Archive of Numerical Software
vol. 3, no. 100, pages 9–23
© 2015

The FEniCS Project Version 1.5

Martin S. Alnæs[1], Jan Blechta[2], Johan Hake[3], August Johansson[4], Benjamin Kehlet[5], Anders Logg[6], Chris Richardson[7], Johannes Ring[8], Marie E. Rognes[9], and Garth N. Wells[10]

[1]Simula Research Laboratory, martinal@simula.no
[2]Charles University in Prague, blechta@karlin.mff.cuni.cz
[3]Simula Research Laboratory, hake@simula.no
[4]Simula Research Laboratory, august@simula.no
[5]Simula Research Laboratory, benjamik@simula.no
[6]Chalmers University of Technology and University of Gothenburg, logg@chalmers.se
[7]University of Cambridge, chris@bpi.cam.ac.uk
[8]Simula Research Laboratory, johannr@simula.no
[9]Simula Research Laboratory, meg@simula.no
[10]University of Cambridge, gnw20@cam.ac.uk

Received: May 28th, 2015; **final revision:** December 1st, 2015; **published:** December 7th, 2015.

Abstract: The FEniCS Project is a collaborative project for the development of innovative concepts and tools for automated scientific computing, with a particular focus on the solution of differential equations by finite element methods. The FEniCS Project consists of a collection of interoperable software components, including DOLFIN, FFC, FIAT, Instant, mshr, and UFL. This note describes the new features and changes introduced in the release of FEniCS version 1.5.

1 Overview

FEniCS version 1.5 was released on 12th January 2015. This article provides an overview of the new features of this release and serves as a citable reference for FEniCS version 1.5. The FEniCS software can be downloaded from the FEniCS Project web page at http://fenicsproject.org.

The FEniCS software consists of a collection of interoperable components. These are, in alphabetical order, DOLFIN, FFC, FIAT, Instant, mshr, and UFL. In addition, FEniCS includes the code generation interface component UFC, which is now technically a part of FFC. All FEniCS components, except mshr, are licensed under the GNU LGPL. Due to upstream dependencies, mshr is licensed under the GNU GPL. The components are released simultaneously with the same major and minor version numbers, making it straightforward for users to find compatible versions of the components. The version number of packages that may be unchanged are still incremented to avoid divergence of version numbers.

2 New features

2.1 Parallel computing

The performance of parallel computations with DOLFIN continues to be enhanced. Notable enhancements and new features in the 1.5 release include:

- switch to local (process-wise) degree-of-freedom indexing; for more details see subsection 2.9;
- full support (including from Python) for problems with over 2^{32} dofs via complete support for 64-bit indices;
- substantial improvements in performance scaling and memory use for degree-of-freedom map construction;
- new parallel mesh refinement strategy;
- support for overlapping mesh 'ghost' layers in parallel;
- interior facet integrals are now supported in parallel, which enables discontinuous Galerkin methods in parallel;
- improvements in parallel IO using HDF5.

DOLFIN has been used to solve the Poisson equation with over 12 billion degrees of freedom [Richardson and Wells, 2015] on 24576 cores on a Cray XC30.

Improvements have been made to decrease the load on the filesystem during just-in-time compilation, which can be a bottleneck on large machines. Further effort is required to address the well-known issue of importing Python modules on parallel computers.

A memory corruption issue during forks on OFED-based (InfiniBand) clusters has been tracked down. Users of OFED machines are advised to read the Instant `README` file and export the `INSTANT_SYSTEM_CALL_METHOD` environment variable.

2.2 New mesh refinement strategy

A new refinement algorithm has been adopted for DOLFIN, following the work of Plaza and Carey [2000]. This algorithm uses edge bisection to subdivide the marked triangles of a `Mesh`, be they the cells themselves in two dimensions, or the facets of tetrahedra in three-dimensions. The subdivision is chosen so as to maximize the internal angles of the new triangulation, thus maintaining mesh quality after multiple refinements; see Figure 1. Edges which are marked for refinement propagate between processes in parallel, whilst the refinement operation is local, resulting in good scaling.

2.3 Mesh generation: the new *mshr* component

The mesh generation functionality of FEniCS, except for simple meshes like `UnitSquareMesh` and `UnitCubeMesh`, has been moved to a new FEniCS Component named *mshr*. The motivation for this change is twofold: first, to simplify the list of dependencies and reduce the amount of resources (CPU time and memory usage) for building DOLFIN (by moving the CGAL dependency to mshr); and second, to simplify the addition of new features to the new meshing component. Currently, both CGAL and Tetgen are available as mesh generation backends for mshr.

Much like the old meshing interface in DOLFIN, the new mshr interface allows simple generation of meshes from Constructive Solid Geometry (CSG) descriptions, as demonstrated by the example listed below. The generated mesh is shown in Figure 2.

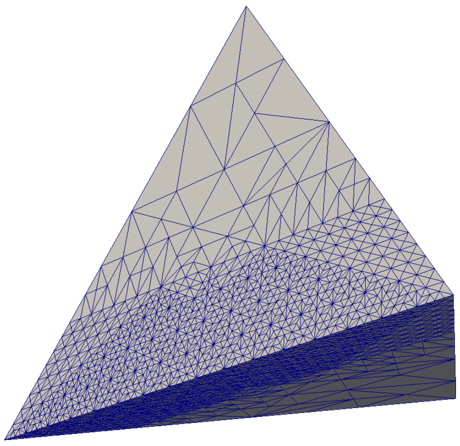
Figure 1: Tetrahedron refined multiple times, whilst preserving mesh quality.

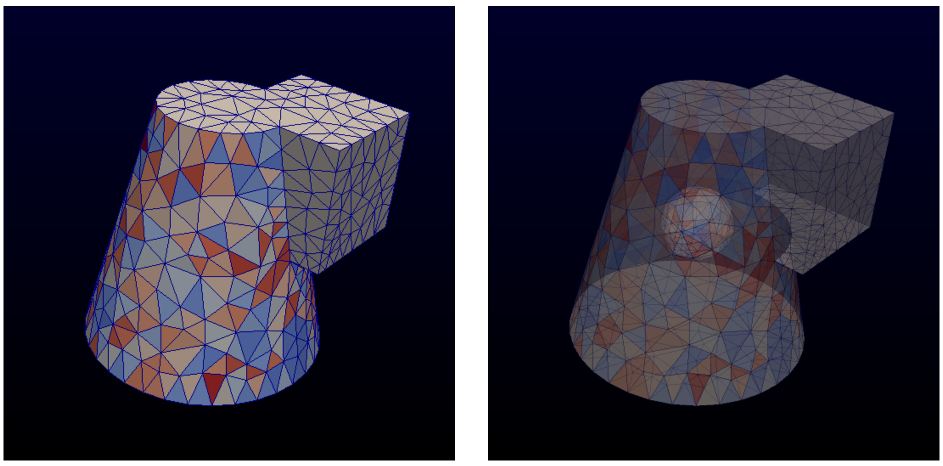
Figure 2: A mesh generated from a CSG description in FEniCS. The left image shows the surface of the generated tetrahedral mesh while the right image shows a volume rendering in which the subtracted sphere in the center is visible.

Python code

```python
from dolfin import *
from mshr import *

# Define geometry
box = Box(Point(0, 0, 0), Point(1, 1, 1))
sphere = Sphere(Point(0, 0, 0), 0.3)
cylinder = Cylinder(Point(0, 0, -1), Point(0, 0, 1), 1.0, 0.5)
geometry = box + cylinder - sphere

# Generate mesh
mesh = generate_mesh(geometry, 16)
```

Through a combination of simple CSG primitives and Boolean operations (and, or, negation), meshes can be created for geometries ranging from simple domains like the canonical L-shaped domain, to relatively complex geometries. Figure 3 shows a pair of meshes generated using mshr.

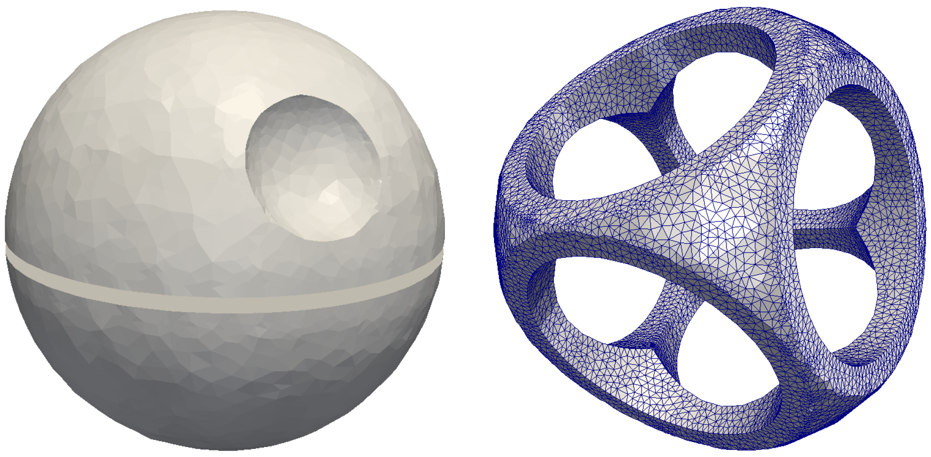

Figure 3: Meshes generated from CSG geometries using the new FEniCS mesh generation component mshr.

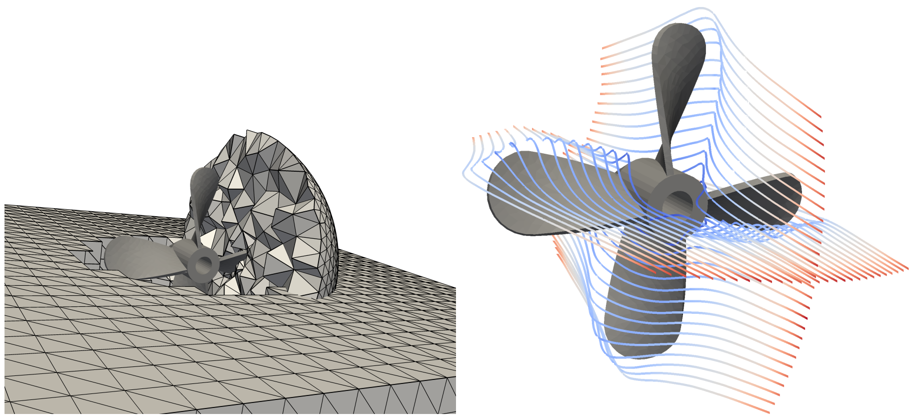

Figure 4: Multimesh solution of the Stokes problem on two overlapping non-matching meshes.

2.4 Multimesh finite element methods

Multimesh finite element methods (CutFEM) are finite element methods posed on two or more possibly non-matching meshes. The formulation of multimesh finite element methods involves the formulation of variational problems on function spaces composed by regular function spaces on the intersecting meshes. Continuity between meshes is imposed using Nitsche's method and stability ensured by adding suitable stabilization terms (ghost penalties) to the variational problem.

FEniCS 1.5 adds support for the formulation of multimesh methods, including automatic and efficient computation of mesh-mesh intersections, quadrature rules for cut cells and interfaces, and expression of integrals that typically appear in the formulation of multimesh/CutFEM methods. Figure 4 shows results for a Taylor–Hood formulation for the Stokes problem on two overlapping and intersecting meshes; a boundary-fitted mesh embedding a propeller-shaped hole, which overlaps a fixed background mesh; for details see Johansson et al. [2015].

The FEniCS interface to multimesh finite element methods currently requires a user to define the integrals in terms of the newly added `custom_integral` interface. This new interface enables code generation for integrals expressed on arbitrary domains, among them cut cells, interfaces, and overlaps, as illustrated by the following UFL code for expression of a multimesh formulation of Poisson's equation:

Python code

```python
# Define custom measures (simplify syntax in future versions)
dc0 = dc(0, metadata={"num_cells": 1})
dc1 = dc(1, metadata={"num_cells": 2})
dc2 = dc(2, metadata={"num_cells": 2})

# Define measures for integration
dx = dx + dc0 # domain integral
di = dc1      # interface integral
do = dc2      # overlap integral

# Parameters
alpha = 4.0
beta = 4.0

# Bilinear form
a = dot(grad(u), grad(v))*dx \
  - dot(avg(grad(u)), jump(v, n))*di \
  - dot(avg(grad(v)), jump(u, n))*di \
  + alpha/h*jump(u)*jump(v)*di \
  + dot(jump(grad(u)), jump(grad(v)))*do
```

On the C++ side, multimesh function spaces must currently be explicitly constructed as demonstrated by the following excerpt from the `multimesh-poisson` demo:

C++ code

```cpp
// Create function spaces
MultiMeshPoisson::FunctionSpace V0(mesh_0);
MultiMeshPoisson::FunctionSpace V1(mesh_1);
MultiMeshPoisson::FunctionSpace V2(mesh_2);

// Build multimesh function space
MultiMeshFunctionSpace V;
V.parameters("multimesh")["quadrature_order"] = 2;
V.add(V0);
V.add(V1);
V.add(V2);
V.build();
```

Future improvements to the multimesh functionality of FEniCS will include the introduction of specific integration measures in UFL, so that the measures don't need to be defined in terms of `custom_measure`, as well as simplified and higher level interfaces for assembly and problem solving in DOLFIN.

2.5 Assembly of point integrals

In addition to the classical cell integrals, exterior facet integrals, and interior facet integrals, FEniCS version 1.5 supports specification of *point integrals*. A point integral represents the integrand evaluated at each vertex, summed over the vertices of the mesh. Starting with FEniCS version 1.5, the DOLFIN assemblers recognize point integrals and assemble contributions from such integrals. We remark that for each vertex, the integrand will be restricted to the arbitrarily chosen first cell associated with this vertex prior to evaluation. This operation is therefore only consistently defined for functions and integrands that are continuous at the vertices.

2.6 Point integral solvers and Rush–Larsen schemes

The FEniCS `PointIntegralSolvers` are special-purpose solvers dedicated to solving multistage integration schemes defined for each vertex of a mesh. Such problems typically occur in connection with spatially varying systems of ODEs, for instance in electrophysiology. In FEniCS version 1.5, we have extended the range of available schemes to also include the Rush–Larsen

[Rush and Larsen, 1978] and the generalized Rush–Larsen [Sundnes et al., 2009] schemes for solving systems of nonlinear ODEs.

2.7 Redesign of core symbolic representation and algorithms

The domain-specific language UFL represents equations as symbolic expression trees, where each node represents some value or mathematical operation. For nonlinear equations with large expression trees, the performance and memory overhead of symbolic computations and form compiler algorithms can become significant, especially when running in parallel, where this overhead is a limiting factor for scalability according to Amdahl's law. This becomes even more important when FEniCS is combined with high-level algorithms, such as the dolfin-adjoint package, that rely extensively on symbolic algorithms from UFL. To reduce this overhead, the core representation of symbolic expressions has been redesigned (see `ufl.core`), followed by a rewrite of the core framework for developing additional symbolic algorithms (see `ufl.corealg`). Each expression tree node type now has the following attributes:

- **`ufl_operands`** Tuple of child nodes in expression tree. Equivalent to deprecated `operands()`.
- **`ufl_shape`** Tuple of value shape dimensions. Equivalent to deprecated `shape()`.
- **`ufl_free_indices`** Tuple of free index integer ids. Replaces deprecated `free_indices()`.
- **`ufl_free_index_dimensions`** Tuple of dimensions associated with corresponding free index ids in `ufl_free_indices`. Replaces deprecated `free_indices()`.

The deprecated functions mentioned above are still available as wrappers around the new attributes. Type traits of symbolic expression classes have also been made available through an efficient property-based API using the naming scheme `ExprType._ufl_*_` (see `ufl.core.expr`). This is mainly intended for internal usage but also for developers building on the symbolic capabilities of FEniCS.

A major guiding principle in the redesign of core algorithms in UFL was to replace all uses of expensive recursion in Python with loops. Algorithms for expression traversal (see module `ufl.corealg.traversal`) have all been reimplemented without recursion, and further fine-tuned for specific iteration scenarios. The former `Transformer` base class for visitor-like algorithms has been deprecated in favor of an implementation that avoids recursion. The new function `map_expr_dag(function, expression)` applies the given function to transform each expression tree node in a non-recursive post-order traversal. The function arguments include the node as well as the previously transformed node operands. This algorithm is further optimized to reduce the number of function calls and avoid duplicated expression tree nodes by (optionally) caching intermediate results in a hashmap with the original expression nodes as keys.

Previous implementations of the `Transformer` class can easily be rewritten as implementations of `MultiFunction`, which implements the same dynamic type dispatch mechanism, and passed as the function argument to `map_expr_dag`. Key algorithms such as automatic differentiation (`apply_derivatives`) and propagation of restrictions (`apply_restrictions`) have been reimplemented in this optimized algorithm framework, and may serve as models. In addition, form signatures are now computed without first applying automatic differentiation, which reduces the symbolic overhead significantly when the signature is found in the form compiler cache. Together these improvements have reduced the symbolic overhead in FEniCS 1.5 by 20–60%. To give a few examples: the initial just-in-time (JIT) compilation of a Poisson equation form was reduced from 5s to 4s including C++ compilation, while for a complicated cardiac hyperelasticity model the JIT time was reduced from 103s to 48s. For the latter model, subsequent program runs now avoid a 2s delay in the signature computation used for disk cache lookup.

$\mathcal{P}_r^-\Lambda^k(\Delta_n)$	$k=0$	$k=1$	$k=2$	$k=3$
$n=1$	P	DP		
$n=2$	P	RT[E,F]	DP	
$n=3$	P	N1E	N1F	DP

$\mathcal{P}_r\Lambda^k(\Delta_n)$	$k=0$	$k=1$	$k=2$	$k=3$
$n=1$	P	DP		
$n=2$	P	BDM[E,F]	DP	
$n=3$	P	N2E	N2F	DP

$\mathcal{Q}_r^-\Lambda^k(\square_n)$	$k=0$	$k=1$	$k=2$	$k=3$
$n=1$	Q	DQ		
$n=2$	Q	RTC[E,F]	DQ	
$n=3$	Q	NCE	NCF	DQ

$\mathcal{S}_r\Lambda^k(\square_n)$	$k=0$	$k=1$	$k=2$	$k=3$
$n=1$	S	DPC		
$n=2$	S	BDMC[E,F]	DPC	
$n=3$	S	AAE	AAF	DPC

Table 1: New notation for finite elements adopted from the Periodic Table of the Finite Elements.

2.8 Notation from Periodic Table of the Finite Elements

FEniCS now supports the notation for finite elements as set out in the Periodic Table of the Finite Elements [Arnold and Logg, 2014] based on the finite element exterior calculus, see Arnold et al. [2006]. The notation introduces new aliases for the existing elements in FEniCS and introduces new placeholder names for the not yet implemented elements on quadrilaterals and hexahedra. Earlier FEniCS notation ("Lagrange", "CG", etc.) is still supported but may be phased out in future versions. Table 1 summarizes the notation. In this table, n is the topological dimension of the finite element domain (1D, 2D, 3D), r is the polynomial degree and k is the form degree; $k = 0$ means scalars, $k = 1$ means 1-forms (oriented line segments), $k = 2$ means 2-forms (oriented surfaces) and $k = 3$ means 3-forms (oriented volumes).

Figure 5 shows a miniature (but high resolution) version of the periodic table. The full version can be downloaded from `http://femtable.org`.

As an example, the following code shows how to instantiate a pair of cubic Nédélec face elements of the first and second kinds on a tetrahedron:

Python code
```python
element_1 = FiniteElement("N1F", "tetrahedron", 3)
element_2 = FiniteElement("N2F", "tetrahedron", 3)
```

2.9 Changes in degree-of-freedom map construction

A number of performance improvements have been made in the degree-of-freedom (DOF) map construction, and changes to how to DOF maps are order have been made. Specifically:

- DOLFIN now uses local (process-wise) numbering of degrees of freedom, which has reduced the memory use of degree-of-freedom maps. This change also permits some new developments in interfacing to linear-algebra backends.
- The degree-of-freedom map construction code has been re-written, reducing the time for construction and substantially improving parallel scaling of construction.
- Node-wise block structure in DOF maps, e.g. a fixed number of DOFs at each node, is now automatically detected. This provides better data locality and faster graph-based DOF map reordering.

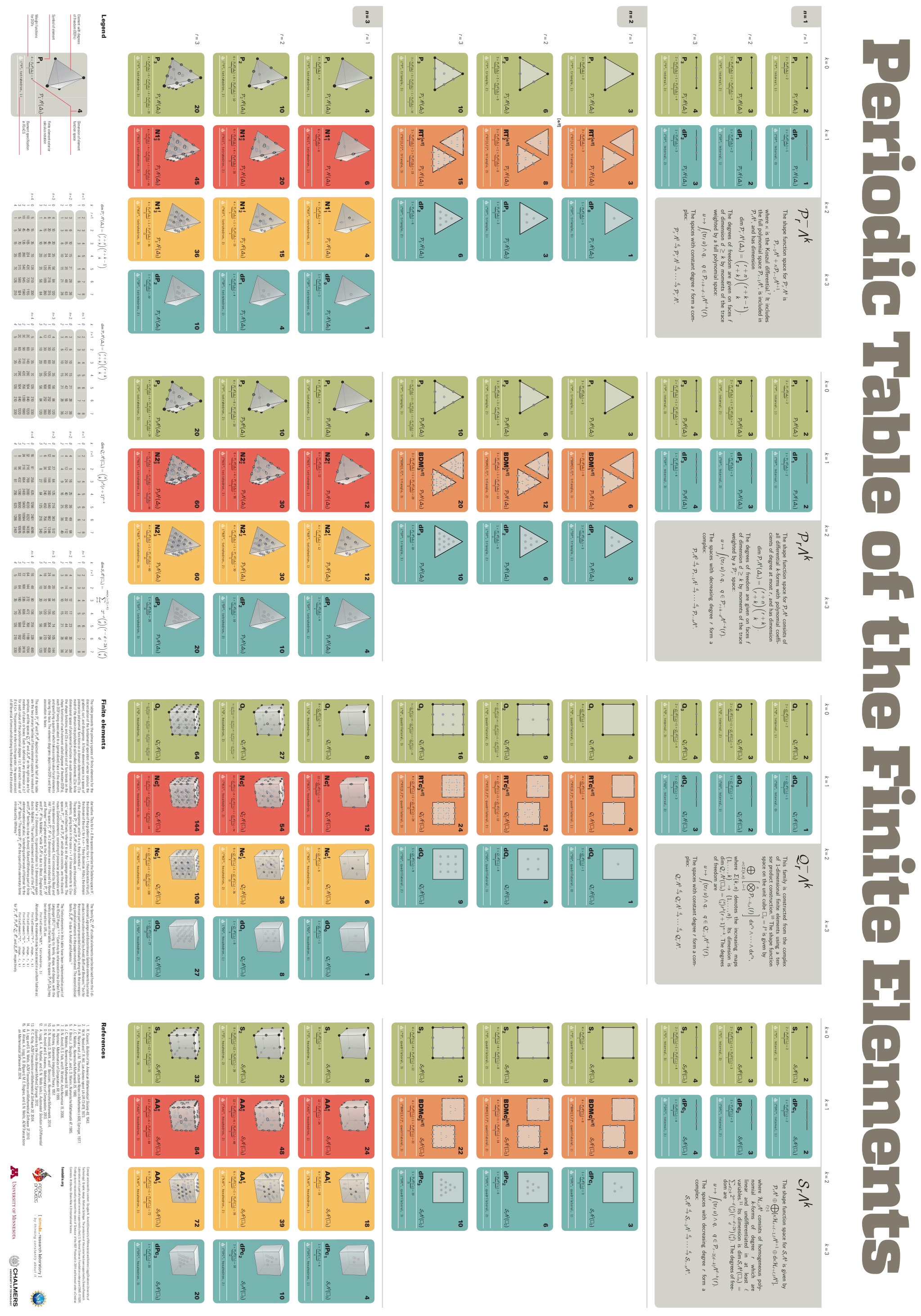

Figure 5: Periodic Table of the Finite Elements.

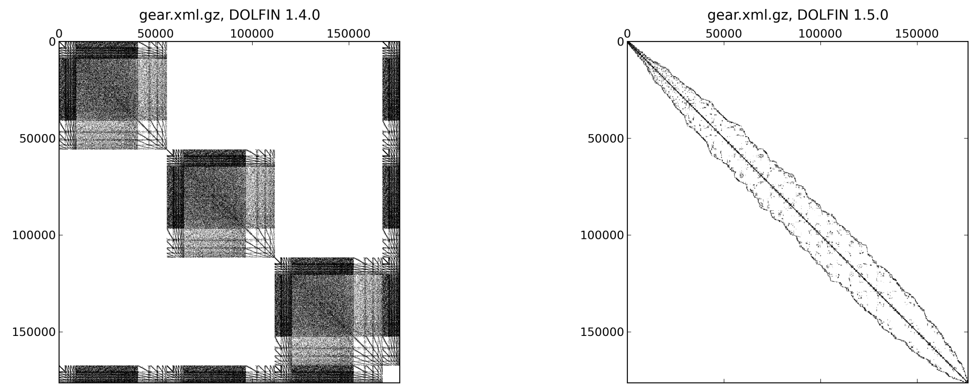

Figure 6: Comparison of sparsity patterns of the Taylor–Hood Stokes operator on the DOLFIN `gear` mesh.

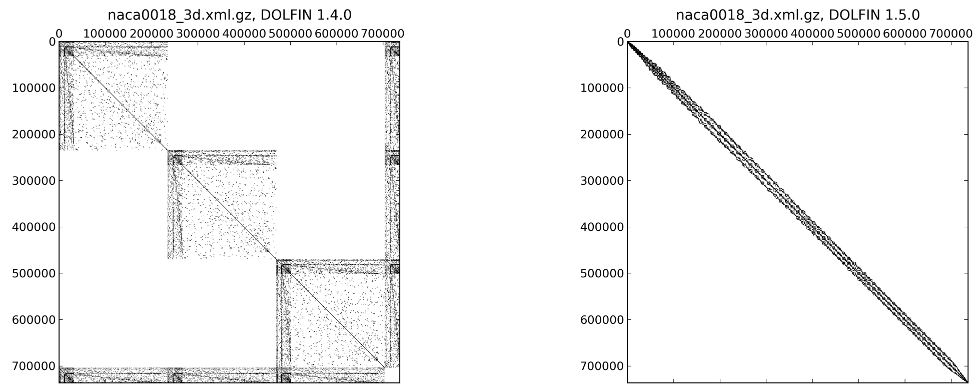

Figure 7: Comparison of sparsity patterns of the Taylor–Hood Stokes operator on a `NACA 0018 airfoil` mesh.

Improvements in sparsity patterns that follow from improved DOF map ordering are shown in Figures 6 and 7. The sparsity patterns produced by DOLFIN 1.5.0 are better clustered around the diagonal. When DOLFIN is configured with SCOTCH, the SCOTCH implementation of Gibbs-Poole–Stockmeyer reordering is used.

The code for reproducing the sparsity patterns in Figures 6 and 7 and patterns on other meshes can be found at `https://www.repository.cam.ac.uk/handle/1810/252670`. The meshes are shown at `http://fenicsproject.org/download/data.html#meshes`.

2.10 Linear algebra interface

FEniCS uses the PETSc and SLEPc libraries as backends providing parallel infrastructure for linear algebra, non-linear solvers, and eigensolvers. The wrapper layer to these packages has been improved, in particular, as follows:

- `PETScTAOSolver`, the DOLFIN interface to the PETSc TAO framework, which provides solvers for non-linear optimization problems, has been introduced;
- `LinearOperator`, the DOLFIN interface to matrix-free operators, can now be unwrapped into a respective PETSc or petsc4py object;

- SLEPcEigenSolver, the DOLFIN interface to eigensolvers, can now be unwrapped into a respective SLEPc or slepc4py object.

The latter two extend the feature already available for other backend wrappers, e.g., PETScMatrix, PETScKrylovSolver, etc. This feature allows for finer, but still convenient, control over a solution process and other details, in such cases when the coverage of backend features or options in DOLFIN is not sufficiently detailed.

In this release, support for the Trilinos Epetra backend has been removed. Support for the newer Trilinos Tpetra backend will be added to a future release.

2.11 Use of C++ 11

The FEniCS team has started using selected C++11 features, allowing removal of the dependency on Boost shared pointers and some simplifications of the C++ source code. The set of features is limited for the time being, to remain compatible with some common compilers. GCC 4.6.3 is known to be compatible.

For users of the DOLFIN C++ interface, the use of C++11 can lead to less verbose code by using the key word auto, initializer lists, and range-based loops, for instance.

2.12 Python versions

FEniCS now requires Python 2.7, and has experimental support for Python 3.x. As part of supporting Python 3, FIAT no longer depends on Scientific Python. In place of Scientific Python, which does not support Python 3, FIAT now depends on SymPy.

2.13 Packaging and installation

As before, FEniCS is available as a binary package as part of the standard Debian and Ubuntu repositories. For users who wish to access the latest FEniCS versions before these have propagated downstream, a personal package archive (PPA) is provided (ppa:fenics-packages/fenics). A binary package is also available for Mac OSX.

With the release of version 1.5, FEniCS has retired the installer Dorsal in favor of the new tool fenics-install.sh. This script relies on HashDist (https://hashdist.github.io/) for installation of FEniCS and its dependencies. Using this script, users can easily build FEniCS with a one-line command:

```
curl -s http://fenicsproject.org/fenics-install.sh | bash
```

FEniCS developers (who need to modify the source code) are encouraged to use the two related scripts fenics-install-all.sh and fenics-install-component.sh as standard tools in their normal workflow. These scripts are part of the FEniCS Developer Tools repository on BitBucket.

In addition, FEniCS can now also be installed via a community-supported Conda recipe, see https://github.com/Juanlu001/fenics-recipes/releases/tag/v1.5.0 as well as via a custom virtualization image provided on the FEniCS web page.

3 New demo programs

Some new example programs have been added to demonstrate the use of new user-level features. The new demo programs are:

- Smoothed aggregation algebraic multigrid for three-dimensional elasticity. The demo illustrates the construction and specification of the near-nullspace which is required by the multigrid preconditioner. It also demonstrates how to construct a scalable solver for elasticity problems.
- Two new demos illustrating the implementation of multimesh-methods: `multimesh-poisson` and `multimesh-stokes`.

4 Interface changes

The following interface changes were made in the release of FEniCS version 1.5:

- The classes `FacetArea`, `FacetNormal`, `CellSize`, `CellVolume`, `SpatialCoordinate`, `Circumradius`, `MinFacetEdgeLength`, and `MaxFacetEdgeLength` now require a `Mesh` argument instead of a `Cell` argument.
- The signature for the `assemble` function has been simplified to not require/accept the arguments `coefficients`, `cells`, and `common_cell`.
- Differentiation of expressions with respect to a `Coefficient` or `Function` using `diff` does not require the function to be wrapped in a `variable` construct.
- Quadrature degree and rule name can be specified as keywords to form integration measures, i.e. `f*dx(degree=3, rule="canonical")`. The syntax `Measure.__getitem__(mesh_function).__call__(subdomain_id)`, e.g., `dx[mesh_function](42)`, has been deprecated in favor of keyword arguments, for example: `dx(subdomain_data=mesh_function, subdomain_id=42)`.
- `GenericTensors` cannot be reinitialized anymore, hence the `reset_sparsity` parameter has been removed from all assemble functions.
- Deprecated `MPI::process_number` and `MPI::num_processes` were removed in favor of `MPI::rank` and `MPI::size` respectively.
- `GenericVector.__setitem__`, e.g., `vec[indices] = values`, now takes local `indices` which can be list of ints, NumPy array of ints, int, or full slice `:`. Type of `values` can be scalar, NumPy array, or `GenericVector` where applicable. Operation is collective and finalizes the vector automatically.

5 Dependencies

FEniCS, mostly DOLFIN, depends on a number of external packages. Here we list and describe these external packages, along with required version numbers (to the best of our knowledge).

Necessary dependencies:

- C++11 compiler (GCC $\geq$ 4.6 and Clang 3.6 are known to work), see section 2.11 for details
- Boost $\geq$ 1.48; provides some low-level data structures and algorithms, DOF reordering and mesh coloring [Schling, 2011]
- CMake $\geq$ 2.8; used by DOLFIN build system and optionally by Python JIT chain

- Eigen ≥ 3.0; provides data structures and operations used by adaptive solvers [Guennebaud et al., 2015]
- Python 2.7 or ≥ 3.2; six (Python 2 and 3 compatibility tool); needed by FIAT, UFL, FFC, and Instant; optionally needed for Python interface to DOLFIN and mshr
- NumPy; Python package providing data structures and operations on N-dimensional arrays [van der Walt et al., 2011]
- SymPy; Python symbolic maths library, used by FIAT [SymPy Development Team, 2014]

Optional dependencies:

- OpenMP; supports parallel computing on shared memory systems
- MPI; supports parallel computing on distributed memory systems
- PETSc ≥ 3.3, < 3.6; serves as parallel (linear) algebra backend; provides data structures and routines for handling parallel vectors, matrices, Krylov solvers, sparse direct solvers, various preconditioners and non-linear solvers [Balay et al., 2014b,a, 1997]
- petsc4py ≥ 3.3, < 3.6; provides Python bindings to PETSc [Dalcin et al., 2011]
- SLEPc ≥ 3.3, < 3.6; provides parallel eigen-problem solvers [Hernandez et al., 2005]
- slepc4py $\geq 3.5.1$, < 3.6; provides Python bindings to SLEPc [Dalcin et al., 2011]
- UMFPACK; provides sequential sparse LU decomposition using multifrontal method [Davis, 2004]
- CHOLMOD; provides sequential sparse Cholesky decomposition using supernodal method [Chen et al., 2008]
- PaStiX $\geq 5.2.1$; provides parallel (both distributed and threaded) sparse LU and Cholesky decomposition [Hénon et al., 2002]
- Trilinos $\geq 11.0.0$; provides mesh partitioning and coloring [Heroux et al., 2005]
- SCOTCH; provides mesh partitioning and DOF reordering [Chevalier and Pellegrini, 2006]
- ParMETIS $\geq 4.0.2$; provides mesh partitioning [Karypis and Kumar, 1998]
- SWIG ≥ 2.0 (with Python 2) or $\geq 3.0.3$ (with Python 3); tool needed for generating Python interface to DOLFIN and mshr
- flufl.lock; implements file locking (used by Instant) on NFS file system
- HDF5; provides parallel, scalable IO backend; needs to be built with parallel support [The HDF Group, 2015]
- zlib; reading compressed XML files and compressed VTK output
- VTK ≥ 5.2; provides interactive plotting in DOLFIN
- CGAL 4.5.1, TetGen 1.5.0; mshr backends, built automatically by mshr build system [The CGAL Project, 2015, Si, 2015]
- GMP, MPFR; arbitrary-precision and multiple-precision arithmetic libraries, required by mshr [Granlund and the GMP development team, 2014, Fousse et al., 2007]
- Scipy; needed by UFL for evaluation of error and Bessel functions

- pytest; Python testing tool, needed for running unit tests
- Sphinx ≥ 1.1.0; enables building documentation
- Qt4; provides API for writing graphical applications
- Soya; Python 3D engine, used for plotting finite elements

6 How to cite FEniCS 1.5

Users of FEniCS version 1.5 are encouraged to cite this article, in addition to one or more of the publications listed on the FEniCS Project web site:

http://fenicsproject.org/citing/

7 Acknowledgments

Although this article lists as authors only those developers who have made significant contributions specifically to the release of FEniCS 1.5, we gratefully acknowledge the contributions by many people to the FEniCS Project, including Martin S. Alnæs, Jan Blechta, Juan Luis Cano Rodríguez Patrick Farrell, Johan Hake, Johan Hoffman, Johan Jansson, Niclas Jansson, August Johansson, Claes Johnson, Benjamin Kehlet, Robert C. Kirby, Matthew Knepley, Miroslav Kuchta, Hans Petter Langtangen, Anders Logg, Kent-Andre Mardal, Andre Massing, Mikael Mortensen, Harish Narayanan, Chris Richardson, Johannes Ring, Marie E. Rognes, L. Ridgway Scott, Ola Skavhaug, Andy Terrel, Garth N. Wells, and Kristian Ølgaard.

The FEniCS Project is supported by The Research Council of Norway through a Centre of Excellence grant to the Center for Biomedical Computing at Simula Research Laboratory, project number 179578. Anders Logg acknowledges support by the Swedish Research Council Grant No. 2014–6093. Jan Blechta acknowledges the support of the project LL1202 in the programme ERC-CZ funded by the Ministry of Education, Youth and Sports of the Czech Republic.

References

D. N. Arnold and A. Logg. Periodic table of the finite elements. *SIAM News*, 2014.

D. N. Arnold, R. S. Falk, and R. Winther. Finite element exterior calculus, homological techniques, and applications. *Acta numerica*, 15:1–155, 2006. URL http://dx.doi.org/10.1017/S0962492906210018.

S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith. Efficient management of parallelism in object oriented numerical software libraries. In E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhäuser Press, 1997. URL http://dx.doi.org/10.1007/978-1-4612-1986-6_8.

S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, K. Rupp, B. F. Smith, and H. Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 3.5, Argonne National Laboratory, 2014a. URL http://www.mcs.anl.gov/petsc.

S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, K. Rupp, B. F. Smith, and H. Zhang. PETSc Web page, 2014b. URL http://www.mcs.anl.gov/petsc.

Y. Chen, T. A. Davis, W. W. Hager, and S. Rajamanickam. Algorithm 887: CHOLMOD, supernodal sparse Cholesky factorization and update/downdate. *ACM Trans. Math. Softw.*, 35(3):22:1–22:14, Oct. 2008. ISSN 0098-3500. URL http://doi.acm.org/10.1145/1391989.1391995.

C. Chevalier and F. Pellegrini. PT-Scotch: A tool for efficient parallel graph ordering. In *4th International Workshop on Parallel Matrix Algorithms and Applications (PMAA'06), IRISA, Rennes, France*, Sept. 2006.

L. D. Dalcin, R. R. Paz, P. A. Kler, and A. Cosimo. Parallel distributed computing using Python. *Advances in Water Resources*, 34(9):1124 – 1139, 2011. ISSN 0309-1708. URL `http://dx.doi.org/10.1016/j.advwatres.2011.04.013`. New Computational Methods and Software Tools.

T. A. Davis. Algorithm 832: UMFPACK v4.3—an unsymmetric-pattern multifrontal method. *ACM Trans. Math. Softw.*, 30(2):196–199, June 2004. ISSN 0098-3500. URL `http://doi.acm.org/10.1145/992200.992206`.

L. Fousse, G. Hanrot, V. Lefèvre, P. Pélissier, and P. Zimmermann. MPFR: A multiple-precision binary floating-point library with correct rounding. *ACM Transactions on Mathematical Software*, 33(2):13:1–13:15, June 2007. URL `http://doi.acm.org/10.1145/1236463.1236468`.

T. Granlund and the GMP development team. *GNU MP: The GNU Multiple Precision Arithmetic Library*, 6.0.0 edition, 2014. URL `http://gmplib.org/`.

G. Guennebaud, B. Jacob, et al. Eigen v3, 2015. URL `http://eigen.tuxfamily.org`.

P. Hénon, P. Ramet, and J. Roman. PaStiX: A High-Performance Parallel Direct Solver for Sparse Symmetric Definite Systems. *Parallel Computing*, 28(2):301–321, Jan. 2002. URL `http://dx.doi.org/10.1016/S0167-8191(01)00141-7`.

V. Hernandez, J. E. Roman, and V. Vidal. SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems. *ACM Trans. Math. Software*, 31(3):351–362, 2005. URL `http://dx.doi.org/10.1145/1089014.1089019`.

M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, and K. S. Stanley. An overview of the Trilinos project. *ACM Trans. Math. Softw.*, 31(3):397–423, 2005. ISSN 0098-3500. URL `http://doi.acm.org/10.1145/1089014.1089021`.

A. Johansson, M. G. Larson, and A. Logg. High order cut finite element methods for the stokes problem. *Advanced Modeling and Simulation in Engineering Sciences*, 2(24), 2015. doi: 10.1186/s40323-015-0043-7.

G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20(1):359–392, Dec. 1998. ISSN 1064-8275. URL `http://dx.doi.org/10.1137/S1064827595287997`.

A. Plaza and G. Carey. Local refinement of simplicial grids based on the skeleton. *Applied Numerical Mathematics*, 32:195–218, 2000. URL `http://dx.doi.org/10.1016/S0168-9274(99)00022-7`.

C. N. Richardson and G. N. Wells. Parallel scaling of DOLFIN on ARCHER. *figshare.com*, `http://figshare.com/articles/Parallel_scaling_of_DOLFIN_on_ARCHER/1304537`, 2015. URL `http://dx.doi.org/10.6084/m9.figshare.1304537`.

S. Rush and H. Larsen. A practical algorithm for solving dynamic membrane equations. *IEEE Trans Biomed Eng*, 25(4):389–392, Jul 1978. URL `http://dx.doi.org/10.1109/TBME.1978.326270`.

B. Schling. *The Boost C++ Libraries*. XML Press, 2011. ISBN 0982219199, 9780982219195.

H. Si. TetGen, a Delaunay-based quality tetrahedral mesh generator. *ACM Trans. Math. Softw.*, 41(2):11:1–11:36, Feb. 2015. ISSN 0098-3500. URL `http://doi.acm.org/10.1145/2629697`.

J. Sundnes, R. Artebrant, O. Skavhaug, and A. Tveito. A second-order algorithm for solving dynamic cell membrane equations. *IEEE Trans Biomed Eng*, 56(10):2546–2548, Oct 2009. URL http://dx.doi.org/10.1109/TBME.2009.2014739.

SymPy Development Team. *SymPy: Python library for symbolic mathematics*, 2014. URL http://www.sympy.org.

The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 4.6 edition, 2015. URL http://doc.cgal.org/4.6/Manual/packages.html.

The HDF Group. Hierarchical data format, version 5, 2015. URL http://www.hdfgroup.org/HDF5/.

S. van der Walt, S. Colbert, and G. Varoquaux. The NumPy Array: A structure for efficient numerical computation. *Computing in Science Engineering*, 13(2):22–30, March 2011. ISSN 1521-9615. URL http://dx.doi.org/10.1109/MCSE.2011.37.