Help

```
#include "math/ImportanceSampling_jl/src/wrapper.hpp"


extern "C" {
#include  "hes1d_std.h"
#include "enums.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2016+2) //The "#els
  static int CHK_OPT(MC_Multilevel)(void *Opt, void *Mod)
  {
    return NONACTIVE;
  }
  int CALC(MC_Multilevel)(void *Opt, void *Mod, PricingMethod *Met)
  {
    return AVAILABLE_IN_FULL_PREMIA;
  }
#else

  int CALC(MC_Multilevel)(void *Opt, void *Mod, PricingMethod *Met)
  {
    TYPEOPT *ptOpt = (TYPEOPT *)Opt;
    TYPEMOD *ptMod = (TYPEMOD *)Mod;
    NumFunc_1  *p = ptOpt->PayOff.Val.V_NUMFUNC_1;
    double r, divid;

    r = log(1. + ptMod->R.Val.V_DOUBLE / 100.);
    divid = log(1. + ptMod->Divid.Val.V_DOUBLE / 100.);

    double option_type = 0.;
    if ((p->Compute) == &Call)
      option_type = 1;
    else if ((p->Compute) == &Put)
      option_type = -1;


    Param P;
    PnlRng *rng = pnl_rng_create(Met->Par[3].Val.V_ENUM.value);
    pnl_rng_sseed(rng, 0);
    std::vector<double> payoff_coeffs(1, option_type);
```

```
    std::vector<double> spot(1, ptMod->S0.Val.V_PDOUBLE);
    std::vector<double> dividend_rate(1, divid);

    P.insert("model type", T_STRING, std::string("heston"));
    P.insert("option size", T_INT, 1);
    P.insert("strike", T_DOUBLE, option_type * p->Par[0].Val.V_DOUBLE);
    P.insert("spot", T_VECTOR, spot);
    P.insert("maturity", T_DOUBLE, ptOpt->Maturity.Val.V_DATE - ptMod->T.Val.V_D
    P.insert("interest rate", T_DOUBLE, r);
    P.insert("dividend rate", T_VECTOR, dividend_rate);
    P.insert("correlation", T_DOUBLE, 0.);
    P.insert("extra vol correlation", T_DOUBLE, 0.);
    P.insert("asset vol correlation", T_DOUBLE, ptMod->Rho.Val.V_PDOUBLE);
    P.insert("option type", T_STRING, std::string("basket"));
    P.insert("payoff coefficients", T_VECTOR, payoff_coeffs);
    P.insert("long run variance", T_VECTOR, std::vector<double>(1, ptMod->LongRu
    P.insert("initial volatility", T_VECTOR, std::vector<double>(1, ptMod->Sigma
    P.insert("volatility of volatility", T_VECTOR, std::vector<double>(1, ptMod-
    P.insert("reverting rate", T_VECTOR, std::vector<double>(1, ptMod->MeanRever
    P.insert("timestep number finest level", T_INT, Met->Par[0].Val.V_INT);
    P.insert("timestep number level one", T_INT, 4);


    bool use_IS = false;
    if (Met->Par[2].Val.V_ENUM.value == 1) use_IS = true;

    MultiLevelMonteCarloWrapper(rng, P, use_IS, Met->Res[0].Val.V_DOUBLE, Met->R

  return OK;
  }

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

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

    return  WRONG;
  }
#endif //PremiaCurrentVersion
```

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

        Met->Par[0].Val.V_INT = 128;
        Met->Par[1].Val.V_ENUM.value = 0;
        Met->Par[1].Val.V_ENUM.members = &PremiaEnumMCRNGs;
        Met->Par[2].Val.V_ENUM.value = 0;
        Met->Par[2].Val.V_ENUM.members = &PremiaEnumBool;
      }

    return OK;
  }

  PricingMethod MET(MC_Multilevel) =
  {
    "MC_Multilevel",
    {
      {"TimeStepNumber finest level", INT, {100}, ALLOW},
      {"RandomGenerator", ENUM, {100}, ALLOW},
      {"Use Importance Sampling", ENUM, {100}, ALLOW},
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CALC(MC_Multilevel),
    { {"Price", DOUBLE, {100}, FORBID},
      {"Std_dev", DOUBLE, {100}, FORBID},
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CHK_OPT(MC_Multilevel),
    CHK_mc,
    MET(Init)
  };
}
```