```
   Help
#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
#ifndef FD_OPERATORS_COMMON_H
#define FD_OPERATORS_COMMON_H

#include "fd_solver.h"

typedef struct _FDOperatorJam
{

  unsigned dim;
  unsigned char L0;
  unsigned char *L1, *L2;
  double L0v, *L1v, *L2v;
  unsigned state[FDSOLVERMAXDIM];

} FDOperatorJam;

typedef int (*FDOperatorJamCoMatricesFillerEqDef_t)(FDOpera
    torJam *, void *);
typedef int (*FDOperatorJamCoMatricesFillerEqApply_t)(FDSol
    ver *,
                                                          FDOpe
    ratorJam *,
                                                          unsi
    gned *coord, void *);

typedef struct _FDOperatorJamCoMatricesFillerData
{

  FDOperatorJam jam;
  FDSolverCoordWalkerData wd;
  unsigned first[FDSOLVERMAXDIM];      // TODO: This two sh
    ould be thrown away
  unsigned size[FDSOLVERMAXDIM];       // and replaced by an
     inner walker.
  unsigned offs[FDSOLVERMAXDIM];
  unsigned Ars,Brs;
```

```
  unsigned i1,c1,i2,c2;
  int first_run;

  FDOperatorJamCoMatricesFillerEqDef_t eq_def;
  FDOperatorJamCoMatricesFillerEqApply_t eq_apply;
  void *eq_data;

} FDOperatorJamCoMatricesFillerData;


#define FDOPERATORJAM_LEFT 0
#define FDOPERATORJAM_CENTER 1
#define FDOPERATORJAM_RIGHT 2

#define FDOPERATORJAM_RESET_STATE(jam)                {
  do                                                  {
  {                                                   {
    unsigned _k;                                      {
                                                      {
    for(_k=0; _k<(jam)->dim; _k++)                    {
      (jam)->state[_k] = FDOPERATORJAM_LEFT;          {
                                                      {
  } while(0);

//
// Operators mask definition
//

// Constraints:
//    - i1 > i2 => i1 > 0
//    - c1, c2 in {0,2}
//
#define FDOPERATORJAM_L2_ACCESS(jam,i1,c1,i2,c2)
            {
  (*((jam)->L2 +
            {
  ((i1)*((((i1)-1)*2)+(c1))*sizeof(unsigned char)+
            {
   (((c1)/2)+(((i1))*2))*sizeof(unsigned) +
            {
   2*(i2)*sizeof(unsigned char) + ((c2)>>1))))
```

```
#define FDOPERATORJAM_L2_COUNTER2(jam,i,c)
            {
  (*(unsigned *)((jam)->L2 +
            {
  ((i)*((((i)-1)*2)+(c))*sizeof(unsigned char) +
            {
   (((c)/2)+(((i)-1)*2)+1)*sizeof(unsigned))))

#define FDOPERATORJAM_L2_COUNTER1(jam) (*((unsigned *)((jam
    )->L2)))

#define FDOPERATORJAM_L1_ACCESS(jam,i,c) {
  (*((jam)->L1 + sizeof(unsigned) + 2*(i)*sizeof(unsigned
    char) + ((c)>>1)))

#define FDOPERATORJAM_L1_COUNTER(jam) (*((unsigned *)((jam)
    ->L1)))

#define FDOPERATORJAM_L0_ACCESS(jam) ((jam)->L0)

#define FDOPERATORJAM_L0_MARK(jam) FDOPERATORJAM_L0_ACCESS(
    jam) = 1

#define FDOPERATORJAM_L1_MARK(jam,i,c)
           {
  do
           {
  {
           {
    if(!FDOPERATORJAM_L1_ACCESS(jam,i,c))
           {
    {
           {
      FDOPERATORJAM_L1_ACCESS(jam,i,c) = 1;
           {
      FDOPERATORJAM_L1_COUNTER(jam)++;
           {
    }
           {
  }
```

```
          {
  while(0);

#define FDOPERATORJAM_L2_MARK(jam,i1,c1,i2,c2)
          {
  do
          {
  {
          {
    if(!FDOPERATORJAM_L2_ACCESS(jam,i1,c1,i2,c2))
          {
    {
          {
      FDOPERATORJAM_L2_ACCESS(jam,i1,c1,i2,c2) = 1;
          {
      FDOPERATORJAM_L2_COUNTER1(jam)++;
          {
      FDOPERATORJAM_L2_COUNTER2(jam,i1,c1)++;
          {
    }
          {
  }
          {
  while(0);

#define FDOPERATORJAM_L2_VACCESS(jam,i1,c1,i2,c2) {
  ((jam)->L2v[((i1)*((i1)-1)+(i2))*2+(i1)*(c1)+((c2)/2)])

#define FDOPERATORJAM_L1_VACCESS(jam,i,c) ((jam)->L1v[2*(i)
    +((c)/2)])

#define FDOPERATORJAM_L0_VACCESS(jam) ((jam)->L0v)

////////////////////////////////////
// Zero-order term

#define FDOPERATORJAM_Z0_SET_MASK(jam) FDOPERATORJAM_L0_MAR
    K(jam)

////////////////////////////////////
// Time derivatives
```

```
// Forward
#define FDOPERATORJAM_TDFF_SET_MASK(jam) FDOPERATORJAM_L0_
    MARK(jam)

////////////////////////////////////
// First spatial derivatives

// Upwind (forward)
#define FDOPERATORJAM_SDFU_SET_MASK(jam,i)
               {
  do
               {
  {
               {
    FDOPERATORJAM_L1_MARK(jam,i,FDOPERATORJAM_RIGHT);
               {
    FDOPERATORJAM_L0_MARK(jam);
               {
  }
               {
  while(0);

// Centered
#define FDOPERATORJAM_SDFC_SET_MASK(jam,i)
               {
  do
               {
  {
               {
    FDOPERATORJAM_L1_MARK(jam,i,FDOPERATORJAM_RIGHT);
               {
    FDOPERATORJAM_L1_MARK(jam,i,FDOPERATORJAM_LEFT);
               {
  }
               {
  while(0);

////////////////////////////////////
// Second spatial derivatives
```

```
// Uniform, centered
#define FDOPERATORJAM_SDSUC_SET_MASK(jam,i)
               {
  do
               {
  {
               {
    FDOPERATORJAM_L1_MARK(jam,i,FDOPERATORJAM_LEFT);
               {
    FDOPERATORJAM_L1_MARK(jam,i,FDOPERATORJAM_RIGHT);
               {
    FDOPERATORJAM_L0_MARK(jam);
               {
  }
               {
  while(0);

// Mixed, centered
#define FDOPERATORJAM_SDSMC_SET_MASK(jam,i1,i2)
                       {
  do
                       {
  {
                       {
    FDOPERATORJAM_L2_MARK(jam,i1,FDOPERATORJAM_LEFT,i2,FDO
    PERATORJAM_LEFT);   {
    FDOPERATORJAM_L2_MARK(jam,i1,FDOPERATORJAM_LEFT,i2,FDO
    PERATORJAM_RIGHT);  {
    FDOPERATORJAM_L2_MARK(jam,i1,FDOPERATORJAM_RIGHT,i2,FDO
    PERATORJAM_LEFT);  {
    FDOPERATORJAM_L2_MARK(jam,i1,FDOPERATORJAM_RIGHT,i2,FDO
    PERATORJAM_RIGHT); {
  }
                       {
  while(0);

// Mixed, centered, Bouchut corrected
#define FDOPERATORJAM_SDSMCB_SET_MASK(jam,i1,i2)
         {
  do
         {
```

```
  {
        {
    FDOPERATORJAM_L2_MARK(jam,i1,FDOPERATORJAM_LEFT,
        {
                                i2,FDOPERATORJAM_LEFT);
        {
    FDOPERATORJAM_L2_MARK(jam,i1,FDOPERATORJAM_RIGHT,
        {
                                i2,FDOPERATORJAM_RIGHT);
        {
    FDOPERATORJAM_L1_MARK(jam,i1,FDOPERATORJAM_LEFT);
        {
    FDOPERATORJAM_L1_MARK(jam,i2,FDOPERATORJAM_LEFT);
        {
    FDOPERATORJAM_L1_MARK(jam,i1,FDOPERATORJAM_RIGHT);
        {
    FDOPERATORJAM_L1_MARK(jam,i2,FDOPERATORJAM_RIGHT);
        {
    FDOPERATORJAM_L0_MARK(jam);
        {
  }
        {
  while(0);

//
// Operators value setting
//

/////////////////////////////////////
// Zero-order term

#define FDOPERATORJAM_Z0_SET_VALUE(jam,v) FDOPERATORJAM_L0_
    VACCESS(jam) += v

/////////////////////////////////////
// Time derivatives

// Forward
#define FDOPERATORJAM_TDFF_SET_VALUE(jam,v) FDOPERATORJAM_
    L0_VACCESS(jam) += v
```

```
//////////////////////////////////////
// First spatial derivatives

// Upwind (forward)
#define FDOPERATORJAM_SDFU_SET_VALUE(jam,i,v)
               {
  do
               {
  {
               {
    FDOPERATORJAM_L1_VACCESS(jam,i,FDOPERATORJAM_RIGHT) +=
    (v);   {
    FDOPERATORJAM_L0_VACCESS(jam) += -1.0*(v);
               {
  }
               {
  while(0);

// Centered
#define FDOPERATORJAM_SDFC_SET_VALUE(jam,i,v)
               {
  do
               {
  {
               {
    FDOPERATORJAM_L1_VACCESS(jam,i,FDOPERATORJAM_RIGHT) +=
    0.5*(v);   {
    FDOPERATORJAM_L1_VACCESS(jam,i,FDOPERATORJAM_LEFT) -= 0
    .5*(v);    {
  }
               {
  while(0);

//////////////////////////////////////
// Second spatial derivatives

// Uniform, centered
#define FDOPERATORJAM_SDSUC_SET_VALUE(jam,i,v)
               {
  do
               {
```

```
  {
              {
    FDOPERATORJAM_L1_VACCESS(jam,i,FDOPERATORJAM_LEFT) +=
    v;           {
    FDOPERATORJAM_L1_VACCESS(jam,i,FDOPERATORJAM_RIGHT) +=
    v;          {
    FDOPERATORJAM_L0_VACCESS(jam) += -2.0*(v);
              {
  }
              {
  while(0);

// Mixed, centered
#define FDOPERATORJAM_SDSMC_SET_VALUE(jam,i1,i2,v)
                {
  do
                {
  {
                {
    FDOPERATORJAM_L2_VACCESS(jam,i1,FDOPERATORJAM_LEFT,
                {
                                i2,FDOPERATORJAM_LEFT) +=
    0.25*(v);    {
    FDOPERATORJAM_L2_VACCESS(jam,i1,FDOPERATORJAM_LEFT,
                {
                                i2,FDOPERATORJAM_RIGHT) +=
     -0.25*(v);  {
    FDOPERATORJAM_L2_VACCESS(jam,i1,FDOPERATORJAM_RIGHT,
                {
                                i2,FDOPERATORJAM_LEFT) +=
    -0.25*(v);   {
    FDOPERATORJAM_L2_VACCESS(jam,i1,FDOPERATORJAM_RIGHT,
                {
                                i2,FDOPERATORJAM_RIGHT) +=
     0.25*(v);   {
  }
                {
  while(0);

// Mixed, centered, Bouchut corrected
#define FDOPERATORJAM_SDSMCB_SET_VALUE(jam,i1,i2,v)
```

```
                {
  do
                {
  {
                {
    FDOPERATORJAM_L2_VACCESS(jam,i1,FDOPERATORJAM_LEFT,
                {
                                    i2,FDOPERATORJAM_LEFT) +=
    0.5*(v);     {
    FDOPERATORJAM_L2_VACCESS(jam,i1,FDOPERATORJAM_RIGHT,
                {
                                    i2,FDOPERATORJAM_RIGHT) +=
     0.5*(v);    {
    FDOPERATORJAM_L1_VACCESS(jam,i1,FDOPERATORJAM_LEFT) -=
    0.5*(v);     {
    FDOPERATORJAM_L1_VACCESS(jam,i2,FDOPERATORJAM_LEFT) -=
    0.5*(v);     {
    FDOPERATORJAM_L1_VACCESS(jam,i1,FDOPERATORJAM_RIGHT) -=
     0.5*(v);    {
    FDOPERATORJAM_L1_VACCESS(jam,i2,FDOPERATORJAM_RIGHT) -=
     0.5*(v);    {
    FDOPERATORJAM_L0_VACCESS(jam) += v;
                {
  }
                {
  while(0);

#endif

#endif //PremiaCurrentVersion
```

References