Preprint manuscript No.
(will be inserted by the editor)

A detailed introduction to density-based topology optimisation of fluid flow problems with implementation in MATLAB

Joe Alexandersen

Received: date / Accepted: date

Abstract This article presents a detailed introduction to density-based topology optimisation of fluid flow problems. The goal is to allow new students and researchers to quickly get started in the research area and to skip many of the initial steps, often consuming unnecessarily long time from the scientific advancement of the field. This is achieved by providing a step-by-step guide to the components necessary to understand and implement the theory, as well as extending the supplied MATLAB code. The continuous design representation used and how it is connected to the Brinkman penalty approach, for simulating an immersed solid in a fluid domain, is illustrated. The different interpretations of the Brinkman penalty term and how to chose the penalty parameters are explained. The accuracy of the Brinkman penalty approach is analysed through parametric simulations of a reference geometry. The chosen finite element formulation and the solution method is explained. The minimum dissipated energy optimisation problem is defined and how to solve it using an optimality criteria solver and a continuation scheme is discussed. The included MATLAB implementation is documented, with details on the mesh, pre-processing, optimisation and post-processing. The code has two benchmark examples implemented and the application of the code to these is reviewed. Subsequently, several modifications to the code for more complicated examples are presented through provided code modifications and explanations. Lastly, the computational performance of the code is examined through studies of the computational time and memory usage, along with recommendations to decrease computational time through approximations.

Keywords topology optimisation · fluid flow · education · MATLAB

This paper is dedicated to my former supervisor and colleague Ole Sigmund, the pioneer of educational topology optimisation codes and so much more.

J. Alexandersen
Department of Mechanical and Electrical Engineering
University of Southern Denmark
Campusvej 55, DK-5230 Odense M
Tel.: +45 65507465
E-mail: joal@sdu.dk

1 Introduction

1.1 Topology optimisation

The topology optimisation method emerged from sizing and shape optimisation at the end of the 1980s in the field of solid mechanics. The homogenisation method by Bendsøe and Kikuchi (1988) is cited as being the seminal paper on topology optimisation. Topology optimisation is posed as a material distribution technique that answers the question “where should material be placed?” or alternatively “where should the holes be?”. This differentiates it from the classical disciplines of sizing and shape optimization, since it does not need an initial structure with the topology defined *a priori*. Although a range of topology optimisation approaches exists, such as the level set and phase field methods, the most popular method remains the so-called “density-based” method. The review papers by Sigmund and Maute (2013) and Deaton and Grandhi (2014) give a general overview of topology optimisation methods and applications.

Due to the maturity of the method for solid mechanics, most finite element analysis (FEA) and computer-aided design (CAD) packages now have built-in topology optimisation capabilities. Further, many public codes have been made available, often together with educational articles explaining their use. The first published code was the famous "99-line" MATLAB code by Sigmund (2001), used broadly around the world in teaching and initial experiments with topology optimisation. Subsequently, this code was updated to a more efficient "88-line" version by Andreassen et al. (2011) and most recently a hyper-efficient "neo99" version by Ferrari and Sigmund (2020). Over the years, many educational codes have been published for topology optimisation using e.g. level set methods (Challis, 2010; Wei et al., 2018), unstructured polygonal meshes (Talischi et al., 2012), energy-based homogenisation (Xia and Breitkopf, 2015), truss ground structures (Zegard and Paulino, 2014) and large-scale parallel computing (Aage et al., 2015). The recent review paper by Wang et al. (2021) provides a comprehensive overview of educational articles in the field of structural and multidisciplinary optimisation.

1.2 Fluid flow

The density-based topology optimisation approach was extended to the design of Stokes flow problems by Borrvall and Petersson (2003) and further to Navier-Stokes flow by Gersborg-Hansen et al. (2005). Since those seminal papers, topology optimisation has been applied to a large range of flow-based problems. The recent review paper by Alexandersen and Andreasen (2020) details the many contributions to this still rapidly developing field. Topology optimisation of pure fluid flow has now become reasonably mature, with only minor recent contributions for steady laminar flow (Alexandersen and Andreasen, 2020). Therefore, research efforts should be focused on building further upon this technology, calling for an educational article and a simple code that allows for quickly getting started and skipping initial repetitive implementations.

Pereira et al. (2016) presented an extension of their polygonal mesh code (Talischi et al., 2012) to Stokes flow topology optimisation. Whereas Pereira et al. (2016) focused on stable discretisations of polygonal elements for *Stokes* flow, this educational article focuses on giving a **detailed and complete introduction** to density-based topology optimisation of fluid flow problems governed by the *Navier-Stokes* equations. The aim is to serve as the first point of contact for new students and researchers to quickly get started in the research area and to skip many of the initial steps, often consuming unnecessarily long time from the scientific advancement of the field. The goal is not to give an overarching overview of the literature on the subject, so references herein are rather sparse on purpose and the reader is referred to the recent review paper instead (Alexandersen and Andreasen, 2020).

This article provides a step-by-step introduction to all the needed steps, along with a relatively simple and efficient implementation in MATLAB - where compatibility of the main files with the open-source alternative GNU Octave has been ensured. The code presented in this paper builds on the "88-line" code by Andreassen et al. (2011) and uses the same basic code structure and variable names.

1.3 Paper layout

The article is structured as follows:

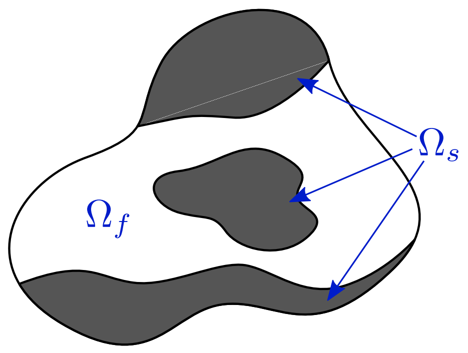
- Section 2 introduces the design representation used for density-based topology optimisation of fluid flow problems;
- Section 3 presents the physical formulation and an in-depth investigation of the error convergence for important problem parameters;
- Section 4 details the finite element discretisation and the state solution procedure used;
- Section 5 discusses the optimisation formulation, sensitivity analysis and design solution procedure;
- Section 6 introduces the two benchmark examples included in the code;
- Section 7 presents a description of the accompanying MATLAB implementation;
- Section 8 shows results obtained using the code for the benchmark examples;
- Section 9 discusses modifications to the code for solving more advanced problems;
- Section 10 discusses the computational time and memory usage;
- Section 11 presents concluding remarks.

2 Design representation

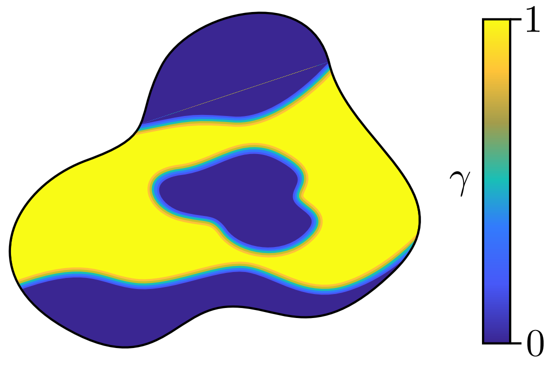
This section describes the design representation introduced to model solid domains immersed in a fluid, allowing for density-based topology optimisation. First, the true separated discrete domains that should be represented are introduced, and secondly, a continuous design representation is described.

2.1 Discrete separate domains

Figure 1a shows an arbitrary example of a truly discrete problem with separated fluid and solid domains, denoted as Ω_f and Ω_s, respectively. Both Ω_f and Ω_s can



(a) Discrete separation of domains



(b) Continuous design representation

Fig. 1: Arbitrary computational domain consisting of fluid and solid domains.

be comprised as the union of non-overlapping subdomains, as illustrated for Ω_s in Figure 1a. The interface between the solid and fluid domains is clearly defined due to the separation of the two domains.

2.2 Continuous relaxed representation

In order to perform density-based topology optimisation using gradient-based methods, the discrete representation must be relaxed. Firstly, a spatially-varying field, $\gamma(\mathbf{x})$, is introduced, which will be termed the design field. It is equivalent to the characteristic function of the fluid domain Ω_f:

$$\gamma(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} \in \Omega_f \\ 0 & \text{if } \mathbf{x} \in \Omega_s \end{cases} \tag{1}$$

Secondly, the design field is relaxed from binary values to a continuous field, allowing intermediate values: $\gamma(\mathbf{x}) \in [0;1]$. Figure 1b shows a continuous relaxed representation of the discrete separated domains of Figure 1a. The continuous representation has an implicit representation of the solid-fluid interface, represented by a transition region of intermediate design field values as shown in Figure 1b. Generally, during the optimisation process there will be many regions of intermediate values, whereas a converged design will have significantly less, if any – depending on various details as will be discussed later. By coupling the continuous design representation to coefficients of the governing equations, it is possible to model an immersed geometry which is implicitly defined by the design field.

3 Physical formulation

This section presents the physical formulation of the fluid flow problems treated in this article. The Navier-Stokes equations are the governing equations and a so-called Brinkman penalty term is introduced to facilitate density-based topology optimisation.

3.1 Governing equations

This article restricts itself to steady-state laminar and incompressible flow, governed by the Navier-Stokes equations:

$$\rho u_j \frac{\partial u_i}{\partial x_j} - \mu \frac{\partial}{\partial x_j}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) + \frac{\partial p}{\partial x_i} - f_i = 0 \tag{2a}$$

$$\frac{\partial u_i}{\partial x_i} = 0 \tag{2b}$$

where u_i is the i-th component of the velocity vector $\mathbf{u}$, p is the pressure, ρ is the density, μ is the dynamic viscosity, and f_i is the i-th component of the body force vector $\mathbf{f}$. The Reynolds number is commonly used to describe a flow and is defined as:

$$Re = \frac{\rho U L}{\mu} \tag{3}$$

where U and L are the reference velocity and length-scale, respectively. The Reynolds number describes the ratio of inertial to viscous forces in the fluid: if $Re < 1$, the flow is dominated by viscous diffusion; if $Re > 1$, the flow is dominated by inertia.

3.2 Brinkman penalty term

In order to facilitate topology optimisation, an artificial body force is introduced, termed the Brinkman penalty term:

$$f_i = -\alpha u_i \tag{4}$$

which represents a resistance term and a momentum sink, drawing energy from the flow. The Brinkman penalty factor, α, is a spatially-varying parameter defined as follows for the separated domains introduced in Figure 1a:

$$\alpha(\mathbf{x}) = \begin{cases} 0 & \text{if } \mathbf{x} \in \Omega_f \\ \infty & \text{if } \mathbf{x} \in \Omega_s \end{cases} \tag{5}$$

where the 0 in the fluid domain recovers the original equations and the infinite value inside the solid domain theoretically ensures identically zero velocities. This allows for extending the Navier-Stokes equations to the entire computational domain:

$$\left.\begin{aligned} \rho u_j \frac{\partial u_i}{\partial x_j} - \mu \frac{\partial}{\partial x_j}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) \\ + \frac{\partial p}{\partial x_i} + \alpha\left(\gamma(\mathbf{x})\right) u_i = 0 \\ \frac{\partial u_i}{\partial x_i} = 0 \end{aligned}\right\} \text{for } \mathbf{x} \in \Omega \tag{6}$$

where solid domains are modelled as immersed geometries through the Brinkman penalty term.

The Brinkman penalty term can be interpreted in three different ways:

1. Volumetric penalty term to ensure zero velocities in the solid regions
2. Friction term from out-of-plane viscous resistance
3. Friction term from an idealised porous media

The first interpretation is a purely algorithmic approach, where the only purpose is to simulate an immersed fully solid geometry inside a fluid domain. The second and third are both physical interpretations, but require different lower and upper bounds of Equation 5 formulated based on physical parameters. The second interpretation is limited to only two-dimensional problems, where the computational domain has a finite and relatively small out-of-plane thickness. The first and third interpretations can be applied to three-dimensional problems.

The seminal work by Borrvall and Petersson (2003) for Stokes flow and Gersborg-Hansen et al. (2005) for Navier-Stokes flow, both introduced a design parametrisation based on the out-of-plane channel height for two-dimensional problems following the second interpretation above. Both works also note the comparison to the third interpretation and Gersborg-Hansen et al. (2005) argues for the first interpretation as the ultimate goal of topology optimisation for fluid flow. The recent review paper by Alexandersen and Andreasen (2020) shows that the dominant interpretation is the first, a purely algorithmic approach to ensure zero velocities. This is because it easily extends to three-dimensions and no

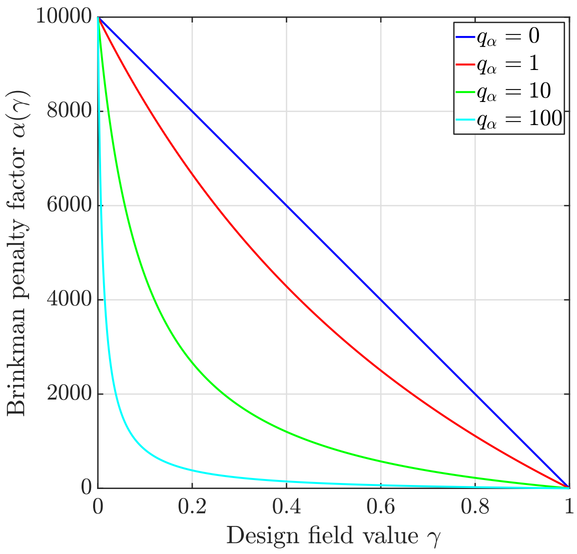


Fig. 2: Interpolation of α for various q_α.

physical reflections are required. However, as it will be discussed in Sections 3.2.2 and 3.2.3 and shown in Section 3.3, it is highly relevant to couple the parameters to the physical interpretations to ensure correct scaling of the equations.

3.2.1 Numerical relaxation

In practise, the bounds of the Brinkman penalty factor are defined as follows, based on the design field, γ, introduced in Figure 1b:

$$\alpha(\gamma(\mathbf{x})) = \begin{cases} \alpha_{\min} & \text{if } \gamma(\mathbf{x}) = 1 \\ \alpha_{\max} & \text{if } \gamma(\mathbf{x}) = 0 \end{cases} \tag{7}$$

For numerical reasons, a finite value must be used in the solid domain, $\alpha_{\max}$, being large enough to ensure negligible velocities but small enough to ensure stability. The choice of this maximum value will be discussed in Sections 3.2.3 and 3.3. Furthermore, if two-dimensional problems with a finite out-of-plane thickness are treated, a minimum value, $\alpha_{\min}$, should remain to account for the out-of-plane viscous resistance as discussed in Section 3.2.2.

In order to interpolate between solid and fluid domains, the Brinkman penalty factor is interpolated using the following function:

$$\alpha(\gamma) = \alpha_{\min} + (\alpha_{\max} - \alpha_{\min})\frac{1-\gamma}{1+q_\alpha \gamma} \tag{8}$$

where q_α is a parameter determining the shape of the interpolation, as illustrated in Figure 2 for $\alpha_{\min} = 0$ and $\alpha_{\max} = 10^4$. This function is not the same as introduced by Borrvall and Petersson (2003), but rather the Rational Approximation of Material Properties (RAMP) function (Stolpe and Svanberg, 2001). This ensures a linear interpolation is directly achieved by setting q_α to 0, whereas the function used by Borrvall and Petersson (2003) only has the linear case as an asymptotic limit. Effectively, q_α herein is the inverse of the q used by Borrvall and Petersson (2003).

3.2.2 Minimum penalty factor

The seminal work by Borrvall and Petersson (2003) for Stokes flow and Gersborg-Hansen et al. (2005) for Navier-Stokes flow, introduced topology optimisation of fluid flow problems using a design parametrisation based on the out-of-plane channel height for two-dimensional problems. By assuming parallel plates distanced $2h$ apart and a fully-developed pressure-driven flow (Poiseuille flow), an out-of-plane parabolic velocity profile is assumed between the two plates. This allows for analytical through-thickness integration, yielding an additional term to the momentum equations from the out-of-plane viscous resistance in the form of Equation 4. When treating two-dimensional problems with a finite out-of-plane thickness and lateral no-slip walls, the minimum Brinkman penalty must be set to:

$$\alpha_{\min} = \frac{5\mu}{2h^2} \tag{9}$$

where h is half of the domain thickness. However, when treating three-dimensional problems, the minimum Brinkman penalty factor should be set to 0 to recover the original unencumbered Navier-Stokes momentum equations.

3.2.3 Maximum penalty factor

Finding the correct maximum penalty factor is not trivial. It must be large enough to ensure negligible flow through solid regions, but small enough to ensure numerical stability. While the Brinkman term is often seen purely as a volumetric penalty approach for imposing the no-flow conditions inside the solid, it is beneficial to ensure consistent dimensional scaling of the penalty factor. This can be done by relating it to the friction factor from a fictitious porous media:

$$\alpha_{\max} = \frac{\mu}{\kappa} \tag{10}$$

where κ is the permeability of the fictitious porous media. This permeability must then be small enough to ensure negligible velocities inside the solid regions, but as will be shown in Section 8.1.3, the minimum dissipated energy optimisation is pretty forgiving. For a given constant permeability, Equation 10 ensures an almost constant ratio between the order of magnitude for the velocities in the flow and solid regions for a range of Reynolds numbers, as demonstrated in Section 3.3.

3.2.4 Note on physical interpretations

It is easily seen from Equations 9 and 10, that the two physical interpretations are analogous by connecting the permeability to the out-of-plane thickness:

$$\kappa = \frac{2h^2}{5} \tag{11}$$

as noted by both Borrvall and Petersson (2003) and Gersborg-Hansen et al. (2005).

3.2.5 Dissipated energy

This article focuses on minimising the dissipated energy by a flow through a channel structure. The dissipated energy due to viscous resistance is defined as:

$$\phi = \frac{1}{2}\int_{\Omega_f} \mu \frac{\partial u_i}{\partial x_j}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) dV \tag{12}$$

The dissipated energy is often used as the objective functional of topology optimisation and it is, thus, relevant to investigate the accuracy of this measure when using the Brinkman penalty method.

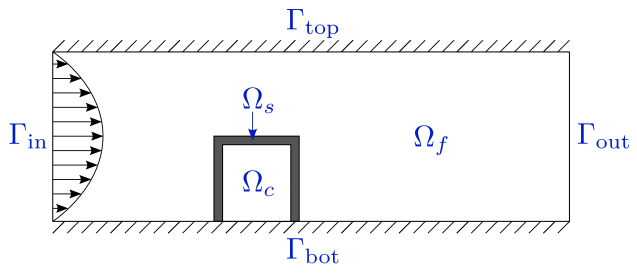
3.3 Effect of varying parameters

This section presents the effect of varying the Reynolds number Re and Brinkman penalty factor α on the accuracy of the Brinkman-penalised Navier-Stokes equations.

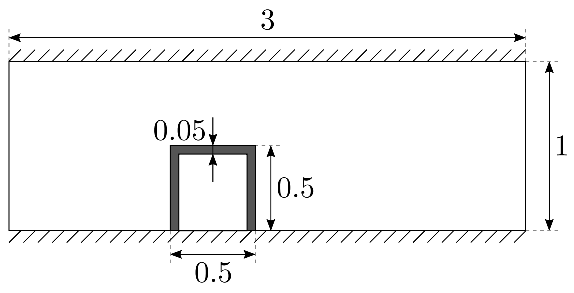
3.3.1 Example setup

Figure 3 presents the problem setup used to explore the accuracy of the Brinkman-penalised Navier-Stokes equations. A hollow box obstacle is placed inside a channel, where a fully-developed parabolic flow with a maximum of U_{in} enters at the left-hand side inlet and exits at the right-hand side zero normal-stress outlet. This model will be solved using both a discrete design representation, modelling only the fluid domain, and a continuous design representation, with the Brinkman-penalised Navier-Stokes equations in the entire computational domain. The same locally-refined mesh is used for both models to ensure that the discretisation error does not affect the comparison in a significant sense.

The porous media definition in Equation 10 is chosen for the Brinkman penalty factor and the problem is non-dimensionalised by setting the inlet velocity to $U_{\text{in}} = 1$, the density to $\rho = 1$, the dynamic viscosity to $\mu = {}^1\!/_{Re}$, and the permeability to $\kappa = Da$. The Darcy number, $Da = \frac{\kappa}{L^2}$, is the non-dimensional permeability. The Reynolds and Darcy numbers will be varied.



(a) Domain and boundary definitions



(b) Dimensionless lengths

Fig. 3: Problem setup for the example used in Section 3.3.

3.3.2 Fluid solution

Figure 4 shows the velocity magnitude and pressure fields using the discrete design representation for $Re = 20$. Since the fluid cavity inside the solid structure is disconnected, the pressure is here set equal to a reference pressure of 0. It can be seen that the flow moves above the obstacle and leaves a re-circulation zone behind it.

Figure 5 shows the velocity magnitude field using the Brinkman approach for $Re = 20$ and $Da \in \{10^{-4}, 10^{-6}, 10^{-8}\}$. For the highest permeability case, $Da = 10^{-4}$, it is seen that a significant amount of fluid passes through the solid structure and clearly non-zero velocities exist in the cavity. When this is increased to $Da = 10^{-6}$, the flow solution becomes more accurate, but the recirculation zone behind the obstacle is not fully captured. This is improved for the last case, $Da = 10^{-8}$, but even here flow is passing through the obstacle. This will always be the case for the Brinkman penalty approach, but the velocities can be driven towards zero by decreasing the permeability sufficiently.

Figure 6 shows the pressure field using the Brinkman approach for $Re = 20$ and $Da \in \{10^{-4}, 10^{-6}, 10^{-8}\}$. Due to the continuous nature of the design representation, pressure gradients exist inside the solid structure and a non-zero pressure exists in the cavity. The cavity pressure approximately becomes the mean of the high and low surface pressures - for this case $\approx (4+0)/2 = 2$. For fluid dissipation problems, this is not really an issue because this pressure is not important for the overall dissipation and since disconnected fluid regions never occur in the optimised designs. But for fluid-structure-interaction, where the fluid exerts a pressure on the solid, the internal non-zero pressure can be an issue (Lundgaard et al., 2018).

3.3.3 Error convergence study

To clearly show the convergence of the approximate solution from the continuous representation towards the true solution from the discrete representation, a number of error measures are computed. The spatial average of the velocity magnitude inside the solid structure and inside the cavity are computed as:

$$\varepsilon_{u,\text{s}} = \frac{1}{|\Omega_s|} \int_{\Omega_s} \|\mathbf{u}\|_2 \, dV \tag{13a}$$

$$\varepsilon_{u,\text{c}} = \frac{1}{|\Omega_c|} \int_{\Omega_c} \|\mathbf{u}\|_2 \, dV \tag{13b}$$

These are seen as directly equivalent to a relative error measure, since the true solution for these quantities are zero and the reference (inlet) value is 1. Furthermore, the relative error is found for the dissipated energy and the pressure drop:

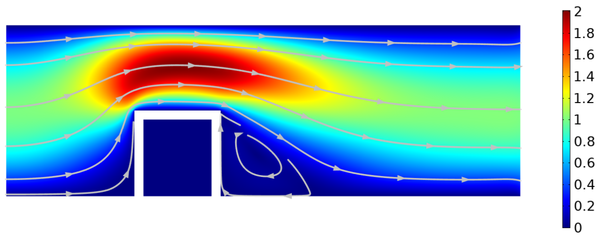
$$\varepsilon_{\phi} = \frac{|\phi_{\text{disc}} - \phi_{\text{cont}}|}{\phi_{\text{disc}}} \tag{14a}$$

$$\varepsilon_{\Delta p} = \frac{|\Delta p_{\text{disc}} - \Delta p_{\text{cont}}|}{\Delta p_{\text{disc}}} \tag{14b}$$

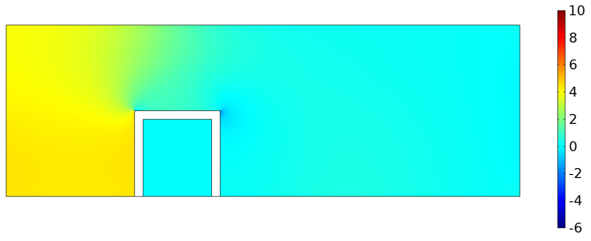
where subscripts ‘disc’ and ‘cont’ denote discrete and continuous design representations, respectively, ϕ is the dissipated energy of the fluid subdomain defined in Equation 12 and Δp is the pressure drop from inlet to outlet as computed from:

$$\Delta p = \frac{1}{|\Gamma_{\text{in}}|} \int_{\Gamma_{\text{in}}} p \, dS - \frac{1}{|\Gamma_{\text{out}}|} \int_{\Gamma_{\text{out}}} p \, dS \tag{15}$$

Figure 7 shows the error measures as a function of the Darcy number for $Re = 20$. It is clearly seen that the velocity error measures converge at an order of 1 and the pressure and dissipation measures at an order of ½. This shows that compared to the velocity field, the pressure field needs a significantly higher Brinkman penalty factor to reach a certain accuracy. The dissipated energy also requires a larger Brinkman penalty factor to achieve a certain accuracy. However, this is not necessarily important for minimum dissipated energy optimisation as will be demonstrated in Section 8.1.3. Additional accuracy becomes important if these

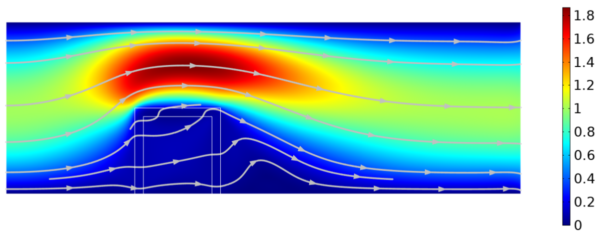


(a) Velocity magnitude

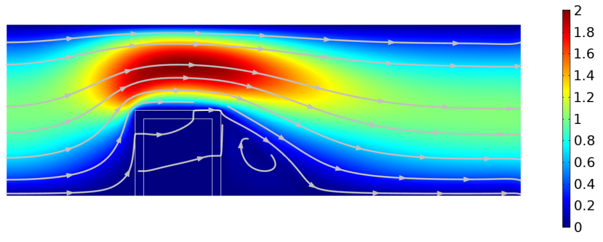


(b) Pressure field

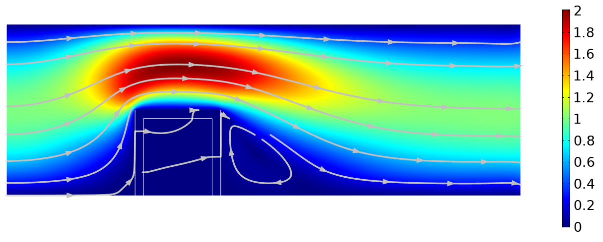
Fig. 4: Velocity magnitude and pressure fields using the discrete design representation for $Re = 20$.



(a) $Da = 10^{-4}$

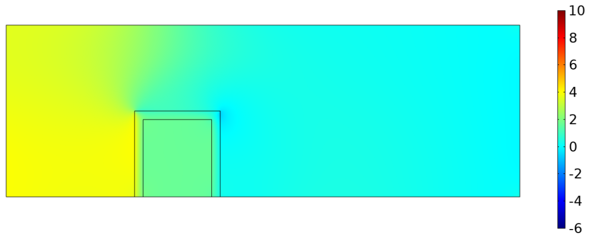


(b) $Da = 10^{-6}$

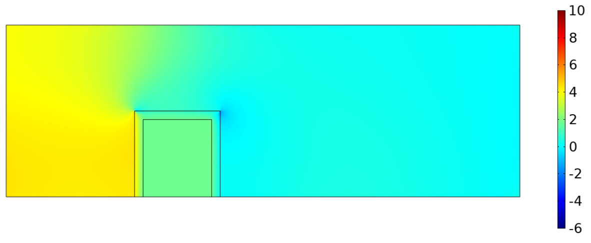


(c) $Da = 10^{-8}$

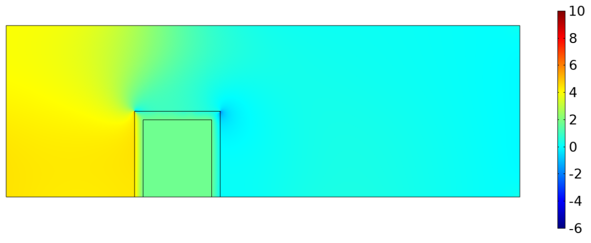
Fig. 5: Velocity magnitude fields using the Brinkman approach for $Re = 20$ and $Da \in \{10^{-4}, 10^{-6}, 10^{-8}\}$.



(a) $Da = 10^{-4}$



(b) $Da = 10^{-6}$



(c) $Da = 10^{-8}$

Fig. 6: Pressure fields using the Brinkman approach for $Re = 20$ and $Da \in \{10^{-4}, 10^{-6}, 10^{-8}\}$.

functionals are used as constraints where the limit represents a physical value. The constraint looses its physicality, if the functional is not resolved to a large enough accuracy[1]. However, if treated in relative terms as in Section 9.2.2, then this is less of an issue. The accuracy of the pressure field is also very important if it is used to drive additional physics, such as fluid-structure interaction (FSI) which was clearly shown by Lundgaard et al. (2018).

Figure 8 shows the error measures as a function of the Reynolds number for $Da = 10^{-6}$. It can be seen that defining the Brinkman penalty factor using Equation 10 ensures a constant error for a range of Reynolds numbers. However, as the Reynolds number

[1] This is very similar to stress constraints in structural topology optimisation, where a maximum stress limit becomes meaningless, unless the stresses and aggregated maximum is accurate Le et al. (2010); da Silva et al. (2019).

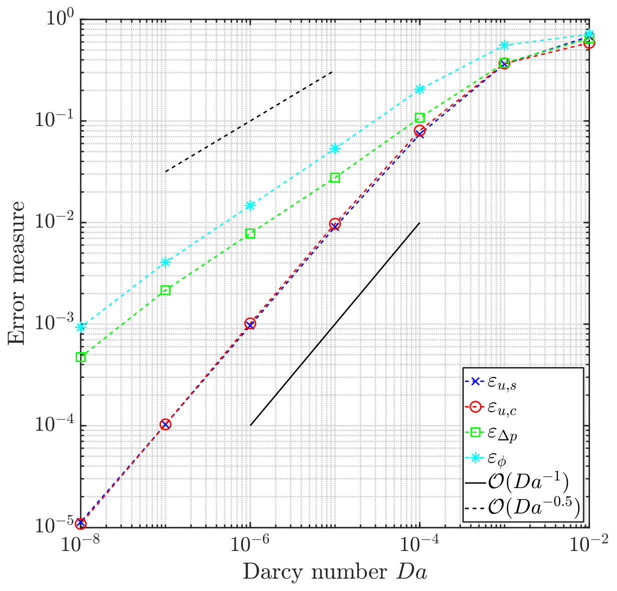


Fig. 7: Error measures as a function of permeability for a Reynolds number of $Re = 20$ and $Da \in \{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}, 10^{-7}, 10^{-8}\}$.

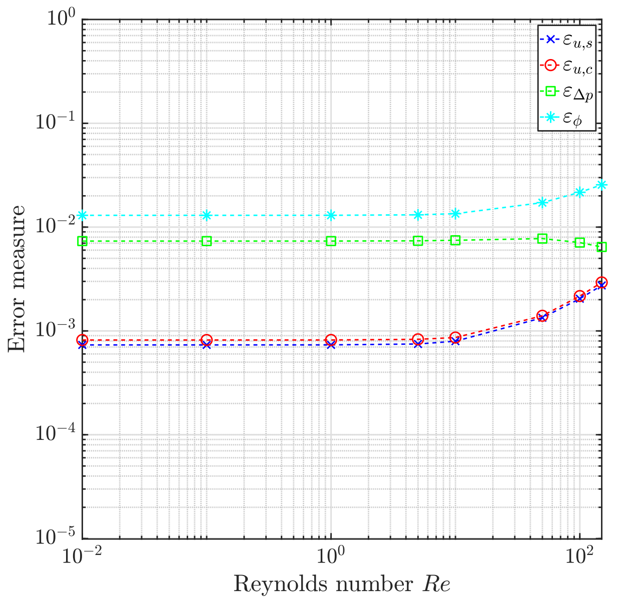


Fig. 8: Error measures as a function of Reynolds number for a Darcy number of $Da = 10^{-6}$ and $Re \in \{0.01, 0.1, 1, 5, 10, 50, 100, 150\}$.

becomes larger than 10, the error begins to slowly grow. This makes sense, since the porous media analogy is based on scaling with the viscous diffusion term and a Brinkman porous media model. As the Reynolds number increases, the convective term begins to dominate in Equation 6 and, thus, it makes sense that the penalty term should scale differently. However, the slight increase in the errors are not seen as significant for the moderate Reynolds numbers treatable with the present steady-state implementation. An alternative scaling law proposed by Kondoh et al. (2012) is discussed in Appendix A.

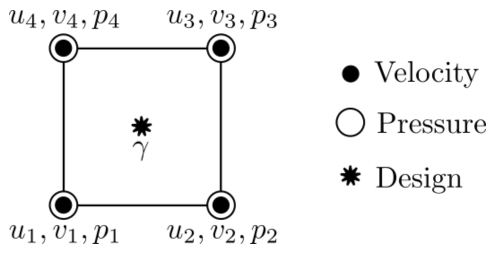


Fig. 9: Element degrees-of-freedom (DOFs) and nodal numbering.

4 Discretisation and solution

4.1 Finite element formulation

The governing equations in Equation 2 are discretised using a stabilised continuous Galerkin finite element formulation. Both the velocity and pressure fields are approximated using piecewise bi-linear interpolation functions and the design field is approximated using piecewise constant values, rendering a *u1p1g0* element as shown in Figure 9. This yields 8 velocity degrees-of-freedom (DOFs), 4 pressure DOFs, and 1 design DOF per element. Due to the equal-order interpolation for velocity and pressure, Pressure-Stabilising Petrov-Galerkin (PSPG) stabilisation is applied to get a stable finite element formulation. Furthermore, to avoid spurious oscillations due to convection, Streamline Upwind Petrov-Galerkin (SUPG) stabilisation is also applied. The detailed formulation is given in Appendix B.

The discretisation gives rise to a non-linear system of equations, which is posed in residual form:

$$\mathbf{r}(\mathbf{s}, \boldsymbol{\gamma}) = \mathbf{A}(\mathbf{s}, \boldsymbol{\gamma})\, \mathbf{s} = \mathbf{0} \qquad (16)$$

where $\mathbf{A}$ is the system coefficient matrix, $\mathbf{s} = \left[\mathbf{u}^T \mathbf{p}^T\right]^T$ is the vector of global state DOFs, $\mathbf{u}$ is the vector of velocity DOFs, $\mathbf{p}$ is the vector of pressure DOFs, and $\boldsymbol{\gamma}$ is the vector of design field values. There are many non-linear and design-dependent terms in the system: the convection term is dependent on the velocity field; the Brinkman term is dependent on the design field; and the stabilisation terms are dependent on both the velocity and design field (either directly through convection and Brinkman terms or indirectly through the stabilisation parameter). Thus, a robust non-linear solver is needed and the Newton solver is detailed in the subsequent section. Furthermore, derivation of accurate partial derivatives, such as the Jacobian matrix and those necessary for optimisation, can be cumbersome. Therefore, these derivatives will be automatically computed using the Symbolic Toolbox in MATLAB.

4.2 Newton solver

To solve the non-linear system of equations, a damped Newton method is used. At each non-linear iteration, k, the following linearised problem is solved for the Newton step, $\boldsymbol{\Delta}\mathbf{s}_k$:

$$\left.\frac{\partial \mathbf{r}}{\partial \mathbf{s}}\right|_{k-1} \boldsymbol{\Delta}\mathbf{s}_k = -\mathbf{r}(\mathbf{s}_{k-1}) \tag{17}$$

where the solution is then updated according to:

$$\mathbf{s}_k = \mathbf{s}_{k-1} + \delta_k \boldsymbol{\Delta}\mathbf{s}_k \tag{18}$$

with $\delta_k \in [0.01; 1]$ being the damping factor[2]. In order to gain robust convergence for moderate Reynolds numbers, the damping factor is updated according to the minimum of a quadratic fit of the residual norm dependence on δ. The residual and its 2-norm is calculated for $\delta \in \{0, 0.5, 1\}$, with the current value ($\delta = 0$) already known. A quadratic equation is then fitted to the residual norms with the unique minimising damping factor expressed analytically. The damping factor is then set to:

$$\delta_k = \max\left(0.01, \min\left(1, \frac{3||\mathbf{r}_0||_2 - 4||\mathbf{r}_{0.5}||_2 + ||\mathbf{r}_1||_2}{4||\mathbf{r}_0||_2 - 8||\mathbf{r}_{0.5}||_2 + 4||\mathbf{r}_1||_2}\right)\right) \tag{19}$$

where $\mathbf{r}_\delta$ is the residual evaluated for the solution updated with a given δ.

The Newton solver is run until the current residual norm has been reduced relative to the initial below a predefined threshold. To speed up convergence, the state solution from the previous design iteration is used as the initial solution for the Newton solver. If the solver does not converge before a maximum iteration number is reached, it tries again from a zero (except for boundary conditions) initial solution. If that also fails, the Reynolds number may be too large to have a steady-state solution for the problem. It is also possible that the problem is so non-linear, that it might need ramping of the inlet velocity or Reynolds number.

[2] The lower bound is chosen to avoid stagnation.

5 Optimisation formulation

5.1 Objective functional

The dissipated energy was originally introduced as the objective functional by Borrvall and Petersson (2003). Equation 12 is the dissipated energy in the fluid domain due to viscous resistance only. The addition of the Brinkman penalty term introduces a body force, which must be taken into account in the dissipated energy:

$$\phi = \frac{1}{2}\int_{\Omega} \left(\mu \frac{\partial u_i}{\partial x_j}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) + \alpha(\mathbf{x}) u_i u_i \right) dV \tag{20}$$

which is now integrated over the entire computational domain.

5.2 Volume constraint

In order to avoid trivial solutions, a maximum allowable volume of fluid is imposed as a constraint. The relative volume of fluid is in continuous form defined as:

$$V = \frac{1}{|\Omega|}\int_{\Omega} \gamma(\mathbf{x})\, dV \tag{21}$$

which represents the volumetric average of the design field. The relative fluid volume should be restricted to be below $V_f \in]0; 1[$, which is the fraction of the total domain allowed to be fluid. After discretisation as described in Section 4, the relative volume is computed by:

$$V = \frac{1}{n_{el}} \sum_{e=1}^{n_{el}} \gamma_e \tag{22}$$

where n_{el} is the number of elements and it has been exploited that all elements have the same volume in the current implementation.

5.3 Optimisation problem

The final discretised optimisation problem is:

$$\begin{aligned} \underset{\boldsymbol{\gamma}}{\text{minimise:}} \quad & f_\phi\left(\mathbf{s}(\boldsymbol{\gamma}), \boldsymbol{\gamma}\right) = \phi \\ \text{subject to:} \quad & V(\boldsymbol{\gamma}) \leq V_f \\ \text{with:} \quad & \mathbf{r}(\mathbf{s}(\boldsymbol{\gamma}), \boldsymbol{\gamma}) = \mathbf{0} \\ & 0 \leq \gamma_i \leq 1,\ i = 1, \dots, n_{el} \end{aligned} \tag{23}$$

5.4 Adjoint sensitivity analysis

The gradients of any functionals dependent on the state field are found using adjoint sensitivity analysis. A general description is given in Appendix C. The sensitivities for a generic functional f are found from:

$$\frac{df}{d\gamma_e} = \frac{\partial f}{\partial \gamma_e} - \boldsymbol{\lambda}^T \frac{\partial \mathbf{r}}{\partial \gamma_e} \tag{24}$$

where $\boldsymbol{\lambda}$ is the vector of adjoint variables for the state residual constraint found from the adjoint problem:

$$\frac{\partial \mathbf{r}}{\partial \mathbf{s}}^T \boldsymbol{\lambda} = \frac{\partial f}{\partial \mathbf{s}}^T \tag{25}$$

The above is valid for any equation system formulated as a residual, $\mathbf{r}$, and any state-dependent functional, f, be it objectives or constraints. For a given system, two partial derivatives of the residual are necessary, and for a given functional, two partial derivatives of the functional are necessary: with respect to the design field and the state field. These partial derivatives can be found analytically by hand, but in the presented implementation they are found automatically using symbolic differentiation in MATLAB.

5.5 Optimality criteria optimiser

An optimality criteria (OC) based optimisation algorithm is used in the presented code. The implemented OC method is the modified version suggested by Bendsøe and Sigmund (2004) for compliant mechanism problems, where the design gradients can be both positive and negative:

$$\gamma_e^{new} = \begin{cases} \gamma_e^L & \text{if } \gamma_e {B_e}^{\eta} \leq \gamma_e^L \\ \gamma_e^U & \text{if } \gamma_e {B_e}^{\eta} \geq \gamma_e^U \\ \gamma_e {B_e}^{\eta} & \text{otherwise} \end{cases} \tag{26}$$

where $\eta = \frac{1}{3}$ is a damping factor and the upper bounds at each iteration are given as:

$$\gamma_e^L = \max(0, \gamma_e - m) \tag{27a}$$

$$\gamma_e^U = \min(1, \gamma_e + m) \tag{27b}$$

with m being the movelimit. The optimality criteria ratio is given as:

$$B_e = \max\left(\varepsilon, \frac{-\frac{\partial \phi}{\partial \gamma_e}}{l \frac{\partial \mathcal{V}}{\partial \gamma_e}}\right) \tag{28}$$

where ε is a small positive number (e.g. 10^{-10}) and l is the Lagrange multiplier for the volume constraint. This means that positive gradients (less fluid is better) are ignored and the design process is driven only by the negative gradients (more fluid is better). However, this has worked extremely well for the problems at hand, because mostly the dissipated energy gradients are negative.

The Lagrange multiplier is found using bisection. To speed up the process, the upper bound estimate suggested by Ferrari and Sigmund (2020) is modified for $\eta = \frac{1}{3}$:

$$\bar{l} = \left(\frac{1}{n_{el} V_f} \sum_{e=1}^{n_e} \gamma_e \left(-\frac{\frac{\partial \phi}{\partial \gamma_e}}{\frac{\partial V}{\partial \gamma_e}} \right)^{\frac{1}{3}} \right)^3 \tag{29}$$

5.6 Continuation scheme

As shown by Borrvall and Petersson (2003), the minimisation of dissipated energy is self-penalising for the Brinkman approach with linear interpolation of $\alpha(\gamma)$. This means that intermediate design values are not beneficial and the optimal solutions are discrete 0-1. However, it is often beneficial to relax the problem using the interpolation factor, q_α, in the initial stages of the optimisation process in order to avoid convergence to particularly poor local minima. Therefore, it is common to solve the minimisation problem for a sequence of q_α, known as a continuation scheme.

In the presented code, a heuristic approach is implemented to automatically choose the interpolation factor. The sequence for the continuation scheme is given by:

$$q_\alpha \in \left\{ q_\alpha^0, \frac{q_\alpha^0}{2}, \frac{q_\alpha^0}{10}, \frac{q_\alpha^0}{20} \right\} \tag{30}$$

where:

$$q_\alpha^0 = \frac{(\alpha_{\max} - \alpha_0) - x_0 (\alpha_{\max} - \alpha_{\min})}{x_0 (\alpha_0 - \alpha_{\min})} \tag{31}$$

is the initial interpolation factor necessary to ensure an initial Brinkman penalty of α_0 for an initial design field value of x_0. The interpolation factor is changed when the subproblem is considered converged or after a predefined number of design iterations. For the problems implemented in the presented code, an initial Brinkman penalty of $\alpha_0 = \frac{2.5\mu}{0.1^2}$ works very well. However, do bear in mind that this is a heuristic value and will not work for all problems and all settings.

Due to the non-convexity, the initial Brinkman penalty of the domain plays an important role in which direction the design will progress. The initial flow field determines the exploration of the design space, because if the flow is not moving through some parts of the design domain, sensitivities will often be near zero in those areas. Therefore, it is often beneficial to start with an

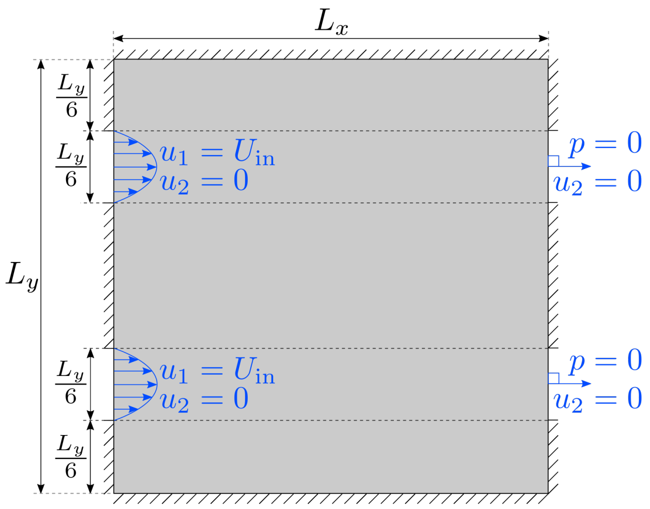


Fig. 10: Problem setup for the double pipe problem.

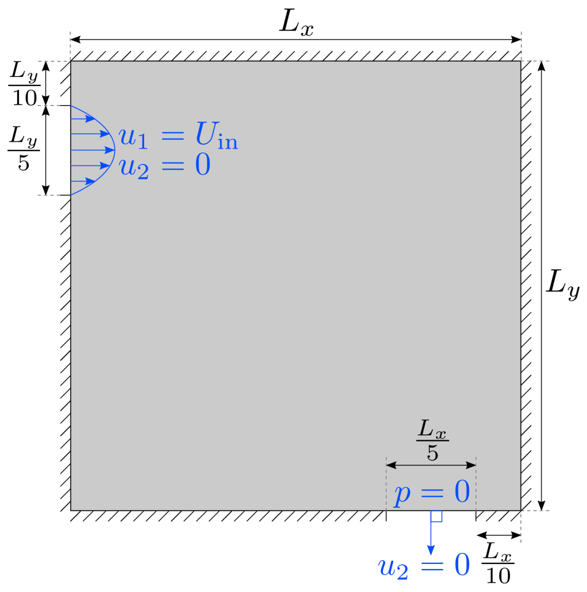


Fig. 11: Problem setup for the pipe bend problem.

initial Brinkman penalty that allows for the fluid to move into most areas of the domain. This was for instance recently observed in the topology optimisation of heat exchanger designs by Høghøj et al. (2020).

6 Benchmark examples

6.1 Double pipe problem

This problem was first introduced by Borrvall and Petersson (2003) and is illustrated in Figure 10. On the left-hand side, there are two inlets at which parabolic normal velocity profiles are prescribed with a maximum velocity of U_{in}. On the right-hand side, there are two zero-pressure outlets at which the flow is specified to exit in the normal direction. This is a more realistic boundary condition traditionally used in the computational fluid dynamics community, compared to prescribed parabolic flow profiles at the outlets (Borrvall and Petersson, 2003). However, this does introduce new and better minima as was shown by Papadopoulos et al. (2021), which makes the problem more difficult.

6.2 Pipe bend problem

This problem was also introduced by Borrvall and Petersson (2003) and is illustrated in Figure 11. On the left-hand side, there is an inlet at which a parabolic normal velocity profile is prescribed with a maximum velocity of U_{in}. On the bottom, there is a zero-pressure outlet at which the flow is specified to exit in the normal direction - a minor change from Borrvall and Petersson (2003) as for the double pipe problem.

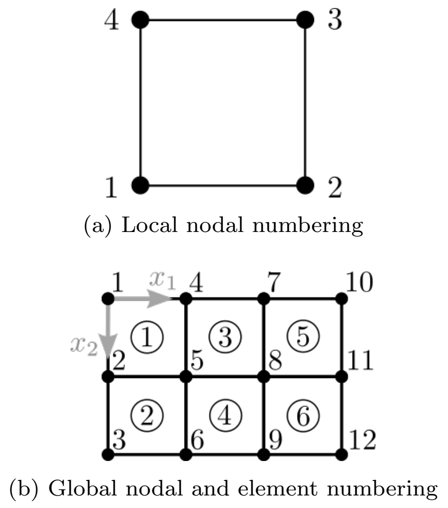

4
3
1
2
(a) Local nodal numbering
1
x_1
4
7
10
x_2
2
5
8
11
3
6
9
12
①
③
⑤
②
④
⑥
(b) Global nodal and element numbering

Fig. 12: Local and global numbering. Numbers without circles indicate nodal numbering and number with circles indicate element numbering.

7 MATLAB implementation

7.1 Mesh and numbering

Figure 12 shows the local and global numbering used in the presented implementation. As shown in Figure 12b, for the global numbering of both the nodes and elements begins in the upper-left corner, increasing from top to bottom and then from left to right. The origin of the global coordinate system is placed in the upper-left corner, with the first spatial direction, x_1, being positive from left to right and the second spatial direction, x_2, being positive from top to bottom. This orientation has been chosen to simplify easy plotting of the various spatial fields in MATLAB. The velocity and pressure DOFs follow the ordering of the nodes and the design field follows the ordering of the elements.

7.2 Brief description of code

The full code consists of multiple files, the main ones of which are available in Appendices D-H. The full code base is available on GitHub (Alexandersen, 2022).

The main file `topFlow.m` (Appendix D) can be divided into three parts: pre-processing, optimisation loop, and post-processing. Each part will be very briefly described herein, except the definition of problem parameters which is treated in detail below since this will be of importance to the subsequent definition of examples and results. A detailed description of the code is available in the Supplementary Material.

7.2.1 Definition of input parameter (lines 6-29)

The pre-processing part starts with the setting of input parameters (lines 6-29) to define the problem and optimisation parameters. In the presented code, two different problems are already implemented and the `probtype` variable determines which one is to be optimised:

```
 7 % PROBLEM TO SOLVE (1 = DOUBLE PIPE; 2 = ...
       PIPE BEND)
 8 probtype = 1;
```

Next the dimensions of the domain and the discretisation are given:

```
 9 % DOMAIN SIZE
10 Lx = 1.0; Ly = 1.0;
11 % DOMAIN DISCRETISATION
12 nely = 30; nelx = nely*Lx/Ly;
```

where `Lx` and `Ly` are the dimensions in the x- and y-directions, respectively. The number of elements in the x-direction, `nelx`, is automatically determined from the supplied number of elements in the y-direction, `nely`, to ensure square elements. The implementation can handle rectangular elements, so `nelx` can be decoupled from `nely` if necessary.

The allowable volume fraction V_f is stored in the `volfrac` variable and the initial design field value is set to be the same to fulfill the volume constraint from the start:

```
13 % ALLOWABLE FLUID VOLUME FRACTION
14 volfrac = 1/3; xinit = volfrac;
```

The inlet velocity U_{in} is stored in the variable `Uin` and the fluid density, ρ and dynamic viscosity, μ, are stored in the variables `rho` and `mu`, respectively:

```
% PHYSICAL PARAMETERS                                 15
Uin = 1e0; rho = 1e0; mu = 1e0;                       16
```

The maximum and minimum Brinkman penalty factors are per default defined with respect to an out-of-plane thickness according to Equation 9 (Borrvall and Petersson, 2003) and the heuristic continuation scheme is automatically computed and stored in `qavec`:

```
% BRINKMAN PENALISATION                               17
alphamax = 2.5*mu/(0.01^2); alphamin = ...            18
    2.5*mu/(100^2);
% CONTINUATION STRATEGY                               19
ainit = 2.5*mu/(0.1^2);                               20
qinit = (-xinit*(alphamax-alphamin) - ...             21
    ainit + ...
    alphamax)/(xinit*(ainit-alphamin));
qavec = qinit./[1 2 10 20]; qanum = ...               22
    length(qavec); conit = 50;
```

where `ainit` is the initial Brinkman penalty, `qinit` is the initial interpolation factor computed using Equation 31, `qanum` is the number of continuation steps, and `conit` is the maximum number of iterations per continuation step.

Various optimisation parameters are then defined:

```
% OPTIMISATION PARAMETERS                             23
maxiter = qanum*conit; mvlim = 0.2; ...               24
    plotdes = 0;
chlim = 1e-3; chnum = 5;                              25
```

where `maxiter` is the maximum number of total design iterations, `mvlim` is the movelimit m for the OC update, `chlim` is the stopping criteria, `chnum` is the number of subsequent iterations the convergence criteria must be met, and `plotdes` is a Boolean determining whether to plot the design during the optimisation process ($0 =$ no, $1 =$ yes).

The parameters for the Newton solver are defined:

```
% NEWTON SOLVER PARAMETERS                            26
nltol = 1e-6; nlmax = 25; plotres = 0;                27
```

where `nltol` is the required residual tolerance for convergence, `nlmax` is the maximum number of non-linear iterations, and `plotres` is a Boolean determining whether to plot the convergence of the residual norm over the optimisation process ($0 =$ no, $1 =$ yes).

Finally, the option to export the contour of the final design as a DXF (Data Exchange Format) file can be activated:

```
% EXPORT FILE                                         28
filename='output'; exportdxf = 0;                     29
```

where `filename` is a specified filename and `exportdxf` is a Boolean determining whether to export the design ($0 =$ no, $1 =$ yes).

7.2.2 Other pre-processing (lines 30-66)

Subsequently, various arrays are defined (lines 30-44) for the quick assembly of the matrices and vectors from the finite element analysis based on the `sparse` function in MATLAB. The boundary conditions for the available problems are defined in the separate MATLAB file `problems.m` (Appendix E) and matrices for quick enforcement of Dirichlet conditions are built (lines 45-51). Finally, various arrays, counters and constants are initialised (lines 52-66).

7.2.3 Optimisation loop (lines 67-167)

Prior to the optimisation loop start, some information is output to the screen and timers are initiated (lines 67-73). The optimisation starts using a `while` loop (line 74) and computation of the greyscale indicator (Sigmund, 2007) and material interpolation per Equation 8 (lines 76-80). The non-linear solver loop iteratively updates the solution (lines 81-112) using Newton's method according to Section 4.2. The finite element analysis is carried out according to Section 4.1 (lines 85-93) using function `RES` and `JAC` to build to residual vector and Jacobian matrix, respectively. For the converged solution, the objective functional is evaluated (line 114) using the function `PHI` according to Section 5.1 and the volume is evaluated for the current design (line 117) according to Section 5.2. The current solution is evaluated (lines 124-126) and if it is not considered converged, then the adjoint problem is formed and solved (lines 131-136) and the sensitivities are computed (lines 137-144) according to Section 5.4. Subsequently, the design field is updated using the optimality criteria optimiser (lines 145-154) according to Section 5.5. Before repeating the process for the new design, the interpolation parameter is updated if necessary according to the continuation scheme (lines 155-159) in Section 5.6.

The functions for evaluating the residual vector, Jacobian matrix, objective functional, and their partial derivatives, are all automatically built using the file `analyticalElement.m`. This script uses the MATLAB Symbolic Toolbox to symbolically derive and differentiate the required vectors and matrices.

7.2.4 Post-processing (lines 168-170)

The results are post-processed and plotted in an external file `postproc.m` (line 169), where 5 standard plots are made:

1. Design field $\gamma(\mathbf{x})$
2. Brinkman penalty field, $\alpha(\gamma(\mathbf{x}))$
3. Velocity magnitude field, $U = \sqrt{u_i u_i}$
4. Pressure field, p
5. Velocity and design field along a cut-line

Finally, if requested by setting `exportdxf = 1`, the contour of the final optimised design is exported to DXF format for verification analysis in external software.

Parameter	Value	Code
L_x	1	`Lx = 1.0`
L_y	1	`Ly = 1.0`
ρ	10^{-3}	`rho = 1e-3`
μ	1	`mu = 1.0`
$\alpha_{\max}$	$2.5\mu/(0.01^2)$	`alphamax = 2.5*mu/(0.01^2)`
$\alpha_{\min}$	$2.5\mu/(100^2)$	`alphamin = 2.5*mu/(100^2)`
V_f	$1/3$	`volfrac = 1/3`
n_x^e	102	`nelx = 102`
n_y^e	102	`nely = 102`

Table 1: Parameter values used for double pipe solutions in Figures 13 and 14.

8 Examples

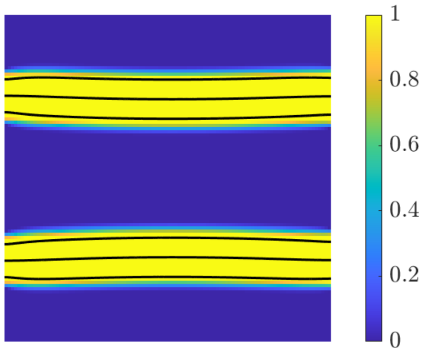
8.1 Double pipe problem

This problem was introduced in Section 6.1 and is the default for the provided code when `probtype = 1`. It was originally introduced by Borrvall and Petersson (2003) for Stokes flow, but will herein also be optimised for Navier-Stokes flow.

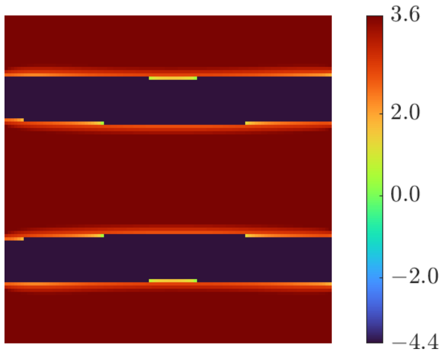
8.1.1 Stokes flow

In order to make the results directly comparable to those of Borrvall and Petersson (2003), the parameter values are set as in Table 1. To simulate Stokes flow ideally the density would be set to 0, but due to the current implementation not being able to handle that, the density is set to a small number, $\rho = 10^{-3}$ instead. This yields an inlet Reynolds number of $Re_{\text{in}} = 1.67 \times 10^{-4}$, which is sufficiently small to ensure Stokes flow.

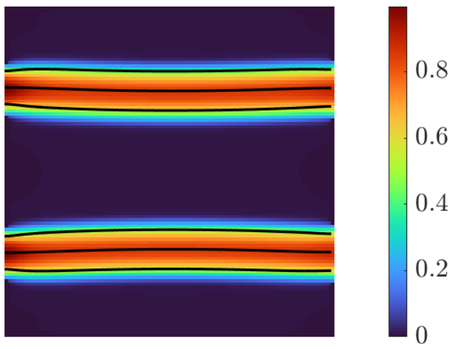
Figure 13 shows the optimised design and corresponding physical fields for a square domain with $Re_{\text{in}} = 1$. From Figure 13b it can be seen that the converged design in Figure 13a ensures the lowest and highest Brinkman penalty factor in the fluid region and solid region, respectively. This results in negligible velocities in the solid region with the flow passing easily through the channels, as seen in Figure 13c. Figure 13d shows the pressure field, where it can be seen that a continuous pressure field exists even in the solid region. The final objective value is $\phi = 22.0956$, which is very close to the value of 25.67 reported by Borrvall and Petersson (2003). Differences can be attributed to different implementations, most likely the fact that the



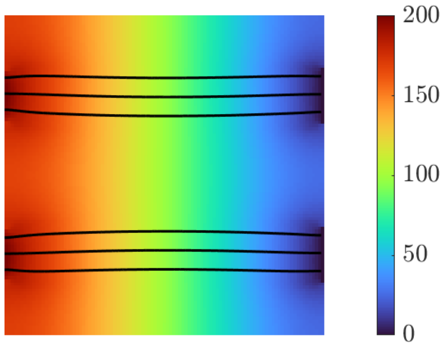
(a) Design field



(b) Brinkman penalty field



(c) Velocity magnitude field



(d) Pressure field

Fig. 13: Optimised design and corresponding physical fields for the double pipe problem with $L_x = 1$ and $Re_{\text{in}} = 10^{-3}$. Final values: $\phi = 22.0956$ after 63 iterations.

Parameter	Value	Code
ρ	1	`rho = 1.0`
μ	$1/6Re_{\text{in}}$	`mu = 1/(6*20)`

Table 2: Updated values used for double pipe solutions in Figure 15. Other values are given in Table 1.

present uses stabilised linear finite elements in contrast to mixed-order quadratic-linear elements (Borrvall and Petersson, 2003). The level of intermediate design variables since the final interpolation factors are almost the same, at 9.85 for the present code and an equivalent of 10 in Borrvall and Petersson (2003), and the level of greyscale is therefore similar.

An elongated domain is investigated by setting $L_x = 1.5$ similar to Borrvall and Petersson (2003), with the number of elements increased proportionally in the same direction. Figure 14 shows the optimised design and corresponding physical fields. The optimised design is now a "double wrench" design, where the channels merge at the centre to minimise viscous losses along the channel walls. The final objective value is $\phi = 23.5732$, which is very close to the value of 27.64 reported by Borrvall and Petersson (2003).

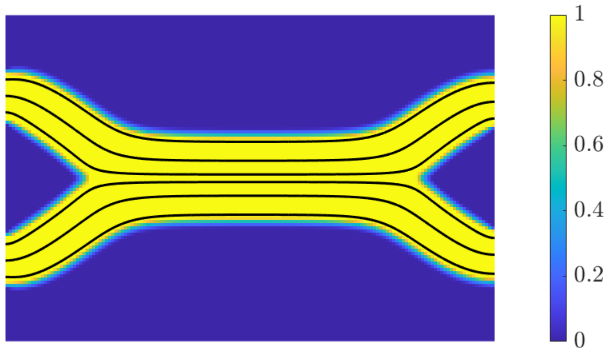
8.1.2 Navier-Stokes

In order to non-dimensionalise the problem and control it using the Reynolds number, Re_{in}, the density is set to $\rho = 1$ (`rho = 1.0`) and the viscosity is set to $\mu = 1/6Re_{\text{in}}$ (`mu = 1/(6*20)`). The 6 in the denominator ensures that the Reynolds number, Re_{in}, is based on the inlet height which is $\frac{L_y}{6}$ as shown in Figure 10. Other parameter values are as previously.

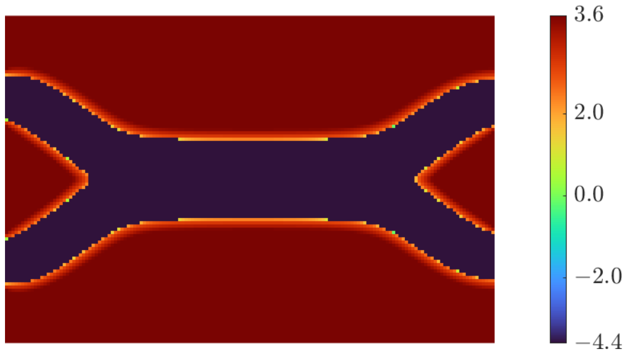
Figure 15 shows the optimised designs obtained for increasing Reynolds number. It is observed that for $Re_{\text{in}} = 40$ and above, the OC solver converges to the local minimum of two "straight" channels. It is observed using COMSOL simulations, that this local minimum is worse than the double wrench channel for Reynolds numbers up to 200. The MMA solver introduced in Section 9.2.1 is actually able to converge to a significantly better design for $Re_{\text{in}} = 40$.

8.1.3 Variation of α_{max}

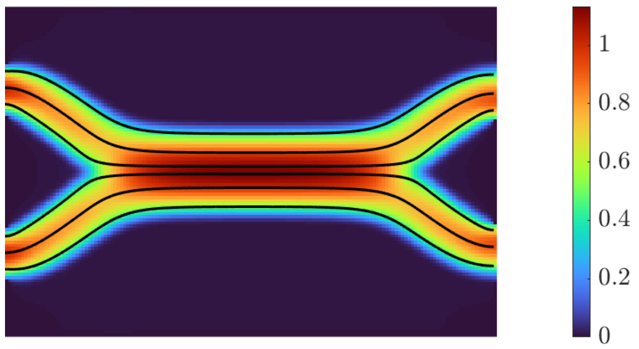
To demonstrate the robustness of the proposed heuristic initial Brinkman penalty and continuation scheme, as well as the insensitivity of minimum dissipation optimisation to the maximum Brinkman penalty, the double pipe problem will be optimised for $Re_{\text{in}} = 20$ using 5 additional $\alpha_{\max}$. Getting to the double wrench channel for $Re_{\text{in}} = 20$ seems pretty difficult and the poorer



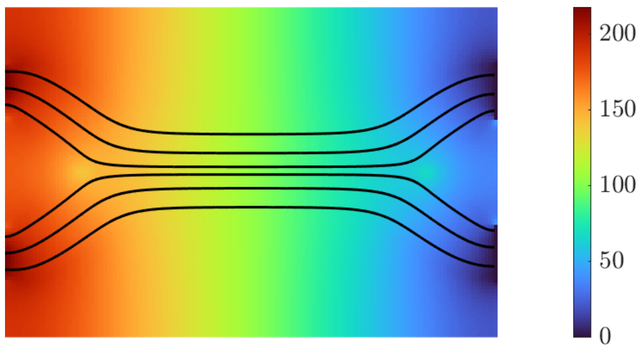
(a) Design field



(b) Brinkman penalty field

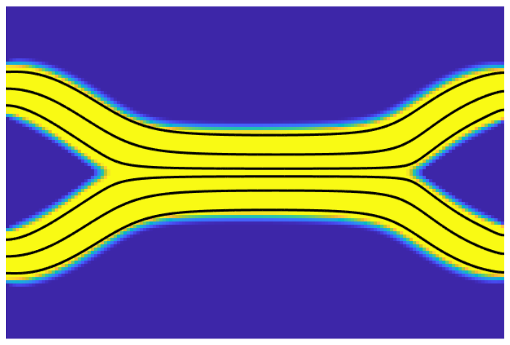


(c) Velocity magnitude field

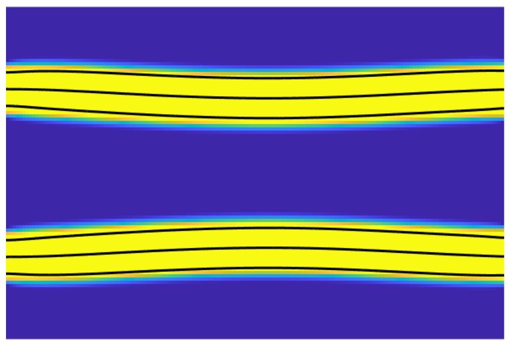


(d) Pressure field

Fig. 14: Optimised design and corresponding physical fields for the double pipe problem with $L_x = 1.5$ and $Re_{\text{in}} = 10^{-3}$. Final values: $\phi = 23.5732$ after 87 iterations.



(a) $Re_{\text{in}} = 20$



(b) $Re_{\text{in}} = 40$

Fig. 15: Optimised designs for the double pipe problem with increasing Reynolds number and $L_x = 1.5$. Final values: (a) $\phi = 0.2126$ after 102 iterations; (b) $\phi = 0.1380$ after 59 iterations.

local minima double pipe design is obtained very easily. Having a constant fixed initial Brinkman penalty, α_{init}, of the entire design domain ensures that the same initial flow field is obtained independent of maximum Brinkman penalty, α_{max}.

The values predicted by the presented MATLAB code are compared to verification analyses using COMSOL of post-processed designs. The `export.m` file provides export capabilities of the design field contour to the DXF file format. The DXF file can then be imported into COMSOL and used to trim the computational domain. An example COMSOL file is included in the Supplementary Material.

Figure 16 shows the dissipated energy of the final optimised designs. The final designs look basically identical to Figure 15a and are, thus, not shown here. It can be seen that the post-processed performance of the designs are more or less constant independent of α_{max}. The values predicted by the continuous Brinkman model are seen to be quite far off for lower maximum penalties. However, this shows that even a low maximum Brinkman penalty, which in theory yields a large error in the prediction of the dissipated energy (Figure

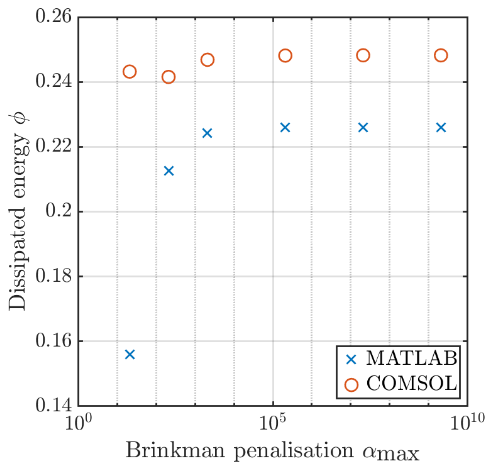


Fig. 16: $Re_{\text{in}} = 20$

Parameter	Value	Code
L_x	1	`Lx = 1.0`
L_y	1	`Ly = 1.0`
ρ	10^{-3}	`rho = 1e-3`
μ	1	`mu = 1.0`
$\alpha_{\max}$	$^{2.5\mu}/_{(1/0.01^2)}$	`alphamax = 2.5*mu/(1/0.01^2)`
$\alpha_{\min}$	$^{2.5\mu}/_{(100^2)}$	`alphamin = 2.5*mu/(100^2)`
V_f	0.25	`volfrac = 0.25`
n_x^e	100	`nelx = 100`
n_y^e	100	`nely = 100`

Table 3: Parameter values used for pipe bend solution in Figure 17.

7), ends up giving the same or close to the same final design and performance. This insensitivity was already noted originally by Borrvall and Petersson (2003), but not explicitly shown.

This conclusion is somewhat disconnected from the investigations of the flow accuracy in Section 3.3 and Figure 5. However, in contrast to the reference geometry in Section 3.3, the final optimised design for minimum dissipated energy do not have re-circulation zones, since this is detrimental to the energy functional. Therefore, for the optimised designs, a lower maximum Brinkman penalty appears to be forgiving.

8.2 Pipe bend problem

This problem was introduced in Section 6.2 and is selected in the code by setting `probtype = 2`. It was originally introduced by Borrvall and Petersson (2003) for Stokes flow and also treated by Gersborg-Hansen et al. (2005) for Navier-Stokes flow.

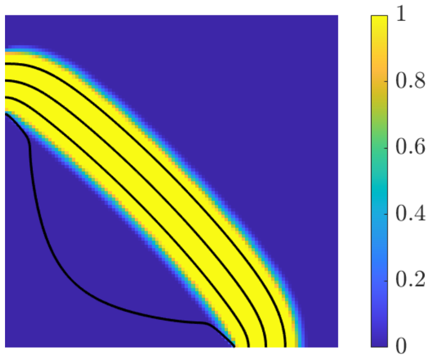


Fig. 17: Optimised design for the pipe bend problem using values from Borrvall and Petersson (2003). Final values: $\phi = 9.1862$ after 66 iterations.

Parameter	Value	Code
L_x	1	`Lx = 1.0`
L_y	1	`Ly = 1.0`
ρ	10^{-3}	`rho = 1.0`
μ	$^1/_{(5\,Re_{\text{in}})}$	`mu = 1/(5*10)`
$\alpha_{\min}$	$^{2.5\mu}/_{(1/5^2)}$	`alphamin = 2.5*mu/(1/5^2)`
$\alpha_{\max}$	$10^4\alpha_{\min}$	`alphamax = 1e4*alphamin`
V_f	0.25	`volfrac = 0.25`
n_x^e	100	`nelx = 100`
n_y^e	100	`nely = 100`

Table 4: Parameter values used for pipe bend solution in Figure 18.

8.2.1 Stokes flow

In order to make the results comparable to those of Borrvall and Petersson (2003), the parameter values are set as in Table 3. As previously, the density is set to a small number, $\rho = 10^{-3}$, yielding an inlet Reynolds number of $Re_{\text{in}} = 2 \times 10^{-4}$ which is small enough to approximate Stokes flow.

Figure 17 shows the optimised design obtained using the values from Borrvall and Petersson (2003) listed in Table 3. The design is a straight channel from the inlet to the outlet with a final objective value of $\phi = 22.0956$, which is very close to the value of 25.67 reported by Borrvall and Petersson (2003). The streamlines show that the maximum Brinkman penalty is not large enough to provide enough resistance to flow through the solid regions. However, as previously shown, while increasing the $\alpha_{\max}$ may increase the accuracy of the flow modelling, almost identical optimised designs are obtained for minimum dissipated energy.

8.2.2 Navier-Stokes flow

In order to investigate the effect of inertia, the out-of-plane channel height is set to the width of the inlet. This ensures a hydraulic diameter of the inlet width/height

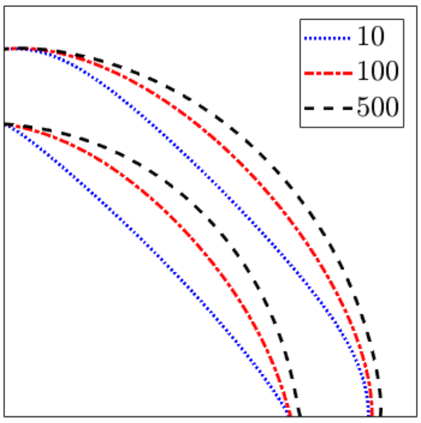


Fig. 18: Design contours for the pipe bend problem with increasing Reynolds number. Final values: (a) $\phi = 2.484 \times 10^{-1}$ after 58 iterations; (b) $\phi = 2.830 \times 10^{-2}$ after 63 iterations; (c) $\phi = 6.502 \times 10^{-3}$ after 40 iterations.

	Optimised for:		
Analysed at:	$Re = 10$	$Re = 100$	$Re = 500$
$Re = 10$ $[\times 10^{-1}]$	**2.484**	2.731	3.071
$Re = 100$ $[\times 10^{-2}]$	2.889	**2.830**	3.105
$Re = 500$ $[\times 10^{-3}]$	8.459	6.553	**6.502**

Table 5: Cross-check table of the optimised designs for the pipe bend problem shown in Figure 18. Values should be multiplied by. The best performing design for each analysis Reynolds number is highlighted in bold.

and a well-defined Reynolds number. The problem is non-dimensionalised and controlled using the inlet Reynolds number, Re_{in}, leading to the parameter values listed in Table 4. The 5 in the denominator of the viscosity ensures that the Reynolds number is based on the inlet width, which is $\frac{L_y}{5}$ as shown in Figure 11.

Figure 18 shows the contours of the optimised designs for increasing Reynolds number. For a low Reynolds number, $Re_{\text{in}} = 10$, a more or less straight pipe is formed similar to that for Stokes flow. When the Reynolds number increases, inertia begins to play a role and the curvature of the pipe changes to accommodate this. This is confirmed by the cross-check of the three designs with the objective values in Table 5. A cross-check is strictly necessary prior to drawing conclusions from designs optimised for different conditions. A cross-check is where the optimised designs are tested for the other flow conditions and compared. A successful cross-check is where the design optimised for certain conditions is also the best for these conditions. This means it has the lowest value in its own row, such that the diagonal has the best values as highlighted in bold in Table 5.

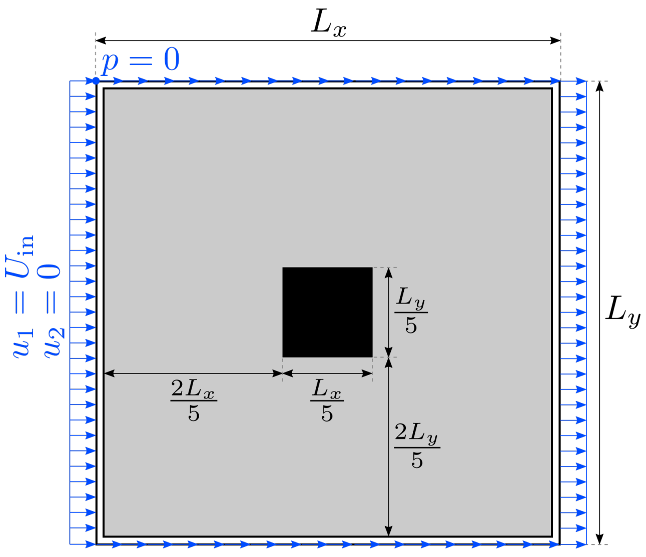


Fig. 19: Problem setup for the modified rugby ball problem.

9 Modifications

This Section presents a series of extensions to the provided code, showing how it is possible to extend it to solve a range of more complicated examples including fixed regions and other objective functionals.

9.1 Fixed regions of solid and fluid

It is common that the design domain is only a subset of the computational domain, where the rest is defined as fixed solid or fluid domains. In the discretised problem this can be handled by restricting optimisation to the *active* elements in the defined design domain, with *passive* elements in the fixed domains remaining constant at their prescribed value.

The implementation of passive and active elements will be illustrated using the so-called rugby ball problem, introduced by Borrvall and Petersson (2003) in the context of topology optimisation, but it has its roots in the work by Pironneau (1973). Figure 19 shows the external flow problem with a free flow velocity of U_{in} applied around the entire outer boundary and a single zero-pressure point constraint in the upper-left corner. A thin region next to the boundary all the way around the outside will be prescribed as fluid regions and a solid square will be prescribed in the centre of the domain. The solid square is not imposed originally, but is done so here to demonstrate the definition of both types of passive domains. Table 6 lists the parameter values used herein.

Parameter	Value	Code
L_x	1	`Lx = 1.0`
L_y	1	`Ly = 1.0`
ρ	10^{-3}	`rho = 1e-3`
μ	1	`mu = 1.0`
$\alpha_{\max}$	$^{2.5\mu}/_{(0.01^2)}$	`alphamax = 2.5*mu/(0.01^2)`
$\alpha_{\min}$	$^{2.5\mu}/_{(100^2)}$	`alphamin = 2.5*mu/(100^2)`
V_f	0.94	`volfrac = 0.94`
n_x^e	100	`nelx = 100`
n_y^e	100	`nely = 100`

Table 6: Parameter values used for rugby ball solutions in Figure 20.

First of all, the new problem should be added to `problems.m`. Add the following after line 56 to create a third problem:

```
elseif (probtype == 3) % RUGBY BALL PROBLEM
if ( mod(nelx,5) > 0 || mod(nely,5) > 0 )
error('ERROR: Number of elements per side ...
    must be divisable by 5.');
end
```

The element number check is needed, since the solid square will be defined in terms of fifths of the domain height and width. Then define the boundary conditions and Reynolds number as follows:

```
nodesOuter  = [1:nody-1 nody:nody:nodtot ...
    nody+1:nody:nodtot ...
    (nodx-1)*nody+1:nodtot-1];
nodesPressure = 1;
fixedDofsOuterX = 2*nodesOuter-1; ...
    fixedDofsOuterY = 2*nodesOuter;
fixedDofsU = [fixedDofsOuterX ...
    fixedDofsOuterY];
fixedDofsP = 2*nodtot+nodesPressure;
fixedDofs  = [fixedDofsU fixedDofsP];
% DIRICHLET VECTORS
DIRU=zeros(nodtot*2,1); DIRP=zeros(nodtot,1);
DIRU(fixedDofsOuterX) = Uin; DIR = [DIRU; ...
    DIRP];
% INLET REYNOLDS NUMBER
Renum = Uin*Ly*rho/mu;
```

The fluid region will be defined as a single element wide layer around the outer edge:

```
% PASSIVE ELEMENTS
fluid = [1:nely nely:nely:neltot ...
    (nely+1):nely:neltot ...
    (nelx-1)*nely+1:neltot-1];
```

The solid square is defined as the central one fifth of the domain height and width:

```
x = 2*nelx/5:3*nelx/5; y = 2*nely/5:3*nely/5;
[X,Y] = meshgrid(x,y);
solid = sub2ind([nely nelx],Y(:),X(:))';
```

Finally, all elements of the computational domain will be classified as either *passive* or *active*:

```
passive = [solid fluid]';
active = setdiff(1:neltot,passive)';
nactive = length(active);
```

where `nactive` is the number of active elements to be used in the main script. In order to make the updated code backwards compatible with the first two problems defined in `problem.m`, the following must be added at the bottom of each definition:

```
% PASSIVE ELEMENTS
solid = []; fluid = [];
passive = [solid fluid]'; ...
active = setdiff(1:neltot,passive)';
nactive = length(active);
```

Now, modifications needs to be made to the main script `topFlow.m` in order to ensure that passive elements are given their prescribed value and that the optimisation is only applied to the active elements. After line 57, where the initial design field is defined, add the corrections for the prescribed values:

```
xPhys(solid) = 0.0; xPhys(fluid) = 1.0;
```

Next, the greyscale indicator should be calculated for the design domain only, so change line 77 to:

```
Md = 100*full(4*sum(xPhys(active).* ...
    (1-xPhys(active)))/nactive);
```

The volume constraint should also be modified to only account for the design domain. The evaluation on line 117 should be updated to:

```
V = mean(xPhys(active));
```

and the sensitivities updated accordingly on line 144:

```
dV = ones(nely,nelx)/nactive; dV(passive) ...
    = 0.0;
```

The sensitivities of the objective for the passive elements should also be set to 0, so add the following after the sensitivity calculation on line 142:

```
sens(passive) = 0.0;
```

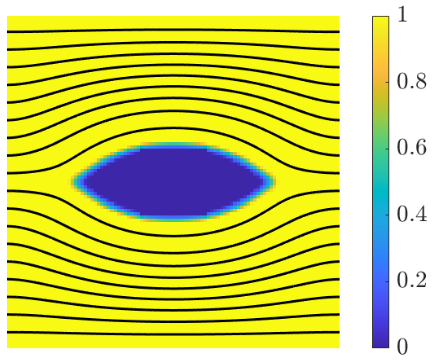
In order to only update the active elements, the pre- and post-steps of the OC solver should be updated. Lines 146-147 should be updated to:

```
xnew = xPhys(active); ...
xlow = xPhys(active)-mvlim; ...
xupp = xPhys(active)+mvlim;
ocfac = xPhys(active).*max(1e-10, ...
    (-sens(active)./dV(active))).^(1/3);
```

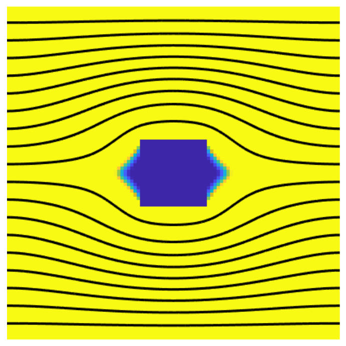
and line 154 should be updated to:

```
xPhys(active) = xnew;
```

In order to solve the rugby ball problem, it is not necessary to use a continuation strategy for the interpolation parameter. As noted by Borrvall and Petersson (2003), the problem is very well-conditioned and it is plenty to use a single parameter value from the start.



(a) $V_f = 0.94$



(b) $V_f = 0.99$

Fig. 20: Optimised designs for the modified rugby ball problem with different allowable fluid volume fractions. Final values: (a) $\phi = 13.503$ after 19 iterations; (b) $\phi = 9.839$ after 17 iterations.

Therefore, the first definition of line 22 should be updated to `qavec = 10` and lines 20 and 21 can be deleted. Beware that changing $\alpha_{\max}$ will now change the initial Brinkman penalty.

Figure 20 shows the optimised design for `volfrac = 0.94` and `volfrac = 0.99`. In order to compare directly with the results of Borrvall and Petersson (2003), the volume fraction has to account for the enforced solid square, which means 0.04 has been added. Figure 20a is visually identical to the result obtained by Borrvall and Petersson (2003), with a final dissipated energy of $\phi = 13.503$ compared to 14.44. Figure 20b shows the optimised design for a higher fluid volume fraction, where the enforced solid square is clearly seen. In contrast to before, the design is now completely different since the solid square dominates the design and triangles have been added in the up- and downwind directions to minimise dissipation. This also causes a significantly higher dissipated energy of $\phi = 9.839$ compared to the freely-optimised design by Borrvall and Petersson (2003) with 8.35.

9.2 Other objective functionals

In order to treat other objective functionals than the dissipated energy, a general optimisation solver is needed. Herein the popular Method of Moving Asymptotes (MMA) will be implemented.

9.2.1 Method of Moving Asymptotes (MMA)

The Method of Moving Asymptotes (MMA) by Svanberg (1987) is a very popular optimisation solver within the topolog optimisation community because it handles general optimisation problems with many design variables extremely well. The MATLAB version of MMA is freely available for download under the GNU GPLv3 license SMOPTIT (2022) and the files `mmasub.m` and `subsolv.m` are needed.

In order to use MMA, there are a series of variables and parameters that must be defined under the initialisation block. After line 66, add the following:

```
% Initialise MMA parameters
numconstr = 1;
xold1 = xPhys(:); xold2 = xPhys(:); xnew = ...
    xPhys(:);
low = 0; upp = 0;
a0 = 1; ai = 0*ones(numconstr,1);
c = 1000*ones(numconstr,1); ...
d = ones(numconstr,1);
```

where `numconstr` is the number of constraints and `a0`, `ai`, `c` and `d` all are parameters for setting the type of problem to be solved by MMA. Please see the document by Svanberg (2007) for details on these parameters. In this document, it is also discussed that MMA works best when the objective and constraint functionals are scaled properly. A block computing the scaling should be added after line 117:

```
%% MMA SCALING
if (loop == 0); obj0 = obj/10; end
f0 = obj/obj0; fi = V/volfrac - 1;
```

This scales the objective to start at 10 for the first iteration and the volume constraint to be at most 1 in magnitude. The constraint is scaled to be at most around one, by normalising the volume:

$$\texttt{fi} = \frac{V}{V_f} - 1 \tag{32}$$

which should be less than 0. The sensitivities should of course be scaled accordingly, which will come later. In order to monitor the scaled values during the optimisation, add the following output in the result printing block of the code after line 128:

```
fprintf('      MMA: f0 = %4.3e - f1 = ...
    %4.3e \n',f0,fi);
```

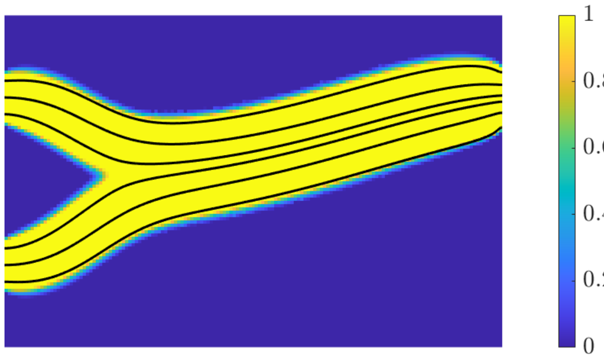


Fig. 21: Optimised design for the double pipe problem using MMA and a random initial design field. Final values: $\phi = 0.0876$ after 200 iterations (maximum).

The entire OC solver block (lines 145-154) can now be replaced with:

```
%% MMA UPDATE
df0 = sens(:)/obj0; dfi = dV(:)'/volfrac;
xlow = max(0,xPhys(:)-mvlim); ...
  xupp = min(1,xPhys(:)+mvlim);
[xnew,¬,¬,¬,¬,¬,¬,¬,¬,low,upp] = ...
    mmasub(numconstr,neltot,loop,xPhys(:), ...
    xlow,xupp,xold1,xold2,f0,df0,fi,dfi, ...
    low,upp, a0,ai,c,d);
xold2 = xold1; xold1 = xPhys(:); xPhys(:) ...
    = xnew;
```

As was shown by Papadopoulos et al. (2021) for Stokes flow, even better minima exists for the double pipe problem when using open boundary conditions at the outlet as in Figure 10. The better designs still uses merged channels, but only a single of the outlets are used. By decreasing the stopping tolerance for the relative change in the objective to `chlim = 1e-4`, MMA is actually able to find the better performing design shown in Figure 21. This is likely because MMA uses all gradients, both positive and negative, whereas OC only uses positive gradients (add fluid). Removing one of the outlet channels requires to remove fluid caused by negative gradients.

9.2.2 Flow reversal

Gersborg-Hansen et al. (2005) introduced the flow reversal problem, which is analogous to the compliant mechanism problem in topology optimisation of solid mechanics. Instead of giving the exact code modifications needed, instructions will be given and the implementation will be left as an exercise for the reader.

The goal is to maximise the velocity of the flow in the reverse direction of the incoming flow, denoted U_p as seen in Figure 22. A parabolic flow profile enters the domain at the left-hand side and at the outlet a zero-pressure and straight-out flow condition is imposed. The optimisation problem is formulated as a minimisation problem:

$$\begin{aligned} \underset{\boldsymbol{\gamma}}{\text{minimise:}} \quad & f_u\left(\mathbf{s}(\boldsymbol{\gamma})\right) = -U_p \\ \text{subject to:} \quad & V(\boldsymbol{\gamma}) \leq V_f \\ & p_{in}\left(\mathbf{s}(\boldsymbol{\gamma})\right) \leq \beta p_{ref} \\ \text{with:} \quad & \mathbf{r}(\mathbf{s}(\boldsymbol{\gamma}),\boldsymbol{\gamma}) = \mathbf{0} \\ & 0 \leq \gamma_i \leq 1, \ i = 1,\dots,n_{el} \end{aligned} \tag{33}$$

where the minus infront of U_p is to turn the maximisation problem into a minimisation problem. The velocity to be maximised is defined as the negative of the velocity component in the x_2-direction at the point $(x_1, x_2) = \left(\frac{L_x}{2}, \frac{L_y}{2}\right)$:

$$U_p = -u_2\left(\frac{L_x}{2}, \frac{L_y}{2}\right) \tag{34}$$

For the discretised problem, the objective functional then becomes:

$$f_u = \mathbf{l}_u{}^T\mathbf{s} \tag{35}$$

where $\mathbf{l}_u$ is a vector of zeros except at the velocity DOF of interest, where a 1 is placed. The vector product then extracts the velocity value at the DOF of interest. According to Equation 25, the adjoint right-hand side is easily found to be simply this vector:

$$\frac{\partial f_u}{\partial \mathbf{s}}^T = \mathbf{l}_u \tag{36}$$

This problem is more complex than minimum dissipation, since it includes a constraint on the inlet pressure, p_{in}, and thus requires solving an additional adjoint problem. The inlet pressure is defined as:

$$p_{in} = \frac{1}{|\Gamma_{in}|}\int_{\Gamma_{in}} p\, dS \tag{37}$$

which is equivalent to the pressure drop since zero pressure is imposed at the outlet. When discretised, this can be formulated as:

$$p_{in} = \mathbf{l}_p{}^T\mathbf{s} \tag{38}$$

where $\mathbf{l}_p$ results from the integration of the shape functions for pressure over the inlet boundary. This can be approximated as a vector of zeros with $\frac{1}{n_{in}}$ on the pressure DOFs along the inlet boundary, where n_{in} is the number of nodes along the inlet boundary. According to Equation 25, the adjoint right-hand side is easily found to be simply the vector:

$$\frac{\partial p_{in}}{\partial \mathbf{s}}^T = \mathbf{l}_p \tag{39}$$

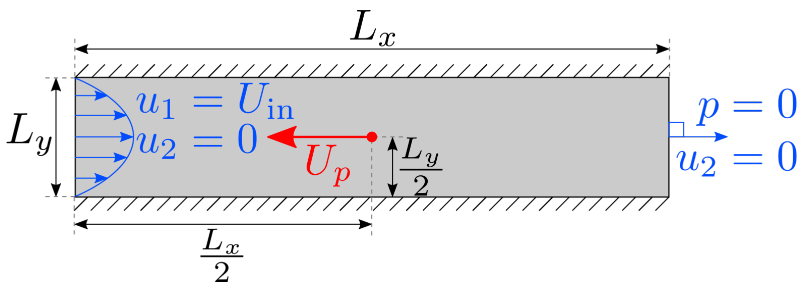


Fig. 22: Problem setup for the flow reversal problem.

To evaluate the objective and inlet pressure, the user must first define the vectors $\mathbf{l}_u$ and $\mathbf{l}_p$ under pre-processing and then the values can be evaluated in the optimisation loop using the vector products given by Equations 35 and 38. To evaluate the sensitivities, two adjoint problems must be solved. In order to do so efficiently, the right-hand side of the adjoint problem can be defined as an array with two columns `RHS = [lu lp]`, where `lu` and `lp` are $\mathbf{l}_u$ and $\mathbf{l}_p$, respectively. When solving the adjoint problem, `L = J'\RHS`, MATLAB solves both adjoint problems using the same factorisation of the Jacobian matrix. This yields two columns of `L` corresponding to the adjoint solution for the objective and inlet pressure, respectively. The sensitivities can then be computed separately for the two functionals. Please note that for both functionals, the partial derivative with respect to the design variables, $\frac{\partial f}{\partial \gamma}$, are zero.

The problem now has two constraints, one on the fluid volume and one on the inlet pressure, so `numconstr` should be set to 2. Furthermore, the arrays of constraints and sensitivities, `fi` and `dfi`, should now have two rows each corresponding to the two constraints. The inlet pressure constraint should be scaled similarly to the volume constraint, `pin/pmax - 1`, where `pin` is p_{in} and `pmax` is p_{max}, which should be defined under the definition of input parameters. Better behaviour of MMA has been observed by setting the scaling of the objective functional to just the initial objective value, rather than a tenth of this value.

It is possible for the optimiser to get stuck at a solution with zero velocity at the DOF of interest. This occurs mainly when the initial design field is constant and symmetric. In order to push the optimiser away from this local minima, a random initial design field is recommended for this problem. It is recommended to generate it using the `rand` function in MATLAB and save the generated design field once. In order to ensure the random design field is close to the prescribed design field value, in order to ensure the initial Brinkman penalty is hit, the design field can be defined as:

Parameter	Value	Code
ρ	1	`rho = 1.0`
L_x	5	`Lx = 5.0`
L_y	1	`Ly = 1.0`
μ	$^1/_{(Re_{\text{in}})}$	`mu = 1/1`
α_{min}	0	`alphamin = 0.0`
α_{max}	$^{\mu}/_{10^{-5}}$	`alphamax = mu/(1e-5)`
V_f	0.6	`volfrac = 0.6`
n_x^e	250	`nelx = 250`
n_y^e	50	`nely = 50`

Table 7: Parameter values for the flow reversal problem in Section 9.2.2.

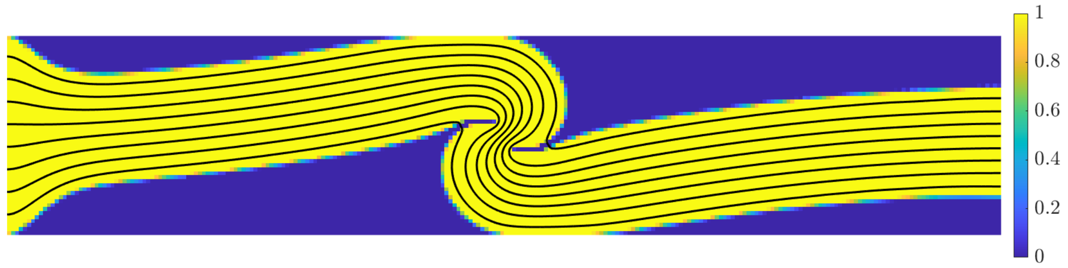
```
xPhys = xinit + 0.1*(-0.5 + ...
    rand(nely,nelx)); save('xRand.mat');
```

The same field should then be loaded when changing design parameters, ensuring a fair comparison between different parameter settings:

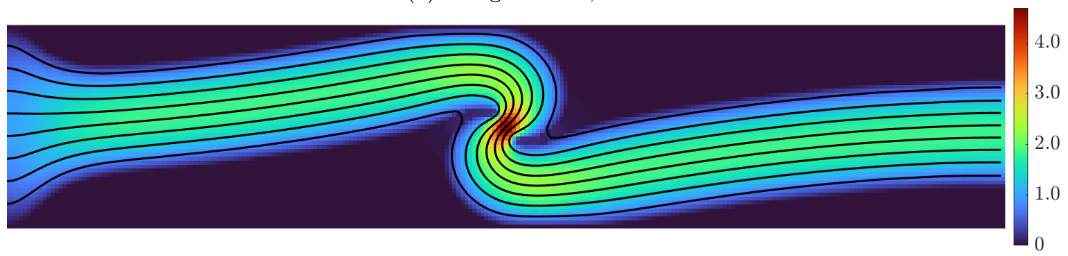
```
load('xRand.mat','xPhys');
```

The problem parameters are defined in Table 7. The maximum Brinkman penalty is set one order of magnitude higher than Gersborg-Hansen et al. (2005) used, since their value seems to still allow significant flow through solid feature, especially the thin features that appear. The maximum pressure drop is set to different factors, β, relative to the pressure drop of an empty channel, p_{ref}.

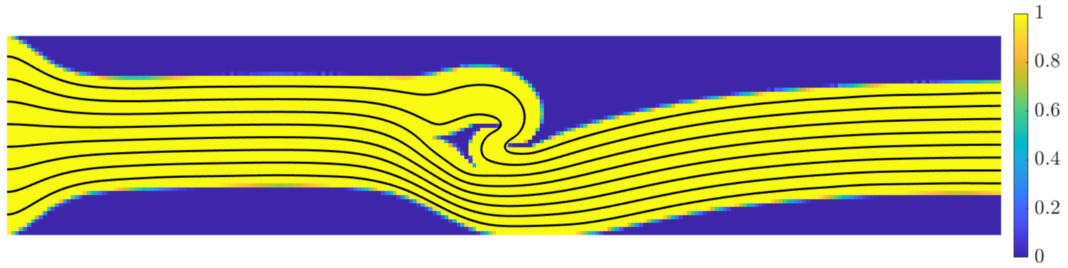
Figure 23 shows two optimised designs for $Re_{\text{in}} = 1$ that qualitatively agree with those presented by Gersborg-Hansen et al. (2005). Both designs have a characteristic S-shaped flow feature at the centre, the purpose of which is to reverse the flow direction as requested. It can be seen that for the larger pressure drop, a single channel connects the inlet to the outlet and because all the fluid has to pass through, the velocity becomes large at the centre. For the smaller pressure drop, the topology changes and a bypass channel is included to reduce the pressure drop. The 46% reduction in pressure drop leads to a significant reduction of 79% in the objective value.



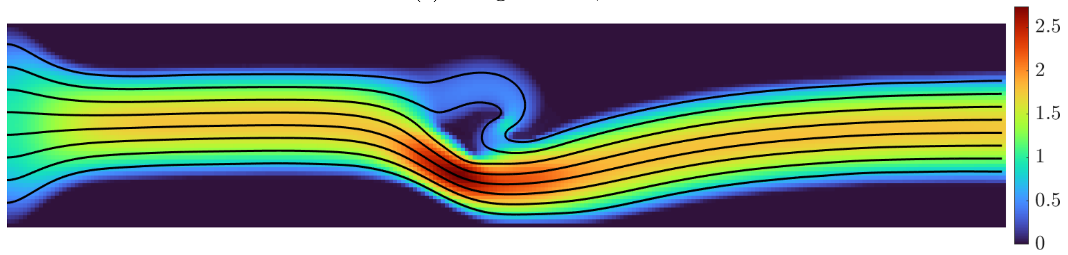
(a) Design field - $\beta = 15$



(b) Velocity magnitude - $\beta = 15$



(c) Design field - $\beta = 8$



(d) Velocity magnitude - $\beta = 8$

Fig. 23: Optimised designs and velocity fields for the flow reversal problem with $Re_{\text{in}} = 1$ and variable maximum pressure drop relative to $p_{ref} = 39.870$. Final values: (a,b) $U_p = 3.362$ after 80 iterations; (c,d) $U_p = 0.693$ after 112 iterations.

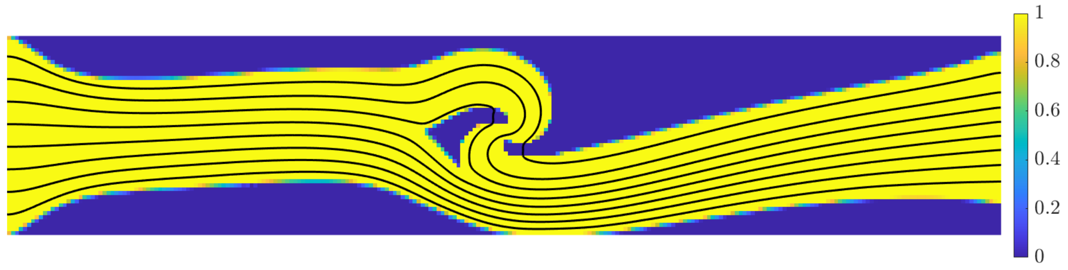
Figure 24 shows the optimised designs for $Re_{\text{in}} = 100$ for two different maximum pressure drops. It can be seen that for this higher Reynolds number, the bypass channel is present for both designs. This is because inertia is dominant in the flow and having a sharp S-turn for the entire flow volume as in Figure 23a, causes a 12 times higher pressure drop compared to the design with the smallest bypass channel in Figure 24b.

9.2.3 Drag and lift

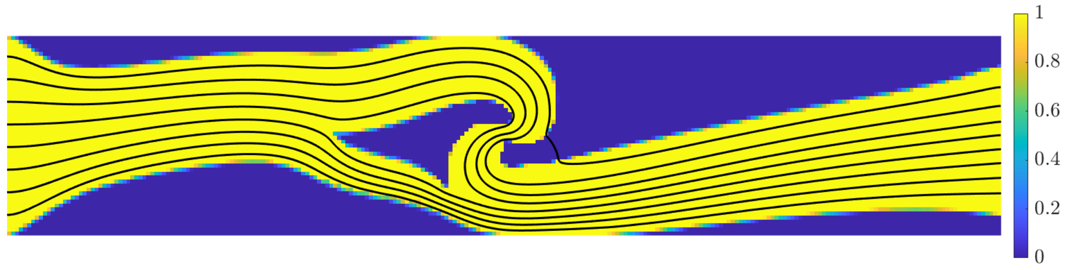
This example is inspired by the work of Kondoh et al. (2012), who presented drag minimisation and lift maximisation using topology optimisation based on a variety of objective function formulations. The simplest definition of calculating the drag and lift force are based on body forces resulting from the Brinkman penalty:

$$D = \int_{\Omega} \alpha u_1 \, dV \tag{40a}$$

$$L = \int_{\Omega} \alpha u_2 \, dV \tag{40b}$$



(a) $\beta = 15$



(b) $\beta = 30$

Fig. 24: Optimised designs and velocity fields for the flow reversal problem with $Re_{\text{in}} = 100$ and variable maximum pressure drop relative to $p_{ref} = 0.39819$. Final values: (a) $U_p = 2.264$ after 102 iterations; (b) $U_p = 4.075$ after 88 iterations.

where D is the drag force and L is the lift force. Being a volumetric integration over the computational domain, the minimum drag objective can easily be implemented by changing the objective functional in the element derivation code `analyticalElement.m` to:

```
phi = alpha*ux(1);
```

for drag and equivalent for lift. However, because the drag and lift only relate to either the x_1 or x_2 velocity DOFs, respectively, the automatically generated output from the Symbolic Toolbox for the partial derivatives $\frac{\partial D}{\partial \mathbf{s}}$ and $\frac{\partial L}{\partial \mathbf{s}}$ will contain single scalar zeros, instead of vectors of zeros for a vectorised function. The output should be modified from:

```
out1 = [t2;0;t2;0;t20;t2;0];
```

to:

```
out1 = [t2;zeros(1,length(t2));t2; ...
    (1,length(t2));t2;zeros(1,length(t2)); ...
    t2;zeros(1,length(t2))];
```

Figure 25 shows the problem setup for the drag and lift problem. Similar to the rugby ball problem, a constant velocity is imposed in the x_1-direction along the entire outer boundary with a single reference pressure point in the upper-left corner. The design domain is a subset of the computational flow inspired by the work of Kondoh et al. (2012). However, details of the physical dimensions are missing from the original work, so the values used herein are given in Table 8. Kondoh et al. (2012) defines the Reynolds number in term of a characteristic length given by the square root of the area of the solid profile:

Parameter	Value	Code
ρ	1	`rho = 1.0`
L_x	3	`Lx = 3.0`
L_y	1	`Ly = 1.0`
L_c	$\sqrt{V_f\lvert\Omega_d\rvert}$	`Lc = sqrt(volfrac*Lx/2*Ly/2)`
μ	$U_{\text{in}}L_c\rho/Re_{\text{in}}$	`mu = Uin*Lc*rho/1`
α_{min}	0	`alphamin = 0.0`
α_{max}	$\mu/(10^{-5}L_c)$	`alphamax = mu/(1e-5*Lc)`
V_f	0.85	`volfrac = 0.85`
n_x^e	300	`nelx = 300`
n_y^e	100	`nely = 100`
q_α	10	`qavec = 10`
Max. iter.	100	`conit = 100`

Table 8: Parameter values for the drag and lift problem in Section 9.2.3.

$$L_c = \sqrt{V_f|\Omega_d|} \tag{41}$$

where $|\Omega_d|$ is the size of the design domain and V_f the prescribed volume fraction. The maximum Brinkman penalty is defined as $\alpha_{\text{max}} = \frac{\mu}{(10^{-5}L_c)}$ in order to ensure a value of 10^5 for a Reynolds number of 1.

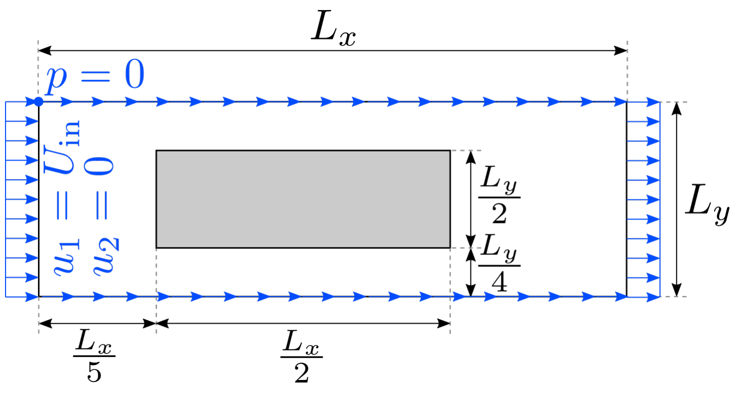


Fig. 25: Problem setup for the drag and lift problem.

The optimisation problem for minimum drag is formulated as:

$$\begin{aligned} \underset{\gamma}{\text{minimise:}} \quad & f_D\left(\mathbf{s}(\boldsymbol{\gamma}), \boldsymbol{\gamma}\right) = D \\ \text{subject to:} \quad & V(\boldsymbol{\gamma}) \leq V_f \\ \text{with:} \quad & \mathbf{r}(\mathbf{s}(\boldsymbol{\gamma}), \boldsymbol{\gamma}) = \mathbf{0} \\ & 0 \leq \gamma_i \leq 1, \; i = 1, \ldots, n_{el} \end{aligned} \tag{42}$$

which only requires a single adjoint problem for the drag force, similar to the minimum dissipation problem. For maximum lift, a maximum constraint on the drag force is necessary to avoid very thin and sharply rotated designs:

$$\begin{aligned} \underset{\gamma}{\text{minimise:}} \quad & f_L\left(\mathbf{s}(\boldsymbol{\gamma}), \boldsymbol{\gamma}\right) = -L \\ \text{subject to:} \quad & V(\boldsymbol{\gamma}) \leq V_f \\ & D\left(\mathbf{s}(\boldsymbol{\gamma}), \boldsymbol{\gamma}\right) \leq \beta D_{\text{ref}} \\ \text{with:} \quad & \mathbf{r}(\mathbf{s}(\boldsymbol{\gamma}), \boldsymbol{\gamma}) = \mathbf{0} \\ & 0 \leq \gamma_i \leq 1, \; i = 1, \ldots, n_{el} \end{aligned} \tag{43}$$

which means that a second adjoint problem is necessary, similar to the flow reversal problem. The maximum drag is set to different factors, β, relative to that of the drag minimised design, D_{ref}. Kondoh et al. (2012) also impose a constraint on the centre-of-gravity of the profile[3], which is not included herein for simplicity. Furthermore, similar to the rugby ball problem, a single interpolation factor of $q_\alpha = 10$ is sufficient for this external flow problem.

The MMA version of the code is required and this can easily be updated to handle passive elements by following the same steps as in Section 9.1 and basically replacing `xPhys`, `sens` and `dV` with only the active elements (`active`) in the definition of arrays for MMA and changing `neltot` to `nactive` in the call to MMA. It is important to scale the drag constraint similarly to the volume constraint, $^{D}/_{(\beta D_{\text{ref}})} - 1 \leq 0$, when inputting it to MMA. Furthermore, the objective functional should be scaled using the absolute of the initial value, especially for the maximisation of lift since otherwise the sign changes incorrectly.

Figure 26 shows optimised designs for increasing Reynolds number. The optimised designs are qualitatively similar to those presented by Kondoh et al. (2012). It can be seen that for $Re = 10$, a more or less symmetric profile is observed. For $Re = 100$ and $Re = 1000$, the profiles become elongated and tapered towards the trailing edge as is well-known from aerodynamics.

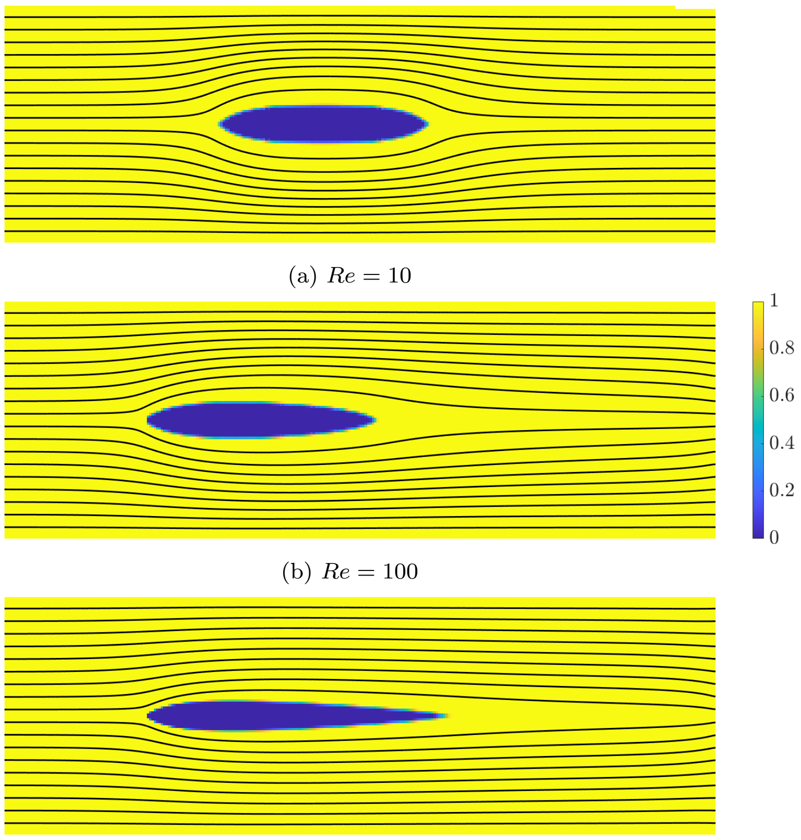
Figure 27 shows optimised designs for maximum lift at $Re = 10$. The drag optimised design from Figure 26a is used as the initial design field and the drag is constrained to be less than 10%, 20% and 100% higher than the drag optimised case. It can be seen that the optimised design becomes further elongated as larger drag is allowed. The optimised designs are qualitatively similar to those presented by Kondoh et al. (2012), except the design for $\beta = 2.0$ which has a blunt tip due to reaching the edge of the design domain.

9.3 Other extensions

Due to the similarity in structure of the presented code to the well-known 88-line code for static mechanics, many of the extensions made available to that code are applicable here. For some problems, it might be relevant to include length-scale control. For that purpose, filtering, projection and robust formulations can be directly copied from extensions to the 88-line code.

The purpose of the presented code is to stand on its own as an introductory tool for newcomers to the field. However, it is also the idea that it will serve as the basis for extension to flow-driven multiphysics. A sequel pa-

[3] The constraint is not formulated explicitly anywhere in the paper.



(a) $Re = 10$

(b) $Re = 100$

(c) $Re = 1000$

Fig. 26: Optimised designs for minimum drag for increasing Reynolds number. Final values: (a) $D = 2.4207$ after 38 iterations; (b) $D = 0.3399$ after 37 iterations; (c) $D = 0.0800$ after 44 iterations.

per is planned treating fluid-structure-interaction and conjugate heat transfer.

10 Computational performance

This subsection covers the computational time and memory use of the code. The results are given using both a workstation and a laptop to compare the upper and lower bounds of performance. The workstation was running Ubuntu 18.04.6LTS with an Intel Core i9-9980XE CPU with 18 cores at 3.0GHz and 128GB of memory. The laptop was running Windows 10 with an Intel Core i5-8350U CPU with 4 cores at 1.7GHz and 8GB of memory. Both machines were running MATLAB R2021b and had all unnecessary processes shut down. MATLAB was completely shut down and restarted to clear all memory cache between runs.

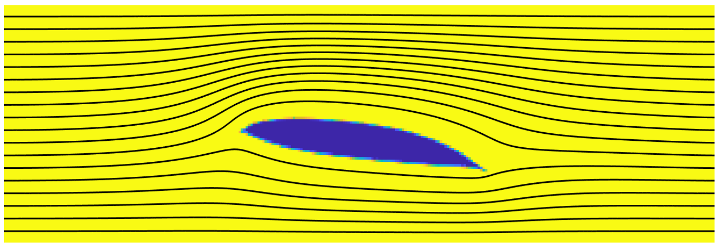
10.1 Computational time

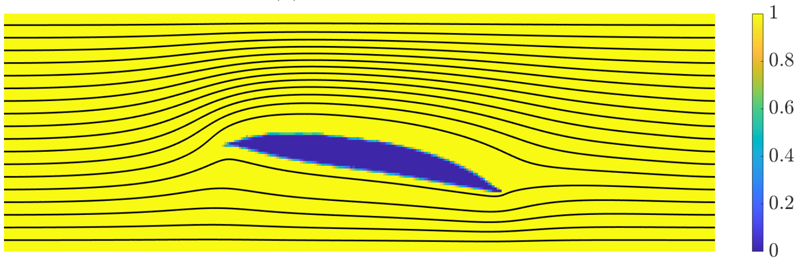
Due to the non-linear governing equations and the vastly more complex finite element formulation, the present code is significantly slower than the available codes for static mechanics. The computational time of a selected number of results from the paper are listed in Table 9. It can be seen that overall all the examples can be generated in 10 minutes or less on both of the types of machine. Generally, the laptop takes around 50% longer than the workstation, which is mostly attributed to the significantly lower clock-speed of the CPU, but a high number of cores are likely also helpful for the vectorised assembly procedure.

The computational time can easily be decreased by doing one or both of the following:

1. Loosening the tolerance of the non-linear solver
2. Re-using the last Jacobian from the non-linear solver for the adjoint solver

The tolerance of the non-linear solver, `nltol`, determines the number of iterations needed to provide a solution with the required accuracy. Loosening the toler-


(a) $\beta = 1.1$



(b) $\beta = 1.2$

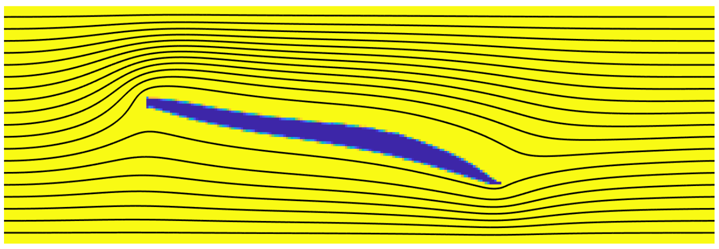

(c) $\beta = 2.0$

Fig. 27: Optimised designs for maximum lift at $Re = 10$ under increasing drag constraint relative to $D_{\text{ref}} = 2.4207$ on the drag. Final values: (a) $L = 2.0148$ after 100 iterations (maximum); (b) $L = 3.1882$ after 100 iterations (maximum); (c) $L = 7.2880$ after 60 iterations.

				Workstation			Laptop		
Problem	*Re*	Elements	Iter.	Total [min]	Iteration [s]	Memory [GB]	Total [min]	Iteration [s]	% slower
Double pipe	20	15606	102	4.10	2.41	1.6	6.08	3.58	48
Pipe bend	500	10000	40	1.34	2.01	1.6	1.2	2.92	44
Rugby ball	1	10000	19	0.49	1.55	1.6	1.3	2.26	47
Flow reversal	100	12500	90	3.96	2.33	1.4	6.00	3.50	52
Maximum lift	10	30000	100	7.07	4.24	2.3	10.60	6.36	50

Table 9: Computational time for selected results from the paper using both a workstation and a laptop with specifications given in the text. Total time is given in minutes and the average time per design iteration is given in second. The values correspond to the following results: double pipe = Figure 15a; pipe bend = Figure 18; rugby ball = Figure 20a; flow reversal = Figure 24a; maximum lift = Figure 27b.

ance (increasing the number) gives a less accurate solution to the non-linear system of equations in Equation 16. This means that errors will be introduced into the adjoint sensitivity analysis due to a non-zero residual. However, as detailed by Amir et al. (2010, 2014) for linear solvers, solver tolerances can be loosened significantly without impacting accuracy significantly.

The transposed Jacobian is necessary for solving the adjoint problem and by default it is built using the final converged (to the desired accuracy) state solution. This ensures an accurate (to solution accuracy) evaluation of the adjoint variables and, thus, the sensitivities. However, if the non-linear tolerance is tight (a low number), the state field will not have changed significantly from the second-to-last non-linear iteration to the last one. Therefore, the Jacobian already built for the last non-linear iteration can be used for solving the adjoint problem and this saves the time of assemblying the Jacobian a single time per design iteration.

	Standard		Re-use Jacobian			Loose tolerance			Both modifications		
Problem	Obj.	Iter.	Obj.	Iter.	Time [%]	Obj.	Iter.	Time [%]	Obj.	Iter.	Time [%]
Double pipe	2.126	102	2.129	100	89.7	2.129	100	65.4	2.130	99	55.9
Flow reversal	2.264	102	2.257	105	88.7	2.232	89	67.4	2.260	83	63.7
Maximum lift	3.188	$100m$	3.232	$100m$	91.2	3.243	$100m$	68.6	3.330	$100m$	58.3

Table 10: Final objective value, number of design iterations, and average computational time per design iteration for selected results from the paper using a laptop with re-use of Jacobian from non-linear solver for the adjoint solver and/or loose tolerance of non-linear solver. All objective values are evaluated with the default tolerance of `nltol = 1e-6`. For the double pipe, the objective values should be multiplied by 10^{-1}. All runs for maximum lift maxed out at 100 iterations (100m).

Table 10 shows the computational time for three examples using the two computational time reduction methods on the laptop. It can be seen that re-using the Jacobian can reduce the computational time per design iteration, but only by 8-11%. Loosening the tolerance to `nltol = 1e-2`, compared to the default value of `nltol = 1e-6`, reduces the computational cost more significantly more by 31-35%. Generally, the looser tolerance reduces the number of non-linear iterations by 1 or 2 per design iteration, depending on the non-linearity of the flow problem. Lastly, combining the two modifications together yields a total reduction of 36-44%. The reduction in computational time does come at a cost of accuracy, which is reflected in the different number of design iterations and different final values of the objective functionals. Despite the differences in objective values, visually-identical optimised designs are obtained for all cases. However, this may not be a general conclusion and it will depend on the physical problem and optimisation problem to be solved.

10.2 Memory use

The memory use of the code is of interest, because the local finite element matrices are formed all-at-once using the vectorised functionality of the auto-generated functions from the Symbolic Toolbox. Ideally, this means that the computer should be able to store all local finite element matrices in memory all at once - in addition to the factorisation of the Jacobian matrix for the solution of the systems of equations.

Table 9 also shows the amount of peak RAM memory used for the 5 examples, when run on the workstation (similar values were observed on the laptop). This is in addition to the approximately 1.2GB that MATLAB used in idle after startup. The laptop was able to handle a mesh of 90 thousand elements (300×300) without exceeding the 8GB of RAM and avoiding to resort to swap storage. A design iteration for the pipe bend problem took around 30 seconds for 5 Newton iterations. The workstation could easily handle a mesh of 1 million elements (1000×1000), peaking at 50GB of RAM memory use and solving one design iteration in around 400 seconds (6.7 minutes).

By default, the `topFlow.m` script starts with the `clear` command. This ensures that any cached functions are not cleared and should speed up running of the code, after the first initial run. However, memory use can build up and if time is not an issue, the command can be updated to `clear all`. This clears the memory everytime the script is run. Beware that in this case, the initial design iterations will be slower, because MATLAB realises many of the functions are being called repeatedly.

11 Concluding remarks

This article presents a detailed introduction to density-based topology optimisation of fluid flow problems. The goal is to be the first point of contact for new students and researchers, allowing them to quickly get started in the research area and to skip many of the initial steps, often consuming unnecessarily long time from the scientific advancement of the field.

A step-by-step guide is provided to the components necessary to understand and implement topology optimisation for fluid flow. The continuous design representation and Brinkman penalty approach are examined in detail using parametric simulations of a reference geometry, as well as through optimisation examples. Under the way, several recommendations are given to the choice of minimum, maximum and initial Brinkman penalty value.

The guide is aided by a MATLAB code based on the well-known “88-line” code for static mechanics, but with significant modifications to treat Navier-Stokes flow. The code uses vectorised functions for building all element-level matrices and vectors, which have been generated using the MATLAB Symbolic Toolbox. All partial derivatives necessary for both the Newton solver and adjoint sensitivity analysis are automatically computed using symbolic differentiation.

The extendability of the code is demonstrated through various additional modifications and explanations. The code is shown to be efficient, but the computational time is significantly higher than that for static mechanics, due to the significantly more complex finite element formulation and the non-linear nature of the governing equations.

Acknowledgements The author would like to acknowledge Yupeng Sun, for providing feedback on a close-to-finished manuscript, and Christian Lundgaard, for providing feedback on a very early draft. The author also wishes to thank the DTU TopOpt group for discussions over the many years it has taken to form the experienced embodied in this article.

Conflict of interest

The author has no conflict of interest.

Reproduction of results

The code to reproduce all results is either directly provided or the required modifications are explained thoroughly. Upon proof-of-attempt, the code for the examples not provided can be obtained from the author. Lastly, manual derivations of the analytical sensitivities can be provided upon request.

References

Aage N, Andreassen E, Lazarov BS (2015) Topology optimization using PETSc: An easy-to-use, fully parallel, open source topology optimization framework. Structural and Multidisciplinary Optimization 51(3):565–572, DOI 10.1007/s00158-014-1157-0

Alexandersen J (2022) Code base on GitHub,. GitHub URL Link will be added upon publication.

Alexandersen J, Andreasen CS (2020) A review of topology optimisation for fluid-based problems. Fluids 5(1), DOI 10.3390/fluids5010029

Amir O, Stolpe M, Sigmund O (2010) Efficient use of iterative solvers in nested topology optimization. Structural and Multidisciplinary Optimization 42(1):55–72, DOI 10.1007/s00158-009-0463-4

Amir O, Aage N, Lazarov BS (2014) On multigrid-CG for efficient topology optimization. Structural and Multidisciplinary Optimization 49(5):815–829, DOI 10.1007/s00158-013-1015-5

Andreassen E, Clausen A, Schevenels M, Lazarov BS, Sigmund O (2011) Efficient topology optimization in MATLAB using 88 lines of code. Structural and Multidisciplinary Optimization 43(1):1–16, DOI 10.1007/s00158-010-0594-7

Bendsøe MP, Kikuchi N (1988) Generating optimal topologies in structural design using a homogenization method. Computer Methods in Applied Mechanics and Engineering 71(2):197–224, DOI https://doi.org/10.1016/0045-7825(88)90086-2

Bendsøe MP, Sigmund O (2004) Topology Optimization: Theory, Methods, and Applications, 2nd edn. Springer-Verlag Berlin Heidelberg, DOI 10.1007/978-3-662-05086-6

Borrvall T, Petersson J (2003) Topology optimization of fluids in Stokes flow. International Journal for Numerical Methods in Fluids 41(1):77–107, DOI 10.1002/fld.426

Challis VJ (2010) A discrete level-set topology optimization code written in Matlab. Structural and Multidisciplinary Optimization 41(3):453–464, DOI 10.1007/s00158-009-0430-0

Choi KK, Kim NH (2005a) Structural Sensitivity Analysis and Optimization 1. Mechanical Engineering Series, Springer, DOI 10.1007/b138709

Choi KK, Kim NH (2005b) Structural Sensitivity Analysis and Optimization 2. Mechanical Engineering Series, Springer, DOI 10.1007/b138895

Deaton JD, Grandhi RV (2014) A survey of structural and multidisciplinary continuum topology optimization: post 2000. Structural and Multidisciplinary Optimization 49(1):1–38, DOI 10.1007/s00158-013-0956-z

Ferrari F, Sigmund O (2020) A new generation 99 line Matlab code for compliance topology optimization and its extension to 3D. Structural and Multidisciplinary Optimization 62(4):2211–2228, DOI 10.1007/s00158-020-02629-w

Gersborg-Hansen A, Sigmund O, Haber RB (2005) Topology optimization of channel flow problems. Structural and Multidisciplinary Optimization 30(3):181–192, DOI 10.1007/s00158-004-0508-7

Høghøj LC, Nørhave DR, Alexandersen J, Sigmund O, Andreasen CS (2020) Topology optimization of two fluid heat exchangers. International Journal of Heat and Mass Transfer 163, DOI 10.1016/j.ijheatmasstransfer.2020.120543

Kondoh T, Matsumori T, Kawamoto A (2012) Drag minimization and lift maximization in laminar flows via topology optimization employing simple objective function expressions based on body force integration. Structural and Multidisciplinary Optimization 45(5):693–701, DOI 10.1007/s00158-011-0730-z

Le C, Norato J, Bruns T, Ha C, Tortorelli D (2010) Stress-based topology optimization for continua. Structural and Multidisciplinary Optimization 41(4):605–620, DOI 10.1007/s00158-009-0440-y

Lundgaard C, Alexandersen J, Zhou M, Andreasen CS, Sigmund O (2018) Revisiting density-based topology optimization for fluid-structure-interaction problems. Structural and Multidisciplinary Optimization 58(3):969–995, DOI 10.1007/s00158-018-1940-4

Michaleris P, Tortorelli DA, Vidal CA (1994) Tangent operators and design sensitivity formulations for transient non-linear coupled problems with applications to elastoplasticity. International Journal for Numerical Methods in Engineering 37(14):2471–2499, DOI https://doi.org/10.1002/nme.1620371408

Papadopoulos IPA, Farrell PE, Surowiec TM (2021) Computing multiple solutions of topology optimization problems. SIAM Journal on Scientific Computing 43(3):A1555–A1582, DOI 10.1137/20m1326209

Pereira A, Talischi C, Paulino GH, M Menezes IF, Carvalho MS (2016) Fluid flow topology optimization in PolyTop: stability and computational implementation. Structural and Multidisciplinary Optimization 54(5):1345–1364, DOI 10.1007/s00158-014-1182-z

Pironneau O (1973) On optimum profiles in Stokes flow. Journal of Fluid Mechanics 59(1):117–128, DOI 10.1017/S002211207300145X

Sigmund O (2001) A 99 line topology optimization code written in matlab. Structural and Multidisciplinary Optimization 21(2):120–127, DOI 10.1007/s001580050176

Sigmund O (2007) Morphology-based black and white filters for topology optimization. Structural and Multidisciplinary Optimization 33(4):401–424, DOI 10.1007/s00158-006-0087-x

Sigmund O, Maute K (2013) Topology optimization approaches. Structural and Multidisciplinary Optimization 48(6):1031–1055, DOI 10.1007/s00158-013-0978-6

da Silva GA, Beck AT, Sigmund O (2019) Stress-constrained topology optimization considering uniform manufacturing uncertainties. Computer Methods in Applied Mechanics and Engineering 344:512–537, DOI https://doi.org/10.1016/j.cma.2018.10.020

SMOPTIT (2022) Svanberg matematisk optimering och IT AB,. SMOPTIT URL http://www.smoptit.se/

Stolpe M, Svanberg K (2001) An alternative interpolation scheme for minimum compliance topology optimization. Structural and Multidisciplinary Optimization 22(2):116–124, DOI 10.1007/s001580100129

Svanberg K (1987) The method of moving asymptotes - a new method for structural optimization. International Journal for Numerical Methods in Engineering 24(2):359–373, DOI https://doi.org/10.1002/nme.1620240207

Svanberg K (2007) MMA and GCMMA – two methods for nonlinear optimization. Report, KTH, URL https://people.kth.se/~krille/mmagcmma.pdf

Talischi C, Paulino GH, Pereira A, Menezes IFM (2012) PolyTop: a Matlab implementation of a general topology optimization framework using unstructured polygonal finite element meshes. Structural and Multidisciplinary Optimization 45(3):329–357, DOI 10.1007/s00158-011-0696-x

Wang C, Zhao Z, Zhou M, Sigmund O, Zhang XS (2021) A comprehensive review of educational articles on structural and multidisciplinary optimization. Structural and Multidisciplinary Optimization 64(5):2827–2880, DOI 10.1007/s00158-021-03050-7

Wei P, Li Z, Li X, Wang MY (2018) An 88-line MATLAB code for the parameterized level set method based topology optimization using radial basis functions. Structural and Multidisciplinary Optimization 58(2):831–849, DOI 10.1007/s00158-018-1904-8

Xia L, Breitkopf P (2015) Design of materials using topology optimization and energy-based homogenization approach in Matlab. Structural and Multidisciplinary Optimization 52(6):1229–1241, DOI 10.1007/s00158-015-1294-0

Zegard T, Paulino GH (2014) GRAND — Ground structure based topology optimization for arbitrary 2d domains using MATLAB. Structural and Multidisciplinary Optimization 50(5):861–882, DOI 10.1007/s00158-014-1085-z

A Alternative scaling of Brinkman penalty factor

The order of magnitude of the convective term is constant due to the non-dimensional formulation of Equation 2. Thus, the Brinkman penalty factor should no longer become smaller with increasing Reynolds number. Kondoh et al. (2012) suggested the following scaling:

$$\alpha_{\max} = \left(1 + \frac{1}{Re}\right)\frac{1}{Da} \tag{44}$$

which aims to ensure that the Brinkman penalty factor is large enough even for increasing Reynolds numbers. From a scaling point of view, it can be argued that $\frac{1}{Da}$ makes sense for purely convective flow, which is recovered from the above for $Re \longrightarrow \infty$.

Figure 28 shows the error measures as a function of Reynolds number when using Equation 44. In practise, it is seen that the error decreases for increasing Reynolds number over 1, meaning the flow in the solid is penalised increasingly hard. It appears that the error is beginning to stagnate. However, the flow becomes non-steady for $Re > 150$ and, thus, the limit cannot be investigated presently. This is an area worth more investigation, as the community transitions to treating larger Reynolds number flows.

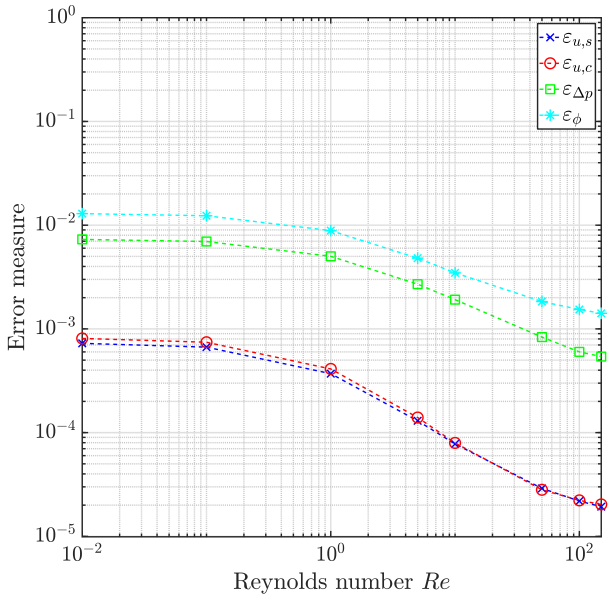


Fig. 28: Error measures, when penalising using Equation 44, as a function of Reynolds number for a Darcy number of $Da = 10^{-6}$ and $Re \in \{0.01, 0.1, 1, 5, 10, 50, 100, 150\}$.

B Finite element formulation

The weak form of the momentum conservation equations is derived by multiplying the strong form by the test function w_i for the velocity field, integrating over the volume, applying integration-by-parts and introducing a zero normal stress natural boundary condition:

$$\int_{\Omega} \rho w_i u_j \frac{\partial u_i}{\partial x_j} \, dV + \int_{\Omega} \mu \frac{\partial w_i}{\partial x_j} \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) dV - \int_{\Omega} \frac{\partial w_i}{\partial x_i} p \, dV + \int_{\Omega} \alpha w_i u_i \, dV + \sum_{e=1}^{n_e} \int_{\Omega_e} \tau w_k \frac{\partial u_i}{\partial x_k} \left(\rho u_j \frac{\partial u_i}{\partial x_j} + \frac{\partial p}{\partial x_i} + \alpha u_i \right) dV = 0 \quad (45)$$

where the last integral is the additional SUPG stabilisation terms with τ as the stabilisation parameter. Likewise, the weak form of the mass conservation equation is derived by multiplying the strong form with the test function q for the pressure field:

$$\int_{\Omega} q \frac{\partial u_i}{\partial x_i} \, dV + \sum_{e=1}^{n_e} \int_{\Omega_e} \frac{\tau}{\rho} \frac{\partial q}{\partial x_i} \left(u_j \frac{\partial u_i}{\partial x_j} + \frac{\partial p}{\partial x_i} + \alpha u_i \right) dV = 0 \quad (46)$$

where the last integral is the additional PSPG stabilisation terms with τ as the stabilisation parameter. The diffusive term has been left out of the SUPG and PSPG stabilisation, since it is negligible for bi-linear interpolation functions due to the second-order derivative.

The stabilisation parameter is computed using an approximate minimum function:

$$\tau = \left({\tau_1}^{-2} + {\tau_3}^{-2} + {\tau_4}^{-2} \right)^{-1/2} \quad (47)$$

based on three limit cases:

$$\tau_1 = \frac{h}{2\sqrt{u_i u_i}} \quad (48a)$$

$$\tau_3 = \frac{\rho h^2}{12\mu} \quad (48b)$$

$$\tau_4 = \frac{\rho}{\alpha} \quad (48c)$$

where τ_1 is the convective limit, τ_3 is the diffusive limit, and τ_4 is the reactive limit[4]. The reactive limit τ_4 is very important to ensure stability in the solid domain and at the interface, especially for large Brinkman penalty parameters. The stabilisation parameters are assumed to be constant within each element and τ_1 is, thus, computed based on the velocity components evaluated in the element centroid.

C Adjoint sensitivity analysis

Adjoint sensitivity analysis need not be difficult or especially derived for every type of problem. The derivation is very simply by posing the system of equations (be they uncoupled, weakly coupled or strongly coupled) as a common residual:

$$\mathbf{r} = \mathbf{A}\,\mathbf{s} - \mathbf{b} = \mathbf{0} \quad (49)$$

where $\mathbf{A}$ is the system coefficient matrix, $\mathbf{s}$ is the vector of all state variables and $\mathbf{b}$ is the forcing vector. To find the derivatives of a given functional f, the Lagrangian $\mathcal{L}$ is defined as:

$$\mathcal{L} = f - \boldsymbol{\lambda}^T \mathbf{r} \quad (50)$$

where $\boldsymbol{\lambda}$ is the vector of adjoint variables. The total derivative with respect to a design variable γ_e is then taken of the Lagrangian:

$$\frac{d\mathcal{L}}{d\gamma_e} = \frac{df}{d\gamma_e} - \boldsymbol{\lambda}^T \frac{d\mathbf{r}}{d\gamma_e} \quad (51)$$

where the total derivative is given by:

$$\frac{df}{d\gamma_e} = \frac{\partial f}{\partial \gamma_e} + \frac{\partial f}{\partial \mathbf{s}} \frac{\partial \mathbf{s}}{\partial \gamma_e} \quad (52)$$

due to the implicit dependence of f on the state field. Expanding the total derivative of the Lagrangian gives:

$$\frac{d\mathcal{L}}{d\gamma_e} = \frac{\partial f}{\partial \gamma_e} + \frac{\partial f}{\partial \mathbf{s}} \frac{\partial \mathbf{s}}{\partial \gamma_e} - \boldsymbol{\lambda}^T \left(\frac{\partial \mathbf{r}}{\partial \gamma_e} + \frac{\partial \mathbf{r}}{\partial \mathbf{s}} \frac{\partial \mathbf{s}}{\partial \gamma_e} \right) \quad (53)$$

which can be rewritten to:

$$\frac{d\mathcal{L}}{d\gamma_e} = \frac{\partial f}{\partial \gamma_e} - \boldsymbol{\lambda}^T \frac{\partial \mathbf{r}}{\partial \gamma_e} + \left(\frac{\partial f}{\partial \mathbf{s}} - \boldsymbol{\lambda}^T \frac{\partial \mathbf{r}}{\partial \mathbf{s}} \right) \frac{\partial \mathbf{s}}{\partial \gamma_e} \quad (54)$$

by collecting the terms multiplied by the derivative of the state field. The adjoint problem is then defined as what is inside the brackets:

$$\frac{\partial \mathbf{r}}{\partial \mathbf{s}}^T \boldsymbol{\lambda} = \frac{\partial f}{\partial \mathbf{s}}^T \quad (55)$$

[4] τ_2 is for the transient case, which is traditionally numbered as number 2, and left out since steady-state flow is treated herein.

When $\boldsymbol{\lambda}$ is the solution to the adjoint problem, the terms inside the brackets become zero and it is avoided to compute the design sensitivities of the state field:

$$\frac{d\mathcal{L}}{d\gamma_e} = \frac{\partial f}{\partial \gamma_e} - \boldsymbol{\lambda}^T \frac{\partial \mathbf{r}}{\partial \gamma_e} \tag{56}$$

Since the state solution will be updated after a design change to make the residual equal to zero, the total derivative of the residual with respect to the design variable is equal to zero. Thus, the total derivative of the Lagrangian will be equal to that of the functional and Equation 51 gives:

$$\frac{d\mathcal{L}}{d\gamma_e} = \frac{df}{d\gamma_e} \tag{57}$$

Thus, the final sensitivities of the given functional become:

$$\frac{df}{d\gamma_e} = \frac{\partial f}{\partial \gamma_e} - \lambda^T \frac{\partial \mathbf{r}}{\partial \gamma_e} \tag{58}$$

The above result is valid for ALL systems of equations, be they linear/non-linear or un/weakly/strongly coupled. This is the methodology laid forth in various textbooks, e.g. the "Structural Sensitivity Analysis and Optimization" series by Choi and Kim (2005a,b), and papers from the 1990s Michaleris et al. (1994).

D MATLAB code: `topFlow.m`

```matlab
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%    NAVIER-STOKES TOPOLOGY OPTIMISATION CODE, MAY 2022    %
% COPYRIGHT (c) 2022, J ALEXANDERSEN. BSD 3-CLAUSE LICENSE %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear; close all; clc;
%% DEFINITION OF INPUT PARAMETERS
% PROBLEM TO SOLVE (1 = DOUBLE PIPE; 2 = PIPE BEND)
probtype = 1;
% DOMAIN SIZE
Lx = 1.0; Ly = 1.0;
% DOMAIN DISCRETISATION
nely = 30; nelx = nely*Lx/Ly;
% ALLOWABLE FLUID VOLUME FRACTION
volfrac = 1/3; xinit = volfrac;
% PHYSICAL PARAMETERS
Uin = 1e0; rho = 1e0; mu = 1e0;
% BRINKMAN PENALISATION
alphamax = 2.5*mu/(0.01^2); alphamin = 2.5*mu/(100^2);
% CONTINUATION STRATEGY
ainit = 2.5*mu/(0.1^2);
qinit = (-xinit*(alphamax-alphamin) - ainit + alphamax)/(xinit*(ainit-alphamin));
qavec = qinit./[1 2 10 20]; qanum = length(qavec); conit = 50;
% OPTIMISATION PARAMETERS
maxiter = qanum*conit; mvlim = 0.2; plotdes = 0;
chlim = 1e-3; chnum = 5;
% NEWTON SOLVER PARAMETERS
nltol = 1e-6; nlmax = 25; plotres = 0;
% EXPORT FILE
filename='output'; exportdxf = 0;
%% PREPARE FINITE ELEMENT ANALYSIS
dx = Lx/nelx; dy = Ly/nely;
nodx = nelx+1; nody = nely+1; nodtot = nodx*nody;
neltot = nelx*nely; doftot = 3*nodtot;
% NODAL CONNECTIVITY
nodenrs = reshape(1:nodtot,nody,nodx);
edofVecU = reshape(2*nodenrs(1:end-1,1:end-1)+1,neltot,1);
edofMatU = repmat(edofVecU,1,8)+repmat([0 1 2*nely+[2 3 0 1] -2 -1],neltot,1);
edofVecP = reshape(nodenrs(1:end-1,1:end-1),neltot,1);
edofMatP = repmat(edofVecP,1,4)+repmat([1 nely+[2 1] 0],neltot,1);
edofMat = [edofMatU 2*nodtot+edofMatP];
iJ = reshape(kron(edofMat,ones(12,1))',144*neltot,1);
jJ = reshape(kron(edofMat,ones(1,12))',144*neltot,1);
iR = reshape(edofMat',12*neltot,1); jR = ones(12*neltot,1);
jE = repmat(1:neltot,12,1);
%% DEFINE BOUNDARY CONDITIONS
% DEFINE THE PROBLEMS IN SEPARATE MATLAB FILE
run('problems.m');
% NULLSPACE MATRICES FOR IMPOSING BOUNDARY CONDITIONS
EN=speye(doftot); ND=EN; ND(fixedDofs,fixedDofs)=0.0; EN=EN-ND;
% VECTORS FOR FREE DOFS
alldofs = 1:doftot; freedofs = setdiff(alldofs,fixedDofs);
%% INITIALISATION
% SOLUTION VECTOR
S = zeros(doftot,1); dS = S; L = S;
S(fixedDofs) = DIR(fixedDofs);
% DESIGN FIELD
xPhys = xinit*ones(nely,nelx);
% COUNTERS
loop = 0; loopcont = 0; nlittot = 0; chcnt = 0;
% CHANGE
change = Inf; objOld = Inf;
% CONTINUATION
qastep = 1; qa = qavec(1);
% VECTORISED CONSTANTS
dxv = dx*ones(1,neltot); dyv = dy*ones(1,neltot);
muv = mu*ones(1,neltot); rhov = rho*ones(1,neltot);
%% OUTPUT PROBLEM INFORMATION
fprintf('=========================================================\n');
fprintf('      Problem number: %2i - Reynolds number: %3.2e\n',probtype,Renum);
fprintf('=========================================================\n');
fprintf('      Design it.:   0\n');
%% START ITERATION
destime = tic; ittime = tic;
while (loop ≤ maxiter)
    if (plotdes); figure(1); imagesc(xPhys); colorbar; caxis([0 1]); axis equal; axis off; ...
        drawnow; end
    %% GREYSCALE INDICATOR
    Md = 100*full(4*sum(xPhys(:).*(1-xPhys(:)))/neltot);
    %% MATERIAL INTERPOLATION
    alpha = alphamin + (alphamax-alphamin)*(1-xPhys(:))./(1+qa*xPhys(:));
    dalpha = (qa*(alphamax - alphamin)*(xPhys(:) - 1))./(xPhys(:)*qa + 1).^2 - (alphamax - ...
        alphamin)./(xPhys(:)*qa + 1);
    %% NON-LINEAR NEWTON SOLVER
    normR = 1; nlit = 0; fail = -1; nltime = tic;
    while (fail ≠ 1)
        nlit = nlit+1; nlittot = nlittot+1;
        % BUILD RESIDUAL AND JACOBIAN
        sR = RES(dxv,dyv,muv,rhov,alpha(:)',S(edofMat'));
        R = sparse(iR,jR,sR(:)); R(fixedDofs) = 0;
        if (nlit == 1); r0 = norm(R); end
        r1 = norm(R); normR = r1/r0;
        if (plotres); figure(6); semilogy(nlittot,normR,'x'); axis square; grid on; hold on; end
        if (normR < nltol); break; end
        sJ = JAC(dxv,dyv,muv,rhov,alpha(:)',S(edofMat'));
        J = sparse(iJ,jJ,sJ(:)); J = (ND'*J*ND+EN);
        % CALCULATE NEWTON STEP
        dS = -J\R;
        % L2-NORM LINE SEARCH
        Sp = S + 0.5*dS;
        sR = RES(dxv,dyv,muv,rhov,alpha(:)',Sp(edofMat'));
        R = sparse(iR,jR,sR(:)); R(fixedDofs) = 0; r2 = norm(R);
        Sp = S + 1.0*dS;
        sR = RES(dxv,dyv,muv,rhov,alpha(:)',Sp(edofMat'));
        R = sparse(iR,jR,sR(:)); R(fixedDofs) = 0; r3 = norm(R);
        % SOLUTION UPDATE WITH "OPTIMAL" DAMPING
        lambda = max(0.01,min(1.0,(3*r1 + r3 - 4*r2)/(4*r1 + 4*r3 - 8*r2)));
        S = S + lambda*dS;
        % IF FAIL, RETRY FROM ZERO SOLUTION
        if (nlit == nlmax && fail < 0); nlit = 0; S(freedofs) = 0.0; normR=1; fail = fail+1; end
        if (nlit == nlmax && fail < 1); fail = fail+1; end
    end
    nltime=toc(nltime);
    fprintf('      Newton it.: %2i - Res. norm: %3.2e - Sol. time: %6.3f sec\n',nlit,normR,nltime);
    if (fail == 1); error('ERROR: Solver did not converge after retry from zero!\n      Stopping ...
        optimisation.\n'); end
    %% OBJECTIVE EVALUATION
    obj = sum( PHI(dxv,dyv,muv,alpha(:)',S(edofMat')) );
    change = abs(objOld-obj)/objOld; objOld = obj;
    %% VOLUME CONSTRAINT
    V = mean(xPhys(:));
    %% PRINT RESULTS
    ittime = toc(ittime);
    fprintf('      Obj.: %3.2e - Constr.: %3.2e - Md: %3.2f\n',obj,V,Md);
    fprintf('      Change: %4.3e - It. time: %6.3f sec\n',change,ittime);
    fprintf('      Contin. step: %2i - qa: %4.3e\n',qastep,qa);
    ittime = tic;
    %% EVALUATE CURRENT ITERATE - CONTINUE UNLESS CONSIDERED CONVERGED
    if (change < chlim); chcnt = chcnt + 1; else; chcnt = 0; end
    if (qastep == qanum && ( (chcnt == chnum) || (loopcont == conit) ) ); break; end
    %% PRINT HEADER FOR ITERATION
    loop = loop + 1; loopcont = loopcont + 1;
    fprintf('---------------------------------------------------------\n');
    fprintf('      Design it.:%4i\n',loop);
    %% ADJOINT SOLVER
    sR = [dPHIds(dxv,dyv,muv,alpha(:)',S(edofMat')); zeros(4,neltot)];
    RHS = sparse(iR,jR,sR(:)); RHS(fixedDofs) = 0;
    sJ = JAC(dxv,dyv,muv,rhov,alpha(:)',S(edofMat'));
    J = sparse(iJ,jJ,sJ(:)); J = (ND'*J*ND+EN);
    L = J'\RHS;
    %% COMPUTE SENSITIVITIES
    % OBJECTIVE
    sR = dRESdg(dxv,dyv,muv,rhov,alpha(:)',dalpha(:)',S(edofMat'));
    dRdg = sparse(iR(:),jE(:),sR(:));
    dphidg = dPHIdg(dxv,dyv,muv,alpha(:)',dalpha(:)',S(edofMat'));
    sens = reshape(dphidg - L'*dRdg,nely,nelx);
    % VOLUME CONSTRAINT
    dV = ones(nely,nelx)/neltot;
    %% OPTIMALITY CRITERIA UPDATE OF DESIGN VARIABLES AND PHYSICAL DENSITIES
    xnew = xPhys; xlow = xPhys(:)-mvlim; xupp = xPhys(:)+mvlim;
    ocfac = xPhys(:).*max(1e-10,(-sens(:)./dV(:))).^(1/3);
    l1 = 0; l2 = ( 1/(neltot*volfrac)*sum( ocfac ) )^3;
    while (l2-l1)/(l1+l2) > 1e-3
        lmid = 0.5*(l2+l1);
        xnew(:) = max(0,max(xlow,min(1,min(xupp,ocfac/(lmid^(1/3))))));
        if mean(xnew(:)) > volfrac; l1 = lmid; else; l2 = lmid; end
    end
    xPhys = xnew;
    %% CONTINUATION UPDATE
    if (qastep < qanum && (loopcont == conit || chcnt == chnum) )
        loopcont = 0; chcnt = 0;
        qastep = qastep + 1; qa = qavec(qastep);
    end
end
%% PRINT FINAL INFORMATION
destime = toc(destime);
fprintf('=========================================================\n');
fprintf('      Number of design iterations: %4i\n',loop);
fprintf('      Final objective: %4.3e\n',obj);
fprintf('      Total time taken: %6.2f min\n',destime/60);
fprintf('=========================================================\n');
%% PLOT RESULTS
run('postproc.m');
if (exportdxf); run('export.m'); end
%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This code was written by: Joe Alexandersen                             %
%                           Department of Mechanical and                 %
%                                         Electrical Engineering         %
%                           University of Southern Denmark               %
%                           DK-5230 Odense M, Denmark.                   %
% Please send your comments and questions to: joal@sdu.dk                %
%                                                                        %
% The code is intended for educational purposes and theoretical details  %
% are discussed in the paper: "A detailed introduction to density-based  %
% topology optimisation of fluid flow problems including implementation  %
% in MATLAB", J. Alexandersen, SMO 2022, doi:                            %
%                                                                        %
% A preprint version of the paper can be downloaded from the author's    %
% website: joealexandersen.com                                           %
% The code is available from GitHub: github.com/sdu-multiphysics/topflow  %
%                                                                        %
% The basic structure of the code is based on the 88-line code for       %
% elastic compliance from: "Efficient topology optimization in MATLAB    %
% using 88 lines of code", E. Andreassen, A. Clausen, M. Schevenels,     %
% B. S. Lazarov and O. Sigmund, SMO 2010, doi:10.1007/s00158-010-0594-7   %
%                                                                        %
% Disclaimer:                                                            %
% The author does not guarantee that the code is free from errors.       %
% Furthermore, the author shall not be liable in any event caused by the  %
% use of the program.                                                    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

E MATLAB code: problems.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%    NAVIER-STOKES TOPOLOGY OPTIMISATION CODE, MAY 2022    %
% COPYRIGHT (c) 2022, J ALEXANDERSEN. BSD 3-CLAUSE LICENSE %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% DEFINE THE PROBLEMS TO BE SOLVED  %%%%
if (probtype == 1) % DOUBLE PIPE PROBLEM
    if ( mod(nely,6) > 0 > 0 )
        error('ERROR: Number of elements in y-dir. must be divisable by 6.');
    elseif ( nely < 30 )
        error('ERROR: Number of elements in y-dir. must be above 30.');
    end
    inletLength = 1/6*nely; inlet1 = 1/6*nely+1; inlet2 = 4/6*nely+1;
    outletLength = 1/6*nely; outlet1 = 1/6*nely+1; outlet2 = 4/6*nely+1;
    nodesInlet  = [inlet1:(inlet1+inletLength)' inlet2:(inlet2+inletLength)'];
    nodesOutlet = (nodx-1)*nody+[outlet1:(outlet1+outletLength)' outlet2:(outlet2+outletLength)'];
    nodesTopBot = [1:nely+1:nodtot nody:nody:nodtot];
    nodesLefRig = [2:nely (nelx)*nody+2:nodtot-1];
    fixedDofsTBx = 2*nodesTopBot-1; fixedDofsTBy = 2*nodesTopBot;
    fixedDofsLRx = 2*setdiff(nodesLefRig,[nodesInlet nodesOutlet])-1;
    fixedDofsLRy = 2*nodesLefRig;
    fixedDofsInX  = 2*nodesInlet-1; fixedDofsInY  = 2*nodesInlet;
    fixedDofsOutY = 2*nodesOutlet; fixedDofsOutP = 2*nodtot+nodesOutlet;
    fixedDofsU = [fixedDofsTBx fixedDofsTBy fixedDofsLRx fixedDofsLRy ...
                  fixedDofsInX fixedDofsInY fixedDofsOutY];
    fixedDofsP = [fixedDofsOutP];
    fixedDofs  = [fixedDofsU fixedDofsP];
    % DIRICHLET VECTORS
    u = @(y) -4*y.^2+4*y; Uinlet = Uin*u([0:inletLength]'/inletLength);
    DIRU=zeros(nodtot*2,1); DIRU(fixedDofsInX) = [Uinlet' Uinlet'];
    DIRP=zeros(nodtot,1); DIR = [DIRU; DIRP];
    % INLET REYNOLDS NUMBER
    Renum = Uin*(inletLength*Ly/nely)*rho/mu;
elseif (probtype == 2) % PIPE BEND PROBLEM
    if ( mod(nelx,10) > 0 || mod(nely,10) > 0 )
        error('ERROR: Number of elements per side must be divisable by 10.');
    end
    inletLength = 2/10*nely; inlet1 = 1/10*nely+1;
    outletLength = 2/10*nelx; outlet1 = 7/10*nelx+1;
    nodesInlet  = [inlet1:(inlet1+inletLength)];
    nodesOutlet = nody*[outlet1:(outlet1+outletLength)];
    nodesTopBot = [1:nely+1:nodtot nody:nody:nodtot];
    nodesLefRig = [2:nely (nodx-1)*nody+2:nodtot-1];
    fixedDofsTBx = 2*nodesTopBot-1; fixedDofsTBy = 2*setdiff(nodesTopBot,nodesOutlet);
    fixedDofsLRx = 2*setdiff(nodesLefRig,nodesInlet)-1; fixedDofsLRy = 2*nodesLefRig;
    fixedDofsInX  = 2*nodesInlet-1; fixedDofsInY  = 2*nodesInlet;
    fixedDofsOutX = 2*nodesOutlet-1; fixedDofsOutP = 2*nodtot+nodesOutlet;
    fixedDofsU = [fixedDofsTBx fixedDofsTBy fixedDofsLRx fixedDofsLRy ...
                  fixedDofsInX fixedDofsInY fixedDofsOutX];
    fixedDofsP = [fixedDofsOutP];
    fixedDofs  = [fixedDofsU fixedDofsP];
    % DIRICHLET VECTORS
    DIRU=zeros(nodtot*2,1); DIRP=zeros(nodtot,1);
    u = @(y) -4*y.^2+4*y; Uinlet = Uin*u([0:inletLength]'/inletLength);
    DIRU(fixedDofsInX) = Uinlet'; DIR = [DIRU; DIRP];
    % INLET REYNOLDS NUMBER
    Renum = Uin*(inletLength*Ly/nely)*rho/mu;
end
```

F MATLAB code: `postproc.m`

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%    NAVIER-STOKES TOPOLOGY OPTIMISATION CODE, MAY 2022    %
% COPYRIGHT (c) 2022, J ALEXANDERSEN. BSD 3-CLAUSE LICENSE %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% POST-PROCESSING
% SETTING UP NODAL COORDINATES FOR STREAMLINES
[X,Y] = meshgrid(0.5:nodx,0.5:nody); sx = 0.5*ones(1,21); sy = linspace(0,nody,21);
U = S(1:(nely+1)*(nelx+1)*2); umag=reshape(sqrt(U(1:2:end).^2+U(2:2:end).^2),nely+1,nelx+1);
% DESIGN FIELD
figure(1); imagesc(xPhys); colorbar; caxis([0 1]); axis equal; axis off;
h = streamline(X,Y,reshape(U(1:2:end),nody,nodx),-reshape(U(2:2:end),nody,nodx),sx,sy); ...
    set(h,'Color','black');
% BRINKMAN PENALTY FACTOR
figure(2); imagesc(reshape(log10(alpha),nely,nelx)); colorbar; caxis([0 log10(alphamax)]); axis ...
    equal; axis off; %colormap turbo;
% VELOCITY MAGNITUDE FIELD
figure(3); imagesc(umag); colorbar; axis equal; axis on; hold on; %colormap turbo;
h = streamline(X,Y,reshape(U(1:2:end),nody,nodx),-reshape(U(2:2:end),nody,nodx),sx,sy); ...
    set(h,'Color','black');
```

```matlab
% PRESSURE FIELD
P = S(2*nodtot+1:3*nodtot);
figure(4); imagesc(reshape(P,nody,nodx)); colorbar; axis equal; axis off; %colormap turbo;
h = streamline(X,Y,reshape(U(1:2:end),nody,nodx),-reshape(U(2:2:end),nody,nodx),sx,sy); ...
    set(h,'Color','black');
% VELOCITY ALONG A LINE
if (probtype == 2)
    uline=flipud(diag(fliplr(umag))); xline=flipud(diag(fliplr(xPhys)));
else
    uline=umag(:,floor((end-1)/2)); xline=xPhys(:,floor(end/2));
end
figure(5);
subplot(3,1,1); plot(uline,'-x'); grid on;
subplot(3,1,2); plot(log10(uline),'-x'); grid on;
subplot(3,1,3); plot(xline,'-x'); grid on; drawnow
```

G MATLAB code: `export.m`

```matlab
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%    NAVIER-STOKES TOPOLOGY OPTIMISATION CODE, MAY 2022    %
% COPYRIGHT (c) 2022, J ALEXANDERSEN. BSD 3-CLAUSE LICENSE %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% DXF
xMod = padarray(padarray(xPhys,[1 1],'replicate'),[1 1],1);
figure('visible','off');
C = contour(flipud(xMod),[0.5 0.5]);
C = unique(C','rows','stable')';
C = C-2.5; C(1,:) = C(1,:)*dx; C(2,:) = C(2,:)*dy;
h = sqrt(dx^2+dy^2);
ind = 1:size(C,2);
ind1 = intersect(find(C(1,:)<Lx+2*dx),find(C(2,:)<Ly+2*dy));
ind2 = intersect(find(C(1,:)>0-dx),find(C(2,:)>0-dy));
ind = intersect(ind1,ind2);
X = []; Y = []; c = 0;
indused = [];
for i = 1:(length(ind))
    x0 = C(1,ind(i)); y0 = C(2,ind(i));
    redind = ind([1:max(1,i-5) i+1:min(length(ind),i+2)]);
    redind = setdiff(redind,indused(max(1,c-10):end));
    R = sqrt( (C(1,redind)-x0).^2 + (C(2,redind)-y0).^2 );
    [minR,idx] = min(R);
    if (minR < 2*h && minR > 0)
        c = c + 1;
        indused(c) = i;
        X(c,:) = [x0 C(1,redind(idx))];
        Y(c,:) = [y0 C(2,redind(idx))];
    end
end
fid=fopen([filename '.dxf'],'w');
fprintf(fid,'0\nSECTION\n2\nENTITIES\n0\n');
for i=1:size(X,1)
      fprintf(fid,'LINE\n8\n0\n');
      fprintf(fid,'10\n%.4f\n20\n%.4f\n30\n%.4f\n',X(i,1),Y(i,1),0);
      fprintf(fid,'11\n%.4f\n21\n%.4f\n31\n%.4f\n',X(i,2),Y(i,2),0);
      fprintf(fid,'0\n');
end
fprintf(fid,'ENDSEC\n0\nEOF\n');
fclose(fid);
```

H MATLAB code: `analyticalElement.m`

```matlab
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%    NAVIER-STOKES TOPOLOGY OPTIMISATION CODE, MAY 2022    %
% COPYRIGHT (c) 2022, J ALEXANDERSEN. BSD 3-CLAUSE LICENSE %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear; close all; clc;
%% Switches
buildJR = 1; exportJR = 1;
buildPhi = 1; exportPhi = 1;
%% Initialisation
syms rho mu alpha dalpha xi eta dx dy;
syms u1 u2 u3 u4 u5 u6 u7 u8 p1 p2 p3 p4;
time = tic;
%% Shape functions and matrices
xv = [-1 1 1 -1]'; yv = [-1 -1 1 1]';
Np = 1/4*(1+xi*xv').*(1+eta*yv');
Nu = sym(zeros(2,8));
Nu(1,1:2:end-1) = Np;
Nu(2,2:2:end) = Np;
%% Nodal coordinates, interpolation and coordinate transforms
xc = dx/2*xv; yc = dy/2*yv;
x = Np*xc; y = Np*yc;
J = [diff(x,xi) diff(y,xi); diff(x,eta) diff(y,eta)];
iJ = inv(J); detJ = det(J);
%% Derivatives of shape functions
dNpdx = iJ(:,1)*diff(Np,xi) + iJ(:,2)*diff(Np,eta);
dNudx = sym(zeros(2,8,2));
for i = 1:2
   dNudx(i,:,:) = transpose(iJ(:,1)*diff(Nu(i,:),xi) + iJ(:,2)*diff(Nu(i,:),eta));
end
%% Nodal DOFs
u = [u1; u2; u3; u4; u5; u6; u7; u8];
p = [p1; p2; p3; p4]; s = [u; p];
ux = Nu*u; px = Np*p;
dudx = [dNudx(:,:,1)*u dNudx(:,:,2)*u];
dpdx = dNpdx*p;
%% Stabilisation parameters
h = sqrt(dx^2 + dy^2);
u0 = subs(ux,[xi,eta],[0,0]);
ue = sqrt(transpose(u0)*u0);
tau1 = h/(2*ue);
tau3 = rho*h^2/(12*mu);
tau4 = rho/alpha;
tau  = (tau1^(-2) + tau3^(-2) + tau4^(-2))^(-1/2);
%% Loop over the tensor weak form to form residual
if (buildJR)
    Ru = sym(zeros(8,1)); Rp = sym(zeros(4,1));
    % Momentum equations
    for g = 1:8
        for i = 1:2
            Ru(g) = Ru(g) + alpha*Nu(i,g)*ux(i); % Brinkman term
            for j = 1:2
                Ru(g) = Ru(g) + mu*dNudx(i,g,j)*( dudx(i,j) + dudx(j,i) ); % Viscous term
                Ru(g) = Ru(g) + rho*Nu(i,g)*ux(j)*dudx(i,j); % Convection term
                Ru(g) = Ru(g) + tau*ux(j)*dNudx(i,g,j)*alpha*ux(i); % SUPG Brinkman term
                for k = 1:2
                    Ru(g) = Ru(g) + tau*ux(j)*dNudx(i,g,j)*rho*ux(k)*dudx(i,k); % SUPG convection term
                end
                Ru(g) = Ru(g) + tau*ux(j)*dNudx(i,g,j)*dpdx(i); % SUPG pressure term
            end
            Ru(g) = Ru(g) - dNudx(i,g,i)*px; % Pressure term
        end
    end
    % Incompressibility equations
    for g = 1:4
        for i = 1:2
            Rp(g) = Rp(g) + Np(1,g)*dudx(i,i); % Divergence term
            Rp(g) = Rp(g) + tau/rho*dNpdx(i,g)*alpha*ux(i); % PSPG Brinkman term
            for j = 1:2
                Rp(g) = Rp(g) + tau*dNpdx(i,g)*ux(j)*dudx(i,j); % PSPG convection term
            end
            Rp(g) = Rp(g) + tau/rho*dNpdx(i,g)*dpdx(i); % PSPG pressure term
        end
    end
    fprintf('Simplifying Ru... \n');
    Ru = simplify(detJ*Ru);
    fprintf('Simplifying Rp... \n');
    Rp = simplify(detJ*Rp);
    %% Integrate analytically
    fprintf('Integrating Ru... \n');
    Ru = int(int(Ru,xi,[-1 1]),eta,[-1 1]);
    fprintf('Integrating Rp... \n');
    Rp = int(int(Rp,xi,[-1 1]),eta,[-1 1]);
    fprintf('Simplifying Ru... \n');
    Ru = simplify(Ru);
    fprintf('Simplifying Rp... \n');
    Rp = simplify(Rp);
    %% Differentiate residual to form Jacobian
    Re = [Ru; Rp];
    Je = sym(zeros(12,12));
    for b = 1:12
        fprintf('Computing dR/ds%2i... \n',b);
        Je(:,b) = diff(Re,s(b));
    end
end
%% Export residual and Jacobian
if (exportJR)
    fprintf('Exporting residual... \n');
    f = matlabFunction(Re,'File','RES','Comments','Version: ...
        0.99','Vars',{dx,dy,mu,rho,alpha,transpose([u1 u2 u3 u4 u5 u6 u7 u8 p1 p2 p3 p4])});
    fprintf('Exporting Jacobian... \n');
    f = matlabFunction(Je(:),'File','JAC','Comments','Version: ...
        0.99','Vars',{dx,dy,mu,rho,alpha,transpose([u1 u2 u3 u4 u5 u6 u7 u8 p1 p2 p3 p4])});
end
%%
time = toc(time);
fprintf('Finished analysis part in %1.3e seconds. \n',time)
%% OPTIMISATION PART
time = tic;
if (buildPhi)
    %% Compute objective functional
    fprintf('Computing phi... \n');
    phi = 1/2*alpha*transpose(ux)*ux;
    for i = 1:2
        for j = 1:2
            phi = phi + 1/2*mu*dudx(i,j)*( dudx(i,j) + dudx(j,i) );
        end
    end
    %% Intregrate and simplify objective functional
    fprintf('Integrating phi... \n');
    phi = int(int(detJ*phi,xi,[-1 1]),eta,[-1 1]);
    phi = simplify(phi);
    %% Compute the partial derivative wrt. design field
    fprintf('Computing dphi/dgamma... \n');
    dphidg = simplify( diff(phi,alpha)*dalpha );
    %% Compute the partial derivative wrt. state field
    dphids = sym(zeros(12,1));
    for a = 1:12
        fprintf('Computing dphi/ds%2i... \n',a);
        dphids(a) = simplify(diff(phi,s(a)));
    end
end
%% Compute partial derivative of residual wrt. design field
if (buildJR)
    fprintf('Computing dr/dgamma... \n');
    drdg = simplify( diff(Re,alpha)*dalpha );
end
%% Export optimisation functions
if (exportPhi)
    fprintf('Exporting phi... \n');
    f = matlabFunction(phi,'File','PHI','Comments','Version: ...
        0.9','Vars',{dx,dy,mu,alpha,transpose([u1 u2 u3 u4 u5 u6 u7 u8 p1 p2 p3 p4])});
    fprintf('Exporting dphi/dg... \n');
    f = matlabFunction(dphidg,'File','dPHIdg','Comments','Version: ...
        0.9','Vars',{dx,dy,mu,alpha,dalpha,transpose([u1 u2 u3 u4 u5 u6 u7 u8 p1 p2 p3 p4])});
    fprintf('Exporting dphi/ds... \n');
    f = matlabFunction(dphids(1:8),'File','dPHIds','Comments','Version: ...
        0.9','Vars',{dx,dy,mu,alpha,transpose([u1 u2 u3 u4 u5 u6 u7 u8 p1 p2 p3 p4])});
end
if (exportJR)
    fprintf('Exporting dr/ds... \n');
    f = matlabFunction(drdg,'File','dRESdg','Comments','Version: ...
        0.9','Vars',{dx,dy,mu,rho,alpha,dalpha,transpose([u1 u2 u3 u4 u5 u6 u7 u8 p1 p2 p3 p4])});
```

```
end
%%
```