Help

```
#include"
href../../../mod/bs1d/bs1d_std/bs1d_std_h_src.pdfbs1d_std.h"

#define AP_carr_eps 1e-7
#define AP_carr_h 1e-4


static double Call_euro_n(double S, double  K, double T, double r, double divid,

static double Put_euro_n(double S, double K, double T, double r, double divid, d

static int newton(double (*f)(double *, int), int n, double S, double K, double

/*Pow_int*/
static double pow_int(double x, int n)
{
  int i;
  double x1 = 1.;

  for (i = 1; i <= n; i++)
    x1 *= x;

  return x1;
}

/*Gamma*/
static double ap_carr_gamma(double r, double divid, double sigma)
{
  return 0.5 - (r - divid) / (sigma * sigma);
}

/*DELTA*/
static double ap_carr_delta(double T, int n)
{
  return T / n;
}

/*R*/
static double ap_carr_R(double r, double T, int n)
```

```
{
  return 1. / (1. + r * ap_carr_delta(T, n));
}

/*D*/
static double ap_carr_D(double divid, double T, int n)
{
  return 1. / (1. + divid * ap_carr_delta(T, n));
}

/*epsilon*/
static double ap_carr_epsilon(double r, double divid, double sigma, double T, in
{
  return sqrt(SQR(ap_carr_gamma(r, divid, sigma)) + 2 / (ap_carr_R(r, T, n) * ap
}

/*p*/
static double ap_carr_p(double r, double divid, double sigma, double T, int n)
{
  return (ap_carr_epsilon(r, divid, sigma, T, n) - ap_carr_gamma(r, divid, sigma
}

/*q*/
static double ap_carr_q(double r, double divid, double sigma, double T, int n)
{
  return 1 - ap_carr_p(r, divid, sigma, T, n);
}

/*phat*/
static double ap_carr_phat(double r, double divid, double sigma, double T, int n
{
  return (ap_carr_epsilon(r, divid, sigma, T, n) - ap_carr_gamma(r, divid, sigma
}

/*qhat*/
static double ap_carr_qhat(double r, double divid, double sigma, double T, int n
{
  return 1 - ap_carr_phat(r, divid, sigma, T, n);
}

/*Factor*/
```

```
static double Factor(int n)
{
  int i;
  double x = 1;
  if (n != 0)
    {
      for (i = 1; i <= n; i++)
        {
          x *= (double) i;
        }
      return x;
    }
  else
    {
      return 1;
    }
}

/*Combi*/
static double Combi(int n, int k)
{
  return Factor(n) / (Factor(k) * Factor(n - k));
}

/*Calleuro_n*/
static double Call_euro_n(double S, double K, double T, double r, double divid,
{
  double d = ap_carr_D(divid, T, n);
  double epsilon = ap_carr_epsilon(r, divid, sigma, T, n);
  double gamma_carr = ap_carr_gamma(r, divid, sigma);
  double R1 = ap_carr_R(r, T, n);
  double q1 = ap_carr_q(r, divid, sigma, T, n);
  double p1 = ap_carr_p(r, divid, sigma, T, n);
  double q_1 = ap_carr_qhat(r, divid, sigma, T, n);
  double p_1 = ap_carr_phat(r, divid, sigma, T, n);
  double S1, S2;
  int k, l;

  if (S > K)
    {
      return S * pow_int(d, n) - K * pow_int(R1, n) + Put_euro_n(S, K, T, r, div
```

```
    }
  else
    {
      S1 = 0;
      for (k = 0; k <= n - 1; k++)
        {
          S2 = 0;
          for (l = 0; l <= n - k - 1; l++)
            {
              S2 += Combi(n - 1 + l, n - 1) * (K * pow_int(d, n) * pow_int(q_1,
            }
          S1 += (pow_int(2 * epsilon * log(K / S), k) / Factor(k)) * S2;
        }
      return pow(S / K, gamma_carr + epsilon) * S1;
    }
}

/*Puteuro_n*/
static double Put_euro_n(double S, double K, double T, double r, double divid, d
{
  double d = ap_carr_D(divid, T, n);
  double epsilon = ap_carr_epsilon(r, divid, sigma, T, n);
  double gamma_carr = ap_carr_gamma(r, divid, sigma);
  double R1 = ap_carr_R(r, T, n);
  double q1 = ap_carr_q(r, divid, sigma, T, n);
  double p1 = ap_carr_p(r, divid, sigma, T, n);
  double q_1 = ap_carr_qhat(r, divid, sigma, T, n);
  double p_1 = ap_carr_phat(r, divid, sigma, T, n);
  double S1, S2;
  int k, l;

  if (S <= K)
    {
      return K * pow_int(R1, n) - S * pow_int(d, n) + Call_euro_n(S, K, T, r, di
    }
  else
    {
      S1 = 0;
      for (k = 0; k <= n - 1; k++)
        {
          S2 = 0;
```

```
          for (l = 0; l <= n - k - 1; l++)
            {
              S2 += Combi(n - 1 + l, n - 1) * (K * pow_int(R1, n) * pow_int(q1,
            }
          S1 += (pow_int(2 * epsilon * log(S / K), k) / Factor(k)) * S2;
        }
      return pow(S / K, gamma_carr - epsilon) * S1;
    }
}

/*derivx*/
static double deriv_x(double(*f)(double *, int), double *tab, int n)
{
  double tmp1;
  tab[0] += AP_carr_h;
  tmp1 = (*f)(tab, n);
  tab[0] -= AP_carr_h;
  return (tmp1 - (*f)(tab, n)) / AP_carr_h;
}

/*v*/
static double ap_carr_v(int i, int n, double s, double K, double T, double r, do
{
  return K * pow_int(ap_carr_R(r, T, n), n - i + 1) - s * pow_int(ap_carr_D(divi
}

/*A*/
static double ap_carr_a(int i, int h, int n, double S, double K, double T, doubl
{
  int j, k, l;
  double S1, S2, S3;
  double d = ap_carr_D(divid, T, n);
  double epsilon = ap_carr_epsilon(r, divid, sigma, T, n);
  double gamma_carr = ap_carr_gamma(r, divid, sigma);
  double delta = ap_carr_delta(T, n);
  double R1 = ap_carr_R(r, T, n);
  double q1 = ap_carr_q(r, divid, sigma, T, n);
  double p1 = ap_carr_p(r, divid, sigma, T, n);
  double q_1 = ap_carr_qhat(r, divid, sigma, T, n);
  double p_1 = ap_carr_phat(r, divid, sigma, T, n);
```

```
  S1 = 0.;
  for (j = h; j <= n - i + 1; j++)
    {
      S2 = 0.;
      for (k = 0; k <= j - 1; k++)
        {
          S3 = 0.;
          for (l = 0; l <= j - k - 1; l++)
            {
              S3 += Combi(j - 1 + l, j - 1) * (pow_int(p1 * R1, j) * pow_int(q1,
            }
          S2 += (pow_int(2 * epsilon * log(s[n - j + 1] / V), k) / Factor(k)) *
        }
      S1 += pow(V / s[n - j + 1], gamma_carr + epsilon) * S2;
    }
  return S1;
}

/*b*/
static double ap_carr_b(int i, int n, double S, double K, double T, double r, do
{
  int j, k, l;
  double S1, S2, S3;
  double d = ap_carr_D(divid, T, n);
  double epsilon = ap_carr_epsilon(r, divid, sigma, T, n);
  double gamma_carr = ap_carr_gamma(r, divid, sigma);
  double delta = ap_carr_delta(T, n);
  double R1 = ap_carr_R(r, T, n);
  double q1 = ap_carr_q(r, divid, sigma, T, n);
  double p1 = ap_carr_p(r, divid, sigma, T, n);
  double q_1 = ap_carr_qhat(r, divid, sigma, T, n);
  double p_1 = ap_carr_phat(r, divid, sigma, T, n);

  S1 = 0.;
  for (j = 1; j <= n - i + 1; j++)
    {
      S2 = 0.;
      for (k = 0; k <= j - 1; k++)
        {
          S3 = 0.;
```

```
          for (l = 0; l <= j - k - 1; l++)
            {
              S3 += Combi(j - 1 + l, j - 1) * (pow_int(q1 * R1, j) * pow_int(p1,


            }
          S2 += (pow_int(2 * epsilon * log(S / s[n - j + 1]), k) / Factor(k)) *
        }
      S1 += pow(S / s[n - j + 1], gamma_carr - epsilon) * S2;

    }

  return S1;
}

/*f1*/
static double f1(double *tab, int n)
{

  double s1 = tab[0];
  /* double S=tab[1];*/
  double K = tab[2];
  double T = tab[3];
  double r = tab[4];
  double divid = tab[5];
  double sigma = tab[6];
  double d = ap_carr_D(divid, T, n);
  double epsilon = ap_carr_epsilon(r, divid, sigma, T, n);
  double gamma_carr = ap_carr_gamma(r, divid, sigma);
  double delta = ap_carr_delta(T, n);
  double R1 = ap_carr_R(r, T, n);
  double p1 = ap_carr_p(r, divid, sigma, T, n);
  double p_1 = ap_carr_phat(r, divid, sigma, T, n);

  return pow(s1 / K, gamma_carr + epsilon) * K * (d * p_1 - R1 * p1) - delta * (
}


/*f2*/
static double f2(double *tab, int n)
{
```

```
  double s2 = tab[0];
  double S = tab[1];
  double K = tab[2];
  double T = tab[3];
  double r = tab[4];
  double divid = tab[5];
  double sigma = tab[6];
  double s1 = tab[7];
  double d = ap_carr_D(divid, T, n);
  double epsilon = ap_carr_epsilon(r, divid, sigma, T, n);
  double gamma_carr = ap_carr_gamma(r, divid, sigma);
  double delta = ap_carr_delta(T, n);
  double R1 = ap_carr_R(r, T, n);
  double p1 = ap_carr_p(r, divid, sigma, T, n);
  double p_1 = ap_carr_phat(r, divid, sigma, T, n);
  double q1 = ap_carr_q(r, divid, sigma, T, n);
  double q_1 = ap_carr_qhat(r, divid, sigma, T, n);
  double s_1[2];

  s_1[0] = K;
  s_1[1] = s1;

  return K * SQR(d * p_1) * (1 + 2 * q_1) - K * SQR(R1 * p1) * (1 + 2 * q1) - ap
}


/*f3*/
static double f3(double *tab, int n)
{
  double s3 = tab[0];
  double S = tab[1];
  double K = tab[2];
  double T = tab[3];
  double r = tab[4];
  double divid = tab[5];
  double sigma = tab[6];
  double s1 = tab[7];
  double s2 = tab[8];
  double d = ap_carr_D(divid, T, n);
  double epsilon = ap_carr_epsilon(r, divid, sigma, T, n);
  double gamma_carr = ap_carr_gamma(r, divid, sigma);
```

```
  double delta = ap_carr_delta(T, n);
  double R1 = ap_carr_R(r, T, n);
  double p1 = ap_carr_p(r, divid, sigma, T, n);
  double p_1 = ap_carr_phat(r, divid, sigma, T, n);
  double q1 = ap_carr_q(r, divid, sigma, T, n);
  double q_1 = ap_carr_qhat(r, divid, sigma, T, n);
  double s_1[3];

  s_1[0] = K;
  s_1[1] = s1;
  s_1[2] = s2;

  return K * pow_int(d * p_1, 3) * (1 + 3 * q_1 + 6 * SQR(q_1)) - K * pow_int(R1
}


/*critical_stripped_prices*/
static int critical_stripped_prices(double S, double K, double T, double r, doub
{
  double xinit[4];
  double s1_1, s1_2, s1_3;
  double s2_2, s2_3;
  double s3_3;

  xinit[0] = K;
  xinit[1] = 0.;
  xinit[2] = 0.;
  xinit[3] = 0.;


  newton(&f1, 1, S, K, T, r, divid, sigma, xinit, &s1_1, 1);
  newton(&f1, 2, S, K, T, r, divid, sigma, xinit, &s1_2, 1);
  newton(&f1, 3, S, K, T, r, divid, sigma, xinit, &s1_3, 1);



  s_1[0] = s1_1;
  s_2[0] = s1_2;
  s_3[0] = s1_3;
  xinit[0] = s_2[0];
  xinit[1] = s_2[0];
```

```
  newton(&f2, 2, S, K, T, r, divid, sigma, xinit, &s2_2, 2);
  xinit[0] = s_3[0];
  xinit[1] = s_3[0];
  newton(&f2, 3, S, K, T, r, divid, sigma, xinit, &s2_3, 2);

  s_2[1] = s2_2;
  s_3[1] = s2_3;

  xinit[0] = s_3[1];
  xinit[1] = s_3[0];
  xinit[2] = s_3[1];
  newton(&f3, 3, S, K, T, r, divid, sigma, xinit, &s3_3, 3);

  s_3[2] = s3_3;

  return OK;
}


/*Newton's algorithm*/
static int newton(double (*f)(double *, int), int n, double S, double K, double
{
  double tab1[7];
  double tab2[8];
  double tab3[9];
  double tab4[10];
  double *adresse;

  tab1[0] = xinit[0];
  tab1[1] = S;
  tab1[2] = K;
  tab1[3] = T;
  tab1[4] = r;
  tab1[5] = divid;
  tab1[6] = sigma;

  tab2[0] = xinit[0];
  tab2[1] = S;
  tab2[2] = K;
  tab2[3] = T;
```

```
tab2[4] = r;
tab2[5] = divid;
tab2[6] = sigma;
tab2[7] = xinit[1];

tab3[0] = xinit[0];
tab3[1] = S;
tab3[2] = K;
tab3[3] = T;
tab3[4] = r;
tab3[5] = divid;
tab3[6] = sigma;
tab3[7] = xinit[1];
tab3[8] = xinit[2];

tab4[0] = xinit[0];
tab4[1] = S;
tab4[2] = K;
tab4[3] = T;
tab4[4] = r;
tab4[5] = divid;
tab4[6] = sigma;
tab4[7] = xinit[1];
tab4[8] = xinit[2];
tab4[9] = xinit[3];


if (type == 1)
  {
    adresse = tab1;
  }
else if (type == 2)
  {
    adresse = tab2;
  }
else if (type == 3)
  {
    adresse = tab3;
  }
else
  {
```

```
      adresse = tab4;
    }
  *x = xinit[0];
  if (deriv_x(f, adresse, n) == 0)
    {
      return WRONG;
    }
  else
    {
      while (fabs((*f)(adresse, n) / deriv_x(f, adresse, n)) > AP_carr_eps)
        {
          if (deriv_x(f, adresse, n) == 0)
            {
              return WRONG;
            }
          else
            {
              *x -= ((*f)(adresse, n) / deriv_x(f, adresse, n));
              adresse[0] = *x;
            }
        }
      return OK;
    }
}


/*P1*/
static int pricing1(double S, double K, double T, double r, double divid, double
{
  int i, n = 1;

  if (S > K)
    {
      *P1 = Put_euro_n(S, K, T, r, divid, sigma, n) + ap_carr_b(1, n, S, K, T, r
    }
  else if (S <= s_1[n])
    {
      *P1 = K - S;
    }
  else
    {
```

```
      for (i = 1; i <= n; i++)
        {
          if ((S <= s_1[i - 1]) && (S > s_1[i]))
            {
              *P1 = ap_carr_v(i, n, S, K, T, r, divid) + ap_carr_b(i, n, S, K, T
            }
        }
    }
  /*printf("%f\ n",*P1);*/
  return OK;

}

/*P2*/
static int pricing2(double S, double K, double T, double r, double divid, double
{
  int i, n = 2;

  if (S > K)
    {
      *P2 = Put_euro_n(S, K, T, r, divid, sigma, n) + ap_carr_b(1, n, S, K, T, r
    }
  else if (S <= s_2[n])
    {
      *P2 = K - S;
    }
  else
    {
      for (i = 1; i <= n; i++)
        {
          if ((S <= s_2[i - 1]) && (S > s_2[i]))
            {
              *P2 = ap_carr_v(i, n, S, K, T, r, divid) + ap_carr_b(i, n, S, K, T
            }
        }
    }
  return OK;

}

/*P3*/
```

```
static int pricing3(double S, double K, double T, double r, double divid, double
{
  int i, n = 3;

  if (S > K)
    {
      *P3 = Put_euro_n(S, K, T, r, divid, sigma, n) + ap_carr_b(1, n, S, K, T, r
    }
  else
    {
      if (S <= s_3[n])
        {
          *P3 = K - S;
        }
      else
        {
          for (i = 1; i <= n; i++)
            {
              if ((S <= s_3[i - 1]) && (S > s_3[i]))
                {
                  *P3 = ap_carr_v(i, n, S, K, T, r, divid) + ap_carr_b(i, n, S,
                }
            }
        }

    }

  return OK;

}


/*decalage*/
static void decalage(double *tab1, int n, double *tab2, double K)
{
  int i;
  tab2[0] = K;
  for (i = 1; i <= n; i++)
    {
      tab2[i] = tab1[i - 1];
    }
```

```
}

/*PRICING*/
static int putamer_carr(double S, NumFunc_1 *p, double T, double r, double divid

{
  double s1[1], s2[2], s3[3], s_1[2], s_2[3], s_3[4];
  double s1h[1], s2h[2], s3h[3], s_1h[2], s_2h[3], s_3h[4];
  double P1, P2, P3, P1_h, P2_h, P3_h;

  critical_stripped_prices(S, p->Par[0].Val.V_DOUBLE, T, r, divid, sigma, s1, s2
  decalage(s1, 1, s_1, p->Par[0].Val.V_DOUBLE);
  decalage(s2, 2, s_2, p->Par[0].Val.V_DOUBLE);
  decalage(s3, 3, s_3, p->Par[0].Val.V_DOUBLE);

  pricing1(S, p->Par[0].Val.V_DOUBLE, T, r, divid, sigma, &P1, s_1);
  pricing2(S, p->Par[0].Val.V_DOUBLE, T, r, divid, sigma, &P2, s_2);
  pricing3(S, p->Par[0].Val.V_DOUBLE, T, r, divid, sigma, &P3, s_3);

  critical_stripped_prices(S + AP_carr_h, p->Par[0].Val.V_DOUBLE, T, r, divid, s

  decalage(s1h, 1, s_1h, p->Par[0].Val.V_DOUBLE);
  decalage(s2h, 2, s_2h, p->Par[0].Val.V_DOUBLE);
  decalage(s3h, 3, s_3h, p->Par[0].Val.V_DOUBLE);

  pricing1(S + AP_carr_h, p->Par[0].Val.V_DOUBLE, T, r, divid, sigma, &P1_h, s_1
  pricing2(S + AP_carr_h, p->Par[0].Val.V_DOUBLE, T, r, divid, sigma, &P2_h, s_2
  pricing3(S + AP_carr_h, p->Par[0].Val.V_DOUBLE, T, r, divid, sigma, &P3_h, s_3

  /*Price*/
  *put_price = 2.*P2 - P1;
  /**put_price=4.5*P3-4*P2+0.5*P1;*/

  /*Delta*/
  *put_delta = (2.*P2_h - P1_h - *put_price) / AP_carr_h;
  /**put_delta=((4.5*P3_h-4*P2_h+0.5*P1_h)-(*put_price))/AP_JU_h;*/

  return OK;
}

int CALC(AP_Carr_PutAmer)(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 putamer_carr(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_Carr_PutAmer)(void *Opt, void *Mod)
{
  if (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_Carr_PutAmer) =
{
  "AP_Carr_PutAmer",
  {{" ", PREMIA_NULLTYPE, {0}, FORBID}},
  CALC(AP_Carr_PutAmer),
  {{"Price", DOUBLE, {100}, FORBID}, {"Delta", DOUBLE, {100}, FORBID} , {" ", PR
  CHK_OPT(AP_Carr_PutAmer),
  CHK_ok ,
  MET(Init)
} ;
```