```
/** \file timeDiscretization_PETSc.c
 *
 * \brief Contains the functions for performing the time stepping process
using PETSc
 *
 * \author Thibault Bridel-Bertomeu
 */

#ifdef with_petsc

#include <petsc/private/tsimpl.h> /**< Add this one to register the
custom TSAdapt */

#include "equation.h"
#include "output.h"
#include "petscinterface.h"
#include "timeDiscretization.h"

/** \brief Main time discretization loop
 *
 * Selection of temporal integration method, as well as management of
data
 * output and analysis tools.
 *
 * \warning Specific to PETSc usage
 */
void timeDisc_PETSc(void)
{

    PetscErrorCode    ierr;
    PetscReal         ftime;
    PetscInt          nsteps;
    TSConvergedReason reason;
    PetscReal         dt_init, minRadius;
    DM                plex;

    PetscFunctionBegin;

    PetscPrintf(user->model->comm, "\nStarting Computation:\n");

    /* Compute the first time step manually (then TSAdapt takes over) */
    ierr    = DMConvert(dm, DMPLEX, &plex); CHKERRV(ierr);
    ierr    = DMPlexGetGeometryFVM(plex, NULL, NULL, &minRadius);
CHKERRV(ierr);
    ierr    = MPI_Allreduce(&user->model->physics->maxspeed, &user-
>model->maxspeed,
                            1, MPIU_REAL, MPIU_MAX,
PetscObjectComm((PetscObject)ts)); CHKERRV(ierr);
    dt_init = ((PetscReal)cfl) * minRadius / user->model->maxspeed;
    ierr    = TSSetTimeStep(ts, dt_init); CHKERRV(ierr);

    /* Solve the problem */
    ierr = TSSolve(ts, sol);                    CHKERRV(ierr);

    /* Get back the solver infos, like final time, number of steps, why
it stopped */
    ierr = TSGetSolveTime(ts, &ftime);          CHKERRV(ierr);
    ierr = TSGetStepNumber(ts, &nsteps);        CHKERRV(ierr);
```

```
    ierr = TSGetConvergedReason(ts, &reason); CHKERRV(ierr);
    ierr = PetscPrintf(user->model->comm,"| %s at time %g after %D
steps\n",
           TSConvergedReasons[reason],(double)ftime,nsteps);
CHKERRV(ierr);

    /* Dump a last solution */
    ierr = vtkOutput(sol, nsteps, (void*)user); CHKERRV(ierr);

    /* Destroy the timestepping manager */
    ierr = TSDestroy(&ts);                    CHKERRV(ierr);

    PetscPrintf(user->model->comm, "| Done.\n");
}

/**
 * Below all the methods for the CFL-based timestep adaptation strategy
 */
static PetscErrorCode PetscCalcMaxSpeed(PetscInt dim, PetscReal time,
const PetscReal x[], PetscInt Nf, PetscScalar *u, void *userctx)
{
    PetscErrorCode ierr;
    User           user = (User)           userctx;
    Model          mod  = (Model)          user->model;
    Physics        phys = (Physics)        mod->physics;
    Physics_Euler  *eu  = (Physics_Euler*) phys->data;
    EulerNode      *uu  = (EulerNode*)     u;
    PetscReal c;

    PetscFunctionBegin;

    eu->sound(&gam, uu, &c);
    c = (uu->ru[0]/uu->r) + c;
    if (c > phys->maxspeed) phys->maxspeed = c;

    PetscFunctionReturn(0);
}

PetscErrorCode TSAdaptCreate_MyCFLAdapt(TSAdapt adapt)
{
    PetscErrorCode ierr;
    TSAdapt_MyCFLAdapt *data;

    PetscFunctionBegin;

    ierr = PetscNewLog(adapt,&data); CHKERRQ(ierr);
    adapt->data         = (void*)data;
    adapt->ops->choose  = TSAdaptChoose_MyCFLAdapt;
    adapt->ops->reset   = TSAdaptReset_MyCFLAdapt;
    adapt->ops->destroy = TSAdaptDestroy_MyCFLAdapt;

    PetscFunctionReturn(0);
}

PetscErrorCode TSAdaptReset_MyCFLAdapt(TSAdapt adapt)
{
  TSAdapt_MyCFLAdapt *data = (TSAdapt_MyCFLAdapt*) adapt->data;
  PetscErrorCode ierr;
```

```
  PetscFunctionBegin;
  ierr = VecDestroy(&data->Y); CHKERRQ(ierr);
  PetscFunctionReturn(0);
}

PetscErrorCode TSAdaptDestroy_MyCFLAdapt(TSAdapt adapt)
{
  PetscErrorCode ierr;

  PetscFunctionBegin;
  ierr = TSAdaptReset_MyCFLAdapt(adapt); CHKERRQ(ierr);
  ierr = PetscFree(adapt->data);         CHKERRQ(ierr);
  PetscFunctionReturn(0);
}

PetscErrorCode TSAdaptChoose_MyCFLAdapt(TSAdapt adapt, TS ts, PetscReal
h,
                                        PetscInt *next_sc, PetscReal
*next_h,
                                        PetscBool *accept, PetscReal
*wlte,
                                        PetscReal *wltea, PetscReal
*wlter)
{
    PetscErrorCode     ierr;
    User               user = NULL;
    PetscReal          new_dt;
    DM                 plex, ts_dm;
    PetscErrorCode     (*func[1]) (PetscInt dim, PetscReal time, const
PetscReal x[], PetscInt Nf, PetscScalar *u, void *ctx);
    void               *ctx[1];
    PetscReal          waqt;
    TSAdapt_MyCFLAdapt *data = (TSAdapt_MyCFLAdapt*) adapt->data;
    PetscReal          minRadius;

    PetscFunctionBegin;

    *next_sc =  0;  /* Reuse the same order scheme */
    *wltea   = -1;  /* Weighted absolute local truncation error is not
used */
    *wlter   = -1;  /* Weighted relative local truncation error is not
used */
    *wlte    = -1;  /* Weighted local truncation error is not going to be
evaluated anyways */

    /* Get the context and the DM through the TS */
    ierr    = TSGetApplicationContext(ts, &user); CHKERRQ(ierr);
    ierr    = TSGetTime(ts, &waqt);               CHKERRQ(ierr);
    ierr    = TSGetDM(ts, &ts_dm);                CHKERRQ(ierr);
    ctx[0]  = (void*) user;
    func[0] = PetscCalcMaxSpeed;

    /* Set the current solution vector as data to the adapter */
    if (!data->Y) {
        ierr = VecDuplicate(ts->vec_sol, &data->Y); CHKERRQ(ierr);
    }
    /* - Then compute the sound speed in the local domain and
```

```
     * - the minimum inscribed circle radius over the global domain
     */
    ierr = DMProjectFunction(ts_dm, waqt, func, ctx, INSERT_ALL_VALUES,
data->Y); CHKERRQ(ierr);
    ierr = DMConvert(ts_dm, DMPLEX, &plex); CHKERRQ(ierr);
    ierr = DMPlexGetGeometryFVM(plex, NULL, NULL, &minRadius);
CHKERRQ(ierr);
    /* - In case of several mpi processes, reduce the maxspeed to get the
max over the domain */
    ierr = MPI_Allreduce(&user->model->physics->maxspeed, &user->model-
>maxspeed, 1, MPIU_REAL, MPIU_MAX, PetscObjectComm((PetscObject)ts));
CHKERRQ(ierr);
    /* - Evaluate the new DT (inviscid-like) */
    new_dt = ((PetscReal)cfl) * minRadius / user->model->maxspeed;

    *accept = PETSC_TRUE;
    *next_h = new_dt;

    PetscFunctionReturn(0);
}

/**
 * Below all the methods for implicit time stepping
 */
PetscErrorCode PetscIJacobian(
                                TS ts,         /*!< Time stepping object
(see PETSc TS)*/
                                PetscReal t,   /*!< Current time */
                                Vec Y,         /*!< Solution vector */
                                Vec Ydot,      /*!< Time-derivative of
solution vector */
                                PetscReal a,   /*!< Shift */
                                Mat A,         /*!< Jacobian matrix */
                                Mat B,         /*!< Preconditioning matrix
*/
                                void *ctxt     /*!< Application context */
                              )
{
    PetscErrorCode ierr;
    User           user = (User) ctxt;

    PetscFunctionBegin;

    user->isImplicit = isImplicit;
    user->shift = a;
    user->waqt = t;
    user->ts   = &ts;
    /* Construct preconditioning matrix */
    if (user->flag_use_precon) { ierr = PetscComputePreconMatImpl(B, Y,
user); CHKERRQ(ierr); }

    PetscFunctionReturn(0);
}

PetscErrorCode PetscComputePreconMatImpl(
                                          Mat Pmat,    /*!< Preconditioning
matrix to construct [local] */
```

```
                                          Vec locY,   /*!< Solution vector
[local] */
                                          void *ctxt  /*!< Application
context */
                                         )
{
    PetscErrorCode ierr;
    User           user  = (User) ctxt;
    TS             locts = NULL;
    DM             locdm = NULL, plex = NULL;
    PetscInt       dim, depth;
    IS             cellIS;
    PetscInt       cStart, cEnd, numCells;
    const PetscInt *cells;
    PetscInt       c;

    PetscFunctionBegin;

    /* Get the PETSc objects */
    locts = (TS) *(user->ts);
    ierr = TSGetDM(locts, &locdm); CHKERRQ(ierr);
    ierr = DMGetDimension(locdm, &dim); CHKERRQ(ierr);
    ierr = DMConvert(locdm, DMPLEX, &plex); CHKERRQ(ierr);

    /* Get the "cell" layer in the DM */
    ierr = DMPlexGetDepth(plex, &depth); CHKERRQ(ierr);
    ierr = DMGetStratumIS(plex, "dim", depth, &cellIS); CHKERRQ(ierr);
    if (!cellIS) {
        ierr = DMGetStratumIS(plex, "depth", depth, &cellIS);
CHKERRQ(ierr);
    }
    ierr = ISGetPointRange(cellIS, &cStart, &cEnd, &cells);
CHKERRQ(ierr);
    numCells = cEnd - cStart;

    /* Loop on cells */
    for (c = cStart; c < cEnd; ++c) {
    }

    /* Free local memory */
    ierr = DMDestroy(&plex); CHKERRQ(ierr);

    PetscFunctionReturn(0);
}

PetscErrorCode PetscJacobianFunction_JFNK(
                                          Mat Jacobian, /*!< Jacobian
matrix [local] */
                                          Vec locY,     /*!< Input
vector [local] */
                                          Vec locF      /*!< Output
vector (Jacobian times input vector) [local] */
                                         )
{
    PetscErrorCode ierr;
    User           user = NULL;
    PetscReal      normY, epsilon;
    DM             locdm = NULL, plex = NULL;
```

```
    TS              locts = NULL;
    Vec             RHS_hyp, F;

    PetscFunctionBegin;

    /* Get back the context to access everything */
    ierr = MatShellGetContext(Jacobian, &user); CHKERRQ(ierr);
    /* Get the norm of the increment vector Y */
    ierr = VecNorm(locY, NORM_2, &normY); CHKERRQ(ierr);
    /* - If the increment is infinitesimal */
    if (normY < 1e-16) {
        ierr = VecZeroEntries(locF); CHKERRQ(ierr);
        ierr = VecAXPBY(locF, user->shift, 0.0, locY); CHKERRQ(ierr);
    }
    /* - If the increment is non negligible */
    else {
        /* -- Get the PETSc objects */
        locts = (TS) *(user->ts);
        ierr = TSGetDM(locts, &locdm); CHKERRQ(ierr);
        ierr = DMConvert(locdm, DMPLEX, &plex); CHKERRQ(ierr);
        /* -- U0 + eps * Y */
        epsilon = user->jfnk_eps / normY;
        ierr = VecAYPX(locY, epsilon, locts->vec_sol); CHKERRQ(ierr);
        /* -- F(U0 + eps * Y) */
        ierr = DMGetGlobalVector(plex, &RHS_hyp); CHKERRQ(ierr);
        ierr = VecZeroEntries(RHS_hyp); CHKERRQ(ierr);
        ierr = DMPlexTSComputeRHSFunctionFVM(plex, user->waqt, locY,
RHS_hyp, (void*) user); CHKERRQ(ierr);
        /* -- J = F(U0 + eps * Y) - F(U0) */
        ierr = DMGetGlobalVector(plex, &F); CHKERRQ(ierr);
        ierr = VecZeroEntries(F); CHKERRQ(ierr);
        ierr = VecWAXPY(F, -1.0, user->RHS_ref, RHS_hyp); CHKERRQ(ierr);
        /* -- J = shift * Y - 1/eps * (F(U0+eps*Y)-F(U0)) */
        ierr = VecZeroEntries(locF); CHKERRQ(ierr);
        ierr = DMGlobalToLocalBegin(plex, F, INSERT_VALUES, locF);
CHKERRQ(ierr);
        ierr = DMGlobalToLocalEnd(plex, F, INSERT_VALUES, locF);
CHKERRQ(ierr);
        ierr = VecAXPBY(locF, user->shift, (-1.0/epsilon), locY);
CHKERRQ(ierr);
        /* -- Restore accesses */
        ierr = DMRestoreGlobalVector(plex, &RHS_hyp); CHKERRQ(ierr);
        ierr = DMRestoreGlobalVector(plex, &F); CHKERRQ(ierr);
        ierr = DMDestroy(&plex); CHKERRQ(ierr);
    }

    PetscFunctionReturn(0);
}

PetscErrorCode PetscJacobianFunction_Linear(Mat Jacobian, Vec Y, Vec F)
{
    PetscErrorCode ierr;

    PetscFunctionBegin;

    PetscFunctionReturn(0);
}
```

```
#endif
```