Help

```
#include  "bs1d_lim.h"

static int Cryer_DownIn(double s,NumFunc_1  *p,
    double l,double rebate,double t,double r,double divid,
    double sigma,int N,int M,double *ptprice,double *ptde
    lta)
{
  int      Index,PriceIndex,TimeIndex,ssl,dummy;
  double   k,vv,loc,h,z,alpha,beta,gamma,y,down,u
    pwind_alphacoef,price1;
  double   *Obst,*A,*B,*C,*P,*S,*Z,*Q,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 Localisation*/
vv=0.5*sigma*sigma;
z=(r-divid)-vv;
loc=sigma*sqrt(t)*sqrt(log(1.0/PRECISION))+fa
  bs(z*t);

/*Space Step*/
y=log(s);
down=log(l);
h=(y+loc-down)/(double)(N);

/*Peclet Condition-Coefficient of diffusion aug
  mented */
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;
  }

/*Ternminal 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[PriceInd
  ex - 2] +
    beta * Obst[PriceIndex - 1] + gamma * Ob
  st[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.;
  Z[PriceIndex] = 0.;
  }
ssl = false;

/*Finite Difference Cycle*/
for (TimeIndex= 1; TimeIndex<= M; TimeIndex++)
    {
    if (TimeIndex==1)
    for (PriceIndex = 0; PriceIndex <= N- 2;
   PriceIndex++)
      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;
  PriceIndex++)
    Q[PriceIndex]=P[PriceIndex]-Z[PriceInd
  ex];

  price1=Boundary(l,p,(double)TimeIndex*k,r,
```

```
  divid,sigma);
  Q[0] += alpha*price1;
  Q[N-2]+=gamma*(p->Compute)(p->Par,exp(y+loc
  ));

  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[PriceInd
  ex];

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

/*Price*/
*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*/
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_DownIn)(void *Opt,void *Mod,Pr
    icingMethod *Met)
{
  TYPEOPT* ptOpt=(TYPEOPT*)Opt;
  TYPEMOD* ptMod=(TYPEMOD*)Mod;
  double r,divid,limit,rebate;

    r=log(1.+ptMod->R.Val.V_DOUBLE/100.);
    divid=log(1.+ptMod->Divid.Val.V_DOUBLE/100.);
    limit=((ptOpt->Limit.Val.V_NUMFUNC_1)->Compu
    te)((ptOpt->Limit.Val.V_NUMFUNC_1)->Par,ptMod->T.
    Val.V_DATE);
    rebate=((ptOpt->Rebate.Val.V_NUMFUNC_1)->
    Compute)((ptOpt->Rebate.Val.V_NUMFUNC_1)->Par,ptMod-
    >T.Val.V_DATE);

  return Cryer_DownIn(ptMod->S0.Val.V_PDOUBLE,pt
    Opt->PayOff.Val.V_NUMFUNC_1,
      limit,rebate, ptOpt->Maturity.Val.V_DATE-pt
    Mod->T.Val.V_DATE,r,divid,ptMod->Sigma.Val.V_PDOU
    BLE,
      Met->Par[0].Val.V_INT2,Met->Par[1].Val.V_
```

```
    INT2,
      &(Met->Res[0].Val.V_DOUBLE),&(Met->Res[1].
    Val.V_DOUBLE));
}

int CHK_OPT(FD_Cryer_DownIn)(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,"
    CallDownInAmer")==0) || (strcmp( ((Option*)Opt)->Name,"
    PutDownInAmer")==0) )
        return OK;

  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_DownIn)=
{
  "FD_Cryer",
  {{"SpaceStepNumber",INT2,100,ALLOW    },{"TimeS
    tepNumber",INT2,100,ALLOW},
   {" ",END,0,FORBID}},
```

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

References