Help

```
extern "C" {
#include  "hes1d_pad.h"
#include "enums.h"
}

#include "heston_kusuoka.h"

extern "C" {

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2007+2) //The "#els
  static int CHK_OPT(MC_AsianKusuokaEuler_Heston)(void *Opt, void *Mod)
  {
    return NONACTIVE;
  }
  int CALC(MC_AsianKusuokaEuler_Heston)(void *Opt, void *Mod, PricingMethod *Met
  {
    return AVAILABLE_IN_FULL_PREMIA;
  }
#else
  static int MCAsianKusuokaEuler(double x0, NumFunc_2  *p, double T, double r, d
  {
    int init_mc;
    double K, nu, nprice, ndelta, nerror_price, nerror_delta;
    int simulation_dim = 1;

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

    /*MC sampling*/
    init_mc = pnl_rand_init(generator, simulation_dim, niter);
    if (init_mc == OK)
      {
        heston_kusuoka(generator, alpha, beta, theta, nu, rho, K, T, x0, y0, r,

      }
    /* Call Price estimator */
    if ((p->Compute) == &Call_OverSpot2)
      {
        *ptprice = nprice;
```

```
        *pterror_price = nerror_price;

        /* Delta estimator */
        *ptdelta = ndelta;
        *pterror_delta = nerror_delta;
      }

    else /*Parity Results*/
      /*if ((p->Compute) == &Put_OverSpot2)*/
      {
        if (r == divid)
          {
            *ptprice = nprice - x0 * exp(-divid * T) + K * exp(-r * T);
            *ptdelta = ndelta - exp(-divid * T);
          }
        else
          {
            *ptprice = nprice - x0 * exp(-divid * T) * (1 - exp(-(r - divid) * T
            *ptdelta = ndelta - exp(-divid * T) * (1 - exp(-(r - divid) * T)) /
          }
        *pterror_price = nerror_price;
        *pterror_delta = nerror_delta;
      }
    return OK;
  }

  int CALC(MC_AsianKusuokaEuler_Heston)(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 MCAsianKusuokaEuler(ptMod->S0.Val.V_PDOUBLE,
                               ptOpt->PayOff.Val.V_NUMFUNC_2,
                               ptOpt->Maturity.Val.V_DATE - ptMod->T.Val.V_DATE,
                               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_ENUM.value,
                                    Met->Par[3].Val.V_ENUM.value,
                                    Met->Par[4].Val.V_ENUM.value,
                                    &(Met->Res[0].Val.V_DOUBLE),
                                    &(Met->Res[1].Val.V_DOUBLE),
                                    &(Met->Res[2].Val.V_DOUBLE),
                                    &(Met->Res[3].Val.V_DOUBLE));

  }

  static int CHK_OPT(MC_AsianKusuokaEuler_Heston)(void *Opt, void *Mod)
  {

    if ((strcmp(((Option *)Opt)->Name, "AsianCallFixedEuro") == 0) || (strcmp(((
      return OK;

    return  WRONG;
  }
#endif //PremiaCurrentVersion

  static PremiaEnumMember FlagSchemeMembers[] =
  {
    { "Euler", 0 },
    { "Euler+Romberg", 1 },
    { "Ninomiya-Victoir", 2 },
    { NULL, NULLINT }
  };

  static DEFINE_ENUM(FlagScheme, FlagSchemeMembers)


  static int MET(Init)(PricingMethod *Met, Option *Opt)
  {
    static int first = 1;
    int type_generator;


    if (first)
      {
```

```
      Met->Par[0].Val.V_LONG = 10000;
      Met->Par[1].Val.V_INT = 100;
      Met->Par[2].Val.V_ENUM.value = 0;
      Met->Par[2].Val.V_ENUM.members = &PremiaEnumBool;
      Met->Par[3].Val.V_ENUM.value = 0;
      Met->Par[3].Val.V_ENUM.members = &FlagScheme;
      Met->Par[4].Val.V_ENUM.value = 0;
      Met->Par[4].Val.V_ENUM.members = &PremiaEnumMCRNGs;

      first = 0;
    }
  type_generator = Met->Par[4].Val.V_ENUM.value;
  if (pnl_rand_or_quasi(type_generator) == PNL_QMC)
    {
      Met->Res[2].Viter = IRRELEVANT;
      Met->Res[3].Viter = IRRELEVANT;

    }
  else
    {
      Met->Res[2].Viter = ALLOW;
      Met->Res[3].Viter = ALLOW;
    }
  return OK;
}


PricingMethod MET(MC_AsianKusuokaEuler_Heston) =
{
  "MC_AsianKusouka_Hes",
  { {"N iterations", LONG, {100}, ALLOW},
    {"TimeStepNumber", LONG, {100}, ALLOW},
    {"Flag Control Variate", ENUM, {100}, ALLOW},
    {"Flag Scheme", ENUM, {100}, ALLOW},
    {"RandomGenerator(Quasi Random not allowed)", ENUM, {100}, ALLOW},
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CALC(MC_AsianKusuokaEuler_Heston),
  { {"Price", DOUBLE, {100}, FORBID},
    {"Delta", DOUBLE, {100}, FORBID} ,
    {"Error Price", DOUBLE, {100}, FORBID},
```

```
      {"Error Delta", DOUBLE, {100}, FORBID} ,
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CHK_OPT(MC_AsianKusuokaEuler_Heston),
    CHK_mc,
    MET(Init)
  };

}
```