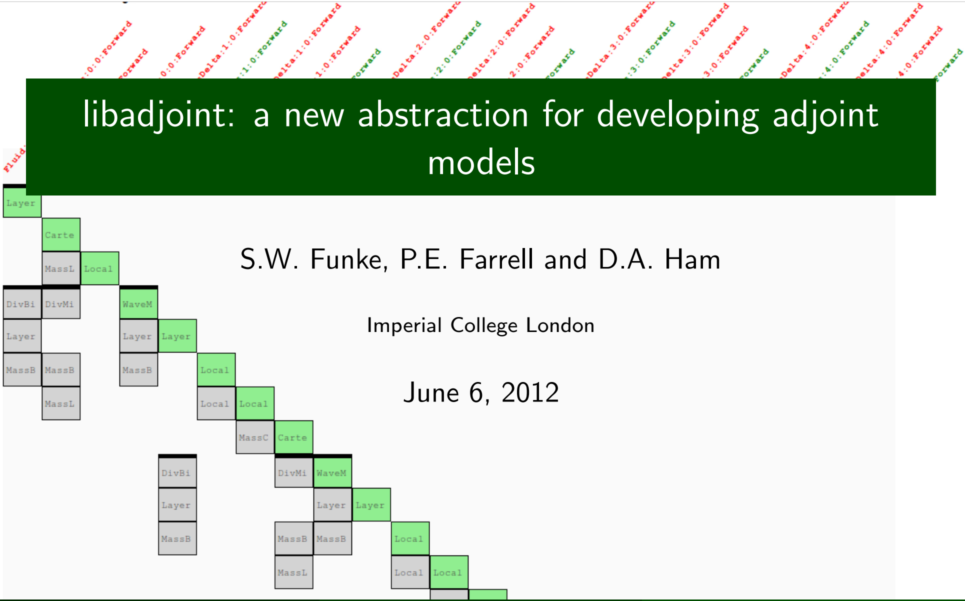


libadjoint: a new abstraction for developing adjoint models

S.W. Funke, P.E. Farrell and D.A. Ham

Imperial College London

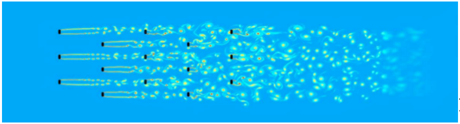
June 6, 2012

Outline

Outline

Example problem

What is the optimal turbine layout in a tidal stream to extract most energy from the tidal current?

 [1]

Problem formulation

$$\max_m \mathsf{Power}(u, m)$$

$$\text{s.t.} \quad u_t + \nabla\eta = mu,$$

$$\eta_t + \nabla \cdot u = 0.$$

m: turbine positions

u: velocity

η: water elevation.

To solve this problem efficiently, we want to apply gradient based optimisation.

How do we compute $\frac{d\text{Power}}{dm}$?

[1] Divett et al. Optimisation of multiple turbine arrays in a channel, 2011.

Derivation of the adjoint equation

The general form of the example problem is:

$$\min_m J(u, m) \qquad \text{subject to} \quad F(u, m) = 0, \tag{1}$$

$J(u, m) \in \mathbb{R}$ is the functional of interest, m are the control variables and F is the PDE operator with solution $u(m)$.

Derivation of the adjoint equation

The general form of the example problem is:

$$\min_m J(u,m) \qquad \text{subject to} \quad F(u,m) = 0, \tag{1}$$

$J(u,m) \in \mathbb{R}$ is the functional of interest, m are the control variables and F is the PDE operator with solution $u(m)$.

We seek the total derivative of J with respect to the controls m:

$$\frac{dJ}{dm} = J_u \frac{du}{dm} + J_m. \tag{2}$$

Taking the derivative of the constraint in (1) w.r.t. m yields:

$$F_u \frac{du}{dm} + F_m = 0. \tag{3}$$

Derivation of the adjoint equation

The general form of the example problem is:

$$\min_m J(u,m) \qquad \text{subject to} \quad F(u,m) = 0, \tag{1}$$

$J(u,m) \in \mathbb{R}$ is the functional of interest, m are the control variables and F is the PDE operator with solution $u(m)$.
We seek the total derivative of J with respect to the controls m:

$$\frac{dJ}{dm} = J_u \frac{du}{dm} + J_m. \tag{2}$$

Taking the derivative of the constraint in (1) w.r.t. m yields:

$$F_u \frac{du}{dm} + F_m = 0. \tag{3}$$

(3) in (2) yields:

$$\frac{dJ}{dm} = -\overbrace{J_u F_u^{-1}}^{:=\lambda^*} F_m + J_m,$$

where λ is the adjoint solution.

Adjoint equation

The adjoint equation is therefore:

$$F_u^*(u, m)\lambda = J_u^*(u, m)$$

Key properties

1. The adjoint equation is a linear.
2. The adjoint equation is solved backward in time.
3. The functional gradient is obtained by computing

$$\frac{dJ}{dm} = -\lambda^* F_m + J_m.$$

Hence the derivative computation requires **one** forward solve for u and **one** adjoint solve for λ, independently of the choice of m!

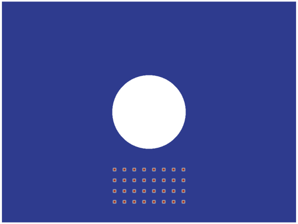
Outline

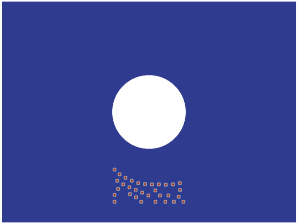
Efficient gradient computation

Applications

- Sensitivity analysis
- Data assimilation
- Design optimisation
- Inverse problems

The turbine layout optimisation problem





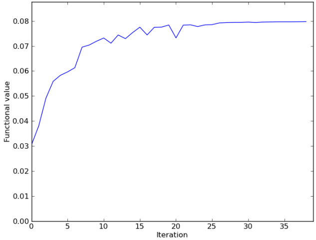


Figure: Initial and optimised turbine positions and the power increase.

Goal-oriented adaptivity

Goal-oriented adaptivity

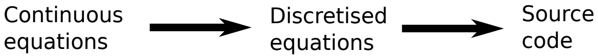
Goal-oriented adaptivity and error control optimises the computational resources by targeting the numerical simulations at a specific quantity of interest.

Figure 1.2: *Meshes with 5,000 cells obtained by the vorticity indicator (left) and the new DWR indicator (right).* [2]

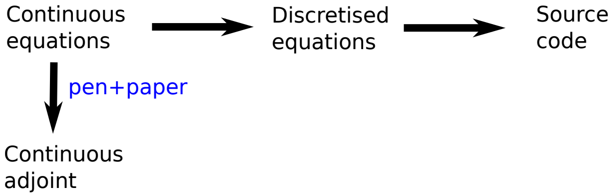
[2] W. Bangerth, R. Rannacher. Adaptive Finite Element Methods for Differential Equations, 2003.

Outline

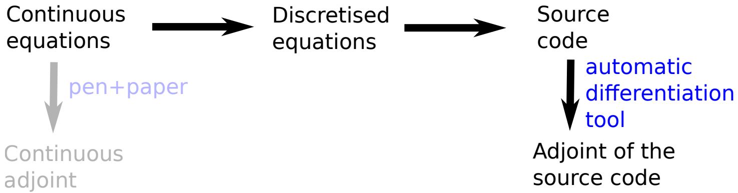
The stages of developing a model



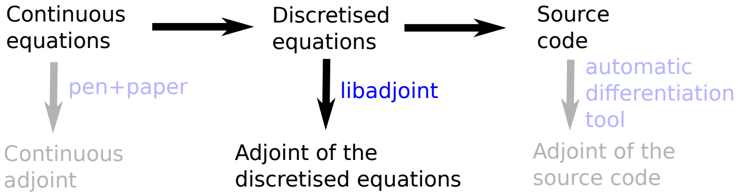
Continuous adjoint



Algorithmic differentiation



Libadjoint's approach



Outline

The fundamental idea of Libadjoint

libadjoint is a library that facilitates the development of discrete adjoint models.

The fundamental idea of AD

A model is a sequence of elementary instructions.

The fundamental idea of Libadjoint

libadjoint is a library that facilitates the development of discrete adjoint models.

The fundamental idea of AD

A model is a sequence of elementary instructions.

The fundamental idea of libadjoint

A model is a sequence of equation solves.

Example: Burgers equation

The non-viscous Burgers equation has the form:

$$\frac{\partial u}{\partial t} + u \cdot \nabla u = 0.$$

The (explicit) discretisation with one nonlinear iteration per time step yields:

$$\underbrace{-(M + \Delta t A(u^n))}_{:=T(u^n)} u^n + M u^{n+1} = 0,$$

where M is the mass matrix, A is the discretised advection operator and Δt is the time step. We linearise the advection term using the velocity at the previous time step.

Example: Burgers equation

Three time steps can be written as a block matrix:

$$\underbrace{\begin{pmatrix} I & & & \\ T(u^0) & M & & \\ & T(u^1) & M & \\ & & T(u^2) & M \end{pmatrix}}_{F(u)} \underbrace{\begin{pmatrix} u^0 \\ u^1 \\ u^2 \\ u^3 \end{pmatrix}}_{u} = \underbrace{\begin{pmatrix} u_{\text{init}} \\ 0 \\ 0 \\ 0 \end{pmatrix}}_{b}$$

We have cast the model in the form $F(u)u = b$.

Example: Burgers equation

Three time steps can be written as a block matrix:

$$\underbrace{\begin{pmatrix} I & & & \\ T(u^0) & M & & \\ & T(u^1) & M & \\ & & T(u^2) & M \end{pmatrix}}_{F(u)} \underbrace{\begin{pmatrix} u^0 \\ u^1 \\ u^2 \\ u^3 \end{pmatrix}}_{u} = \underbrace{\begin{pmatrix} u_{\text{init}} \\ 0 \\ 0 \\ 0 \end{pmatrix}}_{b}$$

We have cast the model in the form $F(u)u = b$.

The associated adjoint equation is:

$$\left(F(u) + \frac{\partial F(u)}{\partial u} u\right)^* \lambda = \frac{\partial J}{\partial u}^*.$$

Example: Burgers equation

Three time steps can be written as a block matrix:

$$\underbrace{\begin{pmatrix} I & & & \\ T(u^0) & M & & \\ & T(u^1) & M & \\ & & T(u^2) & M \end{pmatrix}}_{F(u)} \underbrace{\begin{pmatrix} u^0 \\ u^1 \\ u^2 \\ u^3 \end{pmatrix}}_{u} = \underbrace{\begin{pmatrix} u_{\text{init}} \\ 0 \\ 0 \\ 0 \end{pmatrix}}_{b}$$

We have cast the model in the form $F(u)u = b$.
The associated adjoint equation is:

$$\left(F(u) + \frac{\partial F(u)}{\partial u} u\right)^* \lambda = \frac{\partial J}{\partial u}^*.$$

Therefore the adjoint equation reads:

$$\begin{pmatrix} I^* & \left(T(u^0) + \frac{\partial T(u^0)}{\partial u^0} u^0\right)^* & & \\ & M^* & \left(T(u^1) + \frac{\partial T(u^1)}{\partial u^1} u^1\right)^* & \\ & & M^* & \left(T(u^2) + \frac{\partial T(u^2)}{\partial u^2} u^2\right)^* \\ & & & M^* \end{pmatrix} \begin{pmatrix} \lambda^0 \\ \lambda^1 \\ \lambda^2 \\ \lambda^3 \end{pmatrix} = \frac{\partial J}{\partial u}^*.$$

The development of an adjoint model with `libadjoint` requires two steps:

1. Annotation
2. Callback registration

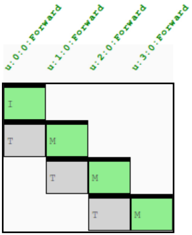
Step 1: Annotation

- `libadjoint` provides a set of library calls with which a model may be annotated at runtime
- Each equation solve is annotated to record what is being computed, what operators are acting on previously computed values, and their nonlinear dependencies

The annotation

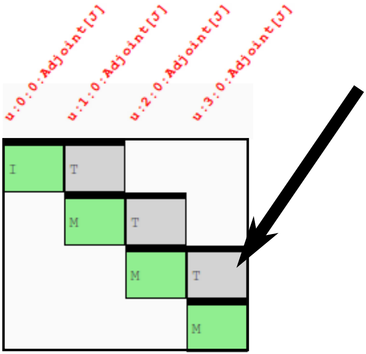
is sufficient to describe the discretisation matrix of the forward model...

$$\begin{pmatrix} I & & & \\ T(u^0) & M & & \\ & T(u^1) & M & \\ & & T(u^2) & M \end{pmatrix} \longrightarrow$$



Step 1: Annotation

...and so libadjoint can derive the associated adjoint system:



Targets: u:2:0:Adjoint[]
Timestep:2
Iteration:0

===== Block description =====

T

Coefficient: 1.000000
Hermitian: true
Nonlinear Block: A Dependencies: u:2:0

+

Derivative of A
with respect to u:2:0:Forward
contracted with u:2:0:Forward

Step 2. Register function callbacks

- `libadjoint` offers the facility to register function callbacks for computing the action of the operators in the annotation
- ... and their derivatives (e.g. by using AD)
- It also offers the facility to record solutions

With the callbacks ...

... `libadjoint` can **automatically assemble the adjoint equations**.

Key properties of libadjoint

- \+ Works with modern language features and external libraries
- \+ The approach meshes well with AD
- \+ Comes with an optimal checkpointing strategy: `Revolve`[3]
- – The annotation and callback implementation has to be done by hand, however in some cases this can be automated (DOLFIN)

[3] A. Griewank, A. Walther, Revolve: an implementation of checkpointing for the reverse or adjoint mode of computational differentiation, TOMS (2000)

Summary

We have seen:

- An introduction and applications to adjoints
- Three ways how to adjoint a model
- How to adjoint a model using `libadjoint`