New features in deal.II

Matthias Maier

School of Mathematics
University of Minnesota Twin Cities
Minneapolis, MN, USA

Fifth deal.II Users and Developers Workshop

College Station, TX, USA

August 3, 2015

UNIVERSITY OF MINNESOTA

Organizational

Organizational

Switch to git

- distributed development structure with review policy
- only 8 developers with direct commit access

Move to github

- around 200 unique visitors per week
- around 50 unique clones per week
- Everyone is encouraged to participate!

Release	8.0 (2012)	8.1 (2013)	8.2 (2015)	8.3 (2015)
Authors	71	76	95	119

Organizational

Everyone is encouraged to participate!

- demonstration on Tuesday
- If interested, we can also do a short git session.

We hope to

- further increase user contribution and developer base
- increase bus factor
- gain support for more platforms (Windows)
- increase packaging for (Linux) distributions
- ...

Code cleanup

A small remark:

A big part of the deal.II 8.3 release is code cleanup.

Code cleanup

Changelog

```
Removed: This release removes a number of functions that have long been
deprecated and that were previously already marked as deprecated (i.e.,
they would have yielded warnings by the compiler whenever you tried to
use them). In almost all cases, there is a function with same name but
different argument list that should be used instead. Specifically, the
removed functions and classes are:

With headers in deal.II/base/:
- ThreadManagement::spawn.
- Threads::ThreadCondition and Threads::ThreadMutex.
- DataOutBase::create_xdmf_entry with 3 arguments.
- DataOutBase::write_hdf5_parallel with 2 arguments.
- The versions of FunctionParser::initialize that took a
  use_degrees or constants argument.
  The implementation as it is now no longer supports either of
  these two concepts (since we switched from the FunctionParser
  library to the muparser library after the deal.II 8.1 release).
- GridOutFlags::XFig::level_color.
- class BlockList.
- The MPI support functions in namespace Utilities and Utilities::System.
- Deprecated members of namespace types.
- Namespace deal_II_numbers.
- MultithreadInfo::n_default_threads.
- Table::data.
```

Code cleanup

Changelog cont.

```
With headers in deal.II/lac/:
- The deprecated constructors of SparseMIC,
  SparseILU, and SparseLUDecomposition.
- SparseMIC::decompose and SparseILU::decompose.
- SparseMIC::reinit and SparseLUDecomposition::reinit.
- SparseILU::apply_decomposition.
- SparseLUDecomposition::decompose and SparseLUDecomposition::is_decomposed.
- The compress() functions without argument in the various vector
  classes. You should use the versions with a VectorOperation
  argument instead.
- Vector::scale.
- TrilinosWrappers::*Vector*::compress with an Epetra_CombineMode
  argument.
- SparsityPattern and ChunkSparsityPattern functions that take an
  optimize_diagonal argument.
- SparsityPattern::partition.
- SparsityPattern::get_rowstart_indices and
  SparsityPattern::get_column_numbers.
- SparsityPattern::row_iterator and corresponding row_begin() and row_end()
  functions.
- CompressedSparsityPattern::row_iterator and corresponding row_begin()
  and row_end() functions.
- The typedef CompressedBlockSparsityPattern.
- The deprecated constructors of SparsityPattern iterator classes.
```

Code cleanup

Changelog cont.

```
- The deprecated variants of DoFTools::make_periodicity_constraints.
- BlockMatrixArray and BlockTrianglePreconditioner functions that
  take an explicit VectorMemory object.
- The SolverSelector constructor that takes a VectorMemory argument.
- The version of parallel::distributed::Vector::compress_finish
  function that takes a boolean as argument.
- The version of BlockVector::scale and
  parallel::distributed::Vector::scale,
  parallel::distributed::BlockVector::scale
  function that takes a scalar as argument.
- PreconditionBlock::size.
- Classes PreconditionedMatrix and PreconditionLACSolver.
- PETScWrappers::VectorBase::update_ghost_values.
- PETScWrappers::MPI::Vector constructors and reinit variants.
- SparseMatrixIterators::Accessor and SparseMatrixIterators::Iterator
  constructors.
- SparseMatrix::raw_entry and SparseMatrix::global_entry.
- The ConstraintMatrix functions that transform a matrix, vector, or
  linear system into a smaller by not just setting the corresponding
  rows and columns to zero, but actually shrinking the size of the
  linear system.

With headers in deal.II/deal.II/:
- GridGenerator::laplace_transformation.
```

Code cleanup

Changelog cont.

```
- The version of GridGenerator::parallelogram where the corners are given
  as a rank-2 tensor rather than as an array of points.
- GridTools::create_union_triangulation.
- GridTools::extract_boundary_mesh.
- Triangulation::distort_random.
- Triangulation::clear_user_pointers.
- The refinement listener concept of the Triangulation class. This
  approach to getting notified about what happens to triangulations
  has been superseded by the signals defined by the triangulation
  class.

With headers in deal.II/fe/:
- In FEValues and related classes, the functions that contain the
  term 2nd_derivatives were removed in favor of those
  with names containing hessian. Similarly, functions
  with names including function_grads were removed in
  favor of those called function_gradients. Finally,
  the cell_normal_vector functions were replaced by
  normal_vector ones. In all cases, the new functions
  have been around for a while.
- Mapping::transform_covariant and Mapping::transform_contravariant.

With headers in deal.II/dofs/:
- DoFRenumbering::downstream_dg.
```

Code cleanup

Changelog cont.

```
- DoFTools::count_dofs_per_component.
- DoFTools::make_sparsity_pattern with a vector-of-vector mask.

With headers in deal.II/multigrid/:
- The constructors of classes MGSmoother, MGSmootherRelaxation and
  MGSmootherPrecondition that take a VectorMemory object.
- MGLevelObject::get_minlevel and MGLevelObject::get_maxlevel.
- MGConstrainedDoFs::non_refinement_edge_index
- MGConstrainedDoFs::at_refinement_edge_boundary
- MGTools::count_dofs_per_component.
- MGTools::apply_boundary_values.
- MGTools::extract_inner_interface_dofs.
- Class MGMatrix.
- Multigrid::vmult and friends.

With headers in deal.II/matrix_free/:
- Classes FEEvaluationDGP, FEEvaluationGeneral and FEEvaluationGL.

With headers in deal.II/mesh_worker/:
- Deprecated variants of MeshWorker::loop and MeshWorker::integration_loop.

With headers in deal.II/algorithm/:
- Algorithms::ThetaTimestepping::operator().
- Algorithms::ThetaTimestepping::initialize.
```

Code cleanup

Changelog cont.

```
- Algorithms::Newton::initialize.

With headers in deal.II/numerics/:
- TimeDependent::end_sweep (with an argument).
- PointValueHistory::mark_locations.
- The DataPostprocessor::compute_derived_quantities_scalar and
  DataPostprocessor::compute_derived_quantities_vector functions without
  evaluation points. If you have
  data postprocessor classes implemented in your program that overload these
  functions, you will have to change it in a way that they overload the
  functions of same name but with the evaluation point argument instead.
This release also removes the deprecated class MGDoFHandler. The
functionality of this class had previously been incorporated into
the DoFHandler class. Unlike the changes above, if you were still
using this class, you will need to do the following changes to
your code:
- Where you called mg_dof_handler.distribute_dofs()
  you now also need to explicitly call
  mg_dof_handler.distribute_mg_dofs().
- If you called mg_dof_handler.begin(level), you
  will now have to write this as
  mg_dof_handler.begin_mg(level) to make clear that
  you are not just interested in an iterator to a cell on a given
  level, but in fact to a cell that can access the degrees of
  freedom on a particular level of a multigrid hierarchy.
```

Code cleanup

Changelog cont.

```
- The type previously referred to as
  MGDoFHandler::cell_iterator now corresponds to
  MGDoFHandler::level_cell_iterator.
- Where you previously called DoFRenumbering::component_wise
  for the entire MGDoFHandler object, you now need to call
  this function for the DoFHandler object, and then call the
  same function with the level argument for each
  of the levels of the triangulation individually.

(Wolfgang Bangerth, 2014/12/29-2015/01/22)
```

etc.

Changes to the build system

Changes to the build system

- User configurable testsuite

- Support for git branch and revision information

Unit tests

- A unit test is a small test for the expected output of a building block

Implementation in deal.II:

- `./tests/category/test.cc` - small program that links against the library and runs a specific function/method/data structure with defined input.
- `./tests/category/test.output` - a comparison file the output of the program is compared against.

Unit tests

bunny.cc

```
#include<cstdlib>
int main()
{
  return std::system("cowsay␣-f␣bunny␣Muh\\!");
}
```

bunny.output

```
 ------
< Muh! >
 ------
  \
   \   \
        \ /\
        ( )
      .( o ).
```

Unit tests

This mechanism is readily available for user projects:

CMakeLists.txt

```
add_subdirectory(src)
ENABLE_TESTING()
add_subdirectory(tests)
```

src/CMakeLists.txt

```
add_library(foo [...])
DEAL_II_SETUP_TARGET(foo)

add_executable(bar [...])
```

tests/CMakeLists.txt

```
set(TEST_LIBRARIES foo)
DEAL_II_PICKUP_TESTS()
```

Reproducible computations

An observation

- Source code evolves over time.
- It is surprisingly hard to reproduce a computation made half a year ago exactly with source code that is under development.

Consequence

- Use a version control system to manage your source code!
- Annotate all your computations with the sha1 of the git commit used - for both, deal.II library and your source code.

Reproducible computations

CMakeLists.txt

```
DEAL_II_QUERY_GIT_INFORMATION()
```

src/CMakeLists.txt

```
SET_PROPERTY(TARGET bar APPEND PROPERTY COMPILE_DEFINITIONS
  DEMO_GIT_BRANCH="${GIT_BRANCH}"
  DEMO_GIT_REVISION="${GIT_REVISION}"
  DEMO_GIT_SHORTREV="${GIT_SHORTREV}"
  DEAL_II_VERSION="${DEAL_II_VERSION}"
  DEAL_II_GIT_BRANCH="${DEAL_II_GIT_BRANCH}"
  DEAL_II_GIT_REVISION="${DEAL_II_GIT_REVISION}"
  DEAL_II_GIT_SHORTREV="${DEAL_II_GIT_SHORTREV}"
  )
```

src/bar.cc

```
// uses the preprocessor definitions
```

Reproducible computations

- If only information about deal.II is needed:

src/bar.cc

```
#include <deal.II/base/revision.h>

// DEAL_II_GIT_BRANCH
// DEAL_II_GIT_REVISION
// DEAL_II_GIT_SHORTREV
```

Abstract concepts of linear operators

Abstract concepts of linear operator

- deal.II prevents space leaks by prohibiting (in terms of not implementing) certain functionality:

Examples:

- `Triangulation<dim>` copy constructor throws an exception, `copy_from()` has to be used
- same for `SparseMatrix<double>`, etc.
- vector types only implement inline variants of operators: `operator+=`, `operator-=`, . . .

Abstract concepts of linear operator

. . . but this has its draw-backs:

- Consider “`Vector<double> result = b - A*x - c;`”, where `A` is a matrix type and `b`, `c` are vectors:

```
Vector<double> result = c;
A.vmult_add(result, x);
result *= -1.;
result += b;
```

Abstract concepts of linear operator

. . . but this has its draw-backs:

- Consider a custom (completely contrived) preconditioner "`B + k * C`", that shall be used in an iterative solver:

```
class MyPreconditioner {
public:
  void vmult (VECTOR &dst, const VECTOR &src) {
    C.vmult(dst, src);
    dst *= k;
    B.vmult_add(dst, src);
  }

  // ...
} my_preconditioner;

Solver.solve(matrix, dst, src, my_preconditioner);
```

Abstract concepts of linear operator

Idea

- Allow for syntax like

  ```
  A + B * C;
  b - A * x - c;
  B + k * C;
  ```

but avoid unnecessary creation of temporaries:

- use the known concept of expression templates
- use C++11 features like `std::function` and lambda expressions.
- Rationale: The overhead of `std::function` is cheap compared to a typical invocation of `vmult()`.

Abstract concepts of linear operator

LinearOperator

```cpp
template <typename Range, typename Domain>
class LinearOperator
{
public:
  std::function<void(Range &v, const Domain &u)> vmult;
  std::function<void(Range &v, const Domain &u)> vmult_add;
  std::function<void(Domain &v, const Range &u)> Tvmult;
  std::function<void(Domain &v, const Range &u)> Tvmult_add;

  std::function<void(Range &v, bool fast)>
    reinit_range_vector;
  std::function<void(Domain &v, bool fast)>
    reinit_domain_vector;

  // ...
};
```

Abstract concepts of linear operator

- Encapsulation of a matrix:

```
    SparseMatrix<double> A;
    auto op_a = linear_operator(A);
```

- Addition of two `LinearOperator` objects:

```
... operator+(... &first_op, ... &second_op)
{
  LinearOperator<Range, Domain> return_op;
  return_op.vmult =
    [first_op, second_op](Range &v, const Domain &u) {
      first_op.vmult(v, u);
      second_op.vmult_add(v, u);
    };
  // ...
  return return_op;
}
```

Abstract concepts of linear operator

Example `B + k * C`:

```
#include <deal.II/lac/linear_operator.h>

const auto op_b = linear_operator(B);
const auto op_c = linear_operator(C);

Solver.solve(matrix, dst, src, op_a + k * op_c);
```

Abstract concepts of linear operator

`#include <deal.II/lac/linear_operator.h>` to

- encapsulate a matrix with `Vector<double>`:

```
auto op_a = linear_operator(A);
```

- encapsulate a matrix with custom range and domain:

```
auto op_a =
  linear_operator<Vector<float>, Vector<float>>(A);
```

- to have all vector operations for `LinearOperator` in scope:

```
op_a + op_b;
op_a - op_b;
   k * op_a;
op_a * op_b;
```

Abstract concepts of linear operator

- A class to store a computation:

PackagedOperation

```
template <typename Range>
class PackagedOperation
{
public:
  std::function<void(Range &v)> apply;
  std::function<void(Range &v)> apply_add;

  std::function<void(Range &v, bool fast)> reinit_vector;

  operator Range() const;

  // ...
};
```

Abstract concepts of linear operator

- Contraction of a `LinearOperator` with a vector:

```
... operator*(const ... &op, const ... &u) {
  // ...
  return_comp.apply = [op, &u](Range &v) {
    op.vmult(v, u); };
  return return_comp;
}
```

- Addition of two vectors:

```
... operator+(const ... &u, const ... &v) {
  // ...
  return_comp.apply = [&u, &v](Range &x) {
    x = u;
    x += v; };
  return return_comp;
}
```

Abstract concepts of linear operator

Example `b - A * x - c`:

```
#include <deal.II/lac/linear_operator.h>
#include <deal.II/lac/packaged_operation.h>

const auto op_a = linear_operator(A);

Vector<double> result = b - op_a * x - c;

// also possible:
Vector<double> result = b + c;
```

Abstract concepts of linear operator

#include <deal.II/lac/packaged_operation.h> to

- to have all vector operations for PackagedOperation in scope:

```
po_a + po_b; po_a - po_b; k * po_a;
```

- to construct a PackagedOperation out of common expressions:

```
Vector<double> u, v;
LinearOperator<> op_a;
op_a * u; u * op_a; u + v; // mixed variants,etc.
```

- to use an explicit or implicit apply:

```
Vector<double> u;
po_a.apply(u);
u = po_a;
```

Abstract concepts of linear operator

- This concept is primarily meant as “syntactic sugar”, but

- we can also guard for a common mistake:

```
A.vmult(x, x);    // error

auto op_a = linear_operator(A);
op_a.vmult(x, x); // will use intermediate storage
```

- provide intermediate storage for operator concatenation

```
auto op_a = linear_operator(A);
auto op_b = linear_operator(B);

(op_a * op_b).vmult(v, u); // with intermediate storage
```

Abstract concepts of linear operator

#include <deal.II/lac/block_linear_operator.h>

- under development, not included in deal.II 8.3
- encapsulation of linear block structures:

```
auto block_op =
  block_operator<2,2>({op_a00, op_a01, op_a10, op_a11});
```

- flexible manipulation of block structures, e. g., to construct preconditioners

Initial support for iso-geometric analysis

> *"Isogeometric analyis is a computational technique that aims at integrating finite element analysis into [conventional CAD design tool that use nonuniform rational B-splines]."*

Initial support

- `MappingFEField`
- `FE_Bernstein`

by

- Luca Heltai `<luca.heltai@sissa.it>`
- Marco Tezzele `<marcotez@gmail.com>`

Initial support for iso-geometric analysis

MappingFEField

- a generalization of the MappingQEulerian class

```
MappingFEField (const DH      &euler_dof_handler,
                const VECTOR  &euler_vector);
```

- relegates all geometrical information to a finite-element vector field that describe absolute positions

```
const FE_Q<dim> fe_q(1);
const FESystem<dim> fe_system(fe_q, dim);

DoFHandler<dim> dof_handler(triangulation);
dof_handler.distribute_dofs(fe_system);

Vector<double> eulerq(dof_handler.n_dofs());

VectorTools::get_position_vector(dof_handler, eulerq);
MappingFEField<dim,spacedim> map(dof_handler, eulerq);
```

Conclusion

. . . there is a lot more to discover. Have a look at the changelog!