PDEs should be the solver's problem

Lawrence Mitchell[1,*] Rob Kirby[2,†] Patrick Farrell[3,‡]

6th June 2018

[1]Departments of Computing and Mathematics, Imperial College London
[*]lawrence.mitchell@imperial.ac.uk

[2]Department of Mathematics, Baylor University
[†]robert_kirby@baylor.edu

[3]Mathematical Institute, University of Oxford
[‡]patrick.farrell@maths.ox.ac.uk

Block preconditioning

A motivating problem,

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

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

and a preconditioner,

For each Newton step, solve

$$\mathcal{K}\left(\begin{bmatrix} F & B^T & M_1 \\ C & 0 & 0 \\ M_2 & 0 & K \end{bmatrix}, \mathbb{J}\right)$$

using a preconditioner from Howle and Kirby (2012):

$$\mathbb{J} = \begin{bmatrix} \mathcal{K}\left(\begin{bmatrix} F & B^T \\ C & 0 \end{bmatrix}, \mathbb{N}\right) & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} I & 0 & -M_1 \\ 0 & I & 0 \\ 0 & 0 & I \end{bmatrix} \begin{bmatrix} I & 0 & 0 \\ 0 & I & 0 \\ 0 & 0 & \mathcal{K}(K, \mathbb{K}) \end{bmatrix}$$

with

$$\mathbb{N} = \begin{bmatrix} F & 0 \\ 0 & \mathcal{K}(S_p, \mathcal{K}(M_p, \mathbb{M})(\mathbb{I} + F_p\,\mathcal{K}(L_p, \mathbb{L}))) \end{bmatrix} \begin{bmatrix} I & 0 \\ -C & I \end{bmatrix} \begin{bmatrix} \mathcal{K}(F, \mathbb{F}) & 0 \\ 0 & I \end{bmatrix}$$

and

$$S_p = -C\mathcal{K}(F, \mathbb{F})\, B^T.$$

and some solver options

```
-ksp_type fgmres
-pc_type fieldsplit
-pc_fieldsplit_type multiplicative
-pc_fieldsplit_0_fields 0,1
-pc_fieldsplit_1_fields 2
   -fieldsplit_0_
      -ksp_type gmres
      -pc_type fieldsplit
      -pc_fieldsplit_type schur
      -pc_fieldsplit_schur_fact_type lower
      -fieldsplit_0_ksp_type preonly
      -fieldsplit_0_pc_type gamg
      -fieldsplit_1_ksp_type preonly
      -fieldsplit_1_pc_type XXX
   -fieldsplit_1_
      -ksp_type gmres
      -pc_type hypre
```

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

$$\begin{bmatrix} F & 0 \\ 0 & S_p^{-1} \end{bmatrix} \begin{bmatrix} I & 0 \\ -C & I \end{bmatrix} \begin{bmatrix} F^{-1} & 0 \\ 0 & I \end{bmatrix}$$

$$F^{-1} \approx \texttt{gamg}(F)$$

$$S_p^{-1} \approx M_p^{-1}(\mathbb{I} + F_p L_p^{-1}) \quad \text{PCD approximation}$$

$$K^{-1} \approx \texttt{hypre}(K)$$

and some solver options

```
-ksp_type fgmres
-pc_type fieldsplit
-pc_fieldsplit_type multiplicative
-pc_fieldsplit_0_fields 0,1
-pc_fieldsplit_1_fields 2
   -fieldsplit_0_
      -ksp_type gmres
      -pc_type fieldsplit
      -pc_fieldsplit_type schur
      -pc_fieldsplit_schur_fact_type lower
      -fieldsplit_0_ksp_type preonly
      -fieldsplit_0_pc_type gamg
      -fieldsplit_1_ksp_type preonly
      -fieldsplit_1_pc_type XXX
   -fieldsplit_1_
      -ksp_type gmres
      -pc_type hypre
```

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

$$\begin{bmatrix} F & 0 \\ 0 & S_p^{-1} \end{bmatrix} \begin{bmatrix} I & 0 \\ -C & I \end{bmatrix} \begin{bmatrix} F^{-1} & 0 \\ 0 & I \end{bmatrix}$$

$$F^{-1} \approx \texttt{gamg}(F)$$

$$S_p^{-1} \approx M_p^{-1}(\mathbb{I} + F_p L_p^{-1}) \quad \text{PCD approximation}$$

$$K^{-1} \approx \texttt{hypre}(K)$$

Problem

How do I get L_p^{-1}, M_p^{-1}, and F_p into the solver?

Idea

- Endow discretised operators with PDE-level information:
    - what equation/function space?
    - boundary conditions, etc…
- Enable standard fieldsplits on these operators.
- Write custom preconditioners that utilise this information appropriately.

Idea

- Endow discretised operators with PDE-level information:
  - what equation/function space?
  - boundary conditions, etc...
- Enable standard fieldsplits on these operators.
- Write custom preconditioners that utilise this information appropriately.

Extend PETSc with Firedrake-level PCs

- PETSc already provides *algebraic* composition of solvers.
- Firedrake can provide auxiliary operators
- We just need to combine these appropriately.

Implementation: two parts

A new matrix type

A shell matrix that implements matrix-free actions, and contains the symbolic information about the bilinear form.

$y \leftarrow Ax$

```
A = assemble(a, mat_type="matfree")
```

Could do this all with assembled matrices if desired.

Custom preconditioners

These matrices do not have entries, we create preconditioners that inspect the UFL and do the appropriate thing.

$y \leftarrow \tilde{A}^{-1}x$

```
solve(a == L, x,
      {"mat_type": "matfree",
       "pc_type": "python",
       "pc_python_type": "AssembledPC"})
```

This sounds like hard work

Fortunately, `petsc4py` makes it easy to write these PCs.

```
class MyPC(object):
    def setUp(self, pc):
        A, P = pc.getOperators()
        # A and P are shell matrices, carrying the symbolic
        # discretisation information.
        # So I have access to the mesh, function spaces, etc...
        # Can inspect options dictionary here
        # do whatever
    def apply(self, pc, r, e):
        # Compute approximation to error given current residual
        # $e \leftarrow A^{-1} r$

solve(..., solver_parameters={"pc_type": "python",
                              "pc_python_type": "MyPC"})
```

PETSc manages all the splitting and nesting already. So this does the right thing *inside* multigrid, etc...

Back to Rayleigh-Bénard

Problem

How do I get L_p^{-1}, M_p^{-1}, and F_p into the solver?

Back to Rayleigh-Bénard

Problem

How do I get L_p^{-1}, M_p^{-1}, and F_p into the solver?

Solution

Write a custom PC for that makes them, calling back to the PDE library.

Back to Rayleigh-Bénard

```
class PCDPC(PCBase):
    def initialize(self, pc):
        _, P = pc.getOperators()
        prefix = pc.getOptionsPrefix()
        ctx = P.getPythonContext()
        p, q = ctx.a.arguments()
        ...
        # convection operator
        fp = Re*dot(grad(p), u0)*q*dx
        self.Fp = assemble(fp, options_prefix=prefix + "fp_")
        # pressure laplacian
        laplace = inner(grad(p), grad(q))*dx
        Lp = assemble(laplace, bcs=bcs, options_prefix=prefix + "lp_")
        self.Lksp = PETSc.KSP().create(comm=pc.comm)
        self.Lksp.incrementTabLevel(1, parent=pc)
        self.Lksp.setOptionsPrefix(prefix + "lp_")
        self.Lksp.setOperators(Kp.petscmat)
        self.Lksp.setFromOptions()
        # pressure mass matrix
        mass = Re*p*q*dx
        Mp = assemble(mass, options_prefix=prefix + "mp_")
        self.Mksp = PETSc.KSP().create(comm=pc.comm)
        self.Mksp.incrementTabLevel(1, parent=pc)
        self.Mksp.setOptionsPrefix(prefix + "mp_")
        self.Mksp.setOperators(Mp.petscmat)
        self.Mksp.setFromOptions()
```

```
    def apply(self, pc, x, y):
        # $y \leftarrow M^{-1}(\mathbb{I} + F_p L_p^{-1})x$
        z = self.work
        x.copy(z)
        self.bcs.apply(z)
        self.Lksp.solve(z, y)
        self.Fp.petscmat.mult(y, z)
        z.axpy(1.0, x)
        self.Mksp.solve(z, y)
```

and some more solver options

```
-ksp_type fgmres
-pc_type fieldsplit
-pc_fieldsplit_type multiplicative
-pc_fieldsplit_0_fields 0,1
-pc_fieldsplit_1_fields 2
   -fieldsplit_0_
      -ksp_type gmres
      -pc_type fieldsplit
      -pc_fieldsplit_type schur
      -pc_fieldsplit_schur_fact_type lower
      -fieldsplit_0_ksp_type preonly
      -fieldsplit_0_pc_type gamg
      -fieldsplit_1_
         -ksp_type preonly
         -pc_type python
         -pc_python_type PCDPC
         -lp_ksp_type preonly
         -lp_pc_type hypre
         -mp_ksp_type preonly
         -mp_pc_type sor
   -fieldsplit_1_
      -ksp_type gmres
      -pc_type hypre
```

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

$$\begin{bmatrix} F & 0 \\ 0 & S_p^{-1} \end{bmatrix} \begin{bmatrix} I & 0 \\ -C & I \end{bmatrix} \begin{bmatrix} F^{-1} & 0 \\ 0 & I \end{bmatrix}$$

$$F^{-1} \approx \texttt{gamg}(F)$$

$$S_p^{-1} \approx M_p^{-1}(\mathbb{I} + F_p L_p^{-1}) \quad \text{PCD approximation}$$

$$L_p^{-1} \approx \texttt{hypre}(L_p)$$

$$M_p^{-1} \approx \texttt{sor}(M_p)$$

$$K^{-1} \approx \texttt{hypre}(K)$$

Kirby and Mitchell (2018, §B.4) shows a complete solver configuration.

Schwarz smoothers

Now that I have a hammer...

...can I find some nails?

Now that I have a hammer...

...can I find some nails?

Schwarz building blocks

1. Subspace decomposition
2. Operators on subspaces
3. Solvers on subspaces
4. Coarse spaces (not yet)

Now that I have a hammer...

...can I find some nails?

Schwarz building blocks

1. Subspace decomposition
2. Operators on subspaces
3. Solvers on subspaces
4. Coarse spaces (not yet)

`-pc_type patch`

- `DMPlex` + `PetscSection` for subspace decomposition
- Callback interface (to Firedrake for now) to build operators
- `KSP` on each patch for solves

Subspace definition

Each patch defined by set of mesh points on which dofs are free.

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

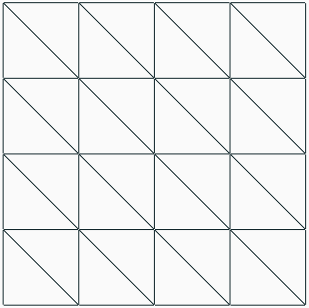
`star`	points in star(p_i)
`vanka`	points in closure(star(p_i))

User-defined

Callback provides `IS`es for each patch, plus iteration order.

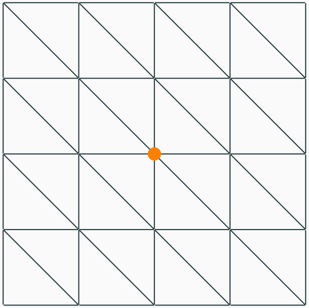
```
PetscErrorCode UserPatches(PC, PetscInt*, IS**, IS*, void*);
```

Patch construction: `star`



Looping over vertices

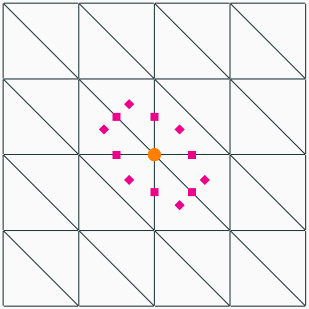
Patch construction: `star`



Looping over vertices

- Select mesh point

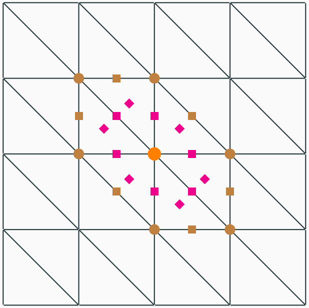
Patch construction: `star`



Looping over vertices

- Select mesh point
- Add points in star

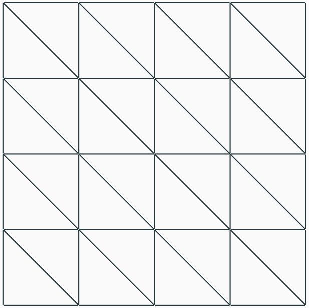
Patch construction: `star`



Looping over vertices

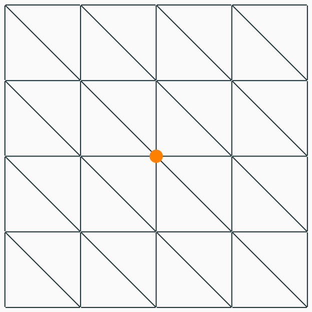
- Select mesh point
- Add points in star
- Complete with FEM adjacency

Patch construction: vanka



Looping over vertices

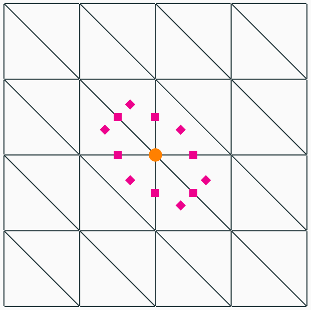
Patch construction: vanka



Looping over vertices

- Select mesh point

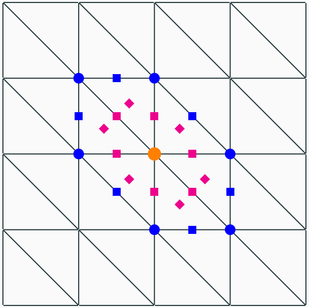
Patch construction: vanka



Looping over vertices

- Select mesh point
- Add points in star

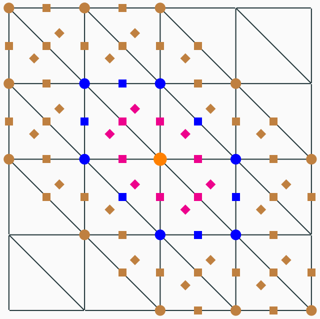
Patch construction: vanka



Looping over vertices

- Select mesh point
- Add points in star
- Add points in closure

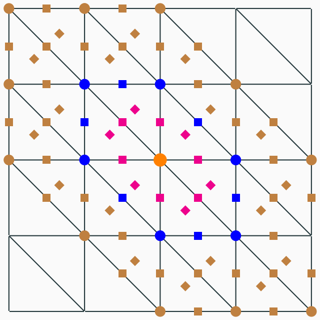
Patch construction: vanka



Looping over vertices

- Select mesh point
- Add points in star
- Add points in closure
- Complete with FEM adjacency

Patch construction: vanka



Looping over vertices

- Select mesh point
- Add points in star
- Add points in closure
- Complete with FEM adjacency

Discretisation-independent

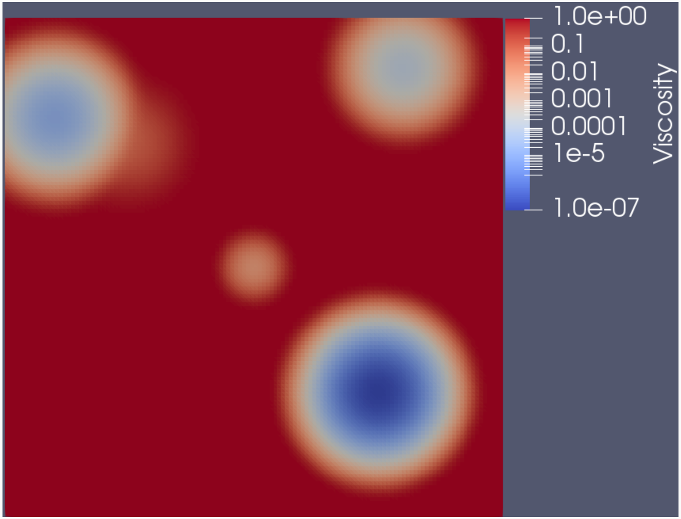
- With points selected, `PetscSection` gives dofs
- Operators "just do assembly" on the patch

Another tool in the box

- Requires slightly more setup than purely algebraic PCs
- Need to feed in operator callback, and some discretisation information
- I do this with the same Python interface as for the block PCs
- Opens up ability for “monolithic” multigrid in PETSc

Code available at `github.com/wence-/ssc`, hopefully in PETSc RSN.

Example: P2-P1 Stokes



Example: P2-P1 Stokes

Monolithic multigrid with Vanka smoother on each level.

```
monolithic_solver_parameters = {
    "mat_type": "matfree",
    "snes_type": "ksponly",
    "ksp_rtol": 1e-8,
    "ksp_type": "fgmres",
    "pc_type": "mg",
    "mg_levels": {"ksp_type": "gmres",
        "ksp_max_it": 5,
        "pc_type": "python",
        "pc_python_type": "ssc.PatchPC",
        "patch_pc_patch_save_operators": True,
        "patch_pc_patch_construction_type": "vanka",
        "patch_pc_patch_partition_of_unity": True,
        "patch_pc_patch_vanka_dim": 0,
        "patch_pc_patch_construction_dim": 0,
        "patch_pc_patch_exclude_subspace": 1,
        "patch_pc_patch_sub_mat_type": "seqaij",
        "patch_sub_ksp_type": "preonly",
        "patch_sub_pc_type": "lu",
        "patch_sub_pc_factor_shift_type": "nonzero"},
    "mg_coarse": {"ksp_type": "preonly",
        "pc_type": "python",
        "pc_python_type": "firedrake.AssembledPC",
        "assembled_pc_type": "svd",}
}
```

```
0 SNES Function norm 51.1133
  Residual norms for  solve.
   0 KSP Residual norm 51.1133
   1 KSP Residual norm 4.24056
   2 KSP Residual norm 1.29486
   3 KSP Residual norm 0.207982
   4 KSP Residual norm 0.1554
   5 KSP Residual norm 0.0422543
   6 KSP Residual norm 0.0278166
   7 KSP Residual norm 0.00664682
   8 KSP Residual norm 0.00307886
   9 KSP Residual norm 0.000731788
  10 KSP Residual norm 0.000238784
  11 KSP Residual norm 3.09127e-05
  12 KSP Residual norm 4.9848e-06
  13 KSP Residual norm 1.07483e-06
  14 KSP Residual norm 1.00638e-07
1 SNES Function norm 1.00638e-07
```

Conclusions

- Composable solvers, using PDE library to easily develop complex block preconditioners.
- Model formulation decoupled from solver configuration.
- Automatically takes advantage of any improvements in both PETSc and Firedrake.
- Same approach works for Schwarz-like methods.

`www.firedrakeproject.org`

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

EPSRC
Engineering and Physical Sciences Research Council

References

Brown, J. et al. (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`.

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]`.