Automated Adjoints of Finite Element Discretizations

S. W. Funke, P. E. Farrell, D. A. Ham, M. E. Rognes

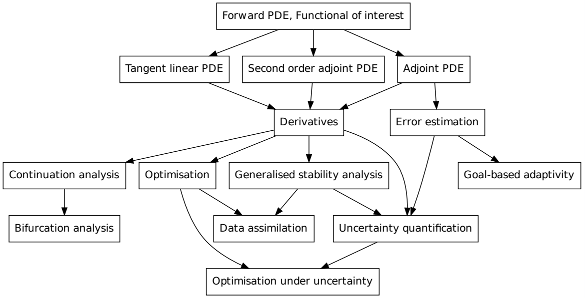
Imperial College London
Simula Research Laboratory

February 28, 2013

Vision

Vision

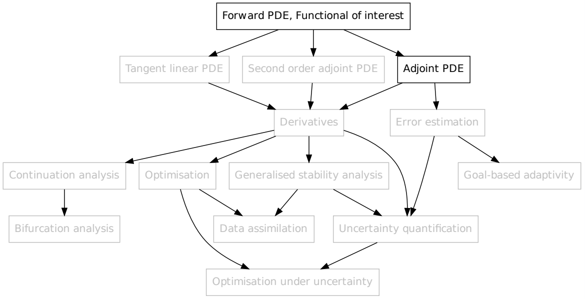
Building tools with a high degree of automation for solving:



Vision

Vision

Building tools with a high degree of automation for solving:



Traditional approach

$$\begin{tikzcd} \boxed{\text{discrete forward PDE}} \arrow[r, "\text{implement model by hand}"] & \boxed{\text{forward code}} \arrow[d, "\text{algorithmic differentiation (AD)}"'] \\ & \boxed{\text{adjoint code}} \end{tikzcd}$$

Traditional approach

$$\begin{tikzcd}
\boxed{\text{discrete forward PDE}} \arrow[r, "\text{implement model by hand}"] & \boxed{\text{forward code}} \arrow[d, "\text{algorithmic differentiation (AD)}"'] \\
& \boxed{\text{adjoint code}}
\end{tikzcd}$$

Fundamental idea of AD

A model is a sequence of elementary instructions.
Differentiate each in turn, and compose with the chain rule.

Traditional approach

$$
\begin{tikzcd}
\boxed{\text{discrete forward PDE}} \arrow[r, "\text{implement model by hand}"] & \boxed{\text{forward code}} \arrow[d, "\text{algorithmic differentiation (AD)}"'] \\
& \boxed{\text{adjoint code}}
\end{tikzcd}
$$

Problematic for AD

- pointers
- expressions with side effects
- preprocessor directives
- external libraries
- mixed-language programming
- parallel directives

Our approach

$$
\begin{tikzcd}
\boxed{\text{discrete forward equations}} \arrow[r, "\text{code generation}"] \arrow[d, "\text{symbolic manipulation}"] & \boxed{\text{forward code}} \\
\boxed{\text{discrete adjoint equations}} \arrow[r, "\text{code generation}"] & \boxed{\text{adjoint code}}
\end{tikzcd}
$$

Our approach

$$
\begin{tikzcd}[column sep=huge]
\boxed{\text{discrete forward equations (UFL)}} \arrow[r, "\text{code generation (FEniCS)}"] \arrow[d, "\text{symbolic manipulation (dolfin-adjoint)}"] & \boxed{\text{forward code}} \\
\boxed{\text{discrete adjoint equations (UFL)}} \arrow[r, "\text{code generation (FEniCS)}"] & \boxed{\text{adjoint code}}
\end{tikzcd}
$$

FEniCS

Advection-diffusion equation (variational formulation)

$$(a \cdot \nabla u, v) + \nu (\nabla u, \nabla v) = 0 \quad \forall v$$

Advection-diffusion equation (UFL)

```
F = inner(a*grad(u), v)*dx + nu*inner(grad(u), grad(v))*dx
solve(F == 0, u)
```

Automated adjoints in FEniCS

Key steps performed by dolfin-adjoint:

1. Building the tape
2. Symbolic derivation of the adjoint equations
3. Solve the adjoint equations

Building the tape

Code

```
->  F1 = inner(...)
    solve(F1 == 0, u1)


    F2 = inner(...)
    solve(F2 == 0, u2)


    F3 = inner(...)
    solve(F3 == 0, u3)
```

Tape

Equation (UFL)	Solution

Building the tape

Code

```
    F1 = inner(...)
->  solve(F1 == 0, u1)


    F2 = inner(...)
    solve(F2 == 0, u2)


    F3 = inner(...)
    solve(F3 == 0, u3)
```

Tape

Equation (UFL)	**Solution**
F1 $=$ inner(...) $= 0$	$u1 = (0.1, 1.2, ...)$

Building the tape

Code

```
    F1 = inner(...)
    solve(F1 == 0, u1)

->  F2 = inner(...)
    solve(F2 == 0, u2)

    F3 = inner(...)
    solve(F3 == 0, u3)
```

Tape

Equation (UFL)	**Solution**
F1 = inner(...) = 0	$u1 = (0.1, 1.2, ...)$

Building the tape

Code

```
    F1 = inner(...)
    solve(F1 == 0, u1)

    F2 = inner(...)
->  solve(F2 == 0, u2)

    F3 = inner(...)
    solve(F3 == 0, u3)
```

Tape

Equation (UFL)	**Solution**
F1 $=$ inner(...) $= 0$	$u1 = (0.1, 1.2, ...)$
F2 $=$ inner(...) $= 0$	$u2 = (1.1, 2.2, ...)$

Building the tape

Code

```
    F1 = inner(...)
    solve(F1 == 0, u1)


    F2 = inner(...)
    solve(F2 == 0, u2)


->  F3 = inner(...)
    solve(F3 == 0, u3)
```

Tape

Equation (UFL)	**Solution**
F1 = inner(...) = 0	$u1 = (0.1, 1.2, ...)$
F2 = inner(...) = 0	$u2 = (1.1, 2.2, ...)$

Building the tape

Code

```
    F1 = inner(...)
    solve(F1 == 0, u1)


    F2 = inner(...)
    solve(F2 == 0, u2)


    F3 = inner(...)
->  solve(F3 == 0, u3)
```

Tape

Equation (UFL)	Solution
F1 = inner(...) = 0	$u1 = (0.1, 1.2, ...)$
F2 = inner(...) = 0	$u2 = (1.1, 2.2, ...)$
F3 = inner(...) = 0	$u3 = (3.4, 0.6, ...)$

Automated adjoints in FEniCS

Key steps performed by dolfin-adjoint:

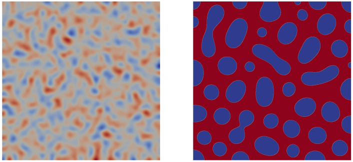
1. Building the tape

2. Symbolic derivation of the adjoint equations

3. Solving the adjoint equations

Cahn-Hilliard equation

$$\frac{\partial c}{\partial t} - \nabla \cdot M \left(\nabla \left(\frac{df}{dc} - \epsilon^2 \nabla^2 c \right) \right) = 0$$



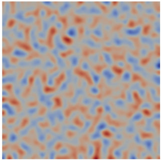
t = 0 t = 0.00025

Code

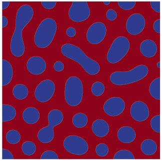
```
# Code for solving the Cahn-Hilliard equation
# ...

J = Functional(...)
m = InitialConditionParameter(c)
dJdm = compute_gradient(J, m)
```

Cahn-Hilliard equation

$$\frac{\partial c}{\partial t} - \nabla \cdot M \left(\nabla \left(\frac{df}{dc} - \epsilon^2 \nabla^2 c \right) \right) = 0$$



t = 0



t = 0.00025

	Runtime (s)	Ratio
Forward model (5 Newton iter.)	103.9	
Forward + adjoint model	127.1	**1.22**

Key features

Fully automated adjoint derivation.

Key features

Fully automated adjoint derivation.

Assumes the forward model can be expressed in UFL.

Key features

Fully automated adjoint derivation.

Assumes the forward model can be expressed in UFL.

Works naturally in parallel.

Key features

Fully automated adjoint derivation.

Assumes the forward model can be expressed in UFL.

Works naturally in parallel.

Adjoint model is very efficient.

Key features

Fully automated adjoint derivation.

Assumes the forward model can be expressed in UFL.

Works naturally in parallel.

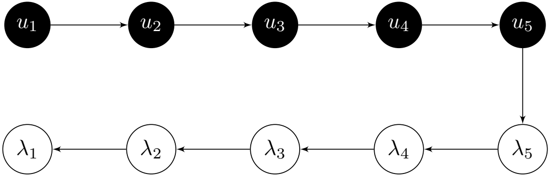
Adjoint model is very efficient.

Tangent linear and second order adjoint model work the same way:

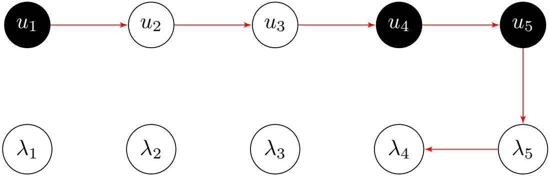
Code

```
h = hessian(J, m)(delta_m)
```

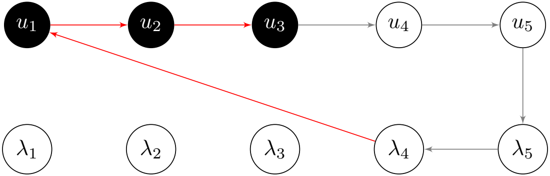
Optimal checkpointing with `revolve`



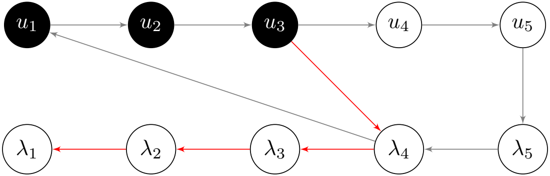
Optimal checkpointing with `revolve`



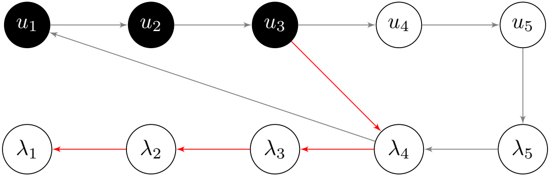
Optimal checkpointing with `revolve`



Optimal checkpointing with `revolve`



Optimal checkpointing with `revolve`



Recomputation cost	# checkpoints	Max. # timesteps
$\times 2$	50	1, 000
$\times 2$	100	5, 000
$\times 3$	50	23, 000
$\times 3$	100	177, 000

Logarithmic growth of computation and spatial complexity in the number of time levels can be achieved (Griewank, 1992).

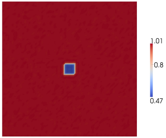
Optimal checkpointing

Code

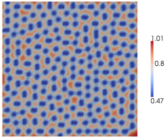
```
use_checkpointing(strategy='multistage', steps=20000,
                  snaps_on_disk=50, snaps_in_ram=50)
```

Gray-Scott equations

$$\frac{\partial u}{\partial t} = D_u \nabla^2 u - uv^2 + F(1-u),$$
$$\frac{\partial v}{\partial t} = D_v \nabla^2 v + uv^2 - (F+k)v,$$



$T = 0$



$T = 20,000$

	Forward steps	Runtime of forward steps
Without checkpointing	100	21 s
With 7 checkpoints	255	54 s
Ratio	2.55	2.60

PDE-constrained optimisation

PDE-constrained optimisation

General problem

$$\min_{u,m} J(u,m)$$
$$\text{subject to} \quad F(u,m) = 0$$

Notation

$J(u,m)$ Functional
$F(u,m) = 0$ Forward PDE
u PDE solution
m Control
$u(m)$ Solution operator

PDE-constrained optimisation

General problem

$$\min_{u,m} J(u,m)$$
$$\text{subject to} \quad F(u,m) = 0$$

Notation

$J(u,m)$ Functional
$F(u,m) = 0$ Forward PDE
u PDE solution
m Control
$u(m)$ Solution operator

Reduced problem

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

PDE-constrained optimisation

Simple optimisation loop

Choose initial m

do

Compute functional J

Compute the gradient dJ/dm

Update optimisation variables m by performing a line seach

while tolerance not reached

PDE-constrained optimisation

Simple optimisation loop

Choose initial m

do

Compute functional J

$\rightarrow$ by replaying the tape

Compute the gradient dJ/dm

$\rightarrow$ by deriving the adjoint PDE

Update optimisation variables m by performing a line seach

$\rightarrow$ by updating tape

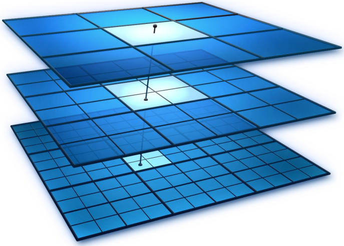
while tolerance not reached

Code

```
Jhat = ReducedFunctional(J, m)
m_opt = minimize(Jhat)
```

Key features

High degree of automation

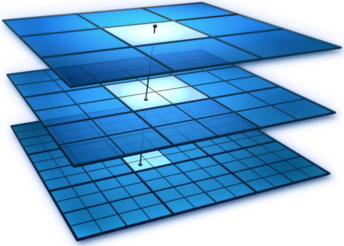


Key features

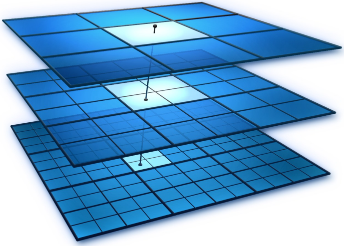
High degree of automation

Optimisation algorithms

- SQP
- L-BFGS-B
- Newton-CG (using Hessian information)
- Multigrid optimisation
- ...



Key features



High degree of automation

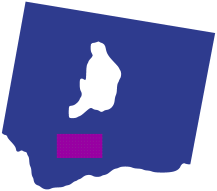
Optimisation algorithms

- SQP
- L-BFGS-B
- Newton-CG (using Hessian information)
- Multigrid optimisation
- ...

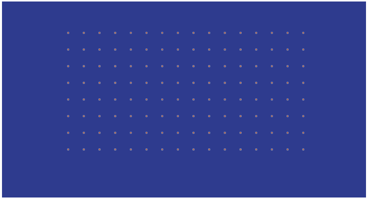
Constraints

- Control bounds
- (In-)Equality constraints

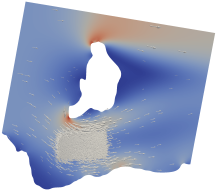
Optimal placement of tidal turbines



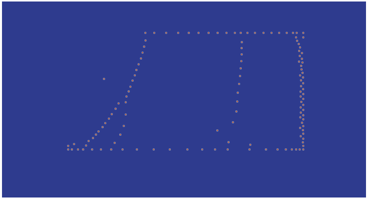
Domain and turbine site



Initial turbine layout



Velocity



Optimised turbine layout

Conclusion and future work

Summary

Automated adjoints for finite element discretisations.

Supports **coupled, nonlinear and time-dependent** problems.

Framework for solving **PDE-constrained optimisation** problems.

Future work

Advanced multigrid and reduced order optimisation methods.

Automated shape derivatives and optimisation.

Further applications.

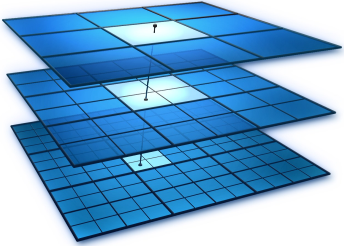
...

`dolfin-adjoint`

http://dolfin-adjoint.org

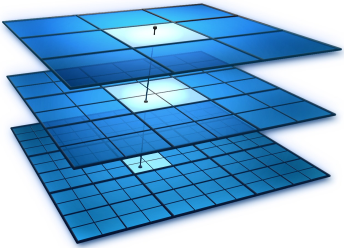
Multigrid optimisation

Solve the optimisation problem iteratively on different mesh resolutions.



Multigrid optimisation

Solve the optimisation problem iteratively on different mesh resolutions.



Idea

Replace the underlying meshes on the tape!

```
m_opt = minimize(Jhat, method = "successive", levels = 2)
```

Bilinear elliptic equation (problem from M. Vellejos, 2010)

$$\min_{u,m} \frac{1}{2} ||u - z||^2 + \frac{\nu}{2} ||m||^2$$
$$\text{subject to:} \ -\nabla^2 u - um = f$$

Single grid

# Iterations	Runtime (s)
10	329

Multigrid, 2 levels

Refinement	# Iterations	Runtime (s)
Level 2	11	40
Level 1	2	76
Total		116