Help

```
extern "C" {
#include "nonpar1d_vol.h"
}

#include <fstream>
#include <vector>

extern "C" {

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2008+2) //The "#els
  static int CHK_OPT(AP_NONPAR_VARIANCESWAP)(void *Opt, void *Mod)
  {
    return NONACTIVE;
  }
  int CALC(AP_NONPAR_VARIANCESWAP)(void *Opt, void *Mod, PricingMethod *Met)
  {
    return AVAILABLE_IN_FULL_PREMIA;
  }
#else

  static int readvol(std::vector<double> *pstrikes, std::vector<double> *pivol,

  static int ap_nonpar_varswap(char *ivfname, double S0, double Strike, double T
  {
    // S0  is a forward price!!
    // Arrays :
    // replStrikes are percentages of forward price, read from file
    std::vector<double> replStrikes;
    //replOptions are BS vanillas prices for given implied volatility
    std::vector<double> replOptions;
    //implVolatil are implied volatilities, read from file
    std::vector<double> implVolatil;
    // replWeights are weights of each vanilla option in replicating portfolio
    std::vector<double> replWeights;
    // CallPuts are logical indicators to identify the type of the option
    std::vector<int> CallPuts;

    int flag;
    double kfirst;
```

```
double pvfactor = exp(-r * T);
double divid = 0.0;

int k, k0, res, replN = 0;
double optprice, optdelta, tstrike, tprice;

// get implied volatility and strikes
if (int err_code = readvol(&replStrikes, &implVolatil, &replN, ivfname))
  return err_code;

replOptions.resize(replN);
replWeights.resize(replN);
CallPuts.resize(replN);

tprice = 0.0;
tstrike = S0;
k = 0;
flag = 1;
//find a separator between call and puts
while ((k < replN) && (flag))
  {
    CallPuts[k] = (S0 <= replStrikes[k]);
    flag = !CallPuts[k];
    k++;
  }

k0 = k - 2;
for (; k < replN; k++)
  {
    CallPuts[k] = 1;
  }
//weights and prices for puts
tstrike = replStrikes[k0 + 1];
for (k = k0; k >= 0; k--)
  {
    replWeights[k] = -(replStrikes[k] - tstrike) / (replStrikes[k] * replStr
    res = pnl_cf_put_bs(S0 * pvfactor, replStrikes[k], T, r, divid, implVola
    if (res)
      {
        return 1;
      }
```

```
      replOptions[k] = optprice;
      tstrike = replStrikes[k];
      tprice += replOptions[k] * replWeights[k];
    }

  //weights and prices for calls
  tstrike = replStrikes[k0];
  for (k = k0 + 1; k < replN; k++)
    {
      replWeights[k] = (replStrikes[k] - tstrike) / (replStrikes[k] * replStri
      res = pnl_cf_call_bs(S0 * pvfactor, replStrikes[k], T, r, divid, implVol
      if (res)
        {
          return 1;
        }
      replOptions[k] = optprice;
      tstrike = replStrikes[k];
      tprice += replOptions[k] * replWeights[k];
    }

  //portfolio value
  tprice *= 2.0 / T; //*252.0/251.0;

  //fair strike of variance swap, in annual volatility points
  *fairval = sqrt(tprice / pvfactor) * 100;
  // strike in variance points
  kfirst = pvfactor * Strike * Strike;
  // price of var swap
  *ptprice = tprice * 10000 - kfirst;

  return OK;
}

//------------------------------------------------------------------------
static int readvol(std::vector<double> *pstrikes, std::vector<double> *pivol,
{
  std::ifstream fin(finname);

  int i;

  double str, vol;
```

```
  if (!(fin))
    {
      printf("Unable to open input File %s\ n", finname);
      return UNABLE_TO_OPEN_FILE;
    }

  fin >> i;

  *nn = i;

  for (i = 0; i < *nn; i++)
    {
      fin >> str >> vol;
      pstrikes->push_back(str);
      pivol->push_back(vol);
    }

  return 0;
}


int CALC(AP_NONPAR_VARIANCESWAP)(void *Opt, void *Mod, PricingMethod *Met)
{
  TYPEOPT *ptOpt = (TYPEOPT *)Opt;
  TYPEMOD *ptMod = (TYPEMOD *)Mod;
  double r, strike, spot;
  NumFunc_1 *p;

  r = log(1. + ptMod->R.Val.V_DOUBLE / 100.);
  p = ptOpt->PayOff.Val.V_NUMFUNC_1;
  strike = p->Par[0].Val.V_DOUBLE;
  spot = ptMod->S0.Val.V_DOUBLE;

  return ap_nonpar_varswap(
          ptMod->implied_volatility.Val.V_FILENAME, spot, strike,
          ptOpt->Maturity.Val.V_DATE, r,
          &(Met->Res[0].Val.V_DOUBLE)/*FAIR STRIKE*/,
          &(Met->Res[1].Val.V_DOUBLE)/*PRICE*/);

}
```

```
  static int CHK_OPT(AP_NONPAR_VARIANCESWAP)(void *Opt, void *Mod)
  {
    if ((strcmp(((Option *)Opt)->Name, "VarianceSwap") == 0))
      return OK;

    return WRONG;
  }

#endif //PremiaCurrentVersion
  static int MET(Init)(PricingMethod *Met, Option *Opt)
  {

    return OK;
  }

  PricingMethod MET(AP_NONPAR_VARIANCESWAP) =
  {
    "AP_NONPARAM_VARIANCESWAP_CARRLEE",
    {{" ", PREMIA_NULLTYPE, {0}, FORBID}},
    CALC(AP_NONPAR_VARIANCESWAP),
    { {"Fair strike, in annual volatility points", DOUBLE, {100}, FORBID},
      {"Price, in 10000 variance points", DOUBLE, {100}, FORBID},
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CHK_OPT(AP_NONPAR_VARIANCESWAP),
    CHK_ok ,
    MET(Init)
  } ;

  /*////////////////////////////////////*/
}
```