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

static int boundary(int bound,NumFunc_1 *p,double y,double
    l,double h,double *ptbound1,double *ptbound2)
{
  /*Natural Dirichlet Boundary Conditions*/
  if (bound==0) {
    *ptbound1=(p->Compute)(p->Par,exp(y-l));
    *ptbound2=(p->Compute)(p->Par,exp(y+l));
  }

  /*Natural Neumann Boundary Conditions*/
  else
    {
      if (( (p->Compute) == &Call))
        {
          *ptbound1=0.;
          *ptbound2=exp(y+l)*h;
        }
      else
      if (( (p->Compute) == &Put))
        {
          *ptbound1=-exp(y-l)*h;
          *ptbound2=0.;
        }
      else
  /*if (( (p->Compute) ==  &CallSpread )||((p->Compute) ==
      &Digit))*/
        {
          *ptbound1=0.;
          *ptbound2=0.;
        }
    }

  return OK;
```

```
}

static int GaussThetaSchema(int am,double s,NumFunc_1  *p,
    double t,double r,double divid,double sigma,int N,int M,double
    theta,int bound,double *ptprice,double *ptdelta)
{
  int Index,PriceIndex,TimeIndex;
  double k,vv,l,h,z,alpha,beta,gamma,y,alpha1,beta1,gamma1,
    bound1,bound2,upwind_alphacoef;
  double  *Obst,*A,*B,*C,*A1,*B1,*C1,*P,*S;

  /*Memory Allocation*/
  if (N%2==1) N++;
  Obst= malloc((N+1)*sizeof(double));
  if (Obst==NULL)
    return MEMORY_ALLOCATION_FAILURE;
  A= malloc((N+1)*sizeof(double));
  if (A==NULL)
    return MEMORY_ALLOCATION_FAILURE;
  B= malloc((N+1)*sizeof(double));
  if (B==NULL)
    return MEMORY_ALLOCATION_FAILURE;
  C= malloc((N+1)*sizeof(double));
  if (C==NULL)
    return MEMORY_ALLOCATION_FAILURE;
  A1= malloc((N+1)*sizeof(double));
  if (A1==NULL)
    return MEMORY_ALLOCATION_FAILURE;
  B1= malloc((N+1)*sizeof(double));
  if (B1==NULL)
    return MEMORY_ALLOCATION_FAILURE;
  C1= malloc((N+1)*sizeof(double));
  if (C1==NULL)
    return MEMORY_ALLOCATION_FAILURE;
  P= malloc((N+1)*sizeof(double));
  if (P==NULL)
    return MEMORY_ALLOCATION_FAILURE;
  S= malloc((N+1)*sizeof(double));
  if (S==NULL)
    return MEMORY_ALLOCATION_FAILURE;
```

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

/*Space Localisation*/
vv=0.5*SQR(sigma);
z=(r-divid)-vv;
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 of theta-schema*/
alpha=theta*k*(-vv/(h*h)+z/(2.0*h));
beta=1.0+k*theta*(r+2.*vv/(h*h));
gamma=k*theta*(-vv/(h*h)-z/(2.0*h));

for(PriceIndex=1;PriceIndex<=N-1;PriceIndex++)
  {
    A[PriceIndex]=alpha;
    B[PriceIndex]=beta;
    C[PriceIndex]=gamma;
  }

/*Neumann Boundary Condition*/
if (bound==1) {
  B[1]=beta+alpha;
  B[N-1]=beta+gamma;
}

/*Rhs Factor of theta-schema*/
alpha1=k*(1.0-theta)*(vv/(h*h)-z/(2.0*h));
beta1=1.0-k*(1.0-theta)*(r+2.*vv/(h*h));
```

```
gamma1=k*(1.0-theta)*(vv/(h*h)+z/(2.0*h));

for(PriceIndex=1;PriceIndex<=N-1;PriceIndex++)
  {
    A1[PriceIndex]=alpha1;
    B1[PriceIndex]=beta1;
    C1[PriceIndex]=gamma1;
  }

/*Neumann Boundary Condition*/
if (bound==1) {
  B1[1]=beta1+alpha1;
  B1[N-1]=beta1+gamma1;
}

/*Set Gauss*/
for(PriceIndex=N-2;PriceIndex>=1;PriceIndex--)
  B[PriceIndex]=B[PriceIndex]-C[PriceIndex]*A[PriceIndex+
  1]/B[PriceIndex+1];
for(PriceIndex=1;PriceIndex<N;PriceIndex++)
  A[PriceIndex]=A[PriceIndex]/B[PriceIndex];
for(PriceIndex=1;PriceIndex<N-1;PriceIndex++)
  C[PriceIndex]=C[PriceIndex]/B[PriceIndex+1];

/*Terminal Values*/
y=log(s);
for(PriceIndex=1;PriceIndex<N;PriceIndex++) {
  Obst[PriceIndex]=(p->Compute)(p->Par,exp(y-l+(double)
  PriceIndex*h));
  P[PriceIndex]= Obst[PriceIndex];
}

boundary(bound,p,y,l,h,&bound1,&bound2);

/*Finite Difference Cycle*/
for(TimeIndex=1;TimeIndex<=M;TimeIndex++)
  {
    /*Set Rhs*/
    S[1]=B1[1]*P[1]+C1[1]*P[2]+A1[1]*bound1-alpha*bound1;
    for(PriceIndex=2;PriceIndex<N-1;PriceIndex++)
      S[PriceIndex]=A1[PriceIndex]*P[PriceIndex-1]+
```

```
        B1[PriceIndex]*P[PriceIndex]+
        C1[PriceIndex]*P[PriceIndex+1];
    S[N-1]=A1[N-1]*P[N-2]+B1[N-1]*P[N-1]+C1[N-1]*bound2-
  gamma*bound2;

    /*Solve the system*/
    for(PriceIndex=N-2;PriceIndex>=1;PriceIndex--)
      S[PriceIndex]=S[PriceIndex]-C[PriceIndex]*S[PriceI
  ndex+1];

    P[1] =S[1]/B[1];
    for(PriceIndex=2;PriceIndex<N;PriceIndex++)
      P[PriceIndex]=S[PriceIndex]/B[PriceIndex]-A[PriceI
  ndex]*P[PriceIndex-1];

    /*Splitting for the american case*/
    if (am)
      for(PriceIndex=1;PriceIndex<N;PriceIndex++)
        P[PriceIndex]=MAX(Obst[PriceIndex],P[PriceIndex])
  ;
  }

Index=(int) floor ((double)N/2.0);

/*Price*/
*ptprice=P[Index];

/*Delta*/
*ptdelta  = (P[Index+1]-P[Index-1])/(2.0*s*h);

/*Memory Desallocation*/
free(Obst);
free(A);
free(B);
free(C);
free(A1);
free(B1);
free(C1);
free(P);
free(S);
```

```
  return OK;
}

int CALC(FD_Gauss)(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 GaussThetaSchema(ptOpt->EuOrAm.Val.V_BOOL,ptMod->
    S0.Val.V_PDOUBLE,
                          ptOpt->PayOff.Val.V_NUMFUNC_1,pt
    Opt->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->Par[2].Val.V_RGDOUBLE051,Met->Par[3].Val.
    V_ENUM.value,
                          &(Met->Res[0].Val.V_DOUBLE),&(
    Met->Res[1].Val.V_DOUBLE));
}

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

static PremiaEnumMember BoundaryCondMembers[] =
  {
    { "Dirichlet", 0 },
    { "Neumann", 1 },
    { NULL, NULLINT }
};

DEFINE_ENUM(BoundaryCondition, BoundaryCondMembers);
```

```
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;
      Met->Par[2].Val.V_RGDOUBLE=0.5;
      Met->Par[3].Val.V_ENUM.value=1;
      Met->Par[3].Val.V_ENUM.members=&BoundaryCondition;


    }

  return OK;
}

PricingMethod MET(FD_Gauss)=
{
  "FD_Gauss",
  {{"SpaceStepNumber",INT2,{100},ALLOW    },{"TimeStepNumb
    er",INT2,{100},ALLOW},
   {"Theta",RGDOUBLE051,{100},ALLOW},
   {"Boundary Condition",ENUM,{1},ALLOW},
   {" ",PREMIA_NULLTYPE,{0},FORBID}},
  CALC(FD_Gauss),
  {{"Price",DOUBLE,{100},FORBID},
   {"Delta",DOUBLE,{100},FORBID} ,
   {" ",PREMIA_NULLTYPE,{0},FORBID}},
  CHK_OPT(FD_Gauss),
  CHK_split,
  MET(Init)
};
```

References