Help

```
#include <stdlib.h>
#include  "mer1d_std.h"
#include "enums.h"


#if defined(PremiaCurrentVersion) && PremiaCurrentVersion <
     (2007+2) //The "#else" part of the code will be freely av
    ailable after the (year of creation of this file + 2)
static int CHK_OPT(FD_AndersenAndreasen)(void *Opt, void *
    Mod)
{
  return NONACTIVE;
}
int CALC(FD_AndersenAndreasen)(void *Opt,void *Mod,Pricing
    Method *Met)
{
  return AVAILABLE_IN_FULL_PREMIA;
}
#else

static int asym_1d(int am,PARAM p, DENSITY g,MESH m, WEIGHT
     w, IMESH Im,NumFunc_1  *p_func, int bound,double *pt
    price, double *ptdelta)
{
  int j,i,ii;
  unsigned long n2;
  double dum,*data,*sol_a,*sol_b,*p1,*p2,*p3,*tnoto,*bounda
    ry,*ftg,*fftg,*Obst;

  n2 = m.N<<1;
  /* vector allocation */
  sol_a = malloc((m.N+1)*sizeof(double));
  if (sol_a==NULL) return MEMORY_ALLOCATION_FAILURE;
  memset(sol_a,0,(m.N+1)*sizeof(double));
  Obst= malloc((m.N+1)*sizeof(double));
  if (Obst==NULL) return MEMORY_ALLOCATION_FAILURE;
  memset(Obst,0,(m.N+1)*sizeof(double));
  sol_b = malloc((m.N+1)*sizeof(double));
  if (sol_b==NULL) return MEMORY_ALLOCATION_FAILURE;
  memset(sol_b,0,(m.N+1)*sizeof(double));
```

```
tnoto = malloc((n2+1)*sizeof(double));
if (tnoto==NULL) return MEMORY_ALLOCATION_FAILURE;
memset(tnoto,0,(n2+1)*sizeof(double));
p1= malloc((m.N+1)*sizeof(double));
if (p1==NULL) return MEMORY_ALLOCATION_FAILURE;
memset(p1,0,(m.N+1)*sizeof(double));
p2= malloc((m.N+1)*sizeof(double));
if (p2==NULL) return MEMORY_ALLOCATION_FAILURE;
memset(p2,0,(m.N+1)*sizeof(double));
p3= malloc((m.N+1)*sizeof(double));
if (p3==NULL) return MEMORY_ALLOCATION_FAILURE;
memset(p3,0,(m.N+1)*sizeof(double));
data = malloc((m.N+3)*sizeof(double));
if (data==NULL) return MEMORY_ALLOCATION_FAILURE;
memset(data,0,(m.N+3)*sizeof(double));
boundary = malloc((m.N)*sizeof(double));
if (boundary==NULL) return MEMORY_ALLOCATION_FAILURE;
memset(boundary,0,(m.N)*sizeof(double));
fftg = malloc((n2+1)*sizeof(double));
if (fftg==NULL) return MEMORY_ALLOCATION_FAILURE;
memset(fftg,0,(n2+1)*sizeof(double));
ftg = malloc((m.N+3)*sizeof(double));
if (ftg==NULL) return MEMORY_ALLOCATION_FAILURE;
memset(ftg,0,(m.N+3)*sizeof(double));
g.d = malloc((m.N+1)*sizeof(double));
if (g.d==NULL) return MEMORY_ALLOCATION_FAILURE;
for (j=1;j<=m.N;j++) ftg[j]=(g.d)[j]=1.0/(sqrt(2.0*M_PI*
  g.par2))*exp(-SQR(g.zmin+(j-1)*m.h-g.par1)/(2.0*g.par2));

/* FFT of the density function */
drealft(ftg,m.N,1);

/*Terminal Values*/
for (j=1;j<=m.N;j++) {
  boundary[j-1]=sol_a[j] = (p_func->Compute)(p_func->Par,
  exp(m.xmin+(j-1)*m.h));
  Obst[j]=sol_a[j];
}

/* boundary */
set_boundaryAA(bound,m,p,Im,boundary,boundary);
```

```
/* "Probabilities" associated to points */
for (j=1;j<=Im.min;j++) {
  p1[j]= 0.;
  p2[j]= 1.0;
  p3[j] = 0.;
}
for (j=Im.min;j<=Im.max;j++){
  p1[j]= -m.k/2.0*w.p1;
  p2[j]= 1.0+m.k/2.0*w.p2;
  p3[j] = -m.k/2.0*w.p3;
  /*
      p1[j]= -m.k*w.p1;
      p2[j]= 1.0+m.k*w.p2;
      p3[j] = -m.k*w.p3;
  */
}
for (j=Im.max+1;j<=m.N;j++){
  p1[j]= 0.;
  p2[j]= 1.0;
  p3[j] = 0.;
}
/* Finite Difference Cycle */
for (i=1;i<=m.M;i++){

  /* step 1 */
  for (j=1;j<=Im.min;j++){ data[j]=boundary[j-1];}/* boun
  dary */
  for (j=Im.max+1;j<=m.N;j++){ data[j]=boundary[j-1];}/*
  boundary */
  for (j=Im.min;j<Im.max;j++){
    data[j] = sol_a[j];
  }

  /* okcorrel=dcorrel(data,g.d,m.N,tnoto);
     if (okcorrel != RETURNOK) return okcorrel; */
  /* -- */
  drealft(data,m.N,1);
  for (ii=2;ii<=m.N+2;ii+=2) {
    tnoto[ii-1]=(data[ii-1]*(dum=ftg[ii-1])+data[ii]*ftg[
  ii])/(m.N/2);
```

```
  tnoto[ii]=(data[ii]*dum-data[ii-1]*ftg[ii])/(m.N/2);
}
tnoto[2]=tnoto[m.N+1];
drealft(tnoto,m.N,-1);

for (j=1;j<=Im.min;j++){ data[j]=boundary[j-1];}/* boun
dary */
for (j=Im.max;j<=m.N;j++){ data[j]=boundary[j-1];}/* bo
undary */
/* -- */

for (j=Im.min;j<Im.max;j++){
  data[j]=m.k/2.*w.p1*sol_a[j-1]+(1.0-m.k/2.*w.p2)*sol_
a[j]+m.k/2.*w.p3*sol_a[j+1]+m.k*p.lambda*m.h*tnoto[j-Im.mi
n+1];
  /* data[j]=sol_a[j]+m.k/2.0*p.lambda*m.h*tnoto[j-Im.
min+1]; */
}
/* tridiagonal system */
tridiag_bis(p1,p2,p3,data,sol_b,m.N);

/* step 2 */
/*
    data[0]=boundary[0];
    for (j=1;j<=m.N-1;j++){
    data[j]=m.k/2.*w.p1*sol_b[j-1]+(1.0-m.k/2.*w.p2)*
sol_b[j]+m.k/2.*w.p3*sol_b[j+1];
    }
    data[m.N-1]=boundary[m.N-1];

    dtwofft(data-1,g.d-1,tnoto-1,fftg-1,m.N);
    for (j=1;j<=m.N+2;j+=2){
    r = tnoto[j-1]*(1-m.k/2.*p.lambda*fftg[j-1])-tnoto[
j]*m.k/2.*p.lambda*fftg[j];
    im = tnoto[j-1]*m.k/2.*p.lambda*fftg[j]+tnoto[j]*(1
-m.k/2.*p.lambda*fftg[j-1]);
    den = SQR(1-m.k/2.*p.lambda*fftg[j-1])+SQR(m.k/2.*
p.lambda*fftg[j]);
    fftg[j-1] = r/(den);
    fftg[j] = im/(den);
    }
```

```
        fftg[1]=fftg[m.N];
        drealft(fftg-1,m.N,-1);
    */
    for (j=0;j<m.N;j++)
      {
        sol_a[j]=sol_b[j];
        /* sol_a[j]=2.0/m.N*fftg[j]; */
        if (am)
          sol_a[j]=MAX(Obst[j],sol_a[j]);
      }
  }


  /* Price */
  *ptprice=sol_a[m.Index+1];
  /*Delta*/
  *ptdelta  = (sol_a[m.Index+1]-sol_a[m.Index-1])/(2.0*p.s*
    m.h);

  /* Memory Desallocation */
  free(sol_a);
  free(sol_b);
  free(p1);
  free(p2);
  free(p3);
  free(tnoto);
  free(data);
  free(boundary);
  free(g.d);
  free(fftg);
  free(ftg);
  free(Obst);

  return RETURNOK;
}
static int AndersenAndreasen(int am,double s,NumFunc_1  *p_
    func,double t,double r,double divid,double sigma,double lambd
    a,double mu,double gamma2,int N,int M,int bound,double *pt
    price,double *ptdelta)
{
  MESH m;
```

```
  WEIGHT w;
  IMESH Im;
  PARAM p;
  DENSITY g;
  EQ eq;
  double K;

  K=p_func->Par[0].Val.V_DOUBLE;
  Gaussian_data(mu,gamma2,&g);
  set_parameter(s,K,t,r,sigma,divid,lambda,g.Eu,&p);
  equation(p,&eq);

  if (N%2==1) N++;

  initgrid_1Dbis(p,g,eq,N,&m,&Im);
  set_weights_impl(M,p.T,eq,&m,&w);
  /* Gaussian_vect(0,Im.N,m.h,&g);   */
  asym_1d(am,p,g,m,w,Im,p_func,bound,ptprice,ptdelta);

  /*    freeDensity(&g); */

  return OK;
}

int CALC(FD_AndersenAndreasen)(void *Opt,void *Mod,Pricing
    Method *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 AndersenAndreasen(ptOpt->EuOrAm.Val.V_BOOL,
                           ptMod->S0.Val.V_PDOUBLE,
                           ptOpt->PayOff.Val.V_NUMFUNC_1,
                           ptOpt->Maturity.Val.V_DATE-pt
    Mod->T.Val.V_DATE,
                           r,
                           divid,
```

```
                           ptMod->Sigma.Val.V_PDOUBLE,
                           ptMod->Lambda.Val.V_PDOUBLE,
                           ptMod->Mean.Val.V_PDOUBLE,
                           ptMod->Variance.Val.V_PDOUBLE,
                           Met->Par[0].Val.V_INT,
                           Met->Par[1].Val.V_INT,
                           Met->Par[2].Val.V_ENUM.value,
                           &(Met->Res[0].Val.V_DOUBLE),
                           &(Met->Res[1].Val.V_DOUBLE));
}

static int CHK_OPT(FD_AndersenAndreasen)(void *Opt, void *
    Mod)
{
  /*
   *     Option* ptOpt=(Option*)Opt;
   * TYPEOPT* opt=(TYPEOPT*)(ptOpt->TypeOpt);
   */
  return  OK;
}
#endif //PremiaCurrentVersion


static int MET(Init)(PricingMethod *Met,Option *Opt)
{

  if ( Met->init == 0)
    {
      Met->init=1;

      Met->Par[0].Val.V_INT2=8192;
      Met->Par[1].Val.V_INT2=500;
      Met->Par[2].Val.V_ENUM.value=0;
      Met->Par[2].Val.V_ENUM.members=&PremiaEnumBoundaryCon
    d;

    }

  return OK;
}
```

```
PricingMethod MET(FD_AndersenAndreasen)=
{
  "FD_AndersenAndreasen",
  {{"SpaceStepNumber MUST be an integer power of 2 (this is
     not checked for!)",INT2,{100},ALLOW},
   {"TimeStepNumber",INT2,{100},ALLOW},
   {"Boundary Condition",ENUM,{1},ALLOW},
   {" ",PREMIA_NULLTYPE,{0},FORBID}},
  CALC(FD_AndersenAndreasen),
  {{"Price",DOUBLE,{100},FORBID},{"Delta",DOUBLE,{100},FORB
    ID},{" ",PREMIA_NULLTYPE,{0},FORBID}},
  CHK_OPT(FD_AndersenAndreasen),
  CHK_split,
  MET(Init)
};
```

References