Help

```
/*
 Author: Syoiti Ninomiya
 Tokyo Institute of Technology
 Implementation of Kusuoka-Ninomyia-Ninomyia algorithm "A
    new Weak approximation of stochastic differential equations
     by using Runge-Kutta method"
 */

#include  "hes1d_pad.h"
/**********************************************************
    ******/
/* */
/* */
/**********************************************************
    *****/

#include <math.h>
#include <stdlib.h>
#include <stdio.h>


static double dt, sq_dt, mu, rho_gl, alpha_gl, theta_gl,
    beta_gl;
static const double c1 = 0.5;

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion <
     (2007+2) //The "#else" part of the code will be freely av
    ailable after the (year of creation of this file + 2)
static int CHK_OPT(MC_AsianKNN_Heston)(void *Opt, void *
    Mod)
{
  return NONACTIVE;
}
int CALC(MC_AsianKNN_Heston)(void *Opt, void *Mod, Pricing
    Method *Met)
{
return AVAILABLE_IN_FULL_PREMIA;
}
#else
```

```
static double *vZ(double *sig, double *initial, double *de
    stination){
  double sq_y2;

  sq_y2 = sqrt(fabs(initial[1]));
  destination[0] =
  c1*dt*initial[0]*(mu-0.5*initial[1]-rho_gl*beta_gl/4.0)
  + sig[0]*sq_dt*initial[0]*sq_y2;
  destination[1] =
  c1*dt*(alpha_gl*(theta_gl-initial[1])-beta_gl*beta_gl/4.0
    )
  + sig[0]*sq_dt*rho_gl*beta_gl*sq_y2
  + sig[1]*sq_dt*beta_gl*sqrt((1.0-rho_gl*rho_gl)*fabs(ini
    tial[1]));
  destination[2] = c1*dt*initial[0];
  return destination;
}

static double *ExpZ5th(double *sig, double *initial,
    double *destination){
  /*
   * order 5 method (6-stages)
   *
   * 0   |
   * 2/5 | 2/5
   * 1/4 | 11/64  5/64
   * 1/2 |  0      0      1/2
   * 3/4 | 3/64   -15/64  3/8    9/16
   *  1  |  0     5/7     6/7    -12/7   8/7
   * ----+------------------------------------------------
   *     | 7/90    0     32/90   12/90  32/90   7/90
   *
   * Corresponding Butcher Array
   */
  double Y1[3], Y2[3], Y3[3], Y4[3], Y5[3];
  double fY0[3], fY1[3], fY2[3], fY3[3], fY4[3], fY5[3];
  double *pf0, *pf1, *pf2, *pf3, *pf4, *pf5;
  int i;

  pf0 = vZ(sig, initial, fY0);
  for (i=0; i<3; i++)
```

```
    Y1[i] = initial[i] + pf0[i];
  pf1 = vZ(sig, Y1, fY1);
  for (i=0; i<3; i++)
    Y2[i] = initial[i] + (11.0/64.0)*pf0[i] + (5.0/64.0)*pf
    1[i];
  pf2 = vZ(sig, Y2, fY2);
  for (i=0; i<3; i++)
    Y3[i] = initial[i] + (1.0/2.0)*pf2[i];
  pf3 = vZ(sig, Y3, fY3);
  for (i=0; i<3; i++)
    Y4[i] = initial[i] + (3.0/64.0)*pf0[i] - (15.0/64.0)*pf
    1[i]
    + (3.0/8.0)*pf2[i] + (9.0/16.0)*pf3[i];
  pf4 = vZ(sig, Y4, fY4);
  for (i=0; i<3; i++)
    Y5[i] = initial[i] + (5.0/7.0)*pf1[i] + (6.0/7.0)*pf2[
    i]
    -(12.0/7.0)*pf3[i] + (8.0/7.0)*pf4[i];
  pf5 = vZ(sig, Y5, fY5);
  for (i=0; i<3; i++)
    destination[i] = initial[i] + (7.0/90.0)*pf0[i] + (32.0
    /90.0)*pf2[i]
    + (12.0/90.0)*pf3[i] + (32.0/90.0)*pf4[i] + (7.0/90.0)*
    pf5[i];
  return destination;
}

static int MCAsianKNN(double x0, NumFunc_2  *p, double T,
    double r, double divid, double y0,double alpha,double theta,
    double beta,double rho, long niter, int n_steps,double inc,
    double *ptprice, double *ptdelta)
{
  double K, sqrt2;
  double *u_seq1, *u_seq2, *n_seq1, *n_seq2;

  K=p->Par[0].Val.V_DOUBLE;
  mu=r-divid;
  dt = T/(double)n_steps; sq_dt = sqrt(dt);
  rho_gl = rho; alpha_gl = alpha; beta_gl = beta; theta_gl
    = theta;
  sqrt2 = sqrt(2.0);
```

```
u_seq1 = (double *)calloc(4*n_steps, sizeof(double));
u_seq2 = u_seq1 + 2*n_steps;
n_seq1 = (double *)calloc(4*n_steps, sizeof(double));
n_seq2 = n_seq1 + 2*n_steps;
{
  double sum, x[2][3], dsum, dx[2][3];
  double *last=NULL, *dlast=NULL;
  long int i;
  int j;

  for (dsum=sum=0.0, i=0; i < niter; i++){
    b2_g_sobol_seq("G_SOBOL_1", 4*n_steps, u_seq1);
    {
      int k;
      for (k=0; k<2*n_steps; k++){
        n_seq1[k] = sqrt(-2.0*log(u_seq1[k]))*cos(2.0*M_
  PI*u_seq2[k]);
        n_seq2[k] = sqrt(-2.0*log(u_seq1[k]))*sin(2.0*M_
  PI*u_seq2[k]);
      } /** for (k) **/
    }
    for(x[0][0]=x0, dx[0][1]=x[0][1]=y0, dx[0][2]=x[0][2]
  =0.0,
        dx[0][0]=x0*(1.0+inc), j=0;
        j < n_steps; j++){
      double sig1[2], sig2[2]; /** sig1 is for Z_1, sig2
  is for Z_2 **/
      sig1[0] = 0.5*n_seq1[2*j] + n_seq1[2*j+1]/sqrt2;
      sig2[0] = 0.5*n_seq1[2*j] - n_seq1[2*j+1]/sqrt2;
      sig1[1] = 0.5*n_seq2[2*j] + n_seq2[2*j+1]/sqrt2;
      sig2[1] = 0.5*n_seq2[2*j] - n_seq2[2*j+1]/sqrt2;
      last = ExpZ5th(sig2, ExpZ5th(sig1, x[0], x[1]), x[0
  ]);
      dlast = ExpZ5th(sig2, ExpZ5th(sig1, dx[0], dx[1]),
  dx[0]);
    } /** for (j) **/
    if ((p->Compute) == &Call_OverSpot2){
      sum += (last[2]/(double)T - K > 0)? last[2]/(
  double)T - K : 0;
      dsum += (dlast[2]/(double)T - K > 0)? dlast[2]/(
    double)T - K : 0;
      }else{
        if ((p->Compute) == &Put_OverSpot2){
          sum += (K-last[2]/(double)T  > 0)? K-last[2]/(
    double)T  : 0;
          dsum += (K-dlast[2]/(double)T  > 0)? K-dlast[2]/(
    double)T  : 0;
        }
      }
    } /** for (i) **/
    *ptprice = exp(-r*T)*sum/(double)niter;
    *ptdelta=exp(-r*T)*(dsum-sum)/(double)niter/inc/x0;
  }
  free(u_seq1);
  free(n_seq1);
  b2_g_sobol_free();

  return OK;
}


int CALC(MC_AsianKNN_Heston)(void *Opt, void *Mod, Pricing
    Method *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 MCAsianKNN(ptMod->S0.Val.V_PDOUBLE,
                    ptOpt->PayOff.Val.V_NUMFUNC_2,
                    ptOpt->Maturity.Val.V_DATE-ptMod->T.Val
    .V_DATE,r,divid,
                    ptMod->Sigma0.Val.V_PDOUBLE,ptMod->Mea
    nReversion.hal.V_PDOUBLE,
                    ptMod->LongRunVariance.Val.V_PDOUBLE,
                    ptMod->Sigma.Val.V_PDOUBLE,
                    ptMod->Rho.Val.V_PDOUBLE,
                    Met->Par[0].Val.V_LONG,
                    Met->Par[1].Val.V_INT,
                    Met->Par[2].Val.V_DOUBLE,
                    &(Met->Res[0].Val.V_DOUBLE),
                    &(Met->Res[1].Val.V_DOUBLE)
    );

}

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

  return  WRONG;
}

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

      Met->Par[0].Val.V_LONG=10000;
      Met->Par[1].Val.V_INT=100;
      Met->Par[2].Val.V_PDOUBLE=0.001;
    }
  return OK;
}


PricingMethod MET(MC_AsianKNN_Heston)=
{
  "MC_Asian_KNN_Hes",
  {{"N iterations",LONG,{100},ALLOW},
   {"TimeStepNumber",LONG,{100},ALLOW},
   {"Delta Increment Rel",PDOUBLE,{100},ALLOW},
   {" ",PREMIA_NULLTYPE,{0},FORBID}},
  CALC(MC_AsianKNN_Heston),
  {{"Price",DOUBLE,{100},FORBID},
   {"Delta",DOUBLE,{100},FORBID} ,
   {" ",PREMIA_NULLTYPE,{0},FORBID}},
  CHK_OPT(MC_AsianKNN_Heston),
  CHK_mc,
  MET(Init)
};
```

References