = k; i < n; i++)
        {
            pm[n][k] = pm[n][k] + bin_dia(n, i) * powl(x[3], i) * powl(x[4] /
            qm[n][k] = qm[n][k] + bin_dia(n, i) * powl(1 - x[3], i) * powl(x[3]
        }
    }
}

std::vector<std::vector<long double> > pmb, qmb;
pmb.resize(Nb + 1);
for (i = 0; i <= Nb; i++)
    pmb[i].resize(i + 2);

```

```

qmb.resize(Nb + 1);
for (i = 0; i <= Nb; i++)
    qmb[i].resize(i + 1);

for (n = 1; n <= Nb; n++)
{
    pmb[n][1] = 0;
    for (i = 1; i <= n; i++)
        pmb[n][1] = pmb[n][1] + qm[n][i] * pow(eta2 / (eta2 + eta1), i);
    for (i = 2; i <= n + 1; i++)
    {
        pmb[n][i] = pm[n][i - 1];
    }
    for (i = 1; i <= n; i++)
    {
        qmb[n][i] = 0;
        for (j = i; j <= n; j++)
            qmb[n][i] = qmb[n][i] + (eta1 / (eta1 + eta2)) * pow(eta2 / (eta1 +
    }
}

long double s1 = zr, s2 = zr, s3 = zr, s4 = zr;
long double c1 = expl(sigma * eta1 * sigma * eta1 * T / 2) / (sigma * sqrtl(2
long double c2 = expl(sigma * eta2 * sigma * eta2 * T / 2) / (sigma * sqrtl(2
long double c = expl(sigma * eta1 * sigma * eta1 * T / 2) * eta1 / sqrtl(2 * M
long double c3, c4;
std::vector<long double> Inv1(Nb + 1), Inv2(Nb);

Inv1 = In(a - nu * T, -eta1, -1. / (sigma * sqrtl(T)), -sigma * eta1 * sqrtl(T)
Inv2 = In(a - nu * T, eta2, 1. / (sigma * sqrtl(T)), -sigma * eta2 * sqrtl(T),
c3 = sigma * sqrtl(T) * eta1;
c4 = sigma * sqrtl(T) * eta2;
for (n = 1; n <= Nb; n++)
{
    for (k = 1; k <= n; k++)
    {
        s1 += pi[n] * pm[n][k] * powl(c3, k) * Inv1[k - 1];
        s3 += pi[n] * pmb[n][k] * powl(c3, k) * Inv1[k - 1];
        s2 += pi[n] * qm[n][k] * powl(c4, k) * Inv2[k - 1];
        s4 += pi[n] * qmb[n][k] * powl(c4, k) * Inv2[k - 1];
    }
}

```

```

        s3 += pi[n] * pmb[n][n + 1] * powl(c3, n + 1) * Inv1[n];
    }
    if (c > 1e+100)
    {
        c = Hh0((a - nu * T) / (sigma * sqrtl(T))) / (sqrtl(2 * M_PI));
    }
    else
        c *= I0(a - nu * T, -eta1, -1. / (sigma * sqrt(T)), -eta1 * sigma * sqrt(T))
    if (c1 > 1e+100)
    {
        c1 = 1.;
        s1 = (1 - pi[0]) * p * Hh0((a - nu * T) / (sigma * sqrtl(T))) / sqrtl(2 *
        s3 = s1;
    }
    if (c2 > 1e+100)
    {
        c2 = 1.;
        s2 = (1 - pi[0]) * (1 - p) * Hh0((a - nu * T) / (sigma * sqrtl(T))) / sqrt
        s4 = s2;
    }
    y[0] = c1 * s1 + c2 * s2 + pi[0] * Hh0((a - nu * T) / (sigma * sqrtl(T))) / sq
    y[1] = c1 * s3 + c2 * s4 + pi[0] * c;

    return y;
}
//psi=P[ZT>=a,maxZs>=b s dans[0 T]] ou Z levy poisson compose long double expone
long double psiB(long double *x, const long double &T)
{
    long double sm = 0.0, pas = 0.01, s = 0.0, y[8], a, b;
    int n, N;
    std::vector<long double> z;
    z.resize(2);
    N = (int)(T / pas);
    if (x[7] <= 0)
    {
        cout << "Fonction psiB : parametres non valides" << endl;
        exit(0);
    }
    if (T - N * pas < 1e-16)
        N -= 1;
    for (n = 0; n < 8; n++)

```

```

    y[n] = x[n];
    y[6] = x[6] - x[7];
    for (n = 1; n <= N; n++)
    {
        s = n * pas;
        y[7] = T - s;
        z = psiVNB(y);
        a = psiMA(x, s);
        b = psiMB(x, s);
        sm += (a * z[0] + b * z[1]) * pas;
    }
    return sm;
};
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//prix du put loockback floating strike
long double PLB(long double *x, const long double &T)
{
    LPLB LF(x);
    inverselaplace F(LF);
    return F.InvLF(T);
};
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//prix du call loockback floating strike
long double CLB(long double *x, const long double &T)
{
    LCLB LF(x);
    inverselaplace F(LF);
    return F.InvLF(T);
};
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//delta du put loockback floating strike
long double dPLB(long double *x, const long double &T)
{
    DLPLB LF(x);
    inverselaplace F(LF);
    return F.InvLF(T);
};
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//delta du call loockback floating strike
long double dCLB(long double *x, const long double &T)
{

```

```

    DLCLB LF(x);
    inverselaplace F(LF);
    return F.InvLF(T);
};
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//psi=P[maxZs>=b s dans[0 T]] ou Z levy poisson compose long double exponentiell
long double psiM(long double *x, const long double &T)
{
    LPsiM LF(x);
    inverselaplace F(LF);
    return F.InvLF(T);
};
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
long double dpsim(long double *x, const long double &T)
{
    LdPsiM LF(x);
    inverselaplace F(LF);
    return F.InvLF(T);
};
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
long double rebateproba(long double *x, const long double &r, const long double
{
    long double sm = 0.0, pas = 0.01, s = 0.0;
    int N;
    N = (int)(T / pas);
    if (T - N * pas < 1e-16)
        N -= 1;
    for (int n = 1; n <= N; n++)
    {
        s = n * pas;
        sm += exp(-r * s) * dpsim(x, s) * pas;
    }
    return sm;
};
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//psi=P[maxZs>=b s dans[0 T]] ou Z levy poisson compose long double exponentiell
long double psiMA(long double *x, const long double &T)
{
    LPsiMA LF(x);
    inverselaplace F(LF);

```

[illegible]