Help

```
#include "bs1d_doublim.h"

static int Cryer_In(double s,NumFunc_1  *L,
    NumFunc_1  *U,NumFunc_1  *Rebate,NumFunc_1  *p,double
    t,double r,double divid,double sigma,int N,int M,
    double *ptprice,double *ptdelta)
{
  int      Index,PriceIndex,TimeIndex,ssl,dummy;
  double   k,vv,h,z,alpha,beta,gamma,y,down,u,l,
    rebate,up,upwind_alphacoef;
  double   *Obst,*A,*B,*C,*P,*S,*Q,*Z,pricenh,pr
    icen2h,priceph;

  /*Memory Allocation*/
  Obst=(double *)malloc((N+1)*sizeof(double));
  if (Obst==NULL)
    return MEMORY_ALLOCATION_FAILURE;
  A=(double *)malloc((N+1)*sizeof(double));
  if (A==NULL)
    return MEMORY_ALLOCATION_FAILURE;
  B=(double *)malloc((N+1)*sizeof(double));
  if (B==NULL)
    return MEMORY_ALLOCATION_FAILURE;
  C=(double *)malloc((N+1)*sizeof(double));
  if (C==NULL)
    return MEMORY_ALLOCATION_FAILURE;
  P=(double *)malloc((N+1)*sizeof(double));
  if (P==NULL)
    return MEMORY_ALLOCATION_FAILURE;
  S=(double *)malloc((N+1)*sizeof(double));
  if (S==NULL)
    return MEMORY_ALLOCATION_FAILURE;
  Z=(double *)malloc((N+1)*sizeof(double));
  if (Z==NULL)
    return MEMORY_ALLOCATION_FAILURE;
  Q=(double *)malloc((N+1)*sizeof(double));
  if (Q==NULL)
    return MEMORY_ALLOCATION_FAILURE;

  /*Time Step*/
```

```
k=t/(double)M;

/*Space Step*/
u=(U->Compute)(U->Par,0);
l=(L->Compute)(L->Par,0);
rebate=(Rebate->Compute)(Rebate->Par,0);
y=log(s);
down=log(l);
up=log(u);
h=(up-down)/(double)(N);

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

/*Lhs Factor of theta-schema*/
alpha=k*(-vv/(h*h)+z/(2.0*h));
beta=1.0+k*(r+2.*vv/(h*h));
gamma=k*(-vv/(h*h)-z/(2.0*h));

for(PriceIndex=0;PriceIndex<=N-2;PriceIndex++)
  {
    A[PriceIndex]=alpha;
    B[PriceIndex]=beta;
    C[PriceIndex]=gamma;
  }

/*Terminal Values*/
y=log(s);
for (PriceIndex = 1; PriceIndex < N; PriceInd
  ex++)
Obst[PriceIndex - 1]=(p->Compute)(p->Par,exp(
  down+PriceIndex* h));
```

```
for (PriceIndex = 2; PriceIndex <= N - 2; Pric
  eIndex++)
{
  P[PriceIndex - 1] = alpha * Obst[PriceIndex
  - 2] +
  beta * Obst[PriceIndex - 1] + gamma * Obst[
  PriceIndex];
}

P[0] =beta*Obst[0]+gamma*Obst[1];
P[N - 2] = alpha*Obst[N-3] + beta * Obst[N-2];

for (PriceIndex = 0; PriceIndex <= N - 2; Pric
  eIndex++)
{
  S[PriceIndex] = 0.0;
  Z[PriceIndex] = 0.0;
}
ssl = false;

/*Finite Difference Cycle*/
for (TimeIndex= 1; TimeIndex<= M; TimeIndex++)
{
  if (TimeIndex==1)
  for (PriceIndex = 0; PriceIndex <= N- 2; Pr
  iceIndex++)
    Z[PriceIndex] =rebate;
  else
  for (PriceIndex = 0; PriceIndex <= N- 2; Pr
  iceIndex++)
    Z[PriceIndex] =Z[PriceIndex]+Obst[PriceI
  ndex];

  for (PriceIndex = 0; PriceIndex <= N - 2; Pr
  iceIndex++)
  Q[PriceIndex]=P[PriceIndex]-Z[PriceIndex];
  Q[0] += alpha*Boundary(l,p,(double)TimeInd
  ex*k,r,divid,sigma);;
  Q[N-2]+=gamma*Boundary(u,p,(double)TimeInd
  ex*k,r,divid,sigma);
```

```
  dummy=AlgCrayer(N,Z,ssl,A,B,C,Q,S);

  for (PriceIndex = 0; PriceIndex <=N-2; Pric
  eIndex++)
  S[PriceIndex] = Z[PriceIndex];

  ssl = true;
}

for (PriceIndex = 0; PriceIndex <= N - 2; Pric
  eIndex++)
P[PriceIndex]=Z[PriceIndex]+Obst[PriceIndex];

Index=(int)floor((y-down)/h)-1;

/*Price*/
if ((y==up)&&(y==down))
  *ptprice=P[0];
else
  *ptprice=P[Index]+(P[Index+1]-P[Index])*(exp(
  y)-exp(down+h+Index*h))/(exp(down+h+(Index+1)*h)
  -exp(down+h+Index*h));

/*Delta*/
if ((y==up)&&(y==down))
  *ptdelta=0.0;
else {
  pricenh=P[Index+1]+(P[Index+2]-P[Index+1])*(
  exp(y+h)-exp(down+h+(Index+1)*h))/(exp(down+h+(Ind
  ex+2)*h)-exp(down+h+(Index+1)*h));
  if (Index>0) {
    priceph=P[Index-1]+(P[Index]-P[Index-1])*(
  exp(y-h)-exp(down+h+(Index-1)*h))/(exp(down+h+(Ind
  ex)*h)-exp(down+h+(Index-1)*h));
    *ptdelta=(pricenh-priceph)/(2*s*h);
  } else {
    pricen2h=P[Index+2]+(P[Index+3]-P[Index+2])
  *(exp(y+2*h)-exp(down+h+(Index+2)*h))/(exp(down+
  h+(Index+3)*h)-exp(down+h+(Index+2)*h));
    *ptdelta=(4*pricenh-pricen2h-3*(*ptprice))/
```

```
    (2*s*h);
    }
  }

  /*Memory Desallocation*/
  free(Obst);
  free(A);
  free(B);
  free(C);
  free(P);
  free(S);
  free(Z);
  free(Q);

  return OK;
}

int CALC(FD_Cryer_In)(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 Cryer_In(ptMod->S0.Val.V_PDOUBLE,
          ptOpt->LowerLimit.Val.V_NUMFUNC_1,
    ptOpt->UpperLimit.Val.V_NUMFUNC_1,ptOpt->Rebate.
    Val.V_NUMFUNC_1,ptOpt->PayOff.Val.V_NUMFUNC_1,
          ptOpt->Maturity.Val.V_DATE-ptMod->
    T.Val.V_DATE,r,divid,ptMod->Sigma.Val.V_PDOUBLE,
          Met->Par[0].Val.V_INT,Met->Par[1].
    Val.V_INT,
          &(Met->Res[0].Val.V_DOUBLE),&(Met->
    Res[1].Val.V_DOUBLE));
}

int CHK_OPT(FD_Cryer_In)(void *Opt, void *Mod)
{
```

```
  Option* ptOpt=(Option*)Opt;
  TYPEOPT* opt=(TYPEOPT*)(ptOpt->TypeOpt);

  if ((opt->Parisian).Val.V_BOOL==WRONG)
  if ( (strcmp( ((Option*)Opt)->Name,"
    DoubleCallInAmer")==0) || (strcmp( ((Option*)Opt)->Name,"
    DoublePutInAmer")==0) )
    return OK;
  return WRONG;

  return  WRONG;
}

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

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

      first=0;
    }

  return OK;
}

PricingMethod MET(FD_Cryer_In)=
{
  "FD_Cryer_In",
  {{"SpaceStepNumber",INT2,100,ALLOW},{"TimeStep
    Number",INT2,100,ALLOW},{" ",END,0,FORBID}},
  CALC(FD_Cryer_In),
  {{"Price",DOUBLE,100,FORBID},{"Delta",DOUBLE,10
    0,FORBID} ,{" ",END,0,FORBID}},
  CHK_OPT(FD_Cryer_In),
  CHK_split,
  MET(Init)
};
```

References