Chapter 1

Hijacking degrees of freedom in oomph-lib

Certain problem formulations require changing the equation corresponding to (having the same global number as) a particular degree of freedom.

For example, in steady free surface problems, there is a constraint on the total fluid volume, but no "obvious" associated degree of freedom. For these problems, a fluid pressure degree of freedom may be "traded" for the volume constraint. The "traded" degree of freedom remains a fluid pressure and must appear in the governing equations in that role. However, the equation associated with that unknown is no longer the continuity equation, but the volume constraint, see `another tutorial` for more details.

A similar situation arises when applying complex boundary conditions: perhaps the fluid velocity on the boundary is itself the solution of an equation depending on a local chemical concentration. One way of implementing such boundary conditions is to introduce a new variable representing the boundary velocity, solve the additional equation for that velocity and then impose that as a Dirichlet condition. However, the condition can be implemented without introducing a new variable by changing the equation associated with the fluid velocity on the boundary.

In `oomph-lib`, this type of process is known as **hijacking** and uses the `Hijacked` wrapper class, which adds new member functions to the element:

```
Data* hijack_internal_value(const unsigned &n, const unsigned &i)
Data* hijack_external_value(const unsigned &n, const unsigned &i)
Data* hijack_nodal_value(const unsigned &n, const unsigned &i)
Data* hijack_nodal_position_value(const unsigned &n, const unsigned &i)
Data* hijack_nodal_spine_value(const unsigned &n, const unsigned &i)
```

Calling any of these functions instructs the bulk element to set the specified residual and associated row in the Jacobian matrix to zero. For example,

```
Hijacked<QPoissonElement<1,2> > *el_pt = new Hijacked<QPoissonElement<1,3> >;
el_pt->hijack_nodal_value(0,0);
```

sets to zero the residual associated with the first data value stored at the first (local) node in the element.

Note that hijacking is on a per element basis, so that if a degree of freedom is shared between elements it can be hijacked in all, none or some of them, as required.

The return type of each member function is a pointer to a `Data` object. This `Data` object is a "custom" object that consists of a (copy of a) single value corresponding to the "hijacked" degree of freedom and its global equation number. This object is provided so that it can be used as external data in other elements that assemble the "new" equations for the unknown, see `another tutorial` for an example. Note that this can lead to memory leaks unless the new object is safely deleted, so if you do not want to make use of the `Data` object you should precede the call to the member function by `delete`, e.g.

```
delete el_pt->hijack_internal_value(0,0);
```

1.1 Notes and Guidelines

- If you do not assign a new equation to the "hijacked" unknown the Jacobian matrix will be singular because there will be a zero row.
- Hijacking should take place before equation numbering.
- The member function `Hijacked::unhijack_all_data()` can be used to reset all hijacked values in an element.
- Hijacking will persist throughout adaptation, but only if the degree of freedom remains "active". It is safest to `unhijack_all_data()` before adaptation and then hijack again in `actions_after_adapt()`.
- All free surface elements are actually `Hijacked`, because hijacking is used to enforce the contact angle condition in certain circumstances.
- The member function
  `double* &Hijacked::residual_multiplier_pt()`

  can be used to set a double value that multiplies the hijacked residuals. The default and most common case it that the multiplier is zero, but non-zero values may be useful in homotopy continuation calculations in order to move smoothly between different types of boundary condition.

1.2 PDF file

A `pdf version` of this document is available.