```
#-------------------------------------------------------------------#
### Petsc options

## Variables to monitor during the run
-monitor density,pressure

## Implicit time stepping options

# Nonlinear solver (SNES) type
-snes_type newtonls
# Set relative tolerance
-snes_rtol 1e-4
# Set absolute tolerance
-snes_atol 1e-4
# Set step size tolerance
-snes_stol 1e-16

# Linear solver (KSP) type
-ksp_type gmres
# Set relative tolerance
-ksp_rtol 1e-5
# Set absolute tolerance
-ksp_atol 1e-4
# Set maximum number of iterations
-ksp_max_it 5

# Preconditioner (PC)
-pc_type jacobi
-pc_factor_reuse_ordering

# apply right preconditioner
-ksp_pc_side RIGHT
```

```
        /* ---- RHS here */
        ierr = DMTSSetRHSFunctionLocal(dm, PetscRHSFunctionExpl, user);
CHKERRV(ierr);
        /* ---- Matrix-free representation of the jacobian for LHS now */
        // SNES           snes;
        // DM             sdm;
        // KSP            ksp;
        // PC             pc;
        // TSProblemType ptype;
        // SNESType       snestype;
        // Mat            A, B;
        // Vec            tmp;
        // PetscInt       local_size, global_size;
        // /**/
        // ierr = TSGetSNES(ts, &snes); CHKERRV(ierr);
        // ierr = SNESGetType(snes, &snestype); CHKERRV(ierr);
        // ierr = TSGetProblemType(ts, &ptype); CHKERRV(ierr);

        // ierr = SNESGetDM(snes, &sdm); CHKERRV(ierr);
        // ierr = DMGetGlobalVector(sdm, &tmp); CHKERRV(ierr);
        // ierr = VecGetLocalSize(tmp, &local_size); CHKERRV(ierr);
        // ierr = VecGetSize(tmp, &global_size); CHKERRV(ierr);
        // ierr = DMRestoreGlobalVector(sdm, &tmp); CHKERRV(ierr);

        // ierr = MatCreateShell(user->model->comm, local_size,
local_size, global_size, global_size, user, &A); CHKERRV(ierr);
        // if ((!strcmp(snestype, SNESKSPONLY)) || (ptype == TS_LINEAR))
{
        //     /* linear problem */
        //     user->flag_is_linear = 1;
        //     ierr = MatShellSetOperation(A, MATOP_MULT, (void
(*)(void))PetscJacobianFunction_Linear); CHKERRV(ierr);
        //     ierr = SNESSetType(snes, SNESKSPONLY); CHKERRV(ierr);
        // } else {
        //     /* nonlinear problem */
        //     user->flag_is_linear = 0;
        //     user->jfnk_eps = 1e-7;
        //     ierr = PetscOptionsGetReal(PETSC_NULL, PETSC_NULL, "-
jfnk_epsilon", &user->jfnk_eps, PETSC_NULL);          CHKERRV(ierr);
        //     ierr = MatShellSetOperation(A, MATOP_MULT, (void
(*)(void))PetscJacobianFunction_JFNK); CHKERRV(ierr);
        // }
        // ierr = MatSetUp(A); CHKERRV(ierr);
        // /* ----- Preconditioner */
        // user->flag_use_precon = 0;
        // ierr = PetscOptionsGetBool(PETSC_NULL, PETSC_NULL, "-with_pc",
(PetscBool*)&user->flag_use_precon, PETSC_NULL); CHKERRV(ierr);
        // if (user->flag_use_precon) {
        //     /* ------ Set up preconditioner matrix */
        //     PetscInt dim;
        //     ierr = DMGetDimension(dm, &dim); CHKERRV(ierr);
        //     ierr = MatCreateAIJ(user->model->comm, local_size,
local_size, global_size, global_size,
        //                         (dim*2+1)*user->model->physics->dof,
NULL,
        //                         2*dim*user->model->physics->dof, NULL,
&B); CHKERRV(ierr);
```

```
        //      ierr = MatSetBlockSize(B, user->model->physics->dof);
CHKERRV(ierr);
        //      /* Set the RHSJacobian function for TS */
        //      ierr = TSSetIJacobian(ts, A, B, PetscIJacobian, user);
CHKERRV(ierr);
        // } else {
        //      /* ------ Just roll without any preconditioner */
        //      /* Set the RHSJacobian function for TS */
        //      ierr = TSSetIJacobian(ts, A, A, PetscIJacobian, user);
CHKERRV(ierr);
        //      /* Set PC (preconditioner) to none */
        //      ierr = SNESGetKSP(snes, &ksp); CHKERRV(ierr);
        //      ierr = KSPGetPC(ksp, &pc);     CHKERRV(ierr);
        //      ierr = PCSetType(pc, PCNONE);  CHKERRV(ierr);
        // }
        /* ---- Create coloring context */
        Mat J = NULL; // Jacobian
        ierr = DMSetMatType(dm, MATAIJ); CHKERRV(ierr);
        ierr = DMCreateMatrix(dm, &J); CHKERRV(ierr);
        ISColoring iscoloring;
        MatColoring matcoloring;
        MatFDColoring matfdcoloring;
        PetscBool hascolor;
        ierr = DMHasColoring(dm, &hascolor); CHKERRV(ierr);
        if (hascolor) {
            ierr = DMCreateColoring(dm, IS_COLORING_GLOBAL, &iscoloring);
CHKERRV(ierr);
        } else {
            ierr = MatColoringCreate(J, &matcoloring); CHKERRV(ierr);
            ierr = MatColoringSetType(matcoloring, MATCOLORINGSL);
CHKERRV(ierr);
            ierr = MatColoringSetFromOptions(matcoloring); CHKERRV(ierr);
            ierr = MatColoringApply(matcoloring, &iscoloring);
CHKERRV(ierr);
            ierr = MatColoringDestroy(&matcoloring); CHKERRV(ierr);
        }
        ierr = MatFDColoringCreate(J, iscoloring, &matfdcoloring);
CHKERRV(ierr);
        ierr = MatFDColoringSetFunction(matfdcoloring,(PetscErrorCode
(*)(void))SNESTSFormFunction,ts); CHKERRV(ierr);
        ierr = MatFDColoringSetFromOptions(matfdcoloring); CHKERRV(ierr);
        ierr = MatFDColoringSetUp(J,iscoloring,matfdcoloring);
CHKERRV(ierr);
        SNES snes;
        ierr = TSGetSNES(ts, &snes); CHKERRV(ierr);
        ierr = SNESSetFromOptions(snes); CHKERRV(ierr);
        ierr =
SNESSetJacobian(snes,J,J,SNESComputeJacobianDefaultColor,matfdcoloring);
CHKERRV(ierr);
        ierr = ISColoringDestroy(&iscoloring); CHKERRV(ierr);
        /* ---- Set the preconditioner */
        KSP ksp;
        ierr = SNESGetKSP(snes,&ksp); CHKERRV(ierr);
        ierr = KSPSetFromOptions(ksp); CHKERRV(ierr);
        PC pc;
        ierr = KSPGetPC(ksp,&pc); CHKERRV(ierr);
        ierr = PCSetFromOptions(pc); CHKERRV(ierr);
    }
```