Help

```
#include <stdlib.h>
#include  "bs1d_limdisc.h"
#include "pnl/pnl_mathtools.h"

#define ACTPADE 1
/*--------------------------------------------------------------------*/

#define MAXIT 10
#if 0
static void gaulag(double x[], double w[], int n, double alf)
{
  void nrerror(char error_text[]);
  int i, its, j;
  double ai;
  double p1, p2, p3, pp, z, z1;

  for (i = 1; i <= n; i++)
    {
      if (i == 1)
        {
          z = (1.0 + alf) * (3.0 + 0.92 * alf) / (1.0 + 2.4 * n + 1.8 * alf);
        }
      else if (i == 2)
        {
          z += (15.0 + 6.25 * alf) / (1.0 + 0.9 * alf + 2.5 * n);
        }
      else
        {
          ai = i - 2;
          z += ((1.0 + 2.55 * ai) / (1.9 * ai) + 1.26 * ai * alf /
                (1.0 + 3.5 * ai)) * (z - x[i - 2]) / (1.0 + 0.3 * alf);
        }
      for (its = 1; its <= MAXIT; its++)
        {
          p1 = 1.0;
          p2 = 0.0;
          for (j = 1; j <= n; j++)
            {
              p3 = p2;
```

```
              p2 = p1;
              p1 = ((2 * j - 1 + alf - z) * p2 - (j - 1 + alf) * p3) / j;

            }
          pp = (n * p1 - (n + alf) * p2) / z;
          z1 = z;
          z = z1 - p1 / pp;
          if (fabs(z - z1) <= EPS) break;
        }
      if (its > MAXIT) printf("too many iterations in gaulag");
      x[i] = z;
      w[i] = -exp(pnl_lgamma(alf + n) - pnl_lgamma((double)n)) / (pp * n * p2);
    }
}
#endif

#undef EPS
#undef MAXIT
/*--------------------------------------------------------------------*/
#define EPS 3.0e-11
#undef EPS
/*--------------------------------------------------------------------*/
static dcomplex mu(int m, dcomplex q)
{

  dcomplex  mum, logq, term, root;
  double pg = 3.14159265358979358;
  double imroot;

  logq = Clog(q); /*logq*/
  term = Complex(0.0, 2 * pg * m); /*2 pg m I*/
  root = Csqrt(Cadd(logq, term));

  imroot = Cimag(root); /*IMM(root)*/

  if (imroot > 0) mum = root;
  else
    /*if(imroot <=0 )*/
    mum = RCmul(-1, root);

  return mum;
```

```
}

/*-------------------------------------------------------------------*/
static double sign(double x)
{
  double segno;

  if (x >= 0.0) segno = 1.0;
  else
    {
      segno = -1.0;
    }

  return segno;
}


dcomplex L(dcomplex u, dcomplex q)
{

  dcomplex u2 = Cmul(u, u);
  dcomplex eu2 = Cexp(RCmul(-1.0, u2));
  return Csub(Complex(1.0, 0.0), Cmul(q, eu2));
}

/*-------------------------------------------------------------------*/
/*I find the first term in the sum in the paper*/
static dcomplex term1(double z, double k, double l, double alpha, double gamma,
{
  /*attenzione se sign =-1 ottengo il termine per calcolare reale2,
    se sign = 1 ottengo il termine per calcolare reale3*/

  int n;
  /* double pg =3.14159265358979358;*/

  dcomplex cOne = Complex(0.0, 1.0);
  dcomplex num;
  dcomplex den1, den2, den3, den;

  dcomplex term;
```

```
  /**starting value for the sum when k=0***/
  dcomplex sum = Complex(0.0, 0.0);


  /***computation of the sum****/
  for (n = -nmax; n <= nmax; n++)
    {

      /***the numerator****/
      if (z > k) num = Cexp(Cmul(cOne, RCmul((z - k) / gamma, mu(n, q))));
      if (z == k) num = Complex(1.0, 0.0);
      if (z < k) num = Cexp(Cmul(cOne, RCmul((k - z) / gamma, mu(n, q))));

      /***the denominator****/
      den1 = mu(n, q);
      den2 = Csub(den1, RCmul(alpha * gamma * sign(z - k), cOne));
      den3 = Csub(den1, Complex(0.0, (alpha - 1) * gamma * sign(z - k)));
      den  = Cmul(Cmul(den1, den2), den3);

      /***the ratio num/den****/
      term = Cdiv(num, den);

      sum = Cadd(term, sum);
    }

  sum = Cmul(sum, Complex(0.0, -1 * gamma * exp(k * (1 - alpha)) / 2.0));

  return sum;
}


/*compute the argument of the integral defining the function L+*/
static dcomplex argLplus(double z, dcomplex u, dcomplex q)
{
  dcomplex num;
  dcomplex den;


  num = Clog(Csub(Complex(1.0, 0.0), RCmul(exp(-z * z), q)));
  den = Csub(Complex(z * z, 0.0), Cmul(u, u));
  return Cdiv(num, den);
```

```
}


static dcomplex Lplus(dcomplex u, dcomplex q, int npoints, double zmax)
{

  int i;
  double sumr, sumi, *z, *w;
  double pg = 3.14159265358979358;
  dcomplex result, alplus;

  sumr = 0.0;
  sumi = 0.0;

  /*Memory Allocation*/
  z = malloc((npoints + 2) * sizeof(double));
  w = malloc((npoints + 2) * sizeof(double));


  /* Integration using gauss-legendre*/
  gauleg(0, zmax, z, w, npoints);

  for (i = 1; i <= npoints; i++)
    {
      alplus = argLplus(z[i], u, q);
      sumr += (w[i] * alplus.r);
      sumi += (w[i] * alplus.i);
    }

  /* Integration using gauss-laguerre */
  /* double alf=1.0;
    gaulag(z, w, npoints, alf);
    for (i=1;i<=npoints;i++) {
    alplus = RCmul(exp(-z[i]),argLplus(z[i], u, q));
    sumr += (w[i]*alplus.r);
    sumi += (w[i]*alplus.i);
    }
  */
  result = Complex(sumr, sumi);
  result = Cexp(Cdiv(Cmul(u, result), Complex(0.0, pg)));
```

```
  /*Memory Desallocation*/
  free(w);
  free(z);

  return result;
}

#if 0
static dcomplex Lminus(dcomplex u, dcomplex q, int npoints, double zmax)
{
  return Lplus(RCmul(-1.0, u), q, npoints, zmax);
}
#endif


static dcomplex term3(double z, double k, double l, double alpha,
                      double gamma, dcomplex q, int nmax, int mmax, int npoints,
{
  /* double pg =3.14159265358979358; */
  int in, ii, im, min_nm, max_nm, indice;

  double  *lplus_r, *lplus_i;

  dcomplex lplus_c;

  dcomplex num1, num2, num, den, term;
  dcomplex den1, den2, den3, den4;

  dcomplex summ;
  dcomplex sumn = Complex(0.0, 0.0);

  /*find the number of times we need to compute the function Lplus*/
  min_nm = -nmax;
  if (-mmax < -nmax) min_nm = -mmax;
  max_nm = -min_nm;

  indice = 2 * max_nm + 1;

  /*allocate the vector where to store the function Lplus*/
  /*Memory Allocation*/
  lplus_r = malloc((indice + 1) * sizeof(double));
```

```
lplus_i = malloc((indice + 1) * sizeof(double));

/*compute the values of Lplus*/
for (ii = min_nm; ii <= max_nm; ii++)
  {
    lplus_c = Lplus(mu(ii, q), q, npoints, zmax);
    lplus_r[ii + max_nm + 1] = lplus_c.r;
    lplus_i[ii + max_nm + 1] = lplus_c.i;
  }

for (in = -nmax; in <= nmax; in++)
  {
    num = Cexp(Cmul(Complex(0.0, z / gamma), mu(in, q)));
    den = mu(in, q);
    term = Cmul(Complex(lplus_r[in + max_nm + 1], lplus_i[in + max_nm + 1]), C

    summ = Complex(0.0, 0.0);

    for (im = -mmax; im <= mmax; im++)
      {
        num1 = Complex(lplus_r[im + max_nm + 1], lplus_i[im + max_nm + 1]);
        num2 = Cexp(Cmul(Complex(0.0, k / gamma), mu(im, q)));
        num = Cmul(num1, num2);

        den1 = mu(im, q);
        den2 = Cadd(mu(im, q), Complex(0.0, alpha * gamma));
        den3 = Cadd(mu(im, q), Complex(0.0, (alpha - 1) * gamma));
        den4 = Cadd(mu(im, q), mu(in, q));
        den  = Cmul(Cmul(Cmul(den1, den2), den3), den4);

        summ = Cadd(summ, Cdiv(num, den));
      }

    sumn = Cadd(sumn, Cmul(term, summ));
  }

/*Memory Desallocation*/
free(lplus_r);
free(lplus_i);

return Cmul(Complex(0.0, -1 * gamma * exp((1 - alpha) * k) / 4.0), sumn);
```

```
}

static dcomplex ztransform(double z, double k, double l, double alpha,
                           double gamma, dcomplex q, int n1max, int n3max, int m
{
  dcomplex sum1, sum2;

  sum1 = term1(z, k, l, alpha, gamma, q, n1max);

  sum2 = term3(z, k, l, alpha, gamma, q, n3max, m3max, npoints, zmax);

  return Cadd(sum1, sum2);
}


static double InverseZT(double z, double k, double l, double alpha, double gamma
{
  dcomplex q;
  int j;
  double pg = 3.14159265358979358;
  double sum;
  double accuracy = 8.0;
  double rpar = POW(10.0, -accuracy / (2.0 * ndates));
  double termAW, term1AW, term2AW;


  term1AW = Creal(ztransform(z, k, l, alpha, gamma, Complex(rpar, 0.0), n1max, n


  term2AW = Creal(ztransform(z, k, l, alpha, gamma, Complex(-rpar, 0.0), n1max,

  sum = 0.0;

  for (j = 1; j <= ndates - 1; j++)
    {
      q = RCmul(rpar, Cexp(Complex(0.0, pg * j / ndates)));
      termAW = Creal(ztransform(z, k, l, alpha, gamma, q, n1max, n3max, m3max, n
      sum = sum + POW(-1, j) * termAW;
    }
  return (term1AW + term2AW * POW(-1.0, ndates) + 2.0 * sum) / (2.0 * ndates * P
```

```
}



/***BEGINS CODE FOR COMPUTING THE DELTA***/
/*The first term in the sum for the delta*/
static dcomplex deltaterm1(double z, double k, double l, double alpha, double ga
{

  int n;
  /* double pg =3.14159265358979358;*/

  dcomplex cOne = Complex(0.0, 1.0);
  dcomplex num;
  dcomplex den1, den2, den3, denterm1, denterm2;

  dcomplex term1, term2, term;

  /**starting value for the sum when k=0***/
  dcomplex sum = Complex(0.0, 0.0);

  /***computation of the sum****/
  for (n = -nmax; n <= nmax; n++)
    {

      /***the numerator****/
      if (z > k) num = Cexp(Cmul(cOne, RCmul((z - k) / gamma, mu(n, q))));
      if (z == k) num = Complex(1.0, 0.0);
      if (z < k) num = Cexp(Cmul(cOne, RCmul((k - z) / gamma, mu(n, q))));

      /***the denominator****/
      den1 = mu(n, q);
      den2 = Csub(den1, RCmul(alpha * gamma * sign(z - k), cOne));
      den3 = Csub(den1, Complex(0.0, (alpha - 1) * gamma * sign(z - k)));
      denterm1  = Cmul(Cmul(den1, den2), den3);
      denterm2  = Cmul(den2, den3);

      /***the ratio num/den****/
      term1 = RCmul(alpha, Cdiv(num, denterm1));
```

```
      term2 = Cmul(Complex(0.0, 1.0 * sign(z - k) / gamma), Cdiv(num, denterm2))
      term  = Cadd(term1, term2);

      sum = Cadd(term, sum);
    }

  sum = Cmul(sum, Complex(0.0, -l * gamma * exp(k * (1 - alpha)) / 2.0));

  return sum;
}


static dcomplex deltaterm3(double z, double k, double l, double alpha,
                           double gamma, dcomplex q, int nmax, int mmax, int npo
{
  /* double pg =3.14159265358979358;*/
  int in, ii, im, min_nm, max_nm, indice;

  double  *lplus_r, *lplus_i;

  dcomplex lplus_c;

  dcomplex num1, num2, num, den, term, term1, term2, term3;
  dcomplex den1, den2, den3, den4;

  dcomplex summ;
  dcomplex sumn = Complex(0.0, 0.0);

  /*find the number of times we need to compute the function Lplus*/
  min_nm = -nmax;
  if (-mmax < -nmax) min_nm = -mmax;
  max_nm = -min_nm;

  indice = 2 * max_nm + 1;

  /*allocate the vector where to store the function Lplus*/
  /*Memory Allocation*/
  lplus_r = malloc((indice + 1) * sizeof(double));
  lplus_i = malloc((indice + 1) * sizeof(double));

  /*compute the values of Lplus*/
```

```
for (ii = min_nm; ii <= max_nm; ii++)
  {
    lplus_c = Lplus(mu(ii, q), q, npoints, zmax);
    lplus_r[ii + max_nm + 1] = lplus_c.r;
    lplus_i[ii + max_nm + 1] = lplus_c.i;
  }

for (in = -nmax; in <= nmax; in++)
  {
    num = Cexp(Cmul(Complex(0.0, z / gamma), mu(in, q)));
    den = mu(in, q);

    term1 = Cmul(Complex(lplus_r[in + max_nm + 1], lplus_i[in + max_nm + 1]),
    term2 = Cdiv(term1, den);
    term3 = Cmul(Complex(0.0, 1.0 / gamma), term1);
    term  = Cadd(term3, RCmul(alpha, term2));

    summ = Complex(0.0, 0.0);

    for (im = -mmax; im <= mmax; im++)
      {
        num1 = Complex(lplus_r[im + max_nm + 1], lplus_i[im + max_nm + 1]);
        num2 = Cexp(Cmul(Complex(0.0, k / gamma), mu(im, q)));
        num = Cmul(num1, num2);

        den1 = mu(im, q);
        den2 = Cadd(mu(im, q), Complex(0.0, alpha * gamma));
        den3 = Cadd(mu(im, q), Complex(0.0, (alpha - 1) * gamma));
        den4 = Cadd(mu(im, q), mu(in, q));
        den  = Cmul(Cmul(Cmul(den1, den2), den3), den4);

        summ = Cadd(summ, Cdiv(num, den));
      }

    sumn = Cadd(sumn, Cmul(term, summ));
  }

free(lplus_r);
free(lplus_i);

return Cmul(Complex(0.0, -1 * gamma * exp((1 - alpha) * k) / 4.0), sumn);
```

```
}

static dcomplex deltaztransform(double z, double k, double l, double alpha,
                                double gamma, dcomplex q, int n1max, int n3max,
{
  dcomplex sum1, sum2;

  sum1 = deltaterm1(z, k, l, alpha, gamma, q, n1max);
  sum2 = deltaterm3(z, k, l, alpha, gamma, q, n3max, m3max, npoints, zmax);

  return Cadd(sum1, sum2);
}

static double deltaInverseZT(double z, double k, double l, double alpha,
                             double gamma, double ndates, int n1max, int n3max,
{
  dcomplex q;
  int j;
  double pg = 3.14159265358979358;
  double sum;
  double accuracy = 8.0;
  double rpar = POW(10.0, -accuracy / (2.0 * ndates));
  double termAW, term1AW, term2AW;

  term1AW = Creal(deltaztransform(z, k, l, alpha, gamma, Complex(rpar, 0.0), n1m
  term2AW = Creal(deltaztransform(z, k, l, alpha, gamma, Complex(-rpar, 0.0), n1


  sum = 0.0;

  for (j = 1; j <= ndates - 1; j++)
    {
      q = RCmul(rpar, Cexp(Complex(0.0, pg * j / ndates)));
      termAW = Creal(deltaztransform(z, k, l, alpha, gamma, q, n1max, n3max, m3m
      sum = sum + POW(-1, j) * termAW;

    }

  return (term1AW + term2AW * POW(-1.0, ndates) + 2.0 * sum) / (2.0 * ndates * P
}
```

```
static int Integration_call_down_out_FAS(double matu, double strike, double r, d
{
  double term1, term2, term3, term4, term1af, term1df, dt;
  double z, k, m, alpha, gamma, alpha1barr, beta1barr, db, db_delta, ztiterm2;

  dt = matu / (double)nb_monit;
  z = log(spot / lowbarr);
  k = log(strike / lowbarr);

  m = r - sg * sg / 2.0;

  alpha = -m / (sg * sg);
  gamma = sg * sqrt(dt / 2.0);

  alpha1barr = -m / (sg * sg);
  beta1barr = alpha1barr * m + (alpha1barr * sg) * (alpha1barr * sg) / 2.0 - r;

  /*Price Computation*/
  db = InverseZT(z, k, lowbarr, alpha, gamma, nb_monit, n1max, n3max, m3max, npo


  ztiterm2 = 0.0;

  if (z >= k)
    {
      term1 = exp(z) * exp(nb_monit * gamma * gamma * (alpha - 1) * (alpha - 1))
      term2 = exp(k) * exp(nb_monit * (alpha * gamma) * (alpha * gamma));
      ztiterm2 = lowbarr * exp(-z * alpha) * (term1 - term2);
    }

  db = (db + ztiterm2) * exp(alpha1barr * z + beta1barr * dt * nb_monit);

  /*Delta Computation*/
  db_delta = deltaInverseZT(z, k, lowbarr, alpha, gamma, nb_monit, n1max, n3max,

  ztiterm2 = 0.0;

  if (z >= k)
```

```
    {
      term1 = exp(z) * exp(nb_monit * gamma * gamma * (alpha - 1) * (alpha - 1))
      term2 = exp(k) * exp(nb_monit * (alpha * gamma) * (alpha * gamma));
      term1af = alpha * (term1 - term2);

      term3 = (1 - alpha) * term1;
      term4 = alpha * term2;
      term1df = term3 + term4;

      ztiterm2 = lowbarr * exp(-z * alpha) * (term1af + term1df);
    }

  db_delta = (db_delta + ztiterm2) * exp((alpha1barr - 1) * z + beta1barr * dt *

  /*Price*/
  *pt_price = db;

  /*Delta*/
  *pt_delta = db_delta;

  return OK;
}

int CALC(AP_FusaiAbrahamsSgarra)(void *Opt, void *Mod, PricingMethod *Met)
{
  TYPEOPT *ptOpt = (TYPEOPT *)Opt;
  TYPEMOD *ptMod = (TYPEMOD *)Mod;
  double r, limit;
  int return_value;

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

  if ((ptMod->Divid.Val.V_DOUBLE > 0) || (limit > (ptOpt->PayOff.Val.V_NUMFUNC_1
    {
      Fprintf(TOSCREEN, "Untreated case\ n\ n\ n");
      return_value = WRONG;
    }
  else if ((limit > (ptOpt->PayOff.Val.V_NUMFUNC_1)->Par[0].Val.V_PDOUBLE))
    {
      Fprintf(TOSCREEN, "Untreated case\ n\ n\ n");
```

```
      return_value = WRONG;
    }
  else
    return_value = Integration_call_down_out_FAS(ptOpt->Maturity.Val.V_DATE - pt
                   Met->Par[0].Val.V_INT,
                   Met->Par[1].Val.V_INT,
                   Met->Par[2].Val.V_INT,
                   Met->Par[3].Val.V_INT,
                   Met->Par[4].Val.V_PDOUBLE,
                   &(Met->Res[0].Val.V_DOUBLE), &(Met->Res[1].Val.V_DOUBLE));

  return return_value;

}

static int CHK_OPT(AP_FusaiAbrahamsSgarra)(void *Opt, void *Mod)
{
  return strcmp(((Option *)Opt)->Name, "CallDownOutDiscEuro");
}

static int MET(Init)(PricingMethod *Met, Option *Opt)
{
  if (Met->init == 0)
    {
      Met->init = 1;
      Met->Par[0].Val.V_INT = 15;
      Met->Par[1].Val.V_INT = 15;
      Met->Par[2].Val.V_INT = 15;
      Met->Par[3].Val.V_INT = 50;
      Met->Par[4].Val.V_PDOUBLE = 10.;

    }
  return OK;
}

PricingMethod MET(AP_FusaiAbrahamsSgarra) =
{
  "AP_FusaiAbrahamsSgarra",
  { {"Number of Series Points of First Sum", INT, {100}, ALLOW},
    {"Number of Series Points of Second Sum", INT, {100}, ALLOW},
    {"Number of Series Points of Third Sum", INT, {100}, ALLOW},
```

```
    {"Number of Quadrature Points for LPlus", INT, {100}, ALLOW},
    {"Upper Bound in the Integral for LPlus", DOUBLE, {100}, ALLOW},
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CALC(AP_FusaiAbrahamsSgarra),
  {{"Price", DOUBLE, {100}, FORBID}, {"Delta", DOUBLE, {100}, FORBID} , {" ", PR
  CHK_OPT(AP_FusaiAbrahamsSgarra),
  CHK_ok,
  MET(Init)
} ;
```