ThermalPorous: simulating non-isothermal flow in porous media using Firedrake and PETSc

Thomas Roy

June 27, 2019

Project	Preconditioning for Non-Isothermal Flow in Porous Media
Supervisors	Prof. Andy Wathen, Dr. Tom Jönsthövel, Dr. Christopher Lemon

Schlumberger

Overview of this session

- Firedrake and PETSc
- Discretizating with Firedrake
- Solving with PETSc
- Features of ThermalPorous

PETSc

PETSc is a suite of data structures and routines for the scalable (parallel) solution of scientific applications modelled by partial differential equations.
Features include:

- Parallel vectors and matrices (sparse storage formats. easy, efficient assembly)
- **Scalable parallel preconditioners, Listing of preconditioners methods**
- Krylov subspace methods, Listing of Krylov methods
- Parallel Newton-based nonlinear solvers, as well as a variety other nonlinear solvers
- Support for Nvidia GPUs
- Complete documentation
- Automatic profiling of floating point and memory usage
- Consistent user interface
- Intensive error checking
- PETSc is supported and will be actively enhanced for many years

Firedrake

Firedrake is an automated system for the solution of partial differential equations using the finite element method (FEM).
Features include:

- Expressive specification of any PDE using the Unified Form Language (UFL) from the FEniCS Project.
- Sophisticated, programmable solvers through seamless coupling with PETSc.
- Triangular, quadrilateral, and tetrahedral unstructured meshes.
- Layered meshes of triangular wedges or hexahedra.
- Vast range of finite element spaces.
- Customisable operator preconditioners.

Weak formulation of Poisson

We want to solve the Poisson equation on the unit square.

$$-\nabla^2 u = f \quad \text{in } \Omega = [0,1] \times [0,1],$$

$$u = 0 \quad \text{on } \Gamma_D \qquad \nabla u \cdot \mathbf{n} = g \quad \text{on } \Gamma_N.$$

where Γ_D are the left and right edges, and Γ_N are the top and bottom. We can rewrite this in variational form:

$$\begin{aligned} -\int_\Omega \nabla^2 u \; v \, \mathrm{d}x &= \int_\Omega f \; v \, \mathrm{d}x \quad \text{for all } v \\ \int_\Omega \nabla u \cdot \nabla v \, \mathrm{d}x - \int_{\partial\Omega} v \, \nabla u \cdot \mathbf{n} \, \mathrm{d}s &= \int_\Omega f \; v \, \mathrm{d}x \quad \text{for all } v \\ \int_\Omega \nabla u \cdot \nabla v \, \mathrm{d}x - \int_{\Gamma_N} g \; v \, \mathrm{d}s &= \int_\Omega f \; v \, \mathrm{d}x \quad \text{for all } v \end{aligned}$$

since u satisfies the boundary conditions above, and choosing $v = 0$ on Γ_D.

Finite Element for Poisson

We want to find an approximation of the weak formulation, for e.g. a piecewise linear approximation. Find $u_h \in V_h = \{u \in \mathbb{P}_1 \mid u = 0 \text{ on } \Gamma_D\}$ such that

$$\int_\Omega \nabla u_h \cdot \nabla v_h \, \mathrm{d}x - \int_{\Gamma_N} g \; v_h \, \mathrm{d}s = \int_\Omega f \; v_h \, \mathrm{d}x \quad \text{for all } v_h \in V_h$$

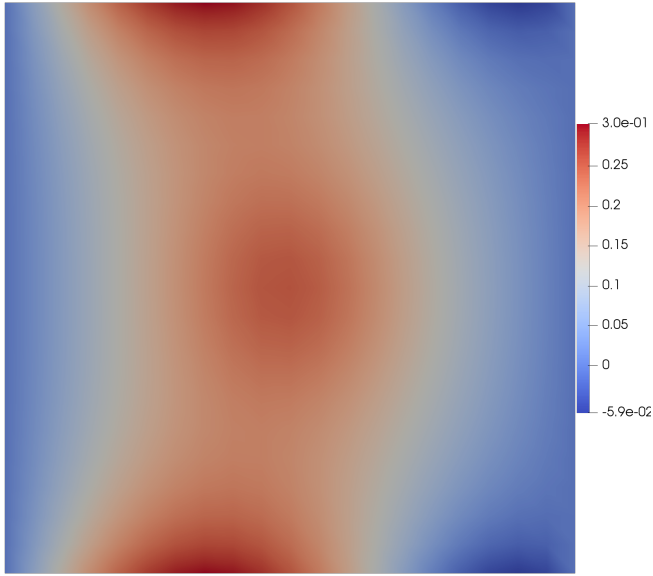
Let's solve this problem using Firedrake for a particular choice f and g.

```
mesh = UnitSquareMesh(20, 20)
V = FunctionSpace(mesh, "CG", 1)
bc = DirichletBC(V, Constant(0), [1, 2])
x,y = mesh.coordinates
f = interpolate(10*exp(-((x - 0.5)**2 \
    + (y - 0.5)**2) / 0.02), V)
g = interpolate(sin(5*x), V)
```

Firedrake code for Poisson


```
from firedrake import *
mesh = UnitSquareMesh(20, 20)
V = FunctionSpace(mesh, "CG", 1)
bc = DirichletBC(V, Constant(0), [1, 2])
x,y = mesh.coordinates
f = interpolate(10*exp(-((x - 0.5)**2 \
    + (y - 0.5)**2) / 0.02), V)
g = interpolate(sin(5*x), V)

u = TrialFunction(V)
v = TestFunction(V)
a = inner(grad(u), grad(v))*dx
L = f*v*dx + g*v*ds

u = Function(V)
solve(a == L, u, bc)
```

Solution



Finite Volume as DG0 (heat equation)

We want to write the Finite Volume method with Two-point flux approximation as a Finite Element method. We will use piecewise constant Discontinuous Galerkin (DG0). For example, the heat equation:

$$\frac{\partial u}{\partial t} - \nabla^2 u = 0 \quad \text{in } \Omega,$$

Weak form on a single cell E_i is: find u such that

$$\int_{E_i} \frac{\partial u}{\partial t} v \,\mathrm{d}x + \int_{E_i} \nabla u \cdot \nabla v \,\mathrm{d}x - \int_{\partial E_i} v \nabla u \cdot \mathbf{n} \,\mathrm{d}s = 0 \quad \text{for all test functions } v.$$

Finite Volume as DG0 (heat equation)

Take u, v to be piecewise constant (u_i, v_i in cell E_i). Then $\nabla u = 0$ inside cell E_i. Let $\mathcal{N}(i)$ be indices of neighboring cells, e_{ij} facets, h_i, h_j cell centers. We use Two-Point flux approximation:

$$\int_{E_i} \frac{\partial u_i}{\partial t} v_i \,\mathrm{d}x - \sum_{j \in \mathcal{N}(i)} \int_{e_{ij}} v_i \frac{u_j - u_i}{\|h_j - h_i\|} \,\mathrm{d}s = 0 \quad \text{for all } v_i.$$

This is equivalent to the Finite Volume discretization:

$$\frac{\partial u_i}{\partial t} |E_i| - \sum_{j \in \mathcal{N}(i)} \frac{u_j - u_i}{\|h_j - h_i\|} |e_{ij}| = 0 \quad \text{for every } i.$$

Here, $|E_i|$ is the volume of cell E_i and $|e_{ij}|$ is the area of the facet between cells E_i and E_j.

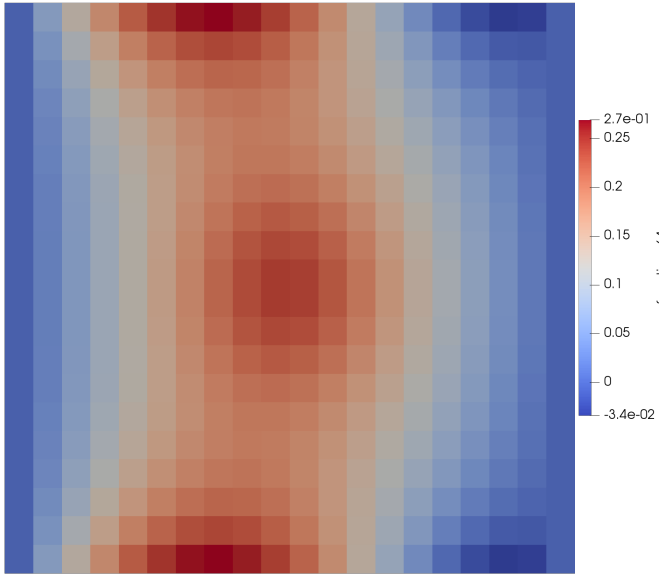
DG0 Poisson equation in Firedrake

Given ordering of indices in $\mathcal{I}$, denote by u^+ and u^- the limit value of u for two cells sharing an edge. Denote *jump* of v as $[v] = v^+ - v^-$. Summing over all $i \in \mathcal{I}$: find $u \in \mathbb{P}_0$ s.t.

$$\int_{\Gamma_{\text{int}}} [v] \frac{[u]}{\|h^+ - h^-\|} \, \mathrm{d}S = f \quad \text{for all } v \in \mathbb{P}_0.$$

```
mesh = UnitSquareMesh(20, 20, quadrilateral = True)
V = FunctionSpace(mesh, "DG", 0)
bc = DirichletBC(V, Constant(0), [1, 2], method = "geometric")

x_func = interpolate(x, V)
y_func = interpolate(y, V)
Delta_h = sqrt(jump(x_func)**2 + jump(y_func)**2)

a = jump(u)/Delta_h*jump(v)*dS
```

Solution



Heterogeneous coefficient

Consider a Heterogeneous diffusion equation

$$-\nabla \cdot \kappa \nabla u = f \quad \text{in } \Omega,$$

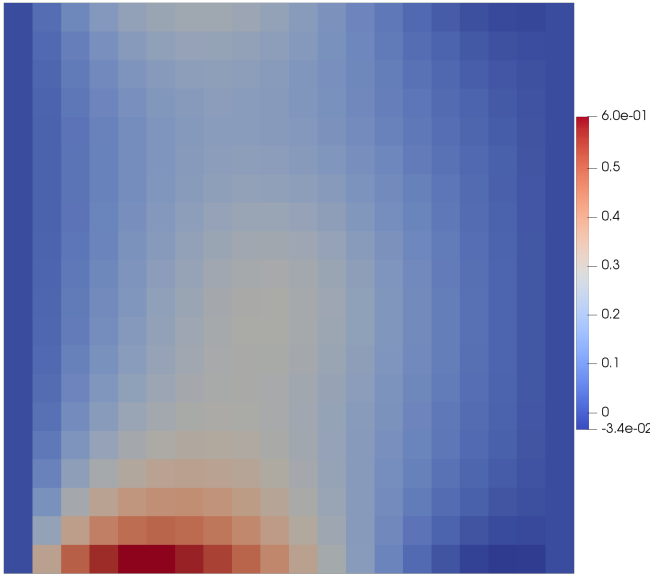
where κ varies in space. We take the harmonic average (as is done for the permeability in reservoir simulation).
Find $u \in \mathbb{P}_0$ s.t.

$$\int_{\Gamma_{\text{int}}} \{\!\{\kappa\}\!\}[v] \frac{[u]}{\|h^+ - h^-\|} \, \mathrm{d}S = 0 \quad \text{for all } v \in \mathbb{P}_0,$$

where $\{\!\{\cdot\}\!\}$ is the harmonic average across facets.

```
kappa = interpolate(x + y, V)
kappa_facet = conditional(gt(avg(kappa), 0.0), \
              kappa('+')*kappa('-')/avg(kappa), 0.0)
a = kappa_facet*jump(u)/Delta_h*jump(v)*dS
```

Solution



Isothermal flow in porous media

Consider Darcy flow in 2D. We want to find the pressure p such that

$$\phi \frac{\partial \rho}{\partial t} - \nabla \cdot \left(\rho \frac{\mathbf{K}}{\mu} \nabla p \right) = f \quad \text{in } \Omega.$$

Here ρ is density, μ is viscosity, $\mathbf{K}$ is permeability. The DG0 formulation is: find $p \in \mathbb{P}_0$ such that

$$\int_\Omega \phi \frac{\partial \rho}{\partial t} q \, \mathrm{d}x + \int_{\Gamma_{\text{int}}} [q] \{\!\{ \mathbf{K} \}\!\} \frac{\rho^{\text{up}}}{\mu^{\text{up}}} \frac{[p]}{\|h^+ - h^-\|} \, \mathrm{d}S - \int_\Omega f q \, \mathrm{d}x = 0, \quad \text{for all } q \in \mathbb{P}_0.$$

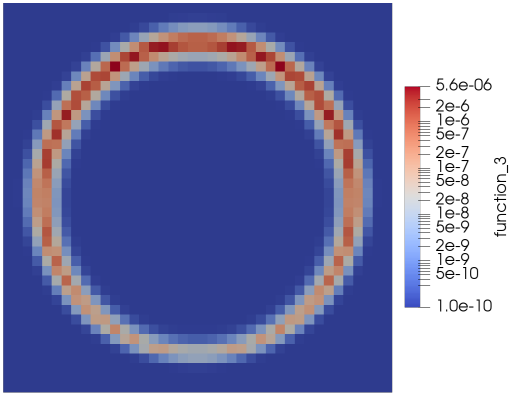
The upwind quantities are determined by:

$$u^{\text{up}} = \begin{cases} u\,|^e_{E_1} & \text{if } [p] \geq 0, \\ u\,|^e_{E_2} & \text{if } [p] < 0. \end{cases}$$

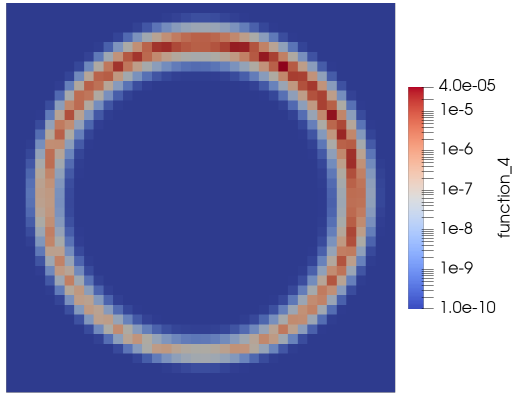
Isothermal flow in porous media


```
n = FacetNormal(mesh)

K_x_facet = conditional(gt(avg(K_x), 0.0), K_x('+')*K_x('-') / avg(K_x), 0.0)
K_y_facet = conditional(gt(avg(K_y), 0.0), K_y('+')*K_y('-') / avg(K_y), 0.0)
K_facet = (K_x_facet*(abs(n[0]('+'))+abs(n[0]('-')))/2 \
          +  K_y_facet*(abs(n[1]('+'))+abs(n[1]('-')))/2)

p = Function(V)
p_ = Function(V)
q = TestFunction(V)

a_accum = phi*(rho(p) - rho(p_))/dt*q*dx
a_flow = K_facet/mu*conditional(gt(jump(p), 0.0), rho(p('+')), rho(p('-'))) \
         *jump(q)*jump(p)/Delta_h*dS
F = a_accum + a_flow

prod_well = well_delta((5.0,L/2.0))
inj_well = well_delta((L-5.0,L/2.0))

F -= prod_rate*prod_well*rho(p)*q*dx

F -= inj_rate*inj_well*rho(p)*q*dx
```

Permeability field (Log scale)

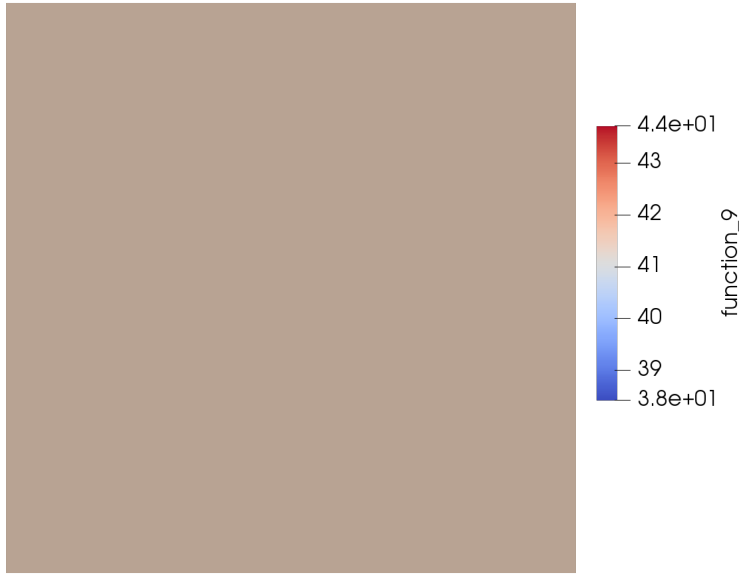


K_x

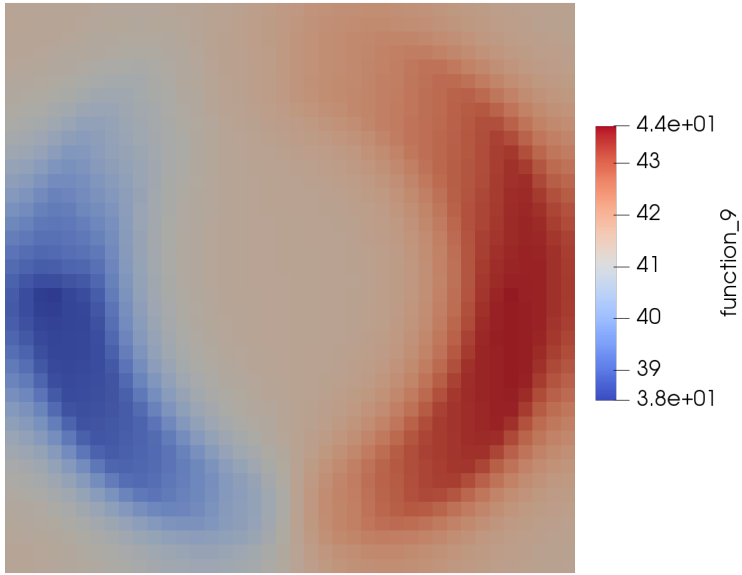


K_y

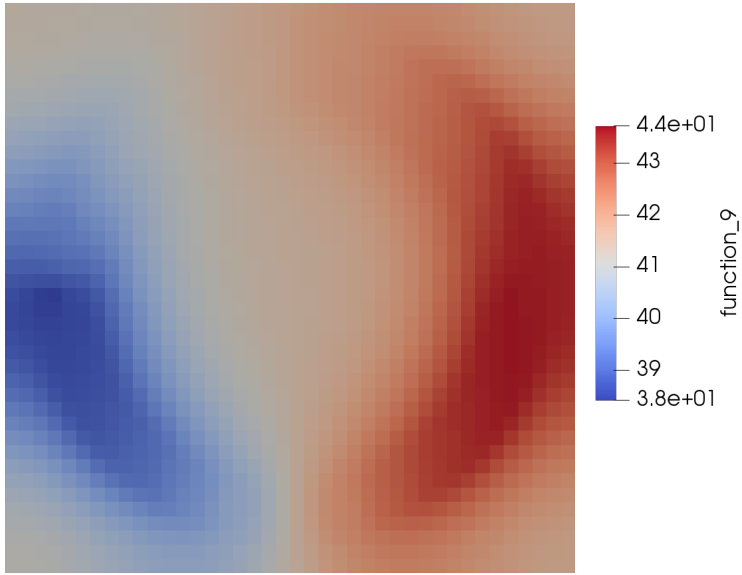
Solution



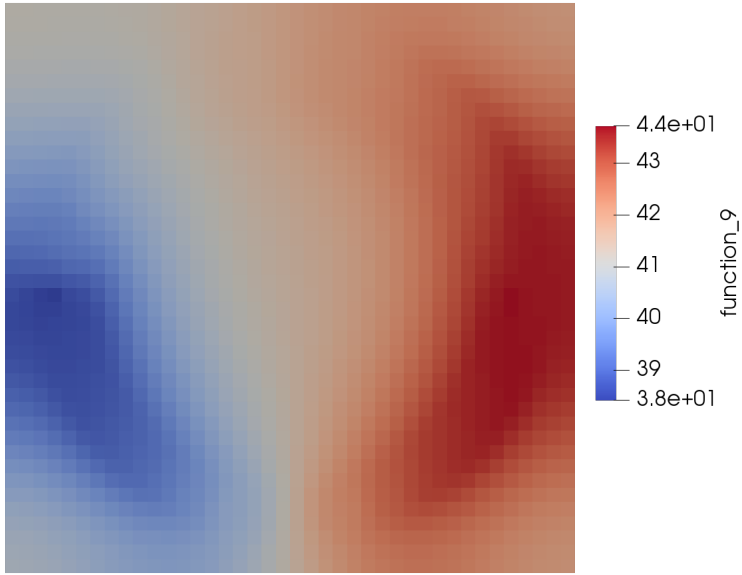
Solution



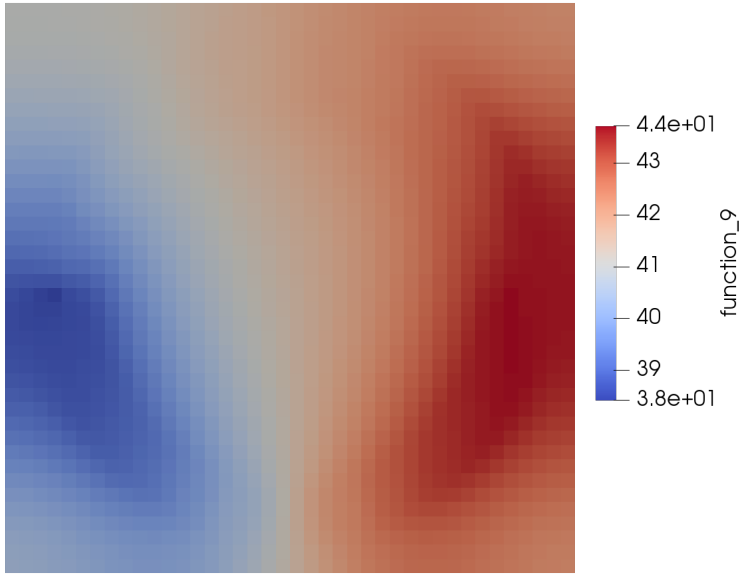
Solution



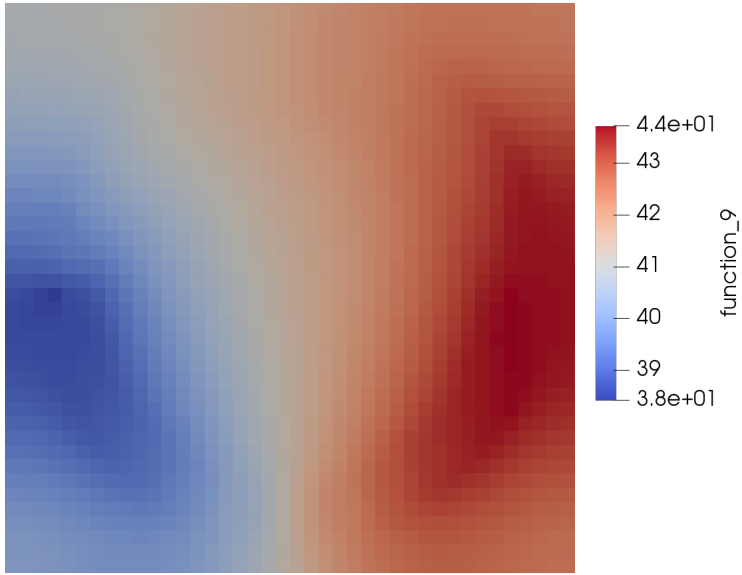
Solution



Solution



Solution



Systems of PDEs


```
V = FunctionSpace(mesh, "DQ", 0)
W = V*V

u = Function(W)
u_ = Function(W)
(p, T) = split(u)
(p_, T_) = split(u_)
q, r = TestFunctions(W)
```

Adding conservation of energy:

```
a_Eaccum = phi*c_v*(rho(p,T)*T - rho(p_,T_)*T_)/dt*r*dx \
           + (1-phi)*rho_r*c_r*(T - T_)/dt*r*dx
a_advec = K_facet*conditional(gt(jump(p), 0.0), \
          T('+')*rho(p('+'),T('+'))/mu(T('+')), \
          T('-')*rho(p('-'),T('-'))/mu(T('-')))*c_v*jump(r)*jump(p)/Delta_h*dS
a_diff = kT_facet*jump(T)/Delta_h*jump(r)*dS
F = a_accum + a_flow + a_Eaccum + a_diff + a_advec
```

Setting PETSc parameters

PETSc language:

- `snes`: System of Nonlinear Equations Solver
- `ksp`: Krylov methods (linear solver)
- `pc`: Preconditioner

Solver parameters can be given to Firedrake in a dictionary.
For example, we want to use Newton's method with line search for the nonlinear solver.
For the linear solver, we want GMRES with ILU(0) preconditioning.

```
solver_parameters =
 {"snes_type": "newtonls",
  "ksp_type": "gmres",
  "pc_type": "ilu",
  }

solve(F == 0, u, solver_parameters = solver_parameters)
```

Tuning of PETSc parameters

Using GMRES(20) with block ILU(1)

```
solver_parameters =
 {"snes_type": "newtonls",
  "snes_max_it": 15,
  "snes_rtol": 1e-5,
  "snes_stol": 1e-5,

  "ksp_type": "gmres",
  "ksp_pc_side": "right"
  "ksp_max_it": 200,
  "ksp_gmres_restart": 20,
  "ksp_rtol": 1e-3,

  "pc_type": "bjacobi",
  "sub_pc_type": "ilu",
  "sub_pc_factor_levels": 1,
  }
```

Tuning of PETSc parameters

Using GMRES(20) with one AMG V-cycle (Using hypre's BoomerAMG)

```
solver_parameters =
 {"snes_type": "newtonls",
  "snes_max_it": 15,
  "snes_rtol": 1e-5,
  "snes_stol": 1e-5,

  "ksp_type": "gmres",
  "ksp_pc_side": "right"
  "ksp_max_it": 200,
  "ksp_gmres_restart": 20,
  "ksp_rtol": 1e-3,

  "pc_type": "hypre",
  "pc_hypre_type" : "boomeramg",
  "pc_hypre_boomeramg_max_iter": 1,
  }
```

Solving systems of PDEs with block preconditioners

Jacobian of the form:

$$J = \begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix}$$

PETSc allows the use of block preconditioners through PCFieldsplit. For example, we want the following block diagonal preconditioner:

$$P^{-1} = \begin{bmatrix} \texttt{ksp0}(A_{00}) & 0 \\ 0 & \texttt{ksp1}(A_{11}) \end{bmatrix}$$

for given linear solvers `ksp0`, `ksp1`. We set the following options:

```
"pc_type": "fieldsplit",
"pc_fieldsplit_type": "additive",
"fieldsplit_0": ksp0,
"fieldsplit_1": ksp1,
```

Solving systems of PDEs with block preconditioners

Jacobian of the form:

$$J = \begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix}$$

PETSc allows the use of block preconditioners through PCFieldsplit. For example, we want the following block triangular preconditioner:

$$P^{-1} = \begin{bmatrix} I & 0 \\ 0 & \texttt{ksp1}(A_{11}) \end{bmatrix} \begin{bmatrix} I & 0 \\ -A_{10} & I \end{bmatrix} \begin{bmatrix} \texttt{ksp0}(A_{00}) & 0 \\ 0 & I \end{bmatrix} \approx \begin{bmatrix} A_{00} & 0 \\ A_{10} & A_{11} \end{bmatrix}^{-1}$$

for given linear solvers `ksp0`, `ksp1`. We set the following options:

```
"pc_type": "fieldsplit",
"pc_fieldsplit_type": "multiplicative",
"fieldsplit_0": ksp0,
"fieldsplit_1": ksp1,
```

Solving systems of PDEs with block preconditioners

Jacobian of the form:

$$J = \begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} = \begin{bmatrix} I & 0 \\ A_{10}A_{00}^{-1} & I \end{bmatrix} \begin{bmatrix} A_{00} & 0 \\ 0 & S \end{bmatrix} \begin{bmatrix} I & A_{00}^{-1}A_{01} \\ 0 & I \end{bmatrix}$$

where $S = A_{11} - A_{10}A_{00}^{-1}A_{01}$. For example, we want the following Schur complement block preconditioner:

$$P^{-1} = \begin{bmatrix} I & -\texttt{ksp0}(A_{00})A_{01} \\ 0 & I \end{bmatrix} \begin{bmatrix} \texttt{ksp0}(A_{00}) & 0 \\ 0 & \texttt{ksp1}(\tilde{S}) \end{bmatrix} \begin{bmatrix} I & 0 \\ -A_{10}\texttt{ksp0}(A_{00}) & I \end{bmatrix} \approx \begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix}^{-1}$$

for given linear solvers `ksp0`, `ksp1`, and a sparse Schur complement approximation $\tilde{S}$.

```
"pc_fieldsplit_type": "schur",
"pc_fieldsplit_schur_fact_type": "full",
"pc_fieldsplit_schur_precondition": "a11",
"fieldsplit_0": ksp0,
"fieldsplit_1": ksp1,
```

$$\tilde{S} = A_{11}$$

Solving systems of PDEs with block preconditioners

Jacobian of the form:

$$J = \begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} = \begin{bmatrix} I & 0 \\ A_{10}A_{00}^{-1} & I \end{bmatrix} \begin{bmatrix} A_{00} & 0 \\ 0 & S \end{bmatrix} \begin{bmatrix} I & A_{00}^{-1}A_{01} \\ 0 & I \end{bmatrix}$$

where $S = A_{11} - A_{10}A_{00}^{-1}A_{01}$. For example, we want the following Schur complement block preconditioner:

$$P^{-1} = \begin{bmatrix} I & -\texttt{ksp0}(A_{00})A_{01} \\ 0 & I \end{bmatrix} \begin{bmatrix} \texttt{ksp0}(A_{00}) & 0 \\ 0 & \texttt{ksp1}(\tilde{S}) \end{bmatrix} \begin{bmatrix} I & 0 \\ -A_{10}\texttt{ksp0}(A_{00}) & I \end{bmatrix} \approx \begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix}^{-1}$$

for given linear solvers `ksp0`, `ksp1`, and a sparse Schur complement approximation $\tilde{S}$.

```
"pc_fieldsplit_type": "schur",
"pc_fieldsplit_schur_fact_type": "full",
"pc_fieldsplit_schur_precondition": "selfp",
"fieldsplit_0": ksp0,
"fieldsplit_1": ksp1,
```

$$\tilde{S} = A_{11} - A_{10}\mathrm{diag}(A_{00})^{-1}A_{01}$$

Custom Schur complement approximations


```
"pc_fieldsplit_type": "schur",
"pc_fieldsplit_schur_fact_type": "full",
"fieldsplit_0": ksp0,
"fieldsplit_1_ksp_type": "preonly",
"fieldsplit_1_pc_type": "python",
"fieldsplit_1_pc_python_type": "MyCustomPC",
"fieldsplit_1_schur": ksp1,
```

Creating a custom preconditioner using Firedrake:

```
MyCustomPC(PCBase):
    def initialize(self, pc):
        # build Schur complement approximation
        # create self.ksp solver using parameters given in ksp1
    def update(self, pc):
        # e.g. update Schur complement approximation
    def apply(self, pc, X, Y):
        self.ksp.solve(X, Y)
```

ThermalPorous: non-isothermal flow in porous media

Models

- Single phase and two-phase flow (water and heavy oil)
- Source terms: wells (Peaceman), simple heaters
- Cartesian grids with heterogeneous, anisotropic permeability (e.g. SPE10)

Solvers

- All nonlinear and linear solvers from PETSc (Newton + FGMRES)
- Preconditioners: CPR, Block preconditioner, System AMG, CPTR

Missing features:

- Gases and phase change
- Newton with different damping factors for different variables.

https://github.com/tlroy/thermalporous

Let's look at the code now

Thank you!

T. Roy, T.B. Jönsthövel, C. Lemon, and A.J. Wathen. A block preconditioner for non-isothermal flow in porous media. To appear in the *Journal of Computational Physics* (2019). arXiv:1902.00095. 10.1016/j.jcp.2019.06.038

T. Roy, T.B. Jönsthövel, C. Lemon, and A.J. Wathen. A constrained pressure-temperature residual (CPTR) method for non-isothermal multiphase flow in porous media. To be submitted to the *Journal of Computational Physics*.

https://github.com/tlroy/thermalporous