```
   Help
#include <stdlib.h>
#include  "bs1d_std.h"
#define PRECISION 1.0e-7 /*Precision for the localization
    of FD methods*/

typedef struct InfoOfBlock {
  int *IndexFirstOfBlock;
  int *IndexLastOfBlock;
  int *NextBlock;
  int *PreviousBlock;
  long FirstBlock;
  long LastBlock;
  int  *ForwardSol;
  int *BackwardSol;
  int Size;
} InfoOfBlock;

static int BackwardPass(int N,double *ZZ, double *Bfor,
    double *Zfor,
      double *Bbac, double *Zbac, long *Npiv,
      long *Npass, long *p, InfoOfBlock *Info, double *
    A,
      double *B, double *C, double *Q, long NOfBlock);

static void ForElimination(long Indexp,long Indexd, double
    *ZZ,double *Bfor,double *Zfor,double *A,double *B,double *
    C,double *Q)
{
  double x;
  long Index;

  for (Index= Indexp + 1; Index<= Indexd;Index++) {
    x = A[Index- 1] / Bfor[Index- 2];
    Bfor[Index- 1] = B[Index- 1] - x * C[Index- 2];
    Zfor[Index- 1] = -Q[Index- 1] - x * Zfor[Index- 2];
  }

  ZZ[Indexd] = Zfor[Indexd - 1] / Bfor[Indexd - 1];

  return;
```

```
}

static void BacElimination(long Indexp,long Indexd, double
    *ZZ,double *Bbac,double *Zbac,double *A,double *B,double *
    C,double *Q)
{
  double x;
  long Index;

  for (Index= Indexd - 1; Index>= Indexp; Index--) {
    x = C[Index- 1] / Bbac[Index];
    Bbac[Index- 1] = B[Index- 1] - x * A[Index];
    Zbac[Index- 1] = -Q[Index- 1] - x * Zbac[Index];
  }

  ZZ[Indexp] = Zbac[Indexp - 1] / Bbac[Indexp - 1];

  return;
}

static int SolEquation(int N,double *ZZ,double *Bfor,
    double *Zfor, double *Bbac, double *Zbac, double *A,double *B,
    double *C,double *Q,InfoOfBlock Info,long NOfBlock)

{
  int *forsol, *bacsol;
  long IndexBloc, FORLIM1,Index,Ip,Id;


  forsol= malloc((N+1)*sizeof(int));
  if (forsol==NULL)
    return 1;
  bacsol= malloc((N+1)*sizeof(int));
  if (bacsol==NULL)
    return 1;

  for (IndexBloc = 0;IndexBloc<NOfBlock; IndexBloc++) {
    forsol[IndexBloc] = Info.ForwardSol[IndexBloc];
    bacsol[IndexBloc] = Info.BackwardSol[IndexBloc];
    Ip = Info.IndexFirstOfBlock[IndexBloc + 1];
    Id = Info.IndexLastOfBlock[IndexBloc + 1];
    while (forsol[IndexBloc] == false && bacsol[IndexBloc]
    == false) {
      ForElimination(Ip, Id, ZZ, Bfor, Zfor, A, B, C, Q);
      forsol[IndexBloc] = true;
    }
    if (forsol[IndexBloc] == true) {
      FORLIM1 = Ip;
      for (Index = Id - 1; Index >= FORLIM1; Index--)
  ZZ[Index] = (Zfor[Index - 1] - C[Index - 1] *
         ZZ[Index + 1]) / Bfor[Index - 1];
    }
    if (bacsol[IndexBloc] == true) {
      FORLIM1 = Id;
      for (Index = Ip + 1; Index <= FORLIM1; Index++)
  ZZ[Index] = (Zbac[Index - 1] - A[Index - 1] *
         ZZ[Index - 1]) / Bbac[Index - 1];
    }
  }

  free(forsol);
  free(bacsol);

  return OK;

}

static int MethodA(int N,long *NBl,long *NSigneChange,Info
    OfBlock *Info,double *Q,double *S)
{
  long Neg;
  int *Np,*Nd;
  long FORLIM,Index;

  Np= malloc((N+1)*sizeof(int));
  if (Np==NULL)
    return 1;
  Nd= malloc((N+1)*sizeof(int));
  if (Nd==NULL)
    return 1;
```

```
if (Q[0] < 0 || S[0] > 0) {
  Neg = 1;
  *NSigneChange = 0;
  *NBl = 1;
  Np[1] = 1;
  Nd[1] = 1;
} else {
  Neg = 0;
  *NSigneChange = 0;
  *NBl = 0;
}
for (Index= 2; Index< N; Index++) {
  if (Neg == 1) {
    if (Q[Index- 1] < 0 || S[Index- 1] > 0) {
Nd[*NBl]++;
Neg = 1;
    } else {
(*NSigneChange)++;
Neg = 0;
    }
  } else {
    if (Q[Index- 1] < 0 || S[Index- 1] > 0) {
(*NSigneChange)++;
(*NBl)++;
Np[*NBl] = Index;
Nd[*NBl] = Index;
Neg = 1;
    } else
Neg = 0;
  }

}

for (Index=0;Index<N;Index++) {
  Info->IndexFirstOfBlock[Index]=0;
  Info->IndexLastOfBlock[Index]=0;
  Info->NextBlock[Index] = 0;
  Info->PreviousBlock[Index] = 0;

}
Info->IndexFirstOfBlock[0] = 0;
```

```
  Info->IndexLastOfBlock[0] =N;
  for (Index= 1; Index<N; Index++) {
    Info->ForwardSol[Index- 1] = false;
    Info->BackwardSol[Index- 1] = false;
  }
  FORLIM =*NBl;
  for (Index= 1; Index<= FORLIM; Index++) {
    Info->IndexFirstOfBlock[Index] = Np[Index];
    Info->IndexLastOfBlock[Index] = Nd[Index];
  }
  FORLIM = *NBl;
  for (Index= 1; Index< FORLIM; Index++)
    Info->NextBlock[Index] = Index+ 1;
  Info->NextBlock[*NBl] = 0;
  FORLIM = *NBl;
  for (Index= 2; Index<= FORLIM; Index++)
    Info->PreviousBlock[Index] = Index- 1;
  Info->PreviousBlock[1] = 0;
  if (*NBl == 0) {
    Info->FirstBlock = 0;
    Info->LastBlock = 0;
  } else {
    Info->FirstBlock = 1;
    Info->LastBlock = *NBl;
  }

  free(Nd);
  free(Np);

  return OK;
}

static void MethodB(int N,long *NOfBlock,long *NbreSigneCha
    nge,InfoOfBlock  *Info,double *ZZ,double  *Bfor, double *Zf
    or,double *Bbac,double *Zbac,
        double *A,double *B,double *C,double  *Q,double *
    S,int sll)

{
  long IndexPrime, FORLIM, FORLIM1,Index,Ip,Id;
  if (sll == true) {
    MethodA(N,NOfBlock,NbreSigneChange,Info,Q,S);
    SolEquation(N,ZZ, Bfor, Zfor, Bbac, Zbac, A, B, C, Q, *
    Info,*NOfBlock);
    FORLIM = *NOfBlock;
    for (Index=1; Index <= FORLIM;Index++) {
      Ip = Info->IndexFirstOfBlock[Index];
      Id = Info->IndexLastOfBlock[Index];
      FORLIM1 = Id;
      for (IndexPrime = Ip; IndexPrime <= FORLIM1; IndexP
    rime++) {
  if (ZZ[IndexPrime] <= 0)
    sll = false;
      }
    }
  }
  if (sll == false){
    for (Index = 1; Index <N; Index++)
      S[Index - 1] = 0.0;
    MethodA(N,NOfBlock,NbreSigneChange,Info,Q,S);
  }

  return;

}

static void ForFondreDeuxBloc(InfoOfBlock *Info, long *Ind
    exd,long NBloc)
{
  long nk, nnk;

  nk=Info->NextBlock[NBloc];
  nnk = Info->NextBlock[nk];
  *Indexd = Info->IndexLastOfBlock[nk];
  Info->IndexLastOfBlock[NBloc] = *Indexd;
  Info->PreviousBlock[nnk] = Info->PreviousBlock[nk];
  Info->NextBlock[NBloc] = Info->NextBlock[nk];

  return;

}
```

```
static void BacFondreDeuxBloc(InfoOfBlock *Info,long *Ind
    exp,long NBloc)
{
  long nk, nnk;

  nk = Info->PreviousBlock[NBloc];
  nnk = Info->PreviousBlock[nk];
  *Indexp = Info->IndexFirstOfBlock[nk];
  Info->IndexFirstOfBlock[NBloc] = *Indexp;
  Info->NextBlock[nnk] = Info->NextBlock[nk];
  Info->PreviousBlock[NBloc] = Info->PreviousBlock[nk];

  return;

}

static int ForwardPass(int N,double *ZZ,double *Bfor,
    double *Zfor,
           double *Bbac, double *Zbac, long *Npiv,
           long *Npass, long *p, InfoOfBlock *Info,
    double *A,
           double *B, double *C, double *Q, long NOfBlock
    )
{
  long NBloc,  NNextBlock, I, II,Ip,Id;
  double d=0., x;
  int *forsol, *bacsol;

  forsol= malloc((N+1)*sizeof(int));
  if (forsol==NULL)
    return 1;
  bacsol= malloc((N+1)*sizeof(int));
  if (bacsol==NULL)
    return 1;


  *Npiv = 0;
  (*Npass)++;
  NBloc = Info->FirstBlock;
```

```
do {

  if (NBloc == 0) {
    if (*Npiv == 0 && *Npass >= 2)
SolEquation(N,ZZ, Bfor, Zfor, Bbac, Zbac, A, B, C, Q, *
  Info,NOfBlock);
    else
BackwardPass(N,ZZ, Bfor, Zfor, Bbac, Zbac, Npiv, Npass,
  p, Info, A, B,C,Q,NOfBlock);
  } else {
    NNextBlock=Info->NextBlock[NBloc];
    forsol[NBloc - 1] = Info->ForwardSol[NBloc - 1];
    bacsol[NBloc - 1] = Info->BackwardSol[NBloc - 1];
    if (forsol[NBloc - 1] == true && *Npass >= 3)
NBloc = NNextBlock;
    else {
Ip = Info->IndexFirstOfBlock[NBloc];
Id = Info->IndexLastOfBlock[NBloc];
if (Id ==N - 1)
  NBloc = NNextBlock;
else {
  while (forsol[NBloc - 1] == false) {
    ForElimination(Ip, Id, ZZ, Bfor, Zfor, A, B, C, Q);
    forsol[NBloc - 1] = true;
    Info->ForwardSol[NBloc - 1] = forsol[NBloc - 1];
  }

  do {

    if (Id ==N - 1)
      NBloc = NNextBlock;
    else {
      d = ZZ[Id] * A[Id] + Q[Id];
      if (ZZ[Id + 2] != 0)
  d += ZZ[Id + 2] * C[Id];
      if (d >= 0)
  NBloc = NNextBlock;
      else {
  (*p)++;
  (*Npiv)++;
  Id++;
  Info->IndexLastOfBlock[NBloc] = Id;
  Info->BackwardSol[NBloc - 1] = false;
  x = A[Id - 1] / Bfor[Id - 2];
  Bfor[Id - 1] = B[Id - 1] - x * C[Id - 2];
  Zfor[Id - 1] = -Q[Id - 1] - x * Zfor[Id - 2];
  ZZ[Id] = Zfor[Id - 1] / Bfor[Id - 1];
  I = Info->IndexFirstOfBlock[NNextBlock];
  if (Id + 1 == I) {
   II = Info->IndexLastOfBlock[NNextBlock];
   ForElimination(Id, II, ZZ, Bfor, Zfor, A, B, C, Q);
   ForFondreDeuxBloc(Info,&Id,NBloc);
  }
    }
   }
  } while ((Id !=(N - 1)) && (d < 0));
  NBloc = NNextBlock;
 }
   }
  }

 } while (NBloc != 0);

 if (*Npiv == 0 && *Npass >= 2)
  SolEquation(N,ZZ, Bfor, Zfor, Bbac, Zbac, A, B, C, Q, *
  Info,NOfBlock);
 else
  BackwardPass(N,ZZ, Bfor, Zfor, Bbac, Zbac, Npiv, Npass,
   p, Info, A, B, C, Q,
      NOfBlock);

 free(forsol);
 free(bacsol);

 return OK;

}

static int  BackwardPass(int N,double *ZZ, double *Bfor,
  double *Zfor,
    double *Bbac, double *Zbac, long *Npiv,
       long *Npass, long *p, InfoOfBlock *Info, double *
    A,
       double *B, double *C, double *Q, long NOfBlock)
{
  long NBloc, NPreviousBlock, I, II,Ip,Id;
  double d, x;
  int *forsol, *bacsol;

  forsol= malloc((N+1)*sizeof(int));
  if (forsol==NULL)
    return 1;
  bacsol= malloc((N+1)*sizeof(int));
  if (bacsol==NULL)
    return 1;

  *Npiv = 0;
  (*Npass)++;
  NBloc = Info->LastBlock;
  do {

    if (NBloc == 0) {
      if (*Npiv == 0 && *Npass >= 2)
  SolEquation(N,ZZ,Bfor, Zfor, Bbac, Zbac, A, B, C, Q, *
    Info,NOfBlock);
      else
  ForwardPass(N,ZZ,Bfor,Zfor,Bbac,Zbac,Npiv,Npass,p,Info,
    A,B,C,Q,NOfBlock);
    } else {
      NPreviousBlock = Info->PreviousBlock[NBloc];
      forsol[NBloc - 1] = Info->ForwardSol[NBloc - 1];
      bacsol[NBloc - 1] = Info->BackwardSol[NBloc - 1];
      if (bacsol[NBloc - 1] == true && *Npass >= 3)
  NBloc = NPreviousBlock;
      else {
  Ip = Info->IndexFirstOfBlock[NBloc];
  Id = Info->IndexLastOfBlock[NBloc];
  if (Ip == 1)
    NBloc = NPreviousBlock;
  else {
    while (bacsol[NBloc - 1] == false) {
      BacElimination(Ip,Id, ZZ, Bbac, Zbac, A, B, C, Q);
      bacsol[NBloc - 1] = true;
      Info->BackwardSol[NBloc - 1] = bacsol[NBloc - 1];
    }

    do {
      if (Ip == 1)
        NBloc = NPreviousBlock;
      else {
        d = ZZ[Ip] * C[Ip - 2] + Q[Ip - 2];
        if (ZZ[Ip - 2] != 0)
    d += ZZ[Ip - 2] * A[Ip - 2];
        if (d >= 0)
    NBloc = NPreviousBlock;
        else {
    (*p)++;
    (*Npiv)++;
    Ip--;
    Info->IndexFirstOfBlock[NBloc] = Ip;
    Info->ForwardSol[NBloc - 1] = false;
    x = C[Ip - 1] / Bbac[Ip];
    Bbac[Ip - 1] = B[Ip - 1] - x * A[Ip];
    Zbac[Ip - 1] = -Q[Ip - 1] - x * Zbac[Ip];
    ZZ[Ip] = Zbac[Ip - 1] / Bbac[Ip - 1];
    I = Info->IndexLastOfBlock[NPreviousBlock];
    if (Ip - 1 == I) {
      II = Info->IndexFirstOfBlock[NPreviousBlock];
      BacElimination(II, Ip, ZZ, Bbac, Zbac, A, B, C, Q);
      BacFondreDeuxBloc(Info,&Ip,NBloc);
    }
        }
      }
    } while ((Ip != 1) && (d < 0));
    NBloc=NPreviousBlock;
  }
      }
    }

  } while (NBloc != 0);
  if (*Npiv == 0 && *Npass >= 2)
    SolEquation(N,ZZ, Bfor, Zfor, Bbac, Zbac, A, B, C, Q, *
    Info,NOfBlock);
  else
    ForwardPass(N,ZZ, Bfor, Zfor, Bbac, Zbac, Npiv, Npass,
    p,Info, A, B, C, Q,
          NOfBlock);

  free(forsol);
  free(bacsol);

  return OK;

}

int AlgCrayer(int N,double *Z,int  ssl,double *A,double *B,
    double *C,double*Q,double  *S)
{
  double *ZZ,*Bbac, *Bfor, *Zbac, *Zfor;
  long NOfBlock, NbreSigneChange, Npiv,Npass,p,Index,Ip,Id;
  InfoOfBlock Info;

  ZZ= malloc((N+1)*sizeof(double));
  if (ZZ==NULL)
    return 1;
  Bbac= malloc((N+1)*sizeof(double));
  if (Bbac==NULL)
    return 1;
  Bfor= malloc((N+1)*sizeof(double));
  if (Bfor==NULL)
    return 1;
  Zbac= malloc((N+1)*sizeof(double));
  if (Zbac==NULL)return 1;
  Zfor= malloc((N+1)*sizeof(double));
  if (Zfor==NULL)
    return 1;

  Info.Size=N+1;

  Info.IndexFirstOfBlock= malloc((N+1)*sizeof(int));
  if (Info.IndexFirstOfBlock==NULL)
    return 1;
  Info.IndexLastOfBlock= malloc((N+1)*sizeof(int));
  if (Info.IndexLastOfBlock==NULL)
    return 1;
  Info.NextBlock= malloc((N+1)*sizeof(int));
  if (Info.NextBlock==NULL)
    return 1;
  Info.PreviousBlock= malloc((N+1)*sizeof(int));
  if (Info.PreviousBlock==NULL)
    return 1;
  Info.ForwardSol= malloc((N+1)*sizeof(int));
  if (Info.ForwardSol==NULL)
    return 1;
  Info.BackwardSol= malloc((N+1)*sizeof(int));
  if (Info.BackwardSol==NULL)
    return 1;

  for (Index = 1; Index< N; Index++)
    {
      Bfor[Index - 1] = B[Index - 1];
      Bbac[Index - 1] = B[Index - 1];
      Zfor[Index - 1] = -Q[Index - 1];
      Zbac[Index - 1] = -Q[Index - 1];
      ZZ[Index] = 0.0;
    }
  ZZ[0] = 0.0;
  ZZ[N] = 0.0;

  Npass = 0;
  p = 0;
  MethodB(N,&NOfBlock,&NbreSigneChange,&Info,ZZ,Bfor,Zfor,
    Bbac,Zbac, A,B,C,Q,S,ssl);
  Ip = Info.IndexFirstOfBlock[1];
  Id = Info.IndexLastOfBlock[NOfBlock];
  if (Ip<=N-Id)
    ForwardPass(N,ZZ,Bfor,Zfor,Bbac,Zbac,&Npiv,&Npass,&p,&
    Info, A,B,C,Q,NOfBlock);
  else
    BackwardPass(N,ZZ,Bfor,Zfor,Bbac,Zbac,&Npiv,&Npass,&p,&
    Info,A,B,C,Q,NOfBlock);

  for (Index = 1; Index < N;Index++)
    Z[Index - 1] = ZZ[Index];
  free(ZZ);
  free(Bbac);
  free(Bfor);
  free(Zbac);
  free(Zfor);
  free(Info.IndexFirstOfBlock);
  free(Info.IndexLastOfBlock);
  free(Info.NextBlock);
  free(Info.PreviousBlock);
  free(Info.ForwardSol);
  free(Info.BackwardSol);

  return OK;
}

static int Cryer_84(double s,NumFunc_1  *p,double t,double
    r,double divid,double sigma,int N,int M,double *ptprice,
    double *ptdelta)
{
  double  *Obst,*A,*B,*C,*P,*S,*Z,*Q;
  double  alpha,beta,gamma,h,k,vv,y,l,z,upwind_alphacoef,
    pricenh,priceph;
  int Index,PriceIndex,TimeIndex;
  int ssl;

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

```
S= malloc((N+1)*sizeof(double));
if (S==NULL)
  return 1;
Z= malloc((N+1)*sizeof(double));
if (Z==NULL)
  return 1;
Q= malloc((N+1)*sizeof(double));
if (Q==NULL)
  return 1;

/*Time Step*/
k=t/(double)M;

/*Space Localisation*/
vv=sigma*sigma;
z=r-divid-vv/2.0;
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 Implicit Schema*/
alpha=k*(-vv/(2.0*h*h)+z/(2.0*h));
beta=1+k*(r+vv/(h*h));
gamma=k*(-vv/(2.0*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; PriceIndex++)
  Obst[PriceIndex - 1]=(p->Compute)(p->Par,exp(y-l+PriceI
  ndex* h));

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

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

    for (PriceIndex = 0; PriceIndex <= N - 2; PriceIndex+
  +)
Q[PriceIndex] =  P[PriceIndex]-Z[PriceIndex];
    Q[0] += alpha*(p->Compute)(p->Par,exp(y-l));
    Q[N-2]+=gamma*(p->Compute)(p->Par,exp(y+l));
    /*Algorithm of Cryer*/
    AlgCrayer(N,Z,ssl,A,B,C,Q,S);

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

```
      ssl = true;
    }

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

  Index=(int)floor(l/h)-1;

  /*Price*/
  *ptprice=P[Index]+(P[Index+1]-P[Index])*(exp(y)-exp(y-l+
    h+Index*h))/(exp(y-l+h+(Index+1)*h)-exp(y-l+h+Index*h));

  /*Delta */
  pricenh=P[Index+1]+(P[Index+2]-P[Index+1])*(exp(y+h)-exp(
    y-l+h+(Index+1)*h))/(exp(y-l+h+(Index+2)*h)-exp(y-l+h+(Ind
    ex+1)*h));
  priceph=P[Index-1]+(P[Index]-P[Index-1])*(exp(y-h)-exp(y-
    l+h+(Index-1)*h))/(exp(y-l+h+(Index)*h)-exp(y-l+h+(Index-1)
    *h));
  *ptdelta=(pricenh-priceph)/(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)(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.);

  return  Cryer_84(ptMod->S0.Val.V_PDOUBLE,ptOpt->PayOff.
    Val.V_NUMFUNC_1,ptOpt->Maturity.Val.V_DATE-ptMod->T.Val.V_DA
    TE,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));
}

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

  if ((opt->EuOrAm). Val.V_BOOL==AMER)
    return OK;

  return  WRONG;
}

static int MET(Init)(PricingMethod *Met,Option *Opt)
{
  if ( Met->init == 0)
    {
      Met->init=1;

      Met->Par[0].Val.V_INT2=100;
      Met->Par[1].Val.V_INT2=100;


    }

  return OK;
}

PricingMethod MET(FD_Cryer)=
{
  "FD_Cryer",
  {{"SpaceStepNumber",INT2,{100},ALLOW    },{"TimeStepNumb
    er",INT2,{100},ALLOW},
   {" ",PREMIA_NULLTYPE,{0},FORBID}},
  CALC(FD_Cryer),
  {{"Price",DOUBLE,{100},FORBID},{"Delta",DOUBLE,{100},FORB
    ID} ,{" ",PREMIA_NULLTYPE,{0},FORBID}},
  CHK_OPT(FD_Cryer),
  CHK_fdiff,
  MET(Init)
};
```

References