Chapter 1

Interfaces, Free Surfaces and Surface Transport: Theory and implementation

This document provides the theoretical background underlying `oomph-lib's` free surface, interface and surface transport capabilities. We begin with a review of the relevant theory to establish the overall framework and notation, and then discuss the implementation of the methodology in `oomph-lib`.

Here is an overview of the structure of this document:

If you don't need all the gory details, you may prefer to start by exploring the free-surface flow tutorials in `oomph-lib's list of example driver codes.`

1.1 Theory

A complete theoretical treatment of moving surfaces in space requires the use of differential geometry, described in detail by Aris (1962) *Vectors, Tensors and the Basic Equations of Fluid Mechanics* , Wetherburn (1955) *Differential Geometry of Three Dimensions, Volume I* and Green & Zerna (1968) *Theoretical Elasticity* among others. Here, we shall present only the essential information and the reader is referred to these texts to fill in additional background.

1.1.1 Geometry of Surfaces

For our purposes, a surface is a object of dimension $n-1$ embedded in an n-dimensional Euclidean space, where n is two or three. We can always represent such a surface by a position vector from our chosen origin $\mathbf{R}^*(\zeta^\alpha, t)$, parametrised by time t and intrinsic (surface) coordinates ζ^α, where the Greek index $\alpha = 1, .., n-1$.

Throughout this document we will use the summation convention that repeated Roman indices are to be summed over the range from 1 to n and repeated Greek indices are to be summed over the range from 1 to $n-1$. We will retain the summation signs for all other sums, such as sums over the nodes etc.

The covariant base vectors of the surface are defined to be the partial derivatives of the position vector with respect to the surface coordinates. For a two-dimensional surface,

$$g_1 = \frac{\partial R}{\partial \zeta^1}, \qquad g_2 = \frac{\partial R}{\partial \zeta^2}.$$

The covariant metric tensor of the surface is formed by taking the dot product of the covariant base vectors

$$g_{\alpha\beta} = g_\alpha \cdot g_\beta,$$

and the determinant of the covariant metric tensor is denoted by

$$g = g_{11}g_{22} - g_{12}g_{21}.$$

The contravariant metric tensor is the inverse of the covariant metric tensor and is denoted by $g^{\alpha\beta}$. Thus,

$$g^{11} = g_{22}/g, \quad g^{12} = -g_{12}/g, \quad g^{21} = -g_{21}/g, \quad g^{22} = g_{11}/g.$$

For a one-dimensional surface, there is a single covariant base vector that coincides with the tangent vector

$$g_1 = \frac{\partial R}{\partial \zeta^1}.$$

Here, the covariant metric tensor is simply the inner product of the tangent vector with itself $g_{11} = g_1 \cdot g_1$ with determinant $g = g_{11}$ and the contravariant metric tensor is $g^{11} = 1/g_{11}$.

We will need to integrate quantities over the surface, which uses the result that an infinitesimal unit of area is

$$\mathrm{d}S = \sqrt{g}\, \mathrm{d}\zeta^\alpha,$$

in terms of the intrinsic coordinates.

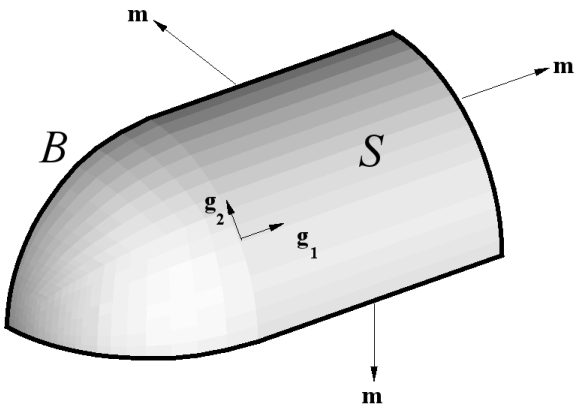

Figure 1.1: Sketch of section of a two-dimensional surface, S , in a three-dimensional space. The covariant base vectors are tangent to the surface and in the direction of the intrinsic surface coordinates. The intrinsic coordinates parametrise the position vector to material points in the surface. The surface is bounded by the curve B and the vector m is perpendicular to the outer unit normal of the surface and tangent to the bounding curve.

1.1.2 Differential Operators On A Surface

The formulation of transport equations within the surface requires the rates of change of surface quantities. The appropriate derivative is the surface gradient, often written as ∇_S and defined unhelpfully in many papers and

textbooks as the gradient operator restricted to the surface or

$$\nabla_S \phi = \nabla \phi - (\nabla \phi \cdot n) n.$$

The problem with the above definition is that if a quantity is defined only on the surface it is impossible to take its gradient ∇.

A more helpful definition in terms of the intrinsic coordinates is that

$$\nabla_s \phi = g^{\alpha\beta} g_\alpha \frac{\partial \phi}{\partial \zeta^\beta}.$$

For a one dimensional surface parametrised by the arc-length, s, this reduces to $\nabla_s \phi = t \partial \phi / \partial s$, where t is a unit tangent vector to the surface. By definition the surface gradient is tangent to the surface and has no normal component.

The surface divergence of an n-dimensional vector quantity defined on the surface is given by

$$\nabla_s \cdot v = g^{\alpha\beta} g_\alpha \cdot \frac{\partial v}{\partial \zeta^\beta}.$$

The divergence theorem applied over the surface (Aris, 1955) is

$$\iint_S \nabla_s \cdot v_t \ \mathrm{d}S = \int_B v_t \cdot m \, \mathrm{d}l,$$

where v_t is a vector tangential to the surface; and m is the unit vector tangent to the surface, but perpendicular to the tangent of the bounding curve B, see the Figure above.

If a general vector is decomposed into normal and tangential components, $v = v_t + v_n n$, we can write

$$\iint_S \nabla_s \cdot v \ \mathrm{d}S = \iint_S \nabla_s \cdot (v_t + v_n n) \ \mathrm{d}S = \iint_S \nabla_s \cdot v_t \ \mathrm{d}S + \iint_S \nabla_s \cdot (v_n n) \ \mathrm{d}S$$

$$= \iint_S \nabla_s \cdot v_t \ \mathrm{d}S + \iint_S v_n \nabla_s \cdot n + n \cdot \nabla_s v_n \ \mathrm{d}S = \iint_S \nabla_s \cdot v_t \ \mathrm{d}S - \iint_S v_n \kappa \ \mathrm{d}S,$$

where κ is twice the mean curvature of the surface and equal to minus the surface divergence of the normal. The term $n \cdot \nabla_s v_n = 0$ because the normal and surface divergence of any scalar quantity are orthogonal.

Hence using the divergence theorem on the first term on the right-hand side, we obtain

$$\iint_S \nabla_s \cdot v \ \mathrm{d}S = -\iint_S v_n \kappa \ \mathrm{d}S + \int_B v_t \cdot m \, \mathrm{d}l = -\iint_S v_n \kappa \ \mathrm{d}S + \int_B v \cdot m \, \mathrm{d}l, \quad (1)$$

because the normal component of v is perpendicular to m and therefore contributes nothing to the line integral.

If we now choose $v = \phi \, e_i$, where e_i is the unit base vector in the i-th Cartesian coordinate direction, then we obtain

$$\iint_S \nabla_s \phi \cdot e_i \ \mathrm{d}S = -\iint_S \phi (e_i \cdot n) \kappa \ \mathrm{d}S + \int_B \phi \, e_i \cdot m \, \mathrm{d}l,$$

which is equivalent to the surface divergence theorem for a scalar field described by Wetherburn (1955; p 240, Eqn 26)

$$\iint_S \kappa \phi n \ \mathrm{d}S = \int_B \phi m \ \mathrm{d}l - \iint_S \nabla_s \phi \ \mathrm{d}S. \quad (2)$$

1.1.3 Free Surface and Interface Boundary Conditions

1.1.3.1 Dynamic condition

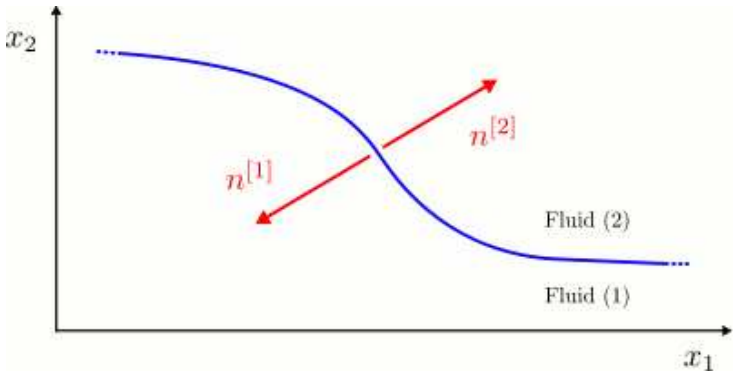

Figure 1.2: Sketch of the interface between two fluids.

The presence of an interface with non-constant surface tension σ^* contributes to the overall force balance, as also described in another tutorial . The surface tension acts a line force bounding the interface and acting in the direction m. Thus using the surface divergence theorem in the form (2),

$$\int_B \sigma^* m \, \mathrm{d}l = \iint_S \sigma^* \kappa + \nabla_S \sigma^* \, \mathrm{d}S,$$

which gives the terms to be included in the force balance.

If we define the lower fluid in the sketch above to be fluid 1 and the upper fluid to be fluid 2. The traction exerted by fluid 1 onto fluid 2, $\mathbf{t}^{[1]*}$ and that exerted by fluid 2 onto fluid 1, $\mathbf{t}^{[2]*}$. Then, balance of forces requires that

$$\mathbf{t}^{[1]*} - \mathbf{t}^{[2]*} \equiv \left[\left[\tau^* \cdot n^{[1]}\right]\right] = \sigma^* \, \kappa^* \, n^{[1]} + \nabla_S \sigma^*$$

where we have been explicit about the fact that the curvature is dimensional and $\kappa^* > 0$ if the centre of curvature lies inside fluid 1.

After using the non-dimensionalisation described in another tutorial, the boundary condition becomes

$$\left[\left[\tau \cdot n^{[1]}\right]\right] = \frac{1}{Ca}\left[\sigma \, \kappa n + \nabla_S \sigma\right]$$

where $\sigma = \sigma^*/\sigma_{ref}$ and $Ca = \mu_{ref}\mathscr{U}/\sigma_{ref}$, is the capillary number based on a reference viscosity and surface tension. This condition can be incorporated directly into the weak form of the momentum equations because the surface integral term in these equations, as described in another tutorial , is

$$\iint_S n \cdot \tau \cdot \psi^{(F)} \, \mathrm{d}S,$$

where $\psi^{(F)}$ are the vector test functions associated with the momentum equations. Note that it is more compact to work with vector test functions rather than the Cartesian components of a vector test function in this case. The terms in the integral are exactly those on the left-hand side of the boundary condition multiplied by the test function. Hence, they become

$$\iint_S \frac{1}{Ca}\left[\sigma \, \kappa n + \nabla_S \sigma\right] \cdot \psi^{(F)} \, \mathrm{d}S.$$

Early contributors computed these terms directly, but an undesirable feature is that computation of the curvature requires taking second derivatives of the position vector, which requires a higher degree of smoothness than previously demanded. We can use the surface divergence theorem again to weaken this requirement. Firstly we must use the product rule to bring the test function into the surface divergence

$$\iint_S \frac{1}{Ca}\left[\sigma \psi^{(F)} \cdot \kappa n + \nabla_S \cdot \left(\sigma \psi^{(F)}\right) - \sigma \nabla_S \cdot \psi^{(F)}\right] \, \mathrm{d}S;$$

and then we can use the surface divergence theorem (1) on the first two terms to obtain

$$\int_B \frac{1}{Ca}\sigma\psi^{(F)}\cdot m \ \mathrm{d}l - \int\!\!\int_S \frac{1}{Ca}\sigma\nabla_s\cdot\psi^{(F)} \ \mathrm{d}S.$$

In index notation these terms become

$$\int_B \frac{1}{Ca}\sigma\psi_i^{(F)}m_i \ \mathrm{d}l - \int\!\!\int_S \frac{1}{Ca}\sigma g^{\alpha\beta}\left[g_\alpha\right]_i\left[\psi_{,\beta}^{(F)}\right]_i \ \mathrm{d}S,$$

where $[g_\alpha]_i$ denotes the i-th component of the covariant base vector g_α and $\sigma_{,\beta}$ represents the derivative $\partial\sigma/\partial\zeta^\beta$ (3). These are the terms that are implemented in `oomph-lib's` free surface elements. Note that variations in surface tension are taken into account in this formulation without the explicit need to take its surface derivative. An important observation is that the intrinsic surface coordinates can be chosen to be the local coordinates of each element so that we do not need to introduce another set of coordinates.

1.1.3.2 Kinematic condition

The kinematic condition is that "particles on the surface must remain on the surface". In other words the normal velocity of the surface must equal the normal rate of change of its position with time. The condition is compactly expressed in non-dimensional form as

$$\left(U - St\frac{\partial\mathbf{R}}{\partial t}\right)\cdot n = 0,$$

where St is the Strouhal number, see another tutorial for details. Note that, in general U the velocity of the surface need *not* coincide with the velocity of the fluid.

Although this equation must be satisfied the details of how exactly it is implemented depend crucially on the mesh-update strategy chosen and we make use of the C++ features of inheritance and templating to avoid code duplication, see below for details.

1.1.4 Surface Transport Equations

Consider a chemical species with surface concentration Γ that is only present on the surface. It can be transported within the moving surface by the standard mechanisms of advection and diffusion, but changes in the surface area can also induce changes in its concentration. The formulation of surface transport equations has been discussed many times in the literature and the main confusion surrounds how material derivatives are taken. Unlike the conventional bulk equations the surface does not occupy every point in the domain, so one cannot simply use the standard form of the material derivative.

Rather than covering the literature, here we shall simply state, *ab initio* , the governing equations formulated by Huang, Lai & Tseng as well as Cermelli et al (2005), who claim it was established by Slattery (1972). We shall demonstrate that it is equivalent to the form stated by Wong & Rumshitski and used by Campana et al (2004), but that it leads to a simpler formulation which avoids explicit calculation of the curvature (and hence second derivatives).

The dimensionless governing equations in weak form governing the transport of a scalar quantity Γ are

$$\int\!\!\int_S \left[St\left(\frac{\partial\Gamma}{\partial t} - \dot{R}\cdot\nabla_s\Gamma\right) + \nabla_s\cdot(\Gamma U) - \frac{1}{Pe_s}\nabla_s\cdot\nabla_s\Gamma\right]\phi \ \mathrm{d}S = 0.$$

In the above equation the time derivative is taken at fixed “nodes” in the finite element formulation and the ALE-like term compensates for tangential movement of these nodes along the surface. The normal movement is enforced to be exactly the same as the surface velocity by the kinematic condition. Note that the velocity in the third term of the governing equation is the full (bulk) fluid velocity. The dimensionless quantity $Pe_s = \mathscr{U}\mathscr{L}/D_s$ is the surface Peclet number.

The formulation of Campana et al is found by decomposing the velocity in this term into normal and tangential components.

$$\nabla_S\cdot(\Gamma U) = \nabla_s\cdot(\Gamma U_t + \Gamma U_n n).$$

The surface gradient yields a vector that is tangential to the surface so that its inner product with the unit normal, n is zero. Thus,

$$\nabla_s\cdot(\Gamma U) = \nabla_s\cdot(\Gamma U_t) + \Gamma U_n\nabla_s\cdot n,$$

which is the starting point for Campana *et al*'s derivation because the surface divergence of the normal may be replaced by the curvature, as described above.

Returning to our formulation we use the surface divergence theorem (1) to integrate the diffusion term and the product rule to handle the third term:

$$\int\!\!\int_S \left[St\left(\frac{\partial \Gamma}{\partial t} - \dot{R}\cdot\nabla_S\Gamma\right) + \Gamma\nabla_S\cdot U + U\cdot\nabla_S\Gamma \right]\phi + \frac{1}{Pe_s}\nabla_S\Gamma\cdot\nabla_S\phi \,\mathrm{d}S - \int_B \frac{1}{Pe_s}\nabla_S\Gamma\cdot m\,\phi \;\mathrm{d}l = 0,$$

$$\Rightarrow \quad \int\!\!\int_S \left[St\frac{\partial \Gamma}{\partial t} + (U - St\,\dot{R})\cdot\nabla_S\Gamma + \Gamma\nabla_S\cdot U \right]\phi + \frac{1}{Pe_s}\nabla_S\Gamma\cdot\nabla_S\phi \,\mathrm{d}S - \int_B \frac{1}{Pe_s}\nabla_S\Gamma\cdot m\,\phi \;\mathrm{d}l = 0,$$

The line term represents no-diffusive flux out of the system. In index notation the equations are

$$\int\!\!\int_S \left[St\frac{\partial \Gamma}{\partial t} + g^{\alpha\beta}[g_\alpha]_i\left\{(U_i - St\,\dot{R}_i)\Gamma_{,\beta} + \Gamma\left[U_{,\beta}\right]_i\right\}\right]\phi + \frac{1}{Pe_s}g^{\alpha\beta}\Gamma_{,\alpha}\phi_{,\beta}\,\mathrm{d}S - \int_B \frac{1}{Pe_s}g^{\alpha\beta}[g_\alpha]_i m_i\Gamma_{,\beta}\,\phi \;\mathrm{d}l = 0,$$

These are the equations implemented in `oomph-lib` using the definitions of surface derivatives given in Differential Operators On A Surface.

1.2 Implementation

We will now discuss how the discrete versions of the equations derived above are actually implemented in `oomph-lib`. The basic idea is that the equations should be implemented independently of the specific element type and mesh-update strategy and a base class oomph::FluidInterfaceElement defines the generic functionality for all fluid interface elements. The only difference between the different surface geometries are in the definitions of the surface derivative operators and these are defined in specific classes oomph::LineDerivatives (1D surface), oomph::AxisymmetricDerivatives and oomph::SurfaceDerivatives (2D surface). The final specific element is created by using a special templated class that determines the node-update strategy and takes the base class, derivative class and bulk element as template arguments.

1.2.1 The FluidInterfaceElement class

The template-free oomph::FluidInterfaceElement class provides storage and member functions that are common to all free-surface and interface elements. The most important functions to be aware of are:

- Storage and access functions for (pointers to) the capillary and Strouhal numbers.
- Storage for a (pointer to) an external pressure degree of freedom if the boundary is a free surface, rather than an interface.
- The virtual function

  ```
  double FluidInterfaceElement::compute_surface_derivatives(...)
  ```

  specifies how the surface gradient operators are computed.
- The function

  ```
  double FluidInterfaceElement::sigma(const Vector<double> &s_local)
  ```

  that returns the surface tension at the given local coordinate; default implementation returns 1.0.
- The function

  ```
  FluidInterfaceElement::fill_in_generic_residual_contribution_interface(..)
  ```

  that is responsible for assembling the residual and jacobian contributions corresponding to the dynamic and kinematic boundary conditions.

- The function

```
FluidInterfaceElement::add_additional_residual_contributions_interface(...)
```

which is called from *within* the integration loop and is used to assemble any additional surface transport equations or equations arising from different node update strategies. This function is virtual so that it can be overloaded in derived classes.

- The function

```
virtual int FluidInterfaceElement::kinematic_local_eqn(...)
```

that is used to specify the local equation number used for the kinematic condition, which depends on the mesh-update strategy chosen.

1.2.2 The LineDerivatives class

The class oomph::LineDerivatives implements the specific surface derivatives associated with a one-dimensional surface in a two-dimensional domain.The global coordinate system is Cartesian so its base vectors do not vary with the surface coordinates and $[\psi^{(F)}_{,\beta}]_i = \psi^{(F)}_{i,\beta}$; and the contribution to each component of the momentum equation is found by setting the appropriate component $\psi^{(F)}_i = 0$ for $i = 1,2$.

1.2.3 The AxisymmetricDerivatives class

The class oomph::AxisymmetricDerivatives implements the specific residuals associated with a two-dimensional surface in a three-dimensional domain, under the assumption of axisymmetry. Thus, the coordinate system is cylindrical polar (r,z,θ), but it is assumed that there are no variations in the θ direction.

It is worthwhile including the required mathematics here because the terms are not the same as in the `LineDerivatives` class. Specifically, if the surface coordinates are $(\zeta^1,\zeta^2) = (s,\theta)$, the position vector to the surface is given by

$$R = \begin{pmatrix} r(s)\cos\theta \\ r(s)\sin\theta \\ z(s) \end{pmatrix} \quad \Rightarrow \quad g_1 = \begin{pmatrix} r'(s)\cos\theta \\ r'(s)\sin\theta \\ z'(s) \end{pmatrix}, \quad g_2 = \begin{pmatrix} -r(s)\sin\theta \\ r(s)\cos\theta \\ 0 \end{pmatrix},$$

where $r'(s) = \partial r/\partial s$ and $z'(s) = \partial z/\partial s$. Hence,

$$g_{11} = (r')^2 + (z')^2, \quad g_{12} = g_{21} = 0, \quad g_{22} = r^2, \quad \text{and} \quad g = r^2\left[(r')^2 + (z')^2\right].$$

In our standard formulation, the vector test function is given by

$$\psi^{(F)} = \begin{pmatrix} \psi^{(F)}_r(s)\cos\theta \\ \psi^{(F)}_r(s)\sin\theta \\ \psi^{(F)}_z(s) \end{pmatrix} \quad \Rightarrow \quad \psi^{(F)}_{,1} = \begin{pmatrix} (\psi')^{(F)}_r(s)\cos\theta \\ (\psi')^{(F)}_r(s)\sin\theta \\ (\psi')^{(F)}_z(s) \end{pmatrix} \quad \text{and} \quad \psi^{(F)}_{,2} = \begin{pmatrix} -\psi^{(F)}_r(s)\sin\theta \\ \psi^{(F)}_r(s)\cos\theta \\ 0 \end{pmatrix}.$$

Thus the contribution to the momentum equation (3) are

$$\int_B \frac{1}{Ca}\sigma\psi^{(F)}_i m_i \; \mathrm{d}l - \iint_S \frac{1}{Ca}\sigma\left[\frac{r'(\psi')^{(F)}_r + z'(\psi')^{(F)}_z}{(r')^2+(z')^2} + \frac{1}{r}\psi^{(F)}_r\right] \mathrm{d}S,$$

Two separate contributions are then derived from the cases $\psi^{(F)}_r = 0$ and $\psi^{(F)}_z = 0$. The difference from the `LineDerivatives` is the final term that accounts for the azimuthal curvature. In addition, we must also multiply all terms by the additional factor of r in the square-root of the determinant of the surface metric tensor.

1.2.4 Surface Transport Implementation

The class oomph::SurfaceDerivatives implements the specific residuals associated with a general two-dimensional surface in a three-dimensional domain. Once again, the global coordinate is Cartesian, so the contribution to each momentum equation is found by setting the two other components of the test function to be zero.

1.3 The SpineLine/Axi/SurfaceFluidInterfaceElement classes

We shall discuss the "line" version of the elements, but the others are essentially the same.

The class oomph::SpineLineFluidInterfaceElement is templated by the bulk element type, `ELEMENT`, and inherits from `FluidInterfaceElement`, LineDerivatives and `Hijacked<SpineElement<FaceGeometry<ELEMENT> > >`. The hijacking is only required for imposition of contact angle boundary conditions, see another tutorial for more details.

Note that the use of templates to make the code generic makes it hard to read. A simplified constructor is given below and simply builds the element based on the `FaceGeometry` of the bulk element and sets the indices associated with the bulk fluid velocity components from the bulk element.

```
SpineLineFluidInterfaceElement(FiniteElement* const &element_pt,
                               const int &face_index) :
 Hijacked<SpineElement<FaceGeometry<ELEMENT> > >(),
 LineFluidInterfaceElement()
  {
   //Attach the geometrical information to the element, by
   //making the face element from the bulk element
   element_pt->build_face_element(face_index,this);

   //Find the index at which the velocity unknowns are stored
   //from the bulk element
   ELEMENT* cast_element_pt = dynamic_cast<ELEMENT*>(element_pt);
   this->U_index_interface.resize(2);
   for(unsigned i=0;i<2;i++)
    {
     this->U_index_interface[i] = cast_element_pt->u_index_nst(i);
    }
  } //End of constructor
```

If a spine method is used to update the nodal positions then the spine height is the unknown associated with the kinematic condition. Thus, the function `kinematic_local_eqn(...)` is overloaded accordingly

```
int kinematic_local_eqn(const unsigned &n)
 {return this->spine_local_eqn(n);}
```

Finally, the element calculates the geometric contributions to the jacobian using the generic functionality in `ElementWithMovingNodes`

```
void fill_in_contribution_to_jacobian(Vector<double> &residuals,
                                      DenseMatrix<double> &jacobian)
 {
  //Call the generic routine with the flag set to 1
  fill_in_generic_residual_contribution_interface(residuals,jacobian,1);

  //Call the generic routine to evaluate shape derivatives
  this->fill_in_jacobian_from_geometric_data(jacobian);
 } //End of jacobian contribution
```

There are no additional contributions to the residuals or jacobian.

1.4 The ElasticLine/Axi/SurfaceFluidInterfaceElement classes

We shall discuss the "line" version of the elements, but the others are essentially the same.

The class oomph::ElasticLineFluidInterfaceElement is templated by the bulk element type, ELEMENT, and inherits from LineDerivatives, FluidInterfaceElement and Hijacked<FaceGeometry<ELEMENT> >. The hijacking is again only required for imposition of contact angle boundary conditions.

The "elastic" versions of the elements are more complicated than the "spine" versions because the kinematic condition is imposed using Lagrange multipliers, following the method described by Cairncross et al A finite element method for free surface flows of incompressible fluids in three dimensions. Part I. Boundary fitted mesh motion' (2000). These Lagrange multipliers must be added to the Nodes on the free surface and their introduction adds additional terms to the equations governing the bulk mesh motion.

The constructor is given below and in addition to building the element based on the FaceGeometry of the bulk element and setting the indices associated with the bulk fluid velocity components from the bulk element, it also adds the additional storage required for the Lagrange multipliers

```
ElasticLineFluidInterfaceElement(FiniteElement* const &element_pt,
                                 const int &face_index,
                                 const unsigned &id=0) :
 // Final optional argument to specify id for lagrange multiplier
 FaceGeometry<ELEMENT>(), LineFluidInterfaceElement(), Id(id)
 {
  //Attach the geometrical information to the element
  //This function also assigned nbulk_value from required_nvalue of the
  //bulk element
  element_pt->build_face_element(face_index,this);

  //Find the index at which the velocity unknowns are stored
  //from the bulk element
  ELEMENT* cast_element_pt = dynamic_cast<ELEMENT*>(element_pt);
  this->U_index_interface.resize(2);
  for(unsigned i=0;i<2;i++)
   {
    this->U_index_interface[i] = cast_element_pt->u_index_nst(i);
   }

  //Read out the number of nodes on the face
  unsigned n_node_face = this->nnode();

  //Set the additional data values in the face
  //There is one additional values at each node --- the lagrange multiplier
  Vector<unsigned> additional_data_values(n_node_face);
  for(unsigned i=0;i<n_node_face;i++)
   {
    additional_data_values[i] = 1;
   }

  // Now add storage for Lagrange multipliers and set the map containing
  // the position of the first entry of this face element's
  // additional values.
  add_additional_values(additional_data_values,id);

 } //End of constructor
```

The kinematic boundary condition is associated with the Lagrange multiplier and kinematic_local_eqn(...) is overloaded accordingly

```
/// (This is the equation for the Lagrange multiplier)
int kinematic_local_eqn(const unsigned &j)
{
 // Get the index of the nodal value associated with Lagrange multiplier
 const unsigned lagr_index=
  dynamic_cast<BoundaryNodeBase*>(node_pt(j))->
  index_of_first_value_assigned_by_face_element(Id);

 // Return nodal value
 return this->nodal_local_eqn(j,lagr_index);
}
```

The element calculates the geometric contributions to the jacobian using the generic functionality in SolidElements

```
void fill_in_contribution_to_jacobian(Vector<double> &residuals,
                                      DenseMatrix<double> &jacobian)
{
 //Call the generic routine with the flag set to 1
 fill_in_generic_residual_contribution_interface(residuals,jacobian,1);
```

```
  //Call the generic finite difference routine for the solid variables
  this->fill_in_jacobian_from_solid_position_by_fd(jacobian);
 }
```

The additional contributions to the residuals and jacobian arise from the Lagrange multiplier contributions to the equations that determine the position of the nodes. The essential loop is the contribution below which adds the normal traction to the governing equations of solid mechanics:

```
 //Loop over the shape functions to assemble contributions
 for(unsigned l=0;l<n_node;l++)
  {
   //Loop over the directions
   for(unsigned i=0;i<2;i++)
    {
     //Now using the same shape functions for the elastic equations,
     //so we can stay in the loop
     local_eqn = this->position_local_eqn(l,0,i);
     if(local_eqn >= 0)
      {
       //Add in a "Lagrange multiplier"
       residuals[local_eqn] -=
        interpolated_lagrange*interpolated_n[i]*psif[l]*W*J;

       //Do the Jacobian calculation
       if(flag)
        {
         //Loop over the nodes
         for(unsigned l2=0;l2<n_node;l2++)
          {
           //Derivatives w.r.t. solid positions will be handled by FDing
           //That leaves the "lagrange multipliers" only
           local_unknown = kinematic_local_eqn(l2);
           if(local_unknown >= 0)
            {
             jacobian(local_eqn,local_unknown) -=
              psif[l2]*interpolated_n[i]*psif[l]*W*J;
            }
          }
        } //End of Jacobian calculation
      }
    }
  } //End of loop over shape functions
```

1.5 Surface Transport Implementation

The recommended strategy for implementing surface transport equations is to inherit from the appropriate `FluidInterfaceElement`, and include the equations independently of the mesh-update strategy by overloading the function

```
FluidInterfaceElement::add_additional_residual_contributions_interface(...)
```

The mesh-update specialisations can be added by further specialisation as required. Alternatively, one could simply inherit directly from the specialised element. This is the approach taken in this `example code.`

An important point is that the axisymmetric formulation of the surface divergence of a vector is not entirely trivial, as discussed in The AxisymmetricDerivatives class. In the context of surface transport the term that is always present is $\nabla_s \cdot U$. Using the same surface coordinates as above, $(\zeta^1, \zeta^2) = (s, \theta)$, the velocity vector is

$$U = \begin{pmatrix} U_r(s)\cos\theta \\ U_r(s)\sin\theta \\ U_z(s) \end{pmatrix} \quad \Rightarrow \quad U_{,1} = \begin{pmatrix} \frac{\partial U_r}{\partial s}\cos\theta \\ \frac{\partial U_r}{\partial s}\sin\theta \\ \frac{\partial U_z}{\partial s} \end{pmatrix}, \quad U_{,2} = \begin{pmatrix} -U_r\sin\theta \\ U_r\cos\theta \\ 0 \end{pmatrix}.$$

Thus, the surface divergence term is

$$\nabla_s \cdot U = \frac{\left(\frac{\partial U_r}{\partial s}\frac{\partial r}{\partial s} + \frac{\partial U_z}{\partial s}\frac{\partial z}{\partial s}\right)}{(r')^2 + (z')^2} + \frac{U_r}{r},$$

which is exactly the same form as the surface divergence of the vector test function derived in the section The AxisymmetricDerivatives class, as expected.

1.6 PDF file

A pdf version of this document is available.