Author: Luca Bertagna
Email: lbertag@emory.edu

CurrentFEManifold

The class `CurrentFEManifold` has replaced the classes `CurrentBoundaryFE` (and `CurrentBoundaryFEBase`) for the handling of boundary finite elements, introducing a significative change in the interfaces. The reason behind the name of the new class (compared to the old ones) is that we hope to generalize the concept of the boundary of a mesh to the concept of a manifold in $\mathbb{R}^n$: a boundary of a mesh can always be seen as a manifold, but sometimes it's convenient to think of a manifold not as the boundary of some higher dimensional set.

The main feature introduced by the new design is inheritance. In face, a boundary finite element is itself a finite element on the boundary, with in addition a few more features, due to the fact that it lies on a manifold (like normals, tangents, metric, etc.). Hence, in the new design `CurrentFEManifold` inherits from `CurrentFE`. The benefits of this change are the following:

i) less code repetition: inheriting from `CurrentFE`, `CurrentFEManifold` does not need to provide methods that are already inside the base class.

ii) uniformity of names: in the previous design, there were methods in `CurrentFE` and `CurrentBoundaryFE` which had different names but had the same pourpose, like for instance `nbFEDof` (in `CurrentFE`) and `nbNode` (in `CurrentBoundaryFE`).

iii) reusability of code: if using polymorphism, a `CurrentFEManifold` can be passed to a function as a `CurrentFE`, without writing an analogous routine that would perform the same action on the boundary (manifold).

iv) uniformity of update procedure: the update of the internal structures of `CurrentFEManifold` follows the same syntax of the base class. In particular, to update the normal in each quadrature node, one can simply write

```
feBd.update(mesh.boundaryFacet(faceID),UPDATE_NORMALS);
```

The update method calls first the update method in `CurrentFE`, then starts to perform all the updates contained in the update flag.

What will change in your code

- The class name. Wherever you have a type `CurrentBoundaryFE`, you should replace it with `CurrentFEManifold`. It's possible that a typedef

```
typedef CurrentFEManifold CurrentBoundaryFE;
```

  will be introduced, to make the code clear wherever the `CurrentFEManifold` is just needed as a finite element on the boundary of the mesh. However, this is not guaranteed to happen.

- All the update methods for the boundary fe now follow the style defined in `CurrentFE`. In particular, a few more update flags have been introduced

i. `UPDATE_TANGENTS`: updates the vectors that span the tangent space at each quadrature point.
ii. `UPDATE_NORMALS`: updates the normal vector at each quadrature point.
iii. `UPDATE_METRIC`: updates the metric tensor at each quadrature point.
iv. `UPDATE_INVERSE_METRIC`: updates the inverse of the metric tensor at each quadrature point.
v. `UPDATE_W_ROOT_DET_METRIC`: updates the square root of the determinant of the metric tensor multiplied by the quadrature weight at each quadrature point.
vi. `UPDATE_SHAPE_TENSOR`: updates the shape tensor (i.e., the second fundamental form) at each quadrature point. Note: this is identically zero in case of linear meshes.

- A few methods' names change. Here are the most common:

  i. `nbNode` is now `nbFEDof` (inherited from `CurrentFE`).
  ii. `meas` is now `detMetric`. although the measure is actually the square root of the metric determinant. However, the name `meas` was somewhat ambiguous, since it does not return the measure of the element, which can be retrieved using the method `measure` (overrides the base class one).
  iii. `weightMeas` is now `wRootDetMetric`.

- all the `hasXXX` method of `CurrentBoundaryFE` (which checked if the quantity `XXX` had been updated) are now suppressed, for uniformity with `CurrentFE` (where they're not present).

Add-ons and new features

As you probably noticed from the update flags list, the new class `CurrentFEManifold` also introduces a couple more features not present in the old `CurrentBoundaryFE` class:

- computation of the inverse of the metric tensor: this is useful in case covariant derivatives for assembly on the boundary are needed.
- computation of shape tensor: in case the mesh is quadratic, the shape tensor (aka, the second fundamental form) can be computed on the element. This can be useful in case one needs an approximation of the local curvature of the boundary (manifold) on arbitrary geometries. In case the mesh is not quadratic, this tensor will be identically zero.