Help

```
#include  "bs1d_stda.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2014+2) //The "#els
static int CHK_OPT(FD_GLWB_BS)(void *Opt, void *Mod)
{
  return NONACTIVE;
}
int CALC(FD_GLWB_BS)(void *Opt, void *Mod, PricingMethod *Met)
{
  return AVAILABLE_IN_FULL_PREMIA;
}
#else

static char *infilename;
static double r, divid, s0, t, sigma, theta_sc;
static double alpha_m;
static int n_passi, Ns, N;
static double gamma_i;
static int ratchet;
static double lm_insurance[100];
static double lm_age[100];
static double Rt[100];
static double Mt[100];
static double Kt[100];
static double Bt[100];
static double Gt[100];

static double rtsec(double (*func)(double), double x1, double x2, double xacc)
{
  int j;
  double fl, f, dx, swap, xl, rts;

  fl = (*func)(x1);
  f = (*func)(x2);
  if (fabs(fl) < fabs(f))
    {
      rts = x1;
      xl = x2;
      swap = fl;
```

```
      fl = f;
      f = swap;
    }
  else
    {
      xl = x1;
      rts = x2;
    }
  for (j = 1; j <= 30; j++)
    {
      dx = (xl - rts) * f / (f - fl);
      xl = rts;
      fl = f;
      rts += dx;
      f = (*func)(rts);
      if (fabs(dx) < xacc || f == 0.0) return rts;
    }
  printf("Maximum number of iterations exceeded in rtsec\ n");
  return 0.0;
}
#undef MAXIT

////////////////////////////////////////////////////////////////////////////////



/*Compute GLWB Price*/
static double compute_price(double alpha_g)
{
  double dt;
  int Index, PriceIndex;
  double k, vv, l, h, z, alpha, beta, gamma, y, alpha1, beta1, gamma1, bound1, b
  double  *Obst, *A, *B, *C, *A1, *B1, *C1, *P, *SP, *vect_s, *P_Old;
  double PRECISION = 0.0000000001;
  int i, flag_monit, i_s;
  double price, new_P, s_value;
  double val, val1;
  double A_IN, new_A;
  int anno;
  double worst_g;
  int current_mt_year, current_mt_year1;
```

```
double s_value1;

A_IN = s0;

A = malloc((N + 1) * sizeof(double));
B = malloc((N + 1) * sizeof(double));
C = malloc((N + 1) * sizeof(double));
A1 = malloc((N + 1) * sizeof(double));
B1 = malloc((N + 1) * sizeof(double));
C1 = malloc((N + 1) * sizeof(double));
P = malloc((N + 1) * sizeof(double));
P_Old = malloc((N + 1) * sizeof(double));
Obst = malloc((N + 1) * sizeof(double));
SP = malloc((N + 1) * sizeof(double));
vect_s = malloc((N + 1) * sizeof(double));

dt = t / (double)Ns;

/*Time Step*/
k = dt;

/*Space Localisation*/
vv = 0.5 * SQR(sigma);
z = (r - divid - alpha_g - alpha_m) - vv;
l = sigma * sqrt(t) * sqrt(log(1.0 / PRECISION)) + fabs(z * t);

/*Space Step*/
h = 2.0 * l / (double)N;

/*Peclet Condition-Coefficient of diffusion augmented */
if ((h * fabs(z)) <= vv)
  upwind_alphacoef = 0.5;
else
  {
    if (z > 0.) upwind_alphacoef = 0.0;
    else  upwind_alphacoef = 1.0;
  }
vv -= z * h * (upwind_alphacoef - 0.5);

/*Lhs Factor of theta_sc-schema*/
alpha = theta_sc * k * (-vv / (h * h) + z / (2.0 * h));
```

```
beta = 1.0 + k * theta_sc * (r + 2.*vv / (h * h));
gamma = k * theta_sc * (-vv / (h * h) - z / (2.0 * h));

for (PriceIndex = 1; PriceIndex <= N - 1; PriceIndex++)
  {
    A[PriceIndex] = alpha;
    B[PriceIndex] = beta;
    C[PriceIndex] = gamma;
  }

//Neuman boundary conditions
B[1] = beta + alpha;
B[N - 1] = beta + gamma;

/*Rhs Factor of theta_sc-schema*/
alpha1 = k * (1.0 - theta_sc) * (vv / (h * h) - z / (2.0 * h));
beta1 = 1.0 - k * (1.0 - theta_sc) * (r + 2.*vv / (h * h));
gamma1 = k * (1.0 - theta_sc) * (vv / (h * h) + z / (2.0 * h));

for (PriceIndex = 1; PriceIndex <= N - 1; PriceIndex++)
  {
    A1[PriceIndex] = alpha1;
    B1[PriceIndex] = beta1;
    C1[PriceIndex] = gamma1;
  }

B1[1] = beta1 + alpha1;
B1[N - 1] = beta1 + gamma1;

/*Set Gauss*/
for (PriceIndex = N - 2; PriceIndex >= 1; PriceIndex--)
  B[PriceIndex] = B[PriceIndex] - C[PriceIndex] * A[PriceIndex + 1] / B[PriceI
for (PriceIndex = 1; PriceIndex < N; PriceIndex++)
  A[PriceIndex] = A[PriceIndex] / B[PriceIndex];
for (PriceIndex = 1; PriceIndex < N - 1; PriceIndex++)
  C[PriceIndex] = C[PriceIndex] / B[PriceIndex + 1];

/*Terminal Values*/
y = log(s0);
for (PriceIndex = 0; PriceIndex <= N; PriceIndex++)
  {
```

```
    vect_s[PriceIndex] = exp(y - l + (double)PriceIndex * h);
    P[PriceIndex] = 0.;
  }

bound1 = 0.;
bound2 = 0.;

anno = (int)t;

/*Dynamic Programming*/
for (i = Ns - 1; i >= 0; i--)
  {
    current_mt_year1 = (int)((i + 1) * k);
    current_mt_year = (int)((i) * k);
    if ((((i) % n_passi) == 0) && (i != Ns) && (i > 0))
      {
        flag_monit = 1;
        anno = anno - 1;
      }
    else flag_monit = 0;

    if (flag_monit)
      {

        for (PriceIndex = 1; PriceIndex < N; PriceIndex++)
          P_Old[PriceIndex] = P[PriceIndex];

        //Ratchet
        if ((ratchet))
          {

            for (PriceIndex = 1; PriceIndex < N; PriceIndex++)
              {
                new_A = MAX(vect_s[PriceIndex], A_IN);
                s_value = A_IN / new_A * vect_s[PriceIndex];
                i_s = 0;
                while (vect_s[i_s] < s_value) i_s++;

                if (i_s == 0)
                  new_P = P_Old[1];
                else
```

```
            {
              val = P_Old[i_s];
              val1 = P_Old[i_s - 1];
              new_P = val + (val - val1) * (s_value - vect_s[i_s]) / (ve
            }
          P_Old[PriceIndex] = new_A / A_IN * new_P;
        }
    }

  for (PriceIndex = 1; PriceIndex < N; PriceIndex++)
    {
      worst_g = -100.;

      if (fabs(gamma_i) < 0.000001) //Bonus Event
        {
          new_A = A_IN * (1 + Bt[anno]);
          s_value = A_IN / new_A * vect_s[PriceIndex];
          i_s = 0;
          while ((vect_s[i_s] < s_value) && (i_s < N))  i_s++;

          if (i_s == 0)
            new_P = P_Old[1];
          else
            {
              val = P_Old[i_s];
              val1 = P_Old[i_s - 1];
              new_P = val + (val - val1) * (s_value - vect_s[i_s]) / (ve
            }
          worst_g = MAX(worst_g, new_A / A_IN * new_P);

        }
      if (gamma_i <= 1) //Withdrawal not exceeding contract amount
        {
          s_value = MAX(vect_s[PriceIndex] - gamma_i * Gt[anno] * A_IN,
          i_s = 0;
          while ((vect_s[i_s] < s_value) && (i_s < N)) i_s++;

          if (i_s == 0)
            new_P = P_Old[1];
          else if (i_s >= N)
            new_P = P_Old[N - 1];
```

```
            else
              {
                val = P_Old[i_s];
                val1 = P_Old[i_s - 1];
                new_P = val + (val - val1) * (s_value - vect_s[i_s]) / (ve
              }
            worst_g = MAX(worst_g, new_P + Rt[anno] * gamma_i * Gt[anno] *

          }

        else if (gamma_i > 1.) //Partial or full surrender
          //gamma_i=2 is full surrender
          {

            new_A = A_IN * (2 - gamma_i);
            s_value1 = MAX(vect_s[PriceIndex] - Gt[anno] * A_IN, 0.);
            s_value = s_value1 * (2 - gamma_i);

            i_s = 0;
            while ((vect_s[i_s] < s_value) && (i_s < N))  i_s++;

            if (i_s == 0)
              new_P = P_Old[1];
            else
              {
                val = P_Old[i_s];
                val1 = P_Old[i_s - 1];
                new_P = val + (val - val1) * (s_value - vect_s[i_s]) / (ve
              }

            worst_g = MAX(worst_g, new_A / A_IN * new_P + Rt[anno] * (Gt[a


          }//end j

        P[PriceIndex] = worst_g;

      }//End PriceIndex

  }//End Monitoring
```

```
    //BS-Cycle
    SP[1] = B1[1] * P[1] + C1[1] * P[2] + A1[1] * bound1 - alpha * bound1;
    for (PriceIndex = 2; PriceIndex < N - 1; PriceIndex++)
      SP[PriceIndex] = A1[PriceIndex] * P[PriceIndex - 1] +
                       B1[PriceIndex] * P[PriceIndex] +
                       C1[PriceIndex] * P[PriceIndex + 1];
    SP[N - 1] = A1[N - 1] * P[N - 2] + B1[N - 1] * P[N - 1] + C1[N - 1] * boun

    //Add M(t)S
    for (PriceIndex = 1; PriceIndex < N; PriceIndex++)
      {
        SP[PriceIndex] += vect_s[PriceIndex] * k * ((1 - theta_sc) * (Mt[curre
      }

    /*Solve the system*/
    for (PriceIndex = N - 2; PriceIndex >= 1; PriceIndex--)
      SP[PriceIndex] = SP[PriceIndex] - C[PriceIndex] * SP[PriceIndex + 1];

    P[1] = SP[1] / B[1];
    for (PriceIndex = 2; PriceIndex < N; PriceIndex++)
      P[PriceIndex] = SP[PriceIndex] / B[PriceIndex] - A[PriceIndex] * P[Price

  }//End time cicle i

Index = (int) floor((double)N / 2.0);
price = P[Index] - s0;


/*Memory Desallocation*/
free(Obst);
free(A);
free(B);
free(C);
free(A1);
free(B1);
free(C1);
free(P);
free(SP);
free(vect_s);
free(P_Old);
```

```
  /*Price*/
  return price;
}


static int FD_glwb_bs(double s, double t_fixed, double r_fixed, double divid_fix
{
  int i,j, iter;
  double val, sum, sum1;
  double pr1, pr2, pracc;
  FILE *f_in;

  N = Nspace;

  n_passi = n_step_for_year;
  Ns = (int)(t_fixed * n_step_for_year);
  theta_sc = theta;

  t = t_fixed;
  s0 = s;
  r = r_fixed;
  divid = divid_fixed;
  sigma = sigma_fixed;
  alpha_m = alpha_m_fixed;
  ratchet = ratchet_fixed;
  gamma_i = gamma_i_fixed;
  f_in = fopen(infilename, "rb");


  for (i = 0; i <= 60; i++)
    {
      fscanf(f_in, "%lf\ n", &val);
      lm_insurance[i] = val;
      lm_age[i] = 65. + i;
    }

  sum = 1.;
  Mt[0] = lm_insurance[0];
  for (i = 1; i <= 60; i++)
    {
      sum *= (1. - lm_insurance[i - 1]);
```

```
      Mt[i] = lm_insurance[i] * sum;
    }

  Rt[0] = 1.;
  sum1 = 0;
  for (i = 1; i <= 60; i++)
    {
      sum1 += Mt[i - 1];
      Rt[i] = 1. - sum1;
    }

  //Surrender fee Kt
  for (i = 0; i <= 60; i++)
    {
  j=0;
  while((pnl_vect_get(timesKt_fixed,j)<(double)i)&&(j<=4))
j++;
  Kt[i] = pnl_vect_get(Kt_fixed,j);
    }

  for (i = 1; i <= 60; i++)
    Bt[i] = B_i_fixed;

  for (i = 1; i <= 60; i++)
    Gt[i] = Gt_i_fixed;

  iter = 1;
  if (iter == 1)
    {
      pracc = 1.e-8;
      pr1 = 0.;
      pr2 = 0.1;
      val = rtsec(compute_price, pr1, pr2, pracc);
    }

  /*Price*/
  *ptprice = val;

  fclose(f_in);

  return OK;
```

```
}

int CALC(FD_GLWB_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.);
  infilename = ptOpt->MortalityData.Val.V_FILENAME;

  return FD_glwb_bs(ptMod->S0.Val.V_PDOUBLE, ptOpt->Maturity.Val.V_DATE - ptMod-
}

static int CHK_OPT(FD_GLWB_BS)(void *Opt, void *Mod)
{
  if ((strcmp(((Option *)Opt)->Name, "GLWB") == 0))
    return OK;
  else
    return WRONG;
}

#endif //PremiaCurrentVersion

static int MET(Init)(PricingMethod *Met, Option *Opt)
{
  if (Met->init == 0)
    {
      Met->init = 1;
      Met->HelpFilenameHint = "fd_glwb_bs";
      Met->Par[0].Val.V_INT2 = 40;
      Met->Par[1].Val.V_INT2 = 2400;
      Met->Par[2].Val.V_RGDOUBLE = 0.5;
    }

  return OK;
}

PricingMethod MET(FD_GLWB_BS) =
{
```

```
  "FD_GLWB_BS",
  { {"Timestep_for_year", INT2, {100}, ALLOW}, {"SpaceStepNumber", INT2, {100},
    {"Theta", RGDOUBLE051, {100}, ALLOW},
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CALC(FD_GLWB_BS),
  { {"Fair fee", DOUBLE, {100}, FORBID},
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CHK_OPT(FD_GLWB_BS),
  CHK_split,
  MET(Init)
};
```