Help

```
#include <stdlib.h>
#include  "cir1d_stdi.h"

/*Product*/
static double dt, dr, r_min, r_max;
static double *r_vect, *disc, * *Option_Price, * *Ps;
static double *pu, *pm, *pd;
static long Ns, NtO;

/* static int j_max;*/

/*Memory Allocation*/
static void memory_allocation(long Nt)
{
  int i;

  if ((r_vect = malloc(sizeof(double) * (Ns + 1))) == NULL)
    {
      printf("Allocation error");
      exit(1);
    }
  if ((disc = malloc(sizeof(double) * (Ns + 1))) == NULL)
    {
      printf("Allocation error");
      exit(1);
    }
  if ((pu = malloc(sizeof(double) * (Ns + 1))) == NULL)
    {
      printf("Allocation error");
      exit(1);
    }
  if ((pm = malloc(sizeof(double) * (Ns + 1))) == NULL)
    {
      printf("Allocation error");
      exit(1);
    }
  if ((pd = malloc(sizeof(double) * (Ns + 1))) == NULL)
    {
      printf("Allocation error");
```

```
      exit(1);
    }
  if ((Ps = malloc(sizeof(double *) * (Nt + 1))) == NULL)
    {
      printf("Allocation error");
      exit(1);
    }
  if ((Option_Price = malloc(sizeof(double *) * (Nt + 1))) == NULL)
    {
      printf("Allocation error");
      exit(1);
    }
  for (i = 0; i <= Nt; i++)
    {
      Option_Price[i] = malloc(sizeof(double) * (Ns + 1));
    }
  for (i = 0; i <= Nt; i++)
    {
      Ps[i] = malloc(sizeof(double) * (Ns + 1));
    }


  return;
}

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

  free(r_vect);
  free(pu);
  free(pm);
  free(pd);
  free(disc);

  for (i = 0; i < Nt + 1; i++)
    free(Ps[i]);
  free(Ps);

  for (i = 0; i < Nt + 1; i++)
```

```
    free(Option_Price[i]);
  free(Option_Price);

  return;
}

/*Compute probabilities*/
static int init_prob(double k, double sigma, double theta, double T, double t0,
{
  double df;
  int j;
  double beta, alpha1, alpha2;

  /*Time and Space Step*/
  dt = (T - t0) / (double)Nt;
  dr = sigma * sqrt(3. / 4.*dt);

  /*Localization*/
  alpha1 = (4.*k * theta - SQR(sigma)) / 8.;
  alpha2 = k / 2.;
  beta = dr / (2.*dt);
  r_min = (-beta + sqrt(SQR(beta) + 4.*alpha1 * alpha2)) / (2.*alpha2);
  r_max = (beta + sqrt(SQR(beta) + 4.*alpha1 * alpha2)) / (2.*alpha2);
  Ns = (int)ceil((r_max - r_min) / dr);
  memory_allocation(Nt);

  /*Compute probabilities*/
  for (j = 0; j <= Ns; j++)
    {
      r_vect[j] = r_min + (double)j * dr;
      disc[j] = exp(-SQR(r_vect[j]) * dt);
      df = ((4.*k * theta - SQR(sigma)) / (8.*r_vect[j]) - r_vect[j] * k / 2.) *

      /*Boundary*/
      if (j == 0)
        {
          pu[j] = 1. / 6. + (SQR(df) - df) / 2.;
          pm[j] = df - 2.*pu[j];
          pd[j] = 1. - pu[j] - pm[j];
        }
      else if (j == Ns)
```

```
        {
          pd[j] = 1. / 6. + (SQR(df) + df) / 2.;
          pm[j] = -df - 2.*pd[j];
          pu[j] = 1. - pd[j] - pm[j];

        }
      /*Not Boundary*/
      else
        {
          pu[j] = 1. / 6. + (SQR(df) + df) / 2.;
          pd[j] = pu[j] - df;
          pm[j] = 1. - pu[j] - pd[j];
        }
    }

  return OK;
}

/*Compute Coupon Bearing*/
static int zcb_cir(long Nt0, long Nt, double K, double periodicity, double first
{
  int i, j, z;
  /*Maturity conditions for CB*/
  for (j = 0; j <= Ns; j++)
    Ps[Nt][j] = 1. + K * periodicity;

  /*Dynamic Programming*/
  for (i = Nt - 1; i >= Nt0; i--)
    for (j = 0; j <= Ns; j++)
      {
        if (j == 0)
          Ps[i][j] = disc[j] * (pu[j] * Ps[i + 1][j + 2] + pm[j] * Ps[i + 1][j +
        else if (j == Ns)
          Ps[i][j] = disc[j] * (pd[j] * Ps[i + 1][j - 2] + pm[j] * Ps[i + 1][j -
        else
          Ps[i][j] = disc[j] * (pu[j] * Ps[i + 1][j + 1] + pm[j] * Ps[i + 1][j]
        /*Coupon adjustment*/
        for (z = 0; z < nb_coupon; z++)
          {
            if ((i != 0) && (fabs((double)i * dt - (first_payement + (double)z *
              {
```

```
                Ps[i][j] += K * periodicity;
              }
          }
      }
  return 1.;
}

/*Swaption=Option on Coupon-Bearing Bond*/
static int swaption_cir1d(double r0, double k, double t0, double sigma, double t
{
  int i, j, nb_coupon, Nt;
  double val, val1, tmp, first_payement;

  /*Compute probabilities*/
  Nt = NtY * (long)((T - t0) / periodicity);
  init_prob(k, sigma, theta, T, t0, Nt);

  /*Number of Step for the Option*/
  NtO = NtY * (long)((t - t0) / periodicity);

  /*Compute Zero Coupon Prices*/
  first_payement = t + periodicity;
  nb_coupon = (int)((T - first_payement) / periodicity);
  zcb_cir(NtO, Nt, K, periodicity, first_payement, nb_coupon);

  /*Maturity conditions for the option*/
  tmp = p->Par[0].Val.V_DOUBLE;
  p->Par[0].Val.V_DOUBLE = 1.;
  for (j = 0; j <= Ns; j++)
    Option_Price[NtO][j] = (p->Compute)(p->Par, Ps[NtO][j]);

  /*Explicit Finite Difference Cycle*/
  for (i = NtO - 1; i >= 0; i--)
    for (j = 0; j <= Ns; j++)
      {
        /*Boundary*/
        if (j == 0)
          Option_Price[i][j] = disc[j] * (pu[j] * Option_Price[i + 1][j + 2] + p
        else if (j == Ns)
          Option_Price[i][j] = disc[j] * (pd[j] * Option_Price[i + 1][j - 2] + p
```

```
        /*Not Boundary*/
        else
          Option_Price[i][j] = disc[j] * (pu[j] * Option_Price[i + 1][j + 1] + p

        /*American Case*/
        /*if(am)
          Option_Price[i][j]=MAX(Option_Price[i][j],(p->Compute)(p->Par,Ps[i][j]
      }

  /*Linear Interpolation*/
  j = 0;
  while (SQR(r_vect[j]) < r0)
    j++;
  val = Option_Price[0][j];
  val1 = Option_Price[0][j - 1];

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

  /*Memory Disallocation*/
  p->Par[0].Val.V_DOUBLE = tmp;
  free_memory(Nt);


  return OK;
}

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


  return swaption_cir1d(ptMod->r0.Val.V_PDOUBLE, ptMod->k.Val.V_DOUBLE, ptMod->T
                        ptMod->theta.Val.V_PDOUBLE, ptOpt->BMaturity.Val.V_DATE,
                        ptOpt->EuOrAm.Val.V_BOOL, ptOpt->Nominal.Val.V_PDOUBLE,
}


static int CHK_OPT(FD_SWAPTION)(void *Opt, void *Mod)
```

```
{

  if ((strcmp(((Option *)Opt)->Name, "PayerSwaption") == 0) || (strcmp(((Option
    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_LONG = 400;


    }
  return OK;
}

PricingMethod MET(FD_SWAPTION) =
{
  "FD_Explicit_Cir1d_SWAPTION",
  { {"TimeStepNumber", LONG, {100}, ALLOW},
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CALC(FD_SWAPTION),
  {{"Price", DOUBLE, {100}, FORBID}, {" ", PREMIA_NULLTYPE, {0}, FORBID}},
  CHK_OPT(FD_SWAPTION),
  CHK_ok,
  MET(Init)
} ;
```