Help

```
extern "C" {
#include "
href../../../mod/nonpar1d/nonpar1d_vol/nonpar1d_vol_h_src.pdfnonpar1d_vol.h"
}
#include <
href../../../common/math/highdim_solver/highdim_vector_h_src.pdfvector>
#include <cmath>
#include <pnl/pnl_finance.h>
#include <stdio.h>
#include "pnl/pnl_vector.h"

extern "C" {

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


typedef struct option {
    double strike;
    double call_price;
    double put_price;
} option;

//compute volatility index in non-parametric Levy model, fn - file name with opt
//option strikes in file must be sorted
//s - asset price
//T - time to maturity
int levy_vol_index(const char* fn, double s, double T, double r,double *ptprice)
{
    const int max_options=50;
    option options[max_options];
```

```
    FILE* f=fopen(fn,"r");;
 if (f == NULL)
    {
        perror("Could not open file");
        return 0;
    }
    //Reading option data from file fn

    int n=0;
    int iatm=-1; // index of atm put
    //double min_dif=-1;
    //double k0=0;
////find k0 at-the-money strike price ////////////////
    while (!feof(f))
    {
        fscanf(f, "%lf %lf %lf\ n",&options[n].strike,&options[n].call_price,&op
        if (n>=1)
            if ((options[n-1].strike<=s) && (s<options[n].strike))
                iatm=n-1;
        /*
        double dif=fabs(options[n].call_price-options[n].put_price);
        if ((dif<=min_dif) or (min_dif==-1))
        {
            min_dif=dif;
            k0=options[n].strike;
        }*/
        n++;
    }
    double k0=options[iatm].strike; ////the at-the-money strike price
    double f_price=k0+exp(r*T)*(options[iatm].call_price-options[iatm].put_price

    double price_call[max_options],price_put[max_options],k_call[max_options],k_
    int n_call=0;
    int n_put=0;
    for (int i=0;i<=iatm;i++)
        if (options[i].put_price!=0)
        {
            price_put[n_put]=options[i].put_price;
            k_put[n_put]=options[i].strike;
            n_put++;
```

```
        }
    for (int i=iatm+1;i<n;i++)
        if (options[i].call_price!=0)
        {
            price_call[n_call]=options[i].call_price;
            k_call[n_call]=options[i].strike;
            n_call++;
        }
    double log_k0=log(k0);
    double sum1=price_call[0]/pow(k_call[0],2)*(k_call[0]-k_put[n_put-1]);
    for (int i=1;i<n_call;i++)
        sum1+=price_call[i]/pow(k_call[i],2)*(k_call[i]-k_call[i-1]);
    double sum2=price_put[0]/pow(k_put[0],2)*(k_put[1]-k_put[0]);
    for (int i=1;i<n_put;i++)
        sum2+=price_put[i]/pow(k_put[i],2)*(k_put[i]-k_put[i-1]);
    double expected=log_k0+f_price/k0-1-exp(r*T)*(sum1+sum2);
    sum1=price_call[0]*(1-log(k_call[0]))/pow(k_call[0],2)*(k_call[0]-k_put[n_pu
    for (int i=1;i<n_call;i++)
        sum1+=price_call[i]*(1-log(k_call[i]))/pow(k_call[i],2)*(k_call[i]-k_cal
    sum2=price_put[0]*(1-log(k_put[0]))/pow(k_put[0],2)*(k_put[1]-k_put[0]);
    for (int i=1;i<n_put;i++)
        sum2+=price_put[i]*(1-log(k_put[i]))/pow(k_put[i],2)*(k_put[i]-k_put[i-1
    double expected_quad=pow(log_k0,2)+2*log_k0*(f_price/k0-1)+2*exp(r*T)*(sum1+
    double var=expected_quad-pow(expected,2);

 fclose(f);

    *ptprice=sqrt(var/T)*100.0;

 return OK;
}

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

    r = log(1. + ptMod->R.Val.V_DOUBLE / 100.);
    p = ptOpt->PayOff.Val.V_NUMFUNC_1;
```

```
    spot = ptMod->S0_volindex.Val.V_DOUBLE;
    return levy_vol_index(ptMod->option_prices.Val.V_FILENAME, spot,
  ptOpt->Maturity.Val.V_DATE, r, &(Met->Res[0].Val.V_DOUBLE));


  }

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

    return WRONG;
  }

#endif //PremiaCurrentVersion

  static int MET(Init)(PricingMethod *Met, Option *Opt)
  {
    Met->HelpFilenameHint = "ap_nonparam_volatilityindex";
    return OK;
  }

  PricingMethod MET(AP_NONPAR_LEVY_VOLATILITYINDEX) =
  {
    "AP_NONPAR_LEVY_VOLATILITYINDEX",
    {{" ", PREMIA_NULLTYPE, {0}, FORBID}},
    CALC(AP_NONPAR_LEVY_VOLATILITYINDEX),
    { {"VolatilityIndex", DOUBLE, {100}, FORBID},
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CHK_OPT(AP_NONPAR_LEVY_VOLATILITYINDEX),
    CHK_ok ,
    MET(Init)
  } ;

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