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_GMDB_BS)(void *Opt, void *Mod)
{
  return NONACTIVE;
}
int CALC(AP_SPECTRAL_RISK_GMDB_BS)(void *Opt, void *Mod, PricingMethod *Met)
{
  return AVAILABLE_IN_FULL_PREMIA;
}
}
#else

static double A0, x0, nu, sigma, kappa, r, maturity, alpha, rollup_rate, risk_le

static double divid,mu,me,m;

static int n_, N, Tn;

static PnlVect *PQ;

static double binomial(double n, double k){
    if (k >= n) return 1;
    return pnl_fact(n) / (pnl_fact(k) * pnl_fact(n - k));
}
```

```
static double weight(double k){
  return std::pow(-1, N - k) * std::pow(k, N) / pnl_fact(k) / pnl_fact(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 / std::pow(sigma,2)) / 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 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 * 100 - 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;
```

```
}

static double SumP(double V)
{
    double sum = 0.;
    int i;
    for (i= 1; i < Tn+1; i++)
    {
      sum = sum + pnl_vect_get(PQ,i-1)*P(i/n_, (std::exp((rollup_rate-r) * i/ n_
    }
    return sum;
}

double Y(double V)
{
    return SumP(V) - (1 - alpha2) ;
}

static double derivative_Y(double V) {
    double dt = 0.0001;
    double right;
    double t1, t2;
    t1 = Y(V);
    right = V - dt;
    t2 = Y(right);
    double dev = (t1 - t2) / dt;
    V = right;
    return dev;
}

static double Multiplier(double V,double T)
{
return std::exp((rollup_rate-r)*T)*(A0*alpha)*P(T, (std::exp((rollup_rate-r) * T
}

static double CTE(double V)
{
  double sum = 0, i;
    for ( i= 1; i < Tn+1;i++)
    {
        sum = sum + pnl_vect_get(PQ,i-1)*Multiplier(V, i/n_);
```

```
    }
    return sum/(1-alpha2);
}

static double CTE2(double V)
{
    return CTE(V)*(1-alpha2)/(1-risk_level);
}

int AP_GMDB_Spectral_VaR_CTE(double A0, double alpha, double maturity, double r,
{

double t;

PnlVect *Tpx, *Qx;

alpha2 = risk_level;

double left, B, xi;

Tn = (int) maturity/n_;

    Tpx = pnl_vect_create_from_list(Tn+1, 1.0,0.98246,0.96348,0.94304,0.92113,0.
    Qx = pnl_vect_create_from_list(Tn+1, 0.01753,0.01932,0.02122,0.02323,0.02538

    PQ = pnl_vect_create(Tn+1);
    for (int j=0; j< Tn+1; j++)
    {
        pnl_vect_set(PQ, j, pnl_vect_get(Tpx, j)*pnl_vect_get(Qx, j)/n_);
    }

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;

    xi= 1. - SumP(0);
    if (risk_level < xi )
    {
        alpha2 = xi;
    }
```

```
    double right = 30;
    do
    {
        left = right;
        right = left - Y(left) / derivative_Y(left);
    } while (fabs(right - left) >= 0.00001);


    B = (std::exp(-r*maturity)*(A0*alpha) - m) / (A0);


if (xi<risk_level)
    {
    *ptcte=CTE(left);
    }
    else
    {
      *ptcte=CTE2(left);
    }

    *ptvar=left;

    pnl_vect_free (&Tpx);
    pnl_vect_free (&Qx);
    pnl_vect_free (&PQ);

  return  1;

}



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

 n_=1;//number of steps per year in the mortality table
 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;
 rollup_rate=rollup_rate_main;

  AP_GMDB_Spectral_VaR_CTE(A0_main,alpha_main,maturity_main,r_main,sigma_main,ri

  *ptvar=var;
  *ptcte=cte;

  return OK;
}

int CALC(AP_SPECTRAL_RISK_GMDB_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_GMDB_BS(ptMod->S0.Val.V_PDOUBLE,ptOpt->Maturity.Val.V_
   }
}

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

```
  if ((strcmp(((Option *)Opt)->Name, "GMDB_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_GMDB_BS) =
{
  "AP_SPECTRAL_RISK_GMDB_BS",
  {
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CALC(AP_SPECTRAL_RISK_GMDB_BS),
  { {"VaR", DOUBLE, {100}, FORBID},
  {"CTE", DOUBLE, {100}, FORBID},
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CHK_OPT(AP_SPECTRAL_RISK_GMDB_BS),
  CHK_ok,
  MET(Init)
};
}
```