FEniCS: Automated FEM

Johan Jansson

April 7, 2011

Table of contents

Who we are and what we do

Computational Technology Laboratory (CTL):

Adaptive FEM/software for continuum mechanics/high Re

Johan Hoffman - Group leader
: Turbulent incompressible flow/geometry

Johan Jansson - Researcher
: Unicorn, turbulent flow/FSI

Jeanette Spuhler - Phd student
: Heart simulation

Rodrigo Vilela de Abreu - Phd student
: Aeroacoustics

Niclas Jansson - Phd student
: Massively parallel adaptive FEM (DOLFIN-HPC)

Cem Degirmenci - Phd student
: FSI, mesh algorithms

Murtazo Nazarov - Phd student
: Compressible flow

Research and software development of Unicorn/DOLFIN/FEniCS

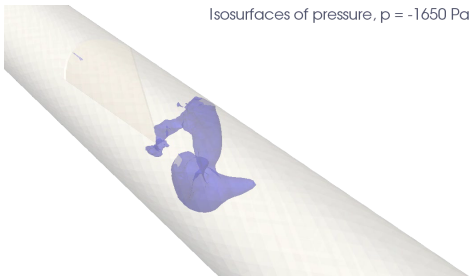
`http://ctl.csc.kth.se`

Unicorn: Aim

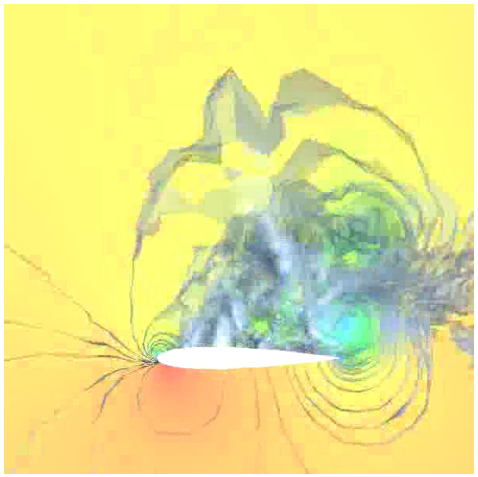
Automated high-performance mathematical modeling of industrial/realistic continuum mechanics

- Automated/canonical Unified Continuum model
- Automated weak form evaluation/tensor assembly
- Automated duality-based error control/adaptivity
- Strong scaling on massively parallel hardware

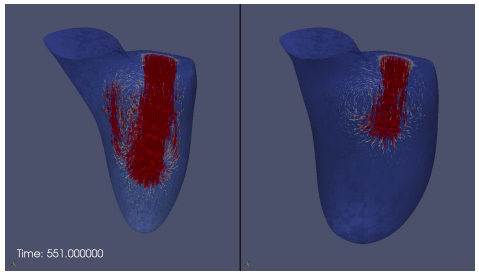
Aeroacoustics



Flight simulation



Biomechanics



Unified Continuum (UC) formulation

Conservation equations (momentum, mass)
in Euler (laboratory) coordinates, stress σ as data:

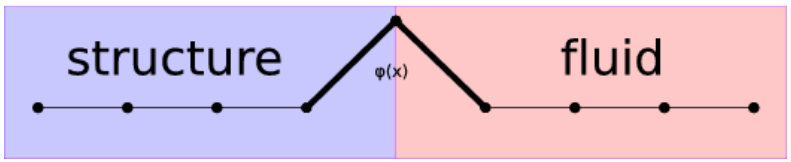
$$\rho(D_t u_i + D_{x_j} u_j u_i) - D_{x_j}\sigma_{ij} - D_{x_i} p - f = 0$$
$$D_{x_j} u_j = 0$$
$$D_t\theta + D_{x_j} u_j \theta = 0$$

Different constitutive equations for phases:

$$\sigma = \theta\sigma_f + (1-\theta)\sigma_s$$

$$D_t\sigma_s = 2\mu_s\epsilon + \nabla u\sigma_s + \sigma_s\nabla u^\top \qquad \sigma_f = 2\mu_f\epsilon$$

Discretize as one continuum/domain:

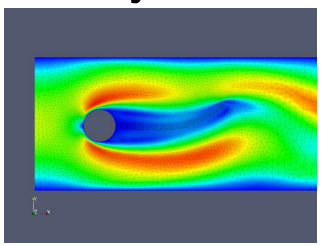


Unified Continuum (UC) G2/FEM discretization

Unicorn/FEniCS simulator with G2 (General Galerkin):

- Can use simple elements (piecewise linear)
- Least-squares/streamline diffusion type stabilization (necessary for convection dominated flow).
- ALE map (tilted space-time elements)
- Elastic smoothing/local mesh modification (MAdLib, [Compere, Remacle, Jansson, Hoffman 2009])
- Model turbulent dissipation by adaptive residual-based DNS/LES.

2D dynamic FSI benchmark:



Within 2% of reference in mesh convergence study
[Hoffman, Jansson, Stockli, M3AS 2010]

Goal of FEniCS: Automation of solution of PDE by FEM

- One goal: automatic solution of PDE
- Why?
  - Efficiency
    - Execution time
    - Development time
    - Complexity is high, so manual efficient implementation typically not possible.
    - Implement automated kernel thoroughly (efficiency, parallel) - can solve all problems vs. one specialized solver for every problem: need to repeat optimization/verification for every problem.
  - Safety - reduce chance of human error (bugs)

Unicorn/FEniCS components

FEniCS: `www.fenicsproject.org` Open source software project for automated solution of PDE with many involved universities/individuals

- Automated generation of finite elements/basis functions (FIAT)

$$e = (K, P, \mathcal{N})$$

- Automated evaluation of variational forms (FFC)

$$a(v, u) = \int_\Omega \nabla v \cdot \nabla u dx$$

- Automated assembly of discrete systems (DOLFIN)

$$A = 0$$
$$\textit{for all elements } K \in \mathcal{T}_\Omega$$
$$\quad A \mathrel{+}= A^K$$

- Automated Unified Continuum modeling (Unicorn)

$$L(v) = R(U, \sigma, \theta) = (D_t U_i + D_{x_j} U_j U_i) - D_{x_j} \sigma_{ij} - f, v)$$

Component dependencies

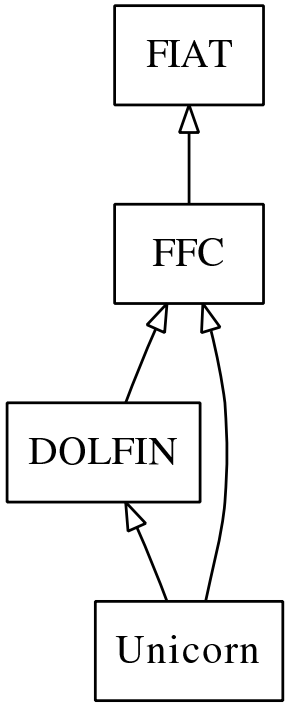
Unicorn
- UC forms
- `TimeDependentPDE`
- ErrorEstimate
- coefficients (eg. stabilization)
- mesh adaptation/smoothing

DOLFIN
- mesh representation
- global tensor assembly,
- Linear algebra interface
- load balancing

FFC
- form evaluation code generation

FIAT
- basis function representation

Dependencies



Basic FEniCS Poisson demo

```
from dolfin import *

# Create mesh and define function space
mesh = UnitSquare(32, 32)
V = FunctionSpace(mesh, "CG", 1)

# Define Dirichlet boundary (x = 0 or x = 1)
def boundary(x):
    return x[0] < DOLFIN_EPS or x[0] > 1.0 - DOLFIN_EPS

# Define boundary condition
u0 = Constant(0.0)
bc = DirichletBC(V, u0, boundary)

# Define variational problem
v = TestFunction(V)
u = TrialFunction(V)
f = Expression("10*exp(-(pow(x[0] - 0.5, 2) + pow(x[1] - 0.5, 2)) / 0.02)")
g = Expression("sin(5*x[0])")
a = inner(grad(v), grad(u))*dx
L = v*f*dx - v*g*ds

# Compute solution (assemble matrix/vector, solve linear system)
problem = VariationalProblem(a, L, bc)
u = problem.solve()

# Plot solution
plot(u, interactive=True)
```

Automated matrix/vector assembly I

In general the matrix A_h, representing a bilinear form

$$a(u, v) = (A(u), v),$$

is given by

$$(A_h)_{ij} = a(\varphi_j, \hat{\varphi}_i).$$

and the vector b_h representing a linear form

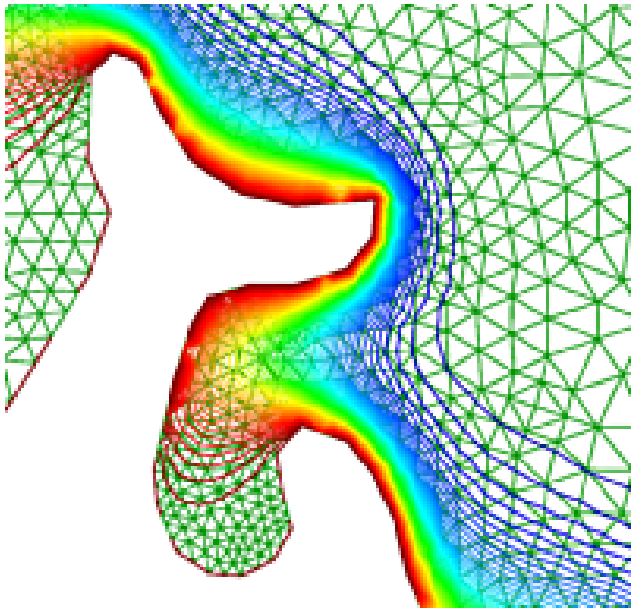
$$L(v) = (f, v),$$

is given by

$$(b_h)_i = L(\hat{\varphi}_i).$$

Example (Poisson 1D):
$a(u, v) = (u', v') = \int_0^1 u'v'dx, \quad (A_h)_{ij} = a(\phi_j, \phi_i) = \int_0^1 \phi_j'\phi_i'dx$
$L(v) = (f, v) = \int_0^1 fvdx, \quad (b_h)_i = L(\phi_i) = \int_0^1 f\phi_i dx$

Automated matrix/vector assembly II



Noting that $a(v,u) = \sum_{K \in \mathcal{T}} a_K(v,u)$, the matrix A can be assembled by

$A = 0$

for all elements $K \in \mathcal{T}$

A += A^K

The *element matrix* A^K is defined by

$$A_{ij}^K = a_K(\hat{\phi}_i, \phi_j)$$

for all local basis functions $\hat{\phi}_i$ and ϕ_j on K

Automated matrix/vector assembly III

for all elements $K \in \mathcal{T}$

  for all test functions $\hat{\varphi}_i$ on K

    for all trial functions φ_j on K

      1. Compute $I = a(\varphi_j, \hat{\varphi}_i)_K$
      2. Add I to $(A_h)_{ij}$

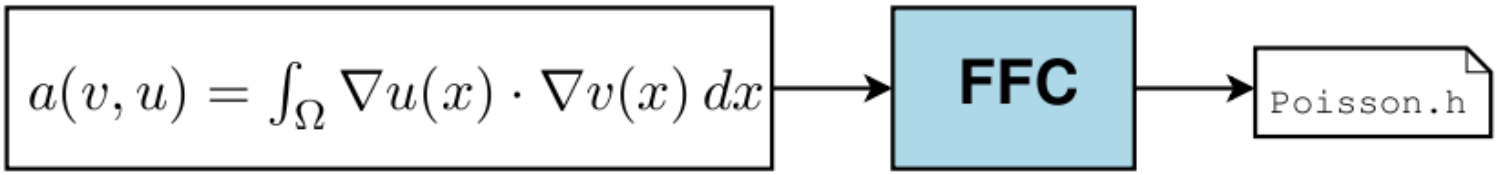
    end

  end

end

Form compilation/code generation

- Automates a key step in the implementation of finite element methods for partial differential equations
- Input: a variational form and a finite element
- Output: C/C++ function for element tensor




```
>> ffc [-l language] poisson.form
```

Generated quadrature code

```
virtual void tabulate_tensor(double* A, const double* const* w,
      const ufc::cell& c) const
{
...
  // Quadrature weight
  const static double W0 = 0.5;
  // Tabulated basis functions and arrays of non-zero columns
const static double Psi_w[1][3] =\
  {{0.33333333333, 0.33333333333, 0.33333333333}};
 const static double Psi_vu[1][2] = {{-1, 1}};
 static const unsigned int nzc0[2] = {0, 1};
 static const unsigned int nzc1[2] = {0, 2};
 // Geometry constants
 const double G0 = Jinv_00*Jinv_10*W0*det;
 const double G1 = Jinv_01*Jinv_11*W0*det;
 const double G2 = Jinv_00*Jinv_00*W0*det;
 const double G3 = Jinv_01*Jinv_01*W0*det;
 const double G4 = Jinv_10*Jinv_10*W0*det;
 const double G5 = Jinv_11*Jinv_11*W0*det;
 // Loop integration points
 for (unsigned int ip = 0; ip < 1; ip++)
 {
   // Compute function value
   double F0 = 0;
   for (unsigned int r = 0; r < 3; r++)
     F0 += Psi_w[ip][r]*w[0][r];
   const double Gip0 = (G0 + G1)*F0;
   const double Gip1 = (G2 + G3)*F0;
   const double Gip2 = (G4 + G5)*F0;
   for (unsigned int i = 0; i < 2; i++)
   {
     for (unsigned int j = 0; j < 2; j++)
     {
       A[nzc0[i]*3 + nzc0[j]] += Psi_vu[ip][i]*Psi_vu[ip][j]*Gip1;
       A[nzc0[i]*3 + nzc1[j]] += Psi_vu[ip][i]*Psi_vu[ip][j]*Gip0;
       A[nzc1[i]*3 + nzc0[j]] += Psi_vu[ip][i]*Psi_vu[ip][j]*Gip0;
       A[nzc1[i]*3 + nzc1[j]] += Psi_vu[ip][i]*Psi_vu[ip][j]*Gip2;
     }
   }
 }
}
```

UC Momentum form (FFC language)

```
Sf = mult(P, Identity(d)) - mult(nu, grad(UP))
Ss = mult(P, Identity(d)) - mult(1.0, sigmaM)
S = mult(phi, Sf) + mult(1.0 - phi, Ss)

def f(u, v):
    return -(dot(ugradu(U_ALE, u), v) - dot(S, grad(v))) + \
        -dot(mult(d2, div(u)), div(v)) + \
        -mult(d1, dot(ugradu(U_ALE, u), ugradu(U_ALE, v))) + \
         dot(mult(1.0 - phi, ff), v)

def dfdu(u, k, v):
    return -dot(ugradu(U_ALE, u), v) + \
        -mult(1 - phi, mult(k, dot(E(epsilon(u), mu, lmbda), grad(v)))) + \
        -mult(phi, mult(nu, dot(grad(u), grad(v)))) + \
        -mult(d2, dot(div(u), div(v))) + \
        -mult(d1, dot(ugradu(U_ALE, u), ugradu(U_ALE, v)))

# cG(1)
def F(u, u0, k, v):
    uc = mult(0.5, u + u0)
    return (-dot(u, v) + dot(u0, v) + mult(k, f(uc, v)))

def dFdu(u, u0, k, v):
    uc = mult(0.5, u)
    return (-dot(u, v) + mult(k, dfdu(uc, k, v)))

a = (dFdu(U1, U0, k, v)) * dx
L = (dFdu(UP, U0, k, v) - F(UP, U0, k, v)) * dx
```

FIAT: FInite element Automatic Tabulator

- Automates the generation of finite element basis functions
- Simplifies the specification of new elements
- Continuous and discontinuous Lagrange elements of arbitrary order
- Crouziex-Raviart (CR) elements
- Nedelec elements
- Raviart-Thomas (RT) elements
- Brezzi-Douglas-Marini (BDM) elements
- ...

FIAT: Implementation

- Use orthonormal basis on triangles and tets (Dubiner)
- Express basis functions in terms of the orthonormal basis
- Translate conditions of function space into linear algebraic relations
- Implemented in Python (as a Python module)

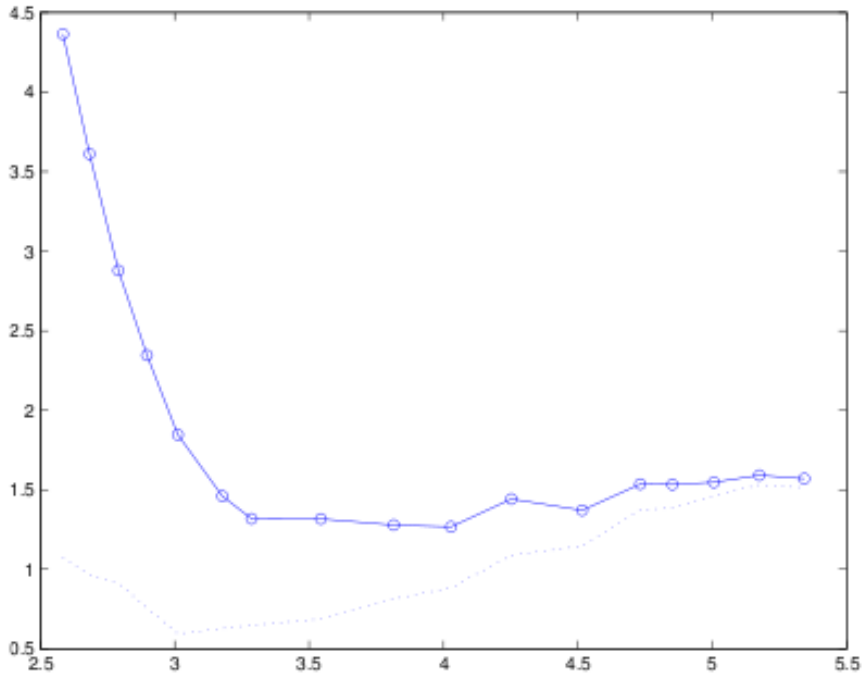
FIAT: Example

```
>>> from FIAT.Lagrange import *
>>> from FIAT.shapes import *
>>> element = Lagrange(TRIANGLE, 2)
>>> basis = element.function_space()
>>> v = basis[0]
>>> v([-1.0, -1.0])
1.0
```

(note reference triangle starts at (-1, -1) in FIAT)

Previous results on adaptive error control for turbulent flow

Surface-mounted cube: drag coeff. converges to 5% variation from reference result (same as experiment)



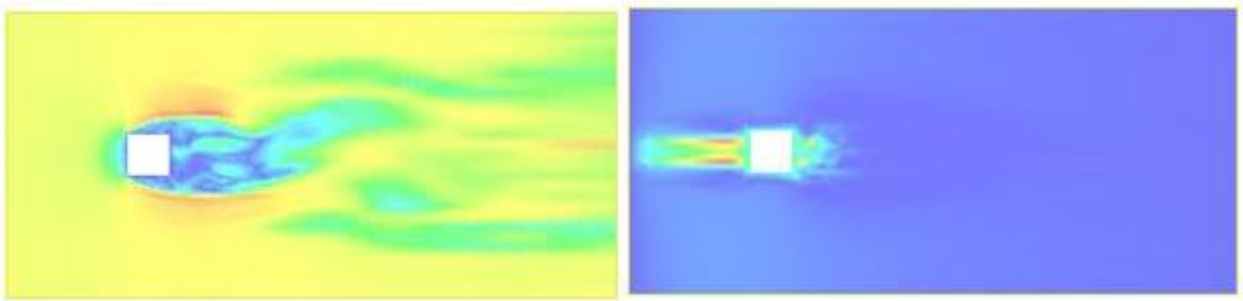
[Hoffman, 2005], [Hoffman/Johnson, 2007]
More reference results..

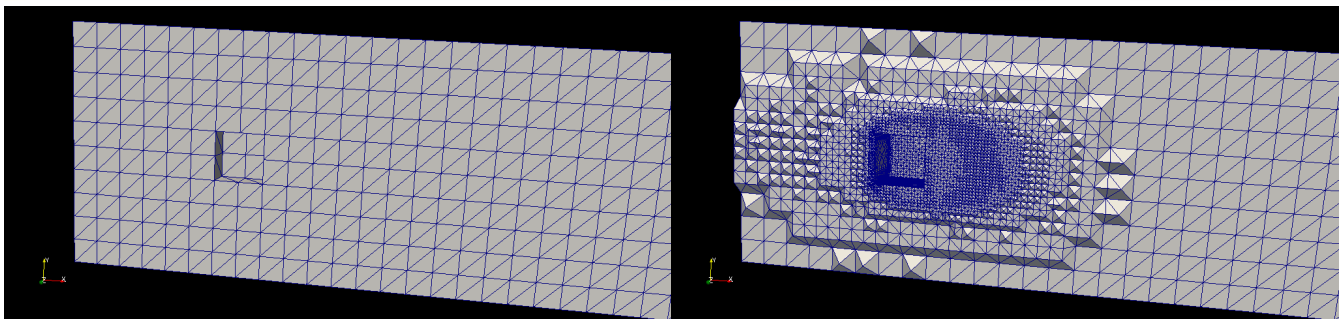
Adaptive error control: output

Abstract error estimate using duality:

$$(e, \psi) = (e, A^* \phi) = (Ae, \phi) = (-R(W), \phi)$$

Implemented in Unicorn as `ErrorEstimate`





Parallel software architecture

Design parallel algorithms for basic primitives, based on MPI and distributed mesh (ghost points define communication):

Assembly loop
DOLFIN-HPC (Parallel matrix representation with PETSc)

Mesh refinement
Unicorn-HPC: Parallel recursive Rivara

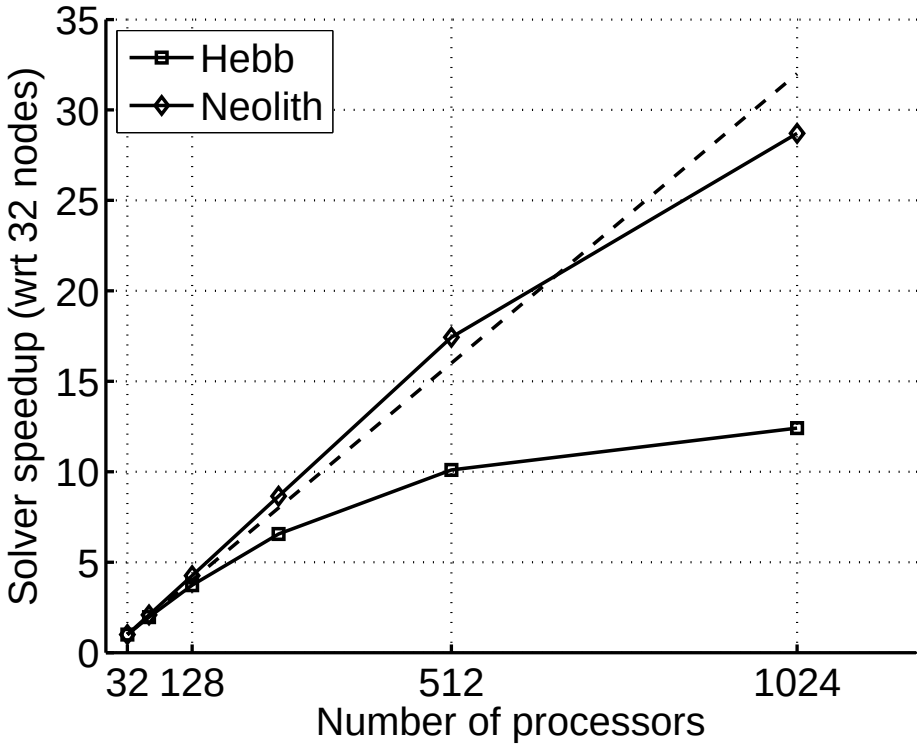
Linear solvers
PETSc: Parallel Krylov

Load balancing
DOLFIN-HPC: Remapping (PLUM)

Partitioning
ParMetis

Strong scaling verification

Full adaptive solve of incompressible flow up to 1024 processors



[Jansson, Hoffman, Jansson, 2010 Preprint]

Scientific method and source code

Construct falsifiable model

Intuition, symbolic manipulation, software construction

Reproduce/Falsify

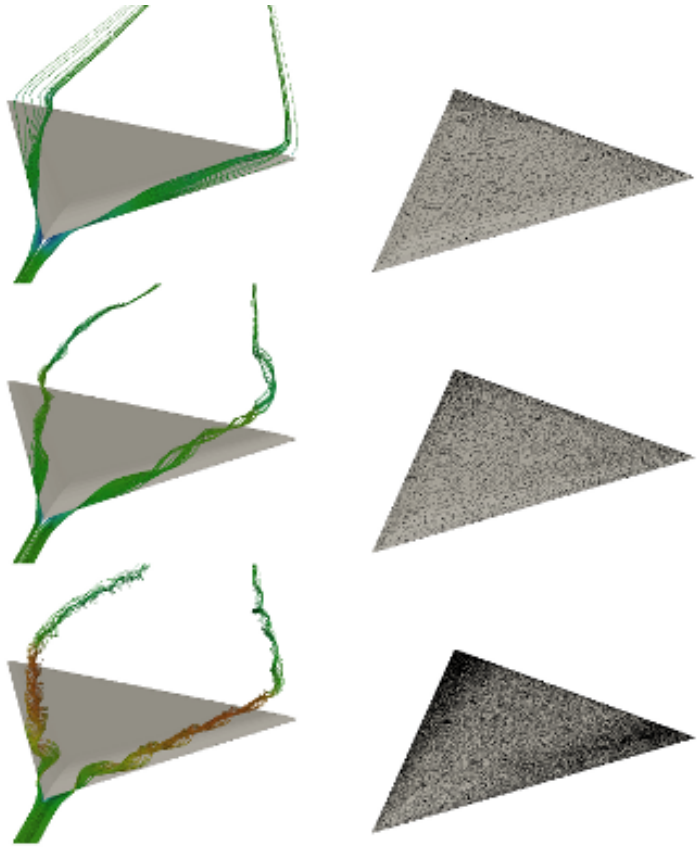
Other scientists reproduce and try to falsify

Extend

Build on previous work to construct new models

Source code and formal mathematical system are equivalent [Turing 1937].
High complexity $\Rightarrow$ High disclosure
Make results easily falsifiable/reproducible $\Rightarrow$ stronger science

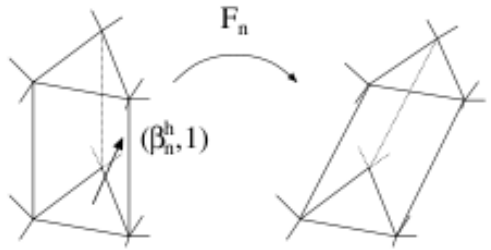
Parallel duality-based adaptivity for delta wing



Ref: Niclas Jansson

ALE map arising from FEM with moving basis

Map between reference and "tilted" space-time element:



$$
\begin{aligned}
D_t\phi(t,\bar{x}) &= \hat{v}(t,\bar{x}) \\
(x,t) &= \phi(\bar{x},t) \\
\int_K u(t,x)dxdt &= \int_{\bar{K}} \bar{u}(t,\phi(t,\bar{x}))|detJ|d\bar{x}dt \\
D_t u(x) + u\cdot\nabla u &= D_t\bar{u}(\bar{x}) + (\bar{u}-\hat{v})\cdot\nabla\bar{u})
\end{aligned}
\tag{1}
$$

Choosing mesh vel. $\hat{v} = U$ in solid part $\Rightarrow$ phase equation becomes trivial: $D_t\Theta = 0$.

Elastic smoothing

Elastic model (variant of FSI solid model):

$$\dot{v} = \nabla \cdot \sigma \quad \text{for } x \in \Omega(t) \quad \text{(momentum conservation)}$$

$$F = \frac{\partial x}{\partial X} \quad \text{(deformation)}$$

$$\dot{F^{-1}} = -F^{-1}\nabla v \quad \text{(deformation)}$$

$$\sigma = E(e) \quad \text{(stress (Hooke))}$$

$$e = \frac{1}{2}(I - F^{-\top}F^{-1}) \quad \text{(strain)}$$

Set deformation from reference cell as initial value:

$$F_0 = \bar{F}\bar{h}$$

Local mesh modification

Local mesh modification (MAdLib) (edge split/collapse/swap) as basis for 2D/3D mesh algorithms:
Applications:

- Mesh refinement/coarsening [Rivara, GRUMMP]
- Mesh motion (+ elastic smoothing)
- Mesh quality optimization [Barry]
- (Mesh generation)

Developed by Gaetan Compere and Jean-Francois Remacle (Gmsh developers) at Louvain, Belgium.

`http://www.madlib.be`

[Compere, Remacle, Marchandise, Proc. 17th International Meshing Roundtable, 2008]

Visualization of mesh adaptation

Conclusions

- Automated FEM
- Unicorn: UC + automated discretization as part of FEniCS
- Verified UC model for FSI against 2D benchmark
- Overview of previous results for convergence to turbulent benchmarks
- Demonstrated strong linear scaling for parallel implementation
- Demonstrated convergence of duality-based adaptive error control for turbulent aeroacoustic model problem with fixed mixer plate

`ctl.csc.kth.se`
`fenicsproject.org`