Help

```
extern "C" {
#include "
href../../../mod/bmspectraly_negative1d/bmspectraly_negative1d_std/bmspectraly_n
#include "pnl/pnl_specfun.h"
#include "pnl/pnl_integration.h"
#include <pnl/pnl_complex.h>
}

typedef struct params_bmsntsl {
double C;
double G ;
double M;
double vol ;
double Y;
double r;
double mu;   // the drift fixed by the EMM requirement
double T;   // time to expiry
int N;     // the algorithm parameter: number of terms in the approximate formul
double K; //rebate
double S; //spot
double H; // barrier
    double x; // x-mu T
double alp; // parameter of the parabolic transform
double eps; // modified accuracy parameter epsilon_1 from the paper
double om; // the shift parameter
double d;  // the algorithm parameter omega_alpha_f from the paper
double zet; // the space step in the approximate integral formula


} params_bmsntsl;


static void Set_params(params_bmsntsl *p, double r, double divid, double Sigma,
{

p->C = Cminus;    //in the paper C=c, Y=nu, G=lambda_+.
p->G = Lambdaminus; //lm=lambda_+
p->M = 20; //artificial lambda_-
p->vol = Sigma*Sigma / 2; //volatility
```

```
p->Y = Alphaminus;
p->T = t;
p->N = N;
p->K = k;
p->S = s;
p->H = h;
p->r = r;
p->mu = r - p->vol - divid + pnl_sf_gamma(-Alphaminus)*Cminus*(exp(Alphaminus*lo
    p->x = log(s/h)+(p->mu)*t;
p->alp = 1.4;
p->eps=M_PI*eps*exp(r*t);
if (p->x<0)//Call-like  //
    {
p->om=-0.4*p->M;
p->d = -p->M + pow(p->M, 1 - p->alp)*pow(p->M + 1.8*p->om, p->alp);
}
else    // Put-like
{
p->om = 0.4*p->G;
p->d = p->G - pow(p->G, 1 - p->alp)*pow(p->G - 1.8*p->om, p->alp);
}
p->zet = 1.8*M_PI*p->om / (-log(p->eps) + p->mu*p->d*p->x - t*(-p->vol*p->d*p->d

}

static void psimTSL(params_bmsntsl *p, dcomplex u, dcomplex *res)//computation f
{
dcomplex aux1, aux2, aux3;

aux1 = RCmul(p->vol,Cmul(u, u));

aux2.r = p->G - u.i;
aux2.i = u.r;

aux3=Cexp(RCmul(p->Y,Clog(aux2)));

res->r = aux1.r+p->C*pnl_sf_gamma(-p->Y)*(exp(p->Y*log(p->G)) - aux3.r);
res->i = aux1.i-p->C*pnl_sf_gamma(-p->Y)*aux3.i;
}

static void psimTSLdrift(params_bmsntsl *p, dcomplex u, dcomplex *res)//computat
```

```
{
dcomplex aux1, aux2, aux3, aux4;

aux1 = RCmul(p->vol, Cmul(u, u));

aux2.r = p->G - u.i;
aux2.i = u.r;

aux3 = Cexp(RCmul(p->Y, Clog(aux2)));

aux4.r = 0;
aux4.i = -p->mu;
aux4 = Cmul(aux4, u);

res->r = aux1.r + p->C*pnl_sf_gamma(-p->Y)*(exp(p->Y*log(p->G)) - aux3.r)+aux4.r
res->i = aux1.i - p->C*pnl_sf_gamma(-p->Y)*aux3.i+aux4.i;
}

static dcomplex chip(double eta, double al, double om, double lp)
{
    dcomplex z, aux1, aux2, res;
z.i=lp;
z.r=0;
aux1.i=eta;
aux1.r=lp-om;
aux2=Cexp(RCmul(al,Clog(aux1)));

aux1.i=-pow(lp,1-al);
aux1.r=0.;
aux2=Cmul(aux1,aux2);
res=Cadd(z,aux2);

  return res;
  }

static dcomplex chim(double eta, double al, double om, double lm)// taken in the
                                                                  //"Efficient variati
//Journal of Computational Finance 18(2), 57-90
{
    dcomplex z, aux1, aux2, res;
z.i=-lm;
```

```
z.r=0;
aux1.i=-eta;
aux1.r=lm+om;
aux2=Cexp(RCmul(al,Clog(aux1)));

aux1.i=pow(lm-1.,1-al);
aux1.r=0.;
aux2=Cmul(aux1,aux2);
res=Cadd(z,aux2);

  return res;
  }

static dcomplex ZMueller(dcomplex beta, dcomplex q, double b1, double b2, void *

params_bmsntsl *p = (params_bmsntsl*)pp;

double d, d1, d2, x1, x2, x3, y1, y2, y3;
double u1, u2, u3, v1, v2, v3, xl1, xl2;
double a, a1, a2, b, c1, c2, e1, e2;
double e = 0.000001;
double x0, y0;
dcomplex aux, aux1, res;
int k;
int n = 1000; // the maximum number of itereations
// Start calculations
k = 0;
x3 = beta.r; y3 = beta.i;
x1 = x3 - b1; y1 = y3 - b2;
x2 = x3 + b1; y2 = y3 + b2;
res.r = beta.r; res.i = beta.i;

do
{
k = k + 1; x0 = res.r; y0 = res.i;
d = (x2 - x1)*(x2 - x1) + (y2 - y1)*(y2 - y1);
// Avoid divide by zero
if (d == 0) d = 1e-7;
xl1 = (x3 - x2)*(x2 - x1) + (y3 - y2)*(y2 - y1);
xl1 = xl1 / d;
xl2 = (x2 - x1)*(y3 - y2) + (x3 - x2)*(y2 - y1);
```

```
xl2 = xl2 / d;
d1 = (x3 - x1)*(x2 - x1) + (y3 - y1)*(y2 - y1);
d1 = d1 / d;
d2 = (x2 - x1)*(y3 - y1) + (x3 - x1)*(y2 - y1);
d2 = d2 / d;
// Get function values in 3 points
aux.r = x1;
aux.i = y1;
psimTSLdrift(p, aux, &aux1);
u1 = aux1.r + q.r; v1 = aux1.i + q.i;

aux.r = x2;
aux.i = y2;
psimTSLdrift(p, aux, &aux1);
u2 = aux1.r + q.r; v2 = aux1.i + q.i;

aux.r = x3;
aux.i = y3;
psimTSLdrift(p, aux, &aux1);
u3 = aux1.r + q.r; v3 = aux1.i + q.i;

// Calculate Mueller parameters
e1 = u1*(xl1*xl1 - xl2*xl2) - 2.0*v1*xl1*xl2 - u2*(d1*d1 - d2*d2);
e1 = e1 + 2.0*v2*d1*d2 + u3*(xl1 + d1) - v3*(xl2 + d2);
e2 = 2.0*xl1*xl2*u1 + v1*(xl1*xl1 - xl2*xl2) - 2.0*d1*d2*u2 - v2*(d1*d1 - d2*d2)
e2 = e2 + u3*(xl2 + d2) + v3*(xl1 + d1);
c1 = xl1*xl1*u1 - xl1*xl2*v1 - d1*xl1*u2 + xl1*d2*v2 + u3*xl1;
c1 = c1 - u1*xl2*xl2 - v1*xl1*xl2 + u2*xl2*d2 + v2*d1*xl2 - v3*xl2;
c2 = xl1*xl2*u1 + xl1*xl1*v1 - d2*xl1*u2 - xl1*d1*v2 + v3*xl1;
c2 = c2 + u1*xl1*xl2 - v1*xl2*xl2 - u2*xl2*d1 + v2*d2*xl2 + u3*xl2;
b1 = e1*e1 - e2*e2 - 4.0*(u3*d1*c1 - u3*d2*c2 - v3*d2*c1 - v3*d1*c2);
b2 = 2.0*e1*e2 - 4.0*(u3*d2*c1 + u3*d1*c2 + v3*d1*c1 - v3*d2*c2);
// Guard against divide by zero
if (b1 == 0) b1 = 1e-7;
a = atan(b2 / b1); a = a / 2.0;
b = sqrt(sqrt(b1*b1 + b2*b2));
b1 = b*cos(a); b2 = b*sin(a);
a1 = (e1 + b1)*(e1 + b1) + (e2 + b2)*(e2 + b2);
a2 = (e1 - b1)*(e1 - b1) + (e2 - b2)*(e2 - b2);

if (a1 > a2) { a1 = e1 + b1; a2 = e2 + b2; }
```

```
else
{
a1 = e1 - b1; a2 = e2 - b2;
}
a = a1*a1 + a2*a2;
xl1 = a1*d1*u3 - a1*d2*v3 + a2*u3*d2 + a2*v3*d1;
// Guard against divide by zero
if (a == 0) a = 1e-7;
xl1 = -2.0*xl1 / a;
xl2 = -d1*u3*a2 + d2*v3*a2 + a1*u3*d2 + a1*v3*d1;
xl2 = -2.0*xl2 / a;
// Calculate new estimate
res.r = x3 + xl1*(x3 - x2) - xl2*(y3 - y2);
res.i = y3 + xl2*(x3 - x2) + xl1*(y3 - y2);

// Test for number of iterations
if (k >= n){ x0 = res.r; y0 = res.i; };
if (fabs(u3) + fabs(v3) < e){ res.r = x3; res.i = y3; x0 = res.r; y0 = res.i; }
// Test if beta_+ is negative
if (res.i >= 0){ res.i = -res.i - 20;}
// Continue
x1 = x2; y1 = y2; x2 = x3; y2 = y3; x3 = res.r; y3 = res.i;
} while (fabs(res.r - x0) + fabs(res.i - y0) > e);
return res;
} // ZMueller()


static double RePutInt(double eta,  void *pp) ///Re( exp(ixChi_+(eta+i om)-T psi
{

params_bmsntsl *p = (params_bmsntsl*)pp;

dcomplex ix, aux, aux1, aux2, aux3, q, beta;

ix.i=p->x;
ix.r=0.;
//////////////exponent computation/////////////////////////ok
aux=chip(eta,p->alp, p->om, p->G);
psimTSL(p, aux, &aux1);
aux2=Cmul(aux, ix);
aux1=CRmul(aux1,-p->T);
```

```
aux1=Cadd(aux1,aux2);
/////////////////end////////////////////////
/////////exp*G0 computation/////////////////
aux2 =Cdiv(Cexp(aux1),aux);
aux3.r=-aux2.i;
aux3.i = aux2.r;
psimTSLdrift(p, aux, &aux1);
aux3 =Cmul(aux3,aux1);
aux2 = CRadd(aux1, p->r);
aux3 = Cdiv(aux3, aux2);
/////////////////end////////////////////////
////////// WH computation///////
q.r = -aux1.r;
q.i = -aux1.i;
aux2 = Cexp(CRmul(Clog(CRdiv(q, p->vol)), 0.5));

beta.r = 0;
beta.i = -aux2.r; //guess approximate root of the equation q+psi(-ibeta_+)=0

beta = ZMueller(beta, q, 10, 1, p); //computation of beta=-ibeta_+ with the Muel
aux2.r = 0;
aux2.i = 1;
beta = Cmul(beta, aux2);/// beta_+///

aux2 = Cdiv(aux, beta);///   computation of
aux.r = 1 + aux2.i; ///           beta_+(q) - i xi
aux.i = -aux2.r; ///              -----------------
aux3 = Cmul(aux3, aux); ///           beta_+(q)
/////////end////////////////////
////////////chi derivative computation////////////ok
aux.i=eta/(p->G);
aux.r=1-(p->om)/(p->G);
aux1=Cexp(RCmul(p->alp-1,Clog(aux)));
/////////////////end////////////////////////////
aux2=Cmul(aux3,aux1);
if (eta == p->N*p->zet){ aux2.r = 2.*aux2.r; }
return aux2.r*(p->alp);
}

static double ReCallInt(double eta,  void *pp) ///Re( exp(ixChi_-(eta+i om)-T ps
{
```

```
params_bmsntsl *p = (params_bmsntsl*)pp;

dcomplex ix, aux, aux1, aux2, aux3, q, beta;

ix.i=p->x;
ix.r=0.;
//////////////exponent computation////////////////////////ok
aux=chim(eta,p->alp, p->om, p->M);
psimTSL(p, aux, &aux1);
aux2=Cmul(aux, ix);
aux1=CRmul(aux1,-p->T);
aux1=Cadd(aux1,aux2);
/////////////////end////////////////////////
/////////exp*G0 computation//////////////////
aux2 = Cdiv(Cexp(aux1), aux);
aux3.r = -aux2.i;
aux3.i = aux2.r;
psimTSLdrift(p, aux, &aux1);
aux3 = Cmul(aux3, aux1);
aux2 = CRadd(aux1, p->r);
aux3 = Cdiv(aux3, aux2);
/////////////////end////////////////////////
////////// WH computation///////
q.r = -aux1.r;
q.i = -aux1.i;
aux2 = Cexp(CRmul(Clog(CRdiv(q, p->vol)), 0.5));

beta.r = 0;
beta.i = -aux2.r; //guess approximate root of the equation q+psi(-ibeta_+)=0

beta = ZMueller(beta, q, 10, 1, p); //computation of beta=-ibeta_+ with the Muel
aux2.r = 0;
aux2.i = 1;
beta = Cmul(beta, aux2);/// beta_+///

aux2 = Cdiv(aux, beta);///   computation of
aux.r = 1 + aux2.i; ///           beta_+(q) - i xi
aux.i = -aux2.r; ///              ----------------
aux3 = Cmul(aux3, aux); ///           beta_+(q)
/////////end//////////////////////
```

```
////////////chi derivative computation////////////
aux.i=-eta/(p->M-1);
aux.r=(p->M+p->om)/(p->M-1);
aux1=Cexp(RCmul(p->alp-1,Clog(aux)));
////////////////end//////////////////////////////
aux2=Cmul(aux3,aux1);
if (eta == p->N*p->zet){ aux2.r = 2.*aux2.r; }
return aux2.r*(p->alp);
}


extern "C" {
#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2018+2) //The "#els
  static int CHK_OPT(AP_Levendorskii)(void *Opt, void *Mod)
  {
    return NONACTIVE;
  }
  int CALC(AP_Levendorskii)(void *Opt, void *Mod, PricingMethod *Met)
  {
    return AVAILABLE_IN_FULL_PREMIA;
  }
#else

  static int Levendorskii(double Spot,double rebate, NumFunc_1  *payoff, double
  {
    double Strike;
    Strike = payoff->Par[0].Val.V_DOUBLE;
params_bmsntsl p;
double eps;
double UP_PARFT;
PnlFunc  Put, Call;

eps = 0.0005;//Accuracy

Set_params(&p, r, divid, Sigma, Alphaminus, Lambdaminus, Cminus, T, rebate, Spot
if (p.S>p.H)// the barrier is not crossed
    {
if (p.x>0)
{
 UP_PARFT = p.N*p.zet;
 Put.F =  &RePutInt;
```

```
 Put.params = &p;
 *ptprice = pnl_integration (&Put, 0, UP_PARFT, p.N , "trap");
 *ptprice =*ptprice*p.K/M_PI;
    }
else {
 UP_PARFT = p.N*p.zet;
 Call.F =  &ReCallInt;
 Call.params = &p;
 *ptprice = pnl_integration (&Call, 0, UP_PARFT, p.N , "trap");
 *ptprice =*ptprice*p.K/M_PI;
 }
*ptprice = exp(-r*T)**ptprice;
}
else   // the barrier is already crossed
 {    *ptprice = p.K;
   //printf("The barrier is already crossed. Please increase the spot price. \ n
 }
    return OK;
  }

  int CALC(AP_Levendorskii)(void *Opt, void *Mod, PricingMethod *Met)
  {
    TYPEOPT *ptOpt = (TYPEOPT *)Opt;
    TYPEMOD *ptMod = (TYPEMOD *)Mod;
    double r, divid;
double rebate;
NumFunc_1 *p;

    r = log(1. + ptMod->R.Val.V_DOUBLE / 100.);
    divid = log(1. + ptMod->Divid.Val.V_DOUBLE / 100.);
p = ptOpt->PayOff.Val.V_NUMFUNC_1;
rebate = p->Par[1].Val.V_DOUBLE;

  return  Levendorskii(ptMod->S0.Val.V_PDOUBLE, rebate,ptOpt->PayOff.Val.V_NUMFU
  }

  static int CHK_OPT(AP_Levendorskii)(void *Opt, void *Mod)
  {
    if ((strcmp(((Option *)Opt)->Name, "DigitAmer") == 0))
      return OK;
```

```
    return WRONG;
  }


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

    if (first)
      {
        Met->Par[0].Val.V_PINT = 12 ;
        first = 0;
      }

 return OK;
  }

  PricingMethod MET(AP_Levendorskii) =
  {
    "AP_Levendorskii",
    {{"Number of Space Steps", PINT, {100}, ALLOW    },
 {" ", PREMIA_NULLTYPE, {0}, FORBID}},
    CALC(AP_Levendorskii),
    {{"Price", DOUBLE, {100}, FORBID}, {" ", PREMIA_NULLTYPE, {0}, FORBID}},
    CHK_OPT(AP_Levendorskii),
    CHK_ok,
    MET(Init)
  } ;
}
```