Help

```
/* We need Nd1 here */
#define USE_ND1
#include "bs1d_std.h"

#define AP_JU_Nmax 3
#define AP_JU_err 1e-7
#define AP_JU_Infinity 100.0
#define AP_JU_Neginfinity -100.0 /* AP_JU_Neginfinity for -infinity */
#define AP_JU_h 1e-4

/*Put Whaley Exponent*/
static double WhaleyPut_Exp(double r, double divid, double sigma, double T)
{
  double ratio = 2.0 * (r - divid) / (sigma * sigma);
  double delta = (ratio - 1.0);

  if (r == 0.)
    delta = SQR(delta) + 4.0 * (2.0 / (sigma * sigma)) / T;
  else
    delta = SQR(delta) + 4.0 * (2.0 * r / (sigma * sigma)) / (1.0 - exp(-r * T))

  return 0.5 * (1. - ratio - sqrt(delta));
}

static double Contact_PointPut(double r, double divid, double sigma,
                               double T, double K, double (*exponent_method)(dou
{
  const double precision = 0.00001;
  double previous;
  double exponent = (*exponent_method)(r, divid, sigma, T);
  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 = -exponent * (K - put_price) / ((1. - exp(-divid * T)
                * Nd1(previous, r, divid, -sigma, T, K)) - exponent);
    }
  while (!(fabs((previous - current) / current) <= precision));

  return current;
}

double critical_price(double r, double divid, double sigma,
                      double T, double K)
{
  double x;
  r = (r != 0. ? r : 1e-6);
  x = Contact_PointPut(r, divid, sigma, T, K, WhaleyPut_Exp);
  return x;
}



/* Mathematical functions */

/*derivx */
static double deriv_x(double(*f)(double *), double *tab)
{
  double tmp1;

  tab[0] += AP_JU_h;
  tmp1 = (*f)(tab);
  tab[0] -= AP_JU_h;
  return (tmp1 - (*f)(tab)) / AP_JU_h;

}

/*derivy*/
static double deriv_y(double(*f)(double *), double *tab)
{
  double tmp1;

  tab[1] += AP_JU_h;
  tmp1 = (*f)(tab);
  tab[1] -= AP_JU_h;
```

```
  return (tmp1 - (*f)(tab)) / AP_JU_h;

}

/*function d1*/
static double ap_ju_d1(double x, double y, double t, double r, double divid, dou
{
  if (t != 0.)
    {
      return (log(x / y) + (r - divid) * t) / (sigma * sqrt(t)) + sigma * sqrt(t

    }
  else
    {
      if (x == y)
        {
          return 0.;
        }
      else if (x > y)
        {
          return AP_JU_Infinity;
          /* we take 100 for AP_JU_Infinity because N(100)=1=N(AP_JU_Infinity)*/
        }
      else
        {
          return AP_JU_Neginfinity;
          /* we take -100 for AP_JU_Neginfinity because N(-100)=0=N(-AP_JU_Infin
        }
    }
}


/* function I */
static double ap_ju_I(double t1, double t2, double x, double y, double z, double
{
  double z1 = (r - divid - z + 0.5 * Phi * sigma * sigma) / sigma;
  double z2 = log(x / y) / sigma;
  double z3 = sqrt(z1 * z1 + 2.*Nu);
  double res;
  double sqrtt1, sqrtt2;
```

```
  sqrtt1 = sqrt(t1);
  sqrtt2 = sqrt(t2);

  if (t1 != 0.)
    {
      /* case t1 different of 0*/
      res =  exp(-Nu * t1) * cdf_nor(z1 * sqrtt1 + z2 / sqrtt1) - exp(-Nu * t2)
    }
  else
    {
      if (x == y)
        {
          /* case   x=y ( i.e. z2=0 ) and t1=0 */
          res =  0.5 - exp(-Nu * t2) * cdf_nor(z1 * sqrtt2) + 0.5 * (z1 / z3 + 1
        }
      else if (x > y)
        {
          /* case   x>y ( i.e. z2>0 ) and t1=0*/
          res =   1 - exp(-Nu * t2) * cdf_nor(z1 * sqrtt2 + z2 / sqrtt2) + 0.5 *
        }
      else
        {
          /* case   x<y ( i.e. z2<0 )  and t1=0*/
          res =   -exp(-Nu * t2) * cdf_nor(z1 * sqrtt2 + z2 / sqrtt2) + 0.5 * (z
        }
    }
  /*printf("%f %f\ n",res,z1);*/
  return res;
}

/* function IS*/
static double ap_ju_IS(double t1, double t2, double x, double y, double z, doubl
{
  double z1 = (r - divid - z + 0.5 * Phi * sigma * sigma) / sigma;
  double z2 = log(x / y) / sigma;
  double z3 = sqrt(z1 * z1 + 2.*Nu);
  double res;
  double sqrtt1, sqrtt2;

  sqrtt1 = sqrt(t1);
  sqrtt2 = sqrt(t2);
```

```
  if (t1 != 0.)
    {
      /* case  t1 different of 0 */
      res = (exp(-Nu * t1) * pnl_normal_density(z1 * sqrtt1 + z2 / sqrtt1) / sqr
    }
  else
    {
      if (x == y)
        {
          /* case x=y ( i.e. z2=0 ) and t1=0 */
          res = -exp(-Nu * t2) * pnl_normal_density(z1 * sqrtt2) / sqrtt2 / (sig
        }
      else if (x > y)
        {
          /* case x>y ( i.e. z2>0 ) and t1=0*/
          res = -exp(-Nu * t2) * pnl_normal_density(z1 * sqrtt2 + z2 / sqrtt2) /
        }
      else
        {
          /* case  x<y ( i.e. z2<0 ) and t1=0*/
          res = -exp(-Nu * t2) * pnl_normal_density(z1 * sqrtt2 + z2 / sqrtt2) /
        }
    }
  return res;
}

/* det*/
static int det(double(*f1)(double *), double(*f2)(double *), double *tab, double
{
  if (deriv_x(f1, tab)*deriv_y(f2, tab) - deriv_x(f2, tab)*deriv_y(f1, tab) == 0
    {
      return WRONG;
    }
  else
    {
      *d = deriv_x(f1, tab) * deriv_y(f2, tab) - deriv_x(f2, tab) * deriv_y(f1,
      return OK;
    }
}
```

```
/* coefficients of the inverse of the jacobian matrix */
/* coefficient 00 */
static double InvJ_00(double(*f1)(double *), double(*f2)(double *), double *tab)
{
  double d;
  if (det(f1, f2, tab, &d) != WRONG)
    return deriv_y(f2, tab) / d;
  else return 0.;
}

/* coefficient 01  */
static double InvJ_01(double(*f1)(double *), double(*f2)(double *), double *tab)
{
  double d;
  if (det(f1, f2, tab, &d) != WRONG)
    return  -(deriv_y(f1, tab)) / d;
  else return 0.;
}
/* coefficient 10 */
static double InvJ_10(double(*f1)(double *), double(*f2)(double *), double *tab)
{
  double d;
  if (det(f1, f2, tab, &d) != WRONG)
    return -(deriv_x(f2, tab)) / d;
  else return 0.;
}

/* coefficient 11 */
static double InvJ_11(double(*f1)(double *), double(*f2)(double *), double *tab)
{
  double d;
  if (det(f1, f2, tab, &d) != WRONG)
    return deriv_x(f1, tab) / d;
  else return 0.;
}

/*  inverse of the jacobian matrix */
static void create_InvJac(double(*InvJac[2][2])(double(*)(double *), double(*)(d
{
  InvJac[0][0] = &InvJ_00;
  InvJac[0][1] = &InvJ_01;
```

```
  InvJac[1][0] = &InvJ_10;
  InvJac[1][1] = &InvJ_11;
}

/* method of Newton-Raphson */
static int Newton_Raphson(double(*f1)(double *), double(*f2)(double *), double S
{
  double x[AP_JU_Nmax];
  double tab1[8];
  double tab2[10];
  double tab3[12];/*={x[1],x[2],S,K,T,r,divid,sigma,x1[0],x1[0],x2[0],x2[0]};*/
  double d;
  double *adresse;
  double(*InvJac[2][2])(double(*)(double *), double(*)(double *), double *);
  double(*f[2])(double *);
  double first_term, second_term, f0_ad, f1_ad;

  x[0] = 0.;
  x[1] = x1[0];
  x[2] = x2[0];

  tab1[0] = x[1];
  tab1[1] = x[2];
  tab1[2] = S;
  tab1[3] = K;
  tab1[4] = T;
  tab1[5] = r;
  tab1[6] = divid;
  tab1[7] = sigma;

  tab2[0] = x[1];
  tab2[1] = x[2];
  tab2[2] = S;
  tab2[3] = K;
  tab2[4] = T;
  tab2[5] = r;
  tab2[6] = divid;
  tab2[7] = sigma;
  tab2[8] = x1[0];
  tab2[9] = x2[0];
```

```
tab3[0] = x[1];
tab3[1] = x[2];
tab3[2] = S;
tab3[3] = K;
tab3[4] = T;
tab3[5] = r;
tab3[6] = divid;
tab3[7] = sigma;
tab3[8] = x1[0];
tab3[9] = x1[0];
tab3[10] = x2[0];
tab3[11] = x2[0];

create_InvJac(InvJac);
f[0] = f1;
f[1] = f2;

if (type == 1)
  {
    adresse = tab1;
  }
else if (type == 2)
  {
    adresse = tab2;
  }
else
  {
    x[1] = x2[0];
    x[2] = x2[1];
    tab3[0] = x[1];
    tab3[1] = x[2];
    tab3[9] = x1[1];
    tab3[11] = x2[1];
    adresse = tab3;
  }
if (det(f1, f2, adresse, &d) == WRONG)
  {
    return WRONG;
  }
else
  {
```

```
      f0_ad = f[0](adresse);
      f1_ad = f[1](adresse);
      first_term = InvJac[0][0](f1, f2, adresse) * f0_ad + InvJac[0][1](f1, f2,
      second_term = InvJac[1][0](f1, f2, adresse) * f0_ad + InvJac[1][1](f1, f2,

      while ((fabs(first_term) > AP_JU_err) || (fabs(second_term) > AP_JU_err))
        {
          x[1] -= first_term;
          x[2] -= second_term;

          adresse[0] = x[1];
          adresse[1] = x[2];

          f0_ad = f[0](adresse);
          f1_ad = f[1](adresse);
          first_term = InvJac[0][0](f1, f2, adresse) * f0_ad + InvJac[0][1](f1,
          second_term = InvJac[1][0](f1, f2, adresse) * f0_ad + InvJac[1][1](f1,
        }

      *coeff_B = x[1];
      *coeff_b = x[2];
      return OK;
    }
}


/* APPROXIMATION  BY ONE EXPONENTIAL */
/* Functions for which (B11,b11) is solution */

static double f1_11(double *tab)
{
  double B11 = tab[0];
  double b11 = tab[1];
  /* double S=tab[2]; */
  double K = tab[3];
  double T = tab[4];
  double r = tab[5];
  double divid = tab[6];
  double sigma = tab[7];
  double put_price, put_delta;
```

```
  pnl_cf_put_bs(B11, K, T, r, divid, sigma, &put_price, &put_delta);

  return K - B11 - put_price - K * (1 - exp(-r * T)) + B11 * (1 - exp(-divid * T
}

static double f2_11(double *tab)
{
  double B11 = tab[0];
  double b11 = tab[1];
  /* double S=tab[2]; */
  double K = tab[3];
  double T = tab[4];
  double r = tab[5];
  double divid = tab[6];
  double sigma = tab[7];
  double exp_minus_divid_T = exp(-divid * T);

  return -1. + exp_minus_divid_T * cdf_nor(-ap_ju_d1(B11, K, T, r, divid, sigma)
}

/*P1*/
static int ap_ju_pricing1(double S, double K, double T, double r, double divid,
{
  double B11, b11;
  double put_price, put_delta;
  double temp1[1];
  double temp2[1];

  temp1[0] = critical_price(r, divid, sigma, T, K);
  temp2[0] = 0.;

  pnl_cf_put_bs(S, K, T, r, divid, sigma, &put_price, &put_delta);
  Newton_Raphson(&f1_11, &f2_11, S, K, T, r, divid, sigma, &B11, &b11, 1, temp1,

  if (S <= B11)
    *P1 = K - S;
  else
    *P1 = put_price + K * (1 - exp(-r * T)) - S * (1 - exp(-divid * T)) - K * ap

  return OK;
}
```

```
/* APPROXIMATION BY TWO EXPONENTIAL PIECES  */
/* functions which (B21,b21) is solution */

static double f1_21(double *tab)
{
  double B21 = tab[0];
  double b21 = tab[1];
  /* double S=tab[2]; */
  double K = tab[3];
  double T = tab[4];
  double r = tab[5];
  double divid = tab[6];
  double sigma = tab[7];
  double put_price, put_delta;
  double B21_exp = B21 * exp(b21 * T / 2.);

  pnl_cf_put_bs(B21_exp, K, T / 2, r, divid, sigma, &put_price, &put_delta);
  return K - B21_exp - put_price - K * (1 - exp(-r * T / 2)) + B21_exp * (1 - ex
}

static double f2_21(double *tab)
{
  double B21 = tab[0];
  double b21 = tab[1];
  /* double S=tab[2];*/
  double K = tab[3];
  double T = tab[4];
  double r = tab[5];
  double divid = tab[6];
  double sigma = tab[7];
  double exp_minus_divid_ToverTwo = exp(-divid * T / 2.);
  double B21_exp = B21 * exp(b21 * T / 2.);


  return -1. + exp_minus_divid_ToverTwo * cdf_nor(-ap_ju_d1(B21_exp, K, T / 2.,
}

/* functions for which (B22,b22) is solution  */
```

```
static double f1_22(double *tab)
{
  double B22 = tab[0];
  double b22 = tab[1];
  /* double S=tab[2]; */
  double K = tab[3];
  double T = tab[4];
  double r = tab[5];
  double divid = tab[6];
  double sigma = tab[7];
  double B21 = tab[8];
  double b21 = tab[9];
  double put_price, put_delta, value;

  pnl_cf_put_bs(B22, K, T, r, divid, sigma, &put_price, &put_delta);
  value = K - B22 - put_price - K * (1 - exp(-r * T)) + B22 * (1 - exp(-divid *

  return value;
}


static double f2_22(double *tab)
{
  double B22 = tab[0];
  double b22 = tab[1];
  /* double S=tab[2]; */
  double K = tab[3];
  double T = tab[4];
  double r = tab[5];
  double divid = tab[6];
  double sigma = tab[7];
  double B21 = tab[8];
  double b21 = tab[9];
  double exp_minus_divid_T = exp(-divid * T);

  return -1. + exp_minus_divid_T * cdf_nor(-ap_ju_d1(B22, K, T, r, divid, sigma)

}

/*P2*/
static int ap_ju_pricing2(double S, double K, double T, double r, double divid,
```

```
{
  double B11, b11;
  double B21, b21;
  double B22, b22;
  double BT = MIN(K, divid != 0. ? K * r / divid : K);
  double temp1[1];
  double temp2[1];
  double put_price, put_delta;

  temp1[0] = critical_price(r, divid, sigma, T, K);
  temp2[0] = 0.;

  if (fabs(BT - (*temp1)) < 0.05 * BT)
    {
      Newton_Raphson(&f1_11, &f2_11, S, K, T, r, divid, sigma, &B11, &b11, 1, te
      Newton_Raphson(&f1_21, &f2_21, S, K, T, r, divid, sigma, &B21, &b21, 1, &B
      Newton_Raphson(&f1_22, &f2_22, S, K, T, r, divid, sigma, &B22, &b22, 2, &B

    }
  else
    {
      Newton_Raphson(&f1_11, &f2_11, S, K, T, r, divid, sigma, &B11, &b11, 1, te
      Newton_Raphson(&f1_21, &f2_21, S, K, T, r, divid, sigma, &B21, &b21, 1, &B
      Newton_Raphson(&f1_22, &f2_22, S, K, T, r, divid, sigma, &B22, &b22, 2, &B

    }

  pnl_cf_put_bs(S, K, T, r, divid, sigma, &put_price, &put_delta);

  if (S <= B22)
    *P2 = K - S;
  else
    *P2 = put_price + K * (1 - exp(-r * T)) - S * (1 - exp(-divid * T)) - K * ap

  return OK;
}

/* APPROXIMATION BY THREE EXPONENTIAL PIECES*/
/*functions for which (B31,b31) is solution */

static double f1_31(double *tab)
```

```
{
  double B31 = tab[0];
  double b31 = tab[1];
  /* double S=tab[2];*/
  double K = tab[3];
  double T = tab[4];
  double r = tab[5];
  double divid = tab[6];
  double sigma = tab[7];
  double put_price, put_delta, value;
  double B31_exp = B31 * exp(2.*b31 * T / 3.);


  value = pnl_cf_put_bs(B31_exp, K, T / 3, r, divid, sigma, &put_price, &put_del
  value = K - B31_exp - put_price - K * (1. - exp(-r * T / 3.)) + B31_exp * (1 -

  return value;


}

static double f2_31(double *tab)
{
  double B31 = tab[0];
  double b31 = tab[1];
  /* double S=tab[2]; */
  double K = tab[3];
  double T = tab[4];
  double r = tab[5];
  double divid = tab[6];
  double sigma = tab[7];
  double value;
  double exp_minus_divid_ToverThree = exp(-divid * T / 3.);
  double B31_exp = B31 * exp(2.*b31 * T / 3.);

  value = -1. + exp_minus_divid_ToverThree * cdf_nor(-ap_ju_d1(B31_exp, K, T / 3

  return value;

}
```

```
/* functions for which (B32,b32) is solution */

static double f1_32(double *tab)
{
  double B32 = tab[0];
  double b32 = tab[1];
  /* double S=tab[2];*/
  double K = tab[3];
  double T = tab[4];
  double r = tab[5];
  double divid = tab[6];
  double sigma = tab[7];
  double B31 = tab[8];
  double b31 = tab[9];
  double put_price, put_delta, value;
  double B31_exp = B31 * exp(b31 * T / 3.);
  double B32_exp = B32 * exp(b32 * T / 3.);
  double twoT_over_three = 2 * T / 3;

  pnl_cf_put_bs(B32_exp, K, twoT_over_three, r, divid, sigma, &put_price, &put_d
  value = K - B32_exp - put_price - K * (1 - exp(-r * twoT_over_three)) + B32_ex

  return value;
}


static double f2_32(double *tab)
{
  double B32 = tab[0];
  double b32 = tab[1];
  /* double S=tab[2]; */
  double K = tab[3];
  double T = tab[4];
  double r = tab[5];
  double divid = tab[6];
  double sigma = tab[7];
  double B31 = tab[8];
  double b31 = tab[9];
  double value;
  double exp_minus_divid_twoToverThree = exp(-divid * 2.*T / 3.);
```

```
  double B31_exp = B31 * exp(b31 * T / 3.);
  double B32_exp = B32 * exp(b32 * T / 3.);
  double twoT_over_three = 2 * T / 3;

  value = -1. + exp_minus_divid_twoToverThree * cdf_nor(-ap_ju_d1(B32_exp, K, tw

  return value;
}

/* functions for which (B33,b33) is solution */

static double f1_33(double *tab)
{
  double B33 = tab[0];
  double b33 = tab[1];
  /* double S=tab[2]; */
  double K = tab[3];
  double T = tab[4];
  double r = tab[5];
  double divid = tab[6];
  double sigma = tab[7];
  double B31 = tab[8];
  double b31 = tab[9];
  double B32 = tab[10];
  double b32 = tab[11];
  double put_price, put_delta, value;

  pnl_cf_put_bs(B33, K, T, r, divid, sigma, &put_price, &put_delta);
  value = K - B33 - put_price - K * (1 - exp(-r * T)) + B33 * (1 - exp(-divid *

  return value;

}

static double f2_33(double *tab)
{
  double B33 = tab[0];
  double b33 = tab[1];
  /* double S=tab[2];*/
  double K = tab[3];
  double T = tab[4];
```

```
  double r = tab[5];
  double divid = tab[6];
  double sigma = tab[7];
  double B31 = tab[8];
  double b31 = tab[9];
  double B32 = tab[10];
  double b32 = tab[11];
  double value;
  double exp_minus_divid_T = exp(-divid * T);
  double twoT_over_three = 2.*T / 3.;

  value = -1. + exp_minus_divid_T * cdf_nor(-ap_ju_d1(B33, K, T, r, divid, sigma

  return value;
}

/*P3*/
static int ap_ju_pricing3(double S, double K, double T, double r, double divid,
{
  double B11, b11;
  double B31, b31;
  double B32, b32;
  double B33, b33;
  double BT = MIN(K, divid != 0 ? K * r / divid : K);
  double temp1[1];
  double temp2[1];
  double temp3[2];
  double temp4[2];
  double put_price, put_delta;

  temp1[0] = critical_price(r, divid, sigma, T, K);
  temp2[0] = 0.;
  if (fabs(BT - (*temp1)) < 0.05 * BT)
    {
      Newton_Raphson(&f1_11, &f2_11, S, K, T, r, divid, sigma, &B11, &b11, 1, te
      Newton_Raphson(&f1_31, &f2_31, S, K, T, r, divid, sigma, &B31, &b31, 1, &B
      Newton_Raphson(&f1_32, &f2_32, S, K, T, r, divid, sigma, &B32, &b32, 2, &B

      temp3[0] = B31;
      temp3[1] = b31;
      temp4[0] = B32;
```

```
      temp4[1] = 0;
      Newton_Raphson(&f1_33, &f2_33, S, K, T, r, divid, sigma, &B33, &b33, 3, te
    }
  else
    {
      Newton_Raphson(&f1_11, &f2_11, S, K, T, r, divid, sigma, &B11, &b11, 1, te
      Newton_Raphson(&f1_31, &f2_31, S, K, T, r, divid, sigma, &B31, &b31, 1, &B
      Newton_Raphson(&f1_32, &f2_32, S, K, T, r, divid, sigma, &B32, &b32, 2, &B

      temp3[0] = B31;
      temp3[1] = b31;
      temp4[0] = B32;
      temp4[1] = b32;
      Newton_Raphson(&f1_33, &f2_33, S, K, T, r, divid, sigma, &B33, &b33, 3, te
    }

  pnl_cf_put_bs(S, K, T, r, divid, sigma, &put_price, &put_delta);

  if (S <= B33)
    *P3 = K - S;
  else
    *P3 = put_price + K * (1. - exp(-r * T)) - S * (1. - exp(-divid * T)) - K *

  return OK;
}

/*PRICING*/
static int PutAmer_Ju(double S, NumFunc_1 *p, double T, double r, double divid,
{
  double P1, P2, P3, K;
  double P1_h, P2_h, P3_h;

  K = p->Par[0].Val.V_DOUBLE;

  ap_ju_pricing1(S, K, T, r, divid, sigma, &P1);
  ap_ju_pricing2(S, K, T, r, divid, sigma, &P2);
  ap_ju_pricing3(S, K, T, r, divid, sigma, &P3);

  ap_ju_pricing1(S + AP_JU_h, K, T, r, divid, sigma, &P1_h);
  ap_ju_pricing2(S + AP_JU_h, K, T, r, divid, sigma, &P2_h);
  ap_ju_pricing3(S + AP_JU_h, K, T, r, divid, sigma, &P3_h);
```

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

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

  return OK;
}

int CALC(AP_Ju_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_Ju(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_Ju_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_Ju_PutAmer) =
{
  "AP_Ju_PutAmer",
  {{" ", PREMIA_NULLTYPE, {0}, FORBID}},
  CALC(AP_Ju_PutAmer),
  {{"Price", DOUBLE, {100}, FORBID}, {"Delta", DOUBLE, {100}, FORBID} , {" ", PR
  CHK_OPT(AP_Ju_PutAmer),
  CHK_ok ,
  MET(Init)
};
```