SECTION C: CONTINUOUS OPTIMISATION
LECTURE 5: CONJUGATE GRADIENTS

HONOUR SCHOOL OF MATHEMATICS, OXFORD UNIVERSITY
HILARY TERM 2005, DR RAPHAEL HAUSER

1. The Need for Further Methods. In Lecture 4 we motivated quasi-Newton methods by a reduced complexity per iteration as compared to the Newton-Raphson method. The notion of complexity we used was based on counting "basic computer operations", without taking into account that some operations are less costly than others. It is possible to be more rigorous and introduce mathematical models of computers, so-called Turing machines, that can operate on rational, real or complex numbers directly, but developing the necessary theory would lead us too far astray. For our purposes it usually suffices to think of a computer as being able to operate on real numbers, and occasionally we will take roundoff errors into account.

Any notion of complexity obtained in the framework of such a computational model reflects the running time of the implementation of an algorithm on an actual computer only to a first approximation. Real existing computers work with floating point numbers and have multiple levels of memory. When the data that needs to be carried over between iterations exceeds the active memory of the central processor, data has to be shifted back and forth between different levels of memory (for example between CPU and hard drive) and the machine can end up spending most of its processing time on data transfers rather than actual computations. In addition, different kinds of operations take different amounts of time: divisions are usually more costly than multiplications which are themselves more costly than additions. In practical implementations, these issues have to be taken into account in order to produce fast and reliable computer programs.

Quasi-Newton methods are amongst the most widely used algorithms in unconstrained optimisation, but any practical implementation makes it necessary to keep a $n \times n$ matrix H_k (the inverse of the approximate Hessian B_k) or L_k (the Cholesky factor of B_k) in the computer memory. In other words, there is an $O(n^2)$ *memory requirement* for quasi-Newton methods. In comparison, the steepest descent method only occupies $O(n)$ memory at any given time: after x_{k+1} has been computed, the registers that were previously used to store the n components of $\nabla f(x_k)$ can be overwritten with the data $\nabla f(x_{k+1})$, then x_{k+1} can be overwritten with x_{k+2} etc.

In situations where n is very large, the steepest descent method can therefore still cope with keeping all the necessary data in the main memory when quasi-Newton methods have long ceased to be effective and spend most of their time in data transfers. In this lecture we will analyse the *conjugate gradient method* which has the same memory requirements and complexity per iteration as the steepest descent method but typically converges much faster.

2. The Conjugate Gradient Method. We first develop the method in the context of minimising convex quadratic functions. Later, we will generalise the approach to arbitrary objective functions and call it *Fletcher–Reeves method*.

Let us consider the minimisation of a strictly convex quadratic objective function

$$\text{(P)} \qquad \min_{x\in\mathbb{R}^n} f(x) = x^{\mathrm{T}}Bx + b^{\mathrm{T}}x + a,$$

where B is a symmetric positive definite matrix. Using the first order necessary

optimality conditions from Lecture 2, we find that (P) is equivalent to the problem of solving the linear system $2Bx = -b$.

Here are a few facts about real symmetric matrices that you should remember from your course on linear algebra: let $A \in \mathbb{R}^{n\times n}$. If A is symmetric, then A can be written in decomposed form as $A = QDQ^{\mathrm{T}}$, where $Q^{\mathrm{T}}Q = \mathrm{I}$ (that is, Q is orthogonal), and D is the diagonal matrix $D = \mathrm{Diag}(\lambda(A))$ which has the vector $\lambda(A) = [\,\lambda_1(A)\ \dots\ \lambda_n(A)\,]^{\mathrm{T}} \in \mathbb{R}^n$ of eigenvalues of A on its diagonal, and where the components of $\lambda(A)$ are ordered in nonincreasing order: $\lambda_1(A) \geq \cdots \geq \lambda_n(A)$. All eigenvalues are real. The columns of Q form an orthogonal basis of eigenvectors of A. A is invertible if and only if D is, and then $A^{-1} = QD^{-1}Q^{\mathrm{T}}$, where $D^{-1} = \mathrm{Diag}(\lambda(A)^{-1})$ is simply obtained as the component-wise inverse of D. A is positive semidefinite (respectively positive definite) if and only if $\lambda_i(A) \geq 0$ (respectively $\lambda_i(A) > 0$) for all i. In these cases the matrix $D^{\frac{1}{2}} := \mathrm{Diag}(\sqrt{\lambda(A)})$, obtained as the component-wise square root of D, is well-defined, and we can set $A^{\frac{1}{2}} := QD^{\frac{1}{2}}Q^{\mathrm{T}}$. $A^{\frac{1}{2}}$ is the *unique* symmetric positive semidefinite matrix for which the identity $A^{\frac{1}{2}}A^{\frac{1}{2}} = A$ holds.

Let us now return to our optimisation problem (P) and observe that an additive constant in the objective function does not change its minimiser. Therefore, we can reformulate the problem as

$$\text{(P')} \qquad \min f(x) = (x - x^*)^{\mathrm{T}} B (x - x^*) = y^{\mathrm{T}} y = g(y),$$

where $x^* = -(1/2)B^{-1}b$ and $y = B^{\frac{1}{2}}(x - x^*)$. Thus, the objective function of our minimisation problem looks particularly simple in the transformed variables y. Of course, transforming the problem into y-coordinates is the same as solving the original problem: we would first have to find x^*, which is precisely the minimiser of (P)! So, there seems to be no gain from considering (P').

Nevertheless, we can use y-coordinates conceptually to understand how the conjugate gradient algorithm works in the original coordinates x: we would like to construct an iterative sequence $(x_k)_{k\in\mathbb{N}}$ such that the corresponding sequence of $y_k = B^{\frac{1}{2}}(x_k - x^*)$ behaves sensibly. Let us assume that the current iterate is x_k and that the search direction d_k has been computed. The exact line search $\alpha_k = \arg\min_\alpha f(x_k + \alpha d_k)$ is the same as $\alpha_k = \arg\min_\alpha g(y_k + \alpha p_k)$, where $p_k = B^{\frac{1}{2}} d_k$ is the search direction in y-coordinates. Therefore,

$$\alpha_k = \arg\min_{\alpha} \|y_k\|^2 + 2\alpha p_k^{\mathrm{T}} y_k + \alpha^2 \|p_k\|^2.$$

Since this is a univariate quadratic minimisation problem, it is trivial to establish that

$$\alpha_k = -\frac{p_k^{\mathrm{T}} y_k}{\|p_k\|^2}.$$

If we set $y_{k+1} = y_k + \alpha_k p_k$, then we find

$$y_{k+1}^{\mathrm{T}} p_k = \left(y - \frac{p_k^{\mathrm{T}} y_k}{\|p_k\|^2} p_k \right)^{\mathrm{T}} p_k = y_k^{\mathrm{T}} p_k - y_k^{\mathrm{T}} p_k = 0. \tag{2.1}$$

2.1. A key observation. A *key observation* is that the relation (2.1) is *independent* of the location of x_k: as long as we use the same search direction $d = \pm d_k$ but an arbitrary starting point $x \in \mathbb{R}^n$ for the line search

$$\alpha^* = \arg\min_{\alpha\in\mathbb{R}} f(x + \alpha d),$$

the point $x_+ = x + \alpha^* d$ ends up lying in the affine hyper-plane $\pi_k := x^* + B^{-1/2} p_k^\perp$, where

$$p_k^\perp = \{y : p_k^{\mathrm{T}} y = 0\}$$

is the orthogonal complement of p_k. In subsequent searches, it therefore never makes sense to leave π_k again, in fact, f always improves when moving from a point $x \in \mathbb{R}^n$ to its (skewed) projection along d_k into π_k, see Figure 2.1.

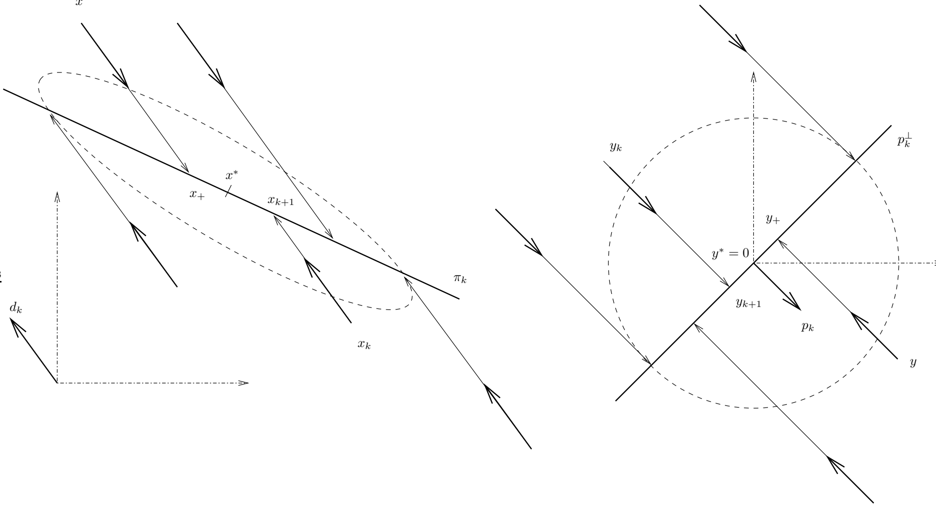

FIG. 2.1. *One step of the conjugate gradient method applied at different starting points x with search directions $\pm d_k$ in x-coordinates (left-hand side of diagram) and the corresponding directions $\pm p_k$ in the transformed coordinates y (right-hand side of diagram). The computation takes place in x-coordinates, but the geometry is best understood in y-coordinates.*

The requirement that all subsequent line searches are to be conducted within π_k amounts to the condition $p_j \perp p_k$ for all $j > k$, or equivalently expressed in x-coordinates,

$$d_k^{\mathrm{T}} B d_j = 0 \qquad \forall j \geq k+1. \tag{2.2}$$

If this relation holds, we say that d_k and d_j are B-conjugate (which is the same as orthogonality with respect to the Euclidean inner product defined by B).

Note that the restriction $f|_{\pi_k}$ is a strictly convex quadratic function on the affine subpace π_k. After choosing a search direction d_{k+1} that satisfies (2.2), we can repeat our argument and find that x_{k+2} will lie in an affine hyper-plane π_{k+1} of π_k to which any future line-search must be restricted. Arguing this way starting with $k = 0$ and $\pi_0 = \mathbb{R}^n$, we find that the dimension of the search space π_k is reduced by 1 in each iteration. This procedure must therefore terminate after n iterations at the latest (that is, after iteration $k = n-1$). In the process, we will have chosen search directions d_k $(k = 0, \ldots, n-1)$ which are mutually B-conjugate:

$$d_i^{\mathrm{T}} B d_j = 0 \qquad \forall\, i \neq j.$$

We have just developed all the different parts that constitute a proof of the following termination result:

THEOREM 2.1. *Let $f(x) = x^{\mathrm{T}}Bx + b^{\mathrm{T}}x + a$ be defined on $\mathbb{R}^n$ with $B \succ 0$, let $x_0 \in \mathbb{R}^n$ be a starting point and $d_k \in \mathbb{R}^n \setminus \{0\}$ $(k = 0, \dots, n-1)$ mutually B-conjugate search directions. Furthermore, for $(k = 0, \dots, n-1)$ let $x_{k+1} = x_k + \alpha_k d_k$, where $\alpha_k = arg\min_{\alpha\in\mathbb{R}} f(x_k + \alpha d_k)$. Then x_n is the global minimiser of f.*

Note that since B is nonsingular, there cannot exist more than n mutually B-conjugate search directions d_k. This yields an alternative argument for the proof of termination after n steps.

2.2. How to choose B-conjugate search directions. So far we have a finite termination result for a family of algorithms that iterate over exact line searches along mutually conjugate search directions. We did not otherwise specify how to compute these directions. In general, we could do this by choosing any descent direction v_k and then applying a procedure that corresponds to a Gram–Schmidt orthogonalisation in the transformed space:

LEMMA 2.2. *Let $v_0, \dots, v_{n-1} \in \mathbb{R}^n$ be linearly independent vectors, and let $d_0, \dots, d_{n-1}$ be recursively defined as follows,*

$$d_k = v_k - \sum_{j=0}^{k-1} \frac{d_j^{\mathrm{T}} B v_k}{d_j^{\mathrm{T}} B d_j} d_j. \tag{2.3}$$

Then $d_i^{\mathrm{T}} B d_k = 0$ for all $i \neq k$.

Proof. We proceed by induction over k and show that all distinct d_i with $i \leq k$ are mutually B-conjugate. For $k = 0$ there is nothing to prove. We may therefore assume that $d_i^{\mathrm{T}} B d_j = 0$ for all $i, j \in \{0, \dots, k-1\}$, $i \neq j$. Let $i < k$. Then

$$d_i^{\mathrm{T}} B d_k = d_i^{\mathrm{T}} B v_k - \sum_{j=0}^{k-1} \frac{d_j^{\mathrm{T}} B v_k}{d_j^{\mathrm{T}} B d_j} d_i^{\mathrm{T}} B d_j = d_i^{\mathrm{T}} B v_k - d_i^{\mathrm{T}} B v_k = 0.$$

Note that the linear independence of the v_j guarantees that none of the d_j is zero, and hence that $d_j^{\mathrm{T}} B d_j > 0$ for all j. □

Unfortunately, this procedure would require that we hold the vectors d_j $(j < k)$ in the computer memory. Thus, as k approaches n the method would require $O(n^2)$ memory, defying one of the main purposes of the new algorithm. Luckily, a second key observation shows that we can get away with $O(n)$ storage if we choose the steepest descent direction as v_k:

LEMMA 2.3. *Let the procedure of Theorem 2.1 be applied to the B-conjugate search directions d_k $(k = 0, \dots, n-1)$, where $d_0 = -\nabla f(x_0)$ and d_k is computed via (2.3) with $v_k = -\nabla f(x_k)$ $(k = 1, \dots, n-1)$. Then $\nabla f(x_j)^{\mathrm{T}} \nabla f(x_k) = 0$ for $(j = 0, \dots, k-1)$, and $(k = 1, \dots, n-1)$.*

Proof. First, note that $\nabla f(x_k) = 2Bx_k + b$ for all k. We claim that this implies that $d_j^{\mathrm{T}} \nabla f(x_k) = 0$ for $(j = 0, \dots, k-1)$. In order to prove this claim, we proceed by induction over k. The statement holds trivially true for $k = 0$. We can therefore

assume that it holds for k, and it suffices to prove that it holds for $k+1$ too. By the induction hypothesis we have

$$\begin{aligned} d_j^{\mathrm{T}} \nabla f(x_{k+1}) &= d_j^{\mathrm{T}} \left(2B(x_k + \alpha_k d_k) + b\right) \\ &= d_j^{\mathrm{T}} \nabla f(x_k) + 2\alpha_k d_j^{\mathrm{T}} B d_k \\ &= 0, \qquad (j = 0, \ldots, k-1). \end{aligned}$$

On the other hand, $d_k^{\mathrm{T}} \nabla f(x_{k+1}) = 0$ is the first order optimality condition for the line search $\min_\alpha \, f(x_k + \alpha d_k)$ that defines x_{k+1}. Hence, our claim is true. Next, note that (2.3) implies that

$$\mathrm{span}(d_0, \ldots, d_k) = \mathrm{span}(\nabla f(x_0), \ldots, \nabla f(x_k)) \qquad (k = 0, \ldots, n-1).$$

For $j < k$ there exist therefore $\lambda_1, \ldots, \lambda_j$ such that $\nabla f(x_j) = \sum_{i=0}^{j} \lambda_i d_i$, and we have

$$\nabla f(x_j)^{\mathrm{T}} \nabla f(x_k) = \sum_{i=1}^{j} \lambda d_i^{\mathrm{T}} \nabla f(x_k) = 0.$$

□

We are now ready to put all the pieces of the conjugate gradient algorithm together: substituting $\nabla f(x_{j+1}) - \nabla f(x_j) = 2\alpha_j B d_j$ into (2.3) with $v_k = -\nabla f(x_k)$, we obtain

$$d_k = -\nabla f(x_k) + \sum_{j=0}^{k-1} \frac{\nabla f(x_{j+1})^{\mathrm{T}} \nabla f(x_k) - \nabla f(x_j)^{\mathrm{T}} \nabla f(x_k)}{\nabla f(x_{j+1})^{\mathrm{T}} d_j - \nabla f(x_j)^{\mathrm{T}} d_j} d_j.$$

Lemma 2.3 implies that all but the last summand in the the right hand side expression are zero. Moreover, multiplying Equation (2.3) by $\nabla f(x_k)^{\mathrm{T}}$ and then replacing k by $k-1$, we deduce from Lemma 2.3 that $d_{k-1}^{\mathrm{T}} \nabla f(x_{k-1}) = -\|\nabla f(x_{k-1})\|^2$. Together with $d_{k-1}^{\mathrm{T}} \nabla f(x_k) = 0$ this implies that

$$d_k = -\nabla f(x_k) + \frac{\|\nabla f(x_k)\|^2}{\|\nabla f(x_{k-1})\|^2} d_{k-1}. \tag{2.4}$$

This is the *conjugate gradient* rule for updating the search direction.

Note that for the computation of d_k we only need to keep two vectors and one number stored in the main memory: d_{k-1}, x_k and $\|\nabla f(x_{k-1})\|^2$, which can be overwritten when the corresponding new data are computed. Let us now summarise the conjugate gradient algorithm:

Algorithm 2.4.

S0 *Choose* x_0*, set* $d_0 = -\nabla f(x_0)$.

S1 *For* $k = 0, 1, \ldots, n-1$ *do the following:*

1. *compute* $\alpha_k = \arg\min_\alpha f(x_k + \alpha d_k)$,
2. *set* $x_{k+1} = x_k + \alpha_k d_k$,
3. *if* $k < n-1$*, compute*

$$d_{k+1} = -\nabla f(x_{k+1}) + \frac{\|\nabla f(x_{k+1})\|^2}{\|\nabla f(x_k)\|^2} d_k.$$

S2 *Return* $x^* = x_n$.

Apart from the low memory requirements, the method has great advantages when a quick approximation to x^* is needed: in general x_k approximates x^* closely after very few iterations, and the remaining iterations are used for fine-tuning the result. On the other hand, the conjugate gradient algorithm runs into numerical problems when B is ill-conditioned, that is, when the ratio $\kappa(B) = \lambda_1(B)/\lambda_n(B)$ between the largest and smallest eigenvalues of B is very large. In these cases the algorithm can be improved through the use of so-called preconditioners. For further details see for example G. Golub and C. Van Loan "Matrix Computations", Third Edition, Johns Hopkins University Press.

3. The Fletcher-Reeves Method. Algorithm 2.4 can be adapted for the minimisation of an arbitrary C^1 objective function f and is then called *Fletcher-Reeves method*. The main differences are the following:

1. Exact line-searches have to be replaced by practical line-searches. Unfortunately, the Wolfe conditions are not always sufficient for this purpose, because we should also require that the new search direction is a descent direction, that is,

$$-\|\nabla f(x_{k+1})\|^2 + \frac{\|\nabla f(_{k+1})\|^2}{\|\nabla f(x_k)\|^2} d_k^{\mathrm{T}} \nabla f(x_{k+1}) < 0.$$

2. A termination criterion $\|\nabla f(x_k)\| < \epsilon$ has to be used to guarantee that the algorithm terminates in finite time.
3. Since Lemma 2.3 only holds for quadratic functions, the conjugacy of the search directions d_k can only be achieved approximately and is lost over time, unless d_k is set to the steepest descent direction again from time to time. Usually, one chooses such a reset every n iterations. If such a restart occurs in a sufficiently small neighbourhood of a local minimiser x^* of f, then the Fletcher-Reeves method can be expected to behave similarly to the conjugate gradient algorithm applied to the second order Taylor approximation of f around x^*.