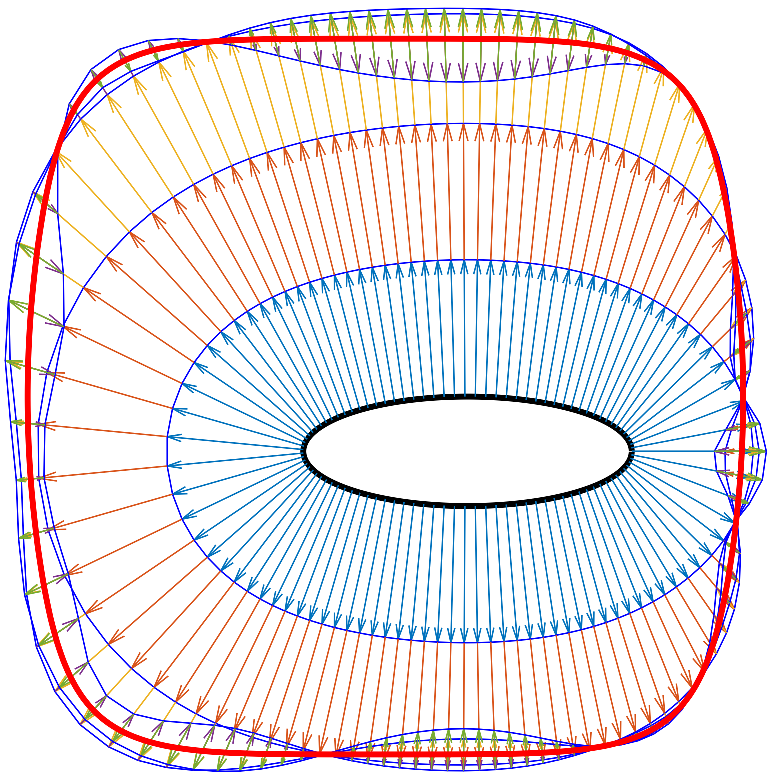
Shape Optimisation with PDE Constraints

Mike Fuller (fuller@maths.ox.ac.uk)
Supervised by Dr Alberto Paganini (paganini@maths.ox.ac.uk)

30th July 2018



Abstract

Shape optimisation is typically concerned with finding the shape which is optimal in the sense that it minimises a certain cost functional while satisfying given constraints. Often we find that the functional being solved depends on the solution of a PDE constraint.

In this report we briefly discuss unconstrained optimisation methods before introducing the notion of a shape derivative, which will help us to create shape optimisation methods. We then explore a test case that, despite its relative simplicity, exhibits an unexpected behaviour: solving this problem via Newton's method with a *truncated* second shape derivative performs better than using full second order information. We will perform numerical and theoretical investigations using *Chebfun* to try to shed light on this unexpected behaviour.

Contents

1 Unconstrained Optimisation

We first recap on some iterative methods for solving unconstrained optimisation problems.

1.1 Setting the Scene

Let $f : \mathbb{R}^n \to \mathbb{R}$ be a function. Suppose we want to find a value x^* such that f is minimal, i.e

$$x^* \in \arg\min_x f(x).$$

We can find such a value by means of an iterative algorithm. That is, given an initial guess $x^{(0)}$ for x^*, we construct a *minimising sequence* $(x^{(k)})$ of points such that

$$f(x^{(k)}) \longrightarrow f(x^*) \quad \text{as } k \to \infty.$$

Often we only require an approximate solution, so we can reduce computational effort by choosing a tolerance $\varepsilon > 0$ such that when $f(x^{(k)}) - f(x^*) \leqslant \varepsilon$, we terminate the algorithm.

1.2 Descent Methods

The minimising sequences we are going to discuss in this section are all of the form

$$x^{(k+1)} = x^{(k)} + t^{(k)}\Delta x^{(k)},$$

where

- $\Delta x^{(k)}$ represents a *search direction* - which direction we travel in to approach x^*,
- $t^{(k)} \geqslant 0$ represents a *step length* - how far we travel along the vector $\Delta x^{(k)}$.

Recall that our aim is to minimise f, so we would like the sequence to have the property

$$f(x^{(k+1)}) < f(x^{(k)}) \tag{1}$$

for all k, unless $x^{(k)}$ is optimal. If f is differentiable, we can Taylor expand up to first-order and write

$$f(x^{(k+1)}) = f(x^{(k)} + t^{(k)}\Delta x^{(k)}) \approx f(x^{(k)}) + t^{(k)}\mathrm{d}f(x^{(k)}, \Delta x^{(k)}),$$

where $\mathrm{d}f$ denotes the *directional derivative*

$$\mathrm{d}f(x, v) := \lim_{t\to 0} \frac{f(x+tv) - f(x)}{t}.$$

Hence, if $x^{(k)}$ is not optimal (so that $t^{(k)} \neq 0$), we can ensure (1) holds up to first-order if

$$\mathrm{d}f(x^{(k)}, \Delta x^{(k)}) < 0. \tag{2}$$

A search direction $\Delta x^{(k)}$ such that (2) holds is called a *descent direction.*

A generic descent method then goes as follows:

```
1 Given starting point $x$
2 repeat
3     (1) Determine descent direction, $\Delta x$
4     (2) Choose step size, $t > 0$
5     (3) Set $x \leftarrow x + t\Delta x$
6 until stopping criterion satisfied
```

At the moment this is rather vague. We have a few questions to answer:

- How do we choose the step size?
- What is the descent direction?
- When do we stop the algorithm?

1.3 Line Search

To answer the first question, we discuss two methods of deciding the step size, called *exact line search (ELS)* and *backtracking line search (BLS)*. Here we assume that Δx is a descent direction.

In ELS, we choose t such that f is minimised along the ray $\{x + t\Delta x : t \geqslant 0\}$, i.e

$$t = \arg\min_{s \geqslant 0} f(x + s\Delta x).$$

It may be difficult to find t exactly, so it is generally cost effective to choose a t value such that f is only approximately minimised along the ray. For this reason we may choose to apply BLS, which generates the step size by the following algorithm:

Algorithm 1: Backtracking Line Search

```
1 Given descent direction $\Delta x$, $\alpha \in (0, 0.5), \beta \in (0, 1)$
2 $t := 1$
3 while $f(x + t\Delta x) > f(x) + \alpha t \, \mathrm{d}f(x, \Delta x)$ do
4     Set $t \leftarrow \beta t$
5 end
```

The resulting value of t is chosen as the step size. The line search is called "backtracking" as it starts with unit step size, then reduces it by the factor β until the stopping criterion

$$f(x + t\Delta x) > f(x) + \alpha t \, \mathrm{d}f(x, \Delta x)$$

holds. Since Δx is a descent direction, we have $\mathrm{d}f(x, \Delta x) < 0$, so

$$f(x + t\Delta x) \approx f(x) + t \, \mathrm{d}f(x, \Delta x) < f(x) + \alpha t \, \mathrm{d}f(x, \Delta x)$$

for small enough t, demonstrating that BLS eventually terminates. The choice of parameters α, β has a noticeable but not dramatic effect on convergence (see [1]); ELS can sometimes improve convergence rate but not dramatically. It is generally not worth the computational effort of using ELS.

1.4 Steepest Descent

We can find a descent direction by analysing the Taylor expansion of $f(x + \Delta x)$ about x up to first-order. The approximation is

$$f(x + \Delta x) \approx f(x) + \mathrm{d}f(x, \Delta x),$$

so we find the direction of *steepest descent* when we minimise the directional derivative. We define a *normalised steepest descent direction* Δx_{nsd} as a descent direction of unit length that minimises $\mathrm{d}f(x, \Delta x)$, i.e

$$\Delta x_{\mathrm{nsd}} \in \arg\min_{||\Delta x|| = 1} \mathrm{d}f(x, \Delta x)$$

for some norm $|| \cdot ||$ on $\mathbb{R}^n$.

To piece together the *Steepest Descent* method, it will be useful to introduce the concept of inner products and gradients, particularly when it comes to defining stopping criteria. Let V be a real vector space. An *inner product* $(\cdot,\cdot) : V^2 \to \mathbb{R}$ is a function that satisfies the following properties:

- *Linearity:* $(\alpha u + \beta v, w) = \alpha(u, w) + \beta(v, w) \quad \forall u, v, w \in V,\ \alpha, \beta \in \mathbb{R}$
- *Symmetry:* $(u, v) = (v, u) \quad \forall u, v \in V$
- *Positive definitenesss:* $(u, u) \geqslant 0 \quad \forall u \in V, \quad (u, u) = 0 \iff u = 0.$

It follows from the first two properties that $(\cdot,\cdot)$ is *bilinear.*

We call $\nabla f(x)$ the *gradient* of f at x with respect to an inner product $(\cdot,\cdot)$ if

$$(\nabla f(x), v) = \mathrm{d}f(x, v) \quad \forall v.$$

We occasionally write $\nabla_{(\cdot,\cdot)} f(x)$ to make it clear what the associated inner product is.

Lemma 1.1: If a norm $||\cdot||$ is induced by an inner product $(\cdot,\cdot)$, i.e $||v|| = \sqrt{(v, v)}$, then

$$\Delta x_{\mathrm{nsd}} = -\frac{\nabla f(x)}{||\nabla f(x)||}. \tag{3}$$

Proof: We use the method of Lagrange multipliers. Let

$$L(\Delta x, \lambda) := \mathrm{d}f(x, \Delta x) + \frac{\lambda}{2}(||\Delta x||^2 - 1).$$

Then by definition of Δx_{nsd},

$$\begin{aligned} & \partial_{\Delta x} L(\Delta x_{\mathrm{nsd}}, v) = \mathrm{d}f(x, v) + \lambda(\Delta x_{\mathrm{nsd}}, v) = 0 \quad \forall v \\ \iff \quad & (\lambda \Delta x_{\mathrm{nsd}}, v) = -\mathrm{d}f(x, v) \quad \forall v \\ \iff \quad & \lambda \Delta x_{\mathrm{nsd}} = -\nabla f(x). \end{aligned}$$

Taking norms of both sides, and using the fact that $||\Delta x_{\mathrm{nsd}}|| = 1$, we obtain $|\lambda| = ||\nabla f(x)||$. The result follows, using the fact that Δx_{nsd} is a steepest descent direction.

□

Geometrically speaking, we see that Δx_{nsd} is a step in the unit ball of $||\cdot||$ which extends farthest in the direction of $-\nabla f(x)$. It is more convenient in practice to use an unnormalised descent direction

$$\Delta x_{\mathrm{sd}} := ||\nabla f(x)||_* \Delta x_{\mathrm{nsd}},$$

where $||\cdot||_*$ denotes the *dual norm*

$$||v||_* := \sup_{||x|| \leqslant 1} v^T x.$$

We find that Δx_{sd} is simply the unnormalised version of (3), that is

$$\Delta x_{\mathrm{sd}} = -\nabla_{(\cdot,\cdot)} f(x),$$

and we arrive at the *Steepest Descent* method, which uses Δx_{sd} as a descent direction:

Algorithm 2: Steepest Descent

```
1 Given starting point $x$
2 repeat
3     (1) Set $\Delta x \leftarrow \Delta x_{\mathrm{sd}}$
4     (2) Choose $t$ via ELS/BLS
5     (3) Set $x \leftarrow x + t\Delta x$
6 until stopping criterion satisfied
```

To define a stopping criterion, recall that when x^* is optimal we have

$$\mathrm{d}f(x^*, v) = 0 \quad \forall v,$$

so for a sufficiently smooth f, we expect the gradient to be small for $x \approx x^*$. For this reason, we can define a stopping criterion of the form

$$||\nabla f(x)|| \leqslant \varepsilon$$

for some chosen small $\varepsilon > 0$. Most practical uses of Steepest Descent check the stopping criterion after the first step.

1.5 Newton's Method

Which inner product do we use to define Δx_{sd}? An interesting choice, which forms the basis of *Newton's Method*, is to consider the inner product induced by the second derivative

$$\mathrm{d}^2 f(x, u, v) := \lim_{t \to 0} \frac{\mathrm{d}f(x + tv, u) - \mathrm{d}f(x, u)}{t}.$$

If it exists, we can extend our Taylor approximation up to second-order to get

$$f(x + \Delta x) \approx f(x) + \mathrm{d}f(x, \Delta x) + \frac{1}{2}\mathrm{d}^2 f(x, \Delta x, \Delta x) =: g(\Delta x).$$

If $\mathrm{d}^2 f(x, \cdot, \cdot)$ is positive definite then g is *strictly convex*[*], thus has a unique minimiser Δx_{nt}, called the *Newton step*. Since the second derivative exists, $\mathrm{d}^2 f(x, \cdot, \cdot)$ is also symmetric and linear, hence an inner product. By minimality, the Newton step must satisfy

$$\begin{aligned} & \mathrm{d}g(\Delta x_{\text{nt}}, v) = 0 \quad \forall v \\ \iff \quad & \mathrm{d}f(x, v) + \mathrm{d}^2(x, \Delta x_{\text{nt}}, v) = 0 \quad \forall v \\ \iff \quad & \mathrm{d}^2(x, \Delta x_{\text{nt}}, v) = -\mathrm{d}f(x, v) \quad \forall v \end{aligned}$$

so Δx_{nt} is the negative gradient with respect to the inner product induced by $\mathrm{d}^2 f$, that is,

$$\Delta x_{\text{nt}} = -\nabla_{\mathrm{d}^2 f(x, \cdot, \cdot)} f(x).$$

If f is quadratic, then $f(\Delta x_{\text{nt}}) = g(\Delta x)$ exactly, so $x + \Delta x_{\text{nt}}$ is the *exact* minimiser of f. Hence it should also be a very good estimate for x^* if f is approximately quadratic. Since f is assumed to be twice continuously differentiable, the quadratic model of f is very accurate for x near x^*, so $x + \Delta x_{\text{nt}}$ is a very good estimate for x^*.

Before we introduce *Newton's method*, we will construct a stopping criterion using the directional derivative, as demonstrated in the following lemma:

Lemma 1.2: $\frac{1}{2}\mathrm{d}f(x, \Delta x_{\text{nt}})$ estimates the error $f(x) - f(x^*)$, based on the quadratic approximation of f.

Proof: The error based on the quadratic approximation is

$$\begin{aligned} f(x) - \inf_y g(y) = f(x) - g(\Delta x_{\text{nt}}) &= f(x) - \left(f(x) + \underbrace{\mathrm{d}f(x, \Delta x_{\text{nt}})}_{=0} + \frac{1}{2}\mathrm{d}^2 f(x, \Delta x_{\text{nt}}, \Delta x_{\text{nt}}) \right) \\ &= f(x) - \left(f(x) - \frac{1}{2}\mathrm{d}f(x, \Delta x_{\text{nt}}) \right) \\ &= \frac{1}{2}\mathrm{d}f(x, \Delta x_{\text{nt}}). \end{aligned}$$

□

[*]A function g is said to be *strictly convex* if the line segment connecting any two distinct points on the surface of g lies strictly above g, except at the endpoints.

We use this as a stopping criterion for Newton's Method, which goes as follows:

```
Algorithm 3: Newton's Method
1 Given starting point $x$, tolerance $\varepsilon > 0$
2 repeat
3     (1) Set $\Delta x \leftarrow \Delta x_{\mathrm{nt}}$
4     (2) Stop if $\mathrm{d}f(x, \Delta x)/2 \leqslant \varepsilon$
5     (3) Choose $t$ via BLS
6     (4) Set $x \leftarrow x + t\Delta x$
```

Observe that this is more or less a general descent method, with the difference that the stopping criterion is checked after computing Δx_{nt}, rather than after updating the value of x.

Newton's Method has many advantages over Steepest Descent. Most importantly it has *quadratic convergence* near x^* [1, page 496] - roughly speaking, this means that the number of correct digits doubles after each iteration. As a result, Newton's Method also scales with the size of the problem: it will perform just as well for a problem in $\mathbb{R}^{10000}$ as it would for problems in $\mathbb{R}^{10}$ say, with only a moderate increase in the number of iterations required.

A pitfall of Newton's Method is the cost of computing Δx_{nt}, which requires solving a system of linear equations. *Quasi-Newton* methods require less cost to form the search direction, sharing some advantages of Newton's Method such as rapid convergence near x^*, but we will not discuss such methods here.

2 Introduction to Shape Optimisation

In the previous section, we had a function $f : \mathbb{R}^n \to \mathbb{R}$ and we used its derivatives to find the points x^* which minimised $f(x)$. What if we were instead given a cost functional $\mathcal{J}[\Omega]$, with the aim of minimising $\mathcal{J}$ over bounded domains (shapes) Ω? We would need a notion of *shape differentiation.*

In this section we define the *shape derivative.* We will ultimately use such derivatives to find a domain Ω^* in a collection of admissible shapes $\mathcal{U}_{\text{ad}}$ which minimises a given cost functional $\mathcal{J}$. For simplicity, we restrict ourselves to functionals of the form

$$\mathcal{J}[\Omega] := \int_\Omega f \,\mathrm{d}x.$$

In the style of Section 1.1, we write

$$\Omega^* \in \operatorname*{arg\,min}_{\Omega \in \mathcal{U}_{\text{ad}}} \mathcal{J}[\Omega].$$

2.1 The Shape Derivative, $\mathrm{d}\mathcal{J}(\Omega, \mathcal{V})$

Let $T : \mathbb{R}^n \to \mathbb{R}^n$ be a sufficiently smooth vector field. Then

$$\mathcal{J}[T(\Omega)] = \int_{T(\Omega)} f \,\mathrm{d}x = \int_\Omega (f \circ T)|\det \mathbf{D}T| \,\mathrm{d}x,$$

where $\mathbf{D}T$ denotes the *Jacobian* of T. Note that $T(\Omega) \neq \Omega$ in general, and similarly $\mathcal{J}[T(\Omega)] \neq \mathcal{J}[\Omega]$.

Furthermore let $\mathcal{V} : \mathbb{R}^n \to \mathbb{R}^n$ be a vector field, and define a duality pairing

$$\Big\langle \frac{\mathrm{d}}{\mathrm{d}T} \mathcal{J}[T(\Omega)], \mathcal{V} \Big\rangle := \lim_{t \to 0} \frac{\mathcal{J}[(T + t\mathcal{V})(\Omega)] - \mathcal{J}[T(\Omega)]}{t}.$$

We then define the *Eulerian (shape) derivative* of $\mathcal{J}$ in the direction $\mathcal{V}$ as

$$\mathrm{d}\mathcal{J}(\Omega, \mathcal{V}) := \Big\langle \frac{\mathrm{d}}{\mathrm{d}T} \mathcal{J}[T(\Omega)], \mathcal{V} \Big\rangle \Big|_{T=I},$$

where I is the identity map $I(\Omega) = \Omega$. Therefore, we can write

$$\mathrm{d}\mathcal{J}(\Omega, \mathcal{V}) = \Big\langle \frac{\mathrm{d}}{\mathrm{d}T} \Big(\int_\Omega (f \circ T)|\det \mathbf{D}T| \,\mathrm{d}x \Big), \mathcal{V} \Big\rangle \Big|_{T=I} = \int_\Omega \Big\langle \frac{\mathrm{d}}{\mathrm{d}T} ((f \circ T) \det \mathbf{D}T), \mathcal{V} \Big\rangle \Big|_{T=I} \mathrm{d}x.$$

Ideally we want $\mathrm{d}\mathcal{J}$ to exist for all $\mathcal{V}$, so we say that $\mathcal{J}$ is *shape differentiable* at Ω if the map

$$\mathcal{V} \mapsto \mathrm{d}\mathcal{J}(\Omega, \mathcal{V})$$

is linear and bounded on $C^1(\mathbb{R}^n, \mathbb{R}^n)$, the set of continuously differentiable maps from $\mathbb{R}^n$ to $\mathbb{R}^n$.

2.2 Computing $\mathrm{d}\mathcal{J}(\Omega, \mathcal{V})$

Using the above, we prove two formulae for the shape derivative $\mathrm{d}\mathcal{J}$, which will help us to compute it in practice.

Theorem 2.1: Let $\nabla \cdot \mathcal{V}$ denote the *divergence* of $\mathcal{V}$. Then

$$\mathrm{d}\mathcal{J}(\Omega, \mathcal{V}) = \int_\Omega \nabla f \cdot \mathcal{V} + f \nabla \cdot \mathcal{V} \,\mathrm{d}x.$$

Proof: By the product rule,

$$\mathrm{d}\mathcal{J}(\Omega, \mathcal{V}) = \int_\Omega \Big\{ \Big\langle \frac{\mathrm{d}}{\mathrm{d}T} (f \circ T), \mathcal{V} \Big\rangle \det \mathbf{D}T + \Big\langle \frac{\mathrm{d}}{\mathrm{d}T} \det \mathbf{D}T, \mathcal{V} \Big\rangle (f \circ T) \Big\} \Big|_{T=I} \mathrm{d}x.$$

Considering the first term, we have

$$\left\langle \frac{\mathrm{d}}{\mathrm{d}T}(f \circ T), \mathcal{V} \right\rangle = (\nabla f \circ T) \cdot \mathcal{V},$$

hence

$$\left\langle \frac{\mathrm{d}}{\mathrm{d}T}(f \circ T), \mathcal{V} \right\rangle \det \mathbf{D}T \Big|_{T=I} = \nabla f \cdot \mathcal{V}.$$

Considering the second term and using Jacobi's formula [2], we have

$$\begin{aligned}\left\langle \frac{\mathrm{d}}{\mathrm{d}T} \det \mathbf{D}T, \mathcal{V} \right\rangle &= \lim_{t\to 0} \frac{\det \mathbf{D}(T + t\mathcal{V}) - \det \mathbf{D}T}{t} \\ &= \lim_{t\to 0} \frac{\det(\mathbf{D}T + t\mathbf{D}\mathcal{V}) - \det \mathbf{D}T}{t} \\ &= \lim_{t\to 0} \frac{(\det \mathbf{D}T + t\,\mathrm{Tr}(\mathrm{adj}(\mathbf{D}T)\mathbf{D}\mathcal{V}) + O(t^2)) - \det \mathbf{D}T}{t} = \mathrm{Tr}(\mathrm{adj}(\mathbf{D}T)\mathbf{D}\mathcal{V}),\end{aligned}$$

where $\mathrm{Tr}(A), \mathrm{adj}(A)$ denote the *trace* and *adjugate*[†] of A respectively. Hence

$$\left\langle \frac{\mathrm{d}}{\mathrm{d}T} \det \mathbf{D}T, \mathcal{V} \right\rangle (f \circ T) \Big|_{T=I} = \mathrm{Tr}(\mathrm{adj}(\mathrm{Id})\mathbf{D}\mathcal{V}) f = \mathrm{Tr}(\mathbf{D}\mathcal{V}) f = f \nabla \cdot \mathcal{V}$$

since $\mathrm{adj}(\mathrm{Id}) = \mathrm{Id}$ is the identity matrix, and the result follows. $\square$

Theorem 2.2: The shape derivative is equal to the surface integral

$$\mathrm{d}\mathcal{J}(\Omega, \mathcal{V}) = \int_{\partial\Omega} f(\mathcal{V} \cdot n)\, \mathrm{d}S,$$

where n is the outward pointing unit normal of the boundary $\partial\Omega$.

Proof: Using the Divergence Theorem,

$$\begin{aligned}\mathrm{d}\mathcal{J}(\Omega, \mathcal{V}) &= \int_{\Omega} \nabla f \cdot \mathcal{V} + f \nabla \cdot \mathcal{V}\, \mathrm{d}x \\ &= \int_{\Omega} \nabla \cdot (f\mathcal{V})\, \mathrm{d}x \\ &= \int_{\partial\Omega} (f\mathcal{V}) \cdot n\, \mathrm{d}S\end{aligned}$$

where the second equality follows from vector calculus. The result follows. $\square$

2.3 Descent Methods for Shapes

To construct a descent method for shapes, we need a minimising sequence of shapes, say

$$\Omega^{(k+1)} = T^{(k)}(\Omega^{(k)})$$

for a sequence of smooth vector fields $T^{(k)}$. But how could we define the next shape in the sequence in a similar manner to Section 1.2? We could define the shape by its boundary, then *add* shape boundaries together by taking their parametrisations and summing component-wise. It then makes sense to write

$$\partial\Omega^{(k+1)} = \partial\Omega^{(k)} + t^{(k)} \Delta\partial\Omega^{(k)},$$

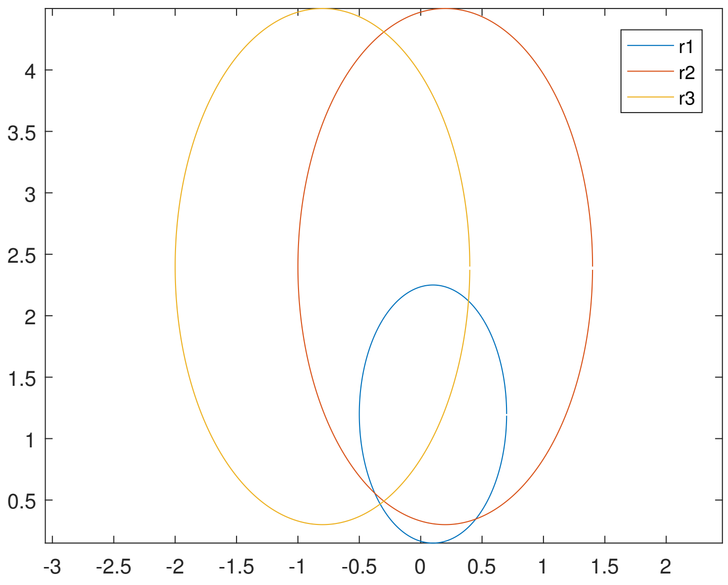
where

- $\Delta\partial\Omega^{(k)}$ represents a *(boundary) search direction*,
- $t^{(k)} \geqslant 0$ represents a *step length.*

[†]For a matrix A, the *adjugate* of A is the transpose of the *cofactor matrix* C, where $c_{ij} = (-1)^{i+j} \det A_{ij}$, and A_{ij} is the matrix A with row i and column j removed. For an invertible matrix, this is simply $\mathrm{adj} A = (\det A)A^{-1}$.

In MATLAB, we can illustrate this with the help of the *Chebfun* package. For a 2-dimensional problem, define the boundary $\partial\Omega$ as being in the complex plane, and parametrise as follows:

```
r = @(t) 0.1+1.2i + 0.6*cos(t) + 1.05*1i*sin(t);
t = linspace(0,2*pi);
r1 = chebfun(@(t) r(t), [0, 2*pi], 'trig'); %Parametrised boundary
r2 = 2*r1; %Scaled boundary
r3 = r2-[1+0i]; %Scaled and shifted boundary
```



Plotting `r1`, `r2` and `r3` gives the (intuitive) figure above.

2.3.1 Finding a Search Direction, $\Delta\partial\Omega$

We can find a suitable shape search direction using the gradient definition in Section 1.4. If the shape derivative is linear with respect to $\mathcal{V}$, there is a unique element of $\nabla\mathcal{J} \in L^2(\partial\Omega)$ [3], called the *shape gradient* of $\mathcal{J}$. If $(\cdot,\cdot)_{\partial\Omega}$ is the L^2-inner product on $\partial\Omega$, then

$$\mathrm{d}\mathcal{J}(\Omega,\mathcal{V}) = (\nabla\mathcal{J},\mathcal{V})_{\partial\Omega} := \int_{\partial\Omega} \nabla\mathcal{J}\cdot\mathcal{V}\,\mathrm{d}S.$$

Using Theorem 2.2, we deduce that

$$\nabla\mathcal{J}\big|_{\partial\Omega} = fn.$$

So if we choose our boundary search direction to be the negative gradient

$$\Delta\partial\Omega := -\nabla\mathcal{J}\big|_{\partial\Omega} = -fn,$$

then the shape derivative with search direction $\Delta\partial\Omega$ is

$$\mathrm{d}\mathcal{J}(\Omega,\Delta\partial\Omega) = -\int_{\partial\Omega} f(fn\cdot n)\,\mathrm{d}S = -\int_{\partial\Omega} f^2\,\mathrm{d}S < 0 \tag{4}$$

when $f \neq 0$ on $\partial\Omega$. In line with Section 1.2, we will call any search direction with property (4) a *(boundary) descent direction.*

From (4), we deduce that the shape derivative vanishes when $f = 0$ on $\partial\Omega$. This intuitively makes sense; to minimise $\mathcal{J}$, we want to integrate over the shape $\Omega^* = \{x : f(x) \leqslant 0\}$, with the boundary being the 0-level set of f. Integrating over shapes outside of Ω^* can only increase the value of $\mathcal{J}$.

Example 2.3: Take $f : \mathbb{R}^2 \to \mathbb{R}$ to be the function defined by

$$f(x) = x_1^2 + x_2^2 - 1.$$

We know that $\mathcal{J}(\Omega)$ is minimised when integrated over the domain $\Omega = \{x : x_1^2 + x_2^2 \leqslant 1\}$.

We implement a basic shape optimisation method in MATLAB, with initial guess

$$\Omega^{(0)} := \left\{ x : \frac{(x_1 - 0.2)^2}{0.6^2} + \frac{(x_2 - 0.3)^2}{0.2^2} \leqslant 1 \right\},$$

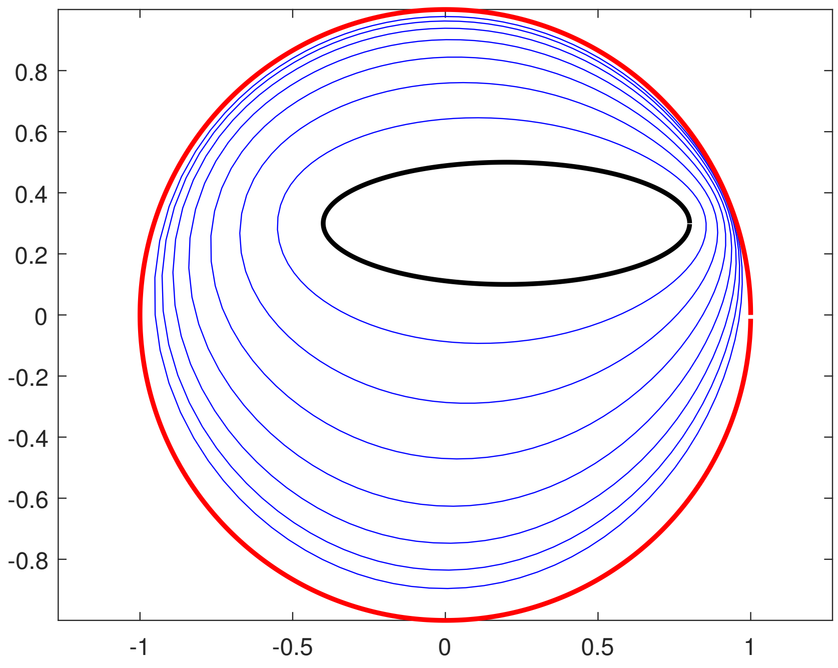
shape search direction $\Delta\partial\Omega$ and fixed step size $t = 0.2$.

```
g_ = @(t) [0.2+0.3i] + 0.6*cos(t) + 0.2*1i*sin(t);
t = linspace(0,2*pi);
g = chebfun(@(t) g_(t), [0, 2*pi], 'trig'); %Initial boundary
init = plot(g, 'k') %Plot of initial boundary
set(init,'LineWidth',2);
dg = diff(g); n_ = -1i*dg;
n = n_./abs(n_); %Unit normal to boundary

f_ = @(x,y) x.^2 + y.^2 - 1;
f = chebfun2(@(x,y) f_(x,y), [-5 5 -5 5]); %Integrand, f

for k=1:7
    hold on
    fn = chebfun(@(t) n(t).*f(real(g(t)), imag(g(t))), [0 2*pi]);
    g = g - 0.2*fn; %Updated boundary
    plot(real(g(t)), imag(g(t)), 'b')
end

exact = fimplicit(f_, 'r');
set(exact,'LineWidth',2);
axis equal
```



The above figure shows the first 7 iterations $\partial\Omega^{(k)}$ in blue, together with the initial shape boundary $\partial\Omega^{(0)}$ in black and the optimum shape boundary $\partial\Omega^*$ in red.

2.3.2 Choosing the Initial Boundary, $\partial\Omega^{(0)}$

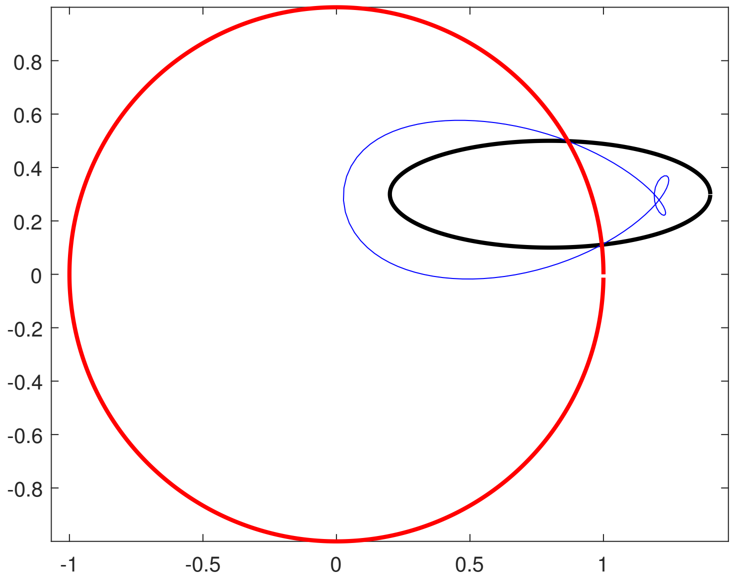
Recall that the optimum shape boundary is the set of points such that f vanishes. If the initial shape boundary intersects the optimum boundary, then at the intersection points we have

$$\Delta\partial\Omega = -fn = 0,$$

so each boundary iteration will always include those points as $\partial\Omega^{(k+1)} = \partial\Omega^{(k)}$ there. This can be problematic - suppose we instead took the initial boundary in Example 2.1 to be

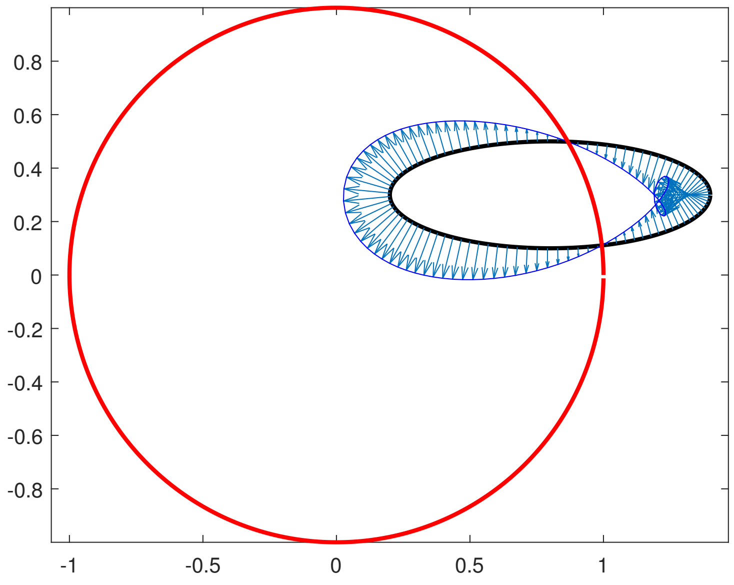
$$\Omega^{(0)} := \left\{ x : \frac{(x_1 - 0.8)^2}{0.6^2} + \frac{(x_2 - 0.3)^2}{0.2^2} \leqslant 1 \right\},$$

which intersects the optimum boundary at two points. The first iteration would look like this:



A problem arises in that the curve is no longer simple. Why does this happen? We can visualise the direction that points of the initial boundary are being transformed to using `quiver`:

```
quiver(real(g(t)), imag(g(t)), −0.2*real(fn(t)), −0.2*imag(fn(t)),0)
```



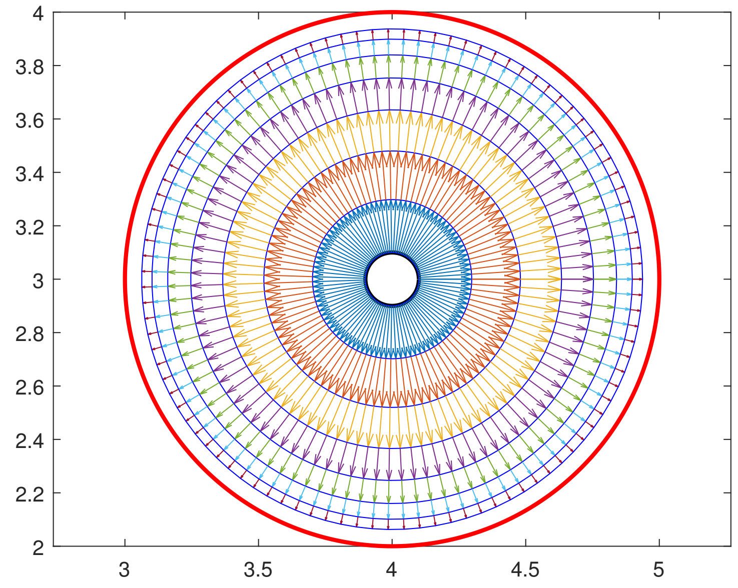
In this example, points further away from the optimum boundary have a greater value of f, so they are being transformed towards the red circle with a greater scale factor than those points closer to the 0-level set of f. For optimisation problems with simple boundary solutions, if the initial curve intersects/lies outside of the optimum shape, we may run into problems.

At the cost of some computational time, we can avoid this boundary-crossing behaviour by defining $\partial\Omega^{(0)}$ to be a small ball around the point at which f is minimised, which we can find with `min2`:

```
[Y,X] = min2(f); x1 = X(1); x2 = X(2);
r = sqrt(x1.^2+x2.^2); theta = atan2(x2,x1);
g_ = @(t) r*(cos(theta)+1i*sin(theta)) + 0.1*cos(t) + 0.1*1i*sin(t);
```

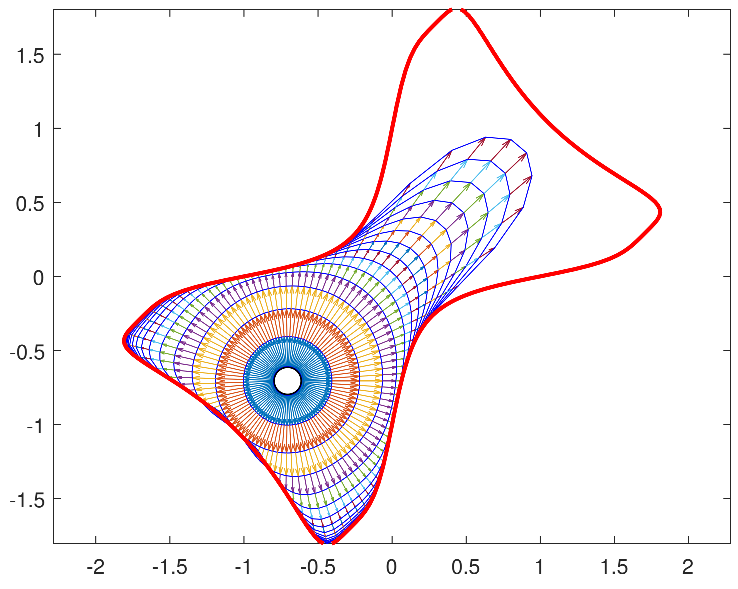
The below figure plots the first 7 iterations for the new initial boundary with a shifted integrand

$$f(x) = (x_1 - 4)^2 + (x_2 - 3)^2 - 1$$



and the next figure with 14 iterations and a step size $t = 0.1$ on a slightly more complicated integrand

$$f(x) = (x_1 - x_2)^2 + (2x_1x_2 - 1)^4 - 5/2.$$



Note that the method struggles with the latter shape due to its non-convexity: this is not a big issue as in practical applications, for example the shape optimisation of microlenses, we will restrict the set of admissible shapes $\mathcal{U}_{\text{ad}}$ to

$$\mathcal{U}_{\text{ad}} = \{\Omega : \Omega \text{ convex}\}.$$

2.4 Line Search for Shapes

In the previous example we fixed the step size t, but we could try to implement a line search-like method to reduce the number of iterations. In an ideal world, we could use an ELS-like method: choosing t such that $\mathcal{J}$ is minimised along the ray $\{\Omega + t\Delta\Omega : t \geqslant 0\}$, i.e

$$t = \arg\min_{s \geqslant 0} \mathcal{J}(\Omega + s\Delta\Omega).$$

Here we write $\Omega + t\Delta\Omega$ to mean the shape with boundary $\partial\Omega + t\Delta\partial\Omega$.

Recall that it was difficult to find t exactly in Section 1.3, so to find t practically we will need to consider a BLS-like algorithm here. We will then observe the improvements (if any) on our shape optimisation method. *BLS for Shapes* goes as follows:

Algorithm 4: Backtracking Line Search for Shapes

```
1 Given shape descent direction $\Delta\partial\Omega$, $\alpha \in (0, 0.5), \beta \in (0,1)$
2 $t := 1$
3 while $\mathcal{J}(\Omega + t\Delta\Omega) > \mathcal{J}(\Omega) + \alpha t \, \mathrm{d}\mathcal{J}(\Omega, \Delta\partial\Omega)$ do
4     Set $t \leftarrow \beta t$
5 end
```

We can compute the functionals $\mathcal{J}(\Omega)$, $\mathcal{J}(\Omega + t\Delta\Omega)$ and shape derivatives $\mathrm{d}\mathcal{J}(\Omega, \mathcal{V})$ with the *Chebfun* functions `integral2` and `integral`. Conveniently, $\mathcal{J}$ is calculated using the shape boundary. We can then replace the `for` lop in the MATLAB code from Example 2.1 with a `while` loop involving BLS for Shapes, with $\alpha = 0.1$ and $\beta = 0.7$ say:

```
J = @(f,g) integral2(f,g); %Cost functional, J(\Omega)
dJ = @(f,g,V) integral(f.*dot(V,n),g); %Shape derivative, dJ(\Omega,V)

tol = 0.05;
iteration = 0;

while abs(dJ(f,g,fn)) ≥ tol

    hold on
    iteration  = iteration + 1
    fn = chebfun(@(t) n(t).*f(real(g(t)), imag(g(t))), [0 2*pi]);

    % BLS for shapes; step size algorithm
        alpha = 0.1; beta = 0.7; s = 1;
        while J(f,g-s*fn) > J(f,g) + alpha*s*dJ(f,g,-fn);
              s = beta*s;
        end

    q = quiver(real(g(t)), imag(g(t)), -s*real(fn(t)), -s*imag(fn(t)),0);
    set(q,'MaxHeadSize',0.075,'AutoScaleFactor',1);
    g = g - s*fn; %Updated boundary
    plot(real(g(t)), imag(g(t)), 'b')

end
```

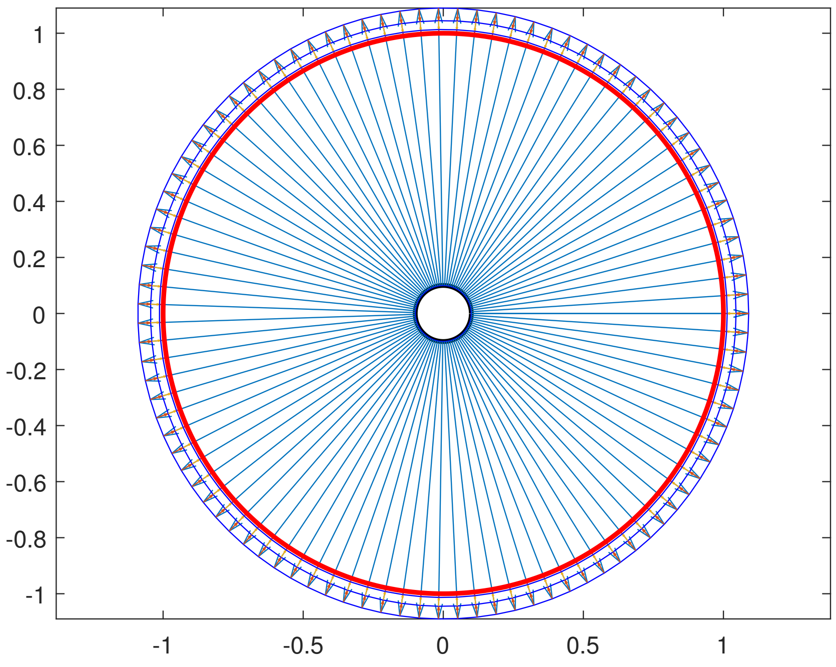
2.5 Steepest Descent for Shapes

The MATLAB code above bears close resemblance to the Steepest Descent algorithm we saw in Section 1.4. We could formally write the corresponding algorithm for shapes as follows:

```
Algorithm 5: Steepest Descent for Shapes
1 Given starting shape $\Omega$
2 repeat
3     (1) Set $\Delta\partial\Omega \leftarrow -fn$
4     (2) Choose $t$ via BLS for Shapes
5     (3) Set $\partial\Omega \leftarrow \partial\Omega + t\Delta\partial\Omega$
6 until stopping criterion satisfied
```

Here we could use a stopping criterion of the form $|\mathrm{d}\mathcal{J}(\Omega, \Delta\partial\Omega)| < \varepsilon$ for a given tolerance $\varepsilon > 0$, as in the MATLAB code.

Running the code for integrand $f(x) = x_1^2 + x_2^2 - 1$ produces the following figure:



Though the first iteration shoots outside the level set, we achieve an extremely good approximation when the algorithm terminates after only 4 iterations. The first iteration uses step size 1 - if we chose the initial step size for BLS to be such that the iteration stays within the level set, say $t = 0.9$, the algorithm takes only 2 iterations and a fraction of the time.

While BLS for Shapes does generally help to reduce the number of iterations required, the effort of computing $\mathcal{J}$ and $\mathrm{d}\mathcal{J}$ can make the Steepest Descent algorithm a lot slower. Comparing the two descent methods for solving Example 2.1 using `tic` and `toc`, we find that the non-BLS method takes 0.635179 seconds, compared to the BLS method's 19.108205 seconds[‡].

2.6 Newton's Method for Shapes

In Section 1.5 we saw that *Newton's Method* on functions was a dramatic improvement on convergence to the optimal point in comparison to Steepest Descent, with the pitfall of increased computational effort. We have seen in Chebfun that the time taken to compute $\mathcal{J}$ and $\mathrm{d}\mathcal{J}$ is very

[‡]Computed on a Toshiba Satellite P50-C-18L with Intel Core i7.

long, so if we want to do such computations we need to use an efficient optimisation method that approximates the optimal shape in few iterations. For this reason we will attempt to develop a *Newton's Method* for shapes.

2.6.1 The Shape Hessian, $\mathrm{d}^2\mathcal{J}(\Omega, \mathcal{V}, \mathcal{W})$

We can derive the *Shape Hessian* in a similar manner to how we computed the shape derivative in Section 2.2. We obtain [4, page 1087]

$$\begin{aligned}\mathrm{d}^2\mathcal{J}(\Omega, \mathcal{V}, \mathcal{W}) &= \int_{\Omega} \nabla \cdot (\mathcal{W}\nabla \cdot (f\mathcal{V}))\,\mathrm{d}x \\ &= \int_{\partial\Omega} \nabla \cdot (f\mathcal{V})(\mathcal{W} \cdot n)\,\mathrm{d}S \\ &= \int_{\partial\Omega} (\nabla f \cdot \mathcal{V} + f\nabla \cdot \mathcal{V})(\mathcal{W} \cdot n)\,\mathrm{d}S,\end{aligned}$$

where $\mathcal{W} : \mathbb{R}^n \to \mathbb{R}^n$ is another sufficiently smooth vector field and the last line above follows from the same vector identity used in proving Theorem 2.2. Ideally we would then like an approximation to the cost functional to resemble a Taylor approximation, i.e

$$\mathcal{J}(\Omega + \partial\Omega) \approx \mathcal{J}(\Omega) + \mathrm{d}\mathcal{J}(\Omega, \partial\Omega) + \frac{1}{2}\mathrm{d}^2\mathcal{J}(\Omega, \partial\Omega, \partial\Omega). \tag{5}$$

How would we choose the descent direction $\partial\Omega$ to minimise this second-order approximation?

3 Constrained Shape Optimisation

4 Conclusion

References

[1] Stephen Boyd and Lieven Vandenberghe. *Convex Optimisation.* Cambridge University Press, 2004. `web.stanford.edu/~boyd/cvxbook/bv_cvxbook.pdf`

[2] Jan R Magnus and Heinz Neudecker. *Matrix Differential Calculus with Applications in Statistics and Econometrics.* Wiley, 1999.

[3] *Shape Optimization.* Wikipedia. `en.wikipedia.org/wiki/Shape_optimization`

[4] Ralf Hiptmair and Jingzhi Li. *Shape Derivatives in Differential Forms I: An Intrinsic Perspective.* Annali di Matematica Pura ed Applicata, 2013.