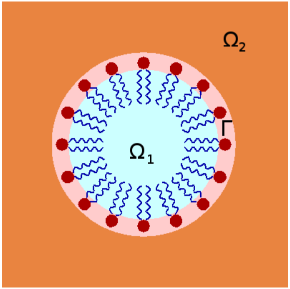
The Trace Finite Element Method for PDEs on Surfaces

T. Jawecki, M. Wess

June 25, 2015

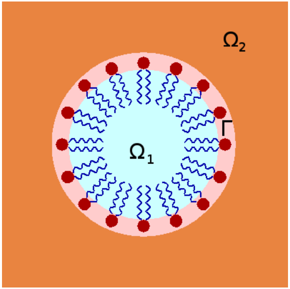
Table of Contents

General setting



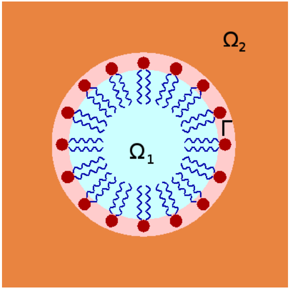
- two fluids Ω_1, Ω_2 contained in an open domain, seperated by Γ

General setting



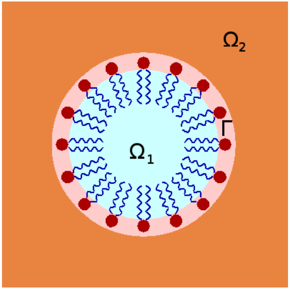
- two fluids Ω_1, Ω_2 contained in an open domain, seperated by Γ
- given velocity field $\vec{v}$

General setting



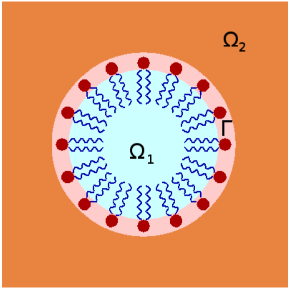
- two fluids Ω_1, Ω_2 contained in an open domain, seperated by Γ
- given velocity field $\vec{v}$
- concentration c of surfactant agent on Γ

General setting



- two fluids Ω_1, Ω_2 contained in an open domain, seperated by Γ
- given velocity field $\vec{v}$
- concentration c of surfactant agent on Γ

General setting



- two fluids Ω_1, Ω_2 contained in an open domain, seperated by Γ
- given velocity field $\vec{v}$
- concentration c of surfactant agent on Γ

Problem

For given initial data c_0, model the evolution of c.

Surface gradient and divergence

- C^2-hypersurface $\Gamma \subseteq \mathbb{R}^d$, outward normal $\vec{n}_\Gamma$

Surface gradient and divergence

- C^2-hypersurface $\Gamma \subseteq \mathbb{R}^d$, outward normal $\vec{n}_\Gamma$
- C^1-functions $f : \Gamma \to \mathbb{R}$, $\vec{g} : \Gamma \to \mathbb{R}^d$

Surface gradient and divergence

- C^2-hypersurface $\Gamma \subseteq \mathbb{R}^d$, outward normal $\vec{n}_\Gamma$
- C^1-functions $f : \Gamma \to \mathbb{R}$, $\vec{g} : \Gamma \to \mathbb{R}^d$

Definition

$$\nabla_\Gamma f := P \nabla f := \left(I - \vec{n}_\Gamma \vec{n}_\Gamma^T \right) \nabla f$$

$$\mathrm{div}_\Gamma \vec{g} := \nabla_\Gamma \cdot \vec{g} := \sum_{i=1}^{d} \sum_{j=1}^{d} p_{ij} \frac{\partial g_i}{\partial x_j}$$

Reynolds transport theorem on an interface

Theorem (Reynold's transport theorem on an interface)

The rate of change for a smooth function $f(x,t)$ on $W(t) \subseteq \Gamma$ with a given flux $\vec{v}$ can be described by

$$\frac{d}{dt}\int_{W(t)} f(x,t)\,ds = \int_{W(t)} \dot{f}(x,t) + f(x,t)\,\mathrm{div}_\Gamma(\vec{v})\,ds \qquad (1)$$

with the material derivative $\dot{f}$ defined as

$$\dot{f} := \frac{\partial f}{\partial t} + \vec{v}\cdot\nabla f \qquad (2)$$

Conservation of mass

- source function f

Conservation of mass

- source function f
- velocity $\vec{q}$

Conservation of mass

- source function f
- velocity $\vec{q}$

$$\frac{d}{dt}\int_{W(t)} c\,ds = -\int_{\partial W(t)} \vec{q}\cdot n_W\,d\tilde{s} + \int_{W(t)} f\,ds$$

Conservation of mass

- source function f
- velocity $\vec{q} := -\alpha \nabla_\Gamma c$

$$\frac{d}{dt}\int_{W(t)} c\, ds = -\int_{\partial W(t)} \vec{q}\cdot n_W\, d\tilde{s} + \int_{W(t)} f\, ds$$

Conservation of mass

- source function f
- velocity $\vec{q} := -\alpha \nabla_\Gamma c$

$$
\begin{aligned}
\frac{d}{dt} \int_{W(t)} c \, ds &= - \int_{\partial W(t)} \vec{q} \cdot n_W \, d\tilde{s} + \int_{W(t)} f \, ds \\
&= \int_{W(t)} \mathrm{div}_\Gamma(\alpha \nabla_\Gamma c) \, ds + \int_{W(t)} f \, ds
\end{aligned}
$$

Conservation of mass

- source function f
- velocity $\vec{q} := -\alpha \nabla_\Gamma c$

$$
\begin{aligned}
\frac{d}{dt} \int_{W(t)} c \, ds &= -\int_{\partial W(t)} \vec{q} \cdot n_W \, d\tilde{s} + \int_{W(t)} f \, ds \\
&= \int_{W(t)} \mathrm{div}_\Gamma(\alpha \nabla_\Gamma c) \, ds + \int_{W(t)} f \, ds \\
&= \int_{W(t)} \dot{c} + c \, \mathrm{div}_\Gamma(\vec{v}) \, ds \\
&= \int_{W(t)} \frac{\partial c}{\partial t} + \vec{v} \cdot \nabla c + c \, \mathrm{div}_\Gamma(\vec{v}) \, ds
\end{aligned}
$$

Model equation in strong form

since $W(t)$ was arbitrary:

Model equation in strong form

since $W(t)$ was arbitrary:

$$\frac{\partial c}{\partial t} + \vec{v} \cdot \nabla c + c \operatorname{div}_\Gamma(\vec{v}) - \operatorname{div}_\Gamma(\alpha \nabla_\Gamma c) = f. \tag{3}$$

Model equation in strong form

since $W(t)$ was arbitrary:

$$\frac{\partial c}{\partial t} + \vec{v} \cdot \nabla c + c \, \mathrm{div}_\Gamma(\vec{v}) - \mathrm{div}_\Gamma(\alpha \nabla_\Gamma c) = f. \tag{3}$$

$\vec{v}$ is tangential to Γ

Model equation in strong form

since $W(t)$ was arbitrary:

$$\frac{\partial c}{\partial t} + \vec{v} \cdot \nabla c + c \, \mathrm{div}_\Gamma(\vec{v}) - \mathrm{div}_\Gamma(\alpha \nabla_\Gamma c) = f. \tag{3}$$

$\vec{v}$ is tangential to Γ $\quad \Rightarrow \vec{v} \cdot \nabla c = P\vec{v} \cdot \nabla c = \vec{v} \cdot P\nabla c = \vec{v} \cdot \nabla_\Gamma c$

Model equation in strong form

since $W(t)$ was arbitrary:

$$\frac{\partial c}{\partial t} + \mathrm{div}_\Gamma(c\vec{v}) - \mathrm{div}_\Gamma(\alpha \nabla_\Gamma c) = f. \qquad (3)$$

Model equation in strong form

For a given (tangential) flux $\vec{v}$, a source function f and initial data c_0 find c such that

$$\frac{\partial c}{\partial t} + \mathrm{div}_\Gamma(c\vec{v}) - \mathrm{div}_\Gamma(\alpha \nabla_\Gamma c) = f. \tag{3}$$

Weak formulation

- Integration by parts holds on Γ i.e.

$$\int_\Gamma g \,\mathrm{div}_\Gamma \vec{f}\, ds = -\int_\Gamma \nabla_\Gamma g \cdot \vec{f}\, ds \tag{4}$$

Weak formulation

- Integration by parts holds on Γ i.e.

$$\int_\Gamma g \,\mathrm{div}_\Gamma \vec{f}\, ds = -\int_\Gamma \nabla_\Gamma g \cdot \vec{f}\, ds \tag{4}$$

- multiplication with a test function $w \in H^1(\Gamma)$, integration by parts yields

$$\int_\Gamma \frac{\partial c}{\partial t} w\, ds - \int_\Gamma c\vec{v} \cdot \nabla_\Gamma w\, ds + \alpha \int_\Gamma \nabla_\Gamma c \cdot \nabla_\Gamma w\, ds = \int_\Gamma fw\, ds \tag{5}$$

Discrete formulation

Problem

Find $c_h \in ??$ such that for all $w_h \in ??$

$$\int_{\Gamma_h} \frac{\partial c_h}{\partial t} w_h \, ds - \int_{\Gamma_h} c_h \vec{v} \cdot \nabla_{\Gamma_h} w_h \, ds + \alpha \int_{\Gamma_h} \nabla_{\Gamma_h} c_h \cdot \nabla_{\Gamma_h} w_h \, ds = \int_{\Gamma_h} f w_h \, ds \tag{6}$$

with Γ_h is a linearization of Γ,

Discrete formulation

Problem

Find $c_h \in V_{trace}$ such that for all $w_h \in V_{trace}$

$$\int_{\Gamma_h} \frac{\partial c_h}{\partial t} w_h \, ds - \int_{\Gamma_h} c_h \vec{v} \cdot \nabla_{\Gamma_h} w_h \, ds + \alpha \int_{\Gamma_h} \nabla_{\Gamma_h} c_h \cdot \nabla_{\Gamma_h} w_h \, ds = \int_{\Gamma_h} f w_h \, ds \tag{6}$$

with Γ_h is a linearization of Γ, $V_{trace} := V_h|_{\Gamma_h}$ for the space of piecewise linears V_h

Time discretization

for time discretization we use an implicit Euler method i.e. we have to solve in each time-step:

$$\left(\frac{1}{\Delta t}M - D + S\right)\Delta\vec{c}_k = \vec{f} - \left(\frac{1}{\Delta t}M - D + S\right)\vec{c}_k \tag{7}$$

where for basis functions ψ_i

$$M := (m_{ij}) := \int_{\Gamma_h} \psi_i \psi_j \, ds \tag{8}$$

$$D := (d_{ij}) := \int_{\Gamma_h} \psi_i \vec{v} \cdot \nabla_\Gamma \psi_j \, ds \tag{9}$$

$$S := (d_{ij}) := \alpha \int_{\Gamma_h} \nabla_\Gamma \psi_i \cdot \nabla_\Gamma \psi_j \, ds \tag{10}$$

Implementation

We need...

- FEspace

Implementation

We need...

- FEspace
- Integrators

Implementation

We need...

- FEspace
- Integrators
- numproc for instationary part

Implementation

We need...

- FEspace
- Integrators
- numproc for instationary part
- Output

tracemass.pde

```
... #load gometry, mesh, define constants
define coefficient lset  #interface description as zero-level
( sqrt(x*x+y*y) - R),
define fespace fesh1 -type=h1ho -order=1
define fespace tracefes -type=xfespace -type_std=h1ho -ref_space=1
numproc informxfem npix -xfespace=tracefes -fespace=fesh1 -coef_levelset=lset
gridfunction u -fespace=tracefes
bilinearform a -fespace=tracefes -symmetric
tracemass 1.0

linearform f -fespace=tracefes
tracesource (x)

define preconditioner c -type=local -bilinearform=a -test #-block
numproc bvp npbvp -gridfunction=u -bilinearform=a -linearform=f -solver=cg -preconditioner=c -maxsteps=1000 -prec=1e-6
bilinearform evalu -fespace=tracefes -nonassemble
exttrace 1.0

numproc drawflux npdf -solution=u -bilinearform=evalu -applyd -label=u
numproc markinterface npmi -fespace=tracefes
```

FEspace

We use "xfespace"

FEspace

We use "xfespace"

- like "xstdfespace" from lecture

FEspace

We use "xfespace"

- like "xstdfespace" from lecture
- just enrichment functions, no standard functions

traceintegrators.cpp

```
    const XFiniteElement * xfe =
      dynamic_cast<const XFiniteElement *> (&base_fel);

    elmat = 0.0;
    if (!xfe) return;

    const ScalarFiniteElement<D> & scafe =
      dynamic_cast<const ScalarFiniteElement<D> & > (xfe->GetBaseFE
    ());

    int ndof = scafe.GetNDof();
    FlatVector<> shape(ndof,lh);
    FlatMatrixFixWidth<D> dshape(ndof,lh);
    FlatMatrixFixWidth<D> proj(ndof,lh);
    const FlatXLocalGeometryInformation & xgeom(xfe->
    GetFlatLocalGeometry());
    const FlatCompositeQuadratureRule<D> & fcompr(xgeom.
    GetCompositeRule<D>());
    const FlatQuadratureRuleCoDim1<D> & fquad(fcompr.
    GetInterfaceRule());
```

traceintegrators.cpp contd.

```
for (int i = 0; i < fquad.Size(); ++i)
{
  IntegrationPoint ip(&fquad.points(i,0),0.0);
  MappedIntegrationPoint<D,D> mip(ip, eltrans);
  Vec<D> convvec;
  conv->Evaluate(mip,convvec);

  Mat<D,D> Finv = mip.GetJacobianInverse();
  const double absdet = mip.GetMeasure();

  Vec<D> nref = fquad.normals.Row(i);
  Vec<D> normal = absdet * Trans(Finv) * nref ;
  double len = L2Norm(normal);
  normal /= len;
  const double weight = fquad.weights(i) * len;

  scafe.CalcShape (mip.IP(),shape);
  scafe.CalcMappedDShape(mip, dshape);
  proj = dshape * (Id<D>() - normal * Trans(normal)) ;
  elmat -= weight * proj * convvec * Trans(shape);
}
```

Output

- For 2D output: standard Ngsolve output with

Output

- For 2D output: standard Ngsolve output with

```
bilinearform evalu -fespace=tracefes -nonassemble
exttrace 1.0
numproc drawflux npdf -solution=u -bilinearform=evalu -applyd -label=u
```

Output

- For 2D output: standard Ngsolve output with

```
bilinearform evalu -fespace=tracefes -nonassemble
exttrace 1.0
numproc drawflux npdf -solution=u -bilinearform=evalu -applyd -label=u
```

- For 3D output: VTK and Paraview

traceoutput.cpp

```cpp
XFESpace & xfes = * dynamic_pointer_cast<XFESpace>(gfu->GetFESpace());
int ne = gfu->GetMeshAccess()->GetNE();
for ( int i : Range(ne))
{
        if (!xfes.IsElementCut(i)) continue;
        HeapReset hr(lh);

        ElementTransformation & eltrans = gfu->GetMeshAccess()->GetTrafo (i, 0, lh);
        const FiniteElement & fel = xfes.GetFE (i, lh);
        const XFiniteElement & xfe =
          dynamic_cast<const XFiniteElement &> (fel);
        const ScalarFiniteElement<3> & scafe =
          dynamic_cast<const ScalarFiniteElement<3> & > (xfe.GetBaseFE());

        int ndof = scafe.GetNDof();
        FlatVector<> shape(ndof,lh);

        Array<int> dnums (ndof, lh);
        xfes.GetDofNrs (i, dnums);
        FlatVector<> elvec(ndof,lh);
        gfu->GetVector().GetIndirect (dnums, elvec);

        int offset = points.Size();
```

traceoutput.cpp contd

```
      for ( auto ip : ref_vertices)
          {
            MappedIntegrationPoint<3,3> mip(ip, eltrans);
            points.Append(mip.GetPoint());
            values_lset.Append(coef_lset->Evaluate(mip));

            scafe.CalcShape (mip.IP(),shape);
            values_tracesol.Append(InnerProduct(shape,elvec));
          }

          for ( auto tet : ref_tets)
          {
            INT<4> new_tet = tet;
            for (int i = 0; i < 4; ++i)
              new_tet[i] += offset;
            cells.Append(new_tet);
          }

        }
        PrintPoints();
        PrintCells();
        PrintCellTypes();
        PrintFieldData();

```

traceoutput.cpp contd.

```
void PrintPoints()
{
  *fileout << "POINTS " << points.Size() << " float" << endl;
  for (auto p : points)
    *fileout << p << endl;
}

void PrintCells()
{
  *fileout << "CELLS " << cells.Size() << " " << 5 * cells.Size() << endl;
  for (auto c : cells)
    *fileout << "4 " << c << endl;
}
```

traceoutput.cpp contd.

```
void PrintCellTypes()
{
  *fileout << "CELL_TYPES " << cells.Size() << endl;
  for (auto c : cells)
    *fileout << "10 " << endl;
  *fileout << "CELL_DATA " << cells.Size() << endl;
  *fileout << "POINT_DATA " << points.Size() << endl;
}

void PrintFieldData()
{
  *fileout << "FIELD FieldData 2"<< endl;

  *fileout << "levelset 1 " << values_lset.Size() << " float" << endl;
  for (auto v : values_lset)
    *fileout << v << " ";
  *fileout << endl;

  *fileout << "solution 1 " << values_tracesol.Size() << " float" << endl;
  for (auto v : values_tracesol)
    *fileout << v << " ";
  *fileout << endl;
}
```

traceinstat.cpp

```cpp
const BaseMatrix & mata = bfa->GetMatrix();
const BaseMatrix & matm = bfm->GetMatrix();
const BaseVector & vecf = lff->GetVector();
BaseVector & vecu = gfu->GetVector();
auto summat = matm.CreateMatrix();
auto d = vecu.CreateVector();
auto w = vecu.CreateVector();

double per=1;
summat->AsVector() = (1.0/dt) * matm.AsVector() + mata.AsVector();
auto invmat = summat->InverseMatrix();

for (double t = 0; t <= tend; t += dt)
{
   if (periodicrhs)
   per=cos(t);
   cout << "t = " << t << endl;
   d = per * vecf - mata * vecu;
   w = *invmat * d;
   vecu += w;
   npto->Do(lh);
   Ng_Redraw ();
}
```

Rotating circle/sphere

- 2D:

$$\vec{v} := (-y, x)^T \tag{11}$$

$$c_0 := x \tag{12}$$

Rotating circle/sphere

- 2D:

$$\vec{v} := (-y, x)^T \tag{11}$$
$$c_0 := x \tag{12}$$

- 3D:

$$\vec{v} := (-z, 0, x)^T \tag{13}$$
$$c_0 := \exp(50\,(x - R)) \tag{14}$$
$$f := 5\,(\cos(7t) + 1)\exp(50\,(x - R)) \tag{15}$$

run movie

Two phase stokes flow

- Bubble rising with velocity v_0

Two phase stokes flow

- Bubble rising with velocity v_0
- viscosities μ_1, μ_2

Two phase stokes flow

- Bubble rising with velocity v_0
- viscosities μ_1, μ_2
- solution of Stokes equations:

$$\vec{v} = \frac{1}{2} \frac{\mu_2}{\mu_1 + \mu_2} v_0 \sin \theta \vec{e}_\theta \tag{16}$$

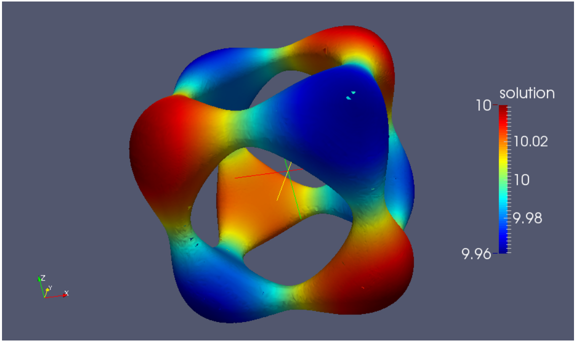
Two phase stokes flow

- Bubble rising with velocity v_0
- viscosities μ_1, μ_2
- solution of Stokes equations:

$$\vec{v} = \frac{1}{2}\frac{\mu_2}{\mu_1 + \mu_2} v_0 \sin\theta \vec{e}_\theta \tag{16}$$

- run movie

awesome torus



References

- S. Gross and A. Reusken: Numerical Methods for Two-phase Incompressible Flows, Springer 2011
- C. Lehrenfeld: Extenden finite element methods for interface problems, lecture notes, TU Wien 2015