Imperial College London

Firedrake

Making point sets first class in UFL and Firedrake

Reuben W. Nixon-Hill[1,2], Daniel Shapero[3], Colin J. Cotter[2], David A. Ham[2]

[1] Science and Solutions for a Changing Planet DTP, Grantham Institute for Climate Change and the Environment, Imperial College London
[2] Department of Mathematics, Imperial College London
[3] Polar Science Center, Applied Physics Laboratory, University of Washington

Grantham Institute Climate Change and the Environment An institute of Imperial College London

Science and Solutions for a Changing Planet DTP

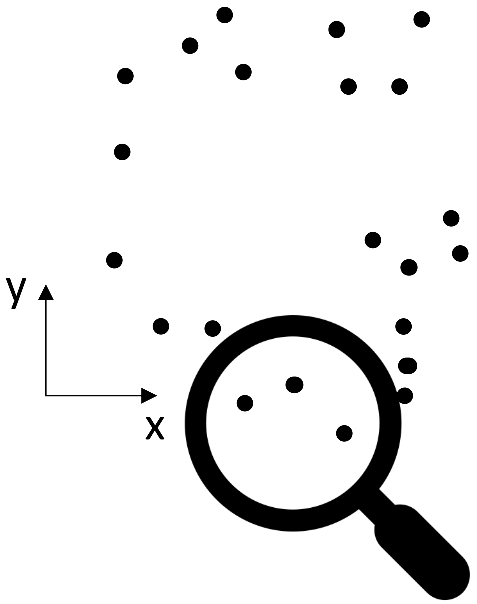
UKRI Natural Environment Research Council

Why care about point data?

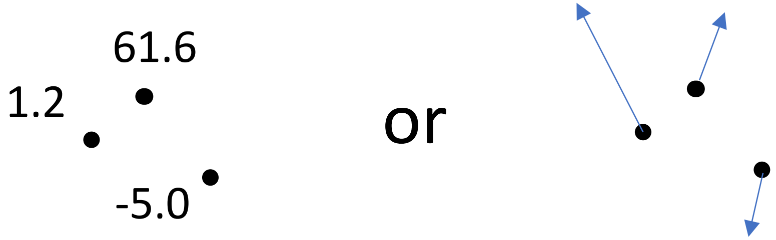
- Geoscience Examples
  - Ice Sheets: elevation from **satellite altimetry** and **ice cores**
  - Ocean: salinity and temperature from **drifting buoys**
  - Atmosphere Climate: conditions at **weather stations**
  - **And more**!
- Typical uses
  - **Point evaluation** of PDE solutions
  - PDE Constrained Optimisation
    - Variational **data assimilation**
    - Goal based error estimation
    - And more!

Definition: Point Data

1. A “point cloud” $\{X_i\}$ – a set of spatial coordinates and



2. Values y_i (scalar, vector or tensor valued) at those coordinates.



How we deal with point data at the moment

```
# UFL
# evaluate coefficient f at two
# 2-dimensional points
vals[0] = f([0.2, 0.4])
vals[1] = f([1.2, 0.5])
```

```
# Firedrake (dolfin similar)
# evaluate Function f at two
# 2-dimensional points
from firedrake import *
...
vals = f.at([0.2, 0.4], [1.2, 0.5])
```

- UFL expressions of coefficients can be evaluated at given point coordinates
- Can get values from point clouds

How we deal with point data at the moment

```
# UFL
# evaluate coefficient f at two
# 2-dimensional points
vals[0] = f([0.2, 0.4])
vals[1] = f([1.2, 0.5])
```

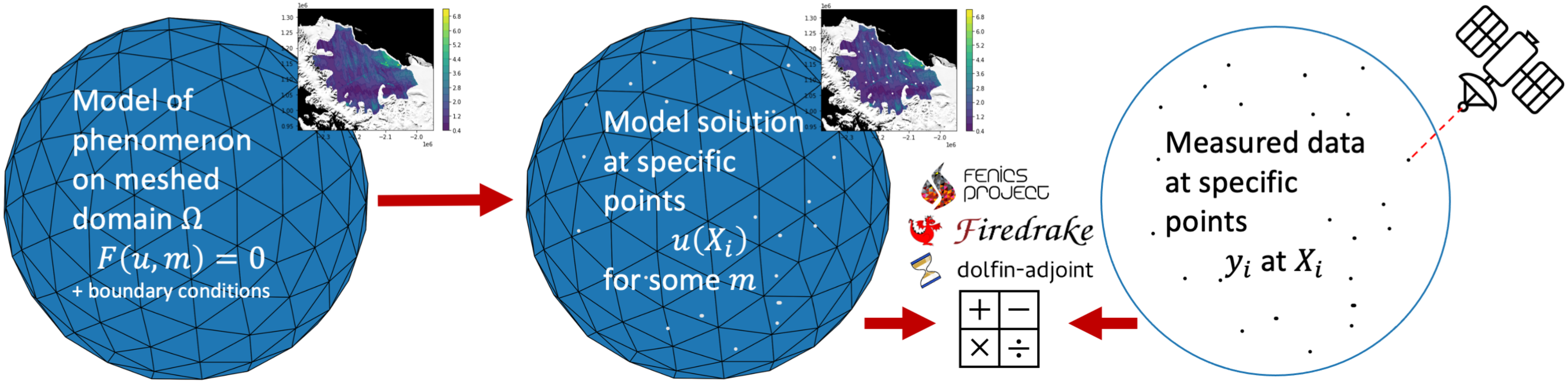
- **outside of the UFL type system of fields in function spaces on meshes**
- Value driven: no symbolic point evaluation.
- ~~dolfin adjoint~~

```
# Firedrake
# evaluate Function f at two
# 2-dimensional points
from firedrake import *
...
vals = f.at([0.2, 0.4], [1.2, 0.5])
```

- end up special-casing point evaluation code pathways
- Often slow!

Example Use Case

Point Data Assimilation: A PDE Constrained Optimisation Problem



Minimization Problem:

$$\min_{u,m} J[u,m] \quad \text{where } J = \underbrace{\|u(X_i) - y_i\|^2}_{\substack{\text{misfit} \\ \text{(Fit of model to data)}}} + \underbrace{c\|m\|^2}_{\substack{\text{regularization} \\ \text{(e.g. to ensure smooth fit)}}}$$

subject to

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

+ boundary conditions

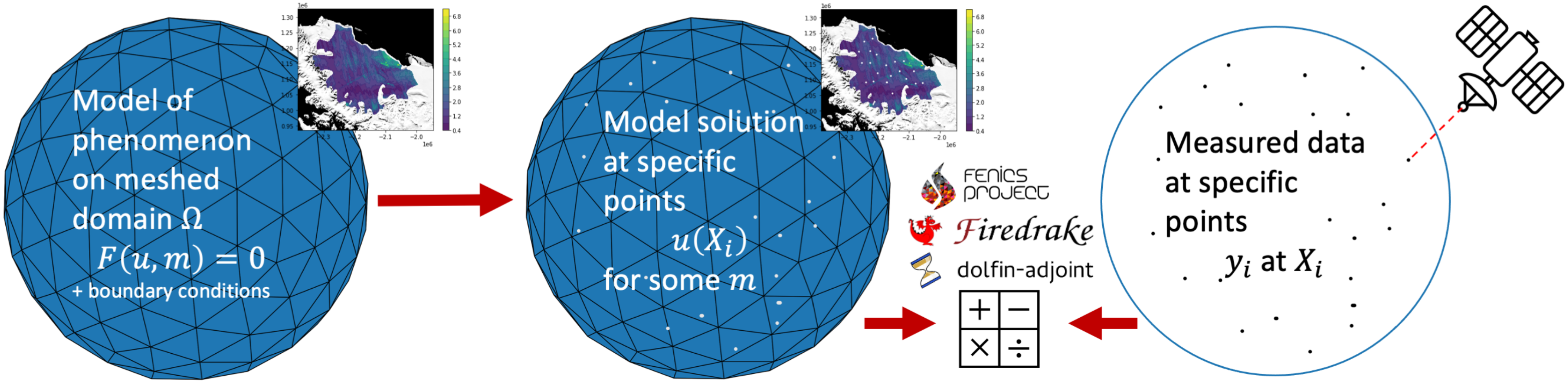
Gradient Based Optimisation Needs

$$\frac{\partial J}{\partial u}\frac{du}{dm}$$

Find via solution to the adjoint problem using dolfin-adjoint

Example Use Case

Point Data Assimilation: A PDE Constrained Optimisation Problem



Minimization Problem:

$$\min_{u,m} J[u,m] \quad \text{where} \quad J = \|u(X_i) - y_i\|^2 + c\|m\|^2$$

Need to express this in UFL and annotate it with dolfin-adjoint/pyadjoint

Gradient Based Optimisation Needs

$$\frac{\partial J}{\partial u}\frac{du}{dm}$$

Find via solution to the adjoint problem using dolfin-adjoint

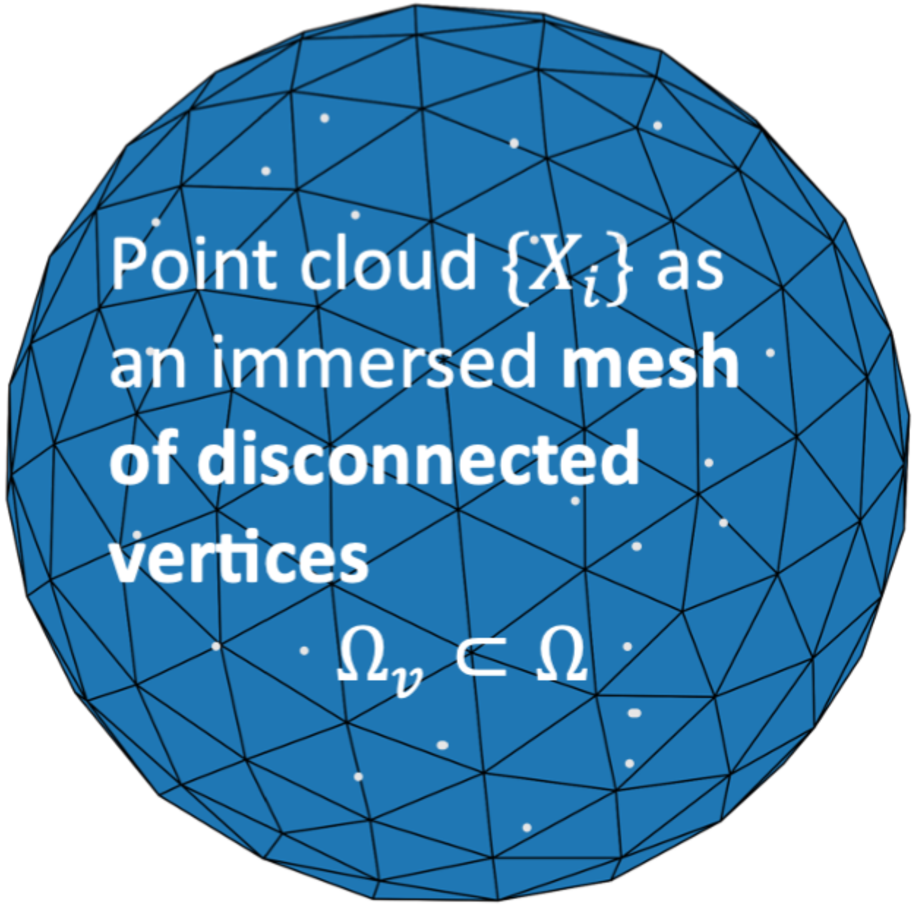
Solution Part 1: A point cloud is a mesh Ω_v

UFL

- Vertex Cells
- Topological dimension = 0
- Geometric dimension = $\dim(X_i)$ (point cloud coordinate dimension)

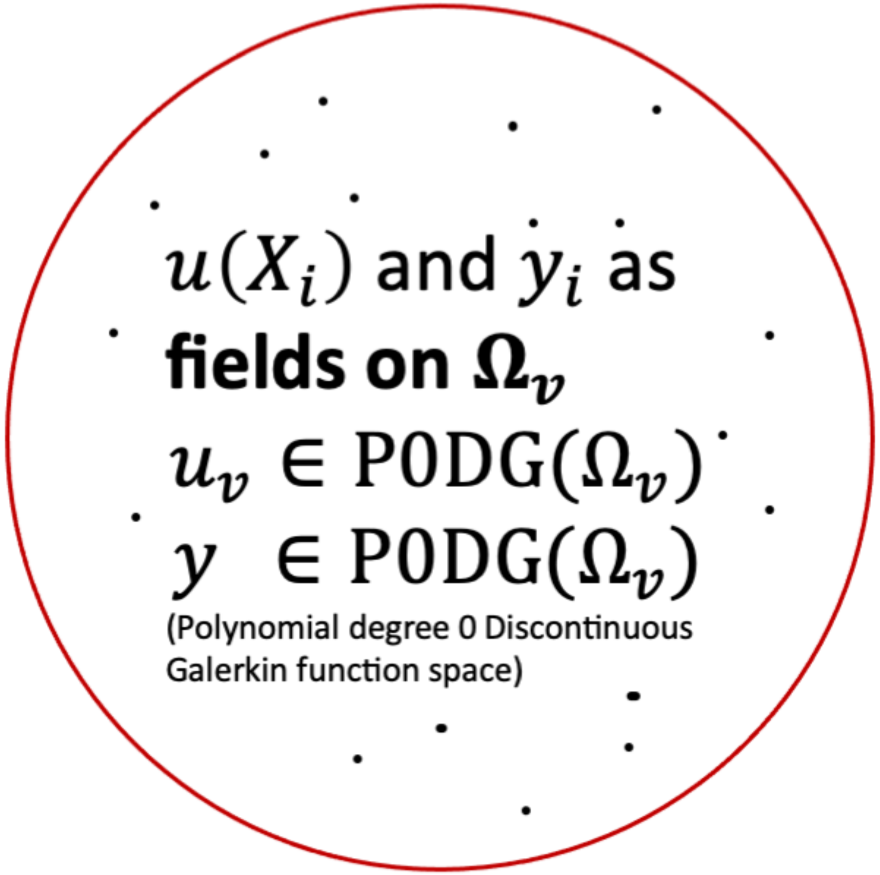
Firedrake

- Vertices at point cloud coordinates $\{X_i\}$
- Immersed (for now)
    - makes implementation simpler
    - care about the point data with respect to field on "parent" mesh Ω (e.g. a PDE solution)



Solution Part 2: Point data are functions in a P0DG function space on Ω_v

- Scalar, vector or tensor valued
- Each function in this function space represents **a complete set of point data**
  - Vertex coordinates $\{X_i\}$ are fixed*
  - Can **declare UFL arguments** to **solve for values** at points $\{X_i\}$
- Changes: UFL ✔ FIAT ✔ Firedrake ✔



*unless we represent moving points – something for the future!

How do we use this? Interpolate into the point data space

Point evaluation is now interpolation into $\mathrm{W} = \mathrm{P0DG}(\Omega_v)$

$$u_v = I(u, W)$$

Rewrite $J = \|u(X_i) - y_i\|^2 + c\|m\|^2$

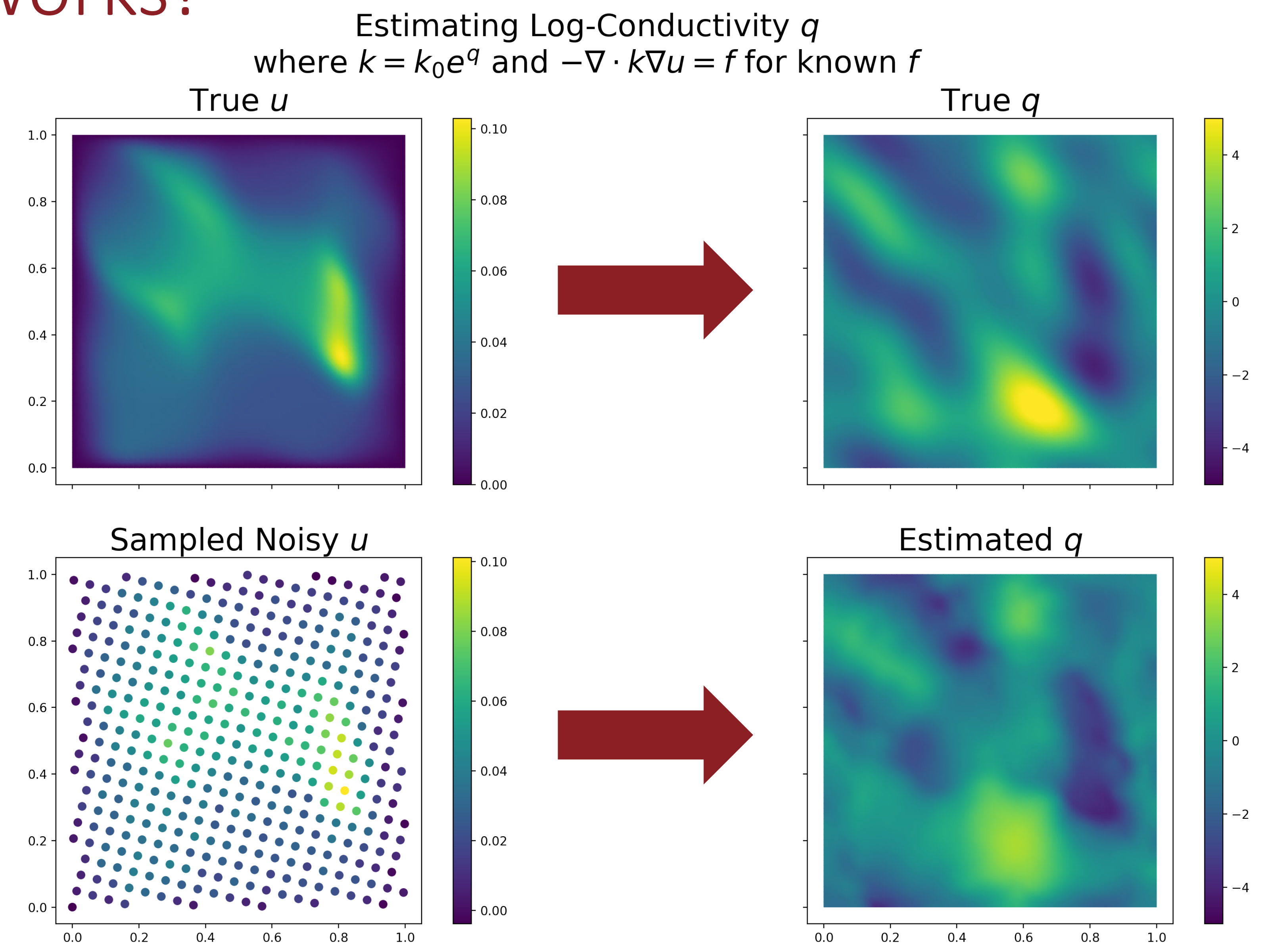
$$J = \int_{\Omega_v} (I(u, W) - y)^2 dx + c\|m\|^2$$

$$\frac{\partial J}{\partial u} = \int_{\Omega_v} (I(u, W) - y) \underbrace{\frac{\boldsymbol{\partial I(u, W)}}{\boldsymbol{\partial u}}}_{} dx$$

Newly added to Firedrake **and** dolfin-adjoint

```
# UFL+Firedrake Pseudocode
# blue is proposed new UFL

# u is a solution from a function space on parent_mesh
# m is source term from some function space

vm = VertexOnlyMesh(parent_mesh, point_cloud_coords)
W = FunctionSpace(vm, ”DG”, 0)

# interpolate
v = TestFunction(W).dual()
u_v = interp(u, v)  # or u_v = v(u)

...  # create y in W from observation data

# Functional for minimisation
J = assemble( (u_v – y)**2 * dx + c*inner(m, m) * dx )

# Reduce using control from parameter space
m̂ = firedrake_adjoint.Control(m)
Ĵ = firedrake_adjoint.ReducedFunctional(J, m̂)

# Find optimal control
m_min = firedrake_adjoint.minimize(Ĵ, method='Newton-CG’)
```

It really works!

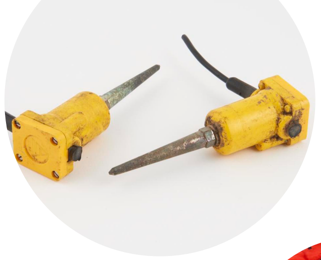


Future: UFL for Automated Diagnostics

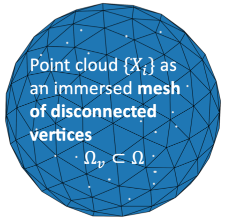
- Aim to develop UFL into a Domain Specific Language (DSL) for specifying model diagnostics
  - A user will specify, as some high level integration (e.g. over points or planes), the diagnostic then code will be generated to calculate it
  - Example: Ocean current from multiple climate models e.g. Atlantic Meridional Overturning Circulation in Medhaug and Furevik 2011 [4]
- Form compiler can then generate code for calculating the diagnostic
- **Scalable** and able to run on **big datasets** (e.g. climate) on the **HPCs closest to the data**.

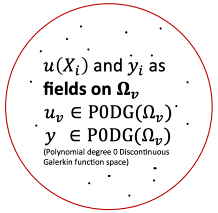
[1] Medhaug, I., & Furevik, T. (2011). North Atlantic 20th century multidecadal variability in coupled climate models: Sea surface temperature and ocean overturning circulation. *Ocean Science*, *7*(3), 389-404.

Conclusion



- Point data are everywhere and can be treated more rigorously
- We can represent point data as a function in a P0DG function space on a point cloud mesh
- Point evaluations are interpolations into this function space
- `interp` proposed as new UFL operator for interpolation
- Can use point data in PDE constrained optimisation problems by annotating interpolation with pyadjoint/dolfin-adjoint
- First step towards turning UFL into a DSL for automated diagnostics





Get in touch! **reuben.nixon-hill10@imperial.ac.uk**

Firedrake Imperial College London Grantham Institute Climate Change and the Environment An institute of Imperial College London Science and Solutions for a Changing Planet DTP UKRI Natural Environment Research Council

Possible work-around existing limitations

Minimization Problem:

$$\min_{u,m} J[u,m] \quad \text{where} \quad J = \|u(X_i) - y_i\|^2 + c\|m\|^2$$

subject to

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

+ boundary conditions

- Interpolate the point data into some function space on my mesh then calculate

$$J = \left\|u - y_{approx}\right\|^2 + c'\|m\|^2$$

- Have to make difficult-to-test decisions about appropriate interpolation hyperparameters to get y_{approx},
- Particularly a problem if the point data is sparse

Solution Part 2: Point data are functions in a P0DG function space on Ω_v

Some Properties:

- Integrating sums values at points

$$\int_{\Omega_v} u_v dx = \sum_i u_v(X_i)$$

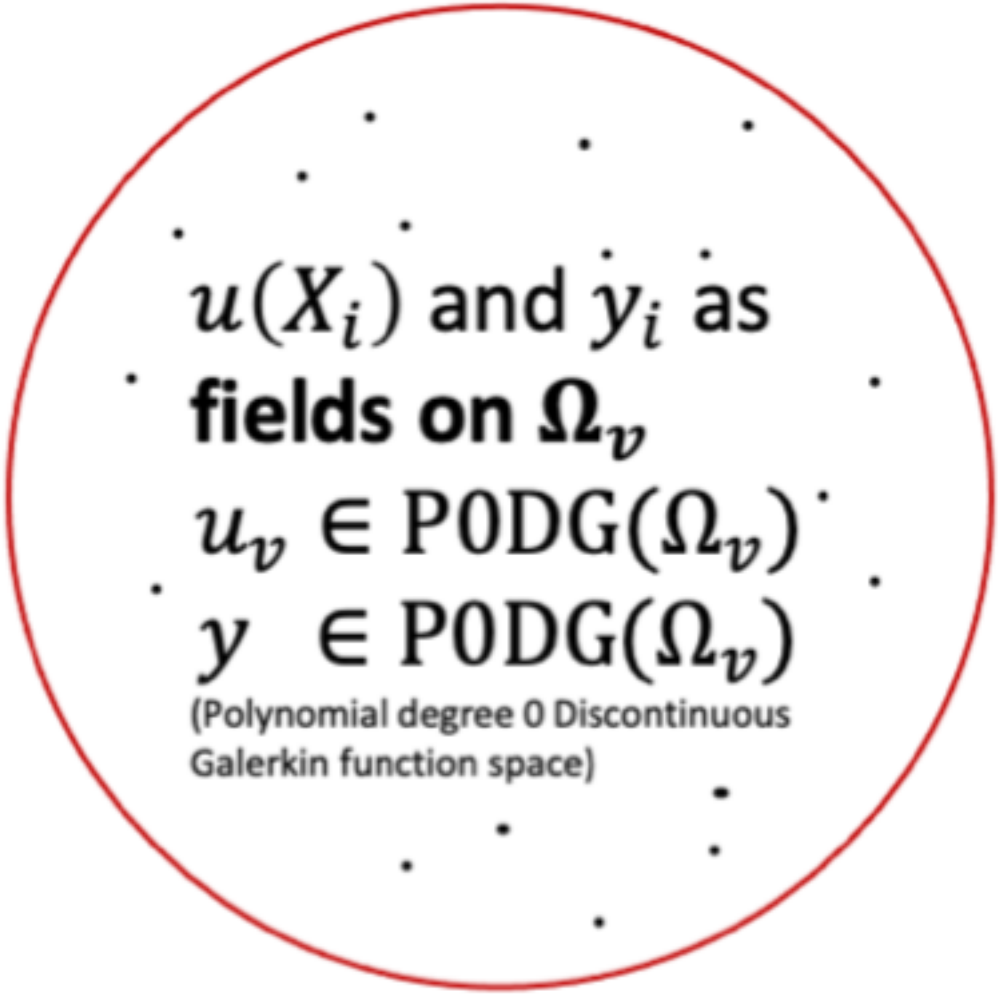
- L^2 inner product equivalent to l_2 inner product of each component (if vector or tensor valued)

$$\langle u_v, y \rangle_{L^2} = \int_{\Omega_v} u_v \, y \, dx = \cdots = \sum_i u_v(X_i) y(X_i) = \langle u_v, y \rangle_{l_2}$$

- Mass matrix is identity matrix
- Non-differentiable
- Reisz map is L^2 inner product

$$\langle u_v, \cdot \rangle_{L^2} = u_v'(\cdot)$$

- Nodal interpolation is L^2 Galerkin projection



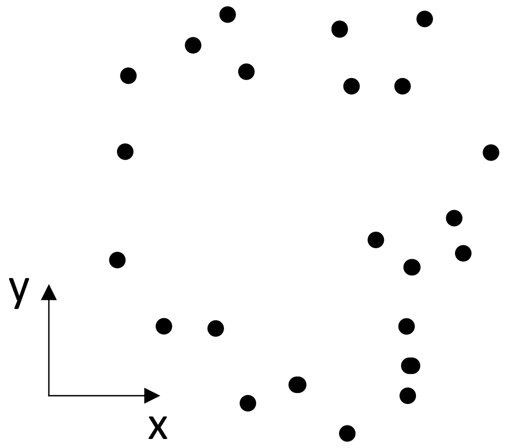
Why is this helpful in this example?

- No extra approximations necessary
- Can **be rigorous about the statistical interpretation** of data assimilation results via misfit functional (check if errors are normally distributed for example) and directly investigate different regularisations.
- **Feed back from modeller to experimenter**: can quickly model say, the impact of more measurements vs better SNR with simulated data.
- Dan and I working on a paper right now to make this point!

Future: UFL for Automated Diagnostics

- Possible approach:
  1. Read-in gridded field data as equivalent finite element field
  2. Create mesh to represent region to integrate:
     - Points - “Vertex Only Mesh”
     - lines “mesh of disconnected lines”
     - Planes “mesh of disconnected planes” etc.
  3. Calculate the diagnostic by interpolating onto the region and performing desired integration
     - Point evaluations
     - Fluxes etc.
- **Note:** Requires UFL to be extended to include interpolation operations in the language e.g. via the `interp` form.

Next Steps



Point Data

- Demonstration on real models showing advantages
- Moving points (optional)

Automated Diagnostics

- Higher dimension disconnected mesh abstractions (lines, planes and polyhedra) for interpolation onto
- Define the interpolation operations e.g. via supermeshing
- Improve dataset parsing tools
- Integration with existing tool-chains e.g. Pangeo