Help

```
extern "C" {
#include  "
href../../../mod/jump1d/jump1d_stdg/jump1d_stdg_h_src.pdfjump1d_stdg.h"
}
#include "
href../../../common/math/levy_fd_swing_h_src.pdfmath/levy_fd_swing.h"

extern "C" {
#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2008+2) //The "#els
  static int CHK_OPT(FD_ImpExpSwing)(void *Opt, void *Mod)
  {
    return NONACTIVE;
  }
  int CALC(FD_ImpExpSwing)(void *Opt, void *Mod, PricingMethod *Met)
  {
    return AVAILABLE_IN_FULL_PREMIA;
  }
#else
  static int ImpExpSwing(int am, double S0, NumFunc_1  *p, double T, int Nd, dou
  {
    double price0, delta0;
    int flag_callput, flag_stdbarrier;
    double rebate = 0.;
    int Nu;/*Number of Call Exercise*/

    /*American Put choosen by default*/

    /*Construction of the model*/
    double gamma2 = 0.000000000000001;
    double delta = sqrt(gamma2);
    Merton_measure measure(mu, delta, lambda, sigma, dx);

    double K = p->Par[0].Val.V_DOUBLE;;
    double k = 3;
    double Al = log(2. / 3) + T * measure.espX1 - k * sqrt(T * measure.varX1);
    double Ar = log(2.) + r * T + k * sqrt(T * measure.varX1);
    if (Al < -30) Al = -30;
    if (Ar > 30) Ar = 30;
    int Nl = (int)ceil(-Al / dx);
```

```
  int Nr = (int)ceil(Ar / dx);
  int N = Nl + Nr;
  Al = -Nl * dx;
  Ar = Nr * dx;

  if ((p->Compute) == &Put)
    flag_callput = 2;
  else /*if ((p->Compute)==&Call)*/
    flag_callput = 1;

  flag_stdbarrier = 1;
  Nu = 0;
  /*Price Computation*/
  price2_swing(am, measure, flag_callput, flag_stdbarrier, r, divid, S0, K, re


  /*Price */
  *ptprice = price0;

  /*Delta */
  *ptdelta = delta0;

  return OK;
}

int CALC(FD_ImpExpSwing)(void *Opt, void *Mod, PricingMethod *Met)
{
  TYPEOPT *ptOpt = (TYPEOPT *)Opt;
  TYPEMOD *ptMod = (TYPEMOD *)Mod;
  double r, divid;

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


  return ImpExpSwing(ptOpt->EuOrAm.Val.V_BOOL, ptMod->S0.Val.V_PDOUBLE,
                     ptOpt->PayOff.Val.V_NUMFUNC_1, ptOpt->Maturity.Val.V_DATE
}

static int CHK_OPT(FD_ImpExpSwing)(void *Opt, void *Mod)
{
```

```
    Option *ptOpt = (Option *)Opt;
    TYPEOPT *opt = (TYPEOPT *)(ptOpt->TypeOpt);

    if ((opt->EuOrAm).Val.V_BOOL == AMER)
      return OK;

    return WRONG;
  }

#endif //PremiaCurrentVersion
  static int MET(Init)(PricingMethod *Met, Option *Opt)
  {
    static int first = 1;

    if (first)
      {
        Met->Par[0].Val.V_PDOUBLE = 0.01;
        Met->Par[1].Val.V_INT2 = 100;
        first = 0;
      }

    return OK;
  }

  PricingMethod MET(FD_ImpExpSwing) =
  {
    "FD_Swing_Jump",
    { {"Space Discretization Step", DOUBLE, {500}, ALLOW}, {"TimeStepNumber", IN
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CALC(FD_ImpExpSwing),
    {{"Price", DOUBLE, {100}, FORBID}, {"Delta", DOUBLE, {100}, FORBID}, {" ", P
    CHK_OPT(FD_ImpExpSwing),
    CHK_split,
    MET(Init)
  };

}
```