```
   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
/**********************************************************
    ***************
 * Multidimensional Linear PDE Solver, Premia Project
 *
 * Authors:
 *         Maya Briani      <mbriani@iac.cnr.it>
 *         Roberto Natalini <rnatalini@iac.cnr.it>
 *         Cristiano Paris  <cparis@iac.cnr.it>
 *
 **********************************************************
    ***************/

#include <string.h>

#include "error.h"
#include "fd_operators.h"

int FDOperatorJamInit(FDOperatorJam *j, unsigned d)
{
  unsigned L2s = 2*(d-1)*(d*(sizeof(unsigned char)) + size
    of(unsigned)) +
                 sizeof(unsigned);

  unsigned L2vs = 2*(d-1)*d*sizeof(double);
  unsigned L1s = 2*d*sizeof(unsigned char) + sizeof(unsign
    ed);
  unsigned L1vs = 2*d*sizeof(double);

  FDDEBUG(("Size of L2 = %d{n", L2s));
  FDDEBUG(("Size of L1 = %d{n", L1s));
  FDDEBUG(("Size of L2v = %d{n", L2vs));
  FDDEBUG(("Size of L1v = %d{n", L1vs));

  j->L2 = malloc(L2s);

  if(!j->L2)
```

```
{
  FDERROR("Can’t allocate memory for buffer L2 of
  OperatorJam");
  return 1;
}

j->L1 = malloc(L1s);

if(!j->L1)
{
  FDERROR("Can’t allocate memory for buffer L1 of
  OperatorJam");
  goto free_L2;
}

j->L2v = malloc(L2vs);

if(!j->L2v)
{
  FDERROR("Can’t allocate memory for buffer L2v of
  OperatorJam");
  goto free_L1;
}

j->L1v = malloc(L1vs);

if(!j->L1v)
{
  FDERROR("Can’t allocate memory for buffer L1v of
  OperatorJam");
  goto free_L2v;
}

memset(j->L1,0,L1s);
memset(j->L2,0,L2s);
memset(j->L1v,0,L1vs);
memset(j->L2v,0,L2vs);

j->L0 = 0U;
j->dim = d;
```

```
  FDOPERATORJAM_RESET_STATE(j);

  return 0;

free_L2v:
  free(j->L2v);

free_L1:
  free(j->L2v);

free_L2:
  free(j->L2v);

  return 1;
}

void FDOperatorJamFree(FDOperatorJam *j)
{
  free(j->L1);
  free(j->L2);
  free(j->L1v);
  free(j->L2v);
}

int FDOperatorJamGetRowSizes(FDOperatorJam *j, unsigned *Ar
    s, unsigned *Brs)
{
  unsigned t = 0;
  int k,h;

  // L1
  for(k=j->dim-1; k>=0; k--)
    if(j->state[k] != FDOPERATORJAM_CENTER)
      t += FDOPERATORJAM_L1_ACCESS(j,k,j->state[k]);

  // L2
  for(k=j->dim-1; k>0; k--)
  {
    if(j->state[k] != FDOPERATORJAM_CENTER)
    {
      t += FDOPERATORJAM_L2_COUNTER2(j,k,j->state[k]);
      for(h=k-1; h>=0; h--)
        if(j->state[h] != FDOPERATORJAM_CENTER)
          t += FDOPERATORJAM_L2_ACCESS(j,k,2-j->state[k],h,
    j->state[h]);
    }
    else
    {
      for(h=k-1; h>=0; h--)
        if(j->state[h] != FDOPERATORJAM_CENTER)
          t += FDOPERATORJAM_L2_ACCESS(j,k,FDOPERATORJAM_
    LEFT,h,j->state[h]);

      for(h=k-1; h>=0; h--)
        if(j->state[h] != FDOPERATORJAM_CENTER)
          t += FDOPERATORJAM_L2_ACCESS(j,k,FDOPERATORJAM_
    RIGHT,h,j->state[h]);
    }
  }

  *Brs = t;
  *Ars = FDOPERATORJAM_L2_COUNTER1(j) + FDOPERATORJAM_L1_
    COUNTER(j) +
         FDOPERATORJAM_L0_ACCESS(j) - *Brs;

  return 0;
}

// CoMatrices filler

static int filler_init(FDSolver *s, FDSolverCoMatricesFil
    ler *cmf)
{
  unsigned k;
  FDOperatorJamCoMatricesFillerData *fdata =
                (FDOperatorJamCoMatricesFillerData *)cmf->
    data;

  fdata->offs[0] = 1;

  for(k=0; k < s->dim; k++)
  {
    fdata->first[k] = 1;
    fdata->size[k] = s->size[k] - 2;

    if(k>0) fdata->offs[k] = fdata->offs[k-1]*s->size[k-1];
  }

  FDOperatorJamInit(&fdata->jam, s->dim);
  FD_SLICE_WALKER_RESET(&fdata->wd,s->dim,fdata->first,fda
    ta->size);

  fdata->first_run = 1;

  if(fdata->eq_def(&fdata->jam,fdata->eq_data))
  {
    FDERROR("Error calling equation definition.{n");
    FDOperatorJamFree(&fdata->jam);
    return 1;
  }

  return 0;
}

static int filler_next_row(FDSolver *s, FDSolverCoMatrices
    Filler *cmf,
                     unsigned *Ars, unsigned *Brs)
{
  int *pt_notify;
  int notify = 1;
  FDOperatorJamCoMatricesFillerData *fdata =
              (FDOperatorJamCoMatricesFillerData *)cmf->
    data;

  pt_notify = &notify;
  if(!fdata->first_run)
    FD_SLICE_WALKER_UPDATE(&fdata->wd,fdata->jam.state,pt_
    notify);
  else
    fdata->first_run = 0;

  if(notify)
```

```
    FDOperatorJamGetRowSizes(&fdata->jam, &fdata->Ars, &fda
    ta->Brs);

  *Ars = fdata->Ars;
  *Brs = fdata->Brs;

  // Reset the state
  memset(fdata->jam.L1v,0,2*fdata->jam.dim*sizeof(double));
  memset(fdata->jam.L2v,0,2*(fdata->jam.dim-1)*fdata->jam.
    dim*sizeof(double));
  fdata->jam.L0v = 0.;

  fdata->i1 = fdata->jam.dim-1;
  fdata->i2 = fdata->jam.dim-2;
  fdata->c1 = FDOPERATORJAM_LEFT;
  fdata->c2 = FDOPERATORJAM_LEFT;

  FDDEBUG(("row: c1=%d, i1=%d, c2=%d, i2=%d{n",
           fdata->c1, fdata->i1,
           fdata->c2, fdata->i2));

  // TODO: Check return value and signal accordingly to ou
    r caller
  fdata->eq_apply(s,&fdata->jam,fdata->wd.coord,fdata->eq_
    data);

  return 0;
}

static unsigned posB(int dim, unsigned *offsA, unsigned *of
    fsB,
                     unsigned *x, unsigned *s, unsigned i1,
     unsigned c1,
                     unsigned i2, unsigned c2)
{
  unsigned t = 0, border = 0, xi;
  int k;

  for(k=dim-1; k>=0; k--)
  {
    xi = x[k];
    if(k == i1) xi += c1-1;
    else if(k == i2) xi += c2-1;

    if(border) t += offsA[k]*xi;
    else
    {
      if(xi>1) t += offsB[k]*(xi-1) + offsA[k];
      else if(xi>0) t += offsA[k];
    }

    if(!border && s[k] != FDOPERATORJAM_CENTER &&
        ((k == i1 && s[k] == c1) || (k == i2 && s[k] == c2)
    ))
      border = 1;
  }

  return t;
}

static int filler_next_elem(FDSolver *s, FDSolverCoMatrices
    Filler *cmf,
                            unsigned *pos, double *val,unsi
    gned *isA)
{
  int last = 0;
  FDOperatorJamCoMatricesFillerData *fdata =
                (FDOperatorJamCoMatricesFillerData *)cmf->
    data;

  while(!last)
  {
    FDDEBUG(("c1=%d, i1=%d, c2=%d, i2=%d{n",
             fdata->c1, fdata->i1,
             fdata->c2, fdata->i2));

    if(fdata->c2 == FDOPERATORJAM_CENTER)
    {
      if(fdata->c1 == FDOPERATORJAM_CENTER)
      {
        if(FDOPERATORJAM_LO_ACCESS(&fdata->jam))
        {
          *val = fdata->jam.L0v;
          last = 1;
        }
      }
      else
      {
        if(FDOPERATORJAM_L1_ACCESS(&fdata->jam,fdata->i1,
    fdata->c1))
        {
          *val = FDOPERATORJAM_L1_VACCESS(&fdata->jam,fdata
    ->i1,fdata->c1);
          last = 1;
        }
      }
    }
    else if(FDOPERATORJAM_L2_ACCESS(&fdata->jam,fdata->i1,
    fdata->c1,
                                      fdata->i2, fdata->c2)
    )
    {
      *val = FDOPERATORJAM_L2_VACCESS(&fdata->jam,fdata->i1
    , fdata->c1,
                                      fdata->i2, fdata->c2)
    ;
      last = 1;
    }

    if(last)
    {
      *isA = !((fdata->c1 == FDOPERATORJAM_CENTER ? 0 :
                      (fdata->jam.state[fdata->i1] == fdata
    ->c1)) ||
               (fdata->c2 == FDOPERATORJAM_CENTER ? 0 :
                      (fdata->jam.state[fdata->i2] == fdata
    ->c2)));

      if(*isA)
        *pos = (fdata->c1 - 1)*s->offsA[fdata->i1] +
               (fdata->c2 - 1)*s->offsA[fdata->i2];
      else
```

```
      *pos = posB(fdata->jam.dim, fdata->offs, s->offsB,
    fdata->wd.coord,
                    fdata->jam.state, fdata->i1, fdata->c1,
     fdata->i2,
                    fdata->c2);
    }

    if(fdata->i1 == 0) goto skip1;

    if(fdata->c1 == FDOPERATORJAM_CENTER) goto skip2;

    if(fdata->i2 == 0 || fdata->c2 == FDOPERATORJAM_CENTER)
    {
      if(fdata->c2 == FDOPERATORJAM_RIGHT)
      {
        if(fdata->i1 == 0)
        {
skip1:
          if(fdata->c1 == FDOPERATORJAM_RIGHT)
          {
            return 0;
          }
          else
          {
skip2:
            fdata->c1++;
          }

          fdata->i1 = fdata->jam.dim - 1;
        }
        else fdata->i1--;

        if(fdata->i1 == 0 || fdata->c1 == FDOPERATORJAM_CE
    NTER)
          fdata->c2 = FDOPERATORJAM_CENTER;
        else
          fdata->c2 = FDOPERATORJAM_LEFT;
      }
      else fdata->c2++;

      if(fdata->i1 > 0)
```

```
        fdata->i2 = fdata->i1 - 1;
      else
        fdata->i2 = 0;
    }
    else fdata->i2--;
  }

  return 0;
}

static void filler_free(FDSolver *s, FDSolverCoMatricesFil
    ler *cmf)
{
  FDOperatorJamCoMatricesFillerData *fdata =
                (FDOperatorJamCoMatricesFillerData *)cmf->
    data;

  FDOperatorJamFree(&fdata->jam);
}

void FDOperatorJamCoMatricesFillerSet(
                             FDSolverCoMatricesFiller *cmf,
                             FDOperatorJamCoMatricesFiller
    Data *data,
                             FDOperatorJamCoMatricesFiller
    EqDef_t def,
                             FDOperatorJamCoMatricesFiller
    EqApply_t apply,
                             void *eq_data
                                     )
{
  cmf->init = filler_init;
  cmf->next_row = filler_next_row;
  cmf->next_elem = filler_next_elem;
  cmf->free = filler_free;
  cmf->finish = NULL;
  cmf->data = data;
  data->eq_data = eq_data;
  data->eq_def = def;
  data->eq_apply = apply;
}
```

```
#endif //PremiaCurrentVersion
```

References