# 1 19th of October 2018 — A. Frangioni

## 1.1 Gradient method for quadratic functions

This is the simplest possible family of functions where a minumum exists.

**Do you recall?**

A quadratic function is defined as: $f(x) = \frac{1}{2}x^T Qx + qx$, so its gradient is the following $\nabla f(x) = Qx + q$

We are interested in finding local minima of the function $f$.

**Fact 1.1.** *A quadratic function $f : \mathbb{R}^n \to \mathbb{R}^n$, s.t. $f(x) = \frac{1}{2}x^T Qx + qx$ admits a minimum iff $Q \succeq 0$ ($Q$ is positive semidefinite).*

The gradient method generates points that move along orthognal directions. More formally, $x^{i+1} = x^i + \alpha^i d^i$, where $d^i = -\nabla f(x^i) = -Qx^i - q$

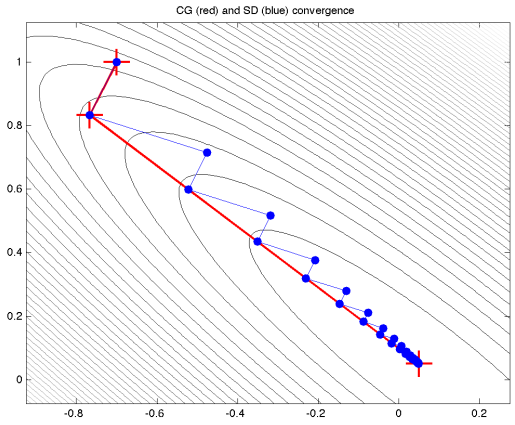


FIGURE 1.1: Some iterations of the gradient method

**Fact 1.2.** $\forall i \ < d^i, d^{i+1} > = 0$.

*Proof.* TODO □

$i\alpha^i = \frac{\|d^i\|^2}{d^{iT}Qd^i}$, $x^i \to \overline{x}$

Our aim is to estimate how fast this converges.

$f(x^i) - f_* = -\frac{1}{2}q^T Q^{-1} q$ we know everything for this function, from last lecture ($x^* = -Q^{-1}q$).

What can we say about $f(x^{i+1}) - f_* = \frac{1}{2}(x^{i+1} - x_*)^T A(x^{i+1} - x_*)$?

I want it in terms of $x^{i+1} = x^i - \frac{\|Qx^i - q\|^2}{(Qx^i-q)^T Q(Qx^i-q)}$

From this fomula we can say that the error at the current step $(i+1)$ is equal to the error of the step before $(i)$ divided by something. More precisely, $f_*(x) = \frac{1}{2}(x - x_*)^T Q(x - x_*) = f(x) + \frac{1}{2}{x_*}^T Q x_* = f(x) - f_*$

If the quantity after the minus is positive but less than 1 the error at the next step will be less than the error at the previous step. This means not only **linear convergence**, but a bit more, because linear convergence only takes into consideration steps in proximity to the limit, while this formula holds also at the beginning.

**Fact 1.3.** *If $Q$ is positive semidefinite $d^T Q d = \|d\|_Q^2$*

**Fact 1.4.** *If $Q$ is positive definite we can say that the error goes like $1 - \left(\frac{\|d_i\|^2}{(d^{i\,T} Q d^i)(d^{i\,T} Q^{-1} d^i)}\right)$ or, equivalently, $1 - \frac{{\|d_i\|_I}^2}{{\|d_i\|_Q}^2} \cdot \frac{{\|d_i\|_I}^2}{{\|d_i\|_{Q^{-1}}}^2}$.*

We are measuring a vector with three different norms.

We would like to estimate $\frac{<d_i, d_i>}{{d_i}^T Q d_i}$.

Given $\lambda_i$ the eigenvalues of $Q$, $\frac{1}{\lambda_i}$ are the eigenvalues of the matrix $Q^{-1}$.

We can say that $\lambda^n \|x\|^2 \leq x^T Q x \leq \lambda^1 \|x\|^2$, where $\lambda^n$ is the smallest eigenvalue, while $\lambda^1$ is the biggest.

We are looking for a close formula for calculating the convergence rate, since it depends reursivley by the steps done. So we want to do a worst case analysis in order to find a faster way to calculate convergence rate.

We want to prove that R is smaller than 1 so we are looking for an upperbound. A coarse upperbound is $(1 - \frac{\lambda^n}{\lambda^1})$, but we can prove more:

$$\forall\, x \in \mathbb{R}^n \quad \frac{\|x\|^4}{(x^T Q x)(x^T Q^{-1} x)} \geq \frac{4\lambda^1 \lambda^n}{(\lambda^1 + \lambda^n)^2}$$

We won't see the proof of this fact.

$R$ close to 0 means that the algorithm is converging fast, so when the larger eigenvalue $(\lambda^1)$ and the smallest eigenvalue $(\lambda^n)$ are very close to eachother the algorithm is converging fast. We can say that, since the eigenvalues are the axis of the ellipsoid, the algorithm is converging fast when the ellipsoid has a round shape.

We can provide an even better estimate, which is $\left(\frac{\lambda^1 - \lambda^n}{\lambda^1 + \lambda^n}\right)^2$

How can we understand if the number of iterations needed to converge is a good result? Well, it depends on how that number is obtained. If it's dimensional independent it's very good, because it scales well (when the size of the space — number of variables — increases). It only depends on the conditioning of the matrix $Q$.

Of course as $n$ grows $Q$ changes, so in practice it may happen that the conditioning of the problem is worsening as $n$ grows.

If the balls are very rounded the zig zags needed to start converging are very few.

**Obs:** We found a bound for the convergence speed of the algorithm when $Q$ is positive definite. What can we say when $Q$ is positive semidefinite? The algorithm works, but we can't provide an upperbound for the convergence rate. We are even more restrictive when dealing with machine precision, since if there is an eigenvalue which is bigger than zero, but very close to zero, becomes 0 on the machine, so we can't give an upperbound.

We will see how to deal with this case.

## 1.2 MatLab implementation

Let's talk about the implementation. First of all we need to set a proper value for $\epsilon$. A good idea would be the norm of the gradient. We need to give also some performance bound, like maximum number of iterations or the maximum amount of time.

**MatLab call:**

```
1  function[x, status] = SDQ(Q, q, x, fStar, eps, MaxIter)
```

where fStar is the optimal value, which should give us an idea of the convergence.

**Note**

- Always check the coherence of input values (check if the user passed allowed parameters);
- Always give all possible information about your result (for example if the algorithm stopped because the maximum number of iterations was reached or because the epsilon value was reached);
- Always check in your code the accepted values of variables during calculations: for example if you need to divide by a quantity that may be smaller than the precision check it before computing the ratio;
- Design a good log, in order to understand what's happening at each step.

In the code of that function we also kept track of the actual ratio between the error at one step and the error at the next one. This information tells us how many orders of magnitude the error decrease. We can find starting points where the ratio is exaclty $R$. We can observe that when the conditioning is quite good the error decrease faster than the $R$ limitation, but as soon as we change the values for $Q$ and $q$ things may be worse.

We saw from some examples that if the conditioning grows the number of iterations needed to find the minumum increases as well.

Trying the algorithm on some examples shows that the theoric results are reflected well in the practical case.

### 1.2.1 Error

When we are running an algorithm, starting from a point that will not lead to the optimum we would like to stop anyway, at a certain point, when we are close enough to the solution.
$\varepsilon_A = f(x^i) - f_* \leq \varepsilon$ (absolute error)

In this context, we introduce the consept of **relative error**, since the error may be big or small if compared with the value of the function.

This relative error is invriant for scaling transformations. Notice that if $f^*$ might be zero the forumla should be changed.

$\varepsilon_R = (\, f(x^i) - f_* \,) \,/\, |\, f_* \,| = \varepsilon_A \,/\, |\, f_* \,| \leq \varepsilon$ (relative error)

The problem is that often we don't know $f^*$, so it's impossible to compute this kind of error.

In this very common case a good lower bound $\underline{f} \leq f^*$for $f^*$. In this course we won't focus on finding $\underline{f}$.

Another stopping conditions may be the following:

- $\| \nabla f(x^i) \| \leq \varepsilon$ ("absolute version")
- $\| \nabla f(x^i) \| \,/\, \| \nabla f(x^0) \| \leq \varepsilon$ ("relative version")

The sencond stopping condition is expressed in relation to the first value for the norm of the gradient.

We usually chose the norm of the gradient as a threshold for precision, but we don't know how this quantity relates to $\epsilon_A$ or $\epsilon_B$.

**Example 1.1.** *for $X = \mathcal{B}(0, r)$ and $f$ convex, estimate $\varepsilon_A$ when $\| \nabla f(x^i) \| \leq \varepsilon$*

*I've got a convex function and I know the the minimum is in a ball. We can minimize the linear function in the range of the ball and that minumum is surely a lower bound.*