

Help

```
/* Céline Labart and Jérôme Lelong
 * First version : August 2004
 * Last modified : July 2006
 * Computation of the prices of Parisian options using
 * Laplace transforms. This implementation is based on a
 * Research Report available on cermics website
 *
 * http://cermics.enpc.fr/reports/CERMICS-2005/CERMICS-2005-294.pdf
 */

extern "C" {
#include "
href../../mod/bs1d/bs1d_lim/bs1d_lim_h_src.pdfbs1d_lim.h"
}

#include <cmath>
#include <
href../../common/math/complex_erf_h_src.pdfcomplex>

using namespace std;

typedef complex<double> complex_double;

typedef struct
{
    double K;
    double T;
    double t;
    double D;
    double d;
    double L;
    double sigma;
    double r;
    double delta;
    double So;
} parisian_t;

/* compute Laplace Transform of the price with respect to
```

```

* maturity time. l is the laplace parameter
* p : Parisian
* d or u: down or out
* i or o: in or out
* c or p: call or put
*/
static complex_double pdic(complex_double l,    parisian_t *opt);
static complex_double pdoc(complex_double l,    parisian_t *opt);
static complex_double puic(complex_double l,    parisian_t *opt);
static complex_double puoc(complex_double l,    parisian_t *opt);

/* Laplace transform of the price of the call with respect
* to maturity time */
static complex_double bs(complex_double l,    parisian_t *opt);

/* defined in Src/common/complex_erf.c */
extern complex_double normal_cerf(const complex_double z);

static parisian_t *NewParisian_t(parisian_t *orig)
{
    parisian_t *opt = new parisian_t;
    opt->K = orig->K;
    opt->T = orig->T;
    opt->t = orig->t;
    opt->D = orig->D;
    opt->d = orig->d;
    opt->L = orig->L;
    opt->sigma = orig->sigma;
    opt->r = orig->r;
    opt->So = orig->So;
    opt->delta = orig->delta;
    return opt;
}

static complex_double psi(complex_double z)
{
    complex_double res;
    double racine = sqrt(2.0 * M_PI);
    res = 1.0 + z * racine * exp(pow(z, 2) / 2.0) * normal_cerf(z);
}

```

```

    return (res);
}

/* Laplace transform of the price of the call with respect
 * to maturity time */
static complex_double bs(complex_double l, parisian_t *opt)
{
    complex_double theta;
    double m, k;
    m = (opt->r - opt->delta - pow(opt->sigma, 2) / 2.0) / opt->sigma;
    k = log(opt->K / opt->So) / opt->sigma;
    theta = sqrt(2.0 * l);

    /* K < X */
    if (opt->K <= opt->So)
        return (2.0 * opt->K / (m * m - 2.0 * l) - 2.0 * opt->So / (pow(m + opt->sigma, 2) - 2.0 * l));
    /* K > x */
    if (opt->K > opt->So)
        return (opt->K * exp((m - theta) * k) / theta * (1.0 / (m - theta) - 1.0 / (m + theta)));
    return -1;
}

static complex_double pdic(complex_double l, parisian_t *opt)
{
    complex_double theta;
    double m;
    double b;
    double k;
    double d;
    double d3;
    double racine = sqrt(2.0 * M_PI);
    m = 1.0 / opt->sigma * (opt->r - opt->delta - pow(opt->sigma, 2) / 2.0);
    b = 1.0 / opt->sigma * log(opt->L / opt->So);
    k = 1.0 / opt->sigma * log(opt->K / opt->So);
    theta = sqrt(2.0 * l);
    d = sqrt(opt->D);
    d3 = (b - k) / d;
    /* K < L < x */
    if (opt->D > opt->T) return 0.0;
    if (opt->K <= opt->L && opt->L <= opt->So)

```

```

    {
        return (exp((m + theta) * b) / psi(theta * d) * (2.0 * opt->K / (m * m - 2.0 * theta * theta)));
    }
    /*x<L<K*/
    if (opt->So <= opt->L && opt->L <= opt->K)
    {
        return (opt->K / theta * exp((m - theta) * k) * (1.0 / (m - theta) - 1.0 / (m + theta)));
    }
    /*L<x et L<K*/
    if (opt->L <= opt->So && opt->L <= opt->K)
    {
        return (psi(-theta * d) / psi(theta * d) * opt->K / theta * exp(2.0 * b * theta * d));
    }
    /*x<K<L*/
    if (opt->So <= opt->K && opt->K <= opt->L)
    {
        return (opt->K / theta * exp((m - theta) * k) * (1.0 / (m - theta) - 1.0 / (m + theta)));
    }
    /*K<x<L*/
    if (opt->K <= opt->So && opt->So <= opt->L)
    {
        return (2.0 * opt->K / (m * m - 2.0 * 1) - 2.0 * opt->So / (pow(m + opt->sigma, 2)));
    }

    return -1;
}

```

```

static complex_double pdoc(complex_double l, parisian_t *opt)
{
    complex_double theta;
    double b, m, k, d;
    double racine = sqrt(2.0 * M_PI);

    m = (opt->r - opt->delta - pow(opt->sigma, 2) / 2.0) / opt->sigma;
    b = log(opt->L / opt->So) / opt->sigma;
    k = log(opt->K / opt->So) / opt->sigma;
    d = sqrt(opt->D);
    theta = sqrt(2.0 * 1);
}

```

```

/* L < K < X */
if (opt->D > opt->T) return (bs(l, opt));
if (opt->L <= opt->K && opt->K <= opt->So)
    return (2.0 * opt->K / (m * m - 2.0 * l) - 2.0 * opt->So / (pow(m + opt->sig

/* L < x < K */
if (opt->L <= opt->So && opt->So <= opt->K)
    return ((1.0 - exp(2.0 * b * theta) + (theta * exp(2.0 * b * theta) * racine

/* x < L < K */
if (opt->So <= opt->L && opt->L <= opt->K)
    return (opt->L * (exp((m - theta) * b) * normal_cerf(theta * d - b / d) + ex

/* K < L < x */
if (opt->K <= opt->L && opt->L <= opt->So)

    return (2.0 * opt->K / (m * m - 2.0 * l) * (1.0 - exp((m + theta) * b) / psi

/* K<L and x<L */
if (opt->K <= opt->L && opt->So <= opt->L)
    return (opt->L * (exp((m - theta) * b) * normal_cerf(theta * d - b / d) + ex

return -1;
}

static complex_double puoc(complex_double l, parisian_t *opt)
{
    complex_double theta;
    double b, m, d;
    m = (opt->r - opt->delta - opt->sigma * opt->sigma / 2.0) / opt->sigma;
    b = log(opt->L / opt->So) / opt->sigma;
    theta = sqrt(2.0 * l);
    d = sqrt(opt->D);
    if (opt->D > opt->T)
        return bs(l, opt);
    else if (opt->L >= opt->So)
        return bs(l, opt) - puic(l, opt);
    else
    {
        parisian_t opt_0;

```

```

    opt_0 = *opt;
    opt_0.K = opt->K / opt->L;
    opt_0.So = 1.0;
    opt_0.L = 1.0;

    return opt->L * (exp((m + theta) * b) * normal_cerf(theta * d + b / d) + e
}
}

static complex_double puic(complex_double l, parisian_t *opt)
{
    complex_double theta;
    double b, m, k, d, d3;
    double racine = sqrt(2.0 * M_PI);
    m = (opt->r - opt->delta - pow(opt->sigma, 2) / 2.0) / opt->sigma;
    b = log(opt->L / opt->So) / opt->sigma;
    k = log(opt->K / opt->So) / opt->sigma;
    d = sqrt(opt->D);
    theta = sqrt(2.0 * l);
    d3 = (b - k) / d;
    if (opt->D > opt->T) return 0.0;
    /* X < L < K*/
    if (opt->So <= opt->L && opt->L <= opt->K)
    {
        return (exp((m - theta) * b) * racine * d / psi(theta * d) * (2.*opt->K /
    }
    /* x>L */
    if (opt->So > opt->L)
        return (bs(l, opt) - puoc(l, opt));

    /* K<L and x<L*/
    if (opt->K <= opt->L && opt->So <= opt->L)
        return (exp((m - theta) * b) / psi(theta * d) * (2.0 * opt->K / (m * m - 2.0

    return -1;
}

/* computes the Laplace Transforms of the price of single
 * barrier Parisian options using pdic,
 * pdoc, puic, puoc. Put prices are computed using parity

```

```

    * relationships */
static complex_double Ltransform(complex_double l, int choice, parisian_t *opt)
{
    switch (choice)
    {
        case 1:
            return pdic(l, opt);
            break;
        case 2:
            return pdoc(l, opt);
            break;
        case 3:
            return puic(l, opt);
            break;
        case 4:
            return puoc(l, opt);
            break;
        case 9:
            return bs(l, opt);
            break;
    }
    return WRONG;
}

```

```

/* compute the numerical inversion of Laplace transforms
 * using Euler summation */
static double euler(int choice, parisian_t *opt, int N, int M)
{
    int i, Cnp;
    double sum, a, pit, run_sum, m;
    /* int N=15 ;
     * int M=15; */
    double A = 13.8;
    complex_double I = complex_double(0.0, 1.0);

    a = A / (2.0 * opt->T);
    pit = M_PI / opt->T;
    sum = exp(A / 2.0) * 0.5 * (Ltransform(a, choice, opt)).real();

    for (i = 1; i < N + 1; i++)

```

```

    sum = sum + exp(A / 2.0) * PNL_ALTERNATE(i) * (Ltransform(a + pit * i * I, c
run_sum = sum; /* partial sum of sn */
sum = 0.0; /* partial exponential sum */
Cnp = 1; /* binomial coefficients */

for (i = 0; i < M + 1; i++)
{
    sum = sum + run_sum * (double) Cnp ;
    run_sum = run_sum + exp(A / 2.0) * PNL_ALTERNATE(i + N + 1) *
        (Ltransform(a + pit * (i + N + 1) * I, choice, opt)).real();
    Cnp = (Cnp * (M - i)) / (i + 1);
}
m = (opt->r - opt->delta - opt->sigma * opt->sigma / 2.0) / opt->sigma;
return (exp(-(opt->r + m * m / 2.0) * opt->T) * sum / opt->T / pow(2.0, M));
}

```

```

/* Parameter choice defines the option type as follows

```

```

* 1. Parisian Down and In Call
* 2. Parisian Down and Out Call
* 3. Parisian Up and In Call
* 4. Parisian Up and Out Call
* 5. Parisian Down and In Put
* 6. Parisian Down and Out Put
* 7. Parisian Up and In Put
* 8. Parisian Up and Out Put
*/

```

```

/* Computes the price of the corresponding single barrier

```

```

* Parisian option using Laplace inversion*/

```

```

static double SingleParisian(int choice, parisian_t *opt, int N, int M)
{

```

```

    parisian_t *new_opt = NewParisian_t(opt);
    double res;

```

```

    switch (choice)
    {

```

```

        case 1:
            new_opt->T = opt->T - opt->t;

```



```

    res = euler(choice, new_opt, N, M);
    break;
case 2:
    new_opt->T = opt->T - opt->t;
    res = euler(choice, new_opt, N, M);
    break;
case 3:
    new_opt->T = opt->T - opt->t;
    res = euler(choice, new_opt, N, M);
    break;
case 4:
    new_opt->T = opt->T - opt->t;
    res = euler(choice, new_opt, N, M);
    break;
case 5:
{
    new_opt->So = 1.0 / opt->So;
    new_opt->L = 1.0 / opt->L;
    new_opt->K = 1.0 / opt->K;
    new_opt->r = opt->delta;
    new_opt->delta = opt->r;
    res = opt->K * opt->So * SingleParisian(3, new_opt, N, M);
}
break;
case 6:
{
    new_opt->So = 1.0 / opt->So;
    new_opt->L = 1.0 / opt->L;
    new_opt->K = 1.0 / opt->K;
    new_opt->r = opt->delta;
    new_opt->delta = opt->r;
    res = opt->K * opt->So * SingleParisian(4, new_opt, N, M);
}
break;
case 7:
{
    new_opt->So = 1.0 / opt->So;
    new_opt->L = 1.0 / opt->L;
    new_opt->K = 1.0 / opt->K;
    new_opt->r = opt->delta;
    new_opt->delta = opt->r;

```

```

    res = opt->K * opt->So * SingleParisian(1, new_opt, N, M);
}
break;
case 8:
{
    new_opt->So = 1.0 / opt->So;
    new_opt->L = 1.0 / opt->L;
    new_opt->K = 1.0 / opt->K;
    new_opt->r = opt->delta;
    new_opt->delta = opt->r;
    res = opt->K * opt->So * SingleParisian(2, new_opt, N, M);
}
break;
case 9 :
    new_opt->T = opt->T - opt->t;
    res = euler(9, new_opt, N, M);
    break;
case 10 :
{
    new_opt->So = 1.0 / opt->So;
    new_opt->L = 1.0 / opt->L;
    new_opt->K = 1.0 / opt->K;
    new_opt->r = opt->delta;
    new_opt->delta = opt->r;
    res = opt->K * opt->So * SingleParisian(9, new_opt, N, M);
}
break;
default:
{
    printf("wrong choice in SingleParisian\ n");
    abort();
}
}

delete new_opt;
return (res);
}

```

```

static int LaplaceParisian(int utorin, int upordown, double s, NumFunc_1 *p, do
{

    int choice;
    parisian_t *opt = new parisian_t;
    opt->T = t;
    opt->t = 0.0;
    opt->D = delay;
    opt->r = r;
    opt->sigma = sigma;
    opt->delta = divid;
    opt->So = s;
    opt->L = l;
    opt->d = 0.0;
    opt->K = p->Par[0].Val.V_DOUBLE;

    if ((p->Compute) == &Put)
    {
        /* puop */
        if (utorin && upordown)
            choice = 8;
        /* pdop */
        else if (utorin && !upordown)
            choice = 6;
        /* puip */
        else if (!utorin && upordown)
            choice = 7;
        /* pdip */
        else /* (!utorin && !upordown) */
            choice = 5;
    }
    else
    {
        /* puoc */
        if (utorin && upordown)
            choice = 4;
        /* pdoc */
        else if (utorin && !upordown)
            choice = 2;
    }
}

```

```

        /* puic */
        else if (!outorin && upordown)
            choice = 3;
        /* pdic */
        else /*if(!outorin && !upordown)*/
            choice = 1;
    }
    /*Price*/
    *ptprice = SingleParisian(choice, opt, N, M);

    /*Delta*/
    opt->So = opt->So * (1.0 + inc);
    *ptdelta = (SingleParisian(choice, opt, N, M) - *ptprice) / (s * inc);

    delete opt;
    return OK;
}

extern "C" {

int CALC(AP_LaplaceParisian)(void *Opt, void *Mod, PricingMethod *Met)
{
    TYPEOPT *ptOpt = (TYPEOPT *)Opt;
    TYPEMOD *ptMod = (TYPEMOD *)Mod;
    double r, divid, limit;
    int upordown, outorin;

    r = log(1. + ptMod->R.Val.V_DOUBLE / 100.);
    divid = log(1. + ptMod->Divid.Val.V_DOUBLE / 100.);
    limit = ((ptOpt->Limit.Val.V_NUMFUNC_1)->Compute)((ptOpt->Limit.Val.V_NUMFUN

    if ((ptOpt->DownOrUp).Val.V_BOOL == DOWN)
        upordown = 0;
    else upordown = 1;

    if ((ptOpt->OutOrIn).Val.V_BOOL == OUT)
        outorin = 1;
    else outorin = 0;

    return LaplaceParisian(outorin, upordown, ptMod->S0.Val.V_PDOUBLE,
        ptOpt->PayOff.Val.V_NUMFUNC_1,

```

```

        limit,
        ptOpt->Maturity.Val.V_DATE - ptMod->T.Val.V_DATE,
        (ptOpt->Limit.Val.V_NUMFUNC_1)->Par[4].Val.V_PDOUBLE,
        r,
        divid,
        ptMod->Sigma.Val.V_PDOUBLE, Met->Par[0].Val.V_DOUBLE,
        Met->Par[1].Val.V_PINT,
        Met->Par[2].Val.V_PINT,
        &(Met->Res[0].Val.V_DOUBLE),
        &(Met->Res[1].Val.V_DOUBLE)
    );
}

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

    if ((opt->RebOrNo).Val.V_BOOL == NOREBATE)
        if ((opt->EuOrAm).Val.V_BOOL == EURO)
            if ((opt->Parisian).Val.V_BOOL == TRUE)
                return OK;

    return WRONG;
}

static int MET(Init)(PricingMethod *Met, Option *Opt)
{
    static int first = 1;

    if (first)
    {
        Met->Par[0].Val.V_PDOUBLE = 0.01;
        Met->Par[1].Val.V_PINT = 15;
        Met->Par[2].Val.V_PINT = 15;
        first = 0;
    }

    return OK;
}

```

```

PricingMethod MET(AP_LaplaceParisian) =
{
    "AP_Laplace_Parisian",
    { {"Delta Increment Rel", PDOUBLE, {100}, ALLOW},
      {"sum truncation", PINT, {15}, ALLOW},
      {"window average", PINT, {15}, ALLOW},
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CALC(AP_LaplaceParisian),
    { {"Price", DOUBLE, {100}, FORBID},
      {"Delta", DOUBLE, {100}, FORBID} ,
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CHK_OPT(AP_LaplaceParisian),
    CHK_ok,
    MET(Init)
} ;

}

```