Help

```
#include "bs1d_doublim.h"
#include "pnl/pnl_specfun.h"

/*Computation of Laplace transform*/
double fnRf_3(dcomplex z, double aa, double bb, double hh, double nn)
{
  double Rfs;
  dcomplex Cun, Cnn, Cnn2, z1, z2, z3, z4, z5, z6, z7, z8, z9, z10, z11, z12, z1

  Cun = Complex(1.0, 0.0);
  Cnn = Complex(nn, 0.0);
  mu = RCmul(2.0, z);
  Cnn2 = Cmul(Cnn, Cnn);
  mu = Csqrt(Cadd(mu, Cnn2));

  z1 = Complex(cos(bb * mu.i) * sinh(mu.r * bb), sin(bb * mu.i) * cosh(mu.r * bb
  z2 = Complex(cos(aa * mu.i) * sinh(mu.r * aa), sin(aa * mu.i) * cosh(mu.r * aa
  z3 = Complex(cos((aa + bb) * mu.i) * sinh(mu.r * (aa + bb)), sin((aa + bb) * m
  z5 = RCmul(exp(-aa * mu.r), Complex(cos(-mu.i * aa), sin(-mu.i * aa))); /*exp(

  z4 = RCmul(pow(hh, nn + 1.0 - mu.r), Complex(cos(-mu.i * log(hh)), sin(-mu.i *
  z6 = Cmul(mu, Complex(mu.r - nn, mu.i));
  z7 = Cmul(z6, Complex(mu.r - nn - 1.0, mu.i));

  z8 = Cmul(z4, z5);
  z9 = Cmul(z8, z1);
  z10 = Cmul(z7, z3);
  Q_1 = Cdiv(z9, z10);

  z4 = Cmul(mu, mu);
  z5 = Cdiv(Cun, Complex(z4.r - (nn + 1.0) * (nn + 1.0), z4.i));
  z6 = Cdiv(Cun, Complex(z4.r - nn * nn, z4.i));
  z5 = RCmul(exp((nn + 1.0) * bb), z5);
  z6 = RCmul(hh * exp(nn * bb), z6);
  z7 = RCmul(2.0, Csub(z5, z6));


  z8 = RCmul(exp(-bb * mu.r), Complex(cos(-mu.i * bb), sin(-mu.i * bb))); /*exp(
  z9 = RCmul(pow(hh, nn + 1.0 + mu.r), Complex(cos(mu.i * log(hh)), sin(mu.i * l
```

```
  z10 = Cmul(mu, Complex(mu.r + nn, mu.i));
  z11 = Cmul(z10, Complex(mu.r + nn + 1.0, mu.i));
  z12 = Cdiv(Cmul(z8, z9), z11);

  z13 = Cadd(z7, z12);

  Q_2 = Cdiv(Cmul(z2, z13), z3);

  Q = Cadd(Q_1, Q_2);

  Rfs = Q.r;

  return Rfs;
}


static int Out_Laplace(double s, NumFunc_1 *L, NumFunc_1 *Up, NumFunc_1  *Rebate
{
  int N = 15, M = 11;
  int i;
  double price, delta, price2, delta2;
  double xx, y, hh, sum, sum2, Avg, Avg2, Fun, Fun2, j, S[13], Q[13], U, tt, d,
  double St, St2, Lower, Upper, v, pp;
  double sigma2;
  double nu, h, h2, a, a2, b, b2, CTtK;

  /* Inversion Variables*/
  double A;
  dcomplex z;

  /*Inversion parameters*/
  A = 19.1;

  Upper = (Up->Compute)(Up->Par, 0.0);
  Lower = (L->Compute)(L->Par, 0.);
  K = PayOff->Par[0].Val.V_PDOUBLE;
  pp = 1.e-8;
  St = s;
  St2 = s * (1. + pp);
  v = r - divid;
  sigma2 = sigma * sigma;
```

```
nu = (1.0 / sigma2) * (v - 0.5 * sigma2);
h = K / St;
h2 = K / St2;
a = log(St / Lower);
a2 = log(St2 / Lower);
b = log(Upper / St);
b2 = log(Upper / St2);

/* INVERSION  */
tt = t;
xx = A / (2 * tt);
hh = M_PI / tt;
z = Complex(xx / sigma2, 0.0);

sum = fnRf_3(z, a, b, h, nu) * .5 / sigma2;
sum2 = fnRf_3(z, a2, b2, h2, nu) * .5 / sigma2;

/* Computation of S[0]=s(n) which approximate f(t) */
for (i = 1; i <= N; i++)
  {
    y = (double)i * hh;
    z = Complex(xx / sigma2, y / sigma2);
    j = PNL_ALTERNATE(i);
    sum = sum + j * fnRf_3(z, a, b, h, nu) / sigma2;
    sum2 = sum2 + j * fnRf_3(z, a2, b2, h2, nu) / sigma2;
  }

S[0] = sum;
Q[0] = sum2;
/* End of Inversion */

/*  Computation of s(n+p)  p<=M+1 for Euler appromations */
for (i = 1; i <= M + 1; i++)
  {
    y = (double)(N + i) * hh;
    z = Complex(xx / sigma2, y / sigma2);
    j = PNL_ALTERNATE(N + i);
    S[i] = S[i - 1] + j * fnRf_3(z, a, b, h, nu) / sigma2;
    Q[i] = Q[i - 1] + j * fnRf_3(z, a2, b2, h2, nu) / sigma2;
  }
```

```
  /* Computation of Euler appromations */
  Avg = 0.;
  Avg2 = 0.;
  for (i = 1; i <= M + 1; i++)
    {
      Avg = Avg + pnl_sf_choose(M, i - 1) * S[i - 1];
      Avg2 = Avg2 + pnl_sf_choose(M, i - 1) * Q[i - 1];
    }

  d = pow(2.0, (double)M);
  U = exp(A / 2.) / tt;

  /*f(t) values*/
  Fun = U * Avg / d;
  Fun2 = U * Avg2 / d;

  /*Black-Sholes price for call option*/

  pnl_cf_call_bs(1., h, t, r, divid, sigma, &price, &delta);
  pnl_cf_call_bs(1., h2, t, r, divid, sigma, &price2, &delta2);

  CTtK = St * price - St * exp(-r * t) * Fun;

  /*Price*/
  *ptprice = CTtK;

  /*Delta*/
  *ptdelta = (CTtK - (price2 - price) / (h2 - h) * K) / St - exp(-r * t) * (Fun2

  return OK;
}


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

  r = log(1. + ptMod->R.Val.V_DOUBLE / 100.);
```

```
  divid = log(1. + ptMod->Divid.Val.V_DOUBLE / 100.);

  return  Out_Laplace(ptMod->S0.Val.V_PDOUBLE, ptOpt->LowerLimit.Val.V_NUMFUNC_1
}

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

  if ((opt->TwoDoubleStep).Val.V_BOOL == FALSE)
    if ((opt->Parisian).Val.V_BOOL == FALSE)
      if ((opt->RebOrNo).Val.V_BOOL == NOREBATE)
        if ((strcmp(((Option *)Opt)->Name, "DoubleCallOutEuro") == 0))
          return OK;

  return WRONG;
}


static int MET(Init)(PricingMethod *Met, Option *Opt)
{
  return OK;
}

PricingMethod MET(AP_Out_Laplace) =
{
  "AP_Out_Laplace",
  {{" ", PREMIA_NULLTYPE, {0}, FORBID}},
  CALC(AP_Out_Laplace),
  {{"Price", DOUBLE, {100}, FORBID}, {"Delta", DOUBLE, {100}, FORBID} , {" ", PR
  CHK_OPT(AP_Out_Laplace),
  CHK_ok,
  MET(Init)
} ;
```