Help

```
extern "C" {
#include  "
href../../../mod/roughbergomi2d/roughbergomi2d_std/roughbergomi2d_std_h_src.pdfr
#include "
href../../../common/enums_h_src.pdfenums.h"
}
#include "pnl/pnl_random.h"
#include "pnl/pnl_finance.h"
#include "
href../../../mod/roughbergomi2d/roughbergomi2d_std/RfBm_h_src.pdfRfBm.hpp"
#include <
href../../../common/math/highdim_solver/highdim_vector_h_src.pdfvector>
#include <
href../../../common/math/numerics_h_src.pdfnumeric>

/* The model considered is defined as
*
*  dS_t = \ sqrt{v_t} S_t dZ_t,
*  v_t = \ xi_0(t) \ exp\ left( \ eta \ tilde{W}_t - \ frac{1}{2} \ eta^2 t^{2H}
*
* where $Z, W$ are correlated Brownian motions with correlation \ rho.
* The process $\ tilde W$ is a variant of the fractional Brownian motion
* \ tilde{W}_t = \ int_0^t K(t,s) dW_s, \ quad K(t,s) = \ sqrt{2H} (t-s)^{H - 1/
*/


#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2018+2) //The "#els
extern "C" {
static int CHK_OPT(MC_Hybrid_RoughBergomi)(void *Opt, void *Mod)
{
  return NONACTIVE;
}
int CALC(MC_Hybrid_RoughBergomi)(void *Opt, void *Mod, PricingMethod *Met)
{
  return AVAILABLE_IN_FULL_PREMIA;
}
}
#else
```

```
typedef std::vector<double> Vector;


// x = s*x
template<typename T>
void scaleVector(std::vector<T>& x, T s) {
  for (size_t i = 0; i < x.size(); ++i)
    x[i] *= s;
}

// Note that Wtilde plays the role of the old WtildeScaled!
static void compute_V(Vector& v, const Vector& Wtilde, double H, double eta, dou
    double dt) {
  v[0] = xi;
  for (size_t i = 1; i < v.size(); ++i)
    v[i] = xi * exp( eta * Wtilde[i - 1] - 0.5 * eta * eta * pow((i - 1) * dt, 2
}

static double intVdt(const Vector & v, double dt) {
  return dt * std::accumulate(v.begin(), v.end(), 0.0);
}

static double intRootVdW(const Vector & v, const Vector & W1, double sdt) {
  double IsvdW = 0.0;
  for (size_t i = 0; i < v.size(); ++i)
    IsvdW += sqrt(v[i]) * sdt * W1[i];
  return IsvdW;
}

static double updatePayoff_cholesky(Vector& Wtilde, const Vector& W1,
    Vector& v, double eta, double H, double rho, double xi,
    double T, double K, int N){
  double dt = T / N;
  double sdt = sqrt(dt);
  scaleVector(Wtilde, pow(T, H)); // scale Wtilde for time T
  compute_V(v, Wtilde, H, eta, xi, dt); // compute instantaneous variance v
// now compute \ int v_s ds, \ int \ sqrt{v_s} dW_s with W = W1
  double Ivdt = intVdt(v, dt);
  double IsvdW = intRootVdW(v, W1, sdt);
// now compute the payoff by inserting properly into the BS formula
  double BS_vol = sqrt((1.0 - rho * rho) * Ivdt);
```

```
  double BS_spot = exp(-0.5 * rho * rho * Ivdt + rho * IsvdW);
  return pnl_bs_call(BS_spot, K, 1.0, 0., 0., BS_vol);
}

/*
 * Re-interpreting K as moneyness, we may have S0 = 1, without loss of generalit
 */
static void mc_bayer_roughbergomi_moneyness_cholesky(double eta, double H, doubl
    double xi, double K, double T, int M, int N, PnlRng *rng, double &price,
    double &stat) {

  RfBm rfbm(N, H, rng);

  // Allocate memory for Gaussian random numbers and the random vector v (instan
  // Note that W1, W1perp correspond to UNNORMALIZED increments of Brownian moti
  // i.e., are i.i.d. standard normal.
  Vector W1(N);
  Vector Wtilde(N);
  Vector v(N);

  double mean = 0.0; // will eventually be the mean
  double mu2 = 0.0; // will become second moment
  double var; // will eventually become variance (i.e., MC error).

  // The big loop which needs to be parallelized in future
  for (int m = 0; m < M; ++m) {
    // generate W and Wtilde
    rfbm(W1, Wtilde);

    double payoff = updatePayoff_cholesky(Wtilde, W1, v, eta, H, rho, xi, T, K,
    mean += payoff;
    mu2 += payoff * payoff;
  }

// compute mean and variance
  mean = mean / M;
  mu2 = mu2 / M;
  var = mu2 - mean * mean;

  price = mean;
  stat = sqrt(var / M);
```

```
}

static void mc_bayer_roughbergomi_cholesky(double S0, double eta, double H, doub
    double xi, double K, double T, int M, int N, PnlRng *rng, double &price,
    double &stat)
{
  mc_bayer_roughbergomi_moneyness_cholesky(eta, H, rho, xi, K / S0, T, M, N, rng
  price *= S0;
  stat *= S0;
}


extern "C" {
int MCBayer1_RoughBergomi(NumFunc_1  *p, double S0, double eta, double H, double
{
  //------------------Declaration of variable

  int call_put;
  double K;
  double price, stat;
  PnlRng *rng;

  if ((p->Compute) == &Call)
    call_put = 1;
  else
    call_put = 0;
  K = p->Par[0].Val.V_PDOUBLE;

  rng = pnl_rng_create(generator);
  pnl_rng_sseed(rng, 0);

  mc_bayer_roughbergomi_cholesky(S0, eta, H, rho, xi, K, T, M, N, rng, price, st

  *ptprice=price;

  //Put case
  if(call_put==0)
   *ptprice = *ptprice - S0 + K ;

  pnl_rng_free(&rng);
```

```
  return OK;
}

int CALC(MC_Hybrid_RoughBergomi)(void *Opt, void *Mod, PricingMethod *Met)
{
  TYPEOPT *ptOpt = (TYPEOPT *)Opt;
  TYPEMOD *ptMod = (TYPEMOD *)Mod;

  return MCBayer1_RoughBergomi(ptOpt->PayOff.Val.V_NUMFUNC_1,
                               ptMod->S0.Val.V_PDOUBLE,
                               ptMod->sigma.Val.V_PDOUBLE,
                               ptMod->H.Val.V_PDOUBLE,
                               ptMod->rho.Val.V_PDOUBLE,
                               ptMod->sigma0.Val.V_PDOUBLE,
                               ptOpt->Maturity.Val.V_DATE - ptMod->T.Val.V_DATE,
                               Met->Par[0].Val.V_LONG,
                               Met->Par[1].Val.V_PINT,
                               Met->Par[2].Val.V_ENUM.value,
                               &(Met->Res[0].Val.V_DOUBLE));
}

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

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

  return  WRONG;
}
}
#endif //PremiaCurrentVersion


extern "C" {
static int MET(Init)(PricingMethod *Met, Option *Opt)
{
  if (Met->init == 0)
    {
      Met->init = 1;
      Met->Par[0].Val.V_LONG = 50000;
      Met->Par[1].Val.V_PINT = 200;
```

```
      Met->Par[2].Val.V_ENUM.value = 0;
      Met->Par[2].Val.V_ENUM.members = &PremiaEnumMCRNGs;
      Met->HelpFilenameHint = "MC_Hybrid_RoughBergomi";
    }
  return OK;
}


PricingMethod MET(MC_Hybrid_RoughBergomi) =
{
  "MC_Hybrid_RoughBergomi",
  {
    {"Nb MC iterations", LONG, {100}, ALLOW},
    {"Nb discretisation steps", LONG, {100}, ALLOW},
    {"RandomGenerator", ENUM, {100}, ALLOW},
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CALC(MC_Hybrid_RoughBergomi),
  { {"Price", DOUBLE, {100}, FORBID},
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CHK_OPT(MC_Hybrid_RoughBergomi),
  CHK_mc,
  MET(Init)
};
}
```