Help

```
#include <stdlib.h>
#include  "vasicek1d_stdi.h"
#include "error_msg.h"

/*Product*/
static double dt,dr,r_min,r_max;
static double *r_vect;
static double *V,*Vp,*Ps;
static double *beta,*alpha_r,*beta_r,*gamma_r_,*alpha_l,*
    beta_l,*gamma_l;

/*Memory Allocation*/
static int memory_allocation(int Nt,int Ns)
{

  r_vect= malloc((Ns+1)*sizeof(double));
  if (r_vect==NULL)
    return MEMORY_ALLOCATION_FAILURE;

  V= malloc((Ns+1)*sizeof(double));
  if (V==NULL)
    return MEMORY_ALLOCATION_FAILURE;

  Vp= malloc((Ns+1)*sizeof(double));
  if (Vp==NULL)
    return MEMORY_ALLOCATION_FAILURE;

  Ps= malloc((Ns+1)*sizeof(double));
  if (Ps==NULL)
    return MEMORY_ALLOCATION_FAILURE;

  beta= malloc((Ns+1)*sizeof(double));
  if (beta==NULL)
    return MEMORY_ALLOCATION_FAILURE;

  alpha_l= malloc((Ns+1)*sizeof(double));
  if (alpha_l==NULL)
    return MEMORY_ALLOCATION_FAILURE;

  beta_l= malloc((Ns+1)*sizeof(double));
```

```
  if (beta_l==NULL)
    return MEMORY_ALLOCATION_FAILURE;

  gamma_l= malloc((Ns+1)*sizeof(double));
  if (gamma_l==NULL)
    return MEMORY_ALLOCATION_FAILURE;

  alpha_r= malloc((Ns+1)*sizeof(double));
  if (alpha_r==NULL)
    return MEMORY_ALLOCATION_FAILURE;

  beta_r= malloc((Ns+1)*sizeof(double));
  if (beta_r==NULL)
    return MEMORY_ALLOCATION_FAILURE;

  gamma_r_= malloc((Ns+1)*sizeof(double));
  if (gamma_r_==NULL)
    return MEMORY_ALLOCATION_FAILURE;

  return OK;
}

/*Memory Desallocation*/
static void free_memory(int Nt)
{

  free(beta);
  free(alpha_r);
  free(beta_r);
  free(gamma_r_);
  free(alpha_l);
  free(beta_l);
  free(gamma_l);
  free(r_vect);
  free(V);
  free(Vp);
  free(Ps);

  return;
}
```

```
/*Zero Coupon Bond*/
static int zcb_vasicek(int Nt,int Ns)
{
  int i,TimeIndex;

  /*Maturity conditions for pure discount Bond*/
  for(i=1;i<=Ns;i++)
    Ps[i]=1.;

  /*Finite Difference Cycle*/
  for(TimeIndex=Nt-1;TimeIndex>=0;TimeIndex--)
    {
      /*Right factor*/
      for (i=1;i<Ns;i++)
  V[i]=alpha_r[i]*Ps[i-1]+beta_r[i]*Ps[i]+gamma_r_[i]*Ps[
    i+1];

      /*Backward Steps*/
      Vp[Ns-1]=V[Ns-1];
      beta[Ns-1]=beta_l[Ns-1];
      for(i=Ns-2;i>=1;i--)
  {
    beta[i]=beta_l[i]-gamma_l[i]*alpha_l[i+1]/beta[i+1];
    Vp[i]=V[i]-gamma_l[i]*Vp[i+1]/beta[i+1];
  }

      /*Forward Steps*/
      Ps[1]=Vp[1]/beta[1];
      for (i=2;i<Ns;i++)
  Ps[i]=(Vp[i]-alpha_l[i]*Ps[i-1])/beta[i];
    }

  return 1.;
}

/*Zero Bond Computation*/
static int zbo_vasicek1d(double r0,double k,double t0,
    double sigma,double theta,double T,int Nt,int Ns,double cn_thet
    a,double *price)
{
  int i,j;
```

```
double val,val1,sigma2;

/*Space Localisation*/
memory_allocation(Nt,Ns);
sigma2=SQR(sigma);
dt=(T-t0)/(double)Nt;
r_min=-3.;
r_max=3.;
dr=(r_max-r_min)/(double)Ns;
r_vect[0]=r_min;
for(i=0;i<=Ns;i++)
  r_vect[i]=r_min+(double)i*dr;

/*Computation of the Matrix*/
for(i=1;i<Ns;i++)
  {
    /*Computation of Rhs coefficients*/
    alpha_r[i]=(1.-cn_theta)*(0.5*sigma2*(dt/SQR(dr))-0.5
  *k*(theta-r_vect[i])*(dt/dr));
    beta_r[i]=1.-(1.-cn_theta)*(sigma2*(dt/SQR(dr))+r_vec
  t[i]*dt);
    gamma_r_[i]=(1.-cn_theta)*(0.5*sigma2*(dt/SQR(dr))+0.
  5*k*(theta-r_vect[i])*(dt/dr));

    /*Computation of Lhs coefficients*/
    alpha_l[i]=cn_theta*(-0.5*sigma2*(dt/SQR(dr))+0.5*k*(
  theta-r_vect[i])*(dt/dr));
    beta_l[i]=1.+cn_theta*(sigma2*(dt/SQR(dr))+r_vect[i]*
  dt);
    gamma_l[i]=cn_theta*(-0.5*sigma2*(dt/SQR(dr))-0.5*k*(
  theta-r_vect[i])*(dt/dr));
  }

/*Compute Zero Coupon Prices*/
zcb_vasicek(Nt,Ns);

/*Linear Interpolation*/
j=0;
while(r_vect[j]<r0)
  j++;
val= Ps[j];
```

```
  val1= Ps[j-1];

  /*Price*/
  *price=val+(val-val1)*(r0-r_vect[j])/(r_vect[j]-r_vect[j-
    1]);

  /*Memory Disallocation*/
  free_memory(Nt);


  return OK;
}

int CALC(FD_GaussZCBond)(void *Opt,void *Mod,PricingMethod
    *Met)
{
  TYPEOPT* ptOpt=(TYPEOPT*)Opt;
  TYPEMOD* ptMod=(TYPEMOD*)Mod;

  return zbo_vasicek1d(ptMod->r0.Val.V_PDOUBLE,ptMod->k.Val
    .V_DOUBLE,ptMod->T.Val.V_DATE,ptMod->Sigma.Val.V_PDOUBLE,
    ptMod->theta.Val.V_PDOUBLE,ptOpt->BMaturity.Val.V_DATE,Met-
    >Par[0].Val.V_INT,Met->Par[1].Val.V_INT,Met->Par[2].Val.V_
    RGDOUBLE,&(Met->Res[0].Val.V_DOUBLE));
}

static int CHK_OPT(FD_GaussZCBond)(void *Opt, void *Mod)
{

  if ((strcmp(((Option*)Opt)->Name,"ZeroCouponBond")==0))
    return OK;
  else
    return WRONG;
}

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

```
      Met->Par[0].Val.V_INT2=300;
      Met->Par[1].Val.V_INT2=300;
      Met->Par[2].Val.V_RGDOUBLE=0.5;


    }
  return OK;
}

PricingMethod MET(FD_GaussZCBond)=
{
  "FD_Gauss_Vasicek1d_ZCBond",
  {{"SpaceStepNumber",INT2,{100},ALLOW  },{"TimeStepNumber"
    ,INT2,{100},ALLOW},{"Theta",RGDOUBLE051,{100},ALLOW},
   {" ",PREMIA_NULLTYPE,{0},FORBID}},
  CALC(FD_GaussZCBond),
  {{"Price",DOUBLE,{100},FORBID},{" ",PREMIA_NULLTYPE,{0},
    FORBID}},
  CHK_OPT(FD_GaussZCBond),
  CHK_ok,
  MET(Init)
} ;
```

References