Medical Physics / Biomedical Engineering 574

Homework Set 2 - SOLUTIONS

Diego Hernando

Due date: Monday, February 25, 2019
(Due by the end of the day, either electronically or on paper. Note that your Matlab implementation of the algorithms in problem 2 needs to be handed in electronically even if the rest of the homework is handed in on paper)

1. Reflection and Refraction (15 points)

A. (7.5 points) Reflection

We would like to find the quickest path to travel from (x_1, y_1) to (x_2, y_2) such that we bounce off the reflective surface. Determine the angle of reflection (ie: θ_2 in terms of θ_1).

SOLUTION:

Assume the reflective surface is on the line $(x,0)$ (ie: the horizontal axis). Let us denote the point of reflection as $(x_0,0)$. Then, the total travel time (as a function of the point of reflection) will be

$$T(x_0) = \frac{1}{v}\sqrt{(x_0-x_1)^2+y_1^2}+\frac{1}{v}\sqrt{(x_2-x_0)^2+y_2^2}$$

where $v > 0$ is the velocity of our object in this medium.

In order to find the minimum of $T(x_0)$, we can set its derivative as equal to zero:

$$\frac{dT}{dx_0} = \frac{1}{v}\frac{2(x_0-x_1)}{\sqrt{(x_0-x_1)^2+y_1^2}} - \frac{1}{v}\frac{2(x_2-x_0)}{\sqrt{(x_2-x_0)^2+y_2^2}} = 0$$

Which we can simplify as:

$$\frac{(x_0-x_1)}{\sqrt{(x_0-x_1)^2+y_1^2}} = \frac{(x_2-x_0)}{\sqrt{(x_2-x_0)^2+y_2^2}}$$

Which, based on the definition of the sine (and using the diagram in figure 1.1A), can be expressed as:

$$\sin(\theta_1) = \sin(\theta_2)$$

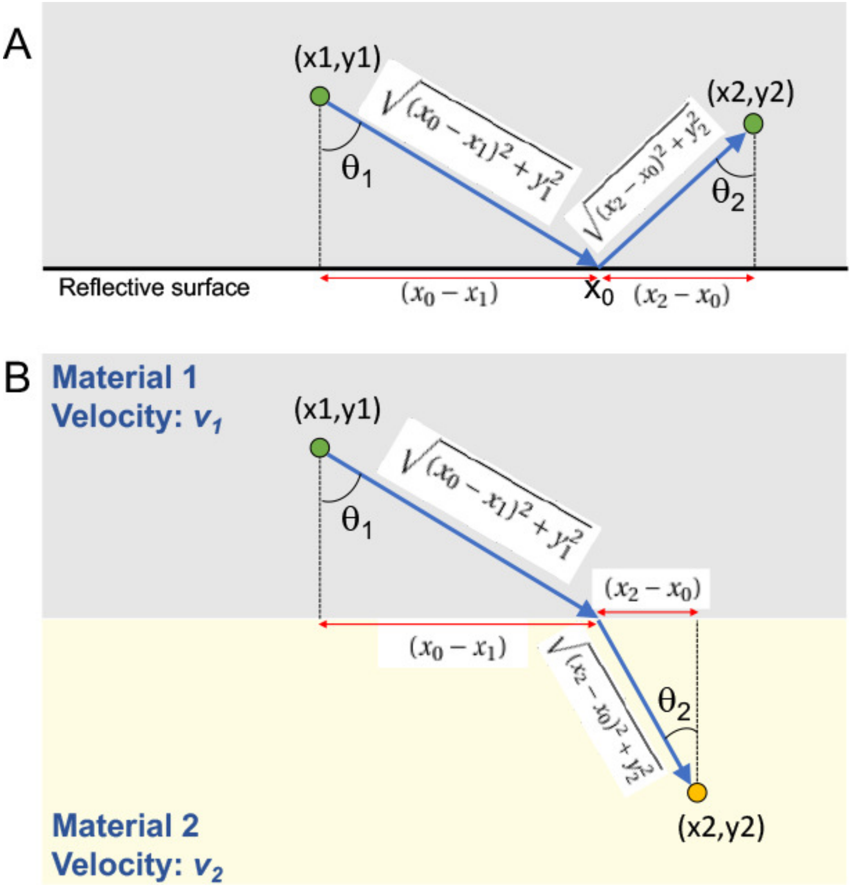


Figure 1.1: Bouncing off a reflective surface (A), and traveling through two media with different velocities (B): what is the optimum (fastest) path in each case?

which has a solution at $\theta_1 = \theta_2$.

Is this solution a minimizer or a maximizer of $T(x_0)$? We can calculate the second derivative:

$$\frac{d^2T}{dx_0^2} = \frac{1}{v}\frac{4(x_0-x_1)^2}{((x_0-x_1)^2+y_1^2)^{3/2}} + \frac{1}{v}\frac{2}{\sqrt{(x_0-x_1)^2+y_1^2}} + \frac{1}{v}\frac{4(x_2-x_0)^2}{((x_2-x_0)^2+y_2^2)^{3/2}} + \frac{1}{v}\frac{2}{\sqrt{(x_2-x_0)^2+y_2^2}}$$

Since all the terms in the second derivative are positive, $\frac{d^2T}{dx_0^2} > 0$ everywhere (and at our solution specifically) so our optimum point is a minimizer, not a maximizer.

B. (7.5 points) Refraction

We would like to find the quickest path between two points, as we travel at different velocities on different sides of an interface. Determine the angle of refraction (ie: the relationship between θ_2 and θ_1 in terms of the ratio between the velocities). How does this compare to Snell's law for the refraction of light?

SOLUTION:

Assume the refractive surface is on the line $(x,0)$ (ie: the horizontal axis), such that $y_1 > 0$ and $y_2 < 0$. Let us denote the point of refraction as $(x_0,0)$. Then, the total travel time (as a function of the point of reflection) will be

$$T(x_0) = \frac{1}{v_1}\sqrt{(x_0-x_1)^2+y_1^2} + \frac{1}{v_2}\sqrt{(x_2-x_0)^2+y_2^2}$$

.

In order to find the minimum of $T(x_0)$, we can set its derivative as equal to zero:

$$\frac{dT}{dx_0} = \frac{1}{v_1}\frac{2(x_0-x_1)}{\sqrt{(x_0-x_1)^2+y_1^2}} - \frac{1}{v_2}\frac{2(x_2-x_0)}{\sqrt{(x_2-x_0)^2+y_2^2}} = 0$$

Which we can simplify as:

$$\frac{1}{v_1}\frac{(x_0-x_1)}{\sqrt{(x_0-x_1)^2+y_1^2}} = \frac{1}{v_1}\frac{(x_2-x_0)}{\sqrt{(x_2-x_0)^2+y_2^2}}$$

Which, based on the definition of the sine (and using the diagram in figure 1.1A), can be expressed as:

$$\frac{\sin(\theta_1)}{v_1} = \frac{\sin(\theta_2)}{v_2}$$

which has a solution at

$$\frac{\sin(\theta_2)}{\sin(\theta_1)} = \frac{v_2}{v_1}$$

.

Is this solution a minimizer or a maximizer of $T(x_0)$? We can calculate the second derivative:

$$\frac{d^2T}{dx_0^2} = \frac{1}{v_1}\frac{4(x_0 - x_1)^2}{((x_0 - x_1)^2 + y_1^2)^{3/2}} + \frac{1}{v_1}\frac{2}{\sqrt{(x_0 - x_1)^2 + y_1^2}} + \frac{1}{v_2}\frac{4(x_2 - x_0)^2}{((x_2 - x_0)^2 + y_2^2)^{3/2}} + \frac{1}{v_2}\frac{2}{\sqrt{(x_2 - x_0)^2 + y_2^2}}$$

Since all the terms in the second derivative are positive, $\frac{d^2T}{dx_0^2} > 0$ everywhere (and at our solution specifically) so our optimum point is a minimizer, not a maximizer.

This solution is also known as Snell's law.

2. Implement Optimization Algorithms (85 points)

Suppose we want to solve the minimization problem $\min_{\mathbf{x}} f(\mathbf{x})$, where $f(\mathbf{x})$ is a quadratic function defined as follows:

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{Q}\mathbf{x} - \mathbf{x}^T\mathbf{b} \tag{2.1}$$

for some symmetric positive definite matrix $\mathbf{Q} = \mathbf{Q}^T > 0$ and some vector $\mathbf{b}$.

For this problem, you need to implement the three algorithms described below to solve this optimization problem in a case where $\mathbf{Q} = \mathbf{A}^T\mathbf{C}\mathbf{A}$ (note that we define $\mathbf{Q}$ in this way to ensure that it is positive definite), with

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 2 & 1 \\ 4 & 0 & 1 \end{bmatrix} \tag{2.2}$$

$$\mathbf{C} = \begin{bmatrix} 400 & 0 & 0 \\ 0 & 20 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

and

$$\mathbf{b} = \begin{bmatrix} 100 \\ 100 \\ 100 \end{bmatrix} \tag{2.4}$$

For each of the algorithms below, run it until convergence or for 500 iterations (whichever happens first), with initial guess $\mathbf{x} = [0,0,0]^T$ in each case. For each algorithm, report:

i The path of the estimated $\mathbf{x}^k$ through the iterations (eg: plot the path of each of the three components of $\mathbf{x}^k$ as a function of k, for each algorithm).

ii The evolution of the corresponding $f(\mathbf{x}^k)$ through the iterations (eg: plot $f(\mathbf{x}^k)$ as a function of k, for each algorithm).

iii The "solution" $\hat{\mathbf{x}}$ and corresponding $f(\hat{\mathbf{x}})$ after iterations are done

In addition to reporting these results/plots in the homework solutions you hand in, make sure to provide your code electronically to the instructor as a matlab script named 'runOptim.m' (this script may call other functions or scripts, which also need to be attached). This code should generate the plots/outputs described above.

A. (25 points) Steepest Descent

In this algorithm, we iterate in the direction of steepest descent (ie: the direction opposite to the gradient). Therefore, at each step we obtain the updated estimate as follows:

$$\mathbf{x}^{(k+1)} \quad = \quad \mathbf{x}^{(k)} - \alpha\nabla f(\mathbf{x}^{(k)}) \tag{2.5}$$

with α (the step size) chosen to minimize the 1D function $h(\alpha) = f(\mathbf{x}^{(k)} - \alpha\nabla f(\mathbf{x}^{(k)}))$. In this example, let us use a golden section search to determine α, with an initial range (for the golden section search) between $[0,1]$ at each iteration, ie: $a_0 = 0$ and $b_0 = 1$ using the notation from the lectures as an initialization for the golden section search performed at each iteration.

B. (25 points) Conjugate Gradients

The conjugate gradients (CG) algorithm is an iterative algorithm for solving quadratic optimization problems[1]. CG is designed to overcome the limitations of steepest descent (which sometimes converges very slowly because it travels in a zig-zag trajectory). We will not go over the proofs of the properties of CG in our lectures or homework[2], but we will note that CG is guaranteed to converge to the solution of a quadratic problem of the form described above in N iterations (where N is the size of our vector $\mathbf{x}$.

CG can be described as follows:

1. Set $k = 0$, select the initial point $\mathbf{x}^{(0)}$
2. $\mathbf{g}^{(0)} = \nabla f(\mathbf{x}^{(0)})$. If $\mathbf{g}^{(0)} = 0$, stop (done!), otherwise set $\mathbf{d}^{(0)} = -\mathbf{g}^{(0)}$.
3. $\alpha_k = -\frac{\mathbf{g}^{(k)H}\mathbf{d}^{(k)}}{\mathbf{d}^{(k)H}\mathbf{Q}\mathbf{d}^{(k)}}$.
4. $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)}$.
5. $\mathbf{g}^{(k+1)} = \nabla f(\mathbf{x}^{(k+1)})$. If $\mathbf{g}^{(k+1)} = 0$, stop.
6. $\beta_k = \frac{\mathbf{g}^{(k+1)H}\mathbf{Q}\mathbf{d}^{(k)}}{\mathbf{d}^{(k)H}\mathbf{Q}\mathbf{d}^{(k)}}$.
7. $\mathbf{d}^{(k+1)} = -\mathbf{g}^{(k+1)} + \beta_k \mathbf{d}^{(k)}$.
8. Set $k = k+1$: if $k > k_{MAX}$, stop. Otherwise, go to step 3 above.

Where we use a predetermined maximum number of iterations k_{MAX}. Note that the condition that $\mathbf{g}^{(k+1)} = 0$ can be replaced in a realistic implementation by condition that $\|\mathbf{g}^{(k+1)}\| < \epsilon$ for some small $\epsilon > 0$ (or course, what is 'small' depends on the problem - you can check the norm of the initial gradients in this problem to pick a reasonable threshold ϵ that is many orders of magnitude smaller).

C. (25 points) Newton's Method

In this algorithm, we approximate the function as a quadratic near the current estimate $\mathbf{x}^{k-1}$ (based on the gradient and Hessian), and iterate as follows:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \left(\nabla^2 f(\mathbf{x}^{(k)})\right)^{-1} \nabla f(\mathbf{x}^{(k)}) \tag{2.6}$$

[1] CG is also commonly used to solve non-quadratic optimization problems, although this requires some practical modifications and the theoretical guarantees of convergence in N iterations no longer hold.

[2] For further details into the derivation and properties of CG, there are several references and texts, including "An Introduction to the Conjugate Gradient Method Without the Agonizing Pain" by Jonathan Shewchuk, available at https://www.cs.cmu.edu/ quake-papers/painless-conjugate-gradient.pdf

where $\nabla f(\mathbf{x}^{(k)})$ represents the gradient of $f(\mathbf{x})$ evaluated at $\mathbf{x}^{(k)}$, $\nabla^2 f(\mathbf{x}^{(k)})$ represents the Hessian (second derivative matrix) of $f(\mathbf{x})$ evaluated at $\mathbf{x}^{(k)}$, and $\left(\nabla^2 f(\mathbf{x}^{(k)})\right)^{-1}$ is the matrix inverse of the Hessian.

Important note: Matlab has built-in functions that perform some of these algorithms. However, you are expected to implement the algorithms from scratch using only basic matrix-vector operations, matrix inverses (for Newton's method), loops, etc.

Hint: You may want to implement a single Matlab function that returns the value, gradient, and Hessian of our cost function, and then simply call this function as needed from each of the algorithms. This approach may help ensure that these evaluations are consistent across algorithms, and will also help generalize your algorithms to other cost functions.

D. (10 points) Discussion

Comment briefly on the relative performance of each algorithm in terms of speed of convergence. Demonstrate whether the relative performance of the algorithms changes if they were used to solve a very similar quadratic optimization problem as above, except with $\mathbf{Q} = \mathbf{A}^T\mathbf{C}\mathbf{A}$:

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 2 & 1 \\ 4 & 0 & 1 \end{bmatrix} \tag{2.7}$$

$$\mathbf{C} = \begin{bmatrix} 20 & 0 & 0 \\ 0 & 20 & 0 \\ 0 & 0 & 20 \end{bmatrix} \tag{2.8}$$

and

$$\mathbf{b} = \begin{bmatrix} 100 \\ 100 \\ 100 \end{bmatrix} \tag{2.9}$$

(note that only the **C** matrix is different). Can you give an intuitive/pictorial explanation (eg: depicting analogous 2D cases where you can plot the isocontours and iterations for each algorithm) for the difference between these two problems as far as convergence is concerned?

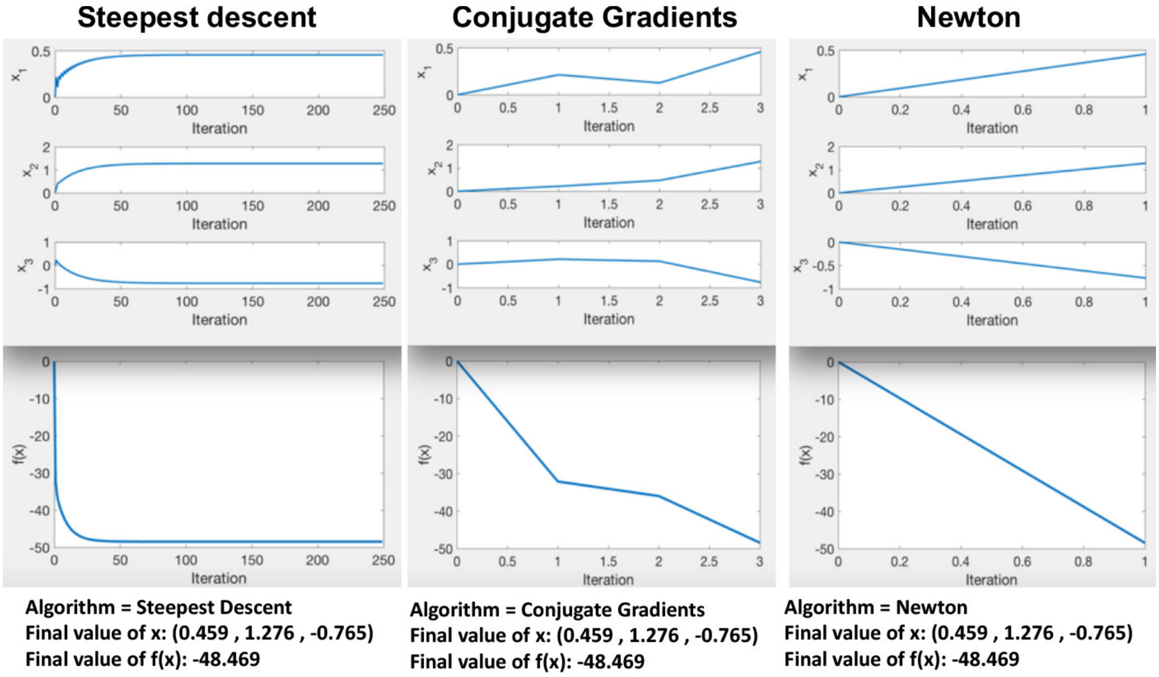
SOLUTION:

Please see files `test_optim.m` (main script) and `evalGradients.m` (auxiliary function) for an example implementation. Example results for the three algorithms are shown in figure 2.1. These results illustrate the very different performance of the three methods:

- *Steepest descent:* As mentioned in class, steepest descent requires many many iterations to converge, and results in a 'seesaw' pattern in the estimates through the iterations. This is particularly problematic in this quadratic function becase the matrix **Q** has very 'anisotropic' eigenvalue structure (eigenvalues are $\approx (8, 35, 5674)$), which leads to very anisotropic isocontours.

- *Conjugate gradients:* As also briefly mentioned in class, CG requires as many iterations as the size of the unknown vector **x** (ie: 3 in this case). This is a vast improvement over steepest descent.

- *Newton's algorithm:* Since this is a quadratic problem, and Newton approximates our function as a quadratic, which it solves at each iteration, Newton's algorithm converges in a single iteration.



```
Algorithm = Steepest Descent
Final value of x: (0.459 , 1.276 , -0.765)
Final value of f(x): -48.469
```

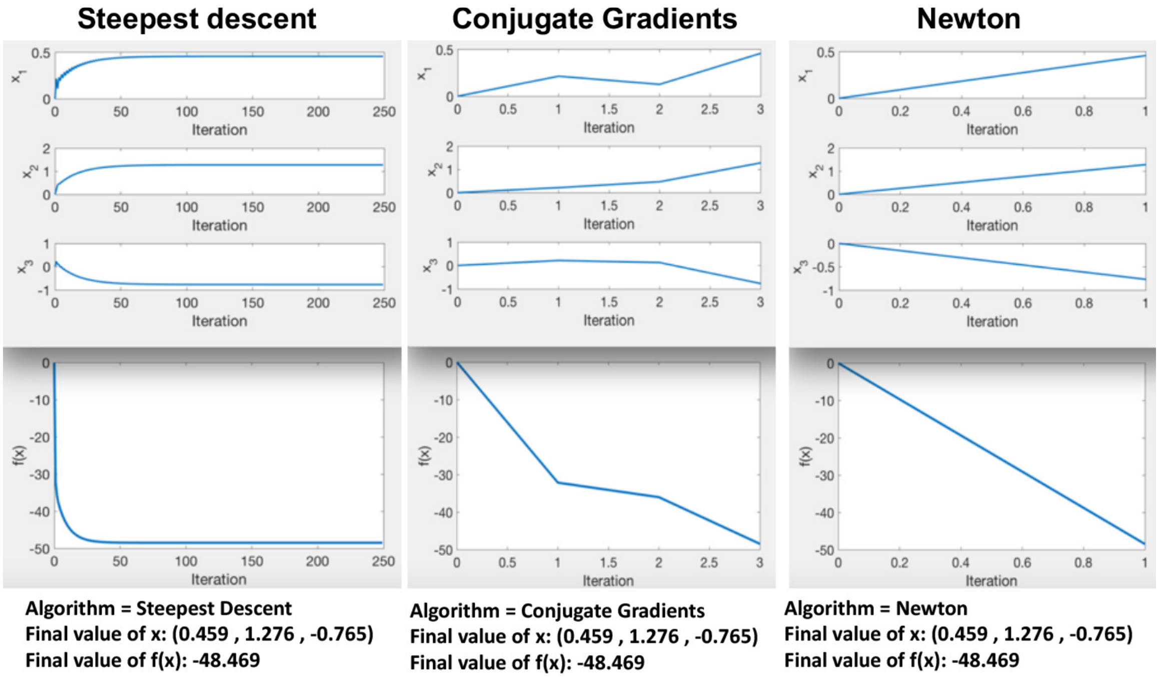
```
Algorithm = Conjugate Gradients
Final value of x: (0.459 , 1.276 , -0.765)
Final value of f(x): -48.469
```

```
Algorithm = Newton
Final value of x: (0.459 , 1.276 , -0.765)
Final value of f(x): -48.469
```

Figure 2.1: Iterative optimization of a quadratic cost function with very 'anisotropic' isocontours (ie: very uneven eigenvalues in our matrix **Q**) using three algorithms: steepest descent, conjugate gradients, and Newton's algorithm.

If instead we try these algorithms on the alternative problem proposed in section D (with more clustered eigenvalues in **Q**, which has eigenvalues $\approx (15, 222, 483)$), then steepest descent performs a lot better (although still slower than CG or Newton). The performance of CG[3] and Newton is similar to the original problem. A 2D graphical depiction of this effect is shown in figure 2.3.

[3] In reality, CG also benefits from more isotropic cost functions (when working with high-dimensional problems), although this was not demonstrated in this low-dimensional example. This is the reason that in practice, CG algorithms often include some form of pre-conditioning which attempts to 'push' the cost function into looking more isotropic.



Algorithm = Steepest Descent
Final value of x: (0.459 , 1.276 , -0.765)
Final value of f(x): -48.469

Algorithm = Conjugate Gradients
Final value of x: (0.459 , 1.276 , -0.765)
Final value of f(x): -48.469

Algorithm = Newton
Final value of x: (0.459 , 1.276 , -0.765)
Final value of f(x): -48.469

Figure 2.2: Iterative optimization of a quadratic cost function with less ‘anisotropic’ isocontours (ie: relatively less difference between eigenvalues in our matrix **Q**) using three algorithms: steepest descent, conjugate gradients, and Newton’s algorithm.

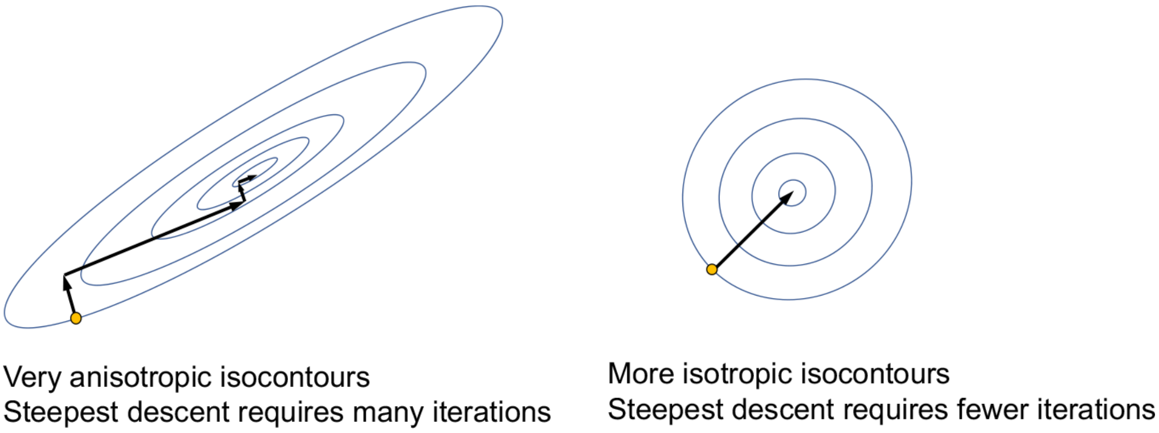

Very anisotropic isocontours
Steepest descent requires many iterations
More isotropic isocontours
Steepest descent requires fewer iterations

Figure 2.3: Graphical depiction of the difficulties of the steepest descent algorithm with very anisotropic functions (where the gradient at many locations may not point in the direction of the minimizer). In reality, CG also benefits from more isotropic cost functions, although this was not demonstrated in this low-dimensional example.