Flexible computational abstractions for complex preconditioners

Lawrence Mitchell[1,*]

P. E. Farrell (Oxford) R. C. Kirby (Baylor) M. G. Knepley (Buffalo) F. Wechsung (Oxford)

March 7, 2019

[1]Department of Computer Science, Durham University

*lawrence.mitchell@durham.ac.uk

Setting

Firedrake www.firedrakeproject.org [...] *is an automated system for the solution of partial differential equations using the finite element method.*

- Finite element problems specified with *embedded* domain specific language, UFL (Alnæs, Logg, Ølgaard, Rognes, and Wells 2014) from the FEniCS project.
- *Runtime* compilation to optimised, low-level (C) code.
- PETSc for meshes and (algebraic) solvers.

Rathgeber et al. (2016) arXiv: 1501.01809 [cs.MS]

UFL makes it easy to write complex PDEs

Rayleigh-Bénard convection

$$-\Delta u + u \cdot \nabla u + \nabla p + \frac{\mathrm{Ra}}{\mathrm{Pr}} \hat{g} T = 0$$

$$\nabla \cdot u = 0$$

$$-\frac{1}{\mathrm{Pr}} \Delta T + u \cdot \nabla T = 0$$

Newton

$$\begin{bmatrix} F & B^T & M_1 \\ C & 0 & 0 \\ M_2 & 0 & K \end{bmatrix} \begin{bmatrix} \delta u \\ \delta p \\ \delta T \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix}$$

```python
from firedrake import *
...
V = VectorFunctionSpace(mesh, "CG", 2)
W = FunctionSpace(mesh, "CG", 1)
Z = V * W * W
Ra = Constant(200)
Pr = Constant(6.18)
upT = Function(Z)
u, p, T = split(upT)
v, q, S = TestFunctions(Z)
bcs = [...] # no-flow + temp gradient

F = (inner(grad(u), grad(v))
   + inner(dot(grad(u), u), v)
   - inner(p, div(v))
   + (Ra/Pr)*inner(T*g, v)
   + inner(div(u), q)
   + inner(dot(grad(T), u), S)
   + (1/Pr) * inner(grad(T), grad(S)))*dx

solve(F == 0, upT, bcs=bcs)
```

UFL makes it easy to write complex PDEs

Ohta–Kawasaki

$$u_t - \Delta w + \sigma(u - m) = 0$$
$$w + \epsilon^2 \Delta u - u(u^2 - 1) = 0$$

Implicit timestepping + Newton

$$\begin{bmatrix} (1 + \Delta t \theta \sigma) M & \Delta t \theta K \\ -\epsilon^2 K - M_E & M \end{bmatrix} \begin{bmatrix} \delta u \\ \delta w \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix}$$

```
from firedrake import *
...
V = FunctionSpace(mesh, "CG", 1)
Z = V*V
ϵ = Constant(0.02);     σ = Constant(100)
dt = Constant(eps**2); θ = Constant(0.5)
v, q = TestFunctions(Z)
z = Function(Z)
z0 = Function(Z)
u, w = split(z)
u0, w0 = split(z0)
uθ = (1 - θ)*u0 + θ*u
wθ = (1 - θ)*w0 + θ*w
dfdu = u**3 - u
F = ((u-u0)*v + dt*dot(grad(wθ), grad(v))
    + dt*σ*(uθ - m)*v
    + w*q - dfdu*q
    - ϵ**2*dot(grad(u), grad(q)))*dx
while t < ...:
    z0.assign(z)
    solve(F == 0, z)
```

What about solvers?

Some motivating problems: block preconditioners

Stokes equations

$$
\begin{aligned}
-\nabla^2 u + \nabla p = f \quad &\text{in } \Omega,\\
\nabla \cdot u = 0 \quad &\text{in } \Omega,
\end{aligned}
$$

Discretising with inf-sup stable element pair results in:

$$
Jx := \begin{bmatrix} A & B^T \\ B & 0 \end{bmatrix} \begin{bmatrix} u \\ p \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix}.
$$

Some motivating problems: block preconditioners

...and a preconditioner

Use a diagonal Schur complement factorisation (Silvester and Wathen 1994)

$$\tilde{J}^{-1} = \begin{bmatrix} \tilde{A}^{-1} & 0 \\ 0 & \tilde{S}^{-1} \end{bmatrix}$$

With multigrid for $\tilde{A}^{-1}$, and $\tilde{S}^{-1} = -Q^{-1}$ (Q the pressure mass matrix).

Some motivating problems: block preconditioners

Stationary Rayleigh-Bénard convection

$$
\begin{aligned}
-\Delta u + u \cdot \nabla u + \nabla p + \frac{\mathrm{Ra}}{\mathrm{Pr}} \hat{g} T &= 0 \\
\nabla \cdot u &= 0 \\
-\frac{1}{\mathrm{Pr}} \Delta T + u \cdot \nabla T &= 0
\end{aligned}
$$

Newton linearisation

$$
\begin{bmatrix} F & B^T & M_1 \\ C & 0 & 0 \\ M_2 & 0 & K \end{bmatrix}
\begin{bmatrix} \delta u \\ \delta p \\ \delta T \end{bmatrix}
=
\begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix}
$$

Some motivating problems: block preconditioners

...and a preconditioner

For each Newton step, invert the 3×3 block system using a preconditioner from Howle and Kirby (2012):

$$\begin{bmatrix} \widetilde{\begin{bmatrix} F & B^T \\ C & 0 \end{bmatrix}}^{-1} & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} I & 0 & -M_1 \\ 0 & I & 0 \\ 0 & 0 & I \end{bmatrix} \begin{bmatrix} I & 0 & 0 \\ 0 & I & 0 \\ 0 & 0 & \tilde{K}^{-1} \end{bmatrix}$$

with

$$\widetilde{\begin{bmatrix} F & B^T \\ C & 0 \end{bmatrix}}^{-1} = \begin{bmatrix} F & 0 \\ 0 & \tilde{S}^{-1} \end{bmatrix} \begin{bmatrix} I & 0 \\ -C & I \end{bmatrix} \begin{bmatrix} \tilde{F}^{-1} & 0 \\ 0 & I \end{bmatrix}$$

with $S = -C\tilde{F}^{-1}B^T$ the Schur complement, whose inverse is approximated with PCD:

$$\tilde{S}^{-1} = \tilde{M}_p^{-1}(\mathbb{I} + F_p\tilde{L}_p^{-1})$$

Some motivating problems: block preconditioners

Ohta–Kawasaki equation: phase separation in polymers

$$
\begin{aligned}
u_t - \Delta w + \sigma(u - m) &= 0 \\
w + \epsilon^2 \Delta u - u(u^2 - 1) &= 0
\end{aligned}
$$

Newton linearisation

$$
\begin{bmatrix} (1 + \Delta t \theta \sigma) M & \Delta t \theta K \\ -\epsilon^2 K - M_E & M \end{bmatrix} \begin{bmatrix} \delta u \\ \delta w \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix}
$$

Some motivating problems: block preconditioners

...and a preconditioner

For each Newton step, invert the 2×2 block system using a preconditioner from Farrell and Pearson (2017):

$$\begin{bmatrix} [(1+\Delta t\theta\sigma)M]^{-1} & 0 \\ 0 & \widetilde{S}^{-1} \end{bmatrix} \begin{bmatrix} I & 0 \\ (\epsilon^2 K + M_E)\,[(1+\Delta t\theta\sigma)M]^{-1} & I \end{bmatrix}.$$

Where the Schur complement is preconditioned by

$$\tilde{S}^{-1} = \hat{S}^{-1} M \hat{S}^{-1}$$

with

$$\hat{S} = M + \epsilon\sqrt{(\Delta t\theta)/(1+\Delta t\theta\sigma)}K.$$

Some motivating problems: block preconditioners

Time dependent Stokes

Implicit time-stepping schemes lead to a saddle-point system

$$\begin{bmatrix} I - \epsilon^2 \Delta & -\,\mathrm{grad} \\ \mathrm{div} & 0 \end{bmatrix}.$$

For which the "canonical" diagonal block preconditioner is

$$\begin{bmatrix} (I - \epsilon^2 \Delta)^{-1} & 0 \\ 0 & (-\Delta)^{-1} + \epsilon^2 I \end{bmatrix}.$$

Mardal and Winther (2011)

Unifying observation

Auxiliary operators

Many schemes require access, *in the preconditioner*, to matrix blocks that are not in the original operator.

Example

- PCD requires a pressure Laplacian, mass matrix, and convection
- Time dependent Stokes needs a pressure Laplacian, and mass matrix

- These operators are typically *easy* for the discretisation library to build.
- How do we get them into the solver?

Idea: "PCFIREDRAKE"

- Endow discretised operators with PDE-level information:
  - what equation/function space?
  - boundary conditions, etc...
- Enable use of PETSc's `fieldsplit` preconditioner for these operators

⇒ gets access to all the block factorisation schemes

Extend PETSc with Firedrake-level (discretisation) preconditioners

✓ PETSc provides *algebraic* composition of solvers.

✓ Firedrake can provide auxiliary operators

✗? Require model developer to write a preconditioner

Kirby and Mitchell (2018) `arXiv:1706.01346 [cs.MS]`

That sounds like hard work

- Thanks to Python, and `petsc4py`, it's not as bad as it sounds.
- We just write a little Python class that implements the application of the preconditioner
- For auxiliary operators, Firedrake provides some extra sugar.
- PETSc manages all the splitting and nesting already. So this does the right thing *inside* multigrid, etc...

That Stokes PC

$$\begin{bmatrix} \tilde{A}^{-1} & 0 \\ 0 & \tilde{Q}^{-1} \end{bmatrix} \begin{bmatrix} A & B^T \\ B & 0 \end{bmatrix}$$

Example: Stokes again

```
from firedrake import *
...
V = VectorFunctionSpace(mesh, "CG", 2)
Q = FunctionSpace(mesh, "CG", 1)
W = V*Q
v, q = TestFunctions(W)

w = Function(W)
u, p = split(w)
F = (inner(grad(u), grad(v))*dx
    - p*div(v)*dx - div(u)*q*dx


class MassMatrix(AuxiliaryOperatorPC):
    _prefix = "mass_"
    def form(self, pc, test, trial):
        a = -inner(test, trial)*dx
        return (a, None)

solve(F == 0, w, solver_parameters=params)
```

```
# Params set up as
-ksp_type gmres
-pc_type fieldsplit
-pc_fieldsplit_type schur
-pc_fieldsplit_schur_fact_type diag
-fieldsplit_0_
   -ksp_type preonly
   -pc_type gamg
-fieldsplit_1_
   -ksp_type chebyshev
   -ksp_max_it 2
   -pc_type python
   # Callback to Firedrake
   -pc_python_type MassMatrix
   -mass_pc_type sor
```

Example: Ohta–Kawasaki

Schur complement approximation

$$\tilde{S}^{-1} = \hat{S}^{-1} M \hat{S}^{-1}$$

where

$$\hat{S} = M + \epsilon\sqrt{(\Delta t\theta)/(1 + \Delta t\theta\sigma)}K.$$

```python
class OKPC(PCBase):
    def setUp(self, pc):
        _, P = pc.getOperators()
        ctx = P.getPythonContext()
        # User information about Δt, θ, etc...
        dt, θ, ϵ, σ = ctx.appctx["parameters"]
        V = ctx.a.arguments()[0].function_space()
        c = (dt * θ)/(1 + dt * θ * σ)
        w = TrialFunction(V)
        q = TestFunction(V)
        # Ŝ = ⟨q, w⟩ + ϵ√c ⟨∇q, ∇w⟩, c = Δtθ/(1+Δtθσ)
        op = assemble(inner(w, q)*dx +
                      ϵ*sqrt(c)*inner(grad(w), grad(q))*dx)
        self.ksp = KSP().create(comm=pc.comm)
        self.ksp.setOptionsPrefix(pc.getOptionsPrefix + "hats_")
        self.ksp.setOperators(op, op)
        self.ksp.setFromOptions()
        self.mass = assemble(w*q*dx)
```

```python
def apply(self, pc, x, y):
    t1, t2 = self.work
    # t_1 ← Ŝ^{-1}x
    self.ksp.solve(x, t1)
    # t_2 ← Mt_1
    self.mass.mult(t1, t2)
    # y ← Ŝ^{-1}t_2 = Ŝ^{-1}MŜ^{-1}x
    self.ksp.solve(t2, y)
```

Solvers on the blocks

Now that I have a hammer...

...can I find some nails?

- This approach gets me auxiliary operators
- and *algebraic* solvers for the blocks
- Many hard problems require multigrid
- ...with something other than point smoothers

Now that I have a hammer...

...can I find some nails?

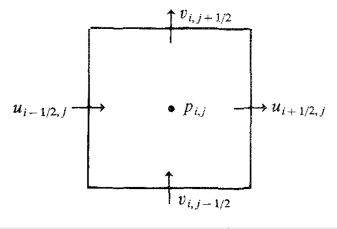
- This approach gets me auxiliary operators
- and *algebraic* solvers for the blocks
- Many hard problems require multigrid
- ...with something other than point smoothers

$\Rightarrow$ Overlapping Schwarz methods

Some motivating problems: multigrid smoothers

Coupled multigrid for Stokes/Navier–Stokes

In the SCGS scheme four velocites and one pressure corresponding to one finite difference node are simultaneously updated by inverting a (small) matrix of equations.

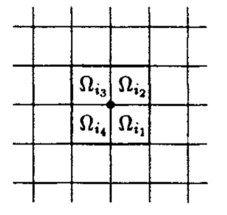


Vanka (1986)

Some motivating problems: multigrid smoothers

p-independent preconditioners for elliptic problems

[Each subspace is generated from] $V_i^p = V^p \cap H_0^1(\Omega_i')$ *where* Ω_i' *is the open square centered at the ith vertex*

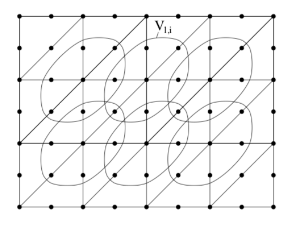


Pavarino (1993)

Some motivating problems: multigrid smoothers

Multigrid for nearly incompressible elasticity

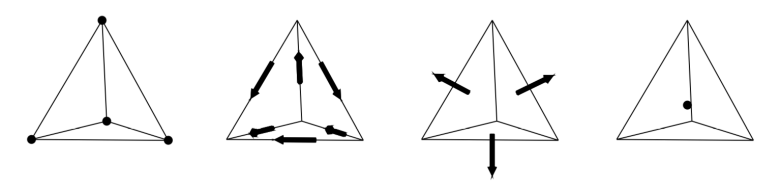
The suggested smoother is block Jacobi smoother, which takes care of the kernel [...]. These kernel basis functions are captured by subspaces $V_{l,i}$ as shown



Schöberl (1999)

Multigrid in $H(\mathrm{div})$ and $H(\mathrm{curl})$

To define the Schwarz smoothers, we can use a decomposition of V_h into local patches consisting of all elements surrounding either an edge or a vertex.

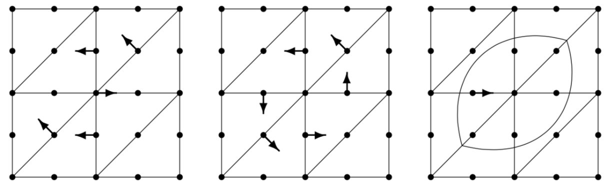


Arnold, Falk, and Winther (2000)

Some motivating problems: multigrid smoothers

An augmented Lagrangian approach to the Oseen problem

We use a block Gauss-Seidel method [...] based on the decomposition $V_h = \sum_{i=0}^{l} V_i$ *[...For] P2-P0 finite elements the natural choice is to gather nodel DOFs for velocity inside ovals [around a vertex]*



Benzi and Olshanskii (2006)

Unifying observation

Abstract relaxation method

Choose a subspace decomposition

$$V = \sum_i V_i$$

solve the problem on each subspace and combine the updates.

Example

If V_i is the span of a single basis function, then we have Jacobi or Gauß-Seidel relaxation (parallel or sequential subspace solves)

- Decompose space (usually) based on some mesh decomposition
- Build and solve little problems on the resulting patches
- Combine additively or multiplicatively

Idea: “PCPATCH”

- Separate topological decomposition from algebraic operators
- Ask discretisation library to make the operators once decomposition is obtained

Idea: "PCPATCH"

- Separate topological decomposition from algebraic operators
- Ask discretisation library to make the operators once decomposition is obtained

Topological decomposition

Use `DMPlex` to provide decomposition of mesh into patches.

Space decomposition

Use topological decomposition plus `PetscSection` to determine degrees of freedom in each patch.

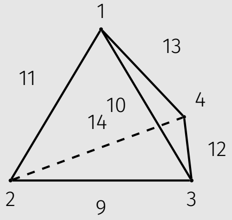
Operators

Callback interface to discretisation/PDE library.

DMPlex notation

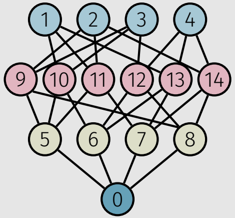
Description of patches uses DMPlex nomenclature

Mesh



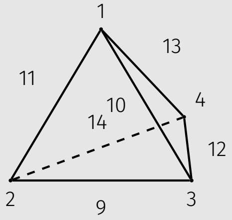
Vertices and edges labelled.

Graph representation



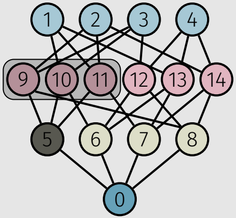
Description of patches uses DMPlex nomenclature

Mesh



Vertices and edges labelled.

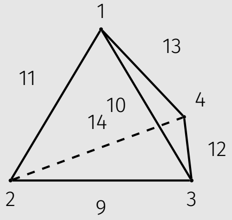
Graph representation



$\mathtt{cone}(5) = \{9, 10, 11\}$

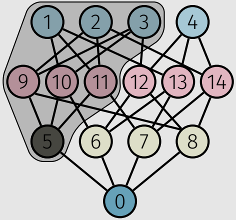
Description of patches uses DMPlex nomenclature

Mesh



Vertices and edges labelled.

Graph representation

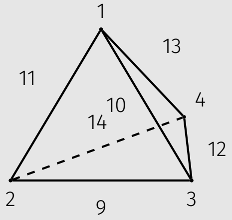


$\texttt{closure}(5) = \{1, 2, 3, 9, 10, 11\}$

DMPlex notation

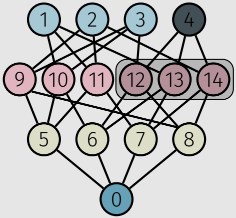
Description of patches uses DMPlex nomenclature

Mesh



Vertices and edges labelled.

Graph representation

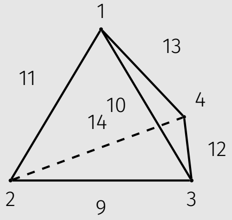


$\texttt{support}(4) = \{12, 13, 14\}$

DMPlex notation

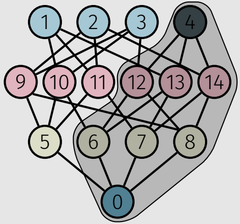
Description of patches uses DMPlex nomenclature

Mesh



Vertices and edges labelled.

Graph representation



$\texttt{star}(4) = \{0, 6, 7, 8, 12, 13, 14\}$

Topological subspace definition via `DMPlex`

- Each patch defined by set of mesh points (entities) on which the dofs we're going to solve for in the patch live

Builtin

Specify patches by selecting:

1. Mesh points $\{p_i\}$ to iterate over (e.g. vertices, cells)
2. Adjacency relation that gathers points in patch

`star` points in star(p_i)
`vanka` points in closure(star(p_i))

User-defined

Callback provides list of entities in each patch, plus iteration order.

Patch assembly

- If we only want homogeneous Dirichlet, can use list of dofs to select from assembled global operator
- Doesn't work for other transmission conditions
- Instead, callback interface
- Extends to nonlinear smoothers

PDE library support

- Works if you use `DMPlex` + `PetscDS`

```
-pc_type patch
```

- Works in Firedrake

```
-pc_type python -pc_python_type firedrake.PatchPC
# Also
-snes_type python -snes_python_type firedrake.PatchSNES
```

Examples

What subspace to choose?

Consider the problem: for $\alpha, \beta \in \mathbb{R}$, find $u \in V$ such that

$$\alpha a(u, v) + \beta b(u, v) = (f, v) \quad \forall v \in V,$$

where a is SPD, and b is symmetric positive semidefinite.

Theorem (Schöberl (1999); Lee, Wu, Xu, Zikatanov (2007))

Let the kernel be

$$\mathcal{N} := \{u \in V : b(u, v) = 0 \ \forall v \in V\}.$$

If the subspace decomposition captures the kernel

$$\mathcal{N} = \sum_i \mathcal{N} \cap V_i,$$

then convergence of the relaxation defined by this decomposition will be independent *of α and β.*

What subspace to choose?

Theorem (Schöberl (1999); Lee, Wu, Xu, Zikatanov (2007))

Let the kernel be

$$\mathcal{N} := \{u \in V : b(u,v) = 0 \ \forall v \in V\}.$$

If the subspace decomposition captures the kernel

$$\mathcal{N} = \sum_i \mathcal{N} \cap V_i,$$

then convergence of the relaxation defined by this decomposition will be independent *of* α *and* β*.*

Corollary

"All" we need to do is characterise the kernel: in particular the support of the basis.

Appropriate discrete de Rham complexes can help.

Nearly incompressible elasticity

Find $u \in V \subset H^1$ s.t. $\quad (\operatorname{grad} u, \operatorname{grad} v) + \gamma(\operatorname{div} u, \operatorname{div} v) = (f, v) \quad \forall v \in V.$

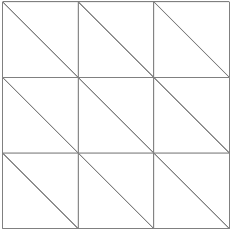
Stokes complex (2D)

$$\mathbb{R} \xrightarrow{\text{id}} H^2 \xrightarrow{\text{grad}^\perp} H^1 \xrightarrow{\text{div}} L^2 \xrightarrow{\text{null}} 0,$$

Decomposition must capture $\ker \operatorname{div} = \operatorname{range} \operatorname{grad}^\perp$.

Appropriate discrete spaces are developed in Morgan and Scott (1975): use star patch around vertices (degree $k \geq 4$).

```
-ksp_type cg
-pc_type mg
-mg_levels_
   -pc_type python
   -pc_python_type firedrake.PatchPC
   -patch_
```



Nearly incompressible elasticity

Find $u \in V \subset H^1$ s.t. $\quad (\operatorname{grad} u, \operatorname{grad} v) + \gamma(\operatorname{div} u, \operatorname{div} v) = (f, v) \quad \forall v \in V.$

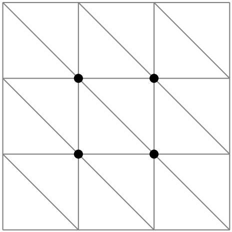
Stokes complex (2D)

$$\mathbb{R} \xrightarrow{\text{id}} H^2 \xrightarrow{\text{grad}^\perp} H^1 \xrightarrow{\text{div}} L^2 \xrightarrow{\text{null}} 0,$$

Decomposition must capture $\ker \operatorname{div} = \operatorname{range} \operatorname{grad}^\perp$.

Appropriate discrete spaces are developed in Morgan and Scott (1975): use star patch around vertices (degree $k \geq 4$).

```
-ksp_type cg
-pc_type mg
-mg_levels_
   -pc_type python
   -pc_python_type firedrake.PatchPC
   -patch_
      -pc_patch_construct_dim 0
```



Nearly incompressible elasticity

Find $u \in V \subset H^1$ s.t. $\quad (\operatorname{grad} u, \operatorname{grad} v) + \gamma(\operatorname{div} u, \operatorname{div} v) = (f, v) \quad \forall v \in V.$

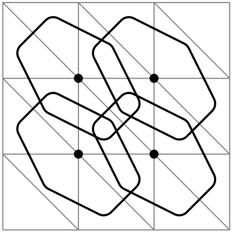
Stokes complex (2D)

$$\mathbb{R} \xrightarrow{\text{id}} H^2 \xrightarrow{\text{grad}^\perp} H^1 \xrightarrow{\text{div}} L^2 \xrightarrow{\text{null}} 0,$$

Decomposition must capture $\ker \operatorname{div} = \operatorname{range} \operatorname{grad}^\perp$.

Appropriate discrete spaces are developed in Morgan and Scott (1975): use star patch around vertices (degree $k \geq 4$).

```
-ksp_type cg
-pc_type mg
-mg_levels_
   -pc_type python
   -pc_python_type firedrake.PatchPC
   -patch_
      -pc_patch_construct_dim 0
      -pc_patch_construct_type star
```



Nearly incompressible elasticity

Find $u \in V \subset H^1$ s.t. $\quad (\operatorname{grad} u, \operatorname{grad} v) + \gamma(\operatorname{div} u, \operatorname{div} v) = (f, v) \quad \forall v \in V.$

Stokes complex (2D)

$$\mathbb{R} \xrightarrow{\text{id}} H^2 \xrightarrow{\operatorname{grad}^\perp} H^1 \xrightarrow{\operatorname{div}} L^2 \xrightarrow{\text{null}} 0,$$

Decomposition must capture $\ker \operatorname{div} = \operatorname{range} \operatorname{grad}^\perp$.

Appropriate discrete spaces are developed in Morgan and Scott (1975): use star patch around vertices (degree $k \geq 4$).

$k \backslash \gamma$	10^0	10^1	10^2	10^3	10^4	10^5
2	11	17	35	54	89	100
3	10	13	23	49	62	82
4	9	10	13	13	13	12
5	9	10	11	12	11	10

Table 1: Convergence in 2D with vertex star patch smoother for P_k elements

Nearly incompressible elasticity

Find $u \in V \subset H^1$ s.t. $\quad (\operatorname{grad} u, \operatorname{grad} v) + \gamma(\operatorname{div} u, \operatorname{div} v) = (f, v) \quad \forall v \in V.$

Stokes complex (3D)

$$\mathbb{R} \xrightarrow{\text{id}} H^2 \xrightarrow{\text{grad}} H^1(\text{curl}) \xrightarrow{\text{curl}} H^1 \xrightarrow{\text{div}} L^2 \xrightarrow{\text{null}} 0,$$

Decomposition must capture $\ker \operatorname{div} = \operatorname{range} \operatorname{curl}$.

Appropriate discrete spaces are developed in Neilan and Sap (2015): use star patch around vertices (degree $k \geq 7$).

$k \backslash \gamma$	10^0	10^1	10^2	10^3	10^4	10^5
3	11	16	29	66	173	458
4	11	13	19	26	54	110
5	11	12	16	19	20	19
6	10	11	14	15	16	15
7	10	11	13	14	14	13

Table 2: Convergence in 3D with vertex star patch smoother for P_k elements

H(div) Riesz map

$$\text{Find } u \in V \subset H(\text{div}) \text{ s.t.} \quad (u, v) + \gamma(\text{div}\, u, \text{div}\, v) = (f, v) \quad \forall v \in V.$$

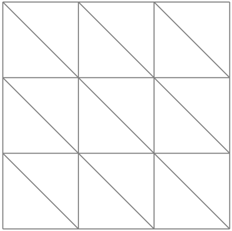
Whitney-Nédélec complex

$$\mathbb{R} \xrightarrow{\text{id}} H^1 \xrightarrow{\text{grad}} H(\text{curl}) \xrightarrow{\text{curl}} H(\text{div}) \xrightarrow{\text{div}} L^2 \xrightarrow{\text{null}} 0,$$

Decomposition must capture $\ker \text{div} = \text{range}\, \text{curl}$.

Choose V to be a Raviart-Thomas space, the matching $H(\text{curl})$ space has dofs on edges: use star patch around edges (or vertices).

```
-ksp_type cg
-pc_type mg
-mg_levels_
   -pc_type python
   -pc_python_type firedrake.PatchPC
   -patch_
```



H(div) Riesz map

$$\text{Find } u \in V \subset H(\mathrm{div}) \text{ s.t.} \quad (u, v) + \gamma(\mathrm{div}\, u, \mathrm{div}\, v) = (f, v) \quad \forall v \in V.$$

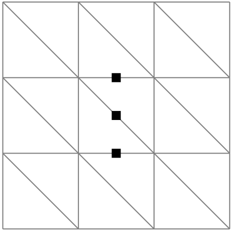
Whitney-Nédélec complex

$$\mathbb{R} \xrightarrow{\mathrm{id}} H^1 \xrightarrow{\mathrm{grad}} H(\mathrm{curl}) \xrightarrow{\mathrm{curl}} H(\mathrm{div}) \xrightarrow{\mathrm{div}} L^2 \xrightarrow{\mathrm{null}} 0,$$

Decomposition must capture $\ker \mathrm{div} = \mathrm{range}\, \mathrm{curl}$.

Choose V to be a Raviart-Thomas space, the matching $H(\mathrm{curl})$ space has dofs on edges: use star patch around edges (or vertices).

```
-ksp_type cg
-pc_type mg
-mg_levels_
   -pc_type python
   -pc_python_type firedrake.PatchPC
   -patch_
      -pc_patch_construct_dim 1
```



H(div) Riesz map

$$\text{Find } u \in V \subset H(\mathrm{div}) \text{ s.t.} \quad (u, v) + \gamma(\mathrm{div}\, u, \mathrm{div}\, v) = (f, v) \quad \forall v \in V.$$

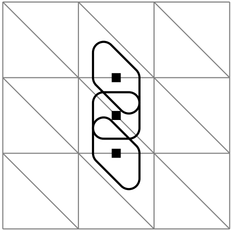
Whitney-Nédélec complex

$$\mathbb{R} \xrightarrow{\mathrm{id}} H^1 \xrightarrow{\mathrm{grad}} H(\mathrm{curl}) \xrightarrow{\mathrm{curl}} H(\mathrm{div}) \xrightarrow{\mathrm{div}} L^2 \xrightarrow{\mathrm{null}} 0,$$

Decomposition must capture $\ker \mathrm{div} = \mathrm{range}\, \mathrm{curl}$.

Choose V to be a Raviart-Thomas space, the matching $H(\mathrm{curl})$ space has dofs on edges: use star patch around edges (or vertices).

```
-ksp_type cg
-pc_type mg
-mg_levels_
   -pc_type python
   -pc_python_type firedrake.PatchPC
   -patch_
      -pc_patch_construct_dim 1
      -pc_patch_construct_type star
```



H(div) Riesz map

$$\text{Find } u \in V \subset H(\mathrm{div}) \text{ s.t.} \quad (u, v) + \gamma(\mathrm{div}\, u, \mathrm{div}\, v) = (f, v) \quad \forall v \in V.$$

Whitney-Nédélec complex

$$\mathbb{R} \xrightarrow{\mathrm{id}} H^1 \xrightarrow{\mathrm{grad}} H(\mathrm{curl}) \xrightarrow{\mathrm{curl}} H(\mathrm{div}) \xrightarrow{\mathrm{div}} L^2 \xrightarrow{\mathrm{null}} 0,$$

Decomposition must capture $\ker \mathrm{div} = \mathrm{range}\, \mathrm{curl}$.

Choose V to be a Raviart-Thomas space, the matching $H(\mathrm{curl})$ space has dofs on edges: use star patch around edges (or vertices).

$k\backslash\gamma$	10^0	10^1	10^2	10^3	10^4	10^5
1 (edge)	32	32	32	32	32	32
2 (edge)	15	15	15	15	15	15
1 (vertex)	11	11	11	11	11	11
2 (vertex)	7	7	7	7	7	7

Table 3: Convergence in 3D with vertex and edge star patch smoothers for RT_k elements

H(curl) Riesz map

$$\text{Find } u \in V \subset H(\mathrm{curl}) \text{ s.t.} \quad (u, v) + \gamma(\mathrm{curl}\, u, \mathrm{curl}\, v) = (f, v) \quad \forall v \in V.$$

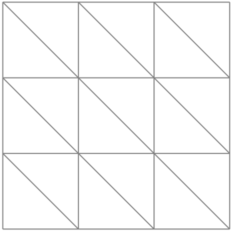
Whitney-Nédélec complex

$$\mathbb{R} \xrightarrow{\mathrm{id}} H^1 \xrightarrow{\mathrm{grad}} H(\mathrm{curl}) \xrightarrow{\mathrm{curl}} H(\mathrm{div}) \xrightarrow{\mathrm{div}} L^2 \xrightarrow{\mathrm{null}} 0,$$

Decomposition must capture $\ker \mathrm{curl} = \mathrm{range}\, \mathrm{grad}$.

Choose V to be a Nédélec space, the matching H^1 space has dofs at vertices: use star patch around vertices.

```
-ksp_type cg
-pc_type mg
-mg_levels_
   -pc_type python
   -pc_python_type firedrake.PatchPC
   -patch_
```



H(curl) Riesz map

$$\text{Find } u \in V \subset H(\text{curl}) \text{ s.t.} \quad (u, v) + \gamma(\text{curl}\, u, \text{curl}\, v) = (f, v) \quad \forall v \in V.$$

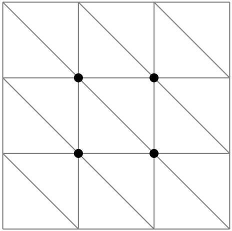
Whitney-Nédélec complex

$$\mathbb{R} \xrightarrow{\text{id}} H^1 \xrightarrow{\text{grad}} H(\text{curl}) \xrightarrow{\text{curl}} H(\text{div}) \xrightarrow{\text{div}} L^2 \xrightarrow{\text{null}} 0,$$

Decomposition must capture $\ker \text{curl} = \text{range}\, \text{grad}$.

Choose V to be a Nédélec space, the matching H^1 space has dofs at vertices: use star patch around vertices.

```
-ksp_type cg
-pc_type mg
-mg_levels_
   -pc_type python
   -pc_python_type firedrake.PatchPC
   -patch_
      -pc_patch_construct_dim 0
```



H(curl) Riesz map

$$\text{Find } u \in V \subset H(\text{curl}) \text{ s.t.} \quad (u, v) + \gamma(\text{curl}\, u, \text{curl}\, v) = (f, v) \quad \forall v \in V.$$

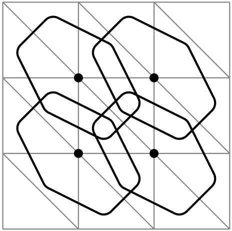
Whitney-Nédélec complex

$$\mathbb{R} \xrightarrow{\text{id}} H^1 \xrightarrow{\text{grad}} H(\text{curl}) \xrightarrow{\text{curl}} H(\text{div}) \xrightarrow{\text{div}} L^2 \xrightarrow{\text{null}} 0,$$

Decomposition must capture $\ker \text{curl} = \text{range}\, \text{grad}$.

Choose V to be a Nédélec space, the matching H^1 space has dofs at vertices: use star patch around vertices.

```
-ksp_type cg
-pc_type mg
-mg_levels_
   -pc_type python
   -pc_python_type firedrake.PatchPC
   -patch_
      -pc_patch_construct_dim 0
      -pc_patch_construct_type star
```



H(curl) Riesz map

Find $u \in V \subset H(\text{curl})$ s.t. $\quad (u, v) + \gamma(\text{curl}\, u, \text{curl}\, v) = (f, v) \quad \forall v \in V.$

Whitney-Nédélec complex

$$\mathbb{R} \xrightarrow{\text{id}} H^1 \xrightarrow{\text{grad}} H(\text{curl}) \xrightarrow{\text{curl}} H(\text{div}) \xrightarrow{\text{div}} L^2 \xrightarrow{\text{null}} 0,$$

Decomposition must capture $\ker \text{curl} = \text{range}\, \text{grad}$.

Choose V to be a Nédélec space, the matching H^1 space has dofs at vertices: use star patch around vertices.

$k \backslash \gamma$	10^0	10^1	10^2	10^3	10^4	10^5
1	13	13	13	13	13	13
2	10	10	10	10	10	10

Table 4: Convergence with vertex star patch preconditioned Richardson smoothers and N1curl_k elements

Vanka for Stokes

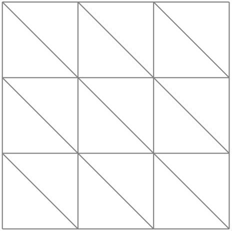
Find $(u, p) \in V \times Q \subset (H^1)^d \times L^2$ s.t.

$$(\operatorname{grad} u, \operatorname{grad} v) - (p, \operatorname{div} v) - (\operatorname{div} u, q) = (f, v) \quad \forall (v, q) \in V \times Q.$$

Vanka patch

Solve simultaneously for (u, p) on each pressure dof, gathering those velocity dofs that couple to the pressure dof.

Marker-and-Cell: loop over cells, gather closure of star



```
-ksp_type cg
-pc_type mg
-mg_levels_
   -pc_type python
   -pc_python_type firedrake.PatchPC
   -patch_
```

Vanka for Stokes

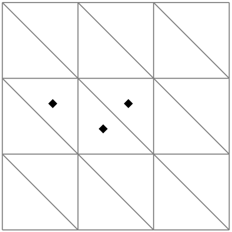
Find $(u, p) \in V \times Q \subset (H^1)^d \times L^2$ s.t.

$$(\operatorname{grad} u, \operatorname{grad} v) - (p, \operatorname{div} v) - (\operatorname{div} u, q) = (f, v) \quad \forall (v, q) \in V \times Q.$$

Vanka patch

Solve simultaneously for (u, p) on each pressure dof, gathering those velocity dofs that couple to the pressure dof.

Marker-and-Cell: loop over cells, gather closure of star



```
-ksp_type cg
-pc_type mg
-mg_levels_
   -pc_type python
   -pc_python_type firedrake.PatchPC
   -patch_
      -pc_patch_construct_codim 0
```

Vanka for Stokes

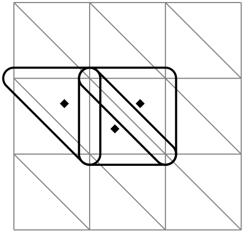
Find $(u, p) \in V \times Q \subset (H^1)^d \times L^2$ s.t.

$$(\operatorname{grad} u, \operatorname{grad} v) - (p, \operatorname{div} v) - (\operatorname{div} u, q) = (f, v) \quad \forall (v, q) \in V \times Q.$$

Vanka patch

Solve simultaneously for (u, p) on each pressure dof, gathering those velocity dofs that couple to the pressure dof.

Marker-and-Cell: loop over cells, gather closure of star



```
-ksp_type cg
-pc_type mg
-mg_levels_
   -pc_type python
   -pc_python_type firedrake.PatchPC
   -patch_
      -pc_patch_construct_codim 0
      -pc_patch_construct_type vanka
      -pc_patch_exclude_subspaces 1
```

Vanka for Stokes

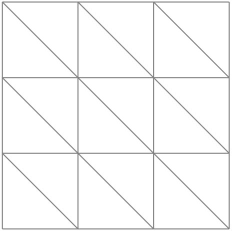
Find $(u, p) \in V \times Q \subset (H^1)^d \times L^2$ s.t.

$$(\text{grad}\, u, \text{grad}\, v) - (p, \text{div}\, v) - (\text{div}\, u, q) = (f, v) \quad \forall (v, q) \in V \times Q.$$

Vanka patch

Solve simultaneously for (u, p) on each pressure dof, gathering those velocity dofs that couple to the pressure dof.

Taylor-Hood: loop over vertices, gather closure of star



```
-ksp_type cg
-pc_type mg
-mg_levels_
   -pc_type python
   -pc_python_type firedrake.PatchPC
   -patch_
```

Vanka for Stokes

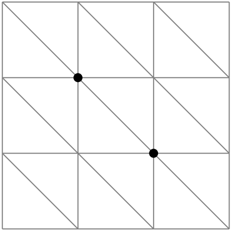
Find $(u, p) \in V \times Q \subset (H^1)^d \times L^2$ s.t.

$$(\mathrm{grad}\, u, \mathrm{grad}\, v) - (p, \mathrm{div}\, v) - (\mathrm{div}\, u, q) = (f, v) \quad \forall (v, q) \in V \times Q.$$

Vanka patch

Solve simultaneously for (u, p) on each pressure dof, gathering those velocity dofs that couple to the pressure dof.

Taylor-Hood: loop over vertices, gather closure of star



```
-ksp_type cg
-pc_type mg
-mg_levels_
   -pc_type python
   -pc_python_type firedrake.PatchPC
   -patch_
      -pc_patch_construct_dim 0
```

Vanka for Stokes

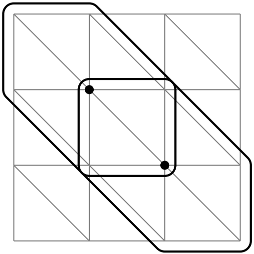
Find $(u, p) \in V \times Q \subset (H^1)^d \times L^2$ s.t.

$$(\operatorname{grad} u, \operatorname{grad} v) - (p, \operatorname{div} v) - (\operatorname{div} u, q) = (f, v) \quad \forall (v, q) \in V \times Q.$$

Vanka patch

Solve simultaneously for (u, p) on each pressure dof, gathering those velocity dofs that couple to the pressure dof.

Taylor-Hood: loop over vertices, gather closure of star



```
-ksp_type cg
-pc_type mg
-mg_levels_
   -pc_type python
   -pc_python_type firedrake.PatchPC
   -patch_
      -pc_patch_construct_dim 0
      -pc_patch_construct_type vanka
      -pc_patch_exclude_subspaces 1
```

Vanka for Stokes

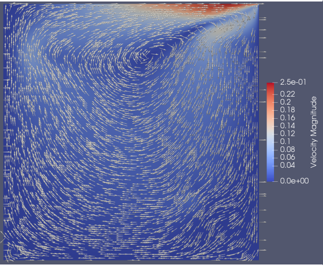
Find $(u, p) \in V \times Q \subset (H^1)^d \times L^2$ s.t.

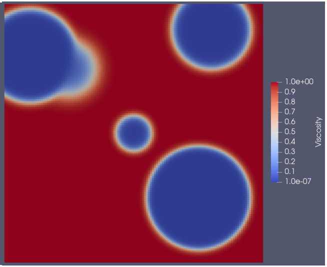
$$(\operatorname{grad} u, \operatorname{grad} v) - (p, \operatorname{div} v) - (\operatorname{div} u, q) = (f, v) \quad \forall (v, q) \in V \times Q.$$

Vanka patch

Solve simultaneously for (u, p) on each pressure dof, gathering those velocity dofs that couple to the pressure dof.

Taylor-Hood: loop over vertices, gather closure of star





- Converges in around 16 iterations, even with large viscosity contrasts (Gaussian bumps)
- Falls over when jumps appear

Divergence free Navier–Stokes with augmented Lagrangian

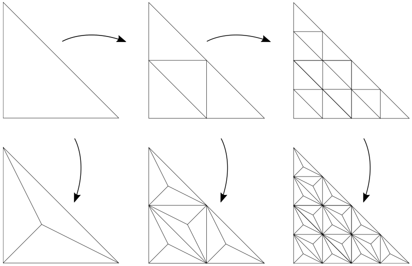
Find $u \in (H^1)^d$ s.t. $\nu(\operatorname{grad} u, \operatorname{grad} v) + (u \cdot \operatorname{grad} u, v) + \gamma(\operatorname{div} u, \operatorname{div} v) = (f, v) \quad \forall v \in V.$

Stokes complex on Alfeld splits

$$\mathbb{R} \xrightarrow{\text{id}} H^2 \xrightarrow{\text{grad}} H^1(\text{curl}) \xrightarrow{\text{curl}} H^1 \xrightarrow{\text{div}} L^2 \xrightarrow{\text{null}} 0,$$

Decomposition must capture $\ker \operatorname{div} = \operatorname{range} \operatorname{curl}$.

Appropriate discrete spaces are constructed in Fu, Guzmán, and Neilan (2018): use barycentrically refined meshes, piecewise continuous space with $k \geq d$, "macro star" patch around vertices.



Divergence free Navier–Stokes with augmented Lagrangian

Find $u \in (H^1)^d$ s.t. $\nu(\operatorname{grad} u, \operatorname{grad} v) + (u \cdot \operatorname{grad} u, v) + \gamma(\operatorname{div} u, \operatorname{div} v) = (f, v) \quad \forall v \in V.$

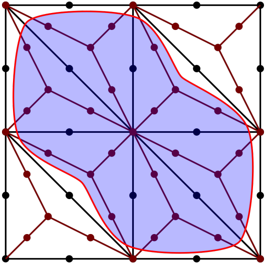
Stokes complex on Alfeld splits

$$\mathbb{R} \xrightarrow{\text{id}} H^2 \xrightarrow{\text{grad}} H^1(\text{curl}) \xrightarrow{\text{curl}} H^1 \xrightarrow{\text{div}} L^2 \xrightarrow{\text{null}} 0,$$

Decomposition must capture $\ker \operatorname{div} = \operatorname{range} \operatorname{curl}$.

Appropriate discrete spaces are constructed in Fu, Guzmán, and Neilan (2018): use barycentrically refined meshes, piecewise continuous space with $k \geq d$, “macro star” patch around vertices.

```
-ksp_type fgmres
-pc_type mg
-mg_levels_
   -ksp_type gmres
   -pc_type python
   -pc_python_type firedrake.PatchPC
   -patch_
      -pc_patch_construct_type python
      -pc_patch_construct_python_type MacroStar
```



Conclusions

- Bidirectional solver/discretisation interface useful
- Playground for preconditioner design: it's easy to get all the operators you need
- With scalability to large problems
- Overlapping Schwarz methods also benefit from this
- With topological decompositions in hand, can provide a *generic* interface, encompassing many useful smoothing schemes

Thanks!

References i

Alnæs, M. S., A. Logg, K. B. Ølgaard, M. E. Rognes, and G. N. Wells (2014). “Unified Form Language: A Domain-specific Language for Weak Formulations of Partial Differential Equations”. *ACM Trans. Math. Softw.* 40. doi:10.1145/2566630. arXiv: 1211.4047 [cs.MS].

Arnold, D. N., R. S. Falk, and R. Winther (2000). “Multigrid in $H(\mathrm{div})$ and $H(\mathrm{curl})$”. *Numerische Mathematik* 85. doi:10.1007/s002110000137.

Benzi, M. and M. A. Olshanskii (2006). “An Augmented Lagrangian-Based Approach to the Oseen Problem”. *SIAM Journal on Scientific Computing* 28. doi:10.1137/050646421.

Brown, J., M. G. Knepley, D. A. May, L. C. McInnes, and B. Smith (2012). “Composable Linear Solvers for Multiphysics”. *Proceedings of the 2012 11th International Symposium on Parallel and Distributed Computing*. ISPDC ’12. Washington, DC, USA: IEEE Computer Society. doi:10.1109/ISPDC.2012.16.

Farrell, P. E. and J. W. Pearson (2017). “A preconditioner for the Ohta-Kawasaki equation”. *SIAM Journal on Matrix Analysis and Applications* 38. arXiv: 1603.04570 [ma.NA].

Fu, G., J. Guzmán, and M. Neilan (2018). *Exact smooth piecewise polynomial sequences on Alfeld splits*. arXiv: 1807.05883 [math.NA].

References ii

Howle, V. E. and R. C. Kirby (2012). "Block preconditioners for finite element discretization of incompressible flow with thermal convection". *Numerical Linear Algebra with Applications* 19. doi:10.1002/nla.1814.

Kirby, R. C. and L. Mitchell (2018). "Solver composition across the PDE/linear algebra barrier". *SIAM Journal on Scientific Computing* 40. doi:10.1137/17M1133208. arXiv: 1706.01346 [cs.MS].

Lee, Y.-J., J. Wu, J. Xu, and L. Zikatanov (2007). "Robust subspace correction methods for nearly singular systems". *Mathematical Models and Methods in Applied Sciences* 17. doi:10.1142/s0218202507002522.

Mardal, K.-A. and R. Winther (2011). "Preconditioning discretizations of systems of partial differential equations". *Numerical Linear Algebra with Applications* 18. doi:10.1002/nla.716.

Morgan, J. and R. Scott (1975). "A nodal basis for C^1 piecewise polynomials of degree $n \geq 5$". *Math. Comp.* 29.

Neilan, M. and D. Sap (2015). "Stokes elements on cubic meshes yielding divergence-free approximations". *Calcolo*. doi:10.1007/s10092-015-0148-x.

Pavarino, L. F. (1993). "Additive Schwarz methods for the p-version finite element method". *Numerische Mathematik* 66. doi:10.1007/BF01385709.

Rathgeber, F. et al. (2016). “Firedrake: automating the finite element method by composing abstractions”. *ACM Transactions on Mathematical Software* 43. doi:10.1145/2998441. arXiv: 1501.01809 [cs.MS].

Schöberl, J. (1999). “Multigrid methods for a parameter dependent problem in primal variables”. *Numerische Mathematik* 84. doi:10.1007/s002110050465.

Silvester, D. and A. J. Wathen (1994). “Fast iterative solution of stabilised Stokes systems. Part II: Using general block preconditioners”. *SIAM Journal on Numerical Analysis* 31. doi:10.1137/0731070.

Vanka, S. (1986). “Block-implicit multigrid solution of Navier-Stokes equations in primitive variables”. *Journal of Computational Physics* 65. doi:10.1016/0021-9991(86)90008-2.