Help

```
#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2008+2) //The "#els
#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 <stdlib.h>
#include <stdio.h>

#include "laspack/qmatrix.h"
#include "laspack/highdim_vector.h"
#include "laspack/operats.h"
#include "laspack/itersolv.h"
#include "laspack/rtc.h"
#include "laspack/errhandl.h"

#include "cps_utils.h"
#include "cps_pde_problem.h"
#include "cps_pde.h"
#include "cps_pde_term.h"
#include "cps_boundary_description.h"
#include "cps_grid.h"
#include "cps_grid_tuner.h"
#include "cps_grid_node.h"
#include "cps_problem_solver.h"
#include "cps_utils.h"
#include "cps_assertions.h"


/*
* hidden implementations: static functions which are only called
* from public functions
*/
```

```
/* compute_stencils: compute stencil for each PDE term */
static int compute_stencils(pde_problem *problem)
{

  pde_term *pterm;
  REQUIRE("problem_not_null", problem != NULL);

  for (pde_term_start(problem->equation);
       !pde_term_after(problem->equation);
       pde_term_forth(problem->equation))
    {
      pde_term_item(problem->equation, &pterm);
      pde_term_create_stencil(pterm, problem->discretization_grid);
    }
  return OK;
}

/* public interface functions */

int pde_problem_create(pde_problem **problem)
{

  STANDARD_CREATE(problem, pde_problem);
  return OK;
}

int pde_problem_destroy(pde_problem **problem)
{
  /* destroy problem structure and all related objects */
  if ((*problem)->equation)
    pde_destroy(&((*problem)->equation));
  if ((*problem)->boundary)
    boundary_description_destroy(&((*problem)->boundary));
  if ((*problem)->discretization_grid)
    grid_destroy(&((*problem)->discretization_grid));
  if ((*problem)->solver)
    problem_solver_destroy(&((*problem)->solver));
  STANDARD_DESTROY(problem);
  return OK;
}
```

```
int pde_problem_setup(pde_problem *problem)
{
  int dim;
  /* setup some internal parameters */
  REQUIRE("problem_not_null", problem != NULL);
  REQUIRE("grid_is_set", problem->discretization_grid != NULL);
  REQUIRE("grid_is_rescaled", problem->discretization_grid->is_rescaled);
  REQUIRE("equation_is_set", problem->equation != NULL);



  problem->solution_size = 1;

  for (dim = X_DIM; dim <= problem->discretization_grid->space_dimensions; dim++
    {
      problem->solution_size *= grid_iterator_span(problem->discretization_grid,
    }

  ENSURE("solution_size_set", problem->solution_size > 0);
  return OK;
}

int pde_problem_set_desired_accuracy(pde_problem *problem, double a)
{
  /* set accuracy of problem */
  REQUIRE("problem_not_null", problem != NULL);
  REQUIRE("valid_accuracy", a > 0.0);

  problem->desired_accuracy = a;

  return OK;
}

int pde_problem_set_equation(pde_problem *problem, pde *pde)
{
  /* set equation in problem */
  REQUIRE("problem_not_null", problem != NULL);
  REQUIRE("equation_not_null", pde != NULL);

  problem->equation = pde;
```

```
  return OK;
}

int pde_problem_set_grid(pde_problem *problem, grid *g)
{
  /* set grid in problem */
  REQUIRE("problem_not_null", problem != NULL);
  REQUIRE("grid_not_null", g != NULL);

  problem->discretization_grid = g;

  return OK;
}

int pde_problem_set_boundary(pde_problem *problem, boundary_description *descr)
{
  /* set boundary in problem */
  REQUIRE("problem_not_null", problem != NULL);
  REQUIRE("description_not_null", descr != NULL);

  problem->boundary = descr;

  return OK;
}

int pde_problem_set_plotting(pde_problem *problem, int b)
{
  /* toggle plotting on/of */
  REQUIRE("problem_not_null", problem != NULL);

  problem->plotting_enabled = b;

  ENSURE("plotting_set", problem->plotting_enabled == b);
  return OK;
}

int pde_problem_set_plotfile(pde_problem *problem, const char *fn)
{
  /* set plot file name */
  REQUIRE("problem_not_null", problem != NULL);
```

```
  REQUIRE("valid_filename_len", strlen(fn) < MAX_FILENAME);
  memcpy(problem->plotfile, fn, strlen(fn));
  return OK;
}

int pde_problem_solve(pde_problem *problem)
{
  /* solve problem */
  REQUIRE("problem_not_null", (problem != NULL));
  REQUIRE("grid_is_rescaled", (problem->discretization_grid->is_rescaled));
  REQUIRE("problem_is_setup", problem->solution_size > 0);

  /* 1 - compute stencils */
  compute_stencils(problem);

  /* 2 - set up solver */
  problem_solver_create(&(problem->solver));

  /* 4 - setup and iteration */
  problem_solver_setup(problem->solver, problem);

  /* explicit part */
  problem_solver_set_mode(problem->solver, SOLVER_MODE_EXP);
  grid_tuner_apply(problem->discretization_grid->tuner, EXPLICIT_TUNER, problem-
  problem_solver_reset(problem->solver);

  for (grid_time_start(problem->discretization_grid);
       !grid_time_after(problem->discretization_grid)
       && problem->solver->step < problem->max_explicit_steps;
       grid_time_forth(problem->discretization_grid))
    {
      problem_solver_step(problem->solver);
    }

  /* implicit, Crank-Nicolson part */
  problem_solver_set_mode(problem->solver, SOLVER_MODE_IMP);
  problem_solver_set_algorithm(problem->solver, SOLVER_ALG_BICGS);
  grid_tuner_apply(problem->discretization_grid->tuner, IMPLICIT_TUNER, problem-
  problem_solver_reset(problem->solver);

  for (grid_time_start(problem->discretization_grid);
```

```
       !grid_time_after(problem->discretization_grid);
       grid_time_forth(problem->discretization_grid))
    {

      problem_solver_step(problem->solver);
    }
  return OK;
}

int pde_problem_get_solution(pde_problem *problem, double *sol)
{
  /* get solution */
  grid_node *sol_node;
  REQUIRE("problem_not_null", (problem != NULL));


  grid_focus_item(problem->discretization_grid, &sol_node);
  problem_solver_get_solution_element(problem->solver, sol_node->order, sol);
  grid_node_destroy(&sol_node);
  return OK;
}

int pde_problem_get_delta_x(pde_problem *problem, double *delta)
{
  /* get solution */
  double s_left, s_right;

  grid_node *sol_node;
  REQUIRE("problem_not_null", (problem != NULL));

  grid_focus_item(problem->discretization_grid, &sol_node);

  CHECK("node_is_nearly_centered",
        sol_node->order > 1 && sol_node->order < problem->solution_size - 1);

  problem_solver_get_solution_element(problem->solver, sol_node->order - 1, &s_l
  problem_solver_get_solution_element(problem->solver, sol_node->order + 1, &s_r
  (*delta) = (s_right - s_left) / (4.0 * problem->discretization_grid->delta[X_D
  grid_node_destroy(&sol_node);
  return OK;
}
```

```
int pde_problem_plot_solution(const pde_problem *problem)
{
  /* plot to stdout, suitable to feed gnuplot */
  char filename[MAX_FILENAME];
  FILE *F;
  problem_solver *solver;
  grid *grid;
  grid_node *node;

  REQUIRE("problem_not_null", problem != NULL);
  solver = problem->solver;
  grid = problem->discretization_grid;

  sprintf(filename, "%s_%d.dat", problem->plotfile, solver->step);

  F = fopen(filename, "w+");

  for (grid_plain_start(problem->discretization_grid, X_DIM);
       !grid_plain_after(problem->discretization_grid, X_DIM);
       grid_plain_forth(problem->discretization_grid, X_DIM))
    {

      for (grid_plain_start(problem->discretization_grid, Y_DIM);
           !grid_plain_after(problem->discretization_grid, Y_DIM);
           grid_plain_forth(problem->discretization_grid, Y_DIM))
        {

          grid_plain_item(grid, &node);
          if (grid_node_is_boundary(node))
            {
              fprintf(F, "%.4f %.4f %.4f\ n",
                      node->value[X_DIM], node->value[Y_DIM],
                      boundary_description_evaluate(problem->boundary, grid, nod
            }
          else
            {
              fprintf(F, "%.4f %.4f %.4f\ n",
                      node->value[X_DIM], node->value[Y_DIM], V_GetCmp(&solver->
            }
          grid_node_destroy(&node);
```

```
        }
      fprintf(F, "\ n");
    }
  fclose(F);
  return OK;
}
/* end -- pde_problem.c */

#endif //PremiaCurrentVersion
```