Help

```
#include "
href../../../mod/bs1d/bs1d_std/bs1d_std_h_src.pdfbs1d_std.h"
#define INC 1.0e-5 /*Relative Increment for Delta-Hedging*/

/*assign_var_temp*/
static void assign_var_temp(double *sst,
                            double *alpha,
                            double *phi,
                            double *f,
                            double *b,
                            const double K,
                            const double T,
                            const double sigma,
                            const double delta,
                            const double r)
{
  *sst = sigma * sqrt(T);
  *b = delta - r + sigma * sigma / 2.;
  *f = sqrt((*b) * (*b) + 2.*r * sigma * sigma);
  *phi = ((*b) - (*f)) / 2.;
  *alpha = ((*b) + (*f)) / 2.;
}

/*assign_var_temp_L*/
static void assign_var_temp_L(const double sst,
                              const double alpha,
                              const double phi,
                              const double f,
                              const double b,
                              double *lambda,
                              double *d0,
                              double *d1pL,
                              double *d1pK,
                              double *d1mL,
                              double *d1mK,
                              const double K,
                              const double T,
                              const double sigma,
                              const double delta,
```

```
                                const double r,
                                const double x,
                                const double L)
{
  *lambda = x / L;
  *d0 = (log(*lambda) - f * T) / sst;
  *d1pL = (log(*lambda) + b * T) / sst;
  *d1pK = (log(*lambda * K / L) + b * T) / sst;
  *d1mL = (-log(*lambda) + b * T) / sst;
  *d1mK = (log(K / x) + b * T) / sst;
}

/*call_up_out*/
static double call_up_out(const double sst,
                          const double lambda,
                          const double alpha,
                          const double phi,
                          const double f,
                          const double b,
                          const double d0,
                          const double d1pL,
                          const double d1pK,
                          const double d1mL,
                          const double d1mK,
                          const double K,
                          const double T,
                          const double sigma,
                          const double delta,
                          const double r,
                          const double x,
                          const double L)
{
  double sig2 = sigma * sigma;
  double loglam = log(lambda);
  double ct =
    (L - K) * (exp(2 * phi / sig2 * loglam) * cdf_nor(d0) + exp(2 * alpha / sig2
    + x * exp(-delta * T) * (cdf_nor(d1mL - sst) - cdf_nor(d1mK - sst))
    - exp(-2 * (r - delta) / sig2 * loglam - delta * T) * L * (cdf_nor(d1pL - ss
    - K * exp(-r * T) * (cdf_nor(d1mL) - cdf_nor(d1mK) - exp((1 - 2 * (r - delta
  return ct;
}
```

```
/*dCdl*/
static double dCdL(const double sst,
                   const double lambda,
                   const double alpha,
                   const double phi,
                   const double f,
                   const double b,
                   const double d0,
                   const double d1pL,
                   const double d1pK,
                   const double d1mL,
                   const double d1mK,
                   const double K,
                   const double T,
                   const double sigma,
                   const double delta,
                   const double r,
                   const double x,
                   const double L)
{

  double sig2 = sigma * sigma;
  double loglam = log(lambda);
  double dCsdL =
    (1 - (L - K) / L * (2.*phi / sig2)) * exp(2.*phi * loglam / sig2) * cdf_nor(
    + (1 - (L - K) / L * (2.*alpha / sig2)) * exp(2.*alpha * loglam / sig2) * cd
    + exp(-delta * T) * 2.*(b - sig2) / sig2 * exp(-2.*(r - delta) * loglam / si
    * (cdf_nor(d1pL - sst) - cdf_nor(d1pK - sst))
    - exp(-r * T) * 2.*b * K / (sig2 * L) * exp(2.*b * loglam / sig2) * (cdf_nor
  return dCsdL;
}

/*maximise_C*/
static void maximise_C(double *Lmax,
                       const double sst,
                       double *lambda,
                       const double alpha,
                       const double phi,
                       const double f,
                       const double b,
```

```
                        double *d0,
                        double *d1pL,
                        double *d1pK,
                        double *d1mL,
                        double *d1mK,
                        const double K,
                        const double T,
                        const double sigma,
                        const double delta,
                        const double r,
                        const double x)
{
  double L1, L2, Ltmp, pas, derive;
  int i;
  L1 = x;
  L2 = 1000 * (x + K);
  pas = L2 - L1;
  for (i = 0; i <= 42; i++)
    {
      pas = pas / 2.;
      Ltmp = L1 + pas;
      assign_var_temp_L(sst, alpha, phi, f, b, lambda, d0, d1pL, d1pK, d1mL, d1m
      derive = dCdL(sst, *lambda, alpha, phi, f, b, *d0, *d1pL, *d1pK, *d1mL, *d
      if (derive <= 0) L2 = Ltmp;
      else L1 = Ltmp;
    };
  *Lmax = Ltmp;
}

/*low_coeff*/
static double low_coeff(const double K,
                        const double T,
                        const double sigma,
                        const double delta,
                        const double r,
                        const double x,
                        const double CLow)
{
  double c_euro, c_delta, x1, x2, x3, x4, x5, x6, x7, x8, x9, x10, y1, y2;
  pnl_cf_call_bs(x, K, T, r, delta, sigma, &c_euro, &c_delta);
  if ((CLow == c_euro) || (CLow <= (x - K))) return 1;
```

```
  else
    {
      x1 = T;
      x2 = sqrt(T);
      x3 = x / K;
      x4 = r;
      x5 = delta;
      x6 = MIN(r / MAX(delta, 0.00001), 5);
      x7 = x6 * x6;
      x8 = (CLow - c_euro) / K;
      x9 = x8 * x8;
      x10 = CLow / c_euro;
      y1 = 1.002 - 1.485 * 0.001 * x1 + 6.693 * 0.001 * x2
           - 1.451 * 0.001 * x3 - 3.430 * 0.01 * x4 + 6.301 * 0.01 * x5
           - 1.954 * 0.001 * x6 + 2.740 * 0.0001 * x7 - 1.043 * 0.1 * x8
           + 5.077 * 0.1 * x9 - 2.509 * 0.001 * x10;
      y2 = MAX(MIN(y1, 1.0133), 1);
      return y2;
    }
}

/*call_low_approx*/
static double call_low_approx(const double K,
                              const double T,
                              const double sigma,
                              const double delta,
                              const double r,
                              const double x)
{
  double sst, lambda, alpha, phi, f, b, d0, d1pL, d1pK, d1mL, d1mK, L;
  double CLow, LLow, coef, approx;

  assign_var_temp(&sst, &alpha, &phi, &f, &b, K, T, sigma, delta, r);
  L = x * 1.5;
  assign_var_temp_L(sst, alpha, phi, f, b, &lambda, &d0, &d1pL, &d1pK, &d1mL, &d
  maximise_C(&LLow, sst, &lambda, alpha, phi, f, b, &d0, &d1pL, &d1pK, &d1mL, &d
  L = LLow;
  CLow = call_up_out(sst, lambda, alpha, phi, f, b, d0, d1pL, d1pK, d1mL, d1mK,
  coef = low_coeff(K, T, sigma, delta, r, x, CLow);
  approx = coef * CLow;
```

```
  return approx;
}

/*call_low_delta*/
static double call_low_delta(const double K,
                             const double T,
                             const double sigma,
                             const double delta,
                             const double r,
                             const double x,
                             const double lba)
{
  double lba1, low_delta;
  lba1 = call_low_approx(K, T, sigma, delta, r, x * (1 + INC));
  low_delta = (lba1 - lba) / (x * INC);
  return low_delta;
}

/*put_low_delta*/
static double put_low_delta(const double K,
                            const double T,
                            const double sigma,
                            const double delta,
                            const double r,
                            const double x,
                            const double lba)
{
  double lba1, low_delta;
  lba1 = call_low_approx(x * (1 + INC), T, sigma, r, delta, K);
  low_delta = (lba1 - lba) / (x * INC);
  return low_delta;
}


static int CallAmer_Lba(double x,
                        NumFunc_1 *p,
                        double T,
                        double r,
                        double delta,
                        double sigma,
                        double *call_price,
```

```
                        double *call_delta)
{
  double K;
  if ((p->Compute) == &Call)
    {
      K = p->Par[0].Val.V_DOUBLE;
      *call_price = call_low_approx(K, T, sigma, delta, r, x);
      *call_delta = call_low_delta(K, T, sigma, delta, r, x, *call_price);
    }
  else if ((p->Compute) == &Put)
    {
      K = p->Par[0].Val.V_DOUBLE;
      *call_price = call_low_approx(x, T, sigma, r, delta, K);
      *call_delta = put_low_delta(K, T, sigma, delta, r, x, *call_price);
    }

  return OK;

}




int CALC(AP_Lba_CallAmer)(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 CallAmer_Lba(ptMod->S0.Val.V_PDOUBLE,
                      ptOpt->PayOff.Val.V_NUMFUNC_1,
                      ptOpt->Maturity.Val.V_DATE - ptMod->T.Val.V_DATE, r, divid
                      ptMod->Sigma.Val.V_PDOUBLE,
                      &(Met->Res[0].Val.V_DOUBLE), &(Met->Res[1].Val.V_DOUBLE));
}

static int CHK_OPT(AP_Lba_CallAmer)(void *Opt, void *Mod)
```

```
{
  if ((strcmp(((Option *)Opt)->Name, "CallAmer") == 0)
      || (strcmp(((Option *)Opt)->Name, "PutAmer") == 0))

    return OK;
  return WRONG;
}

static int MET(Init)(PricingMethod *Met, Option *Opt)
{
  if (Met->init == 0)
    {
      Met->init = 1;
    }

  return OK;
}

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