Multiscale and multiphysics models: High level implementation & preconditioning

Miroslav Kuchta
Simula Research Laboratory & University of Oslo

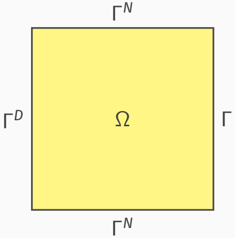
simula

ENUMATH 2019, Egmond aan Zee October 3, 2019

Two language problem of FEniCS

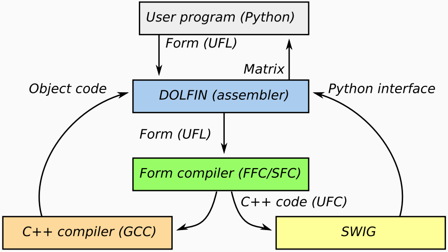
Behind the scenes of "Hello World!" problem

$$
\begin{aligned}
-\Delta u &= f && \text{in } \Omega, \\
u &= g && \text{on } \Gamma^D \cup \Gamma, \\
\partial_n u &= 0 && \text{on } \Gamma^N.
\end{aligned}
$$



Assembly (C++)code generated by FFC(Python) from UFL definition

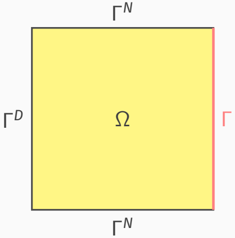
```python
from fenics import *

mesh = UnitSquareMesh(32, 32)

V = FunctionSpace(mesh, "Lagrange", 1)

bc = DirichletBC(V, 1, 'on_boundary')

u, v = TrialFunction(V), TestFunction(V)
a = inner(grad(u), grad(v))*dx
L = inner(Constant(1), v)*dx

A, b = assemble_system(a, L, bc)
```



Two language problem of FEniCS

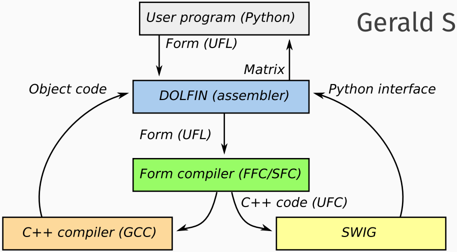
Making multiscale "Hello World!" problem work[1]

$$
\begin{aligned}
-\Delta u &= f && \text{in } \Omega,\\
\text{(weakly) } u &= g && \text{on } \Gamma,\\
u &= g && \text{on } \Gamma^D,\\
\partial_n u &= 0 && \text{on } \Gamma^N.
\end{aligned}
$$



'A diamond is very pretty. But it is very hard to add to a diamond.'

Gerald Sussman paraphrasing Joel Moses

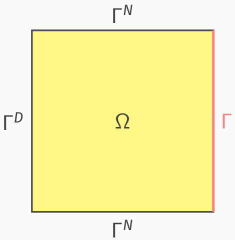


[1] Babuška, I. (1973). The finite element method with Lagrangian multipliers. Numerische Mathematik, 20(3), 179-192.

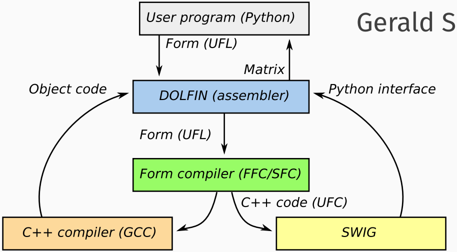
Two language problem of FEniCS

Making multiscale "Hello World!" problem work[1]

$$
\begin{aligned}
-\Delta u &= f && \text{in } \Omega,\\
\text{(weakly) } u &= g && \text{on } \Gamma,\\
u &= g && \text{on } \Gamma^D,\\
\partial_n u &= 0 && \text{on } \Gamma^N.
\end{aligned}
$$



'A diamond is very pretty. But it is very hard to add to a diamond.'
Gerald Sussman paraphrasing Joel Moses



'A ball of mud is not so pretty. But you can always add more mud to a ball of mud.' ——

[1] Babuška, I. (1973). The finite element method with Lagrangian multipliers. Numerische Mathematik, 20(3), 179-192.

Multiscale form interpreter

Block structured operators in $W = V \times Q$, $V = V(\Omega)$, $Q = Q(\Gamma)$

$$\mathcal{A} = \begin{pmatrix} A & B' \\ B & \end{pmatrix} : W \to W' \qquad \langle Bu, q \rangle = \int_\Gamma uq \mathrm{d}x$$

Coupling operator is composite $B = M \circ T$ using trace space V_T

$$T : V \to V_T, Tu = \bar{u} = u|_\Gamma \qquad M : V_T \to Q', \langle Mu, q \rangle = \int_\Gamma uq \mathrm{d}x$$

[2] Mardal, K. A., & Haga, J. B. (2012). Block preconditioning of systems of PDEs. In Automated solution of differential equations by the finite element method (pp. 643-655). Springer, Berlin, Heidelberg.

Observations on structure of multiscale problems

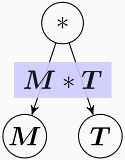
Block structured operators in $W = V \times Q$, $V = V(\Omega)$, $Q = Q(\Gamma)$

$$\mathcal{A} = \begin{pmatrix} A & B' \\ B & \end{pmatrix} : W \to W' \qquad \langle Bu, q \rangle = \int_\Gamma uq \, \mathrm{d}x$$

Coupling operator is composite $B = M \circ T$ using trace space V_T

$$T : V \to V_T, Tu = \bar{u} = u|_\Gamma \qquad M : V_T \to Q', \langle Mu, q \rangle = \int_\Gamma uq \, \mathrm{d}x$$

Matrix-expression *language* to represent the structure[2]




```python
from block import *

# Problem operator
AA = block_mat([[A, B.T],
                [B,   0]])

# AA supports *, +, -
# *vector is application/matvec
# * enough for Krylov solver
```

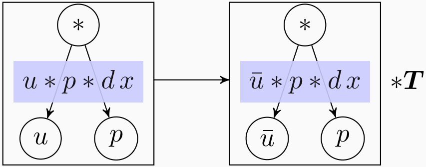
```python
from block import *

# declare Schur complement
S = B*LU(A)*B.T
# A^-1 not computed

# Preconditioner
BB = block_mat([[A, 0],
                [0, S]])
```

Computations are delayed until application needed

[2] Mardal, K. A., & Haga, J. B. (2012). Block preconditioning of systems of PDEs. In Automated solution of differential equations by the finite element method (pp. 643-655). Springer, Berlin, Heidelberg.

Assembly of multiscale forms

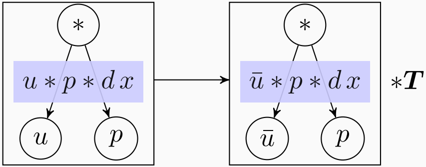
Reflect composition structure; $u \in V(\Omega)$, $p \in Q(\Gamma)$, $\bar{u} \in V_T(\Gamma)$



UFL expression is transformed into matrix expression

Assembly of multiscale forms

Reflect composition structure; $u \in V(\Omega)$, $p \in Q(\Gamma)$, $\bar{u} \in V_T(\Gamma)$



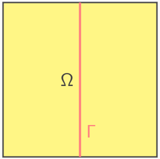
UFL expression is transformed into matrix expression

```python
# xii.py
def assemble(form):
  '''Assemble multidimensional form'''
  if isinstance(form, Form):
    arity = form_arity(form)

    tensor = trace_assembler.assemble(form, arity)
    if tensor is not None:
      return tensor
    # Fallback
    return dolfin.assemble(form)  # <---

  # We might get number
  if is_number(form): return form

  # Recurse on block structured
  blocks = reshape_list(
     map(assemble, flatten_list(form)),
     shape_list(form)
  )
  # cbc.block object
  return (block_vec
          if len(shape) == 1 else block_mat)(blocks)
```

```python
# trace_assembler.py
def assemble(self, form, arity):
  '''Trace assembler'''
  trace_integrals = self.select_integrals(form)
  # Signal for ii assembler
  if not trace_integrals: return None
  # Otherwise we can reduce
  integral = trace_integrals()[0]
  trace_mesh = integral.ufl_domain().ufl_cargo()
  # ... Find argument to be restricted (termimal)

  V = terminal.function_space()
  TV = self.trace_space(V, trace_mesh)
  # Make matrix of trace operator
  T = self.trace_matrix(V, TV)

  if is_trial_function(terminal):
    ubar = dolfin.TrialFunction(TV)
    # Transform form
    integrand = replace(integrand, terminal, ubar)
    trace_form = Form([integral.reconstruct(#...)])
    # Call outside for the rest
    return xii.assemble(trace_form)*T  # <---
```

With $V_H = V_H(\Omega)$, $Q_h = Q_h(\Gamma)$ we consider problem[3]: Find $u \in V_H$, $p \in Q_h$

$$
\begin{aligned}
\int_\Omega \nabla u \cdot \nabla v \,\mathrm{d}x + \quad & \int_\Gamma p v \,\mathrm{d}x = \int_\Omega f v \,\mathrm{d}x \quad \forall v \in V_H, \\
\int_\Gamma q u \,\mathrm{d}x + \quad & - \sum_{F \in \partial \mathcal{S}} \int_F h^2 [\![p]\!][\![q]\!] \mathrm{d}x = \int_\Gamma g q \,\mathrm{d}x \quad \forall q \in Q_h.
\end{aligned}
$$



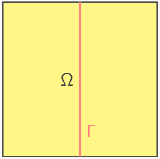

```python
from xii import *
# ... define domains
V = FunctionSpace(omega, 'CG', 1)
Q = FunctionSpace(gamma, 'DG', 0)  # <--
W = [V, Q]

u, p = map(TrialFunction, W)
v, q = map(TestFunction, W)
Tu, Tv = Trace(u, gamma), Trace(v, gamma)

# The line integral
dx_ = Measure('dx', domain=gamma)

a = block_form(W, 2)
a[0][0] = inner(grad(u), grad(v))*dx
a[0][1] = inner(p, Tv)*dx_
a[1][0] = inner(q, Tu)*dx_
# Stabilization term
hk = CellDiameter(gamma)
a[1][1] = -avg(hk)**2*inner(jump(p), jump(q))*dS

A = assemble(a)
```

3 Burman, E. (2014). Projection stabilization of Lagrange multipliers for the imposition of constraints on interfaces and boundaries. Numerical Methods for Partial Differential Equations, 30(2), 567-592.

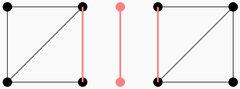
Handling non-conforming geometries

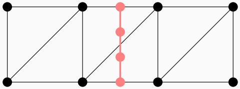
With $V_H = V_H(\Omega)$, $Q_h = Q_h(\Gamma)$ we consider problem[4]: Find $u \in V_H$, $p \in Q_h$

$$
\begin{aligned}
\int_\Omega \nabla u \cdot \nabla v \,\mathrm{d}x + \int_\Gamma p v \,\mathrm{d}x &= \int_\Omega f v \,\mathrm{d}x \quad \forall v \in V_H,\\
\int_\Gamma q u \,\mathrm{d}x + - \sum_{F \in \partial\mathcal{S}} \int_F h^2 [\![p]\!][\![q]\!] \mathrm{d}x &= \int_\Gamma g q \,\mathrm{d}x \quad \forall q \in Q_h.
\end{aligned}
$$



Geometric conformity of meshes is not required





```
omega = UnitSquareMesh(32, 32)
facets = MeshFunction('size_t', omega, 2, 1)
CompiledSubDomain('near(x[0], 0.5)').mark(facets, 1)

gamma = EmbeddedMesh(facets, 1)
```

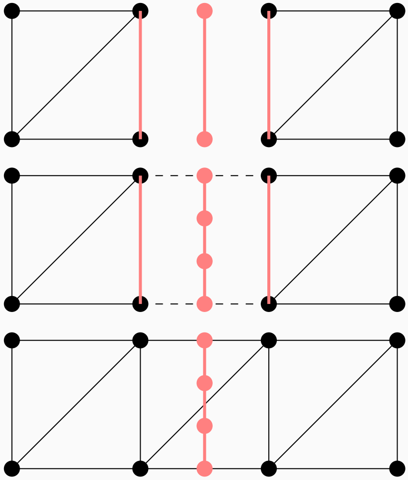
```
omega = UnitSquareMesh(31, 32)  # Odd

A = np.array([0.5, 0])
B = np.array([0.5, 1])
gamma = LineMesh(A, B, 128)
```

The only difference is dedicated trace matrix $\boldsymbol{T}$

[4] Burman, E. (2014). Projection stabilization of Lagrange multipliers for the imposition of constraints on interfaces and boundaries. Numerical Methods for Partial Differential Equations, 30(2), 567-592.

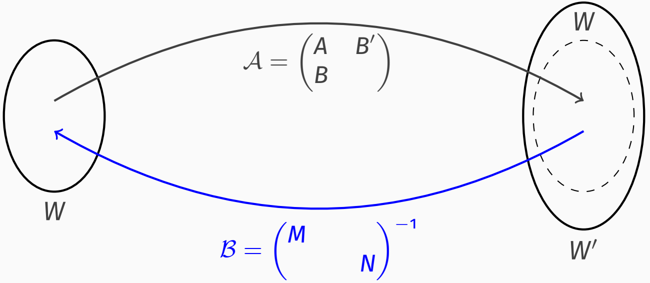
Error convergence using stabilized formulation with P1-P0 elements

H/H_0	$\|u-u_H\|_1$	$\|p-p_H\|_0$
1	8.62E-1(–)	1.04E0(–)
2^{-1}	4.4E-1(0.99)	3.8E-1(1.46)
2^{-2}	2.2E-1(1.00)	1.4E-1(1.49)
2^{-3}	1.1E-1(1.00)	4.8E-2(1.50)
2^{-4}	5.5E-2(1.00)	1.7E-2(1.50)
2^{-5}	2.7E-2(1.00)	6.0E-3(1.50)
H/H_0	$\|u-u_H\|_1$	$\|p-p_h\|_0$
1	8.6E-1(–)	5.4E-1(–)
2^{-1}	4.3E-1(0.98)	2.3E-1(1.20)
2^{-2}	2.2E-1(1.00)	1.0E-1(1.19)
2^{-3}	1.1E-1(1.00)	4.7E-2(1.13)
2^{-4}	5.5E-2(1.00)	2.2E-2(1.08)
2^{-5}	2.7E-2(1.00)	1.1E-2(1.04)
H/H_0	$\|u-u_H\|_1$	$\|p-p_h\|_0$
1	1.3E0(–)	1.7E0(–)
2^{-1}	8.2E-1(0.68)	9.2E-1(0.92)
2^{-2}	5.4E-1(0.63)	4.6E-1(1.00)
2^{-3}	3.6E-1(0.58)	2.3E-1(1.01)
2^{-4}	2.5E-1(0.54)	1.1E-1(1.01)
2^{-5}	1.7E-1(0.52)	5.7E-2(1.01)



Ingredients for multiscale preconditioners

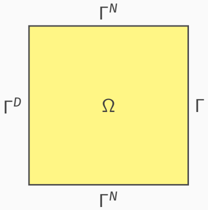
Robust preconditioners by mapping properties[5]



- Let $\mathcal{A} : W \to W'$ an isomorphism (by Brezzi theory)
- Preconditioner $\mathcal{B}$ an isomorphism $W' \to W$; e.g. Riesz map
- Stable discretization yields bounded condition number of $\mathcal{B}_h\mathcal{A}_h$

[5] Mardal, K., & Winther, R. (2011). Preconditioning discretizations of systems of partial differential equations. Numerical Lin. Alg. with Applic., 18, 1-40.

Enforcing Dirichlet boundary condition on Γ by Lagrange multiplier



$$\begin{aligned} -\Delta u &= f & \text{in } \Omega, \\ u &= g & \text{on } \Gamma^D \cup \Gamma, \\ \partial u \cdot n &= 0 & \text{on } \Gamma^N. \end{aligned}$$

Well-posed[6] saddle point problem $W = H^1_{0,\Gamma^D}(\Omega) \times H^{-1/2}(\Gamma)$

$$\mathcal{A}_1 = \begin{pmatrix} -\Delta_\Omega & T' \\ T & \end{pmatrix} \text{ preconditioned by } \mathcal{B} = \begin{pmatrix} -\Delta_\Omega & \\ & -\Delta_\Gamma^{-1/2} \end{pmatrix}^{-1}$$

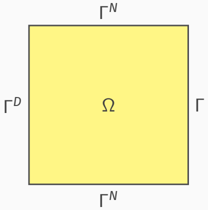
h	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}	2^{-9}
n_{iters}	32	32	32	30	30	28

P1-P1 elements

```
def preconditioner(AA, W):
  '''Babuska preconditioner'''
  _, Q = W

  frac_lap = HsNorm(Q, s=-0.5)
  return block_mat([[AMG(A[0][0]), 0]
                    [0, frac_lap**-1]])
```

6 Babuška, I. (1973). The finite element method with Lagrangian multipliers. Numerische Mathematik, 20(3), 179-192.

Enforcing Dirichlet boundary condition on Γ by Lagrange multiplier



$$
\begin{aligned}
-\nabla(\nabla\cdot u) &= f && \text{in } \Omega,\\
u\cdot n &= g && \text{on } \Gamma^D\cup\Gamma,\\
\nabla\cdot u &= 0 && \text{on } \Gamma^N.
\end{aligned}
$$

Well-posed saddle point problem $W = H_{0,\Gamma^D}(\mathrm{div},\Omega)\times H^{1/2}(\Gamma)$

$$
\mathcal{A}_{\mathrm{div}} = \begin{pmatrix} -\nabla_\Omega(\nabla_\Omega\cdot)+I & T_n' \\ T_n & \end{pmatrix} \qquad \mathcal{B} = \begin{pmatrix} -\nabla_\Omega(\nabla_\Omega\cdot)+I & \\ & -\Delta_\Gamma^{1/2} \end{pmatrix}^{-1}
$$

h	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}	2^{-9}
n_{iters}	19	19	19	17	17	15

RT0-P0 elements, H(div) algebraic multigrid[7]

```
def preconditioner(AA, W):
  '''Hdiv Babuska preconditioner'''
  V, Q = W

  frac_lap = HsNorm(Q, s=0.5)
  return block_mat([[HypreAMS(V),  0]
                    [0, frac_lap**-1]])
```

[7] Kolev, T. V., & Vassilevski, P. S. (2012). Parallel auxiliary space AMG solver for H(div) problems. SIAM Journal on Scientific Computing, 34(6), A3079-A3098.

Realization of fractional operators

Eigenvalue decomposition

- approx. in terms of matrix powers: $-\Delta_h \leftrightarrow A$, $-\Delta^s \approx H^s$

$$H^s = (MU)\Lambda^s(MU)^T \text{ where } AU = MU\Lambda \text{ and } U^T MU = I$$

Geometric multigrid approach[8]

- additive to avoid $-\Delta^s u = f$ on each level, Jacobi smoothers
- extension to $s < 0$ by composition[9]

$$-\Delta^s = (-\Delta)^{\frac{1+s}{2}}(-\Delta)^{-1}(-\Delta)^{\frac{1+s}{2}}$$

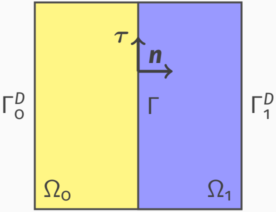
MINRES iterations for $\mathcal{A}_{\text{div}}$, $\mathcal{A}_1$, with Riesz map preconditioners

h	$s = 1/2$				$s = -1/2$			
	#MG			#Eig	#MG			#Eig
	$J = 2$	$J = 3$	$J = 4$		$J = 2$	$J = 3$	$J = 4$	
2^{-6}	25	27	28	22	67	93	103	36
2^{-7}	25	27	29	22	68	92	111	35
2^{-8}	23	27	27	22	66	90	112	35
2^{-9}	22	27	29	22	64	90	112	34
2^{-10}	22	25	29	22	64	88	108	33

[8] Bærland, T., K., M., & Mardal, K. A. (2019). Multigrid Methods for Discrete Fractional Sobolev Spaces. SIAM SISC, 41(2), A948-A972.

[9] Bærland, T. (2019). An Auxiliary Space Preconditioner for Fractional Laplacian of Negative Order. arXiv preprint arXiv:1908.04498.

Coupled Darcy-Stokes



$$
\begin{aligned}
-\nabla \cdot (\sigma(u_0, p_0)) &= f_0 && \text{in } \Omega_0,\\
\nabla \cdot u_0 &= 0 && \text{in } \Omega_0,\\
\sigma(u_0, p_0) &= -p_0 I + 2\mu \nabla u_0, &&\\
K^{-1} u_1 + \nabla p_1 &= f_1 && \text{in } \Omega_1,\\
\nabla \cdot u_1 &= 0 && \text{in } \Omega_1,\\
u_0 \cdot n - u_1 \cdot n &= g_D && \text{on } \Gamma,\\
-n \cdot (\sigma \cdot n) &= p_1 - g_n && \text{on } \Gamma,\\
u_1 \cdot \tau &= -\alpha_{\text{BJS}}^{-1} \sqrt{\frac{K}{\mu}}\, \tau \cdot (\sigma \cdot n) + g_t && \text{on } \Gamma.
\end{aligned}
$$

Lagrange multiplier to enforce mass conservation, $\lambda = n \cdot \sigma \cdot n$

$$
\mathcal{A} = \begin{pmatrix}
\alpha_{\text{BJS}} \sqrt{\frac{\mu}{K}} T_\tau' T_\tau - 2\mu \Delta_{\Omega_0} & (\nabla_{\Omega_0} \cdot)' & & & T_0' \\
\nabla_{\Omega_0} \cdot & & & & \\
 & & K^{-1} I & (\nabla_{\Omega_1} \cdot)' & -T_1' \\
 & & \nabla_{\Omega_1} \cdot & & \\
T_0 & & -T_1 & &
\end{pmatrix}
$$

A well posed problem[10] in $H^1(\Omega_0) \times L^2(\Omega_0) \times H(\mathrm{div}, \Omega_1) \times L^2(\Omega_1) \times H^{1/2}(\Gamma)$

Corresponding Riesz map preconditioner not parameter robust

[10] Layton, W. J., Schieweck, F., & Yotov, I. (2002). Coupling fluid flow with porous media flow. SINAL, 40(6), 2195-2218.

Alternative function space setting[11]

$$(K^{-1/2}L^2(\Omega_1) \cap K^{-1/2}H(\mathrm{div},\Omega_1)) \times K^{1/2}L^2(\Omega_1)$$

Related Riesz map yields K-robust Darcy preconditioner

$$\mathcal{B} = \begin{pmatrix} K^{-1}(I - \nabla(\nabla\cdot)) & \\ & KI \end{pmatrix}^{-1}$$

K	h				
	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}
1	6	6	6	6	6
10^{-2}	6	6	6	6	6
10^{-4}	6	6	6	7	7
10^{-6}	6	6	7	7	7
10^{-8}	6	7	7	6	7

[11] Vassilevski, P. S., & Villa, U. (2013). A block-diagonal algebraic multigrid preconditioner for the Brinkman problem. SIAM SISC, 35(5), S3-S17.

Multi-physics/scale preconditioning with gap 1

Alternative function space setting[11]

$$(K^{-1/2}L^2(\Omega_1) \cap K^{-1/2}H(\mathrm{div},\Omega_1)) \times K^{1/2}L^2(\Omega_1)$$

Related Riesz map yields K-robust Darcy preconditioner

$$\mathcal{B} = \begin{pmatrix} K^{-1}(I - \nabla(\nabla\cdot)) & \\ & KI \end{pmatrix}^{-1}$$

K	h				
	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}
1	6	6	6	6	6
10^{-2}	6	6	6	6	6
10^{-4}	6	6	6	7	7
10^{-6}	6	6	7	7	7
10^{-8}	6	7	7	6	7

Reinterpret the mass conservation coupling term

$$\langle \underbrace{u_0\cdot n - u_1\cdot n}_{V=\mu^{1/2}H^{1/2}(\Gamma)\cup K^{-1/2}H^{-1/2}(\Gamma)}, p\rangle \text{ then } p \ni V' = \mu^{-1/2}H^{-1/2}(\Gamma)\cap K^{1/2}H^{1/2}(\Gamma)$$

[11] Vassilevski, P. S., & Villa, U. (2013). A block-diagonal algebraic multigrid preconditioner for the Brinkman problem. SIAM SISC, 35(5), S3-S17.

Multi-physics/scale preconditioning with gap 1

Alternative function space setting[11]

$$(K^{-1/2}L^2(\Omega_1) \cap K^{-1/2}H(\mathrm{div}, \Omega_1)) \times K^{1/2}L^2(\Omega_1)$$

Related Riesz map yields K-robust Darcy preconditioner

$$\mathcal{B} = \begin{pmatrix} K^{-1}(I - \nabla(\nabla\cdot)) & \\ & KI \end{pmatrix}^{-1}$$

K	h				
	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}
1	6	6	6	6	6
10^{-2}	6	6	6	6	6
10^{-4}	6	6	6	7	7
10^{-6}	6	6	7	7	7
10^{-8}	6	7	7	6	7

Reinterpret the mass conservation coupling term

$$\langle \underbrace{u_0 \cdot n - u_1 \cdot n}_{V = \mu^{1/2}H^{1/2}(\Gamma) \cup K^{-1/2}H^{-1/2}(\Gamma)}, p\rangle \text{ then } p \ni V' = \mu^{-1/2}H^{-1/2}(\Gamma) \cap K^{1/2}H^{1/2}(\Gamma)$$

Coupled Darcy-Stokes preconditioner using intersection space of $H^s(\Gamma)$

$$\begin{pmatrix} \alpha_{\mathrm{BJS}}\sqrt{\frac{\mu}{K}}T_\tau' T_\tau - 2\mu\Delta & & & & \\ & \mu^{-1}I & & & \\ & & K^{-1}(I - \nabla(\nabla\cdot)) & & \\ & & & KI & \\ & & & & -\mu^{-1}\Delta_\Gamma^{-1/2} - K\Delta_\Gamma^{1/2} \end{pmatrix}^{-1}$$

[11] Vassilevski, P. S., & Villa, U. (2013). A block-diagonal algebraic multigrid preconditioner for the Brinkman problem. SIAM SISC, 35(5), S3-S17.

Discretization by (RT0-P0)-(P2-P1)-P0 elements[12]

α_{BJS}	μ	K	$h = 2^{-4}$	2^{-5}	2^{-6}	2^{-7}	2^{-8}
1	1	1	38	38	36	36	36
		10^{-4}	40	40	41	41	40
		10^{-8}	31	31	31	31	30
	10^{-4}	1	52	53	55	55	55
		10^{-4}	42	44	45	47	47
		10^{-8}	34	34	34	34	35
	10^{-8}	1	52	55	56	56	58
		10^{-4}	52	54	55	56	56
		10^{-8}	42	44	45	47	48
10^{-2}	1	1	38	38	37	36	36
		10^{-4}	40	40	40	39	39
		10^{-8}	31	31	31	31	32
	10^{-4}	1	54	56	56	56	56
		10^{-4}	42	44	45	47	49
		10^{-8}	34	34	34	34	35
	10^{-8}	1	51	53	55	59	60
		10^{-4}	52	54	54	56	56
		10^{-8}	42	44	45	46	48
10^{-4}	1	1	38	38	37	36	36
		10^{-4}	40	40	40	40	40
		10^{-8}	30	30	30	30	30
	10^{-4}	1	54	56	56	56	56
		10^{-4}	44	44	46	48	48
		10^{-8}	34	34	35	35	34
	10^{-8}	1	54	57	58	58	62
		10^{-4}	52	53	55	58	59
		10^{-8}	42	44	45	47	47

[12] Holter, K. E., K., M. & Mardal, K.-A. (in prep). Robust preconditioning of monolithically coupled multiphysics problems.

H^s intersection preconditioner for Stokes-Biot coupling

Simplified problem $\delta t = \lambda = \nu = 1$, null storage coefficient and α_{BJS}

```
# Stokes
V0 = VectorFunctionSpace(omega0, 'CG', 2)
Q0 = FunctionSpace(omega0, 'CG', 1)
# Biot
W = VectorFunctionSpace(omega1, 'CG', 2)
V1 = FunctionSpace(omega1, 'RT', 1)
Q1 = FunctionSpace(omega1, 'DG', 0)
# Lagrange
Q = FunctionSpace(gamma, 'DG', 0)

M = [V0, Q0, W, V1, Q1, Q]

u0, p0, eta, u1, p1, p = map(TrialFunction, M)
v0, q0, w, v1, q1, q = map(TestFunction, M)
# Traces
Tw, Tv0, Tv1 = (Trace(f, gamma) for f in (w, v0, v1))

dX = Measure('dx', domain=gamma)  # Iface measure
n0 = Constant((1, 0))  # Normal from Stokes
# Ambartsumyan, Khattatov, Yotov, Zunino (2018)
a = block_form(M, 2)
a.add(2*nu*inner(sym(grad(u0)), sym(grad(v0)))*dx
      -inner(p0, div(v0))*dx) # Stokes bit

a.add(2*mu*inner(sym(grad(eta)), sym(grad(w)))*dx +\
      lmbda*inner(div(eta), div(w))*dx -\
      inner(p1, div(w))*dx) # Biot bit

a.add((1./K)*inner(u1, v1)*dx
      -inner(p1, div(v1))*dx) # Darcy part

# Interface coupling u0.n0 + (u1 + eta).n1 = ...
a.add(inner(p, dot(Tv0, n0))*dX +\
            inner(p, dot(Tw, -n0))*dX +\
            inner(p, dot(Tv1, -n0))*dX)

a = make_selfadjoint(a)  # Save us some typing
```

H^s intersection preconditioner for Stokes-Biot coupling

Simplified problem $\delta t = \lambda = \nu = 1$, null storage coefficient and α_{BJS}

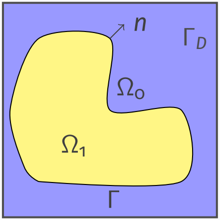
```python
# Stokes
V0 = VectorFunctionSpace(omega0, 'CG', 2)
Q0 = FunctionSpace(omega0, 'CG', 1)
# Biot
W = VectorFunctionSpace(omega1, 'CG', 2)
V1 = FunctionSpace(omega1, 'RT', 1)
Q1 = FunctionSpace(omega1, 'DG', 0)
# Lagrange
Q = FunctionSpace(gamma, 'DG', 0)

M = [V0, Q0, W, V1, Q1, Q]

u0, p0, eta, u1, p1, p = map(TrialFunction, M)
v0, q0, w, v1, q1, q = map(TestFunction, M)
# Traces
Tw, Tv0, Tv1 = (Trace(f, gamma) for f in (w, v0, v1))

dX = Measure('dx', domain=gamma)  # Iface measure
n0 = Constant((1, 0))  # Normal from Stokes
# Ambartsumyan, Khattatov, Yotov, Zunino (2018)
a = block_form(M, 2)
a.add(2*nu*inner(sym(grad(u0)), sym(grad(v0)))*dx
      -inner(p0, div(v0))*dx) # Stokes bit

a.add(2*mu*inner(sym(grad(eta)), sym(grad(w)))*dx +\
      lmbda*inner(div(eta), div(w))*dx -\
      inner(p1, div(w))*dx) # Biot bit

a.add((1./K)*inner(u1, v1)*dx
      -inner(p1, div(v1))*dx) # Darcy part

# Interface coupling u0.n0 + (u1 + eta).n1 = ...
a.add(inner(p, dot(Tv0, n0))*dX +\
            inner(p, dot(Tw, -n0))*dX +\
            inner(p, dot(Tv1, -n0))*dX)

a = make_selfadjoint(a)  # Save us some typing
```

K	μ	h: 2^{-5}	2^{-6}	2^{-7}	2^{-8}
1	1	60	60	62	62
	10^{-2}	65	65	67	68
	10^{-4}	74	74	72	72
	10^{-8}	75	75	75	74
	10^{-10}	73	73	73	71
10^{-2}	1	72	73	73	74
	10^{-2}	74	74	73	73
	10^{-4}	79	77	76	75
	10^{-8}	75	75	74	74
	10^{-10}	73	73	72	72
10^{-4}	1	72	75	75	73
	10^{-2}	72	73	73	73
	10^{-4}	74	76	77	75
	10^{-8}	71	73	74	74
	10^{-10}	68	70	71	71
10^{-8}	1	50	48	48	50
	10^{-2}	50	51	51	53
	10^{-4}	54	54	54	55
	10^{-8}	54	54	56	57
	10^{-10}	53	54	54	56
10^{-10}	1	48	46	47	48
	10^{-2}	49	50	50	50
	10^{-4}	53	53	52	52
	10^{-8}	53	53	53	54
	10^{-10}	53	53	53	53

3D-1D COUPLED PROBLEMS

Crude taxonomy of 3D-1D multiscale models

Simple coupled diffusion with $\Omega \subset \mathbb{R}^3$, Γ of codimension 1



$$
\begin{aligned}
-\nabla \cdot (\kappa_0 \nabla u_0) &= 0 && \text{in } \Omega_0,\\
-\nabla \cdot (\kappa_1 \nabla u_1) &= 0 && \text{in } \Omega_1,\\
-(\kappa_0 \nabla u_0) \cdot n + (\kappa_1 \nabla u_1) \cdot n &= 0 && \text{on } \Gamma,\\
u_0 - u_1 - \epsilon^{-1}(\kappa_0 \nabla u_0) \cdot n &= g_\Gamma && \text{on } \Gamma.
\end{aligned}
$$

Two variable formulation

$$\mathcal{A}_0 = \begin{pmatrix} -\Delta + \epsilon T_0' T_0 & -\epsilon T_1' \\ -\epsilon T_0 & -\Delta + \epsilon I \end{pmatrix}$$

Lagrange multiplier formulation (p)

$$\mathcal{A}_1 = \begin{pmatrix} -\Delta & & T_0' \\ & -\Delta & -T_1' \\ T_0 & -T_1 & -\epsilon^{-1} I \end{pmatrix}$$

Typical coupling terms

$$\int_\Gamma u_0 v_0 \,\mathrm{d}x, \int_\Gamma u_0 v_1 \,\mathrm{d}x \qquad \int_\Gamma v_0 p \,\mathrm{d}x, \int_\Gamma v_1 p \,\mathrm{d}x$$

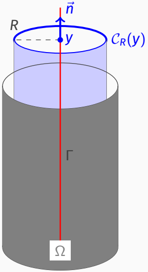
Assume $|\Omega_1| \ll |\Omega_0|$, model order reduction $\Omega_1 \to 0$, Γ a curve, $\Omega = \Omega_0$

For Γ of codimension 2 models differ by "generalization" of coupling terms

Non-symmetric problem in weighted spaces

Generalize coupling terms for $u_0 \in H^1_\alpha(\Omega)$, $v_0 \in H^1_{-\alpha}(\Omega)$[13]

$$
\begin{aligned}
\int_\Gamma u_0 v_0 \mathrm{d}x &\to \int_\Gamma (Tv_0)(\Pi u_0)\mathrm{d}x,\\
\int_\Gamma u_0 v_1 \mathrm{d}x &\to \int_\Gamma (\Pi u_0) v_1 \mathrm{d}x,\\
\int_\Gamma v_0 u_1 \mathrm{d}x &\to \int_\Gamma (Tv_0) u_1 \mathrm{d}x.
\end{aligned}
$$



$(\Pi u)(y) = \frac{1}{|C_R(y)|}\int_{C_R(y)} u \mathrm{d}x$

Problem operator $\mathcal{A} : (H^1_\alpha(\Omega) \times H^1(\Gamma)) \to (H^1_{-\alpha}(\Omega) \times H^1(\Gamma))'$, but standard FEM

$$
\mathcal{A} = \begin{pmatrix} -\Delta + \epsilon T'\Pi & -\epsilon T' \\ -\epsilon \Pi & -\pi R^2 \Delta + \epsilon I \end{pmatrix}
$$

Newly $\epsilon \sim R$

```
# Averaging surface using 3rd order quarature
shape = Circle(radius=radius, degree=3)
c = pi*radius**2

Pi0 = Average(u0, gamma, shape=shape)
Tv0 = Average(v0, gamma, shape=None)  # 3d-1d trace

dX = Measure('dx', domain=gamma)

a = block_form(W)
a[0][0] = inner(grad(u0), grad(v0))*dx + e*inner(Tv0, Piu0)*dX
a[0][1] = -e*inner(u1, Tv0)*dX
a[1][0] = -e*inner(Piu0, v1)*dX
a[1][1] = c*inner(grad(v1), grad(v1))*dX + e*inner(v1, v1)*dX
```

[13] D'Angelo, C., & Quarteroni, A. (2008). On the coupling of 1d and 3d diffusion-reaction equations: application to tissue perfusion problems. Mathematical Models and Methods in Applied Sciences, 18(08), 1481-1504.

Extending multiscale assembler and interpreter

Multiscale assemblers loop to transform UFL expression to matrix expression

```python
def assemble(form):
  '''Assemble multidimensional form'''
  if isinstance(form, Form):
    arity = form_arity(form)



    tensor = trace_assembler.assemble(form, arity)
    if tensor is not None:
      return tensor
     # Fallback
     return dolfin.assemble(form)  # <---
  # Handle other form types as before
```

```python
def assemble(form):
  '''Assemble multidimensional form'''
  if isinstance(form, Form):
    arity = form_arity(form)

    assemblers = (trace_assembler, avg_assembler)
    for assembler in assemblers:
      tensor = trace_assembler.assemble(form, arity)
      if tensor is not None:
        return tensor
     # Fallback
     return dolfin.assemble(form)  # <---
  # ... rest remains unchanged
```

Extending multiscale assembler and interpreter

Multiscale assemblers loop to transform UFL to matrix expression

```python
def assemble(form):
  '''Assemble multidimensional form'''
  if isinstance(form, Form):
    arity = form_arity(form)



    tensor = trace_assembler.assemble(form, arity)
    if tensor is not None:
      return tensor
     # Fallback
     return dolfin.assemble(form)  # <---
  # Handle other form types as before
```

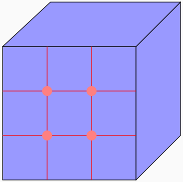
```python
def assemble(form):
  '''Assemble multidimensional form'''
  if isinstance(form, Form):
    arity = form_arity(form)

    assemblers = (trace_assemler, avg_assembler)
    for assembler in assemblers:
      tensor = trace_assembler.assemble(form, arity)
      if tensor is not None:
        return tensor
     # Fallback
     return dolfin.assemble(form)  # <---
  # ... rest remains unchanged
```

Average operator adds new form transformation and discrete Π

```python
def assemble(form):
  '''Trace assembler'''
  trace_integrals = self.select_integrals(form)
  # Signal for ii assembler
  if not trace_integrals: return None
  # Otherwise we can reduce
  integral = trace_integrals()[0]
  trace_mesh = integral.ufl_domain().ufl_cargo()
  # ... Find argument to be restriced (terminal)

  V = terminal.function_space()
  TV = self.trace_space(V, trace_mesh)
  # Make matrix of trace operator
  T = self.trace_matrix(V, TV)

  if is_trial_function(terminal):
    # ... as before
    return xii.assemble(trace_form)*T  # <---
```

```python
def assemble(form):
  '''Average assembler'''
  avg_integrals = self.select_integrals(form)  # <-
  # Signal for ii assembler
  if not avg_integrals: return None
  # Otherwise we can reduce
  integral = avg_integrals()[0]
  line_mesh = integral.ufl_domain().ufl_cargo()
  # ... Find argument to be restriced (terminal)

  V = terminal.function_space()
  TV = self.average_space(V, line_mesh)
  # Make matrix of trace operator
  Pi = self.average_matrix(V, TV, #...)  # <-

  if is_trial_function(terminal):
    # Identical to trace -> avg_form
    return xii.assemble(avg_form)*Pi  # <-
```

Every $3d$-$1d$ of u_0, v_0 rescriction realized by Π yields[14]



$$\mathcal{A} = \begin{pmatrix} -\Delta + \epsilon\Pi'\Pi & -\epsilon\Pi' \\ -\epsilon\Pi & -\pi R^2\Delta + \epsilon I \end{pmatrix}$$

Well posed as $\mathcal{A} : W \rightarrow W'$, $W = H^1(\Omega) \times H^1(\Gamma)$

Functional setting inspired preconditioners

$$\mathcal{B}_1 = \begin{pmatrix} -\Delta + \epsilon\Pi'\Pi & \\ & -\pi R^2\Delta + \epsilon I \end{pmatrix}^{-1} \quad \mathcal{B}_2 = \begin{pmatrix} -\Delta & \\ & -\pi R^2\Delta + \epsilon I \end{pmatrix}^{-1}$$

In addition, system amenable to AMG as $\mathcal{A}$ SPD

PCG iterations[15] for $R = 0.1, 0.05, 0.025$

N	$\mathcal{B}_1$			$\mathcal{B}_2$			$\mathcal{A}^{-1}$		
4	32	29	29	35	27	28	3	3	3
8	33	32	33	35	27	31	3	3	3
16	36	30	33	39	28	31	3	3	3
32	37	31	33	36	27	32	4	4	4
64	40	30	32	40	26	31	4	4	4

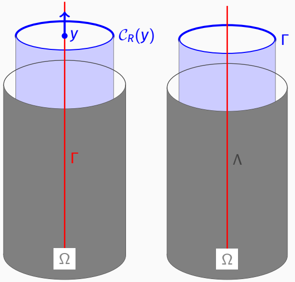
[14] Cerroni, D., Laurino, F., & Zunino, P. (2019). Mathematical analysis, finite element approximation and numerical solvers for the interaction of 3D reservoirs with 1D wells. GEM-International Journal on Geomathematics, 10(1), 4.

[15] Structured mesh of $[-1, 1]^3$ with $6N^3$ cells, wireframe Γ

The coupling manifold not necessary identical to reduced domain [16]

$$\mathcal{A}_1 = \begin{pmatrix} -\kappa\Delta_\Omega & & \epsilon\Pi' \\ & -R^2\hat{\kappa}\Delta_\Gamma & -I' \\ \epsilon\Pi & -I & \end{pmatrix}$$

$$\mathcal{A}_2 = \begin{pmatrix} -\kappa\Delta_\Omega & & \Pi' \\ & -R^2\hat{\kappa}\Delta_\Lambda & -\hat{\Pi}' \\ \Pi & -\hat{\Pi} & \end{pmatrix}$$



Uniform extension (1d-2d) operator Λ to Γ

$$\hat{\Pi}u_1|_{\mathcal{C}_R(y)} = u_1(y)$$

16 K., M., Laurino, F., Mardal, K.A., & Zunino, P. (in prep. 2019). Coupling PDEs on 3D-1D domains with Lagrange multipliers.

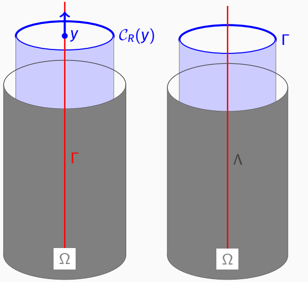
17 Kuchta, M., Mardal, K. A., & Mortensen, M. (2019). Preconditioning trace coupled 3d-1d systems using fractional Laplacian. Numerical Methods for Partial Differential Equations, 35(1), 375-393.

Symmetric problem with fractional spaces for multiplier

The coupling manifold not necessary identical to reduced domain [16]

$$\mathcal{A}_1 = \begin{pmatrix} -\kappa\Delta_\Omega & & \epsilon\Pi' \\ & -R^2\hat{\kappa}\Delta_\Gamma & -I' \\ \epsilon\Pi & -I & \end{pmatrix}$$

$$\mathcal{A}_2 = \begin{pmatrix} -\kappa\Delta_\Omega & & \Pi' \\ & -R^2\hat{\kappa}\Delta_\Lambda & -\hat{\Pi}' \\ \Pi & -\hat{\Pi} & \end{pmatrix}$$



Uniform extension ($1d$-$2d$) operator Λ to Γ

$$\hat{\Pi}u_1|_{\mathcal{C}_R(y)} = u_1(y)$$

Well-posed operators $\mathcal{A}_i : W \to W'$ with $W = H^1(\Omega) \times H^1(\Lambda) \times H^{-1/2}(\Gamma)$

$$\mathcal{B} = \begin{pmatrix} -\Delta_\Omega & & \\ & -R^2\Delta_\Lambda & \\ & & (-\Delta_\Gamma)^{-1/2} \end{pmatrix}^{-1}$$

h	2^{-2}	2^{-3}	2^{-4}	2^{-5}
$\mathrm{cond}(\mathcal{B}\mathcal{A}_1)$	40.5	40.6	40.6	40.6
$\mathrm{cond}(\mathcal{B}\mathcal{A}_2)$	27.4	27.5	27.6	27.6

If $3d$-$1d$ trace used in $\mathcal{A}_1$ then B based on unusual($s = -0.14$) powers of $-\Delta_\Gamma$ [17]

[16] K., M., Laurino, F., Mardal, K.A., & Zunino, P. (in prep. 2019). Coupling PDEs on 3D-1D domains with Lagrange multipliers.

[17] Kuchta, M., Mardal, K. A., & Mortensen, M. (2019). Preconditioning trace coupled 3d-1d systems using fractional Laplacian. Numerical Methods for Partial Differential Equations, 35(1), 375-393.

Conclusions and ongoing work

Robust monolithic solvers for multiphysics/multiscale problems rely on operators in fractional Sobolev spaces

$$\begin{pmatrix} \mathrm{FE} & & \\ & \mathrm{ni} & \\ & & \mathrm{CS} \end{pmatrix}_{ii}$$

`https://github.com/MiroK/fenics_ii/`

- Intersection spaces require efficient solvers for

$$(-\alpha\Delta^s - \beta\Delta^t)u = f$$

- H^s *algebraic* multigrid
- Analysis/solvers for floating domains and self-intersecting interfaces

Conclusions and ongoing work

Robust monolithic solvers for multiphysics/multiscale problems rely on operators in fractional Sobolev spaces

$$\begin{pmatrix} \mathrm{FE} & & \\ & \mathrm{ni} & \\ & & \mathrm{CS} \end{pmatrix}_{ii}$$

https://github.com/MiroK/fenics_ii/

- Intersection spaces require efficient solvers for

$$(-\alpha\Delta^s - \beta\Delta^t)u = f$$

- H^s *algebraic* multigrid
- Analysis/solvers for floating domains and self-intersecting interfaces

Thank you for your attention

FEM implemenation of Stokes-Darcy problem

```
# Stokes
V0 = VectorFunctionSpace(mesh0, 'CG', 2)
Q0 = FunctionSpace(mesh0, 'CG', 1)
# Darcy
V1 = FunctionSpace(mesh1, 'RT', 1)
Q1 = FunctionSpace(mesh1, 'DG', 0)
# The multiplier
Q = FunctionSpace(gamma, 'DG', 0)

W = [V0, Q0, V1, Q1, Q]

u0, p0, u1, p1, p = map(TrialFunction, W)
v0, q0, v1, q1, q = map(TestFunction, W)
# Stokes traces
Tu0, Tv0 = Trace(u0, gamma), Trace(v0, gamma)
# Darcy traces
Tu1, Tv1 = Trace(u1, gamma), Trace(v1, gamma)

# The line integral
dX = Measure('dx', domain=gamma)
n = OuterNormal(gamma)
tau = dot(Constant(((0, 1), (-1, 0))), n)  # Tangent

# ... constants
a = block_form(W, 2)
# Stokes contribution
a.add(inner(2*mu*eps(u0), eps(v0))*dx +\
      BJS*inner(dot(Tu0, tau), dot(Tv0, tau))*dX-\
      inner(p0, div(v0))*dx)
# Darcy contribution
a.add((1/K)*inner(u1, v1)*dx-inner(p1, div(v1))*dx)
# Coupling
a.add(inner(p, dot(Tv0, n))*dX-inner(p, dot(Tv1, n))*dX)

a = make_selfadjoint(a)  # Save time
```