Help

```
extern "C" {
#include  "
href../../../mod/bs1d/bs1d_stdz/bs1d_stdz_h_src.pdfbs1d_stdz.h"
#include "
href../../../common/enums_h_src.pdfenums.h"
#include "
href../../../common/error_msg_h_src.pdferror_msg.h"
}
#include <cmath>
#include "pnl/pnl_integration.h"
#include "pnl/pnl_cdf.h"
#include "pnl/pnl_root.h"
#include "pnl/pnl_specfun.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2018+2) //The "#els
extern "C" {
static int CHK_OPT(AP_SPECTRAL_RISK_GMMB_BS)(void *Opt, void *Mod)
{
  return NONACTIVE;
}
int CALC(AP_SPECTRAL_RISK_GMMB_BS)(void *Opt, void *Mod, PricingMethod *Met)
{
  return AVAILABLE_IN_FULL_PREMIA;
}
}
#else

static double A0, x0, nu, sigma, kappa, r, maturity, alpha, risk_level;
static double divid,mu,me,m;
static int N;

static double binomial(double n, double k){
    if (k >= n) return 1;
    return pnl_sf_choose(n, k);
}

static double WhittakerM(double k, double m, double z){
    double x, y;
    if(z == 0){
```

```
        if (m > -0.5) return z;
        else return INFINITY;
    }
    x = -0.5 * z;
    y = 0.5 + m;
    return std::exp(x) * std::pow(z, y) * pnl_sf_hyperg_1F1(y - k, 1 + 2 * m, z)
}

static double WhittakerW(double k, double m, double z){
    double x, y, g;
    if (z == 0){
        g = fabs(m);
        if (g < 0.5) return z;
        else return INFINITY;
    }
    x = -0.5 * z;
    y = 0.5 + m;
    return std::exp(x) * std::pow(z, y) * pnl_sf_hyperg_U(y - k, 1 + 2 * m, z);
}

static double Pt (double s, double w)
{
  double mmu;
  double K = x0 * w;
  mmu = std::sqrt(nu * nu + 8 * s / (sigma * sigma)) / 2;
  return 4. / std::pow(sigma, 2) * tgamma(0.5+ mmu - kappa) / tgamma(1 + 2 * mmu
}

static double ft(int n, double t, double w){
  int k;
  double sm = 0;
  for (k = 0; k <= n; k++)
    sm += std::pow(-1, k) * binomial(n, k) * Pt((n + k) * std::log(2) / t, w);
  return sm * std::log(2) * pnl_fact(2 * n) / (pnl_fact(n) * pnl_fact(n - 1) * t
}

static double weight(double k){
  return std::pow(-1, N - k) * std::pow(k, N) / pnl_fact(k) / pnl_fact(N - k);
}

static double P(double t, double w){
```

```
  double f1 = 0;
  int k = 1;
  for (; k <= N; k++){
  f1 += weight(k) * ft(k, t, w);
  }
  return f1;
}

static double prob (double V){
  double B = (std::exp(-r * maturity) * alpha * A0 - V) / (A0);
  return P(maturity, B);
}

static double Zt (double s, double w){
  double mmu;
  double K;
  K = x0 * w;
  mmu = std::sqrt(nu * nu + 0.8e1 * s * std::pow(sigma, -0.2e1)) / 0.2e1;
  return(0.4e1 / x0 * std::pow(sigma, -0.2e1) * tgamma(0.1e1 / 0.2e1 + mmu - kap
}

static double ftZ(int n, double t, double w){
  int k;
  double sm = 0;
  for (k = 0; k <= n; k++){
  sm += std::pow(-1, k) * binomial(n, k) * Zt((n + k) * std::log(2) / t, w);
  }
  return sm * std::log(2) * pnl_fact(2 * n) / (pnl_fact(n) * pnl_fact(n - 1) * t
}

static double Z(double t, double w){
  double f1 = 0;
  int k;
  for (k = 1; k <= N; k++){
  f1 += +weight(k) * ftZ(k, t, w);
  }
  return f1;
}

int AP_GMMB_Spectral_VaR_CTE(double A0, double alpha, double maturity, double r,
{
```

```
double Tpx = 0.757;

double temp, target;

double t;

double left, right, B, xi;

nu = 2 * (mu - m - r) / std::pow(sigma, 2);
t = std::pow(sigma, 2)*maturity / 4;
x0 = std::pow(sigma, 2) / 4 / me;
kappa = (1 - nu) / 2;
target=(1-risk_level)/Tpx;

left = 0.0;
right = 100.0;
m = (left+right)/2;
temp=prob(m);

while(fabs(left - right) > 0.0000001){
if(temp>target){
left=m;
}else{
right=m;
}
m=(left+right)/2;
temp=prob(m);
}

xi=1-Tpx*P(maturity,exp(-r*maturity)*A0*alpha /(A0));
B = (std::exp(-r*maturity)*A0*alpha - m) / (A0);
    if (xi<risk_level)
    {
    *ptcte=std::exp(-r*maturity)*A0*alpha - Tpx*A0 / (1 - risk_level)*Z(maturity
    }
    else
    {
    *ptcte=(std::exp(-r*maturity)*A0*alpha-Tpx*A0/(1-xi)*Z(maturity,B))*(1-xi)/0
    }
```

```
    *ptvar=m;


  return  1;

}

extern "C" {
int  AP_SPECTRAL_RISK_GMMB_BS(double A0_main, double maturity_main, double r_mai
{
  double var,cte;

 N=7;//parameter of the spectral method.

 A0=A0_main;
 maturity= maturity_main;
 r=r_main;
 divid=divid_main;
 sigma=sigma_main;
 mu=mu_main-divid;
 me=me_main;
 m=m_main;
 risk_level=risk_level_main;
 alpha=alpha_main;

 AP_GMMB_Spectral_VaR_CTE(A0_main,alpha_main,maturity_main,r_main,sigma_main,ris

  *ptvar=var;
  *ptcte=cte;

  return OK;
}

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

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

```
   if ((ptOpt->Additionalearning1.Val.V_PDOUBLE>0)||(ptOpt->Additionalearning2.V
      {
        return UNTREATED_CASE;
      }
    else
      {
  return AP_SPECTRAL_RISK_GMMB_BS(ptMod->S0.Val.V_PDOUBLE,ptOpt->Maturity.Val.V_
   }
}

static int CHK_OPT(AP_SPECTRAL_RISK_GMMB_BS)(void *Opt, void *Mod)
{
  if ((strcmp(((Option *)Opt)->Name, "GMMB_RISK") == 0))
    return OK;
  else
    return WRONG;
}
}
#endif //PremiaCurrentVersion
extern "C" {
static int MET(Init)(PricingMethod *Met, Option *Opt)
{
  if (Met->init == 0)
    {
      Met->init = 1;
    }

  return OK;
}

PricingMethod MET(AP_SPECTRAL_RISK_GMMB_BS) =
{
  "AP_SPECTRAL_RISK_GMMB_BS",
  {
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CALC(AP_SPECTRAL_RISK_GMMB_BS),
  { {"VaR", DOUBLE, {100}, FORBID},
  {"CTE", DOUBLE, {100}, FORBID},
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
```

```
  },
  CHK_OPT(AP_SPECTRAL_RISK_GMMB_BS),
  CHK_ok,
  MET(Init)
};
}
```