Help

```
/* F. Dubois et T. Lelievre  */
/* revu par T. Lelievre décembre 2003    */
/* revu par T. Lelievre janvier 2004    */

#include <stdlib.h>
#include  "
href../../../mod/bs1d/bs1d_pad/bs1d_pad_h_src.pdfbs1d_pad.h"

#include <stdio.h>
#include <
href../../../common/math/cdo/cdo_math_h_src.pdfmath.h>
#include "
href../../../common/error_msg_h_src.pdferror_msg.h"

#define EPSILON_RODSHI 1.e-10
/* Inversion LU matrice tridiagonale */

void Initinv_Crout(double *A_inf, double *A_diag, double *A_up, int n_0 , int di
{

  int i;

  L_diag[0] = A_diag[n_0 + 0];
  U_up[0] = A_up[n_0 + 0] / L_diag[0];

  for (i = 1; i <= dim - 2; i++)
    {
      L_inf[i - 1] = A_inf[n_0 + i];
      L_diag[i] = A_diag[n_0 + i] - L_inf[i - 1] * U_up[i - 1];
      U_up[i] = A_up[n_0 + i] / L_diag[i];
    }
  L_inf[dim - 2] = A_inf[n_0 + dim - 1];
  L_diag[dim - 1] = A_diag[n_0 + dim - 1] - L_inf[dim - 2] * U_up[dim - 2];

}

void inv_Crout(double *v, double *sol, double *A_inf, double *A_diag, double *A_
{
```

```
  int i;

  Initinv_Crout(A_inf, A_diag, A_up, n_0, dim, L_inf, L_diag, U_diag, U_up);

  /* On resout d'abord L sol = v */
  /* descente */
  sol[n_0] = v[n_0] / L_diag[0];
  for (i = 1; i < dim; i++)
    sol[n_0 + i] = (v[n_0 + i] - L_inf[i - 1] * sol[n_0 + i - 1]) / L_diag[i];

  /* Puis on resout U sol(new) = sol */
  /* remontee */
  for (i = dim - 2; i >= 0; i--)
    sol[n_0 + i] = (sol[n_0 + i] - U_up[i] * sol[n_0 + i + 1]);

}

/* Construction des matrices au temps t_n */
void Construit_Matrices_CN(int n, double *Mn_inf, double *Mn_diag, double *Mn_up
{

  double val1, val2, facmul;
  int i;

  /* Initialisation des matrices */

  /*
    ( - Id / dt + 0.5 * A(t_n} + 0.5 * B_{t_n} )
    ( - Id / dt - 0.5 * A(t_n} - 0.5 * B_{t_n} )
    En fait, ce n'est pas exactement Id : il y a un 0.5 en bas
  */

  /* Mn */
  for (i = n + 1; i <= N + J; i++)
    {
      Mn_inf[i] = 0.;
      Mn_diag[i] = 0.;
      Mn_up[i] = 0.;
    }
```

```
/* M(n+1) */
for (i = n + 1; i <= N + J; i++)
  {
    Mn_plus_1_inf[i] = 0.;
    Mn_plus_1_diag[i] = 0.;
    Mn_plus_1_up[i] = 0.;
  }

/* Mn */
facmul = 0.5 * (sigma * sigma / 2);
for (i = n + 1; i < N + J; i++)
  {
    val1 = facmul * (i - 0.5 - n) * (i - 0.5 - n);
    val2 = facmul * (i + 0.5 - n) * (i + 0.5 - n);
    Mn_inf[i] += val1;
    Mn_diag[i] += -val2 - val1;
    Mn_up[i] += val2;
  }
val1 = facmul * (N + J - 0.5 - n) * (N + J - 0.5 - n);
val2 = facmul * (N + J + 0.5 - n) * (N + J + 0.5 - n);
Mn_inf[N + J] += val1;
Mn_diag[N + J] += -val1;

/* M(n+1) */
facmul = -0.5 * (sigma * sigma / 2);
for (i = n + 1; i < N + J; i++)
  {
    val1 = facmul * (i - 0.5 - (n + 1)) * (i - 0.5 - (n + 1));
    val2 = facmul * (i + 0.5 - (n + 1)) * (i + 0.5 - (n + 1));
    Mn_plus_1_inf[i] += val1;
    Mn_plus_1_diag[i] += -val2 - val1;
    Mn_plus_1_up[i] += val2;
  }
val1 = facmul * (N + J - 0.5 - (n + 1)) * (N + J - 0.5 - (n + 1));
val2 = facmul * (N + J + 0.5 - (n + 1)) * (N + J + 0.5 - (n + 1));
Mn_plus_1_inf[N + J] += val1;
Mn_plus_1_diag[N + J] += -val1;

/* Mn */
facmul = 0.5 * (-(r + sigma * sigma) / 2);
for (i = n + 1; i < N + J; i++)
```

```
    {
      val1 = facmul * (i - 0.5 - n);
      val2 = facmul * (i + 0.5 - n);
      Mn_inf[i] += -val1;
      Mn_diag[i] += -facmul;
      Mn_up[i] += val2;
    }
  val1 = facmul * (N + J - 0.5 - n);
  val2 = facmul * (N + J + 0.5 - n);
  Mn_inf[N + J] += -val1;
  Mn_diag[N + J] += val1;

  /* M(n+1) */
  facmul = -0.5 * (-(r + sigma * sigma) / 2);
  for (i = n + 1; i < N + J; i++)
    {
      val1 = facmul * (i - 0.5 - (n + 1));
      val2 = facmul * (i + 0.5 - (n + 1));
      Mn_plus_1_inf[i] += -val1;
      Mn_plus_1_diag[i] += -facmul;
      Mn_plus_1_up[i] += val2;
    }
  val1 = facmul * (N + J - 0.5 - (n + 1));
  val2 = facmul * (N + J + 0.5 - (n + 1));
  Mn_plus_1_inf[N + J] += -val1;
  Mn_plus_1_diag[N + J] += val1;

  /* Mn et M(n+1) */
  for (i = n + 1; i < N + J; i++)
    {
      Mn_diag[i] += -1. / dt;
      Mn_plus_1_diag[i] += -1. / dt;

    }
  Mn_diag[N + J] += -0.5 / dt;
  Mn_plus_1_diag[N + J] += -0.5 / dt;

}

static int Fixed_RodgerShi_2(double pseudo_stock, double pseudo_strike, NumFunc_
{
```

```
/* Matrices */
double *Mn_inf, * Mn_diag, * Mn_up;
double *Mn_plus_1_inf, * Mn_plus_1_diag, * Mn_plus_1_up;
double *L_diag, * L_inf, * U_up, *U_diag;

/* Vecteur */
double *Fold, * Fnew;
double *scd_membre;

/* Parametres financiers */
double T = maturite;
double r = taux_d_interet - divid;
double S0 = pseudo_stock;
double K = pseudo_strike;

/* Parametres numeriques */

int N = nt; /* nbre de pas de temps */
int J = N / 2; /* nbre de pas d'espaces en plus (à droite de 1) */
double dt = T / N;
double dx = dt / T;
/* double xmax=(N+J)*dx;*/

/* dans les itérations en temps */
int timestep;
double t;
double CL;
int n_0, dim;

/* Calcul du prix */
double x;
int i;
double a, b, c, d;
double aprime, bprime, cprime, dprime;

/* cas d'un taux d'interet nul */
int divid_zero = 0;
if (fabs(r) < EPSILON_RODSHI)
  divid_zero = 1;

/*Memory Allocation*/
```

```
L_diag = malloc((N + J + 1) * sizeof(double));
if (L_diag == NULL)
  return MEMORY_ALLOCATION_FAILURE;

L_inf = malloc((N + J + 1) * sizeof(double));
if (L_inf == NULL)
  return MEMORY_ALLOCATION_FAILURE;

U_diag = malloc((N + J + 1) * sizeof(double));
if (U_diag == NULL)
  return MEMORY_ALLOCATION_FAILURE;
U_up = malloc((N + J + 1) * sizeof(double));
if (U_up == NULL)
  return MEMORY_ALLOCATION_FAILURE;
Mn_inf = malloc((N + J + 1) * sizeof(double));
if (Mn_inf == NULL)
  return MEMORY_ALLOCATION_FAILURE;
Mn_diag = malloc((N + J + 1) * sizeof(double));
if (Mn_diag == NULL)
  return MEMORY_ALLOCATION_FAILURE;
Mn_up = malloc((N + J + 1) * sizeof(double));
if (Mn_up == NULL)
  return MEMORY_ALLOCATION_FAILURE;
Mn_plus_1_inf = malloc((N + J + 1) * sizeof(double));
if (Mn_plus_1_inf == NULL)
  return MEMORY_ALLOCATION_FAILURE;
Mn_plus_1_diag = malloc((N + J + 1) * sizeof(double));
if (Mn_plus_1_diag == NULL)
  return MEMORY_ALLOCATION_FAILURE;
Mn_plus_1_up = malloc((N + J + 1) * sizeof(double));
if (Mn_plus_1_up == NULL)
  return MEMORY_ALLOCATION_FAILURE;
Fold = malloc((N + J + 1) * sizeof(double));
if (Fold == NULL)
  return MEMORY_ALLOCATION_FAILURE;
Fnew = malloc((N + J + 1) * sizeof(double));
if (Fnew == NULL)
  return MEMORY_ALLOCATION_FAILURE;
scd_membre = malloc((N + J + 1) * sizeof(double));
if (scd_membre == NULL)
  return MEMORY_ALLOCATION_FAILURE;
```

```
/* Condition finale : nulle */

for (i = 0; i <= N + J; i++)
  Fold[i] = MAX(0, (1 - i * dx));

/* Iterations en temps */

for (timestep = N - 1; timestep >= 0; timestep--)
  {
    t = timestep * dt;

    n_0 = timestep + 1; /* première coordonnée inconnue */
    dim = N + J - timestep; /* dimension du système à résoudre */

    /* Construction des matrices */
    Construit_Matrices_CN(timestep, Mn_inf, Mn_diag, Mn_up, Mn_plus_1_inf, Mn_

    /* second membre */
    for (i = n_0; i < N + J; i++)
      scd_membre[i] = Fold[i - 1] * Mn_plus_1_inf[i] + Fold[i] * Mn_plus_1_dia
    scd_membre[N + J] = Fold[N + J - 1] * Mn_plus_1_inf[N + J] + Fold[N + J] *
    /* Definition de la condition aux limites de Dirichlet (à gauche) */
    if (!divid_zero)
      CL = (1 / (r * T)) * (1 - exp(-r * (T - t)));
    else
      CL = (T - t) / T;
    scd_membre[n_0] -= CL * Mn_inf[n_0];

    /* calcul de la valeur au temps tn */
    inv_Crout(scd_membre, Fnew, Mn_inf, Mn_diag, Mn_up, n_0, dim, L_inf, L_dia

    /* On passe à la suite et on complète à gauche */
    if (!divid_zero)
      for (i = 0; i <= timestep; i++)
        Fold[i] = (1 / (r * T)) * (1 - exp(-r * (T - t))) - (i * dx - t / T) *
    else
      for (i = 0; i <= timestep; i++)
        Fold[i] = (T - t) / T - (i * dx - t / T);

    for (i = n_0; i <= N + J; i++)
```

```
    Fold[i] = Fnew[i];
  }

/* Calcul du prix et du delta */
for (i = 0; i <= N + J; i++)
  Fnew[i] = Fold[i];

x = K / S0;
i = (int)floor((K / S0) / dx);

/* Interpolation d'ordre 1 */
/*
  a=((i+1)*dx-x)/dx;
  b=(x-i*dx)/dx;

  if ((p->Compute) == &Call_OverSpot2)
  *ptprice=exp(-divid*T)*S0*(a*Fnew[i]+b*Fnew[i+1]);
  else
  if (!divid_zero)
  *ptprice=exp(-divid*T)*S0*(a*Fnew[i]+b*Fnew[i+1])-exp(-divid*T)*exp(-r*T)*((
  else
  *ptprice=exp(-divid*T)*S0*(a*Fnew[i]+b*Fnew[i+1])-exp(-divid*T)*(S0-K);

  if ((p->Compute) == &Call_OverSpot2)
  *ptdelta=exp(-divid*T)*((a*Fnew[i]+b*Fnew[i+1])-(K/S0)*(Fnew[i+1]-Fnew[i])/d
  else
  if (!divid_zero)
  *ptdelta=exp(-divid*T)*((a*Fnew[i]+b*Fnew[i+1])-(K/S0)*(Fnew[i+1]-Fnew[i])/(
  else
  *ptdelta=exp(-divid*T)*((a*Fnew[i]+b*Fnew[i+1])-(K/S0)*(Fnew[i+1]-Fnew[i])/(
  */

/* Interpolation d'ordre 3 */

a = (i * dx - x) * ((i + 1) * dx - x) * ((i + 2) * dx - x) / (dx * 2 * dx * 3
b = (x - (i - 1) * dx) * ((i + 1) * dx - x) * ((i + 2) * dx - x) / (dx * dx *
c = (x - (i - 1) * dx) * (x - i * dx) * ((i + 2) * dx - x) / (2 * dx * dx * dx
d = (x - (i - 1) * dx) * (x - i * dx) * (x - (i + 1) * dx) / (3 * dx * 2 * dx

aprime = ((-1.) * ((i + 1) * dx - x) * ((i + 2) * dx - x) + (i * dx - x) * (-1
bprime = ((1.) * ((i + 1) * dx - x) * ((i + 2) * dx - x) + (x - (i - 1) * dx)
```

```
  cprime = ((1.) * (x - i * dx) * ((i + 2) * dx - x) + (x - (i - 1) * dx) * (1.)
  dprime = ((1.) * (x - i * dx) * (x - (i + 1) * dx) + (x - (i - 1) * dx) * (1.)

  if ((p->Compute) == &Call_OverSpot2)
    *ptprice = exp(-divid * T) * S0 * (a * Fnew[i - 1] + b * Fnew[i] + c * Fnew[
  else if (!divid_zero)
    *ptprice = exp(-divid * T) * S0 * (a * Fnew[i - 1] + b * Fnew[i] + c * Fnew[
  else
    *ptprice = exp(-divid * T) * S0 * (a * Fnew[i - 1] + b * Fnew[i] + c * Fnew[

  if ((p->Compute) == &Call_OverSpot2)
    *ptdelta = exp(-divid * T) * ((a * Fnew[i - 1] + b * Fnew[i] + c * Fnew[i +
  else if (!divid_zero)
    *ptdelta = exp(-divid * T) * ((a * Fnew[i - 1] + b * Fnew[i] + c * Fnew[i +
  else
    *ptdelta = exp(-divid * T) * ((a * Fnew[i - 1] + b * Fnew[i] + c * Fnew[i +


  /*Memory Desallocation*/

  free(L_diag);
  free(L_inf);
  free(U_diag);
  free(U_up);
  free(Mn_inf);
  free(Mn_diag);
  free(Mn_up);
  free(Mn_plus_1_inf);
  free(Mn_plus_1_diag);
  free(Mn_plus_1_up);
  free(Fold);
  free(Fnew);
  free(scd_membre);

  return OK;
}
int CALC(FD_FixedAsian_RodgerShi2)(void *Opt, void *Mod, PricingMethod *Met)
{
  TYPEOPT *ptOpt = (TYPEOPT *)Opt;
  TYPEMOD *ptMod = (TYPEMOD *)Mod;
```

```
  int return_value;
  double r, divid, time_spent, pseudo_spot, pseudo_strike;
  double t_0, T_0;

  r = log(1. + ptMod->R.Val.V_DOUBLE / 100.);
  divid = log(1. + ptMod->Divid.Val.V_DOUBLE / 100.);

  T_0 = ptMod->T.Val.V_DATE;
  t_0 = (ptOpt->PathDep.Val.V_NUMFUNC_2)->Par[0].Val.V_PDOUBLE;

  if (T_0 < t_0)
    {
      Fprintf(TOSCREEN, "T_0 < t_0, untreated case\ n\ n\ n");
      return_value = WRONG;
    }
  /* Case t_0 <= T_0 */
  else
    {
      time_spent = (ptMod->T.Val.V_DATE - (ptOpt->PathDep.Val.V_NUMFUNC_2)->Par[
                   (ptOpt->Maturity.Val.V_DATE - (ptOpt->PathDep.Val.V_NUMFUNC_2
      pseudo_spot = (1. - time_spent) * ptMod->S0.Val.V_PDOUBLE;
      pseudo_strike = (ptOpt->PayOff.Val.V_NUMFUNC_2)->Par[0].Val.V_PDOUBLE - ti

      if (pseudo_strike <= 0.)
        {
          Fprintf(TOSCREEN, "ANALYTIC FORMULA\ n\ n\ n");
          return_value = Analytic_KemnaVorst(pseudo_spot, pseudo_strike, time_sp
        }
      else
        return_value = Fixed_RodgerShi_2(pseudo_spot, pseudo_strike, ptOpt->PayO
    }
  return return_value;
}

static int CHK_OPT(FD_FixedAsian_RodgerShi2)(void *Opt, void *Mod)
{
  if ((strcmp(((Option *)Opt)->Name, "AsianCallFixedEuro") == 0) || (strcmp(((Op
    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 = 500;
      Met->HelpFilenameHint = "FD_FixedAsian_RodgerShi2";

    }

  return OK;
}

PricingMethod MET(FD_FixedAsian_RodgerShi2) =
{
  "FD_FixedAsian_RodgerShi2",
  { {"TimeStepNumber", INT2, {1000}, ALLOW},
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CALC(FD_FixedAsian_RodgerShi2),
  { {"Price", DOUBLE, {100}, FORBID},
    {"Delta", DOUBLE, {100}, FORBID} ,
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CHK_OPT(FD_FixedAsian_RodgerShi2),
  CHK_ok,
  MET(Init)
};
```