# PDE-constrained optimisation in Hilbert spaces

**S.W. Funke and M. Nordass**

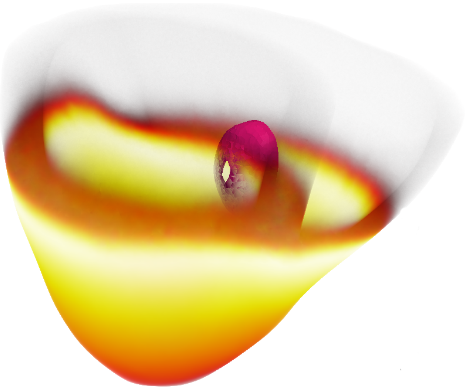
Simula Research Laboratory

**13th August 2014**

# Overview

- Motivation
- Solving a model optimisation problem
- Optimisation algorithms in Hilbert spaces
- Summary

# Treating abnormal cardiac activity: How to find the optimal region to treat atrial fibrillation by ablation?

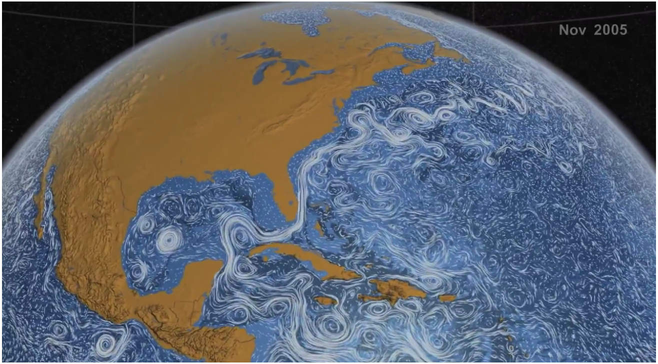


[0]Sjur Gjerald, Johan Hake and Marie Rognes

# Finding the optimal number of placement of tidal stream turbines

[0] OpenTidalFarm

# Recovering the initial ocean/atmosphere state




[0]MITGcm

# These problems have a common structure

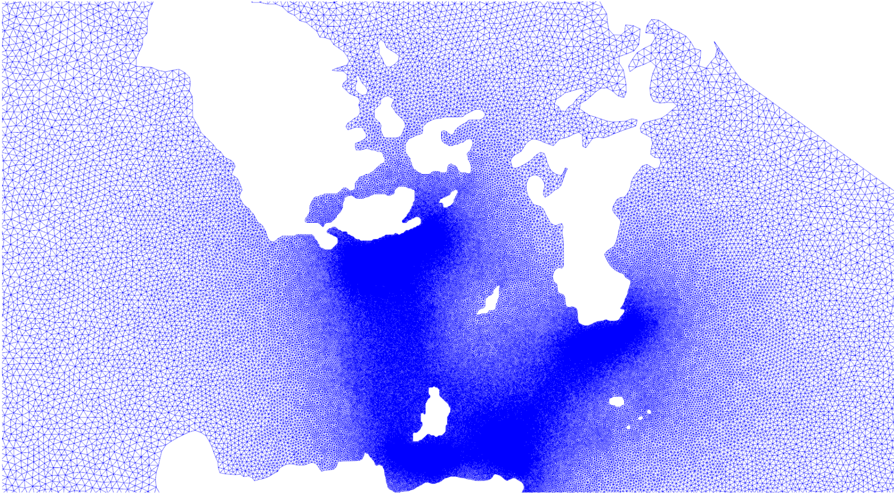
$$\min_{u,m} J(u, m)$$

where

- $J \in (V \times W) \to \mathbb{R}$ is the objective functional,
- $m \in W$ is a control function,
- $u \in V$ is the solution of the PDE

$$F(u, m) = 0.$$

# Mesh-independent convergence is crucial for large-scale applications



# In this talk we use the optimal heating problem as a model problem

$$\min_{u,m} \frac{1}{2}\|u - d\|^2_{L^2(\Omega)} + \frac{\alpha}{2}\|m\|^2_{L^2(\Omega)}$$

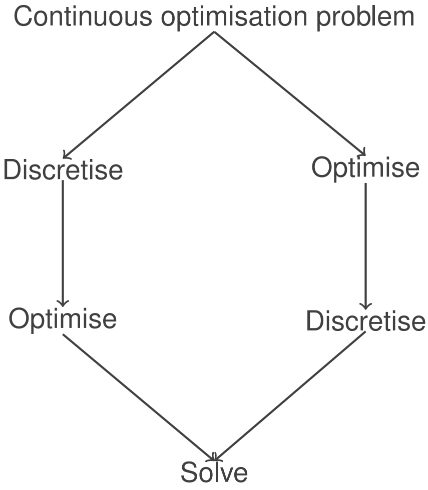
where

- $m \in L^2(\Omega)$ is the heating power
- $u \in H^1_0(\Omega)$ is the weak solution to:

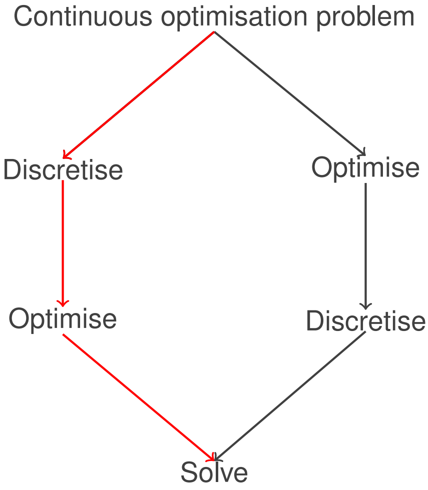
$$-\Delta u = m$$

- $d \in L^2(\Omega)$ is the desired temperature profile
- $\alpha \in \mathbb{R}$ is a regularisation parameter

# There are two strategies for solving the optimisation problem



# There are two strategies for solving the optimisation problem



# Step 1: Discretise the model problem

Continuous problem:

$$\min_{(u,m)\in L^2(\Omega)\times H_0^1(\Omega)} \frac{1}{2}\|u-d\|^2_{L^2(\Omega)} + \frac{\alpha}{2}\|m\|^2_{L^2(\Omega)}$$

$$\text{subject to:}$$

$$-\Delta u = m \text{ in } \Omega$$

Discretised problem:

$$\min_{(\mathbf{u},\mathbf{m})\in\mathbb{R}^{|u|+|m|}} \frac{1}{2}(\mathbf{u}-\mathbf{d})^T M_u (\mathbf{u}-\mathbf{d}) + \frac{\alpha}{2}\mathbf{m}^T M_m \mathbf{m}$$

$$\text{subject to:}$$

$$D\mathbf{u} = M\mathbf{m}$$

where $M$ are mass matrices and $D$ is the discrete Laplace operator.

# Step 2: Derive and solve the optimality conditions

Form the Lagrangian:

$$\begin{aligned}\mathcal{L}(\mathbf{u}, \mathbf{m}, \boldsymbol{\lambda}) =& \frac{1}{2} (\mathbf{u} - \mathbf{d})^T M_u (\mathbf{u} - \mathbf{d}) + \frac{\alpha}{2} \mathbf{m}^T M_m \mathbf{m} \\ &+ \boldsymbol{\lambda}^T (D\mathbf{u} - M\mathbf{m})\end{aligned}$$

# Step 2: Derive and solve the optimality conditions

Form the Lagrangian:

$$\begin{aligned}\mathcal{L}(\mathbf{u},\mathbf{m},\boldsymbol{\lambda}) =&\frac{1}{2}\left(\mathbf{u}-\mathbf{d}\right)^T M_u\left(\mathbf{u}-\mathbf{d}\right)+\frac{\alpha}{2}\mathbf{m}^T M_m\mathbf{m}\\ &+\boldsymbol{\lambda}^T\left(D\mathbf{u}-M\mathbf{m}\right)\end{aligned}$$

Optimality conditions:

$$\begin{aligned}\mathcal{L}_u &= M_u(\mathbf{u}-\mathbf{d})-D^T\boldsymbol{\lambda} &&= 0 &&\text{(adjoint PDE)}\\ \mathcal{L}_m &= \alpha M_m\mathbf{m}-M^T\boldsymbol{\lambda} &&= 0 &&\text{(gradient equation)}\\ \mathcal{L}_\lambda &= D\mathbf{u}-M\mathbf{m} &&= 0 &&\text{(PDE)}\end{aligned}$$

# Step 2: Derive and solve the optimality conditions

Form the Lagrangian:

$$\begin{aligned}\mathcal{L}(\mathbf{u}, \mathbf{m}, \boldsymbol{\lambda}) =& \frac{1}{2}(\mathbf{u} - \mathbf{d})^T M_u (\mathbf{u} - \mathbf{d}) + \frac{\alpha}{2}\mathbf{m}^T M_m \mathbf{m} \\ & + \boldsymbol{\lambda}^T (D\mathbf{u} - M\mathbf{m})\end{aligned}$$

Optimality conditions in matrix form:

$$\begin{bmatrix} M_u & 0 & D^T \\ 0 & \alpha M_m & -M^T \\ D & -M & 0 \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ \mathbf{m} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{d} \\ 0 \\ 0 \end{bmatrix} \qquad \begin{matrix} \text{(adjoint PDE)} \\ \text{(gradient condition)} \\ \text{(state PDE)} \end{matrix}$$

Solve with MINRES and a carefully chosen preconditioner.

# Alternative step 2: Reduce problem, then optimise

$$\min_{\mathbf{u},\, \mathbf{m} \in \mathbb{R}^{|u|+|m|}} \frac{1}{2} \left(\mathbf{u} - \mathbf{d}\right)^T M_u \left(\mathbf{u} - \mathbf{d}\right) + \frac{\alpha}{2} \mathbf{m}^T M_m \mathbf{m}$$

subject to:

$$D\mathbf{u} = M\mathbf{m}$$

# Alternative step 2: Reduce problem, then optimise

$$\min_{\mathbf{m}\in\mathbb{R}^{|m|}} \underbrace{\frac{1}{2}\left(D^{-1}M\mathbf{m}-\mathbf{d}\right)^T M_u \left(D^{-1}M\mathbf{m}-\mathbf{d}\right)+\frac{\alpha}{2}\mathbf{m}^T M_m \mathbf{m}}_{:=J(\mathbf{m})}$$

# Alternative step 2: Reduce problem, then optimise

$$\min_{\mathbf{m}\in\mathbb{R}^{|m|}} \underbrace{\frac{1}{2}\left(D^{-1}M\mathbf{m}-\mathbf{d}\right)^T M_u \left(D^{-1}M\mathbf{m}-\mathbf{d}\right)+\frac{\alpha}{2}\mathbf{m}^T M_m \mathbf{m}}_{:=J(\mathbf{m})}$$

Optimality condition:

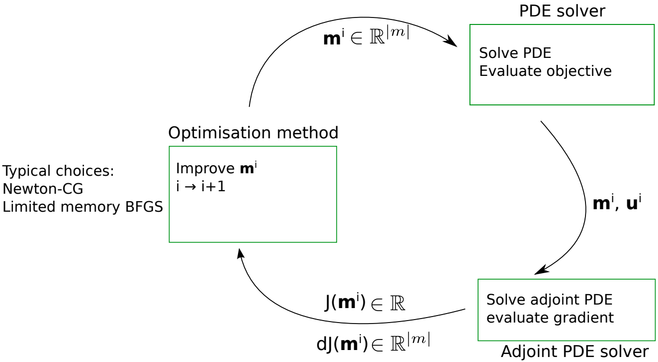
$$dJ(\mathbf{m}) = 0$$

# Alternative step 2: Reduce problem, then optimise

$$\min_{\mathbf{m}\in\mathbb{R}^{|m|}} \underbrace{\frac{1}{2}\left(D^{-1}M\mathbf{m}-\mathbf{d}\right)^T M_u \left(D^{-1}M\mathbf{m}-\mathbf{d}\right)+\frac{\alpha}{2}\mathbf{m}^T M_m \mathbf{m}}_{:=J(\mathbf{m})}$$

Optimality condition:

$$dJ(\mathbf{m}) = 0$$



# The reduced approach performs a block Gauss-Seidel like iteration on the optimality system

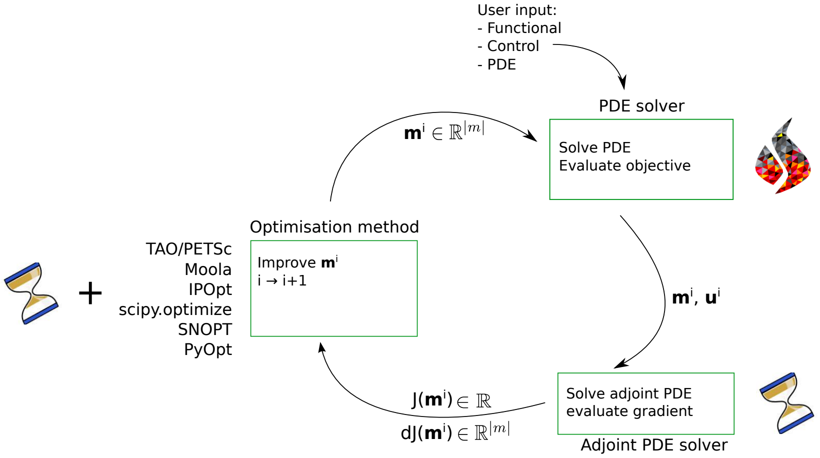
Full-space approach:

$$\begin{array}{c} (2) \\ (3) \\ (1) \end{array} \quad \begin{bmatrix} M_u & 0 & D^T \\ 0 & \alpha M_m & -M^T \\ D & -M & 0 \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ \mathbf{m} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{d} \\ 0 \\ 0 \end{bmatrix} \quad \begin{array}{c} \text{(adjoint PDE)} \\ \text{(gradient condition)} \\ \text{(state PDE)} \end{array}$$

Reduced approach:

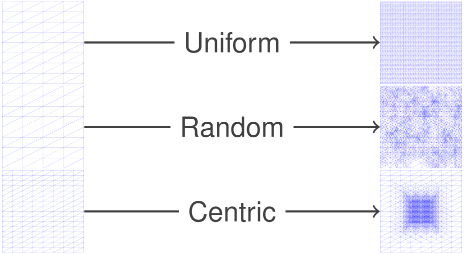
$$\mathbf{m}^0 \xrightarrow[\text{state PDE}]{(1)} \mathbf{u}^0 \xrightarrow[\text{adjoint PDE}]{(2)} \boldsymbol{\lambda}^0 \xrightarrow[\text{optimisation step}]{(3)} \mathbf{m}^1 \to \cdots$$

# FEniCS/dolfin-adjoint allows to automated most steps of the optimisation



# As a numerical experiment we solve the model problem on three meshes and with two solvers

- Solvers: Limited memory BFGS and Newton-CG.
- Meshes:




- Stopping criteria: $\|\mathrm{d}J\|_{L^2} < 10^{-7}$.
- Regularisation parameter: $\alpha = 10^{-6}$.
- Desired temperature: $d = \left[\frac{1}{2\pi^2} + 2\alpha\pi^2\right] \sin(\pi x) \sin(\pi y)$.

# BFGS yields mesh dependent iteration numbers

- Algorithm: TAO limited memory BFGS, initial Hessian approximation based on the Broyden approximation.

**Uniform refinement**

| | | | | |
|---|---|---|---|---|
| Refinement level | 0 | 1 | 2 | 3 |
| BFGS iterations | 14 | 12 | 3 | 4 |

**Random refinement**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Refinement level | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| BFGS iterations | 14 | 52 | 77 | 69 | 74 | 68 | 81 |

**Centric refinement**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Refinement level | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| BFGS iterations | 11 | 39 | 58 | 79 | 80 | 128 | 132 |

# Newton-CG yields mesh-dependent CG iterations

- Optimisation algorithm: TAO line-search Newton-CG

**Uniform refinement**

| Refinement level | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| Newton iterations | 1 | 1 | 1 | 1 |

**Random refinement**

| Refinement level | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| Newton iterations | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

**Centric refinement**

| Refinement level | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| Newton iterations | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

# Newton-CG yields mesh-dependent CG iterations

- Optimisation algorithm: TAO line-search Newton-CG

**Uniform refinement**

| | | | | |
|---|---|---|---|---|
| Refinement level | 0 | 1 | 2 | 3 |
| Newton iterations | 1 | 1 | 1 | 1 |
| CG iterations | 13 | 13 | 8 | 3 |

**Random refinement**

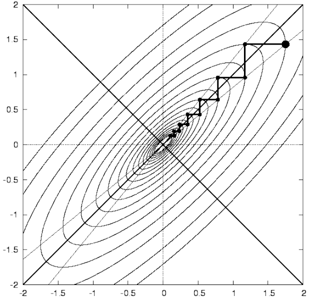
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Refinement level | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| Newton iterations | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| CG iterations | 13 | 21 | 25 | 30 | 36 | 41 | 49 |

**Centric refinement**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Refinement level | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| Newton iterations | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| CG iterations | 13 | 28 | 38 | 55 | 65 | 95 | 131 |

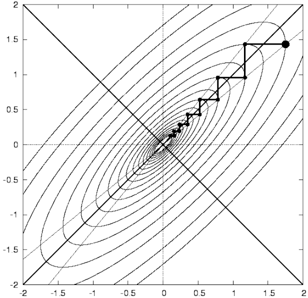
# The textbook steepest descent algorithm

$i \leftarrow 0$
while $|dJ(m^i)| < \mathrm{tol}$:
    $m^{i+1} \leftarrow m^i - \gamma dJ(m^i)$
    $i \leftarrow i + 1$



# The textbook steepest descent algorithm

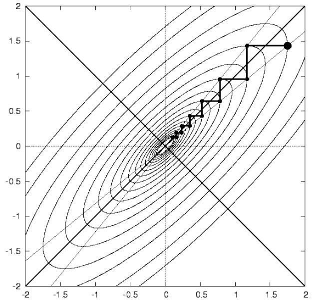
$i \leftarrow 0$
while $|dJ(m^i)| < \mathrm{tol}$:
$\quad m^{i+1} \leftarrow m^i - \gamma dJ(m^i)$
$\quad i \leftarrow i + 1$




But:

- $m^i \in W$
- $J : W \to \mathbb{R}$
- $dJ(m^i) : W \to \mathbb{R} \in W^*$

# Steepest descent algorithm in Hilbert space setting



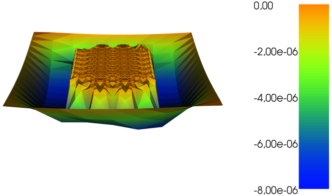

$i \leftarrow 0$
while $\|dJ(m^i)\|_{W^*} < \mathrm{tol}$:
    $m^{i+1} \leftarrow m^i - \gamma \nabla dJ(m^i)$
    $i \leftarrow i + 1$

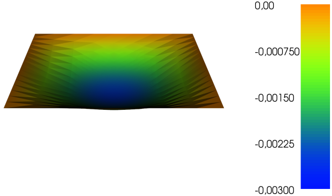
- $m^i \in W$
- $J : W \to \mathbb{R}$
- $dJ(m^i) : W \to \mathbb{R} \in W^*$
- $\nabla J(m^i) \in W$

With Riesz isomoprohism $R : W^* \to W$ the gradient is defined as $\nabla J(m) = R(dJ(m))$, i.e.

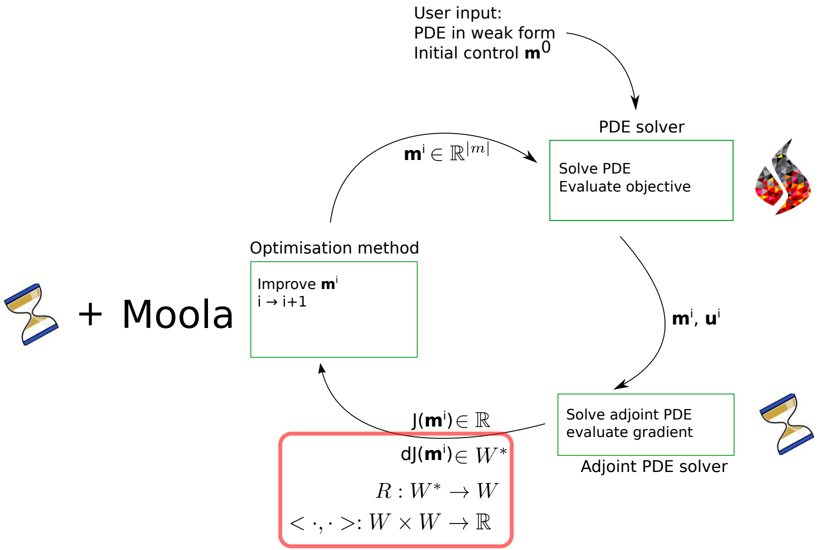
$$\langle dJ(m), \delta m \rangle_{W^*,W} = (\nabla f(m), \delta m)_W$$

# Gradient and derivative operator are, and should be treated as different objects



# Moola is a new Python library that implements optimisation algorithms in Hilbert spaces



# Moola treats vectors in primal and dual space as different objects

```
from fenics import *
from moola import DolfinDualVector

mesh = UnitSquareMesh(2, 2)
W = FunctionSpace(mesh, "CG", 1)
f = Function(W)

dJ = DolfinDualVector(f, inner_product="H1")
gradient = dJ.primal()  # Apply Riesz map

dJ.apply(gradient)      # OK
gradient + dJ           # Error
```

# The BFGS algorithm in Hilbert setting

$i \leftarrow 0$
while $|dJ(m^i)| < \mathrm{tol}$:
    $d^i \leftarrow -B_i^{-1} dJ(m^i)$
    $m^{i+1} \leftarrow m^i - \gamma d^i$
    $i \leftarrow i + 1$

with

$$s_i = m^{i+1} - m^i \in W$$
$$y_i = dJ(m^{i+1}) - dJ(m^i) \in W^*$$

and the inverse Hessian approximation $H_{i+1} : W^* \to W$

$$B_{i+1}^{-1}(x) = \left(\mathrm{id} - \frac{y_i(\mathrm{id})}{y_i(s_i)} s\right) \left[B^{-1} \left(x - \frac{x(s_i)}{y_i(s_i)} y_i\right)\right] - \frac{x(s_i)}{y_i(s_i)} s_i$$

# The resulting code is relatively compact

```
from dolfin import *
from dolfin_adjoint import *
import moola

mesh = UnitSquareMesh(64, 64)
V = FunctionSpace(mesh, "CG", 1)
W = FunctionSpace(mesh, "CG", 1)

m = Function(W, name='Control')
u = Function(V, name='State')
v = TestFunction(V)

# Define weak problem
F = (inner(grad(u), grad(v)) - m*v)*dx
bc = DirichletBC(V, 0.0, "on_boundary")
solve(F == 0, u, bc)

# Define regularisation parameter
alpha = Constant(1e-6)

# Define desired temperature profile
x = SpatialCoordinate(mesh)
d = (1/(2*pi**2) + 2*alpha*pi**2)*sin(pi*x[0])*sin(pi*x[1])

control = SteadyParameter(m)
J = Functional((0.5*inner(u-d, u-d))*dx + alpha/2*m**2*dx)
rf = ReducedFunctional(J, control)

# Set up moola problem and solve optimisation
problem = rf.moola_problem()
m_moola = moola.DolfinPrimalVector(m, inner_product="L2")
solver = moola.BFGS(problem, m_moola)
sol = solver.solve()
```

# Hilbert space based BFGS yields mesh independent iterations

- Algorithm: Moola, limited memory BFGS algorithm, identity as initial Hessian, history length: 20
- $L^2$ inner product.

**Uniform refinement**

| | | | | |
|---|---|---|---|---|
| Refinement level | 0 | 1 | 2 | 3 |
| BFGS iterations | 16 | 20 | 20 | 20 |

**Random refinement**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Refinement level | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| BFGS iterations | 18 | 18 | 18 | 20 | 21 | 20 | 20 |

**Centric refinement**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Refinement level | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| BFGS iterations | 20 | 19 | 20 | 20 | 19 | 19 | 19 |

# Hilbert space based Newton-CG yields mesh independent Newton and CG iterations

- Algorithm: Moola, Newton-CG
- $L^2$ inner product.

**Uniform refinement**

| | | | | |
|---|---|---|---|---|
| Refinement level | 0 | 1 | 2 | 3 |
| Newton iterations | 2 | 2 | 2 | 2 |
| CG iterations | 20 | 24 | 27 | 28 |

**Random refinement**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Refinement level | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| Newton iterations | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| CG iterations | 20 | 19 | 26 | 23 | 25 | 25 | 29 |

**Centric refinement**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Refinement level | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| Newton iterations | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| CG iterations | 24 | 24 | 26 | 27 | 27 | 27 | 27 |

## The choice of inner product is crucial to obtain mesh independent convergence

- Optimisation algorithm: Moola, limited memory BFGS

**Random refinement**

*$H1$ regularisation, L2 inner product in Moola*

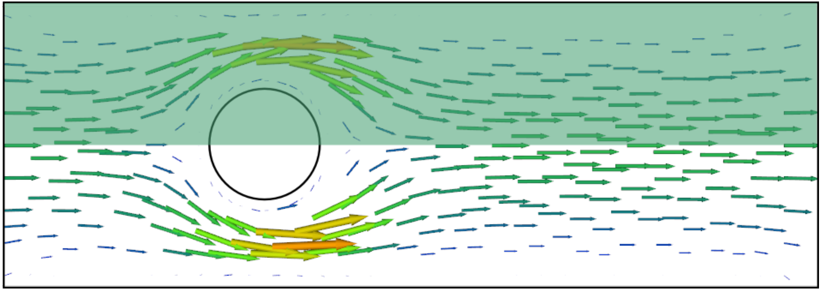
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Refinement level | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| BFGS iterations | 31 | 38 | 64 | 118 | $> 200$ | $> 200$ | |

*$H1$ regularisation, $H1$ inner product in Moola*

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Refinement level | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| BFGS iterations | 5 | 5 | 5 | 5 | 5 | 6 | 4 |

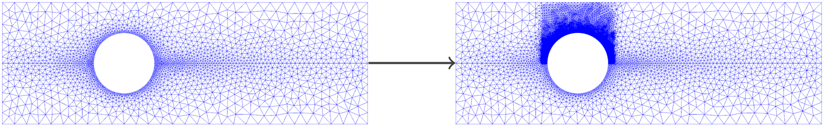
# A more advanced example

Consider Navier-Stokes flow around a cylinder driven by a pressure difference at the left and right boundaries:



# The choice of inner product is crucial to obtain mesh independent convergence

- Optimisation algorithm: Moola, Newton-CG
- Stopping criteria: $\|\mathrm{d}J\|_{L^2} < 10^{-4}$.
- Regularisation parameter: $\alpha = 10^{-4}$.



**Refinement**

| | | | | | |
|---|---|---|---|---|---|
| Refinement level | 0 | 1 | 2 | 3 | 4 |
| Newton iterations | 6 | 5 | 5 | 5 | 5 |
| CG iterations | 11 | 10 | 10 | 10 | 10 |

# Conclusions

- It is important to keep in mind that we solve instances of infinite dimensional problems.
- Mesh-independence convergence is obtained by respecting the inner product of the underlying function spaces.
- Alternatively the optimisation algorithm in Euclidian norm may be preconditioned with a Riesz-preconditioner.
- The choice of inner product is important to achieve mesh-independent convergence.