Journal of
open research software

FEniCSx-pctools: Tools for PETSc Block Linear Algebra Preconditioning in FEniCSx

SOFTWARE METAPAPER

]u[ubiquity press

MARTIN ŘEHOŘ
JACK S. HALE

*Author affiliations can be found in the back matter of this article

ABSTRACT

Discretising partial differential equations with the finite element method leads to large linear systems of equations that must be solved. When these systems have a natural block structure due to multiple field variables, using iterative solvers with carefully designed preconditioning strategies that exploit the underlying physical structure becomes necessary for an efficient and scalable solution process. FEniCSx Preconditioning Tools (FEniCSx-pctools) is a software package that eases the specification of PETSc (Portable, Extensible Toolkit for Scientific Computation) block preconditioning strategies on linear systems assembled using the DOLFINx finite element solver of the FEniCS Project. The package automatically attaches all necessary metadata so that preconditioning strategies can be applied via PETSc's standard options database to monolithic and block assembled systems. The documented examples include a simple mixed Poisson system and more complex pressure convection-diffusion approach to preconditioning the Navier–Stokes equations. We show weak parallel scaling on a coupled Navier–Stokes–Fourier system up to 8192 MPI (Message Passing Interface) processes, demonstrating the applicability of the approach to large-scale problems.

FEniCSx-pctools is available under the LGPLv3 or later license and is developed on GitLab https://gitlab.com/rafinex-external-rifle/fenicsx-pctools. The documentation is available at https://rafinex-external-rifle.gitlab.io/fenicsx-pctools/.

CORRESPONDING AUTHOR:
Martin Řehoř
Rafinex S.à r.l., Luxembourg
martin.rehor@rafinex.com

KEYWORDS:
finite elements; iterative solvers; preconditioning; mixed systems; multiphysics simulations

TO CITE THIS ARTICLE:
Řehoř M, Hale JS 2025 FEniCSx-pctools: Tools for PETSc Block Linear Algebra Preconditioning in FEniCSx. *Journal of Open Research Software,* 13: 18. DOI: https://doi.org/10.5334/jors.494

(1) OVERVIEW

INTRODUCTION

Solving linear systems is a core step in the numerical solution of partial differential equations (PDE). To solve large linear systems efficiently on high-performance computers, iterative solvers are essential. However, iterative solvers require effective preconditioning to be viable; without this, convergence may be unacceptably slow, or fail [1]. When modelling complex physical phenomena involving multiple coupled fields, such as incompressible fluid flow, mixed formulations of elasticity, and electromagnetism, the linear systems have a block structure where the different variables have natural couplings. It is now well established that the iterative solution of block linear systems requires block preconditioning strategies that exploit the structure of the coupled problems [7, 16].

The purpose of this software metadata paper is to describe FEniCSx Preconditioning Tools (henceforth FEniCSx-pctools), an add-on for DOLFINx [6] and PETSc (Portable, Extensible Toolkit for Scientific Computation) [5] that eases the specification of complex PETSc-based block-structured preconditioners and linear solvers. FEniCSx-pctools requires both DOLFINx and PETSc/petsc4py, and was produced as a supporting tool in a larger research project focused on the topology optimisation of fluidic devices.

The main contribution of FEniCSx-pctools is to bridge the gap between DOLFINx and PETSc by offering a set of algorithms that can analyse the high-level Unified Form Language (UFL) [3] representation of a block-structured finite element problem and subsequently attach the necessary PETSc metadata to the existing DOLFINx monolithic and block assembled linear algebra objects. With this metadata, advanced block preconditioning strategies can be specified straightforwardly using PETSc's standard options-based configuration system [10].

We include three fully documented demos. The first shows a Schur complement preconditioner for the mixed Poisson problem, based on a design proposed in [24]. The second sets up a Schur complement preconditioner of the velocity-pressure Navier–Stokes equations using the pressure convection-diffusion (PCD) approach proposed in [16]. In the final demo, an elementary system of algebraic equations is used to illustrate the ability to change the solver configuration at runtime, independently of the model formulation.

RELATED WORK

We focus on software that provides a high-level interface to block preconditioning strategies in lower-level sparse linear algebra libraries like Trilinos [25, 4] and PETSc [5].

1. The Firedrake Project [23] builds directly on top of the PETSc DM package [20] allowing for the straightforward specification of block-structured algebraic systems and composable physics-based preconditioners.
2. CBC.Block [22] provides block preconditioning tools within the legacy DOLFIN library [21, 2] using the Trilinos linear algebra backend. A particularly strong aspect of CBC.Block is its domain-specific algebraic language for specifying block linear algebra preconditioners. CBC.Block is one of the core components of HAZniCS [11], a software toolbox for solving interface-coupled multiphysics problems.
3. FENaPack [9] is a preconditioning package for the legacy DOLFIN library using the PETSc linear algebra backend. A particular focus is on implementing the PCD approach for preconditioning the Navier–Stokes equations.
4. PFIBS [13], like FENaPack, is a parallel preconditioning package for the legacy DOLFIN library using the PETSc linear algebra backend. It contains a class BlockProblem that can split a monolithic PETSc matrix into blocks. PFIBS also contains an example of PCD preconditioning of the Navier–Stokes equations.

The first-mentioned finite element solver, Firedrake [23], offers block preconditioning as standard. The remaining packages are extensions to the legacy DOLFIN finite element solver that ease block preconditioning in Trilinos or PETSc. FEniCSx-pctools is similar in scope to these remaining packages, in that it extends the basic capabilities of the new DOLFINx finite element solver [6], which is linear algebra backend agnostic and does not use PETSc's DM functionality like Firedrake, to support PETSc-based block preconditioning. Additionally, like FENaPack and PFIBS, FEniCSx-pctools contains an implementation of PCD preconditioning which is non-trivial to implement for users.

IMPLEMENTATION AND ARCHITECTURE

In this section we briefly outline the structure of FEniCSx-pctools library before illustrating its use via the example of preconditioning a mixed Poisson problem. We assume basic knowledge of the mathematics of the finite element method, the DOLFINx finite element solver and PETSc, see e.g. [15].

Before continuing we briefly explain the three main types of matrix assembly routines that DOLFINx offers in the current development version `0.10.0.dev0` and whether they allow for block preconditioning.

The first type, monolithic, is created when assembling via `assemble_matrix(..., kind=None)` a finite element form defined on a single `dolfinx.fem.FunctionSpace` containing a single `basix.ufl.FiniteElement`, or a single `basix.ufl.MixedElement` containing multiple `basix.ufl.FiniteElement` objects (one for each variable). In the latter case, the degree-of-freedom maps related to each variable are interleaved into a single global degree-of-freedom map, resulting in strong data locality, but the assembled matrix loses the 'natural' block structure and block preconditioning is not available.

The second type, nest, is created when assembling a finite element form defined on a sequence of `dolfinx.fem.FunctionSpace` objects using the `assemble_matrix(..., kind="nest")` call. Each interaction between a pair of variables (including self-interactions) is assembled into its own matrix and the global system is represented as a PETSc MATNEST matrix composed of these submatrices. The advantage of the nest type is that block preconditioning is supported natively, but in the case the user wants to use a direct solver, only MUMPS can be used.

The third type, block, is created when assembling a finite element form defined on a sequence of `dolfinx.fem.FunctionSpace` objects using the `assemble_matrix(..., kind="mpi")` call. In this case, the degree-of-freedom maps for each variable occupy a contiguous block in a global degree-of-freedom map. The forms are assembled into a standard PETSc MATMPIAIJ matrix and the block structure is retained. An advantage of the block type is that any direct solver can be used for solution, not just MUMPS, as in nest. However, as the degree-of-freedom map has lost explicit information about the variables, block preconditioning is not available.

The key contribution of FEniCSx-pctools is to enable block preconditioning for monolithic and block assembly.

FEniCSx-pctools is implemented as a Python package `fenicsx-pctools` with two subpackages named `mat` (matrix, abbreviated as in PETSc to mat) and `pc` (preconditioners, abbreviated as in PETSc to pc). The package is fully documented and is type-hinted/checked.

The `mat` subpackage primarily contains two fundamental factory functions named `create_splittable_matrix_block` and `create_splittable_matrix_monolithic`, which accept a DOLFINx-assembled PETSc `Mat`, of either block or monolithic type, plus the associated UFL form and return a new PETSc Python-context `Mat` that allows for the matrix to be split into submatrices (blocks) corresponding to the original field variables.

The `pc` subpackage primarily contains a class `WrappedPC`, a PETSc Python-context `PC` that allows for preconditioners to work with the PETSc Python-context `Mat` created by the factory functions in the aforementioned `mat` subpackage. The class `WrappedPC` is not used directly; instead, users tell PETSc to use it as a Python-context preconditioner and combine it with a standard PETSc `PC`. More specifically, the Python-context preconditioner sets the standard preconditioner as its own attribute and uses it to interact with the underlying block matrix. This way it can wrap any type of preconditioner (hence the generic name), but it is designed to extend the configurability of the fieldsplit preconditioner. Additionally, `pc` contains implementations of two PCD preconditioning strategies `PCDPC_vX` and `PCDPC_vY`. All of the prior mentioned preconditioner classes inherit from a shared base class `PCBase` which can be used by advanced users to create their own preconditioners.

The diagram in Figure 1 illustrates the role of FEniCSx-pctools in the FEniCS and PETSc ecosystem, and shows how to set up a fieldsplit preconditioner to allow block preconditioning of a block matrix.

We now move onto an example for DOLFINx version `0.10.0.dev0` showing the use of `create_splittable_matrix_block` and `WrappedPC`. A classic example of a block-structured linear system arises from the finite element discretisation of the mixed Poisson problem. Preconditioning this problem is a foundational example and so is widely used to show the basic aspects of preconditioning in software frameworks. In weak form we seek a vector-valued flux $q_h \in Q_h$ and a scalar-valued pressure $p_h \in P_h$, such that

$$\left(q_h, \tilde{q}\right) + \left(p_h, \operatorname{div}\left(\tilde{q}\right)\right) = 0 \qquad \forall \tilde{q} \in Q_h, \tag{1a}$$

$$\left(\operatorname{div}\left(q_h\right), \tilde{p}\right) = \left(-f, \tilde{p}\right) \qquad \forall \tilde{p} \in P_h, \tag{1b}$$

where $(\cdot,\cdot)$ denotes the usual L^2 inner product on the finite element mesh and f a known forcing term. In discrete block form this can be written as a saddle point linear system with unknown vectors of finite element coefficients q and p

$$\begin{bmatrix} A & B^T \\ B & O \end{bmatrix} \begin{bmatrix} q \\ p \end{bmatrix} = \begin{bmatrix} 0 \\ g \end{bmatrix}, \tag{2}$$

where A is a square matrix arising from the bilinear form $(q_h, \tilde{q})$, B and B^T are non-square matrices arising from the bilinear forms $(\operatorname{div}(q_h), \tilde{p})$ and $(p_h, \operatorname{div}(\tilde{q}))$ respectively, O is a square matrix of zeros, 0 is a vector of zeros and g is a vector arising from the linear form $(-f, \tilde{p})$. Finally, we denote the block matrix on the left of eq. (2) K, the block vector of flux and pressure unknowns x, and the block vector on the right b, i.e. $Kx = b$.

The matrix K and the vector b can be assembled in DOLFINx using the standard code shown in Figure 2. At this point DOLFINx has assembled the left-hand side matrix of eq. (2) in two steps:

1. By creating a PETSc matrix based on the array of bilinear forms; this involves the creation of a merged sparsity pattern with the block layout dictated by the arrangement of forms in the array.
2. Assembling the forms into the matrix initialised in step 1, in a blockwise manner; this involves the creation of index sets that can be used to extract and assemble each individual submatrix.

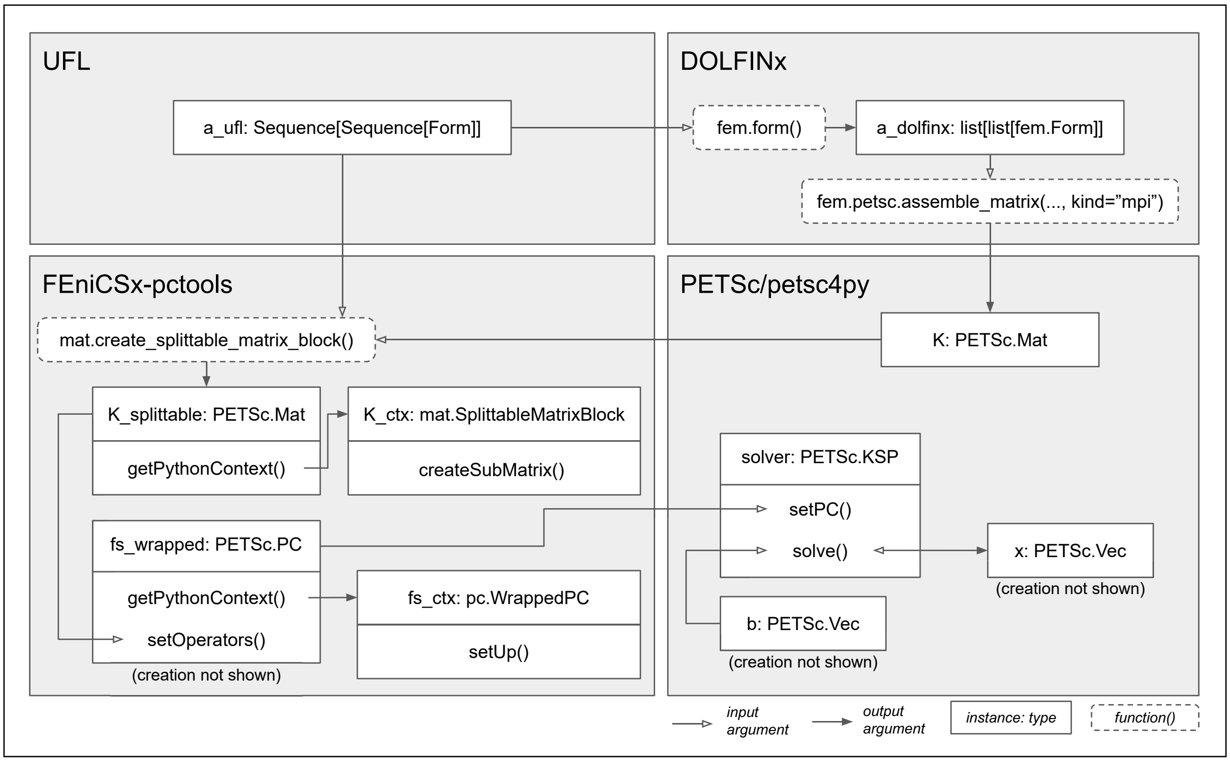

Figure 1 Diagram summarising the workflow to implement a custom fieldsplit preconditioner using the FEniCSx-pctools library. (Top left block, UFL) The symbolic UFL bilinear form defining a block operator is first defined as a 2D list of UFL forms `a_ufl`. (Top right block, DOLFINx) The list of UFL forms is just-in-time compiled to a list of compiled finite element forms `a_dolfinx`, which is then assembled into a block matrix `K` in PETSc's "mpiaij" format (bottom right block, PETSc/petsc4py). We now enter the domain of FEniCSx-pctools, bottom left block. Instead of directly associating the matrix `K` with the linear `solver`, we first pass the original UFL form `a_ufl` and `K` to `create_splittable_matrix_block` to create a splittable matrix `K_splittable`. This is a necessary step to allow block extraction based on arbitrarily combined index sets provided by the fieldsplit preconditioner `fs_wrapped`. The extraction itself is implemented in the `createSubMatrix` method of the splittable matrix' Python-context `K_ctx`, while the index sets are configurable via `setUp` method of the preconditioner's Python-context `fs_ctx`. (Bottom right block, PETSc/petsc4py) The methods in `fs_ctx` and `K_ctx` are transparently executed by the linear `solver` upon calling its `solve` method with the right-hand side vector `b` and a vector `x` to store the solution.

The extraction of a submatrix is the key mechanism exploited by PETSc's fieldsplit-based preconditioners [10]. The splittable block matrix created with the dedicated factory function, see Figure 3, is equipped with metadata including row and column index sets defining the individual blocks. These can be arbitrarily combined (concatenated) and used to define a fieldsplit group to be extracted. The grouping is not limited to neighbouring blocks, which is the case when using PETSc MATNEST matrices directly with the fieldsplit preconditioner.

We now describe and specify an upper-diagonal Schur complement preconditioner using PETSc. Writing the *LDU* decomposition of *K* gives

$$K = LDU = \begin{bmatrix} I & 0 \\ BA^{-1} & I \end{bmatrix} \begin{bmatrix} A & 0 \\ 0 & S \end{bmatrix} \begin{bmatrix} I & A^{-1}B^T \\ 0 & I \end{bmatrix}. \qquad (3)$$

Choosing to use the inverse of the diagonal *D* and upper *U* components of eq. (3) as a preconditioner

$$P_{\text{upper}} = DU = \begin{bmatrix} A & 0 \\ 0 & S \end{bmatrix} \begin{bmatrix} I & A^{-1}B^T \\ 0 & I \end{bmatrix}, \qquad (4)$$

leads to the following *upper* Schur complement *left* preconditioned block system of equations

$$P_{\text{upper}}^{-1} K x = P_{\text{upper}}^{-1} b, \qquad (5)$$

Use of GMRES and upper Schur complement preconditioning can be specified using the PETSc options shown in Figure 4. Note how we use `WrappedPC` as a high-level preconditioner which acts as an embedded layer for reading the precise fieldsplit configuration from the options database and completing the metadata that will be eventually used for the extraction of the submatrices.

In the general case, *S* is a dense matrix that cannot be stored explicitly, let alone inverted. To avoid this, we suppose the existence of a 'good' approximate action for both $A^{-1} \approx \tilde{A}^{-1}$ and $S^{-1} \approx \tilde{S}^{-1}$, i.e. we substitute

$$\tilde{P}_{\text{upper}}^{-1} = \begin{bmatrix} I & -\tilde{A}^{-1}B^T \\ 0 & I \end{bmatrix} \begin{bmatrix} \tilde{A}^{-1} & 0 \\ 0 & \tilde{S}^{-1} \end{bmatrix}, \qquad (6)$$

for P_{upper}^{-1} in eq. (5), where the tilde ($\tilde{\cdot}$) denotes an approximate (inexact) inverse.

```
from mpi4py import MPI
from petsc4py import PETSc

import numpy as np

from basix.ufl import element
from dolfinx import fem, mesh
from dolfinx.fem.petsc import assemble_matrix, assemble_vector
from fenicsx_pctools.mat import create_splittable_matrix_block
from ufl import CellDiameter, FacetNormal, Measure, TestFunction,
    TrialFunction, ZeroBaseForm, avg, div, ds, dS, grad, inner, jump

domain = mesh.create_unit_square(MPI.COMM_WORLD, 1024, 1024, mesh.
    CellType.quadrilateral)

k = 1
Q_el = element("BDMCF", domain.basix_cell(), k)
P_el = element("DG", domain.basix_cell(), k - 1)
Q = fem.functionspace(domain, Q_el)
P = fem.functionspace(domain, P_el)

q = TrialFunction(Q)
q_t = TestFunction(Q)

p = TrialFunction(P)
p_t = TestFunction(P)

f = fem.Function(P)
rng = np.random.default_rng()
f.x.array[:] = rng.uniform(size=f.x.array.shape)

dx = Measure("dx", domain)
a = [[inner(q, q_t) * dx, inner(p, div(q_t)) * dx], [inner(div(q), p_t
    ) * dx, None]]
L = [ZeroBaseForm((q_t,)), -inner(f, p_t) * dx]
a_dolfinx = fem.form(a)
L_dolfinx = fem.form(L)

K = assemble_matrix(a_dolfinx, kind="mpi")
K.assemble()

b = assemble_vector(L_dolfinx, kind="mpi")
b.ghostUpdate(addv=PETSc.InsertMode.INSERT, mode=PETSc.ScatterMode.
    FORWARD)
```

Figure 2 Standard DOLFINx code for assembling the block linear system $Kx = b$ for the mixed Poisson problem. We define a mesh consisting of 1024×1024 quadrilateral cells. For the flux space Q_h we choose Brezzi-Douglas-Marini elements of first-order, and for the pressure space P_h discontinuous Lagrange elements of zeroth-order. The right-hand side forcing term f is drawn from a uniform distribution and then the mixed Poisson variational problem is defined using UFL.

```
K_splittable = create_splittable_matrix_block(K, a)
K_splittable.setOptionsPrefix("mp_")
```

Figure 3 Continuation of Figure 2. The fundamental FEniCSx-pctools factory function `create_splittable_matrix_block` takes the DOLFINx assembled matrix along with the array of bilinear forms that defines it, and returns a PETSc Mat of type "python" with the necessary functionality to apply block preconditioning strategies.

```
solver = PETSc.KSP().create(MPI.COMM_WORLD)
solver.setOptionsPrefix("mp_")
solver.setOperators(K_splittable)

options = PETSc.Options()
options.prefixPush("mp_")
options["ksp_type"] = "gmres"
options["ksp_rtol"] = 1e-8
options["ksp_monitor_true_residual"] = ""
options["pc_type"] = "python"
options["pc_python_type"] = "fenicsx_pctools.pc.WrappedPC"

options.prefixPush("wrapped_")
options["pc_type"] = "fieldsplit"
options["pc_fieldsplit_type"] = "schur"
options["pc_fieldsplit_schur_fact_type"] = "upper"
options["pc_fieldsplit_schur_precondition"] = "user"
options["pc_fieldsplit_0_fields"] = "0"
options["pc_fieldsplit_1_fields"] = "1"
```

Figure 4 Continuation of Figure 3. We first specify that PETSc should use the custom class `fenicsx_pctools.pc.WrappedPC` to wrap the fieldsplit preconditioner, and GMRES as an outer solver for $P_{\text{upper}}^{-1}Kx = P_{\text{upper}}^{-1}b$. At the next level, we ask for upper Schur complement preconditioning with the structure of the preconditioner specified by the user.

```
options.prefixPush("fieldsplit_0_")
options["ksp_type"] = "preonly"
options["pc_type"] = "bjacobi"
options.prefixPop()  # fieldsplit_0_
```

Figure 5 Continuation of Figure 4. We tell PETSc to approximate $\tilde{A}^{-1}$ using one application of block Jacobi preconditioning with the matrix A, the already-assembled upper-left block of K.

To compute $\tilde{P}_{\text{upper}}^{-1}$ in eq. (6) we still must specify the form of both $\tilde{A}^{-1}$ and $\tilde{S}^{-1}$. One reasonable choice is to take $\tilde{A}^{-1}$ as a single application of a block Jacobi preconditioned inverse mass matrix on the finite element flux space Q_h. Note that this mass matrix is in fact the already assembled upper left block A and by default PETSc uses the already assembled operator A as a preconditioner. This can be specified in code as shown in Figure 5.

For $\tilde{S}^{-1}$ we take a single application of algebraic multigrid preconditioned discontinuous Galerkin approximation of the Laplacian on the finite element pressure space P_h. This can be specified as in Figure 6. This choice is justified by the spectral equivalence between the Schur complement and the Laplacian, for further details see e.g. [7]. In the final step, we complete the solver setup and solve the problem using the sequence of commands shown in Figure 7.

We remark that the same configuration of the linear solver can be achieved using the alternative DOLFINx assembly routines that produce the PETSc block-structured matrix of type MATNEST [10], and without the use of the utilities provided by FEniCSx-pctools. The main advantage offered by FEniCSx-pctools is the possibility to change the preconditioner setup at runtime without the need to modify the model specification (typically, the arrangement of finite element function spaces that determines the block layout of the system matrix).

USE IN RESEARCH

In this section we show scaling results demonstrating the relevance of the software to solving real-world problems involving the solution of partial differential equations (PDE) at scale on high-performance computers (HPC).

We perform weak scalability tests up to 8192 MPI (Message Passing Interface) processes on a three-field (temperature, velocity and pressure) problem, describing a steady-state incompressible flow with thermal convection, preconditioned with algebraic multigrid for the temperature block and PCD approach for the velocity-pressure Navier–Stokes blocks [16]. The PDE and problem data (domain, boundary conditions etc.) are exactly the same as those described as the Rayleigh–Bénard problem in [18] and solved with Firedrake [23], so for brevity's sake we do not repeat the details. The solution is described and visualised in Figure 8. Compared with [18] we use a slightly different design for the PCD component of our preconditioner following [8].

```
n = FacetNormal(domain)
alpha = fem.Constant(domain, 4.0)
gamma = fem.Constant(domain, 8.0)
h = CellDiameter(domain)

s = -(
    inner(grad(p), grad(p_t)) * dx
    - inner(avg(grad(p_t)), jump(p, n)) * dS
    - inner(jump(p, n), avg(grad(p_t))) * dS
    + (alpha / avg(h)) * inner(jump(p, n), jump(p_t, n)) * dS
    - inner(grad(p), p_t * n) * ds
    - inner(p * n, grad(p_t)) * ds
    + (gamma / h) * p * p_t * ds
)

S = assemble_matrix(fem.form(s))
S.assemble()


class SchurInv:
    def setUp(self, pc):
        self.ksp = PETSc.KSP().create()
        self.ksp.setOptionsPrefix(pc.getOptionsPrefix() + "SchurInv_")
        self.ksp.setOperators(S)
        self.ksp.setFromOptions()

    def apply(self, pc, x, y):
        self.ksp.solve(x, y)


options.prefixPush("fieldsplit_1_")
options["ksp_type"] = "preonly"
options["pc_type"] = "python"
options["pc_python_type"] = __name__ + ".SchurInv"
options.prefixPush("SchurInv_")
options["ksp_type"] = "preonly"
options["pc_type"] = "hypre"
options.prefixPop()  # SchurInv_
options.prefixPop()  # fieldsplit_1_

options.prefixPop()  # wrapped_
options.prefixPop()  # mp_
```

Figure 6 Continuation of Figure 5. We first setup a class `SchurInv` with a method `apply` that will apply the approximate inverse of the discontinuous Galerkin Laplacian matrix `S` to the vector `x` and place the result in `y`. We then tell PETSc to use this method when it needs the action of $\tilde{S}^{-1}$.

```
solver.setFromOptions()

x = fem.petsc.create_vector(L_dolfinx, kind="mpi")
solver.solve(b, x)
```

Figure 7 Continuation of Figure 6. Finally, we set all of the options on the PETSc objects and solve. This solver setup gives a nearly mesh independent number of GMRES iterations (17) tested up to a mesh size of 1024×1024.

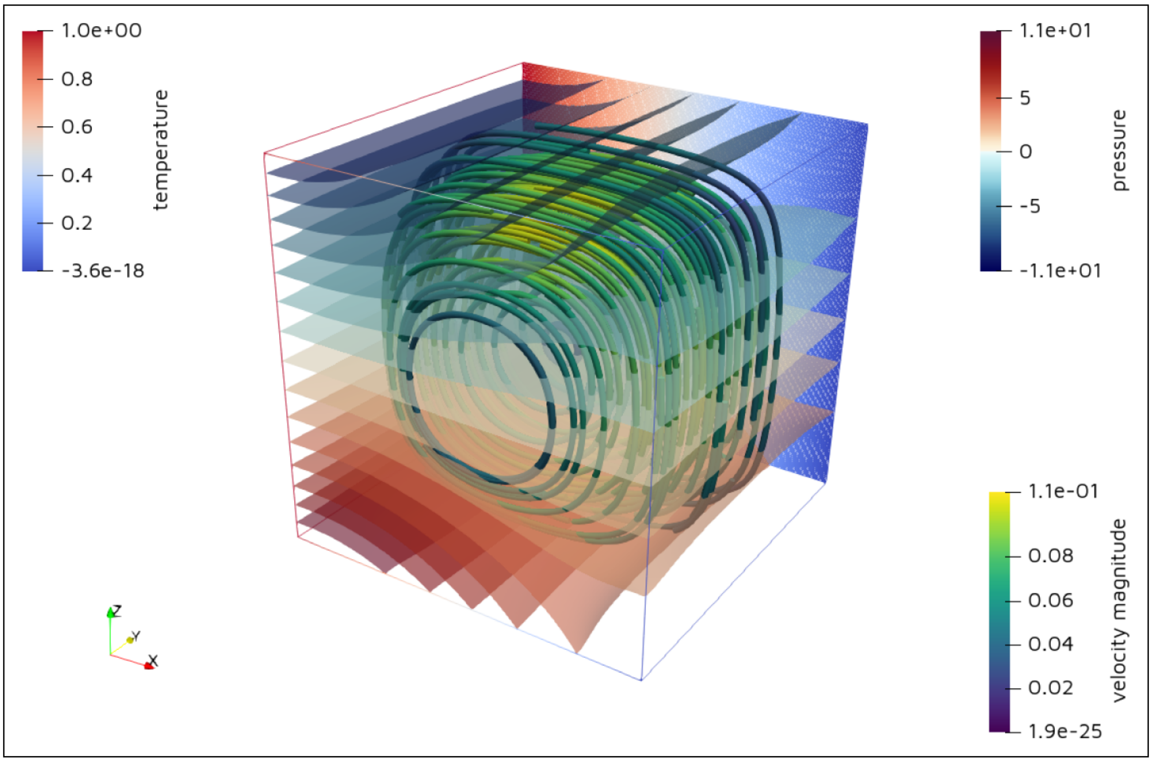

Figure 8 Visualisation of the solution to the Rayleigh-Bénard problem [18]. The fluid velocity is shown as streamlines coloured by the velocity magnitude, the fluid pressure as an isocontour field, and the fluid temperature on the back surface only. Fluid is held at a higher temperature on the left and a lower temperature on the right of the cube. Less dense fluid rises on the left, and denser fluid sinks on the right (the setup takes the gravitational acceleration vector pointing up). At steady-state this forms a distinctive circulation pattern which is clearly shown in the fluid velocity streamlines.

DOF ($\times 10^6$)	MPI PROCESSES	NONLINEAR ITERATIONS	LINEAR ITERATIONS	NAVIER–STOKES ITERATIONS	TEMPERATURE ITERATIONS	WALL TIME FOR SNES SOLVE (s)
6.359	64	2	8	115 (14.4)	49 (6.1)	26.7
12.6	128	2	8	117 (14.6)	49 (6.1)	27.2
25.64	256	2	9	133 (14.8)	56 (6.2)	31.2
101.7	1024	2	7	103 (14.7)	43 (6.1)	25.9
203.5	2048	2	7	102 (14.6)	44 (6.3)	26.1
408.9	4096	2	5	82 (16.4)	31 (6.2)	22.4
816.8	8192	2	6	102 (17)	41 (6.8)	27.1

Table 1 Performance metrics for the Rayleigh-Bénard problem [18] with customised PCD-AMG preconditioning. Weak scaling at 100k DOF per process. Aion Cluster, 50% core utilisation per node. The number in brackets is the average iterations per outer linear solve. Wall time for SNES solve is the time taken to execute `SNES.solve`, which includes PETSc linear algebra operations, DOLFINx assembly operations and the coupling implemented by FEniCSx-pctools.

This results in fewer outer Newton–Raphson iterations and fewer inner Krylov solver iterations than the design proposed in [18].

The weak scalability tests were performed on the University of Luxembourg Aion cluster [26]. The Aion cluster consists of 354 nodes each with two 64 core AMD Epyc ROME 7H12 processors attached to 256 GB RAM. The nodes are connected with an Infiniband HDR network in a 'fat-tree' topology. We built DOLFINx 0.7.0 against PETSc 3.20.0, OpenMPI 4.0.5 and Python 3.8.6 using the GCC 10.2.0 compiler suite with `-march=znver2 -O3` optimisation flags. All experiments were performed at 50% core utilisation per node, i.e. 64 cores per node, as these low-order finite element problems are typically memory-bandwidth constrained. The node allocation was taken in exclusive mode, i.e. with no other jobs running on the same node.

In the weak scaling test we solve the problem on 1 node and double the number of nodes until we reach 128 nodes (8192 MPI processes). We simultaneously increase the problem size so that the number of degrees of freedom (DOF) per MPI rank remains fixed at around 100 000. A breakdown of performance metrics for the weak scaling study is shown in Table 1.

The last column shows the total wall time to execute the `SNES.solve` method, which includes the DOLFINx assembly of the necessary linear operators and their PETSc preconditioning and solution. In summary, the time for solution stays roughly constant, demonstrating the excellent combined weak scaling of DOLFINx, PETSc and the coupling implemented by FEniCSx-pctools. The code to execute these experiments and our raw timing data is available in the supplementary material in the `examples/rayleigh-benard-convection` directory.

QUALITY CONTROL

FEniCSx-pctools contains unit tests that assert that the package functions correctly. In addition, there are three demo problems with checks for correctness. These tests are run as part of a continuous integration pipeline. Users can run these tests themselves by following the instructions in the `README.rst` file. The package is fully documented and Python type-hinted/checked.

(2) AVAILABILITY

OPERATING SYSTEM

DOLFINx, and consequently FEniCSx-pctools, can be built on any modern POSIX-like system, e.g. macOS, Linux, FreeBSD etc. DOLFINx can be built natively on Windows, but at the present time only without PETSc, so FEniCSx-pctools cannot be used on Windows.

PROGRAMMING LANGUAGE

FEniCSx-pctools is written in Python and is compatible with the CPython intepreter version 3.10 and above.

ADDITIONAL SYSTEM REQUIREMENTS

FEniCSx-pctools and its main dependencies DOLFINx and PETSc are designed with scalability on parallel distributed memory systems using MPI. Consequently, they can run on laptops through to large HPC systems.

DEPENDENCIES

FEniCSx-pctools depends on the Python interface to DOLFINx compiled with PETSc/petsc4py support. Because of the many ways to install DOLFINx and PETSc/petsc4py we point users to the upstream DOLFINx instructions. Dependencies and optional dependencies are specified in the standard Python packaging configuration file `pyproject.toml`. We aim to make tagged releases of FEniCSx-pctools that are compatible with DOLFINx releases.

LIST OF CONTRIBUTORS

Not applicable.

SOFTWARE LOCATION

Archive

Name: Figshare [27]
Persistent identifier: https://doi.org/10.6084/m9.figshare.21408294.v6
Licence: LGPLv3 or later
Publisher: Martin Řehoř on behalf of Rafinex S.à r.l.
Version published: Version 6 created from git tag `v0.10.0.dev0-paper`
Date published: 04/09/2025 (ongoing)

Code repository

Name: gitlab.com
Persistent identifier: https://gitlab.com/rafinex-external-rifle/fenicsx-pctools
Licence: LGPLv3 or later
Date published: 27/10/2022 (ongoing)

Emulation environment

Not applicable.

LANGUAGE

English.

(3) REUSE POTENTIAL

The design of parameter and discretisation robust block preconditioning strategies is an active research topic in numerical analysis and computational sciences. We can point to recent developments in designs for the Navier–Stokes equations [17], poroelasticity equations [14], magnetohydrodynamic equations [19] and multiphysics interface problems [12]. Together, FEniCSx-pctools, DOLFINx and PETSc support the straightforward expression and testing of these preconditioning strategies in code, and therefore are useful for researchers who wish to quickly verify the performance of their preconditioning designs. In addition, block preconditioning strategies are an important tool for solving large real-world problems in computational sciences and engineering.

Users can post issues on our GitLab issue tracker at the above repository.

ACKNOWLEDGEMENTS

We would like to thank both reviewers and the editor for their thorough review of this work which led to a significantly improved final article.

The experiments presented in this work were carried out using the HPC facilities of the University of Luxembourg [26] (see https://hpc.uni.lu).

FUNDING STATEMENT

This research was funded in whole, or in part, by the Luxembourg National Research Fund (FNR), grant reference RIFLE/13754363. For the purpose of open access, and in fulfilment of the obligations arising from the grant agreement, the author has applied a Creative Commons Attribution 4.0 International (CC BY 4.0) license to any Author Accepted Manuscript version arising from this submission.

COMPETING INTERESTS

Jack S. Hale has a family member that works at Rafinex S.à r.l. This family member was not involved in this research project. Martin Řehoř declares no competing interests.

AUTHOR AFFILIATIONS

Martin Řehoř orcid.org/0000-0003-4636-6019
Rafinex S.à r.l., Luxembourg; Department of Engineering, Faculty of Science, Technology and Medicine, University of Luxembourg, Luxembourg

Jack S. Hale orcid.org/0000-0001-7216-861X
Department of Engineering, Faculty of Science, Technology and Medicine, University of Luxembourg, Luxembourg

REFERENCES

1. **Allaire G, Kaber SM.** Numerical linear algebra. In: Marsden JE, Sirovich L, Antman SS, editors. *Texts in Applied Mathematics*, vol. 55. New York, NY: Springer; 2008. ISBN: 978-0-387-34159-0 978-0-387-68918-0. DOI: https://doi.org/10.1007/978-0-387-68918-0
2. **Alnæs M,** et al. The FEniCS Project Version 1.5. *Archive of Numerical Software*. Dec. 2015;3(100). ISSN: 2197-8263. DOI: https://doi.org/10.11588/ans. 2015.100.20553
3. **Alnæs MS,** et al. Unified Form Language: A Domain-specific Language for Weak Formulations of Partial Differential Equations. *ACM Trans Math Softw*. Mar. 2014;40(2):9:1–9:37. ISSN: 0098-3500. DOI: https://doi.org/10.1145/2566630
4. **Baker CG, Heroux MA.** Tpetra, and the use of generic programming in scientific computing. *Sci Program*. Apr 2012;20(2):115–128. DOI: https://doi.org/10.1155/2012/693861
5. **Balay S,** et al. *PETSc/TAO Users Manual*. Tech. rep. ANL-21/39 – Revision 3.20. Argonne National Laboratory; 2023. DOI: https://doi.org/10.2172/2205494
6. **Baratta IA,** et al. *DOLFINx: The next generation FEniCS problem solving environment*. Dec. 2023. DOI: https://doi.org/10.5281/zenodo.10447666
7. **Benzi M, Golub GH, Liesen J.** Numerical solution of saddle point problems. *Acta Numerica*. May 2005;14:1–137. Cambridge University Press. ISSN: 1474-0508, 0962-4929. DOI: https://doi.org/10.1017/S0962492904000212
8. **Blechta J.** *Towards efficient numerical computation of flows of non-Newtonian fluids*. PhD thesis. Univerzita Karlova; June 2019. URL: https://dspace.cuni.cz/handle/20.500.11956/108384
9. **Blechta J, Řehoř M.** FENaPack – FEniCS Navier–Stokes preconditioning package. In: Hale JS, editor. *Proceedings of the FEniCS Conference 2017*. Luxembourg: University of Luxembourg; June 2017. pp. 14–15. DOI: https://doi.org/10.6084/m9.figshare.5086369
10. **Brown J,** et al. Composable Linear Solvers for Multiphysics. In: *2012 11th International Symposium on Parallel and Distributed Computing (ISPDC)*. Munich, Germany: IEEE; June 2012. pp. 55–62. ISBN: 978-1-4673-2599-8. URL: http://ieeexplore.ieee.org/document/6341494/
11. **Budiša A,** et al. HAZniCS – Software Components for Multiphysics Problems. *ACM Transactions on Mathematical Software*. Dec. 31, 2023;49(4):1–23. ISSN: 0098-3500, 1557-7295. URL: https://dl.acm.org/doi/10.1145/3625561
12. **Budiša A,** et al. Rational Approximation Preconditioners for Multiphysics Problems. In: Georgiev I, et al. editors. *Numerical Methods and Applications*. *Lecture Notes in Computer Science*. Cham: Springer Nature Switzerland; 2023. 100–113. ISBN: 978-3-031-32412-3. DOI: https://doi.org/10.1007/978-3-031-32412-3_9
13. **Chang J, Protasov K, Allen JM.** *PFIBS: a Parallel FEniCS Implementation of Block Solvers*; 2022. URL: https://github.com/NREL/pfibs
14. **Chen S,** et al. Robust block preconditioners for poroelasticity. In: *Computer Methods in Applied Mechanics and Engineering*. Sept. 2020;369:113229. ISSN: 0045-7825. DOI: https://doi.org/10.1016/j.cma.2020.113229
15. **Dokken JS.** *The FEniCSx tutorial*. URL: https://jsdokken.com/dolfinx-tutorial/
16. **Elman H, Silvester D, Wathen A.** *Finite Elements and Fast Iterative Solvers: with Applications in Incompressible Fluid Dynamics*. Oxford University Press; June 2014. ISBN: 978-0-19-967879-2. DOI: https://doi.org/10.1093/acprof:oso/9780199678792.001.0001
17. **Farrell PE,** et al. A Reynolds-robust preconditioner for the Scott–Vogelius discretization of the stationary incompressible Navier–Stokes equations. In: *The SMAI Journal of Computational Mathematics*. 2021;7:75–96. ISSN: 2426-8399. DOI: https://doi.org/10.5802/smai-jcm.72
18. **Kirby RC, Mitchell L.** Solver Composition Across the PDE/Linear Algebra Barrier. *SIAM Journal on Scientific Computing*. Jan. 2018;40(1):C76–C98. ISSN: 1064-8275, 1095-7197. DOI: https://doi.org/10.1137/17M1133208
19. **Laakmann F, Farrell PE, Mitchell L.** An Augmented Lagrangian Preconditioner for the Magnetohydrodynamics Equations at High Reynolds and Coupling Numbers. *SIAM Journal on Scientific Computing*. Aug. 2022;44(4). B1018–B1044. Society for Industrial and Applied Mathematics. ISSN: 1064-8275. DOI: https://doi.org/10.1137/21M1416539

20. **Lange M,** et al. Efficient Mesh Management in Firedrake Using PETSc DMPlex. *SIAM Journal on Scientific Computing*. Jan. 2016;38(5):S143–S155. Society for Industrial and Applied Mathematics. ISSN: 1064-8275. DOI: https://doi.org/10.1137/15M1026092
21. **Anders L, Wells GN.** DOLFIN: Automated Finite Element Computing. *ACM Trans Math Softw*. Apr. 2010;37(2):20:1–20:28. ISSN: 0098-3500. DOI: https://doi.org/10.1145/1731022.1731030
22. **Kent-Andre M, Joachim BH.** Block preconditioning of sys-tems of PDEs. In: Logg A, Mardal K-A, Wells G, editors. *Automated Solution of Differential Equations by the Finite Element Method*. Lecture Notes in Computational Science and Engineering 84. Berlin Heidelberg: Springer; Jan. 2012. pp. 643–655. ISBN: 978-3-642-23098-1 978-3-642-23099-8. DOI: https://doi.org/10.1007/978-3-642-23099-8_35
23. **Rathgeber F,** et al. Firedrake: Automating the Finite Element Method by Composing Abstractions. *ACM Transactions on Mathematical Software (TOMS)*. Dec. 2016;43(3):24:1–24:27. ISSN: 0098-3500. DOI: https://doi.org/10.1145/2998441
24. **Rusten T, Vassilevski P, Winther R.** Interior penalty preconditioners for mixed finite element approximations of elliptic problems. *Mathematics of Computation*. 1996;65(214):447–466. ISSN: 0025-5718, 1088-6842. DOI: https://doi.org/10.1090/S0025-5718-96-00720-X
25. **The Trilinos Project Team.** *The Trilinos Project Website*. URL: https://trilinos.github.io
26. **Varrette S,** et al. Management of an Academic HPC & Research Computing Facility: The ULHPC Experience 2.0. In: *Proceedings of the 2022 6th High Performance Computing and Cluster Technologies Conference*. HPCCT'22. New York, NY, USA: Association for Computing Machinery; Oct. 2022. pp. 14–24. DOI: https://doi.org/10.1145/3560442.3560445
27. **Řehoř M, Hale JS.** *FEniCSx Preconditioning Tools*. Sept. 2025. DOI: https://doi.org/10.6084/m9.figshare.21408294.v6

TO CITE THIS ARTICLE:
Řehoř M, Hale JS 2025 FEniCSx-pctools: Tools for PETSc Block Linear Algebra Preconditioning in FEniCSx. *Journal of Open Research Software,* 13: 18. DOI: https://doi.org/10.5334/jors.494

Submitted: 24 November 2023 **Accepted:** 05 September 2025 **Published:** 22 September 2025