```
  Help
extern "C"{
#include "fps2d_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;

typedef double (*paramf_t)(double, double);

typedef struct _Model
{
  // Independent model parameters
  paramf_t f;
  double nu_f, nu_s;
  double alpha,delta;
  double rho1,rho2,rho12;
  double r;
  double m_f,m_s;
  double lambda_f,lambda_s;
  double K,T;
  double x0,y0,z0; // Initial coordinates
  unsigned N1,N2,N3;      // Number of grid points per dim
    ension
  double (*boundary)(struct _Model *, double, double);

  // Dependent model parameters
  // Solution domain definition
  double xL,xR;
  double yL,yR;
  double zL,zR;
```

```
  // Offset
  int offx, offy, offz;


} FPS2DModel;

#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

static void setup(FPS2DModel *m)
{
  m->xL = 1;
  m->xR = 200;
  m->yR = -0.000001;
  m->yL = m->yR + 3.0*m->y0;
  m->zR = -0.000001;
  m->zL = m->zR + 3.0*m->z0;

  m->offx = (int)floor((m->x0-m->xL)*(m->N1-1)/(m->xR-m->x
    L));
  m->offy = (int)ceil((m->y0-m->yL)*(m->N2-1)/(m->yR-m->yL)
    );
  m->offz = (int)ceil((m->z0-m->zL)*(m->N3-1)/(m->zR-m->zL)
    );

  m->xR = (m->N1-1)*(m->x0-m->xL)/m->offx+m->xL;
  m->yL = -1.0*((m->N2-1)*(m->yR-m->y0)/(m->N2-m->offy-1)+
    m->yR);
  m->zL = -1.0*((m->N3-1)*(m->zR-m->z0)/(m->N3-m->offz-1)+
    m->zR);
}

// Model to solution space
//
/*static void c_m2s(FPS2DModel *m, double v1, double v2,
    double v3,
                     double *w1, double *w2, double *w3)
{
  if(w1) *w1 = (v1-m->xL)/(m->xR-m->xL);
```

```
  if(w2) *w2 = (v2-m->yL)/(m->yR-m->yL);
  if(w3) *w3 = (v3-m->zL)/(m->zR-m->zL);
}*/

static void v_m2s(FPS2DModel *m, double t, double V,
    double *W)
{
  *W = exp(m->r*(m->T-t))*V;
}

/*static void m2s(FPS2DModel *m, double t, double v1,
    double v2, double v3, double V,
                   double *w1, double *w2, double *w3,
    double *W)
{
  c_m2s(m,v1,v2,v3,w1,w2,w3);
  v_m2s(m,t,V,W);
}*/

// Solution to model space

static void c_s2m(FPS2DModel *m, double w1, double w2,
    double w3,
                     double *v1, double *v2, double *v3)
{
  if(v1) *v1 = (m->xR-m->xL)*w1+m->xL;
  if(v2) *v2 = (m->yR-m->yL)*w2+m->yL;
  if(v3) *v3 = (m->zR-m->zL)*w3+m->zL;
}

static void v_s2m(FPS2DModel *m, double tau, double W,
    double *V)
{
  *V = exp(-1.0*m->r*tau)*W;
}

/*static void s2m(FPS2DModel *m, double tau, double w1,
    double w2, double w3, double W,
                   double *v1, double *v2, double *v3,
    double *V)
{
```

```
  c_s2m(m,w1,w2,w3,v1,v2,v3);
  v_s2m(m,tau,W,V);
}*/

FPS2DModel M;

static double call_boundary(FPS2DModel *m, double t,
    double S)
{
  double v = S-m->K*exp(-m->r*(m->T-t));

  return v > 0 ? v : 0;
}

static double put_boundary(FPS2DModel *m, double t, double
    S)
{
  double v = m->K*exp(-m->r*(m->T-t))-S;

  return v > 0 ? v : 0;
}

static double payoff(FPS2DModel *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,(double)c[0]/(s->size[0]-1),0,0,&S,NULL,NULL);
  v_m2s(&M,M.T-s->t,payoff(&M,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,(double)c[0]/(s->size[0]-1),0,0,&S,NULL,NULL);
  v_m2s(&M,M.T-s->t,M.boundary(&M,M.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,z;
  double wx,wy,wz;

  wx = M.xR - M.xL;
  wy = M.yR - M.yL;
  wz = M.zR - M.zL;

  c_s2m(&M,(double)c[0]/(s->size[0]-1),(double)c[1]/(s->si
    ze[1]-1),
           (double)c[2]/(s->size[2]-1), &x, &y, &z);
```

```
  FIRST_SPATIAL_DERIVATIVE_CENTERED_SET(j,0,factor*x*M.r/wx
    *(M.N1-1)*s->deltaT);
  FIRST_SPATIAL_DERIVATIVE_CENTERED_SET(j,1,factor*M.alpha*
    (M.m_f-y)/wy*(M.N2-1)*s->deltaT);
  FIRST_SPATIAL_DERIVATIVE_CENTERED_SET(j,2,factor*M.delta*
    (M.m_s-z)/wz*(M.N3-1)*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,z;
  double fv,dv,bv;
  double wx,wy,wz;

  wx = M.xR - M.xL;
  wy = M.yR - M.yL;
  wz = M.zR - M.zL;

  c_s2m(&M,(double)c[0]/(s->size[0]-1),(double)c[1]/(s->si
    ze[1]-1),
           (double)c[2]/(s->size[2]-1), &x, &y, &z);

  fv = M.f(y,z)*x;
  bv = M.nu_f*M.rho1*sqrt(2*M.alpha);
```

```
  dv = M.nu_s*M.rho2*sqrt(2*M.delta);

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

  UNIFORM_SECOND_SPATIAL_DERIVATIVE_CENTERED_SET(j,1,
    factor*M.alpha*pow(M.nu_f/wy*(M.N2-1),2)*s->deltaT);

  UNIFORM_SECOND_SPATIAL_DERIVATIVE_CENTERED_SET(j,2,
    factor*M.delta*pow(M.nu_s/wz*(M.N3-1),2)*s->deltaT);


  MIXED_SECOND_SPATIAL_DERIVATIVE_CENTERED_BOUCHUT_SET(j,1,
    0,
    factor*fv*bv/(wx*wy)*(M.N1-1)*(M.N2-1)*s->deltaT);

  MIXED_SECOND_SPATIAL_DERIVATIVE_CENTERED_BOUCHUT_SET(j,2,
    0,
    factor*fv*dv/(wx*wz)*(M.N1-1)*(M.N3-1)*s->deltaT);

  MIXED_SECOND_SPATIAL_DERIVATIVE_CENTERED_BOUCHUT_SET(j,2,
    1,
    factor*2*M.nu_s*M.nu_f*sqrt(M.alpha*M.delta)*
    (M.rho1*M.rho2+M.rho12*sqrt(1-pow(M.rho1,2)))/(wz*wy)*(
    M.N2-1)*(M.N3-1)*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;
}

/////////////////////////////////////////////////////
// Implicit scheme
//

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

  return 0;
}*/

/*static int im_eq_apply_c(FDSolver *s, FDOperatorJam *j,
```

```
    unsigned *c, void *d)
{
  FIRST_TIME_DERIVATIVE_FORWARD_SET(j,1.);

  return 0;
}*/

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

  eq_first_def(j);
  eq_second_def(j);

  return 0;
}*/

/*static int im_eq_apply_n(FDSolver *s, FDOperatorJam *j,
    unsigned *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 double f_exp(double y, double z)
{
  return exp(y+z);
}

static int FD_FPS2D(double x0,double y0,double z0,double T,
    double r,double divid,NumFunc_1  *p,double delta,double alpha,
    double m_s,double m_f,double nu_s,double nu_f,double rho1,
    double rho2,double rho12,int N1,int N2,int N3,double *ptprice,
    double *ptdelta)
{
  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.boundary = call_or_put ? call_boundary : put_boundary;

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

M.r = r-divid;
M.m_f = m_f;
M.m_s = m_s;
M.nu_f = nu_f;
M.nu_s = nu_s;
M.rho1 = rho1;
M.rho2 = rho2;
M.rho12 = rho12;
M.K = K;
M.T = T;
M.f = f_exp;
M.x0 = x0;
M.y0 = y0;
M.z0 = z0;
M.alpha = alpha;
M.delta = delta;

M.N1 = N1 % 2 ? N1 : N1 + 1;
M.N2 = N2 % 2 ? N2 : N2 + 1;
M.N3 = N3 % 2 ? N3 : N3 + 1;

setup(&M);

offset = (N1-2)*(N2-2)*(M.offz-1)+(N1-2)*(M.offy-1)+M.of
  fx;
s.dim = 3;
```

```
s.is_A_symmetric = FALSE;

// Evaluate CFL for explicit method
// Assumption: f(y,z) is increasing in its arguments
s.deltaT = pow(M.xR-M.xL,2)/(0.5*pow(((M.N1-1)*M.xR*M.f(
  M.yR,M.zR)),2));

if (pow(M.yR-M.yL,2)/(M.alpha*pow(((M.N2-1)*M.nu_f),2)) <
   s.deltaT)
  s.deltaT = pow(M.yR-M.yL,2)/(M.alpha*pow(((M.N2-1)*M.
  nu_f),2));

if (pow(M.zR-M.zL,2)/(M.delta*pow(((M.N3-1)*M.nu_s),2)) <
   s.deltaT)
 s.deltaT = pow(M.zR-M.zL,2)/(M.delta*pow(((M.N3-1)*M.nu_
  s),2));

s.deltaT = 0.01*s.deltaT;

s.size[0] = M.N1;
s.size[1] = M.N2;
s.size[2] = M.N3;

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;

k = 1;

for(;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,s.t,V_GetCmp(s.xc,offset),ptprice);

/*Delta*/
v_s2m(&M,s.t,V_GetCmp(s.xc,offset-1),&Vleft);
v_s2m(&M,s.t,V_GetCmp(s.xc,offset+1),&Vright);

*ptdelta = M.N1*(Vright-Vleft)/(2.0*(M.xR-M.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_NataliniBrianiFPS2D)(void *Opt, void
    *Mod)
{
  return NONACTIVE;
}
int CALC(FD_NataliniBrianiFPS2D)(void *Opt, void *Mod,
    PricingMethod *Met)
{
return AVAILABLE_IN_FULL_PREMIA;
}
#else
int CALC(FD_NataliniBrianiFPS2D)(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 FD_FPS2D(ptMod->S0.Val.V_PDOUBLE,
      ptMod->InitialSlow.Val.V_DOUBLE,ptMod->InitialFast.
    Val.V_DOUBLE,

      ptOpt->Maturity.Val.V_DATE-ptMod->T.Val.V_DATE,
      r,
      divid,ptOpt->PayOff.Val.V_NUMFUNC_1,
      ptMod->MeanReversionSlow.Val.V_PDOUBLE, ptMod->Mea
    nReversionFast.Val.V_PDOUBLE,
      ptMod->LongRunVarianceSlow.Val.V_DOUBLE,ptMod->Lon
    gRunVarianceFast.Val.V_DOUBLE,
      ptMod->SigmaSlow.Val.V_PDOUBLE,ptMod->SigmaFast.Val
    .V_PDOUBLE,
```

```
      ptMod->Rho1.Val.V_DOUBLE,ptMod->Rho2.Val.V_DOUBLE,
    ptMod->Rho12.Val.V_DOUBLE,
      Met->Par[0].Val.V_INT,Met->Par[1].Val.V_INT,Met->
    Par[2].Val.V_INT,
      &(Met->Res[0].Val.V_DOUBLE),
      &(Met->Res[1].Val.V_DOUBLE)
      );
}

static int CHK_OPT(FD_NataliniBrianiFPS2D)(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=31;
      Met->Par[1].Val.V_INT2=31;
      Met->Par[2].Val.V_INT2=31;
    }

  return OK;
}

PricingMethod MET(FD_NataliniBrianiFPS2D)=
{
  "FD_NataliniBriani_FPS2d",
  {{"SpaceStepNumber 1 ",INT2,{100},ALLOW},{"SpaceStepNumb
    er 2 ",INT2,{100},ALLOW},{"SpaceStepNumber 3",INT2,{100},ALL
    OW}
```

```
   ,{" ",PREMIA_NULLTYPE,{0},FORBID}},
  CALC(FD_NataliniBrianiFPS2D),
  {{"Price",DOUBLE,{100},FORBID},
   {"Delta",DOUBLE,{100},FORBID} ,
   {" ",PREMIA_NULLTYPE,{0},FORBID}},
  CHK_OPT(FD_NataliniBrianiFPS2D),
  CHK_ok,
  MET(Init)
};
}
```

References