Help

```
extern "C" {
#include  "
href../../../mod/sabr1d/sabr1d_std/sabr1d_std_h_src.pdfsabr1d_std.h"

#include "
href../../../common/enums_h_src.pdfenums.h"
#include "
href../../../common/error_msg_h_src.pdferror_msg.h"
#include "pnl/pnl_finance.h"
#include "pnl/pnl_random.h"
#include "pnl/pnl_vector.h"
#include "pnl/pnl_mathtools.h"
#include "pnl/pnl_specfun.h"
#include "pnl/pnl_cdf.h"
#include "pnl/pnl_root.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2017+2) //The "#els
static int CHK_OPT(MC_TruncatedEuler_Sabr)(void *Opt, void *Mod)
{
  return NONACTIVE;
}
int CALC(MC_TruncatedEuler_Sabr)(void *Opt, void *Mod, PricingMethod *Met)
{
  return AVAILABLE_IN_FULL_PREMIA;
}
#else

/* AZEVEDO CHAVES HERVE
herve.azevedo-chaves@eleves.enpc.fr
hervetolentino@me.com
*/

static double beta_param;
static double maximum(double a){
return (a>0.0)?a:0.0;
}

static double f1(double x){return pow(x,beta_param);};
```

```
// Full Truncated Euler scheme for the SABR-SV model (Stochastic Alpha Beta Rho
int MCTruncatedEulerSabr(double S0, NumFunc_1  *p, double T,double sigma0,double
{
  double K;
  double delta;

  delta=T/(double)n;
  K = p->Par[0].Val.V_PDOUBLE;
  beta_param=beta;
  pnl_rand_init(gen, 1, N);
   PnlVect *S_0 = pnl_vect_create_from_scalar(N,S0);// S_0 is a vector containin
PnlVect *S_delta = pnl_vect_create_from_zero(N);// S_delta is a vector which wil

PnlVect *sigma_0 = pnl_vect_create_from_scalar(N,sigma0);// sigma_0 is a vector
PnlVect *sigma_delta = pnl_vect_create_from_zero(N);// sigma_delta is a vector w

PnlRng *rng = pnl_rng_create(PNL_RNG_MRGK3);// creating a generator
pnl_rng_sseed(rng, 0);/* rng must be initialized. When sseed=0, a default value

// Loop on each time step passing from S_0 to S_delta and sigma_0 to sigma_delta
for(int i = 0; i < n; i++){

PnlVect *W2 = pnl_vect_new();// dehors?
pnl_vect_rng_normal(W2, N, rng);
pnl_vect_mult_scalar(W2,sqrt(delta));//W2 is the N vector Brownian motion from t

//Volatility computation : sigma_(i+1)*delta
pnl_vect_clone(sigma_delta,W2);
pnl_vect_mult_scalar(sigma_delta,alpha);
pnl_vect_minus_scalar(sigma_delta,0.5*delta*pnl_pow_i(alpha,2.0));
pnl_vect_map_inplace (sigma_delta,exp);
pnl_vect_mult_vect_term (sigma_delta,sigma_0);//sigma_delta = sigma_0 x exp(alph

//low-biased scheme for SABR simulation

PnlVect *W1 = pnl_vect_new();// dehors?
pnl_vect_rng_normal(W1, N, rng);
pnl_vect_axpby(rho,W2,sqrt((1.0-pnl_pow_i(rho,2.0))*delta),W1);//W1 is the N vec

//Euler full truncature scheme to compute S_delta : S_delta =  max(S_0 + max(S_0
pnl_vect_clone(S_delta,S_0);
```

```
pnl_vect_map_inplace(S_delta,maximum);
pnl_vect_map_inplace(S_delta,f1);
pnl_vect_mult_vect_term(S_delta,W1);
pnl_vect_mult_vect_term(S_delta,sigma_0);
pnl_vect_plus_vect(S_delta,S_0);
pnl_vect_map_inplace(S_delta,maximum);
pnl_vect_free (&W1);

//Euler scheme to compute S_delta : S_delta =  S_0 + S_0^beta *sigma_0.*W1;
/*pnl_vect_clone(S_delta,S_0);
pnl_vect_mult_vect_term(S_delta,W1);
pnl_vect_mult_vect_term(S_delta,sigma_0);
pnl_vect_plus_vect(S_delta,S_0);
pnl_vect_free (&W1);*/

//}

pnl_vect_clone(S_0,S_delta);
pnl_vect_clone(sigma_0,sigma_delta);

pnl_vect_free (&W2);

}
pnl_vect_minus_scalar(S_delta,K);
pnl_vect_map_inplace(S_delta,maximum);
*price = pnl_vect_sum(S_delta)/N;
pnl_vect_minus_scalar(S_delta,*price);
pnl_vect_mult_vect_term(S_delta,S_delta);
double std = pnl_vect_sum(S_delta)/(N-1);
*up = *price + 1.96*sqrt(std)/sqrt(1.0*N);
*down = *price - 1.96*sqrt(std)/sqrt(1.0*N);
pnl_rng_free (&rng);
pnl_vect_free (&S_0);
pnl_vect_free (&S_delta);
pnl_vect_free (&sigma_0);
pnl_vect_free (&sigma_delta);



  return OK;
}
```

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

      return MCTruncatedEulerSabr(ptMod->S0.Val.V_PDOUBLE,
                                  ptOpt->PayOff.Val.V_NUMFUNC_1,
                                  ptOpt->Maturity.Val.V_DATE - ptMod->T.Val.V_DATE,
                                  ,ptMod->Beta.Val.V_PDOUBLE,
                                  ptMod->Sigma.Val.V_PDOUBLE,
                                  ptMod->Rho.Val.V_PDOUBLE,
                                  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), &(Met->Res[1].Val.V_DOUBLE), &(Met->Res[2].Val.V_D
                                 );
}

static int CHK_OPT(MC_TruncatedEuler_Sabr)(void *Opt, void *Mod)
{
  if ((strcmp(((Option *)Opt)->Name, "CallEuro") == 0) /* || (strcmp(((Option *)

    return OK;
  return WRONG;
}

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

  Met->HelpFilenameHint = "mc_truncatedeulersabr";
      Met->Par[0].Val.V_LONG = 100000;
  Met->Par[1].Val.V_PINT = 16;
      Met->Par[2].Val.V_ENUM.value = 0;
      Met->Par[2].Val.V_ENUM.members = &PremiaEnumMCRNGs;
    }
```

```
  return OK;
}

PricingMethod MET(MC_TruncatedEuler_Sabr) =
{
  "MC_TruncatedEuler_Sabr",
  { {"N iterations", LONG, {100}, ALLOW},
{"Time Steps", PINT, {100}, ALLOW},
    {"RandomGenerator", ENUM, {100}, ALLOW},
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CALC(MC_TruncatedEuler_Sabr),
  { {"Price", DOUBLE, {100}, FORBID},
{"Inf Price", DOUBLE, {100}, FORBID},
 {"Sup Price", DOUBLE, {100}, FORBID},
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CHK_OPT(MC_TruncatedEuler_Sabr),
  CHK_mc,
  MET(Init)
};
}
```