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
/*************************************************************
 *   CPS - A simple C PDE solver                             *

 *                                                           *
 *   Copyright (c) 2007,                                     *

 *     Maya Briani       <m.briani@iac.rm.cnr.it>,           *

 *     Francesco Ferreri <francesco.ferreri@gmail.com>,      *

 *     Roberto Natalini  <r.natalini@iac.rm.cnr.it>,         *
 *     Marco Papi        <m.papi@iac.rm.cnr.it>              *
 *                                                           *

 *************************************************************/
#include "cps_problem_solver.h"
#include "cps_pde_problem.h"
#include "cps_pde_term.h"
#include "cps_pde_integral_term.h"
#include "cps_pde.h"
#include "cps_grid.h"
#include "cps_grid_tuner.h"
#include "cps_grid_node.h"
#include "cps_boundary_description.h"
#include "cps_stencil.h"
#include "cps_stencil_operator.h"
#include "cps_stencil_pattern.h"
#include "cps_assertions.h"
#include "cps_utils.h"

#define VALID_MODE(m)                                    {
  (m == SOLVER_MODE_IMP || m == SOLVER_MODE_EXP)

#define VALID_CORRECTION_MODE(m)                         {
  (m == FULL_CORRECTION || m == FAST_CORRECTION)
```

```
/* private functions */

static double accuracy;
static int iterno;

static Boolean RTCAux(int Iter, double rNorm, double bNorm,
     IterIdType IterId)
{
  accuracy = rNorm/bNorm;
  iterno = Iter;
  return True;
}

static int setup_solution(problem_solver *solver)
{
  double value;
  grid_node *node;
  pde_problem *problem;

  REQUIRE("problem_not_null", solver->problem != NULL);
  REQUIRE("valid_solution_size",
          V_GetDim(&(solver->uc)) == solver->problem->solu
    tion_size);

  problem = solver->problem;

  /* iterate on nodes of first grid row */

  grid_time_initial(problem->discretization_grid);
  for(grid_space_start(problem->discretization_grid);
      !grid_space_after(problem->discretization_grid);
      grid_space_forth(problem->discretization_grid)){

    grid_item(problem->discretization_grid, &node);

    value = boundary_description_evaluate(problem->boundary
    , problem->discretization_grid, node);
    V_SetCmp(&(solver->uc),node->order,value);
    grid_node_destroy(&node);
  }
  return OK;
```

```
}

static int setup_dc_matrix(problem_solver *solver){

  pde_term                    *pterm;
  grid_node                  *cur_node;
  stencil_pattern       *st_pattern;
  stencil_application   *st_app;
  pde_problem *problem;

  QMatrix *D;
  unsigned int mode = 0;
  unsigned int row = 1;
  double dt;

  /* setup D(n) matrix */
  REQUIRE("solver_not_null", solver != NULL);
  REQUIRE("problem_is_set", solver->problem != NULL);

  problem = solver->problem;
  dt = problem->discretization_grid->delta[T_DIM];

  if(solver->mode == SOLVER_MODE_EXP)
    mode = MODE_EXP;
  if(solver->mode == SOLVER_MODE_IMP)
    mode = MODE_IMP;

  if(&(solver->Dc))
    Q_Destr(&(solver->Dc));

  Q_Constr(&(solver->Dc),"D_current", problem->solution_si
    ze, False, Rowws, Normal, True);
  Q_Lock(&(solver->Dc));
  D = &(solver->Dc);

  /* iterate over core */
  for(grid_space_start(problem->discretization_grid);
      !grid_space_after(problem->discretization_grid);
      grid_space_forth(problem->discretization_grid),row++)
    {
```

```
  grid_item(problem->discretization_grid, &cur_node);

  Q_SetLen(D,row, MAX_STENCIL_SIZE); /* TODO: adjust */

  if(cur_node->order == row){ /* we're on the diagonal */
    Q_SetEntry(D,row,XY,cur_node->order,1.0);
  }

  /* iterate over PDE terms */
  for(pde_term_start(problem->equation);
      !pde_term_after(problem->equation);
      pde_term_forth(problem->equation)){

    pde_term_item(problem->equation, &pterm);
    stencil_apply(pterm->generated_stencil, problem->dis
  cretization_grid, TIME_CUR, mode, cur_node, &st_pattern);

    for(stencil_pattern_start(st_pattern);
        !stencil_pattern_after(st_pattern);
        stencil_pattern_forth(st_pattern)){

      stencil_pattern_item(st_pattern, &st_app);
      if(stencil_application_is_internal(st_app)){
        if(Q_GetVal(D,row,st_app->position) == 0.0){
          CHECK("not_diagonal", st_app->order != row);
          Q_SetEntry(D, row, st_app->position,
                     st_app->order, dt*st_app->value);
        }
        else{
          Q_AddVal(D, row, st_app->position, dt*st_app->
  value);
        }
      }
    } /* end -- stencil_pattern loop */
    stencil_pattern_destroy(&st_pattern);
  } /* end -- pde_term loop */
  grid_node_destroy(&cur_node);
} /* end -- core loop */
CHECK_LASPACK("matrix_dc_created");
Q_Unlock(&(solver->Dc));
return OK;
```

```
}

static int setup_dn_matrix(problem_solver *solver){


  pde_term             *pterm;
  grid_node            *cur_node;
  stencil_pattern      *st_pattern;
  stencil_application  *st_app;
  pde_problem          *problem;


  QMatrix *D;
  double dt;
  unsigned int  row = 1;
  unsigned int mode = 0;

  problem = solver->problem;
  dt = problem->discretization_grid->delta[T_DIM];
  /* setup D(n+1) matrix */
  REQUIRE("solver_not_null", solver != NULL);
  REQUIRE("problem_is_set", solver->problem != NULL);

  if(solver->mode == SOLVER_MODE_EXP)
    mode = MODE_EXP;
  if(solver->mode == SOLVER_MODE_IMP)
    mode = MODE_IMP;

  if(&(solver->Dn))
    Q_Destr(&(solver->Dn));

  Q_Constr(&(solver->Dn),"D_next", problem->solution_size,
    False, Rowws, Normal, True);
  Q_Lock(&(solver->Dn));
  D = &(solver->Dn);

  /* iterate over core */

  for(grid_space_start(problem->discretization_grid);
      !grid_space_after(problem->discretization_grid);
      grid_space_forth(problem->discretization_grid),row++)
```

```
    {

    grid_item(problem->discretization_grid, &cur_node);
    Q_SetLen(D,row, MAX_STENCIL_SIZE); /* TODO: adjust */

    if(cur_node->order == row){ /* we're on the diagonal */
      Q_SetEntry(D,row,XY,cur_node->order,1.0);
    }
    /* iterate over PDE terms */
    for(pde_term_start(problem->equation);
        !pde_term_after(problem->equation);
        pde_term_forth(problem->equation)){

      pde_term_item(problem->equation, &pterm);
      stencil_apply(pterm->generated_stencil, problem->dis
    cretization_grid, TIME_NXT, mode, cur_node, &st_pattern);
      for(stencil_pattern_start(st_pattern);
          !stencil_pattern_after(st_pattern);
          stencil_pattern_forth(st_pattern)){

        stencil_pattern_item(st_pattern, &st_app);
        if(stencil_application_is_internal(st_app)){
          if(Q_GetVal(D,row,st_app->position) == 0.0){
            CHECK("not_diagonal", st_app->order != row);
            Q_SetEntry(D, row, st_app->position,
                       st_app->order, -dt * st_app->value);
          }
          else{
            Q_AddVal(D, row, st_app->position, -dt*st_app->
    value);
          }
        }
      } /* end -- stencil_pattern loop */
      stencil_pattern_destroy(&st_pattern);
    } /* end -- pde_term loop */
    grid_node_destroy(&cur_node);
  } /* end -- core loop */
  CHECK_LASPACK("matrix_dn_created");
  Q_Unlock(&(solver->Dn));
  return OK;
}
```

```
static int setup_correction(problem_solver *solver){
  /* creates boundary correction */
  REQUIRE("solver_not_null", solver != NULL);

  V_Constr(&(solver->bc), "bc", solver->problem->solution_
    size, Normal, True);
  V_SetAllCmp(&(solver->bc),0.0);
  return OK;
}

static int compute_corrections(problem_solver *solver){


  pde_term                *pterm;
  stencil_pattern         *stp;
  stencil_application     *st_app;
  grid_node               *cur_node,*neigh;

  pde_problem *problem;
  grid *grid;
  double dt;
  double value = 0.0;
  int mode = 0;

  problem = solver->problem;
  grid=problem->discretization_grid;
  dt=grid->delta[T_DIM];

  /* compute boundary correction */
  REQUIRE("solver_not_null", solver != NULL);
  V_SetAllCmp(&(solver->bc), 0.0);

  if(solver->mode == SOLVER_MODE_EXP){
    mode = MODE_EXP;
  }
  else{
    mode = MODE_IMP;
  }

  /* iterate over core nodes */
```

```
for(grid_space_start(grid);
    !grid_space_after(grid);
    grid_space_forth(grid)){

  grid_item(grid,&cur_node);

  /*
    BOUNDARY CORRECTION
  */

  if(grid_node_is_guard(cur_node)){

    for(pde_term_start(problem->equation);
        !pde_term_after(problem->equation);
        pde_term_forth(problem->equation)){

      pde_term_item(problem->equation,&pterm);

      /* explicit/implicit current part */
      stencil_apply(pterm->generated_stencil, grid,
  TIME_CUR, mode, cur_node, &stp);

      for(stencil_pattern_start(stp);
          !stencil_pattern_after(stp);
          stencil_pattern_forth(stp)){

        stencil_pattern_item(stp, &st_app);

        if(stencil_application_is_boundary(st_app)){
          grid_node_neighbour(grid, st_app->position,
  cur_node, &neigh);
          CHECK("node_is_boundary", grid_node_is_boundary
  (neigh));
          value = boundary_description_evaluate(problem->
  boundary, grid, neigh);
          grid_node_destroy(&neigh);
          V_AddCmp(&(solver->bc), cur_node->order, dt *
  value * st_app->value);
        }
      } /* end -- stencil_pattern loop */
      stencil_pattern_destroy(&stp);
```

```
    /* implicit next part */
    if(mode == MODE_IMP){
      stencil_apply(pterm->generated_stencil, grid,
TIME_NXT, MODE_IMP, cur_node, &stp);

      for(stencil_pattern_start(stp);
          !stencil_pattern_after(stp);
          stencil_pattern_forth(stp)){

        stencil_pattern_item(stp, &st_app);

        if(stencil_application_is_boundary(st_app)){
          grid_node_neighbour(grid, st_app->position,
cur_node, &neigh);
          grid_node_time_forth(neigh);
          CHECK("node_is_boundary", grid_node_is_bounda
ry(neigh));
          value = boundary_description_evaluate(problem
->boundary, grid, neigh);
          grid_node_destroy(&neigh);
          V_AddCmp(&(solver->bc), cur_node->order, dt *
 value * st_app->value);
        }
      } /* end -- stencil_pattern loop */
      stencil_pattern_destroy(&stp);
    }
  }   /* end -- pde_term loop */
} /* end -- boundary correction */

/*      source term correction      */
if(pde_has_source_term(problem->equation)){
  value = cps_function_evaluate(problem->equation->sou
rce_term,cur_node);
  grid_node_time_forth(cur_node);
  value += cps_function_evaluate(problem->equation->sou
rce_term,cur_node);
  V_AddCmp(&(solver->bc), cur_node->order, 0.5 * dt *
value);
}
```

```
    /*  integral term correction  */
    if(pde_has_integral_term(problem->equation)){
      value = pde_integral_term_evaluate(problem->equation-
    >integral_term,
                                         cur_node, &(
    solver->uc));
      V_AddCmp(&(solver->bc), cur_node->order, dt * value);

    }

    grid_node_destroy(&cur_node);
  } /* end -- core loop */
  return OK;
}

/* public interface functions */

int problem_solver_create(problem_solver **solver){

  STANDARD_CREATE(solver,problem_solver);
  return OK;
}

int problem_solver_destroy(problem_solver **solver){

  Q_Destr(&((*solver)->Dn));
  Q_Destr(&((*solver)->Dc));
  V_Destr(&((*solver)->un));
  V_Destr(&((*solver)->uc));
  V_Destr(&((*solver)->bc));

  STANDARD_DESTROY(solver);
  return OK;
}

int problem_solver_setup(problem_solver *solver, pde_proble
    m *problem){
  /* setup solver with given problem */
  REQUIRE("solver_not_null", solver != NULL);
  REQUIRE("problem_not_null", problem != NULL);
```

```
  solver->problem = problem;
  solver->step = 0;
  problem_solver_set_mode(solver, SOLVER_MODE_EXP);
  problem_solver_set_algorithm(solver, SOLVER_ALG_BICGS);

  if(pde_has_integral_term(problem->equation) ||
     pde_has_source_term(problem->equation)){
    problem_solver_set_correction_mode(problem->solver, FUL
    L_CORRECTION);
  }
  else{
    problem_solver_set_correction_mode(problem->solver, FAS
    T_CORRECTION);
  }

  setup_correction(solver);

  V_Constr(&(solver->un),"U(n+1)", problem->solution_size,
    Normal, True);
  V_SetAllCmp(&(solver->un),0.0);
  V_Constr(&(solver->uc),"U(n)", problem->solution_size,
    Normal, True);
  V_SetAllCmp(&(solver->uc),0.0);

  setup_solution(solver);

  /* setup RTC accuracy */
  SetRTCAccuracy(solver->problem->desired_accuracy);
  SetRTCAuxProc(RTCAux);

  return OK;
}

int problem_solver_reset(problem_solver *solver){
  /* reset solver, re-computing matrices */
  REQUIRE("solver_not_null", solver != NULL);

  switch(solver->mode){
  case SOLVER_MODE_EXP:
    setup_dc_matrix(solver);
    break;
```

```
  case SOLVER_MODE_IMP:
    setup_dn_matrix(solver);
    setup_dc_matrix(solver);
    break;
  }
  return OK;
}

int problem_solver_set_mode(problem_solver *solver, int
    mode){
  /* set resolution mode: implicit or explicit */
  REQUIRE("solver_not_null", solver != NULL);
  REQUIRE("valid_mode", VALID_MODE(mode));

  solver->mode = mode;
  return OK;
}

int problem_solver_set_correction_mode(problem_solver *
    solver, int mode){
  /* set resolution mode: implicit or explicit */
  REQUIRE("solver_not_null", solver != NULL);
  REQUIRE("valid_mode", VALID_CORRECTION_MODE(mode));

  solver->correction_mode = mode;
  return OK;
}

int problem_solver_set_algorithm(problem_solver *solver,
    int alg){
  /* set iterative algorithm for implicit mode */
  REQUIRE("solver_not_null", solver != NULL);

  solver->algorithm = alg;

  switch(alg){
  case SOLVER_ALG_GMRES:
    solver->iterative_solver = GMRESIter;
    break;
  case SOLVER_ALG_BICGS:
    solver->iterative_solver = BiCGSTABIter;
```

```
    break;
  case SOLVER_ALG_CG:
    solver->iterative_solver = CGIter;
    break;
  default:
    solver->algorithm = SOLVER_ALG_BICGS;
    solver->iterative_solver = BiCGSTABIter;
  }
  return OK;
}

int problem_solver_step(problem_solver *solver){
  /* performs a time step, computing next time level */
  REQUIRE("solver_not_null", solver != NULL);

  solver->step++;

  compute_corrections(solver);

  switch(solver->mode){
  case SOLVER_MODE_IMP:

    /* step */
    Q_Lock(&(solver->Dn));
    Q_Lock(&(solver->Dc));
    V_Lock(&(solver->un));
    V_Lock(&(solver->uc));

    solver->iterative_solver(&(solver->Dn), &(solver->un),
                             Add_VV(Mul_QV(&(solver->Dc), &
    (solver->uc)),&(solver->bc)),
                             MAX_MAIN_SOLVER_ITERATIONS, SS
    ORPrecond, 1.2);
    CHECK_LASPACK("cn_step_computed");

    /* GMRES backup */
    if(solver->algorithm != SOLVER_ALG_GMRES &&
       (accuracy > solver->problem->desired_accuracy || ac
    curacy < 0.0)){

      PRINT_DEBUG(DEBUG_WARN,"falling back to GMRES");
```

```
      solver->iterative_solver = GMRESIter;
      solver->iterative_solver(&(solver->Dn), &(solver->un)
    ,
                               Add_VV(Mul_QV(&(solver->Dc),
     &(solver->uc)),&(solver->bc)),
                               MAX_BACKUP_SOLVER_ITERATIONS
    , SSORPrecond, 1.2);
      CHECK_LASPACK("fallback_gmres_successful");
    }
    Q_Unlock(&(solver->Dn));
    Q_Unlock(&(solver->Dc));
    break;

  case SOLVER_MODE_EXP:
    Q_Lock(&(solver->Dc));
    V_Lock(&(solver->un));
    V_Lock(&(solver->uc));

    Asgn_VV(&(solver->un),Add_VV(Mul_QV(&(solver->Dc),&(
    solver->uc)),&(solver->bc)));
    CHECK_LASPACK("explicit_step_computed");

    Q_Unlock(&(solver->Dc));
    break;
  }

  /* swap vectors uc := un */

  Asgn_VV(&(solver->uc),&(solver->un));
  V_SetAllCmp(&(solver->un),0.0);
  V_Unlock(&(solver->un));
  V_Unlock(&(solver->uc));
  return OK;
}

int problem_solver_get_solution_element(problem_solver *
    solver, unsigned int ord, double *value){
  REQUIRE("solver_not_null", solver != NULL);
  REQUIRE("valid_order", ord > 0 && ord <= solver->problem-
    >solution_size);
```

```
  *value = V_GetCmp(&(solver->uc),ord);

  return OK;
}
/* end -- problem_solver.c */

#endif //PremiaCurrentVersion
```

References