Help

```
extern "C"{
#include  "hes1d_std.h"

#include "math/highdim_solver/laspack/highdim_vector.h"
#include "math/highdim_solver/laspack/qmatrix.h"
#include "math/highdim_solver/laspack/highdim_matrix.h"
#include "math/highdim_solver/laspack/operats.h"

#include "math/highdim_solver/fd_solver.h"
#include "math/highdim_solver/fd_operators.h"
#include "math/highdim_solver/fd_operators_easy.h"
#include "math/highdim_solver/error.h"
}
#include <cmath>
using namespace std;

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion <
     (2008+2) //The "#else" part of the code will be freely av
    ailable after the (year of creation of this file + 2)
#else

typedef struct _Model
{
  // Independent model parameters
  double kappa, theta, sigma, rho, r, V0, S0, K, T;
  unsigned N1,N2;      // Number of grid points per dimens
    ion

  // Solution domain definition
  double xL,xR;
  double yL,yR;

  // Offset
  int offx, offy;

  // Payoff
  double (*boundary)(struct _Model *, double, double);

} HestonModel;
```

```
static HestonModel M_Heston;

static void setup(HestonModel *m)
{
  m->xL = 1;
  m->xR = 800;
  m->yL = 0.000001;
  m->yR = m->yL + 8.0*m->V0;

  m->offx = (int)floor((m->S0-m->xL)*(m->N1-1)/(m->xR-m->x
    L));
  m->offy = (int)floor((m->V0-m->yL)*(m->N2-1)/(m->yR-m->yL
    ));

  m->xR = (m->N1-1)*(m->S0-m->xL)/m->offx+m->xL;
  m->yR = (m->N2-1)*(m->V0-m->yL)/m->offy+m->yL;
}

// Model to solution space
//
/*static void c_m2s(HestonModel *m, double v1, double v2,
    double *w1, double *w2)
{
  if(w1) *w1 = (v1-m->xL)/(m->xR-m->xL);
  if(w2) *w2 = (v2-m->yL)/(m->yR-m->yL);
}*/

static void v_m2s(HestonModel *m, double t, double V,
    double *W)
{
  *W = exp(m->r*(m->T-t))*V;
}

/*static void m2s(HestonModel *m, double t, double v1,
    double v2, double V,
                    double *w1, double *w2, double *W)
{
  c_m2s(m,v1,v2,w1,w2);
  v_m2s(m,t,V,W);
}*/
```

```
// Solution to model space

static void c_s2m(HestonModel *m, double w1, double w2,
    double *v1, double *v2)
{
  if(v1) *v1 = (m->xR-m->xL)*w1+m->xL;
  if(v2) *v2 = (m->yR-m->yL)*w2+m->yL;
}

static void v_s2m(HestonModel *m, double tau, double W,
    double *V)
{
  *V = exp(-1.0*m->r*tau)*W;
}
/*
static void s2m(HestonModel *m, double tau, double w1,
    double w2, double W,
                   double *v1, double *v2, double *V)
{
  c_s2m(m,w1,w2,v1,v2);
  v_s2m(m,tau,W,V);
}*/


static double call_boundary(HestonModel *m, double t,
    double S)
{
  // Call artificial boundary condition
  double v = S-m->K*exp(-m->r*(m->T-t));

  return v > 0 ? v : 0;
}

static double put_boundary(HestonModel *m, double t,
    double S)
{
  // Call artificial boundary condition
  double v = m->K*exp(-m->r*(m->T-t))-S;

  return v > 0 ? v : 0;
}
```

```
static double payoff(HestonModel *m, double S)
{
  return m->boundary(m,m->T,S);
}


/////////////////////////////////////////////////////
// Initial & boundary conditions
//
static int ic_f_next_elem(struct _FDSolver *s, FDSolverVec
    torFiller *f,
                          unsigned *c, double *v)
{
  double S;

  c_s2m(&M_Heston,(double)c[0]/(s->size[0]-1),0,&S,NULL);
  v_m2s(&M_Heston,M_Heston.T-s->t,payoff(&M_Heston,S),v);

  return 0;
}

static int b_f_next_elem(struct _FDSolver *s, FDSolverVec
    torFiller *f,
                         unsigned *c, double *v)
{
  double S;

  c_s2m(&M_Heston,(double)c[0]/(s->size[0]-1),0,&S,NULL);
  v_m2s(&M_Heston,M_Heston.T-s->t,M_Heston.boundary(&M_
    Heston,M_Heston.T-s->t,S),v);

  return 0;
}

/////////////////////////////////////////////////////
// Equation definition
//

static void eq_first_def(FDOperatorJam *j)
```

```
{
  unsigned k;

  for(k=0; k < j->dim; k++)
    FIRST_SPATIAL_DERIVATIVE_CENTERED_MASK(j,k);
}

static int eq_first_apply(FDSolver *s, FDOperatorJam *j, un
    signed *c, void *d,
                        double factor)
{
  double x,y;
  double wx,wy;

  wx = M_Heston.xR - M_Heston.xL;
  wy = M_Heston.yR - M_Heston.yL;

  c_s2m(&M_Heston,(double)c[0]/(s->size[0]-1),(double)c[1]/
    (s->size[1]-1), &x, &y);

  FIRST_SPATIAL_DERIVATIVE_CENTERED_SET(j,0,factor*x*M_
    Heston.r/wx*(M_Heston.N1-1)*s->deltaT);
  FIRST_SPATIAL_DERIVATIVE_CENTERED_SET(j,1,factor*M_
    Heston.kappa*(M_Heston.theta-y)/wy*(M_Heston.N2-1)*s->deltaT);

  return 0;
}

static void eq_second_def(FDOperatorJam *j)
{
  unsigned k,h;

  for(k=0; k < j->dim; k++)
    UNIFORM_SECOND_SPATIAL_DERIVATIVE_CENTERED_MASK(j,k);

  for(h=1; h<j->dim; h++)
    for(k=0; k<h; k++)
      MIXED_SECOND_SPATIAL_DERIVATIVE_CENTERED_BOUCHUT_MAS
    K(j,h,k);
}
```

```
static int eq_second_apply(FDSolver *s, FDOperatorJam *j,
    unsigned *c, void *d,
                    double factor)
{
  double x,y;
  double wx,wy;

  wx = M_Heston.xR - M_Heston.xL;
  wy = M_Heston.yR - M_Heston.yL;

  c_s2m(&M_Heston,(double)c[0]/(s->size[0]-1),(double)c[1]/
    (s->size[1]-1), &x, &y);

  UNIFORM_SECOND_SPATIAL_DERIVATIVE_CENTERED_SET(j,0,
    factor*0.5*y*pow(x/wx*(M_Heston.N1-1),2)*s->deltaT);

  UNIFORM_SECOND_SPATIAL_DERIVATIVE_CENTERED_SET(j,1,
    factor*0.5*y*pow(M_Heston.sigma/wy*(M_Heston.N2-1),2)*
    s->deltaT);   // I think it should be V^2

  MIXED_SECOND_SPATIAL_DERIVATIVE_CENTERED_BOUCHUT_SET(j,1,
    0,
    factor*M_Heston.rho*M_Heston.sigma*x*y/(wx*wy)*(M_
    Heston.N1-1)*(M_Heston.N2-1)*s->deltaT);

  return 0;
}

/////////////////////////////////////////////////////
// Explicit scheme
//
static int ex_eq_def_c(FDOperatorJam *j, void *d)
{
  FIRST_TIME_DERIVATIVE_FORWARD_MASK(j);

  eq_first_def(j);
  eq_second_def(j);

  return 0;
}
```

```
static int ex_eq_apply_c(FDSolver *s, FDOperatorJam *j, un
    signed *c, void *d)
{
  FIRST_TIME_DERIVATIVE_FORWARD_SET(j,1.);

  eq_first_apply(s,j,c,d,1.0);
  eq_second_apply(s,j,c,d,1.0);

  return 0;
}

static int ex_eq_def_n(FDOperatorJam *j, void *d)
{
  FIRST_TIME_DERIVATIVE_FORWARD_MASK(j);

  return 0;
}

static int ex_eq_apply_n(FDSolver *s, FDOperatorJam *j, un
    signed *c, void *d)
{
  FIRST_TIME_DERIVATIVE_FORWARD_SET(j,1.);

  return 0;
}

/////////////////////////////////////////////////////
// Crank-Nicolson scheme
//
static int cn_eq_def_c(FDOperatorJam *j, void *d)
{
  FIRST_TIME_DERIVATIVE_FORWARD_MASK(j);

  eq_first_def(j);
  eq_second_def(j);

  return 0;
}

static int cn_eq_apply_c(FDSolver *s, FDOperatorJam *j, un
    signed *c, void *d)
```

```
{
  FIRST_TIME_DERIVATIVE_FORWARD_SET(j,1.);

  eq_first_apply(s,j,c,d,1.0);
  eq_second_apply(s,j,c,d,0.5);

  return 0;
}

static int cn_eq_def_n(FDOperatorJam *j, void *d)
{
  FIRST_TIME_DERIVATIVE_FORWARD_MASK(j);

  eq_second_def(j);

  return 0;
}

static int cn_eq_apply_n(FDSolver *s, FDOperatorJam *j, un
    signed *c, void *d)
{
  FIRST_TIME_DERIVATIVE_FORWARD_SET(j,1.);

  eq_second_apply(s,j,c,d,-0.5);

  return 0;
}




static int FDHeston(double S0,double V0,NumFunc_1  *p,
    double T,double r,double divid,double sigma,double rho,double ka
    ppa,double theta,int N1,int N2,double *ptprice,double *ptde
    lta)
{
  double K;
  int k,h,offset,call_or_put;
  double Vleft,Vright;

  K=p->Par[0].Val.V_DOUBLE;
```

```
if ((p->Compute)==&Call)
  call_or_put=1;
else
  call_or_put=0;

M_Heston.boundary = call_or_put ? call_boundary : put_bo
  undary;

FDSolver s;
FDSolverVectorFiller ic_f, b_f;
FDSolverCoMatricesFiller AcBcf, AnBnf;
FDOperatorJamCoMatricesFillerData jfdc_ex,jfdn_ex;
FDOperatorJamCoMatricesFillerData jfdc_cn,jfdn_cn;

M_Heston.kappa = kappa;
M_Heston.theta = theta;
M_Heston.sigma = sigma;
M_Heston.rho = rho;
M_Heston.r = r-divid;
M_Heston.V0 = V0;
M_Heston.S0 = S0;
M_Heston.K = K;
M_Heston.T = T;

M_Heston.N1 = N1 % 2 ? N1 : N1+1;
M_Heston.N2 = N2 % 2 ? N2 : N2+1;

setup(&M_Heston);

offset = (N1-2)*(M_Heston.offy-1)+M_Heston.offx;

s.dim = 2;
s.is_A_symmetric = FALSE;

// Evaluate CFL for explicit part
s.deltaT = pow(M_Heston.xR-M_Heston.xL,2)/(0.5*M_Heston.
  yR*(pow((M_Heston.N1*M_Heston.xR),2)));

if (pow(M_Heston.yR-M_Heston.yL,2)/(0.5*M_Heston.yR*pow((
  M_Heston.N2*M_Heston.sigma),2)) < s.deltaT)
  s.deltaT = pow(M_Heston.yR-M_Heston.yL,2)/(0.5*M_
```

```
  Heston.yR*pow((M_Heston.N2*M_Heston.sigma),2));

s.deltaT *= 0.1;

s.size[0] = M_Heston.N1;
s.size[1] = M_Heston.N2;

ic_f.init = NULL;
ic_f.next_elem = ic_f_next_elem;
ic_f.finish = NULL;
ic_f.free = NULL;

b_f.init = NULL;
b_f.next_elem = b_f_next_elem;
b_f.finish = NULL;
b_f.free = NULL;

s.b_filler = &b_f;

// Explicit

s.is_fully_explicit = TRUE;
s.is_fully_implicit = FALSE;

FDOperatorJamCoMatricesFillerSet(&AcBcf,&jfdc_ex,ex_eq_de
  f_c,
                                 ex_eq_apply_c,NULL);
FDOperatorJamCoMatricesFillerSet(&AnBnf,&jfdn_ex,ex_eq_de
  f_n,
                                 ex_eq_apply_n,NULL);


if(FDSolverInit(&s, &ic_f, &AcBcf, &AnBnf)) return 1;

for(k=1;k<=20;k++) FDSolverStep(&s);

// Crank-Nicolson

s.is_fully_explicit = FALSE;
s.is_fully_implicit = FALSE;
```

```
  h = (int)ceil((1.0-s.t)/(sqrt(s.deltaT)));
  s.deltaT = (1.0-s.t)/h;

  FDOperatorJamCoMatricesFillerSet(&AcBcf,&jfdc_cn,cn_eq_de
    f_c,
                                   cn_eq_apply_c,NULL);
  FDOperatorJamCoMatricesFillerSet(&AnBnf,&jfdn_cn,cn_eq_de
    f_n,
                                   cn_eq_apply_n,NULL);

  if(FDSolverResetMatrices(&s, &AcBcf, &AnBnf)) return 1;

  for(;k<=h+20;k++) FDSolverStep(&s);

  /*Price*/
  v_s2m(&M_Heston,s.t,V_GetCmp(s.xc,offset),ptprice);
  v_s2m(&M_Heston,s.t,V_GetCmp(s.xc,offset-1),&Vleft);
  v_s2m(&M_Heston,s.t,V_GetCmp(s.xc,offset+1),&Vright);

  /*Delta*/
  *ptdelta = (M_Heston.N1-1)*(Vright-Vleft)/(2.0*(M_Heston.
    xR-M_Heston.xL));


  return OK;
}
#endif //PremiaCurrentVersion

extern "C"{
#if defined(PremiaCurrentVersion) && PremiaCurrentVersion <
     (2008+2) //The "#else" part of the code will be freely av
    ailable after the (year of creation of this file + 2)
static int CHK_OPT(FD_Heston1)(void *Opt, void *Mod)
{
  return NONACTIVE;
}
int CALC(FD_Heston1)(void *Opt, void *Mod, PricingMethod *
    Met)
{
return AVAILABLE_IN_FULL_PREMIA;
}
```

```
#else
int CALC(FD_Heston1)(void *Opt, void *Mod, PricingMethod *
    Met)
{
  TYPEOPT* ptOpt=(TYPEOPT*)Opt;
  TYPEMOD* ptMod=(TYPEMOD*)Mod;
  double r,divid;

  if(ptMod->Sigma.Val.V_PDOUBLE==0.0)
    {
      Fprintf(TOSCREEN,"BLACK-SCHOLES MODEL{n{n{n");
      return WRONG;
    }
  else
    {
      r=log(1.+ptMod->R.Val.V_DOUBLE/100.);
      divid=log(1.+ptMod->Divid.Val.V_DOUBLE/100.);

      return FDHeston(ptMod->S0.Val.V_PDOUBLE,
          ptMod->Sigma0.Val.V_PDOUBLE,
          ptOpt->PayOff.Val.V_NUMFUNC_1,
          ptOpt->Maturity.Val.V_DATE-ptMod->T.Val.V_DATE,
          r,
          divid,ptMod->Sigma.Val.V_PDOUBLE,
          ptMod->Rho.Val.V_PDOUBLE,
          ptMod->MeanReversion.hal.V_PDOUBLE,
          ptMod->LongRunVariance.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_Heston1)(void *Opt, void *Mod)
{
if ( (strcmp( ((Option*)Opt)->Name,"CallEuro")==0) || (strc
    mp( ((Option*)Opt)->Name,"PutEuro")==0) )
    return OK;
```

```
  return WRONG;
}

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

      Met->Par[0].Val.V_INT2=201;
      Met->Par[1].Val.V_INT2=51;
    }

  return OK;
}

PricingMethod MET(FD_Heston1)=
{
  "FD_NataliniBriani_Heston",
  {{"SpaceStepNumber S",INT2,{100},ALLOW},{"SpaceStepNumb
    er V",INT2,{100},ALLOW}
,{" ",PREMIA_NULLTYPE,{0},FORBID}},
  CALC(FD_Heston1),
  {{"Price",DOUBLE,{100},FORBID},
   {"Delta",DOUBLE,{100},FORBID} ,
   {" ",PREMIA_NULLTYPE,{0},FORBID}},
  CHK_OPT(FD_Heston1),
  CHK_ok,
  MET(Init)
};
}
```

References