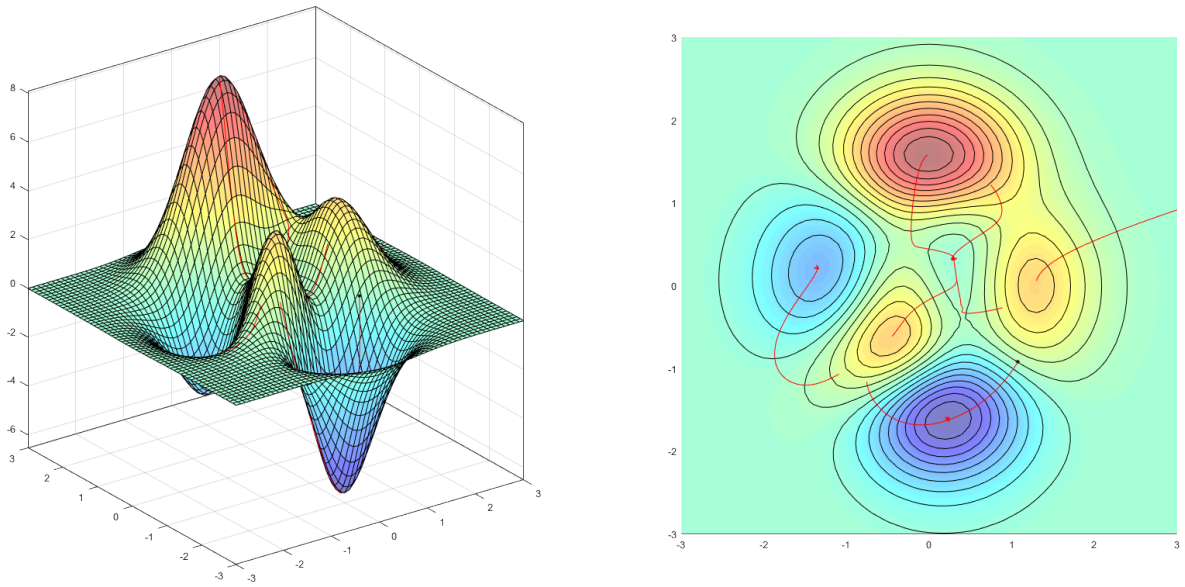
Shape Optimisation with PDE Constraints

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

9th 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 focussing on a shape optimisation test case that, despite its relative simplicity, exhibits an unexpected and interesting behaviour: solving this problem via Newton's method with a *truncated* second shape derivative performs better than using full second order information. We perform numerical (and some theoretical) investigations to try to shed light on this unexpected behaviour.

*One method we will explore is the *Gradient Descent* method; an example using `gradientDescentDemo` from the Machine Learning Toolbox is pictured below the title. MATLAB code by *Roger Jang* and *MIR Lab*, available at `mirlab.org/jang/matlab/toolbox/machineLearning`

Contents

1 Unconstrained Optimisation

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

1.1 Setting the Scene

We will be concerned with convex, twice continuously differentiable functions $f : \mathbb{R}^n \to \mathbb{R}$. Assuming that there is a unique minimiser of f

$$x^* = \arg\min_x f(x)$$

we have the necessary and sufficient condition

$$\nabla f(x^*) = 0.$$

This is a system of n equations in the variables $x_1, \ldots, x_n$, which we often solve by an iterative algorithm. That is, given an initial guess $x^{(0)}$ for x^*, we try to find a *minimising sequence* $(x^{(k)})$ of points such that

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

Often we only seek 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 descend to x^*,
- $t^{(k)}$ 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

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

for all k, unless $x^{(k)}$ is optimal. This is the property of a *descent method*. From convexity we have

$$\nabla f(x^{(k)})^T (y - x^{(k)}) \geqslant 0 \Longrightarrow f(y) \geqslant f(x^{(k)}) \quad \forall x, y$$

so if we set $y = x^{(k)} + \Delta x^{(k)}$, we deduce that the search direction in a descent method must satisfy

$$\nabla f(x^{(k)})^T \Delta x^{(k)} < 0.$$

That is, $\Delta x^{(k)}$ must make an acute angle with the negative gradient $-\nabla f(x^{(k)})$. Such a direction is called a *descent direction*.

A generic descent method goes as follows:

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

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

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

1.3 Line Search

To answer the second question, we discuss two common methods of deciding the step size, called *exact line search (ELS)* and *backtracking line search (BLS).*

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

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

It might not be possible to find t in practice, but even if we can, 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

Given descent direction Δx, $\alpha \in (0, 0.5), \beta \in (0, 1)$
$t := 1$
while $f(x + t\Delta x) > f(x) + \alpha t \nabla f(x)^T \Delta x$ **do**
    Set $t \leftarrow \beta t$
end

The resulting value of t is chosen for our 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

$$f(x + t\Delta x) > f(x) + \alpha t \nabla f(x)^T \Delta x$$

holds. Since Δx is a descent direction, we have $\nabla f(x)^T \Delta x < 0$, so

$$f(x + t\Delta x) \approx f(x) + t\nabla f(x)^T \Delta x < f(x) + \alpha t \nabla f(x)^T \Delta x$$

for small enough t, demonstrating that BLS eventually terminates. The choice of parameters α, β has a noticeable but not dramatic effect on convergence; ELS can sometimes improve convergence rate but not by much. It is generally not worth the computational effort of implementing ELS.

1.4 Gradient Descent

A natural choice of descent direction is $\Delta x = -\nabla f(x)$, which gives the *Gradient Descent* method:

Algorithm 2: Gradient Descent

Given starting point x
repeat
    (1) Set $\Delta x \leftarrow -\nabla f(x)$
    (2) Choose t via ELS/BLS
    (3) Set $x \leftarrow x + t\Delta x$
until *stopping criterion satisfied*

Recall we want to approximate x^* where $\nabla f(x^*) = 0$, so we expect $\nabla f(x)$ to be small for $x \approx x^*$. For this reason, the stopping criterion is usually of the form $||\nabla f(x)||_2 \leqslant \varepsilon$ for some chosen small $\varepsilon > 0$. Most practical uses of Gradient Descent check the stopping criterion after step (1).

It can be shown (see [1]) that Gradient Descent with ELS results in the inequality

$$f(x^{(k)}) - f(x^*) \leqslant c^k(f(x_0) - f(x^*))$$

where $c < 1$. This bound shows that Gradient Descent is *linearly convergent*. If we instead use BLS, we get the same inequality (with a slightly different value of c depending on α, β, but still less than 1) so again we have linear convergence. Gradient Descent is very simple, but convergence can be very slow and so it is rarely used in practice.

1.5 Steepest Descent

Alternatively we could choose the descent direction by considering the Taylor expansion of $f(x+\Delta x)$ around x. The first-order approximation is

$$f(x + \Delta x) \approx f(x) + \nabla f(x)^T \Delta x$$

where the term $\nabla f(x)^T \Delta x$ is the *directional derivative* of f at x in the direction Δx.

To minimise the directional derivative and thus find the *steepest descent*, we first need to fix the size of Δx. If we don't do this, we could make Δx as large as we like: recall that $\nabla f(x)^T \Delta x < 0$ for a descent direction Δx. We define the *normalised steepest descent direction*

$$\Delta x_{\text{nsd}} := \underset{||\Delta x|| \leqslant 1}{\arg\min} \nabla f(x)^T \Delta x$$

for any norm $||\cdot||$ on $\mathbb{R}^n$. Geometrically speaking, this is a step in the unit ball of $||\cdot||$ which extends farthest in the direction $-\nabla f(x)$. It is more convenient in practice to use an unnormalised descent direction

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

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

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

We arrive at the *Steepest Descent* method, which uses Δx_{sd} as a search direction:

Algorithm 3: Steepest Descent

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

Note that if using ELS, scale factors in the descent direction Δx have no effect on the next iteration, so we may use Δx_{sd} or Δx_{nsd}.

Again we have unanswered questions:

- What norm on $\mathbb{R}^n$ do we use in the definition of Δx_{sd}?
- Does the choice of norm affect efficiency of the Steepest Descent method?

If we take the Euclidean norm $||\cdot||_2$, then the steepest descent direction is found to be

$$\Delta x_{\text{sd}} = -\nabla f(x)$$

so in fact the Steepest Descent method coincides with the Gradient Descent method.

What if we were to take a different norm?

1.6 Alternate Norms for Steepest Descent

We briefly explore a selection of non-Euclidean norms that could be employed in the Steepest Descent method, commenting on the advantages for using such norms.

1.6.1 ℓ^1-norm

Suppose we find Δx_{nsd} with respect to the ℓ^1-norm defined by $||x||_1 := \sum |x_i|$. Then

$$\Delta x_{\text{nsd}} = -\text{sgn}\left(\frac{\partial f(x)}{\partial x_i}\right) e_i$$

where e_i is the standard basis vector and i is any index where $||\nabla f(x)||_\infty = |(\nabla f(x))_i|$. Hence

$$\Delta x_{\text{sd}} = ||\nabla f(x)||_\infty \Delta x_{\text{nsd}} = -\frac{\partial f(x)}{\partial x_i} e_i.$$

An advantage of using the ℓ^1-norm is that the descent direction can always be expressed as a multiple of a standard basis vector, so it is relatively easy to determine.

1.6.2 Quadratic Norm

Suppose we find Δx_{nsd} with respect to the quadratic norm defined by $||x||_P := (x^T P x)^{1/2}$, where $P \in M_n(\mathbb{R})$ is a symmetric, positive-definite matrix. The steepest descent direction is

$$\Delta x_{\text{sd}} = -P^{-1}\nabla f(x).$$

Steepest Descent with respect to the quadratic norm can be thought of as applying Gradient Descent to the problem after a change of coordinates $\tilde{x} = P^{1/2}x$. The chosen P can have a dramatic effect on convergence rate: to see this, we define the *condition number* of a convex set C as

$$\text{cond}(C) := \frac{W_{max}^2}{W_{min}^2}$$

where $W_{\text{max}}, W_{\text{min}}$ are the maximum and minimum widths of C respectively. This gives a measure of the eccentricity of C. From [1], we find that Gradient Descent works well for low condition numbers of the *sublevel sets* $C_\alpha = \{x : f(x) \leqslant \alpha\}$, and poorly for large values of $\text{cond}(C_\alpha)$. We can therefore expect faster convergence if the sublevel sets, transformed by $P^{-1/2}$, are well-conditioned.

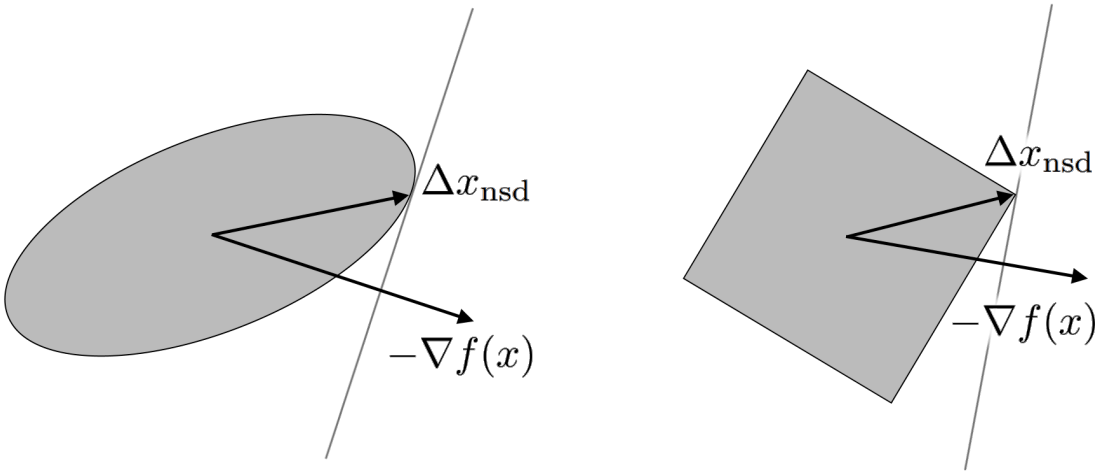


Figure 1: Unit balls with center x for the quadratic and ℓ^1-norm respectively, with arrows depicting the negative gradient $-\nabla f(x)$ and the corresponding descent direction Δx_{nsd}.

1.7 Newton's Method

Newton's Method goes one further than Steepest Descent, choosing the descent direction by analysing the Taylor expansion up to second-order. The approximation is

$$f(x + \Delta x) \approx f(x) + \nabla f(x)^T \Delta x + \frac{1}{2} \Delta x^T H \Delta x$$

where $H_{ij} = \frac{\partial f}{\partial x_i \partial x_j}$ is the *Hessian matrix*. This is a quadratic in Δx, minimised at the value

$$\Delta x_{\text{nt}} := -H^{-1} \nabla f(x).$$

We call Δx_{nt} the *Newton step* for f at x. Positive definiteness implies

$$\nabla f(x)^T \Delta x_{\text{nt}} = -\nabla f(x)^T H^{-1} \nabla f(x) < 0$$

unless $\nabla f(x) = 0$, so the Newton step is indeed a descent direction (unless x is optimal).

If f is quadratic, then $x + \Delta x_{\text{nt}}$ is the *exact* minimiser of f, and hence should 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^*.

We consider one more definition before turning to *Newton's Method*. We call

$$\lambda(x) := (\nabla f(x)^T H^{-1} \nabla f(x))^{1/2}$$

the *Newton decrement* at x. Note that if $\hat{f}$ is the second-order approximation of f at x, then

$$f(x) - \inf_y \hat{f}(y) = f(x) - \hat{f}(x + \Delta x_{\text{nt}}) = \frac{1}{2} \lambda(x)^2$$

so $\lambda(x)^2/2$ is an estimate for the error $f(x) - f(x^*)$, based on the quadratic approximation.

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

Algorithm 4: Newton's Method

Given starting point x, tolerance $\varepsilon > 0$
repeat
  (1) Set $\Delta x \leftarrow \Delta x_{\text{nt}}$
  (2) Set $\lambda^2 \leftarrow \lambda(x)^2$
  (3) **Stop** if $\lambda^2/2 \leqslant \varepsilon$
  (4) Choose t via BLS
  (5) 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 *quadratic convergence* near x^* (see [1]). Roughly speaking, this means that the number of correct digits doubles after each iteration. Newton's Method also scales with the size of the problem: its performance in $\mathbb{R}^{10000}$ is similar to problems in $\mathbb{R}^{10}$ say, with only a modest increase in the number of iterations. Good performance of Newton's Method is independent on choice of algorithm parameters, whereas the choice of norm in Steepest Descent was vital, for example.

A pitfall of Newton's Method is the cost of forming and storing the Hessian matrix H, and 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 Shape Differentiation

3 Shape Optimisation Test Case

4 Conclusion

References

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