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 "pnl/pnl_integration.h"
#include "pnl/pnl_cdf.h"
#include "pnl/pnl_root.h"

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

static double sigma, maturity, rollup_rate, A0, alpha, r,divid,me, mu, m, risk_l

static double tPx[11] = {1,0.98246,0.96348,0.94304,0.92113,0.89775,0.87275,0.846

static double Qx[11] = {0.01753,0.01932,0.02122,0.02323,0.02538,0.02785,0.03059,

static int n_;

static double Normal(double d)
{

  int which;
  double p;
  double q;
```

```
  double mean;
  double sd;
  int status;
  double bound;

  which=1;
  mean=0.0;
  sd=1.0;

  pnl_cdf_nor(&which,&p,&q,&d,&mean,&sd,&status,&bound);

  return p;
}

static double PQ(int i)
{
    int inx = (i - 1) / n_;
    double res = tPx[inx]*Qx[inx] / n_;
    return res;
}

static double p(double T,double z)
{
return exp(-pow((pow(sigma, 2)*T / 2 + log(z)), 2) / (2 * pow(sigma, 2)*T)) / sq
}

static double q(double T,double z)
{
return exp(-pow((pow(sigma, 2)*T + log(z)), 2) / (2 * pow(sigma, 2)*T)) / sqrt(2
}

static double quantile1(double T,double z)
{
return log(z) / sqrt(pow(sigma, 2)*T) + 0.5*sqrt(pow(sigma, 2)*T);
}

static double quantile2(double T,double z)
{
return log(z) / sqrt(pow(sigma, 2)*T) - 0.5*sqrt(pow(sigma, 2)*T);
}
```

```
static double quantile3(double T,double z)
{
return log(z) / sqrt(pow(sigma, 2)*T) + sqrt(pow(sigma, 2)*T);
}

static double quantile4(double T,double z)
{
return log(z) / sqrt(pow(sigma, 2)*T) - sqrt(pow(sigma, 2)*T);
}

static double aT(double T,double z)
{
  if ((Normal(quantile1(T,z)) - Normal(quantile2(T,z))) / (p(T,z)*pow(sigma, 2))
    {
        return T*(z-1)/log(z);
    }
    else
    {
        return (Normal(quantile1(T,z)) - Normal(quantile2(T,z))) / (p(T,z)*pow(s
    }
}

static double bT(double T,double z)
{
  if ((Normal(quantile3(T,z)) - Normal(quantile4(T,z))) / (q(T,z)*pow(sigma, 2))
    {
        return 0;
    }
    else
    {
      return (Normal(quantile3(T,z)) - Normal(quantile4(T,z))) / (q(T,z)*pow(sig
    }
}

static double square_sigma(double T,double z)
{
  if ((2 * (bT(T,z) / aT(T,z) - 1 - z) / (aT(T,z)*pow(sigma, 2)))<=0.0)
    {
        return 0.000000000000001;// This value is set to avoid zero in the domin
    }
    else
```

```
    {
return log(2 * (bT(T,z) / aT(T,z) - 1 - z) / (aT(T,z)*pow(sigma, 2))) / T;
    }
}

static double mean_mu(double T,double z)
{
    if (aT(T,z)<=0)
    {
        return 1.;
    }
    else
    {
        return 1 - 2 * log(aT(T,z)) / (T*square_sigma(T,z));
    }
}

static double B(double V,double T)
{
  double res = (exp((rollup_rate-r)*T)*(A0*alpha) - V) / (A0);
    return res;
}

static double inte_f(double V, double T, double z)
{
  double percentile = (log((B(V,T) - z) / me) + mean_mu(T,z)*square_sigma(T,z)*T

return Normal(percentile);
}

double w(double T)
{
  return exp((rollup_rate-r)*T)*A0*alpha/(A0);
}

static double inte_f2(double T, double z)
{
  double percentile = (log((w(T) - z) / me) + mean_mu(T,z)*square_sigma(T,z)*T /

return Normal(percentile);
}
```

```
static double DPhi(double T,double z)
{
        double temp = log(z) - (mu - m - r)*T;
double s = -pow(temp, 2) / (2 * pow(sigma, 2) * T);
        return exp(s) / (sqrt(2 * M_PI*T)*sigma*z);
}

static double Q(double V, double T,double z)
{
  return inte_f(V, T,z)*DPhi(T,z);
}

static double Q2(double T, double z)
{
return inte_f2(T, z)*DPhi(T,z);
}

static double integral_Q2(double T)
{
double sum = 0, z;
double end = w(T);
double res;
for (z = 0; z<= end; z = z + 0.0001)
{
res = Q2(T, z);
if (isnan(res)) continue;
sum = sum + res*0.0001;
}
return sum;
}

static double inte_f_(double V, double T, double z)
{
  double c1=(mean_mu(T,z)-1)*square_sigma(T,z)*T;
        double multi=exp(-c1/2);
        double percentile1 = (log((B(V,T) - z) / me) + (mean_mu(T,z)-2)*square_s
        return multi*Normal(percentile1);
}

static double Q_(double V,double T,double z)
```

```
{
  return (z*inte_f(V,T,z)+me*inte_f_(V,T,z))*DPhi(T,z);
}

static double c1(double T, double z)
{
  return mean_mu(T,z)*square_sigma(T,z)*T / 2;
}

static double c2(double z)
{
    if (square_sigma(T,z)<=0.0)
    {
        return 0;
    }
    else
    {
        return sqrt(square_sigma(T,z)*T);
    }
}

static double Int_f1(double V,double T,double z)
{
  double percentile1=(log((B(V,T)- z) / me) + c1(T,z)) / c2(z);
    if (square_sigma(T,z)>0)
    {
        return Normal(percentile1);
    }
    else return 0;
}

static double DP(double z)
{
    double temp = log(z) - (mu - m - r)*T;
    double s = -pow(temp, 2) / (2 * pow(sigma, 2) * T);
   return exp(s) / (sqrt(2 * M_PI*T)*sigma*z);
}

static double function1(double u, void *p)
{
    double *tmp;
```

```
    tmp = (double *)p;
    if (isnan(Int_f1(*tmp, T,u)*DP(u)))
    {
        return 0;
    }
    else
    {
      return Int_f1(*tmp, T,u)*DP(u);
    }
}

static double Int_Q1(double V,double T)
{
    double h, result,abserr;
    int neval;
    PnlFunc func;
    func.F = function1;
    func.params = &V;
    h=B(V,T);
    if (h<=0)
    {
        return 0;
    }
    else
    {
        pnl_integration_qag(&func,0.0,h,0.00001,0.0001,0,&result,&abserr,&neval)
             return result;
    }
}

static double Int_f2(double V,double z)
{
  double percentile2=(log((rho*z-(exp(-r*T)*rho*(A0*alpha)*exp(rollup_rate*T)+V)
    return Normal(percentile2);
}

static double function2(double u, void *p)
{
    double *tmp;
    tmp = (double *)p;
    if (isnan(Int_f2(*tmp, u)*DP(u))) {
```

```
        return 0;
    }
    else
    {
    return Int_f2(*tmp, u)*DP(u);
    }
}

static double Int_Q2(double V)
{
    double l, h, result,abserr;
    int neval;
    PnlFunc func;
    func.F = function2;
    func.params = &V;
    l= exp((-r)*T)*(A0*alpha)*exp(rollup_rate*T)/(A0)+V/(rho*(A0));
    h= exp((-r)*T)*(rho*(A0*alpha)*exp(rollup_rate*T)+C)/(rho*(A0));
    pnl_integration_qag(&func,l,h,0.0000001,0.0001,0,&result,&abserr,&neval);
    return result;
}

static double Int_f3(double V,double z)
{
    double percentile3;
    if ((exp(-r*T)*C-V)/(A0)/me<=0)
    {
      percentile3= c1(T,z) / c2(z);
    }
    else percentile3=(log((exp(-r*T)*C-V)/(A0)/me) + c1(T,z)) / c2(z);
    return Normal(percentile3);
}

static double function3(double u, void *p)
{
    double *tmp;
    tmp = (double *)p;
    if (isnan(Int_f3(*tmp, u)*DP(u))) {
        return 0;
    }
    else
    {
```

```
        return Int_f3(*tmp, u)*DP(u);
    }
}

static double Int_Q3(double V)
{
    double l, result,abserr;
    int neval;
    PnlFunc func;
    func.F = function3;
    func.params = &V;
    l= exp(-r*T)*(rho*(A0*alpha)*exp(rollup_rate*T)+C)/(rho*(A0));
    pnl_integration_qag(&func,l, 100, 0.00001,0.0001,0,&result,&abserr,&neval);
    return result;
}

static double Int_Q(double V,double T)
{
  if (rho>0)
    {return Int_Q1(V,T)+Int_Q2(V)+Int_Q3(V);}
  else
    {return Int_Q1(V,T);}
}

static double integral_Q(double x, void *p)
{
    double sum=0;
    int i;
    for (i=1; i<=maturity; i++)
    {
        T=1.0*i;
        sum = sum + tPx[i-1]*Qx[i-1]*Int_Q(x,T);
 }
    return sum - (1 - alpha2);
}

static void bisection(double *var)
{
    double x1, x2, r, tol;
    PnlFunc func;
```

```
    x1 = -1.;
    x2 = 100.;
    tol= 0.00001;
    func.F = integral_Q;
    func.params = NULL;
    r=pnl_root_brent(&func, x1, x2, &tol);
    *var =r;
}

static double g1(double V, double T,double z)
{
    double percentile1;
    double multi=exp(-(mean_mu(T,z)-1)*square_sigma(T,z)*T/2.);
    if (square_sigma(T,z)>0)
    {
      percentile1 = (log((B(V,T) - z) / me) + (mean_mu(T,z)-2)*square_sigma(T,z)
        return multi*Normal(percentile1);
    }
    else return 0;
}

static double z1(double u, void *p)
{
    double *tmp;
    tmp = (double *)p;
    return ((exp((rollup_rate-r)*T)*(A0*alpha)-(A0)*u)*Int_f1(*tmp, T,u)- (A0)*m
}

static double Int_Z1(double V)
{
    double h, result,abserr;
    int neval;
    PnlFunc func;
    func.F = z1;
    func.params = &V;
    h= (exp((rollup_rate-r)*T)*(A0*alpha)-V)/(A0);
    if (h<=0)
    {
        return 0;
    }
    else
```

```
    {
        pnl_integration_qag(&func,0.0,h,0.00001,0.0001,0,&result,&abserr,&neval)
        return result;
    }
}

static double g2(double V, double z)
{
  double multi=exp(-(mean_mu(T,z)-1)*square_sigma(T,z)*T/2);
  double percentile1 = (log((rho*z-(exp(-r*T)*rho*(A0*alpha)*exp(rollup_rate*T)+
    return multi*Normal(percentile1);
}

static double z2(double u, void *p)
{
    double *tmp;
    tmp = (double *)p;
    /*if (u>=10) {
     return 0;
     }
     else
     {
     return (rho*(F0*u-exp(-r*T)*G)*Int_f2(*tmp, u)- F0*me*g2(*tmp, u))*DP(*tmp,
     }*/
    return (rho*((A0)*u-exp((rollup_rate-r)*T)*(A0*alpha))*Int_f2(*tmp, u)- (A0)
}

static double Int_Z2(double V)
{
    double l, h, result,abserr;
    int neval;
    PnlFunc func;
    func.F = z2;
    func.params = &V;
    l= exp((rollup_rate-r)*T)*(A0*alpha)/(A0)+V/(rho*(A0));
    h= exp(-r*T)*(rho*(A0*alpha)*exp(rollup_rate*T)+C)/(rho*(A0));
    pnl_integration_qag(&func,l, h, 0.001,0.01,0,&result,&abserr,&neval);
    return result;
}

static double g3(double V, double z)
```

```
{
  double multi=exp(-(mean_mu(T,z)-1)*square_sigma(T,z)*T/2);
  double percentile1 = (log((exp(-r*T)*C-V)/(A0)/me) + (mean_mu(T,z)-2)*square_s
    return multi*Normal(percentile1);
}

static double z3(double u, void *p)
{
    double *tmp;
    tmp = (double *)p;
    if (u>=1200) {
        return 0;
    }
    else
    {
      return ((exp(-r*T)*C)*Int_f3(*tmp, u)- (A0)*me*g3(*tmp, u))*DP(u);
    }
}

static double Int_Z3(double V)
{
    double l, result,abserr;
    int neval;
    PnlFunc func;
    func.F = z3;
    func.params = &V;
    l=exp(-r*T)*(rho*(A0*alpha)*exp(rollup_rate*T)+C)/(rho*(A0));
    pnl_integration_qag(&func,l, PNL_POSINF, 0.001,0.01,0,&result,&abserr,&neval
    return result;
}

static double CTE(double V)
{
    double sum;
    int i;
    sum=0;
    for (i=1; i<=maturity; i++)
    {
        T=i;
if (rho>0)
  {sum=sum+ tPx[i-1]*Qx[i-1]*(Int_Z1(V)+ Int_Z2(V)+Int_Z3(V));}
```

```
else
  {sum=sum+ tPx[i-1]*Qx[i-1]*Int_Z1(V);}
    }
    return sum/(1-alpha2);
}

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

double  xi(double end)
{
        double sum= 0, i;
        for ( i= 1; i <= end;i=i+1)
{
sum = sum + PQ(i)*integral_Q2(1.0*i/n_);
       }
        return 1-sum;
}

int AP_GMDB_AE_Lognormal_VaR_CTE(double A0, double alpha, double maturity, doubl
{

 double VaR;

 T=maturity;

 alpha2 = xi(10.0*n_);

        if (risk_level > alpha2) alpha2=risk_level;

bisection(&VaR);

        *ptvar=VaR;

        if(risk_level == alpha2) *ptcte=CTE(VaR);

        if(risk_level < alpha2) *ptcte=CTE2(VaR);

return 0;
```

```
}


extern "C" {
int  AP_LOGNORMAL_RISK_GMDB_BS(double A0_main, double maturity_main, double r_ma
{
  double var,cte;

 n_=1;

 A0=A0_main;
 maturity= maturity_main;
 r=r_main;
 divid=divid_main;
 sigma=sigma_main;
 mu=mu_main-divid;
 rollup_rate=rollup_rate_main;
 me=me_main;
 m=m_main;
 rho=rho_main;
 risk_level=risk_level_main;
 C_main*=A0_main;
 C=C_main;
 alpha=alpha_main;

  AP_GMDB_AE_Lognormal_VaR_CTE(A0_main,alpha_main,maturity_main,r_main,sigma_mai

  *ptvar=var;
  *ptcte=cte;

  return OK;
}

int CALC(AP_LOGNORMAL_RISK_GMDB_BS)(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 AP_LOGNORMAL_RISK_GMDB_BS(ptMod->S0.Val.V_PDOUBLE,ptOpt->Maturity.Val.V
}

static int CHK_OPT(AP_LOGNORMAL_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_LOGNORMAL_RISK_GMDB_BS) =
{
  "AP_LOGNORMAL_RISK_GMDB_BS",
  {
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CALC(AP_LOGNORMAL_RISK_GMDB_BS),
  { {"VaR", DOUBLE, {100}, FORBID},
  {"CTE", DOUBLE, {100}, FORBID},
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CHK_OPT(AP_LOGNORMAL_RISK_GMDB_BS),
  CHK_ok,
  MET(Init)
};
```

```
}
```