```
   Help
#include  "mer1d_std.h"

static int asym_1d(int am,PARAM p, DENSITY g,MES
    H m, WEIGHT w, IMESH Im,NumFunc_1  *p_func, int
    bound,double *ptprice, double *ptdelta)
{

  int j,i;
  double integral,*weight,*sol_a,*sol_b,*p1,*p2,*
    p3,*tnoto,*boundary,*Obst;

  /* vector allocation */
  sol_a = (double *)malloc(m.N*sizeof(double));
  if (sol_a==NULL) return MEMORY_ALLOCATION_FAILU
    RE;
  Obst= (double *)malloc(m.N*sizeof(double));
  if (Obst==NULL) return MEMORY_ALLOCATION_FAILU
    RE;
  memset(sol_a,0,m.N*sizeof(double));
  sol_b = (double *)malloc(m.N*sizeof(double));
  if (sol_b==NULL) return MEMORY_ALLOCATION_FAILU
    RE;
  memset(sol_b,0,m.N*sizeof(double));
  tnoto = (double *)malloc(m.N*sizeof(double));
  if (tnoto==NULL) return MEMORY_ALLOCATION_FAILU
    RE;
  memset(tnoto,0,m.N*sizeof(double));
  p1 = (double *)malloc(m.N*sizeof(double));
  if (p1==NULL) return MEMORY_ALLOCATION_FAILURE;
  memset(p1,0,m.N*sizeof(double));
  p2 = (double *)malloc(m.N*sizeof(double));
  if (p2==NULL) return MEMORY_ALLOCATION_FAILURE;
  memset(p2,0,m.N*sizeof(double));
  p3 = (double *)malloc(m.N*sizeof(double));
  if (p3==NULL) return MEMORY_ALLOCATION_FAILURE;
  memset(p3,0,m.N*sizeof(double));
  weight = (double *)malloc(Im.N*sizeof(double));
  if (weight==NULL) return MEMORY_ALLOCATION_FAIL
    URE;
  memset(weight,0,Im.N*sizeof(double));
```

```
boundary = (double *)malloc(m.N*sizeof(double))
  ;
if (boundary==NULL) return MEMORY_ALLOCATION_FA
  ILURE;
memset(boundary,0,m.N*sizeof(double));

/* set integral weights */
d1_intcomp(Im.N,m.h,weight,g.d,SIMP);

/* Terminal Values */
for (j=0;j<m.N;j++){ /* opt = 1 call option;
  opt=0 put option */
  sol_a[j] = (p_func->Compute)(p_func->Par,exp(
  m.xmin+j*m.h));
  Obst[j]=sol_a[j];
}
/* set boundary */
set_boundaryAA(bound,m,p,Im,sol_a,boundary);

/* "Probabilities" associated to points */
for (j=0;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.*w.p1;
  p2[j]= 1.0+m.k/2.*w.p2;
  p3[j] = -m.k/2.*w.p3;
}
for (j=Im.max;j<m.N;j++){
  p1[j]= 0.;
  p2[j]= 1.0;
  p3[j] = 0.;
}
/* Finite Difference Cycle */
for (i=1;i<=m.M;i++)
  {
    for (j=0;j<Im.min;j++){ tnoto[j]=boundary[
  j];}/* boundary */
    for (j=Im.max;j<m.N;j++){ tnoto[j]=boundary
```

```
    [j];}/* boundary */
      for (j=Im.min;j<Im.max;j++){
  integral = calc_int(Im.N,weight,&sol_a[j-Im.mi
    n]);
  tnoto[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*
    integral;
      }
      /* tridiagonal system */
      tridiagsystem(p1,p2,p3,tnoto,sol_b,m.N);

      for (j=0;j<m.N;j++)
  {
    sol_a[j]=sol_b[j];

    if (am)
      sol_a[j]=MAX(Obst[j],sol_a[j]);
  }
    }

  /* Price */
  *ptprice=sol_a[m.Index];
  /*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(weight);
  free(p1);
  free(p2);
  free(p3);
  free(tnoto);
  free(boundary);

  return RETURNOK;

}

static int ImpExp(int am,double s,NumFunc_1  *p_
```

```
    func,double t,double r,double divid,double sigma,
    double lambda,double mu,double gamma2,int N,int M,int
    bound,double *ptprice,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,ptde
    lta);

  freeDensity(&g);

  return OK;
}

int CALC(FD_ImpExp)(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 ImpExp(ptOpt->EuOrAm.Val.V_BOOL,ptMod->
    S0.Val.V_PDOUBLE,
    ptOpt->PayOff.Val.V_NUMFUNC_1,ptOpt->Maturi
    ty.Val.V_DATE-ptMod->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_INT,&(Met->Res[0].Val.V_
    DOUBLE),&(Met->Res[1].Val.V_DOUBLE));
}

int CHK_OPT(FD_ImpExp)(void *Opt, void *Mod)
{
  Option* ptOpt=(Option*)Opt;
  TYPEOPT* opt=(TYPEOPT*)(ptOpt->TypeOpt);

  return  OK;
}

static int MET(Init)(PricingMethod *Met)
{
  static int first=1;

  if (first)
    {
      Met->Par[0].Val.V_INT2=2000;
      Met->Par[1].Val.V_INT2=100;
      Met->Par[2].Val.V_INT2=0;
      first=0;
    }

  return OK;
}

PricingMethod MET(FD_ImpExp)=
{
  "FD_ImpExp",
  {{"SpaceStepNumber",INT2,500,ALLOW},{"TimeStep
    Number",INT2,100,ALLOW},
   {"Boundary Condition:Dirichlet(0) or Andrease
    n(1)?",INT,1,ALLOW},
```

```
   {" ",END,0,FORBID}},
  CALC(FD_ImpExp),
  {{"Price",DOUBLE,100,FORBID},{"Delta",DOUBLE,10
    0,FORBID},{" ",END,0,FORBID}},
  CHK_OPT(FD_ImpExp),
  CHK_split,
  MET(Init)
};
```

References