

Chapter 6. Least Squares Problems

A basic problem in science is to fit a model to observations subject to errors. It is clear that the more observations that are available, the more accurately will it be possible to calculate the parameters in the model. This gives rise to the problem of “solving” an overdetermined linear or nonlinear system of equations. It can be shown that the solution which minimizes a weighted sum of the squares of the residual is optimal in a certain sense.

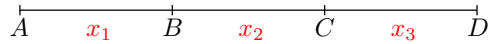
Åke Björck, Numerical Methods for Least Squares Problems, SIAM, 1996.

Least squares problems appear very naturally when one would like to estimate values of parameters of a mathematical model from measured data, which are subject to errors (see quote above). They appear however also in other contexts, and form an important subclass of more general optimization problems, see Chapter 12. After several typical examples of least squares problems, we start in Section 6.2 with the linear least squares problem and the natural solution given by the normal equations. There were two fundamental contributions to the numerical solution of linear least squares problems in the last century: the first one was the development of the QR factorization by Golub in 1965, and the second one was the implicit QR algorithm for computing the singular value decomposition (SVD) by Golub and Reinsch (1970). We introduce the SVD, which is fundamental for the understanding of linear least squares problems, in Section 6.3. We postpone the description of the algorithm for its computation to Chapter 7, but use the SVD to study the condition of the linear least squares problem in Section 6.4. This will show why the normal equations are not necessarily a good approach for solving linear least squares problems, and motivates the use of orthogonal transformations and the QR decomposition in Section 6.5. Like in optimization, least squares problems can also have constraints. We treat the linear least squares problem with linear constraints in full detail in Section 6.6, and a special class with nonlinear constraints in Section 6.7. We then turn to nonlinear least squares problems in Section 6.8, which have to be solved by iteration. We show classical iterative methods for such problems, and like in the case of nonlinear equations, linear least squares problems arise naturally at each iteration. We conclude this chapter with an interesting example of least squares fitting with piecewise functions in Section 6.9. The currently best and most thorough reference for least squares methods is the book by Åke Björck [9].

6.1 Introductory Examples

We start this chapter with several typical examples leading to least squares problems.

EXAMPLE 6.1. *Measuring a road segment (Stiefel in his lectures at ETH¹).*



Assume that we have performed 5 measurements,

$$AD = 89\text{m}, AC = 67\text{m}, BD = 53\text{m}, AB = 35\text{m und } CD = 20\text{m},$$

and we want to determine the length of the segments $x_1 = AB$, $x_2 = BC$ und $x_3 = CD$.

According to the observations we get a linear system with more equations than unknowns:

$$\begin{array}{rcl} x_1 + x_2 + x_3 & = & 89 \\ x_1 + x_2 & = & 67 \\ x_2 + x_3 & = & 53 \\ x_1 & = & 35 \\ x_3 & = & 20 \end{array} \iff A\mathbf{x} = \mathbf{b}, \quad A = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} 89 \\ 67 \\ 53 \\ 35 \\ 20 \end{pmatrix}.$$

Notice that if we use the last three equations then we get the solution $x_1 = 35$, $x_2 = 33$ and $x_3 = 20$. However, if we check the first two equations by inserting this solution we get

$$\begin{array}{rcl} x_1 + x_2 + x_3 - 89 & = & -1, \\ x_1 + x_2 - 67 & = & 1. \end{array}$$

So the equations contradict each other because of the measurement errors, and the over-determined system has no solution.

A remedy is to find an approximate solution that satisfies the equations as well as possible. For that purpose one introduces the residual vector

$$\mathbf{r} = \mathbf{b} - A\mathbf{x}.$$

One then looks for a vector \mathbf{x} that minimizes in some sense the residual vector.

EXAMPLE 6.2. *The amount f of a component in a chemical reaction decreases with time t exponentially according to:*

$$f(t) = a_0 + a_1 e^{-bt}.$$

¹Nehmen wir an, der Meister schickt seine zwei Lehrlinge aus, Strassenstücke zu vermessen. . .

If the material is weighed at different times, we obtain a table of measured values:

| | | | |
|-----|-------|----------|-------|
| t | t_1 | \cdots | t_m |
| y | y_1 | \cdots | y_m |

The problem now is to estimate the model parameters a_0 , a_1 and b from these observations. Each measurement point (t_i, y_i) yields an equation:

$$f(t_i) = a_0 + a_1 e^{-bt_i} \approx y_i, \quad i = 1, \dots, m. \quad (6.1)$$

If there were no measurement errors, then we could replace the approximate symbol in (6.1) by an equality and use three equations from the set to determine the parameters. However, in practice, measurement errors are inevitable. Furthermore, the model equations are often not quite correct and only model the physical behavior approximately. The equations will therefore in general contradict each other and we need some mechanism to balance the measurement errors, e.g. by requiring that (6.1) be satisfied as well as possible.

EXAMPLE 6.3. The next example comes from coordinate metrology. Here a coordinate measuring machine measures two sets of points on two orthogonal lines (see Figure 6.1). If we represent the line g_1 by the equations

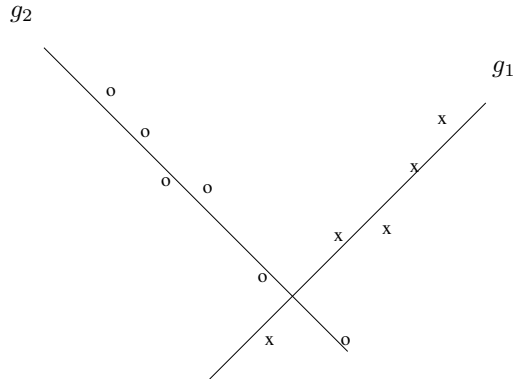


FIGURE 6.1. Measured points on two orthogonal lines.

$$g_1 : \quad c_1 + n_1 x + n_2 y = 0, \quad n_1^2 + n_2^2 = 1, \quad (6.2)$$

then $\mathbf{n} = (n_1, n_2)^\top$ is the normal vector on g_1 . The normalizing equation $n_1^2 + n_2^2 = 1$ ensures the uniqueness of the parameters c , n_1 and n_2 .

If we insert the coordinates of a measured point $P_i = (x_i, y_i)$ into Equation (6.2), we obtain the residual $r_i = c_1 + n_1 x_i + n_2 y_i$ and $d_i = |r_i|$ is the distance of P_i from g_1 . The equation of a line g_2 orthogonal to g_1 is

$$g_2 : \quad c_2 - n_2 x + n_1 y = 0, \quad n_1^2 + n_2^2 = 1. \quad (6.3)$$

If we now insert the coordinates of q measured points Q_i into (6.3) and of p points P_i into Equation (6.2), we obtain the following system of equations for determining the parameters c_1 , c_2 , n_1 and n_2 :

$$\begin{pmatrix} 1 & 0 & x_{P_1} & y_{P_1} \\ 1 & 0 & x_{P_2} & y_{P_2} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & 0 & x_{P_p} & y_{P_p} \\ 0 & 1 & y_{Q_1} & -x_{Q_1} \\ 0 & 1 & y_{Q_2} & -x_{Q_2} \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 1 & y_{Q_q} & -x_{Q_q} \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ n_1 \\ n_2 \end{pmatrix} \approx \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \quad \text{subject to } n_1^2 + n_2^2 = 1. \quad (6.4)$$

Note the difference between the least squares equations and the constraint: whereas equations of the type g_1 or g_2 only need to be satisfied approximately, the constraint $n_1^2 + n_2^2 = 1$ must be satisfied exactly by the solution.

EXAMPLE 6.4. In control theory, one often considers a system like the one in Figure 6.2. The vectors \mathbf{u} and \mathbf{y} are the measured input and output signals at various points in time. Let $y_{t+i} = y(t + i\Delta t)$. A simple model assumes a linear relationship between the output and the input signal of the form

$$y_{t+n} + a_{n-1}y_{t+n-1} + \cdots + a_0y_t \approx b_{n-1}u_{t+n-1} + b_{n-2}u_{t+n-2} + \cdots + b_0u_t. \quad (6.5)$$

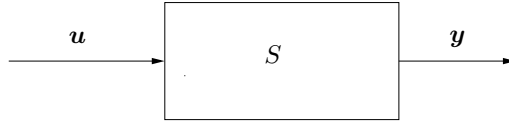


FIGURE 6.2. System with input \mathbf{u} and output \mathbf{y}

The problem is to determine the parameters a_i and b_i from measurements of \mathbf{u} and \mathbf{y} . For each time step we obtain a new equation of the form (6.5). If we write them all together, we get a system of linear equations:

$$\begin{pmatrix} y_{n-1} & y_{n-2} & \cdots & y_0 & -u_{n-1} & -u_{n-2} & \cdots & -u_0 \\ y_n & y_{n-1} & \cdots & y_1 & -u_n & -u_{n-1} & \cdots & -u_1 \\ y_{n+1} & y_n & \cdots & y_2 & -u_{n+1} & -u_n & \cdots & -u_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix} \begin{pmatrix} a_{n-1} \\ a_{n-2} \\ \vdots \\ a_0 \\ b_{n-1} \\ b_{n-2} \\ \vdots \\ b_n \end{pmatrix} \approx \begin{pmatrix} -y_n \\ -y_{n+1} \\ -y_{n+2} \\ \vdots \end{pmatrix} \quad (6.6)$$

A matrix is said to be Toeplitz if it has constant elements on the diagonals. Here in (6.6), the matrix is composed of two Toeplitz matrices. The number of equations is not fixed, since one can generate new equations simply by adding a new measurement.

EXAMPLE 6.5. In robotics and many other applications, one often encounters the Procrustes problem or one of its variants (see [45], Chapter 23). Consider a given body (e.g., a pyramid like in Figure 6.3) and a copy of the same body. Assume that we know the coordinates of m points \mathbf{x}_i on

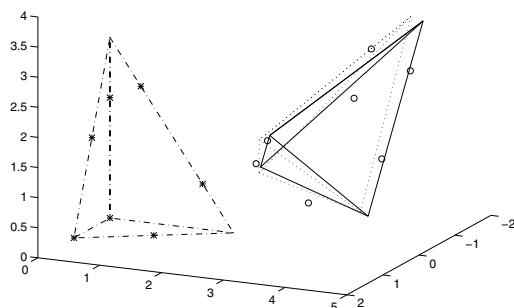


FIGURE 6.3. Procrustes or registration problem

the first body, and that the corresponding points ξ_i have been measured on the other body in another position in space. We would like to rotate and translate the second body so that it can be superimposed onto the first one as well as possible. In other words, we seek an orthogonal matrix Q (the product of three rotations) and a translation vector \mathbf{t} such that $\xi_i \approx Q\mathbf{x}_i + \mathbf{t}$ for $i = 1, \dots, m$.

The above examples are illustrations of different classes of approximation problems. For instance, in Examples 6.1 and 6.4, the equations are linear. However, in Example 6.2 (chemical reactions), the system of equations (6.1) is nonlinear. In the metrology example 6.3, the equations are linear, but they are subject to the nonlinear constraint $n_1^2 + n_2^2 = 1$. Finally, we have also an nonlinear problem in Example 6.5, and it is not clear how to parametrize the unknown matrix Q . Nonetheless, in all the above examples, we would like to satisfy some equations as well as possible; this is indicated by the approximation symbol “ \approx ” and we have to define what we mean by that.

There are also least squares problems that are not connected with measurements like in the following example:

EXAMPLE 6.6. We consider two straight lines g and h in space. Assume

they are given by a point and a direction vector:

$$\begin{aligned} g: \mathbf{X} &= \mathbf{P} + \lambda \mathbf{t} \\ h: \mathbf{Y} &= \mathbf{Q} + \mu \mathbf{s} \end{aligned}$$

If they intersect each other, then there must exist a λ and a μ such that

$$\mathbf{P} + \lambda \mathbf{t} = \mathbf{Q} + \mu \mathbf{s}. \quad (6.7)$$

Rearranging (6.7) yields

$$\begin{pmatrix} t_1 & -s_1 \\ t_2 & -s_2 \\ t_3 & -s_3 \end{pmatrix} \begin{pmatrix} \lambda \\ \mu \end{pmatrix} = \begin{pmatrix} Q_1 - P_1 \\ Q_2 - P_2 \\ Q_3 - P_3 \end{pmatrix} \quad (6.8)$$

a system of three linear equations with two unknowns. If the equations are consistent, then we can use two of them to determine the intersection point. If, however, we have a pair of skew lines (i.e., if (6.8) has no solution) then we may be interested in finding the point \mathbf{X} on g and \mathbf{Y} on h which are closest, i.e. for which the distance vector $\mathbf{r} = \mathbf{X} - \mathbf{Y}$ has minimal length $\|\mathbf{r}\|_2^2 \rightarrow \min$. Thus, we are interested in solving (6.8) as a least squares problem.

6.2 Linear Least Squares Problem and the Normal Equations

Linear least squares problems occur when solving overdetermined linear systems, i.e. we are given more equations than unknowns. In general, such an overdetermined system has no solution, but we may find a meaningful approximate solution by minimizing some norm of the residual vector.

Given a matrix $A \in \mathbb{R}^{m \times n}$ with $m > n$ and a vector $\mathbf{b} \in \mathbb{R}^m$ we are looking for a vector $\mathbf{x} \in \mathbb{R}^n$ for which the norm of the residual \mathbf{r} is minimized, i.e.

$$\|\mathbf{r}\| = \|\mathbf{b} - A\mathbf{x}\| \rightarrow \min. \quad (6.9)$$

The calculations are simplest when we choose the 2-norm. Thus we will minimize the square of the length of the residual vector

$$\|\mathbf{r}\|_2^2 = r_1^2 + r_2^2 + \cdots + r_m^2 \rightarrow \min. \quad (6.10)$$

To see that this minimum exists and is attained by some $\mathbf{x} \in \mathbb{R}^n$, note that $E = \{\mathbf{b} - A\mathbf{x} \mid \mathbf{x} \in \mathbb{R}^n\}$ is a non-empty, closed and convex subset of \mathbb{R}^m . Since \mathbb{R}^m equipped with the Euclidean inner product is a Hilbert space, [108, Thm 4.10] asserts that E contains a unique element of smallest norm, so there exists an $\mathbf{x} \in \mathbb{R}^n$ (not necessarily unique) such that $\|\mathbf{b} - A\mathbf{x}\|_2$ is minimized.

The minimization problem (6.10) gave rise to the name *Least Squares Method*. The theory was developed independently by CARL FRIEDRICH

GAUSS in 1795 and ADRIEN-MARIE LEGENDRE who published it first in 1805. On January 1, 1801, using the least squares method, GAUSS made the best prediction of the orbital positions of the planetoid Ceres based on measurements of G. PIAZZI, and the method became famous because of this.

We characterize the least squares solution by the following theorem.

THEOREM 6.1. (LEAST SQUARES SOLUTION) *Let*

$$\mathcal{S} = \{\mathbf{x} \in \mathbb{R}^n \text{ with } \|\mathbf{b} - A\mathbf{x}\|_2 \longrightarrow \min\}$$

be the set of solutions and let $\mathbf{r}_x = \mathbf{b} - A\mathbf{x}$ denote the residual for a specific \mathbf{x} . Then

$$\mathbf{x} \in \mathcal{S} \iff A^\top \mathbf{r}_x = 0 \iff \mathbf{r}_x \perp \mathcal{R}(A), \quad (6.11)$$

where $\mathcal{R}(A)$ denotes the subspace spanned by the columns of A .

PROOF. We prove the first equivalence, from which the second one follows easily.

“ \Leftarrow ”: Let $A^\top \mathbf{r}_x = 0$ and $\mathbf{z} \in \mathbb{R}^n$ be an arbitrary vector. It follows that $\mathbf{r}_z = \mathbf{b} - A\mathbf{z} = \mathbf{b} - A\mathbf{x} + A(\mathbf{x} - \mathbf{z})$, thus $\mathbf{r}_z = \mathbf{r}_x + A(\mathbf{x} - \mathbf{z})$. Now

$$\|\mathbf{r}_z\|_2^2 = \|\mathbf{r}_x\|_2^2 + 2(\mathbf{x} - \mathbf{z})^\top A^\top \mathbf{r}_x + \|A(\mathbf{x} - \mathbf{z})\|_2^2.$$

But $A^\top \mathbf{r}_x = 0$ and therefore $\|\mathbf{r}_z\|_2 \geq \|\mathbf{r}_x\|_2$. Since this holds for every \mathbf{z} then $\mathbf{x} \in \mathcal{S}$.

“ \Rightarrow ”: We show this by contradiction: assume $A^\top \mathbf{r}_x = \mathbf{z} \neq 0$. We consider $\mathbf{u} = \mathbf{x} + \varepsilon \mathbf{z}$ with $\varepsilon > 0$:

$$\mathbf{r}_u = \mathbf{b} - A\mathbf{u} = \mathbf{b} - A\mathbf{x} - \varepsilon A\mathbf{z} = \mathbf{r}_x - \varepsilon A\mathbf{z}.$$

Now $\|\mathbf{r}_u\|_2^2 = \|\mathbf{r}_x\|_2^2 - 2\varepsilon \mathbf{z}^\top A^\top \mathbf{r}_x + \varepsilon^2 \|A\mathbf{z}\|_2^2$. Because $A^\top \mathbf{r}_x = \mathbf{z}$ we obtain

$$\|\mathbf{r}_u\|_2^2 = \|\mathbf{r}_x\|_2^2 - 2\varepsilon \|\mathbf{z}\|_2^2 + \varepsilon^2 \|A\mathbf{z}\|_2^2.$$

We conclude that, for sufficient small ε , we can obtain $\|\mathbf{r}_u\|_2^2 < \|\mathbf{r}_x\|_2^2$. This is a contradiction, since \mathbf{x} cannot be in the set of solutions in this case. Thus the assumption was wrong, i.e., we must have $A^\top \mathbf{r}_x = 0$, which proves the first equivalence in (6.11). \square

The least squares solution has an important statistical property which is expressed in the following *Gauss-Markoff Theorem*. Let the vector \mathbf{b} of observations be related to an unknown parameter vector \mathbf{x} by the linear relation

$$A\mathbf{x} = \mathbf{b} + \boldsymbol{\epsilon}, \quad (6.12)$$

where $A \in \mathbb{R}^{m \times n}$ is a known matrix and $\boldsymbol{\epsilon}$ is a vector of random errors. In this *standard linear model* it is assumed that the random variables ϵ_j are uncorrelated and all have zero mean and the same variance.

THEOREM 6.2. (GAUSS-MARKOFF) *Consider the standard linear model (6.12). Then the best linear unbiased estimator of any linear function $\mathbf{c}^\top \mathbf{x}$ is the least square solution of $\|A\mathbf{x} - \mathbf{b}\|_2^2 \longrightarrow \min$.*

PROOF. Consult a statistics textbook, for example [94, p. 181]. \square

Equation (6.11) can be used to determine the least square solution. From $A^\top \mathbf{r}_x = 0$ it follows that $A^\top (\mathbf{b} - A\mathbf{x}) = 0$, and we obtain the *Normal Equations* of Gauss:

$$A^\top A\mathbf{x} = A^\top \mathbf{b}. \quad (6.13)$$

EXAMPLE 6.7. We return to Example 6.1 and solve it using the Normal Equations.

$$A^\top A\mathbf{x} = A^\top \mathbf{b} \iff \begin{pmatrix} 3 & 2 & 1 \\ 2 & 3 & 2 \\ 1 & 2 & 3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 191 \\ 209 \\ 162 \end{pmatrix}.$$

The solution of this 3×3 system is

$$\mathbf{x} = \begin{pmatrix} 35.125 \\ 32.500 \\ 20.625 \end{pmatrix}.$$

The residual for this solution becomes

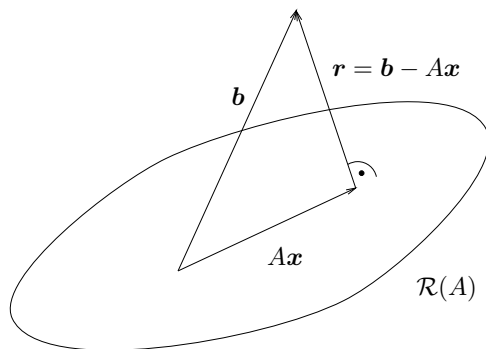
$$\mathbf{r} = \mathbf{b} - A\mathbf{x} = \begin{pmatrix} 0.7500 \\ -0.6250 \\ -0.1250 \\ -0.1250 \\ -0.6250 \end{pmatrix} \quad \text{with } \|\mathbf{r}\|_2 = 1.1726.$$

Notice that for the solution $\mathbf{x} = (35, 33, 20)^\top$ obtained by solving the last three equations we obtain a larger residual $\|\mathbf{r}\|_2 = \sqrt{2} = 1.4142$.

There is also a way to understand the normal equations geometrically from (6.11). We want to find a linear combination of columns of the matrix A to approximate the vector \mathbf{b} . The space spanned by the columns of A is the range of A , $\mathcal{R}(A)$, which is a hyperplane in \mathbb{R}^m , and the vector \mathbf{b} in general does not lie in this hyperplane, as shown in Figure 6.4. Thus, minimizing $\|\mathbf{b} - A\mathbf{x}\|_2$ is equivalent to minimizing the length of the residual vector \mathbf{r} , and thus the residual vector has to be orthogonal to $\mathcal{R}(A)$, as shown in Figure 6.4.

The normal equations (6.13) concentrate data since $B = A^\top A$ is a small $n \times n$ matrix, whereas A is $m \times n$. The matrix B is symmetric, and if $\text{rank}(A) = n$, then it is also positive definite. Thus, the natural way to solve the normal equations is by means of the *Cholesky decomposition* (cf. Section 3.4.1):

1. Form $B = A^\top A$ (we need to compute only the upper triangle since B is symmetric) and compute $\mathbf{c} = A^\top \mathbf{b}$.
2. Decompose $B = R^\top R$ (Cholesky) where R is an upper triangular matrix.

FIGURE 6.4. \mathbf{r} is orthogonal to $\mathcal{R}(A)$

3. Compute the solution by forward- ($R^\top \mathbf{y} = \mathbf{c}$) and back-substitution ($R\mathbf{x} = \mathbf{y}$).

We will see later on that there are numerically preferable methods for computing the least squares solution. They are all based on the use of *orthogonal matrices* (i.e. matrices B for which $B^\top B = I$).

Notice that when solving linear systems $A\mathbf{x} = \mathbf{b}$ with n equations and n unknowns by Gaussian elimination, reducing the system to triangular form, we make use of the fact that *equivalent systems have the same solutions*:

$$A\mathbf{x} = \mathbf{b} \iff BA\mathbf{x} = B\mathbf{b} \quad \text{if } B \text{ is nonsingular.}$$

For a system of equations $A\mathbf{x} \approx \mathbf{b}$ to be solved in the least squares sense, it no longer holds that multiplying by a nonsingular matrix B leads to an equivalent system. This is because the transformed residual $B\mathbf{r}$ may not have the same norm as \mathbf{r} itself. However, if we restrict ourselves to the class of *orthogonal matrices*,

$$A\mathbf{x} \approx \mathbf{b} \iff BA\mathbf{x} \approx B\mathbf{b} \quad \text{if } B \text{ is orthogonal.}$$

then the least squares problems remain equivalent, since $\mathbf{r} = \mathbf{b} - A\mathbf{x}$ and $B\mathbf{r} = B\mathbf{b} - BA\mathbf{x}$ have the same length,

$$\|B\mathbf{r}\|_2^2 = (B\mathbf{r})^\top (B\mathbf{r}) = \mathbf{r}^\top B^\top B\mathbf{r} = \mathbf{r}^\top \mathbf{r} = \|\mathbf{r}\|_2^2.$$

Orthogonal matrices and the matrix decompositions containing orthogonal factors therefore play an important role in algorithms for the solution of linear least squares problems. Often it is possible to simplify the equations by pre-multiplying the system by a suitable orthogonal matrix.

6.3 Singular Value Decomposition (SVD)

The singular value decomposition (SVD) of a matrix A is a very useful tool in the context of least squares problems. It is also very helpful for analyzing properties of a matrix. With the SVD one x-rays a matrix!

THEOREM 6.3. (SINGULAR VALUE DECOMPOSITION, SVD) *Let $A \in \mathbb{R}^{m \times n}$ with $m \geq n$. Then there exist orthogonal matrices $U \in \mathbb{R}^{m \times m}$ and $V \in \mathbb{R}^{n \times n}$ and a diagonal matrix $\Sigma = \text{diag}(\sigma_1, \dots, \sigma_n) \in \mathbb{R}^{m \times n}$ with $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_n \geq 0$, such that*

$$A = U \Sigma V^\top$$

holds. The column vectors of $U = [\mathbf{u}_1, \dots, \mathbf{u}_m]$ are called the left singular vectors and similarly $V = [\mathbf{v}_1, \dots, \mathbf{v}_n]$ are the right singular vectors. The values σ_i are called the singular values of A . If $\sigma_r > 0$ is the smallest non-zero singular value, then the matrix A has rank r .

PROOF. The 2-norm of A is defined by $\|A\|_2 = \max_{\|\mathbf{x}\|_2=1} \|A\mathbf{x}\|_2$. Thus there exists a vector \mathbf{x} with $\|\mathbf{x}\|_2 = 1$ such that

$$\mathbf{z} = A\mathbf{x}, \quad \|\mathbf{z}\|_2 = \|A\|_2 =: \sigma.$$

Let $\mathbf{y} := \mathbf{z}/\|\mathbf{z}\|_2$. This yields $A\mathbf{x} = \sigma\mathbf{y}$ with $\|\mathbf{x}\|_2 = \|\mathbf{y}\|_2 = 1$.

Next we extend \mathbf{x} into an orthonormal basis of \mathbb{R}^n . If $V \in \mathbb{R}^{n \times n}$ is the matrix containing the basis vectors as columns, then V is an orthogonal matrix that can be written as $V = [\mathbf{x}, V_1]$, where $V_1^\top \mathbf{x} = 0$. Similarly, we can construct an orthogonal matrix $U \in \mathbb{R}^{m \times m}$ satisfying $U = [\mathbf{y}, U_1]$, $U_1^\top \mathbf{y} = 0$. Now

$$A_1 = U^\top AV = \begin{bmatrix} \mathbf{y}^\top \\ U_1^\top \end{bmatrix} A [\mathbf{x}, V_1] = \begin{bmatrix} \mathbf{y}^\top A\mathbf{x} & \mathbf{y}^\top AV_1 \\ U_1^\top A\mathbf{x} & U_1^\top AV_1 \end{bmatrix} = \begin{bmatrix} \sigma & \mathbf{w}^\top \\ 0 & B \end{bmatrix},$$

because $\mathbf{y}^\top A\mathbf{x} = \mathbf{y}^\top \sigma\mathbf{y} = \sigma\mathbf{y}^\top \mathbf{y} = \sigma$ and $U_1^\top A\mathbf{x} = \sigma U_1^\top \mathbf{y} = 0$ since $U_1 \perp \mathbf{y}$.

We claim that $\mathbf{w}^\top := \mathbf{y}^\top AV_1 = 0$. In order to prove this, we compute

$$A_1 \begin{pmatrix} \sigma \\ \mathbf{w} \end{pmatrix} = \begin{pmatrix} \sigma^2 + \|\mathbf{w}\|_2^2 \\ B\mathbf{w} \end{pmatrix}$$

and conclude from that equation that

$$\left\| A_1 \begin{pmatrix} \sigma \\ \mathbf{w} \end{pmatrix} \right\|_2^2 = (\sigma^2 + \|\mathbf{w}\|_2^2)^2 + \|B\mathbf{w}\|_2^2 \geq (\sigma^2 + \|\mathbf{w}\|_2^2)^2.$$

Now since V and U are orthogonal, $\|A_1\|_2 = \|U^\top AV\|_2 = \|A\|_2 = \sigma$ holds and

$$\sigma^2 = \|A_1\|_2^2 = \max_{\|\mathbf{x}\|_2 \neq 0} \frac{\|A_1 \mathbf{x}\|_2^2}{\|\mathbf{x}\|_2^2} \geq \frac{\|A_1 \begin{pmatrix} \sigma \\ \mathbf{w} \end{pmatrix}\|_2^2}{\left\| \begin{pmatrix} \sigma \\ \mathbf{w} \end{pmatrix} \right\|_2^2} \geq \frac{(\sigma^2 + \|\mathbf{w}\|_2^2)^2}{\sigma^2 + \|\mathbf{w}\|_2^2}.$$

The last equation reads

$$\sigma^2 \geq \sigma^2 + \|\mathbf{w}\|_2^2,$$

and we conclude that $\mathbf{w} = 0$. Thus we have obtained

$$A_1 = U^\top AV = \begin{bmatrix} \sigma & 0 \\ 0 & B \end{bmatrix}.$$

We can now apply the same construction to the sub-matrix B and thus finally end up with a diagonal matrix. \square

Although the proof is constructive, the singular value decomposition is not usually computed in this way. An efficient numerical algorithm was designed by Golub and Reinsch [148]. They first transform the matrix by orthogonal Householder transformations to bidiagonal form. Then the bidiagonal matrix is further diagonalized in an iterative process by a variant of the QR Algorithm. For details see Section 7.7 in Chapter 7 on eigenvalues.

If we write the equation $A = U\Sigma V^\top$ in partitioned form, in which Σ_r contains only the nonzero singular values, we get

$$A = [U_1, U_2] \begin{pmatrix} \Sigma_r & 0 \\ 0 & 0 \end{pmatrix} [V_1, V_2]^\top \quad (6.14)$$

$$= U_1 \Sigma_r V_1^\top \quad (6.15)$$

$$= \sum_{i=1}^r \sigma_i \mathbf{u}_i \mathbf{v}_i^\top. \quad (6.16)$$

Equation (6.14) is the *full decomposition* with square matrices U and V . When making use of the zeros we obtain the “*economy*” or “*reduced*” version of the SVD given in (6.15). In MATLAB there are two variants to compute the SVD:

```
[U S V]=svd(A)    % gives the full decomposition
[U S V]=svd(A,0)  % gives an m by n matrix U
```

The call `svd(A,0)` computes a version between full and economic with a non-square matrix $U \in \mathbb{R}^{m \times n}$. This form is sometimes referred to as the “thin SVD”.

EXAMPLE 6.8. The matrix A has rank one and its economy SVD is given by

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} \frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \end{pmatrix} (2\sqrt{3}) \begin{pmatrix} \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} \end{pmatrix}.$$

With MATLAB we get the thin version

```
>> [U,S,V]=svd(ones(4,3),0)
U =
   -0.5000    0.8660   -0.0000
   -0.5000   -0.2887   -0.5774
   -0.5000   -0.2887    0.7887
   -0.5000   -0.2887   -0.2113
S =
    3.4641         0         0
         0    0.0000         0
```

$$\begin{array}{ccc}
 & 0 & 0 & 0 \\
 \mathbf{V} = & -0.5774 & 0.8165 & 0 \\
 & -0.5774 & -0.4082 & -0.7071 \\
 & -0.5774 & -0.4082 & 0.7071
 \end{array}$$

THEOREM 6.4. If $A = U\Sigma V^\top$, then the column vectors of V are the eigenvectors of the matrix $A^\top A$ associated with the eigenvalues σ_i^2 , $i = 1, \dots, n$. The column vectors of U are the eigenvectors of the matrix AA^\top .

PROOF.

$$A^\top A = (U\Sigma V^\top)^\top U\Sigma V^\top = V D V^\top, \quad D = \Sigma^\top \Sigma = \text{diag}(\sigma_1^2, \dots, \sigma_n^2). \quad (6.17)$$

Thus $A^\top A V = V D$ and σ_i^2 is an eigenvalue of $A^\top A$. Similarly

$$A A^\top = U \Sigma V^\top (U \Sigma V^\top)^\top = U \Sigma \Sigma^\top U^\top, \quad (6.18)$$

where $\Sigma \Sigma^\top = \text{diag}(\sigma_1^2, \dots, \sigma_n^2, 0, \dots, 0) \in \mathbb{R}^{m \times m}$. \square

THEOREM 6.5. Let $A = U\Sigma V^\top$. Then

$$\|A\|_2 = \sigma_1 \quad \text{and} \quad \|A\|_F = \sqrt{\sum_{i=1}^n \sigma_i^2}.$$

PROOF. Since U and V are orthogonal, we have $\|A\|_2 = \|U\Sigma V^\top\|_2 = \|\Sigma\|_2$. Now

$$\|\Sigma\|_2^2 = \max_{\|\mathbf{x}\|_2=1} \|\Sigma \mathbf{x}\|_2^2 = \max_{\|\mathbf{x}\|_2=1} (\sigma_1^2 x_1^2 + \dots + \sigma_n^2 x_n^2) \leq \sigma_1^2 (x_1^2 + \dots + x_n^2) = \sigma_1^2,$$

and since the maximum is attained for $\mathbf{x} = \mathbf{e}_1$ it follows that $\|A\|_2 = \sigma_1$. For the Frobenius norm we have

$$\|A\|_F = \sqrt{\sum_{i,j} a_{ij}^2} = \sqrt{\text{tr}(A^\top A)} = \sqrt{\sum_{i=1}^n \sigma_i^2},$$

since the trace of a matrix equals the sum of its eigenvalues. \square

In (6.16), we have decomposed the matrix A as a sum of rank-one matrices of the form $\mathbf{u}_i \mathbf{v}_i^\top$. Now we have

$$\|\mathbf{u}_i \mathbf{v}_i^\top\|_2^2 = \max_{\|\mathbf{x}\|_2=1} \|\mathbf{u}_i \mathbf{v}_i^\top \mathbf{x}\|_2^2 = \max_{\|\mathbf{x}\|_2=1} |\mathbf{v}_i^\top \mathbf{x}|^2 \|\mathbf{u}_i\|_2^2 = \max_{\|\mathbf{x}\|_2=1} |\mathbf{v}_i^\top \mathbf{x}|^2,$$

and since

$$\max_{\|\mathbf{x}\|_2=1} |\mathbf{v}_i^\top \mathbf{x}|^2 = \max_{\|\mathbf{x}\|_2=1} (\|\mathbf{v}_i\|_2 \|\mathbf{x}\|_2 \cos \alpha)^2 = \cos^2 \alpha,$$

where α is the angle between the two vectors, we obtain

$$\max_{\|\mathbf{x}\|_2=1} \|\mathbf{v}_i^\top \mathbf{x}\|_2^2 = \|\mathbf{v}_i^\top \mathbf{v}_i\|_2^2 = 1.$$

We see from (6.16) that the matrix A is decomposed into a weighted sum of matrices which have all the same norm, and the singular values are the weights. The main contributions in the sum are the terms with the largest singular values. *Therefore we may approximate A by a lower rank matrix by dropping the smallest singular values, i.e., replacing their values by zero.* In fact we have the

THEOREM 6.6. *Let $A \in \mathbb{R}^{m \times n}$ have rank r and let $A = U\Sigma V^\top$. Let \mathcal{M} denote the set of $m \times n$ matrices with rank $p < r$. The solution of*

$$\min_{X \in \mathcal{M}} \|A - X\|_2$$

is given by $A_p = \sum_{i=1}^p \sigma_i \mathbf{u}_i \mathbf{v}_i^\top$ and we have

$$\min_{X \in \mathcal{M}} \|A - X\|_2 = \|A - A_p\|_2 = \sigma_{p+1}.$$

PROOF. We have $U^\top A_p V = \text{diag}(\sigma_1, \dots, \sigma_p, 0, \dots, 0)$ thus $A_p \in \mathcal{M}$ and

$$\|A - A_p\|_2 = \sigma_{p+1}.$$

Let $B \in \mathcal{M}$ and let the linear independent vectors $\mathbf{x}_1, \dots, \mathbf{x}_{n-p}$ span the null space of B , so that $B\mathbf{x}_j = 0$. The two sets of vectors $\{\mathbf{x}_1, \dots, \mathbf{x}_{n-p}\}$ and $\{\mathbf{v}_1, \dots, \mathbf{v}_{p+1}\}$ contain altogether $n+1$ vectors. Hence, they must be linearly dependent, so we can write

$$\alpha_1 \mathbf{x}_1 + \alpha_2 \mathbf{x}_2 + \dots + \alpha_{n-p} \mathbf{x}_{n-p} + \beta_1 \mathbf{v}_1 + \beta_2 \mathbf{v}_2 + \dots + \beta_{p+1} \mathbf{v}_{p+1} = 0.$$

Moreover, not all $\alpha_i = 0$, otherwise the vectors \mathbf{v}_i would be linearly dependent! Denote by

$$\mathbf{h} = -\alpha_1 \mathbf{x}_1 - \alpha_2 \mathbf{x}_2 - \dots - \alpha_{n-p} \mathbf{x}_{n-p} = \beta_1 \mathbf{v}_1 + \beta_2 \mathbf{v}_2 + \dots + \beta_{p+1} \mathbf{v}_{p+1} \neq 0,$$

and form the unit vector $\mathbf{z} = \mathbf{h}/\|\mathbf{h}\|_2 = \gamma_1 \mathbf{v}_1 + \gamma_2 \mathbf{v}_2 + \dots + \gamma_{p+1} \mathbf{v}_{p+1}$.

Then $B\mathbf{z} = 0$, $\mathbf{z}^\top \mathbf{z} = \gamma_1^2 + \dots + \gamma_{p+1}^2 = 1$ and

$$A\mathbf{z} = U\Sigma V^\top \mathbf{z} = \sum_{i=1}^{p+1} \sigma_i \gamma_i \mathbf{u}_i.$$

It follows that

$$\begin{aligned} \|A - B\|_2^2 &\geq \|(A - B)\mathbf{z}\|_2^2 = \|A\mathbf{z}\|_2^2 = \sum_{i=1}^{p+1} \sigma_i^2 \gamma_i^2 \\ &\geq \sigma_{p+1}^2 \sum_{i=1}^{p+1} \gamma_i^2 = \sigma_{p+1}^2 \|\mathbf{z}\|_2^2 = \sigma_{p+1}^2. \end{aligned}$$

Thus, the distance from A to any other matrix in \mathcal{M} is greater or equal to the distance to A_p . This proves the theorem. \square

6.3.1 Pseudoinverse

DEFINITION 6.1. (PSEUDOINVERSE) Let $A = U\Sigma V^\top$ be the singular value decomposition with

$$\Sigma = \begin{pmatrix} \Sigma_r \\ 0 \end{pmatrix} \in \mathbb{R}^{m \times n}, \quad \Sigma_r := \text{diag}(\sigma_1, \dots, \sigma_r, 0, \dots, 0) \in \mathbb{R}^{n \times n}$$

with $\sigma_1 \geq \dots \geq \sigma_r > 0$. Then the matrix $A^+ = V\Sigma^+U^\top$ with

$$\Sigma^+ = \begin{pmatrix} \Sigma_r^+ & 0 \end{pmatrix} \in \mathbb{R}^{n \times m}, \quad \Sigma_r^+ := \text{diag}\left(\frac{1}{\sigma_1}, \dots, \frac{1}{\sigma_r}, 0, \dots, 0\right) \in \mathbb{R}^{n \times n} \quad (6.19)$$

is called the pseudoinverse of A .

We have discussed the SVD only for the case in which $A \in \mathbb{R}^{m \times n}$ with $m \geq n$. This was mainly for simplicity, since the SVD exists for any matrix: if $A = U\Sigma V^\top$, then $A^\top = V\Sigma^\top U^\top$ is the singular value decomposition of $A^\top \in \mathbb{R}^{n \times m}$. Usually the SVD is computed such that the singular values are ordered decreasingly. The representation $A^+ = V\Sigma^+U^\top$ of the pseudoinverse is thus already a SVD, except that the singular values $\frac{1}{\sigma_1}, \dots, \frac{1}{\sigma_r}$ are ordered increasingly. By simultaneously permuting rows and columns one can reorder the decomposition and bring it into standard form with decreasing elements in Σ^+ .

THEOREM 6.7. (PENROSE EQUATIONS) $Y = A^+$ is the only solution of the matrix equations

$$\begin{array}{ll} (i) & AYA = A \\ (iii) & (AY)^\top = AY \end{array} \quad \begin{array}{ll} (ii) & YAY = Y \\ (iv) & (YA)^\top = YA \end{array}$$

PROOF. It is simple to verify that A^+ is a solution: inserting the SVD e.g. into (i), we get

$$AA^+A = U\Sigma V^\top V\Sigma^+U^\top U\Sigma V^\top = U\Sigma\Sigma^+\Sigma V^\top = U\Sigma V^\top = A.$$

More challenging is to prove uniqueness. To do this, assume that Y is any

solution to (i)–(iv). Then

$$\begin{aligned}
 Y &= YAY \quad \text{because of (ii)} \\
 &= (YA)^\top Y = A^\top Y^\top Y \quad \text{because of (iv)} \\
 &= (AA^+A)^\top Y^\top Y = A^\top (A^+)^\top A^\top Y^\top Y \quad \text{because of (i)} \\
 &= A^\top (A^+)^\top YAY \quad \text{because of (iv)} \\
 &= A^\top (A^+)^\top Y = (A^+A)^\top Y \quad \text{because of (ii)} \\
 &= A^+AY \quad \text{because of (iv)} \\
 &= A^+AA^+AY \quad \text{because of (ii)} \\
 &= A^+(AA^+)^\top (AY)^\top = A^+(A^+)^\top A^\top Y^\top A^\top \quad \text{because of (iii)} \\
 &= A^+(A^+)^\top A^\top \quad \text{because of (i)} \\
 &= A^+(AA^+)^\top = A^+AA^+ \quad \text{because of (iii)} \\
 Y &= A^+ \quad \text{because of (ii)}
 \end{aligned}$$

□

6.3.2 Fundamental Subspaces

There are four *fundamental subspaces* associated with a matrix $A \in \mathbb{R}^{m \times n}$:

DEFINITION 6.2. (FUNDAMENTAL SUBSPACES OF A MATRIX)

1. $\mathcal{R}(A) = \{\mathbf{y} | \mathbf{y} = A\mathbf{x}, \quad \mathbf{x} \in \mathbb{R}^n\} \subset \mathbb{R}^m$ is the range or column space.
2. $\mathcal{R}(A)^\perp$ the orthogonal complement of $\mathcal{R}(A)$.
If $\mathbf{z} \in \mathcal{R}(A)^\perp$ then $\mathbf{z}^\top \mathbf{y} = 0, \quad \forall \mathbf{y} \in \mathcal{R}(A)$.
3. $\mathcal{R}(A^\top) = \{\mathbf{z} | \mathbf{z} = A^\top \mathbf{y}, \quad \mathbf{y} \in \mathbb{R}^m\} \subset \mathbb{R}^n$ the row space.
4. $\mathcal{N}(A) = \{\mathbf{x} | A\mathbf{x} = 0\}$ the null space.

THEOREM 6.8. The following relations hold:

1. $\mathcal{R}(A)^\perp = \mathcal{N}(A^\top)$. Thus $\mathbb{R}^m = \mathcal{R}(A) \oplus \mathcal{N}(A^\top)$.
2. $\mathcal{R}(A^\top)^\perp = \mathcal{N}(A)$. Thus $\mathbb{R}^n = \mathcal{R}(A^\top) \oplus \mathcal{N}(A)$.

In other words, \mathbb{R}^m can be written as a direct sum of the range of A and the null space of A^\top , and an analogous result holds for \mathbb{R}^n .

PROOF. Let $\mathbf{z} \in \mathcal{R}(A)^\perp$. Then for any $\mathbf{x} \in \mathbb{R}^n$, we have $A\mathbf{x} \in \mathcal{R}(A)$, so by definition we have

$$0 = (A\mathbf{x})^\top \mathbf{z} = \mathbf{x}^\top (A^\top \mathbf{z}).$$

Since this is true for all \mathbf{x} , it follows that $A^\top \mathbf{z} = 0$, which means $\mathbf{z} \in \mathcal{N}(A^\top)$ and therefore $\mathcal{R}(A)^\perp \subset \mathcal{N}(A^\top)$.

On the other hand, let $\mathbf{y} \in \mathcal{R}(A)$ and $\mathbf{z} \in \mathcal{N}(A^\top)$. Then we have

$$\mathbf{y}^\top \mathbf{z} = (A\mathbf{x})^\top \mathbf{z} = \mathbf{x}^\top (A^\top \mathbf{z}) = \mathbf{x}^\top \mathbf{0} = 0$$

which means that $\mathbf{z} \in \mathcal{R}(A)^\perp$. Thus also $\mathcal{N}(A^\top) \subset \mathcal{R}(A)^\perp$.

The second statement is verified in the same way. \square

One way to understand the problem of finding least squares approximations is via projections onto the above subspaces. Recall that $P : \mathbb{R}^n \rightarrow \mathbb{R}^n$ is a *projector* onto a subspace $V \subset \mathbb{R}^n$ if $P^2 = P$ and $\mathcal{R}(P) = V$. Additionally, if P is symmetric, then it is an *orthogonal projector*. The following lemma shows a few properties of orthogonal projectors.

LEMMA 6.1. *Let $V \neq 0$ be a non-trivial subspace of \mathbb{R}^n . If P_1 is an orthogonal projector onto V , then $\|P_1\|_2 = 1$ and $P_1 \mathbf{v} = \mathbf{v}$ for all $\mathbf{v} \in V$. Moreover, if P_2 is another orthogonal projector onto V , then $P_1 = P_2$.*

PROOF. We first show that $P_1 \mathbf{v} = \mathbf{v}$ for all $\mathbf{v} \in V$. Let $0 \neq \mathbf{v} \in V$. Since $V = \mathcal{R}(P_1)$, there exists $\mathbf{x} \in \mathbb{R}^n$ such that $\mathbf{v} = P_1 \mathbf{x}$. Thus,

$$P_1 \mathbf{v} = P_1^2 \mathbf{x} = P_1 \mathbf{x} = \mathbf{v}.$$

Taking norms on both sides gives

$$\|P_1 \mathbf{v}\|_2 = \|\mathbf{v}\|_2,$$

which implies $\|P_1\|_2 \geq 1$. To show that $\|P_1\|_2 = 1$, let $\mathbf{y} \in \mathbb{R}^n$ be arbitrary. Then

$$\|P_1 \mathbf{y}\|_2^2 = \mathbf{y}^\top P_1^\top P_1 \mathbf{y} = \mathbf{y}^\top P_1^2 \mathbf{y} = \mathbf{y}^\top P_1 \mathbf{y}.$$

The Cauchy-Schwarz inequality now gives

$$\|P_1 \mathbf{y}\|_2^2 \leq \|\mathbf{y}\|_2 \|P_1 \mathbf{y}\|_2,$$

which shows, upon dividing both sides by $\|P_1 \mathbf{y}\|_2$, that $\|P_1 \mathbf{y}\|_2 \leq \|\mathbf{y}\|_2$. Hence, we conclude that $\|P_1\|_2 = 1$.

Now let P_2 be another orthogonal projector onto V . To show equality of the two projectors, we show that $(P_1 - P_2)\mathbf{y} = \mathbf{0}$ for all $\mathbf{y} \in \mathbb{R}^n$. Indeed, we have

$$\begin{aligned} \|(P_1 - P_2)\mathbf{y}\|_2^2 &= \mathbf{y}^\top (P_1 - P_2)^\top (P_1 - P_2) \mathbf{y} \\ &= \mathbf{y}^\top (P_1 - P_2)^2 \mathbf{y} \\ &= \mathbf{y}^\top (P_1 - P_1 P_2 - P_2 P_1 + P_2) \mathbf{y} \\ &= \mathbf{y}^\top (I - P_1) P_2 \mathbf{y} + \mathbf{y}^\top (I - P_2) P_1 \mathbf{y}. \end{aligned} \quad (6.20)$$

But for any $\mathbf{v} \in V$, we have

$$(I - P_1)\mathbf{v} = \mathbf{v} - P_1 \mathbf{v} = \mathbf{v} - \mathbf{v} = \mathbf{0},$$

and similarly for $I - P_2$. Since $P_2 \mathbf{y} \in V$, we have $(I - P_1)P_2 \mathbf{y} = 0$, so the first term in (6.20) vanishes. Exchanging the roles of P_2 and P_1 shows that the second term in (6.20) also vanishes, so $P_1 = P_2$. \square

Thanks to the above lemma, we see that the orthogonal projector onto a given V is in fact unique; we denote this projector by P_V . With the help of the pseudoinverse, we can describe *orthogonal projectors* onto the fundamental subspaces of A .

THEOREM 6.9. (PROJECTORS ONTO FUNDAMENTAL SUBSPACES)

1. $P_{\mathcal{R}(A)} = AA^+$
2. $P_{\mathcal{R}(A^\top)} = A^+A$
3. $P_{\mathcal{N}(A^\top)} = I - AA^+$
4. $P_{\mathcal{N}(A)} = I - A^+A$

PROOF. We prove only the first relation; the other proofs are similar. Because of Relation (iii) in Theorem 6.7 we have $(AA^+)^\top = AA^+$. Thus $P_{\mathcal{R}(A)}$ is symmetric. Furthermore $(AA^+)(AA^+) = (AA^+A)A^+ = AA^+$ because of (i). Thus $P_{\mathcal{R}(A)}$ is symmetric and idempotent and is therefore an orthogonal projector. Now let $\mathbf{y} = A\mathbf{x} \in \mathcal{R}(A)$; then $P_{\mathcal{R}(A)}\mathbf{y} = AA^+\mathbf{y} = AA^+A\mathbf{x} = A\mathbf{x} = \mathbf{y}$. So elements in $\mathcal{R}(A)$ are projected onto themselves. Finally take $\mathbf{z} \perp \mathcal{R}(A) \iff A^\top \mathbf{z} = 0$ then $P_{\mathcal{R}(A)}\mathbf{z} = AA^+\mathbf{z} = (AA^+)^\top \mathbf{z} = (A^+)^\top A^\top \mathbf{z} = 0$. \square

Note that the projectors can be computed using the SVD. Let $U_1 \in \mathbb{R}^{m \times r}$, $U_2 \in \mathbb{R}^{m \times (n-r)}$, $V_1 \in \mathbb{R}^{n \times r}$, $V_2 \in \mathbb{R}^{n \times (n-r)}$ and $\Sigma_r \in \mathbb{R}^{r \times r}$ in the following SVD

$$A = \begin{pmatrix} U_1 & U_2 \end{pmatrix} \begin{pmatrix} \Sigma_r & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} V_1^\top \\ V_2^\top \end{pmatrix}.$$

Then inserting this decomposition into the expressions for the projectors of Theorem 6.9 we obtain:

1. $P_{\mathcal{R}(A)} = U_1 U_1^\top$
2. $P_{\mathcal{R}(A^\top)} = V_1 V_1^\top$
3. $P_{\mathcal{N}(A^\top)} = U_2 U_2^\top$
4. $P_{\mathcal{N}(A)} = V_2 V_2^\top$

6.3.3 Solution of the Linear Least Squares Problem

We are now ready to describe the general solution for the linear least squares problem. We are given a system of equations with more equations than unknowns,

$$A\mathbf{x} \approx \mathbf{b}.$$

In general \mathbf{b} will not be in $\mathcal{R}(A)$ and therefore the system will not have a solution. A consistent system can be obtained if we project \mathbf{b} onto $\mathcal{R}(A)$:

$$A\mathbf{x} = AA^+\mathbf{b} \iff A(\mathbf{x} - A^+\mathbf{b}) = 0.$$

We conclude that $\mathbf{x} - A^+\mathbf{b} \in \mathcal{N}(A)$. That means

$$\mathbf{x} - A^+\mathbf{b} = (I - A^+A)\mathbf{w}$$

where we have generated an element in $\mathcal{N}(A)$ by projecting an arbitrary vector \mathbf{w} onto it. Thus we have shown

THEOREM 6.10. (GENERAL LEAST SQUARES SOLUTION) *The general solution of the linear least squares problem $A\mathbf{x} \approx \mathbf{b}$ is*

$$\mathbf{x} = A^+\mathbf{b} + (I - A^+A)\mathbf{w}, \quad \mathbf{w} \text{ arbitrary.} \quad (6.21)$$

Using the expressions for projectors from the SVD we obtain for the general solution

$$\mathbf{x} = V_1 \Sigma_r^{-1} U_1^\top \mathbf{b} + V_2 \mathbf{c} \quad (6.22)$$

where we have introduced the arbitrary vector $\mathbf{c} := V_2^\top \mathbf{w}$. Notice that if we calculate $\|\mathbf{x}\|_2^2$ using e.g. (6.21), we obtain

$$\begin{aligned} \|\mathbf{x}\|_2^2 &= \|A^+\mathbf{b}\|_2^2 + 2\mathbf{w}^\top \underbrace{(I - A^+A)^\top A^+ \mathbf{b}}_{=0} + \|(I - A^+A)\mathbf{w}\|_2^2 \\ &= \|A^+\mathbf{b}\|_2^2 + \|(I - A^+A)\mathbf{w}\|_2^2 \geq \|A^+\mathbf{b}\|_2^2. \end{aligned}$$

This calculation shows that any solution to the least squares problem must have norm greater than or equal to that of $A^+\mathbf{b}$; in other words, the pseudoinverse produces the *minimum-norm* solution to the least squares problem $A\mathbf{x} \approx \mathbf{b}$. Thus, we have obtained an algorithm for computing both the general and the minimum norm solution of the linear least squares problem with (possibly) rank deficient coefficient matrix:

ALGORITHM 6.1.

General solution of the linear least squares problem
 $A\mathbf{x} \approx \mathbf{b}$

1. Compute the SVD: $[U, S, V] = \text{svd}(A)$.
 2. Make a rank decision, i.e. choose r such that $\sigma_r > 0$ and $\sigma_{r+1} = \dots = \sigma_n = 0$. This decision is necessary because rounding errors will prevent the zero singular values from being exactly zero.
 3. Set $V1 = V(:, 1:r)$, $V2 = V(:, r+1:n)$, $Sr = S(1:r, 1:r)$, $U1 = U(:, 1:r)$.
 4. The solution with minimal norm is $\mathbf{x}_m = V1 * (Sr \setminus U1' * \mathbf{b})$.
 5. The general solution is $\mathbf{x} = \mathbf{x}_m + V2 * \mathbf{c}$ with an arbitrary $\mathbf{c} \in \mathbb{R}^{n-r}$.
-

If A has full rank ($\text{rank}(A) = n$) then the solution of the linear least squares problem is unique:

$$\mathbf{x} = A^+\mathbf{b} = V\Sigma^+U^\top \mathbf{b}.$$

The matrix A^+ is called pseudoinverse because in the full rank case the solution $A\mathbf{x} \approx \mathbf{b} \Rightarrow \mathbf{x} = A^+\mathbf{b}$ is the analogue of the solution $\mathbf{x} = A^{-1}\mathbf{b}$ of a linear system $A\mathbf{x} = \mathbf{b}$ with nonsingular matrix $A \in \mathbb{R}^{n \times n}$.

The general least squares solution presented in Theorem 6.10 is also valid for a consistent system of equations $A\mathbf{x} = \mathbf{b}$ where $m \leq n$, i.e. an under-determined linear system with fewer equations than unknowns. In this case the $\mathbf{x} = V_1 \Sigma_r^{-1} U_1^T \mathbf{b}$ solves the problem

$$\min \|\mathbf{x}\|_2 \quad \text{subject to } A\mathbf{x} = \mathbf{b}.$$

6.3.4 SVD and Rank

Consider the matrix

$$A = \begin{pmatrix} -1.9781 & 4.4460 & -0.1610 & -3.8246 & 3.8137 \\ 2.7237 & -2.3391 & 2.3753 & -0.0566 & -4.1472 \\ 1.6934 & -0.1413 & -1.5614 & -1.5990 & 1.7343 \\ 3.1700 & -7.1943 & -4.5438 & 6.5838 & -1.1887 \\ 0.3931 & -3.1482 & 3.1500 & 3.6163 & -5.9936 \\ -7.7452 & 2.9673 & -0.1809 & 4.6952 & 1.7175 \\ -1.9305 & 8.9277 & 2.2533 & -10.1744 & 5.2708 \end{pmatrix}.$$

The singular values are computed by `svd(A)` as

$$\begin{aligned} \sigma_1 &= 20.672908496836218 \\ \sigma_2 &= 10.575440102610981 \\ \sigma_3 &= 8.373932796689537 \\ \sigma_4 &= 0.000052201761324 \\ \sigma_5 &= 0.000036419750608 \end{aligned}$$

and we can observe a *gap* between σ_3 and σ_4 . The two singular values σ_4 and σ_5 are about 10^5 times smaller than σ_3 . Clearly the matrix A has rank 5. However, if the matrix elements comes from measured data and if the measurement uncertainty is $5 \cdot 10^{-5}$, one could reasonably suspect that A is in fact a perturbed representation of a rank-3 matrix, where the perturbation is due to measurement errors of the order of $5 \cdot 10^{-5}$. Indeed, reconstructing the matrix by setting $\sigma_4 = \sigma_5 = 0$ we get

```
[U,S,V]=svd(A);
S(4,4)=0; S(5,5)=0;
B=U*S*V'
```

We see no difference between A and B when using the MATLAB-format `short` because, according to Theorem 6.6, we have $\|A - B\|_2 = \sigma_4 = 5.2202 \cdot 10^{-5}$. Thus, in this case, one might as well declare that the matrix has *numerical rank* 3.

In general, if there is a distinct gap between the singular values one can define a threshold and remove small nonzero singular values which only occur

because of the rounding effects of finite precision arithmetic or maybe because of measurement errors in the data. The default tolerance in MATLAB for the rank command is `tol=max(size(A))*eps(norm(A))`. Smaller singular values are considered to be zero.

There are full rank matrices whose singular values decrease to zero with no distinct gap. It is well known that the Hilbert matrix is positive definite (see Problem 7.1 in Chapter 7). The MATLAB statement `eig(hilb(14))` gives us as smallest eigenvalue (which is here equal to σ_{14}) the negative value $-3.4834 \cdot 10^{-17}$! With `svd(hilb(14))` we get $\sigma_{14} = 3.9007 \cdot 10^{-18}$ which is also not correct. Using the MAPLE commands

```
with(LinearAlgebra);
Digits:=40;
n:=14;
A:=Matrix(n,n);
for i from 1 to n do
  for j from 1 to n do
    A[i,j]:=1/(i+j-1);
  end do;
end do;
evalm(evalf(Eigenvalues(A)));
```

we obtain for $n = 14$ the singular values

```
1.8306
4.1224 · 10-01
5.3186 · 10-02
4.9892 · 10-03
3.6315 · 10-04
2.0938 · 10-05
9.6174 · 10-07
3.5074 · 10-08
1.0041 · 10-09
2.2100 · 10-11
3.6110 · 10-13
4.1269 · 10-15
2.9449 · 10-17
9.8771 · 10-20
```

For `A=hilb(14)` MATLAB computes `rank(A)=12` which is mathematically not correct but a reasonable numerical rank for such an ill-conditioned matrix. The SVD is the best tool to assign numerically a rank to a matrix.

6.4 Condition of the Linear Least Squares Problem

The principle of Wilkinson states that *the result of a numerical computation is the result of an exact computation for a slightly perturbed problem* (see Section 2.7). This result allows us to estimate the influence of finite precision arithmetic. A problem is said to be well conditioned if the results do not

differ too much when solving a perturbed problem. For an ill-conditioned problem the solution of a perturbed problem may be very different.

Consider a system of linear equations $A\mathbf{x} = \mathbf{b}$ with $A \in \mathbb{R}^{n \times n}$ non-singular and a perturbed system $(A + \epsilon E)\mathbf{x}(\epsilon) = \mathbf{b}$, where ϵ is small, e.g. the machine precision. How do the solutions $\mathbf{x}(\epsilon)$ and $\mathbf{x} = \mathbf{x}(0)$ differ? We have already shown one way of estimating this difference in Chapter 3 (cf. Theorem 3.5), but here we illustrate another technique, which we will apply in the next section to the linear least squares problem. Let us consider the expansion

$$\mathbf{x}(\epsilon) = \mathbf{x}(0) + \dot{\mathbf{x}}(0)\epsilon + O(\epsilon^2).$$

The derivative $\dot{\mathbf{x}}(0)$ is obtained by differentiating:

$$\begin{aligned} (A + \epsilon E)\mathbf{x}(\epsilon) &= \mathbf{b} \\ E\mathbf{x}(\epsilon) + (A + \epsilon E)\dot{\mathbf{x}}(\epsilon) &= 0 \\ \Rightarrow \dot{\mathbf{x}}(0) &= -A^{-1}E\mathbf{x}(0). \end{aligned}$$

Thus, we get

$$\mathbf{x}(\epsilon) = \mathbf{x}(0) - A^{-1}E\mathbf{x}(0)\epsilon + O(\epsilon^2).$$

Neglecting $O(\epsilon^2)$ and taking norms, we get

$$\|\mathbf{x}(\epsilon) - \mathbf{x}(0)\|_2 \leq \|A^{-1}\|_2 \|\epsilon E\|_2 \|\mathbf{x}(0)\|_2.$$

From the last equation, we conclude that the relative error satisfies

$$\frac{\|\mathbf{x}(\epsilon) - \mathbf{x}\|_2}{\|\mathbf{x}\|_2} \leq \underbrace{\|A^{-1}\|_2 \|A\|_2}_{\kappa, \text{ condition number}} \frac{\|\epsilon E\|_2}{\|A\|_2}. \quad (6.23)$$

If we use the 2-norm as matrix norm,

$$\|A\|_2 := \max_{\mathbf{x} \neq 0} \frac{\|A\mathbf{x}\|_2}{\|\mathbf{x}\|_2} = \sigma_{\max}(A),$$

then the condition number is given by

$$\kappa = \frac{\sigma_{\max}(A)}{\sigma_{\min}(A)} = \text{cond}(A) \text{ in MATLAB.}$$

Thus, if $\|E\|_2 \approx \|A\|_2$, then according to the principle of Wilkinson we have to expect that the numerical solution may deviate by about κ units in the last digit from the exact solution.

We will now present an analogous result due to [56] for comparing the solutions of

$$\|\mathbf{b} - A\mathbf{x}\|_2 \longrightarrow \min \quad \text{and} \quad \|\mathbf{b} - (A + \epsilon E)\mathbf{x}(\epsilon)\|_2 \longrightarrow \min.$$

6.4.1 Differentiation of Pseudoinverses

Let A be a matrix whose elements are functions of k variables

$$A(\boldsymbol{\alpha}) \in \mathbb{R}^{m \times n}, \quad \boldsymbol{\alpha} = \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_k \end{pmatrix}.$$

DEFINITION 6.3. (FRÉCHET DERIVATIVE) *The Fréchet derivative of the matrix $A(\boldsymbol{\alpha})$ is the 3-dimensional tensor*

$$\mathcal{D}A(\boldsymbol{\alpha}) = \left(\frac{\partial a_{ij}(\boldsymbol{\alpha})}{\partial \alpha_s} \right), \quad s = 1, \dots, k$$

$\mathcal{D}A(\boldsymbol{\alpha})$ collects the gradients of all the matrix elements with respect to $\boldsymbol{\alpha}$.

We first derive some calculation rules for the operator \mathcal{D} :

1. If A is constant then $\mathcal{D}A = 0$.
2. Let $A \in \mathbb{R}^{m \times k}$ be constant. Then $\mathcal{D}[A\boldsymbol{\alpha}] = A$.
3. Product rule:

$$\mathcal{D}[A(\boldsymbol{\alpha})B(\boldsymbol{\alpha})] = \mathcal{D}A(\boldsymbol{\alpha})B(\boldsymbol{\alpha}) + A(\boldsymbol{\alpha})\mathcal{D}B(\boldsymbol{\alpha}).$$

Here, the product $\mathcal{D}A(\boldsymbol{\alpha})B(\boldsymbol{\alpha})$ means every layer of the tensor $\mathcal{D}A(\boldsymbol{\alpha})$ is multiplied by the matrix $B(\boldsymbol{\alpha})$. This rule is evident if we consider one element of the product which is differentiated with respect to α_s :

$$(\mathcal{D}[A(\boldsymbol{\alpha})B(\boldsymbol{\alpha})])_{i,j,s} = \frac{\partial}{\partial \alpha_s} \left(\sum_{t=1}^n a_{it} b_{tj} \right) = \sum_{t=1}^n \left(\frac{\partial a_{it}}{\partial \alpha_s} b_{tj} + a_{it} \frac{\partial b_{tj}}{\partial \alpha_s} \right).$$

4. Taylor expansion:

$$A(\boldsymbol{\alpha} + \mathbf{h}) = A(\boldsymbol{\alpha}) + \mathcal{D}A(\boldsymbol{\alpha})(\mathbf{h}) + O(\|\mathbf{h}\|_2^2).$$

Here, $\mathcal{D}A(\boldsymbol{\alpha})(\mathbf{h})$ is a matrix with the same size as $A(\boldsymbol{\alpha})$, where each matrix element is obtained by computing the scalar product between the gradient $\nabla_{\boldsymbol{\alpha}} a_{ij}$ (of length k) and \mathbf{h} , also of length k .

5. Derivative of the inverse:

$$\mathcal{D}[A^{-1}(\boldsymbol{\alpha})] = -A^{-1}(\boldsymbol{\alpha})\mathcal{D}A(\boldsymbol{\alpha})A^{-1}(\boldsymbol{\alpha}).$$

This results from differentiating $AA^{-1} = I$:

$$\mathcal{D}A(\boldsymbol{\alpha})A^{-1}(\boldsymbol{\alpha}) + A(\boldsymbol{\alpha})\mathcal{D}[A^{-1}(\boldsymbol{\alpha})] = 0.$$

THEOREM 6.11. (FRECHET PROJECTOR) *Let $\Omega \subset \mathbb{R}^k$ be an open set and let $A(\alpha) \in \mathbb{R}^{m \times n}$ have constant rank for all $\alpha \in \Omega$. Let $P_{\mathcal{R}(A)} = AA^+$ be the projector on the range of A . Then*

$$\mathcal{D}P_{\mathcal{R}(A)} = P_{\mathcal{N}(A^\top)}\mathcal{D}AA^+ + (P_{\mathcal{N}(A^\top)}\mathcal{D}AA^+)^{\top}, \quad \forall \alpha \in \Omega. \quad (6.24)$$

PROOF.

$$\begin{aligned} P_{\mathcal{R}(A)}A &= A \quad \text{because of Penrose Equation (i)} \\ \Rightarrow \mathcal{D}P_{\mathcal{R}(A)}A + P_{\mathcal{R}(A)}\mathcal{D}A &= \mathcal{D}A \\ \mathcal{D}P_{\mathcal{R}(A)}A &= (I - P_{\mathcal{R}(A)})\mathcal{D}A = P_{\mathcal{N}(A^\top)}\mathcal{D}A \end{aligned}$$

Multiplying the last equation from the right with A^+ and observing that $P_{\mathcal{R}(A)} = AA^+$ we get

$$\mathcal{D}P_{\mathcal{R}(A)}P_{\mathcal{R}(A)} = P_{\mathcal{N}(A^\top)}\mathcal{D}AA^+. \quad (6.25)$$

A projector is idempotent, therefore

$$\mathcal{D}P_{\mathcal{R}(A)} = \mathcal{D}\left(P_{\mathcal{R}(A)}^2\right) = \mathcal{D}P_{\mathcal{R}(A)}P_{\mathcal{R}(A)} + P_{\mathcal{R}(A)}\mathcal{D}P_{\mathcal{R}(A)}. \quad (6.26)$$

Furthermore, $P_{\mathcal{R}(A)}$ is symmetric, which implies each layer of the tensor $\mathcal{D}P_{\mathcal{R}(A)}$ is also symmetric. Therefore

$$(\mathcal{D}P_{\mathcal{R}(A)}P_{\mathcal{R}(A)})^{\top} = P_{\mathcal{R}(A)}\mathcal{D}P_{\mathcal{R}(A)}. \quad (6.27)$$

Substituting into (6.26) gives

$$\mathcal{D}P_{\mathcal{R}(A)} = \mathcal{D}P_{\mathcal{R}(A)}P_{\mathcal{R}(A)} + (\mathcal{D}P_{\mathcal{R}(A)}P_{\mathcal{R}(A)})^{\top},$$

which, together with (6.25), yields (6.24). \square

THEOREM 6.12. *Consider the projector $P_{\mathcal{R}(A^\top)} = A^+A$. With the same assumptions as in Theorem 6.11 we have*

$$\mathcal{D}P_{\mathcal{R}(A^\top)} = A^+\mathcal{D}AP_{\mathcal{N}(A)} + (A^+\mathcal{D}AP_{\mathcal{N}(A)})^{\top}. \quad (6.28)$$

PROOF. The proof follows from the proof of Theorem 6.11 by exchanging $A \leftrightarrow A^+$ and $A^\top \leftrightarrow A$. \square

THEOREM 6.13. *Let $\Omega \subset \mathbb{R}^k$ be an open set and let the matrix $A(\alpha)$ be Fréchet differentiable for all $\alpha \in \Omega$, with constant rank $r \leq \min(m, n)$. Then for every $\alpha \in \Omega$ we have*

$$\mathcal{D}A^+(\alpha) = -A^+\mathcal{D}AA^+ + A^+(A^+)^{\top}(\mathcal{D}A)^{\top}P_{\mathcal{N}(A^\top)} + P_{\mathcal{N}(A)}(\mathcal{D}A)^{\top}(A^+)^{\top}A^+. \quad (6.29)$$

PROOF. First, we have by Theorem 6.9

$$A^+P_{\mathcal{N}(A^\top)} = A^+(I - AA^+) = A^+ - A^+AA^+ = 0,$$

where the last equality follows from Penrose Equation (ii). Differentiating the above gives

$$\mathcal{D}A^+P_{\mathcal{N}(A^\top)} + A^+\mathcal{D}P_{\mathcal{N}(A^\top)} = 0$$

or

$$\mathcal{D}A^+P_{\mathcal{N}(A^\top)} = -A^+\mathcal{D}P_{\mathcal{N}(A^\top)}.$$

But $P_{\mathcal{N}(A^\top)} = I - P_{\mathcal{R}(A)}$, which implies

$$\mathcal{D}P_{\mathcal{N}(A^\top)} = -\mathcal{D}P_{\mathcal{R}(A)},$$

so by Theorem 6.11, we have

$$\begin{aligned} \mathcal{D}A^+P_{\mathcal{N}(A^\top)} &= A^+\mathcal{D}P_{\mathcal{R}(A)} \\ &= \underbrace{A^+P_{\mathcal{N}(A^\top)}}_{=0} \mathcal{D}AA^+ + A^+(A^+)^\top (\mathcal{D}A^+)^\top P_{\mathcal{N}(A^\top)} \\ &= A^+(A^+)^\top (\mathcal{D}A^+)^\top P_{\mathcal{N}(A^\top)}. \end{aligned} \quad (6.30)$$

Similarly, using the relation $P_{\mathcal{N}(A)}A^+ = 0$ and Theorem 6.12, we derive

$$\begin{aligned} P_{\mathcal{N}(A)}\mathcal{D}A^+ &= \mathcal{D}P_{\mathcal{R}(A^\top)}A^+ \\ &= A^+\mathcal{D}A \underbrace{P_{\mathcal{N}(A)}A^+}_{=0} + P_{\mathcal{N}(A)}(\mathcal{D}A)^\top (A^+)^\top A^+ \\ &= P_{\mathcal{N}(A)}(\mathcal{D}A)^\top (A^+)^\top A^+ \end{aligned} \quad (6.31)$$

Finally, differentiating the relation $A^+ = A^+AA^+$ gives

$$\begin{aligned} \mathcal{D}A^+ &= (\mathcal{D}A^+)AA^+ + A^+(\mathcal{D}A)A^+ + A^+A(\mathcal{D}A^+) \\ &= \mathcal{D}A^+(AA^+ - I) + A^+(\mathcal{D}A)A^+ + (A^+A - I)\mathcal{D}A^+ + 2\mathcal{D}A^+ \\ &= 2\mathcal{D}A^+ + A^+(\mathcal{D}A)A^+ - \mathcal{D}A^+P_{\mathcal{N}(A^\top)} - P_{\mathcal{N}(A)}\mathcal{D}A^+. \end{aligned}$$

Thus, after rearranging we get

$$\mathcal{D}A^+ = -A^+\mathcal{D}AA^+ + \mathcal{D}A^+P_{\mathcal{N}(A^\top)} + P_{\mathcal{N}(A)}\mathcal{D}A^+. \quad (6.32)$$

Substituting (6.30) and (6.31) into (6.32) yields the desired result. \square

6.4.2 Sensitivity of the Linear Least Squares Problem

In this section, we want to compare the solution of the least squares problem $\|\mathbf{b} - A\mathbf{x}\|_2$ to that of the perturbed problem $\|\mathbf{b} - (A + \epsilon E)\mathbf{x}(\epsilon)\|_2$. Define $A(\epsilon) = A + \epsilon E$. The solution of the perturbed problem is

$$\mathbf{x}(\epsilon) = A(\epsilon)^+ \mathbf{b} = \left[A(0)^+ + \frac{dA^+}{d\epsilon} \epsilon + O(\epsilon^2) \right] \mathbf{b} = \mathbf{x} + \epsilon \frac{dA^+}{d\epsilon} \mathbf{b} + O(\epsilon^2). \quad (6.33)$$

Applying Theorem 6.13, we get

$$\frac{dA^+}{d\epsilon} = -A^+ \frac{dA}{d\epsilon} A^+ + A^+ A^{+\top} \frac{dA^\top}{d\epsilon} P_{\mathcal{N}(A^\top)} + P_{\mathcal{N}(A)} \frac{dA^\top}{d\epsilon} A^{+\top} A^+.$$

Multiplying from the right with \mathbf{b} and using $dA/d\epsilon = E$, $A^+ \mathbf{b} = \mathbf{x}$ and

$$P_{\mathcal{N}(A^\top)} \mathbf{b} = (I - AA^+) \mathbf{b} = \mathbf{b} - A\mathbf{x} = \mathbf{r},$$

we get

$$\frac{dA^+}{d\epsilon} \mathbf{b} = -A^+ E \mathbf{x} + A^+ A^{+\top} E^\top \mathbf{r} + P_{\mathcal{N}(A)} E^\top A^{+\top} \mathbf{x}.$$

Introducing this into (6.33), we obtain

$$\mathbf{x}(\epsilon) - \mathbf{x} = \epsilon \left(-A^+ E \mathbf{x} + P_{\mathcal{N}(A)} E^\top A^{+\top} \mathbf{x} + A^+ A^{+\top} E^\top \mathbf{r} \right) + O(\epsilon^2).$$

Neglecting the term $O(\epsilon^2)$ and taking norms, we get

$$\begin{aligned} \|\mathbf{x}(\epsilon) - \mathbf{x}\|_2 \leq |\epsilon| \left(\|A^+\|_2 \|E\|_2 \|\mathbf{x}\|_2 + \|P_{\mathcal{N}(A)}\|_2 \|E^\top\|_2 \|A^{+\top}\|_2 \|\mathbf{x}\|_2 \right. \\ \left. + \|A^+\|_2 \|A^{+\top}\|_2 \|E^\top\|_2 \|\mathbf{r}\|_2 \right). \end{aligned}$$

Introducing the *condition number*

$$\kappa := \|A\|_2 \|A^+\|_2 = \frac{\sigma_1(A)}{\sigma_r(A)},$$

and observing that $\|P_{\mathcal{N}(A)}\|_2 = 1$ we get the estimate:

THEOREM 6.14. (GOLUB-PEREYRA 1973)

$$\frac{\|\mathbf{x}(\epsilon) - \mathbf{x}\|_2}{\|\mathbf{x}\|_2} \leq \left(2\kappa + \kappa^2 \frac{\|\mathbf{r}\|_2}{\|A\|_2 \|\mathbf{x}\|_2} \right) \frac{\|\epsilon E\|_2}{\|A\|_2} + O(\epsilon^2). \quad (6.34)$$

Equation (6.34) tells us again what accuracy we can expect from the numerical solution. Here, we have to distinguish between good and bad models: when the model is good, the residual $\|\mathbf{r}\|_2$ must be small, since it is possible to find parameters that fit the data well. In this case, the error in the solution may deviate by about κ units in the last digit from the exact solution, just like for linear equations (6.23). However, when the model is bad, i.e. when $\|\mathbf{r}\|_2$ is large, the condition becomes much worse, so we must expect a larger error in the computed solution.

6.4.3 Normal Equations and Condition

If we want to solve $A\mathbf{x} \approx \mathbf{b}$ numerically using the normal equations, we have to expect worse numerical results than predicted by (6.34), even for good models. Indeed, if $A = U\Sigma V^\top$ has rank n then

$$\kappa(A^\top A) = \kappa(V\Sigma^\top U^\top U\Sigma V^\top) = \kappa(V\Sigma^\top \Sigma V^\top) = \frac{\sigma_1^2}{\sigma_n^2} = \kappa(A)^2. \quad (6.35)$$

Thus, forming $A^\top A$ leads to a matrix with a squared condition number compared to the original matrix A . Intuitively, one also sees that forming $A^\top A$ may result in a loss of information, as shown by the following famous example by P. L  uchli:

$$A = \begin{pmatrix} 1 & 1 \\ \delta & 0 \\ 0 & \delta \end{pmatrix}, \quad A^\top A = \begin{pmatrix} 1 + \delta^2 & 1 \\ 1 & 1 + \delta^2 \end{pmatrix}.$$

If $\delta < \sqrt{\epsilon}$ (with ϵ = machine precision) then numerically $1 + \delta^2 = 1$ and the matrix of the normal equations becomes singular, even though A numerically has rank 2.

EXAMPLE 6.9. *We illustrate the theory with the following example. We generate a matrix A by taking the 6×5 segment of the inverse Hilbert-matrix divided by 35791. Next we construct a compatible right hand side \mathbf{y}_1 for the solution*

$$\mathbf{x} = \left(1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}\right)^\top.$$

Then we compare the numerical solution of $A\mathbf{x} \approx \mathbf{y}_1$ obtained by orthogonal transformations using MATLAB's `\` operator with the solution of the the normal equations.

In the second calculation we make the right hand side incompatible by adding a vector orthogonal to $\mathcal{R}(A)$. The amplification factor

$$2\kappa + \kappa^2 \frac{\|\mathbf{r}\|_2}{\|A\|_2 \|\mathbf{x}\|_2},$$

which is in the first case essentially equal to the condition number, grows in the second case because of the large residual and influences the accuracy of the numerical solution.

```
format compact, format long e
A=invhilb(6)/35791; A=A(:,1:5);           % generate matrix
y1=A*[1 1/2 1/3 1/4 1/5]';               % consistent right hand side
x11=A\y1;
K=cond(A);
factor1=K*(2+K*norm(y1-A*x11)/norm(A)/norm(x11));
x21=(A'*A)\(A'*y1);                     % solve with normal equations
```

```

dy=[-4620 -3960 -3465 -3080 -2772 -2520]';
Check=A'*dy % dy is orthogonal to R(A)
y2=y1+dy/35791; % inconsistent right hand side
x12=A\y2;
factor2=K*(2+K*norm(y2-A*x12)/norm(A)/norm(x12));
x22=(A'*A)\(A'*y2); % solve with normal equations
O_solutions=[x11 x12]
factors=[factor1, factor2]
NE_solutions=[x21 x22]
SquaredCondition=K^2

```

We get the results

```

Check =
    -8.526512829121202e-14
   -9.094947017729282e-13
    1.455191522836685e-11
    1.455191522836685e-11
         0
O_solutions =
    9.999999999129205e-01    1.000000008577035e+00
    4.999999999726268e-01    5.000000028639559e-01
    3.33333333220151e-01    3.333333345605716e-01
    2.499999999951682e-01    2.500000005367703e-01
    1.99999999983125e-01    2.000000001908185e-01
factors =
    9.393571042867742e+06    1.748388455005875e+10
NE_solutions =
    1.000039250650201e+00    1.000045308718515e+00
    5.000131762672949e-01    5.000152068771240e-01
    3.33389969395881e-01    3.33398690463054e-01
    2.500024822148246e-01    2.500028642353512e-01
    2.000008837044892e-01    2.000010196590261e-01
SquaredCondition =
    2.205979423052303e+13

```

We see that the first O_solution has about 6 incorrect decimals which correspond well to the amplifying factor $9.3 \cdot 10^6$. The second O_solution with incompatible right hand side has about 8 decimals incorrect, the factor in this case is $1.7 \cdot 10^{10}$ and would even predict a worse result.

The solutions with the normal equations, however, have about 12 incorrect decimal digits, regardless of whether the right hand side is compatible or not. This illustrates the influence of the squared condition number.

6.5 Algorithms Using Orthogonal Matrices

6.5.1 QR Decomposition

Consider a matrix $A \in \mathbb{R}^{m \times n}$ with $m \geq n$ and $\text{rank}(A) = n$. Then there exists the Cholesky decomposition of $A^T A = R^T R$ where R is an upper

triangular matrix. Since R is non-singular we can write $R^{-\top} A^{\top} A R^{-1} = I$ or

$$(AR^{-1})^{\top}(AR^{-1}) = I.$$

This means that the matrix $Q_1 := AR^{-1}$ has *orthogonal columns*. Thus we have found the QR decomposition

$$A = Q_1 R. \quad (6.36)$$

Here, $Q_1 \in \mathbb{R}^{m \times n}$ and $R \in \mathbb{R}^{n \times n}$. We can always augment Q_1 to an $m \times m$ orthogonal matrix $Q := [Q_1, Q_2]$ and instead consider the decomposition

$$A = [Q_1, Q_2] \begin{pmatrix} R \\ 0 \end{pmatrix}. \quad (6.37)$$

The decomposition (6.37) is what MATLAB computes with the command $[Q, R] = \text{qr}(A)$. The decomposition (6.37) exists for any matrix A with full column rank.

We have shown in Section 6.2 that for an orthogonal matrix B the problems

$$A\mathbf{x} \approx \mathbf{b} \quad \text{and} \quad BA\mathbf{x} \approx B\mathbf{b}$$

are equivalent. Now if $A = Q \begin{pmatrix} R \\ 0 \end{pmatrix}$, then $B = Q^{\top}$ is orthogonal and $A\mathbf{x} \approx \mathbf{b}$ and $Q^{\top} A\mathbf{x} \approx Q^{\top} \mathbf{b}$ are equivalent. But

$$Q^{\top} A = \begin{pmatrix} R \\ 0 \end{pmatrix}$$

and the equivalent system becomes

$$\begin{pmatrix} R \\ 0 \end{pmatrix} \mathbf{x} \approx \begin{pmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{pmatrix}, \quad \text{with} \quad \begin{pmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{pmatrix} = Q^{\top} \mathbf{b}.$$

The square of the norm of the residual,

$$\|\mathbf{r}\|_2^2 = \|\mathbf{y}_1 - R\mathbf{x}\|_2^2 + \|\mathbf{y}_2\|_2^2,$$

is obviously minimal for $\hat{\mathbf{x}}$ where

$$R\hat{\mathbf{x}} = \mathbf{y}_1, \quad \hat{\mathbf{x}} = R^{-1}\mathbf{y}_1 \quad \text{and} \quad \min \|\mathbf{r}\|_2 = \|\mathbf{y}_2\|_2.$$

This approach is numerically preferable to the normal equations, since it does not change the condition number. This can be seen by noting that the singular values are not affected by orthogonal transformations: If $A = U\Sigma V^{\top} = Q \begin{pmatrix} R \\ 0 \end{pmatrix}$ then the singular value decomposition of $\begin{pmatrix} R \\ 0 \end{pmatrix}$ is

$$\begin{pmatrix} R \\ 0 \end{pmatrix} = (Q^{\top} U) \Sigma V^{\top},$$

and thus R and A have the same singular values, which leads to

$$\kappa(A) = \kappa(R).$$

In the following section we will show how to compute the QR decomposition.

6.5.2 Method of Householder

DEFINITION 6.4. (ELEMENTARY HOUSEHOLDER MATRIX) An elementary Householder matrix is a matrix of the form $P = I - \mathbf{u}\mathbf{u}^\top$ with $\|\mathbf{u}\|_2 = \sqrt{2}$ is

Elementary Householder matrices have the following properties:

1. P is symmetric.

2. P is orthogonal, since

$$P^\top P = (I - \mathbf{u}\mathbf{u}^\top)(I - \mathbf{u}\mathbf{u}^\top) = I - \mathbf{u}\mathbf{u}^\top - \mathbf{u}\mathbf{u}^\top + \underbrace{\mathbf{u}\mathbf{u}^\top\mathbf{u}\mathbf{u}^\top}_2 = I.$$

3. $P\mathbf{u} = -\mathbf{u}$ and if $\mathbf{x} \perp \mathbf{u}$ then $P\mathbf{x} = \mathbf{x}$. If $\mathbf{y} = \alpha\mathbf{x} + \beta\mathbf{u}$ then $P\mathbf{y} = \alpha\mathbf{x} - \beta\mathbf{u}$. Thus P is a reflection across the hyperplane $\mathbf{u}^\top\mathbf{x} = 0$.

P will be used to solve the following basic problem: Given a vector \mathbf{x} , find an orthogonal matrix P such that

$$P\mathbf{x} = \begin{pmatrix} \sigma \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \sigma\mathbf{e}_1.$$

Since P is orthogonal we have $\|P\mathbf{x}\|_2^2 = \|\mathbf{x}\|_2^2 = \sigma^2$ thus $\sigma = \pm\|\mathbf{x}\|_2$. Furthermore $P\mathbf{x} = (I - \mathbf{u}\mathbf{u}^\top)\mathbf{x} = \mathbf{x} - \mathbf{u}(\mathbf{u}^\top\mathbf{x}) = \sigma\mathbf{e}_1$, thus $\mathbf{u}(\mathbf{u}^\top\mathbf{x}) = \mathbf{x} - \sigma\mathbf{e}_1$ and we obtain by normalizing

$$\mathbf{u} = \frac{\mathbf{x} - \sigma\mathbf{e}_1}{\|\mathbf{x} - \sigma\mathbf{e}_1\|_2} \sqrt{2}.$$

We can still choose the sign of σ , and we choose it such that no cancellation occurs in computing $\mathbf{x} - \sigma\mathbf{e}_1$,

$$\sigma = \begin{cases} \|\mathbf{x}\|_2, & x_1 < 0, \\ -\|\mathbf{x}\|_2, & x_1 \geq 0. \end{cases}$$

For the denominator we get $\|\mathbf{x} - \sigma\mathbf{e}_1\|_2^2 = (\mathbf{x} - \sigma\mathbf{e}_1)^\top(\mathbf{x} - \sigma\mathbf{e}_1) = \mathbf{x}^\top\mathbf{x} - 2\sigma\mathbf{e}_1^\top\mathbf{x} + \sigma^2$. Note that $-2\sigma\mathbf{e}_1^\top\mathbf{x} = 2|x_1|\|\mathbf{x}\|_2$, so the calculations simplify and we get

$$\mathbf{u} = \frac{\mathbf{x} - \sigma\mathbf{e}_1}{\sqrt{\|\mathbf{x}\|_2(|x_1| + \|\mathbf{x}\|_2)}}.$$

In order to apply this basic construction for the computation of the QR decomposition, we construct a sequence of n elementary matrices P_i :

$$P_i = \begin{pmatrix} I & 0 \\ 0 & I - \mathbf{u}_i\mathbf{u}_i^\top \end{pmatrix}.$$

We choose $\mathbf{u}_i \in \mathbb{R}^{m-i+1}$ such that zeros are introduced in the i -th column of A below the diagonal when multiplying $P_i A$. We obtain after n steps

$$P_n P_{n-1} \cdots P_1 A = \begin{pmatrix} R \\ 0 \end{pmatrix}$$

and, because each P_i is symmetric, Q becomes

$$Q = (P_n P_{n-1} \cdots P_1)^\top = P_1 P_2 \cdots P_n.$$

If we store the new diagonal elements (which are the diagonal of R) in a separate vector d we can store the Householder vectors \mathbf{u}_i in the same location where we introduce zeros in A . This leads to the following *implicit QR factorization* algorithm:

ALGORITHM 6.2. *Householder QR Decomposition*

```
function [A,d]=HouseholderQR(A);
% HOUSEHOLDERQR computes the QR-decomposition of a matrix
% [A,d]=HouseholderQR(A) computes an implicit QR-decomposition A=QR
% using Householder transformations. The output matrix A contains
% the Householder vectors u and the upper triangle of R. The
% diagonal of R is stored in the vector d.

[m,n]=size(A);
for j=1:n,
    s=norm(A(j:m,j));
    if s==0, error('rank(A)<n'), end
    if A(j,j)>=0, d(j)=-s; else d(j)=s; end
    fak=sqrt(s*(s+abs(A(j,j))));
    A(j,j)=A(j,j)-d(j);
    A(j:m,j)=A(j:m,j)/fak;
    if j<n, % transformation of the rest of the matrix G:=G-u*(u'*G)
        A(j:m,j+1:n)=A(j:m,j+1:n)-A(j:m,j)*(A(j:m,j)'+A(j:m,j+1:n));
    end
end
```

Algorithm `HouseholderQR` computes an upper triangular matrix R_h which is very similar to R_c obtained by the Cholesky decomposition $A^\top A = R_c^\top R_c$. The only difference is that R_h may have negative elements in the diagonal, whereas in R_c the diagonal entries are positive. Let D be a diagonal matrix with

$$d_{ii} = \begin{cases} 1 & r_{ii}^h > 0 \\ -1 & r_{ii}^h < 0 \end{cases}$$

then $R_c = D R_h$. The matrix Q is only implicitly available through the Householder vectors \mathbf{u}_i . This is not an issue because it is often unnecessary to compute and store the matrix Q explicitly; in many cases, Q is only

needed as an *operator* that acts on vectors by multiplication. Using the implicit representation we can, for instance, compute the transformed right hand side of the least square equations $\mathbf{y} = Q^T \mathbf{b}$ by applying the reflections $\mathbf{y} = P_n P_{n-1} \cdots P_1 \mathbf{b}$. This procedure is numerically preferable to forming the explicit matrix Q and then multiplying with \mathbf{b} .

ALGORITHM 6.3. Transformation $\mathbf{z} = Q^T \mathbf{y}$

```
function z=HouseholderQTy(A,y);
% HOUSEHOLDERQTY applies Householder reflections transposed
%   z=HouseholderQTy(A,y); computes z=Q'y using the Householder
%   reflections Q stored as vectors in the matrix A by
%   A=HousholderQR(A)

[m,n]=size(A); z=y;
for j=1:n,
    z(j:m)=z(j:m)-A(j:m,j)*(A(j:m,j)'+z(j:m));
end;
```

If we wish to compute $\mathbf{z} = Q\mathbf{y}$ then because $Q = P_1 P_2 \cdots P_n$ it is sufficient to reverse the order in the for-loop:

ALGORITHM 6.4. Transformation $\mathbf{z} = Q\mathbf{y}$

```
function z=HouseholderQy(A,y);
% HOUSEHOLDERQY applies Householder reflections
%   z=HouseholderQy(A,y); computes z=Qy using the Householder
%   reflections Q stored as vectors in the matrix A by
%   A=HousholderQR(A)

[m,n]=size(A); z=y;
for j=n:-1:1,
    z(j:m)=z(j:m)-A(j:m,j)*(A(j:m,j)'+z(j:m));
end;
```

EXAMPLE 6.10. We compute the QR decomposition of a section of the Hilbert matrix. We also compute the explicit matrix Q by applying the Householder transformations to the column vectors of the identity matrix:

```
m=8;n=6;
H=hilb(m); A=H(:,1:n);
[AA,d]=HouseholderQR(A)
Q=[];
for i=eye(m),                                     % compute Q explicit
    z=HouseholderQy(AA,i); Q=[Q z];
end
R=triu(AA);
```

```

for i=1:n, R(i,i)=d(i); end           % add diagonal to R
[norm(Q'*Q-eye(m)) norm(A-Q*R)]
[q,r]=qr(A);                         % compare with Matlab qr
[norm(q'*q-eye(m)) norm(A-q*r)]

```

The resulting matrix after the statement $[AA, d] = \text{HouseholderQR}(A)$ contains the Householder vectors and the upper part of R . The diagonal of R is stored in the vector d :

```

AA =
    1.3450   -0.7192   -0.5214   -0.4130   -0.3435   -0.2947
    0.3008    1.1852   -0.1665   -0.1612   -0.1512   -0.1407
    0.2005    0.3840   -1.0188    0.0192    0.0233    0.0254
    0.1504    0.3584    0.2452   -1.3185    0.0015    0.0023
    0.1203    0.3242    0.3945   -0.4332   -1.0779    0.0001
    0.1003    0.2925    0.4703   -0.2515    0.0111   -1.3197
    0.0859    0.2651    0.5059   -0.0801    0.3819   -0.5078
    0.0752    0.2417    0.5190    0.0641    0.8319    0.0235

d =
   -1.2359   -0.1499    0.0118    0.0007    0.0000    0.0000

ans =
    1.0e-15 *
    0.6834    0.9272

ans =
    1.0e-15 *
    0.5721    0.2461

```

We see that the results (orthogonality of Q and reproduction of A by QR) compare well with the MATLAB function `qr`.

6.5.3 Method of Givens

DEFINITION 6.5. (ELEMENTARY GIVENS ROTATION) An elementary Givens rotation is the matrix

$$G := G_{i,k}(\alpha) = \begin{bmatrix} 1 & & & & & & & & \\ & \ddots & & & & & & & \\ & & 1 & & & & & & \\ & & & \cos \alpha & & & \sin \alpha & & \\ & & & & 1 & & & & \\ & & & & & \ddots & & & \\ & & & & & & 1 & & \\ & & & & & & & \cos \alpha & \\ & & & & & & & & 1 \\ & & & & & & & & & \ddots & \\ & & & & & & & & & & 1 \end{bmatrix} \begin{matrix} \\ \\ \\ \leftarrow i \\ \\ \\ \leftarrow k \\ \\ \end{matrix}$$

$G_{i,k}(\alpha)$ differs from the identity matrix only in the two columns and rows i and k with the elements $\cos \alpha$ and $\sin \alpha$. It is an orthogonal matrix: the columns have norm one and different columns are orthogonal to each other. It is called a rotation matrix since if we multiply G by a vector \mathbf{x} , the result $\mathbf{y} = G\mathbf{x}$ is the vector \mathbf{x} rotated in the ik -plane by the angle α .

For least squares problems, it is convenient to work with a slight modification of these elementary matrices. This is because Givens rotation matrices are non-symmetric: thus, if $Q = G_3G_2G_1$ then $Q^\top = G_1^\top G_2^\top G_3^\top \neq G_1G_2G_3$.

DEFINITION 6.6. (ELEMENTARY GIVENS REFLECTION) *An elementary Givens reflection is the matrix*

$$S := S_{i,k}(\alpha) = \begin{pmatrix} 1 & & & & & & & & \\ & \ddots & & & & & & & \\ & & \cos \alpha & & & \sin \alpha & & & \\ & & & 1 & & & & & \\ & & & & \ddots & & & & \\ & & \sin \alpha & & & 1 & & & \\ & & & & & & -\cos \alpha & & \\ & & & & & & & 1 & \\ & & & & & & & & \ddots & \\ & & & & & & & & & 1 \end{pmatrix}$$

$S_{i,k}(\alpha)$ is a modified Givens rotation matrix: row k has been multiplied by -1 and $S_{i,k}(\alpha)$ is now symmetric. It has the following properties:

1. S is orthogonal.
2. SA changes only two rows in A :

$$\begin{aligned} \mathbf{a}_i^{\text{new}} &= c \mathbf{a}_i + s \mathbf{a}_k, \\ \mathbf{a}_k^{\text{new}} &= s \mathbf{a}_i - c \mathbf{a}_k, \end{aligned} \quad \text{where } c = \cos \alpha \text{ and } s = \sin \alpha.$$

3. Connection to Householder matrices: let

$$\mathbf{u} = (0, \dots, 0, \sin \frac{\alpha}{2}, 0, \dots, 0, -\cos \frac{\alpha}{2}, 0, \dots, 0)^\top \sqrt{2},$$

$$\text{then } I - \mathbf{u}\mathbf{u}^\top = S_{i,k}(\alpha).$$

Givens transformations allow us to solve the following basic problem: given a vector \mathbf{x} , we wish to find an S that annihilates the k -th element, i.e., in $\mathbf{x}^{\text{new}} := S\mathbf{x}$, we want to rotate x_k^{new} to zero.

Solution: Because $x_k^{\text{new}} = sx_i - cx_k = 0$

$$\Rightarrow \cot := \frac{x_i}{x_k}, \quad s := \frac{1}{\sqrt{1 + \cot^2}}, \quad c := s * \cot.$$

Note that we do not need to compute the angle α explicitly to determine the matrix $S_{i,k}(\alpha)$.

To compute the QR decomposition of a matrix A , we now can apply Givens reflections to introduce zeros below the diagonal. The following pseudo-code computes the decomposition columnwise:

```

for i=1:n
  for k=i+1:m
    A=S(i,k,alpha)*A;           % annihilate A(k,i)
  end;
end

```

Again we would like to store information about the rotation $S_{i,k}(\alpha)$ at the same place where we introduce the zero. The easiest would be to store the angle α , but we cannot since we do not compute it! Storing c and s separately would require space for two numbers instead of one. We could store only c and rederive s using $c^2 + s^2 = 1$, but this is numerically unstable if $|c| \approx 1$.

An elegant solution was proposed by G. W. Stewart [130]: we store the smaller of the two numbers c and s , and retrieve the other one in a numerically stable way from $c^2 + s^2 = 1$. In order to tell whether c or s has been stored, Stewart proposes to store s or $1/c$. More specifically, we have two possible formulas for computing $x_k^{\text{new}} = sx_i - cx_k = 0$:

1. $\cot = c/s = x_i/x_k$, giving $s = 1/\sqrt{1 + \cot^2}$ and $c = s * \cot$
2. $\tan = s/c = x_k/x_i$, giving $c = 1/\sqrt{1 + \tan^2}$ and $s = c * \tan$.

To avoid overflow, we will choose the one that gives an answer that is smaller than 1 for $|\cot|$ or $|\tan|$. We thus obtain the following pseudocode for computing and storing a reflection:

```

if x(k)==0           % nothing to do, S is the identity
  c=1; s=0; Store 0
elseif abs(x(k))>=abs(x(i))
  h=x(i)/x(k); s=1/sqrt(1+h^2); c=s*h;
  if c~0, Store 1/c else Store 1 end;
else
  h=x(k)/x(i); c=1/sqrt(1+h^2); s=c*h; Store s;
end

```

To reconstruct c and s from a stored number z we use the following pseudocode:

```

if z==1, c=0; s=1;
elseif abs(z)<1 then s=z; c=sqrt(1-s^2);
else c=1/z; s=sqrt(1-c^2)
end

```

We have now all the elements to write a MATLAB function to compute the QR decomposition using Givens reflections:

ALGORITHM 6.5. *Givens QR Decomposition*

```
function A=GivensQR(A);
% GIVENSQR Computes the QR-decomposition using Givens reflections
% A=GivensQR(A); computes the QR-decomposition of the matrix A using
% Givens reflections; after the decomposition, A contains the
% implicit QR-decomposition of A. Instead of the zeros the
% reflections are stored following a proposal of G. W. Stewart
% (cf. srotg in BLAS). The decomposition is computed column wise.

[m n]=size(A);
for i=1:n,
    for k=i+1:m,
        if A(k,i)==0, % Compute co and si
            co=1; si=0; z=0;
        else if abs(A(k,i))>=abs(A(i,i)),
            h=A(i,i)/A(k,i); % cot
            si=1/sqrt(1+h*h); co=si*h;
            if co~=0, z=1/co; else z=1; end
        else
            h=A(k,i)/A(i,i); % tan
            co=1/sqrt(1+h*h); si=co*h; z=si;
        end;
        A(i,i)=A(i,i)*co+A(k,i)*si;
        A(k,i)=z; % store co or si in A(k,i)
        if (si~=0)&(i<n), % Apply reflection
            S=[co,si;si,-co];
            A(i:k-i:k,i+1:n)=S*A(i:k-i:k,i+1:n);
        end;
    end
end
end
```

Note that in Algorithm 6.5, we make use of the MATLAB feature that allows us to select and overwrite several rows of a matrix at the same time. In a traditional programming language like Fortran, we would store the old rows in auxiliary vectors and then combine them to the new rows:

```
h1=A(i,i+1:n); h2=A(k,i+1:n);
A(i,i+1:n)=h1*co+h2*si;
A(k,i+1:n)=h1*si-h2*co;
```

In MATLAB, however, we can use the selection expression $A(i:k-i:k,i+1:n)$ to write the same computation in terms of the 2×2 -matrix S as

```
S=[co,si;si,-co];
A(i:k-i:k,i+1:n)=S*A(i:k-i:k,i+1:n);
```

The function `GivensQR(A)` computes the QR decomposition of A . After the function call, we have $R = \text{triu}(A(1:n, 1:n))$ and Q is only given implicitly as an operator. If we wish to compute $y = Q^T x$ then the following function can be used:

ALGORITHM 6.6. Transformation $y := Q^T x$

```
function y=GivensQTy(A,y);
% GIVENSQTY applies Givens rotations transposed
% y=GivensQTyqrgiv(A,y) computes y=Q'*y; using the Givens
% rotations Q stored in the matrix A computed by A=GivensQR(A).
% For y=Q*y the for loops must be processed in reverse order.

[m,n]=size(A);
for i=1:n,                                     % for i=n:-1:1,   for y=Q*y
    for k=i+1:m,                               % for k=m:-1:i+1, for y=Q*y
        if A(k,i)==1,                         % reconstruct co and si from A(k,i)
            co=0; si=1;
        else
            if abs(A(k,i))<1,
                si=A(k,i); co=sqrt(1-si*si);
            else
                co=1/A(k,i); si=sqrt(1-co*co);
            end;
        end;
        end;
        if si~=0,                             % Apply Givens reflector
            y(i:k-i:k)=[co,si;si,-co]*y(i:k-i:k);
        end
    end
end
```

EXAMPLE 6.11. We compute the same example as before with Householder:

```
format compact
m=8;n=6;
H=hilb(m); A=H(:,1:n);
AA=GivensQR(A)
R=triu(AA(1:n,1:n));
Qt=[];
for y=eye(m)
    z=GivensQTy(AA,y); Qt=[Qt, z];
end
Q=Qt';
[norm(Q'*Q-eye(m)) norm(A-Q(:,1:n)*R)]
[q,r]=qr(A);
[norm(q'*q-eye(m)) norm(A-q*r)]
```

The orthogonality of Q and the representation of $A = QR$ are perfect and compare well with the MATLAB built-in function `qr`:

```
AA =
    1.2359    0.7192    0.5214    0.4130    0.3435    0.2947
    0.4472   -0.1499   -0.1665   -0.1612   -0.1512   -0.1407
    0.2857    0.6805    0.0118    0.0192    0.0233    0.0254
    0.2095    0.5135    1.5948   -0.0007   -0.0015   -0.0023
    0.1653    0.4116    0.6363    1.7994    0.0000    0.0001
    0.1365    0.3431    0.5371    1.4204    1.9844   -0.0000
    0.1162    0.2941    0.4643    0.6192    1.5366    2.1541
    0.1011    0.2572    0.4086    0.5485    0.6768    1.6456
ans =
    1.0e-15 *
    0.3719    0.2115
ans =
    1.0e-15 *
    0.5721    0.2461
```

The method of Givens needs about twice as many operations as the method of Householder. However, it can be efficient for sparse matrices, where only a few zeros have to be introduced to transform the matrix into upper triangular form. Furthermore, the QR decomposition may be computed row-wise, which may be advantageous for linear least squares problems where the number of equations is not fixed and additional equations are generated by new measurements.

EXAMPLE 6.12. A matrix is said to be in upper Hessenberg form if it has zero entries below the first subdiagonal,

$$H = \begin{pmatrix} * & * & * & * & * \\ * & * & * & * & * \\ 0 & * & * & * & * \\ 0 & 0 & * & * & * \\ 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & 0 & * \end{pmatrix}.$$

Least squares problems of the type $H\mathbf{y} = \mathbf{r}$ with an upper Hessenberg matrix H arise naturally in the GMRES algorithm for solving linear systems iteratively, see Section 11.7.6. To obtain the QR decomposition of such a matrix using Givens reflections, let us first eliminate the (2,1) entry; this is done by choosing a reflection $S_{1,2} := S_{1,2}(\alpha_1)$ that operates on the first two rows only, with the angle α_1 chosen so that the (2,1) entry is eliminated. The

transformed matrix then becomes

$$H = \begin{pmatrix} * & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \\ 0 & 0 & * & * & * \\ 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & 0 & * \end{pmatrix}.$$

But now the remaining matrix $H_{2:n,2:n}$ is also in upper Hessenberg form, so we can continue this process and choose $S_{23}, \dots, S_{n,n+1}$ to transform H into an upper triangular matrix, i.e.,

$$\underbrace{S_{n,n+1} \cdots S_{12}}_{Q^T} H = \begin{bmatrix} R \\ 0 \end{bmatrix}.$$

Note that the k -th step requires at most $2(n - k + 1)$ additions and the same number of multiplications, so the total cost is $O(n^2)$, as opposed to $O(mn^2)$ for a general dense matrix.

Givens rotations are also useful for updating QR decompositions, which we will see in Section 6.5.8. Another different use of upper Hessenberg matrices arise in the context of eigenvalue problems, see Section 7.5.2.

6.5.4 Fast Givens

In some architectures (particularly older ones), multiplication is a much more costly operation than addition, so reducing the number of multiplications can lead to faster running times. In the method of Givens, each rotation affects two rows of the matrix and transforms

$$\begin{bmatrix} x_1, x_2, \dots, x_n \\ y_1, y_2, \dots, y_n \end{bmatrix} \quad \text{into} \quad \begin{array}{cc} cx_k + sy_k & \\ sx_k - cy_k & \end{array} \quad k = 1, \dots, n$$

which requires two multiplications per entry. The basic idea of Fast Givens is to consider the rows in factored form and delay some of the multiplications until the end of the algorithm, thus saving about half the multiplications in the process.

Let us apply a Givens reflection to two “scaled rows” in which the coefficients f_1, f_2 have been factored out:

$$\begin{pmatrix} c & s \\ s & -c \end{pmatrix} \begin{array}{l} f_1 [x_1, x_2, \dots, x_n] \\ f_2 [y_1, y_2, \dots, y_n] \end{array}$$

After the multiplication these rows are changed to

$$\begin{array}{cc} cf_1x_k + sf_2y_k & \\ sf_1x_k - cf_2y_k & \end{array} \quad k = 1, \dots, n$$

The goal now is to update the factored coefficient in such a way that one multiplication inside the row vectors is eliminated. We consider two possibilities, the first of which is

$$\begin{matrix} sf_2 & (\beta_1 x_k + y_k) \\ sf_1 & (x_k + \alpha_1 y_k) \end{matrix} \quad \text{with} \quad \beta_1 = \frac{cf_1}{sf_2}, \quad \alpha_1 = -\frac{cf_2}{sf_1}.$$

Notice that y_k in the first row no longer has a coefficient multiplying it, and neither does x_k in the second row.

Now recall that the Givens reflection is chosen to annihilate one element in the matrix, say the first element y_1 of the second line. For this, c and s have to be chosen so that $sf_1 x_1 - cf_2 y_1 = 0$, which implies

$$c = \frac{f_1 x_1}{\sqrt{f_1^2 x_1^2 + f_2^2 y_1^2}}, \quad s = \frac{f_2 y_1}{\sqrt{f_1^2 x_1^2 + f_2^2 y_1^2}}.$$

Thus,

$$\alpha_1 = -\frac{c}{s} \frac{f_2}{f_1} = -\frac{f_1 x_1}{f_2 y_1} \frac{f_2}{f_1} = -\frac{x_1}{y_1}, \quad \beta_1 = \frac{c}{s} \frac{f_1}{f_2} = \frac{f_1 x_1}{f_2 y_1} \frac{f_1}{f_2} = -\alpha_1 \frac{f_1^2}{f_2^2}$$

and the new scaling factors become

$$f_1^{\text{new}} = sf_2, \quad f_2^{\text{new}} = sf_1.$$

For the effective computation of β_1 , we only need the squares of the factors, so we will only store the quantities

$$d_i = \frac{1}{f_i^2}, \quad i = 1, 2.$$

Then using the expressions for s , α_1 and β_1

$$d_1^{\text{new}} = \frac{1}{s^2 f_2^2} = \frac{f_1^2 x_1^2 + f_2^2 y_1^2}{y_1^2 f_2^2 f_2^2} = d_2 \left(1 + \left(\frac{x_1}{y_1} \right)^2 \left(\frac{f_1}{f_2} \right)^2 \right) = d_2 (1 - \alpha_1 \beta_1).$$

Defining $\gamma_1 = -\alpha_1 \beta_1 = (f_1 x_1)^2 / (f_2 y_1)^2$ we obtain

$$d_1^{\text{new}} = d_2 (1 + \gamma_1) \quad \text{and similarly} \quad d_2^{\text{new}} = \frac{1}{s^2 f_1^2} = d_1 (1 + \gamma_1). \quad (6.38)$$

The second choice of factors is

$$\begin{matrix} cf_1 & (x_k + \beta_2 y_k) \\ cf_2 & (\alpha_2 x_k - y_k) \end{matrix} \quad \text{with} \quad \alpha_2 = \frac{sf_1}{cf_2} = -\frac{1}{\alpha_1}, \quad \beta_2 = \frac{sf_2}{cf_1} = \frac{1}{\beta_1}.$$

The equation $y_1^{\text{new}} = 0$ leads to the same expressions for c and s as before. The update of d_i becomes

$$d_1^{\text{new}} = \frac{1}{c^2 f_1^2} = d_1 \frac{f_1^2 x_1^2 + f_2^2 y_1^2}{f_1^2 x_1^2} = d_1 (1 + \gamma_2) \quad \text{with} \quad \gamma_2 = \frac{1}{\gamma_1}. \quad (6.39)$$

A similar computation yields

$$d_2^{\text{new}} = \frac{1}{c^2 f_2^2} = d_2(1 + \gamma_2).$$

Which of the two choices of factors should be used? We are working with scaled rows. Let $D = \text{diag}(d_1, \dots, d_m)$ then

$$Q^\top A = D^{-1/2} \begin{pmatrix} \tilde{R} \\ 0 \end{pmatrix}, \quad \text{with} \quad D^{-1/2} = \begin{pmatrix} f_1 & 0 & \cdots & 0 \\ 0 & f_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & f_m \end{pmatrix}.$$

Since $\gamma_i \geq 0$, the d_i grow after each reflection, meaning there is a danger of overflow. Because of $\gamma_1 \gamma_2 = 1$ it makes sense to choose the smaller $\gamma_i \leq 1$ to minimize the growth. However, even then it might still be necessary to monitor the growth and to multiply the equations with the factors in the diagonal matrix $D^{-1/2}$ from time to time. For simplicity, we choose not to do this in the program below: we simply reduce the system $Ax \approx b$ to

$$\begin{pmatrix} \tilde{R} \\ 0 \end{pmatrix} x \approx \begin{pmatrix} \tilde{y}_1 \\ \tilde{y}_2 \end{pmatrix}$$

The scaling factors are given by the vector \mathbf{d} , the diagonal of the matrix D , and the solution is obtained by solving $\tilde{R}x = \tilde{y}_1$. The update of the diagonals of D according to (6.38) and (6.39) can be combined simply by swapping the diagonal elements in the first case.

ALGORITHM 6.7. *Fast Givens*

```
function [d,R,y]=FastGivens(A,b)
% FASTGIVENS reduce linear system to upper triangular form.
% [d,R,y]=FastGivens(A,b) reduces Ax=B using fast Givens reflections
% to Rx=y. d contains the scaling factors. If [q,r]=qr(A) then
% abs(diag(d)^(-0.5)*R)=abs(r)

[m n]=size(A); d=ones(m,1);
for i=1:n,
    for k=i+1:m,
        if A(k,i)~=0,
            alpha=-A(i,i)/A(k,i); beta=-alpha*d(k)/d(i);
            gamma=-alpha*beta;
            if gamma<=1,
                A(i,i)=A(i,i)*beta+A(k,i);
                if i<n,
                    h=A(i,i+1:n)*beta+A(k,i+1:n);
                    A(k,i+1:n)=A(i,i+1:n)+A(k,i+1:n)*alpha;
```

```

        A(i,i+1:n)=h;
    end;
    h=b(i)*beta+b(k); b(k)=b(i)+b(k)*alpha; b(i)=h;
    h=d(i); d(i)=d(k); d(k)=h; % swap scaling factors
else
    alpha=-1/alpha; beta=1/beta; gamma=1/gamma;
    A(i,i)=A(i,i)+beta*A(k,i);
    if i<n,
        h=A(i,i+1:n)+A(k,i+1:n)*beta;
        A(k,i+1:n)=A(i,i+1:n)*alpha -A(k,i+1:n);
        A(i,i+1:n)=h;
    end;
    h=b(i)+b(k)*beta; b(k)=b(i)*alpha-b(k); b(i)=h;
end;
d(i)=d(i)*(1+gamma); d(k)=d(k)*(1+gamma); % update for both cases
end;
end
R=triu(A); y=b;

```

Note that in Algorithm 6.7, we did not write the row update in terms of multiplication by a 2×2 matrix, like we did in Algorithm 6.5. This is because we want to avoid the multiplications with the factor 1. To compute one standard Givens reflection, we need about $4n$ multiplications and $2n$ additions. Fast Givens reduces this to $2n$ multiplications and $2n$ additions. Experiments show that the speedup is not by a factor of 2, but only about 1.4 to 1.6. This is due to overhead computations and also because multiplications are no longer much more expensive than additions on newer computer architectures.

6.5.5 Gram-Schmidt Orthogonalization

Yet another algorithm for calculating the QR decomposition is based on the idea of constructing an orthogonal basis in $\mathcal{R}(A)$. An overview on algorithms is given in [116]. For that purpose, the column vectors of $A = [\mathbf{a}_1, \dots, \mathbf{a}_n]$ have to be orthogonalized. Assume that the orthonormal vectors $\mathbf{q}_1, \dots, \mathbf{q}_{k-1}$ have already been computed and that they span the same space as $\mathbf{a}_1, \dots, \mathbf{a}_{k-1}$. Now in order to construct the next basis vector \mathbf{q}_k , we take \mathbf{a}_k and subtract its projections onto the previous basis vectors. If \mathbf{a}_k was not linearly dependent from the previous vectors, the remainder will be nonzero and orthogonal to $\mathcal{R}(\mathbf{a}_1, \dots, \mathbf{a}_{k-1})$, so it can be normalized to give the new basis vector. Thus we have to perform three steps:

1. Compute the projections $r_{ik}\mathbf{q}_i$ with $r_{ik} = \mathbf{q}_i^\top \mathbf{a}_k$.
2. Subtract the projections:

$$\mathbf{b}_k := \mathbf{a}_k - \sum_{i=1}^{k-1} r_{ik} \mathbf{q}_i. \quad (6.40)$$

3. Normalize: $r_{kk} := \|\mathbf{b}_k\|_2$ and $\mathbf{q}_k = \mathbf{b}_k / r_{kk}$

If we solve (6.40) for \mathbf{a}_k then

$$\mathbf{a}_k = \sum_{i=1}^k r_{ik} \mathbf{q}_i, \quad k = 1, \dots, n \iff A = QR.$$

Thus, we obtain again the QR decomposition of A . The elements of the upper triangular matrix R are the coefficients of the various projections. The following algorithm `ClassicalGramSchmidt` computes the QR decomposition using this *classical Gram-Schmidt orthogonalization*:

ALGORITHM 6.8. *Classical Gram-Schmidt*

```
function [Q,R]=ClassicalGramSchmidt(A)
% CLASSICALGRAMSCHMIDT classical Gram-Schmidt orthogonalization
% [Q,R]=ClassicalGramSchmidt(A); computes the classical Gram-Schmidt
% orthogonalization of the vectors in the columns of the matrix A

[m,n]=size(A); R=zeros(n);
Q=A;
for k=1:n,
    for i=1:k-1,
        R(i,k)=Q(:,i)'*Q(:,k);
    end
    % remove these two lines for
    % modified-Gram-Schmidt
    for i=1:k-1,
        Q(:,k)=Q(:,k)-R(i,k)*Q(:,i);
    end
    R(k,k)=norm(Q(:,k)); Q(:,k)=Q(:,k)/R(k,k);
end
```

Note that `ClassicalGramSchmidt` is numerically unstable: if we take the 15×10 section of the Hilbert matrix and try to compute an orthogonal basis then we notice that the vectors of Q are not orthogonal at all:

```
>> m=15; n=10; H=hilb(m); A=H(:,1:n);
>> [Q R]=ClassicalGramSchmidt(A);
>> norm(Q'*Q-eye(n))
ans = 2.9971
>> norm(Q*R-A)
ans = 2.9535e-17
```

However, the relation $A = QR$ is correct to machine precision. The reason for the numerical instability is as follows: when computing \mathbf{b}_k using (6.40), cancellation can occur if the vectors are almost parallel. In that case, the result is a very small inaccurate \mathbf{b}_k which, after normalization, is no longer orthogonal on the subspace spanned by the previous vectors because of the inaccuracies.

An interesting modification partly remedies that problem: if we compute the projections $r_{ik} = \mathbf{q}_i^\top \mathbf{a}_k$ and subtract the projection immediately from \mathbf{a}_k

$$\mathbf{a}_k := \mathbf{a}_k - r_{ik} \mathbf{q}_i$$

then this has no influence on the numerical value of subsequent projections because

$$r_{i+1,k} = \mathbf{q}_{i+1}^\top (\mathbf{a}_k - r_{ik} \mathbf{q}_i) = \mathbf{q}_{i+1}^\top \mathbf{a}_k, \text{ since } \mathbf{q}_{i+1}^\top \mathbf{q}_i = 0.$$

Doing so, we reduce the norm of \mathbf{a}_k with each projection and get the *Modified Gram-Schmidt* Algorithm, which is obtained by just eliminating the two lines

```
end                                % remove for
for i = 1:k-1,                    % modified-Gram-Schmidt
```

in Algorithm ClassicalGramSchmidt:

ALGORITHM 6.9. *Modified Gram-Schmidt*

```
function [Q,R]=ModifiedGramSchmidt(A)
% MODIFIEDGRAMSCHMIDT modified Gram-Schmidt orthogonalization
% [Q,R]=ModifiedGramSchmidt(A); computes the modified Gram-Schmidt
% orthogonalization of the vectors in the columns of the matrix A

[m,n]=size(A); R=zeros(n);
Q=A;
for k=1:n,
    for i=1:k-1,
        R(i,k)=Q(:,i)'*Q(:,k);
        Q(:,k)=Q(:,k)-R(i,k)*Q(:,i);
    end
    R(k,k)=norm(Q(:,k)); Q(:,k)=Q(:,k)/R(k,k);
end
```

Now if we run our example again, we obtain better results than with classical Gram-Schmidt:

```
>> m=15; n=10; H=hilb(m); A=H(:,1:n);
>> [Q R]=ModifiedGramSchmidt(A);
>> norm(Q'*Q-eye(n))
ans = 1.7696e-05
>> norm(Q*R-A)
ans = 6.0510e-17
```

The approximate orthogonality of Q is now visible; however, the results are still not as good as with the algorithms of Householder or Givens.

Note that in Algorithm ModifiedGramSchmidt, we only process the k -th column of A in step k ; the columns $k+1, \dots, n$ remain unchanged during

this step. A mathematically and numerically identical version of modified Gram-Schmidt, popular in many textbooks, computes the projections for each new \mathbf{q}_k and subtracts them immediately from all the column vectors of the remaining matrix:

ALGORITHM 6.10.

*Modified Gram-Schmidt, version updating whole
remaining matrix*

```
function [Q,R]=ModifiedGramSchmidt2(A);
% MODIFIEDGRAMSCHMIDT2 modified Gram-Schmidt orthogonalization version 2
% [Q,R]=ModifiedGramSchmidt2(A); computes the modified Gram-Schmidt
% orthogonalization of the vectors in the columns of the matrix A by
% immediately updating all the remaining vectors as well during the
% process

[m,n]=size(A); R=zeros(n);
for k=1:n
    R(k,k)=norm(A(:,k));
    Q(:,k)=A(:,k)/R(k,k);
    R(k,k+1:n)=Q(:,k)'*A(:,k+1:n);
    A(:,k+1:n)=A(:,k+1:n)-Q(:,k)*R(k,k+1:n);
end
```

In the k -th step of Algorithm `ModifiedGramSchmidt2` we compute the vector \mathbf{q}_k from \mathbf{a}_k . Then the k -th row of R is computed and the rest of A is updated by

$$A(:, k+1:n) = (I - \mathbf{q}_k \mathbf{q}_k^\top) A(:, k+1:n).$$

But $(I - \mathbf{q}_k \mathbf{q}_k^\top)$ is the orthogonal projector on the subspace orthogonal to \mathbf{q}_k . This observation by Charles Sheffield in 1968 established a connection between modified Gram-Schmidt and the method of Householder. Consider the matrix

$$\tilde{A} = \begin{pmatrix} O \\ A \end{pmatrix}, \text{ where } O \text{ is the } n \times n \text{ zero matrix.}$$

If we apply the first Householder transformation on \tilde{A} , i.e. we construct a vector \mathbf{u} with $\|\mathbf{u}\|_2 = \sqrt{2}$ such that

$$(I - \mathbf{u} \mathbf{u}^\top) \begin{pmatrix} O \\ \mathbf{a}_1 \end{pmatrix} = \begin{pmatrix} \sigma \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

then

$$\mathbf{u} = \frac{\mathbf{x} - \sigma \mathbf{e}_1}{\sqrt{\|\mathbf{x}\|_2(x_1 + \|\mathbf{x}\|_2)}}, \text{ with } \mathbf{x} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ \mathbf{a}_1 \end{pmatrix}.$$

Since $x_1 = 0$ (no cancellation in the first component) and $\sigma = \pm\|\mathbf{x}\|_2 = \pm\|\mathbf{a}_1\|_2$ we get

$$\mathbf{u} = \begin{pmatrix} \mathbf{e}_1 \\ \frac{\mathbf{a}_1}{\|\mathbf{a}_1\|_2} \end{pmatrix} = \begin{pmatrix} \mathbf{e}_1 \\ \mathbf{q}_1 \end{pmatrix}$$

and \mathbf{q}_1 is the same vector as in modified Gram-Schmidt! The Householder transformation is

$$P_1 \begin{pmatrix} O \\ A \end{pmatrix} = \begin{pmatrix} O \\ A \end{pmatrix} - \begin{pmatrix} \mathbf{e}_1 \\ \mathbf{q}_1 \end{pmatrix} (\mathbf{e}_1^\top \mathbf{q}_1^\top) \begin{pmatrix} O \\ A \end{pmatrix} = \begin{pmatrix} -\mathbf{q}_1^\top A \\ O_{n-1 \times n} \\ (I - \mathbf{q}_1 \mathbf{q}_1^\top) A \end{pmatrix}. \quad (6.41)$$

Thus we obtain the negative first row of R with $-\mathbf{q}_1^\top A$ and the first transformation of the rest of the matrix $(I - \mathbf{q}_1 \mathbf{q}_1^\top)A$ as in modified Gram-Schmidt.

THEOREM 6.15. *The method of Householder applied to the matrix $\begin{pmatrix} O \\ A \end{pmatrix}$ is mathematically equivalent to modified Gram-Schmidt applied to A .*

After the call of `[AA,d]=HouseholderQR([zeros(n);A])` the matrix has been transformed to

$$AA = \begin{pmatrix} \tilde{R} \\ Q_h \end{pmatrix}, \text{ with } \text{diag}(\tilde{R}) = I.$$

R_h is obtained by taking the negative strict upper part of \tilde{R} and subtracting the diagonal elements stored in d . The orthogonal matrix Q_h is extracted from the Householder-vectors stored in AA . The matrices Q_h and R_h obtained this way by Householder transformations are the same matrices as computed by modified Gram-Schmidt. We illustrate this again with a sub-matrix of the Hilbert matrix:

```
format compact
m=7; n=4; H=hilb(m); A=H(:,1:n);
[Q1,R1]=ModifiedGramSchmidt2(A)
[AA,d]=HouseholderQR([zeros(n); A])
Q2=AA(n+1:n+m,:);
R2=-AA(1:n,1:n);
R2=R2-diag(diag(R2))-diag(d);
[norm(Q1-Q2) norm(R1-R2)]
```

We get the following results which illustrate well that both processes compute the same

```
Q1 =
    0.8133    -0.5438     0.1991    -0.0551
    0.4067     0.3033    -0.6886     0.4760
    0.2711     0.3939    -0.2071    -0.4901
    0.2033     0.3817     0.1124    -0.4396
    0.1627     0.3514     0.2915    -0.1123
    0.1356     0.3202     0.3892     0.2309
    0.1162     0.2921     0.4407     0.5206
```

```

R1 =
    1.2296    0.7116    0.5140    0.4059
         0    0.1449    0.1597    0.1536
         0         0    0.0108    0.0174
         0         0         0    0.0006

AA =
    1.0000   -0.7116   -0.5140   -0.4059
         0    1.0000   -0.1597   -0.1536
         0         0    1.0000   -0.0174
         0         0         0    1.0000
    0.8133   -0.5438    0.1991   -0.0551
    0.4067    0.3033   -0.6886    0.4760
    0.2711    0.3939   -0.2071   -0.4901
    0.2033    0.3817    0.1124   -0.4396
    0.1627    0.3514    0.2915   -0.1123
    0.1356    0.3202    0.3892    0.2309
    0.1162    0.2921    0.4407    0.5206

d =
   -1.2296   -0.1449   -0.0108   -0.0006

ans =
    1.0e-12 *
    0.1804    0.0001

```

6.5.6 Gram-Schmidt with Reorthogonalization

Our example indicated that modified Gram-Schmidt improves the orthogonality of Q in comparison with classical Gram-Schmidt a lot — however, it still cannot compete with Householder or Givens in that respect. In fact, for modified Gram-Schmidt the following estimate holds (with some constants c_1 and c_2 , the condition number $\kappa = \kappa(A)$ and the machine precision ε) [9]:

$$\|I - Q^T Q\|_2 \leq \frac{c_1}{1 - c_2 \kappa \varepsilon} \kappa \varepsilon.$$

Thus, we must expect a loss of orthogonality if the condition of the matrix A is bad. A remedy is to reorthogonalize the vector \mathbf{q}_k if it has been constructed from a small (an inaccurate) \mathbf{b}_k . If \mathbf{q}_k is *reorthogonalized* with respect to \mathbf{q}_i , $i = 1, \dots, k-1$ then for full-rank matrices *one* reorthogonalization is sufficient. That “twice is enough” principle is analyzed in [50].

Note that when we reorthogonalize, we must also update R . Let us consider for that purpose the QR decomposition of the matrix

$$B = [\mathbf{q}_1, \dots, \mathbf{q}_{k-1}, \mathbf{b}] = QR_1.$$

Then the following holds:

$$Q = [\mathbf{q}_1, \dots, \mathbf{q}_{k-1}, \mathbf{q}_k], \quad R_1 = \begin{pmatrix} 1 & & d_1 \\ & \ddots & \vdots \\ & & 1 & d_{k-1} \\ & & & \|\mathbf{u}\|_2 \end{pmatrix}$$

with $d_i = \mathbf{q}_i^\top \mathbf{b}$, $i = 1, \dots, k-1$ and $\mathbf{u} = \mathbf{b} - \sum_{i=1}^{k-1} d_i \mathbf{q}_i$.

If we choose not to normalize the last column of Q , then the decomposition is $B = \bar{Q}\bar{R}_1$

with $\bar{Q} = [\mathbf{q}_1, \dots, \mathbf{q}_{k-1}, \mathbf{u}]$ and

$$\bar{R}_1 = \begin{pmatrix} 1 & & d_1 \\ & \ddots & \vdots \\ & & 1 & d_{k-1} \\ & & & 1 \end{pmatrix} = I + \mathbf{d} \mathbf{e}_k^\top \quad \text{where } \mathbf{d} = \begin{pmatrix} d_1 \\ \vdots \\ d_{k-1} \\ 0 \end{pmatrix}.$$

Now assume the QR decomposition of $A = [\mathbf{a}_1, \dots, \mathbf{a}_k]$ is $A = BR_2$ and we want to reorthogonalize the last column of B , i.e. we decompose $B = \bar{Q}\bar{R}_1$,

$$A = BR_2 = \bar{Q}\bar{R}_1R_2.$$

Now

$$\bar{R}_1R_2 = (I + \mathbf{d} \mathbf{e}_k^\top)R_2 = R_2 + \mathbf{d} \mathbf{e}_k^\top r_{kk}^{(2)}.$$

Again, if we do not normalize the last column of B then $r_{kk}^{(2)} = 1$ i.e. $\bar{R}_1R_2 = R_2 + \mathbf{d} \mathbf{e}_k^\top$ and the update is simply

$$r_{ik} := r_{ik} + d_i, \quad i = 1, \dots, k-1.$$

The normalization is only performed after the reorthogonalization step. We shall now modify our function `ModifiedGramSchmidt` so that each vector is reorthogonalized. Doing so we need to compute the projections twice and add them to the new row of R . For this purpose we introduce the auxiliary variable V :

ALGORITHM 6.11.

Modified Gram-Schmidt with Reorthogonalization

```
function [Q,R]=ModifiedGramSchmidtTwice(A);
% MODIFIEDGRAMSCHMIDTTWICE modified Gram-Schmidt with reorthogonalization
% [Q,R]=ModifiedGramSchmidtTwice(A); applies the modified
% Gram-Schmidt procedure with reorthogonalization to the columns in
% matrix A

[m,n]=size(A); R=zeros(n);
Q=A;
for k=1:n,
    for t=1:2                                % reorthogonalize
        for i=1:k-1,
            V=Q(:,i)'+Q(:,k);                % projections
            Q(:,k)=Q(:,k)-V*Q(:,i);
            R(i,k)=R(i,k)+V;
        end
    end
    R(k,k)=norm(Q(:,k)); Q(:,k)=Q(:,k)/R(k,k);
end
```

Using again our example we get this time

```
>> m=15; n=10; H=hilb(m); A=H(:,1:n);
>> [Q R]=ModifiedGramSchmidtTwice(A);
>> norm(Q'*Q-eye(n))
ans = 4.8199e-16
>> norm(Q*R-A)
ans = 6.7575e-17
```

a perfectly orthogonal matrix Q .

6.5.7 Partial Reorthogonalization

If we orthogonalize every vector twice, the number of computer operations is doubled. Therefore, it is natural to consider reorthogonalizing only when necessary. For this *partial reorthogonalization*, we need to decide when \mathbf{b}_k is to be considered small.

In the following algorithm, which is a translation of the Algol program `ortho` by Heinz Rutishauser [111], a simple criterion for reorthogonalization is used. If \mathbf{b}_k is 10 times smaller than \mathbf{a}_k then we must expect to lose at least one decimal digit by cancellation, thus we will reorthogonalize in that case.

If the vectors are linearly dependent (the matrix is rank deficient) then cancellation will also occur when the vector is reorthogonalized. If with repeated reorthogonalizing the vector shrinks to rounding error level then it is assumed that the vector is linearly dependent and it is replaced by a zero vector.

We obtain thus the following algorithm:

ALGORITHM 6.12.

Gram-Schmidt with Reorthogonalization

```
function [Q,R,z]=GramSchmidt(Q);
% GRAMSCHMIDT modified Gram-Schmidt with partial reorthogonalization
% [Q,R,z]=GramSchmidt(Q); computes the QR decomposition of the
% matrix Q using modified Gram-Schmidt with partial
% reorthogonalization; z is a vector that counts the
% reorthogonalization steps per column.
% Translation of the ALGOL procedure in H. Rutishauser: "Algol 60"

z=[]; [m,n]=size(Q); R=zeros(n);
for k=1:n,
    t=norm(Q(:,k));
    reorth=1;
    u=0; % count reorthogonalizations
    while reorth,
        u=u+1;
        for i=1:k-1,
```

```

    s= Q(:,i)'*Q(:,k);
    R(i,k)=R(i,k)+s;
    Q(:,k)=Q(:,k)-s*Q(:,i);
end
tt=norm(Q(:,k));
if tt>10*eps*t & tt<t/10,          % if length short reorthogonalize
    reorth=1; t=tt;
else
    reorth=0;
    if tt<10*eps*t, tt=0; end      % linearly dependent
end
end
z=[z u];
R(k,k)=tt;
if tt*eps~=0, tt=1/tt; else tt=0; end
Q(:,k)=Q(:,k)*tt;
end

```

Note that if the norm of \mathbf{b} becomes very small, e.g., when $\text{tt} < 10 \cdot \text{eps} \cdot t$ in the implementation above, then `GramSchmidt` considers the new column as linearly dependent and inserts a zero column in Q .

EXAMPLE 6.13. *The 7 column vectors of A are in \mathbb{R}^5 . So they are linearly dependent.*

```

A = [ 0      0      0      0      0      1      0
      83     1      0      0      0      1     973
        7    42      1      1    93     85     53
        9    65     42     70    91     76     26
        5    74     33     63    76     99     37]

```

```

[Q0,R0,z] = GramSchmidt(A)
[Q,R] = qr(A)

```

We obtain the results

```

Q0 =
      0          0          0          0          0      1.0000          0
    0.9889   -0.1388    0.0098    0.0515          0      0.0000          0
    0.0834    0.3841   -0.8231   -0.4098          0      0.0000          0
    0.1072    0.5977    0.5669   -0.5566          0     -0.0000          0
    0.0596    0.6899   -0.0309    0.7208          0     -0.0000          0

R0 =
  83.9285   15.8706    6.5532   11.3430   22.0426   22.1260  971.6480
      0   105.8968   48.2545   85.6870  142.5461  146.2353  -73.5970
      0          0   21.9672   36.9133  -27.3109  -29.9315  -20.4856
      0          0          0    6.0398  -33.9831   -5.7251   40.5828
      0          0          0          0          0          0          0
      0          0          0          0          0      1.0000   -0.0000
      0          0          0          0          0          0          0

z =

```

```

      1      1      1      2      1      2      1
Q =
      0      -0.0000      -0.0000      0.0000      -1.0000
    -0.9889     -0.1388      0.0098      0.0515      -0.0000
    -0.0834      0.3841     -0.8231     -0.4098      -0.0000
    -0.1072      0.5977      0.5669     -0.5566      -0.0000
    -0.0596      0.6899     -0.0309      0.7208      0.0000
R =
   -83.9285   -15.8706   -6.5532   -11.3430   -22.0426   -22.1260   -971.6480
         0    105.8968    48.2545    85.6870    142.5461    146.2353   -73.5970
         0         0    21.9672    36.9133   -27.3109   -29.9315   -20.4856
         0         0         0     6.0398   -33.9831   -5.7251    40.5828
         0         0         0         0     -0.0000   -1.0000   -0.0000

```

GramSchmidt discovered that the 5th vector was linearly dependent on the first four and replaced it by a zero vector. The 7th vector is also linearly dependent (as we would expect from the dimension of the column space) and is also replaced by a zero vector. Notice that the 4th and 6th vector had to be reorthogonalized. The standard MATLAB function `qr` gives the same results without displaying the zero rows and columns in R and Q .

| $1 - \lambda$ | G | M | Singular Values | | | | | |
|---------------|---|---|-----------------|-----------|-----------|-----------|-----------|--|
| 1.000e-09 | 3 | 3 | 8.593 | 2.091e-09 | 1.363e-09 | 1.543e-16 | 8.165e-18 | |
| 1.000e-10 | 3 | 3 | 8.593 | 2.091e-10 | 1.363e-10 | 3.458e-17 | 2.122e-17 | |
| 1.000e-11 | 3 | 3 | 8.593 | 2.091e-11 | 1.363e-11 | 5.955e-17 | 4.253e-17 | |
| 1.000e-12 | 3 | 3 | 8.593 | 2.091e-12 | 1.363e-12 | 8.250e-17 | 2.081e-17 | |
| 1.001e-13 | 3 | 3 | 8.593 | 2.095e-13 | 1.364e-13 | 7.927e-17 | 3.179e-17 | |
| 1.010e-14 | 3 | 2 | 8.593 | 2.122e-14 | 1.383e-14 | 1.876e-16 | 4.088e-17 | |
| 1.110e-15 | 2 | 1 | 8.593 | 2.398e-15 | 1.468e-15 | 1.150e-16 | 5.860e-17 | |
| 2.220e-16 | 1 | 1 | 8.593 | 4.422e-16 | 3.197e-16 | 2.348e-16 | 1.697e-17 | |
| 1.110e-16 | 1 | 1 | 8.593 | 5.609e-16 | 3.314e-16 | 1.055e-16 | 8.622e-18 | |

TABLE 6.1. Rank Computation with G (Gram-Schmidt) and M (MATLAB SVD)

EXAMPLE 6.14. As second example, we construct the following matrix $A \in \mathbb{R}^{8 \times 5}$ which has rank 3 if the parameter $\text{lam} = \lambda < 1$. However, for $\lambda = 1$ the matrix has rank 1. For $1 - \lambda \geq 10^{-14}$ *GramSchmidt* finds the linear dependencies correctly and replaces the dependent columns by zeros. If we let $\lambda \rightarrow 1$ we can see in Table 6.1 how both functions *GramSchmidt* and the MATLAB function `rank` compute the rank.

```

format short e
lam=0.99999999; e=0.00000001;
for i=1:9
    e=e/10; lam=lam+e*9;
    a=[1 lam 1 lam 1 1 1 lam]';
    b=[1 1 lam 1 1 1 1 lam 1]';
    c=[lam 1 1 lam 1 lam 1 1]';

```

```

A=[a+0.1*b-0.98*c, -0.321*a+0.07*b, 0.56*c, 0.3*a-3.1*b, b];
[q,r,z]=GramSchmidt(A)
rankgs=sum(any(q));
1-lam
[rankgs rank(A)]
svd(A)'
end

```

6.5.8 Updating and Downdating the QR Decomposition

Suppose we have computed the QR decomposition to find the model parameters that best fit the given data. It often happens that more data become available, data need to be thrown out, or the data have changed due to different operating conditions; in other words, the matrix has changed. Instead of recomputing the QR factorization from scratch, it is often more efficient to update the existing factorization. In this subsection, we show how to update the QR factors for the following types of changes in the matrix:

1. Rank-one update,
2. Deleting a column,
3. Adding a column,
4. Adding a row,
5. Deleting a row.

Rank-one update

Given a QR decomposition of the matrix $A = Q \begin{pmatrix} R \\ 0 \end{pmatrix}$ with $A \in \mathbb{R}^{m \times n}$, $Q \in \mathbb{R}^{m \times m}$ with $m \geq n$ and two vectors $\mathbf{u} \in \mathbb{R}^m$ and $\mathbf{v} \in \mathbb{R}^n$, is there a simple way to compute the QR decomposition of the rank-1 modified matrix $A' = A + \mathbf{u}\mathbf{v}^\top = Q'R'$?

Multiplying with Q^\top we get

$$Q^\top A' = Q^\top A + Q^\top \mathbf{u}\mathbf{v}^\top = \begin{pmatrix} R \\ 0 \end{pmatrix} + \mathbf{w}\mathbf{v}^\top \quad \text{with} \quad \mathbf{w} = Q^\top \mathbf{u}.$$

The idea is to transform the vector \mathbf{w} to a multiple of \mathbf{e}_1 . Then the rank-1 correction $\mathbf{w}\mathbf{v}^\top$ can be added to the matrix R by just changing the first row.

The algorithm consists of three steps:

1. Choose Givens rotation G_k acting on row k and $k+1$ for $k = m, m -$

$1, \dots, n+1$ such that the elements w_{n+2}, \dots, w_m are rotated to zero:

$$\underbrace{G_{n+1}^\top \dots G_{m-2}^\top G_{m-1}^\top Q_1^\top}_{Q_1^\top} A' = \begin{pmatrix} R \\ 0 \end{pmatrix} + G_{n+1}^\top \dots G_{m-2}^\top G_{m-1}^\top \mathbf{w} \mathbf{v}^\top$$

$$= \begin{pmatrix} R \\ 0 \end{pmatrix} + \begin{pmatrix} \mathbf{w}' \\ 0 \end{pmatrix} \mathbf{v}^\top.$$

This process generates

$$\begin{pmatrix} \mathbf{w}' \\ 0 \end{pmatrix} = G_{n+1}^\top \dots G_{m-2}^\top G_{m-1}^\top \mathbf{w}$$

and $Q_1 = Q G_{m-1} G_{m-2} \dots G_{n+1}$.

2. We now apply more Givens rotations to map \mathbf{w}' to $\alpha \mathbf{e}_1$. These rotations also change Q_1 to Q_2 and the matrix R becomes an upper Hessenberg matrix H_1 ,

$$\underbrace{G_1^\top \dots G_n^\top Q_1^\top}_{Q_2^\top} A' = \begin{pmatrix} H_1 \\ 0 \end{pmatrix} + \begin{pmatrix} \alpha \\ 0 \end{pmatrix} \mathbf{v}^\top = \begin{pmatrix} H \\ 0 \end{pmatrix}.$$

The transformed rank-1 correction is now added to the first row of the matrix H_1 , thus the right hand side becomes a new Hessenberg matrix H .

3. In the last step we annihilate with further Givens rotations the sub-diagonal of H and so we obtain the QR decomposition of the modified matrix,

$$\underbrace{G_n^\top \dots G_1^\top Q_2^\top}_{Q^\top} A' = G_n^\top \dots G_1^\top \begin{pmatrix} H \\ 0 \end{pmatrix} = \begin{pmatrix} R' \\ 0 \end{pmatrix}.$$

In the the following algorithm we will make use of the MATLAB function `planerot`, which computes a Givens rotation matrix $G \in \mathbb{R}^{2 \times 2}$. The call `[G,y]=planerot(x)` computes the matrix

$$G = \frac{1}{\|\mathbf{x}\|_2} \begin{pmatrix} x_1 & x_2 \\ -x_2 & x_1 \end{pmatrix} \quad \text{and vector} \quad \mathbf{y} = \begin{pmatrix} \|\mathbf{x}\|_2 \\ 0 \end{pmatrix}.$$

Since it makes use of the MATLAB built in function `norm`, it is safe to compute it [this way](#). It would not be a good idea to replace the norm computation by $\sqrt{x_1^2 + x_2^2}$, see Section 2.7.5 and Problem 2.13.

ALGORITHM 6.13. *Rank-1 Update of QR decomposition*

`function [Qs,Rs]=UpdateQR(Q,R,u,v)`

```

% UPDATEQR Rank-1 update of the QR-Decomposition
%   [Qs,Rs]=UpdateQR(Q,R,u,v); If As=A+u v' and [Q,R]=qr(A) then we
%   compute Qs and Rs such that As=Qs Rs. Uses Matlab's PLANEROT.

[m,n]=size(R); Qs=Q; Rs=R; w=Q'*u;
for k=m:-1:n+2                                % annihilate w(n+2:m)
    G=planerot(w(k-1:k));
    w(k-1:k)=G*w(k-1:k);
    Qs(:,k-1:k)=Qs(:,k-1:k)*G';
end
for k=n+1:-1:2                                % annihilate w(2:n+1)
    G=planerot(w(k-1:k));
    w(k-1:k)=G*w(k-1:k);
    Rs(k-1:k,k-1:n)=G*Rs(k-1:k,k-1:n);
    Qs(:,k-1:k)=Qs(:,k-1:k)*G';
end
Rs(1,:)=Rs(1,:)+w(1)*v';                      % Add rank-1 change to first row
for k=1:n                                      % reduce Hessenberg matrix to Rs
    G=planerot(Rs(k:k+1,k));
    Rs(k:k+1,k:n)=G*Rs(k:k+1,k:n);
    Qs(:,k:k+1)=Qs(:,k:k+1)*G';
end

```

EXAMPLE 6.15. We consider the 15×10 section of the Hilbert-matrix and the vectors $\mathbf{u} = (1, 2, \dots, 15)^\top$ and $\mathbf{v} = (1, 2, \dots, 10)^\top$. The test program

```

m=15; n=10;
A=hilb(m); A=A(:,1:n);
[Q,R]=qr(A);
u=[1:m]'; v=[1:n]';
[Qs,Rs,]=UpdateQR(Q,R,u,v);
As=A+u*v';
disp('||As-Qs*Rs||'), norm(As-Qs*Rs)
disp('||Qs'*Qs-eye(m)||'), norm(Qs'*Qs-eye(m))
[Qk,Rk]=qr(As);
disp('||As-Qk*Rk||'), norm(As-Qk*Rk)
disp('||Qk'*Qk-eye(m)||'), norm(Qk'*Qk-eye(m))
disp('norm(Rs-Rk)'), norm(Rs-Rk)
disp('||abs(Rs)-abs(Rk)||'), norm(abs(Rs)-abs(Rk))

```

compares the updating technique with the explicit QR decomposition of the modified matrix, and gives the output

```

||As-Qs*Rs||
ans =
    3.8508e-13
||Qs'*Qs-eye(m)||
ans =
    1.3560e-15

```

```

||As-Qk*Rk||
ans =
    5.1102e-13
||Qk'*Qk-eye(m)||
ans =
    1.4212e-15
norm(Rs-Rk)
ans =
    1.3825e+03
||abs(Rs)-abs(Rk)||
ans =
    3.6950e-13

```

The results are the same.

Deleting a column

We first consider the partition of $A \in \mathbb{R}^{m \times n}$ into two matrices and the corresponding partitioning of the QR decomposition,

$$A = [A_1, A_2] = Q \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \\ 0 & 0 \end{pmatrix}.$$

By multiplying the block columns, we conclude that

$$A_1 = Q \begin{pmatrix} R_{11} \\ 0 \end{pmatrix}$$

is the QR decomposition of A_1 . So when removing the last column or some of the last columns the update is trivial.

Consider now the columns $A = [\mathbf{a}_1, \dots, \mathbf{a}_k, \dots, \mathbf{a}_n]$ and the permutation matrix P which moves column k to the last column

$$\begin{aligned} AP &= [\mathbf{a}_1, \dots, \mathbf{a}_{k-1}, \mathbf{a}_{k+1}, \dots, \mathbf{a}_n, \mathbf{a}_k] \\ &= QRP = Q[\mathbf{r}_1, \dots, \mathbf{r}_{k-1}, \mathbf{r}_{k+1}, \dots, \mathbf{r}_n, \mathbf{r}_k]. \end{aligned}$$

The permuted matrix R becomes

$$RP = \begin{pmatrix} R_{11} & \mathbf{v} & R_{13} \\ 0 & r_{kk} & \mathbf{w}^\top \\ 0 & 0 & R_{33} \\ 0 & 0 & 0 \end{pmatrix} P = \begin{pmatrix} R_{11} & R_{13} & \mathbf{v} \\ 0 & \mathbf{w}^\top & r_{kk} \\ 0 & R_{33} & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

a Hessenberg matrix. With Givens rotations G_k acting on rows k and $k+1$, then $k+1$ and $k+2$, etc., we can transform the Hessenberg block again to upper triangular form,

$$G_{n-1}^\top \cdots G_k^\top \begin{pmatrix} \mathbf{w}^\top & r_{kk} \\ R_{33} & 0 \end{pmatrix} = \bar{R}_{22}.$$

Thus we get the decomposition

$$AP = \bar{Q}\bar{R}, \quad \text{with} \quad \bar{Q} = Q\bar{G}_k \cdots \bar{G}_{n-1} \quad \text{and} \quad \bar{G}_i = \begin{pmatrix} I & 0 \\ 0 & G_i \end{pmatrix},$$

and the last column can simply be discarded.

ALGORITHM 6.14. *Remove a Column of QR*

```
function [Q,R]=RemoveColumnQR(Q,R,k);
% REMOVECOLUMNQR updating QR when a column is removed
% [Q,R]=RemoveColumnQR(Q,R,k); finds a QR decomposition for the
% matrix [A(:,1),...,A(:,k-1),A(:,k+1),...A(:,n)] where
% [Q,R]=qr(A). Uses Matlab's function PLANEROT.

[m,n]=size(R);
R=R(:, [1:k-1,k+1:n]);
for j=k:n-1,
    G=planerot(R(j:j+1,j));
    R(j:j+1,j:n-1)=G*R(j:j+1,j:n-1);
    Q(:,j:j+1)=Q(:,j:j+1)*G';
end
```

Adding a column

Denote the new column by \mathbf{a}_{n+1} and place it before column k forming the matrix $\bar{A} = [\mathbf{a}_1, \dots, \mathbf{a}_{k-1}, \mathbf{a}_{n+1}, \mathbf{a}_k, \dots, \mathbf{a}_n]$. Using an appropriate permutation matrix P we move the new column to the end $\bar{A} = [A, \mathbf{a}_{n+1}]P$. Using the QR decomposition

$$A = Q \begin{pmatrix} R \\ 0 \end{pmatrix},$$

we get

$$Q^\top \bar{A} = \left[\begin{pmatrix} R \\ 0 \end{pmatrix}, Q^\top \mathbf{a}_{n+1} \right] P.$$

Defining

$$Q^\top \mathbf{a}_{n+1} = \begin{pmatrix} \mathbf{u} \\ \mathbf{v} \\ \mathbf{w} \end{pmatrix}, \quad \mathbf{u} \in \mathbb{R}^k, \mathbf{v} \in \mathbb{R}^{n-k}, \mathbf{w} \in \mathbb{R}^{m-n}$$

and reversing the permutation, we get

$$Q^\top \bar{A} = \begin{pmatrix} R_{11} & \mathbf{u} & R_{12} \\ 0 & \mathbf{v} & R_{22} \\ 0 & \mathbf{w} & 0 \end{pmatrix}.$$

Now we apply Givens rotations G_j acting on rows j and $j+1$ to annihilate elements of \mathbf{w} and \mathbf{v} up to v_1 . When annihilating elements of \mathbf{v} , the

corresponding rows of R_{22} are also changed and we obtain again an upper triangular matrix.

ALGORITHM 6.15. *QR update by Adding a Column*

```
function [Q,R]=AddColumnQR(Q,R,k,w);
% ADDCOLUMNQR_update QR decomposition when a new column is added
% [Q,R]=AddColumnQR(Q,R,k,w); finds the QR decomposition of
% [A(:,1),A(:,2),... A(:,k-1),w,A(:,k),...A(:,n)] when [Q,R]=qr(A).
% Uses Matlab PLANEROT

[m,n]=size(R);
R=[R(:,1:k-1),Q'*w,R(:,k:n)];
for j=m-1:-1:k
    G=planerot(R(j:j+1,k));
    R(j:j+1,k)=G*R(j:j+1,k);           % annihilate new column
    if j<=n,                             % transform also R
        R(j:j+1,j+1:n+1)=G*R(j:j+1,j+1:n+1);
    end
    Q(:,j:j+1)=Q(:,j:j+1)*G';           % update Q
end
```

Adding a row

Let $A = QR$ be given with $A \in \mathbb{R}^{m \times n}$, $Q \in \mathbb{R}^{m \times p}$ and $R \in \mathbb{R}^{p \times n}$. We consider adding a \mathbf{w}^\top as new last row of A ,

$$\bar{A} = \begin{pmatrix} A \\ \mathbf{w}^\top \end{pmatrix}.$$

Then

$$\begin{pmatrix} Q^\top & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} A \\ \mathbf{w}^\top \end{pmatrix} = \begin{pmatrix} R \\ \mathbf{w}^\top \end{pmatrix}.$$

To eliminate the elements of vector \mathbf{w}^\top we use again Givens rotations acting on \mathbf{w}^\top and a row of R . For $p = n$ we get

$$G_n^\top \cdots G_2^\top G_1^\top \begin{pmatrix} R \\ \mathbf{w}^\top \end{pmatrix} = \begin{pmatrix} \bar{R} \\ 0 \end{pmatrix}$$

with G_i^\top changing row i of R and \mathbf{w}^\top . By applying the Givens rotations to Q we get

$$\bar{Q} = \begin{pmatrix} Q & 0 \\ 0 & 1 \end{pmatrix} G_1 \cdots G_n$$

and thus $\bar{A} = \bar{Q} \begin{pmatrix} \bar{R} \\ 0 \end{pmatrix}$.

Note that if the new row is not appended as last row but between the rows of A , then this means simply a permutation of the rows of \bar{Q} , since

$$\bar{Q}^\top P^\top P \bar{A} = (P\bar{Q})^\top \begin{pmatrix} \mathbf{a}_1: \\ \vdots \\ \mathbf{a}_{k-1}: \\ \mathbf{w}^\top \\ \mathbf{a}_k: \\ \vdots \\ \mathbf{a}_m: \end{pmatrix}$$

and \bar{R} remains the same.

ALGORITHM 6.16. *QR update by Adding a Row*

```
function [Q,R]=AddRowQR(Q,R,k,w);
% ADDROWQR QR decomposition of modified matrix
% [Q,R]=AddRowQR(Q,R,k,w); Given [Q,R]=qr(A), computes the QR
% decomposition for [A(1,:);A(2,:), ...,A(k-1,:);w;A(k,:),...A(n,:)]

[p,n]=size(R); m=size(Q,1);           % Q is (m x p)
Q=[Q(1:k-1,:), zeros(k-1,1)            % add row and column to Q
   zeros(1,p), 1                        % and permute
   Q(k:m,:), zeros(m-k+1,1)];
R=[R; w];                               % augment R
k=p+1;
for j=1:min(n,p)
    G=planerot(R(j:k-j:k,j));
    R(j:k-j:k,j:n)=G*R(j:k-j:k,j:n);   % annihilate w
    Q(:,j:k-j:k)=Q(:,j:k-j:k)*G';       % update Q
end
```

EXAMPLE 6.16. In the following example we consider a matrix $A \in \mathbb{R}^{5 \times 3}$ with rank 2:

```
>> A=[17    24   -43
      23     5   244
        4     6  -14
       10    12   -2
       11    18  -55];
>> [m,n]=size(A);
>> [Q,R]=qr(A)
Q =
-0.5234    0.5058    0.3869   -0.1275   -0.5517
-0.7081   -0.6966    0.0492   -0.0642    0.0826
-0.1231    0.1367   -0.6694   -0.7172   -0.0614
-0.3079    0.1911   -0.6229    0.6816   -0.1271
```

```

    -0.3387    0.4514    0.1089   -0.0275    0.8179
R =
   -32.4808   -26.6311  -129.3073
         0    19.8943  -218.8370
         0         0    0.0000
         0         0         0
         0         0         0

```

```
>> p=rank(A)
```

```
p =
```

```
2
```

```
>> q=Q(:,1:p)
```

```
q =
```

```

   -0.5234    0.5058
   -0.7081   -0.6966
   -0.1231    0.1367
   -0.3079    0.1911
   -0.3387    0.4514

```

```
>> r=R(1:p,:)
```

```
r =
```

```

   -32.4808   -26.6311  -129.3073
         0    19.8943  -218.8370

```

```
>> [norm(A-Q*R) norm(A-q*r)]
```

```
ans =
```

```

   1.0e-13 *
   0.1313    0.0775

```

We computed the QR decomposition with $Q \in \mathbb{R}^{5 \times 5}$ and, because A has rank 2, we also obtained the economic version with $q \in \mathbb{R}^{5 \times 2}$ and $r \in \mathbb{R}^{2 \times 3}$. Both products QR and qr represent the matrix A well. Next, we choose a new row w^T and insert it as new third row:

```
>> k=3;
```

```
>> w=[1 2 3];
```

```
>> As=[A(1:k-1,:); w; A(k:m,:)]
```

```
As =
```

```

    17    24   -43
    23     5   244
     1     2     3
     4     6   -14
    10    12    -2
    11    18   -55

```

```
>> [Qs, Rs]=AddRowQR(Q,R,k,w)
```

```
Qs =
```

```

    0.5231    0.5039   -0.0460   -0.1275   -0.5517    0.3869
    0.7078   -0.6966    0.0195   -0.0642    0.0826    0.0492
    0.0308    0.0592    0.9978         0         0   -0.0000
    0.1231    0.1363   -0.0119   -0.7172   -0.0614   -0.6694
    0.3077    0.1902   -0.0208    0.6816   -0.1271   -0.6229
    0.3385    0.4500   -0.0371   -0.0275    0.8179    0.1089

```

```
Rs =
```

```

32.4962    26.6801   129.3384
      0    19.9292  -218.5114
      0      0    11.9733
      0      0      0
      0      0      0
      0      0      0
>> [qs, rs]=AddRowQR(q,r,k,w)
qs =
  0.5231    0.5039    0.0460
  0.7078   -0.6966   -0.0195
  0.0308    0.0592   -0.9978
  0.1231    0.1363    0.0119
  0.3077    0.1902    0.0208
  0.3385    0.4500    0.0371
rs =
  32.4962    26.6801   129.3384
      0    19.9292  -218.5114
      0      0   -11.9733
>> [norm(As-Qs*Rs) norm(As-qs*rs)]
ans =
  1.0e-12 *
   0.1180    0.1165

```

We see that in both cases the update procedure works well. The product of the new matrices represents well the modified matrix.

Deleting a row

Let $A = QR$ be the QR decomposition. If the k -th row should be removed, then we perform first a permutation such that it becomes the first row:

$$A = QR, \quad PA = \begin{pmatrix} \mathbf{a}_k^\top \\ \bar{A} \end{pmatrix} = PQR.$$

Our aim is to compute the QR decomposition of \bar{A} which is A with row k removed. Using Givens rotations, we transform the first row of PQ to the first unit vector

$$PQG_{m-1}^\top \cdots G_1^\top = \begin{pmatrix} \pm 1 & 0 \\ \mathbf{x} & \bar{Q} \end{pmatrix}$$

Note that by this operation the vector \mathbf{x} must be zero! This because the matrix is orthogonal. Thus we get

$$\begin{aligned} \begin{pmatrix} \mathbf{a}_k^\top \\ \bar{A} \end{pmatrix} &= (PQG_{m-1}^\top \cdots G_1^\top) (G_1 \cdots G_{m-1} R) \\ &= \begin{pmatrix} \pm 1 & 0 \\ 0 & \bar{Q} \end{pmatrix} \begin{pmatrix} \mathbf{v}^\top \\ \bar{R} \end{pmatrix}. \end{aligned} \quad (6.42)$$

The multiplications with the Givens rotations $G_1 \cdots G_{m-1}R$ transform the matrix R into a Hessenberg matrix. Looking at (6.42), we can read off the solution $\bar{A} = \bar{Q}\bar{R}$ by discarding the first row.

ALGORITHM 6.17. *QR update by Removing a Row*

```
function [Q,R]=RemoveRowQR(Q,R,k);
% REMOVEROWQR update the QR decomposition when a row is removed
% [Q,R]=RemoveRowQR(Q,R,k); computes the QR decomposition for
% [A(1,:); A(2,:); ...; A(k-1,:); A(k+1,:); ...; A(n,:)] when
% [Q,R]=qr(A).

[m,n]=size(R);
Q=[Q(k,:); Q(1:k-1,:); Q(k+1:m,:)];           % permute Q
for j=m-1:-1:1                                  % map row to unit vector
    G=planerot(Q(1,j:j+1)');
    if j<=n
        R(j:j+1,j:n)=G*R(j:j+1,j:n);          % update R
    end;
    Q(:,j:j+1)=Q(:,j:j+1)*G';                  % update Q
end
Q=Q(2:m,2:m); R=R(2:m,:);
```

REMARKS.

1. Deleting a row is a special case of a rank-one modification: $\mathbf{u} = -\mathbf{e}_1$, $\mathbf{v} = \mathbf{a}_1$.
2. MATLAB offers two functions to add (respectively remove) rows and columns of a QR decomposition: `qrinsert`, `qrdelete`. They are essentially the same as our functions. There is also a MATLAB-built in function `qrupdate` which computes a rank-one update.

6.5.9 Covariance Matrix Computations Using QR

Let $A = Q \begin{pmatrix} R \\ 0 \end{pmatrix}$ be the QR decomposition. Then $A^\top A = R^\top R$ and thus the *covariance matrix* becomes

$$C = (A^\top A)^{-1} = R^{-1}R^{-\top}.$$

In this section, we will show how to compute elements of the covariance matrix using a minimal number of operations.

Computing individual elements

Since the covariance matrix C is symmetric, it is sufficient to compute elements in the upper triangle. Let $i \leq j$. Then

$$c_{ij} = \mathbf{e}_i^\top R^{-1} R^{-\top} \mathbf{e}_j = \mathbf{u}_i^\top \mathbf{u}_j \quad \text{where} \quad \mathbf{u}_i = R^{-\top} \mathbf{e}_i.$$

Thus \mathbf{u}_i can be computed by forward substitution,

$$R^\top \mathbf{u}_i = \mathbf{e}_i.$$

Because the elements $u_i = 0$ for $i = 1, 2, \dots, i-1$ we need to start with forward substitution only at equation $\#i$.

ALGORITHM 6.18. *Computes i th-Row of R^{-1}*

```
function u=ui(i,R)
% UI computes a row of the inverse of a triangular matrix
%   u=ui(i,R) computes the i-th row of the inverse of the upper
%   triangular matrix R

n=min(size(R)); u=zeros(1,i-1);
u(i)=1/R(i,i);
for j=i+1:n,
    u(j)=-u(i:j-1)*R(i:j-1,j)/R(j,j);
end
```

The element c_{ij} is then computed for $i < j$ by the statements

```
u1=ui(i,R); u2=ui(j,R); cij=u1(j:n)*u2(j:n)'
```

For a diagonal element this simplifies to

```
u=ui(i,R); cii=u(i:n)*u(i:n)'
```

Calculating the whole covariance matrix

To compute $C = R^{-1} R^{-\top}$ we need R^{-1} , that is the vectors $\mathbf{u}_1, \dots, \mathbf{u}_n$. Then we multiply $R^{-1} R^{-\top}$ by using the triangular shape of R^{-1} and the symmetry and computing only the upper part of C .

ALGORITHM 6.19. *Covariance via R^{-1}*

```
function C=Covariance(R)
% COVARIANCE computes the covariance matrix
%   C=Covariance(R) computes the upper triangle of the covariance
%   matrix from the R factor of the QR-decomposition [Q,R]=qr(A)

n=min(size(R)); U=[]; % U is R^(-1)
```

```

for i=1:n,
    u=ui(i,R); U=[U; u];
end;
for i=1:n,                                     % compute C=inv(R)*inv(R')
    for j=i:n,
        C(i,j)=U(i,j:n)*U(j,j:n)';
    end
end

```

Björck's Algorithm

Because $R^\top R = A^\top A$ it follows that $R^\top RC = I$ or

$$RC = R^{-\top}. \quad (6.43)$$

The diagonal elements on the right hand side of Equation (6.43) are $1/r_{ii}$. Surprisingly, this information is sufficient to compute by a recurrence the elements of the covariance matrix C .

Assume that the rows and columns $k+1, \dots, n$ of C are known, that is the elements $c_{ij} = c_{ji}$ for $j = k+1, \dots, n$ and $i \leq j$.

Consider the k -th diagonal element in (6.43),

$$\mathbf{r}_k^\top \mathbf{c}_k = \frac{1}{r_{kk}} \iff r_{kk}c_{kk} + \sum_{j=k+1}^n r_{kj}c_{jk} = \frac{1}{r_{kk}}.$$

The elements after the summation sign are known, therefore

$$c_{kk} = \frac{1}{r_{kk}} \left(\frac{1}{r_{kk}} - \sum_{j=k+1}^n r_{kj}c_{jk} \right).$$

For an element on the k -th row with $i < k$ we have

$$\mathbf{r}_i^\top \mathbf{c}_k = 0 \iff r_{ii}c_{ik} + \sum_{j=i+1}^n r_{ij}c_{jk} = 0.$$

Also here the elements after the summation symbol are known according to our assumption, thus

$$c_{ik} = -\frac{1}{r_{ii}} \sum_{j=i+1}^n r_{ij}c_{jk}, \quad i = k-1, \dots, 1.$$

Using these recurrence relations, the whole covariance matrix can be computed starting with the last column. We obtain

ALGORITHM 6.20. *Covariance Matrix, Björck Algorithm*

```

function C=CovarianceBjoerck(R)
% COVARIANCEBJOERCK computes the covariance matrix by Bjoerck's method
%   C=CovarianceBjoerck(R) computes the covariance matrix from the
%   R factor of A=QR by solving the system R C=R'-T

n=min(size(R));
C=zeros(n,n);
for k=n:-1:1,
    C(k,k)=(1/R(k,k)-R(k,k+1:n)*C(k,k+1:n)')/R(k,k);
    for i=k-1:-1:1,
        C(i,k)=-(R(i,i+1:k)*C(i+1:k,k)+R(i,k+1:n)*C(k,k+1:n)')/R(i,i);
    end
end
end

```

6.6 Linear Least Squares Problems with Linear Constraints

Given the matrices $A^{m \times n}$, $C^{p \times n}$ and the vectors \mathbf{b} and \mathbf{d} , we are interested in finding a vector \mathbf{x} such that

$$\|A\mathbf{x} - \mathbf{b}\|_2 \longrightarrow \min \quad \text{subject to} \quad C\mathbf{x} = \mathbf{d}. \quad (6.44)$$

We are interested in the case $p \leq n \leq m$. A solution exists only if the constraints are consistent i.e. if $\mathbf{d} \in \mathcal{R}(C)$.

A straightforward way is to eliminate the constraints and solve the reduced unconstrained problem. This can be done using the general solution of $C\mathbf{x} = \mathbf{d}$, see Section 6.3.3,

$$\mathbf{x} = C^+ \mathbf{d} + P_{N(C)} \mathbf{y}, \quad \mathbf{y} \text{ arbitrary and } P_{N(C)} = I - C^+ C.$$

Now $\|A\mathbf{x} - \mathbf{b}\|_2^2 = \|AP_{N(C)} \mathbf{y} - (\mathbf{b} - AC^+ \mathbf{d})\|_2^2 \longrightarrow \min$ is an unconstrained linear least squares problem in \mathbf{y} . The solution with minimal norm is

$$\tilde{\mathbf{y}} = (AP_{N(C)})^+ (\mathbf{b} - AC^+ \mathbf{d}).$$

Thus $\mathbf{x} = C^+ \mathbf{d} + P_{N(C)} (AP_{N(C)})^+ (\mathbf{b} - AC^+ \mathbf{d})$. We can simplify this expression using the following lemma:

LEMMA 6.2. $P_{N(C)} (AP_{N(C)})^+ = (AP_{N(C)})^+.$

PROOF. The matrix $(AP_{N(C)})^+$ is the solution of the Penrose equations (see Theorem 6.7). We show that $Y = P_{N(C)} (AP_{N(C)})^+$ is also a solution, therefore by uniqueness they must be the same. In the proof below, we will write $P := P_{N(C)}$ and use the relations $PP = P$ and $P^\top = P$, which hold because P is an orthogonal projector. We now substitute Y into the Penrose equations:

$$(i) \quad (AP)Y(AP) = A \underbrace{PP}_P (AP)^+ AP = AP(AP)^+ (AP) = AP.$$

$$(ii) \quad Y(AP)Y = P(AP)^+ A \underbrace{PP}_P (AP)^+ = P((AP)^+ AP(AP)^+) = P(AP)^+ = Y.$$

$$(iii) \quad (APY)^\top = (A \underbrace{PP}_P (AP)^+)^\top = A \underbrace{P}_{PP} (AP)^+ = APY.$$

$$(iv) \quad (YAP)^\top = (P(AP)^+ AP)^\top = [(AP)^+ AP]^\top P = (AP)^+ A \underbrace{PP}_P \\ = (AP)^+ AP = ((AP)^+ AP)^\top = (AP)^\top [(AP)^+]^\top = \underbrace{P}_{PP} A^\top [(AP)^+]^\top \\ = P((AP)^+ AP)^\top = P((AP)^+ AP) = YAP.$$

□

As a consequence of the lemma, we have $P_{N(C)}\tilde{\mathbf{y}} = \tilde{\mathbf{y}}$.

THEOREM 6.16. Let $\mathbf{d} \in \mathcal{R}(C)$ and define $\mathbf{x} = \tilde{\mathbf{x}} + \tilde{\mathbf{y}}$ with

$$\tilde{\mathbf{x}} = C^+ \mathbf{d}, \quad \tilde{\mathbf{y}} = (AP_{N(C)})^+ (\mathbf{b} - A\tilde{\mathbf{x}}),$$

where $P_{N(C)} = I - C^+C$. Then \mathbf{x} is a solution to the problem (6.44); this solution is unique if and only if

$$\text{rank} \begin{pmatrix} A \\ C \end{pmatrix} = n.$$

Moreover, if (6.44) has more than one solution, then \mathbf{x} is the minimal norm solution.

PROOF. We have already shown by construction that \mathbf{x} is a solution. We note that the solution \mathbf{x} cannot be unique if $\text{rank} \begin{pmatrix} A \\ C \end{pmatrix} < n$, since there would exist a vector $\mathbf{w} \neq 0$ such that $\begin{pmatrix} A \\ C \end{pmatrix} \mathbf{w} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ and $\mathbf{x} + \mathbf{w}$ is also a solution. Thus the condition is necessary; we will show in Section 6.6.2 that it is also sufficient.

We now show that \mathbf{x} has minimal norm when the solution is not unique. Let \mathbf{x} and $\hat{\mathbf{x}}$ be two solutions to (6.44). Since both must satisfy the constraint $C\mathbf{x} = \mathbf{d}$, they can be written as

$$\mathbf{x} = C^+ \mathbf{d} + \tilde{\mathbf{y}} = \tilde{\mathbf{x}} + P\tilde{\mathbf{y}}, \\ \hat{\mathbf{x}} = C^+ \mathbf{d} + P\mathbf{y} = \tilde{\mathbf{x}} + P\mathbf{y},$$

where we use again the abbreviation $P := P_{N(C)}$ and the fact that $P\tilde{\mathbf{y}} = \tilde{\mathbf{y}}$. Since $P = P^\top$ and $PC^+ = (I - C^+C)C^+ = 0$ (see Theorem 6.9), we see that $\tilde{\mathbf{x}} \perp P\tilde{\mathbf{y}}$, so we have

$$\|\mathbf{x}\|_2^2 = \|\tilde{\mathbf{x}}\|_2^2 + \|P\tilde{\mathbf{y}}\|_2^2 = \|\tilde{\mathbf{x}}\|_2^2 + \|\tilde{\mathbf{y}}\|_2^2.$$

Similarly, for $\hat{\mathbf{x}}$ we have

$$\|\hat{\mathbf{x}}\|_2^2 = \|\tilde{\mathbf{x}}\|_2^2 + \|P\mathbf{y}\|_2^2.$$

Now since $\tilde{\mathbf{y}}$ is the minimal norm solution, every other solution of the reduced unconstrained linear least squares problem has the form $\mathbf{y} = \tilde{\mathbf{y}} + \mathbf{w}$ with $\mathbf{w} \perp \tilde{\mathbf{y}}$. Also $\tilde{\mathbf{y}} \perp P\mathbf{w}$ holds since $\tilde{\mathbf{y}}^\top P\mathbf{w} = (P\tilde{\mathbf{y}})^\top \mathbf{w} = \tilde{\mathbf{y}}^\top \mathbf{w} = 0$. Thus

$$\|\hat{\mathbf{x}}\|_2^2 = \|\tilde{\mathbf{x}}\|_2^2 + \|P\mathbf{y}\|_2^2 = \|\tilde{\mathbf{x}}\|_2^2 + \|\tilde{\mathbf{y}}\|_2^2 + \|P\mathbf{w}\|_2^2 \geq \|\tilde{\mathbf{x}}\|_2^2 + \|\tilde{\mathbf{y}}\|_2^2 = \|\mathbf{x}\|_2^2.$$

Therefore $\mathbf{x} = \tilde{\mathbf{x}} + \tilde{\mathbf{y}}$ is the minimum norm solution. \square

6.6.1 Solution with SVD

Problem (6.44) can be solved in an elegant and even more general way using the SVD. We will assume that $A \in \mathbb{R}^{m \times n}$, $C \in \mathbb{R}^{p \times n}$ with $p < n < m$, that $\mathbf{b} \notin \mathcal{R}(A)$, that $\text{rank}(C) = r_c < p$ and $\mathbf{d} \notin \mathcal{R}(C)$. This means that $C\mathbf{x} = \mathbf{d}$ is not consistent, but that we consider the more general problem

$$\|A\mathbf{x} - \mathbf{b}\|_2 \longrightarrow \min \quad \text{subject to} \quad \|C\mathbf{x} - \mathbf{d}\|_2 \longrightarrow \min. \quad (6.45)$$

Since C has a nontrivial null space, the constraint $\|C\mathbf{x} - \mathbf{d}\|_2 \longrightarrow \min$ has many solutions and we want to minimize $\|A\mathbf{x} - \mathbf{b}\|_2$ over that set. The algorithm that we develop can then also be used for the special case of equality constraints when $C\mathbf{x} = \mathbf{d}$ is consistent.

In the first step we determine the general (least squares) solution of $C\mathbf{x} \approx \mathbf{d}$. Let $C^\top = T S R^\top$ be the SVD with $T \in \mathbb{R}^{n \times n}$ and $R \in \mathbb{R}^{p \times p}$. Since $\text{rank}(C) = r_c < p$, we partition the matrices accordingly into the sub-matrices $T = [T_1, T_2]$ with $T_1 \in \mathbb{R}^{n \times r_c}$, $T_2 \in \mathbb{R}^{n \times (p-r_c)}$, $S_r = S(1:r_c, 1:r_c)$ and $R = [R_1, R_2]$ with $R_1 \in \mathbb{R}^{p \times r_c}$ and $R_2 \in \mathbb{R}^{p \times (p-r_c)}$. With this partition, $C = R_1 S_r T_1^\top$ and $C^+ = T_1 S_r^{-1} R_1^\top$. The general solution of $\|C\mathbf{x} - \mathbf{d}\|_2 \longrightarrow \min$ then becomes

$$\mathbf{x} = \mathbf{x}_m + T_2 \mathbf{z}, \quad \text{with } \mathbf{z} \in \mathbb{R}^{n-r_c} \text{ arbitrary and } \mathbf{x}_m = T_1 S_r^{-1} R_1^\top \mathbf{d}.$$

We now introduce this general solution into $\|A\mathbf{x} - \mathbf{b}\|_2$ and obtain

$$\|AT_2 \mathbf{z} - (\mathbf{b} - A\mathbf{x}_m)\|_2 \longrightarrow \min, \quad AT_2 \in \mathbb{R}^{m \times (n-r_c)}, \quad (6.46)$$

an unconstrained least squares problem for \mathbf{z} . We compute the SVD of $AT_2 = U \Sigma V^\top$ and partition again the matrices according to the $\text{rank}(AT_2) = r_a$, for which we will assume that $r_a < n - r_c$: $U = [U_1, U_2]$, $\Sigma_r = \Sigma(1:r_a, 1:r_a)$ and $V = [V_1, V_2]$. With this partition, the general solution of (6.46) is

$$\mathbf{z} = \mathbf{z}_m + V_2 \mathbf{w}, \quad \text{with } \mathbf{w} \in \mathbb{R}^{n-r_c-r_a} \text{ arbitrary, and } \mathbf{z}_m = V_1 \Sigma_r^{-1} U_1^\top (\mathbf{b} - A\mathbf{x}_m).$$

Thus the solution of Problem (6.45) is

$$\mathbf{x} = \mathbf{x}_m + T_2 \mathbf{z} = \mathbf{x}_m + T_2 V_1 \Sigma_r^{-1} U_1^\top (\mathbf{b} - A\mathbf{x}_m) + T_2 V_2 \mathbf{w}, \quad \mathbf{x}_m = T_1 S_r^{-1} R_1^\top \mathbf{d} \quad (6.47)$$

with $\mathbf{w} \in \mathbb{R}^{n-r_c-r_a}$ arbitrary.

EXAMPLE 6.17. We consider $A \in \mathbb{R}^{9 \times 6}$, $\text{rank}(A) = 3$, $C \in \mathbb{R}^{3 \times 6}$, $\text{rank}(C) = 2$ and $\text{rank}\left(\begin{smallmatrix} A \\ C \end{smallmatrix}\right) = 5$. Furthermore $\mathbf{b} \notin \mathcal{R}(A)$ and also $\mathbf{d} \notin \mathcal{R}(C)$. The solution (6.47) will not be unique, as we will see.

```
>> A=[5 -1 -1 6 4 0
      -3 1 4 -7 -2 -3
       1 3 -4 5 4 7
       0 4 -1 1 4 5
       4 2 3 1 6 -1
       3 -3 -5 8 0 2
       0 -1 -4 4 -1 3
      -5 4 -3 -2 -1 7
       3 4 -3 6 7 7];
>> [m,n]=size(A);
>> b=[-4 1 -2 3 3 0 -1 3 1]';
>> ranksa=[rank(A) rank([A,b])]
ranksa=      3      4
>> C=[1 3 -2 3 8 0
      -3 0 0 1 9 4
      -2 3 -2 4 17 4];
>> [p n]=size(C);
>> d=[1 2 -3]';
>> ranksc=[rank(C) rank([C,d])]
ranksc=      2      3
```

Now we compute the minimal norm solution of $C\mathbf{x} \approx \mathbf{d}$, its norm and the norm of the residual $\mathbf{r} = \mathbf{d} - C\mathbf{x}_m$:

```
>> [T,S,R]=svd(C');
>> rc=rank(S); Sr=S(1:rc,1:rc);
>> T1=T(:,1:rc); T2=T(:,rc+1:n);
>> R1=R(:,1:rc); R2=R(:,rc+1:p);
>> xm=T1*(Sr\'(R1\'*d));
>> xm'
ans=-0.0783 -0.0778 0.0519 -0.0604 -0.0504 0.0698
>> [norm(xm) norm(d-C*xm)]
ans= 0.1611 3.4641
```

To obtain another solution \mathbf{x}_g of $C\mathbf{x} \approx \mathbf{d}$, we choose e.g. $\mathbf{z} = (1, 2, 3, 4)^\top$ and obtain

```
>> xg=xm+T2*[1 2 3 4]';
>> xg'
ans= 0.1391 0.6588 -2.7590 2.0783 -1.8586 3.7665
>> [norm(xg) norm(d-C*xg)]
ans= 5.4796 3.4641
```

We see that \mathbf{x}_g has the same residual, but $\|\mathbf{x}_g\|_2 > \|\mathbf{x}_m\|_2$, as we would expect. Note that the results for \mathbf{x}_g may differ depending on the version of MATLAB used; this is because the matrix T_2 is not unique. We continue by eliminating the constraints and solve the unconstrained problem for \mathbf{z} :

```
>> As=A*T2; c=b-A*xm;
```

```

>> nrc=n-rc
nrc=      4
>> [U Sig V]=svd(As);
>> ra=rank(Sig)
ra= 3
>> Sigr=Sig(1:ra,1:ra);
>> U1=U(:,1:ra); U2= U(:,ra+1:nrc);
>> V1=V(:,1:ra); V2= V(:,ra+1:nrc);
>> zm=V1*(Sigr\((U1'*c)));
>> zm'
ans=  -0.0364 -0.1657 -0.2783  0.3797

```

The matrix $A_s = AT_2$ is rank deficient, so the reduced unconstrained problem has infinitely many solutions and the vector \mathbf{z}_m is the minimal norm solution. Thus the minimal norm solution of Problem (6.45) is $\mathbf{x}_{min} = \mathbf{x}_m + T_2\mathbf{z}_m$:

```

>> xmin=xm+T2*zm;
>> xmin'
ans=0.1073  0.2230  0.2695 -0.1950 -0.0815  0.3126

```

We finally perform some checks; \mathbf{x}_a is a solution of $A\mathbf{x} \approx \mathbf{b}$ computed by MATLAB.

```

>> xa=A\b;
Warning: Rank deficient, rank=3  tol=  3.0439e-14.
>> [norm(C*xm-d) norm(C*xg-d) norm(C*xmin-d) norm(C*xa-d)]
ans=   3.4641   3.4641   3.4641   6.1413
>> [norm(A*xm-b) norm(A*xg-b) norm(A*xmin-b) norm(A*xa-b)]
ans=   6.8668  93.3010  5.3106  5.3106
>> [norm(xm) norm(xg) norm(xmin) norm(xa)]
ans=   0.1611  5.4796  0.5256  0.5035

```

Of course, \mathbf{x}_a does not minimize $\|C\mathbf{x} - \mathbf{d}\|_2$ and \mathbf{x}_m does not minimize $\|A\mathbf{x} - \mathbf{b}\|_2$. However, \mathbf{x}_{min} does minimize both norms as it should.

Another solution of Problem (6.45) is obtained by adding to \mathbf{x}_{min} some linear combination of the columns of the matrix T_2V_2 : $\mathbf{x} = \mathbf{x}_{min} + T_2V_2\mathbf{w}$. Since $T_2V_2 \in \mathbb{R}^{6 \times 1}$ in our example, \mathbf{w} is a scalar:

```

T2*V2
ans =
    0.4656
   -0.2993
   -0.4989
   -0.6319
    0.1663
    0.1330

```

The general solution of Problem (6.45) for our example is therefore (with

arbitrary parameter λ)

$$\mathbf{x} = \begin{pmatrix} 0.1073 \\ 0.2230 \\ 0.2695 \\ -0.1950 \\ -0.0815 \\ 0.3126 \end{pmatrix} + \lambda \begin{pmatrix} 0.4656 \\ -0.2993 \\ -0.4989 \\ -0.6319 \\ 0.1663 \\ 0.1330 \end{pmatrix}.$$

We compute two other solutions and show that they also minimize the two norms but their own norm is larger than $\|\mathbf{x}_{\min}\|_2$:

```
>> x=xmin +T2*V2;
>> x'
ans= 0.5729 -0.0763 -0.2293 -0.8269 0.0848 0.4457
>> [norm(d-C*x) norm(b-A*x) norm(x)]
ans= 3.4641 5.3106 1.1297
>> x=xmin +5*T2*V2;
>> x'
ans= 2.4355 -1.2737 -2.2249 -3.3547 0.7500 0.9778
>> [norm(d-C*x) norm(b-A*x) norm(x)]
ans= 3.4641 5.3106 5.0276
```

Certainly, the SVD is the best method in such cases where rank decisions have to be made. By providing explicit expressions for the projectors and orthogonal bases for the subspaces, this method is well suited for rank deficient matrices and general solutions.

6.6.2 Classical Solution Using Lagrange Multipliers

Before the widespread use of the SVD as a computational tool, a classical approach for solving Problem (6.44) uses Lagrange multipliers (see Section 12.2.2 for details). The Lagrangian is

$$L(\mathbf{x}, \boldsymbol{\lambda}) = \frac{1}{2} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2 + \boldsymbol{\lambda}^\top (\mathbf{C}\mathbf{x} - \mathbf{d}).$$

Setting the partial derivatives to zero, we obtain

$$\frac{\partial L}{\partial \mathbf{x}} = \mathbf{A}^\top (\mathbf{A}\mathbf{x} - \mathbf{b}) + \mathbf{C}^\top \boldsymbol{\lambda} = 0 \text{ and } \frac{\partial L}{\partial \boldsymbol{\lambda}} = \mathbf{C}\mathbf{x} - \mathbf{d} = 0.$$

Thus, we obtain the *Normal Equations*

$$\begin{pmatrix} \mathbf{A}^\top \mathbf{A} & \mathbf{C}^\top \\ \mathbf{C} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \boldsymbol{\lambda} \end{pmatrix} = \begin{pmatrix} \mathbf{A}^\top \mathbf{b} \\ \mathbf{d} \end{pmatrix} \quad (6.48)$$

The matrix of the Normal Equations (6.48) is symmetric, but not positive definite; for this reason, this system is also known as *saddle point equations*,

since they correspond geometrically to saddle points, i.e., points that are neither local minima nor maxima. Because of the indefiniteness, we cannot use the Cholesky decomposition to solve (6.48); moreover, the matrix becomes singular in the case of rank deficient matrices, like the example in Section 6.6.1, or even when only C is rank deficient.

The Normal Equations (6.48) can be used to finish the proof of Theorem 6.16. We need to prove that if $\text{rank} \begin{pmatrix} A \\ C \end{pmatrix} = n$, then the solution of Problem (6.44) is unique. Assume that we have two solutions \mathbf{x}_1 and \mathbf{x}_2 . Then both are solutions of the normal equations with some $\boldsymbol{\lambda}_1$ and $\boldsymbol{\lambda}_2$. If we take the difference of the normal equations, we obtain

$$\begin{pmatrix} A^\top A & C^\top \\ C & 0 \end{pmatrix} \begin{pmatrix} \mathbf{x}_1 - \mathbf{x}_2 \\ \boldsymbol{\lambda}_1 - \boldsymbol{\lambda}_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}. \quad (6.49)$$

Multiplying the first equation of (6.49) from the left with $(\mathbf{x}_1 - \mathbf{x}_2)^\top$, we get

$$\|A(\mathbf{x}_1 - \mathbf{x}_2)\|_2^2 + \underbrace{(C(\mathbf{x}_1 - \mathbf{x}_2))^\top (\boldsymbol{\lambda}_1 - \boldsymbol{\lambda}_2)}_{=0} = 0.$$

Thus $A(\mathbf{x}_1 - \mathbf{x}_2) = 0$ and also $C(\mathbf{x}_1 - \mathbf{x}_2) = 0$, which means, since $\text{rank} \begin{pmatrix} A \\ C \end{pmatrix} = n$, that $\mathbf{x}_1 = \mathbf{x}_2$, which is what we wanted to prove.

If both A and C have full rank, we may make use of the block structure of the matrix. Consider the ansatz

$$\begin{pmatrix} A^\top A & C^\top \\ C & 0 \end{pmatrix} = \begin{pmatrix} R^\top & 0 \\ G & -U^\top \end{pmatrix} \begin{pmatrix} R & G^\top \\ 0 & U \end{pmatrix}.$$

Multiplying the right hand side and equating terms we obtain

$$R^\top R = A^\top A \text{ thus } R = \text{chol}(A^\top A)$$

$$R^\top G^\top = C^\top \text{ or } GR = C \text{ thus } G = CR^{-1}$$

and

$$GG^\top - U^\top U = 0 \text{ thus } U = \text{chol}(G^*G^*).$$

The whole algorithm becomes:

1. Compute the Cholesky decomposition $R^\top R = A^\top A$.
2. Solve for G^\top by forward substituting $R^\top G^\top = C^\top$.
3. Compute the Cholesky decomposition $U^\top U = GG^\top$.
4. Solve for \mathbf{y}_1 and \mathbf{y}_2 by forward substitution,

$$\begin{pmatrix} R^\top & 0 \\ G & -U^\top \end{pmatrix} \begin{pmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{pmatrix} = \begin{pmatrix} A^\top \mathbf{b} \\ \mathbf{d} \end{pmatrix}.$$

5. Solve for \mathbf{x} and $\boldsymbol{\lambda}$ by backward substitution,

$$\begin{pmatrix} R & G^\top \\ 0 & U \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \boldsymbol{\lambda} \end{pmatrix} = \begin{pmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{pmatrix}.$$

Compared to solving the Normal Equations (6.48) with Gaussian elimination, we save $(n+p)^3/6$ multiplications using this decomposition. This saving may be not significant for large m , because the dominant computational cost in this case is in forming the matrix $A^\top A$, which requires $mn^2/2$ multiplications. Just as in the unconstrained case, the above algorithm should not be used to solve Problem (6.44) numerically, since forming $A^\top A$ and GG^\top is numerically not advisable.

6.6.3 Direct Elimination of the Constraints

We will assume that $C \in \mathbb{R}^{p \times n}$ has full rank and therefore $C\mathbf{x} = \mathbf{d}$ is consistent. Applying Gaussian elimination with *column pivoting*, we obtain the decomposition

$$CP = L[R, F]$$

with P a permutation matrix, L a unit lower triangular matrix, and R an upper triangular matrix. The constraints become

$$CPP^\top \mathbf{x} = L[R, F]\mathbf{y} = \mathbf{d} \text{ with } \mathbf{y} = P^\top \mathbf{x}.$$

If we partition $\mathbf{y} = [\mathbf{y}_1, \mathbf{y}_2]^\top$, we obtain

$$[R, F] \begin{pmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{pmatrix} = L^{-1}\mathbf{d} =: \mathbf{w}$$

with the solution \mathbf{y}_2 arbitrary and $\mathbf{y}_1 = R^{-1}(\mathbf{w} - F\mathbf{y}_2)$. Inserting $\mathbf{x} = P\mathbf{y}$ in $\|A\mathbf{x} - \mathbf{b}\|_2$ and partitioning $AP = [A_1, A_2]$, we get the unconstrained problem

$$(A_2 - A_1 R^{-1} F)\mathbf{y}_2 \approx \mathbf{b} - A_1 R^{-1} \mathbf{w}.$$

A compact formulation of the algorithm is the following:

1. Form the combined system

$$\begin{pmatrix} C \\ A \end{pmatrix} \mathbf{x} \approx \begin{pmatrix} \mathbf{d} \\ \mathbf{b} \end{pmatrix}.$$

2. Eliminate p variables by Gaussian elimination with column pivoting,

$$\begin{pmatrix} R & F \\ 0 & \tilde{A} \end{pmatrix} \begin{pmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{pmatrix} = \begin{pmatrix} \mathbf{w} \\ \tilde{\mathbf{b}} \end{pmatrix}.$$

3. Continue elimination of $\tilde{A}\mathbf{y}_2 \approx \tilde{\mathbf{b}}$ with *orthogonal transformations* (Householder or Givens) to get

$$\begin{pmatrix} R & F \\ 0 & R_2 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{pmatrix} = \begin{pmatrix} \mathbf{w} \\ \mathbf{v}_1 \\ \mathbf{v}_2 \end{pmatrix}, \text{ with } \tilde{A} = Q \begin{pmatrix} R_2 \\ 0 \end{pmatrix} \text{ and } \begin{pmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \end{pmatrix} = Q^T \tilde{\mathbf{b}}.$$

4. Solve for \mathbf{y} by back substitution

$$\begin{pmatrix} R & F \\ 0 & R_2 \end{pmatrix} \mathbf{y} = \begin{pmatrix} \mathbf{w} \\ \mathbf{v}_1 \end{pmatrix} \text{ and } \mathbf{x} = P\mathbf{y}.$$

The algorithm is implemented in the following MATLAB program

ALGORITHM 6.21. *Linearly Constrained Least Squares*

```
function x=LinearlyConstrainedLSQ(A,C,b,d)
% LINEARLYCONSTRAINEDLSQ solves linearly constrained least squares problem
% x=LinearlyConstrainedLSQ(A,C,b,d); solves ||Ax-b|| = min s.t. Cx=d
% by direct Gaussian elimination. The reduced least squares problem
% is solved using the standard Matlab \ operator.

[p,n]=size(C); [m,n]=size(A);
CA=[C,d;A,b]; % augmented matrix
pp=[1:n]; % permutation vector
for i=1:p, % eliminate p unknowns
    [h jmax]=max(abs(CA(i,i:n))); % with Gaussian elimination
    jmax=i-1+jmax;
    if h==0, error('Matrix C is rank deficient'); end
    if jmax~=i % exchange columns
        h=CA(:,i); CA(:,i)=CA(:,jmax); CA(:,jmax)=h;
        zz=pp(i); pp(i)=pp(jmax); pp(jmax)=zz;
    end;
    CA(i+1:p+m,i)=CA(i+1:p+m,i)/CA(i,i); % elimination
    CA(i+1:p+m,i+1:n+1)=CA(i+1:p+m,i+1:n+1) ...
        -CA(i+1:p+m,i)*CA(i,i+1:n+1);
end;
y2=CA(p+1:m+p,p+1:n)\CA(p+1:m+p,n+1); % solve lsq.-problem
y1=triu(CA(1:p,1:p))\ (CA(1:p,n+1)-CA(1:p,p+1:n)*y2);
x(pp)=[y1;y2]; % permute solution
x=x(:);
```

EXAMPLE 6.18. *If we interpolate the following 7 points by an interpolating polynomial of degree 6,*

```
x=[1; 2.5; 3; 5; 13; 18; 20];
y=[2; 3; 4; 5; 7; 6; 3];
```

```

plot(x,y,'o'); hold;
xx=1:0.1:20;
P=polyfit(x,y,6);           % fit degree 6 polynomial
plot(xx, polyval(P,xx),':')
pause

```

we obtain the dashed curve shown in Figure 6.5. The interpolation is really not what one would like. We can obtain a smoother interpolation for example

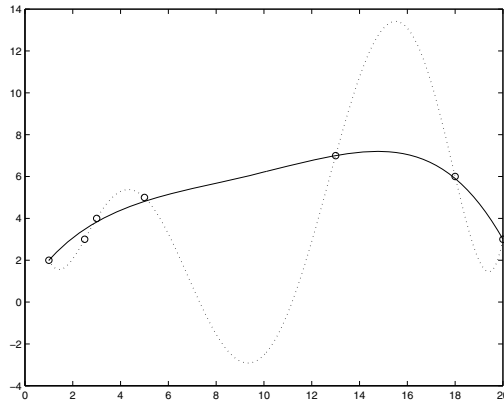


FIGURE 6.5. Polynomial Interpolation

by giving up the interpolation condition, or maybe by demanding interpolation only for a few points and using a least squares fit for the remaining ones.

In MATLAB, a polynomial of degree d with coefficients \mathbf{p} is represented as

$$P_d(x) = p_1x^d + p_2x^{d-1} + \cdots + p_dx + p_{d+1}.$$

The interpolation and approximation conditions $P_d(x_i) = y_i$ resp. $P_d(x_i) \approx y_i$ lead to the constrained least squares problem

$$V\mathbf{p} \simeq \mathbf{y},$$

with the $m \times (d+1)$ Vandermonde matrix $V = (v_{ij})$ with $v_{ij} = x_i^{d-j+1}$. We now choose the degree $d = 4$ and interpolate $p = 3$ points, the first, the last and the fifth:

```

m=length(x); n=5;           % choose degree 4
V=vander(x); V=V(:,m-n+1:m);
p=3;                         % number of interpolating points

```

We reorder the equations so that the first 3 are the ones with the interpolation conditions:

```

in=[1 5 7 2 3 4 6];        % permute equations

```

```

Vp=V(in,:); yp=y(in);
C=Vp(1:p,:); A=Vp(p+1:m,:);
d=yp(1:p); b=yp(p+1:m);
P1=LinearlyConstrainedLSQ(A,C,b,d);
plot(xx, polyval(P1,xx))

```

As we can see from Figure 6.5, we obtain this time a much more satisfactory representation of the data. Comparing the function values VP_1 with \mathbf{y} , we spot the three interpolation points and see that the others are approximated in the least squares sense.

```

>> [V*P1 y]
ans =
    2.0000    2.0000
    3.4758    3.0000
    3.8313    4.0000
    4.8122    5.0000
    7.0000    7.0000
    5.9036    6.0000
    3.0000    3.0000

```

6.6.4 Null Space Method

There are several possibilities to avoid the Normal Equations (6.48) by direct elimination of the constraints. Since we assume that C has full rank p , we can express the general solution of $C\mathbf{x} = \mathbf{d}$ using the QR decomposition instead of the SVD as in Section 6.6.1.

We compute the QR decomposition of C^\top ,

$$C^\top = [Q_1, Q_2] \begin{pmatrix} R \\ 0 \end{pmatrix}, \quad R \in \mathbb{R}^{p \times p}, Q_1 \in \mathbb{R}^{n \times p}.$$

Then the columns of Q_2 span the *null space* of C^\top : $\mathcal{N}(C) = \mathcal{R}(Q_2)$. With $Q = [Q_1, Q_2]$ and the new unknowns $\mathbf{y} = Q^\top \mathbf{x}$, the constraints become

$$C\mathbf{x} = CQ\mathbf{y} = [R^\top, 0]\mathbf{y} = R^\top \mathbf{y}_1 = \mathbf{d}, \quad \text{with} \quad \mathbf{y} = \begin{pmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{pmatrix}.$$

The general solution of the constraints is $\mathbf{y}_1 = R^{-\top} \mathbf{d}$ and \mathbf{y}_2 arbitrary. Introducing

$$A\mathbf{x} = AQQ^\top \mathbf{x} = AQ \begin{pmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{pmatrix} = A(Q_1 \mathbf{y}_1 + Q_2 \mathbf{y}_2)$$

into $\|A\mathbf{x} - \mathbf{b}\|_2$, we get an unconstrained least squares problem

$$\|AQ_2 \mathbf{y}_2 - (\mathbf{b} - AQ_1 \mathbf{y}_1)\|_2 \longrightarrow \min. \quad (6.50)$$

Thus we obtain the algorithm:

1. compute the QR decomposition $C^\top = [Q_1, Q_2] \begin{pmatrix} R \\ 0 \end{pmatrix}$.

2. Compute \mathbf{y}_1 by forward substitution $R^\top \mathbf{y}_1 = \mathbf{d}$ and $\mathbf{x}_1 = Q_1 \mathbf{y}_1$.
3. Form $\tilde{A} = AQ_2$ and $\tilde{\mathbf{b}} = \mathbf{b} - A\mathbf{x}_1$.
4. Solve $\tilde{A}\mathbf{y}_2 \approx \tilde{\mathbf{b}}$.
5. $\mathbf{x} = Q \begin{pmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{pmatrix} = \mathbf{x}_1 + Q_2 \mathbf{y}_2$.

ALGORITHM 6.22. *Null Space Method*

```
function x=NullSpaceMethod(A,C,b,d);
% NULLSPACEMETHOD solves a constrained least squares problem
%   x=NullSpaceMethod(A,C,b,d) solves the constrained least squares
%   problem ||Ax-b||=min s.t. Cx=d using the nullspace method.

[p n]=size(C);
[Q R]=qr(C');
y1=R(1:p,1:p)'\d;
x1=Q(:,1:p)*y1;
y2=(A*Q(:,p+1:n))\ (b-A*x1);
x=x1+Q(:,p+1:n)*y2;
```

6.7 Special Linear Least Squares Problems with Quadratic Constraint

The SVD can be used very effectively to solve a very *particular least squares problem with a quadratic constraint*, as shown in the following theorem.

THEOREM 6.17. *Let $A = U\Sigma V^\top$. Then the problem*

$$\|A\mathbf{x}\|_2 \longrightarrow \min, \quad \text{subject to } \|\mathbf{x}\|_2 = 1 \quad (6.51)$$

has the solution $\mathbf{x} = \mathbf{v}_n$ and the value of the minimum is $\min_{\|\mathbf{x}\|_2=1} \|A\mathbf{x}\|_2 = \sigma_n$.

PROOF. We make use of the fact that for orthogonal V and $V^\top \mathbf{x} = \mathbf{y}$ we have $\|\mathbf{x}\|_2 = \|VV^\top \mathbf{x}\|_2 = \|V\mathbf{y}\|_2 = \|\mathbf{y}\|_2$:

$$\begin{aligned} \min_{\|\mathbf{x}\|_2=1} \|A\mathbf{x}\|_2^2 &= \min_{\|\mathbf{x}\|_2=1} \|U\Sigma V^\top \mathbf{x}\|_2^2 = \min_{\|V^\top \mathbf{x}\|_2=1} \|U\Sigma(V^\top \mathbf{x})\|_2^2 \\ &= \min_{\|\mathbf{y}\|_2=1} \|\Sigma\mathbf{y}\|_2^2 = \min_{\|\mathbf{y}\|_2=1} (\sigma_1^2 y_1^2 + \cdots + \sigma_n^2 y_n^2) \geq \sigma_n^2 \end{aligned}$$

The minimum is attained for $\mathbf{y} = \mathbf{e}_n$ thus for $\mathbf{x} = V\mathbf{y} = \mathbf{v}_n$. □

Such minimization problems appear naturally in a variety of geometric fitting problems, as we show in the following subsections.

6.7.1 Fitting Lines

We consider the problem of fitting lines by minimizing the sum of squares of the distances to given points (see Chapter 6 in [45]). In the plane we can represent a straight line uniquely by the equations

$$c + n_1x + n_2y = 0, \quad n_1^2 + n_2^2 = 1. \quad (6.52)$$

The unit vector (n_1, n_2) is the normal vector orthogonal to the line. A point is on the line if its coordinates (x, y) satisfy the first equation. On the other hand, if $P = (x_P, y_P)$ is some point not on the line and we compute

$$r = c + n_1x_P + n_2y_P,$$

then $|r|$ is its distance from the line. Therefore if we want to determine the line for which the sum of squares of the distances to given points is minimal, we have to solve the constrained least squares problem

$$\begin{pmatrix} 1 & x_{P_1} & y_{P_1} \\ 1 & x_{P_2} & y_{P_2} \\ \vdots & \vdots & \vdots \\ 1 & x_{P_m} & y_{P_m} \end{pmatrix} \begin{pmatrix} c \\ n_1 \\ n_2 \end{pmatrix} \approx \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \quad \text{subject to } n_1^2 + n_2^2 = 1. \quad (6.53)$$

Let A be the matrix of the linear system (6.53). Using the QR decomposition $A = QR$, we can multiply by Q^\top on the left and reduce the linear system to $R\mathbf{x} \approx 0$, i.e., the problem becomes

$$\begin{pmatrix} r_{11} & r_{12} & r_{13} \\ 0 & r_{22} & r_{23} \\ 0 & 0 & r_{33} \end{pmatrix} \begin{pmatrix} c \\ n_1 \\ n_2 \end{pmatrix} \approx \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \quad \text{subject to } n_1^2 + n_2^2 = 1. \quad (6.54)$$

Since the nonlinear constraint only involves two unknowns, we only have to solve

$$\begin{pmatrix} r_{22} & r_{23} \\ 0 & r_{33} \end{pmatrix} \begin{pmatrix} n_1 \\ n_2 \end{pmatrix} \approx \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \quad \text{subject to } n_1^2 + n_2^2 = 1. \quad (6.55)$$

The solution \mathbf{n} is obtained using Theorem 6.17. We then obtain c by inserting the values into the first equation of (6.54).

The quadratically constrained least squares problem

$$A \begin{pmatrix} c \\ \mathbf{n} \end{pmatrix} \approx \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \quad \text{subject to } \|\mathbf{n}\|_2 = 1$$

is therefore solved by the following MATLAB function:

ALGORITHM 6.23.

Quadratically Constrained Linear Least Squares

```

function [c,n]=ConstrainedLSQ(A,dim);
% CONSTRAINEDLSQ solves a constraint least squares problem
% [c,n]=ConstrainedLSQ(A,dim) solves the constrained least squares
% problem A (c n)' ~ 0 subject to norm(n,2)=1, dim=length(n)

[m,p]=size(A);
if p<dim+1, error('not enough unknowns'); end;
if m<dim, error('not enough equations'); end;
m=min(m,p);
R=triu(qr(A));
[U,S,V]=svd(R(p-dim+1:m,p-dim+1:p));
n=V(:,dim);
c=-R(1:p-dim,1:p-dim)\R(1:p-dim,p-dim+1:p)*n;

```

Fitting two Parallel Lines

Suppose we wish to determine two parallel lines by measuring the coordinates of points that lie on them. The measurements are given as two sets of points $\{P_i\}, i = 1, \dots, p$, and $\{Q_j\}, j = 1, \dots, q$, and our goal is to find a pair of parallel lines that best fit the two sets. Since the lines are parallel, their normal vector must be the same. Thus the equations for the lines are

$$\begin{aligned} c_1 + n_1x + n_2y &= 0, \\ c_2 + n_1x + n_2y &= 0, \\ n_1^2 + n_2^2 &= 1. \end{aligned}$$

If we insert the coordinates of the two sets of points into these equations we get the following constrained least squares problem for the four unknowns n_1, n_2, c_1 and c_2 :

$$\begin{pmatrix} 1 & 0 & x_{P_1} & y_{P_1} \\ 1 & 0 & x_{P_2} & y_{P_2} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & 0 & x_{P_p} & y_{P_p} \\ 0 & 1 & x_{Q_1} & y_{Q_1} \\ 0 & 1 & x_{Q_2} & y_{Q_2} \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 1 & x_{Q_q} & y_{Q_q} \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ n_1 \\ n_2 \end{pmatrix} \approx \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \quad \text{subject to } n_1^2 + n_2^2 = 1. \quad (6.56)$$

Again, we can use our function `ConstrainedLSQ` to solve this problem.

EXAMPLE 6.19. *In the following program some measured points are defined and two parallel lines are fitted and plotted.*

```

Px=[1:10]';
Py=[0.2 1.0 2.6 3.6 4.9 5.3 6.5 7.8 8.0 9.0]';

```

```

Qx=[1.5 2.6 3.0 4.3 5.0 6.4 7.6 8.5 9.9 ]'
Qy=[5.8 7.2 9.1 10.5 10.6 10.7 13.4 14.2 14.5]'
A=[ones(size(Px)) zeros(size(Px)) Px Py
   zeros(size(Qx)) ones(size(Qx)) Qx Qy]
[c,n]=ConstrainedLSQ(A,2)
clf; hold on;
axis([-1 11 -1 17])
PlotLine(Px,Py,'o',c(1),n,'-')
PlotLine(Qx,Qy,'+',c(2),n,'-')
hold off;

```

We have used the function *PlotLine* to plot a line through given points.

ALGORITHM 6.24. Plot a line through points

```

function PlotLine(x,y,s,c,n,t)
% PLOTLINE plots a line through a set of points
% PlotLine(x,y,s,c,n,t) plots the set of points (x,y) using the
% symbol s and plots the line c+n1*x+n2*y=0 using the line type
% defined by t

plot(x,y,s)
xrange=[min(x) max(x)]; yrange=[min(y) max(y)];
if n(1)==0, % c+n2*y=0 => y=-c/n(2)
    x1=xrange(1); y1=-c/n(2);
    x2=xrange(2); y2=y1
elseif n(2)==0, % c+n1*x=0 => x=-c/n(1)
    y1=yrange(1); x1=-c/n(1);
    y2=yrange(2); x2=x1;
elseif xrange(2)-xrange(1)>yrange(2)-yrange(1),
    x1=xrange(1); y1=-(c+n(1)*x1)/n(2);
    x2=xrange(2); y2=-(c+n(1)*x2)/n(2);
else
    y1=yrange(1); x1=-(c+n(2)*y1)/n(1);
    y2=yrange(2); x2=-(c+n(2)*y2)/n(1);
end
plot([x1,x2],[y1,y2],t)

```

The results obtained by the program *mainparallel* are the two lines

$$\begin{aligned}
 0.5091 - 0.7146x + 0.6996y &= 0, \\
 -3.5877 - 0.7146x + 0.6996y &= 0,
 \end{aligned}$$

which are plotted in Figure 6.6.

6.7.2 Fitting Ellipses

We want to fit ellipses to measured points by minimizing the *algebraic distance* (see [146]). The solutions $\mathbf{x} = [x_1, x_2]$ of a quadratic equation

$$\mathbf{x}^\top \mathbf{A} \mathbf{x} + \mathbf{b}^\top \mathbf{x} + c = 0 \quad (6.57)$$

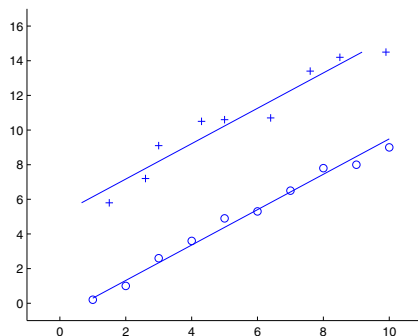


FIGURE 6.6. Two parallel lines through measured points

are points on an ellipse if A is symmetric and positive or negative definite (i.e. if $\det(A) = a_{11}a_{22} - a_{12}^2 > 0$). For each measured point, we substitute its coordinates \mathbf{x}_i into (6.57) to obtain an equation for the unknown coefficients $\mathbf{u} = [a_{11}, a_{12}, a_{22}, b_1, b_2, c]$. Note that $a_{12} = a_{21}$ because of symmetry. Since (6.57) is homogeneous in the coefficients, we need some normalizing condition in order to make the solution unique. A possibility that can handle all cases is to normalize the coefficients by $\|\mathbf{u}\|_2 = 1$. We then obtain a quadratically constrained least squares problem

$$\|B\mathbf{u}\|_2 \longrightarrow \min \quad \text{subject to } \|\mathbf{u}\|_2 = 1.$$

If the rows of the matrix X contain the coordinates of the measured points then the matrix B is computed by the MATLAB statement

```
B=[X(:,1).^2 X(:,1).*X(:,2) X(:,2).^2 X(:,1) X(:,2) ones(size(X(:,1)))]
```

The solution $\mathbf{u} = [a_{11}, a_{12}, a_{22}, b_1, b_2, c]$ is obtained using Theorem 6.17. When the coefficients \mathbf{u} are known, we can compute the geometric quantities, the axes and the center point by a *principle axis transformation*. To find a coordinate system in which the axes of the ellipse are parallel to the coordinate axes, we compute the eigenvalue decomposition

$$A = QDQ^T, \quad \text{with } Q \text{ orthogonal and } D = \text{diag}(\lambda_1, \lambda_2). \quad (6.58)$$

Introducing the new variable $\bar{\mathbf{x}} = Q^T \mathbf{x}$, (6.57) becomes

$$\bar{\mathbf{x}}^T Q^T A Q \bar{\mathbf{x}} + (Q^T \mathbf{b})^T \bar{\mathbf{x}} + c = 0,$$

or, written in components (using $\bar{\mathbf{b}} = Q^T \mathbf{b}$),

$$\lambda_1 \bar{x}_1^2 + \lambda_2 \bar{x}_2^2 + \bar{b}_1 \bar{x}_1 + \bar{b}_2 \bar{x}_2 + c = 0.$$

We transform this equation into its “normal form”

$$\frac{(\bar{x}_1 - \bar{z}_1)^2}{a^2} + \frac{(\bar{x}_2 - \bar{z}_2)^2}{b^2} = 1.$$

Here, (\bar{z}_1, \bar{z}_2) is the center in the rotated coordinate system,

$$(\bar{z}_1, \bar{z}_2) = \left(-\frac{\bar{b}_1}{2\lambda_1}, -\frac{\bar{b}_2}{2\lambda_2} \right).$$

The axes are given by

$$a = \sqrt{\frac{\bar{b}_1^2}{4\lambda_1^2} + \frac{\bar{b}_2^2}{4\lambda_1\lambda_2} - \frac{c}{\lambda_1}}, \quad b = \sqrt{\frac{\bar{b}_1^2}{4\lambda_1\lambda_2} + \frac{\bar{b}_2^2}{4\lambda_2^2} - \frac{c}{\lambda_2}}.$$

To obtain the center in the unrotated system, we have to apply the change of coordinates $\mathbf{z} = Q\bar{\mathbf{z}}$. We now have all the elements to write a MATLAB function that fits an ellipse to measured points:

ALGORITHM 6.25. *Algebraic Ellipse Fit*

```
function [z,a,b,alpha]=AlgebraicEllipseFit(X);
% ALGEBRAICELLIPSEFIT ellipse fit, minimizing the algebraic distance
% [z,a,b,alpha]=AlgebraicEllipseFit(X) fits an ellipse by minimizing
% the algebraic distance to given points P_i=[X(i,1), X(i,2)] in the
% least squares sense x'A x + bb'x + c=0. z is the center, a,b are
% the main axes and alpha the angle between a and the x-axis.

[U S V]=svd([X(:,1).^2 X(:,1).*X(:,2) X(:,2).^2 ...
             X(:,1) X(:,2) ones(size(X(:,1)))]);
u=V(:,6); A=[u(1) u(2)/2; u(2)/2 u(3)];
bb=[u(4); u(5)]; c=u(6);
[Q,D]=eig(A);
alpha=atan2(Q(2,1),Q(1,1));
bs=Q'*bb; zs=-(2*D)\bs; z=Q*zs;
h=-bs'*zs/2-c; a=sqrt(h/D(1,1)); b=sqrt(h/D(2,2));
```

To plot an ellipse, it is best to use polar coordinates:

ALGORITHM 6.26. *Drawing an Ellipse*

```
function DrawEllipse(C,a,b,alpha)
% DRAWELLIPSE plots an ellipse
% DrawEllipse(C,a,b,alpha) plots ellipse with center C, semiaxis a
% and b and angle alpha between a and the x-axis

s=sin(alpha); c=cos(alpha);
Q=[c -s; s c]; theta=[0:0.02:2*pi];
u=diag(C)*ones(2,length(theta)) + Q*[a*cos(theta); b*sin(theta)];
plot(u(1,:),u(2,:));
plot(C(1),C(2),'+');
```

EXAMPLE 6.20. We run the following program to fit an ellipse and generate Figure 6.7:

```
X = [-2.8939    4.1521
      -2.0614    2.1684
      -0.1404    1.9764
        2.6772    3.0323
        5.1746    5.7199
        3.2535    8.1196
       -0.1724    6.8398 ]
axis([0 10 0 10]); axis('equal'); hold
plot(X(:,1),X(:,2),'o');
[z,a,b,alpha]=AlgebraicEllipseFit(X)
DrawEllipse(z,a,b,alpha)
```

We obtain the results

```
z = 1.2348
    4.9871
a = 2.3734
b = 4.6429
alpha = 2.0849
```

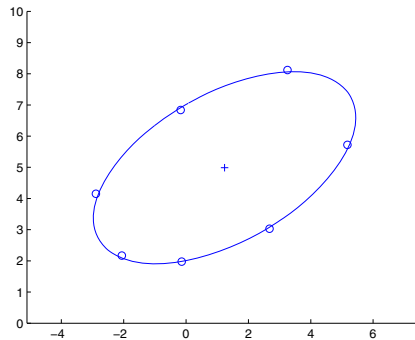


FIGURE 6.7. Fitting an Ellipse to Measured Points

6.7.3 Fitting Hyperplanes, Collinearity Test

The function `ConstrainedLSQ` can be used to fit an $(n-1)$ -dimensional hyperplane in \mathbb{R}^n to given points. Let the rows of the matrix $X = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m]^T$ contain the coordinates of the given points, i.e. point P_i has the coordinates $\mathbf{x}_i = X(i, :)$, $i = 1, \dots, m$. Then the call

```
[c, N] = ConstrainedLSQ ([ones(m,1) X], n);
```

determines the hyperplane in normal form $c + N_1y_1 + N_2y_2 + \dots + N_ny_n = 0$.

In this section, we show how we can also compute best-fit hyperplanes of lower dimensions s , where $1 \leq s \leq n - 1$. We follow the theory developed in [127] and also published in [45]. An s -dimensional hyperplane α in \mathbb{R}^n can be represented in parametric form,

$$\alpha: \quad \mathbf{y} = \mathbf{p} + \mathbf{a}_1t_1 + \mathbf{a}_2t_2 + \dots + \mathbf{a}_st_s = \mathbf{p} + A\mathbf{t}. \quad (6.59)$$

In this equation, \mathbf{p} is a point on the plane and \mathbf{a}_i are linearly independent direction vectors, thus the hyperplane is determined by the parameters \mathbf{p} and $A = [\mathbf{a}_1, \dots, \mathbf{a}_s]$.

Without loss of generality, we assume that A has orthonormal columns, i.e., $A^\top A = I_s$. If we now want to fit a hyperplane to the given set of points X , then we have to minimize the distance of the points to the plane. The distance d_i of a point $P_i = \mathbf{x}_i$ to the hyperplane is given by

$$d_i = \min_{\mathbf{t}} \|\mathbf{p} - \mathbf{x}_i + A\mathbf{t}\|_2.$$

To determine the minimum, we solve $\nabla d_i^2 = 2A^\top(\mathbf{p} - \mathbf{x}_i + A\mathbf{t}) = \mathbf{0}$ for \mathbf{t} , and, since $A^\top A = I_s$, we obtain

$$\mathbf{t} = A^\top(\mathbf{x}_i - \mathbf{p}). \quad (6.60)$$

Therefore the distance becomes

$$d_i^2 = \|\mathbf{p} - \mathbf{x}_i + AA^\top(\mathbf{x}_i - \mathbf{p})\|_2^2 = \|P(\mathbf{x}_i - \mathbf{p})\|_2^2,$$

where we denoted by $P = I - AA^\top$ the projector onto the orthogonal complement of the range of A , i.e. onto the null space of A^\top .

Our objective is to minimize the sum of squares of the distances of all points to the hyperplane, i.e., we want to minimize the function

$$F(\mathbf{p}, A) = \sum_{i=1}^m \|P(\mathbf{x}_i - \mathbf{p})\|_2^2. \quad (6.61)$$

A necessary condition is $\nabla F = \mathbf{0}$. We first consider the first part of the gradient, the partial derivative

$$\frac{\partial F}{\partial \mathbf{p}} = -\sum_{i=1}^m 2P^\top P(\mathbf{x}_i - \mathbf{p}) = -2P\left(\sum_{i=1}^m \mathbf{x}_i - m\mathbf{p}\right) = \mathbf{0},$$

where we made use of the property of an orthogonal projector $P^\top P = P^2 = P$. Since P projects the vector $\sum_{i=1}^m \mathbf{x}_i - m\mathbf{p}$ onto $\mathbf{0}$, this vector must be in the range of A , i.e.

$$\mathbf{p} = \frac{1}{m} \sum_{i=1}^m \mathbf{x}_i + A\boldsymbol{\tau}. \quad (6.62)$$

Inserting this expression into (6.61) and noting that $PA = 0$, the objective function to be minimized simplifies to

$$G(A) = \sum_{i=1}^m \|P\hat{\mathbf{x}}_i\|_2^2 = \|P\hat{X}^\top\|_F^2, \quad (6.63)$$

where we put

$$\hat{\mathbf{x}}_i = \mathbf{x}_i - \frac{1}{m} \sum_{i=1}^m \mathbf{x}_i,$$

and where we used the *Frobenius norm* of a matrix, $\|A\|_F^2 := \sum_{i,j} a_{ij}^2$, see Subsection 2.5.1. Now since P is symmetric, we may also write

$$G(A) = \|\hat{X}P\|_F^2 = \|\hat{X}(I - AA^\top)\|_F^2 = \|\hat{X} - \hat{X}AA^\top\|_F^2. \quad (6.64)$$

If we define $Y := \hat{X}AA^\top$, which is a matrix of rank s , then we can consider the problem of minimizing

$$\|\hat{X} - Y\|_F^2 \longrightarrow \min, \quad \text{subject to} \quad \text{rank}(Y) = s. \quad (6.65)$$

Problem (6.65) is similar to the problem solved in Theorem 6.6. The difference is that now we want to minimize the Frobenius norm and not the 2-norm.

THEOREM 6.18. *Let $A \in \mathbb{R}^{m \times n}$ have rank r and let $A = U\Sigma V^\top$. Let \mathcal{M} denote the set of $m \times n$ matrices with rank $p < r$. A solution of*

$$\min_{X \in \mathcal{M}} \|A - X\|_F^2$$

is given by $A_p = \sum_{i=1}^p \sigma_i \mathbf{u}_i \mathbf{v}_i^\top$ and we have

$$\min_{X \in \mathcal{M}} \|A - X\|_F^2 = \|A - A_p\|_F^2 = \sigma_{p+1}^2 + \cdots + \sigma_n^2.$$

To prove the above theorem, we first need a lemma due to Weyl [147], which estimates the singular values of the sum of two matrices in terms of those of its summands.

LEMMA 6.3. (WEYL) *Let $A, B \in \mathbb{R}^{m \times n}$. Let $\sigma_i(A)$ denote the i th singular value of A in descending order, and similarly for B . Then for $i + j \leq \min\{m, n\} + 1$, we have*

$$\sigma_{i+j-1}(A + B) \leq \sigma_i(A) + \sigma_j(B).$$

PROOF. Let A_p denote the best rank- p approximation of A , i.e., if A has rank r and the SVD of A is $A = \sum_{k=1}^r \sigma_k \mathbf{u}_k \mathbf{v}_k^\top$, then $A_p = \sum_{k=1}^p \sigma_k \mathbf{u}_k \mathbf{v}_k^\top$. Then by Theorem 6.6, we have $\|A - A_i\|_2 = \sigma_{i+1}$. Let us now consider

the matrix $R = A_{i-1} + B_{j-1}$, which has $\text{rank} \leq i + j - 2$. By the best approximation property in Theorem 6.6, we have

$$\begin{aligned}\sigma_{i+j-1}(A+B) &= \|A+B - (A+B)_{i+j-2}\|_2 \leq \|A+B - R\|_2 \\ &\leq \|A - A_{i-1}\|_2 + \|B - B_{j-1}\|_2 = \sigma_i(A) + \sigma_j(B).\end{aligned}$$

□

PROOF. (Theorem 6.18) Let $\Sigma_p = \text{diag}(\sigma_1, \dots, \sigma_p, 0, \dots, 0)$, so that $A_p = U\Sigma_p V^\top$. Then the fact that the Frobenius norm is invariant under orthogonal transformations implies

$$\|A - A_p\|_F^2 = \|\Sigma - \Sigma_p\|_F^2 = \sigma_{p+1}^2 + \dots + \sigma_n^2.$$

To prove that the choice $X = A_p$ minimizes $\|A - X\|_F^2$, let X be a matrix of rank $p < r$. Since $\sigma_{p+1}(X) = 0$, Lemma 6.3 implies for $i + p \leq n$

$$\sigma_{i+p}(A) = \sigma_{i+p}(A - X + X) \leq \sigma_i(A - X) + \sigma_{p+1}(X) = \sigma_i(A - X).$$

Thus, we have

$$\|A - X\|_F^2 = \sum_{i=1}^n \sigma_i^2(A - X) \geq \sum_{i=1}^{n-p} \sigma_i^2(A - X) \geq \sum_{i=p+1}^n \sigma_i^2(A).$$

In other words, no other choice of rank p matrix X can lead to a lower Frobenius norm error than A_p . □

If $\hat{X} = U\Sigma V^\top$, then according to Theorem 6.18 the minimizing matrix of Problem (6.65) is given by $Y = U\Sigma_s V^\top$, where

$$\Sigma_s = \text{diag}(\sigma_1, \sigma_2, \dots, \sigma_s, 0, \dots, 0).$$

Now if $Y = U\Sigma_s V^\top$, we have to find a matrix A with orthonormal columns such that $\hat{X}AA^\top = Y$. It is easy to verify that if we choose $A = V_1$ where $V_1 = V(:, 1:s)$, then $\hat{X}AA^\top = U\Sigma_s V^\top$. Thus, the singular value decomposition of \hat{X} gives us all the lower-dimensional hyperplanes that are best fits of the given set of points:

$$\mathbf{y} = \mathbf{p} + V_1 \mathbf{t}, \quad \text{with} \quad \mathbf{p} = \frac{1}{m} \sum_{i=1}^m \mathbf{x}_i.$$

Notice that $V_2 = V(:, s+1:n)$ also gives us the normal form of the hyperplane: here, the hyperplane is described as the solution of the linear equations

$$V_2^\top \mathbf{y} = V_2^\top \mathbf{p}.$$

A measure for the collinearity is the value of the minimum $\sigma_{s+1}^2 + \dots + \sigma_n^2$.

In order to compute the hyperplanes, we therefore essentially have to compute one singular value decomposition. This is done in Algorithm 6.27.

 ALGORITHM 6.27. *Computation of Hyperplanes.*

```

function [V,p]=HyperPlaneFit(X);
% HYPERPLANEFIT fits a hyperplane to a set of given points
% [V,p]=HyperPlaneFit(X); fits a hyperplane of dimension s<n to a
% set of given points X(i,:) in R^n. The hyperplane is given by
% y=p+V(:,1:s)*tau (Parametric Form) or by the equations
% V(:,s+1:n)'*(y-p)=0 (Normal Form)

m=max(size(X));
p=sum(X)'/m;
Xt=X-ones(size(X))*diag(p);
[U,S,V]=svd(Xt,0);

```

Note that the statement $[U,S,V] = \text{svd}(Qt,0)$ computes the “economy size” singular value decomposition. If Qt is an m -by- n matrix with $m > n$, then only the first n columns of U are computed, and S is an n -by- n matrix.

6.7.4 Procrustes or Registration Problem

We consider a least squares problem in *coordinate metrology* (see [5], [15]): m points of a workpiece, called the *nominal points*, are given by their exact coordinates from construction plans when the workpiece is in its *nominal position* in a reference frame. We denote the coordinate vectors of the nominal points in this position by

$$\mathbf{x}_1, \dots, \mathbf{x}_m, \quad \mathbf{x}_i \in \mathbb{R}^n, \quad 1 \leq n \leq 3.$$

Suppose now that a coordinate measuring machine gathers the same points of another workpiece. The machine records the coordinates of the *measured points*

$$\boldsymbol{\xi}_1, \dots, \boldsymbol{\xi}_m, \quad \boldsymbol{\xi}_i \in \mathbb{R}^n, \quad 1 \leq n \leq 3,$$

which will be in a different frame than the frame of reference. The problem we want to solve is to find a frame transformation that maps the given nominal points onto the measured points. This problem is solved in [45].

We need to find a translation vector \mathbf{t} and an orthogonal matrix Q with $\det(Q) = 1$ i.e., $Q^T Q = I$ such that

$$\boldsymbol{\xi}_i = Q\mathbf{x}_i + \mathbf{t}, \quad \text{for } i = 1, \dots, m. \quad (6.66)$$

For $m > 6$ in 3D-space, Equation (6.66) is an over-determined system of equations and is only consistent if the measurements have no errors. This is not the case for a real machine; therefore our problem is to determine the unknowns Q and \mathbf{t} of the least squares problem

$$\boldsymbol{\xi}_i \approx Q\mathbf{x}_i + \mathbf{t}. \quad (6.67)$$

In the one-dimensional case, we are given two sets of points on the line. The matrix Q is just the constant 1 and we have to determine a scalar t such that

$$\xi_i \approx x_i + t, \quad i = 1, \dots, m.$$

With the notation $A = (1, \dots, 1)^\top$, $\mathbf{a} = (\xi_1, \dots, \xi_m)^\top$ and $\mathbf{b} = (x_1, \dots, x_m)^\top$ the problem becomes

$$At \approx \mathbf{a} - \mathbf{b}. \quad (6.68)$$

Using the normal equations $A^\top A t = A^\top (\mathbf{a} - \mathbf{b})$ we obtain $mt = \sum_{i=1}^m (\xi_i - x_i)$ and therefore

$$t = \bar{\xi} - \bar{x}, \quad \text{with} \quad \bar{\xi} = \frac{1}{m} \sum_{i=1}^m \xi_i \quad \text{and} \quad \bar{x} = \frac{1}{m} \sum_{i=1}^m x_i. \quad (6.69)$$

We can generalize this result for $n > 1$. Consider

$$\xi_i \approx \mathbf{x}_i + \mathbf{t}, \quad i = 1, \dots, m.$$

In matrix notation, this least squares problem becomes (I is the $n \times n$ identity matrix):

$$\begin{pmatrix} I \\ I \\ \vdots \\ I \end{pmatrix} \mathbf{t} \approx \begin{pmatrix} \xi_1 - \mathbf{x}_1 \\ \xi_2 - \mathbf{x}_2 \\ \vdots \\ \xi_m - \mathbf{x}_m \end{pmatrix}.$$

The normal equations are $m\mathbf{t} = \sum_{i=1}^m (\xi_i - \mathbf{x}_i)$ and thus again

$$\mathbf{t} = \bar{\xi} - \bar{\mathbf{x}}, \quad \text{with} \quad \bar{\xi} = \frac{1}{m} \sum_{i=1}^m \xi_i \quad \text{and} \quad \bar{\mathbf{x}} = \frac{1}{m} \sum_{i=1}^m \mathbf{x}_i.$$

Hence, we have shown that *the translation \mathbf{t} is the vector connecting the two centers of gravity of the corresponding sets of points.*

Applying this result to the least squares problem for some fixed Q

$$\xi_i \approx Q\mathbf{x}_i + \mathbf{t} \iff \sum_{i=1}^m \|Q\mathbf{x}_i + \mathbf{t} - \xi_i\|_2^2 \longrightarrow \min, \quad (6.70)$$

we conclude that \mathbf{t} is the vector connecting the two centers of gravity of the point sets ξ_i and $Q\mathbf{x}_i$, i.e.,

$$\mathbf{t} = \bar{\xi} - Q\bar{\mathbf{x}}. \quad (6.71)$$

Using (6.71), we eliminate \mathbf{t} in (6.70) and consider the problem

$$G(Q) = \sum_{i=1}^m \|Q(\mathbf{x}_i - \bar{\mathbf{x}}) - (\xi_i - \bar{\xi})\|_2^2 \longrightarrow \min. \quad (6.72)$$

Introducing the new coordinates

$$\mathbf{a}_i = \mathbf{x}_i - \bar{\mathbf{x}} \quad \text{and} \quad \mathbf{b}_i = \boldsymbol{\xi}_i - \bar{\boldsymbol{\xi}}$$

the problem is:

$$G(Q) = \sum_{i=1}^m \|Q\mathbf{a}_i - \mathbf{b}_i\|_2^2 \longrightarrow \min. \quad (6.73)$$

We can collect the vectors in matrices

$$A = (\mathbf{a}_1, \dots, \mathbf{a}_m), \quad \text{and} \quad B = (\mathbf{b}_1, \dots, \mathbf{b}_m),$$

where $A, B \in \mathbb{R}^{n \times m}$ and rewrite the function G using the *Frobenius norm*

$$G(Q) = \|QA - B\|_F^2.$$

Since the Frobenius norm of a matrix is the same for the transposed matrix, we finally obtain the *Procrustes problem*[51]: find an orthogonal matrix Q such that

$$\|B^\top - A^\top Q^\top\|_F^2 \longrightarrow \min.$$

The standard form of the Procrustes problem is formulated as follows: given two matrices C and D , both in $\mathbb{R}^{m \times n}$ with $m \geq n$, find an orthogonal matrix $P \in \mathbb{R}^{n \times n}$, such that

$$\|C - DP\|_F^2 \longrightarrow \min. \quad (6.74)$$

Thus, it suffices to let $C = B^\top$ and $D = A^\top$.

To solve the Procrustes problem (6.74), we need some properties of the *Frobenius norm*, which is given by

$$\|A\|_F^2 = \sum_{i=1}^m \sum_{j=1}^n a_{i,j}^2 = \sum_{j=1}^n \|\mathbf{a}_j\|_2^2, \quad \text{where } \mathbf{a}_j \text{ is the } j\text{th column of } A. \quad (6.75)$$

Note that

$$\|A\|_F^2 = \text{tr}(A^\top A) = \sum_{i=1}^n \lambda_i(A^\top A). \quad (6.76)$$

Equation (6.76) gives us some useful relations: if P is orthogonal, then $\|PA\|_F = \|A\|_F$. Additionally, since $\|A\|_F = \|A^\top\|_F$, we have $\|AP\|_F = \|A\|_F$.

$$\begin{aligned} \|A + B\|_F^2 &= \text{tr}((A + B)^\top (A + B)) \\ &= \text{tr}(A^\top A + B^\top A + A^\top B + B^\top B) \\ &= \text{tr}(A^\top A) + 2\text{tr}(A^\top B) + \text{tr}(B^\top B) \\ \|A + B\|_F^2 &= \|A\|_F^2 + \|B\|_F^2 + 2\text{tr}(A^\top B) \end{aligned} \quad (6.77)$$

We now apply (6.77) to the Procrustes problem:

$$\|C - DP\|_F^2 = \|C\|_F^2 + \|D\|_F^2 - 2 \operatorname{tr}(P^\top D^\top C) \longrightarrow \min.$$

Computing the minimum is equivalent to maximizing

$$\operatorname{tr}(P^\top D^\top C) = \max.$$

Using the singular value decomposition $D^\top C = U\Sigma V^\top$, we obtain

$$\operatorname{tr}(P^\top D^\top C) = \operatorname{tr}(P^\top U\Sigma V^\top).$$

Since U , V are orthogonal, we may write the unknown matrix P in the following form

$$P = UZ^\top V^\top, \quad \text{with } Z \text{ orthogonal.}$$

Because similar matrices have the same trace, it follows that

$$\begin{aligned} \operatorname{tr}(P^\top D^\top C) &= \operatorname{tr}(VZU^\top U\Sigma V^\top) = \operatorname{tr}(VZ\Sigma V^\top) = \operatorname{tr}(Z\Sigma) \\ &= \sum_{i=1}^n z_{ii}\sigma_i \leq \sum_{i=1}^n |z_{ii}|\sigma_i \leq \sum_{i=1}^n \sigma_i, \end{aligned}$$

where the inequality follows from $|z_{ii}| \leq 1$ for any orthogonal matrix Z . Furthermore, the bound is attained for $Z = I$. Notice that if $D^\top C$ is rank deficient, the solution is not unique (cf. [65]). So we have proved the following theorem:

THEOREM 6.19. *The Procrustes problem (6.74) is solved by the orthogonal polar factor of $D^\top C$, i.e. $P = UV^\top$ where $U\Sigma V^\top$ is the singular value decomposition of $D^\top C$.*

The *polar decomposition* of a matrix is a generalization of the polar representation of complex numbers. The matrix is decomposed into the product of an orthogonal times a symmetric positive (semi-)definite matrix. The decomposition can be computed by the singular value decomposition or by other algorithms [40]. In our case we have

$$D^\top C = U\Sigma V^\top = \underbrace{UV^\top}_{\text{orthogonal}} \underbrace{V\Sigma V^\top}_{\substack{\text{positive} \\ \text{semidefinite}}}.$$

We are now ready to describe the algorithm to solve the Procrustes problem. Given measured points ξ_i and corresponding nominal points x_i for $i = 1, \dots, m$. We want to determine t and Q orthogonal such that $\xi_i \approx Qx_i + t$.

1. Compute the centers of gravity:

$$\bar{\xi} = \frac{1}{m} \sum_{i=1}^m \xi_i \quad \text{and} \quad \bar{x} = \frac{1}{m} \sum_{i=1}^m x_i.$$

2. Compute the *relative coordinates*:

$$\begin{aligned} A &= [\mathbf{a}_1, \dots, \mathbf{a}_m], & \mathbf{a}_i &= \mathbf{x}_i - \bar{\mathbf{x}} \\ B &= [\mathbf{b}_1, \dots, \mathbf{b}_m], & \mathbf{b}_i &= \boldsymbol{\xi}_i - \bar{\boldsymbol{\xi}} \end{aligned}$$

3. The Procrustes problem becomes $\|C - DP\|_F^2 \rightarrow \min$ with $C = B^\top$, $D = A^\top$ and $P = Q^\top$.

Compute the singular value decomposition $AB^\top = U\Sigma V^\top$.

4. Then $Q^\top = UV^\top$ or $Q = VU^\top$ and $\mathbf{t} = \bar{\boldsymbol{\xi}} - Q\bar{\mathbf{x}}$.

For technical reasons, it may be important to decompose the orthogonal matrix Q into elementary rotations. The algorithm that we developed so far computes an orthogonal matrix, but there is no guarantee that Q can be represented as a product of rotations and that no reflection occurs. For Q to be representable as a product of rotations, it is necessary and sufficient to have $\det(Q) = 1$. Thus, if $\det(Q) = -1$, then a reflection is necessary and this may be of no practical use. In this case, one would like to find the best orthogonal matrix with $\det(Q) = 1$.

It is shown in [65] that the constrained Procrustes problem

$$\|C - DP\|_F^2 \rightarrow \min, \quad \text{subject to} \quad \det(P) = 1$$

has the solution

$$P = U \operatorname{diag}(1, \dots, 1, \mu) V^\top,$$

where $D^\top C = U\Sigma V^\top$ is the singular value decomposition and $\mu = \det(UV^\top)$. The proof is based on the fact that for a real orthogonal $n \times n$ matrix Z with $\det(Z) < 1$, the trace is bounded by

$$\operatorname{tr}(Z) \leq n - 2 \quad \text{and} \quad \operatorname{tr}(Z) = n - 2 \iff \lambda_i(Z) = \{1, \dots, 1, -1\}.$$

This can be seen by considering the real Schur form [51] of Z . The maximum of $\operatorname{tr}(Z\Sigma)$ is therefore $\sum_{i=1}^{n-1} \sigma_i - \sigma_n$ and is achieved for $Z = \operatorname{diag}(1, \dots, 1, -1)$.

Thus we obtain the MATLAB function `ProcrustesFit`

ALGORITHM 6.28.

```
function [t,Q]=ProcrustesFit(xi,x);
% PROCRUSTESFIT solves the Procrustes Problem
% [t,Q]=ProcrustesFit(xi,x) computes an orthogonal matrix Q and a
% shift t such that xi=Qx+t

xiq=sum(xi')/length(xi); xiq=xiq';
xq=sum(x')/length(x); xq=xq';
A=x-xq*ones(1,length(x));
B=xi-xiq*ones(1,length(xi));
[U,sigma,V]=svd(A*B');
Q=V*diag([ones(1,size(V,1)-1) det(V*U')])*U';
t=xiq-Q*xq;
```

EXAMPLE 6.21. As an example, we solve a Procrustes problem for $n = 2$. The following MATLAB program defines the nominal points \mathbf{x}_k and the measured points ξ_k . Then it computes and plots the fitted nominal points on the measurements.

```
clf, clear
axis([0 10 0 10]), axis('equal')           % nominal points
hold, grid
x=[-4 -4 -2 -2 -2 -4 -4
   1  2  2  3  4  4  3];
plot(x(1,:),x(2,:),'-')                   % measured points
plot(x(1,:),x(2,:), 'x')

xi=[-5.2572 -4.5528 -3.6564 -2.8239 -2.0555 -3.1761 -4.2007
    6.1206  6.6969  5.4162  6.0886  6.6329  8.2338  7.8175];
plot(xi(1,:),xi(2,:), 'o')
pause
xiq=sum(xi,2)/length(xi);                  % centers of gravity
xq=sum(x,2)/length(x);
[t,Q]=ProcrustesFit(xi,x);
xx=Q*x+t*ones(1,length(x))                % transform nominal points
plot(xx(1,:), xx(2,:), '-'), plot(xx(1,:), xx(2,:), '*')
```

The results are shown in Figure 6.8

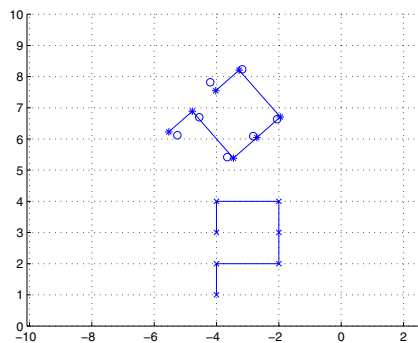


FIGURE 6.8. Procrustes or Registration Problem

6.7.5 Total Least Squares

The linear least squares problem $A\mathbf{x} \approx \mathbf{b}$ has so far been solved by projecting the vector \mathbf{b} onto the range of A ,

$$A\mathbf{x} = P_{\mathcal{R}(A)}\mathbf{b} = AA^+\mathbf{b}.$$

With “Total Least Squares”, the system of equations is made consistent by changing both A and \mathbf{b} : we look for a matrix \hat{A} and a vector $\hat{\mathbf{b}} \in \mathcal{R}(\hat{A})$ which differ as little as possible from the given data

$$\|[A, \mathbf{b}] - [\hat{A}, \hat{\mathbf{b}}]\|_F \longrightarrow \min, \quad \text{subject to} \quad \hat{\mathbf{b}} \in \mathcal{R}(\hat{A}).$$

The constraint states that $\hat{C} = [\hat{A}, \hat{\mathbf{b}}]$ must have rank n . Since in general $C = [A, \mathbf{b}]$ will have rank $n + 1$, our problem involves solving

$$\min_{\text{rank } \hat{C}=n} \|C - \hat{C}\|_F. \quad (6.78)$$

The solution of problem (6.78) is given by Theorem 6.17: let $[A, \mathbf{b}] = C = U\Sigma V^\top$ be the SVD. Then

$$[\hat{A}, \hat{\mathbf{b}}] = \hat{C} = \sum_{i=1}^n \sigma_i \mathbf{u}_i \mathbf{v}_i^\top = U \hat{\Sigma} V^\top, \quad \text{with} \quad \hat{\Sigma} = \text{diag}(\sigma_1, \dots, \sigma_n, 0). \quad (6.79)$$

However, the constraint $\hat{\mathbf{b}} \in \mathcal{R}(\hat{A})$ is more than just a rank condition: if we define $\hat{C} = C + \Delta$ and write the condition $\hat{\mathbf{b}} \in \mathcal{R}(\hat{A})$ as $\hat{C}\mathbf{z} = 0$ with $\mathbf{z} = \begin{pmatrix} \mathbf{x} \\ -1 \end{pmatrix} \neq 0$, then the problem is equivalent to

$$\|\Delta\|_F^2 \longrightarrow \min \quad \text{subject to} \quad \exists \mathbf{x} \in \mathbb{R}^n : (C + \Delta) \begin{pmatrix} \mathbf{x} \\ -1 \end{pmatrix} = 0. \quad (6.80)$$

This is equivalent to saying that there exists a right singular vector \mathbf{v} corresponding to a zero singular value such that its last component does not vanish. Thus, if $\text{rank}(C) = n + 1$ and $v_{n+1, n+1} \neq 0$, then the total least squares solution exists and is given by \hat{C} in (6.79):

$$\hat{A}\hat{\mathbf{x}} = \hat{\mathbf{b}}, \quad \hat{\mathbf{x}} = -\frac{1}{v_{n+1, n+1}} \mathbf{v}(1:n, n+1).$$

In this case, the perturbation is given by

$$\Delta = \hat{C} - C = -\sigma_{n+1} \mathbf{u}_{n+1} \mathbf{v}_{n+1}^\top.$$

This leads to the following MATLAB function:

ALGORITHM 6.29. *Total Least Squares*

```
function [x,v,At,bt]=TLS(A,b);
% TLS total least squares
% [x,v,At,bt]=TLS(A,b) solves Ax~b by allowing changes in A and b.

C=[A,b];
[m,n]=size(C);
[U,Sigma,V]=svd(C,0);
```

```

v=V(:,n);
if v(n)==0
    disp('TLS Solution does not exist')
    x=[];
else
    x=-v(1:n-1)/v(n);
end
Ct=C-Sigma(n)*U(:,n)*V(:,n)';
At=Ct(:,1:n-1); bt=Ct(:,n);

```

THEOREM 6.20. (TOTAL LEAST SQUARES) *The total least squares solution satisfies the equation*

$$(A^\top A - \sigma_{n+1}^2 I) \hat{\mathbf{x}} = A^\top \mathbf{b}. \quad (6.81)$$

PROOF. Since $C = [A, \mathbf{b}] = U\Sigma V^\top$ we have $C\mathbf{v}_{n+1} = \sigma_{n+1}\mathbf{u}_{n+1}$ and therefore

$$C^\top C\mathbf{v}_{n+1} = \sigma_{n+1}C^\top \mathbf{u}_{n+1} = \sigma_{n+1}^2 \mathbf{v}_{n+1}.$$

Dividing the last equation by $v_{n+1,n+1}$ and replacing $C = [A, \mathbf{b}]$ we obtain

$$\left(\begin{array}{c|c} A^\top A & A^\top \mathbf{b} \\ \hline \mathbf{b}^\top A & \mathbf{b}^\top \mathbf{b} \end{array} \right) \begin{pmatrix} \hat{\mathbf{x}} \\ -1 \end{pmatrix} = \sigma_{n+1}^2 \begin{pmatrix} \hat{\mathbf{x}} \\ -1 \end{pmatrix},$$

and the first equation is our claim. \square

A variant of total least squares is given if some elements of the matrix A have no errors and therefore should remain unchanged. Let us consider $A\mathbf{x} \approx \mathbf{b}$ with $A = [A_1, A_2] \in \mathbb{R}^{m \times n}$ where $A_1 \in \mathbb{R}^{m \times p}$ with $p < n$ has no error. The total least squares problem $\|E\|_F^2 + \|\mathbf{r}\|_2^2 \rightarrow \min$ subject to $(A + E)\mathbf{x} = \mathbf{b} + \mathbf{r}$ becomes

$$\|\Delta\|_F^2 \rightarrow \min \quad \text{subject to} \quad (C + \Delta)\mathbf{z} = 0,$$

with $C = [A_1, A_2, \mathbf{b}]$, $\Delta = [0, \Delta_2]$ and $\Delta_2 = [E_2, \mathbf{r}]$. Using the QR decomposition of C , we can transform the constraint by pre-multiplying with Q^\top ,

$$Q^\top (C + \Delta)\mathbf{z} = \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix} \mathbf{z} + \begin{pmatrix} 0 & \tilde{\Delta}_{12} \\ 0 & \tilde{\Delta}_{22} \end{pmatrix} \mathbf{z} = 0, \quad (6.82)$$

and $R_{11} \in \mathbb{R}^{p \times p}$ is upper triangular. Now $\|\Delta\|_F^2 = \|Q^\top \Delta\|_F^2 = \|\tilde{\Delta}_{12}\|_F^2 + \|\tilde{\Delta}_{22}\|_F^2$ is minimized if we choose $\tilde{\Delta}_{12} = 0$ and minimize $\|\tilde{\Delta}_{22}\|_F$. So the algorithm for *constrained total least squares* is:

1. Compute $C = [A_1, A_2, \mathbf{b}] = QR$ and reduce the constraint to (6.82).
2. Compute $\hat{\mathbf{v}}$ the solution of the total least squares problem $(R_{22} + \tilde{\Delta}_{22})\mathbf{v} = 0$.

3. Solve $R_{11}\mathbf{u} = -R_{12}\hat{\mathbf{v}}$ for \mathbf{u} .
4. $\mathbf{z} = \begin{pmatrix} \mathbf{u} \\ \hat{\mathbf{v}} \end{pmatrix} \in \mathbb{R}^{n+1}$ and $\mathbf{x} = -[z_1, \dots, z_n]^\top / z_{n+1}$.

ALGORITHM 6.30. *Constrained Total Least Squares*

```
function [z,x]=ConstrainedTLS(C,p);
% CONSTRAINEDTLS computes constrained total least squares solution
% [z,x]=ConstrainedTLS(C,p); computes the constrained total least
% squares solution with C=[A_1 A_2 b], where A1=C(:,1:p) is without
% errors. One therefore solves [A_1 A_2]x ~ b by modifying A2 and b
% such that [A_1 A_2 b]z=0 or [A_1 A_2] x=b

[m,n]=size(C);
[Q,R]=qr(C);
R11=R(1:p,1:p); R12=R(1:p,p+1:n);
R22=R(p+1:n,p+1:n);
[m1,n1]=size(R22);
[x,v]=TLS(R22(:,1:n1-1),R22(:,n1));
u=-R11\R12*v;
z=[u;v]; x=-z(1:n-1)/z(n);
```

EXAMPLE 6.22. We want to fit a 3-dimensional hyperplane in \mathbb{R}^4 ,

$$z = c + n_1x_1 + n_2x_2 + n_3x_3.$$

We shall simulate measured points, compute the coefficients c, n_1, n_2, n_3 and compare the results for our least squares methods.

The columns of the matrix X^\top contain the coordinates of the “measurements” for x_1, x_2 and x_3 . For simplicity we chose a grid of integer coordinates,

$$X^\top = \begin{pmatrix} 1 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 3 & 3 & 3 & 3 \\ 1 & 1 & 2 & 2 & 1 & 1 & 2 & 2 & 1 & 1 & 2 & 2 \\ 1 & 2 & 1 & 2 & 1 & 2 & 1 & 2 & 1 & 2 & 1 & 2 \end{pmatrix}.$$

We choose $c = 3$, $n_1 = -1$, $n_2 = 2$, $n_3 = 5$, add a first column of ones to X and compute the exact values

$$\mathbf{z} = (\mathbf{1}, \quad X) \begin{pmatrix} c \\ n_1 \\ n_2 \\ n_3 \end{pmatrix}.$$

Then we add some random noise to the exact values and use the algorithms for least squares, total least squares and constrained total least squares to compute the coefficients from the data. With constrained total least squares

we compute 4 variants where we assume one to four columns of X to be without errors.

```
clear, format compact
Xt=[ 1 1 1 1 2 2 2 2 3 3 3 3           % generate points
     1 1 2 2 1 1 2 2 1 1 2 2
     1 2 1 2 1 2 1 2 1 2 1 2];
X=Xt'; [m,n]=size(X);
cc=[3, -1, 2, 5]';           % z=c+n_1*x_1+n_2*x_2+n_3*x_3
X=[ones(m,1) X];
ze=X*cc;                     % exact values for z
z=ze+(rand(m,1)-0.5);       % add noise
[ze,z]
c1=X\z;                      % Least squares fit
[c,N]=ConstrainedLSQ([X,z],4); % Fit plane using ConstrainedLSQ
c2=-[c N(1) N(2) N(3)]'/N(4);
[c3,v,At,bt]=TLS(X,z);      % TLS
C=[X,z];                    % Constrained TLS
[zz,c4]=ConstrainedTLS(C,1); % first column with no errors
[zz,c5]=ConstrainedTLS(C,2); % 2 columns with no errors
[zz,c6]=ConstrainedTLS(C,3); % 3 columns with no errors
[zz,c7]=ConstrainedTLS(C,4); % 4 columns with no errors
[c1 c2 c3 c4 c5 c6 c7]
```

With this program we obtained the values

| <i>exact</i> | <i>with noise</i> |
|--------------|-------------------|
| z_e | z |
| 9 | 9.2094 |
| 14 | 14.2547 |
| 11 | 10.7760 |
| 16 | 16.1797 |
| 8 | 8.1551 |
| 13 | 12.6626 |
| 10 | 9.6190 |
| 15 | 14.9984 |
| 7 | 7.4597 |
| 12 | 11.8404 |
| 9 | 9.0853 |
| 14 | 13.7238 |

and the computed coefficients are displayed in Table 6.2. Observe that the least squares solution (column 1) and the total least squares solution with all four columns without errors (last column) yield the same solution. Also minimizing the geometric distance using *ConstrainedLSQ* (column 2) and *CTLS 1* (column 4) compute the same, which shows that these two methods are mathematically equivalent.

| | LSQ | CLSQ | TLS | CTLS 1 | CTLS 2 | CTLS 3 | CTLS 4 |
|-------|---------|---------|---------|---------|---------|---------|---------|
| c | 3.5358 | 3.4695 | 3.6021 | 3.4695 | 3.4612 | 3.4742 | 3.5358 |
| n_1 | -1.0388 | -1.0416 | -1.0543 | -1.0416 | -1.0388 | -1.0388 | -1.0388 |
| n_2 | 1.8000 | 1.8129 | 1.7820 | 1.8129 | 1.8134 | 1.8000 | 1.8000 |
| n_3 | 4.8925 | 4.9275 | 4.8900 | 4.9275 | 4.9289 | 4.9336 | 4.8925 |

TABLE 6.2. *Computed Coefficients*

6.8 Nonlinear Least Squares Problems

Let us consider some (nonlinear) function $\mathbf{f} : \mathbb{R}^n \mapsto \mathbb{R}^m$ with $n \leq m$. We want to find a point $\mathbf{x} \in \mathbb{R}^n$ such that $\mathbf{f}(\mathbf{x}) \approx 0$, or written in components,

$$f_i(x_1, x_2, \dots, x_n) \approx 0, \quad i = 1, \dots, m. \quad (6.83)$$

EXAMPLE 6.23. In Example 6.2 we have $\mathbf{x} = (a_0, a_1, b)^\top \in \mathbb{R}^3$ and $\mathbf{f} : \mathbb{R}^3 \mapsto \mathbb{R}^m$, where

$$f_i(\mathbf{x}) = y_i - x_1 - x_2 e^{-x_3 t_i} \approx 0, \quad i = 1, \dots, m.$$

Now by $\mathbf{f}(\mathbf{x}) \approx 0$, we mean we should make the 2-norm of \mathbf{f} as small as possible:

$$\|\mathbf{f}(\mathbf{x})\|_2^2 = \sum_{i=1}^m f_i(\mathbf{x})^2 \longrightarrow \min. \quad (6.84)$$

Just like for the linear least squares case, we associate to a given vector \mathbf{x} the *residual vector* $\mathbf{r} = \mathbf{f}(\mathbf{x})$, whose 2-norm is the *residual sum of squares* that we seek to minimize.

6.8.1 Notations and Definitions

In order to develop algorithms to solve Problem (6.84), we need to recall some mathematical notations from calculus. Let $f : \mathbb{R}^n \mapsto \mathbb{R}$ be a continuous and differentiable function.

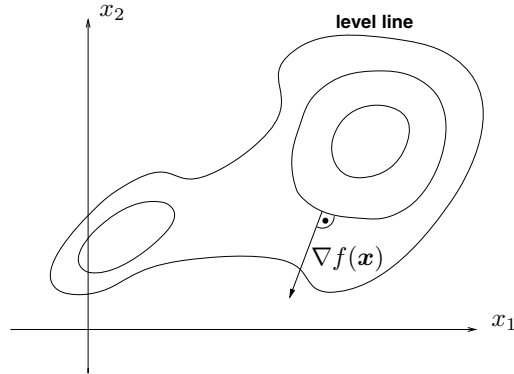
DEFINITION 6.7. (GRADIENT) *The gradient of f is the vector*

$$\text{grad } f = \nabla f = \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} = \left(\frac{\partial f(\mathbf{x})}{\partial x_1}, \frac{\partial f(\mathbf{x})}{\partial x_2}, \dots, \frac{\partial f(\mathbf{x})}{\partial x_n} \right)^\top.$$

DEFINITION 6.8. (HESSIAN) *The Hessian of f is the $n \times n$ matrix*

$$\text{hess } f = \nabla^2 f = \frac{\partial^2 f(\mathbf{x})}{\partial \mathbf{x}^2} = H = (h_{ij}), \quad \text{with } h_{ij} = \frac{\partial^2 f(\mathbf{x})}{\partial x_i \partial x_j}.$$

Note that the Hessian is a symmetric matrix (if the derivatives commute).

FIGURE 6.9. Illustration of the gradient $\nabla f(\mathbf{x})$.

For $n = 2$, we can draw the function f as a set of level curves (see Figure 6.9). The gradient at a given point \mathbf{x} is a vector that points in the direction along which the function f increases most rapidly. It is also orthogonal to the level curve, as one can see using the Taylor expansion seen earlier in (5.94):

$$f(\mathbf{x} + \mathbf{h}) = f(\mathbf{x}) + \nabla f(\mathbf{x})^\top \mathbf{h} + \frac{1}{2} \mathbf{h}^\top \nabla^2 f(\mathbf{x}) \mathbf{h} + O(\|\mathbf{h}\|_2^3). \quad (6.85)$$

Since the a level curve is by definition a curve along which the value of f remains constant, if we choose the direction \mathbf{h} to be along a level curve and only move a short distance away from \mathbf{x} , then we must have $f(\mathbf{x} + \mathbf{h}) = f(\mathbf{x}) + O(\|\mathbf{h}\|_2^2)$. Thus, we deduce that $\nabla f(\mathbf{x})^\top \mathbf{h} = 0$, i.e., the gradient is orthogonal to the level curve.

For a vector function $\mathbf{f} \in \mathbb{R}^m$, we expand each component and obtain

$$\begin{aligned} f_1(\mathbf{x} + \mathbf{h}) &= f_1(\mathbf{x}) + \nabla f_1(\mathbf{x})^\top \mathbf{h} + O(\|\mathbf{h}\|_2^2), \\ \vdots & \quad \quad \quad \vdots \\ f_m(\mathbf{x} + \mathbf{h}) &= f_m(\mathbf{x}) + \nabla f_m(\mathbf{x})^\top \mathbf{h} + O(\|\mathbf{h}\|_2^2), \end{aligned}$$

or in matrix notation

$$\mathbf{f}(\mathbf{x} + \mathbf{h}) \approx \mathbf{f}(\mathbf{x}) + J(\mathbf{x})\mathbf{h},$$

where we have introduced the $m \times n$ Jacobian matrix

$$J(\mathbf{x}) = \begin{pmatrix} \nabla f_1^\top \\ \vdots \\ \nabla f_m^\top \end{pmatrix} = \begin{pmatrix} \nabla f_1, & \dots, & \nabla f_m \end{pmatrix}^\top = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \dots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \vdots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \dots & \frac{\partial f_m}{\partial x_n} \end{pmatrix}.$$

Notice the special case that the Hessian of the scalar function $f(\mathbf{x})$ is the same as the Jacobian of $\nabla f(\mathbf{x})$. In this case, the Jacobian is a square $n \times n$ matrix.

Let us now recall Newton's method for solving a system of nonlinear equations, which was presented in Chapter 5. Let $\mathbf{f}(\mathbf{x}) : \mathbb{R}^n \rightarrow \mathbb{R}^n$; we are looking for a solution of $\mathbf{f}(\mathbf{x}) = 0$. Expanding \mathbf{f} at some approximation \mathbf{x}_k , we obtain

$$\mathbf{f}(\mathbf{x}) = \mathbf{f}(\mathbf{x}_k) + J(\mathbf{x}_k)\mathbf{h} + O(\|\mathbf{h}\|_2^2), \quad \text{with } \mathbf{h} = \mathbf{x} - \mathbf{x}_k.$$

Instead of solving $\mathbf{f}(\mathbf{x}) = 0$, we solve the linearized system

$$\mathbf{f}(\mathbf{x}_k) + J(\mathbf{x}_k)\mathbf{h} = 0$$

for the *Newton correction* \mathbf{h} and obtain a (hopefully better) approximation

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{h} = \mathbf{x}_k - J(\mathbf{x}_k)^{-1}\mathbf{f}(\mathbf{x}_k). \quad (6.86)$$

Note that computationally we would not invert the Jacobian; we would instead solve by Gaussian Elimination the linear system

$$J(\mathbf{x}_k)\mathbf{h} = -\mathbf{f}(\mathbf{x}_k)$$

for the correction \mathbf{h} .

6.8.2 Newton's Method

Given m nonlinear equations with n unknowns ($n \leq m$), we want to solve $\mathbf{f}(\mathbf{x}) \approx 0$ in the least squares sense, that is, we want

$$\Phi(\mathbf{x}) = \frac{1}{2}\|\mathbf{f}(\mathbf{x})\|_2^2 \longrightarrow \min. \quad (6.87)$$

A necessary condition for minimizing $\Phi(\mathbf{x})$ is $\nabla\Phi = 0$. We express this condition in terms of \mathbf{f} :

$$\frac{\partial\Phi(\mathbf{x})}{\partial x_i} = \sum_{l=1}^m f_l(\mathbf{x}) \frac{\partial f_l}{\partial x_i}, \quad i = 1, \dots, n, \quad (6.88)$$

or in matrix notation

$$\nabla\Phi(\mathbf{x}) = J(\mathbf{x})^\top \mathbf{f}(\mathbf{x}).$$

Thus we obtain as a necessary condition for minimizing $\Phi(\mathbf{x})$ a nonlinear system of n equations in n unknowns:

$$J(\mathbf{x})^\top \mathbf{f}(\mathbf{x}) = 0. \quad (6.89)$$

We can compute a solution of (6.89) using Newton's method (6.86). To do so, we need the Jacobian of $\nabla\Phi(\mathbf{x})$, which is the Hessian of $\Phi(\mathbf{x})$. If \mathbf{x}_k is

an approximation then we obtain the Newton correction by solving a linear system:

$$\nabla^2 \Phi(\mathbf{x}_k) \mathbf{h} = -J(\mathbf{x}_k)^\top \mathbf{f}(\mathbf{x}_k).$$

Let us express the Hessian in terms of the function \mathbf{f} . From (6.88), we compute the second derivatives,

$$\frac{\partial^2 \Phi(\mathbf{x})}{\partial x_i \partial x_j} = \sum_{l=1}^m \frac{\partial f_l}{\partial x_j} \frac{\partial f_l}{\partial x_i} + \sum_{l=1}^m f_l(\mathbf{x}) \frac{\partial^2 f_l}{\partial x_i \partial x_j}. \quad (6.90)$$

Now $\partial^2 f_l / \partial x_i \partial x_j$ is the (i, j) element of the Hessian of $f_l(\mathbf{x})$. Furthermore

$$\sum_{l=1}^m \frac{\partial f_l}{\partial x_j} \frac{\partial f_l}{\partial x_i} = \mathbf{J}^\top_{:j} \mathbf{J}_{:i}$$

is the scalar product of columns i and j of the Jacobian matrix. Therefore, we obtain in matrix notation

$$\nabla^2 \Phi(\mathbf{x}) = J^\top J + \sum_{l=1}^m f_l(\mathbf{x}) \nabla^2 f_l(\mathbf{x}).$$

The *Newton iteration for the nonlinear least squares problem* $\mathbf{f}(\mathbf{x}) \approx 0$ becomes

1. solve the linear system for the correction \mathbf{h}

$$\left(J(\mathbf{x}_k)^\top J(\mathbf{x}_k) + \sum_{l=1}^m f_l(\mathbf{x}_k) \nabla^2 f_l(\mathbf{x}_k) \right) \mathbf{h} = -J(\mathbf{x}_k)^\top \mathbf{f}(\mathbf{x}_k) \quad (6.91)$$

2. iterate: $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{h}$.

EXAMPLE 6.24. Let us return to Example 6.2,

$$f_l(\mathbf{x}) = x_1 + x_2 e^{-x_3 t_l} - y_l.$$

The gradient and the Hessian are readily computed using MAPLE:

```
with(LinearAlgebra):
with(VectorCalculus):
BasisFormat(false):
f:=x1+x2*exp(-x3*t)-y;
Gradient(f,[x1,x2,x3]);
Hessian(f,[x1,x2,x3]);
```

We obtain

$$\nabla f_l = \begin{pmatrix} 1 & e^{-x_3 t_l} & -x_2 t_l e^{-x_3 t_l} \end{pmatrix}^\top$$

and

$$\nabla^2 f_l = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & -t_l e^{-x_3 t_l} \\ 0 & -t_l e^{-x_3 t_l} & x_2 t_l^2 e^{-x_3 t_l} \end{pmatrix}$$

We can now write a program for the solution in MATLAB:

ALGORITHM 6.31.

Newton Method for Nonlinear Least Squares

```
function [x,fv]=NewtonLSQ(x,xi,eta)
% NEWTONLSQ Curve fitting with Newton's method
% [x,fv]=NewtonLSQ(x,xi,eta) fits the function f(t)=x1+x2*exp(-x3*t)
% to given points (xi,eta). x is the initial guess and is overwritten
% with fitted parameters x1,x2 and x3. fv contains norm(f) for
% each iteration

h=x; fv=[];
while norm(h)>100*eps*norm(x)
    ee=exp(-x(3)*xi);
    tee=xi.*ee;
    J=[ones(size(xi)),ee,-x(2)*tee ];           % Jacobian
    f=x(1)+x(2)*ee-eta;                          % residual
    s1=f'*tee; s2=x(2)*f'*(xi.*tee);
    A=J'*J+[0 0 0; 0 0 -s1; 0 -s1 s2];           % Hessian of Phi
    h=-A\'(J'*f);
    x=x+h;
    xh=[x,h]
    res=norm(f)
    fv=[fv norm(f)];
end
```

In order to test the program we generate a test example:

```
format compact, format long
a0=1; a1=2; b=0.15;
xi=[1:0.3:7]'; etae=a0+a1*exp(-b*xi);          % compute exact values
rand('seed',0);                                % perturb to simulate
                                                % measurements
eta=etae+0.1*(rand(size(etae))-0.5);
[x1,fv1]= NewtonLSQ([1.8 1.8 0.1]',xi,eta); % first example
plot([1:14],fv1(1:14),'-')
pause                                           % second example using a
[x2,fv2]= NewtonLSQ([1.5 1.5 0.1]',xi,eta); % different initial guess
plot([1:14], fv1(1:14),'-',[1:14], fv2(1:14),'-')
```

The results are very interesting: the iterations converge to different solutions. Because we print intermediate results (the matrix xh , showing the current solution and the correction) we can observe quadratic convergence

in both cases. In the first case with the starting values $[1.8, 1.8, 0.1]$ we obtain the parameters $x_1 = [2.1366, 0.0000, 0.0000]$, which means that the fitted function is the constant $f(t) = 2.1366$. In the second case we obtain $x_2 = [1.1481, 1.8623, 0.1702]$, thus $f(t) = 1.1481 + 1.8623e^{-0.1702 t}$, a much better fit. Figure 6.10 shows the residual sum of squares for both cases. We see that in the first case (solid line) the iteration is trapped in a local minimum! After 6 iteration steps both sequences have reduced the residual sum of squares to about the final value, however, the first sequence does not converge there but moves away to a local minimum with a higher value of the residual sum of squares.

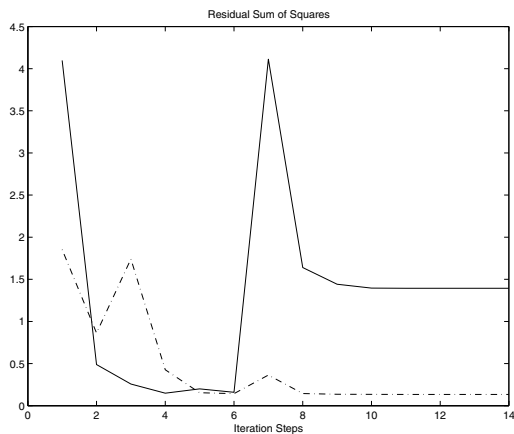


FIGURE 6.10. Residual sum of squares

There is another derivation of Newton's method which gives a different interpretation of the algorithm. Consider approximating Φ at a point \mathbf{x}_k near the minimum with a quadratic form $Q(\mathbf{h})$ with $\mathbf{h} = \mathbf{x} - \mathbf{x}_k$ by expanding in a Taylor series :

$$\Phi(\mathbf{x}) = \frac{1}{2} \|\mathbf{f}(\mathbf{x})\|_2^2 \approx \Phi(\mathbf{x}_k) + \nabla \Phi(\mathbf{x}_k)^\top \mathbf{h} + \frac{1}{2} \mathbf{h}^\top \nabla^2 \Phi(\mathbf{x}_k) \mathbf{h} =: Q(\mathbf{h}).$$

Then instead of minimizing Φ we minimize the quadratic form Q :

$$\nabla Q(\mathbf{h}) = 0 \iff \nabla^2 \Phi(\mathbf{x}_k) \mathbf{h} + \nabla \Phi(\mathbf{x}_k) = 0.$$

But this is again Newton's method. Thus, we have shown that applying Newton's method to the (nonlinear) equations $\nabla \Phi(\mathbf{x}) = 0$ is equivalent to approximating Φ locally by a quadratic form Q and computing its minimum.

6.8.3 Gauss-Newton Method

The approximation of Φ by a quadratic form,

$$\Phi(\mathbf{x}_k + \mathbf{h}) \approx \Phi(\mathbf{x}_k) + \nabla\Phi(\mathbf{x}_k)^\top \mathbf{h} + \frac{1}{2} \mathbf{h}^\top \nabla^2 \Phi(\mathbf{x}_k) \mathbf{h},$$

can be written in terms of \mathbf{f} ,

$$\Phi(\mathbf{x}_k + \mathbf{h}) \approx \frac{1}{2} \mathbf{f}^\top \mathbf{f} + \mathbf{f}^\top J\mathbf{h} + \frac{1}{2} \mathbf{h}^\top J^\top J\mathbf{h} + \frac{1}{2} \mathbf{h}^\top \sum_{l=1}^m f_l(\mathbf{x}_k) \nabla^2 f_l(\mathbf{x}_k) \mathbf{h}.$$

Rearranging yields

$$\Phi(\mathbf{x}_k + \mathbf{h}) \approx \frac{1}{2} \|J(\mathbf{x}_k)\mathbf{h} + \mathbf{f}(\mathbf{x}_k)\|_2^2 + \frac{1}{2} \mathbf{h}^\top \sum_{l=1}^m f_l(\mathbf{x}_k) \nabla^2 f_l(\mathbf{x}_k) \mathbf{h}. \quad (6.92)$$

The quadratic form approximating Φ consists therefore of two parts: the first involves only the Jacobian and the second the Hessians of the functions f_l . If we approximate Φ only by the first part $\Phi(\mathbf{x}) \approx \frac{1}{2} \|J(\mathbf{x}_k)\mathbf{h} + \mathbf{f}(\mathbf{x}_k)\|_2^2$, then the minimum of this quadratic form can be obtained by solving a *linear least squares problem*, namely

$$J(\mathbf{x}_k)\mathbf{h} + \mathbf{f}(\mathbf{x}_k) \approx 0. \quad (6.93)$$

Computing the correction \mathbf{h} by solving (6.93) is one step of the *Gauss-Newton method*. Another derivation of the Gauss-Newton method is obtained by linearizing $\mathbf{f}(\mathbf{x})$,

$$\mathbf{f}(\mathbf{x}) \approx \mathbf{f}(\mathbf{x}_k) + J(\mathbf{x}_k)\mathbf{h} \approx 0.$$

Summarizing, we obtain the algorithm *Gauss-Newton* for solving $\mathbf{f}(\mathbf{x}) \approx 0$:

1. Compute the Jacobian of \mathbf{f} and solve the linear least squares problem for \mathbf{h}

$$J(\mathbf{x}_k)\mathbf{h} \approx -\mathbf{f}(\mathbf{x}_k).$$

2. Iterate $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{h}$.

Convergence is in general quadratic for Newton's method and only linear for Gauss-Newton. However, as we saw in Example 6.24, often the global behavior of the method is more important than local quadratic convergence. We will therefore consider in the next section a method that has been devised to prevent Newton's method from taking excessively large steps. The goal is to avoid convergence situations as shown in Example 6.24 where the residual sum of squares increases after step six.

6.8.4 Levenberg-Marquardt Algorithm

DEFINITION 6.9. (DESCENT DIRECTION) *The vector \mathbf{v} is called a descent direction of $\Phi(\mathbf{x})$ at the point \mathbf{x} if*

$$\nabla\Phi(\mathbf{x})^\top \mathbf{v} < 0.$$

Let us explain this definition for $n = 2$: the gradient points into the direction of largest increase of $\Phi(\mathbf{x})$. If the scalar product of the gradient with a direction vector \mathbf{v} is negative, then the angle α between the two vectors must be larger than 90° . Thus, \mathbf{v} must point downhill, as shown in Figure 6.11. An algebraic explanation is based on the Taylor expansion,

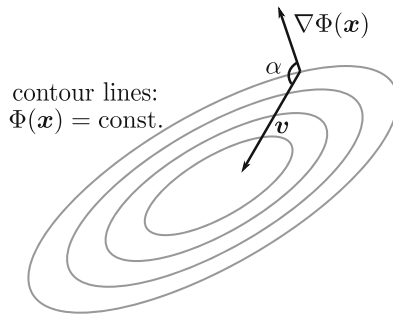


FIGURE 6.11. Illustration of a descent direction \mathbf{v}

$$\Phi(\mathbf{x} + \lambda \mathbf{v}) = \Phi(\mathbf{x}) + \underbrace{\nabla\Phi(\mathbf{x})^\top \mathbf{v}}_{<0} \lambda + O(\lambda^2),$$

where we assumed that $\|\mathbf{v}\|_2 = 1$. For sufficiently small $\lambda > 0$, we have $\Phi(\mathbf{x} + \lambda \mathbf{v}) < \Phi(\mathbf{x})$.

Let us now discuss whether the Newton and the Gauss-Newton corrections use descent directions:

1. Newton correction: $\nabla^2\Phi(\mathbf{x}_k)\mathbf{h} = -\nabla\Phi(\mathbf{x}_k)$. If $\nabla^2\Phi(\mathbf{x}_k)$ is nonsingular, i.e. the linear system has a solution, then

$$\nabla\Phi(\mathbf{x}_k)^\top \mathbf{h} = -\nabla\Phi(\mathbf{x}_k)^\top (\nabla^2\Phi(\mathbf{x}_k))^{-1} \nabla\Phi(\mathbf{x}_k).$$

The matrix $\nabla^2\Phi$ is symmetric. If in addition it is positive definite, then the Newton correction is a descent direction.

Now the matrix $\nabla^2\Phi$ must be positive semidefinite in a neighborhood of a local minimum of Φ , as one can see as follows: if \mathbf{x}^* is a local minimum, then we must have

- (a) $\Phi(\mathbf{x}^*) \leq \Phi(\mathbf{x}^* + \mathbf{h})$ for all $\mathbf{h} \neq 0$ in a small neighborhood of \mathbf{x}^* .

$$(b) \nabla \Phi(\mathbf{x}^*) = 0.$$

The Taylor expansion at \mathbf{x}^* gives

$$\Phi(\mathbf{x}^* + \mathbf{h}) = \Phi(\mathbf{x}^*) + \underbrace{\nabla \Phi(\mathbf{x}^*)^\top}_{=0} \mathbf{h} + \frac{1}{2} \mathbf{h}^\top \nabla^2 \Phi(\mathbf{x}^*) \mathbf{h} + O(\|\mathbf{h}\|_2^3).$$

Because $\Phi(\mathbf{x}^* + \mathbf{h}) \geq \Phi(\mathbf{x}^*)$, it follows that $\mathbf{h}^\top \nabla^2 \Phi(\mathbf{x}^*) \mathbf{h} \geq 0$, which means $\nabla^2 \Phi(\mathbf{x}^*)$ must be positive semidefinite in a neighborhood of \mathbf{x}^* . Thus we have obtained

THEOREM 6.21. *If $\nabla^2 \Phi(\mathbf{x}_k)$ is nonsingular for \mathbf{x}_k in a neighborhood of a local minimum \mathbf{x}^* then the Newton correction is a descent direction.*

2. Gauss-Newton correction: $J(\mathbf{x}_k)^\top J(\mathbf{x}_k) \mathbf{h} = -J(\mathbf{x}_k)^\top \mathbf{f}(\mathbf{x}_k) = -\nabla \Phi(\mathbf{x}_k)$. If $J(\mathbf{x}_k)^\top J(\mathbf{x}_k)$ is nonsingular, then this matrix and its inverse are positive definite and therefore

$$\nabla \Phi(\mathbf{x}_k)^\top \mathbf{h} = -\nabla \Phi(\mathbf{x}_k)^\top (J^\top J)^{-1} \nabla \Phi(\mathbf{x}_k) \quad (6.94)$$

$$= \begin{cases} < 0 & \text{if } \nabla \Phi(\mathbf{x}_k) \neq 0 \\ = 0 & \text{if } \nabla \Phi(\mathbf{x}_k) = 0 \end{cases} \quad (6.95)$$

If $\nabla \Phi(\mathbf{x}_k) = 0$, then we have reached a solution. To summarize, we have the result:

THEOREM 6.22. *If $J(\mathbf{x}_k)$ is not rank deficient in a neighborhood of a local minimum \mathbf{x}^* , then the Gauss-Newton correction is a descent direction.*

Locally the best descent direction is of course the negative gradient,

$$\mathbf{h} = -\nabla \Phi(\mathbf{x}_k) = -J(\mathbf{x}_k)^\top \mathbf{f}(\mathbf{x}_k).$$

However, often the locally optimal minimizing direction may not be the best for global optimization. The principal idea of Levenberg-Marquardt is a compromise between the negative gradient and the Gauss-Newton correction,

$$(J(\mathbf{x}_k)^\top J(\mathbf{x}_k) + \lambda I) \mathbf{h} = -J(\mathbf{x}_k)^\top \mathbf{f}(\mathbf{x}_k). \quad (6.96)$$

In (6.96), we have to choose a parameter λ . For $\lambda = 0$, we obtain the Gauss-Newton correction, whereas for $\lambda \gg 0$, the correction is along the direction of the negative gradient. There are several interpretations of the Levenberg-Marquardt correction:

- *Tikhonov-regularization:* If the matrix $J(\mathbf{x}_k)^\top J(\mathbf{x}_k)$ is singular, then for $\lambda > 0$ the matrix $J(\mathbf{x}_k)^\top J(\mathbf{x}_k) + \lambda I$ is again invertible and the correction can be computed.

- *Approximation of the Hessian:* We can regard λI as an approximation of

$$\sum_{l=1}^m f_l(\mathbf{x}_k) \nabla^2 f_l(\mathbf{x}_k).$$

Depending on the application on hand, one may choose to use a matrix other than λI , such as λD with D a diagonal matrix.

- *Limiting the norm of \mathbf{h} :* Consider the constrained minimization problem

$$\|J\mathbf{h} + \mathbf{f}\|_2^2 \longrightarrow \min \quad \text{subject to} \quad \|\mathbf{h}\|_2^2 \leq \alpha^2. \quad (6.97)$$

The unconstrained problem has a unique minimizer \mathbf{h}_0 given by the Gauss-Newton correction. We now have two cases:

1. if $\|\mathbf{h}_0\|_2 \leq \alpha$, then \mathbf{h}_0 also solves the constrained problem.
2. if $\|\mathbf{h}_0\|_2 > \alpha$, then the solution of the constrained problem must lie on the boundary, since there are no local minima inside the disk $\|\mathbf{h}\|_2 \leq \alpha$.

In the second case, the constrained problem can be solved using Lagrange multipliers as follows: consider the Lagrangian

$$L(\mathbf{h}, \lambda) = \frac{1}{2} \|J\mathbf{h} + \mathbf{f}\|_2^2 + \lambda (\|\mathbf{h}\|_2^2 - \alpha^2).$$

Setting the partial derivatives to zero, we obtain the equations

$$(J(\mathbf{x}_k)^\top J(\mathbf{x}_k) + \lambda I) \mathbf{h} = -J(\mathbf{x}_k)^\top \mathbf{f} \quad (6.98)$$

$$\|\mathbf{h}\|_2^2 = \alpha^2. \quad (6.99)$$

Solving (6.98) gives $\mathbf{h} = -R(\lambda) \mathbf{f}$, where

$$R(\lambda) = (J(\mathbf{x}_k)^\top J(\mathbf{x}_k) + \lambda I)^{-1} J(\mathbf{x}_k)^\top.$$

To calculate the value of α corresponding to this solution, let $J(\mathbf{x}_k) = U \Sigma V^\top$ be the SVD of $J(\mathbf{x}_k)$. Then

$$R(\lambda) = V(\Sigma^\top \Sigma + \lambda I)^{-1} \Sigma^\top U^\top,$$

so $R(\lambda)$ has singular values

$$\mu_i = \frac{\sigma_i(J)}{\sigma_i(J)^2 + \lambda},$$

and

$$\alpha^2 = \|\mathbf{h}\|_2^2 = \sum_{i=1}^n \mu_i^2 \tilde{f}_i^2,$$

where $\tilde{\mathbf{f}} = U^\top \mathbf{f}$. Thus, we see that $\alpha = \alpha(\lambda)$ is a strictly decreasing function of λ , with $\alpha(0) = \|\mathbf{h}_0\|_2$ and $\lim_{\lambda \rightarrow \infty} \alpha(\lambda) = 0$. This means for every $\lambda > 0$, there is an $\alpha(\lambda) < \|\mathbf{h}_0\|_2$ such that the Levenberg-Marquardt step \mathbf{h} solves the constrained minimization problem (6.97) with $\alpha = \alpha(\lambda)$. In other words, the Levenberg-Marquardt method solves the minimization problem over a reduced set of admissible solutions, i.e., those that satisfy $\|\mathbf{h}\|_2 \leq \alpha(\lambda)$, effectively limiting the correction step to within a region near \mathbf{x}_k ; this is known as a *trust region* in optimization, see Subsection 12.3.2.

Choosing a good λ is not easy. Besides trial and error, there are various propositions in the literature, such as the one by Brown and Dennis:

$$\lambda = c \|\mathbf{f}(\mathbf{x}_k)\|_2, \quad c = \begin{cases} 10 & \text{for } 10 \leq \|\mathbf{f}(\mathbf{x}_k)\|_2 \\ 1 & \text{for } 1 \leq \|\mathbf{f}(\mathbf{x}_k)\|_2 < 10 \\ 0.01 & \text{for } \|\mathbf{f}(\mathbf{x}_k)\|_2 < 1 \end{cases}$$

The computation of the Levenberg-Marquardt correction (6.96) is equivalent to solving the linear least squares problem

$$\begin{pmatrix} J \\ \sqrt{\lambda} I \end{pmatrix} \mathbf{h} \approx \begin{pmatrix} -\mathbf{f} \\ 0 \end{pmatrix}, \quad (6.100)$$

which is the numerically preferred way.

In MATLAB there exists the function `nlinfit` for nonlinear least squares data fitting by the Gauss-Newton method (in newer releases like R2011a it is in the Statistics Toolbox). The Jacobian is approximated by a finite difference and augmented by $0.01 I$ (i.e. $\lambda = 10^{-4}$). Thus, the function in fact uses Levenberg-Marquardt-adjusted Gauss-Newton steps, which are computed using (6.100). The step is only accepted if the new residual norm is smaller than the previous one. Otherwise, the step is reduced by $\mathbf{h} = \mathbf{h}/\sqrt{10}$ until the condition is met. The function `nlinfit` therefore does not get trapped for Example 6.24 in the local minimum and is able also to deliver the correct answer for the starting values [1.8, 1.8, 0.1]. With the function

```
>> f=@(beta,x) beta(1)+beta(2)*exp(-beta(3)*x);
```

we obtain the same parameters as before with the Newton method:

```
>> beta=nlinfit(xi,eta,f,[1.8 1.8 0.1])
```

```
beta =
```

```
1.148055903080751    1.862263964120192    0.170154093666413
```

6.9 Least Squares Fit with Piecewise Functions

We end the chapter with a good example of a constrained least squares problem with nonlinear constraints, which is solved in [45]. Consider the data set given in Table 6.3 and displayed in Figure 6.12. The problem is to find piecewise functions *with free knots* that best fit the data set. The choice of the

knots is in fact part of the optimization problem, since the location of these knots may have a physical interpretation, such as phase changes. The constraint is that global function should be continuous in the knots that separate them; optionally, we may also require that the derivative be continuous.

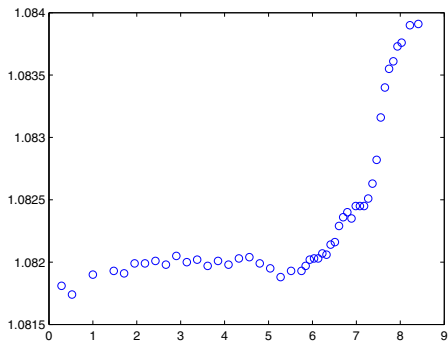


FIGURE 6.12. Data Set

TABLE 6.3. Given Data Set

| x | y | x | y | x | y |
|----------|---------|----------|---------|----------|---------|
| 0.288175 | 1.08181 | 4.562725 | 1.08204 | 6.794990 | 1.08240 |
| 0.525650 | 1.08174 | 4.800200 | 1.08199 | 6.889980 | 1.08235 |
| 1.000600 | 1.08190 | 5.037675 | 1.08195 | 6.984970 | 1.08245 |
| 1.475550 | 1.08193 | 5.275150 | 1.08188 | 7.079960 | 1.08245 |
| 1.713025 | 1.08191 | 5.512625 | 1.08193 | 7.174950 | 1.08245 |
| 1.950500 | 1.08199 | 5.750100 | 1.08193 | 7.269940 | 1.08251 |
| 2.187975 | 1.08199 | 5.845090 | 1.08197 | 7.364930 | 1.08263 |
| 2.425450 | 1.08201 | 5.940080 | 1.08202 | 7.459920 | 1.08282 |
| 2.662925 | 1.08198 | 6.035070 | 1.08203 | 7.554910 | 1.08316 |
| 2.900400 | 1.08205 | 6.130060 | 1.08203 | 7.649900 | 1.08340 |
| 3.137875 | 1.08200 | 6.225050 | 1.08207 | 7.744890 | 1.08355 |
| 3.375350 | 1.08202 | 6.320040 | 1.08206 | 7.839880 | 1.08361 |
| 3.612825 | 1.08197 | 6.415030 | 1.08214 | 7.934870 | 1.08373 |
| 3.850300 | 1.08201 | 6.510020 | 1.08216 | 8.029860 | 1.08376 |
| 4.087775 | 1.08198 | 6.605010 | 1.08229 | 8.219840 | 1.08390 |
| 4.325250 | 1.08203 | 6.700000 | 1.08236 | 8.409820 | 1.08391 |

Since the nature of the piecewise functions is not known, we will use polynomials of low degrees. The special case where all the polynomials have the same degree corresponds to a well-known problem called *least squares approximation by splines with free knots*. Many authors have worked on this problem; a good survey is given for example in [123].

Assume we are given some data points

$$\begin{array}{c|cccc} x & x_1 & x_2 & \dots & x_N \\ \hline y & y_1 & y_2 & \dots & y_N \end{array}. \quad (6.101)$$

where the x -coordinates are ordered: $a = x_1 < x_2 < \dots < x_N = b$. We want to partition this set by two knots ξ and η such that

$$x_1 < \dots < x_{n_1} < \xi \leq x_{n_1+1} < \dots < x_{n_2} < \eta \leq x_{n_2+1} < \dots < x_N.$$

These two knots define three intervals: (x_1, ξ) , (ξ, η) and (η, x_N) . We will fit in the least squares sense three given functions f , g and h in each interval to the data points,

$$\begin{aligned} f(\mathbf{a}, x_i) &\approx y_i^f & i &= 1, \dots, n_1, \\ g(\mathbf{b}, x_i) &\approx y_i^g & i &= n_1 + 1, \dots, n_2, \\ h(\mathbf{c}, x_i) &\approx y_i^h & i &= n_2 + 1, \dots, N. \end{aligned} \quad (6.102)$$

The superscript in the y -coordinates indicates that we have also partitioned the data points accordingly,

$$\mathbf{x} = \begin{pmatrix} \mathbf{x}^f \\ \mathbf{x}^g \\ \mathbf{x}^h \end{pmatrix}, \quad \mathbf{y} = \begin{pmatrix} \mathbf{y}^f \\ \mathbf{y}^g \\ \mathbf{y}^h \end{pmatrix}.$$

The parameters of the functions f , g and h to be determined by the least squares fit are denoted by \mathbf{a} , \mathbf{b} and \mathbf{c} .

To enforce continuity of the global piecewise function, we have to impose the constraints

$$\begin{aligned} f(\mathbf{a}, \xi) - g(\mathbf{b}, \xi) &= 0, \\ g(\mathbf{b}, \eta) - h(\mathbf{c}, \eta) &= 0. \end{aligned} \quad (6.103)$$

If the first derivative is also required to be continuous at the knots, then the additional constraints

$$\begin{aligned} f'(\mathbf{a}, \xi) - g'(\mathbf{b}, \xi) &= 0, \\ g'(\mathbf{b}, \eta) - h'(\mathbf{c}, \eta) &= 0. \end{aligned} \quad (6.104)$$

must also be considered.

Let us introduce the vector functions

$$\mathbf{f}(\mathbf{a}) = \begin{pmatrix} f(\mathbf{a}, x_1) \\ \vdots \\ f(\mathbf{a}, x_{n_1}) \end{pmatrix}, \quad \mathbf{g}(\mathbf{b}) = \begin{pmatrix} g(\mathbf{b}, x_{n_1+1}) \\ \vdots \\ g(\mathbf{b}, x_{n_2}) \end{pmatrix}, \quad \mathbf{h}(\mathbf{c}) = \begin{pmatrix} h(\mathbf{c}, x_{n_2+1}) \\ \vdots \\ h(\mathbf{c}, x_N) \end{pmatrix},$$

and the vector of unknowns

$$\mathbf{z} = \begin{pmatrix} \mathbf{a} \\ \mathbf{b} \\ \mathbf{c} \\ \xi \\ \eta \end{pmatrix}, \quad \mathbf{F}(\mathbf{z}) = \begin{pmatrix} \mathbf{f}(\mathbf{a}) \\ \mathbf{g}(\mathbf{b}) \\ \mathbf{h}(\mathbf{c}) \end{pmatrix} \quad \text{and} \quad \mathbf{G}(\mathbf{z}) = \begin{pmatrix} f(\mathbf{a}, \xi) - g(\mathbf{b}, \xi) \\ g(\mathbf{b}, \eta) - h(\mathbf{c}, \eta) \end{pmatrix}.$$

Then the problem of fitting a continuous global function becomes a *constrained nonlinear least squares problem*

$$\min_{\mathbf{z}} \|\mathbf{F}(\mathbf{z}) - \mathbf{y}\|_2 \quad \text{subject to} \quad \mathbf{G}(\mathbf{z}) = \mathbf{0}. \quad (6.105)$$

If we also require continuity of the derivative, then we have to replace \mathbf{G} in (6.105) by

$$\mathbf{G}(\mathbf{z}) = \begin{pmatrix} f(\mathbf{a}, \xi) - g(\mathbf{b}, \xi) \\ g(\mathbf{b}, \eta) - h(\mathbf{c}, \eta) \\ f'(\mathbf{a}, \xi) - g'(\mathbf{b}, \xi) \\ g'(\mathbf{b}, \eta) - h'(\mathbf{c}, \eta) \end{pmatrix}.$$

We will solve Problem (6.105) by an alternating minimization procedure: in each iteration step, we will use the current values of ξ and η to allocate the points to the functions \mathbf{f} , \mathbf{g} and \mathbf{h} . Then we will apply the *Gauss-Newton method* to solve the nonlinear least squares problem. For this, we linearize the functions \mathbf{F} and \mathbf{G} to obtain a linear least squares problem with linear constraints, from which we calculate the correction $\Delta \mathbf{z}$. Assume $\bar{\mathbf{z}}$ is an approximation of a solution of Problem (6.105). If we expand

$$\mathbf{F}(\bar{\mathbf{z}} + \Delta \mathbf{z}) \approx \mathbf{F}(\bar{\mathbf{z}}) + J_F \Delta \mathbf{z},$$

and do the same for $\mathbf{G}(\mathbf{z})$, we can replace Problem (6.105) by a constrained linear least squares problem with the Jacobian matrices for the correction

$$\begin{aligned} J_F \Delta \mathbf{z} &\approx \mathbf{y} - \mathbf{F}(\bar{\mathbf{z}}), \\ J_G \Delta \mathbf{z} &= -\mathbf{G}(\bar{\mathbf{z}}). \end{aligned} \quad (6.106)$$

We will use the `LinearlyConstrainedLSQ` Algorithm 6.21 for the solution.

6.9.1 Structure of the Linearized Problem

The linearly constrained least squares problem (6.106) is structured: the Jacobian matrix of \mathbf{F} is block diagonal and contains the Jacobin matrices of the three functions f , g and h ,

$$J_F = \begin{bmatrix} J_f & \mathbf{0} & \mathbf{0} & 0 & 0 \\ \mathbf{0} & J_g & \mathbf{0} & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & J_h & 0 & 0 \end{bmatrix}.$$

Since in our case the size of the matrix J_F is small, we will not treat it as a sparse matrix. The following function `DirectSum` comes in handy to construct such block-diagonal matrices :

ALGORITHM 6.32. *Generating Direct Sums*

```
function A=DirectSum(A,varargin)
```

```
% DIRECTSUM computes the direct sum of matrices
%   A=DirectSum(A1,A2,...,An) computes the direct sum of matrices of
%   arbitrary size (Peter Arbenz, May 30, 1997)

for k=1:length(varargin)
    [n,m]=size(A);
    [o,p]=size(varargin{k});
    A=[A zeros(n,p); zeros(o,m) varargin{k}];
end
```

For the continuity condition, the Jacobian of \mathbf{G} is

$$J_G = \begin{bmatrix} \nabla f(\mathbf{a}, \xi)^\top & -\nabla g(\mathbf{b}, \xi)^\top & 0 & f'(\mathbf{a}, \xi) - g'(\mathbf{b}, \xi) & 0 \\ 0 & \nabla g(\mathbf{b}, \eta)^\top & -\nabla h(\mathbf{c}, \eta)^\top & 0 & g'(\mathbf{b}, \eta) - h'(\mathbf{c}, \eta) \end{bmatrix}.$$

Note that we denote with $\nabla f(\mathbf{a}, \xi)$ the gradient of f with respect to the parameters \mathbf{a} , while f' denotes the derivative of f with respect to the independent variable x .

If also the derivatives should be continuous then

$$J_G = \begin{bmatrix} \nabla f(\mathbf{a}, \xi)^\top & -\nabla g(\mathbf{b}, \xi)^\top & 0 & f'(\mathbf{a}, \xi) - g'(\mathbf{b}, \xi) & 0 \\ 0 & \nabla g(\mathbf{b}, \eta)^\top & -\nabla h(\mathbf{c}, \eta)^\top & 0 & g'(\mathbf{b}, \eta) - h'(\mathbf{c}, \eta) \\ \nabla f'(\mathbf{a}, \xi)^\top & -\nabla g'(\mathbf{b}, \xi)^\top & 0 & f''(\mathbf{a}, \xi) - g''(\mathbf{b}, \xi) & 0 \\ 0 & \nabla g'(\mathbf{b}, \eta)^\top & -\nabla h'(\mathbf{c}, \eta)^\top & 0 & g''(\mathbf{b}, \eta) - h''(\mathbf{c}, \eta) \end{bmatrix}. \quad (6.107)$$

6.9.2 Piecewise Polynomials

For polynomials, the Jacobians become Vandermonde matrices if we use the standard representation. If $f(\mathbf{a}, x) = a_1 x^p + a_2 x^{p-1} + \dots + a_p x + a_{p+1}$, then

$$J_f = \begin{pmatrix} x_1^p & x_1^{p-1} & \dots & x_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{n_1}^p & x_{n_1}^{p-1} & \dots & x_{n_1} & 1 \end{pmatrix}.$$

If f , g and h are polynomials of degree p , q and r , then for the continuity of the global function, we need

$$J_G = \begin{pmatrix} \xi^p & \dots & \xi & 1 & -\xi^q & \dots & -\xi & -1 & 0 & \dots & 0 & 0 & f'(\mathbf{a}, \xi) - g'(\mathbf{b}, \xi) & 0 \\ 0 & \dots & 0 & 0 & \eta^q & \dots & \eta & 1 & -\eta^r & \dots & -\eta & -1 & 0 & g'(\mathbf{b}, \eta) - h'(\mathbf{c}, \eta) \end{pmatrix}.$$

For the continuity of the derivative we have

$$f'(\mathbf{a}, \xi) = p a_1 \xi^{p-1} + (p-1) a_2 \xi^{p-2} + \dots + a_p,$$

and thus we use in (6.107) the expression

$$\nabla f'(\mathbf{a}, \xi)^\top = [p \xi^{p-1}, (p-1) \xi^{p-2}, \dots, 2\xi, 1, 0].$$

For the following main program we need to first read the data, stored in a file `xy.m`. This file contains only the definition of the matrix `XY`:

```
XY = [0.288175    1.08181
      0.525650    1.08174
      ...         ...
      8.219840    1.08390
      8.409820    1.08391]
```

For the sake of brevity, we did not print all the data of Table 6.3, but we will print intermediate results and also plot the approximations. To plot the given points, we will use the function `PlotPoints`:

ALGORITHM 6.33. *Plot the Points*

```
function ax=PlotPoints(X)
% PLOTPPOINTS plots the points X and returns the axis handle

clf; hold off;
plot(X(:,1),X(:,2),'o');
ax=axis; hold on;
```

During the iterations, the breakpoints ξ and η will change, so we need to repartition the data after each iteration. This is done with the function `Partition`:

ALGORITHM 6.34. *Partition of the Data*

```
function [n1,n2,xf,yf,xg,yg,xh,yh]=Partition(xi,eta,XY,N)
% PARTITION Partitions the data XY into three data sets
% [n1,n2,xf,yf,xg,yg,xh,yh]=Partition(xi,eta,XY,N) partition the
% date set in XY into three subsets according to xi and eta.

n1=sum(XY(:,1)<xi); n2=sum(XY(:,1)<eta);
xf=XY(1:n1,1); yf=XY(1:n1,2);
xg=XY(n1+1:n2,1); yg=XY(n1+1:n2,2);
xh=XY(n2+1:N,1); yh=XY(n2+1:N,2);
```

The Jacobian matrices are Vandermonde matrices and are generated using the function `van`:

ALGORITHM 6.35. *Generate Jacobian*

```
function J=van(p,x)
% VAN construct a Vandermonde matrix
% J=van(p,x) computes p+1 columns of the Vandermonde matrix of x

n=length(x); J=ones(n,1);
for j=1:p
    J=[x.^j J];
end
```

After each iteration, we will pause and plot the current approximation using the function `PlotFunctions`:

ALGORITHM 6.36. *Plot Functions*

```
function PlotFunctions(xi,eta,a,b,c,ax);
% PLOTFUNCTIONS plots the three polynomials

xx=[0:0.1:xi]; plot(xx,polyval(a,xx),'r'), axis(ax)
xx=[xi:0.1:eta]; plot(xx,polyval(b,xx),'g'), axis(ax)
xx=[eta:0.1:9]; plot(xx,polyval(c,xx),'b'), axis(ax)
```

The initial approximations for coefficients of the polynomials are computed using the MATLAB function `polyfit` to fit each polynomial to the partitioned data sets in the least squares sense. For the evaluation of a polynomial, we use the MATLAB function `polyval`. Finally, to compute the coefficients of the derivative of a polynomial we use `dpoly`:

ALGORITHM 6.37. *Derivative of a Polynomial*

```
function DA=dpoly(n,A)
% DPOLY derivative of a polynomial
% DA=dpoly(n,A) computes the coefficients DA of the derivative of
% the polynomial of degree n given by the coefficients A.

DA=0;
for j=1:n, DA(j)=(n+1-j)*A(j); end
```

The following program `main1` computes the Gauss-Newton approximations. Of course the iterations may or may not converge. Even when the method converges, we cannot guarantee that the limit point is the global minimum, since the problem has many local minima.

The degrees of the polynomials can be chosen by changing the statements for `p`, `q` and `r`. The variable `derivative` is used as a switch to decide whether the derivative should also be continuous at the break points (`derivative=1`) or not.

The iteration is stopped if no convergence has been reached after 40 iterations. It is also stopped if the break points switch their order, i.e. if $\eta < \xi$.

ALGORITHM 6.38. *Fitting Piecewise Polynomials*

```
% Piecewise Polynomial Fit: main1.m
xy; N=max(size(XY));           % read data
ax=PlotPoints(XY)              % plot given points
```

```

p=3; q=3; r=3; % choose degrees and
derivatives=1 % derivatives (1=continuous, 0=no)
xi0=6.4; eta0=8; % initialize break points
% xi < eta
xi=xi0; eta=eta0; % first partition of data
[n1,n2,xf,yf,xg,yg,xh,yh]=Partition(xi,eta,XY,N);
Jf=van(p,xf); Jg=van(q,xg); Jh=van(r,xh);
% initialize polynomial coefficients by individual least
% squares fit
a=polyfit(xf, yf,p); b=polyfit(xg,yg,q); c=polyfit(xh,yh,r);
% plot initial approximation
% functions
PlotFunctions(xi,eta,a,b,c, ax);
k=0; % count iterations
dp=1; % initialize correction
while (norm(dp)>1e-5) & (k<40),
    k= k+1;
    JF=[DirectSum(Jf,Jg,Jh), zeros(N,2)];
    gradfxi=van(p,xi); gradgxi=van(q,xi);
    gradgeta=van(q,eta); gradheta=van(r,eta);
    % coeffs of derivatives
    da=dpoly(p,a); db=dpoly(q,b); dc=dpoly(r,c);
    % constraints for continuity
    JG=[gradfxi -gradgxi zeros(size(gradheta)) ...
        polyval(da,xi)-polyval(db,xi) 0
        zeros(size(gradfxi)) gradgeta -gradheta ...
        0 polyval(db,eta)-polyval(dc,eta)]
    if derivatives, % continuity of derivative
        gradfsxi =[[p:-1:1].*van(p-1,xi),0];
        gradgsxi=[[q:-1:1].*van(q-1,xi),0];
        gradgseta=[[q:-1:1].*van(q-1,eta),0];
        gradhseta=[[r:-1:1].*van(r-1,eta),0];

        dda=dpoly(p-1,da); % coeffs of second derivative
        ddb=dpoly(q-1,db);
        ddc=dpoly(r-1,dc);
        JG=[JG
            gradfsxi -gradgsxi zeros(size(gradhseta))...
            polyval(dda,xi)-polyval(ddb,xi) 0
            zeros(size(gradfsxi)) gradgseta -gradhseta ...
            0 polyval(ddb,eta)-polyval(ddc,eta)]
    end
    % Right hand side for lsq
    z=[yf-polyval(a,xf);yg-polyval(b,xg);yh-polyval(c,xh)]
    % Right hand side for constraints
    mG=-[polyval(a,xi)-polyval(b,xi)
        polyval(b,eta)-polyval(c,eta)]
    if derivatives, % add constraints

```

```

        mG=-[-mG; polyval(da,xi)-polyval(db,xi)
              polyval(db,eta)-polyval(dc,eta)]
    end
                                % solve for corrections
    dp=LinearlyConstrainedLSQ(JF,JG,z,mG);
    a=a+dp(1:p+1)';              % update unknowns
    b=b+dp(p+2:p+q+2)'; c=c+dp(p+q+3:p+q+r+3)';
    xi=xi+dp(p+q+r+4); eta=eta+dp(p+q+r+5);
    if xi>eta, error('xi > eta'); end
    norm(dp)                      % print norm of correction
                                % plot current approximation
    PlotFunctions(xi,eta,a,b,c, ax);
    pause
                                % new partition of the data
    [n1,n2,xf,yf,xg,yg,xh,yh]=Partition(xi,eta,XY,N);
    Jf=van(p,xf); Jg=van(q,xg); Jh=van(r,xh);
end
                                % plot final result
ax=PlotPoints(XY); PlotFunctions(xi,eta,a,b,c, ax);
                                % compute residual and print
                                % results
rf=norm(polyval(a,xf)-XY(1:n1,2));
rg=norm(polyval(b,xg)-XY(n1+1:n2,2));
rh=norm(polyval(c,xh)-XY(n2+1:N,2));
rr=norm([rf, rg, rh])           % residuals
dpnorm=norm(dp)                 % norm of last correction
disp('degrees of polynomials, derivative (yes=1), # of iterations')
[p q r derivatives k]
disp('breakpoints: initial, final')
[xi0 eta0 xi eta]

```

6.9.3 Examples

Figure 6.13 shows an approximation by classical smoothing splines of degree 3 with free knots. Using $\xi = 7.34$ and $\eta = 7.78$ as initial approximations for the breakpoints, after 9 Gauss-Newton iterations the norm of the correction vector drops to $4.3180e-06$ and the break points converge to $\xi = 7.3157$ and $\eta = 7.7275$. The norm of the true residual for the three spline functions is $3.4965e-04$.

However, the solution to which we converged may be only a local minimum of the nonlinear least squares function. In fact, if we use other initial approximations for the break points, we may obtain no convergence or convergence to other solutions. Table 6.4 shows some results. There are three different solutions with almost the same norm of the residual ($3.4965e-04$, $3.6819e-04$ and $3.6499e-04$). This shows that the problem is ill-conditioned.

In the next example, we choose linear functions and ask only for continuity (see Figure 6.14). For the initial values $\xi = 7.34$ and $\eta = 7.78$ we obtain the

FIGURE 6.13.
*Classical Smoothing Splines
with Free Knots*

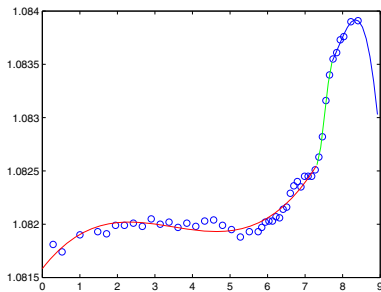


FIGURE 6.14.
*Piecewise Linear Functions
with Free Knots*

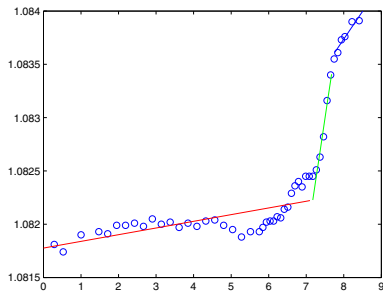


TABLE 6.4.
Different Solutions for Different Initial Breakpoints for Smoothing Splines

| initial appr. final breakp. | norm of corr. norm of res. | # of inter. |
|--------------------------------|--------------------------------------|-------------------|
| 7.34 | 7.78 | 4.3180e-06 |
| 7.31570 | 7.72746 | 3.4965e-04 |
| 6.5 | 8 | 6.3398e-06 |
| 7.31570 | 7.72746 | 3.4965e-04 |
| 6.4 | 8 | 5.3655e-06 |
| 6.03633 | 6.98520 | 3.6819e-04 |
| 5 | 7 | 7.4487e-06 |
| 5.53910 | 7.04320 | 3.6499e-04 |

break points $\xi = 7.1750$ and $\eta = 7.7623$.

Again by choosing different initial values for the break points, the Gauss-Newton method converges to different local minima (see Table 6.5). The ill conditioning here is even more pronounced.

Next we choose different degrees $p = 3$, $q = 1$ and $r = 2$ and fit a continuous global function, see Table 6.6. For the initial values $\xi = 7.34$ and $\eta = 7.78$ the break points become $\xi = 7.3552$ and $\eta = 7.6677$ and we obtain in 4 iterations Figure 6.15 with a residual norm of $3.5422e-04$. However, for the initial values $\xi = 6.5$ and $\eta = 8$ a better solution with break points 5.7587 and 7.3360 and a slightly reduced residual of $2.7396e-04$ is obtained, see Figure 6.16. This example shows again how difficult it is to converge to a global minimum.

Finally, we increase the degrees to $p = 5$, $q = 3$ and $r = 2$ and ask for the continuity of the derivative as well, see Figure 6.17. Now the break points are $\xi = 7.3717$ and $\eta = 7.6494$.

TABLE 6.5. Solutions for Piecewise Linear Functions

| initial appr. final breakp. | | norm of corr. norm of res. | # of inter. |
|--------------------------------|----------|--------------------------------------|-------------|
| 7.34 | 7.78 | 7.8568e-14 | 4 |
| 7.17495 | 7.76231 | 7.6797e-04 | |
| 3 | 6 | 2.4061e-13 | 6 |
| 1.97596 | 6.57421 | 8.1443e-04 | |
| 5 | 7 | 4.5775e-13 | 7 |
| 6.12802 | 7.169537 | 6.7463e-04 | |

FIGURE 6.15.
*Polynomials with degrees $p = 3$,
 $q = 1$ and $r = 2$*

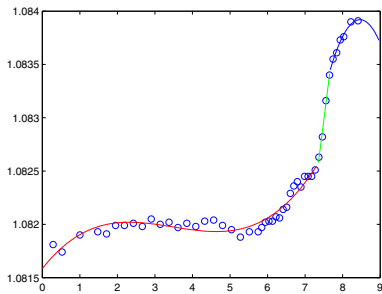
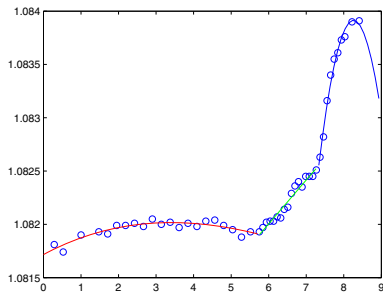


FIGURE 6.16.
*Polynomials with degrees $p = 3$,
 $q = 1$ and $r = 2$, second solution*



Again other starting values for the break points e.g. $\xi = 6$ and $\eta = 8$ lead to another solution, see Figure 6.18, with almost the same residual, see Table 6.7.

6.10 Problems

PROBLEM 6.1. A person brings two packages to the post. One weighs 5 kg, the other 2 kg. However, at the post office both weigh together 8 kg. Adjust their weights using the least squares method.

PROBLEM 6.2. Let the matrix $A \in \mathbb{R}^{m \times n}$ with $m > n$ and $\text{rank}(A) = n$. Prove that the matrix $A^\top A$ of the normal equations is positive definite.

PROBLEM 6.3. Let the matrix $A \in \mathbb{R}^{n \times n}$ be symmetric and positive definite. Prove that the singular values of A are the same as its eigenvalues. What is the relation between singular values and eigenvalues if A is symmetric but not positive definite?

PROBLEM 6.4. Let $A, B \in \mathbb{R}^{m \times n}$ and let $A = QBP^\top$ where Q and P are

TABLE 6.6. Solutions for $p = 3, q = 1$ and $r = 2$

| initial appr. final breakp. | | norm of corr. norm of res. | # of inter. |
|--------------------------------|--------|--------------------------------------|-------------|
| 7.34 | 7.78 | 3.6034e-06 | 4 |
| 7.3552 | 7.6677 | 3.5422e-04 | |
| 6.5 | 8 | 1.4420e-08 | 8 |
| 5.7587 | 7.3360 | 2.7396e-04 | |

FIGURE 6.17.
*Polynomials with degrees $p = 5,$
 $q = 3$ and $r = 2$*

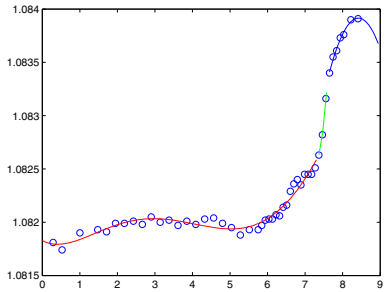
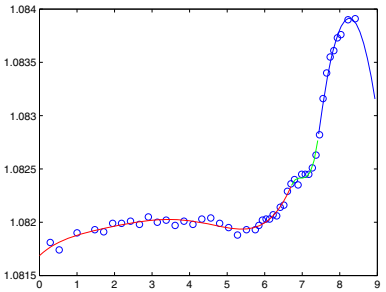


FIGURE 6.18.
*Polynomials with degrees $p = 5,$
 $q = 3$ and $r = 2$, second solution*



orthogonal matrices. Prove that

$$\|A\|_F = \|B\|_F.$$

PROBLEM 6.5. Let $Q \in \mathbb{R}^{m \times n}$ with $m > n$ be an orthogonal matrix. Consider the matrix $P = QQ^\top$. Is P an orthogonal projector? If yes where does it projects? Justify your answer.

TABLE 6.7. Solutions for $p = 5, q = 3$ and $r = 2$, (continuous derivatives)

| initial appr. final breakp. | | norm of corr. norm of res. | # of inter. |
|--------------------------------|--------|--------------------------------------|-------------|
| 7.34 | 7.78 | 5.1228e-09 | 12 |
| 7.3717 | 7.6494 | 2.8155e-04 | |
| 6 | 8 | 8.9962e-06 | 15 |
| 6.7139 | 7.4377 | 2.5334e-04 | |

PROBLEM 6.6. Consider the plane in \mathbb{R}^3 given by the equation

$$x_1 + x_2 + x_3 = 0.$$

Construct a matrix P which projects a given point on this plane. Hint: consider first the orthogonal complement of the plane.

PROBLEM 6.7. Given the matrix

$$A = \begin{pmatrix} 1 & 2 & 6 \\ 1 & 3 & 7 \\ 1 & 4 & 8 \\ 1 & 5 & 9 \end{pmatrix}$$

Compute a Householder matrix P such that

$$PA = \begin{pmatrix} \sigma & x & x \\ 0 & x & x \\ 0 & x & x \\ 0 & x & x \end{pmatrix}$$

It is sufficient to determine σ and the Householder-vector \mathbf{u} .

PROBLEM 6.8. Consider the plane in \mathbb{R}^3 given by the equation

$$2x_1 - 2x_3 = 0.$$

Construct a matrix $P \in \mathbb{R}^{3 \times 3}$ which reflects a given point at this plane (computes the mirror image).

PROBLEM 6.9. Let the measured points (t_k, y_k) for $i = k, \dots, m$ be given. We want to fit the function $f(a, b) = ae^{bt}$ such that

$$\sum_{k=1}^m (ae^{bt_k} - y_k)^2 \longrightarrow \min$$

using the Gauss-Newton method. Write up the system of equations for the first iteration step.

PROBLEM 6.10. Let $\mathbf{b} = (b_1, b_2, b_3, b_4)$ be the measured lengths of the sides of a rectangle. Correct the measurements using least squares and determine side lengths \mathbf{x} such that $\|\mathbf{x} - \mathbf{b}\|_2^2 \longrightarrow \min$ subject to the condition that two corresponding sides have the same length i.e. $x_1 = x_3$ and $x_2 = x_4$. Compute the corrected values.

PROBLEM 6.11. Let $A, B \in \mathbb{R}^{m \times n}$ and let $A = QBP^T$ where Q and P are orthogonal matrices. Prove that

$$\|A\|_2 = \|B\|_2.$$

PROBLEM 6.12. Consider the matrix

$$A = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

1. Compute and describe geometrically the 4 fundamental subspaces
2. Compute the 4 projectors on the fundamental subspaces.

PROBLEM 6.13. Consider the constrained least squares problem with given $A \in \mathbb{R}^{m \times n}$ and $\mathbf{b} \in \mathbb{R}^m$ and $m > n$:

$$\min \|\mathbf{b} - \mathbf{y}\|_2^2 \quad \text{subject to } \mathbf{y} = A\mathbf{x}.$$

1. Interpret the problem geometrically.
2. Assume A has full rank, give an explicit expression for the solution \mathbf{y} .
3. How can you compute \mathbf{y} using the function *ModifiedGramSchmidt* (Algorithm 6.9)?

PROBLEM 6.14. A student wants to update the QR decomposition by removing a column using the function *UpdateQR*. He has defined \mathbf{e}_k to be the k -th unit vector.

```
v=-A(:,k);
[Qs,Rs]=UpdateQR(Q,R,v,e_k)
```

What is he effectively computing?

PROBLEM 6.15. Minimizing the length of the residual vector $\mathbf{r} = \mathbf{b} - A\mathbf{x}$ is equivalent of minimizing the quadratic form

$$Q(\mathbf{x}) = \mathbf{r}^\top \mathbf{r} = (\mathbf{b} - A\mathbf{x})^\top (\mathbf{b} - A\mathbf{x}) = \mathbf{b}^\top \mathbf{b} - 2\mathbf{x}^\top A^\top \mathbf{b} + \mathbf{x}^\top A^\top A \mathbf{x}.$$

By differentiating with respect to \mathbf{x} and equating to zero show that the resulting equations are the normal equations.

PROBLEM 6.16. Savitzky-Golay Filter. Noisy data often have to be filtered. One way to do this is to compute a least squares fit of a polynomial $P(x)$ of degree d through $2N + 1$ points left and right of the middle point x_i and to replace the function value y_i by the smoothed value $P(x_i)$.

The smoothed value is an average of the neighboring points and thus the process is called a moving average (see Chapter 9 in [45]).

If the abscissas x_i are equidistant then the average is the same for all points and depends only on the degree of the polynomial and the number of points used for the average.

We consider therefore the data

$$\begin{array}{c|cccccccc} x & -N & \cdots & -1 & 0 & 1 & \cdots & N \\ \hline y & y_{-N} & \cdots & y_{-1} & y_0 & y_1 & \cdots & y_N \end{array}$$

We want to fit a polynomial

$$P(x) = b_d x^d + b_{d-1} x^{d-1} + \cdots + b_1 x + b_0$$

to this data. The coefficients b_i are obtained as solution of

$$P(i) \approx y_i, \quad \text{for } i = -N, \dots, N$$

which is a linear least squares problem $A\mathbf{b} \approx \mathbf{y}$. The smoothed value is $P(0) = b_0$. Since $\mathbf{b} = A^+ \mathbf{y}$ we obtain

$$b_0 = \mathbf{e}_{p+1}^\top A^+ \mathbf{y} = \mathbf{c}^\top \mathbf{y}$$

Write a MAPLE script to compute the coefficient vector \mathbf{c} .

Hint: in MAPLE's **CurveFitting** package there is a function **LeastSquares** which is useful.

PROBLEM 6.17. A triangle has been measured, the measurements are as follows:

$$\begin{array}{c|c|c|c|c|c} x_1 = \alpha & x_2 = \beta & x_3 = \gamma & x_4 = a & x_5 = b & x_6 = c \\ \hline 67^\circ 30' & 52^\circ & 60^\circ & 172m & 146m & 165m \end{array}$$

The measurements \mathbf{x} have errors. We would like to correct them so that the new values $\tilde{\mathbf{x}} = \mathbf{x} + \mathbf{h}$ are consistent quantities of a triangle. They have to satisfy:

$$\begin{aligned} \text{Sum of angles:} & \quad \tilde{x}_1 + \tilde{x}_2 + \tilde{x}_3 = 180^\circ \\ \text{Sine theorem:} & \quad \tilde{x}_4 \sin \tilde{x}_2 - \tilde{x}_5 \sin \tilde{x}_1 = 0 \\ & \quad \tilde{x}_5 \sin \tilde{x}_3 - \tilde{x}_6 \sin \tilde{x}_2 = 0 \end{aligned} \tag{6.108}$$

Solve the constrained least squares problem $\|\mathbf{h}\|_2 \rightarrow \min$ subject to the constraints (6.108). Replace the nonlinear constraints $\mathbf{f}(\tilde{\mathbf{x}}) = 0$ by the linearized equations $\mathbf{f}(\mathbf{x}) + J\mathbf{h} = 0$ where J is the Jacobian. Solve the linearized problem using e.g. **NullSpaceMethod**. Iterate the process if necessary. Hint: Don't forget to work in radians!

Check that for the new values also e.g. the cosine-theorem $c^2 = a^2 + b^2 - 2ab \cos(\gamma)$ holds.

You will notice that the corrections will be made mainly to the angles and much less to the lengths of the sides of the triangle. This is because the measurements have not the same absolute errors. While the error in last digit of the sides is about 1, the errors in radians of the angles are about 0.01.

Repeat your computation by taking in account with appropriate weighting the difference in measurement errors. Minimize not simply $\|\mathbf{h}\|_2$ but

$$\left\| \begin{pmatrix} 100h_1 \\ 100h_2 \\ 100h_3 \\ h_4 \\ h_5 \\ h_6 \end{pmatrix} \right\|_2$$

PROBLEM 6.18. Prove that the following compact algorithm solves the linear least squares problem $A\mathbf{x} \approx \mathbf{b}$.

1. Form the augmented matrix

$$\bar{A} = [A, \mathbf{b}] \in \mathbb{R}^{m \times n+1}.$$

2. Compute the augmented normal equation matrix and decompose it using the Cholesky decomposition

$$\bar{A}^\top \bar{A} = \bar{R}^\top \bar{R}.$$

3. Partition

$$\bar{R} = \begin{pmatrix} R & \mathbf{y} \\ 0 & \rho \end{pmatrix}. \quad (6.109)$$

4. The least squares solution $\tilde{\mathbf{x}}$ is obtained by solving $R\mathbf{x} = \mathbf{y}$ with back-substitution.
5. The residual is $\|\mathbf{r}\|_2 = \|\mathbf{b} - A\tilde{\mathbf{x}}\|_2 = \rho$.

PROBLEM 6.19. Suppose you are given the decomposition of the matrix $A = LU$ where L is a lower and U an upper triangular matrix. Thus you can solve the system $A\mathbf{x} = \mathbf{b}$ in two steps

1. Solve $L\mathbf{y} = \mathbf{b}$ by forward substitution
2. Solve $U\mathbf{x} = \mathbf{y}$ by backward substitution.

Example

$$L = \begin{pmatrix} 1 & & & & \\ 9 & 1 & & & \\ 1 & 5 & 1 & & \\ 9 & 10 & 5 & 1 & \\ 6 & 10 & 8 & 10 & 1 \end{pmatrix}, \quad U = \begin{pmatrix} 8 & 1 & 2 & 1 & 7 \\ & 3 & 10 & 4 & 0 \\ & & 10 & 9 & 8 \\ & & & 8 & 9 \\ & & & & 7 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} 15 \\ 138 \\ 39 \\ 211 \\ 209 \end{pmatrix}$$

Write two MATLAB functions $\mathbf{y} = \text{ForwardSubstitution}(L, \mathbf{b})$ and $\mathbf{x} = \text{BackSubstitution}(U, \mathbf{y})$ so that they can be used to solve the linear system $LU\mathbf{x} = \mathbf{b}$. Both functions can be programmed using either scalar-product-operations or SAXPY-operations. A SAXPY is defined as a linear combination of two vectors:

$$\mathbf{s} = \alpha \mathbf{x} + \mathbf{y}.$$

For modern processors the SAXPY variant is preferred, therefore program these versions.

PROBLEM 6.20. Fitting of circles. We are given the measured points (ξ_i, η_i) :

| | | | | | | | | |
|--------|-----|-----|-----|-----|------|------|------|------|
| ξ | 0.7 | 3.3 | 5.6 | 7.5 | 6.4 | 4.4 | 0.3 | -1.1 |
| η | 4.0 | 4.7 | 4.0 | 1.3 | -1.1 | -3.0 | -2.5 | 1.3 |

Find the center (z_1, z_2) and the radius r of a circle $(x - z_1)^2 + (y - z_2)^2 = r^2$ that approximate the points as well as possible. Consider the two cases

1. Algebraic fit: Rearrange the equation of the circle as

$$2z_1x + 2z_2y + r^2 - z_1^2 - z_2^2 = x^2 + y^2. \quad (6.110)$$

With $c := r^2 - z_1^2 - z_2^2$, we obtain with (6.110) for each measured point a linear equation for the unknowns z_1, z_2 and c .

2. Geometric fit: Use the Gauss-Newton method to minimize the sum of squares of the distances

$$d_i = |\sqrt{(z_1 - \xi_i)^2 + (z_2 - \eta_i)^2} - r|.$$

Compute and plot in both cases the data points together with the circles. As a second example do the same with the data set

| | | | | | | |
|--------|---|---|---|---|---|---|
| ξ | 1 | 2 | 5 | 7 | 9 | 3 |
| η | 7 | 6 | 8 | 7 | 5 | 7 |

Hint: A circle with center (c_1, c_2) and with radius r is best plotted using the parametric representation:

$$x(t) = c_1 + r \cos t, \quad y(t) = c_2 + r \sin t \quad 0 \leq t \leq 2\pi.$$

PROBLEM 6.21. The parametric form commonly used for the circle is given by

$$\begin{aligned} x &= z_1 + r \cos \varphi \\ y &= z_2 + r \sin \varphi. \end{aligned}$$

The distance d_i of a point $P_i = (x_{i1}, x_{i2})$ may be expressed by

$$d_i^2 = \min_{\varphi_i} (x_{i1} - x(\varphi_i))^2 + (x_{i2} - y(\varphi_i))^2.$$

Now we want again compute a geometric fit, i.e. determine z_1, z_2 and r by minimizing

$$\sum_{i=1}^m d_i^2 \longrightarrow \min.$$

We can simultaneously minimize for z_1, z_2, r and $\{\varphi_i\}_{i=1\dots m}$; i.e. find the minimum of the quadratic function

$$Q(\varphi_1, \varphi_2, \dots, \varphi_m, z_1, z_2, r) = \sum_{i=1}^m (x_{i1} - x(\varphi_i))^2 + (x_{i2} - y(\varphi_i))^2.$$

This is equivalent to solving the nonlinear least squares problem

$$\begin{aligned} z_1 + r \cos \varphi_i - x_{i1} &\approx 0 \\ z_2 + r \sin \varphi_i - x_{i2} &\approx 0 \end{aligned} \quad i = 1, 2, \dots, m.$$

Solve this problem with the Gauss-Newton method. The Jacobian J is highly structured. Taking in account the structure when solving $J\mathbf{h} \approx -\mathbf{f}$ develop an effective algorithm.

PROBLEM 6.22. Write a MATLAB programs to fit two orthogonal lines by minimizing the distance of measured points to the line. Assume that two different sets of points are given for the two lines involved.

PROBLEM 6.23. The function

$$h(x) = 1 - \frac{3 \sinh x - \sin x}{x \cosh x - \cos x}$$

should be fitted to the following data:

| | | | | | | | | | | |
|-----|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| x | 0.100 | 0.800 | 1.500 | 2.200 | 2.900 | 3.600 | 4.300 | 5.000 | 5.700 | 6.400 |
| y | 0.049 | 0.051 | 0.153 | 0.368 | 0.485 | 0.615 | 0.712 | 0.717 | 0.799 | 0.790 |

Find the value of the parameter a such that $h(ax_i) \approx y_i$ for $i = 1, \dots, 10$ in the least squares sense.

Solve the problem using the Gauss-Newton and also the Newton method. Compute the function and the derivatives using algorithmic differentiation (see Section 8.3).

PROBLEM 6.24. We are given the coordinates of the points (ξ_i, η_i) , $i = 1, \dots, m$ in the plane. For a new point $P = (x_1, x_2)$ the distances $s_i = |P_i - P|$ from the given points have been measured:

| | | | | | | |
|-------|----|----|----|----|----|----|
| x_i | 16 | 65 | 85 | 53 | 16 | 25 |
| y_i | 56 | 64 | 37 | 7 | 3 | 32 |
| s_i | 32 | 35 | 44 | 30 | 42 | 16 |

Compute the coordinates of P by solving the nonlinear least squares problem.

PROBLEM 6.25. Fit the function $f(x) = k/(1 + be^{-ax})$ to the data

| | | | | | | | | | | | | | | |
|-----|------|-----|------|-----|-----|-----|------|-----|------|-----|-----|-----|------|------|
| x | 0 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| y | .145 | .19 | .248 | .29 | .78 | .78 | 1.16 | 1.4 | 1.94 | 2.3 | 2.5 | 2.8 | 3.12 | 3.32 |

using the MATLAB function `nlinfit`. Plot the points and the fitted curve.

PROBLEM 6.26. Determine the parameters a and b such that the function $f(x) = ae^{bx}$ fits the following data

| | | | | | | |
|-----|------|------|------|------|------|------|
| x | 30.0 | 64.5 | 74.5 | 86.7 | 94.5 | 98.9 |
| y | 4 | 18 | 29 | 51 | 73 | 90 |

Hint: If you fit $\log f(x)$ the problems becomes very easy!

PROBLEM 6.27. The following program fits a polynomial of degree $n = 5$ to given points. The points are input with the mouse using the MATLAB function `ginput`. After each new point the coefficients are computed and the polynomial is plotted. The program is inefficient because it does not update the QR decomposition – the solution is fully recalculated for every new point.

```
format compact
clf
axis([0 10 0 10])           % plot window
hold
degr=5;                      % degree of polynomial
n=degr+1;                    % number of coefficients
h=degr:-1:0                  % to generate matrix row
t=0:0.1:10;                  % to plot polynomial
[x,y]=ginput(1);
plot(x,y,'*r')               % generate first row
A= x.^h; b=y; k=1;
while 1                      % stop with ctrl-w in plot window
    [x,y]=ginput(1);         % get new point
    plot(x,y,'*r')
    k=k+1;                   % count points
    A=[A; x.^h];              % generate new row
    b=[b;y];                  % and right hand side
    if k>=n
        a=A\b;               % solve for degrew coefficients
        a'                   % display coefficients
    end
    if k>degr
        p=polyval(a,t);      % evaluate polynomial
        plot(t,p)            % and plot
    end
end
end
```

1. Study the program and run a few examples with different degrees.
2. Replace the part between the comment signs so that the solution is updated with Givens rotations or -reflections. Each time when a new point is read the matrix R is updated with n Givens transformations. These Givens transformations annihilate the matrix-elements of the new equation and by back-substitution we obtain the new coefficients of the polynomial. Use the scalar product form for back-substitution.

PROBLEM 6.28. Assume you have decomposed a large matrix $A = QR$ and afterward you discover that the element a_{jk} is wrong. Use our update-techniques to fix the QR decomposition.

Test your Algorithm for the small matrix $A = \text{gallery}(5)$; $A = A(:, 1:3)$

$$A = \begin{pmatrix} -9 & 11 & -21 \\ 70 & -69 & 141 \\ -575 & 575 & -1149 \\ 3891 & -3891 & 7782 \\ 1024 & -1024 & 2048 \end{pmatrix}$$

Change the element $a_{2,3} = 141$ to $a_{2,3} = 1414$ and compute the new decomposition.

PROBLEM 6.29. Consider the augmented matrix $\bar{A} = [A, \mathbf{b}]$. Show that the following “compact” algorithm solves the least squares problem $A\mathbf{x} \approx \mathbf{b}$ and is equivalent with the method of normal equations:

1. decompose $\bar{A}\bar{A}^T = \bar{L}\bar{L}^T$ (Cholesky).
2. Set $R = L(1:n, 1:n)$, $\mathbf{y} = L(n+1, 1:n)'$ and $\rho = L(n+1, n+1)$.
3. Solve $R\mathbf{x} = \mathbf{y}$ by back-substitution.
4. $\min \|\mathbf{b} - A\mathbf{x}\|_2 = \rho$.

PROBLEM 6.30. Derivation of modified Gram-Schmidt via matrix decomposition. Let A be a $m \times n$ matrix and consider the decomposition $A = QR$. If we set $L = R^T$ we can view the factorization as an outer product expansion

$$A = QL^T = \sum_{i=1}^n \mathbf{q}_i \mathbf{l}_i^T$$

where $\mathbf{l}_i^T = (0, \dots, 0, r_{ii}, \dots, r_{in})$ is the i th column vector of R . Observe that the first $i - 1$ columns of the rank one matrix $\mathbf{q}_i \mathbf{l}_i^T$ consist entirely of 0's. Define

$$A^{(k)} = A - \sum_{i=1}^{k-1} \mathbf{q}_i \mathbf{l}_i^T = \sum_{i=k}^n \mathbf{q}_i \mathbf{l}_i^T \quad k = 1, \dots, n+1. \quad (6.111)$$

Clearly the recursion holds

$$A^{(1)} = A, \quad A^{(k+1)} = A^{(k)} - \mathbf{q}_k \mathbf{l}_k^\top, \quad A^{n+1} = 0.$$

Assume now that $k-1$ columns of Q and $k-1$ rows of L are already computed. Multiply the recursion from the right by the unit vector \mathbf{e}_k and from the left by \mathbf{q}_k^\top to get expressions for the k -th column of Q and the k -th row of L . Write a MATLAB-program to compute this way the QR decomposition.

PROBLEM 6.31. Consider the matrix A which is constructed by

```
c=4.11;
m=13;
n=13;
condA_glob=c;
B=inv(pascal(m));
B=B(:,1:n);
[A,R]=qr(B,0);
C=inv(hilb(n));
[B,R]=qr(C,0);
A=A*diag([10.^(0:condA_glob/(n-1):condA_glob)])*B;
[m,n]=size(A);
```

Compute the QR decomposition

1. with classical Gram-Schmidt
2. with modified Gram-Schmidt
3. via Cholesky decomposition of $A^\top A$
4. with MATLAB's function `qr`

In each case, test the departure from orthogonality using $\text{norm}(\text{eye}(13) - Q' * Q)$. (This matrix was communicated by Alicja Smoktunowicz).

PROBLEM 6.32. We are given the following data concerning the growth of pigs. The weight of a pig has been measured over a period of 240 days. The values are given in the following table:

| | | | | | | | | | |
|---|--------|--------|--------|--------|--------|--------|--------|-------|-------|
| t | 0 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 |
| y | 1.64 | 2.68 | 5.81 | 7.45 | 9.98 | 12.51 | 15.34 | 19.07 | 23.24 |
| t | 90 | 100 | 110 | 120 | 130 | 140 | 150 | 160 | 170 |
| y | 28.90 | 35.60 | 42.90 | 51.39 | 61.07 | 69.71 | 79.54 | 88.03 | 95.18 |
| t | 180 | 190 | 200 | 210 | 220 | 230 | 240 | | |
| y | 100.42 | 105.01 | 108.07 | 111.87 | 115.12 | 118.01 | 120.67 | | |

The data suggest an exponential growth of the weight in a first phase followed by an exponential decrease of the growth to a final limit weight (which is not reached since the pigs are transformed to meat before that stage).

Thus it seems reasonable to approximate the data by two piecewise functions

$$F(x) = \begin{cases} f(\mathbf{a}, t) &= a_0 + a_1 \exp(a_3 t) & t < \xi \\ g(\mathbf{b}, t) &= b_0 + b_1 \exp(b_3 t) & t > \xi \end{cases}$$

We expect that by a least squares fit we will obtain exponents $a_3 > 0$ and $b_3 < 0$. The break point ξ is a free knot and the two functions should have the same value and the same derivatives for $t = \xi$.

Use the theory developed in Section 6.9 to determine the parameters by the Gauss-Newton method.

Depending on the initial values, you may have a hard time to converge to a solution. Use the Levenberg-Marquardt correction to avoid large correction steps.

Scientific Computing - An Introduction using Maple and
MATLAB

Gander, W.; Gander, M.J.; Kwok, F.

2014, XVIII, 905 p. 133 illus., 53 illus. in color.,

Hardcover

ISBN: 978-3-319-04324-1