Lecture 8

W. Hebisch

April 12, 2023

1 Conjugate direction methods

We already know about gradient descent and Newton method.

Gradient descent may require very large number of iterations, but per iteration cost is small.

Newton method typically converges in smaller number of iterations, but in each iteration we need to solve linear system of equations involving second derivative.

Conjugate direction methods have per iteration cost slightly larger than gradient descent, but should converge faster. In this sense are intermediate between gradient descent and Newton method.

Remark: In practice we care about conjugate gradient method, but theory is nicer when we make things slightly more general that is consider conjugate direction methods.

Conjugate direction methods originally were introduced as storage efficient method of exact solving of some linear systems. More precisely we know that for positive A minimization of

$$\frac{1}{2}\langle Ax, x\rangle - \langle b, x\rangle$$

is equivalent to solving

$$Ax = b.$$

Original conjugate direction methods minimized quadratic form, in at most n steps reaching exact solution where n is dimension of the problem.

Later it was observed that stopping method earlier one can get approximate solution and frequently one can get good solution in relatively small number of steps.

Conjugate direction methods were generalized to nonlinear problem. In such case method in no longer convergent in finite number of steps.

Let A be a positive definite matrix.

Definition. We say that a sequence of vectors d_i is A-orthogonal (conjugate) if and only if for $i \neq j$

$$\langle Ad_i, d_j\rangle = 0$$

Recall standard linear algebra:

Lemma 1.1 *If vectors d_i are A-conjugate, then they are linearly independent.*

Remark: To better see that this is standard result we can introduce new scalar product by the formula

$$\langle x, y\rangle_A = \langle Ax, y\rangle.$$

Then A-orthogonal simply means orthogonal with respect to this new scalar product.

A-conjugate vectors help in optimizing $f(x) = \frac{1}{2}\langle Ax, x\rangle - \langle b, x\rangle$.

Algorithm:

1. Take arbitrary x_0.
2. For i starting at 1 repeat: $x_i = x_{i-1} + \alpha_i d_i$ where α_i minimizes f on the line $x_{i-1} + \alpha_i d_i$.

First we deal with one dimensional optimization. Put

$$\phi(s) = f(x_{i-1} + sd_i).$$

We have

$$\phi'(s) = \langle A(x_{i-1} + sd_i) - b, d_i\rangle.$$

Writing $r_i = Ax_{i-1} - b$ we have

$$\phi'(s) = \langle r_i, d_i\rangle + s\langle Ad_i, d_i\rangle$$

so solving for $\phi'(\alpha_i) = 0$ we get

$$\alpha_i = \frac{-\langle r_i, d_i\rangle}{\langle Ad_i, d_i\rangle}$$

Lemma 1.2 *Let $x_* = \operatorname{argmin}_{x\in H} f(x)$ where $H = x_0 + \operatorname{lin}\{d_1, \ldots, d_i\}$. Then $x_* = x_i$.*

Proof: For any $x \in H$ we have $x - x_0 = \sum_{j=1}^{i} \beta_j d_j$. By A-orthogonality

$$\langle A(x - x_0), d_j\rangle = \beta_j\langle Ad_j, d_j\rangle$$

so

$$\langle A(x - x_0), x - x_0\rangle = \sum_{j=1}^{i} \beta_j^2\langle Ad_j, d_j\rangle$$

Now

$$f(x) = \frac{1}{2}\langle A((x - x_0) + x_0), (x - x_0) + x_0\rangle - \langle b, (x - x_0) + x_0\rangle$$

$$= \frac{1}{2}\langle A(x - x_0), x - x_0\rangle + \langle A(x - x_0), x_0\rangle - \langle b, x - x_0\rangle + \frac{1}{2}\langle Ax_0, x_0\rangle - \langle b, x_0\rangle$$

$$= \frac{1}{2}\left(\sum_{j=1}^{i} \beta_j^2 \langle Ad_j, d_j\rangle\right) + \sum_{j=1}^{i} \beta_j \langle Ad_j, x_0\rangle - \sum_{j=1}^{i} \beta_j \langle b, d_j\rangle$$

$$+\frac{1}{2}\langle Ax_0, x_0\rangle - \langle b, x_0\rangle$$

$$= c + \sum_{j=1}^{i} \phi_j(\beta_j)$$

where $c = \frac{1}{2}\langle Ax_0, x_0\rangle - \langle b, x_0\rangle$ and

$$\phi_j(\beta_j) = \frac{1}{2}\beta_j^2 \langle Ad_j, d_j\rangle + \beta_j(\langle Ad_j, x_0\rangle - \langle b, d_j\rangle)$$

so $f(x)$ is sum of functions of separate variables. Clearly for such functions optimizing in turn with respect to each variable gives optimal point. But this is exactly what our algorithm is doing.

Lemma 1.3 *$\langle r_{i+1}, d_j\rangle = 0$ for $j = 1, \ldots, i$.*

Proof: This follows from optimality of x_i on H: derivative of f at x_i in direction of d_j is 0. But

$$\partial_s f(x_i + sd_j) = \langle A(x_i + sd_j), d_j\rangle - \langle b, d_j\rangle$$

$$= \langle Ax_i - b, d_j\rangle + s\langle Ad_j, d_j\rangle = \langle r_{i+1}, d_j\rangle + s\langle Ad_j, d_j\rangle.$$

and when $s = 0$ this gives the claim.

1.1 Gram-Schmidt orthogonalization

Now it remains to find sequence of A-orthogonal vectors. There are many such sequences. General procedure is called Gram-Schmidt orthogonalization and works as follows:

- start from arbitrary basis $\{v_i\}$
- take $d_1 = v_1$
- iteratively, given $d_1, \ldots, d_i$ compute $c_{i+1,j} = \langle v_{i+1}, d_j\rangle / \langle d_j, d_j\rangle$ and put $d_{i+1} = v_{i+1} - \sum_{j=1}^{i} c_j d_j$

This is easy to program, but requires rather large number of operations and large memory.

Remark: In our case we would need to use $\langle x, y\rangle_A$ which significantly increases cost of computation.

1.2 Conjugate gradient method

Fortunately one A-orthogonal sequence of vectors is easy to find. Corresponding algorithm is called conjugate gradient method. We put $d_1 = -r_1$ and

$$d_{i+1} = -r_{i+1} + \beta_i d_i$$

where β_i is such that d_{i+1} and d_i are A-orthogonal, that is

$$\beta_i = \frac{\langle Ar_{i+1}, d_i \rangle}{\langle Ad_i, d_i \rangle}.$$

Lemma 1.4 *Assume that $r_1, \ldots, r_i$ are nonzero. Then*

$$\mathrm{lin}\{d_1, \ldots, d_i\} = \mathrm{lin}\{r_1, Ar_1, \ldots, A^{i-1}r_1\}$$

and d_j, $j = 1, \ldots, i$ are A-orthogonal.

Proof: By induction. Denote space on the right hand side above by V_i and on the left hand side by W_i. Clearly $V_1 = W_1$ and sequence consistiong of single vector is A-orthogonal. So we may assume that claim is valid for i and need to prove it for $i+1$. To prove that d_j are A-orthogonal it enough to prove that d_{i+1} is A-orthogonal to W_i. By definition of d_{i+1} we know that d_{i+1} is A-orthogonal to d_i. So it is enough to show that d_{i+1} is A-orthogonal to W_{i-1}.

Now

$$d_{i+1} = -r_{i+1} + \beta_i d_i$$

since d_i is A-orthogonal to W_{i-1} it is enough to show that r_{i+1} is A-orthogonal to W_{i-1}. For $v \in W_{i-1}$ we have

$$\langle Ar_{i+1}, v \rangle = \langle r_{i+1}, Av \rangle.$$

However,

$$Av \in AW_{i-1} = AV_{i-1} \subset V_i = W_i$$

where we used inductive assumption. Also, by inductive assumption d_j, $j = 1, \ldots, i$ are A-orthogonal so by previous lemma r_{i+1} is orthogonal to W_i. Hence

$$\langle r_{i+1}, Av \rangle = 0$$

which shows A-orthogonality of d_j, $j = 1, \ldots, i, i+1$.

It remains to show equality $V_{i+1} = W_{i+1}$. Clearly $AV_i \subset V_{i+1}$. We have $x_i - x_0 \in W_i$, so by inductive assumption $x_i - x_0 \in V_i$. Hence

$$r_{i+1} = Ax_i - b = A(x_i - x_0) + Ax_0 - b = A(x_i - x_0) + r_1 \in V_{i+1}$$

and $d_{i+1} = -r_{i+1} + \beta_i d_i \in V_{i+1}$, so $W_{i+1} \subset V_{i+1}$. But we proved that r_{i+1} is orthogonal to W_i, so since r_{i+1} is nonzero we have $r_{i+1} \notin W_i = V_i$, so r_{i+1} and V_i span V_{i+1}, so $V_{i+1} = W_{i+1}$.

A priori single step of conjugate gradient method needs two application of matrix A to vector: one to compute $r_{i+1} = Ax_i - b$ and second to compute Ad_i.

We can simplify computation of conjugate gradient method as follows: $-\langle r_i, d_i\rangle = \langle r_i, r_i\rangle - \beta_{i-1}\langle r_i, d_{i-1}\rangle = \langle r_i, r_i\rangle$ so

$$\alpha_i = \frac{\langle r_i, r_i\rangle}{\langle Ad_i, d_i\rangle}$$

$$r_{i+1} = r_i + \alpha_i A d_i$$

$$\beta_i = \frac{\langle r_{i+1}, Ad_i\rangle}{\langle Ad_i, d_i\rangle}$$

$$d_{i+1} = -r_{i+1} + \beta_i d_i$$

which needs one multiplication of vector by matrix, 3 scalar products and 2 vector linear combinations.

Recall that we say that a method is first order method if

$$x_i \in x_0 + \mathrm{lin}\{\nabla f(x_0), \ldots, \nabla f(x_{i-1})\}.$$

Lemma 1.5 *Assume f is quadratic as above, $V_i = \mathrm{lin}\{r_1, Ar_1, \ldots, A^{i-1}r_1\}$. For any first order method $x_i \in x_0 + V_i$.*

Proof: We have

$$\nabla f(x) = Ax - b = Ax_0 - b + A(x - x_0) = r_1 + A(x - x_0).$$

Now the claim follows by induction, if $x_i \in x_0 + V_i$, then

$$\nabla f(x_i) \in r_1 + AV_i \subset V_{i+1}$$

so

$$\mathrm{lin}\{\nabla f(x_0), \ldots, \nabla f(x_{i-1})\} \subset V_i$$

which gives the claim.

Note: $V_0 = \{0\}$ so $x_0 \in x_0 + V_0$ which gives starting point for induction.

Together with previous lemma this means that for quadratic functions conjugate gradient method is the best first order method.

Remark: Best here means that we get smallest value of goal function. There are different criteria, for example smallness of $\|\nabla f(x)\|$ or distance between x and optimal point x_∞.

Now we will explore some consequences. Let

$$P(t) = \sum_{j=0}^{i-1} \gamma_j t^j$$

be a polynomial of degree $i - 1$. Let

$$y_i = x_0 + P(A)r_1$$

Similar to previous argument y_i is obtained by in step i of a first order method, so error is at least as big as error of conjugate gradient method, that is

$$\langle A(x_i - x_\infty), x_i - x_\infty \rangle \leq \langle A(y_i - x_\infty), y_i - x_\infty \rangle.$$

But $r_1 = Ax_0 - b$ and $b = Ax_\infty$ so we can write

$$y_i - x_\infty = x_0 + P(A)(Ax_0 - Ax_\infty) - x_\infty = (I + AP(A))(x_0 - x_\infty)$$

so

$$\langle A(x_i - x_\infty), x_i - x_\infty \rangle \leq \langle A(I + AP(A))(x_0 - x_\infty), (I + AP(A))(x_0 - x_\infty) \rangle.$$

From this one can derive more specific estimate, we skip detail of proof, and just give the result.

Lemma 1.6 *Assume that A has $n - m$ eigenvalues in interval $[a, b]$ and remaining m eigenvalues are bigger than b. Then*

$$E(x_{m+1}) \leq \left(\frac{b-a}{b+a}\right)^2 E(x_0)$$

where

$$E(x) = \langle A(x - x_\infty), x - x_\infty \rangle$$

is error at x.

Note that what matters above are distinct eigenvalues. In particular, when A has only $m + 1$ eigenvalues, than we get optimum after $m + 1$ steps.

There are cases when A has small number of distinct eigenvalues, but in general we expect distinct eigenvalues. In fact, in problem 5.2 you should compute eigenvalues of matrix A corresponding to simple differential problem. This matrix has distinct eigenvalues and "spread out". While problem 5.2 is simplified so that eigenvalues of A are easily computable similar behaviour appears in other differential problems. Such problems appear in many engineering tasks.

1.3 Preconditioning conjugate gradient

To improve conditioning we can use transformation

$$x = C^{-1}y$$

In terms of y convergence depends on eigenvalues of $(C^{-1})^T AC^{-1}$. When $W = (C^T C)^{-1}$ is approximate inverse of A, then new problem is well conditioned (note that WA and $(C^{-1})^T AC^{-1}$ have the same eigenvalues).

It is important that there is no need to explicitly compute transformed problem. Namely, denoting by $x_i = C^{-1}y_i$ and using $'$ to denote other transformed quantities in y variables we have

$$A' = (C^{-1})^T AC^{-1},$$

$$b' = (C^{-1})^T b$$

$$r'_k = A'y_{k-1} - b' = A'Cx_{k-1} - b' = (C^{-1})^T(Ax_{k-1} - b).$$

Writing $z_k = Ax_{k-1} - b$ and transforming r'_k to x variables we compute

$$r_k = C^{-1}r'_k = (C^TC)^{-1}(Ax_{k-1} - b) = W(Ax_{k-1} - b) = Wz_k.$$

Next,

$$\langle r'_k, r'_k\rangle = \langle W(Ax_{k-1} - b), (Ax_{k-1} - b)\rangle = \langle Wz_k, z_k\rangle$$

so we can compute this in x coordinates. Similarly $d_1 = C^{-1}d'_1 = -C^{-1}r'_1 = -r_1$ and

$$d_{i+1} = C^{-1}d'_{i+1} = -C^{-1}r'_i + \beta'_i C^{-1}d'_i = -r_k + \beta'_i d_i,$$

$$\langle A'd'_i, d'_i\rangle = \langle AC^{-1}d'_i, C^{-1}d'_i\rangle = \langle Ad_i, d_i\rangle,$$

$$\langle r'_{i+1}, A'd'_i\rangle = \langle r_{i+1}, Ad_i\rangle$$

so indeed we can compute scalar products giving α'_i and β'_i in terms of x variables. Finally

$$z_{i+1} = Ax_i - b = AC^{-1}y_i - b = AC^{-1}(y_{i-1} + \alpha'_i d'_i) - b$$

$$= \alpha'_i AC^{-1}d'_i + AC^{-1}y_{i-1} - b = \alpha'_i Ad_i + Ax_{i-1} - b$$

$$= z_i + \alpha'_i Ad_i$$

so we get whole iteration.

Note that there is no need to explicitly compute W: all we need is ability to compute Wz_k which we can do by solving equations.

1.4 Nonquadratic conjugate gradient methods

When f is not a quadratic function, then there are several possible ways to generalize conjugate gradient method. First, note that our $r_j = \nabla f(x_{j-1})$. As before we use $x_i = x_{i-1} + \alpha_i d_i$ with $d_1 = -r_1$ and $d_{i+1} = -r_{i+1} + \beta_i d_i$. To choose search direction, we need rule for β_i. Two popular choices are Fletcher-Rieves formula

$$\beta_i = \frac{\|\nabla f(x_i)\|^2}{\|\nabla f(x_{i-1})\|^2}$$

and Polak-Ribiere formula

$$\beta_i = \frac{\langle \nabla f(x_i), \nabla f(x_i) - \nabla f(x_{i-1})\rangle}{\|\nabla f(x_{i-1})\|^2}.$$

Both formulas give the same results for quadratic functions, but in practice Polak-Ribiere formula seem to give better results for nonquadratic ones. To determine α_i we need to perform line search. Backtracking line search would give bad results for quadratic functions, so we need better one. Evaluating f at 3 points on line and using quadratic interpolation to find approximate minimum gives exact result for quadratic functions and seem to behave well in general.

For nonquadratic functions search directions are no longer orthogonal and after large number of steps may be quite suboptimal. Usual way to handle this is to use restarts, that is from time to time take $\beta_i = 0$.

Close to optimal point one can show that conjugate gradient method with restarts after n steps (where n is dimension of the space) behaves similar to Newton method, more precisely n steps of conjugate gradient method do similar work as one step of Newton method.

This is easy to see at intuitive level: close to optimal point our function behaves very similar to quadratic function and we could use conjugate gradient to compute $f''(x_i)^{-1}\nabla f(x_i)$. Using f gives slightly different result, but resulting convergence as fast as Newton method (but we need n steps of conjugate gradient to perform one Newton step).

When f'' is well conditioned, then more frequent restarts lead to faster convergence: method behaves like Newton method with approximate inverse of f''.

2 Quasi-Newton methods

Last time we said that conjugate direction methods have cost slightly larger than gradient descent, but should converge faster. They are in this sense intermediate between gradient descent and Newton method.

Quasi-Newton methods were invented earlier than nonquadratic conjugate gradient method. Historically they were the first approach to construct method which tried to converge faster than gradient descent but have lower per iteration cost than Newton method. They try to get good convergence by approximating second derivative using differences of gradients.

Core idea of quasi-Newton methods is to use descent

$$x_{i+1} = x_i + \alpha_i d_i$$

in the search direction d_i given by

$$d_i = -S_i \nabla f(x_i).$$

where S_i is a strictly positive definite matrix. When $S_i = I$, this gives gradient descent. When $S_i = (\nabla^2 f(x_i))^{-1}$, this gives Newton method. Fixed S_i gives preconditioned gradient descent. When $S_i = (\nabla^2 f(x_0))^{-1}$ we get modified Newton method. In general, when S_i approximates $(\nabla^2 f)^{-1}$, then we expect better convergence.

Another view is that we use S_i^{-1} as a new metric. Therefore the first name used for such methods was *variable metric methods*.

Viewing quasi-Newton methods as gradient descent with new metric we can get convergence estimate. First, since d_i are descent direction we see that quasi-Newton methods are descent methods, so values of goal function are nonincreasing. Second, write $S_i = Q_i^T Q_i$ and $A(x) = Q_i^T \nabla^2 f(x) Q_i$. Now using $\langle S_i^{-1} x, y\rangle = \langle Q_i^{-1} x, Q_i^{-1} y\rangle$ as a scalar product we see that $A(x)$ gives Hessian

matrix corresponding to this scalar product. So speed of convergence depends on conditioning of $A(x)$. In particular, when $A(x)$ is well conditioned we get convergence to stationary point.

More precisely we have the following:

Lemma 2.1 *Assume that $mI \leq A(x) \leq MI$, and x_{i+1} uses step $\frac{2}{M+m}$ (or exact line search), then*

$$f(x_{i+1}) - f(x_\infty) \leq C(f(x_i) - f(x_\infty))$$

where $C = 1 - \frac{m}{M}$.

Note: this is lemma from Lecture 6 applied to metric above. Using inexact line search we get similar result with slightly larger C.

Note: We can compute m and M from eigenvalues of $S_i^{-1}\nabla^2 f(x)$. In particular m and M depend only on S_i and f and are independent of Q_i.

How to find reasonable S_k? Write $p_i = x_{i+1} - x_i$, $g_i = \nabla f(x_i)$, $q_i = g_{i+1} - g_i$. For quadratic f we have

$$q_i = \nabla^2 f(x_i) p_i.$$

So reasonable condition is

$$S_{i+1} q_i = p_i.$$

Equivalently when $B_i = S_i^{-1}$:

$$q_i = B_{i+1} p_i.$$

Those equations are called quasi-Newton equations (or secant equations).

Lemma 2.2 *Let A be positive definite matrix and $f(x) = \frac{1}{2}\langle Ax, x\rangle - \langle b, x\rangle$ be quadratic function. Quasi-Newton method using exact line search with S_i satisfying quasi-Newton equations satisfies*

$$\langle Ad_{i+1}, d_i\rangle = 0.$$

Proof: For quadratic f we have $q_i = Ap_i$. By quasi-Newton equations we have

$$S_{i+1} A p_i = S_{i+1} q_i = p_i.$$

Since line search is exact we have $\langle \nabla f(x_{i+1}), d_i\rangle = 0$.

We have

$$p_{i+1} = x_{i+2} - x_{i+1} = \alpha_{i+1} d_{i+1}$$

and $d_{i+1} = -S_{i+1}\nabla f(x_{i+1})$ so

$$\langle Ap_{i+1}, p_i\rangle = -\alpha_{i+1}\langle AS_{i+1}\nabla f(x_{i+1}), p_i\rangle = -\alpha_{i+1}\langle \nabla f(x_{i+1}), S_{i+1} A p_i\rangle$$

$$= -\alpha_{i+1}\langle \nabla f(x_{i+1}), p_i\rangle = \alpha_{i+1}\alpha_i \langle \nabla f(x_{i+1}), d_i\rangle = 0.$$

In other words

$$\langle Ad_{i+1}, d_i\rangle = 0.$$

□

Quasi-Newton equations specify S_i only in single direction, so there is infinite number of solutions. Other reasonable condition is that S_{i+1} should be close to S_i. One meaning of close is to require $U_i = S_{i+1} - S_i$ to be of low rank. It is possible to find symmetric S_{i+1} such that U_i is of rank 1. However, such rule may produce S_{i+1} which is not positive definite and may lead to numerical difficulties. Next level of complexity is rank 2 update: we require update U_i to be of rank at most 2.

Quasi-Newton equations give

$$p_i = S_{i+1}q_i = U_i q_i + S_i q_i$$

so

$$U_i q_i = p_i - S_i q_i$$

That again admits infinite number of solutions.

Quasi-Newton equations involve $S_i q_i$ and p_i. In "general position" $S_i q_i$ and p_i are linearly independent and it is natural to request that update acts only in this plane and maps orthogonal complement to zero. Still admits one dimensional family of solutions. Simplest possibility is

$$U_k(x) = aS_i q_i \langle S_i q_i, x\rangle + bp_i \langle p_i, x\rangle$$

where a and b are numeric parameters. Then

$$p_i - S_i q_i = aS_i q_i \langle S_i q_i, q_i\rangle + bp_i \langle p_i, q_i\rangle$$

so $a = \frac{-1}{\langle S_i q_i, q_i\rangle}$, $b = \frac{1}{\langle p_i, q_i\rangle}$ and

$$S_{i+1}x = S_i x - \frac{\langle S_i q_i, x\rangle}{\langle S_i q_i, q_i\rangle} S_i q_i + \frac{\langle p_i, x\rangle}{\langle p_i, q_i\rangle} p_i.$$

This is called Davidon-Fletcher-Powell update or in short DFP update.

In quasi-Newton equations S_i and B_i play symmetric role, that is

$$q_i = B_{i+1} p_i.$$

so alternatively to DFP update we can request update to B_i build from q_i and $B_i p_i$. This leads to formula invented independently by Broyden, Fletcher, Goldfarb and Shanno or in short BFGS update:

$$B_{i+1}x = B_i x - \frac{\langle B_i p_i, x\rangle}{\langle B_i p_i, p_i\rangle} B_i p_i + \frac{\langle q_i, x\rangle}{\langle p_i, q_i\rangle} q_i.$$

2.1 Further reading

David G. Luenberger, Yinyu Ye, Linear and Nonlinear Programming, chapter 10.

Jorge Nocedal, Stephen J. Wright, Numerical Optimization, chapter 6 and section 2 of chapter 7.