The deal.II Library, Version 8.3

Wolfgang Bangerth[1], Timo Heister[2], Luca Heltai[3], Guido Kanschat[4], Martin Kronbichler[5], Matthias Maier[6], and Bruno Turcksin[7]

[1]Department of Mathematics, Texas A&M University, College Station, TX 77843, USA, bangerth@math.tamu.edu
[2]Mathematical Sciences, O-110 Martin Hall. Clemson University. Clemson, SC 29634, USA, heister@clemson.edu
[3]SISSA - International School for Advanced Studies, Via Bonomea 265, 34136 Trieste, Italy, luca.heltai@sissa.it
[4]Interdisciplinary Center for Scientific Computing (IWR), Universität Heidelberg, Im Neuenheimer Feld 368, 69120 Heidelberg, Germany, kanschat@uni-heidelberg.de
[5]Institute for Computational Mechanics, Technische Universität München, Boltzmannstr. 15, 85748 Garching b. München, Germany, kronbichler@lnm.mw.tum.de
[6]School of Mathematics, University of Minnesota, 127 Vincent Hall, 206 Church Street SE, Minneapolis, MN 55455, USA, msmaier@umn.edu
[7]Department of Mathematics, Texas A&M University, College Station, TX 77843, USA, turcksin@math.tamu.edu

Abstract: This paper provides an overview of the new features of the finite element library deal.II version 8.3.

1 Overview

deal.II version 8.3 was released August 1, 2015. This paper provides an overview of the new features of this release and serves as a citable reference for the deal.II software library version 8.3. deal.II is an object-oriented finite element library used around the world in the development of finite element solvers. It is available for free under the GNU Lesser General Public License (LGPL) from the deal.II homepage at http://www.dealii.org/.

The major changes of this release are:

- Improved handling of parallel distributed meshes, including a better numbering of cells on coarse meshes based on a hierarchical concept instead of Cuthill-McKee numbering.
- New abstract C++11 interface to linear operators.
- All examples have been changed to use the new DynamicSparsityPattern.
- Improved support for periodic boundary conditions with arbitrary orientations.
- New quadrature formulas.
- Full conversion to the new manifold mechanism (manifold_id) for boundary descriptions.
- Better support for complex-valued problems by doing internal arithmetic in the correct data types, rather than defaulting to double precision.
- An implementation of Bernstein polynomial-based finite elements.
- Interface to the new algebraic multilevel package MueLu of the Trilinos project [22].

- More descriptive exception messages in many places for improved user productivity in code development.
- More than 140 other features and bugfixes.

Some of these will be detailed in the following section. Information on how to cite `deal.II` is provided in Section 3.

2 Significant changes to the library

This release of `deal.II` contains a number of large and significant changes that will be discussed in the following sections. It of course also contains a vast number of smaller changes and added functionality; the details of these can be found in the file that lists all changes for this release and that is linked to from the web site of each release as well as the release announcement.

2.1 Abstract concepts of linear operators

`deal.II` now offers a versatile mechanism for storing the concept of a linear operator as well as partially applied computations. The mechanism is inspired by the *expression template* optimization strategy, where C++ templates are used to create an arithmetic expression at compile time that can then be efficiently be evaluated at runtime. In contrast to the template approach, the implementation in `deal.II` utilizes C++11 features* such as *lambda expressions* and `std::function` objects that avoid the majority of the "template overhead" that usually comes with a pure template solution. In particular, it avoids the often very lengthy and cumbersome error messages associated with programming expression templates.

The mechanism is centered around two classes, `LinearOperator` and `PackagedOperation`. Both essentially consist of several `std::function` objects that store the knowledge of how to apply a linear operator, or a packaged operation, and how to initialize vectors of range and domain space:

C++ code

```
template <typename Range, typename Domain> class LinearOperator {
public:
  std::function<void(Range &v, const Domain &u)> vmult;
  // ...
  std::function<void(Range &v, bool fast)> reinit_range_vector;
  std::function<void(Domain &v, bool fast)> reinit_domain_vector;
};

template <typename Range> class PackagedOperation {
public:
  std::function<void(Range &v)> apply;
  // ...;
  std::function<void(Range &v, bool fast)> reinit_vector;
};
```

The primary purpose of the linear operator concept is to provide syntactic sugar for complex matrix-vector operations and free the user from having to create, set up, and handle intermediate storage locations by hand. As an example consider the operation $(A + k B) C$, where A, B and C denote (possible different) matrices (or, in fact, other linear operators). In order to construct a `LinearOperator` `op` that stores the knowledge of this operation, one can write:

C++ code

```
  dealii::SparseMatrix<double> A, B, C;
  const double k = ...;
  // Setup and assembly of matrices
```

```
const auto op_a = linear_operator(A);
const auto op_b = linear_operator(B);
const auto op_c = linear_operator(C);

const auto op = (op_a + k * op_b) * op_c;
```

Now, `op` can be used as a matrix object in its own right for further computation. This includes passing it to any of `deal.II`'s linear solver classes as the operator for which to solve a linear system. To make this work, `deal.II` appropriates overloads arithmetic operators to allow composition of linear operators as above.

The `PackagedOperation` class allows lazy evaluation of expressions involving vectors and linear operators. This is done by storing the computational expression and only performing the computation when either the object is implicitly converted to a vector object, or if `apply()` is invoked by hand. This avoids unnecessary temporary storage of intermediate results. As an example consider the addition of multiple vectors:

C++ code

```
dealii::Vector<double> a, b, c, d;
// ..
dealii::Vector<double> result = a + b - c + d;
```

`operator+` (with two vector arguments) is implemented to return a `PackagedOperation` whose `apply` function is implemented the following way:

C++ code

```
apply = [&a, &b](Range &x) {
  x = a;
  x += b;
};
```

Similarly, addition and subtraction of mixed vector and packaged operation types are implemented. If implemented in the most straightforward way, executing these additions would require the use of multiple temporary vectors, along with the necessary memory allocation and de-allocation. On the other hand, creating a `PackagedOperation` for the expression `a + b - c + d` and converting it to a vector results in code equivalent to the following:

C++ code

```
dealii::Vector<double> a, b, c, d;
// ..
dealii::Vector<double> result = a;
result += b;
result -= c;
result += d;
```

Consequently, this operation avoids the use of any intermediate storage.

Further, in addition to pure vector operations, scalar multiplication (and thus all vector space operations) and application of a `LinearOperator` to a vector can be expressed within a `PackagedOperation`. For example, a residual can be expressed as:

C++ code

```
dealii::Vector<double> residual =  b - op_a * x;
```

In all of these cases, `PackagedOperation` only stores references to vector and linear operator arguments. As a consequence, creating a `PackagedOperation` object and evaluating it multiple times will always use the then-current values of the arguments, not the values at the time the packaged operation was created.

2.2 Initial support for iso-geometric analysis

Nonuniform rational B-splines (NURBS) are the basis of most modern CAD packages. Isogeometric analysis is a computational technique that aims at integrating finite element analysis (FEA) into conventional NURBS-based CAD design tools by directly using NURBS basis functions in the FEA application [15]. The preliminary implementation of the iso-geometric analysis concept in the `deal.II` library is based on two main classes: a new finite element based on Bernstein polynomials called `FE_Bernstein` and a general mapping called `MappingFEField`, which generalizes the iso-parametric model to allow the description of the geometry using arbitrary finite element displacement or location fields.

The scalar Bernstein finite element `FE_Bernstein`, in analogy with `FE_Q`, yields the finite element space of continuous, piecewise Bernstein polynomials of degree p in each coordinate direction [15]. This class is realized using tensor product polynomials of Bernstein basis polynomials.

The `MappingFEField` is a generalization of the `MappingQEulerian` class, for arbitrary vector finite elements. The two main notable differences are that this class interprets the degrees of freedom of the underlying `DoFHandler` as absolute positions instead of displacements; and it allows for arbitrary finite element types. This class effectively decouples topology from geometry, by relegating all geometrical information to the components of a FiniteElement vector field. The underlying `Triangulation` class is only used to infer topological and connectivity information between cells.

(This feature was primarily developed by Marco Tezzele.)

2.3 New quadrature formulas

Two specialized quadrature formulas have been added to the library.

2.3.1 Telles quadrature formulas Many applications require the integration of functions which might be singular in some a-priori known point locations. A common strategy in such cases, which was already implemented in the library, is the use of Lachat-Watson quadrature formulas [27]. These formulas are fairly accurate but very expensive. A new quadrature formula for singular integrals was introduced in `deal.II` that implements the quadrature formula developed by Telles in [35].

The main idea of this quadrature formula is to create an iso-morphism of a standard Gauss quadrature formula according to a nonlinear transformation that has both first and second order derivatives equal to zero at the singularity point. Such a transformation removes the singularity of the integrand and drops the approximation error from about the 10% to 0.3% for a large class of singular integrands [35]. These quadrature formulas are particularly relevant for approximation of Boundary Integral Equations (see, e.g., [18]).

(This feature was primarily developed by Nicola Giuliani.)

2.3.2 Gauss-Chebyshev quadrature formulas

Gauss-Chebyshev quadrature formulas [1] are used to approximate integrals of functions on the interval $[-1, 1]$ with weight $w(x) = (1-x^2)^{-1/2}$. In deal.II the reference interval is $[0, 1]$, hence our implementation of Gauss-Chebyshev formulas are mapped to this interval, and the weight is changed to $w(x) = 1/\sqrt{x(1-x)}$.

As for the other Gaussian quadrature formulas, three flavors of Gauss-Chebyshev quadrature rules have been implemented:

Gauss-Chebyshev (GC) formulas with N points that integrate exactly polynomials up to the order $2N - 1$;

Gauss-Radau-Chebyshev (GRC) formulas with N points where one of the two endpoints of the interval can be specified as a constrained quadrature point; these formulas are exact up to the order $2N - 2$;

Gauss-Lobatto-Chebyshev (GLC) formulas with N points where two quadrature nodes are constrained to lie on the boundary points of the interval; these formulas exactly integrate polynomials up to the order $2N - 3$.

One important application of the GLC quadrature formulas is in the context of Spectral Element Methods (SEM) using Lagrangean interpolants on GLC points as basis functions (this is a good choice for the composite interpolation of "very regular" continuous functions). This pair of basis functions and quadrature formulas generates diagonal mass matrices, that can be efficiently applied both in direct and in inverse form. For saddle point problems, one common choice in SEM [29] is the $\mathbb{Q}_N - \mathbb{Q}_{N-2}$ (inf-sup stable) couple based on Lagrangean interpolants on $N+1$ GLC and $N - 1$ GC nodes respectively.

(This feature was primarily developed by Giuseppe Pitton.)

2.4 Better error messages

deal.II uses assertions extensively to validate that input arguments to functions are within their allowed ranges, are mutually consistent, and satisfy appropriate preconditions. We also use assertions to check for internal conditions as well as for postconditions of many functions. All told, the sources of deal.II contain around 9,000 assertions.

Whenever an assertion is failed, deal.II aborts the program with an error message that includes the failing condition, its location, the surrounding function's name, the call stack, and additional information specific to the exception class associated with this particular error. For example, trying to read an element of a vector that does not exist would trigger the exception in the following code:

C++ code

```cpp
template <typename Number>
inline
Number Vector<Number>::operator() (const size_type i) const
{
  Assert (i<vec_size, ExcIndexRange(i,0,vec_size));
  return val[i];
}
```

The error message produced from this code would look like this:

```
--------------------------------------------------------
An error occurred in line <1222> of file <.../include/deal.II/lac/vector.h> in function
    Number& dealii::Vector<number>::operator()(dealii::Vector<number>::size_type)
      [with Number = double, dealii::Vector<number>::size_type = unsigned int]
The violated condition was:
    i<vec_size
The name and call sequence of the exception was:
    ExcIndexRangeType<size_type>(i,0,vec_size)
Additional Information:
Index 123456 is not in the half-open range [0,679).

Stacktrace:
-----------
#0  ./step-22: dealii::Vector<double>::operator()(unsigned int)
#1  ./step-22: Step22::StokesProblem<2>::assemble_system()
#2  ./step-22: Step22::StokesProblem<2>::run()
#3  ./step-22: main
--------------------------------------------------------
```

In this case, the error message does state the index that is being accessed, as well as the size of the vector, both under “Additional Information”. The error message may not be overly clear, but it helps illustrate the values of variables that participated in the failed condition `i<vec_size`.

However, many error messages produced by `deal.II` either do not produce any additional information at all, or very little that is understandable to a novice user. Observing users deal with errors in the courses we teach has shown that poor error messages are a major obstacle to quickly identifying the causes of problems. Ideally, error messages would attempt to explain what is going wrong, what are typical causes for this, and what possible solutions are.

In an effort to improve this situation, we have rewritten and augmented the error messages produced by several dozen classes of assertions. Going through the remainder of all error messages is a longer-term project.

3 How to cite `deal.II`

In order to justify the work the developers of `deal.II` put into this software, we ask that papers using the library reference one of the `deal.II` papers. This helps us justify the effort we put into it.

There are various ways to reference `deal.II`. To acknowledge the use of a particular version of the library, reference the present document. For up to date information and bibtex snippets for this document see:

https://www.dealii.org/publications.html

The original `deal.II` paper containing an overview of its architecture is [8]. If you rely on specific features of the library, please consider citing any of the following:

- For geometric multigrid: [24, 23];
- For distributed parallel computing: [7];
- For *hp* adaptivity: [13];

– For matrix-free and fast assembly techniques: [26];

– For computations on lower-dimensional manifolds: [17];

– For integration with CAD files and tools: [19].

deal.II can interface with many other libraries:

– ARPACK [28]

– BLAS, LAPACK

– HDF5 [36]

– METIS [25]

– MUMPS [2, 3, 4, 30]

– muparser [31]

– NetCDF [34]

– OpenCASCADE [32]

– p4est [14]

– PETSc [5, 6]

– SLEPc [20]

– Threading Building Blocks [33]

– Trilinos [21, 22]

– UMFPACK [16]

Please consider citing the appropriate references if you use interfaces to these libraries.

Older releases of deal.II can be cited as [9, 10, 11].

4 Acknowledgements

deal.II is a world-wide project with dozens of contributors around the globe. Other than the authors of this paper, the following people contributed code to this release: Daniel Arndt, Mauro Bardelloni, Praveen C, Denis Davydov, Anton Evgrafov, Arezou Ghesmati, Nicola Giuliani, Lukas Korous, Ross Kynch, Giuseppe Pitton, Fernando Posada, Lei Qiao, Angel Rodriguez, Ian Rose, Florian Sonner, Martin Steigemann, Zhen Tao, Marco Tezzele, David Wells.

Their contributions are much appreciated!

deal.II and its developers are financially supported through a variety of funding sources. W. Bangerth and B. Turcksin were partially supported by the National Science Foundation under award OCI-1148116 as part of the Software Infrastructure for Sustained Innovation (SI2) program; and by the Computational Infrastructure in Geodynamics initiative (CIG), through the National Science Foundation under Award No. EAR-0949446 and The University of California – Davis.

L. Heltai was partially supported by the project OpenViewSHIP, “Sviluppo di un ecosistema computazionale per la progettazione idrodinamica del sistema elica-carena”, financed by Regione FVG - PAR FSC 2007-2013, Fondo per lo Sviluppo e la Coesione, and by the project TRIM “Tecnologia e Ricerca Industriale per la Mobilità Marina”, CTN01-00176-163601, funded by MIUR - Ministero dell’Istruzione, dell’Università e della Ricerca.

T. Heister was partially supported by the Computational Infrastructure in Geodynamics initiative (CIG), through the National Science Foundation under Award No. EAR-0949446 and The University of California – Davis.

The Interdisciplinary Center for Scientific Computing (IWR) at Heidelberg University has provided hosting services for the `deal.II` web page and the SVN archive.

References

[1] M. Abramowitz and I. Stegun. *Handbook of Mathematical Functions*. Dover, 1965.

[2] P. Amestoy, I. Duff, and J.-Y. L'Excellent. Multifrontal parallel distributed symmetric and unsymmetric solvers. *Comput. Methods in Appl. Mech. Eng.*, 184:501–520, 2000.

[3] P. R. Amestoy, I. S. Duff, J. Koster, and J.-Y. L'Excellent. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal on Matrix Analysis and Applications*, 23(1):15–41, 2001.

[4] P. R. Amestoy, A. Guermouche, J.-Y. L'Excellent, and S. Pralet. Hybrid scheduling for the parallel solution of linear systems. *Parallel Computing*, 32(2):136–156, 2006.

[5] 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, 2014.

[6] 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. `http://www.mcs.anl.gov/petsc`, 2014.

[7] W. Bangerth, C. Burstedde, T. Heister, and M. Kronbichler. Algorithms and data structures for massively parallel generic adaptive finite element codes. *ACM Trans. Math. Softw.*, 38:14/1–28, 2011.

[8] W. Bangerth, R. Hartmann, and G. Kanschat. deal.II — a general purpose object oriented finite element library. *ACM Trans. Math. Softw.*, 33(4), 2007.

[9] W. Bangerth, T. Heister, L. Heltai, G. Kanschat, M. Kronbichler, M. Maier, B. Turcksin, and T. D. Young. The `deal.II` library, version 8.0. *arXiv preprint* *`http://arxiv.org/abs/1312.2266v3`*, 2013.

[10] W. Bangerth, T. Heister, L. Heltai, G. Kanschat, M. Kronbichler, M. Maier, B. Turcksin, and T. D. Young. The `deal.II` library, version 8.1. *arXiv preprint* *`http://arxiv.org/abs/1312.2266v4`*, 2013.

[11] W. Bangerth, T. Heister, L. Heltai, G. Kanschat, M. Kronbichler, M. Maier, B. Turcksin, and T. D. Young. The `deal.II` library, version 8.2. *Archive of Numerical Software*, 3, 2015.

[12] W. Bangerth and G. Kanschat. Concepts for object-oriented finite element software – the `deal.II` library. Preprint 1999-43, SFB 359, Heidelberg, 1999.

[13] W. Bangerth and O. Kayser-Herold. Data structures and requirements for *hp* finite element software. *ACM Trans. Math. Softw.*, 36(1):4/1–4/31, 2009.

[14] C. Burstedde, L. C. Wilcox, and O. Ghattas. `p4est`: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees. *SIAM J. Sci. Comput.*, 33(3):1103–1133, 2011.

[15] J. A. Cottrell, T. J. R. Hughes, and Y. Bazilevs. *Isogeometric analysis: toward integration of CAD and FEA*. John Wiley & Sons Inc, 2009.

[16] T. A. Davis. Algorithm 832: UMFPACK V4.3—an unsymmetric-pattern multifrontal method. *ACM Trans. Math. Softw.*, 30:196–199, 2004.

[17] A. DeSimone, L. Heltai, and C. Manigrasso. Tools for the solution of PDEs defined on curved manifolds with deal.II. Technical Report 42/2009/M, SISSA, 2009.

[18] N. Giuliani, A. Mola, L. Heltai, and L. Formaggia. FEM SUPG stabilisation of mixed isoparametric BEMs: application to linearised free surface flows. *Engineering Analysis with Boundary Elements*, 59:8–22, 2015.

[19] L. Heltai and A. Mola. Towards the Integration of CAD and FEM using open source libraries: a Collection of deal.II Manifold Wrappers for the OpenCASCADE Library. Technical report, SISSA, 2015. Submitted.

[20] 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.

[21] 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.*, 31:397–423, 2005.

[22] M. A. Heroux et al. Trilinos web page, 2014. http://trilinos.sandia.gov.

[23] B. Janssen and G. Kanschat. Adaptive multilevel methods with local smoothing for H^1- and H^{curl}-conforming high order finite element methods. *SIAM J. Sci. Comput.*, 33(4):2095–2114, 2011.

[24] G. Kanschat. Multi-level methods for discontinuous Galerkin FEM on locally refined meshes. *Comput. & Struct.*, 82(28):2437–2445, 2004.

[25] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20(1):359–392, 1998.

[26] M. Kronbichler and K. Kormann. A generic interface for parallel cell-based finite element operator application. *Comput. Fluids*, 63:135–147, 2012.

[27] J. C. Lachat and J. O. Watson. Effective numerical treatment of boundary integral equations: A formulation for three-dimensional elastostatics. *International Journal for Numerical Methods in Engineering*, 10(5):991–1005, 1976.

[28] R. B. Lehoucq, D. C. Sorensen, and C. Yang. *ARPACK users' guide: solution of large-scale eigenvalue problems with implicitly restarted Arnoldi methods*. SIAM, Philadelphia, 1998.

[29] Y. Maday, A. Patera, and E. Rønquist. A well-posed optimal spectral element approximation for the stokes problem. Technical Report 87-47, ICASE, Hampton, VA, 1987.

[30] MUMPS: a MUltifrontal Massively Parallel sparse direct Solver. http://graal.ens-lyon.fr/MUMPS/.

[31] muparser: Fast Math Parser Library. http://muparser.beltoforion.de/.

[32] OpenCASCADE: Open CASCADE Technology, 3D modeling & numerical simulation. http://www.opencascade.org/.

[33] J. Reinders. *Intel Threading Building Blocks*. O'Reilly, 2007.

[34] R. Rew and G. Davis. NetCDF: an interface for scientific data access. *Computer Graphics and Applications, IEEE*, 10(4):76–82, 1990.

[35] J. Telles. A self-adaptive co-ordinate transformation for efficient numerical evaluation of general boundary element integrals. *International Journal for Numerical Methods in Engineering*, 24(5), 2005.

[36] The HDF Group. Hierarchical Data Format, version 5, 1997-NNNN. http://www.hdfgroup.org/HDF5/.