Help

```
#include  "bs1d_std.h"

/*Critical Price*/
static double PutCriticalPrice(double r, double divid, double sigma, double T, d
{
  const double precision = 0.0001;
  double previous;
  double current = K;
  double put_price, put_delta;
  do
    {
      previous = current;
      pnl_cf_put_bs(previous, K, T, r, divid, sigma, &put_price, &put_delta);
      current = K - put_price;
    }
  while (!(fabs((previous - current) / current) <= precision));
  return current;
}

static double CallCriticalPrice(double r, double divid, double sigma, double T,
{
  const double precision = 0.0001;
  double previous;
  double current = K;
  double call_price, call_delta;
  do
    {
      previous = current;
      pnl_cf_call_bs(previous, K, T, r, divid, sigma, &call_price, &call_delta);
      current = K + call_price;
    }
  while (!(fabs((previous - current) / current) <= precision));
  return current;
}

/* 2-points Bunch-Johnsonn AP*/
static int BunchJohnsonn_92(double s, NumFunc_1 *p, double t, double r, double d

{
```

```
  double p1, p2, crit12, k, price, delta, val, w1, w2, d1, d2, d1c, d2c;

  k = p->Par[0].Val.V_PDOUBLE;
  if ((p->Compute) == &Call)
    {
      val = -1.;
      pnl_cf_call_bs(s, k, t, r, divid, sigma, &price, &delta);
      p1 = price;
      crit12 = CallCriticalPrice(r, divid, sigma, t / 2., k);
    }
  else
    {
      val = 1.;
      pnl_cf_put_bs(s, k, t, r, divid, sigma, &price, &delta);
      p1 = price;
      crit12 = PutCriticalPrice(r, divid, sigma, t / 2., k);
    }
  d1c = (log(s / crit12) + (r - divid + 0.5 * SQR(sigma)) * t / 2.) / (sigma * s
  d2c = d1c - sigma * sqrt(t / 2.);
  d1 = (log(s / k) + (r - divid + 0.5 * SQR(sigma)) * t) / (sigma * sqrt(t));
  d2 = d1 - sigma * sqrt(t);
  w1 = exp(-divid * t / 2.) * cdf_nor(-val * d1c) + exp(-divid * t) * pnl_cdf2no
  w2 = exp(-r * t / 2.) * cdf_nor(-val * d2c) + exp(-r * t) * pnl_cdf2nor(val *
  p2 = val * (k * w2 - s * w1);

  /*Price*/
  *ptprice = 2.*p2 - p1;

  /*Delta*/
  *ptdelta = -2.*val * w1 - delta;

  return OK;
}


int CALC(AP_BunchJohnsonn)(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 BunchJohnsonn_92(ptMod->S0.Val.V_PDOUBLE,
                          ptOpt->PayOff.Val.V_NUMFUNC_1,
                          ptOpt->Maturity.Val.V_DATE - ptMod->T.Val.V_DATE, r, d
                          ptMod->Sigma.Val.V_PDOUBLE,
                          &(Met->Res[0].Val.V_DOUBLE), &(Met->Res[1].Val.V_DOUBL
}

static int CHK_OPT(AP_BunchJohnsonn)(void *Opt, void *Mod)
{
  if ((strcmp(((Option *)Opt)->Name, "CallAmer") == 0) || (strcmp(((Option *)Opt
    return OK;

  return WRONG;
}

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

  return OK;
}

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