Using `oomph-lib` to study bifurcation phenomena in fluid flows

Andrew Hazel, Phil Haines, Matthias Heil & Rich Hewitt

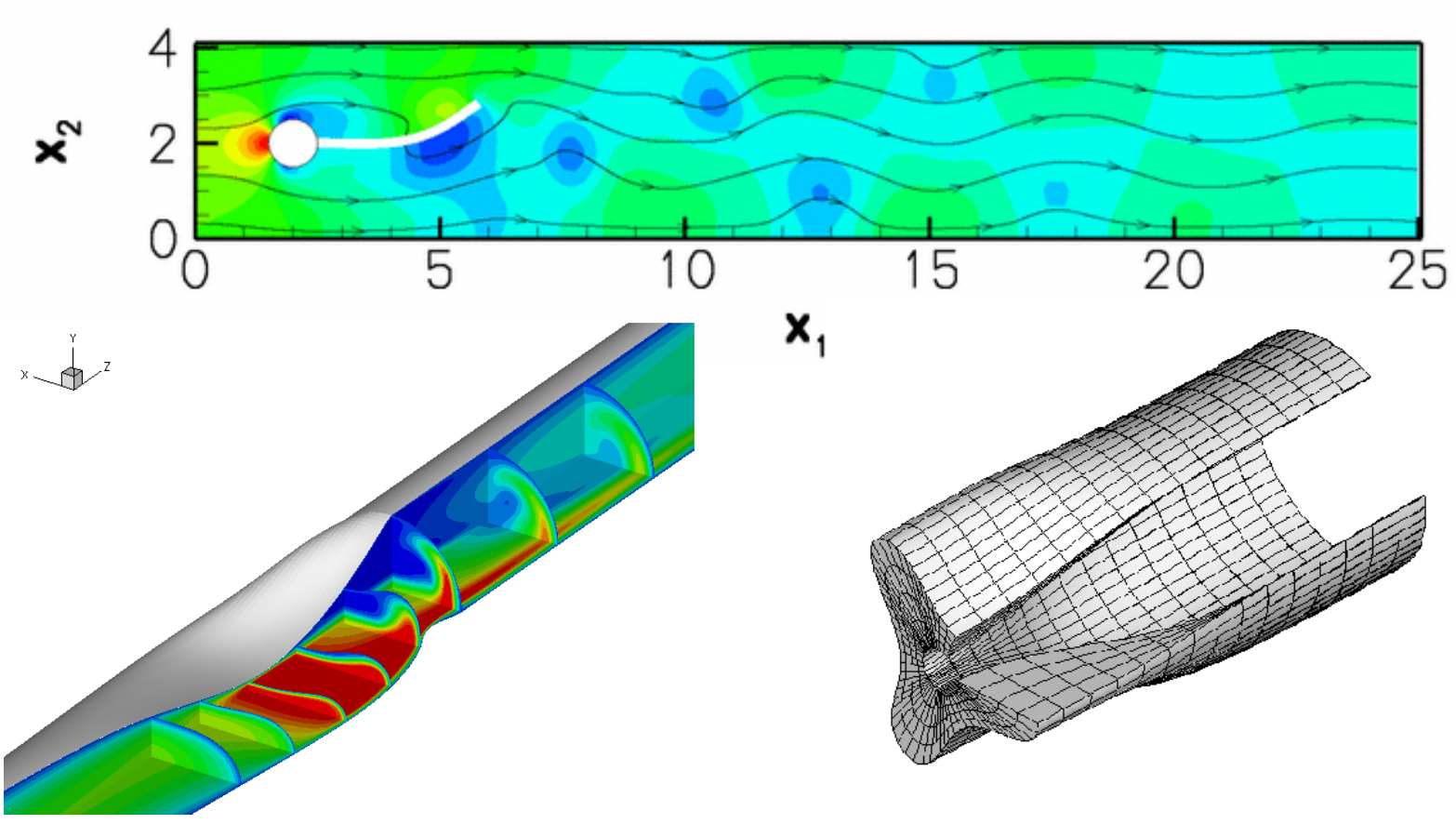
School of Mathematics
The University of Manchester

The framework

oomph-lib

the object-oriented multi-physics finite-element library

- A C++ library designed for the solution of multi-physics problems (*e. g.* fluid-structure interaction).



- OOMPH-LIB "responsible" for problem formulation

  Linear algebra performed by third-party libraries (LAPACK, Trilinos, etc)

The framework

oomph-lib

the object-oriented multi-physics finite-element library

- Design aims:
  - Robust (without being inefficient),
  - Flexible (non-standard boundary conditions),
  - "Easy-to-use" high-level interfaces,
  - Access to parallel methods (MPI), continuation, timestepping, bifurcation detection without change in problem formulation,
  - Lots of documentation and demonstration codes/problems,
  - Freely available at `http://www.oomph-lib.org`
  - Link to directory

Overall Design

- Problems (once discretised) treated as a set of algebraic equations for the unknowns $\mathbf{u}$ with parameters $\boldsymbol{\lambda}$

$$\mathbf{R}(\mathbf{u}; \boldsymbol{\lambda}) = 0.$$

Overall Design

- Problems (once discretised) treated as a set of algebraic equations for the unknowns $\mathbf{u}$ with parameters $\boldsymbol{\lambda}$

$$\mathbf{R}(\mathbf{u}; \boldsymbol{\lambda}) = 0.$$

- Problems assumed to be nonlinear and Newton's method is default solver:

$$\mathbf{u}_{\text{new}} = \mathbf{u}_{\text{old}} + \boldsymbol{\delta}\mathbf{u}, \quad \text{for} \quad \mathcal{J}\boldsymbol{\delta}\mathbf{u} = -\mathbf{R},$$

where $\mathcal{J}$ is the Jacobian matrix $\mathcal{J}_{ij} \equiv \frac{\partial R_i}{\partial u_j}$.

Overall Design

- Problems (once discretised) treated as a set of algebraic equations for the unknowns $\mathbf{u}$ with parameters $\boldsymbol{\lambda}$

$$\mathbf{R}(\mathbf{u}; \boldsymbol{\lambda}) = 0.$$

- Problems assumed to be nonlinear and Newton's method is default solver:

$$\mathbf{u}_{\text{new}} = \mathbf{u}_{\text{old}} + \boldsymbol{\delta}\mathbf{u}, \quad \text{for} \quad \mathcal{J}\boldsymbol{\delta}\mathbf{u} = -\mathbf{R},$$

  where $\mathcal{J}$ is the Jacobian matrix $\mathcal{J}_{ij} \equiv \frac{\partial R_i}{\partial u_j}$.

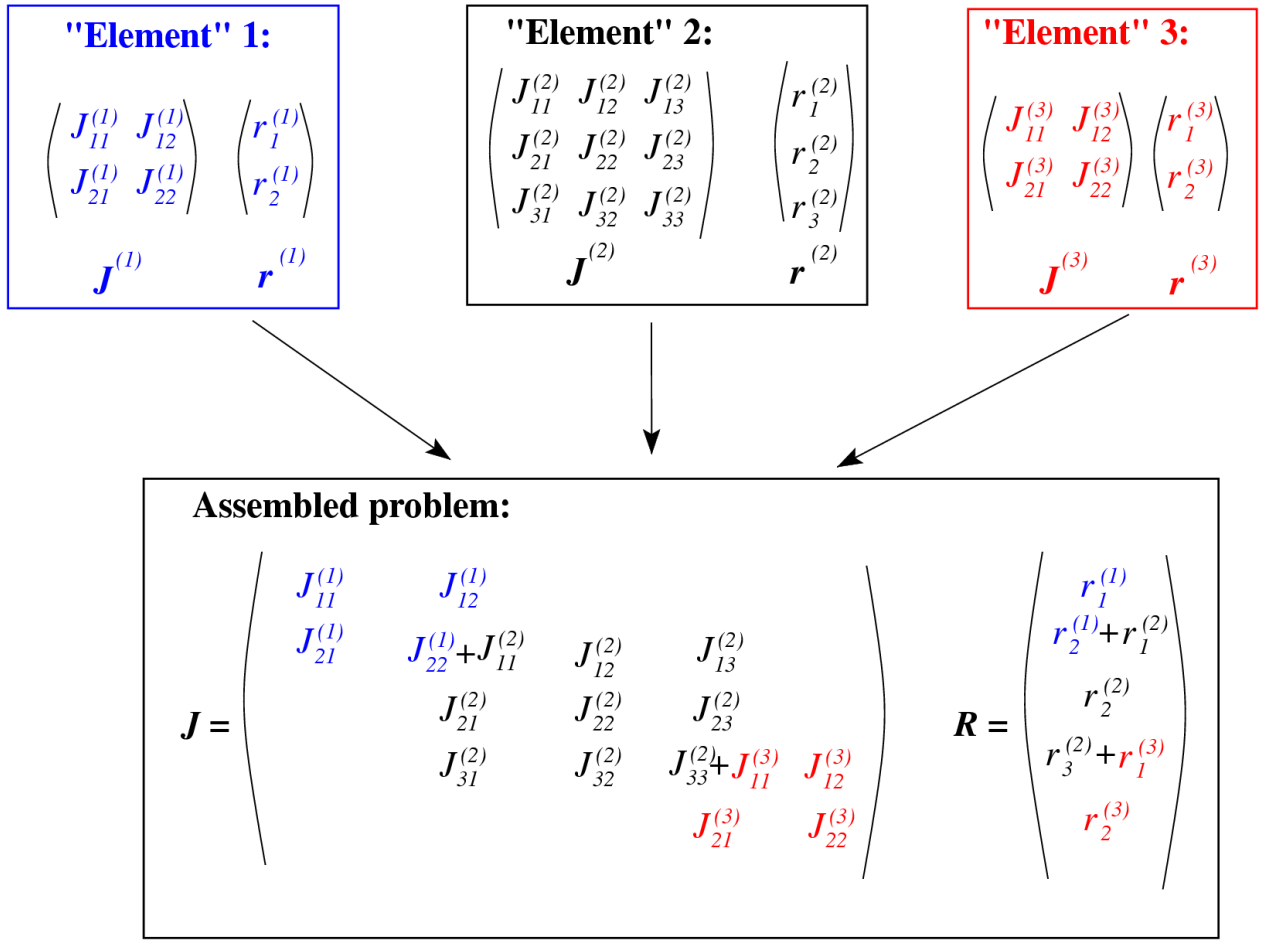
- For linear stability analysis, assume $\mathbf{u}(\mathbf{x}, t) = \mathrm{e}^{\lambda t}\,\mathbf{v}(\mathbf{x})$ and form generalised eigenproblem

$$\mathcal{J}\mathbf{v} = \lambda\mathcal{M}\mathbf{v},$$

  where $\mathcal{M}$ is the mass matrix.
  Real part of λ positive $\Rightarrow$ solution is linearly unstable.

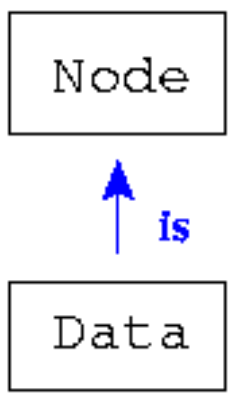
Overall Design

- Element-based framework, each "element" (can) contribute to the residual vector **R**, Jacobian and mass matrices $\mathcal{J}$ and $\mathcal{M}$.



- Could formulate the problem using a single element.

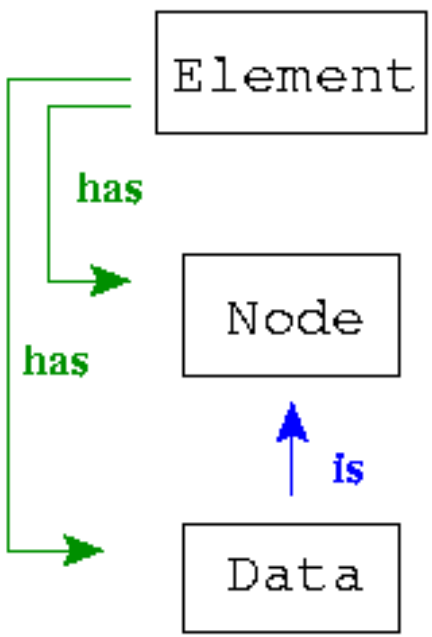
The general data structure



Stores the Eulerian position
[Note: the Eulerian position may be an unknown
and is then represented by `Data`]

Stores "values", their global equation numbers,
and associated "history values"

The general data structure

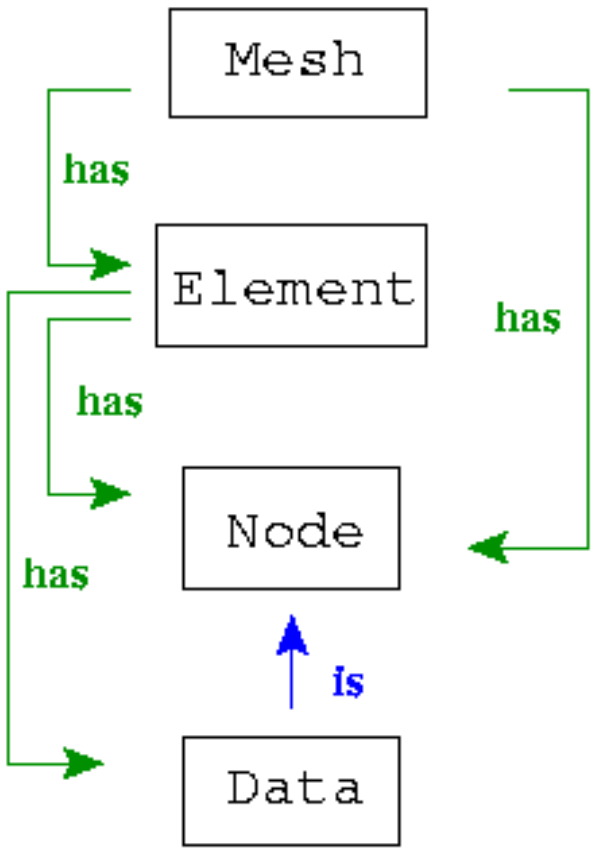


Computes the element's Jacobian matrix
and residual vector

Stores the Eulerian position
[Note: the Eulerian position may be an unknown
and is then represented by `Data`]

Stores "values", their global equation numbers,
and associated "history values"

The general data structure



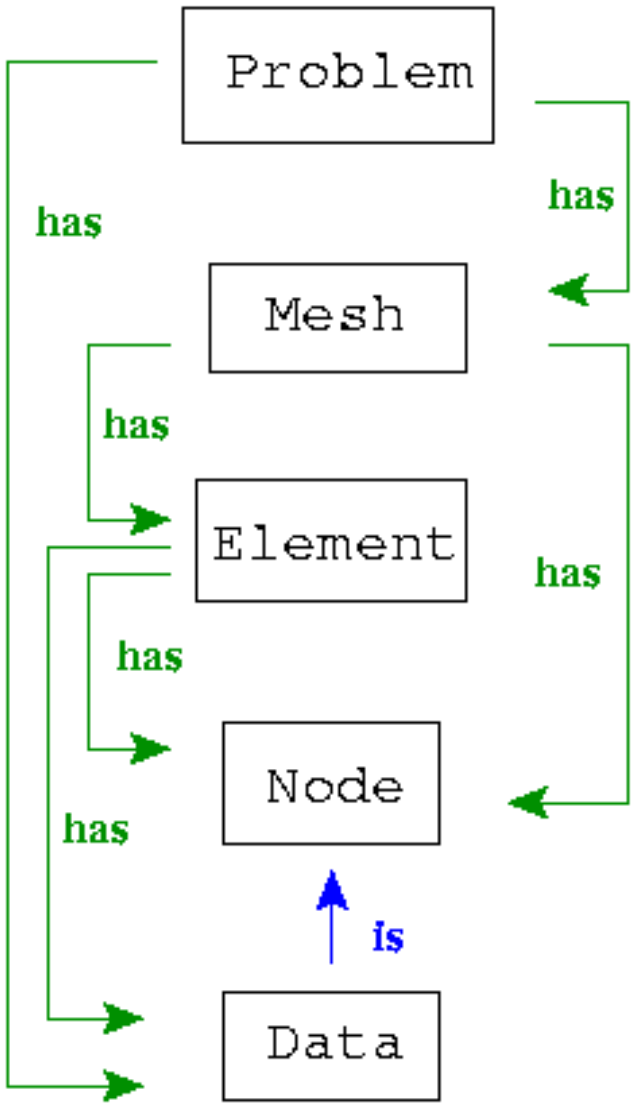
Provides ordered access to the `Elements` and `Nodes`

Computes the element's Jacobian matrix and residual vector

Stores the Eulerian position
[Note: the Eulerian position may be an unknown and is then represented by `Data`]

Stores "values", their global equation numbers, and associated "history values"

The general data structure



Implements the problem formulation; performs standard tasks such as equation numbering, time–stepping, solving, and post–processing

Provides ordered access to the `Elements` and `Nodes`

Computes the element's Jacobian matrix and residual vector

Stores the Eulerian position
[Note: the Eulerian position may be an unknown and is then represented by `Data`]

Stores "values", their global equation numbers, and associated "history values"

High-level Interfaces

- Must write your own specific `Problem` object.
  - Specify domain, equations, boundary conditions, etc.
  - Choose linear solver, eigensolver, timestepping scheme

    Sensible defaults are provided

High-level Interfaces

- Must write your own specific `Problem` object.
  - Specify domain, equations, boundary conditions, etc.
  - Choose linear solver, eigensolver, timestepping scheme

  Sensible defaults are provided
- Then use generic high-level functions, *e. g.*

```
 //Find steady solutions for different Reynolds numbers
for(int i=0;i<700;i++)
 { Re += 0.01; problem.steady_newton_solve(); }
double dt = 0.1;
//Assume system at current solution for all t-
problem.assign_initial_values_impulsive(dt);
//Unsteady simulation at current Reynolds number
for(int t=0;t<2000;t++)
 { problem.unsteady_newton_solve(dt); }
```

Bifurcation Detection

```
 //Solve steady problem
problem.steady_newton_solve();
//Find the four eigenvalues with real part near zero
problem.solve_eigenproblem(4,eigenvalues,eigenvectors);
//Assemble the augmented system associated with a
//Pitchfork bifurcation
problem.activate_pitchfork_tracking(&Re,eigenvectors[0]);
//Solve the augmented system
problem.steady_newton_solve();
//Continue the bifurcation in a second parameter
problem.arc_length_step_solve(&Gamma,ds);
```

Bifurcation Detection

- Finding the bifurcation point "exactly":
  - Solve for the unknowns and critical eigenfunction simultaneously

$$\begin{pmatrix} \mathbf{R} \\ \mathscr{J}\mathbf{v} \\ \mathbf{v}\cdot\mathbf{V}-1 \end{pmatrix} = 0,$$

  $\mathbf{V}$ is a reference vector used to ensure that the trivial solution $\mathbf{v} = 0$ is not possible.

Bifurcation Detection

- Finding the bifurcation point "exactly":
  - Solve for the unknowns and critical eigenfunction simultaneously

$$\begin{pmatrix} \mathbf{R} \\ \mathscr{J}\mathbf{v} \\ \mathbf{v}\cdot\mathbf{V}-1 \end{pmatrix} = 0,$$

    $\mathbf{V}$ is a reference vector used to ensure that the trivial solution $\mathbf{v} = 0$ is not possible.
  - `FoldHandler`, `PitchforkHandler`, `HopfHandler` overload the assembly process to form the appropriate augmented system.

Bifurcation Detection

- Finding the bifurcation point "exactly":
  - Solve for the unknowns and critical eigenfunction simultaneously

$$\begin{pmatrix} \mathbf{R} \\ \mathscr{J}\mathbf{v} \\ \mathbf{v}\cdot\mathbf{V}-1 \end{pmatrix} = 0,$$

    $\mathbf{V}$ is a reference vector used to ensure that the trivial solution $\mathbf{v} = 0$ is not possible.
  - `FoldHandler`, `PitchforkHandler`, `HopfHandler` overload the assembly process to form the appropriate augmented system.
  - (Optionally) block-decompose to reduce problem to a few linear solves with original Jacobian matrix (Fold & Pitchfork).

Bifurcation Detection

- Finding the bifurcation point "exactly":
  - Solve for the unknowns and critical eigenfunction simultaneously

$$\begin{pmatrix} \mathbf{R} \\ \mathcal{J}\mathbf{v} \\ \mathbf{v}\cdot\mathbf{V}-1 \end{pmatrix} = 0,$$

    $\mathbf{V}$ is a reference vector used to ensure that the trivial solution $\mathbf{v} = 0$ is not possible.
  - `FoldHandler`, `PitchforkHandler`, `HopfHandler` overload the assembly process to form the appropriate augmented system.
  - (Optionally) block-decompose to reduce problem to a few linear solves with original Jacobian matrix (Fold & Pitchfork).
  - For Hopf bifurcation assemble the augmented matrix

$$\begin{pmatrix} \mathcal{J} & \omega\mathcal{M} \\ -\omega\mathcal{M} & \mathcal{J} \end{pmatrix},$$

    where ω is the imaginary part (frequency) of the eigenvalue.

Handling large problems: computational approach

- Minimise the size of your problem:
  - Use error estimators* and spatial adaptivity to refine only where needed — `RefinableMesh` objects.

Handling large problems: computational approach

- Minimise the size of your problem:
  - Use error estimators* and spatial adaptivity to refine only where needed — `RefinableMesh` objects.
- Get the linear algebra right:
  - Use optimal solvers* (iterative solvers with good preconditioners) — `LinearSolver` objects.

Handling large problems: computational approach

- Minimise the size of your problem:
  - Use error estimators* and spatial adaptivity to refine only where needed — `RefinableMesh` objects.
- Get the linear algebra right:
  - Use optimal solvers* (iterative solvers with good preconditioners) — `LinearSolver` objects.
- Distribute the problem:
  - Memory is always an ultimate limit. If the problem does not fit on the computer, cannot solve it.
  - Element-by-element assembly leads to natural decompositions — hold a subset of the elements on each processor.

Spatial adaptivity

Strategy:

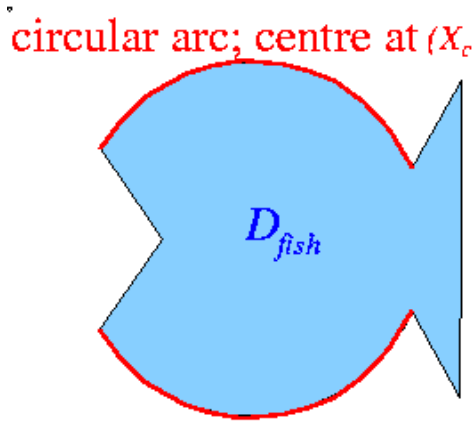
- Start with coarse mesh and refine it automatically.

Spatial adaptivity

Strategy:

- Start with coarse mesh and refine it automatically.

Example: Poisson equation in a fish-shaped domain



$$\nabla^2 u = 1 \quad \text{in } D_{fish}$$

subject to

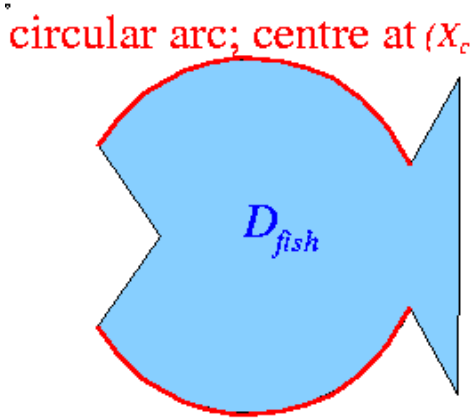
$$u = 0 \quad \text{on } \partial D_{fish}$$

Spatial adaptivity

Strategy:

- Start with coarse mesh and refine it automatically.

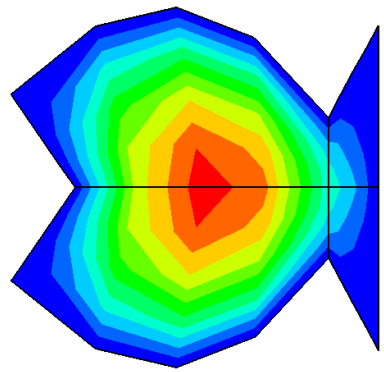
Example: Poisson equation in a fish-shaped domain



$$\nabla^2 u = 1 \quad \text{in } D_{fish}$$

subject to

$$u = 0 \quad \text{on } \partial D_{fish}$$



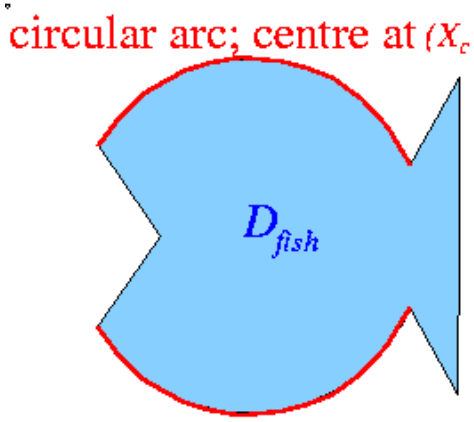
Solve

Spatial adaptivity

Strategy:

- Start with coarse mesh and refine it automatically.

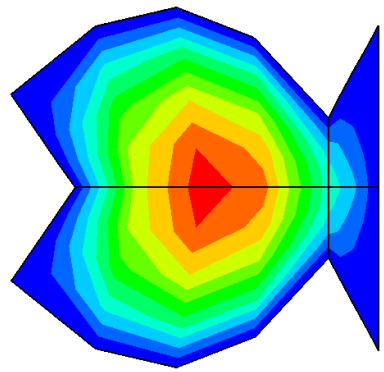
Example: Poisson equation in a fish-shaped domain

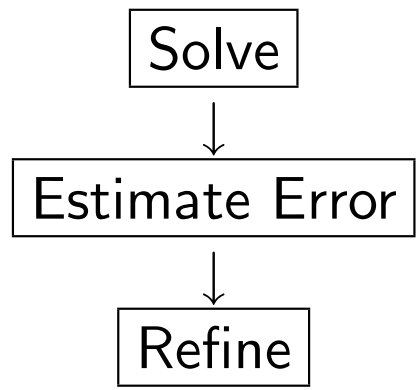


$$\nabla^2 u = 1 \quad \text{in } D_{fish}$$

subject to

$$u = 0 \quad \text{on } \partial D_{fish}$$



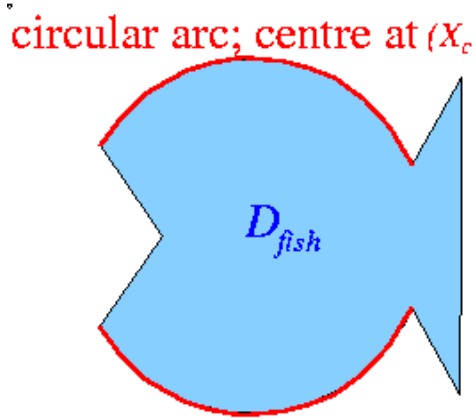


Spatial adaptivity

Strategy:

- Start with coarse mesh and refine it automatically.

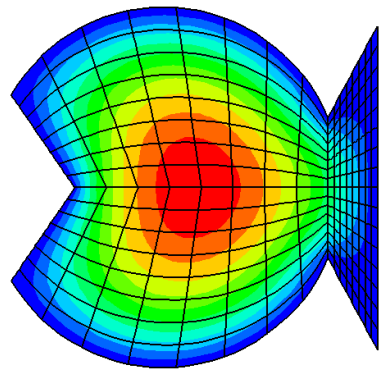
Example: Poisson equation in a fish-shaped domain



$$\nabla^2 u = 1 \quad \text{in } D_{fish}$$

subject to

$$u = 0 \quad \text{on } \partial D_{fish}$$



Solve

↓

Estimate Error

↓

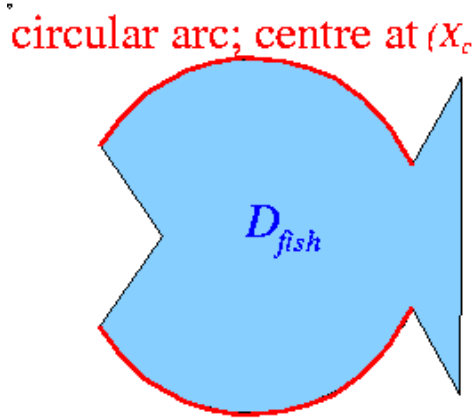
Refine

Spatial adaptivity

Strategy:

- Start with coarse mesh and refine it automatically.

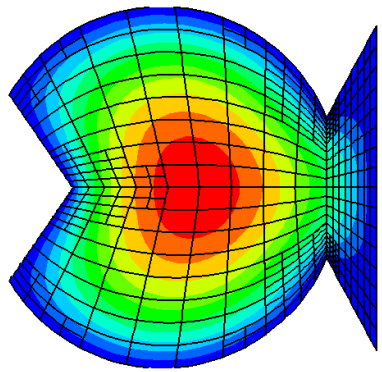
Example: Poisson equation in a fish-shaped domain



$$\nabla^2 u = 1 \quad \text{in } D_{fish}$$

subject to

$$u = 0 \quad \text{on } \partial D_{fish}$$



Solve

↓

Estimate Error

↓

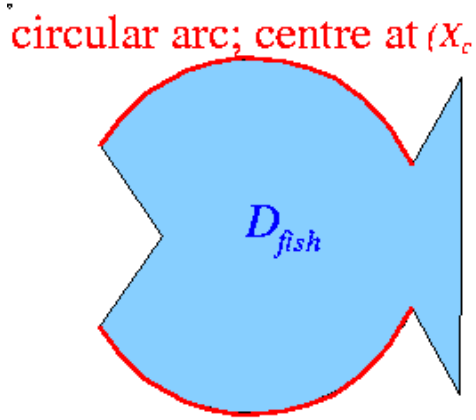
Refine

Spatial adaptivity

Strategy:

- Start with coarse mesh and refine it automatically.

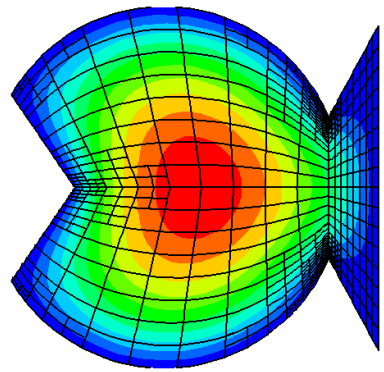
Example: Poisson equation in a fish-shaped domain



$$\nabla^2 u = 1 \quad \text{in } D_{fish}$$

subject to

$$u = 0 \quad \text{on } \partial D_{fish}$$

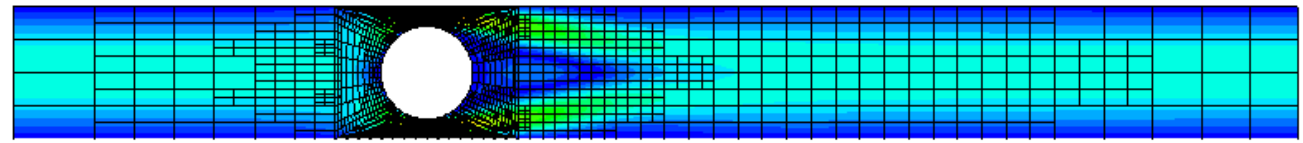


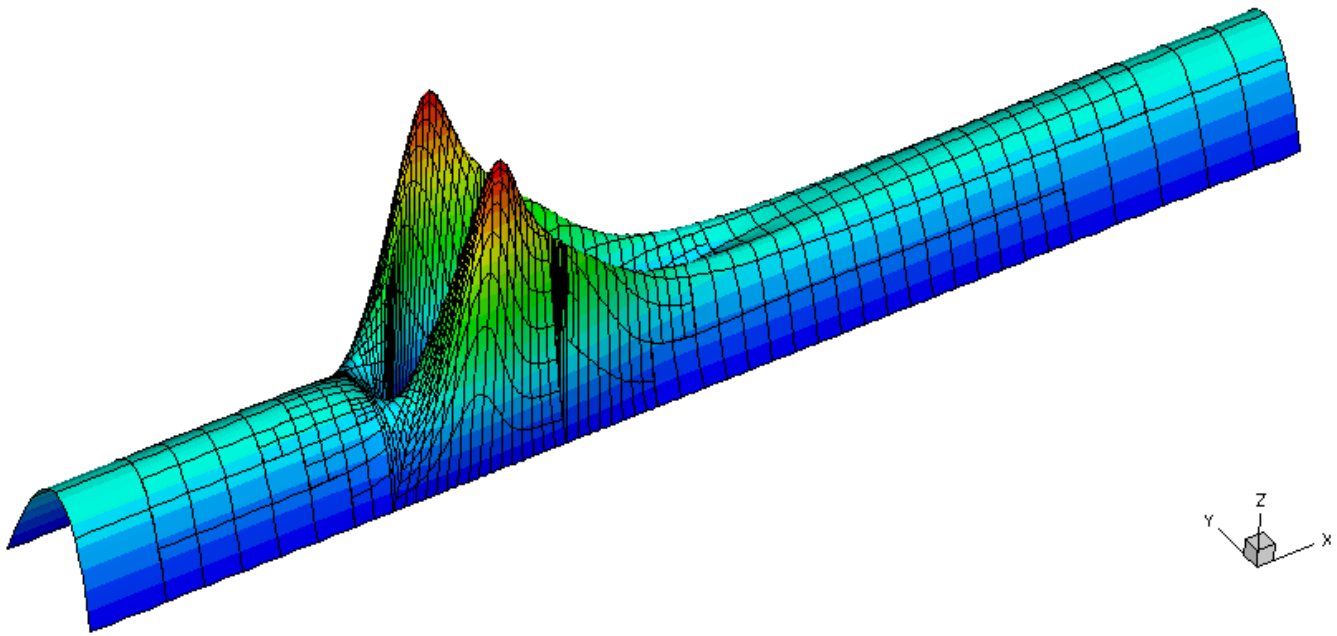
Note the resolution of curvilinear boundaries

```
problem.newton_solve(3);
```

Adaptivity and bifurcation detection

- Must accurately resolve base flow and eigenfunction.



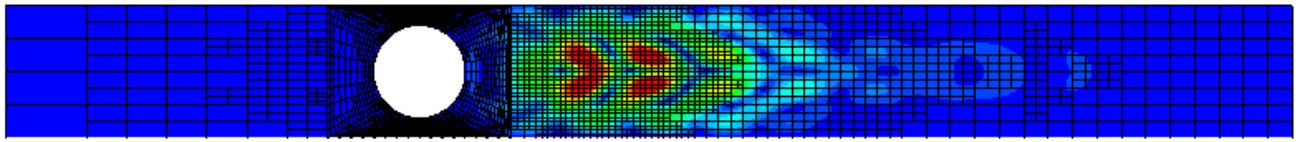


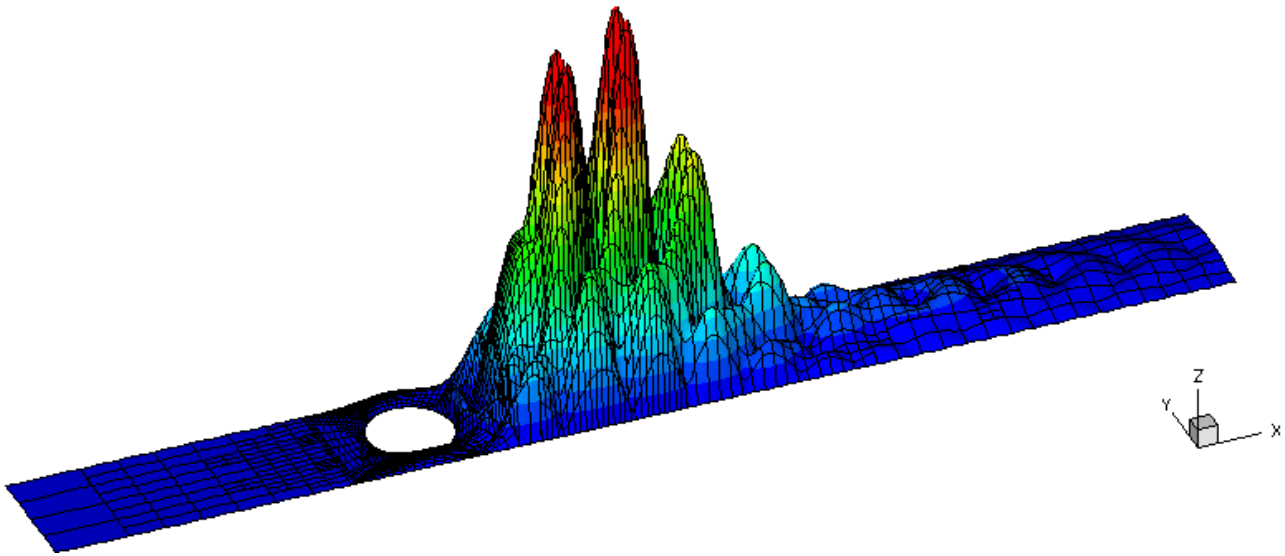
- Flow past cylinder in channel at $Re = 90$
  L = diameter of cylinder, U = inlet flow integrated over cylinder,

  c. f. Cliffe & Tavener (2004)

Adaptivity and bifurcation detection

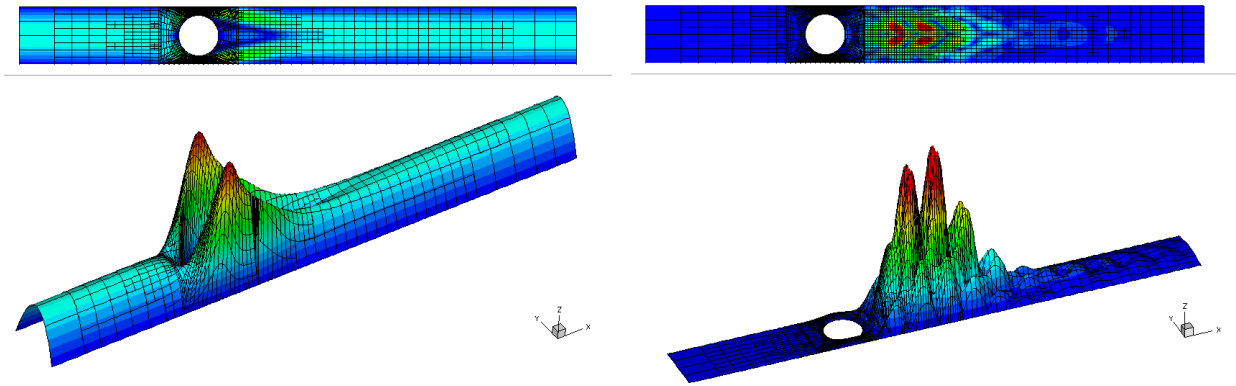
- Must accurately resolve base flow and eigenfunction.





- Critical eigenfunction associated with Hopf bifurcation at $Re \approx 92$ has very different spatial structure.

Adaptivity and bifurcation detection



- Current strategy is to refine mesh based on a weighted sum of the estimated error for base flow and critical eigenfunction.

No. of elements	Re_c	ω	Comments
1458	92.3798	0.568019	Z2 error $\approx$ 0.001 (base flow)
2826	91.9916	0.569385	
3348	91.9927	0.569382	
3366	91.9927	0.569382	combined Z2 error $<$ 0.001
14400	91.9944	0.5694	Cliffe & Tavener (2004), uniform?

Linear solvers & Problem distribution

- Problem distribution complete.
- Bifurcation detection algorithms partially parallelised.
- Block-decomposition of augmented system allows re-use of optimal solvers in detection and tracking of Fold and Pitchfork bifurcations.
  - Implementation of Elman, Silvester & Wathen's LSC preconditioner for the Navier–Stokes equations works well.
  - Resolves are not as cheap for iterative solvers vs direct solvers.
- Need good preconditioners for the augmented matrix required when tracking Hopf bifurcations

$$\begin{pmatrix} \mathcal{J} & \omega\mathcal{M} \\ -\omega\mathcal{M} & \mathcal{J} \end{pmatrix},$$

Application to similarity solutions

Similarity solutions

- In certain cases, Navier–Stokes equations can be simplified by using similarity reductions
  — assume a particular functional form, *e. g.* $u = xF(y)$.
- Such reductions typically remove spatial dimension(s) from the problem, giving ODEs instead of the original PDEs.

Similarity solutions

- In certain cases, Navier–Stokes equations can be simplified by using similarity reductions
  — assume a particular functional form, *e. g.* $u = xF(y)$.
- Such reductions typically remove spatial dimension(s) from the problem, giving ODEs instead of the original PDEs.
- ... By reducing the dimension of the problem, we reduce the number of boundary conditions.
- ... and we are making the tacit assumptions that:
  i) the domain is infinite in (at least) one dimension,
  ii) we shouldn't worry about the boundary conditions "at infinity".

Similarity solutions

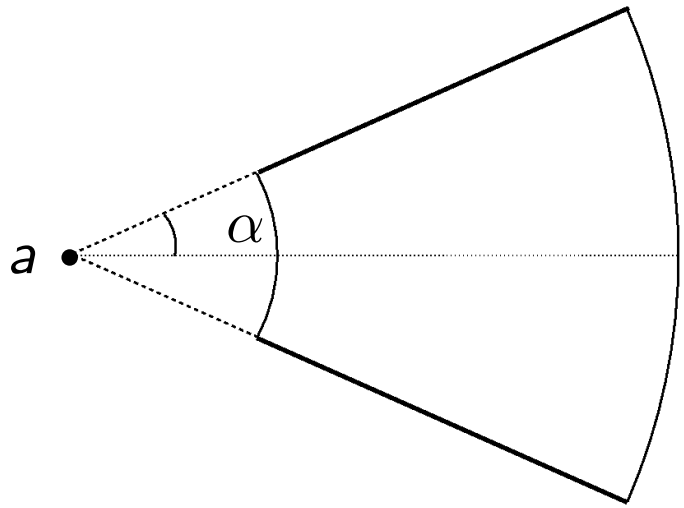
- In certain cases, Navier–Stokes equations can be simplified by using similarity reductions
  — assume a particular functional form, *e. g.* $u = xF(y)$.
- Such reductions typically remove spatial dimension(s) from the problem, giving ODEs instead of the original PDEs.
- ... By reducing the dimension of the problem, we reduce the number of boundary conditions.
- ... and we are making the tacit assumptions that:
  i) the domain is infinite in (at least) one dimension,
  ii) we shouldn't worry about the boundary conditions "at infinity".

Questions:

- Are similarity solutions a good approximation for sufficiently long physical domains?
- Do the (neglected) boundary conditions that must be applied in finite domains influence the system's dynamics?

Jeffrey–Hamel flow

- 2D flow of fluid between two non-parallel plane walls — model for flow in a channel expansion.

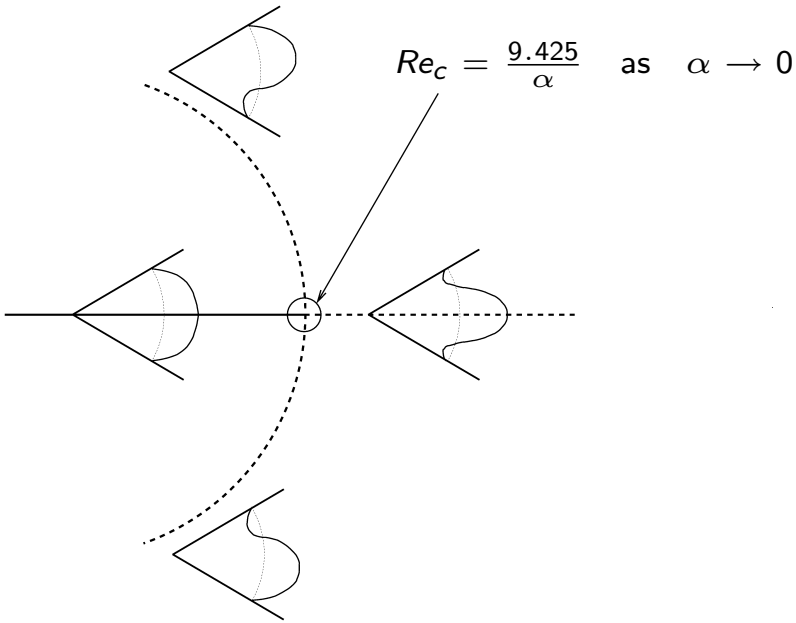


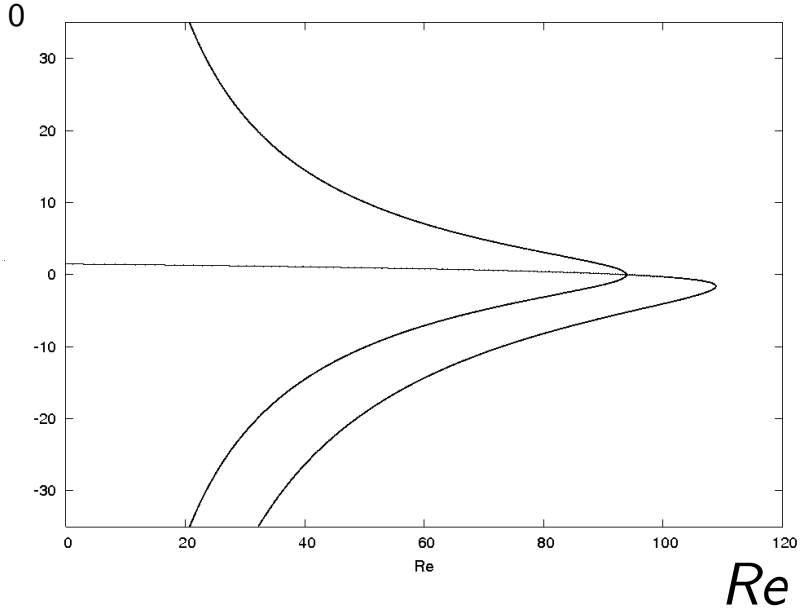
- The idealised problem of flow in an infinite wedge of semi-angle α with a source at the apex a admits a similarity solution of the form

$$\hat{\mathbf{u}} = \frac{\hat{f}(\theta)}{r}\mathbf{e}_r \qquad \text{(Jeffrey–Hamel flow).}$$

Similarity solution behaviour

- Rich dynamics in similarity solution behaviour.
- In particular, the symmetric "pure outflow" solution loses stability via a subcritical pitchfork bifurcation.





Adapted from Kerswell, Tutty & Drazin (2003)

$$\alpha = 0.1$$

- N.B. $Re \equiv \frac{Q}{\nu}$,, where Q is the radial mass flux, $\nu = \mu/\rho$ is the kinematic viscosity of the fluid.

Finite-domain effects

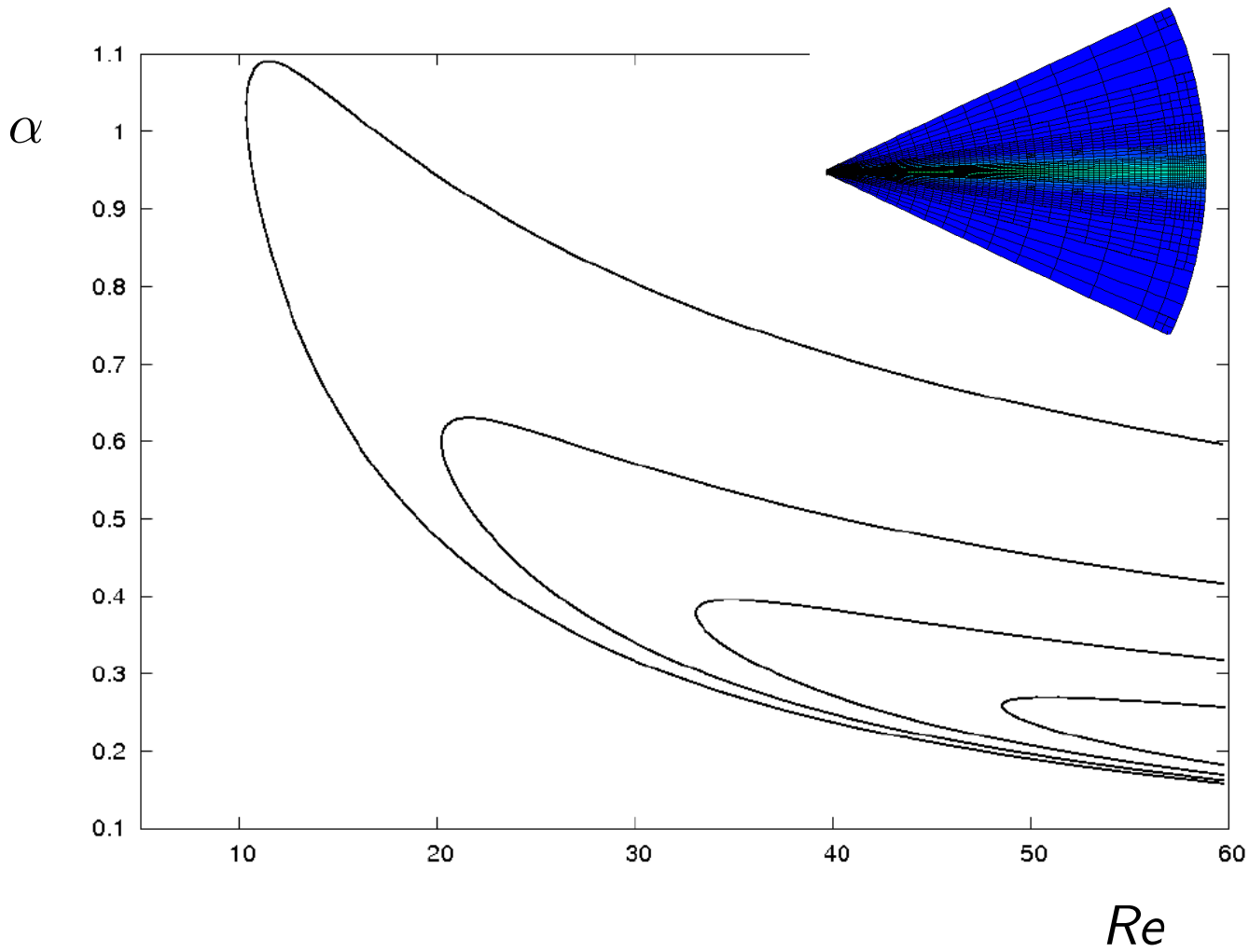
- Can actually be realised experimentally!
- Bifurcation is supercritical when flows are realised in finite domains either experimentally or numerically.
  (also true in sudden-expansion geometry)
- Yet, critical Reynolds number in good agreement with Jeffrey–Hamel solution.

Finite-domain effects

- Can actually be realised experimentally!
- Bifurcation is supercritical when flows are realised in finite domains either experimentally or numerically.
  (also true in sudden-expansion geometry)
- Yet, critical Reynolds number in good agreement with Jeffrey–Hamel solution.

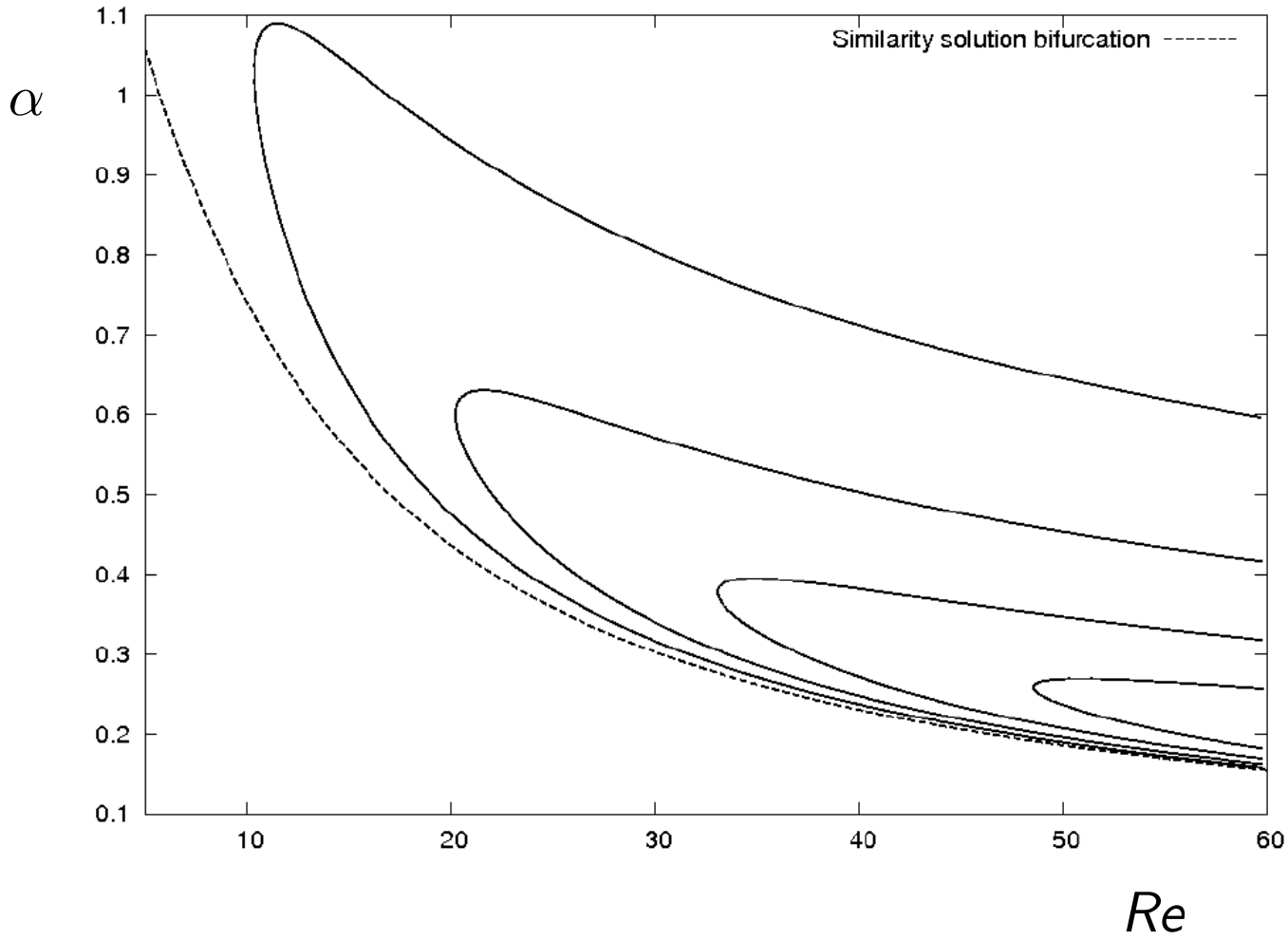
- Radius ratio fixed at 100.
- Two boundary conditions at inlet and outlet.
- Prescribe parabolic inflow with volume flux 1
  $u = \frac{3}{4\alpha r}\left(1 - (\theta/\alpha)^2\right).$
- Leave outlet pseudo-traction free.

Locus of bifurcations in the (Re,α) plane



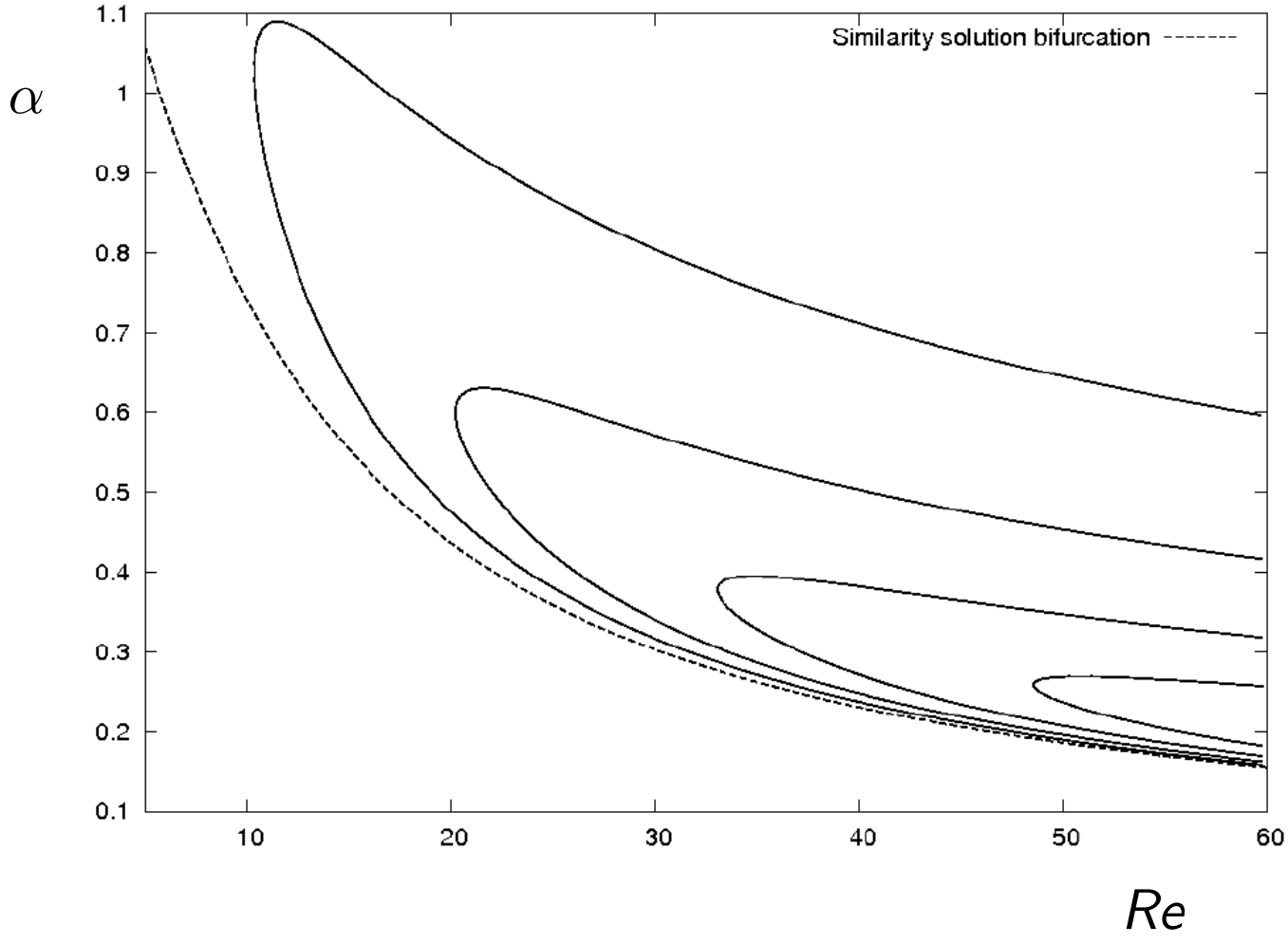
- All bifurcations are supercritical pitchforks in the Reynolds number.

Locus of bifurcations in the (Re,α) plane



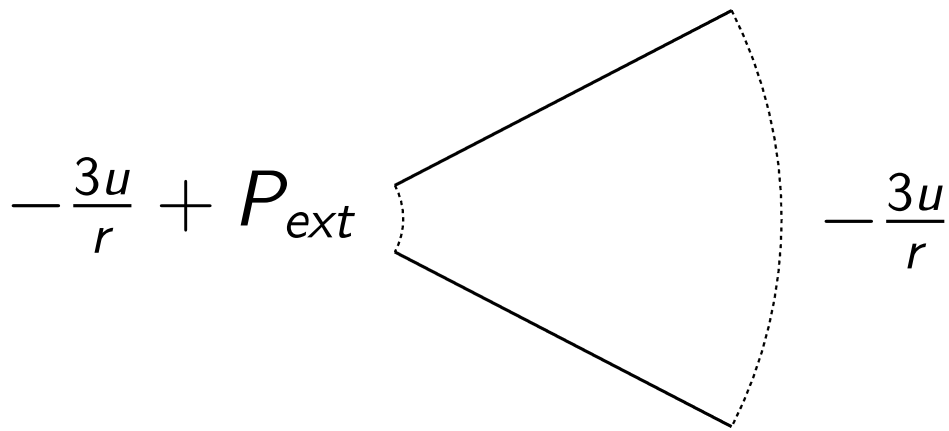
- Supercritical finite-domain pitchforks approach the subcritical similarity-solution pitchfork for small α.

Locus of bifurcations in the (Re,α) plane



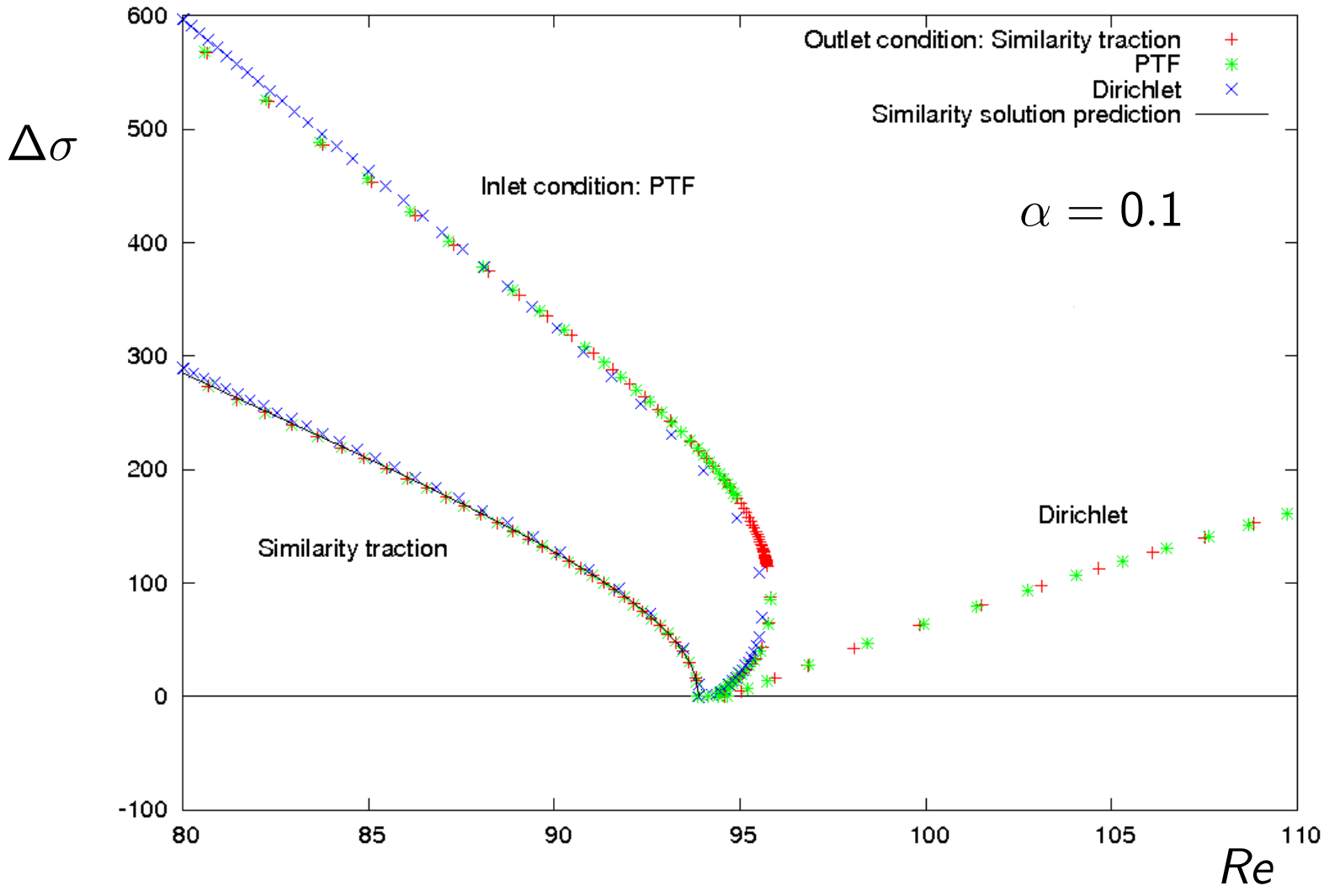
- For small α, eigenfunction over most of finite domain is indistinguishable from a similarity solution (spatial) eigenfunction, but not the critical one.

Forcing the similarity solution



- Replicate similarity solution behaviour (subcriticality) by applying consistent traction conditions at inlet and outlet.
- Specify the desired value of $-p + \frac{\partial u}{\partial r}$ along boundary.
- For the similarity solution, $-p + \frac{\partial u}{\partial r} = -\frac{3u}{r} + C$
- External pressure (P_{ext}) used to drive volume flux of one.
- Recover similarity solution behaviour. Subcritical pitchfork.

Changing boundary conditions — Inlet dominance



- Outlet condition has little effect on solution branches.

Jeffrey–Hamel Conclusions

- Critical Reynolds number in good agreement between similarity solution and finite domains for small angles (irrespective of boundary conditions).
- Critical eigenfunction close to a spatial eigenfunction of the similarity solution.
- Yet, criticality of bifurcation is determined by inflow boundary condition and is, generically, supercritical in finite domains, but subcritical in the similarity solution.
- Structure independent of radius ratio of finite domain.

Similarity Solutions — the verdict

- The base flow is usually well-approximated by the similarity solution over the majority of a finite domain, irrespective of the boundary conditions.
- For special choices of boundary conditions, similarity solutions and their bifurcation structures can be realised in finite domains.
- For more general boundary conditions, modification/suppression of the similarity solution eigenfunctions renders the similarity-solution bifurcation diagrams meaningless.
- The bifurcations in finite domains can be connected to the similarity solution bifurcations via non-trivial paths in a space spanned by the Re the length (aspect ratio) of the domain and a homotopy parameter.

Open questions

- Can we determine *a priori* when a similarity solution can give useful information about the general nonlinear dynamics?
- Temporal dynamics of the systems?
  - Rotating disk and porous channel similarity solutions undergo Hopf bifurcations. Are these realised in finite domains?
  - What about period doubling, chaos?
- Can we make theoretical progress by assuming nonlinear wave-like structures in the eigenfunctions? (*c.f.* Kerswell, Tutty & Drazin)
- Why is the symmetry-breaking bifurcation in Jeffrey–Hamel flow so insensitive to boundary conditions? (fluke?)

Acknowledgements

- Tom Wright (early studies of porous-channel flow)
- Dr. Andy Gait (problem distribution)
- Richard Muddle (block preconditioning & parallel solvers)
- Dr. Jonathan Boyle (interfaces to third-party iterative solvers)
- Cedric Ody (Young–Laplace elements)
- Renaud Schleck (octree-based refinement for 3D problems)
- ... and many others who have worked on or with OOMPH-LIB