MATH 923 — Continuous Optimization*

Manfred R. Trummer
Department of Mathematics and Statistics
Simon Fraser University
Burnaby, British Columbia V5A 1S6, Canada
E-mail: trummer@sfu.ca

September 27, 2000

Contents

1 Matrix decompositions

This chapter contains a number of important matrix factorizations. The first two sections are on the LU and Cholesky factorization. This is followed by one section on the QR factorization, and a brief introduction to the Singular Value Decomposition (SVD).

1.1 Gaussian elimination: LU-factorization

This section reviews the process of Gaussian[1] elimination, and translates it into matrix language, which gives rise to the so-called LU-factorization. Gaussian elimination transforms the original system of equations into an equivalent one, i.e., one which has the same set of solutions, by adding multiples of rows to each other. By eliminating variables from the equations, we end up with a "triangular" system, which can be solved simply through substitution. Let us follow the elimination process in a simple example of three equations in three unknowns.

$$
\begin{array}{l|rcrcrcrl}
\hline
\epsilon_1: & 2u & - & 2v & + & 3w & = & 1 & \\
\epsilon_2: & 6u & - & 7v & + & 14w & = & 5 & \\
\epsilon_3: & 4u & - & 8v & + & 30w & = & 14 & \\
\hline
\epsilon_1' = \epsilon_1: & 2u & - & 2v & + & 3w & = & 1 & \\
\epsilon_2' = \epsilon_2 - 3\epsilon_1: & & - & v & + & 5w & = & 2 & \\
\epsilon_3' = \epsilon_3 - 2\epsilon_1: & & - & 4v & + & 24w & = & 12 & \\
\hline
\epsilon_1'' = \epsilon_1': & 2u & - & 2v & + & 3w & = & 1 & \\
\epsilon_2'' = \epsilon_2': & & - & v & + & 5w & = & 2 & \\
\epsilon_3'' = \epsilon_3' - 4\epsilon_2': & & & & & 4w & = & 4 & \\
\hline
\text{Back-} & & & & & w & = & 1 & \\
\text{substitution:} & & - & v & + & 5 & = & 2 & \Rightarrow v = 3 \\
 & 2u & - & 6 & + & 3 & = & 1 & \Rightarrow u = 2 \\
\hline
\end{array}
\tag{1.1}
$$

We will now translate this process into matrix language.

A matrix is a rectangular array of numbers. Our example here works with 3 by 3 matrices, i.e., 3 rows and 3 columns. Actually, it is often best to think of a matrix as a collection of columns (or a collection of rows, depending on the context), rather than a rectangular array of numbers. It

[1] Carl Friedrich Gauss, 1777-1855, one of the greatest mathematicians. The technique of "Gaussian" elimination has been known prior to Gauss.

is also often useful, to think of a matrix B in terms of what it "does" when it acts on a vector x (or even some other matrix C). This is tantamount to interpreting the matrix B as an *operator* effecting the transformation $x \mapsto Bx$. Let us denote the standard basis vectors by $e_1, \ldots, e_k, \ldots$; for vectors with three components we have

$$e_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \qquad e_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \qquad e_3 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}; \tag{1.2}$$

note, that we always write our vectors as column vectors. The matrix which has the standard basis vectors as its columns is called the identity matrix, I,

$$I = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{1.3}$$

The identity matrix leaves vectors and matrices unchanged, i.e.,

$$Ix = x, \qquad IB = B,$$

for all vectors x and matrices B. On the other hand, the standard basis vectors can be used to pick out columns and rows from a matrix:

$$Be_k = k\text{-th column of } B; \qquad e_k^T B = k\text{-th row of } B;$$

the $\cdot^T$ means transpose: it makes row vectors out of column vectors and vice versa. For a matrix, B^T is the matrix which has the rows of B as its columns.

Finally,

$$e_j^T B e_k = b_{jk} = \text{ the element of } B \text{ in row } j \text{ and column } k.$$

We now return to our example. First, our given system of equations (1.1) is

$$\underbrace{\begin{pmatrix} 2 & -2 & 3 \\ 6 & -7 & 14 \\ 4 & -8 & 30 \end{pmatrix}}_{A} \underbrace{\begin{pmatrix} u \\ v \\ w \end{pmatrix}}_{x} = \underbrace{\begin{pmatrix} 1 \\ 5 \\ 14 \end{pmatrix}}_{b}. \tag{1.4}$$

The first step in the elimination process above is to subtract three times the first equation from the second. Therefore, in the coefficient matrix A, we replace the second row by the the difference between it and three times the first row. Of course, the factor 3 is chosen to obtain a 0 in the (2,1)-position of the matrix. One such elimination step corresponds to multiplying the matrix A by the **elementary** matrix E_{21} from the left:

$$\underbrace{\begin{pmatrix} 1 & 0 & 0 \\ -3 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{=:\, E_{21}} \underbrace{\begin{pmatrix} 2 & -2 & 3 \\ 6 & -7 & 14 \\ 4 & -8 & 30 \end{pmatrix}}_{A} = \underbrace{\begin{pmatrix} 2 & -2 & 3 \\ 0 & -1 & 5 \\ 4 & -8 & 30 \end{pmatrix}}_{E_{21}A}. \tag{1.5}$$

An elementary matrix is equal to the identity matrix I, except for one of the off-diagonal elements which is nonzero. Proceeding with our elimination steps we obtain

$$\underbrace{\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -2 & 0 & 1 \end{pmatrix}}_{=:\, E_{31}} \underbrace{\begin{pmatrix} 2 & -2 & 3 \\ 0 & -1 & 5 \\ 4 & -8 & 30 \end{pmatrix}}_{E_{21}A} = \underbrace{\begin{pmatrix} 2 & -2 & 3 \\ 0 & -1 & 5 \\ 0 & -4 & 24 \end{pmatrix}}_{E_{31}E_{21}A}, \tag{1.6}$$

and

$$\underbrace{\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -4 & 1 \end{pmatrix}}_{=:\, E_{32}} \underbrace{\begin{pmatrix} 2 & -2 & 3 \\ 0 & -1 & 5 \\ 0 & -4 & 24 \end{pmatrix}}_{E_{31}E_{21}A} = \underbrace{\begin{pmatrix} 2 & -2 & 3 \\ 0 & -1 & 5 \\ 0 & 0 & 4 \end{pmatrix}}_{E_{32}E_{31}E_{21}A \,=:\, U}. \tag{1.7}$$

Thus, our "final" matrix U is upper triangular, and corresponds to the coefficients of the final set of equations $(\epsilon_1'', \epsilon_2'', \epsilon_3'')$ in (1.1). In your linear algebra course this may have been referred to as the *row echelon form* of the matrix A. The same matrices need to be applied to the vector b, to obtain our equivalent system of equations. Note that the three elementary matrices E_{21}, E_{31}, and E_{32} contain the information about how we proceeded in the process of Gaussian elimination; they are like entries in a bookkeeping journal. Therefore, it should be possible to reconstruct the original matrix A from U and the bookkeeping matrices E_{21}, E_{31}, and E_{32}. From (1.7) we have

$$\begin{aligned} E_{32}E_{31}E_{21}A &= U \\ E_{31}E_{21}A &= E_{32}^{-1}U \\ E_{21}A &= E_{31}^{-1}E_{32}^{-1}U \\ A &= \underbrace{E_{21}^{-1}E_{31}^{-1}E_{32}^{-1}}_{=:\, L}U. \end{aligned} \tag{1.8}$$

This is the formula for reconstructing the matrix A. To find L, we need to find the inverses of elementary matrices. This is easy to do, if we remember the meaning of these elementary matrices. For example, E_{21} when applied to a matrix from the left, *subtracts* three times the first row from the second row. Clearly, the inverse operation (undoing what we have just done) is to *add* three times the first row to the second row.

$$E_{21} = \begin{pmatrix} 1 & 0 & 0 \\ -3 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad \Rightarrow \qquad E_{21}^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ +3 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}; \tag{1.9}$$

so inverses of elementary matrices are simply obtained by changing the sign of the nonzero off-diagonal element. Now, another interesting phenomenon occurs, when we perform the matrix multiplication to obtain L:

$$\begin{aligned} L &= E_{21}^{-1} E_{31}^{-1} E_{32}^{-1} \\ &= \begin{pmatrix} 1 & 0 & 0 \\ 3 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 2 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 4 & 1 \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 & 0 \\ 3 & 1 & 0 \\ 2 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 4 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 3 & 1 & 0 \\ 2 & 4 & 1 \end{pmatrix}, \end{aligned} \tag{1.10}$$

so the matrix multiplication can be performed by just putting the nonzero off-diagonal elements of the elementary matrices into the appropriate positions in the matrix L. From an algorithmic point of view this means that the entries of L, which are just the multipliers in the Gaussian elimination process, can be easily stored during the process of Gaussian elimination. Please note, that the order in which those elementary matrices appear is crucial, as

$$L^{-1} = E_{32} E_{31} E_{21} = \begin{pmatrix} 1 & 0 & 0 \\ -3 & 1 & 0 \\ 10 & -4 & 1 \end{pmatrix} \neq \begin{pmatrix} 1 & 0 & 0 \\ -3 & 1 & 0 \\ -2 & -4 & 1 \end{pmatrix}. \tag{1.11}$$

Finally, putting everything together we obtain

$$\underbrace{\begin{pmatrix} 2 & -2 & 3 \\ 6 & -7 & 14 \\ 4 & -8 & 30 \end{pmatrix}}_{A} = \underbrace{\begin{pmatrix} 1 & 0 & 0 \\ 3 & 1 & 0 \\ 2 & 4 & 1 \end{pmatrix}}_{L} \underbrace{\begin{pmatrix} 2 & -2 & 3 \\ 0 & -1 & 5 \\ 0 & 0 & 4 \end{pmatrix}}_{U}, \tag{1.12}$$

i.e., Gaussian elimination gives us the LU-**factorization** (sometimes also called the LU-**decomposition**) of the matrix A, $A = LU$, where L is a lower triangular matrix with all diagonal entries equal to 1, and U is an upper triangular matrix.

Once we have this factorization, how can we make use of it to solve $Ax = b$? It is easy to see that

$$Ax = LUx = b$$

is equivalent to solving

$$Ux = L^{-1}b =: c.$$

In general, we do not want to compute the inverse of L. Instead, we can find $c = L^{-1}b$ through the process called *forward substitution*,

$$Lc = b;$$

since L is triangular this is as easy to do as the final step, namely *backward substitution*,

$$Ux = c.$$

These two substitution processes are no more costly than computing $A^{-1}b$, given that A^{-1} is known, so in that sense knowing L and U is as good for solving systems as knowing the inverse. Once the factorization is obtained, it is very cheap to solve many systems with the same matrix A and different right hand sides.

Proposition 1.1 *If Gaussian elimination does not break down, i.e., if all pivots (*the leading diagonal elements at each step*) are nonzero, then the matrix A has a unique LU-factorization, where L is a lower triangular matrix with all diagonal entries equal to* 1*, and U is an upper triangular matrix.*

So what if Gaussian elimination "breaks down", i.e., if there is no LU factorization of our matrix A? For example, the matrix

$$A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

does not have an LU factorization, even though the corresponding system of equations poses no difficulties. Indeed, if we simply switch the two rows (thus obtaining an equivalent system), the resulting matrix does have an LU factorization,

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = LU,$$

where $L = U = I$, and I denotes the identy matrix of appropriate size. Usually, it is less obvious but no more difficult to fix the breakdown. For example, applying Gaussian elimination to

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & -1 \\ 2 & -1 & -1 \end{pmatrix}$$

after two steps results into

$$\begin{pmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ -2 & 0 & -1 \end{pmatrix} A = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & -2 \\ 0 & -3 & -3 \end{pmatrix};$$

to proceed we must switch rows 2 and 3, to move a nonzero element to the current diagonal position (2,2). How can switching rows be expressed in matrix language? Again, by matrix multiplication from the left[2], this time with a **permutation** matrix,

$$P = P_{23} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}.$$

Thus, we obtain

$$\begin{aligned} PA &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & -1 \\ 2 & -1 & -1 \end{pmatrix} \\ = \begin{pmatrix} 1 & 1 & 1 \\ 2 & -1 & -1 \\ 1 & 1 & -1 \end{pmatrix} &= \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 \\ 0 & -3 & -3 \\ 0 & 0 & -2 \end{pmatrix}, \end{aligned}$$

i.e., the matrix PA does have an LU-factorization. Of course, for larger matrices, more of those row exchanges may be necessary, which leads to a more complicated permutation of the rows in the original matrix A, but can easily be handled by good "bookkeeping".

Our strategy consists in switching rows to obtain a nonzero "pivot" (the current diagonal element). Naturally, this is not possible, if all the elements in the present column are zero. But in that case, we may just simply ignore

[2]multiplying from the left will operate on the rows of a matrix, multiplying from the right will operate on the columns of a matrix

the present column and move on to the next one. However, this scenario means that our matrix A is singular, i.e., $Ax = b$ has either no solution, or infinitely many. Although we will ignore this case in the following, we give an example.

$$
\begin{aligned}
A = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 2 & 1 \\ 1 & 1 & 2 & 2 \\ 1 & 1 & 3 & 0 \end{pmatrix} &\rightarrow \begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 2 & 0 \end{pmatrix} \rightarrow \\
\rightarrow \begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -2 \end{pmatrix} &\rightarrow \begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} = U,
\end{aligned}
$$

with

$$
L = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 2 & -2 & 1 \end{pmatrix}.
$$

The resulting upper triangular matrix U has a whole row of zeros, clearly indicating singularity.

Summing up, the main result of this section is

Theorem 1.2 (LU-factorization) *For every n by n matrix A there exists a permutation matrix P, such that PA possesses an LU-factorization, i.e., $PA = LU$, where L is a lower triangular matrix with all diagonal entries equal to 1, and U is an upper triangular matrix.*

In an actual computation, one must switch rows often, even when the pivot is not equal to zero. A good method to ensure an accurate answer is to always switch rows, such that the largest element in absolute value in the present pivot column (all elements below and including the diagonal element) becomes the new pivot. Thus, we always work with the factorization $PA = LU$.

1.2 Cholesky factorization

We will briefly treat the special case of a symmetric matrix $A = A^T$. If we perform the LU-factorization as usual, the symmetry of A is no longer apparent. For example,

$$A = \begin{pmatrix} 9 & -9 & 9 \\ -9 & 10 & -10 \\ 9 & -10 & 14 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 1 & -1 & 1 \end{pmatrix} \begin{pmatrix} 9 & -9 & 9 \\ 0 & 1 & -1 \\ 0 & 0 & 4 \end{pmatrix},$$

and from L and U it is not immediately obvious that A is symmetric. However, we can divide each row of U by its diagonal element, to obtain the LDL^T-factorization,

$$A = \begin{pmatrix} 9 & -9 & 9 \\ -9 & 10 & -10 \\ 9 & -10 & 14 \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 1 & -1 & 1 \end{pmatrix}}_{L} \underbrace{\begin{pmatrix} 9 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 4 \end{pmatrix}}_{D} \underbrace{\begin{pmatrix} 1 & -1 & 1 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{pmatrix}}_{L^T}.$$

Preserving symmetry is usually a worthwhile undertaking, and not only because it is aesthetically more pleasing.

If all the (diagonal) elements of D are positive, we can proceed even further, by taking the "square root" of D. In our numerical example we get

$$A = LD^{1/2}D^{1/2}L^T, \qquad \text{with } D^{1/2} = \begin{pmatrix} 3 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{pmatrix}.$$

Therefore, if we set $R = D^{1/2}L^T$, we obtain

$$\begin{aligned} A &= (LD^{1/2})D^{1/2}L^T \\ &= (D^{1/2}L^T)^T D^{1/2}L^T = R^T R \end{aligned}$$

with R an upper triangular matrix,

$$R = \begin{pmatrix} 3 & -3 & 3 \\ 0 & 1 & -1 \\ 0 & 0 & 2 \end{pmatrix}.$$

Note, that the rows of R are simply the rows of L^T scaled by the square roots of the diagonal elements of D.

This factorization is called the **Cholesky** factorization, $A = R^T R$. It requires A to be symmetric, and **positive definite**. A symmetric matrix is positive definite, if and only if

- $x^T Ax > 0$ for all vectors $x \neq 0$.

- all pivots in Gaussian elimination are (strictly) positive.
- all eigenvalues of A are (strictly) positive.
- all diagonal elements of D in the factorization $A = LDL^T$ are (strictly) positive.
- A can be written as $A = R^T R$, with R a nonsingular upper triangular matrix.

This list gives a number of equivalent criteria; if one criterion is satisfied, than so are all others.

Symmetric positive definite matrices do occur frequently in applications. The Cholesky factorization of such a matrix can be computed twice as fast as the LU factorization, i.e., only $n^3/6$ multiplications and additions (total: $n^3/3$ flops) are required. It is also unnecessary to switch rows. The algorithm itself is straightforward; it is based on finding the entries of R row by row. By comparing elements

$$\underbrace{\begin{pmatrix} r_{11} & 0 & 0 & 0 & \cdots \\ r_{12} & r_{22} & 0 & 0 & \cdots \\ r_{13} & r_{23} & r_{33} & 0 & \cdots \\ r_{14} & r_{24} & r_{34} & r_{44} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}}_{R^T} \underbrace{\begin{pmatrix} r_{11} & r_{12} & r_{13} & r_{14} & \cdots \\ 0 & r_{22} & r_{23} & r_{24} & \cdots \\ 0 & 0 & r_{33} & r_{34} & \cdots \\ 0 & 0 & 0 & r_{44} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}}_{R}$$

$$= \underbrace{\begin{pmatrix} r_{11}^2 & r_{11}r_{12} & r_{11}r_{13} & r_{11}r_{14} & \cdots \\ \star & r_{12}^2 + r_{22}^2 & r_{12}r_{13} + r_{22}r_{23} & r_{12}r_{14} + r_{22}r_{24} & \cdots \\ \star & \star & r_{13}^2 + r_{23}^2 + r_{33}^2 & r_{13}r_{14} + r_{23}r_{24} + r_{33}r_{34} & \cdots \\ \star & \star & \star & r_{14}^2 + r_{24}^2 + r_{34}^2 + r_{44}^2 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}}_{R^T R = A},$$

we find

$$\begin{aligned} a_{kk} &= r_{kk}^2 + \sum_{j=1}^{k-1} r_{jk}^2, \\ a_{mk} = a_{km} &= r_{kk} r_{km} + \sum_{j=1}^{k-1} r_{jk} r_{jm}, \qquad \text{for } k < m. \end{aligned}$$

This leads to the algorithm

$$
\begin{aligned}
\text{for} \quad & k = 1 : n, \\
& r_{kk} = \quad \sqrt{a_{kk} - \sum_{j=1}^{k-1} r_{jk}^2}, \\
& \text{for} \quad m = k+1 : n, \\
& \qquad r_{km} = \frac{a_{km} - \sum_{j=1}^{k-1} r_{jk} r_{jm}}{r_{kk}}.
\end{aligned}
$$

The square root operation will pose no problem, as long as A is positive definite. Once the factorization is obtained, the system $Ax = b$ is solved by forward and back substitution with the matrices R^T and R.

Example 1.3 *The matrix*

$$
A = \begin{pmatrix}
2 & -1 & 0 & 0 & \cdots & & \\
-1 & 2 & -1 & 0 & \ddots & & \\
0 & -1 & 2 & -1 & \ddots & & \\
\vdots & \ddots & \ddots & \ddots & \ddots & & \\
\vdots & \ddots & \ddots & \ddots & \ddots & & \\
\cdots & & 0 & & -1 & 2 & -1 \\
\cdots & & & & 0 & -1 & 2
\end{pmatrix}
$$

arises in the finite difference approximation of a second derivative of a function. It is positive definite, and "banded" (tridiagonal). The matrix R of its Cholesky factorization is also banded (bidiagonal).

1.3 Gram-Schmidt – QR factorization

Every n by n matrix A has a QR-factorization, $A = QR$, where Q is an orthogonal matrix (i.e., $Q^T Q = I$), and R is an upper triangular matrix. To solve a sytem of equations, $Ax = b$, one simply has to backsubstitute $Rx = Q^T b$. The QR factorization also plays a prominent role in the numerical computation of eigenvalues.

Example 1.4 *For the matrix A below we have the QR-factorization*

$$\underbrace{\begin{pmatrix} 3 & 0 & 2 & 2 \\ 3 & 0 & -2 & 0 \\ 3 & -2 & 0 & -1 \\ 3 & -2 & 4 & -1 \end{pmatrix}}_{A} = \underbrace{\begin{pmatrix} -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \end{pmatrix}}_{Q} \underbrace{\begin{pmatrix} -6 & 2 & -2 & 0 \\ 0 & -2 & 2 & -2 \\ 0 & 0 & 4 & 1 \\ 0 & 0 & 0 & -1 \end{pmatrix}}_{R}$$

Note that the QR factorization is not unique, for example,

$$A = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} \end{pmatrix} \begin{pmatrix} 6 & -2 & 2 & 0 \\ 0 & 2 & -2 & 2 \\ 0 & 0 & 4 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

i.e., we can switch the signs in each of the columns of Q (we could enforce uniqueness for nonsingular A by requiring $r_{kk} > 0$). For

$$b = \begin{pmatrix} 7 \\ 5 \\ -4 \\ -8 \end{pmatrix} \qquad \textit{we have} \qquad Q^T b = \begin{pmatrix} 0 \\ -12 \\ -1 \\ -3 \end{pmatrix},$$

and the system

$$Rx = \begin{pmatrix} -6 & 2 & -2 & 0 \\ 0 & -2 & 2 & -2 \\ 0 & 0 & 4 & 1 \\ 0 & 0 & 0 & -1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 0 \\ -12 \\ -1 \\ -3 \end{pmatrix} = Q^T b,$$

has the solution $x = [1 \;\; 2 \;\; -1 \;\; 3]^T$.

We now allow the matrix A to be rectangular, with at least as many rows as columns. That is, A is an m by n matrix with $m \geq n$. We briefly describe the Gram-Schmidt process to compute the QR factorization. It is based on orthogonalization of a basis: The first ℓ columns of Q form an orthonormal basis (the vectors are orthogonal to each other, and of length 1) for the space spanned by the first ℓ column vectors of A. The Gram-Schmidt method has the advantage of being more economical when computing the "reduced" factorization of an m by n matrix A, with $m > n$. Such matrices correspond to systems with more equations than unknowns, and can usually only be solved as least squares problems.

An m by n matrix A with $m > n$ has the full (QR) and reduced ($\bar{Q}\bar{R}$) factorizations

$$\underbrace{\begin{pmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \vdots & \vdots & \vdots \\ \times & \times & \times \end{pmatrix}}_{A} = \underbrace{\begin{pmatrix} \times & \times & \times & \cdots & \times \\ \times & \times & \times & \cdots & \times \\ \times & \times & \times & \cdots & \times \\ \times & \times & \times & \cdots & \times \\ \times & \times & \times & \cdots & \times \\ \vdots & \vdots & \vdots & \ldots & \vdots \\ \times & \times & \times & \cdots & \times \end{pmatrix}}_{Q} \underbrace{\begin{pmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \vdots & \vdots & \vdots \\ 0 & 0 & 0 \end{pmatrix}}_{R} = \underbrace{\begin{pmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \vdots & \vdots & \vdots \\ \times & \times & \times \end{pmatrix}}_{\bar{Q}} \underbrace{\begin{pmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \end{pmatrix}}_{\bar{R}}.$$

As indicated, Q is an m by m (big) orthogonal matrix, R is an m by n "trapezoidal" matrix, the m by n matrix $\bar{Q}$ consists of the first n columns of Q, and the n by n (small) triangular matrix $\bar{R}$ consists of the first n rows of R. The last $m-n+1$ rows of R are all equal to zero, therefore the last $m-n+1$ columns of Q are irrelevant when we compute the product QR. The reduced factorization contains all the information necessary to reconstruct A. The Gram-Schmidt method computes the matrix $\bar{Q}$ column by column from the columns of A, and produces the reduced factorization. To obtain Q, either Householder reflections or Givens rotations are used. Let

$$A = \left(\begin{array}{c|c|c|c|c} | & | & & & | \\ a_1 & a_2 & \cdots & \cdots & a_n \\ | & | & & & | \end{array} \right), \qquad a_j \in \mathbb{R}^m.$$

Classical Gram-Schmidt algorithm:

for $j = 1$ to n
  $u_j = a_j$
for $i = 1$ to $j-1$
  $r_{ij} = q_i^T a_j$
  $u_j = u_j - r_{ij} q_i$
$r_{jj} = ||u_j|| = \sqrt{u_j^T u_j}$
$q_j = u_j / r_{jj}$

Note that the vector u_j is chosen to be orthogonal to the vectors q_1, ..., q_{j-1}. The vectors q_j are the columns of the matrix $\bar{Q}$. The elements r_{ij} of the matrix R are computed column by column.

It turns out, that this algorithm is not numerically stable. Instead, the mathematically equivalent *Modified Gram-Schmidt* method is recommended. It simply rearranges the computation. After a vector q_j is computed, all the elements r_{jk} in row j of the matrix R are computed, i.e., the elements of R are computed row by row. Further, the components of all the vectors a_k with $k > j$ in the direction of q_j are subtracted from the a_j.

Modified Gram-Schmidt algorithm:

for $j = 1$ to n
    $u_j = a_j$
for $j = 1$ to n
    $r_{jj} = ||u_j|| = \sqrt{u_j^T u_j}$
    $q_j = u_j / r_{jj}$
    for $k = j + 1$ to n
        $r_{jk} = q_j^T u_k$
        $u_k = u_k - r_{jk} q_j$

1.4 Least squares problem, Generalized inverse

The matrix A is an m by n matrix, with $m \geq n$. With more equations than unknowns we cannot expect $Ax = b$ to have a solution. Instead we look for a vector v which minimizes the length of the residual $r = b - Av$. The vector v solves the **least squares problem**

$$\min ||b - Ax|| \tag{1.13}$$

if and only if v is a solution to the normal equations

$$A^T Ax = A^T b. \tag{1.14}$$

These equations are derived by observing that for the optimal solution v, the residual $r = b - Av$ must be orthogonal to the column space of A ($= R(A) =$ the space spanned by the column vectors of A). But any vector orthogonal to $R(A)$ must be in $N(A^T)$, the null space of A^T (the set of vectors y for which $A^T y = 0$). This is one of the fundamental results of linear algebra.

The least squares solution v can also be computed using the (reduced) QR-factorization of A, and solving the triangular system

$$\bar{R}x = \bar{Q}^T b. \tag{1.15}$$

In many situations, (1.15) gives results superior to the ones obtained by (1.14), although mathematically the two methods are equivalent.

Example 1.5 *With* $Q =$

$$\frac{1}{4}\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \end{pmatrix}^T$$

and

$$R = \begin{pmatrix} 10.0 & 1.0000 \\ 0 & 0.0001 \end{pmatrix},$$

we obtain the matrix $A = QR =$

$$\begin{pmatrix} 2.5 & 2.5 & 2.5 & 2.5 & \cdots & \cdots & 2.5 & 2.5 \\ .250025 & .249975 & .250025 & .249975 & \cdots & \cdots & .250025 & .249975 \end{pmatrix}^T.$$

For

$$x = \begin{pmatrix} 5 \\ 3 \end{pmatrix}$$

we set $b = Ax$. *The components of* b *are* $b_k = 13.250075$ *for odd* k, *and* $b_k = 13.249925$ *for even* k. *The solution* x^{NE} *computed via the normal equations (1.14), and the solution* x^{QR} *obtained by (1.15) are*

$$x^{NE} = \begin{pmatrix} 5.00000018651747 \\ 2.99999813482531 \end{pmatrix}, \quad \textit{and} \quad x^{QR} = \begin{pmatrix} 4.99999999999971 \\ 3.00000000000289 \end{pmatrix},$$

with errors $||x - x^{NE}|| \approx 1.875 \star 10^{-6}$, *and* $||x - x^{QR}|| \approx 2.9 \star 10^{-12}$, *respectively. The computations are performed in double precision (machine epsilon* $\approx 2.22 \star 10^{-16}$*). Note that the computed* QR *factors* $\tilde{Q}$ *and* $\tilde{R}$ *differ* *from our given matrices Q and R:*

$$\tilde{Q} = \begin{pmatrix} -0.25 & 0.24999999999922 \\ -0.25 & -0.25000000000000 \\ -0.25 & 0.25000000000011 \\ -0.25 & -0.25000000000000 \\ -0.25 & 0.25000000000011 \\ -0.25 & -0.25000000000000 \\ -0.25 & 0.25000000000011 \\ -0.25 & -0.25000000000000 \\ \vdots & \vdots \\ \vdots & \vdots \\ -0.25 & 0.25000000000011 \\ -0.25 & -0.25000000000000 \end{pmatrix}, \qquad \tilde{R} = \begin{pmatrix} -10.0 & -1.0000 \\ 0 & 0.0001 \end{pmatrix}.$$

The relative errors in the second column of $\tilde{Q}$ are larger than 10^{-12} for the first element, and larger than 10^{-13} for the other odd numbered elements, but even though, the product $\tilde{Q}\tilde{R}$ differs from A by less then 10^{-15}.

To prove that $x = \bar{R}^{-1}\bar{Q}^T b$ is indeed the least squares solution, we note that

$$\begin{aligned} ||Ax - b||^2 &= ||Q^T(Ax - b)||^2 = ||Rx - Q^T b||^2 \\ &= ||\bar{R}x - \bar{Q}^T b||^2 + ||Q_2^T b||^2, \end{aligned}$$

where Q_2 denotes the m by $(m-n)$ matrix containing the last $m-n$ columns of Q. Clearly this is minimized, if $\bar{R}x = \bar{Q}^T b$.

The above applies to the full rank case, i.e., $\operatorname{rank}(A) = n$, which in turn implies that the matrix $A^T A$ (and the matrix $\bar{R}$) is nonsingular. For the rank-deficient case (and for the case $m < n$), we must tackle the problem of non-uniqueness of the (least-squares) solution to $Ax = b$.

Definition 1.6 *Let A be an m by n matrix. The minimum norm (least squares) solution u of $Ax = b$ is the (least squares) solution of $Ax = b$ of minimum norm. We denote u by $u = A^\dagger b$.*

Proposition 1.7 (i) *The minimum norm solution $u = A^\dagger b$ is the unique least squares solution of $Ax = b$ in $N(A)^\perp = R(A^T)$.*

(ii) $A^\dagger$ *is an* n *by* m *matrix, the generalized inverse of* A*, satisfying the properties*

(a) $AA^\dagger A = A$.

(b) $A^\dagger AA^\dagger = A^\dagger$.

(c) $AA^\dagger = P_{R(A)}$*, the orthogonal projection of* $\mathbb{R}^m$ *onto* $R(A)$.

(d) $A^\dagger A = P_{R(A^T)}$*, the orthogonal projection of* $\mathbb{R}^n$ *onto* $R(A^T) = N(A)^\perp$.

(e) $R(A^\dagger) = R(A^T) = N(A)^\perp$*, and* $N(A^\dagger) = R(A)^\perp = N(A^T)$.

If $m \geq n$, and $\mathrm{rank}(A) = n$, then $A^\dagger = (A^TA)^{-1}A^T$. To deal with the general case, it is best to work with the singular value decomposition.

1.5 Singular value decomposition

The matrix A is again an m by n matrix, with $m \geq n$ (everything can be done for $m < n$ as well, with obvious modifications).

Theorem 1.8 (SVD) *Every* m *by* n *matrix* A $(m \geq n)$ *has a singular value decomposition (SVD)*

$$A = U\Sigma V^T = \bar{U}\bar{\Sigma}V^T.$$

The matrix U *is an* m *by* m *orthogonal matrix,* V *is an* n *by* n *orthogonal matrix,* Σ *is an* m *by* n *matrix with all elements equal to zero except for the diagonal elements* $\Sigma_{ii} = \sigma_i \geq 0$, $1 \leq i \leq n$*, the singular values of* A $(\sigma_1 \geq \sigma_2 \geq \ldots \sigma_n \geq 0$*; this ordering ensures "uniqueness" of the SVD). The matrix* $\bar{U}$ *consists of the first* n *columns of* U*, and the matrix* $\bar{\Sigma}$ *is the* n *by* n *diagonal matrix consisting of the first* n *rows of* Σ. $A = U\Sigma V^T$ *is the full SVD, and* $A = \bar{U}\bar{\Sigma}V^T$ *is the reduced SVD.*

The column vectors u_j of U are the left singular vectors of A, and the column vectors of V are the right singular vectors of A. We have the following relations:

Proposition 1.9 (i) $Av_j = \sigma_j u_j$, $1 \leq j \leq n$.

(ii) $A^T u_j = \sigma_j v_j,\ \ 1 \le j \le n,$ *and* $A^T u_j = 0,\ \ j > n.$

(iii) $A^T A V = V\bar{\Sigma}^2$, *i.e., the* v_j *are the eigenvectors of* $A^T A$ *with corresponding eigenvalues* $\sigma_1^2, \ldots, \sigma_n^2$.

(iv) *The* u_j *are the eigenvectors of* AA^T *with corresponding eigenvalues* $\sigma_1^2, \ldots, \sigma_n^2, \underbrace{0, 0, \ldots, 0}_{(m-n)\ \textit{times}}$.

The matrix A has rank r if and only if

$$\sigma_1 \ge \sigma_2 \ge \ldots \ge \sigma_r > 0, \qquad \sigma_{r+1} = \sigma_{r+2} = \ldots = \sigma_n = 0.$$

The generalized inverse of

$$\Sigma = \begin{pmatrix} \sigma_1 & 0 & 0 & 0 \\ 0 & \sigma_2 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \sigma_n \\ 0 & 0 & 0 & 0 \\ \vdots & & & \vdots \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

is simply given by

$$\Sigma^\dagger = \begin{pmatrix} \sigma_1^\dagger & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & \sigma_2^\dagger & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & \ddots & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & \sigma_n^\dagger & 0 & \cdots & 0 \end{pmatrix},$$

with

$$\sigma_j^\dagger = \begin{cases} 1/\sigma_j, & \text{for } \sigma_j \ne 0, \text{ i.e., } j \le r, \\ 0, & \text{for } \sigma_j = 0, \text{ i.e., } j > r. \end{cases}$$

Once we have the generalized inverse of Σ, we can easily find the generalized inverse of A:

$$A^\dagger = V\Sigma^\dagger U^T.$$

Thus, the minimum norm least squares solution to $Ax = b$ is

$$x = A^\dagger b = V\Sigma^\dagger U^T b = V\bar{\Sigma}^\dagger \bar{U}^T b = \sum_{j=1}^{r} \frac{u_j^T b}{\sigma_j} v_j.$$

Note that

$$
\begin{aligned}
R(A) &= \operatorname{span}\{u_1, \ldots, u_r\} \\
N(A) &= \operatorname{span}\{v_{r+1}, \ldots, v_n\} \\
R(A^T) &= \operatorname{span}\{v_1, \ldots, v_r\} = N(A)^{\perp} = R(A^{\dagger}) \\
N(A^T) &= \operatorname{span}\{u_{r+1}, \ldots, u_m\} = R(A)^{\perp} = N(A^{\dagger}).
\end{aligned}
$$

2 Descent methods

A typical algorithm to solve the minimization problem

$$\min\{f(x) : x \in \mathbb{R}^n\} \tag{2.1}$$

has the following structure:

Start with initial guess $x_0 \in \mathbb{R}^n$
for $k = 0, 1, 2, \ldots,$
- Step 1. Compute $f(x_k)$ (if not already done). Decide whether to stop or to continue.
- Step 2. Choose (descent) direction p_k.
- Step 3. minimize $f(x_k + \alpha p_k)$, set α_k to the minimizer. Decide whether α_k is acceptable, modify α_k if necessary.
- Step 4. Update: $x_{k+1} = x_k + \alpha_k p_k$.

2.1 Steepest descent

Choose $p_k = -\nabla f(x_k)$, i.e., the direction of steepest descent.

For the quadratic case,

$$f(x) = \frac{1}{2} x^T Q x - x^T b, \tag{2.2}$$

with Q a positive definite matrix, we have

$$\nabla f(x) = Qx - b, \qquad g_k := \nabla f(x_k) = Qx_k - b \tag{2.3}$$

$$\nabla^2 f(x) = Q, \qquad H_k := \nabla^2 f(x_k) = Q. \tag{2.4}$$

A convenient way to measure the error:

$$E(x) = \frac{1}{2}(x - z)^T Q (x - z) = f(x) + \frac{1}{2} z^T Q z, \tag{2.5}$$

where z is the minimizer of f, i.e, $z = Q^{-1} b$. Note, that $(x^T Q x)^{1/2}$ is a **norm** for any positive definite matrix Q.

The parameter α_k can be explicitly determined for the quadratic case.

$$\begin{aligned}
f(x_k - \alpha g_k) &= \frac{1}{2}(x_k - \alpha g_k)^T Q (x_k - \alpha g_k) - (x_k - \alpha g_k)^T b \\
&= \underbrace{\frac{1}{2} x_k^T Q x_k - x_k^T b}_{= f(x_k)} - \alpha g_k^T Q x_k + \frac{1}{2}\alpha^2 g_k^T Q g_k + \alpha g_k^T b \\
&= \text{const} + \frac{1}{2}\alpha^2 g_k^T Q g_k - \alpha g_k^T \underbrace{(Q x_k - b)}_{= g_k};
\end{aligned}$$

the minimum on the line $x_k - \alpha g_k$ is attained for

$$\alpha = \alpha_k = \frac{g_k^T g_k}{g_k^T Q g_k}. \tag{2.6}$$

Proof of convergence:

$$\begin{aligned}
f(x_k) - f(x_{k+1}) &= -\frac{1}{2}\alpha_k^2 g_k^T Q g_k + \alpha_k g_k^T g_k \\
&= \frac{1}{2}\alpha_k g_k^T g_k = \frac{1}{2}\frac{(g_k^T g_k)^2}{g_k^T Q g_k}.
\end{aligned}$$

Setting

$$e_k = x_k - z \qquad \Rightarrow \quad g_k = Q e_k,$$

and using that

$$f(x_k) - f(x_{k+1}) = E(x_k) - E(x_{k+1}),$$

and

$$E(x_k) = \frac{1}{2} e_k^T Q e_k = \frac{1}{2} g_k^T Q^{-1} g_k,$$

we obtain

$$E(x_{k+1}) = \left\{ 1 - \frac{(g_k^T g_k)^2}{(g_k^T Q g_k)(g_k^T Q^{-1} g_k)} \right\} E(x_k). \tag{2.7}$$

This shows $E(x_{k+1}) < E(x_k)$, but does not yet prove $E(x_k) \to 0$. Since Q is a positive definite matrix, its eigenvalues λ_1, ..., λ_n are all positive,

$$0 < m = \lambda_1 \leq \ldots \leq \lambda_n = M,$$

and

$$m = \min_{||x||=1} x^T Q x, \qquad M = \max_{||x||=1} x^T Q x = ||Q||,$$

and therefore

$$\frac{1}{m} = \max_{||x||=1} x^T Q^{-1} x = ||Q^{-1}||.$$

The condition number $\kappa(Q)$ of Q is defined as

$$\kappa(Q) = \frac{M}{m} = ||Q||||Q^{-1}||.$$

To proceed, we need a famous result by Kantorovich (without proof), namely

$$\frac{(x^T x)^2}{(x^T Q x)(x^T Q^{-1} x)} \geq \frac{4mM}{(m+M)^2}. \tag{2.8}$$

Substituting this into (2.7) and using

$$1 - \frac{4mM}{(m+M)^2} = \frac{(m+M)^2 - 4mM}{(m+M)^2} = \frac{(M-m)^2}{(M+m)^2} = \left(\frac{\kappa(Q)-1}{\kappa(Q)+1}\right)^2,$$

finally yields

$$E(x_{k+1}) \leq \left(\frac{\kappa(Q)-1}{\kappa(Q)+1}\right)^2 E(x_k),$$

showing convergence for the quadratic case.

In the nonlinear case, α_k must be computed via a line search, and the algorithm will converge linearly with convergence rate

$$\left(\frac{\kappa(Q)-1}{\kappa(Q)+1}\right)^2,$$

where $Q = \nabla^2 f(z)$ is the Hessian matrix at the minimizer.

2.2 Newton's method

Newton's method in higher dimension is based on the same idea as its one-dimensional version. At each step, the function to be minimized is replaced by its quadratic approximation, and the exact minimizer of the quadratic problem is used as the next approximation. With

$$g_k := \nabla f(x_k), \qquad H_k := \nabla^2 f(x_k),$$

this leads to

$$x_{k+1} = x_k - H_k^{-1} g_k \tag{2.9}$$

This algorithm converges quadratically fast, if x_0 is sufficiently close to the minimizer. Global convergence is a difficult topic. A more practical algorithm is the following:

QUASI-NEWTON algorithm.

Start with initial guess $x_0 \in \mathbb{R}^n$
for $k = 0, 1, 2, \ldots,$

Step 1.	Compute $f(x_k)$ (if not already done). STOP if appropriate, otherwise compute g_k.
Step 2.	Compute H_k, or an approximation to H_k.
Step 3.	Factor H_k (Cholesky factorization $H_k = L_k L_k^T$, or QR factorization $H_k = Q_k R_k$). Estimate the condition number of H_k. If H_k is ill-conditioned, perturb H_k (e.g., replace H_k by $H_k + \mu_k I$).
Step 4.	Solve $H_k s_k = -g_k$, using the factorization.
Step 5.	Take a full Newton step ($\alpha_k = 1$) if appropriate, or choose α_k by alternate strategy.
Step 6.	Update: $x_{k+1} = x_k + \alpha_k s_k$.

The following comments apply to any line search procedure, not just the one involved in the Quasi-Newton method above.

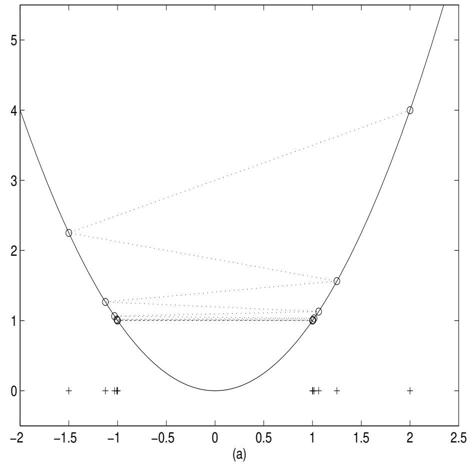

Figure 2.1: Example 2.1 (a)

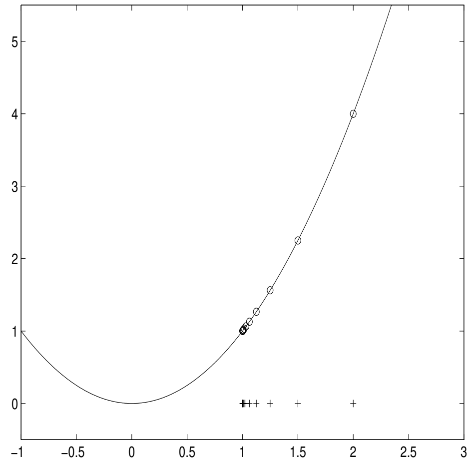

Figure 2.2: Example 2.1 (b)

We assume that p_k is a descent direction (for Newton: $p_k = s_k$, and this is a descent direction, if H_k is positive definite). Naturally we require α_k to be chosen such that $f(x_{k+1}) < f(x_k)$, but, as the following example shows, this is not sufficient for convergence.

Example 2.1 We take $n = 1$, and $f(x) = x^2$, initial value $x_0 = 2$.

(a) Taking $p_k = (-1)^{k+1}$ and $\alpha_k = 2 + 3 * 2^{-(k+1)}$ leads to the sequence $x_k = (-1)^k(1 + 2^{-k})$. Each p_k is a descent direction, $f(x_k)$ is monotonically decreasing, $\lim f(x_k) = 1$. The sequence x_k does not converge, however. See Figure 2.1.

(b) Take $p_k = -1$, $\alpha_k = -2^{-(k+1)}$, which gives $x_k = 1 + 2^{-k}$. $f(x_k)$ is again monotonically decreasing, the sequence (x_k) converges, but its limit is not a minimizer of f. See Figure 2.2.

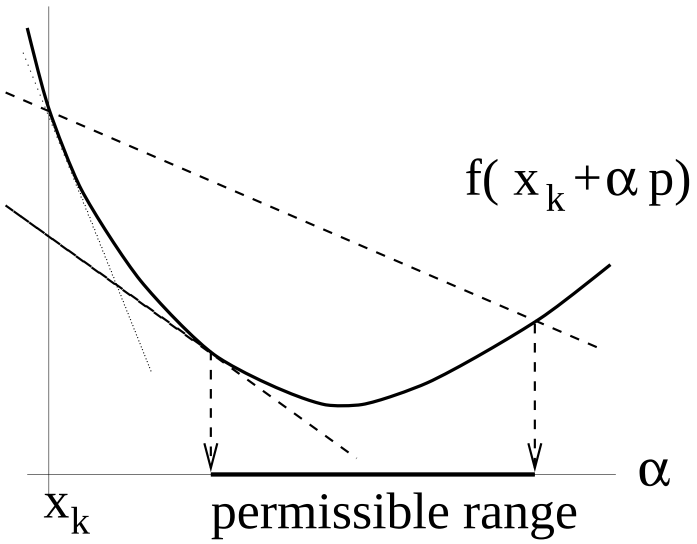

Figure 2.3: Line search conditions

The problem in (a) is that we achieve a very small decrease in f relative to the size of the step. Thus we require that the average rate of decrease should be larger than some prescribed fraction of the initial decrease, leading to the condition

$$f(x_k + \alpha p_k) \leq f(x_k) + \lambda\alpha\nabla f(x_k)^T p_k, \qquad 0 < \lambda < 1. \tag{2.10}$$

In (b), the steps are too small relative to the initial rate of decrease, i.e., it makes a lot of sense to move further along in the direction of p_k. Thus, we require, that the rate of **decrease** at the new point x_{k+1} is smaller than a fraction of the rate of decrease at the previous point x_k:

$$\nabla f(x_{k+1})^T p_k \geq \nu\nabla f(x_k)^T p_k, \qquad \lambda < \nu < 1,$$

or

$$\nabla f(x_{k+1})^T(x_{k+1} - x_k) \geq \nu \nabla f(x_k)^T(x_{k+1} - x_k), \qquad \lambda < \nu < 1. \quad (2.11)$$

The conditions (2.10) and (2.11) are illustrated in Figure 2.3. So we need a way to modify α_k, if one of the criteria (2.10) or (2.11) are violated.

2.3 Conjugate gradient method

Conjugate gradient methods are increasingly important in the field of iterative methods for solving linear equations. We will concentrate on the quadratic model problem

$$\min \frac{1}{2}x^TQx - b^Tx, \quad (2.12)$$

whose minimizer is the solution to the linear system of equations

$$Qx = b.$$

We assume Q is symmetric positive definite.

Definition 2.2 *Let B be a symmetric matrix. Two vectors x and y are called B-orthogonal, if $x^TBy = y^TBx = 0$.*

If Q is symmetric positive definite, than a system of Q-orthogonal vectors is linearly independent. Q-orthogonal directions are called *conjugate directions.*

A basis which is Q-orthogonal is preferable when one tries to minimize the quadratic function (2.12); note that steepest descent works great, when Q is an orthogonal matrix, i.e., when orthogonality and Q-orthogonality are the same.

From now, let $d_0, \ldots, d_{n-1}$ denote a set of n nonzero Q-orthogonal vectors, and let z be the exact solution to $Qz = b$, i.e., $z = Q^{-1}b$. The (d_k) form a basis for $\mathbb{R}^n$, therefore

$$z = \sum_{k=0}^{n-1} \alpha_k d_k.$$

Thus,

$$b = Qz = \sum_{k=0}^{n-1} \alpha_k Q d_k,$$

and, since the (d_k) are Q-orthogonal, we obtain

$$\alpha_j = \frac{d_j^T Q z}{d_j^T Q d_j} = \frac{d_j^T b}{d_j^T Q d_j}. \tag{2.13}$$

Formulated as an iterative algorithm with start vector $x_0 \in \mathbb{R}^n$, we obtain

$$\begin{aligned} \alpha_k &= -\frac{g_k^T d_k}{d_k^T Q d_k}, \qquad g_k = Q x_k - b & (2.14)\\ x_{k+1} &= x_k + \alpha_k d_k, \qquad k = 0, 1, \ldots, n-1. & (2.15) \end{aligned}$$

We have $x_n = z$, i.e., the algorithm converges to the exact solution in no more than n steps. Note that the α_k in (2.14) are not the same as the ones in (2.13); they are the expansion coefficients of $z - x_0$, i.e.,

$$z - x_0 = \sum_{k=0}^{n-1} \alpha_k d_k,$$

and as above, we have

$$\alpha_j = \frac{d_j^T Q(z - x_0)}{d_j^T Q d_j} = \frac{d_j^T (b - Q x_0}{d_j^T Q d_j} = -\frac{d_j^T g_0}{d_j^T Q d_j}.$$

Noting that

$$x_k - x_0 = \sum_{j=0}^{k-1} \alpha_j d_j$$

is Q orthogonal to d_k, $d_k^T Q(x_k - x_0) = 0$, we have

$$\begin{aligned} d_k^T Q(z - x_0) &= d_k^T Q(z - x_0) - \underbrace{d_k^T Q(x_k - x_0)}_{=0} \\ &= d_k^T Q(z - x_k) = -d_k^T g_k, \end{aligned}$$

establishing (2.14).

Let us define the k-dimensional subspaces $\mathcal{B}_k := \operatorname{span}\{d_0, \ldots, d_{k-1}\}$. It is clear, that $\mathcal{B}_k \subset \mathcal{B}_{k+1}$.

Theorem 2.3 (Expanding Subspace Theorem) *Let $x_0 \in \mathbb{R}^n$, and let the sequence x_k be defined by (2.14) and (2.15). The vector x_k minimizes $f(x) = \frac{1}{2} x^T Q x - b^T x$ on the line $x_{k-1} + \alpha d_{k-1}$; however, x_k also minimizes $f(x)$ over the whole affine space $x_0 + \mathcal{B}_k$.*

Proof. f is strictly convex, so h, the restriction of f to the affine subspace $x_0 + \mathcal{B}_k$ is also strictly convex. For x_k to be the unique global minimizer of h it is sufficient that $\nabla h(x_k) = 0$ which is equivalent to

$$\nabla f(x_k) \perp \mathcal{B}_k.$$

We prove that $g_k \perp \mathcal{B}_k$ by induction. Since $\mathcal{B}_0$ is empty, the statement is true for $k = 0$. Now we assume that $g_k \perp \mathcal{B}_k$, i.e.,

$$d_i^T g_k = 0 \qquad \text{for } i < k. \tag{2.16}$$

Obviously,

$$g_{k+1} = g_k + \alpha_k Q d_k. \tag{2.17}$$

Thus,

$$d_k^T g_{k+1} = d_k^T g_k + \alpha_k d_k^T Q d_k = 0,$$

because of (2.14), and, for $i < k$,

$$d_i^T g_{k+1} = \underbrace{d_i^T g_k}_{=0 \ (2.16)} + \alpha_k \underbrace{d_i^T Q d_k}_{=0 \ \text{Q-orthogonality}} = 0.$$

Therefore, $g_{k+1} \perp \mathcal{B}_{k+1}$. □

Usually, the conjugate directions $d_0, \ldots, d_{n-1}$ are not given at the outset. We determine them as we go along. We start with

$$d_0 = -g_0;$$

in the absence of any further information it is best to head into the direction of steepest descent. At the next step, we choose d_1 to be the direction of the component of $-g_1$ orthogonal to d_0:

$$d_1 = -g_1 + \beta_0 d_0,$$

with β_0 chosen such that $d_1^T Q d_0 = 0$. Then we take for d_2 the component of $-g_2$ which orthogonal to the space spanned by d_0 and d_1, and so on. Note that the gradient $g_k \neq 0$, unless the minimum is reached, and because of (2.16), or $g_k \perp \mathcal{B}_k$, g_k is not in the span of the previous directions, and therefore $d_k \neq 0$.

CONJUGATE GRADIENT ALGORITHM. Start with $x_0 \in \mathbb{R}^n$, set $d_0 = -g_0 = b - Qx_0$. For $k = 0, 1, \ldots, n-1$,

$$x_{k+1} = x_k + \alpha_k d_k, \tag{2.18}$$

$$\alpha_k = -\frac{g_k^T d_k}{d_k^T Q d_k}, \tag{2.19}$$

$$g_{k+1} = Q x_{k+1} - b \tag{2.20}$$

$$d_{k+1} = -g_{k+1} + \beta_k d_k, \tag{2.21}$$

$$\beta_k = \frac{g_{k+1}^T Q d_k}{d_k^T Q d_k}. \tag{2.22}$$

Verification of the algorithm.

(a)

$$\begin{aligned} \mathrm{span}(g_0, g_1, \ldots, g_k) &= \mathrm{span}(d_0, d_1, \ldots, d_k) = \\ \mathrm{span}(g_0, Qg_0, \ldots, Q^k g_0) &= \mathcal{K}_k(Q, g_0), \end{aligned}$$

where $\mathcal{K}_k(Q, g_0)$ denotes the $(k+1)$-dimensional *Krylov* subspace spanned by g_0. This can be shown by induction.

(b) $d_k^T Q d_j = 0$ for $j < k$, the d_j are Q-orthogonal. This is true for $k = 0$;

$$d_{k+1}^T Q d_j = -g_{k+1}^T Q d_j + \beta_k d_k^T Q d_j.$$

For $j = k$ we obtain zero because of (2.22). For $j < k$, we have

$$Q d_j \in Q\mathcal{B}_{j+1} = \mathcal{B}_{j+2} \subseteq \mathcal{B}_{k+1},$$

and $g_{k+1} \perp \mathcal{B}_{k+1}$, and $d_k^T Q d_j = 0$ because of the induction hypothesis.

(c)

$$\alpha_k = \frac{g_k^T g_k}{d_k^T Q d_k},$$

as

$$-g_k^T d_k = -g_k^T(-g_k + \beta_{k-1} d_{k-1}) = g_k^T g_k.$$

(d)

$$g_{k+1} = g_k + \alpha_k Q d_k \qquad \Rightarrow \qquad Q d_k = \frac{1}{\alpha_k}\left(g_{k+1} - g_k\right),$$

and therefore

$$g_{k+1}^T Q d_k = \frac{1}{\alpha_k} g_{k+1}^T g_{k+1},$$

as $g_{k+1}^T g_k = 0$ $(g_k \in \mathcal{B}_{k+1})$. So we obtain

$$\beta_k = \frac{1}{\alpha_k}\frac{g_{k+1}^T g_{k+1}}{d_k^T Q d_k} = -\frac{g_{k+1}^T g_{k+1}}{g_k^T g_k}.$$

2.4 Convergence of Conjugate Gradient

In the Conjugate gradient algorithm we have

$$x_{k+1} = x_k + \alpha_k d_k,$$

with $d_k \in \mathcal{B}_{k+1} = \mathcal{K}_k(Q, g_0)$, therefore

$$x_{k+1} - x_0 = p_k(Q) g_0, \qquad g_0 = Q x_0 - b,$$

where p_k is a polynomial of degree k. Denoting by z again the exact solution $Qz = b$, we obtain

$$\begin{aligned} x_{k+1} - z &= x_0 - z - (x_{k+1} - x_0) \\ &= x_0 - z - p_k(Q) Q (x_0 - z) \\ &= [I + Q p_k(Q)] (x_0 - z). \end{aligned}$$

Denoting by λ_i and v_i, $i = 1, \ldots, n$, the eigenvalues and (orthonormal) eigenvectors of Q, we have

$$x_0 - z = \sum_{i=1}^{n} \xi_i v_i, \qquad \Rightarrow \qquad Q(x_0 - z) = \sum_{i=1}^{n} \lambda_i \xi_i v_i,$$

and therefore

$$\begin{aligned} E(x_0) &= \frac{1}{2}(x_0 - z)^T Q (x_0 - z) = \frac{1}{2} \sum_{i=1}^{n} \lambda_i \xi_i^2 \\ E(x_{k+1}) &= \frac{1}{2} \sum_{i=1}^{n} \left[1 + \lambda_i p_k(\lambda_i)^2\right]^2 \lambda_i \xi_i^2; \end{aligned}$$

since conjugate gradient picks the p_k to minimize the function over the whole subspace $x_0 + \mathcal{B}_k$, we obtain

$$E(x_{k+1}) \leq \max_{1 \leq i \leq n} \left[p(\lambda_i)\right]^2 E(x_0),$$

for **any** polynomial p of degree $k+1$ with $p(0) = 1$ (see Figures 2.4 and 2.5).

In particular, we can find a polynomial of degree n which vanishes at all the n eigenvalues λ_i, showing convergence in at most n steps. If Q has multiple eigenvalues, we can find a polynomial p of degree less than n which is zero at the eigenvalues, and the algorithm finds the solution sooner. Furthermore, if the eigenvalues occur in just a few, say ℓ, clusters, there is a polynomial of (low) degree $\ell << n$ which takes small values at the eigenvalues, and the corresponding approximation x_ℓ will be close to the exact solution.

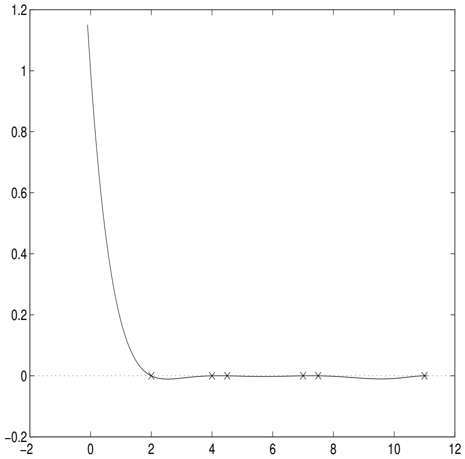

Figure 2.4: Polynomial p_k

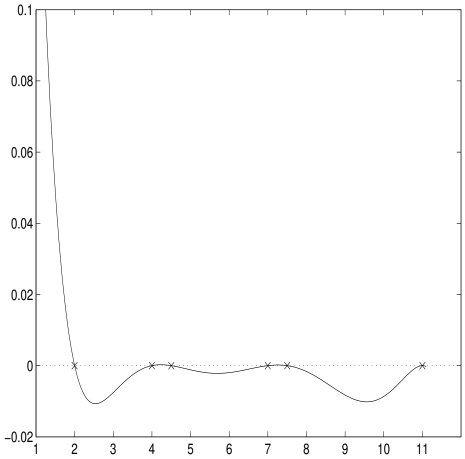

Figure 2.5: Polynomial p_k, blow-up

2.5 Quasi-Newton methods without Hessians

We now investigate the method of "deflected gradients"

$$x_{k+1} = x_k - \alpha_k S_k \nabla f(x_k), \tag{2.23}$$

with S_k a symmetric n by n matrix. If S_k is positive definite, then $p_k = -S_k \nabla f(x_k)$ is a descent direction. For $S_k \equiv I$ we have the steepest descent method, and for

$$S_k = \left[\nabla^2 f(x_k)\right]^{-1} = H_k^{-1}$$

and $\alpha_k = 1$ we have Newton's method. Our goal is now to find and algorithm which has the good convergence behaviour of Newton, without computing the Hessian matrix. The one-dimensional analogue would be the secant method. However the generalization to higher dimensions is not that straightforward, as two points do no longer determine the tangent (hyper)plane.

For the quadratic case,

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

exact line search gives us

$$\alpha_k = \frac{g_k^T S_k g_k}{g_k^T S_k Q S_k g_k},$$

reproducing the answers obtained for steepest descent and Newton.

One way to avoid recomputing the Hessian matrix at every step is to freeze it for a few iterations, i.e., take

$$S_k = \left[\nabla^2 f(x_0)\right]^{-1},$$

which also saves the work involved in factoring the Hessian. However, such a method still requires a procedure to evaluate the Hessian. Instead, we want to use the gradient information to *dynamically* construct an approximation to the Hessian matrix H_k or its inverse H_k^{-1}. The basic idea is as follows:

Denote by T_k an approximation to H_k^{-1},

$$T_k \approx H_k^{-1} = \left[\nabla^2 f(x_k)\right]^{-1}.$$

As always, we denote by g_k the gradients, $g_k = \nabla f(x_k)$. We now write

$$q_k = g_{k+1} - g_k, \qquad p_k = x_{k+1} - x_k,$$

and therefore have the relation

$$q_k \approx H_{k+1} p_k.$$

Thus, we can use this in finding T_k by requiring that

$$p_k = T_{k+1} q_k \tag{2.24}$$

(note that T_{k+1} approximates the inverse of H_{k+1}). But there are many matrices satisfying equation (2.24). So the idea is, to find an algorithm, which for **constant** $H = H_k$ uses n of these equations to determine $T_n = H^{-1}$ (so for the quadratic case we get the exact Hessian matrix!). More precisely, taking

$$P = \left(\begin{array}{c|c|c|c|c} | & | & & & | \\ p_0 & p_1 & \cdots & \cdots & p_{n-1} \\ | & | & & & | \end{array}\right), \qquad Q = \left(\begin{array}{c|c|c|c|c} | & | & & & | \\ q_0 & q_1 & \cdots & \cdots & q_{n-1} \\ | & | & & & | \end{array}\right),$$

the equations

$$H p_k = q_k, \qquad k = 0, 1, \ldots, n-1$$

become

$$HP = Q \qquad \Rightarrow \qquad H = QP^{-1},$$

so in principle, we can reconstruct the Hessian matrix from the gradient information contained in the first n steps of our optimization algorithm. We start with a symmetric, positive definite matrix T_0, for example $T_0 = I$. If we proceed finding T_k through rank 1 updates, we cannot guarantee positive definiteness. The following method using rank 2 updates does the trick:

DFP UPDATE (Davidon-Fletcher-Powell):

Start with initial guess $x_0 \in \mathbb{R}^n$,
T_0 symmetric, positive definite, $g_0 = \nabla f(x_0)$.
for $k = 0, 1, 2, \ldots,$
Step 1. Set $d_k = -T_k g_k$
Step 2. minimize $f(x_k + \alpha d_k)$ w.r.t. α,
Update: $x_{k+1} = x_k + \alpha_k d_k$,
$p_k = \alpha_k d_k, \qquad g_{k+1} = \nabla f(x_{k+1})$.
Step 3. $q_k = g_{k+1} - g_k$; update matrix T_k:

$$T_{k+1} = T_k + \frac{p_k p_k^T}{p_k^T q_k} - \frac{T_k q_k (T_k q_k)^T}{q_k^T T_k q_k}. \tag{2.25}$$

Lemma 2.4 (DFP method) *The matrices T_k obtained through (2.25) are all positive definite, and $p_k = T_{k+1} q_k$.*

Proof. T_0 is positive definite, so we prove the statement by induction. Assume T_k is positive definite. Then, from (2.25)

$$x^T T_{k+1} x = x^T T_k x + \frac{(x^T p_k)^2}{p_k^T q_k} - \frac{(x^T T_k q_k)^2}{q_k^T T_k q_k}.$$

T_k is positive definite, therefore the matrix $T_k^{\frac{1}{2}}$ is defined, and we can set $u = T_k^{\frac{1}{2}} x$ and $v = T_k^{\frac{1}{2}} q_k$ to obtain

$$x^T T_{k+1} x = u^T u - \frac{(u^T v)^2}{v^T v} + \frac{(x^T p_k)^2}{p_k^T q_k} = \frac{(u^T u)(v^T v) - (u^T v)^2}{v^T v} + \frac{(x^T p_k)^2}{p_k^T q_k}.$$

Since x_{k+1} minimizes f along the line $x_k + \alpha d_k$, the gradient of f at that point is orthogonal to the direction of the line, i.e.,

$$p_k^T g_{k+1} = 0; \tag{2.26}$$

therefore,

$$p_k^T q_k = p_k^T g_{k+1} - p_k^T g_k = -p_k^T g_k;$$

but

$$p_k = \alpha_k d_k = -\alpha_k T_k g_k,$$

and we find that

$$p_k^T q_k = -p_k^T g_k = \alpha_k g_k^T T_k g_k > 0, \tag{2.27}$$

because T_k is positive definite, $g_k \neq 0$ and $\alpha_k > 0$. So,

$$x^T T_{k+1} x = \frac{(u^T u)(v^T v) - (u^T v)^2}{v^T v} + \frac{(x^T p_k)^2}{\alpha_k g_k^T T_k g_k}.$$

Both term are nonnegative; the first term because of the CBS (Cauchy, Buniakowski, Schwarz) inequality,

$$|u^T v| \leq ||u||||v||,$$

which also states, that equality implies that u and v are parallel, i.e., $x = \beta q_k$ for some constant β. The second term is zero, if and only if x is perpendicular to p_k. Therefore, if both terms were zero simultaneously, we would conclude that

$$p_k \perp q_k,$$

contradicting (2.27). Thus, T_{k+1} is positive definite. Furthermore, we have

$$T_{k+1} q_k = T_k q_k + p_k - T_k q_k = p_k,$$

which concludes the proof of the lemma. □

Theorem 2.5 (DFP method) *Let f be a quadratic function with positive definite Hessian matrix H (constant, of course). The vectors p_0, p_1, $\ldots, p_{n-1}$ in the DFP method (or, equivalently, the vectors d_0, d_1, $\ldots, d_{n-1}$, $p_k = \alpha_k d_k$) are H-orthogonal, and*

$$T_n = H^{-1}.$$

More precisely we have

$$\begin{aligned} p_i^T H p_k &= 0, \qquad i < k < n, && (2.28) \\ T_{k+1} H p_i &= p_i, \qquad i \leq k < n. && (2.29) \end{aligned}$$

Proof. Since f is quadratic, we have

$$q_k = g_{k+1} - g_k = Hx_{k+1} - Hx_k = Hp_k. \tag{2.30}$$

Because of step 3 in the DFP algorithm, we have

$$T_{k+1}Hp_k = T_{k+1}q_k = p_k,$$

i.e., (2.29) holds for $i = k$. We now prove (2.28) and (2.29) by induction on k. Both hold for $k = 0$; now assume they hold for $k - 1$. Let $i < k$, then

$$\begin{aligned} g_k &= g_{i+1} + ((g_{i+2} - g_{i+1}) + \ldots + (g_k - g_{k-1})) \\ &= g_{i+1} + q_{i+1} + \ldots + q_{k-1} \\ &= g_{i+1} + H(p_{i+1} + \ldots + p_{k-1}), \end{aligned}$$

and therefore,

$$\begin{aligned} p_i^T g_k &= p_i^T g_{i+1} + (p_i^T Hp_{i+1} + \ldots + p_i^T Hp_{k-1}) \\ &= p_i^T g_{i+1} = 0, \end{aligned}$$

because of the induction hypothesis for (2.28), and (2.26) ($p_i \perp g_{i+1}$). Now, using (2.29), we know that $T_k Hp_i = p_i$, and therefore

$$0 = p_i^T g_k = p_i^T HT_k g_k,$$

as both H and T_k are symmetric. But from steps 1 and 2 of the algorithm we know that

$$p_k = -\alpha_k T_k g_k,$$

therefore, as $\alpha_k > 0$,

$$\frac{1}{\alpha_k} p_i^T Hp_k = 0, \qquad \text{for } i < k,$$

establishing (2.28). We also have

$$q_k^T T_k Hp_i = q_k^T p_i = p_k^T Hp_i = 0,$$

because of induction, the fact that f is quadratic and (2.28). Thus,

$$T_{k+1}Qp_i = \underbrace{T_k Qp_i}_{=p_i} + \underbrace{\frac{p_k p_k^T Qp_i}{p_k^T q_k}}_{=0} - \underbrace{\frac{T_k q_k q_k^T T_k Qp_i}{q_k^T T_k q_k}}_{=0} = p_i,$$

and $T_{k+1}Hp_k = p_k$ was shown at the beginning of the proof. Because p_0, $\ldots, p_{n-1}$ are H-orthogonal, they are linearly independent; they are eigenvectors of $T_n H$ with eigenvalues $\lambda_j = 1$, therefore $T_n H = I$. □

For the quadratic problem, the DFP algorithm with $T_0 = I$ is the conjugate gradient algorithm.